Machine Learning Course

Vahid Reza Khazaie

Dimensionality Reduction

- **PCA**
- **MDS**
- **LDA**
- **t-SNE**
- **...**

Dimensionality Reduction

- There are thousands or even millions of features for each training instance.

- Why is that a problem?

  - They can make the training extremely slow.

  - They can make it much harder to find a good solution.

Dimensionality Reduction

▶ On the other hand, there are also unimportant or redundant features in the dataset and you will not lose much information if you remove them.

▶ For example, consider the Hoda images: the pixels on the image borders are almost always white, so you could completely drop these pixels from the training set without losing much information. These pixels are utterly unimportant for the classification task. Additionally, two neighboring pixels are often highly correlated: if you merge them into a single pixel (e.g., by taking the mean of the two pixel intensities), you will not lose much information.

Dimensionality Reduction

▶ So, reducing the dimensionality of the training data may filter out some **noise** and **unnecessary details** and thus result in higher performance.

▶ **Be careful!** Not always reducing the dimensionality of the data will result in a better performance; But always it speeds up the training.

Dimensionality Reduction

▶ Also:

Apart from speeding up training, dimensionality reduction is also extremely useful for data

Visualization, Reducing the number of dimensions down to two (or three) makes it possible to plot a condensed view of a high-dimensional training set on a graph and often gain some important insights by visually detecting patterns, such as clusters
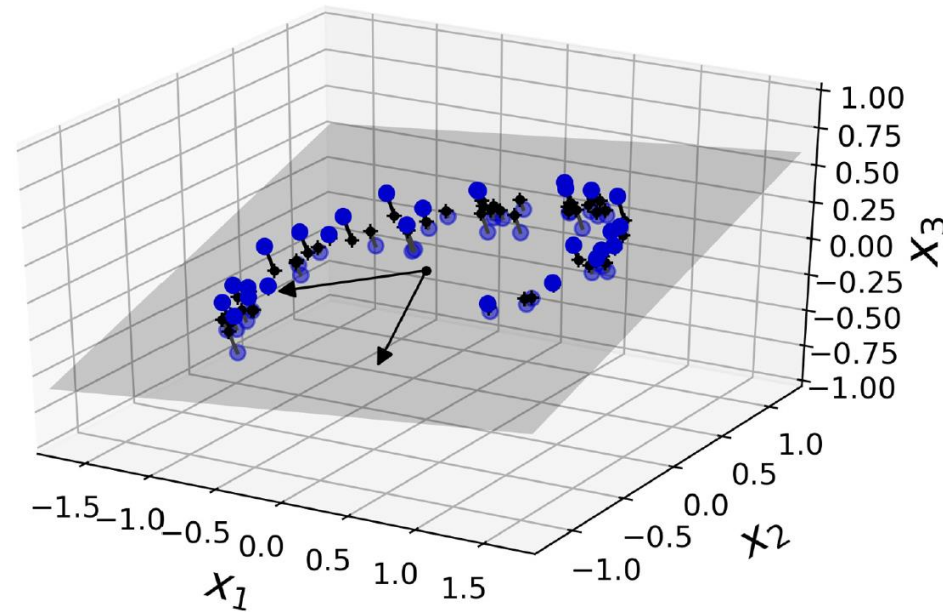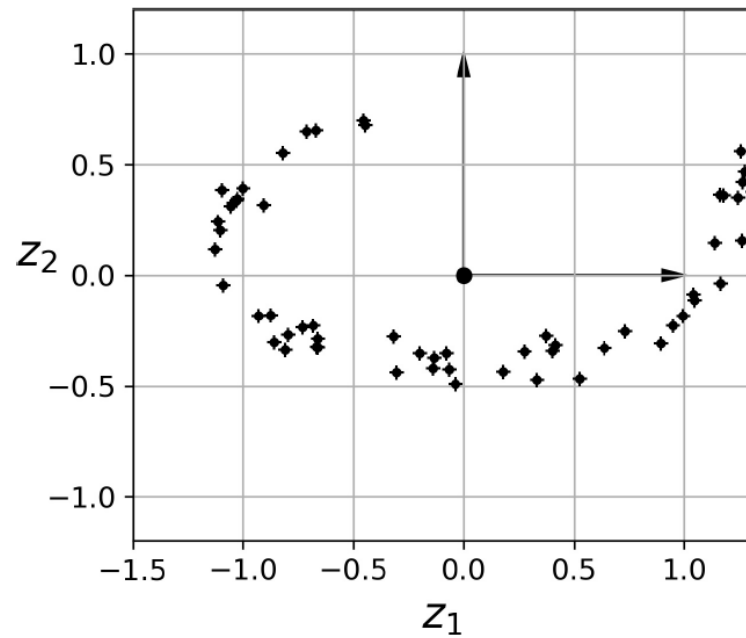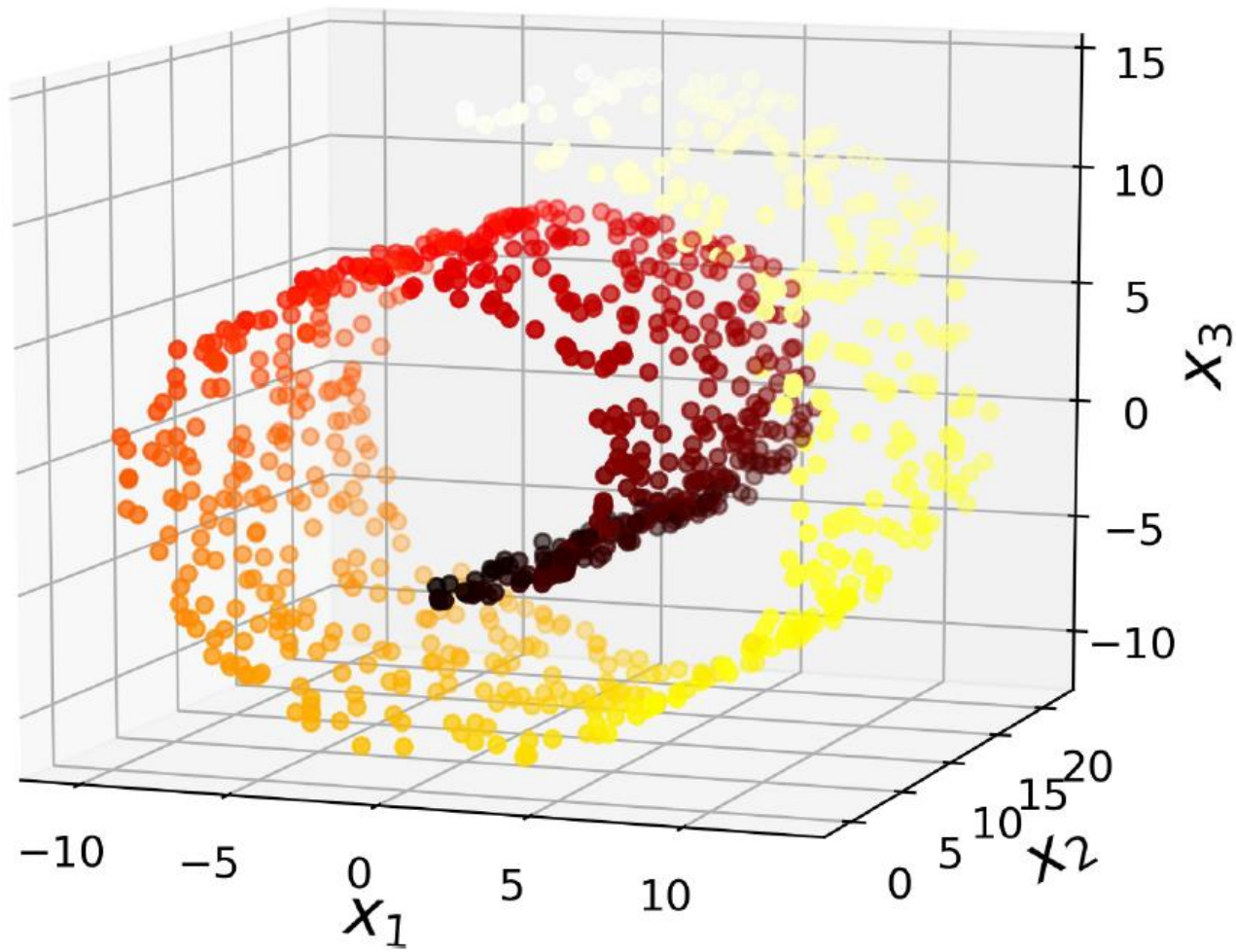
Projection
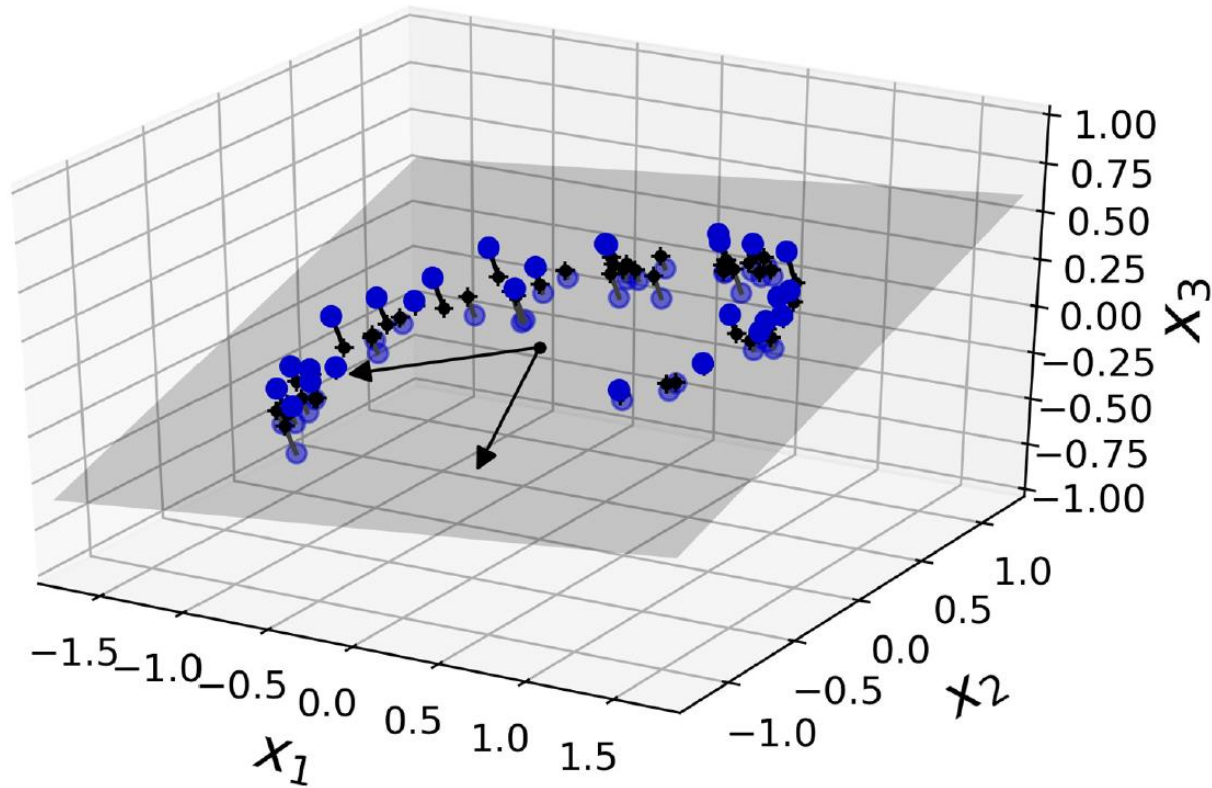
Projection

