CS286: AI for Science and Engineering

Lecture 5: Traditional Machine Learning (Part 3)

Jie Zheng (郑杰)

PhD, Associate Professor

School of Information Science and Technology (SIST), ShanghaiTech University

Fall Semester, 2020

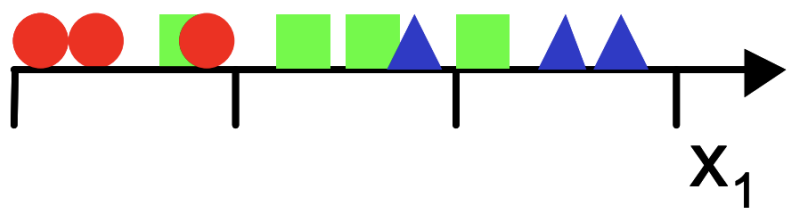
Outline

- Dimensionality reduction
  - Principal Component Analysis (PCA)
- Unsupervised learning
  - K-means
  - DBSCAN
  - Gaussian mixture model (GMM)

The Curse of Dimensionality

A 3-class pattern recognition problem

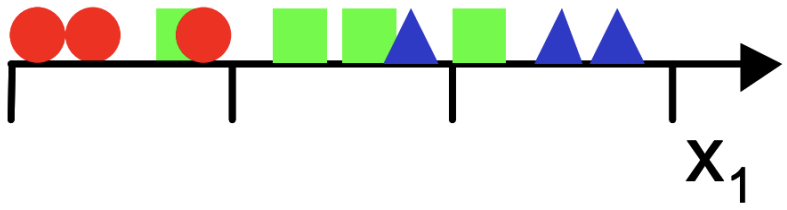
- Three types of objects have to be classified based on the value of a single feature:



- A naive procedure would be to
  - Divide the feature space into uniform bins
  - Compute the ratio of examples for each class at each bin
  - For a new example, find its bin and choose the predominant class in that bin

A 3-class pattern recognition problem

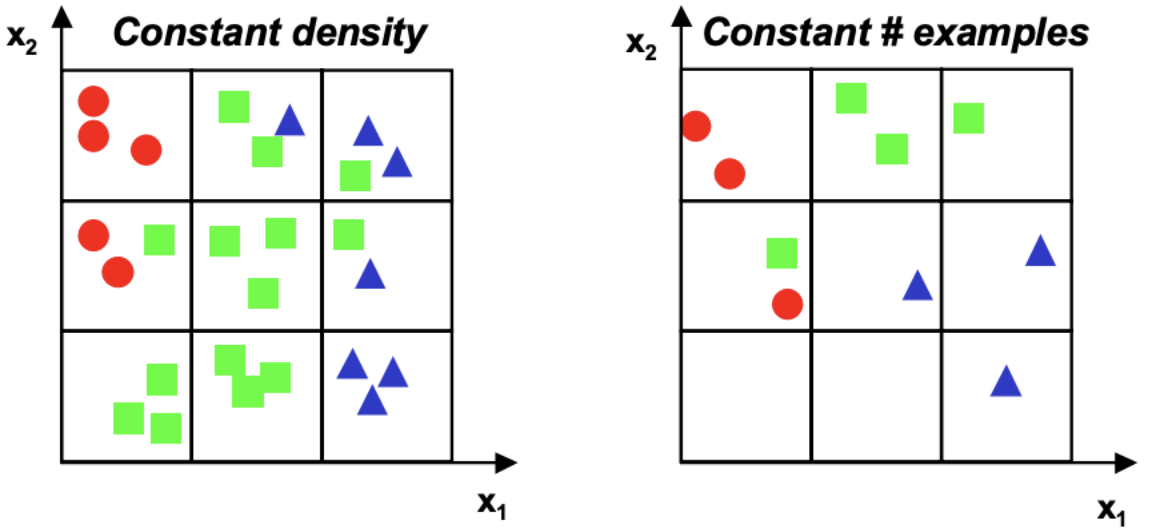
- Three types of objects have to be classified based on the value of a single feature:



- But we decide to start with one feature and divide the real line into 3 bins
  - Notice that there exists a lot of overlap between classes ⇒ to improve discrimination, we decide to incorporate a second feature

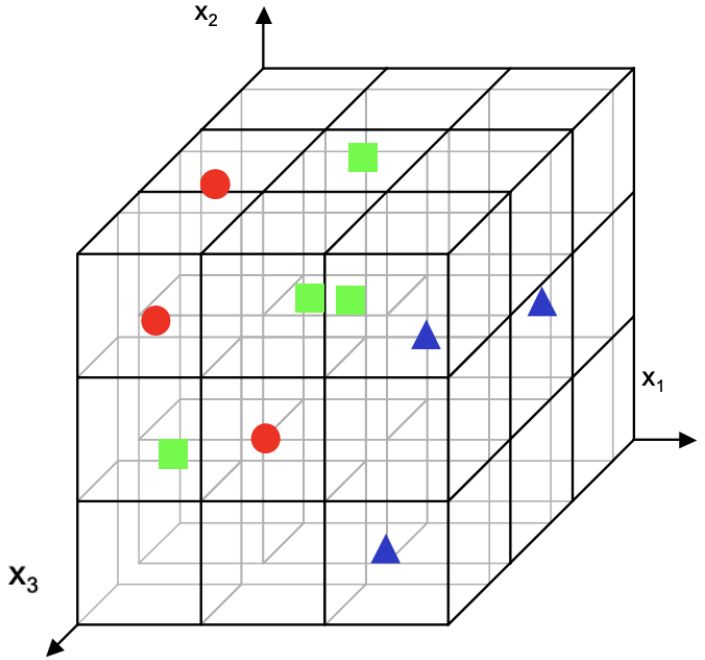
Moving to 2 dimensions

- Moving to 2 dimensions increases the number of bins from 3 to $3^2 = 9$
  - Which should we maintain constant?
    - **The density of examples per bin?** This increases the number of examples from 9 to 27
    - **The total number of examples?** This results in a $2D$ scatter plot that is very sparse



Moving to 3 dimensions

- The number of bins grows to $3^3 = 27$
- To maintain the initial density of examples, the number of required examples grows to 81
- For the same number of examples the $3\mathrm{D}$ scatter plot is fairly sparse



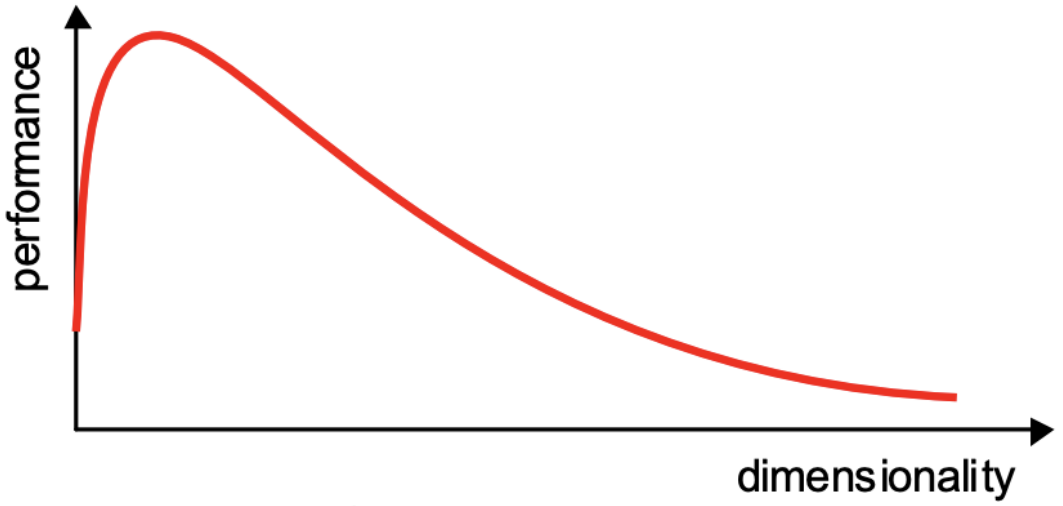
What is the curse of dimensionality?

- It refers to the problems associated with multivariate data analysis as the dimensionality increases

- **This fact implies that high-dimensional datasets are at risk of being very sparse:** most training instances (e.g., the three types of objects in figs) are likely to be far away from each other

Practical meaning of curse of dimensionality

- In practice, the curse of dimensionality means that
  - For a given sample size, there is a maximum number of features above which the performance of our classifier will degrade rather than improve
  - In most cases, the information that was lost by discarding some features is compensated by a more accurate mapping in lower-dimensional space



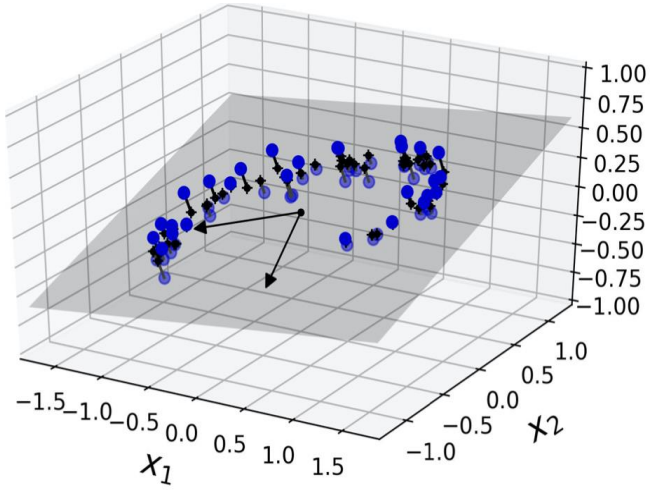
How to overcome the curse of dimensionality?

