Principle Components Analysis (PCA)

Rebecca C. Steorts, Duke University

STA 325, Chapter 10 ISL

Introduction

In our last lecture, we saw that locality sensitive hashing allows us to perform dimension reduction via clustering. In this lecture, we will look at a different form of dimension reduction of the data (features) using linear projections. Principal components analysis (PCA) performs dimension reduction on the data using linear projections, where the variance is preserved. This lecture explains PCA, how to do PCA, an illustrative example, and some issues that arise when we try and interpret the results.[1]

Mathematics of Principal Components

We start with p-dimensional feature vectors, and want to summarize them by projecting down into a q-dimensional subspace. Our summary will be the projection of the original vectors on to q directions, the principal components, which span the sub-space. There are several equivalent ways of deriving the principal components mathematically. The simplest one is by finding the projections which maximize the variance.

The first principal component is the direction in feature space along which projections have the largest variance. The second principal component is the direction which maximizes variance among all directions orthogonal to the first. The k^{th} component is the variance-maximizing direction orthogonal to the previous $k-1$ components. There are p principal components in all.

Rather than maximizing variance, it might sound more plausible to look for the projection with the smallest average (mean-squared) distance between the original vectors and their projections on to the principal components; this turns out to be equivalent to maximizing the variance.

Throughout, assume that the data have been "centered", so that every feature has mean 0. If we write the centered data in a matrix $\boldsymbol{X}$, where rows are objects and columns are features, then $\boldsymbol{X}^T\boldsymbol{X} = nV$, where V is the covariance matrix of the data. (Check the last statement on your own.)

Minimizing Projection Residuals

We'll start by looking for a one-dimensional projection. That is, we have p-dimensional feature vectors, and we want to project them on to a line through the origin. We can specify the line by a unit vector along it, $\vec{w}$, and then the projection of a data vector $\vec{x_i}$ on to the line is $\vec{x_i} \cdot \vec{w}$, which is a scalar. (Sanity check: this gives us the right answer when we project on to one of the coordinate axes.)
This is the distance of the projection from the origin; the actual coordinate in p-dimensional space is $(\vec{x_i} \cdot \vec{w})\vec{w}$. The mean of the projections will be zero, because the mean of the vectors $\vec{x_i}$ is zero:

$$\frac{1}{n}\sum_{i=1}^{n}(\vec{x_i} \cdot \vec{w})\vec{w} = \left(\left(\frac{1}{n}\sum_{i=1}^{n} x_i\right) \cdot \vec{w}\right)\vec{w} \tag{1}$$

If we try to use our projected or image vectors instead of our original vectors, there will be some error, because (in general) the images do not coincide with the original vectors. (When do they coincide?) The difference is the error or residual of the projection. How big is it?

[1]PCA has been rediscovered many times in many fields, so it is also known as the Karhunen-Loeve transformation, the Hotelling transformation, the method of empirical orthogonal functions, and singular value decomposition.

For any one vector, say $\vec{x_i}$, it's

$$\begin{aligned} \|\vec{x_i} - (\vec{w} \cdot \vec{x_i})\vec{w}\|^2 &= \|\vec{x_i}\|^2 - 2(\vec{w} \cdot \vec{x_i})(\vec{w} \cdot \vec{x_i}) + \|\vec{w}\|^2 && (2) \\ &= \|\vec{x_i}\|^2 - 2(\vec{w} \cdot \vec{x_i})^2 + 1 && (3) \end{aligned}$$

(This is a common trick to compute distance matrices; it's really just the Pythagorean theorem.) Add those residuals up across all the vectors:

$$\begin{aligned} RSS(\vec{w}) &= \sum_{i=1}^{n} \|\vec{x_i}\|^2 - 2(\vec{w} \cdot \vec{x_i})^2 + 1 && (4) \\ &= \left(n + \sum_{i=1}^{n} \|\vec{x_i}\|^2\right) - 2\sum_{i=1}^{n} (\vec{w} \cdot \vec{x_i})^2 && (5) \end{aligned}$$

The term in the big parenthesis doesn't depend on $\vec{w}$, so it doesn't matter for trying to minimize the residual sum-of-squares. To make RSS small, what we must do is make the second sum big, i.e., we want to maximize

$$\sum_{i=1}^{n} (\vec{w} \cdot \vec{x_i})^2 \tag{6}$$

Equivalently, since n doesn't depend on $\vec{w}$, we want to maximize

$$\frac{1}{n}\sum_{i=1}^{n} (\vec{w} \cdot \vec{x_i})^2 \tag{7}$$

which we can see is the sample mean of $(\vec{w} \cdot \vec{x_i})^2$. The mean of a square is always equal to the square of the mean plus the variance:

$$\frac{1}{n}\sum_{i=1}^{n} (\vec{w} \cdot \vec{x_i})^2 = \left(\frac{1}{n}\sum_{i=1}^{n} \vec{x_i} \cdot \vec{w}\right)^2 + \mathrm{Var}(\vec{w} \cdot \vec{x_i}) \tag{8}$$

Since we've just seen that the mean of the projections is zero, minimizing the residual sum of squares turns out to be equivalent to maximizing the variance of the projections.