If we project every training instance perpendicularly onto this subspace (as represented by the short lines connecting the instances to the plane), we get the new 2D dataset.
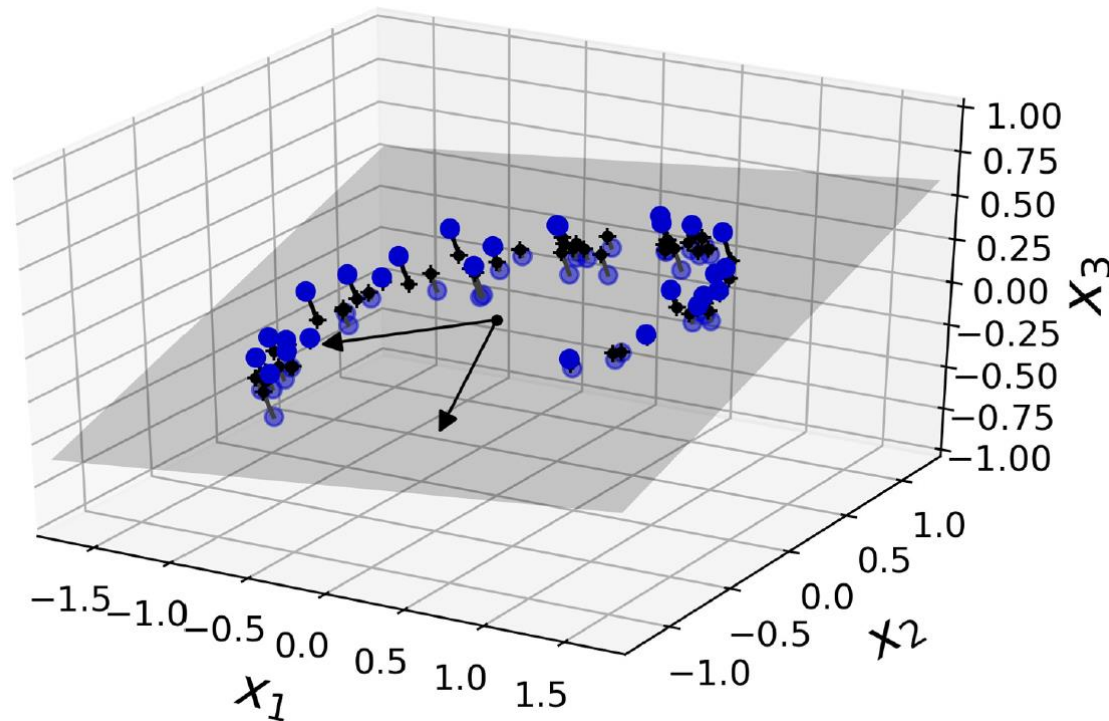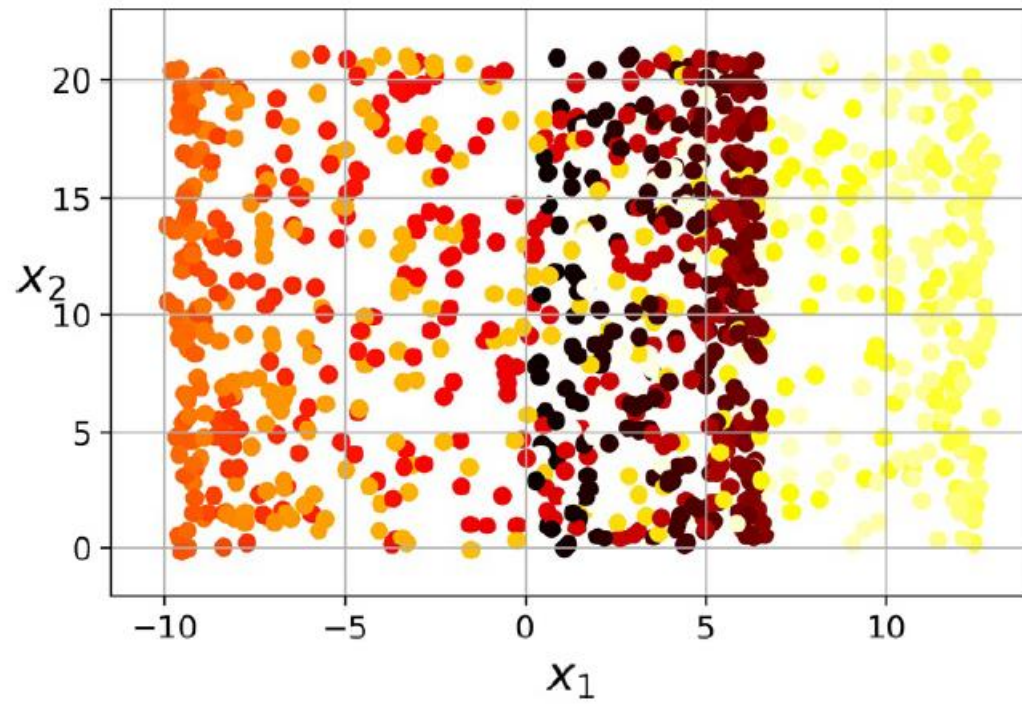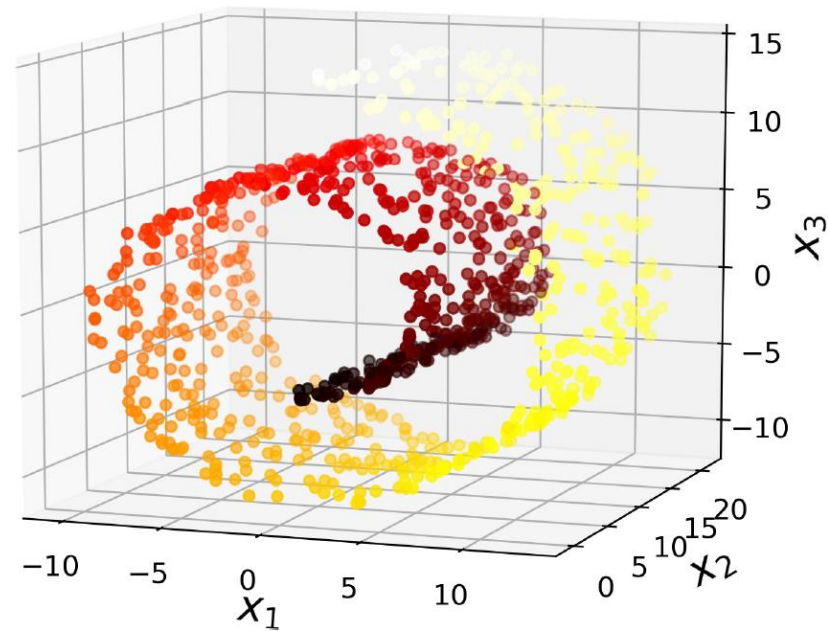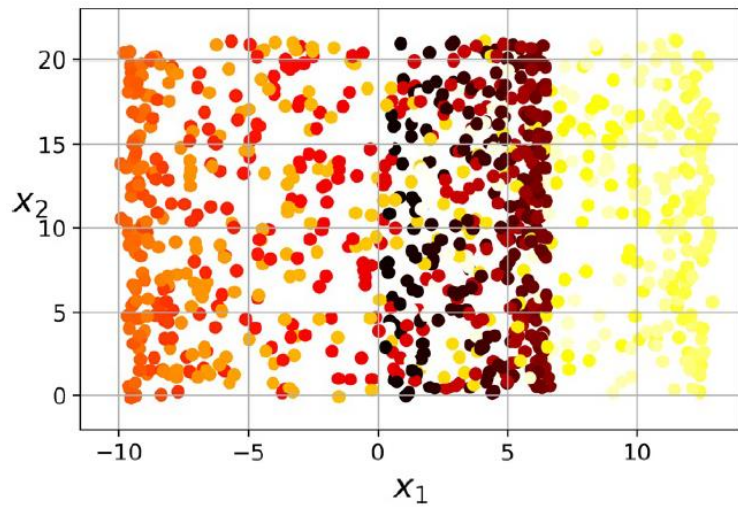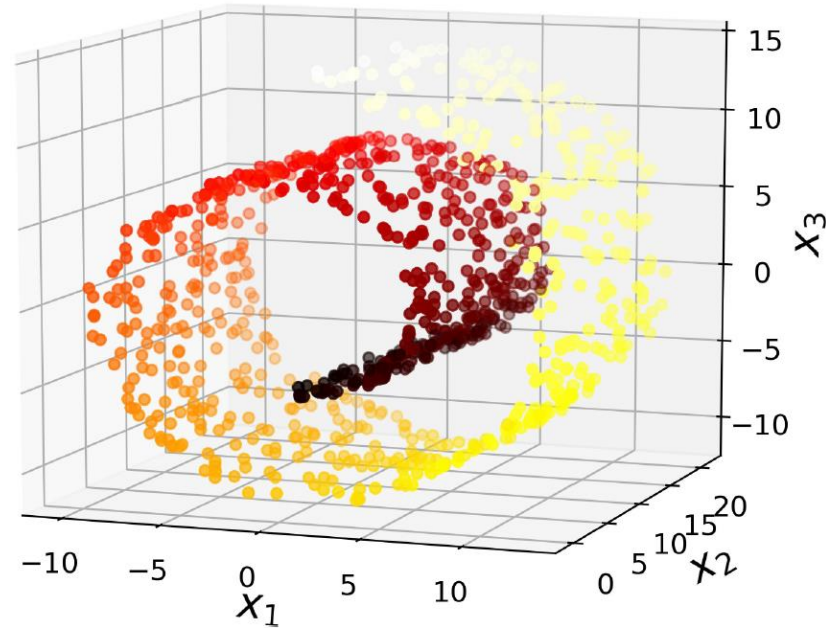
Projection