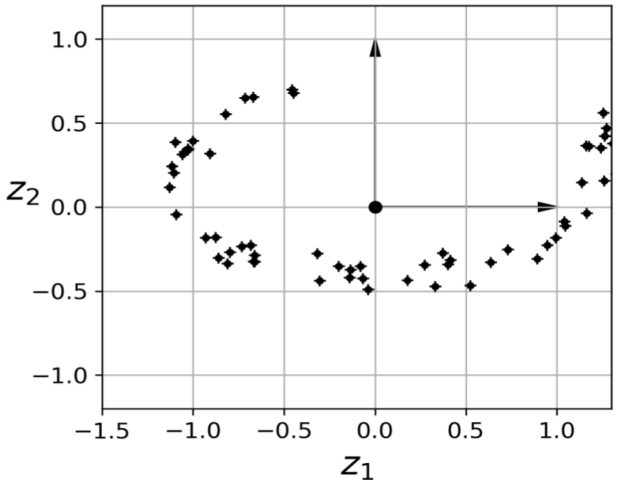
- By incorporating prior knowledge (hard to obtain)
- By increase the size of training set (number of samples required will grow exponentially)
- By reducing the dimensionality ✔

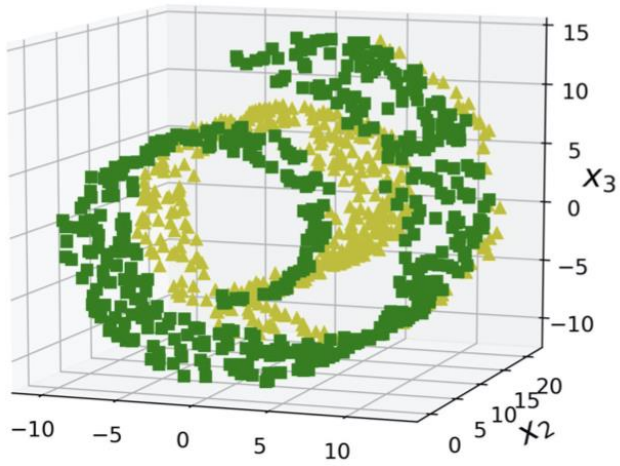
Approaches for Dimensionality Reduction

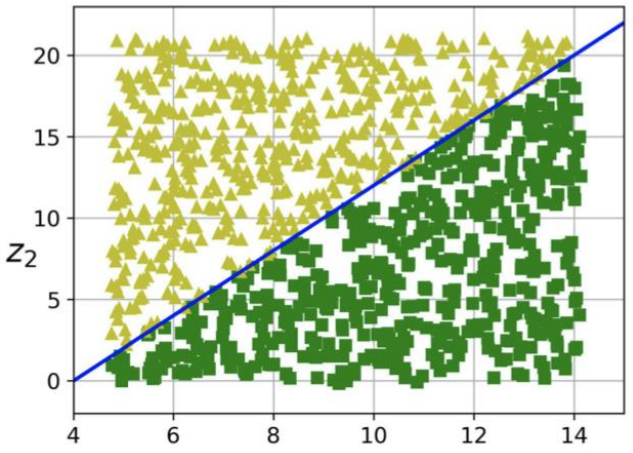
Two main approaches

- Two main approaches
  - Projection
    - Map the high dimension to lower-dimensional space
  - Manifold Learning
    - Modeling the manifold which training samples lie on







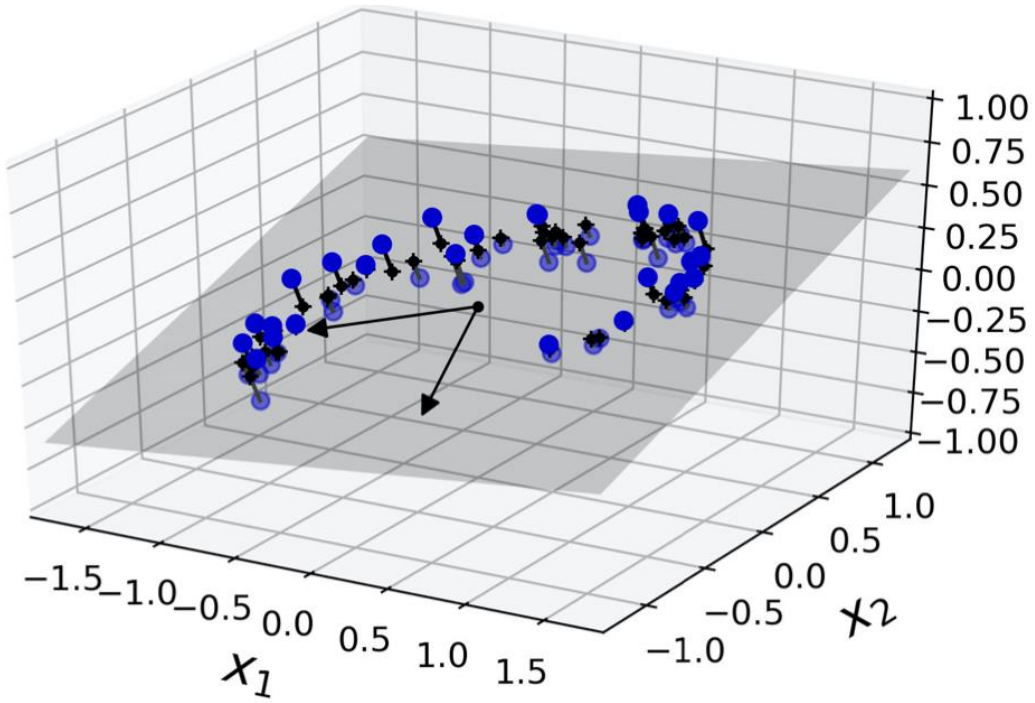


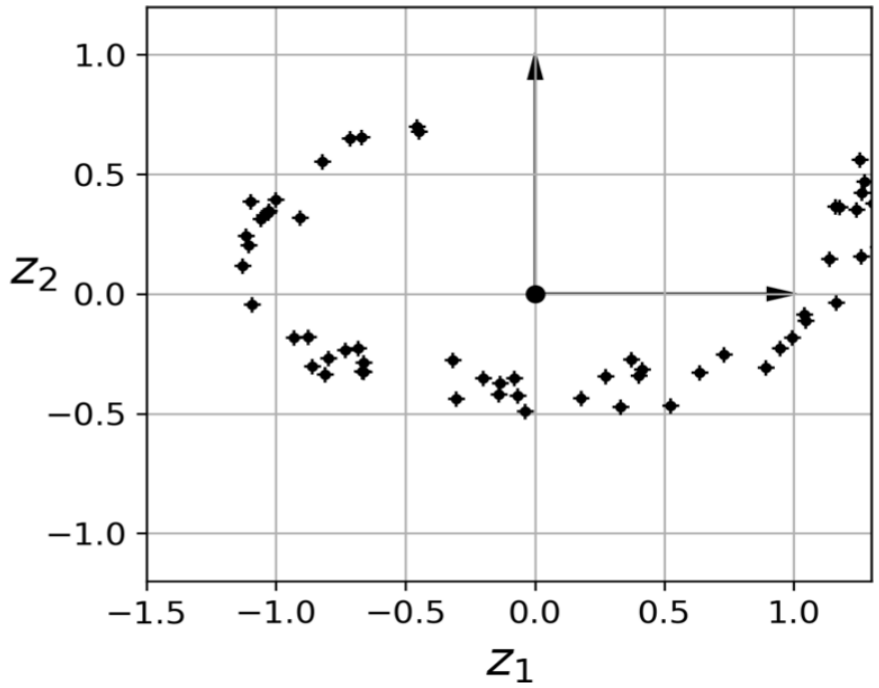
Projection

- Training instances are not spread out uniformly across all dimensions
- Many features are almost constant, while others are highly correlated
- As a result, all training instances actually lie within (or close to) a much lower-dimensional subspace of the high-dimensional space.

Projection (continued)

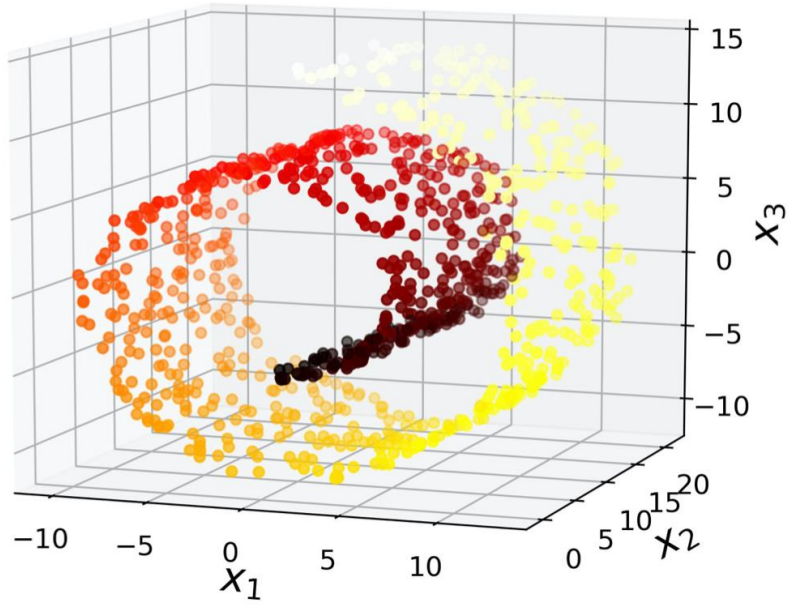
- Notice that all training instances lie close to a plane
- This is a lower-dimensional (2D) subspace of the high-dimensional (3D) space

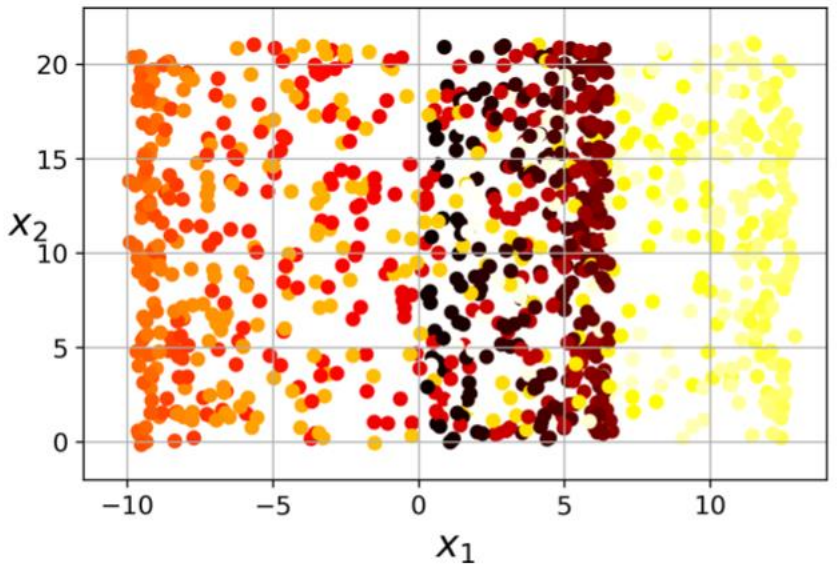




Projection loses its power

- In many cases the subspace may twist and turn, such as in the famous *Swiss roll* toy dataset
- projection is not always the best approach to dimensionality reduction





