Unsupervised Learning

Zhongjian Lin
University of Georgia

October 29, 2024

Unsupervised Learning

Supervised learning methods are concerned with predicting the values of one or more outputs or response variables $Y = (Y_1, \cdots, Y_m)$ for a given set of input or predictor variables $X' = (X_1, \cdots, X_p)$. If one supposes that (X, Y) are random variables represented by some joint probability density $f(X, Y)$, then supervised learning can be formally characterized as a density estimation problem where one is concerned with determining properties of the conditional density $f(Y|X)$. Probability theory has

$$f(X, Y) = f(Y|X) \cdot f(X).$$

Since Y is often of low dimension (usually one),the problem is greatly simplified.

Unsupervised Learning

In unsupervised learning, we address "learning without a teacher." In this case one has a set of n observations $(x_1, x_2, \cdots, x_n)$ of a random p-vector X having joint density $f(X)$. The goal is to directly infer the properties of this probability density without the help of a supervisor or teacher providing correct answers or degree-of-error for each observation.
In low-dimensional problems (say $p \leq 3$), there are a variety of effective nonparametric methods for directly estimating the density $f(X)$ itself at all X-values, and representing it graphically (Silverman, 1986). Owing to the curse of dimensionality, these methods fail in high dimensions.

Unsupervised Learning

Unsupervised learning subsumes all kinds of machine learning where there is no known output, no teacher to instruct the learning algorithm. In unsupervised learning, the learning algorithm is just shown the input data, and asked to extract knowledge from this data. We will look into two kinds of unsupervised learning in this chapter

- Transformation: create a new representation of the data which might be easier for humans or other machine learning algorithms.
  - A common application of unsupervised transformations is dimensionality reduction and a common application for dimensionality reduction is reduction to two dimensions for visualization purposes.
  - Another application for unsupervised transformations is finding the parts or components that "make up" the data.
  - Principal components attempt to identify low-dimensional manifolds within the X-space that represent high data density. This provides information about the associations among the variables and whether or not they can be considered as functions of a smaller set of "latent" variables.
- Clustering: partition data into distinct groups of similar items.
  - Consider the example of uploading photos to a social media site. To allow you to organize your pictures, the site might want to group together pictures that show the same person. However, the site doesn't know which pictures show whom, and it doesn't know how many different people appear in your photo collection. A sensible approach would be to extract all faces, and divide them into groups of faces that look similar.

Challenge

- A major challenge in unsupervised learning is evaluating whether the algorithm learned something useful. Unsupervised learning algorithms are usually applied to data that does not contain any label information, so we don't know what the right output should be. Therefore it is very hard to say whether a model "did well"
- As a consequence, unsupervised algorithms are used often in an exploratory setting, when a data scientist wants to understand the data better, rather than as part of a larger automatic system.
- One must resort to heuristic arguments not only for motivating the algorithms, as is often the case in supervised learning as well, but also for judgments as to the quality of the results. This uncomfortable situation has led to heavy proliferation of proposed methods, since effectiveness is a matter of *opinion* and cannot be verified directly.
- Another common application for unsupervised algorithms is as a preprocessing step for supervised algorithms. Learning a new representation of the data can sometimes improve the accuracy of supervised algorithms, or can lead to reduced memory and time consumption.

Principal Component Analysis

Transforming data using unsupervised learning can have many motivations. The most common motivations are visualization, compressing the data, and finding a representation that is more informative for further processing. One of the simplest and most widely used algorithms for all of these is Principal Component Analysis.

- Principal component analysis (PCA) is a method that rotates the dataset in a way such that the rotated features are statistically uncorrelated. This rotation is often followed by selecting only a subset of the new features, according to how important they are for explaining the data.
- We can use PCA for dimensionality reduction by retaining only some of the principal components.

Principal Component Analysis

Suppose that we wish to visualize n observations with measurements on a set of p features, $X_1, X_2, \cdots, X_p$, as part of an exploratory data analysis. We could do this by examining two-dimensional scatterplots of the data, each of which contains the n observations' measurements on two of the features. However, there are $\binom{p}{2} = p(p-1)/2$ such scatterplots; for example, with $p = 10$ there are 45 plots! If p is large, then it will certainly not be possible to look at all of them; moreover, most likely none of them will be informative since they each contain just a small fraction of the total information present in the data set. Clearly, a better method is required to visualize the n observations when p is large. In particular, we would like to find a low-dimensional representation of the data that captures as much of the information as possible. For instance, if we can obtain a two-dimensional representation of the data that captures most of the information, then we can plot the observations in this low-dimensional space.