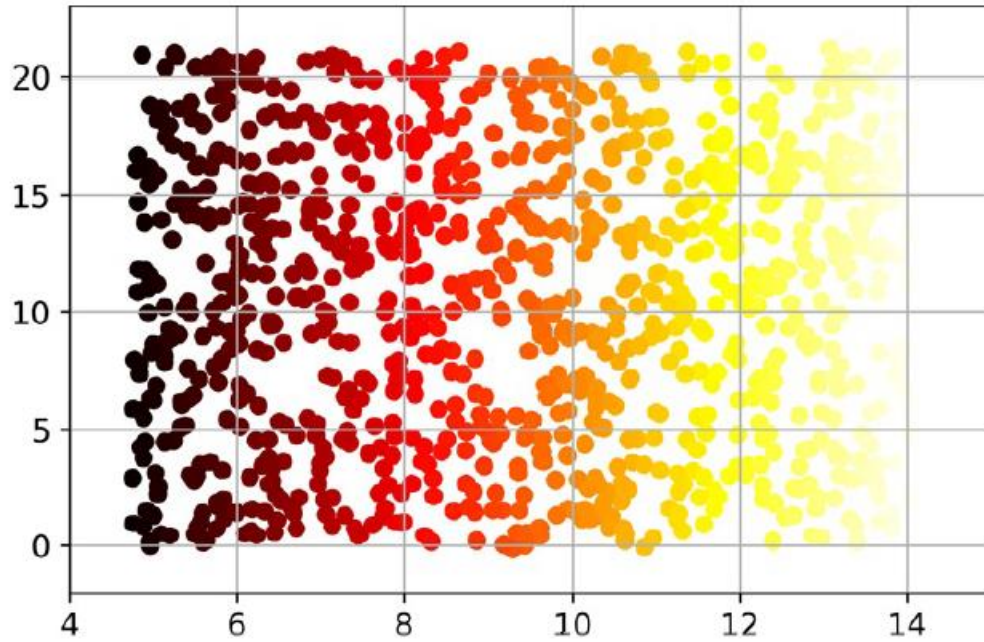# Projection

Projection

# Projection

NOT GOOD PROJECTTION!

Projection

We want this one!

Principal Component Analysis (PCA)

Before you can project the training set onto a lower-dimensional hyperplane, you first need to choose the right hyperplane. For example, a simple 2D dataset is represented on the left, along with three different axes (i.e., 1D hyperplanes). On the right is the result of the projection of the dataset onto each of these axes. As you can see, the projection onto the solid line preserves the maximum variance, while the projection onto the dotted line preserves very little variance and the projection onto the dashed line preserves an intermediate amount of variance.
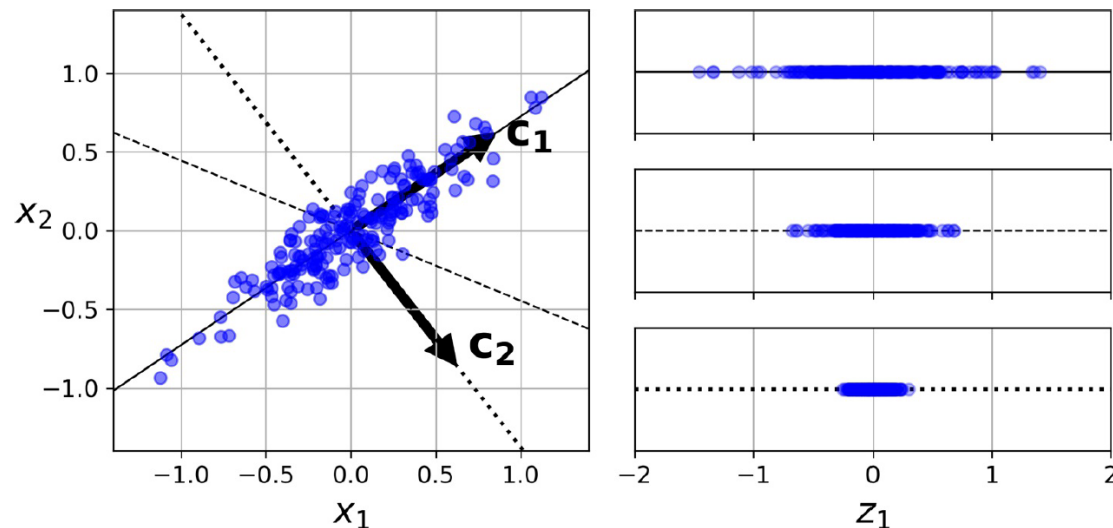
Principal Component Analysis (PCA)