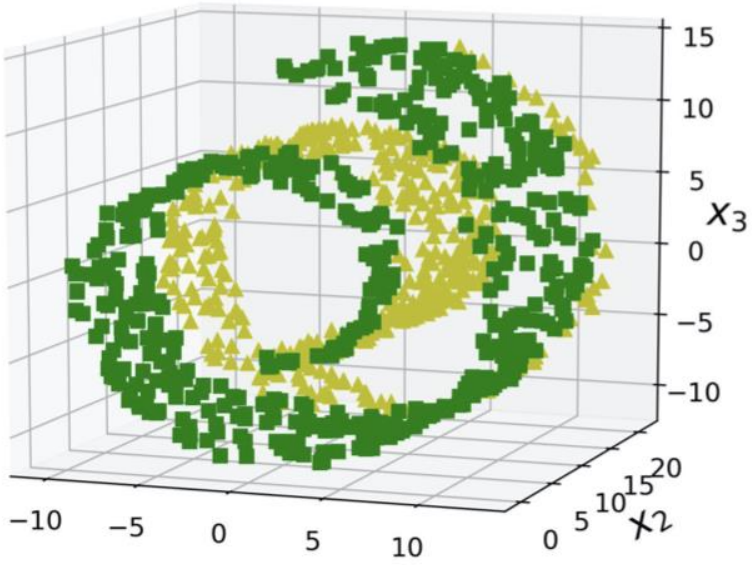
Manifold learning

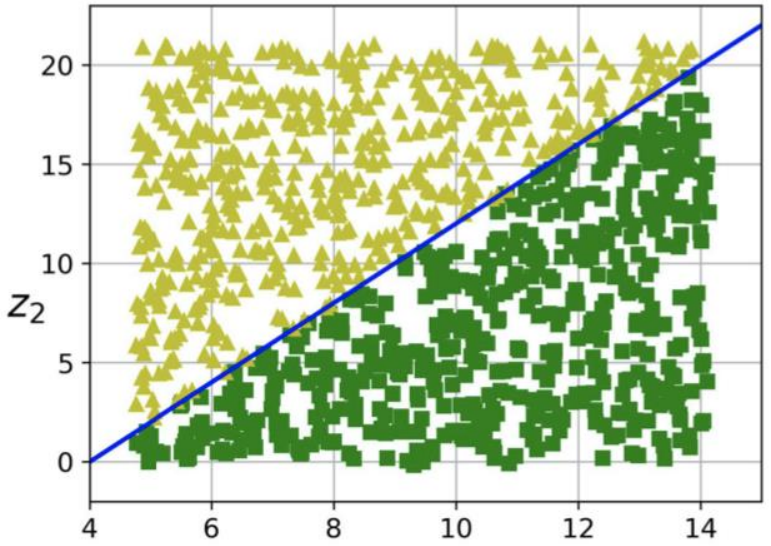
- Manifold Hypothesis: Most real-world high-dimensional datasets lie close to a much lower-dimensional manifold

- The task (e.g., **classification** or **regression**) will be simpler if expressed in the lower-dimensional space of the manifold (Manifold Hypothesis holds)

When manifold learning works well

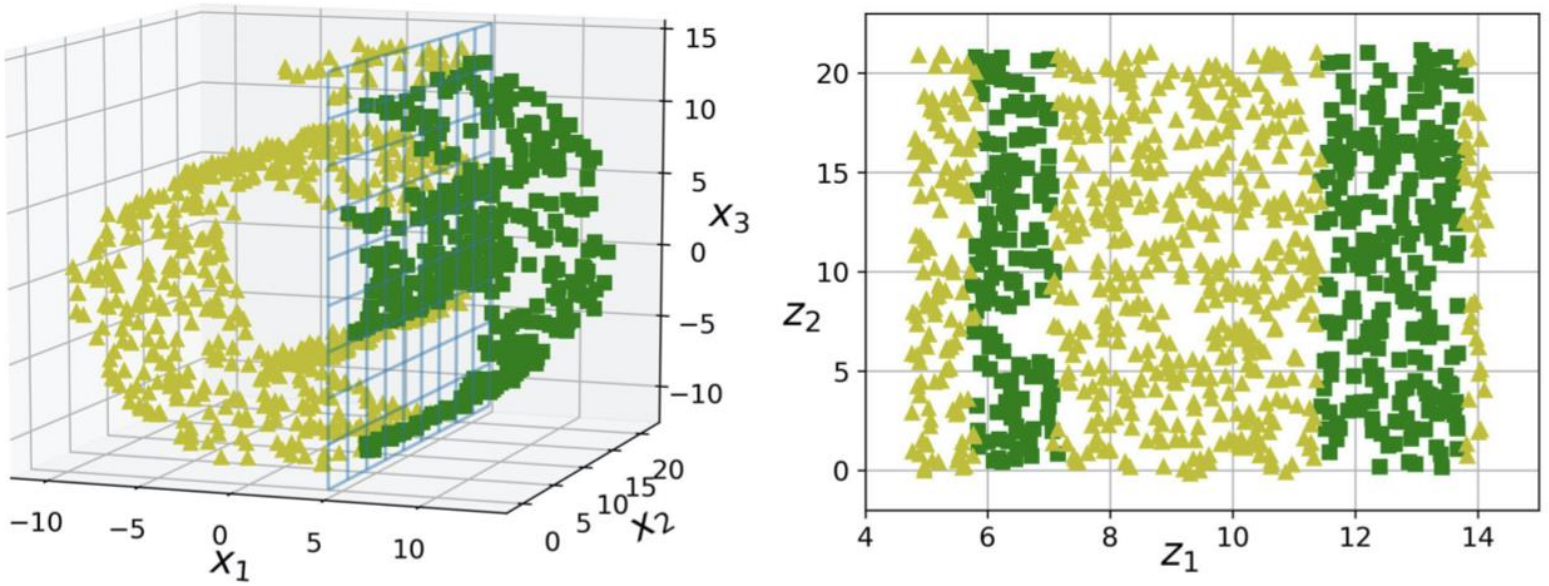
- The Swiss roll is split into two classes:
  - in the 3D space (left), the decision boundary would be fairly complex
  - in the 2D unrolled manifold space (right), the decision boundary is a simple straight line.





When manifold learning doesn’t work well

- What would happen if *Manifold Hypothesis* does not hold? (The decision boundary is located at $X_1 = 5$)
  - This decision boundary looks very simple in the original 3D space (a vertical plane)
  - But it looks more complex in the unrolled manifold



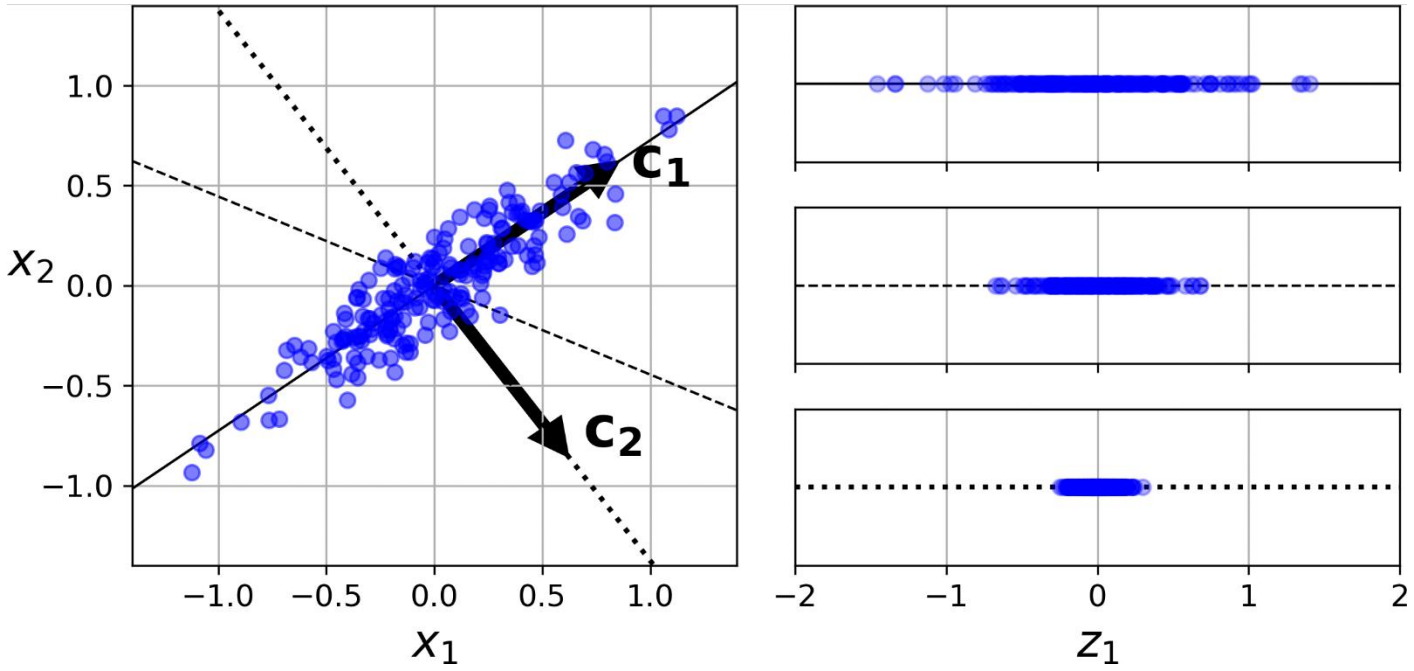
Principal Component Analysis

Overview of PCA

- *Principal Component Analysis* (PCA) is one of the most popular dimensionality reduction algorithms
- It identifies the hyperplane that lies closest to the data
- Then it projects the data onto it

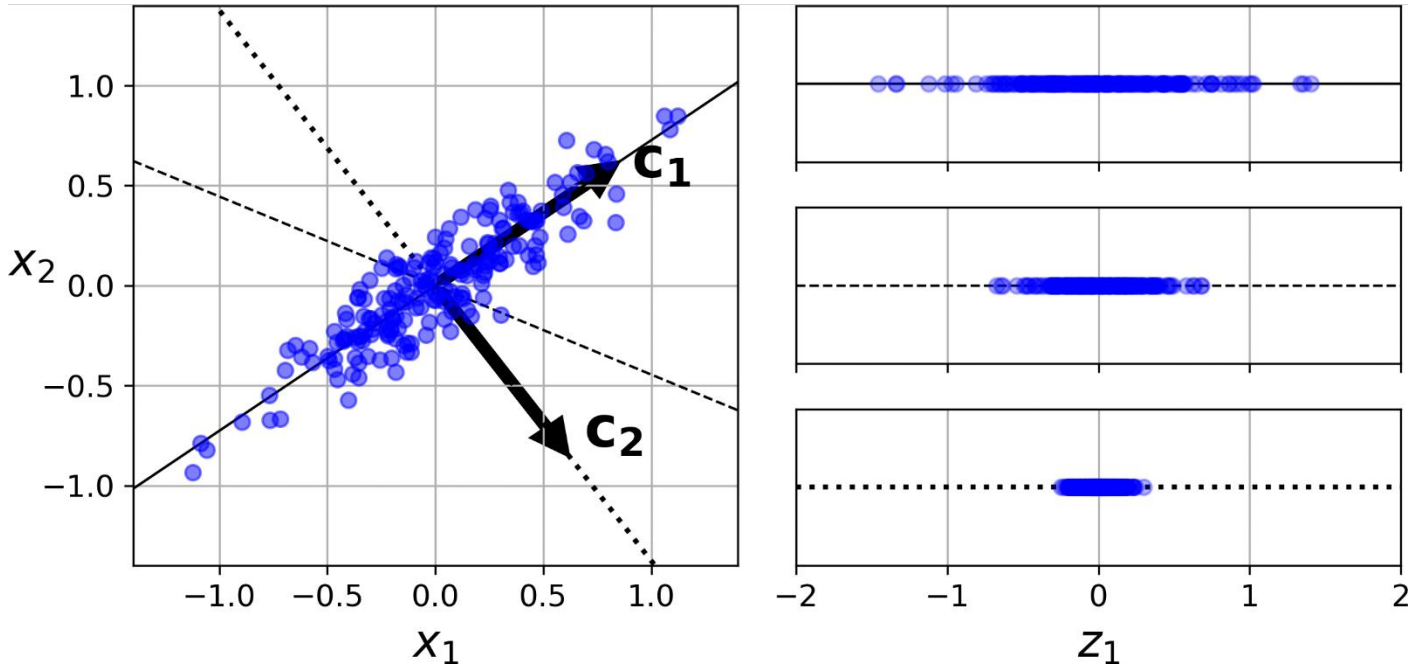
Preserving the variance

- Select the subspace that preserves the maximum amount of variance (lose less information)
- Or minimizes the *mean squared distance* between the original dataset and its projection onto a subspace



