CITS5508 Machine Learning

Débora Corrêa (Unit Coordinator and Lecturer)
UWA

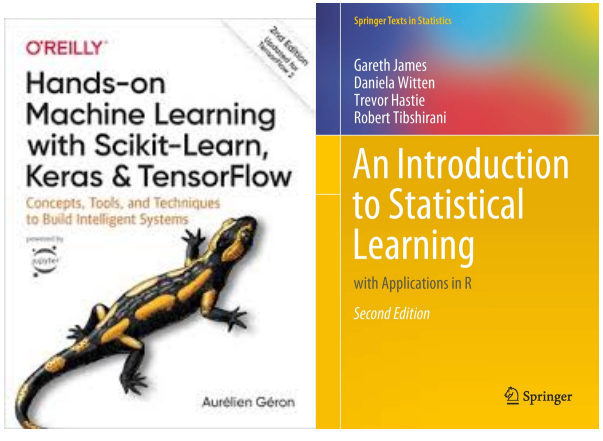
2023

Today

Dimensionality Reduction and Principal Component Analysis.

Hands-on Machine Learning with Scikit-Learn & TensorFlow (chapter 8)

An Introduction to Statistical Learning (chapter 12)



Supervised and Unsupervised Learning

In supervised learning: we have access to n features $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n$ measured on m observations. The goal is to predict an associate response variable $\hat{y}$ (that is, also measured on those m observations) using $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n$.

In unsupervised learning: we only have access to $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n$ measured on m observations. The goal is to discover patterns and interesting things about measurements on $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n$. For instance:

- How can we visualise the data effectively?
- Can we find subgroups among the observations?
- Can we find subgroups among the features and use it for dimensionality reduction?

Unsupervised Learning

Unsupervised learning is more challenging.

- Tends to be more subjective.
- There is no simple goal for the analysis (e.g. prediction of a response).
- Supervised learning is a well-understood area. There are developed set of tools and clear understanding of how to assess the quality of the results.
- It is more challenging to assess the results from unsupervised learning models.

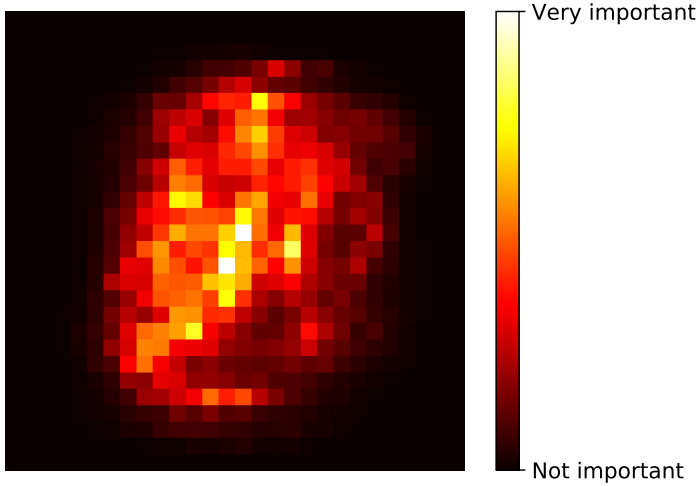
Unsupervised learning is often used as part of exploratory data analysis. However, the field is growing importance in a number of fields. For example, cancer research, recommendation systems, etc.

The Curse of Dimensionality

Training instances in ML often have thousands or even millions of features. This means that it may take a long time to train the ML algorithm, and it might be difficult to find a good solution.

This problem is referred to as the *curse of dimensionality*.

In real-world problems is often possible to reduce the number of features considerably.



For example, consider the MNIST images, we could completely drop the boundary pixels from the training set without losing much information for classification.

Data visualisation

How to visualise m training instances with measurements on a set of n features, $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$?

- Two-dimensional scatterplots of the data, each containing m observation's measurements on two of the features would require $\binom{n}{2} = n(n-1)/2$ of such scatterplots.
- Infeasible for large n. Besides, each scatterplot will contain just a small fraction of the total information present in the data set. Hence, none of them will be much informative.
- Preferably, we would like to find a low-dimension representation of the data that captures as much of the information as possible.
- PCA provides a tool to achieve this aim.

Dimensionality Reduction

We discussed methods that control the variance mainly in two different ways: either by using a subset of the original variables, or by shrinking their coefficients towards zero. These methods use the original features $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n$.

Dimension reduction approaches transform the features, e.g. by finding good linear combinations of the original features.

Let $Z_1, Z_2, ..., Z_Q$ represent $Q < n$ linear combinations of the original n features for some constants $\phi_{1q}, \phi_{2q}, ..., \phi_{nq}$, $q = 1, ..., Q$. That is,

$$Z_q = \sum_{j=1}^{n} \phi_{jq} \boldsymbol{x}_j$$

The choice of $Z_1, Z_2, ..., Z_Q$, or equivalently, the selection of the ϕ_{jq}'s can be achieved in different ways. We will consider PCA to perform this task.

Dimensionality Reduction

Information is lost after reducing the dimensionality of your data.

Although dimensionality reduction will speed up training, it may make your system perform slightly worse, and your pipeline more complex and harder to maintain.

