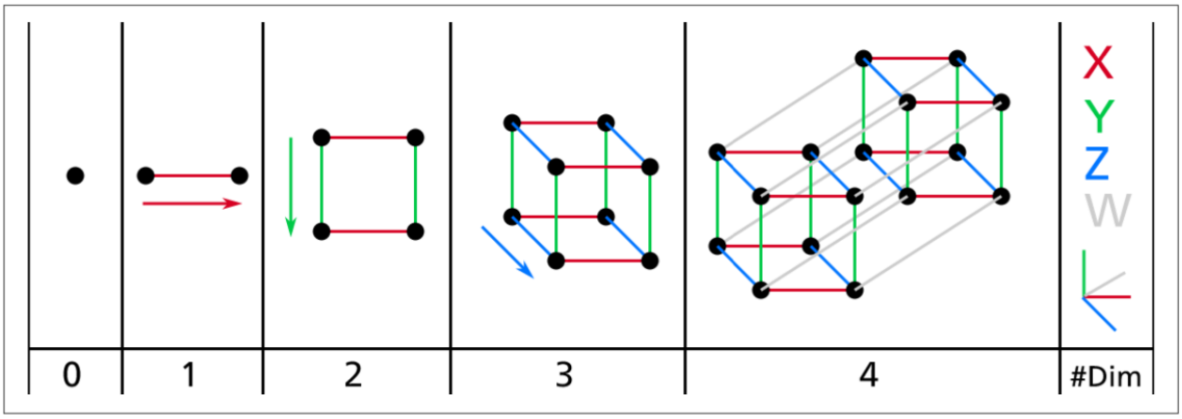
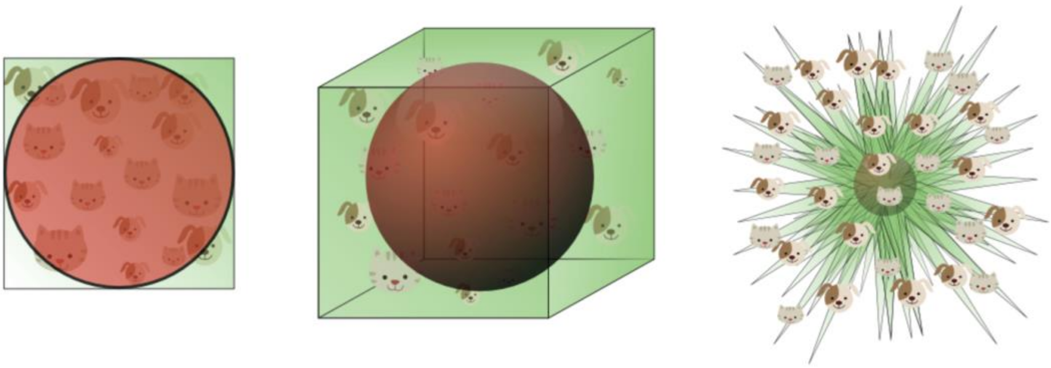
Dimensionality Reduction

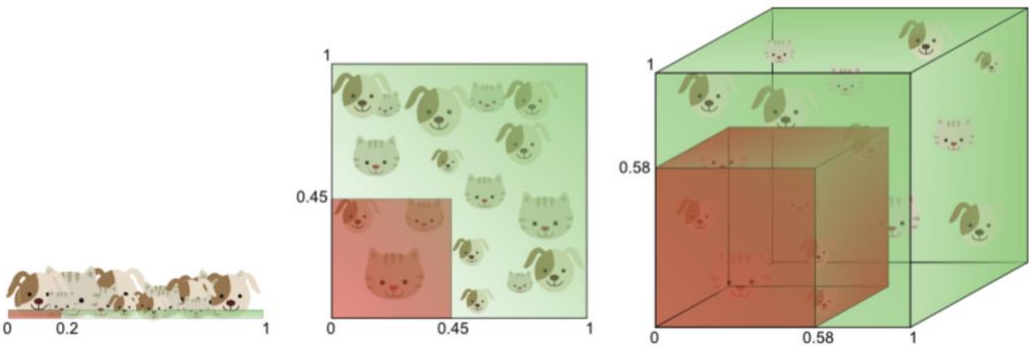


Problem: curse of dimensionality

- The higher the dimension is, the more likely it would be to get close to the border.