PCA provides a tool to do just this. It finds a low-dimensional representation of a data set that contains as much as possible of the variation. The idea is that each of the n observations lives in p-dimensional space, but not all of these dimensions are equally interesting. PCA seeks a small number of dimensions that are as interesting as possible, where the concept of interesting is measured by the amount that the observations vary along each dimension. Each of the dimensions found by PCA is a linear combination of the p features. We now explain the manner in which these dimensions, or principal components, are found. The first principal component of a set of features $X_1, X_2, \cdots, X_p$ is the normalized linear combination of the features

$$Z_1 = \phi_{11} X_1 + \phi_{21} X_2 + \cdots + \phi_{p1} X_p$$

that has the largest variance. We refer to the elements $\phi_{11}, \cdots, \phi_{p1}$ as the *loadings* of the first principal component

Estimation of PCA

Given an $n \times p$ data set X, how do we compute the first principal component? Since we are only interested in variance, we assume that each of the variables in X has been centered to have mean zero. We then look for the linear combination of the sample feature values of the form

$$z_{i1} = \phi_{11} x_{i1} + \phi_{21} x_{i2} + \cdots + \phi_{p1} x_{ip}$$

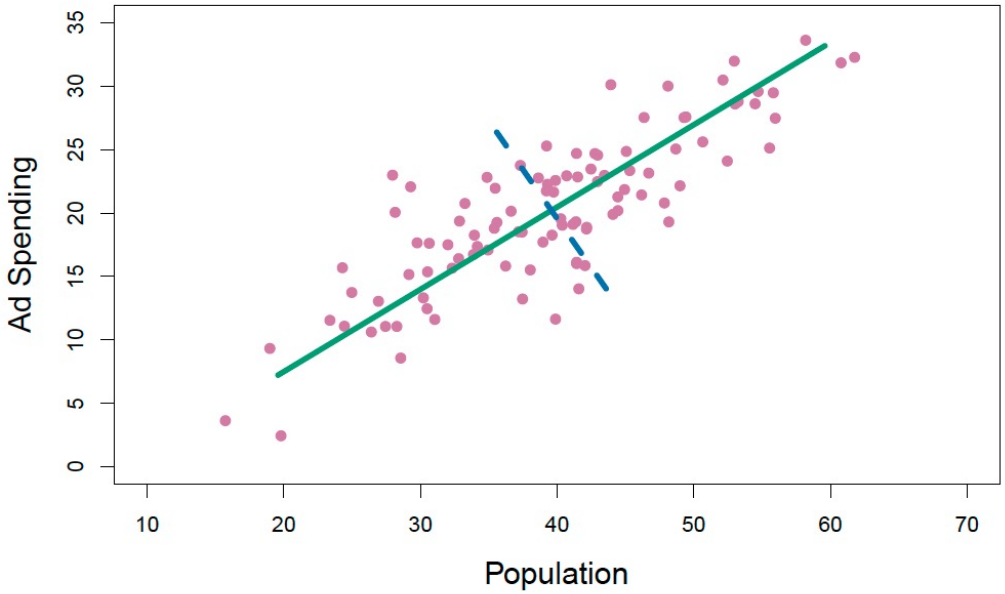
that has largest sample variance, subject to the constraint that $\sum_{j=1}^{p} \phi_{j1}^2 = 1$. In other words, the first principal component loading vector solves the optimization problem

$$\max_{\phi_{11},\cdots,\phi_{p1}} \left\{ \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{j=1}^{p} \phi_{j1} x_{ip} \right)^2 \right\} \text{ s.t. } \sum_{j=1}^{p} \phi_{j1}^2 = 1.$$

The problem can be solved via an eigen decomposition, standard technique in linear algebra.