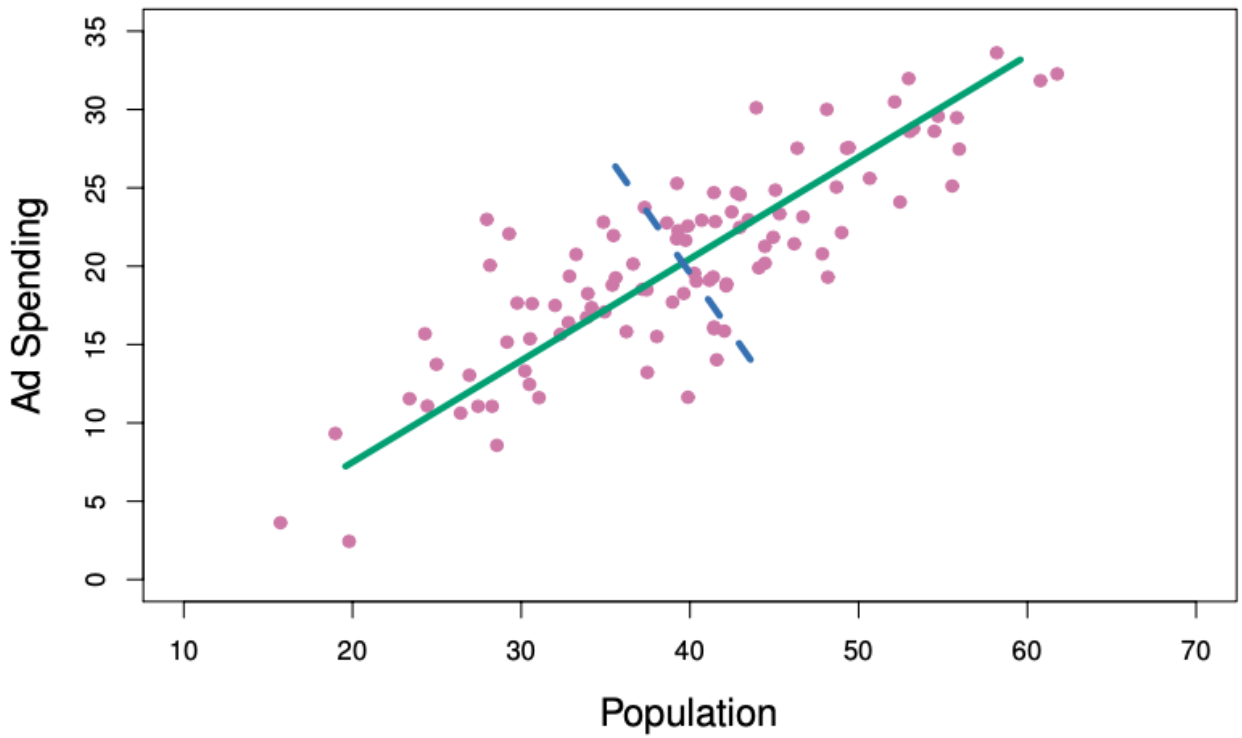
In some cases, dimensionality reduction of the training data may filter out unnecessary details, resulting in higher performance. However in general, it won't increase performance, it will just speed up training.

Usually we try first with the original set of features.

Dimensionality reduction is also useful for data visualization (essential to communicate your conclusions to decision makers who will use your results).

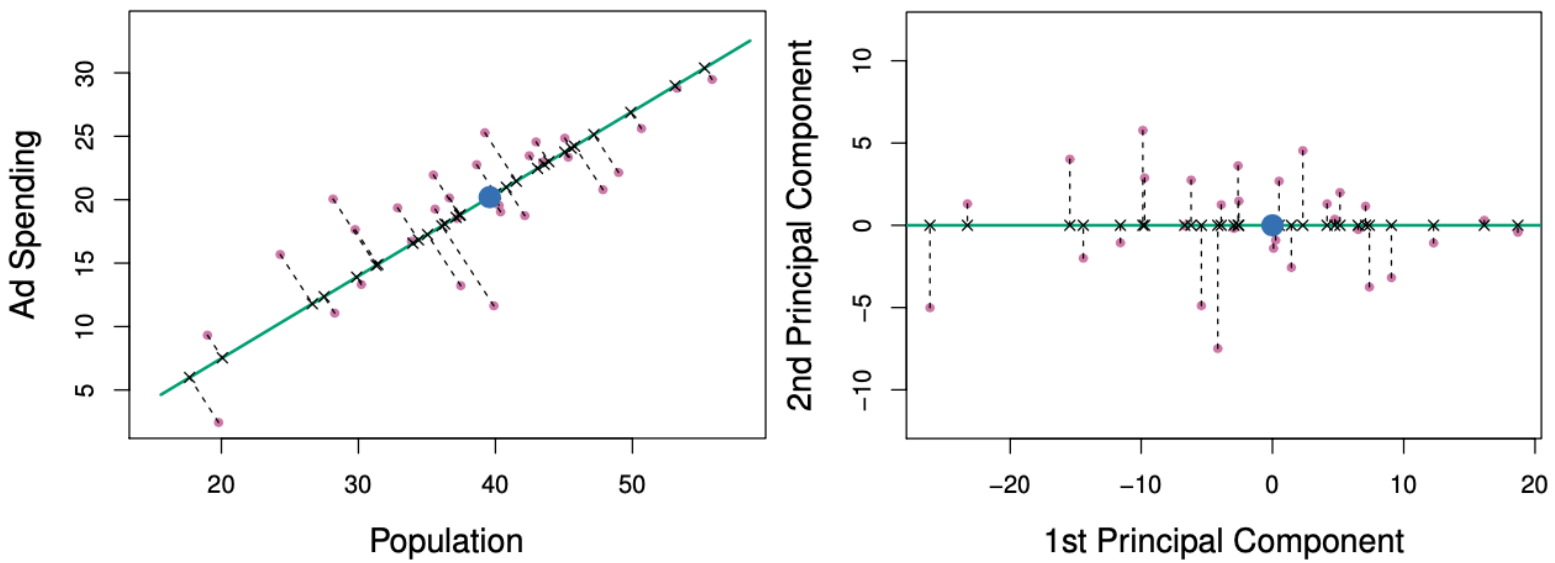
Principal Component Analysis (PCA)

PCA is a linear dimensionality reduction technique to reduce the dimensionality of a data set, while retaining as much as possible of the variation present in the data.



Principal Component Analysis (PCA)

In the example the green solid line represents the first principal component direction of the data. We can see by eye that this is the direction along which there is the greatest variability in the data. The first principal component direction of the data is that along which the observations vary the most.