(Of course in general we don't want to project on to just one vector, but on to multiple principal components. If those components are orthogonal and have the unit vectors $\vec{w_1}, \vec{w_2}, \ldots \vec{w_k}$, then the image of x_i is its projection into the space spanned by these vectors,

$$\sum_{j=1}^{k} (\vec{x_i} \cdot \vec{w_j})\vec{w_j} \tag{9}$$

The mean of the projection on to each component is still zero. If we go through the same algebra for the residual sum of squares, it turns out that the cross-terms between different components all cancel out, and we are left with trying to maximize the sum of the variances of the projections on to the components. EXERCISE: Do this algebra.)

Maximizing Variance

Accordingly, let's maximize the variance! Writing out all the summations grows tedious, so let's do our algebra in matrix form.

If we stack our n data vectors into an $n \times p$ matrix, X, then the projections are given by $X\boldsymbol{w}$, which is an $n \times 1$ matrix. The variance is

$$\begin{aligned}
\sigma^2_{\vec{w}} &= \frac{1}{n}\sum_i (\vec{x_i} \cdot \vec{w})^2 && (10)\\
&= \frac{1}{n}(X\boldsymbol{w})^T(X\boldsymbol{w}) && (11)\\
&= \frac{1}{n}\boldsymbol{w}^T X^T X\boldsymbol{w} && (12)\\
&= \boldsymbol{w}^T \frac{X^T X}{n}\boldsymbol{w} && (13)\\
&= \boldsymbol{w}^T V\boldsymbol{w} && (14)
\end{aligned}$$

We want to chose a unit vector $\vec{w}$ so as to maximize $\sigma^2_{\vec{w}}$. To do this, we need to make sure that we only look at unit vectors — we need to constrain the maximization. The constraint is that $\vec{w} \cdot \vec{w} = 1$, or $\boldsymbol{w}^T\boldsymbol{w} = 1$. This needs a brief excursion into constrained optimization.

We start with a function $f(w)$ that we want to maximize. (Here, that function is $\boldsymbol{w}^T V\boldsymbol{w}$.) We also have an equality constraint, $g(w) = c$. (Here, $g(w) = \boldsymbol{w}^T\boldsymbol{w}$ and $c = 1$.)

We re-arrange the constraint equation so its right-hand side is zero, $g(w) - c = 0$. We now add an extra variable to the problem, the Lagrange multiplier λ, and consider $u(w, \lambda) = f(w) - \lambda(g(w) - c)$. This is our new objective function, so we differentiate with respect to both arguments and set the derivatives equal to zero:

$$\begin{aligned}
\frac{\partial u}{\partial w} = 0 = &\ \frac{\partial f}{\partial w} - \lambda\frac{\partial g}{\partial w} && (15)\\
\frac{\partial u}{\partial \lambda} = 0 = &\ -(g(w) - c) && (16)
\end{aligned}$$

That is, maximizing with respect to λ gives us back our constraint equation, $g(w) = c$. At the same time, when we have the constraint satisfied, our new objective function is the same as the old one. (If we had more than one constraint, we would just need more Lagrange multipliers.)

For our projection problem,

$$\begin{aligned}
u &= \boldsymbol{w}^T V\boldsymbol{w} - \lambda(\boldsymbol{w}^T\boldsymbol{w} - 1) && (17)\\
\frac{\partial u}{\partial \boldsymbol{w}} &= 2V\boldsymbol{w} - 2\lambda\boldsymbol{w} = 0 && (18)\\
V\boldsymbol{w} &= \lambda\boldsymbol{w} && (19)
\end{aligned}$$

Thus, desired vector $\boldsymbol{w}$ is an eigenvector of the covariance matrix V, and the maximizing vector will be the one associated with the largest eigenvalue λ. This is good news, because finding eigenvectors is something which can be done comparatively rapidly (see "Principles of Data Mining" p. 81), and because eigenvectors have many nice mathematical properties, which we can use as follows.

We know that V is a $p \times p$ matrix, so it will have p different eigenvectors.[2] We know that V is a covariance matrix, so it is symmetric, and then linear algebra tells us that the eigenvectors must be orthogonal to one another. Again because V is a covariance matrix, it is a positive matrix, in the sense that $\vec{x} \cdot V\vec{x} \geq 0$ for any $\vec{x}$. This tells us that the eigenvalues of V must all be ≥ 0.