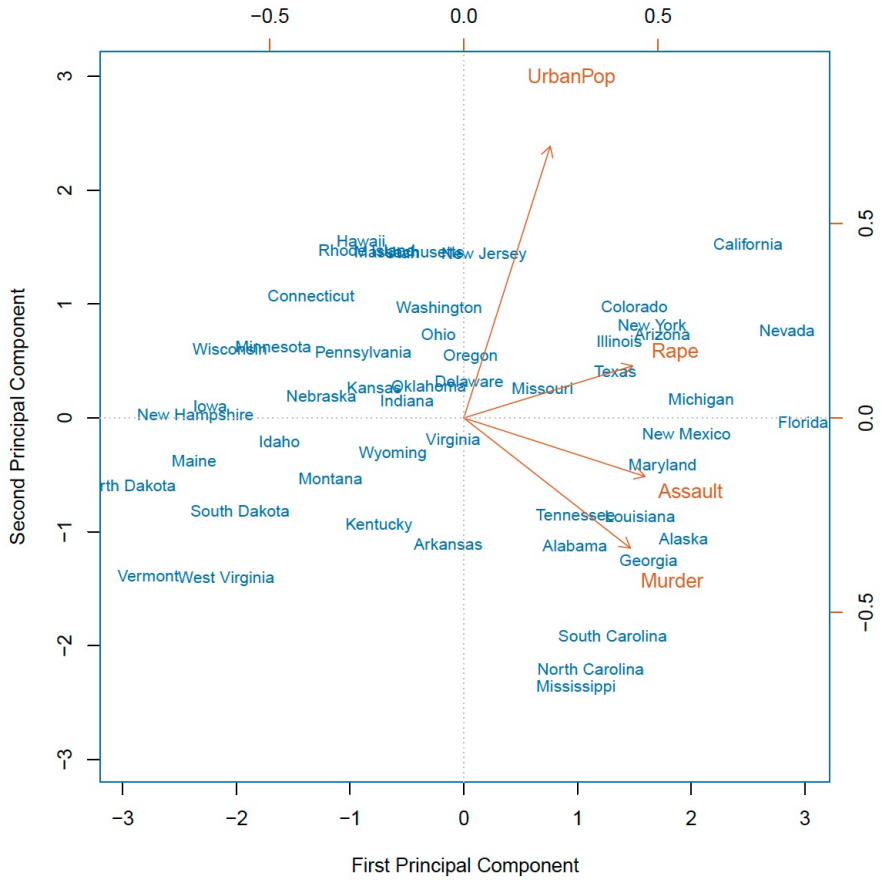
Estimation of PCA

There is a nice geometric interpretation of the first principal component. The loading vector ϕ_1 with elements $\phi_{11}, \cdots, \phi_{p1}$ defines a direction in feature space along which the data vary the most. If we project the n data points $x_1, \cdots, x_n$ onto this direction, the projected values are the principal component scores $z_{11}, \cdots, z_{n1}$ themselves.
After the first principal component Z_1 of the features has been determined, we can find the second principal component Z_2. The second principal component is the linear combination of $X_1, \cdots, X_p$ that has maximal variance out of all linear combinations that are uncorrelated with Z_1.



Principal Component Analysis

Once we have computed the principal components, we can plot them against each other in order to produce low-dimensional views of the data. For instance, we can plot the score vector Z_1 against Z_2, Z_1 against Z_3, Z_2 against Z_3, and so forth. Geometrically, this amounts to projecting the original data down onto the subspace spanned by ϕ_1, ϕ_2, and ϕ_3, and plotting the projected points.



USArrests

We illustrate the use of PCA on the *USArrests* data set. For each of the 50 states in the United States, the data set contains the number of arrests per 100, 000 residents for each of three crimes: *Assault*, *Murder*, and *Rape*. We also record *UrbanPop*.

	PC1	PC2
Murder	0.5358995	-0.4181809
Assault	0.5831836	-0.1879856
UrbanPop	0.2781909	0.8728062
Rape	0.5434321	0.1673186

we see that the first loading vector places approximately equal weight on Assault, Murder, and Rape, but with much less weight on UrbanPop. Hence this component roughly corresponds to a measure of overall rates of serious crimes. The second loading vector places most of its weight on UrbanPop and much less weight on the other three features. Hence, this component roughly corresponds to the level of urbanization of the state.

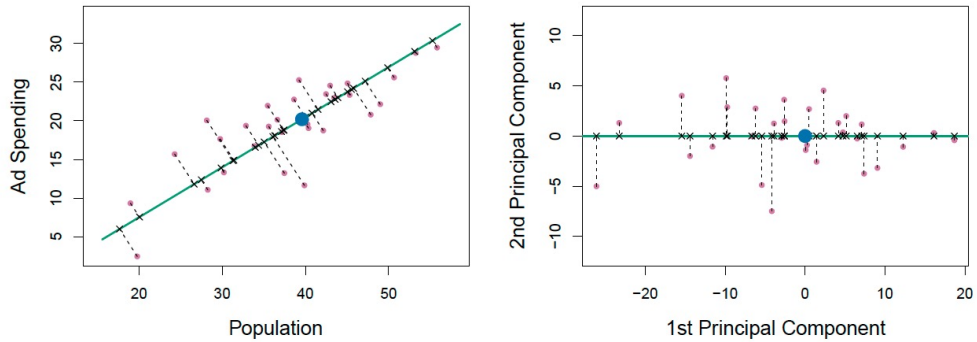
PCA Analysis

Overall, we see that the crime-related variables (Murder, Assault, and Rape) are located close to each other, and that the UrbanPop variable is far from the other three. This indicates that the crime-related variables are correlated with each other—states with high murder rates tend to have high assault and rape rates—and that the UrbanPop variable is less correlated with the other three.

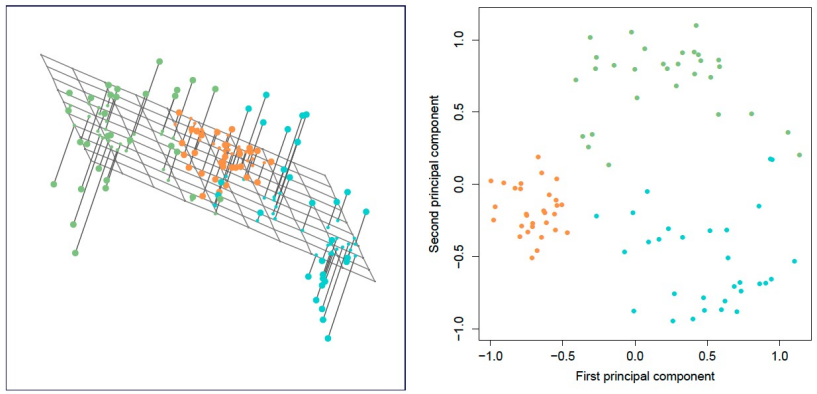
We can examine differences between the states via the two principal component score vectors. Our discussion of the loading vectors suggests that states with large positive scores on the first component, such as California, Nevada and Florida, have high crime rates, while states like North Dakota, with negative scores on the first component, have low crime rates. California also has a high score on the second component, indicating a high level of urbanization, while the opposite is true for states like Mississippi. States close to zero on both components, such as Indiana, have approximately average levels of both crime and urbanization.