Principal Component Analysis (PCA)

The first principal component is also a linear combination of the variables and therefore can be represented as

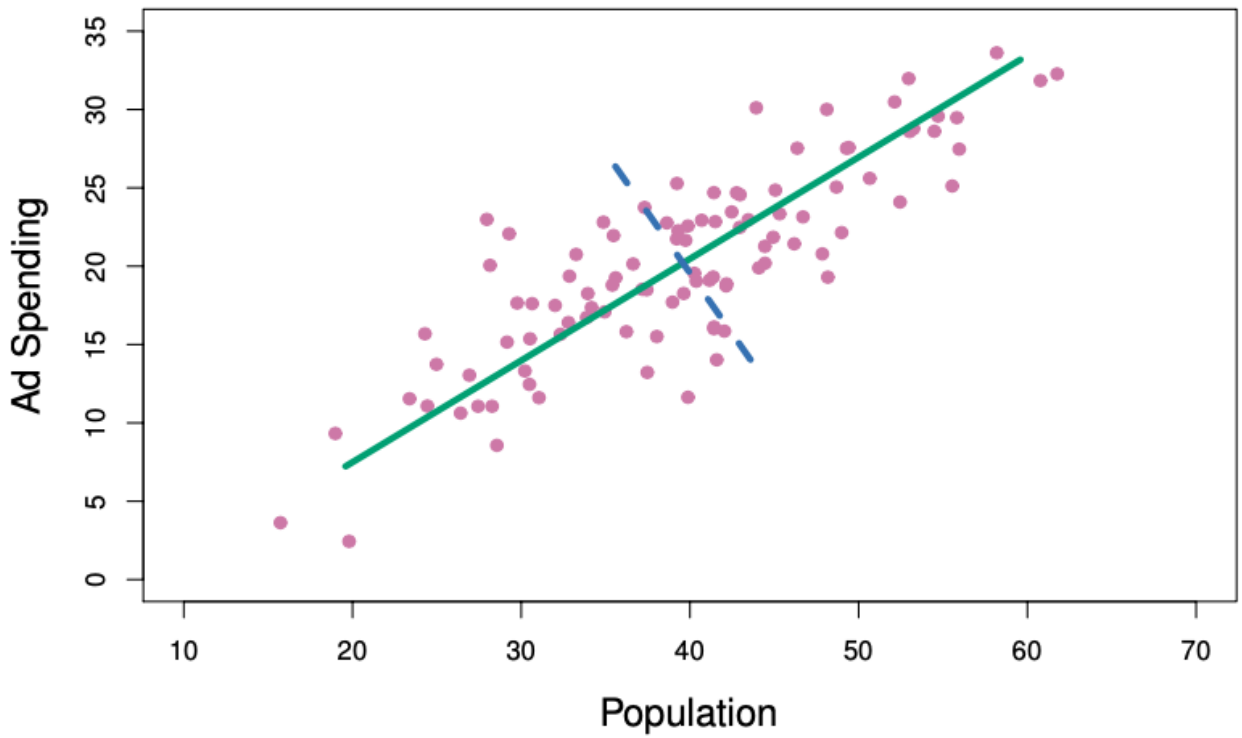
$$Z_1 = 0.839 \times (\text{pop} - \overline{\text{pop}}) + 0.544 \times (\text{ad} - \overline{\text{ad}})$$

where $\overline{\text{pop}}$ indicates the mean of all `population` values in this data set, and $\overline{\text{ad}}$ indicates the mean of all `advertising` spending.

Idea: out of every possible linear combination of `pop` and `ad` such that $\phi_{11}^2 + \phi_{21}^2 = 1$, this particular linear combination yields the highest variance: i.e. this is the linear combination for which $\text{Var}(\phi_{11} \times (\text{pop} - \overline{\text{pop}}) + \phi_{21} \times (\text{ad} - \overline{\text{ad}}))$ is maximized.

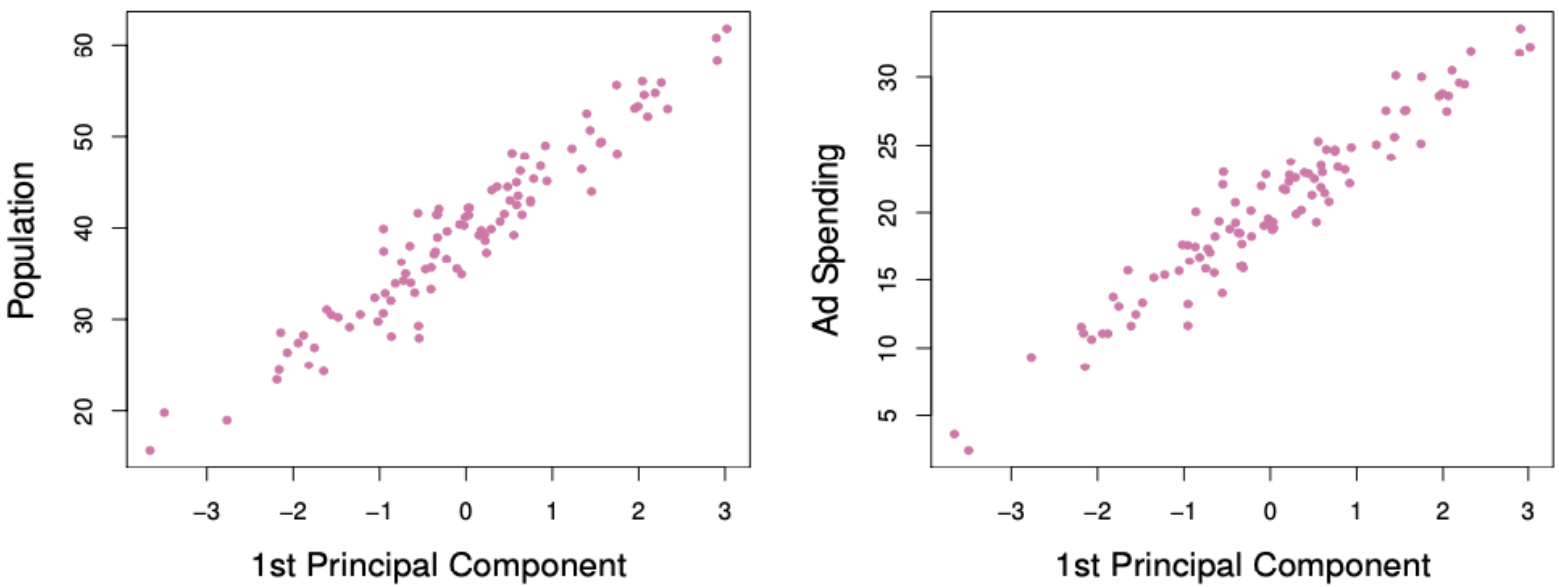
Principal Component Analysis (PCA)

The first principal component vector defines the line that is as close as possible to the data, which is the same as saying the first principal component line minimises the sum of the squared perpendicular distances between each point and the line.



Principal Component Analysis (PCA)

As expected, there is a strong relationship between the first principal component and the two features. In other words, the first principal component appears to capture most of the information contained in the `pop` and `ad` features.



Principal Component Analysis (PCA)

We can construct n distinct principal components.