- Risk of being sparse in high-dimensional datasets
  - More dimensions == overfitting



- Solution:
  - Increase datasets' size – sufficient density
    - Fact: exponentially (size and dimensionality)
  - Feature selection algorithms : ensemble learning (Random Forest)
  - Feature extraction algorithms : projection (PCA)

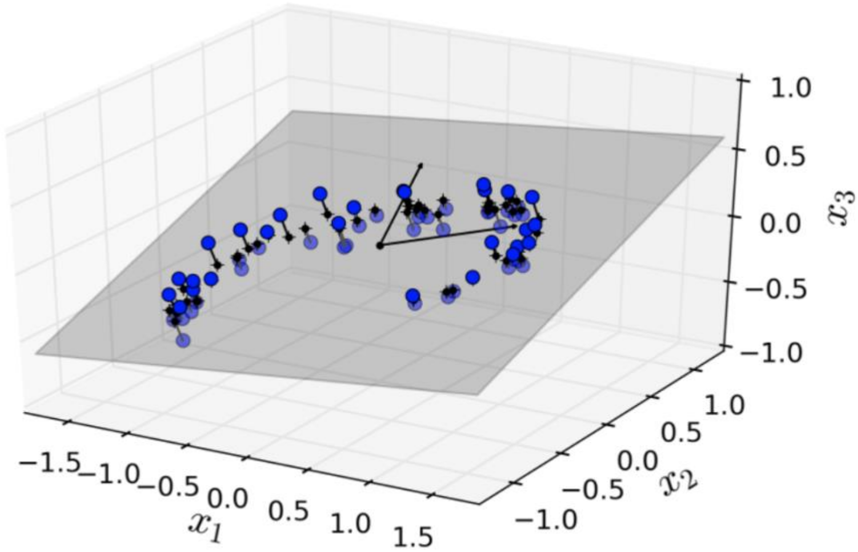
Pros and cons:

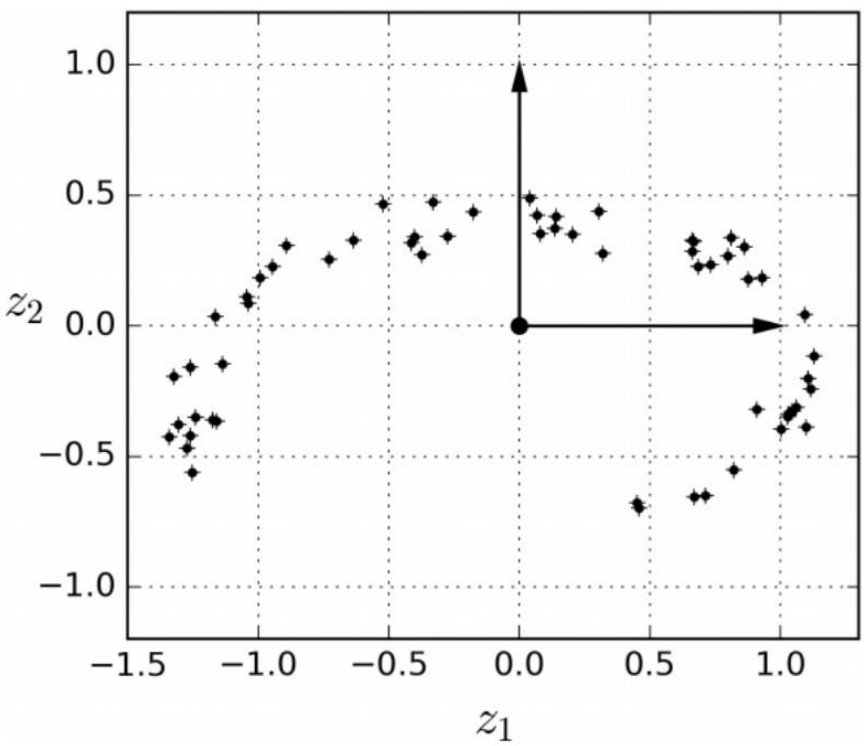
- Pros
    - Filter noise and unnecessary detail – higher performance
    - Speed up training
- Cons
    - Lose information – worse performance