Another Interpretation of PCA

There is also another interpretation of PCA: the first principal component vector defines the line that is as close as possible to the data.



The notion of principal components as the dimensions that are closest to the n observations extends beyond just the first principal component. For instance, the first two principal components of a data set span the plane that is closest to the n observations, in terms of average squared Euclidean distance.



Another Interpretation of PCA

Using this interpretation, together the first M principal component score vectors and the first M principal component loading vectors provide the best M-dimensional approximation to the ith observation x_{ij}

$$x_{ij} \approx \sum_{m=1}^{M} z_{im}\phi_{jm}.$$

We see in above figure that this two-dimensional representation of the three-dimensional data does successfully capture the major pattern in the data: the orange, green, and cyan observations that are near each other in three-dimensional space remain nearby in the two-dimensional representation. Similarly, we can summarize the 50 observations and 4 variables using just the first two principal component score vectors and the first two principal component loading vectors in the USArrests dataset.

PCA

We can now ask a natural question: how much of the information in a given data set is lost by projecting the observations onto the first few principal components? That is, how much of the variance in the data is not contained in the first few principal components?

More generally, we are interested in knowing the proportion of variance explained (PVE) by each principal component. The total variance present in a data set (assuming that the variables have been centered to have mean zero) is defined as

$$\sum_{j=1}^{p} Var(X_j) = \sum_{j=1}^{p} \frac{1}{n} \sum_{i=1}^{n} x_{ij}^2,$$

and the variance explained by the mth principal component is

$$\frac{1}{n} \sum_{i=1}^{n} z_{im}^2 = \frac{1}{n} \sum_{i=1}^{n} \Big(\sum_{j=1}^{p} \phi_{jm} x_{ij} \Big)^2.$$

Therefore, the PVE of the mth principal component is given by

$$\frac{\frac{1}{n} \sum_{i=1}^{n} z_{im}^2}{\sum_{j=1}^{p} \frac{1}{n} \sum_{i=1}^{n} x_{ij}^2} = \frac{\sum_{i=1}^{n} \Big(\sum_{j=1}^{p} \phi_{jm} x_{ij} \Big)^2}{\sum_{j=1}^{p} \sum_{i=1}^{n} x_{ij}^2}$$

It turns out that the variance of the data can be decomposed into the variance of the first M principal components plus the mean squared error of this M-dimensional approximation

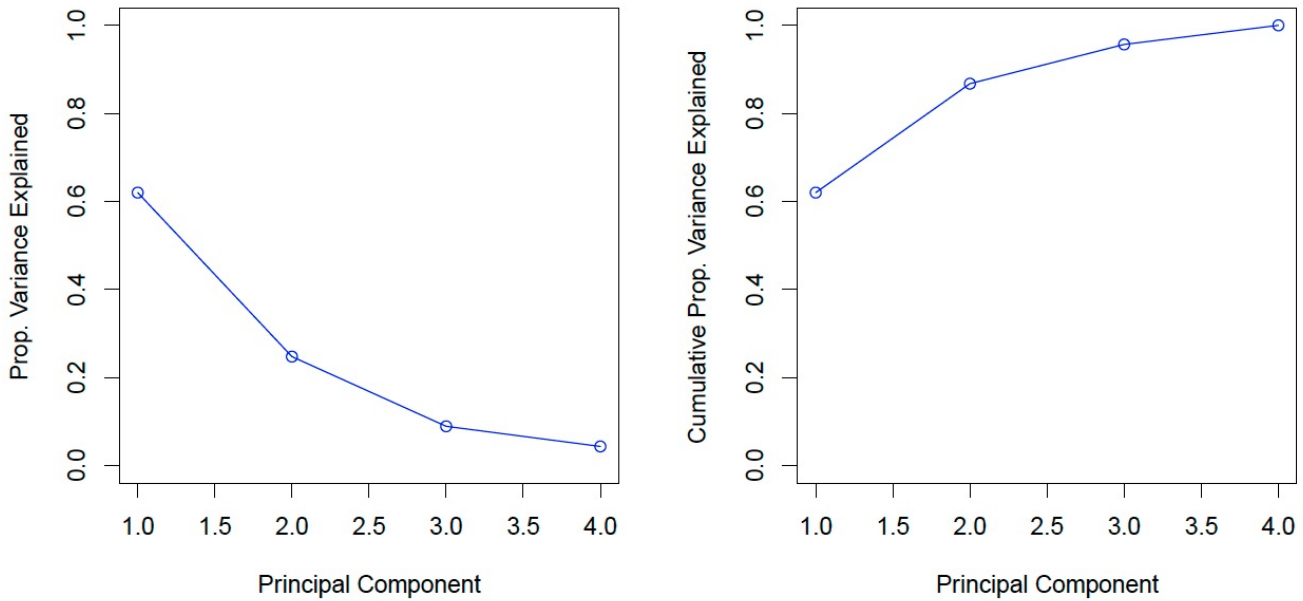
$$\sum_{j=1}^{p} \frac{1}{n} \sum_{i=1}^{n} x_{ij}^2 = \sum_{m=1}^{M} \frac{1}{n} \sum_{i=1}^{n} z_{im}^2 + \underbrace{\frac{1}{n} \sum_{j=1}^{p} \sum_{i=1}^{n} (x_{ij} - \sum_{m=1}^{M} z_{im} \phi_{jm})^2}_{\text{MSE of M-dimensional approximation}}$$

Moreover, we can see that the PVE of M principal components can be expressed as

$$1 - \frac{\sum_{j=1}^{p} \sum_{i=1}^{n} (x_{ij} - \sum_{m=1}^{M} z_{im} \phi_{jm})^2}{\sum_{j=1}^{p} \sum_{i=1}^{n} x_{ij}^2} = 1 - \frac{RSS}{TSS}.$$

This means that we can interpret the PVE as the R^2 of the approximation for X given by the first M principal components.