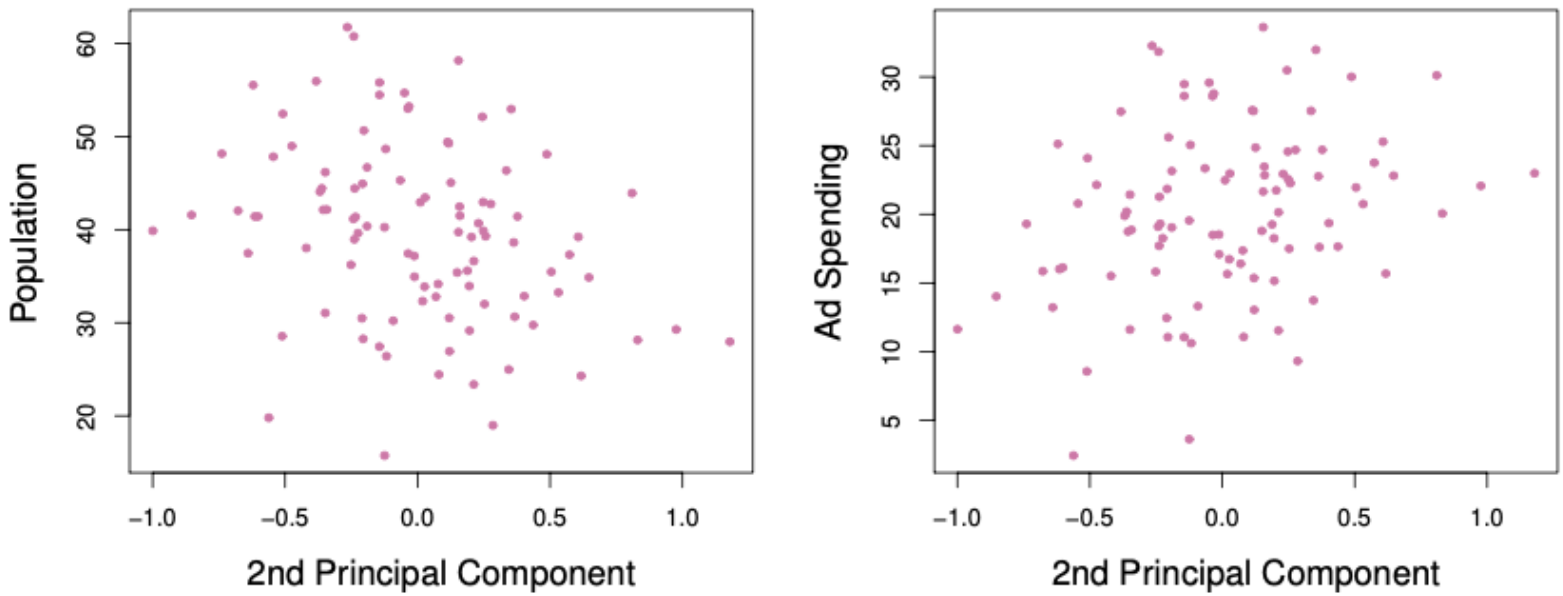
The second principal component Z_2 is a linear combination of the variables that is uncorrelated with Z_1, and has largest variance subject to this constraint.

The zero correlation condition of Z_1 with Z_2 is equivalent to the condition that the direction must be perpendicular, or orthogonal,to the first principal component direction.

$$Z_2 = 0.544\,x\,(\text{pop} - \overline{\text{pop}}) \; - \; 0.839\,x\,(\text{ad} - \overline{\text{ad}})$$

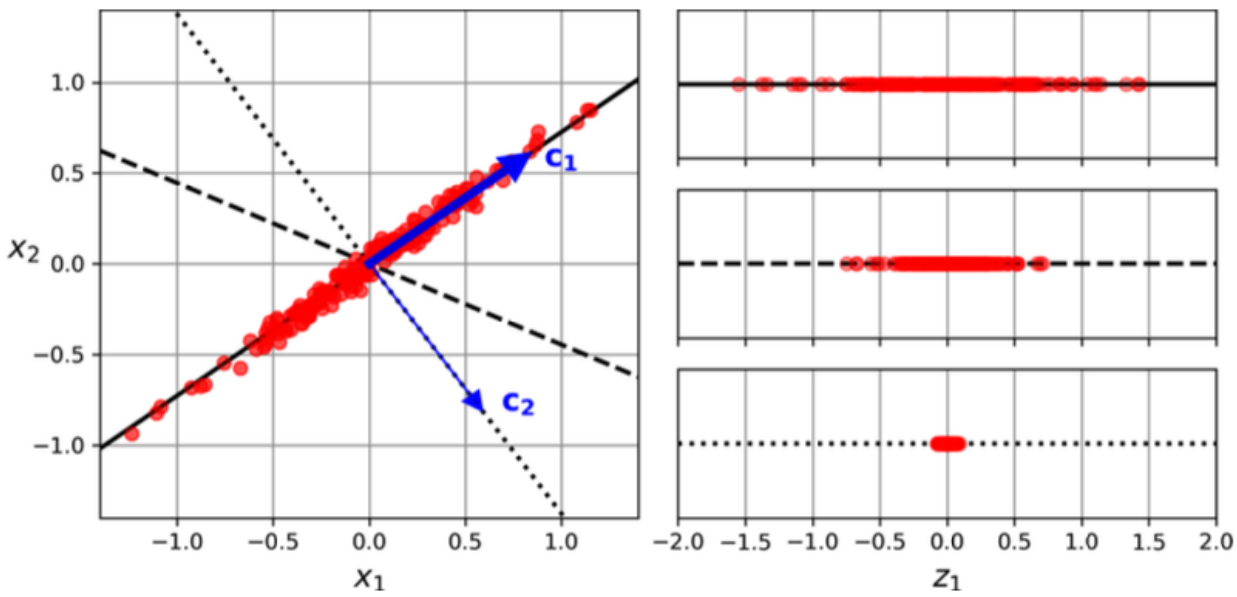
Principal Component Analysis (PCA)

There is much less strong relationship between the second principal component and these two predictors.



Principal Component Analysis (PCA)

We use PCA to find the projection matrix that would preserve the variance of the data as much as possible. That is, PCA first identifies the hyperplane that lies closest to the data, and then it projects the data onto it.