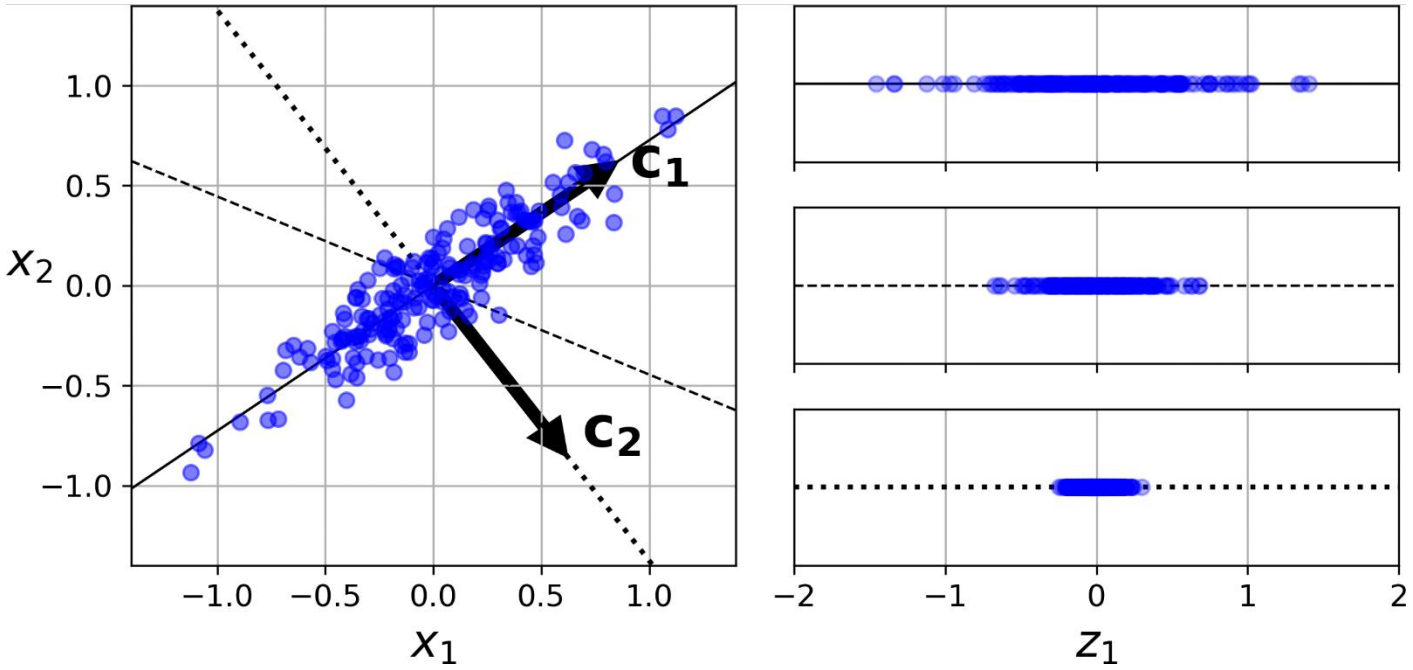
Principal Components

- PCA identifies the axis that accounts for the largest amount of variance
- The unit vector that defines the i^{th} axis is called the i^{th} *principal component* (PC)



Principal Components (continued)

- 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)



How to find the PCs?

- Standard matrix factorization technique called *Singular Value Decomposition* (SVD)
  - decompose the training set matrix $\mathbf{X}$ into the matrix multiplication of three matrices $\boldsymbol{U}\,\boldsymbol{\Sigma}\,\boldsymbol{V}^{\boldsymbol{T}}$
  - $\boldsymbol{V}$ contains all the principal components
  - $\boldsymbol{V} = (\boldsymbol{c_1}, \boldsymbol{c_2}, \ldots, \boldsymbol{c_n})$

How to find the PCs? (continued)

- The following Python code uses NumPy's svd() function to obtain all the principal components of the training set, then extracts the first two PCs:

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

Projecting down to d dimensions

- By projecting it onto the hyperplane defined by the first d principal components

$$\mathbf{X}_{d-proj} = \boldsymbol{X}\,\boldsymbol{W}_{\boldsymbol{d}}$$

- $\boldsymbol{X}$ is the training set matrix
- $\boldsymbol{W}_{\boldsymbol{d}}$ is the matrix containing the first d principal components

Projecting down to d dimensions (continued)

- The following Python code projects the training set onto the plane defined by the first two principal components:

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

Using Scikit-Learn

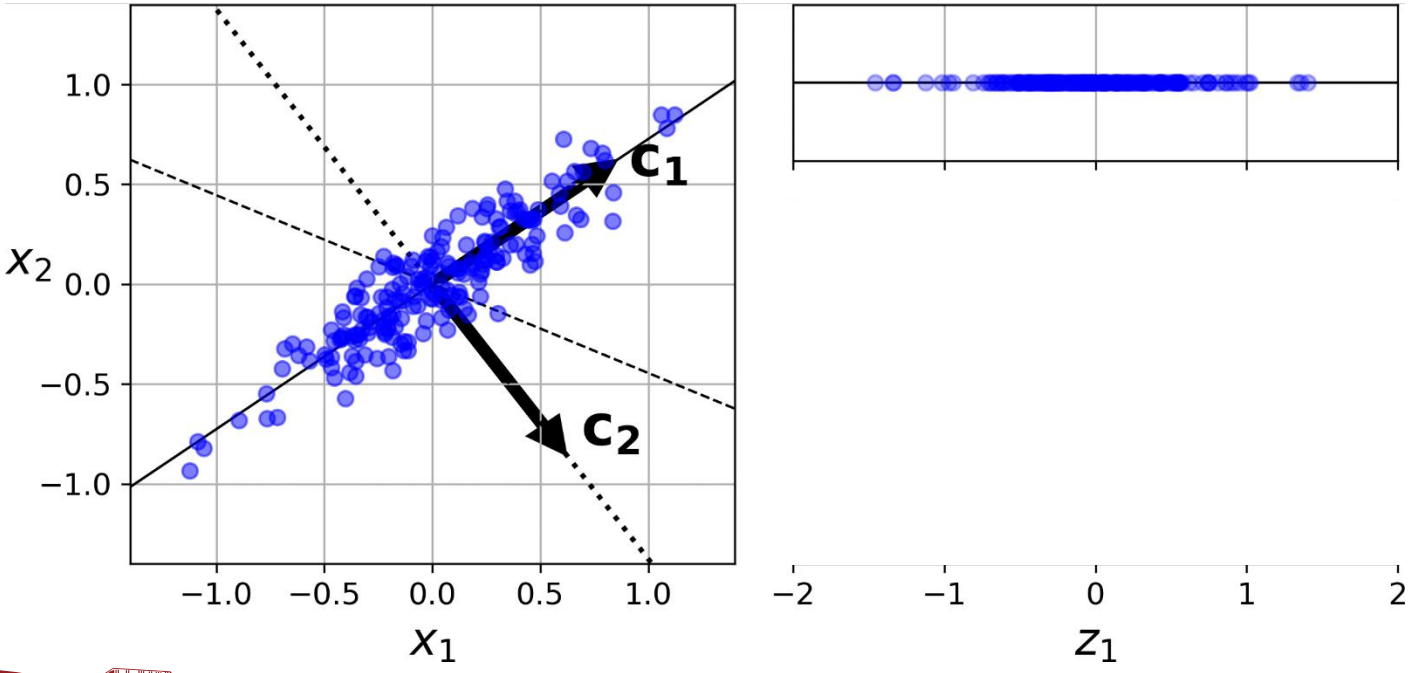
- *Scikit-Learn PCA* class implements PCA using SVD decomposition just like we did before
- The following code applies PCA to reduce the dimensionality of the dataset down to two dimensions

```
from sklearn.decomposition import PCA

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

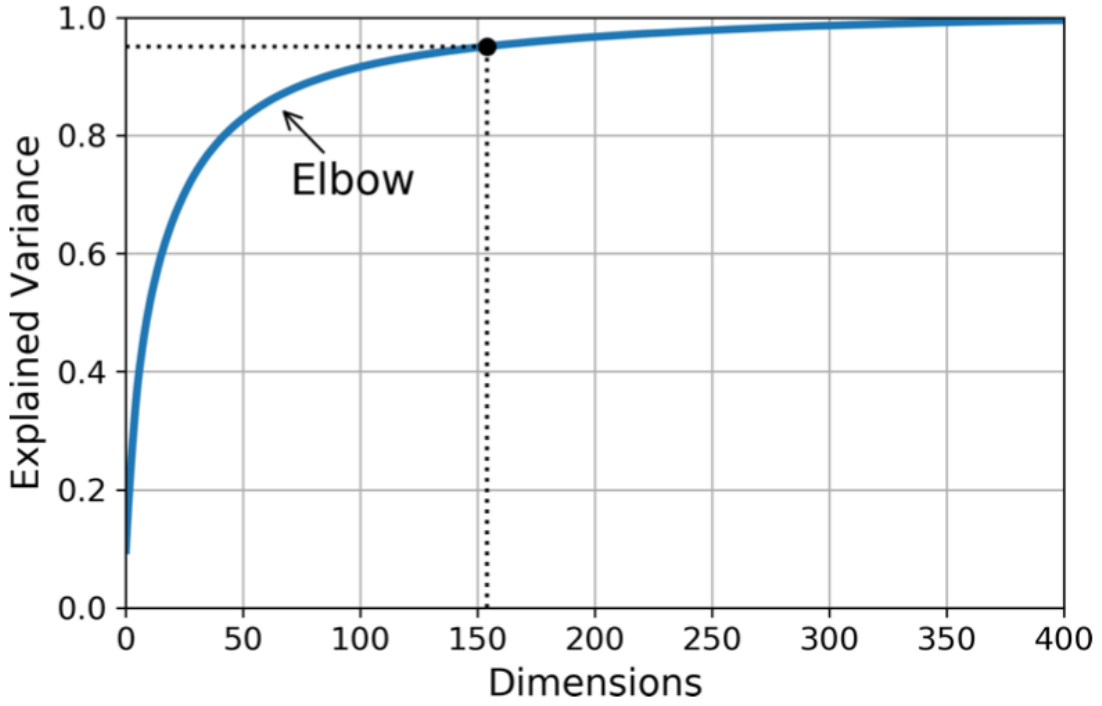
From 3D to 2D

- The 3D dataset is projected down to the 2D plane defined by the first two PCs
- As a result, the 2D projection looks very much like the original 3D dataset.



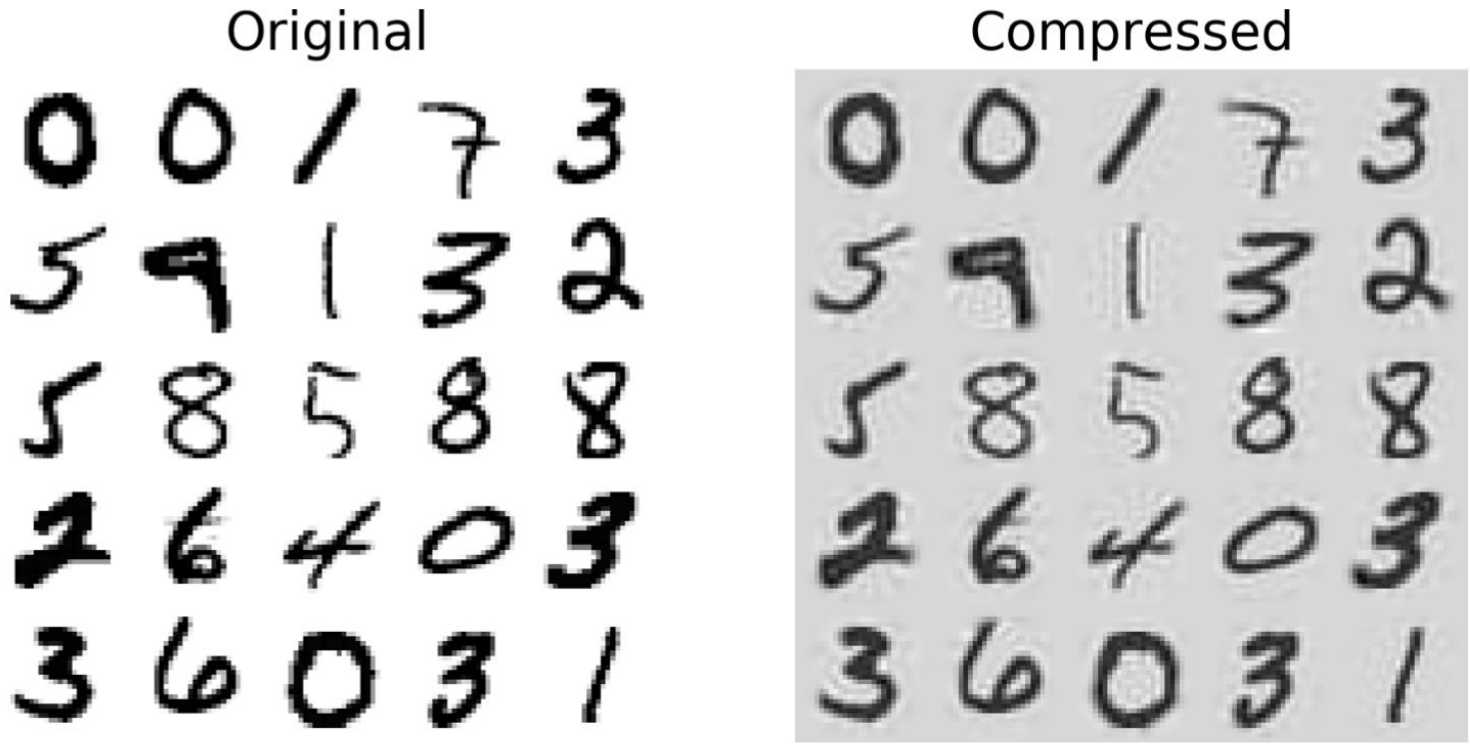
Choosing the right number of dimensions

- This is a tradeoff between the variance and the reduction



Application: PCA for compression

- Apply PCA to the MNIST dataset while preserving 95% of its variance

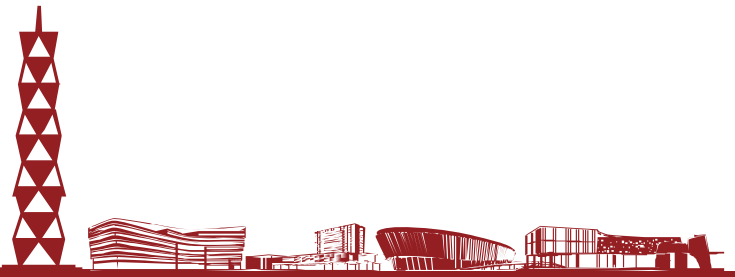


Application: PCA for compression (continued)

- Results
  - Find that each instance will have just over 150 features, instead of the original 784 features
  - The dataset is now less than 20% of its original size
  - Compressed dataset can speed up a classification algorithm

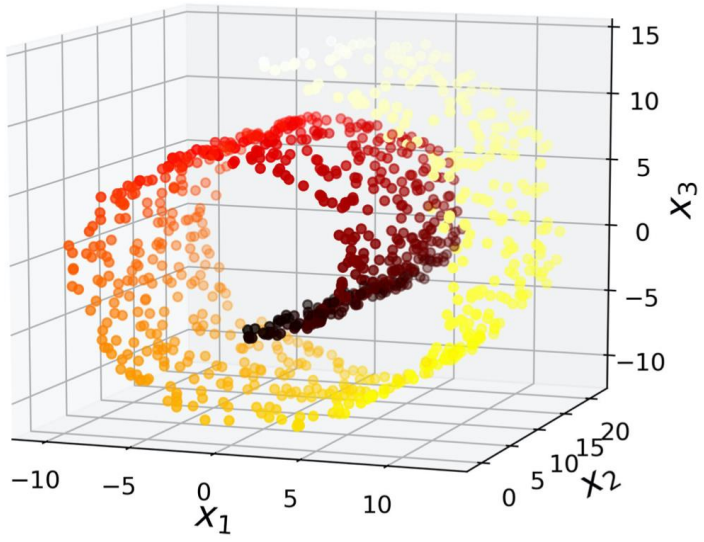
Modified PCA algorithms

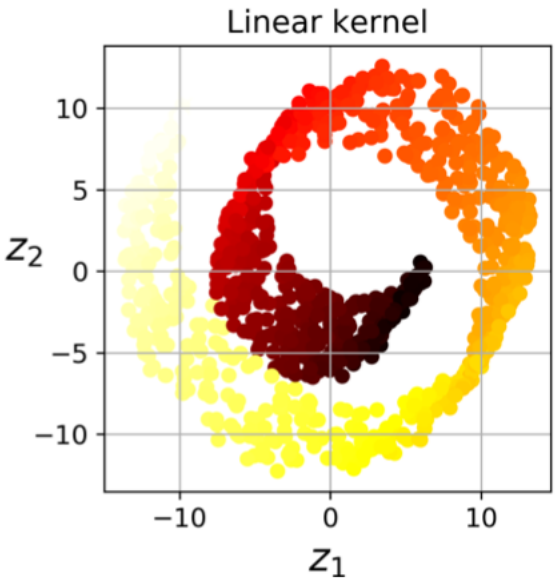
- Randomized PCA
  - Use a stochastic algorithm that quickly finds an approximation of the first d principal components
- Incremental PCA
  - Split the training set into mini-batches and feed an IPCA algorithm one mini-batch at a time instead of the whole training set
- Kernel PCA
  - Be good at preserving clusters of instances after projection, or unrolling datasets that lie close to a twisted manifold

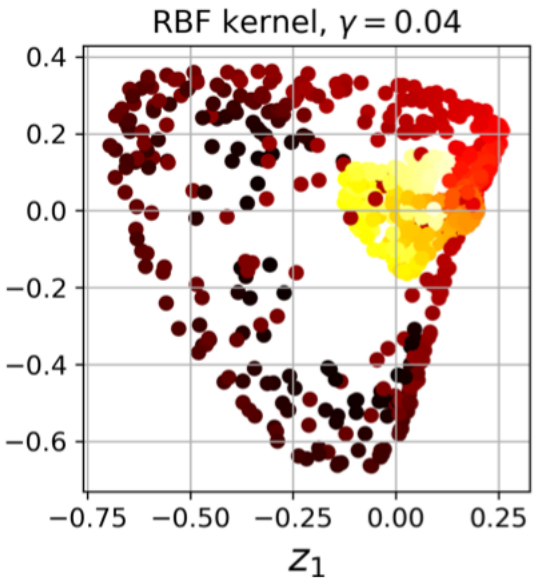


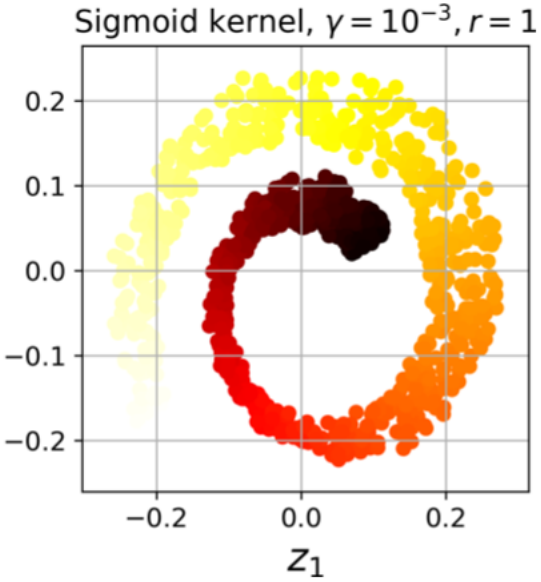
Kernel PCA

- Do you still remember the famous *Swiss roll* ? (the vanilla PCA loses power)
- kPCA makes it possible to perform complex nonlinear projections for dimensionality reduction



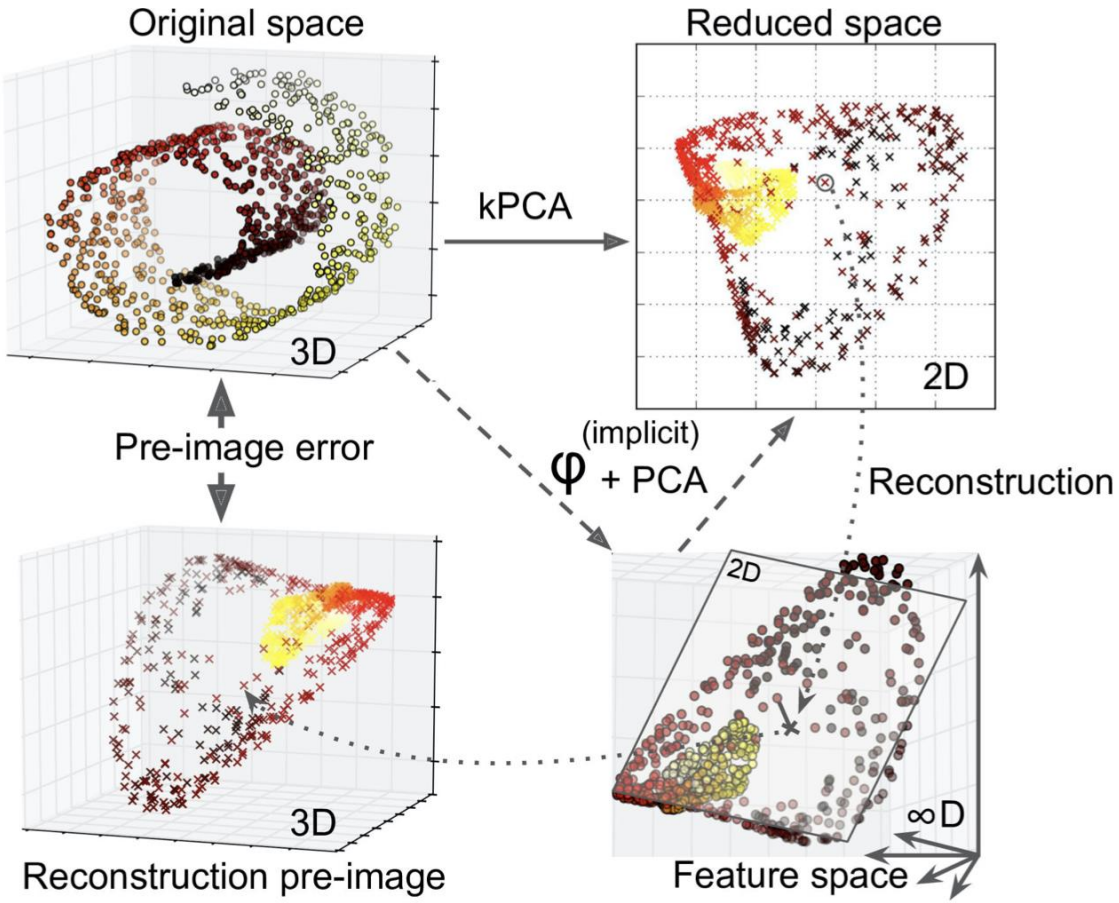






Kernel and hyperparameters

- Select the kernel and hyperparameters that yield the lowest reconstruction error



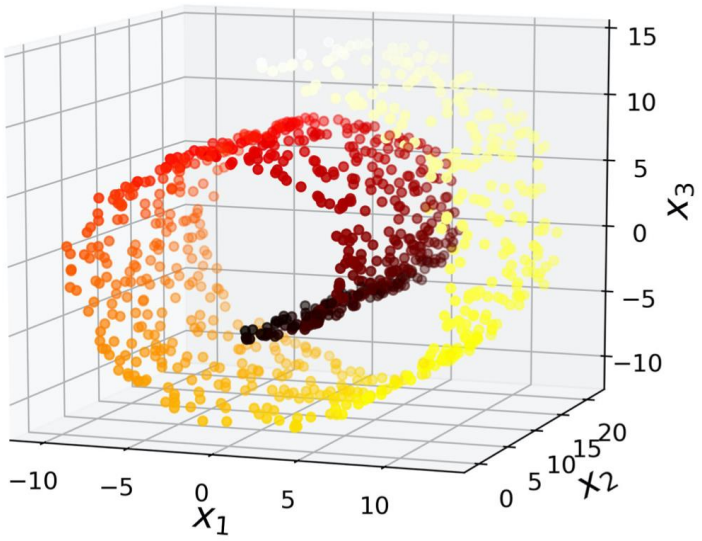
Other Dimensionality Reduction Techniques

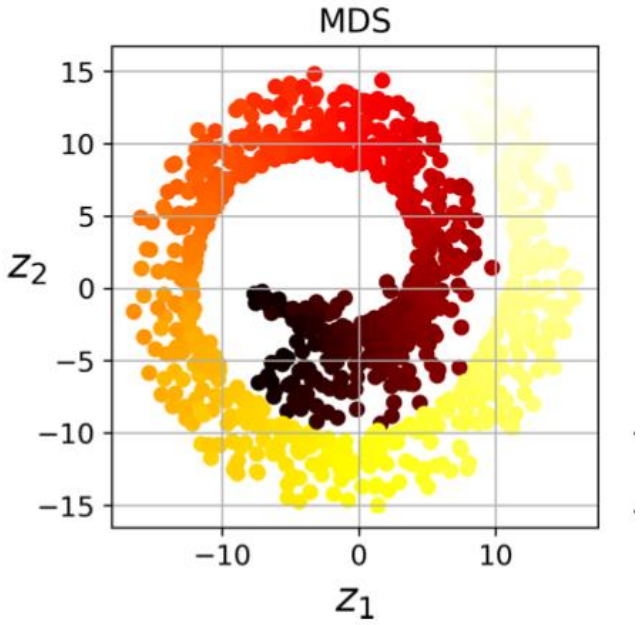
Locally linear embedding (LLE)

- **Locally Linear Embedding (LLE)** is another powerful *nonlinear dimensionality reduction* technique
- It is a Manifold Learning technique that does not rely on projections like the previous algorithms