PVE



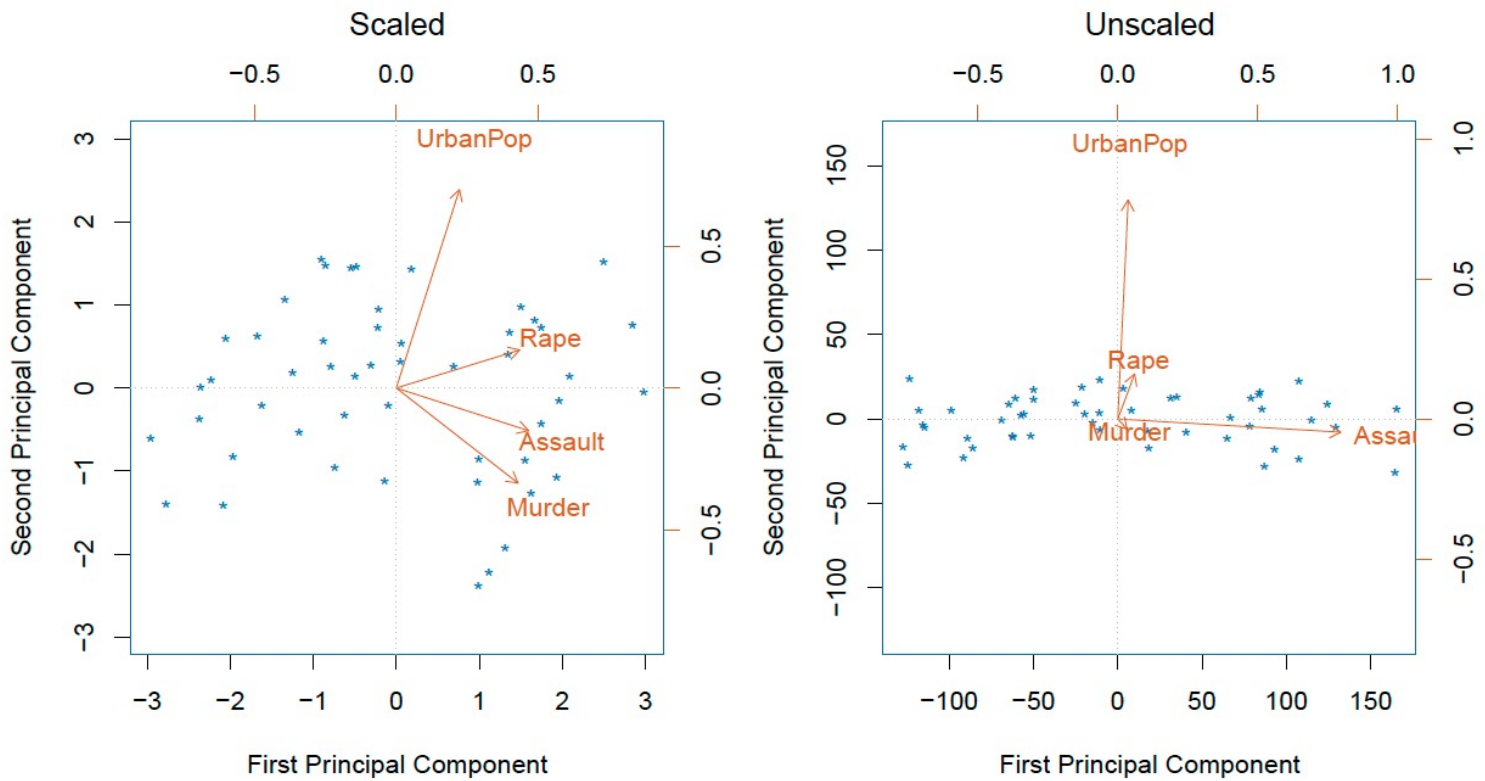
In the USArrests data, the first principal component explains 62.0 % of the variance in the data, and the next principal component explains 24.7 % of the variance. Together, the first two principal components explain almost 87 % of the variance in the data, and the last two principal components explain only 13 % of the variance.

More on PCA: Scaling the Variables

We have already mentioned that before PCA is performed, the variables should be centered to have mean zero. Furthermore, the results obtained when we perform PCA will also *depend on whether the variables have been individually scaled* .

Why does it matter that we scaled the variables? In USArrests data, the variables are measured in different units; Murder, Rape, and Assault are reported as the number of occurrences per 100, 000 people, and UrbanPop is the percentage of the state's population that lives in an urban area. These four variables have variances of 18.97, 87.73, 6945.16, and 209.5, respectively. Consequently, if we perform PCA on the unscaled variables, then the first principal component loading vector will have a very large loading for Assault, since that variable has by far the highest variance.