Selecting the subspace on which to project

Principal Component Analysis (PCA)

For a given n-dimensional dataset, PCA identifies the first axis that accounts for the largest variance in the dataset.

Then, it finds the the second axis, orthogonal to the first axis and that has the largest amount of the remaining variance.

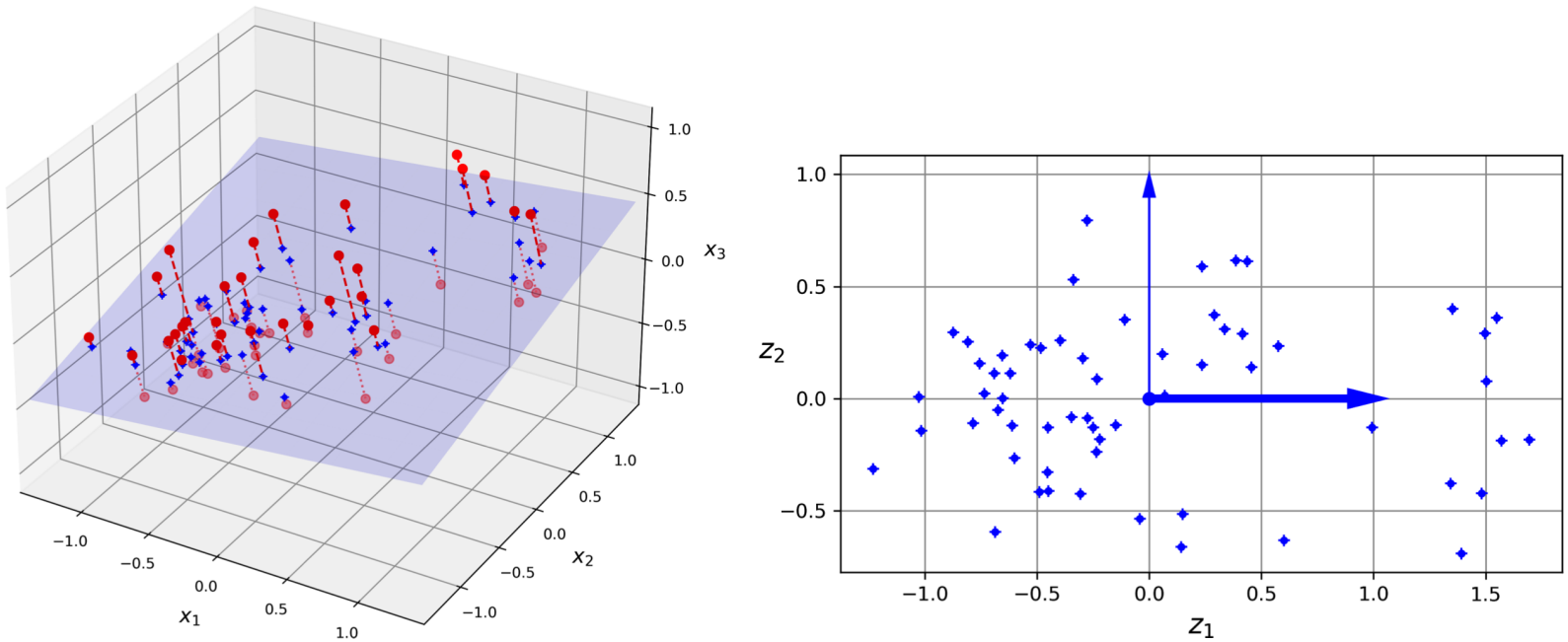
The third axis that is orthogonal to the first two axes and that has the third largest variance, and so on (PCA finds as many axes as the number of dimensions in the dataset).

The i^{th} axis is called the i^{th} *principal component* of the data.

Thus, in dimensionality reduction, to determine the number of principal axes to keep is the same as to determine how many principal components to have after data projection.

Principal Component Analysis (PCA)

After the projection, the first PC corresponds to the Z_1 axis, and the second PC corresponds to the Z_2 axis.



A 3D dataset lying close to a 2D subspace (left) and The new 2D dataset after projection (right)

PCA - How to find the principal axes?

We are interested in the variance, and we assume each of the variables $\boldsymbol{x}_i$ has been centered to have mean zero. The first principal component of a set of features $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n$ is computed by looking for the linear combination of the feature values of the form

$$Z_1 = \phi_{11}\boldsymbol{x}_1 + \phi_{21}\boldsymbol{x}_2 + ... + \phi_{n1}\boldsymbol{x}_n$$

That is,

$$Z_{i1} = \phi_{11}x_{i1} + \phi_{21}x_{i2} + ... + \phi_{n1}x_{in}$$

that has the largest sample variance, subject to the constraint that $\sum_{j=1}^{n} \phi_{j1}^2 = 1$. This is equivalent to solving the optimisation problem

$$\text{maximise}_{\phi_{11},...,\phi_{n1}} = \left\{ \frac{1}{m} \sum_{i=1}^{m} \left(\sum_{j=1}^{n} \phi_{j1} x_{ij} \right)^2 \right\} \quad \text{subject to} \sum_{j=1}^{n} \phi_{j1}^2 = 1$$

PCA - How to find the principal axes?

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 $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n$ that has maximal variance out of all linear combinations that are `uncorrelated` with Z_1. The second principal component scores $z_{12}, z_{22}, ..., z_{m2}$ take the form

$$Z_{i2} = \phi_{12} x_{i1} + \phi_{22} x_{i2} + ... + \phi_{n2} x_{in}$$

where ϕ_2 is the second principal component loading vector, with elements $\phi_{12}, \phi_{22}, ..., \phi_{n2}$. Constraining Z_2 to be uncorrelated with Z_1 is equivalent to constraining the direction of ϕ_2 to be orthogonal (perpendicular) to the direction of ϕ_1.

PCA - How to find the principal axes?

There is a standard matrix factorization technique called *Singular Value Decomposition* (*SVD*) that can decompose the training set matrix $\mathbf{X}$ (m x n matrix) into the matrix multiplication of three matrices $\mathbf{U}$, $\mathbf{\Sigma}$, and $\mathbf{V}$:

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T.$$

If each row of your $\mathbf{X}$ matrix stores one data point, then each column of $\mathbf{V}$ stores the unit vectors that define all the principal components:

$$\mathbf{V} = \begin{bmatrix} | & | & & | \\ \mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_n \\ | & | & & | \end{bmatrix}$$

Note: $\mathbf{X}$ is $m \times n$; $\mathbf{U}$ is $m \times m$, $\mathbf{\Sigma}$ is $m \times n$; $\mathbf{V}$ is $n \times n$.
(m is the total number of data points; $m >> n$).