- **How does LLE work?**
  - By first measuring how each training instance linearly relates to its closest neighbors
  - Then looking for a low-dimensional representation of the training set where these local relationships are best preserved

Multidimensional Scaling (MDS)

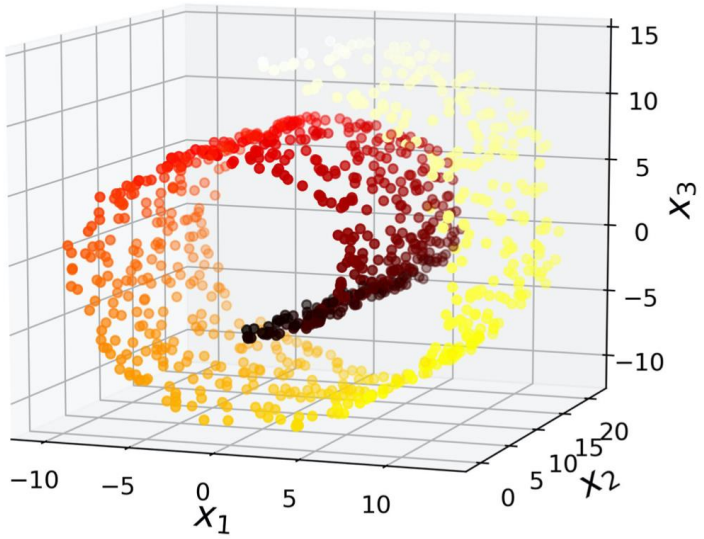
- Reduce dimensionality while trying to preserve the distances between the instances

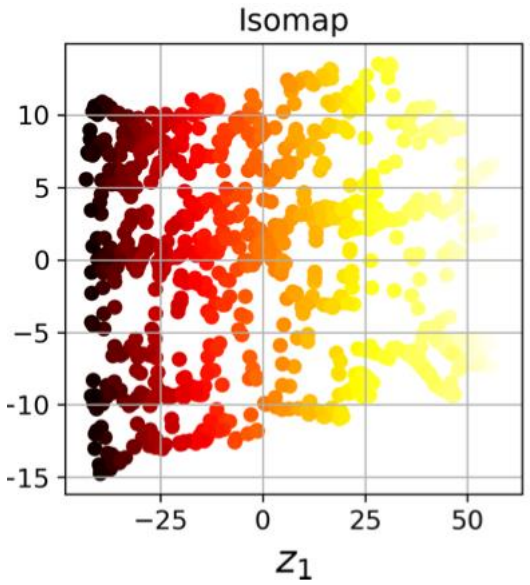




Isomap

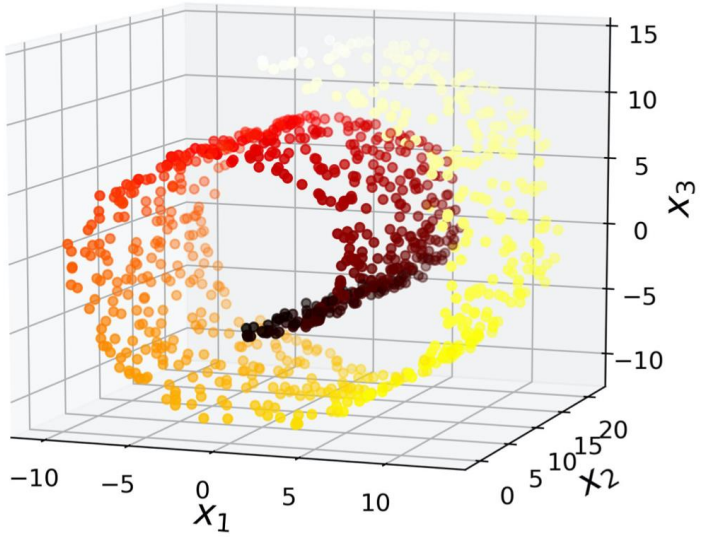
- Create a graph by connecting each instance to its nearest neighbors
- Then reduces dimensionality while trying to preserve the *geodesic distances*9 between the instances

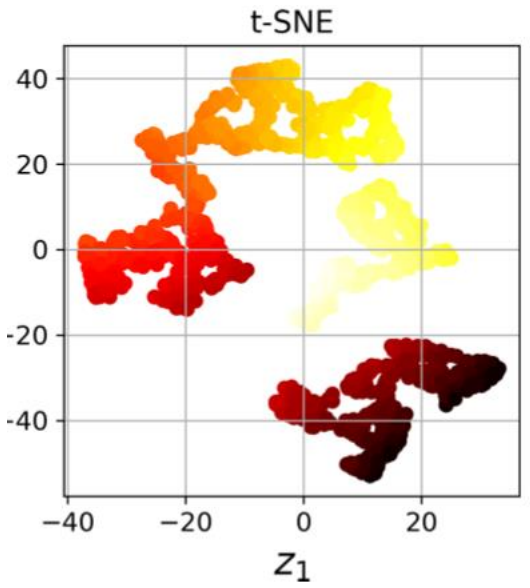




t-Distributed Stochastic Neighbor Embedding (t-SNE)

- Reduce dimensionality while trying to keep similar instances close and dissimilar instances apart
- It is mostly used for visualization





Introduction to Unsupervised Learning

Background

Reinforcement Learning (cherry)

Supervised Learning (icing)

Yann LeCun’ s Cake

Unsupervised Learning or Self-supervised Learning (the cake)

- Vast majority of data available today is unlabeled
- Unsupervised learning: When the input data is unlabeled, the algorithm exploits structures in the data without human teacher, e.g. dimensionality reduction
- The famous cake of Yann LeCun (a founding father of deep learning):
  - “If intelligence was a cake, unsupervised learning would be the cake, supervised learning would be the icing on the cake, and the reinforcement learning would be the cherry on the cake.”

Examples of unsupervised learning

- **Clustering**: To identify similar instances and assigning them to clusters, i.e. groups of similar instances
- **Anomaly detection** (or **outlier detection**): To learn what "normal" instances look like, and use this to detect abnormal instances
- **Density estimation**: To estimate the probability density function (PDF) of the random process that generated the dataset

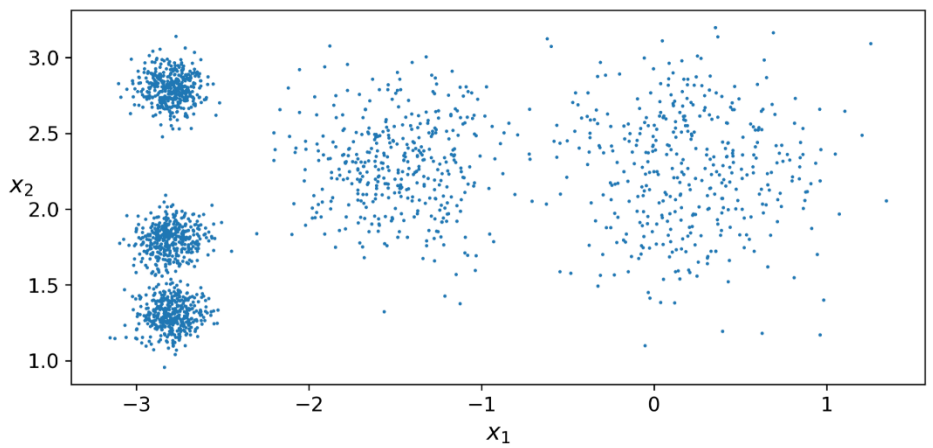
Types of clustering

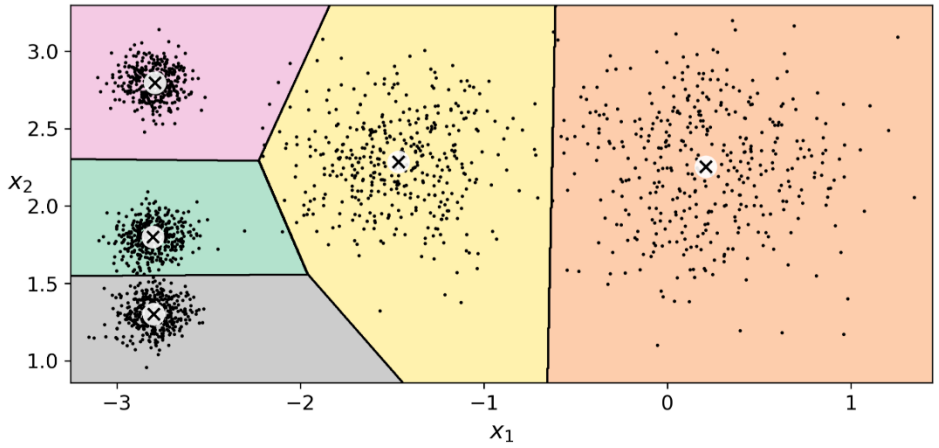
- **Centroid-based clustering**: organize data in clusters such that instances in a cluster are centered around a particular point, called a **centroid**, e.g. K-Means
- **Density-based clustering**: connect areas of high example density into clusters, e.g. DBSCAN
- **Distribution-based clustering**: assume that data are composed of distributions, e.g. Gaussian distribution