More Geometry; Back to the Residuals

Suppose that the data really are q-dimensional. Then V will have only q positive eigenvalues, and $p - q$ zero eigenvalues. If the data fall near a q-dimensional subspace, then $p - q$ of the eigenvalues will be nearly zero.

If we pick the top q components, we can define a projection operator $\mathbf{P}_q$. The images of the data are then $X\mathbf{P}_q$. The projection residuals are $X - X\mathbf{P}_q$ or $X(\mathbf{1} - \mathbf{P}_q)$. (Notice that the residuals here are vectors, not just magnitudes.) If the data really are q-dimensional, then the residuals will be zero. If the data are approximately q-dimensional, then the residuals will be small. In any case, we can define the R^2 of the projection as the fraction of the original variance kept by the image vectors,

$$R^2 \equiv \frac{\sum_{i=1}^{q} \lambda_i}{\sum_{j=1}^{p} \lambda_j} \qquad (20)$$

just as the R^2 of a linear regression is the fraction of the original variance of the dependent variable retained by the fitted values.

The $q = 1$ case is especially instructive. We know, from the discussion of projections in the last lecture, that the residual vectors are all orthogonal to the projections. Suppose we ask for the first principal component of the residuals. This will be the direction of largest variance which is perpendicular to the first principal component. In other words, it will be the second principal component of the data. This suggests a recursive algorithm for finding all the principal components: the k^{th} principal component is the leading component of the residuals after subtracting off the first $k - 1$ components. In practice, it is faster to use eigenvector-solvers to get all the components at once from V, but we will see versions of this idea later.

This is a good place to remark that if the data really fall in a q-dimensional subspace, then V will have only q positive eigenvalues, because after subtracting off those components there will be no residuals. The other $p - q$ eigenvectors will all have eigenvalue 0. If the data cluster around a q-dimensional subspace, then $p - q$ of the eigenvalues will be very small, though how small they need to be before we can neglect them is a tricky question.[3]

Cars dataset

We're going to look at a small dataset of 388 cars from the 2004 model year, with 18 features, with incomplete records removed). Eight features are binary indicators. The other 11 features are numerical. All of the features except `Type` are numerical. PCA only works with numerical features, so we have ten of them to play with. Below we load in the data and show the first few lines of the data.

```r
library(knitr)
cars04 = read.csv("cars-fixed04.dat")
kable(head(cars04))
```

[2]Exception: if $n < p$, there are only n distinct eigenvectors and eigenvalues.

[3]One tricky case where this can occur is if $n < p$. Any two points define a line, and three points define a plane, etc., so if there are fewer data points than features, it is necessarily true that the fall on a low-dimensional subspace. If we look at the bags-of-words for the Times stories, for instance, we have $p \approx 4400$ but $n \approx 102$. Finding that only 102 principal components account for all the variance is not an empirical discovery but a mathematical artifact.

	Sports	SUV	Wagon	Minivan	Pickup	AWD	RWD	Retail	Dealer	Engine	Cylin
Acura 3.5 RL	0	0	0	0	0	0	0	43755	39014	3.5	
Acura 3.5 RL Navigation	0	0	0	0	0	0	0	46100	41100	3.5	
Acura MDX	0	1	0	0	0	1	0	36945	33337	3.5	
Acura NSX S	1	0	0	0	0	0	1	89765	79978	3.2	
Acura RSX	0	0	0	0	0	0	0	23820	21761	2.0	
Acura TL	0	0	0	0	0	0	0	33195	30299	3.2	

Let's now run PCA on the dataset and be sure to scale all the variables to have variance 1.

```
cars04.pca <- prcomp(cars04[,8:18], scale.=TRUE)
```

Let's now extract the weight or loading matrix from the `cars04.pca` object. Specifically, let's just look at the first two principal components.

```
# grab loadings, look at first two PCs
round(cars04.pca$rotation[,1:2], 2)
```

```
##              PC1   PC2
## Retail     -0.26 -0.47
## Dealer     -0.26 -0.47
## Engine     -0.35  0.02
## Cylinders  -0.33 -0.08
## Horsepower -0.32 -0.29
## CityMPG     0.31  0.00
## HighwayMPG  0.31  0.01
## Weight     -0.34  0.17
## Wheelbase  -0.27  0.42
## Length     -0.26  0.41
## Width      -0.30  0.31
```

How can we interpret these two components?

Our results suggest that all the variables except the gas-mileages have a negative projection on to the first component. This means that there is a negative correlation between mileage and everything else. The first principal component tells us about whether we are getting a big, expensive gas-guzzling car with a powerful engine, or whether we are getting a small, cheap, fuel-efficient car with a wimpy engine.