SVD() in Numpy

Numpy has a `svd()` function which can be used to obtain all the principal axes from the training set.

Example:

```
import numpy as np

X = [...] # create a small 3D dataset
X_centered = X - X.mean(axis=0)

U, s, Vt = np.linalg.svd(X_centered)

# extracting the two unit vectors that define the first two PCs
c1 = Vt[0]
c2 = Vt[1]
```

Notes:

- You must centre the data so that the origin is at the mean.

How to project the data?

We generally want to reduce the dimension from n to d by projecting the data onto the hyperplane defined by the first d principal axes while preserving as much variance of the data as possible.

To do so, define an $n \times d$ matrix $\mathbf{W}_d$ to contain the first d columns of matrix $\mathbf{V}$. The projection is then simply:

$$\mathbf{X}_{d\text{-proj}} = \mathbf{X}\mathbf{W}_d$$

($\mathbf{X}$ is $m \times n$, $\mathbf{W}_d$ is $n \times d$, $\mathbf{X}_{d\text{-proj}}$ is $m \times d$)

In Python:

```
W2 = Vt.T[:, :2]
X2D = X_centered.dot(W2)
```

PCA with Scikit-Learn

Instead of Numpy's `svd()` function, you can also use Scikit-Learn's `PCA` class which uses SVD to implement PCA:

```
from sklearn.decomposition import PCA

pca = PCA(n_components = 2)
X2D = pca.fit_transform(X)
```

You can access the PC axes using the `components_` attribute which holds the transpose of $\mathbf{W}_d$: it contains one row for each of the first d principal components.

The line `X2D = pca.fit_transform(X)` above handles the projection. The output `X2D` is the 2D projection of the original 3D data `X` in the previous example.

Scikit-Learns PCA classes take care of centering the data for you.

Explained variance ratio

It is important to find out the proportion of the dataset's variance that lies along each principal component axis. This is the *explained variance ratio*, available via the `explained_variance_ratio_` variable:

```
>>> pca.explained_variance_ratio_
array([0.7578477 , 0.15186921])
```

This output tells us that about 76% of the datasets variance lies along the first PC, and about 15% lies along the second PC.

Adding up the above numbers shows that the third PC that is removed from dimensionality reduction takes up only around 9% of the variance.

Explained variance ratio

The `total variance` in the data set (given the features have been centred to have mean zero) is defined as

$$\sum_{j=1}^{n} Var(\boldsymbol{x}_j) = \sum_{j=1}^{n} \frac{1}{m} \sum_{i=1}^{m} x_{ij}^2$$

and the variance explained by the mth principal component is

$$\frac{1}{m} \sum_{i=1}^{m} z_{iq}^2 = \frac{1}{m} \sum_{i=1}^{m} \left(\sum_{j=1}^{n} \phi_{jq} x_{ij} \right)^2$$

Then, the proportion of variance explained for the mth principal component is given by:

$$\frac{\sum_{i=1}^{m} z_{iq}^2}{\sum_{j=1}^{n} \sum_{i=1}^{m} x_{ij}^2} = \frac{\sum_{i=1}^{m} \left(\sum_{j=1}^{n} \phi_{jq} x_{ij} \right)^2}{\sum_{j=1}^{n} \sum_{i=1}^{m} x_{ij}^2}$$

Choosing the right number of dimensions

It is generally preferable to choose the number of dimensions that add up to a sufficiently large portion of the variance (e.g., 95%).

Example using the MNIST dataset:

```
from sklearn.datasets import fetch_openml

mnist = fetch_openml('mnist_784', as_frame=False)
X_train, y_train = mnist.data[:60_000], mnist.target[:60_000]
X_test, y_test = mnist.data[60_000:], mnist.target[60_000:]

pca = PCA()
pca.fit(X_train)
cumsum = np.cumsum(pca.explained_variance_ratio_)
d = np.argmax(cumsum >= 0.95) + 1 # d equals 154
```

We could then set `n_components=d` and run PCA again, or:

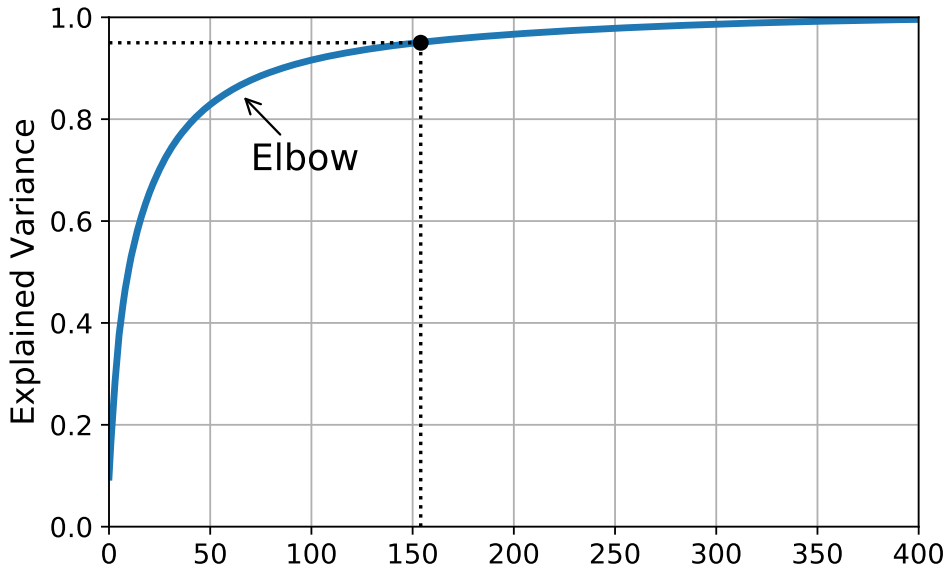
```
pca = PCA(n_components=0.95) # the ratio of variance to preserve
X_reduced = pca.fit_transform(X_train)
```

Choosing the right number of dimensions

The `n_components_` attribute contains the actual number of components determined during training:

```
>>> pca.n_components_
154
```

We can also plot the explained variance as a function of dimensions (simply plot the `cumsum` array). There will usually be an elbow in the curve, where the explained variance stops growing fast.