Scaleing the Variables



Left result was obtained after scaling each of the variables to have standard deviation one. In certain settings, however, the variables may be measured in the same units. In this case, we might not wish to scale the variables to have standard deviation one before performing PCA. For instance, suppose that the variables in a given data set correspond to expression levels for p genes. Then since expression is measured in the same "units" for each gene, we might choose not to scale the genes to each have standard deviation one.

How Many Principal Components to Use

In general, an $n \times p$ data matrix X has $min(n-1, p)$ distinct principal components. However, we usually are not interested in all of them; rather, we would like to use just the first few principal components in order to visualize or interpret the data. In fact, we would like to use the smallest number of principal components required to get a good understanding of the data. How many principal components are needed? Unfortunately, there is no single (or simple!) answer to this question.

We typically decide on the number of principal components required to visualize the data by examining a scree plot. We choose the smallest number of principal components that are required in order to explain a sizable amount of the variation in the data. This is done by eyeballing the scree plot, and looking for a point at which the proportion of variance explained by each subsequent principal component drops off. This drop is often referred to as an *elbow* in the scree plot.

Supplementing Supervised Learning

We can perform regression using the principal component score vectors as features. In fact, many statistical techniques, such as regression, classification, and clustering, can be easily adapted to use the $n \times M$ matrix whose columns are the first $M \ll p$ principal component score vectors, rather than using the full $n \times p$ data matrix. This can lead to less noisy results, since it is often the case that the signal (as opposed to the noise) in a data set is concentrated in its first few principal components. When taking Z's as new features of a supervised learning, we essentially have the number of principal components to use as a tuning parameter. It then can be selected via cross-validation or a related approach.

PCA on Cancer Dataset

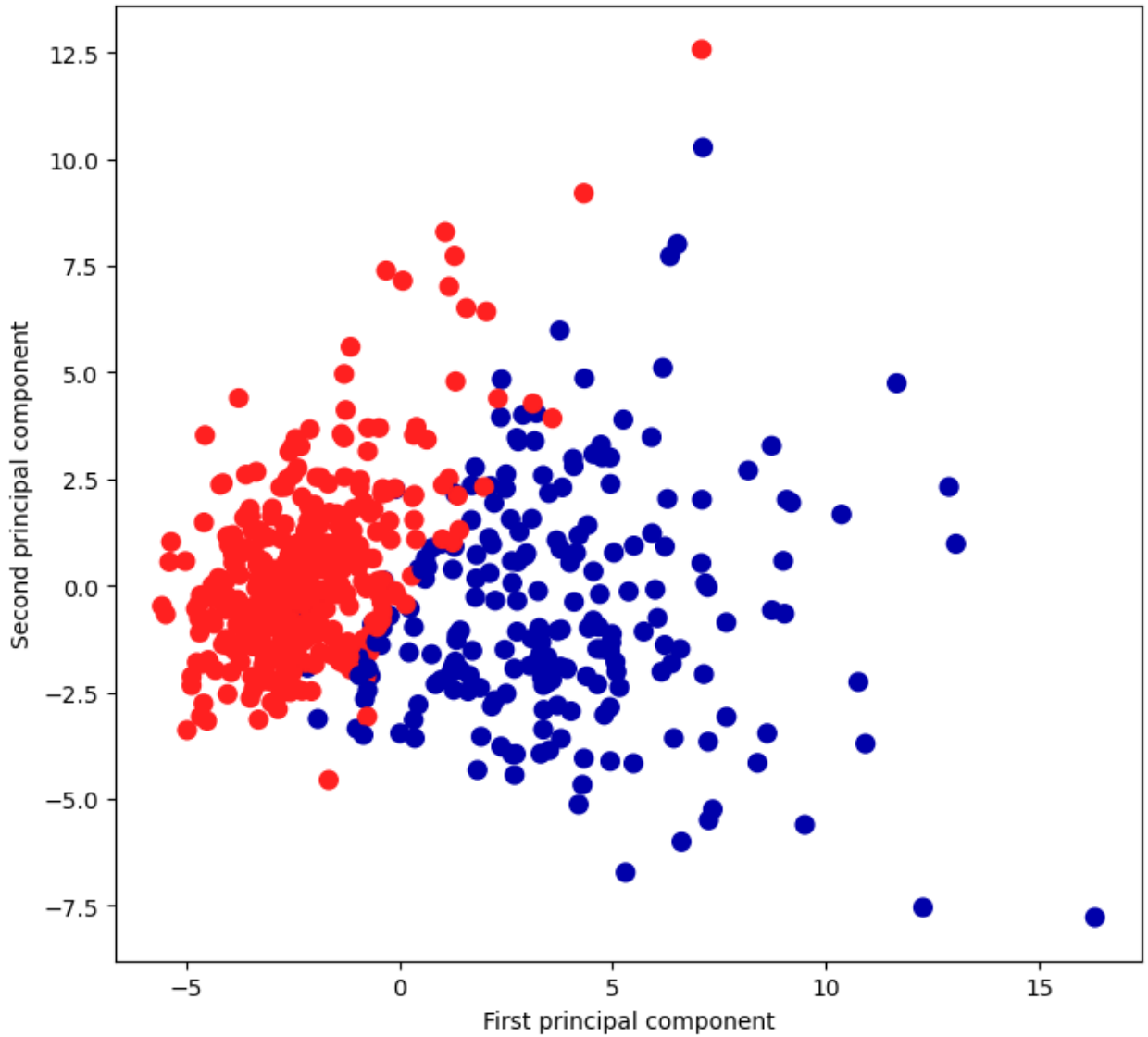
We can find the first two principal components, and visualize the data in this new, two-dimensional space, with a single scatter-plot.