- **Hierarchical clustering**: creates a tree of clusters

- Applications of clustering:
  - Customer segmentation, e.g. for recommendation systems
  - Data analysis
  - Anomaly detection
  - Semi-supervised learning
  - Search engines
  - Image segmentation

K-Means

Key ideas of K-Means

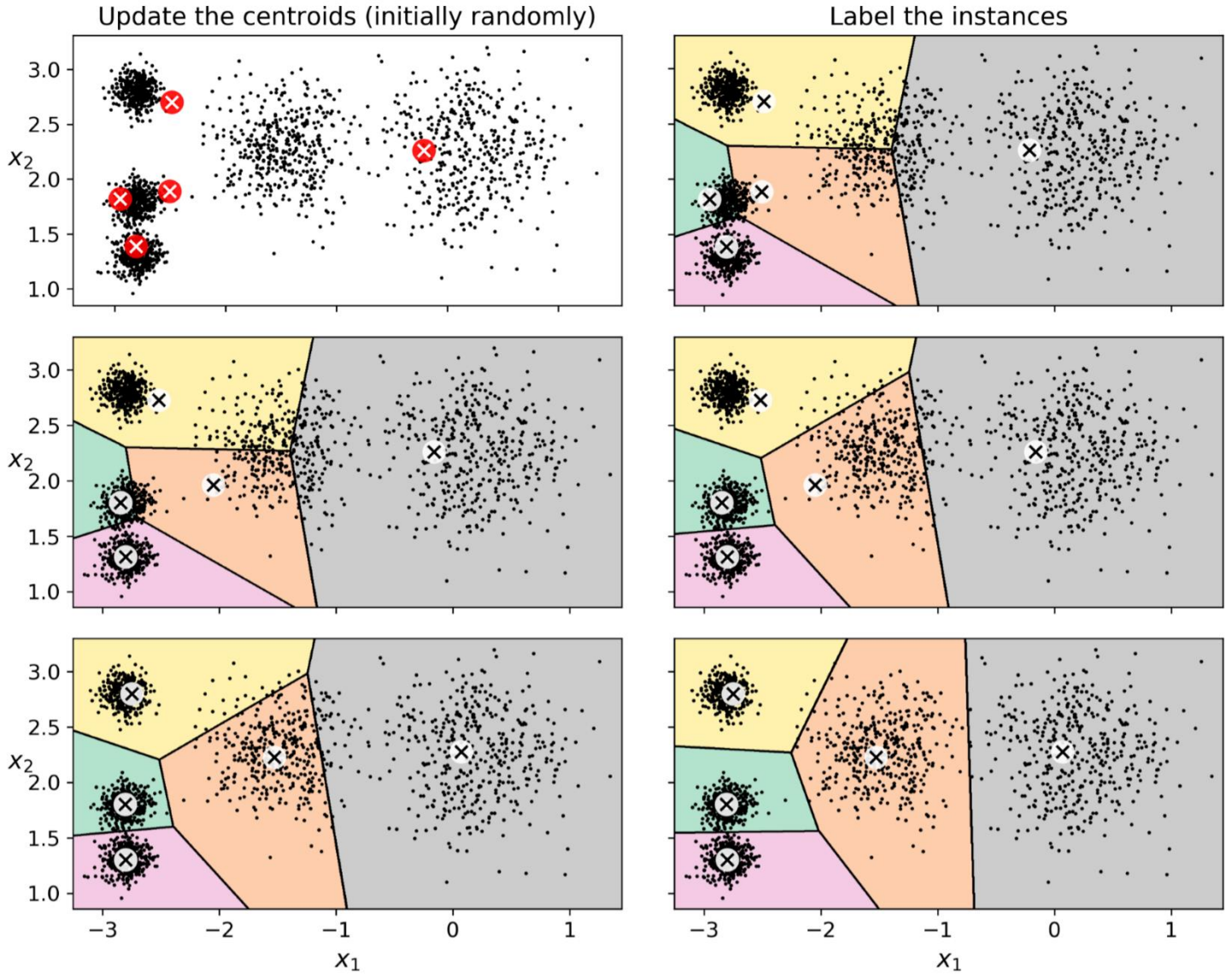




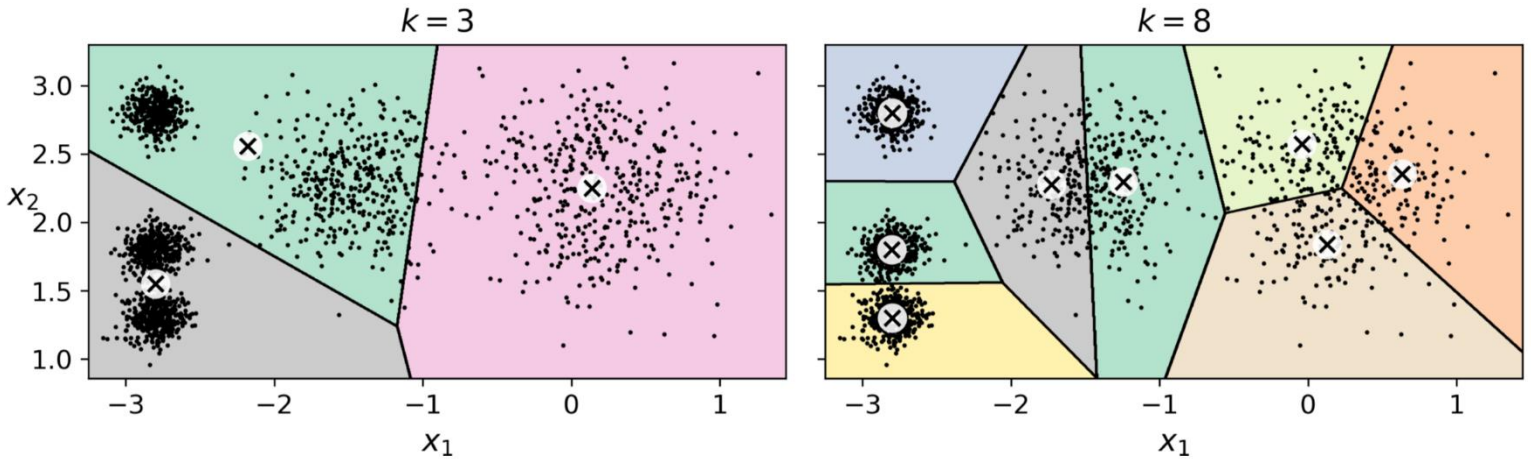
- **K-Means** is an iterative algorithm that partitions data into k pre-defined distinct non-overlapping clusters (each with a centroid), such that:
  - Within the same cluster, points are as similar as possible (i.e. homogeneous)
  - Between different clusters, points are as different as possible
  - **Goal**: The sum of the square distances between data points and its centroid is the minimum

The K-Means algorithm

- **Step 1**: Choose the value of k (i.e. the number of clusters)
- **Step 2 (Centroid initialization)**: Place the centroid randomly (e.g. by picking k instances at random and using their locations as centroids)
- **Step 3 (Label the instances)**: For each instance, compute its distance to the centroid of each of the clusters. If it is not in the cluster with the closest centroid, switch it to that cluster and update the centroids
- **Step 4**: Repeat Step 3 until convergence (i.e. the centroids stop moving)



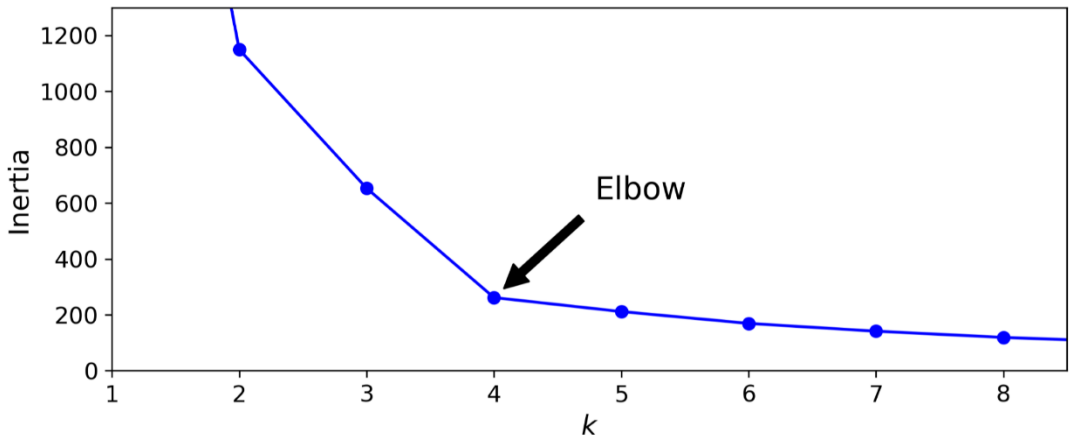
Finding the optimal value of k



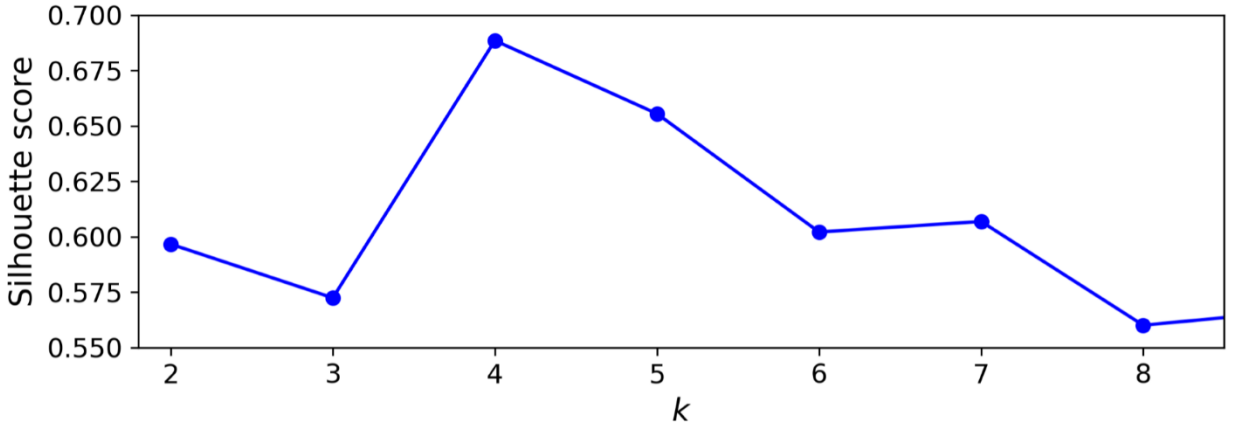
Inertia = 653.2

Inertia = 119.1

- The "elbow rule"
  - **Inertia**: the mean squared distance between each instance and its closest centroid
  - The bigger is k, the lower is the inertia
  - **Elbow**: any lower value of k is dramatic; any higher value of k doesn't help much



Finding the optimal value of k (continued)



- An instance' s **silhouette coefficient** is equal to $(b - a)/\max(a, b)$:
  - a is the mean distance to other instances in the same cluster
  - b is the mean distance to the instances of the next closest cluster
  - The silhouette coefficient varies between −1 and +1. What do values −1, 0, +1 mean, respectively?
- **Silhouette score** is the mean Silhouette coefficient over **all** instances
  - In the above plot, k = 4 is a good choice, but k = 5 is also good (much better than k = 6 or 7)

Strengths and limitations of K-Means

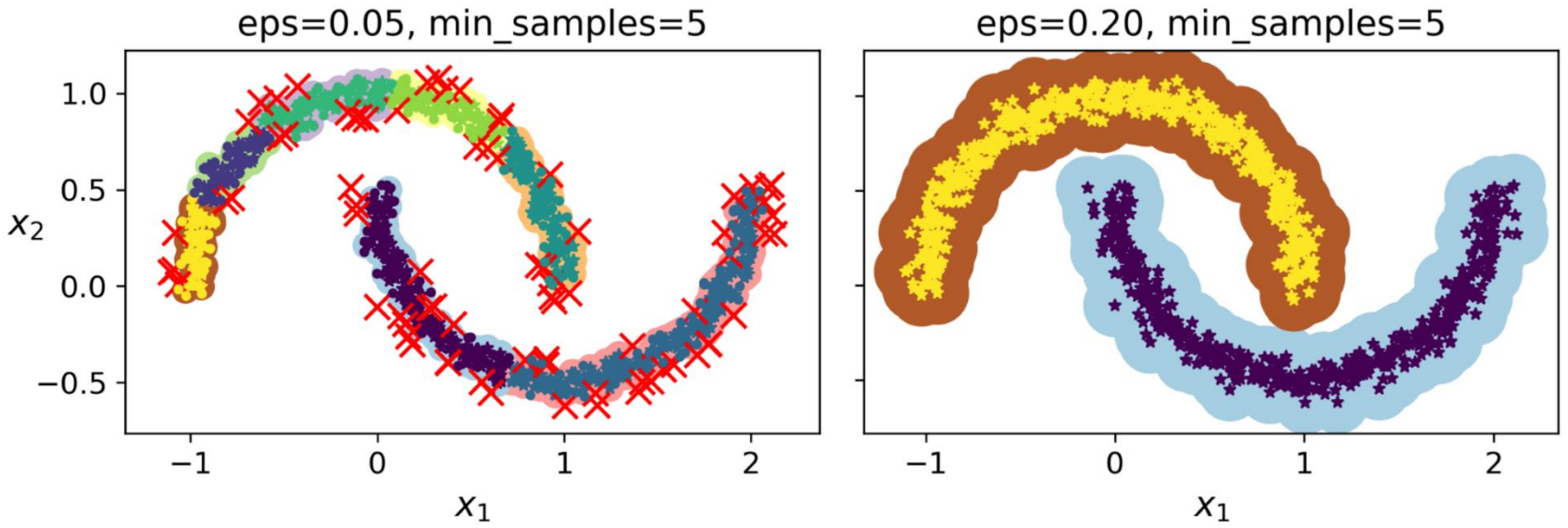
- **Strengths**
  - Easy to implement
  - Fast and scalable
  - Can produce tight clusters

- **Limitations**