Dimensionality reduction as a preprocessing step

We can dimensionality reduction as a preprocessing step for a supervised learning task (e.g., classification) and fine-tune the number of dimensions:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.pipeline import make_pipeline

clf = make_pipeline(PCA(random_state=42),
                    RandomForestClassifier(random_state=42))
param_distrib = {
    "pca__n_components": np.arange(10, 80),
    "randomforestclassifier__n_estimators": np.arange(50, 500)
}
rnd_search = RandomizedSearchCV(clf, param_distrib, n_iter=10, cv=3,
                                random_state=42)
rnd_search.fit(X_train[:1000], y_train[:1000])
```

Hyperparameters found:

```
>>> print(rnd_search.best_params_)
{'randomforestclassifier__n_estimators': 465, 'pca__n_components': 23}
```

PCA for compression

For the MNIST dataset, sacrificing 5% of the variance allows the 784-dimensional features to reduce down to 154 dimensions. This data compression ratio can speed up a classification algorithm (e.g. SVM) tremendously.

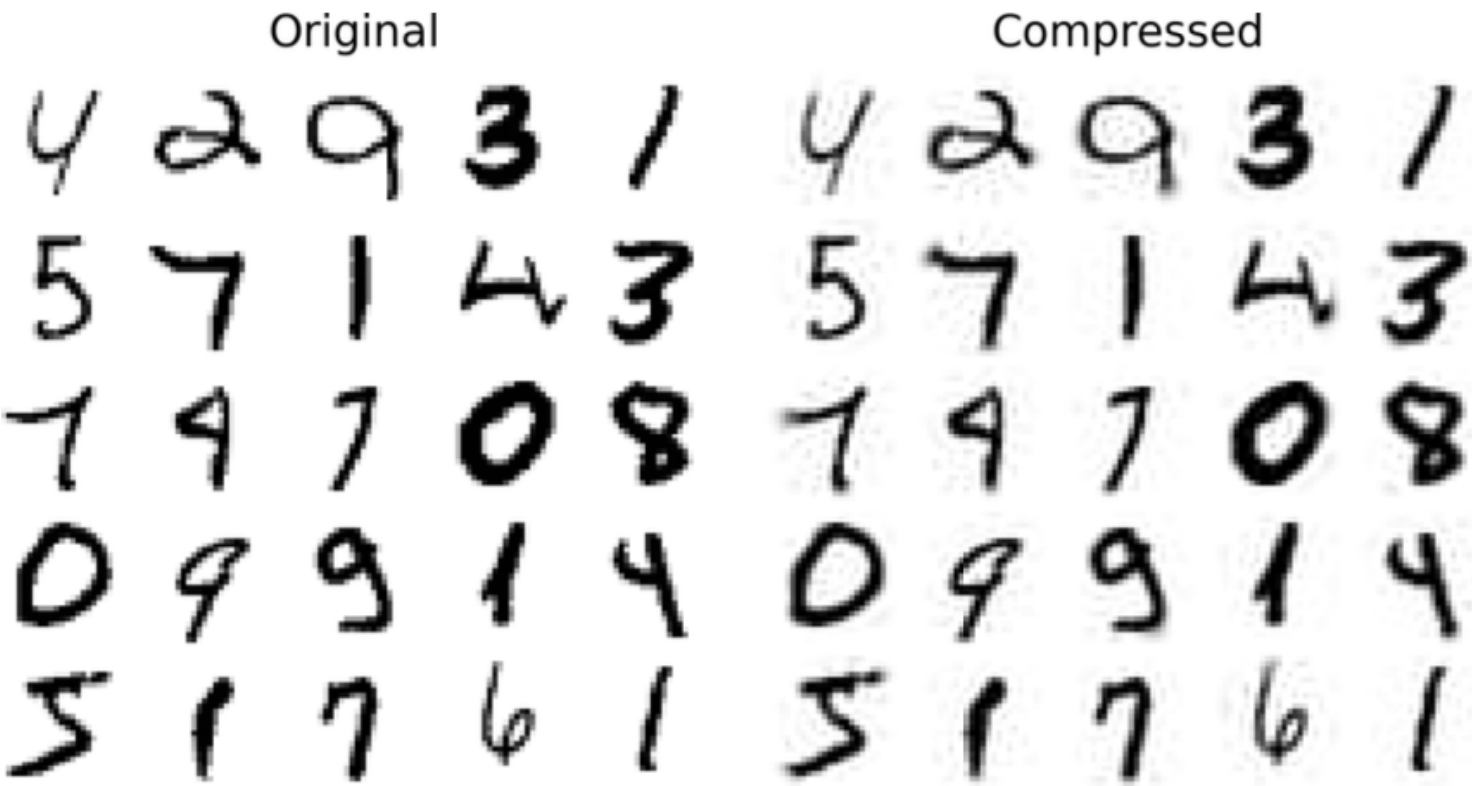
It is also possible to decompress the data back to 784 dimensions by applying the inverse transformation of the PCA projection to reconstruct the data:

$$\mathbf{X}_{\text{recovered}} = \mathbf{X}_{d\text{-proj}}\mathbf{W}_d^T$$