- Before we apply PCA, we scale our data so that each feature has unit variance using *StandardScaler*.
- Learning the PCA transformation and applying it is as simple as applying a preprocessing transformation. We instantiate the PCA object, find the principal components by calling the fit method, and then apply the rotation and dimensionality reduction by calling transform.
- By default, PCA only rotates (and shifts) the data, but keeps all principal components. To reduce the dimensionality of the data, we need to specify how many components we want to keep when creating the PCA object:

PCA

```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(cancer.data)
X_scaled = scaler.transform(cancer.data)
from sklearn.decomposition import PCA
# keep the first two principal components of the data
pca = PCA(n_components=2)
# fit PCA model to beast cancer data
pca.fit(X_scaled)
# transform data onto the first two principal components
X_pca = pca.transform(X_scaled)
print("Original shape: %s" % str(X_scaled.shape))
print("Reduced shape: %s" % str(X_pca.shape))
Original shape: (569, 30)
Reduced shape: (569, 2)
```

We can now plot the first two principal components



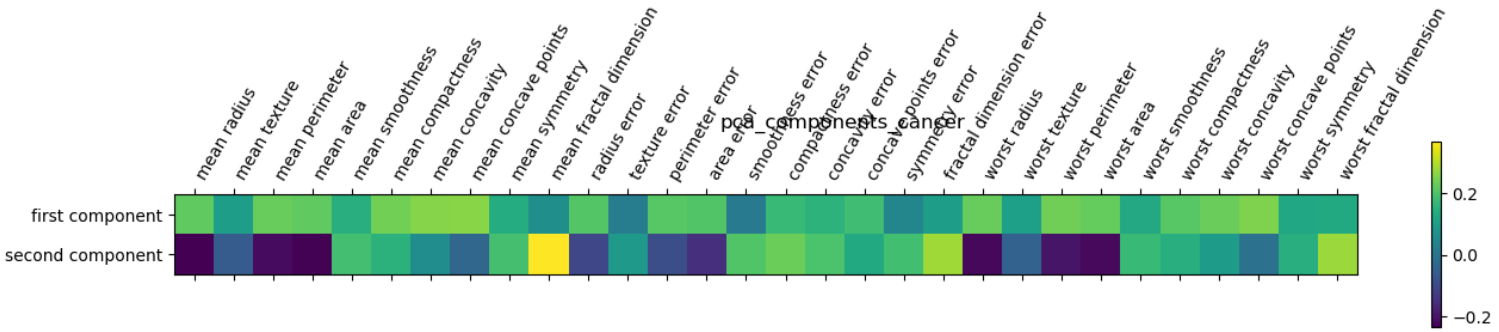
Interpretation

It is important to note is that PCA is an unsupervised method, and does not use any class information when finding the rotation. It simply looks at the correlations in the data. For the scatter plot above, we plotted the first principal component against the second principal component, and then used the class information to color the points. You can see that the two classes separate quite well in this two-dimensional space. This can lead us to believe that even a linear classifier (that would learn a line in this space) could do a reasonably good job at distinguishing the two classes. We can also see that the malignant (red) points are more spread-out than the benign (blue) points

Downside of PCA

A downside of PCA is that the two axes in the plot above are often not very easy to interpret. The principal components are combinations of the original features. However, these combinations are usually very complex.

```
print(pca.components_)
plt.matshow(pca.components_, cmap='viridis')
plt.yticks([0, 1], ["first component", "second component"])
plt.colorbar()
plt.xticks(range(len(cancer.feature_names)),
cancer.feature_names, rotation=60, ha='left');
plt.suptitle("pca_components_cancer")
```



That means that there is a general correlation between all features. As one measurement is high, the others are likely to be high as well. The second component has mixed signs, and both of the components involve all of the 30 features. This mixing of all features is what makes explaining the axes in Figure pca_components_cancer above so tricky.

Eigenfaces for feature extraction

Another application of PCA that we mentioned above is feature extraction. The idea behind feature extraction is that it is possible to find a representation of your data that is better suited to analysis than the raw representation you were given. A great example of an application when feature extraction if helpful is with images. Images are usually stored as red, green and blue intensities for each pixel. But images are made up of many pixels, and only together are they meaningful; objects in images are usually made up of thousands of pixels. We will give a very simple application of feature extraction on images using PCA, using face images from the "labeled faces in the wild" dataset. This dataset contains face images of celebrities downloaded from the internet, and it includes faces of politicians, singers, actors and athletes from the early 2000s.