  - Necessary but difficult to predict the value of k (i.e. the number of clusters)
  - Sensitive to the centroid initialization, order of data, scaling of data (normalization)
  - Not behave well when the clusters have varying sizes, different densities, or non-spherical shapes

Other Clustering Algorithms

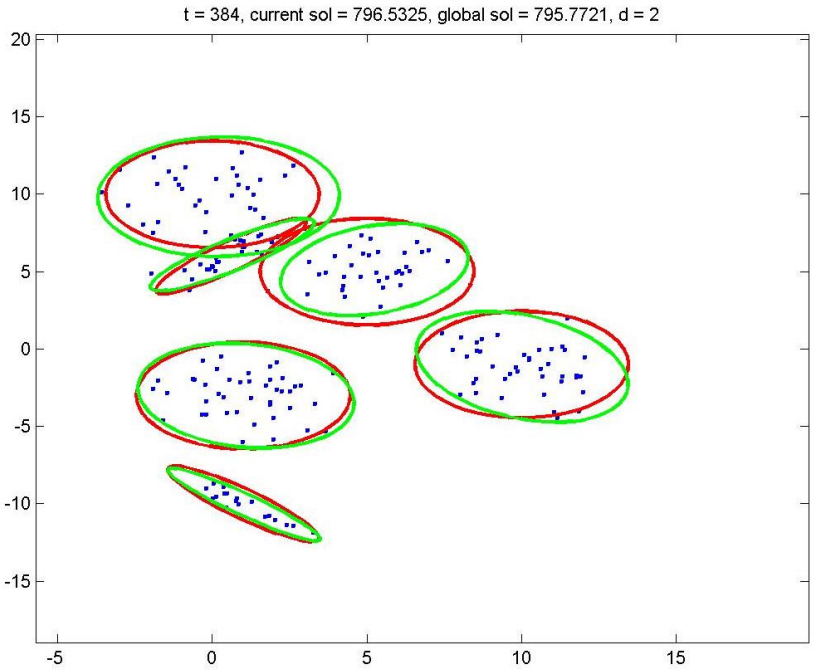
DBSCAN

- **DBSCAN (density-based spatial clustering of applications with noise)** algorithm defines clusters as continuous regions of high density:
  - For each instance, count how many instances are within a small distance ε (epsilon), forming the instance's ε-neighborhood
  - If an instance has at least min_samples instances in its ε-neighborhood (including itself), it is considered a core instance (i.e. located in a dense region)
  - All instances in the neighborhood of a core instance belong to the same cluster; other instances are considered anomalies



Gaussian Mixture Model (GMM)

- A **Gaussian Mixture Model (GMM)** is a probabilistic model that assumes that the instances were generated from a mixture of several Gaussian distributions
- Learning: If the number of distributions k is given, we can fit a set of k Guassian models to the data
  - The training of GMMs with latent variables (i.e. unobserved variables) is often done using the Expectation Maximization (EM) algorithm
- Example: The heights of males and females are both Gaussian. The heights of a group of people (without knowing genders in priori) can be described by a GMM of k=2 Gaussian distributions



Other clustering algorithms

- **Agglomerative clustering**: a hierarchy of clusters is built from the bottom up into a binary tree
- **BIRCH (balanced iterative reducing and clustering using hierarchies)** builds a tree containing just enough information to assign each new instance to a cluster, without storing all the instances in the tree, hence for large data
- **Mean-shift**
- **Affinity propagation**
- **Spectral clustering**

Summary

- In this lecture, we learned several techniques of unsupervised learning:
  - Dimensionality reduction (e.g. PCA)
  - Clustering algorithms, e.g. K-Means, DBSCAN
  - Gaussian mixture models (GMMs) can be used for clustering, anomality detection, etc.

References

- Géron, A. (2019). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media.
- Van Der Maaten, L., Postma, E., & Van den Herik, J. (2009). Dimensionality reduction: a comparative. *J Mach Learn Res*, *10*(66-71), 13.
- http://research.cs.tamu.edu/prism/lectures/iss/iss_l10.pdf