```
X_recovered = pca.inverse_transform(X_reduced)
```

The mean squared distance between the original data and the reconstructed data is called the *reconstruction error*.

PCA for compression



A few digits from the original training set (on the left), and the corresponding digits after compression and decompression.

PCA - Example

Illustration of 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`.

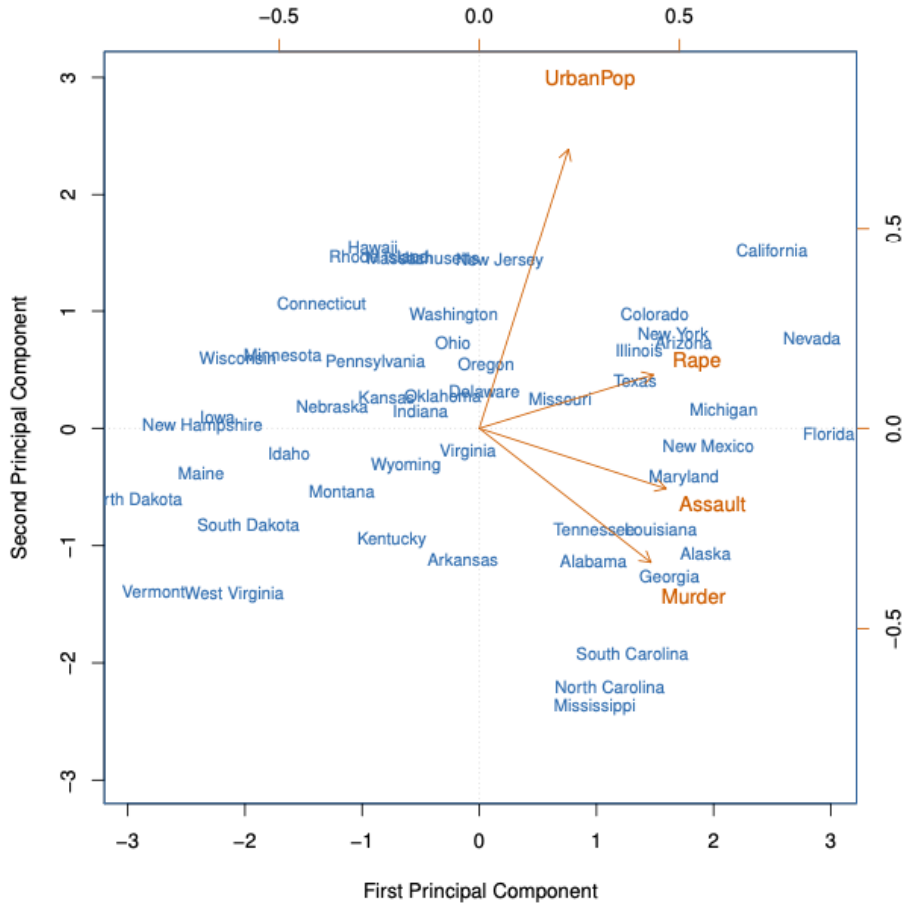
Also available `UrbanPop` (the percent of the population in each state living in urban areas).

The principal component score vectors (Z_1, Z_2, Z_3 and Z_4) have length $m = 50$, and the principal component loading vectors have length $n = 4$ (ϕ_1, ϕ_2, ϕ_3 and ϕ_4).

Each feature was standardised to have zero mean and unit standard deviation.

PCA - Example

The first two principal components, the principal component scores and the loading vectors in a single `biplot` display.



	PC1	PC2
Murder	0.5358	- 0.4181
Assault	0.5831	- 0.1879
UrbanPop	0.2781	0.8728
Rape	0.5434	0.1673

PCA - Example

Crime-related variables seem to be correlated with each other. UrbanPop variable is less correlated with the other three.

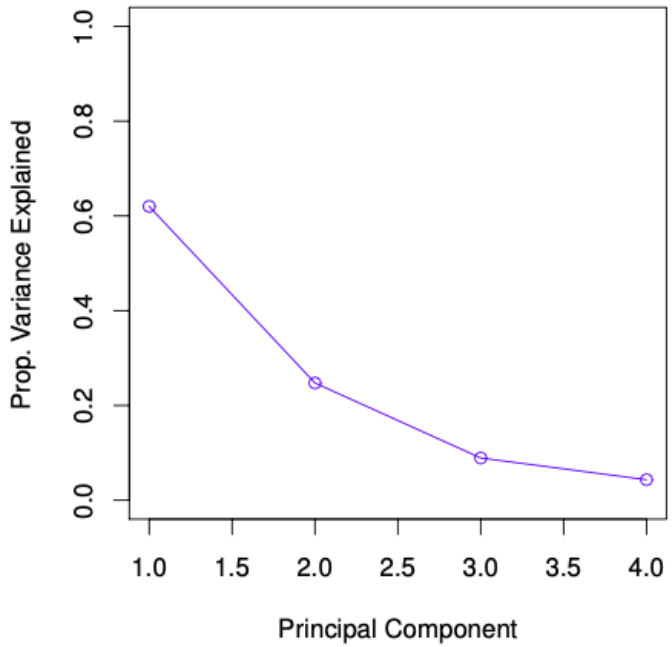
Results also suggests that:

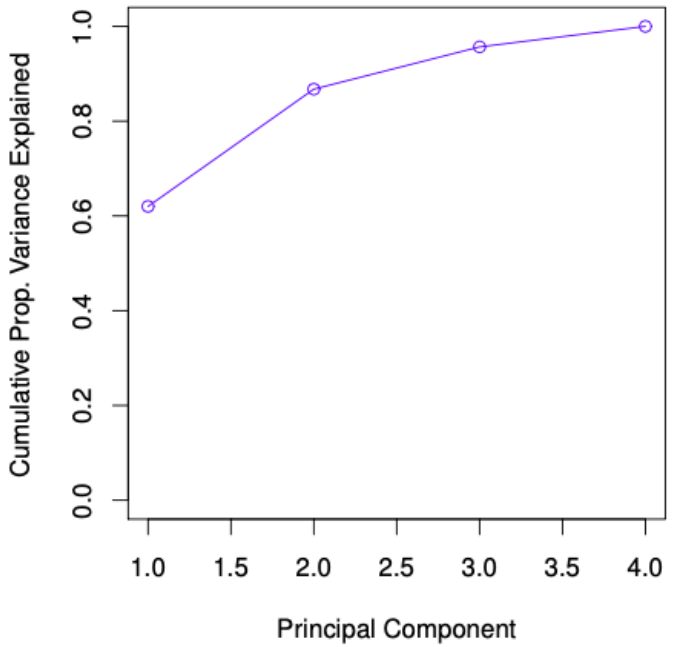
- States with large positive values on the first PC (e.g. California, Nevada): indicative to have high crime rates.
- States with negative scores on the first PC (e.g. North Dakota): indicative to have low crime rates.
- States with high values on the second PC (e.g. California): indicative to have high level of urbanization.
- States with high values on the second PC (e.g. Mississippi): indicative to have low level of urbanization.
- States with values close to zero on both components (e.g. Indiana): indicative to have approximately average levels of both crime and urbanization.

PCA - Example

The principal component explains about 62.0% of the variance in the data, while the second 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.





For next week

Work through Assignment 3 and attend a supervised lab. The Unit Coordinator or a casual Teaching Assistant will be there.

Read up to Chapter 9 on Unsupervised Learning Techniques.