Eigenfaces for feature extraction

A common task in face recognition is to ask if a previously unseen face belongs to a known person from a database. This has applications in photo collection, social media and security. One way to solve this problem would be to build a classifier where each person is a separate class. A simple solution is to use a one-nearest-neighbor classifier which looks for the most similar face image to the face you are classifying.

```
mask = np.zeros(people.target.shape, dtype=bool)
for target in np.unique(people.target):
 mask[np.where(people.target == target)[0][:50]] = 1
X_people = people.data[mask]
y_people = people.target[mask]
# scale the grey-scale values to be between 0 and 1
# instead of 0 and 255 for better numeric stability:
X_people = X_people / 255.
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
# split the data in training and test set
X_train, X_test, y_train, y_test = train_test_split(
   X_people, y_people, stratify=y_people, random_state=0)
# build a KNeighborsClassifier with using one neighbor:
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
knn.score(X_test, y_test)
0.21511627906976744
pca = PCA(n_components=100, whiten=True).fit(X_train)
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train_pca, y_train)
knn.score(X_test_pca, y_test)
0.3023255813953488
```

Missing Values and Matrix Completion

Often datasets have missing values, which can be a nuisance. For example, suppose that we wish to analyze the USArrests data, and discover that 20 of the 200 values have been randomly corrupted and marked as missing. Unfortunately, the statistical learning methods that we have seen in this book cannot handle missing values. How should we proceed?

- We could remove the rows that contain missing observations and perform our data analysis on the complete rows. But this seems wasteful, and depending on the fraction missing, unrealistic.
- Alternatively, if x_{ij} is missing, then we could replace it by the mean of the jth column (using the non-missing entries to compute the mean). Although this is a common and convenient strategy, often we can do better by exploiting the correlation between the variables.
- We show how principal components can be used to *impute* the missing values, through a process known as *matrix completion*. The completed matrix can then be used in a statistical learning method, such as linear regression.
- This approach for imputing missing data is appropriate if the missingness is random.

Matrix Completion

Sometimes data is missing by necessity. If we can impute the missing values well, then we will have an idea of what each customer will think of movies they have not yet seen. Hence matrix completion can be used to power *recommender systems*.
We have showed that the first M principal component score and loading vectors provide the "best" approximation to the data matrix X, in the sense of

$$\min_{A\in\mathbb{R}^{n\times M},B\in\mathbb{R}^{p\times M}} \left\{ \sum_{j=1}^{p}\sum_{i=1}^{n}\left(x_{ij}-\sum_{m=1}^{M}a_{im}b_{jm}\right)^2 \right\}.$$

Suppose that some of the observations x_{ij} are missing. We now show how one can both impute the missing values and solve the principal component problem at the same time.

$$\min_{A\in\mathbb{R}^{n\times M},B\in\mathbb{R}^{p\times M}} \left\{ \sum_{(i,j)\in\mathcal{O}}\left(x_{ij}-\sum_{m=1}^{M}a_{im}b_{jm}\right)^2 \right\}. \tag{1}$$

where $\mathcal{O}$ is the set of all observed pairs of indices (i,j).

Matrix Completion

Once we solve this problem:

- we can estimate a missing observation $\hat{x}_{ij} = \sum_{m=1}^{M} \hat{a}_{im}\hat{b}_{jm}$.
- we can (approximately) recover the M principal component scores and loadings, as we did when the data were complete.

It turns out that solving Equation (1) exactly is difficult, unlike in the case of complete data: the eigen decomposition no longer applies. But the simple iterative approach typically provides a good solution. *Iterative Algorithm for Matrix Completion*

1. Create a complete data matrix $\widetilde{X}$ of dimension $n \times p$ of which the (i,j) element equals x_{ij} if $(i,j) \in \mathcal{O}$ and equals $\bar{x}_j$ if $(i,j) \notin \mathcal{O}$.
2. Repeat steps (a)–(c) until the objective Equation (3) fails to decrease:
   (a) Solve

$$\min_{A \in \mathbb{R}^{n \times M}, B \in \mathbb{R}^{p \times M}} \left\{ \sum_{j=1}^{p} \sum_{i=1}^{n} \left(\tilde{x}_{ij} - \sum_{m=1}^{M} a_{im} b_{jm} \right)^2 \right\}, \tag{2}$$

   by computing the principal components of $\widetilde{X}$.

   (b) For each element $(i,j) \notin \mathcal{O}$, set $\tilde{x}_{ij} = \sum_{m=1}^{M} \hat{a}_{im}\hat{b}_{jm}$.

   (c) Compute the objective

$$\min_{A \in \mathbb{R}^{n \times M}, B \in \mathbb{R}^{p \times M}} \left\{ \sum_{(i,j) \in \mathcal{O}} \left(x_{ij} - \sum_{m=1}^{M} \hat{a}_{im} \hat{b}_{jm} \right)^2 \right\}. \tag{3}$$

Matrix Completion

We illustrate *Iterative Algorithm for Matrix Completion* on the USArrests data. There are $p = 4$ variables and $n = 50$ observations (states). We first standardized the data so each variable has mean zero and standard deviation one. We then randomly selected 20 of the 50 states, and then for each of these we randomly set one of the four variables to be missing. Thus, 10% of the elements of the data matrix were missing. We applied the Iterative Algorithm with $M = 1$ principal component.