The second component is a little more interesting. Engine size and gas mileage hardly project on to it at all. Instead we have a contrast between the physical size of the car (positive projection) and the price and horsepower. Basically, this axis separates mini-vans, trucks and SUVs (big, not so expensive, not so much horse-power) from sports-cars (small, expensive, lots of horse-power).

How can we see this more clearly? We can look at visualizations, such as a bi-plot. A bi-plot plots the data along with the projections of the original features, on to the first two components.

```
biplot(cars04.pca, scale=0, cex=0.4)
```

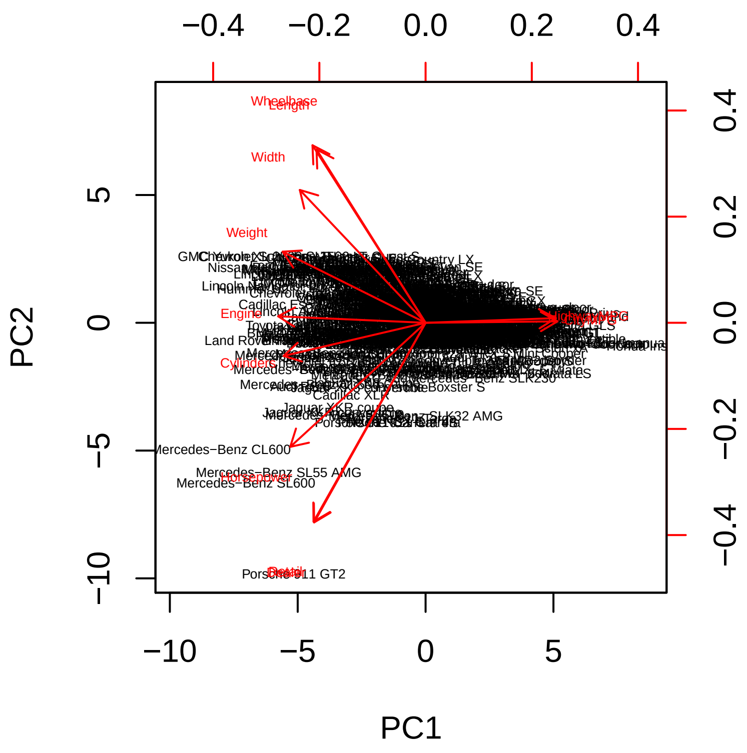

Figure: "Biplot" of the 2004 cars data. The horizontal axis shows projections on to the first principal component, the vertical axis the second component. Car names are written at their projections on to the components (using the coordinate scales on the top and the right). Red arrows show the projections of the original features on to the principal components (using the coordinate scales on the bottom and on the left).

Notice that the car with the lowest value of the second component is a Porsche 911, with pick-up trucks and mini-vans at the other end of the scale. Similarly, the highest values of the first component all belong to hybrids.

How to Interpret PCA Plots

There is a more-or-less standard recipe for interpreting PCA plots, which goes as follows.

To begin with, find the first two principal components of your data. (I say "two" only because that's what you can plot; see below.) It's generally a good idea to standardized all the features first, but not strictly necessary.

Coordinates Using the arrows, summarize what each component means. For the cars, the first component is something like size vs. fuel economy, and the second is something like sporty vs. boxy.

Correlations For many datasets, the arrows cluster into groups of highly correlated attributes. Describe these attributes. Also determine the overall level of correlation (given by the R^2 value). Here we get groups of arrows like the two MPGs (unsurprising), retail and dealer price (ditto) and the physical dimensions of the car (maybe a bit more interesting).

Clusters Clusters indicate a preference for particular combinations of attribute values. Summarize each cluster by its prototypical member. For the cars data, we see a cluster of very similar values for sports-cars, for instance, slightly below the main blob of data.

Funnels Funnels are wide at one end and narrow at the other. They happen when one dimension affects the variance of another, orthogonal dimension. Thus, even though the components are uncorrelated (because they are perpendicular) they still affect each other. (They are uncorrelated but not *independent*.) The cars data has a funnel, showing that small cars are similar in sportiness, while large cars are more varied.

Voids Voids are areas inside the range of the data which are unusually unpopulated. A **permutation plot** is a good way to spot voids. (Randomly permute the data in each column, and see if any new areas become occupied.) For the cars data, there is a void of sporty cars which are very small or very large. This suggests that such cars are undesirable or difficult to make.

Projections on to the first two or three principal components can be visualized; however they may not be enough to really give a good summary of the data. Usually, to get an R^2 of 1, you need to use all p principal components.[4] How many principal components you should use depends on your data, and how big an R^2 you need. In some fields, you can get better than 80% of the variance described with just two or three components. A sometimes-useful device is to plot $1 - R^2$ versus the number of components, and keep extending the curve it until it flattens out.

[4]The exceptions are when some of your features are linear combinations of the others, so that you don't really have p *different* features, or when $n < p$.