PCA identifies the axis that accounts for the **largest amount of variance** in the training set. It also finds a second axis, orthogonal to the first one, that accounts for the largest amount of remaining variance. In this 2D example there is no choice: it is the dotted line. If it were a higher-dimensional dataset, PCA would also find a third axis, orthogonal to both previous axes, and a fourth, a fifth, and so on—as many axes as the number of dimensions in the dataset.
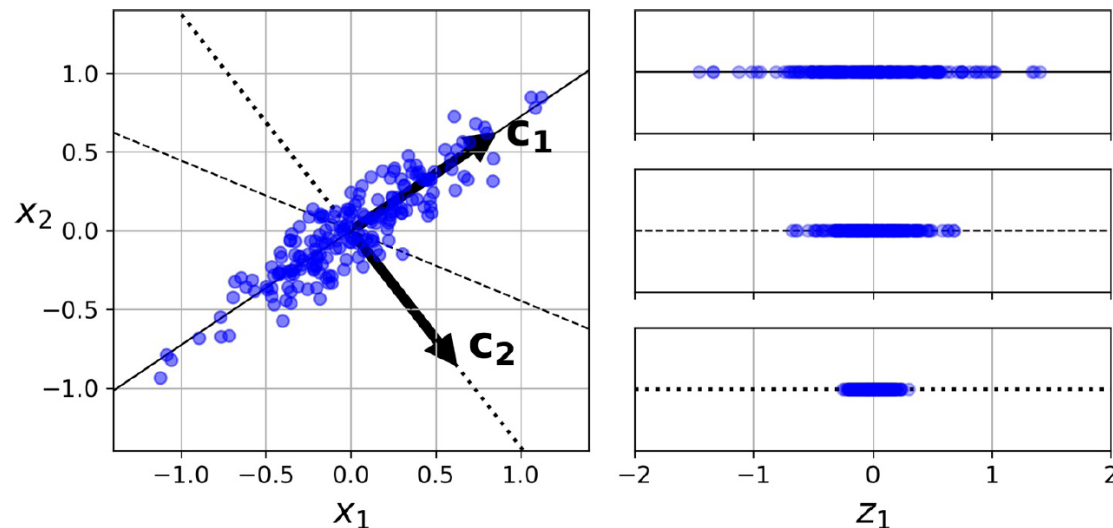
Principal Component Analysis (PCA)

Singular Value Decomposition (SVD) Technique

X into the matrix multiplication of three matrices

```
X_centered = X - X.mean(axis=0)
U, s, Vt = np.linalg.svd(X_centered)
c1 = Vt.T[:, 0]
c2 = Vt.T[:, 1]
```

$$V = \begin{pmatrix} | & | & & | \\ c_1 & c_2 & \cdots & c_n \\ | & | & & | \end{pmatrix}$$

Principal Component Analysis (PCA)

Once you have identified all the principal components, you can reduce the dimensionality of the dataset down to d dimensions by projecting it onto the hyperplane defined by the first d principal components. Selecting this hyperplane ensures that the projection will preserve as much variance as possible.

To project the training set onto the hyperplane and obtain a reduced dataset **X** of dimensionality d, compute the matrix multiplication of the training set matrix **X** by the matrix **W** , defined as the matrix containing the first d columns of **V**.

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

Principal Component Analysis (PCA)

Once you have identified all the principal components, you can reduce the dimensionality of the dataset down to d dimensions by projecting it onto the hyperplane defined by the first d principal components. Selecting this hyperplane ensures that the projection will preserve as much variance as possible.

To project the training set onto the hyperplane and obtain a reduced dataset **X** of dimensionality d, compute the matrix multiplication of the training set matrix **X** by the matrix **W** , defined as the matrix containing the first d columns of **V**.