Two Methods：

(1) Projection

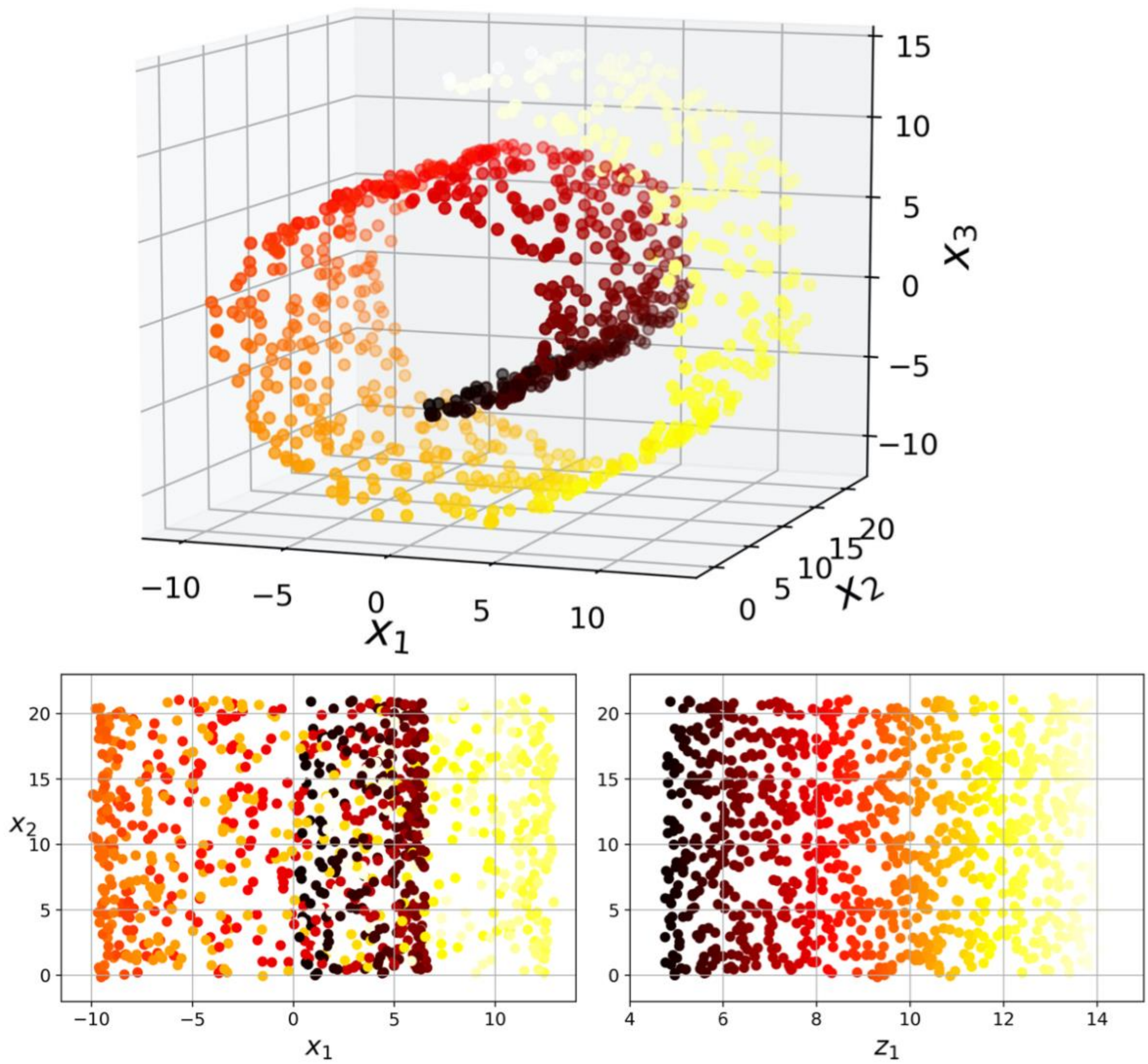
- Features: constant or highly-correlated
- Fearture instance lie within a much lower-dimensional subspace of the high-dimensional space