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

```
X_centered = X - X.mean(axis=0)
U, s, Vt = np.linalg.svd(X_centered)
c1 = Vt.T[:, 0]
c2 = Vt.T[:, 1]

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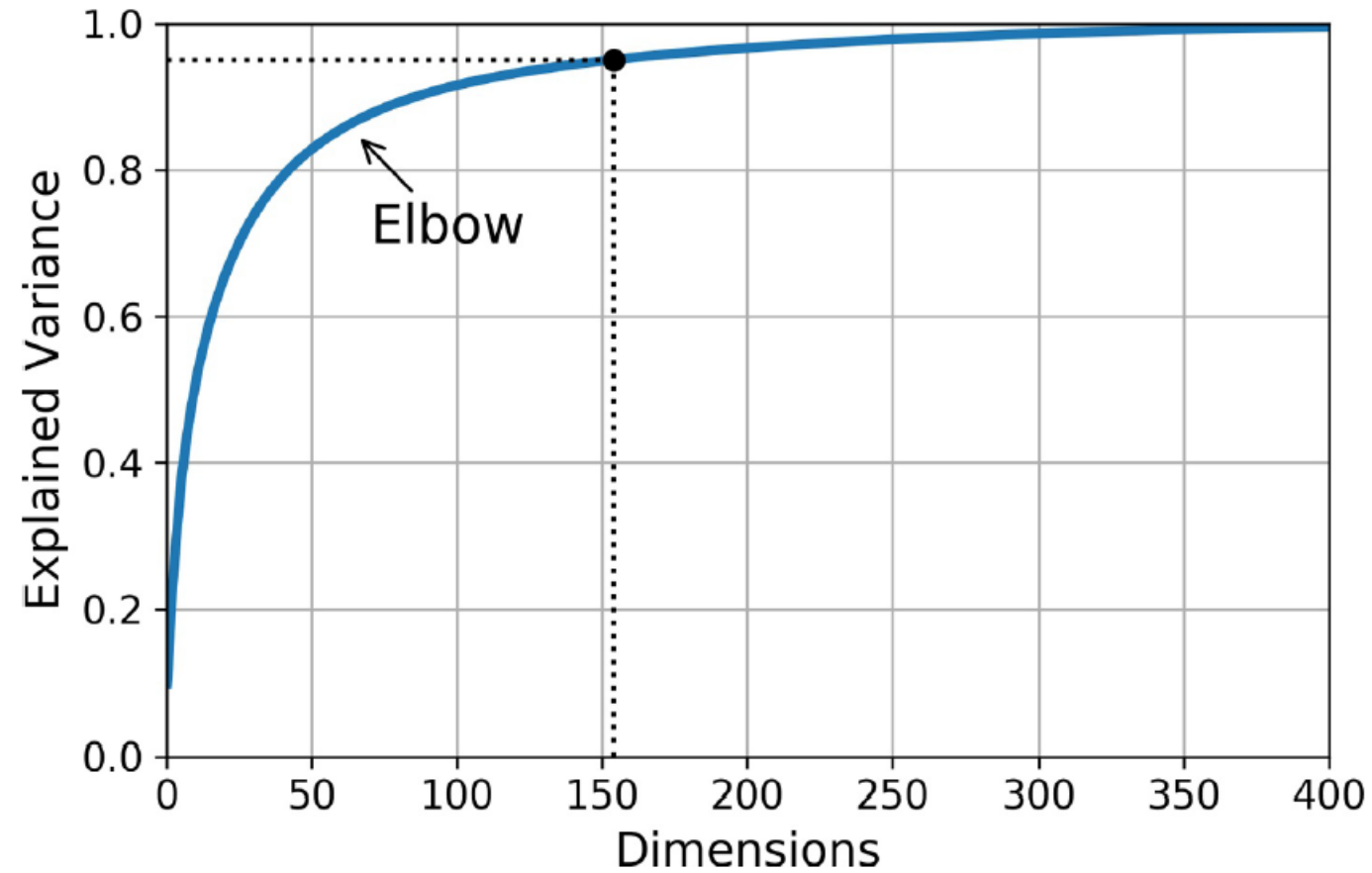
Principal Component Analysis (PCA)

This output tells you that 84.2% of the dataset's variance lies along the first PC, and 14.6% lies along the second PC. This leaves less than 1.2% for the third PC, so it is reasonable to assume that the third PC probably carries little information.

```python
from sklearn.decomposition import PCA

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


>>> pca.explained_variance_ratio_
array([0.84248607, 0.14631839])
```

Principal Component Analysis (PCA)

Principal Component Analysis (PCA)

```python
pca = PCA(n_components=0.95)
X_reduced = pca.fit_transform(X_train)
```

Principal Component Analysis (PCA)

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

```python
pca = PCA(n_components = 154)
X_reduced = pca.fit_transform(X_train)
X_recovered = pca.inverse_transform(X_reduced)
```

Principal Component Analysis (PCA)

Principal Component Analysis (PCA)

- **Randomized PCA**
  - set the svd_solver hyperparameter to "randomized"
- **Incremental PCA**
  - useful for large training sets and for applying PCA online
- **Kernel PCA**
  - making it possible to perform complex nonlinear projections for dimensionality reduction