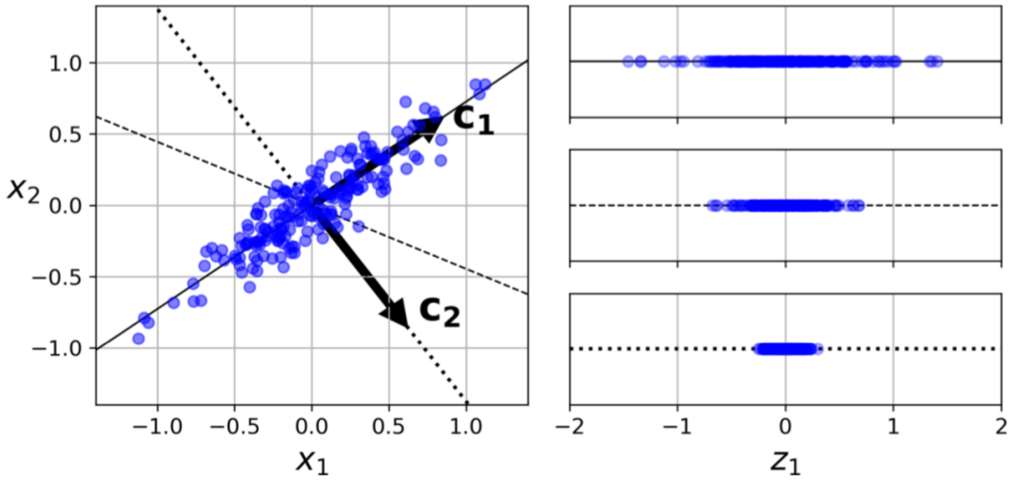
(2) **Manifold Learning**

- Manifold hypothesis: most real-world high-dimensional datasets lie close to a much lower-dimensional manifold.
- Implicit assumption: the task at hand (e.g., classification or regression) will be simpler if expressed in the lower-dimensional space of the manifold.



Projection:

- PCA (Principal Component Analysis)
  - **Essence**:
    - Indentify the hyperlane that lies closest to the data
    - Projects the data onto it
  - **evaluation:**
    - select the axis that preserves the maximum amount of variance
    - select the axis that minimizes the mean squared distance between the original dataset and its projection onto that axis.



- **Process:**
  - identify the axis that accounts for the largest amount of variance
  - find a second axis, orthogonal to the first one, that accounts for the largest amount of remaining variance
  - if high dimension, find the third one, orthogonal to both previous axes until as many as the number of dimensions in the dataset
- **Standard matrix factorization technique:**
  - **SVD(Singular Value Decomposition)**
    - Decompose the training set matrix $\mathbf{X}$ into the matrix multiplication of three matrices $\mathbf{U}\,\Sigma\,\mathbf{V}^T$, where $\mathbf{V}$ contains all the **principal components**.

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

- **Project to hyperplane:**
  - $\mathbf{X}_{d\text{-proj}} = \mathbf{X}\mathbf{W}_d$
  - compute the matrix multiplication of the training set matrix X by the matrix $\mathbf{W}_d$
  - $\mathbf{W}_d$: the matrix containing the first d principal components (i.e., the matrix composed of the first d columns of $\mathbf{V}$)
- **Inverse transformation:**
  - **Retore the original data**
  - $\mathbf{X}_{\text{recovered}} = \mathbf{X}_{d\text{-proj}}\mathbf{W}_d^{\,T}$
  - **Reconstruction error: mean squared distance**
- **Variants:**
  - **Randomized PCA: speed up computation**
    - Its computational complexity is $O(m \times d^2) + O(d^3)$, instead of $O(m \times n^2) + O(n^3)$ for the full SVD approach
  - **Incremental PCA: lower the burdon of memory**
    - split the training set into mini-batches and feed an IPCA algorithm one mini-batch at a time