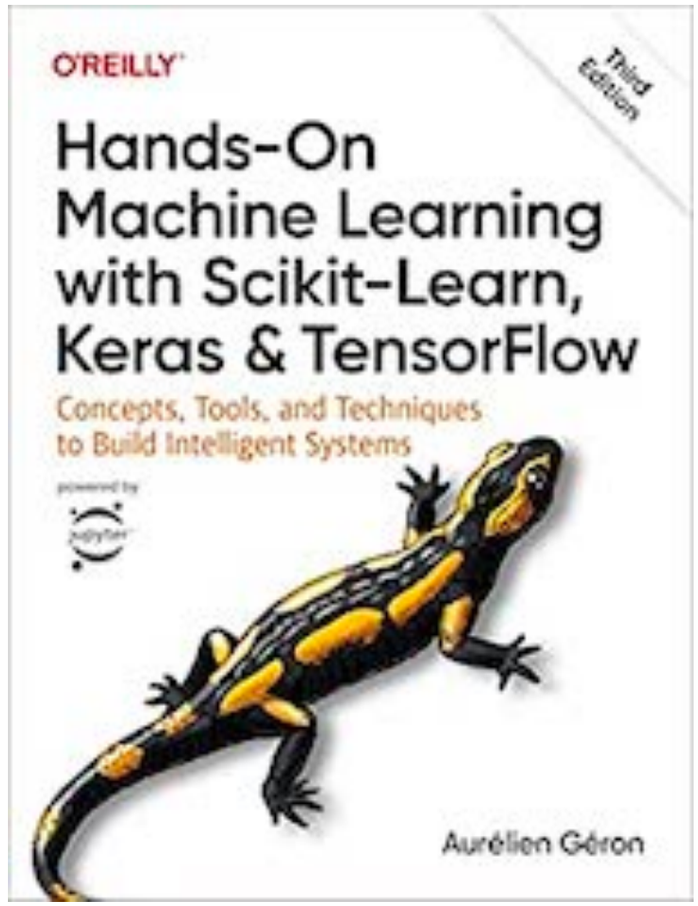
Machine Learning Security

8 Dimensionality Reduction



Revised Oct 6, 2023

Topics

- The Curse of Dimensionality
- Main Approaches for Dimensionality Reduction
- PCA
- Random Projection
- LLE
- Other Techniques for Dimensionality Reduction

High Dimensionality

- On iPhone 15 models, the Main camera resolution is set to 24 MP by default. You can switch between 12 MP, 24 MP, and 48 MP.
- So a single iPhone image has 24 million pixels, each with three colors
- Consider the new Mercedes self-driving car
  - A pair of **lidar** sensors ... a hump just above the rear window for the **GPS** antenna array ... a **front camera** for **3D** image capture, **long-range radar** sensors up front that measure speed and distance, **ultrasonic sensors** that detect the car's surroundings, a **driver monitoring camera** inside, and a **road moisture sensor** ... **A rear-facing camera** keeps an eye out for flashing lights from emergency vehicles, and i**nternal microphones** listen for emergency sirens.
  - https://www.theverge.com/2023/9/27/23892154/mercedes-benz-drive-pilot-autonomous-level-3-test

High Dimensionality

- How can you train or use a model with millions of features in the input data?
- Do you really need all that data?
- **Dimensionality reduction**
  - Discarding the unimportant data
  - Keeping only the useful data

Dimensionality Reduction

- Two main methods
  - **Projection**
  - **Manifold learning**
- Three popular techniques
  - **Principal Component Analysis (PCA)**
  - **Random projection**
  - **Locally Linear Embedding (LLE)**

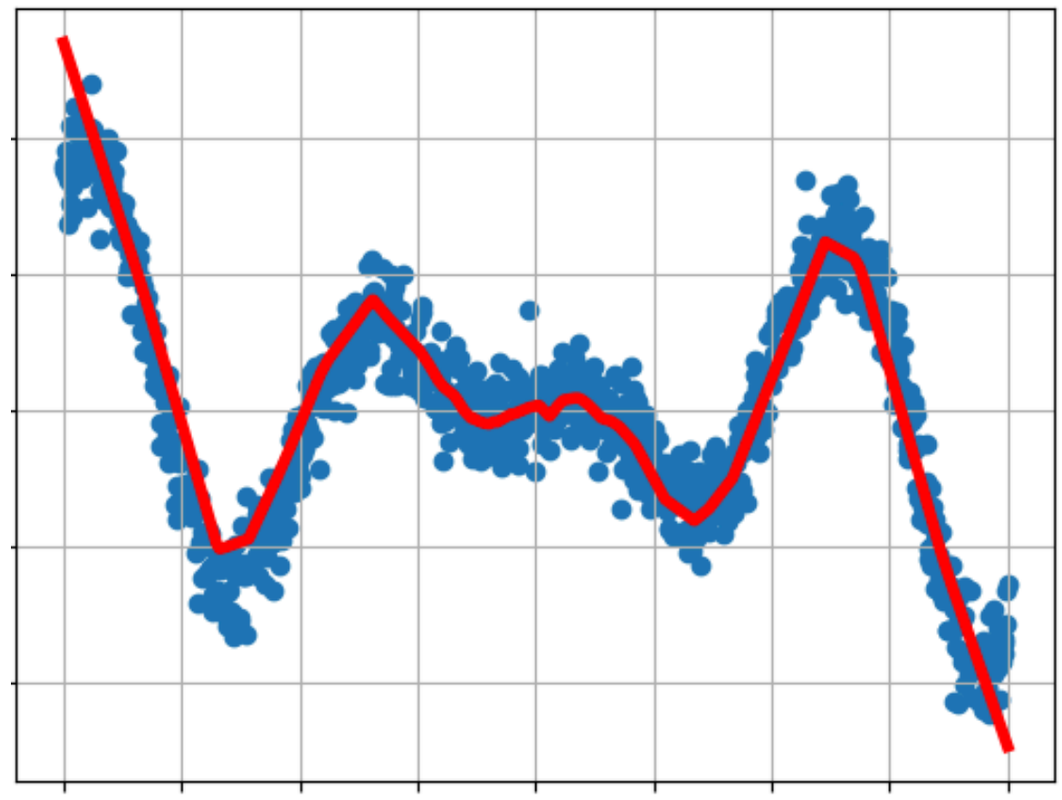
The Curse of Dimensionality

High-Dimensional Spaces

- Consider a 1x1 square
  - Most points are in the middle
  - Not close to the edge
- How about a hypercube with many dimensions?
  - Most of the points are near an edge

Two Dimensions

- You need enough data to define the trend
- If you had only 2 or 3 points, you couldn't make a good model
  - The data can't be too ***sparse***
- If all the points were at the left and right end of X you couldn't define the trend
  - Points too near the edge



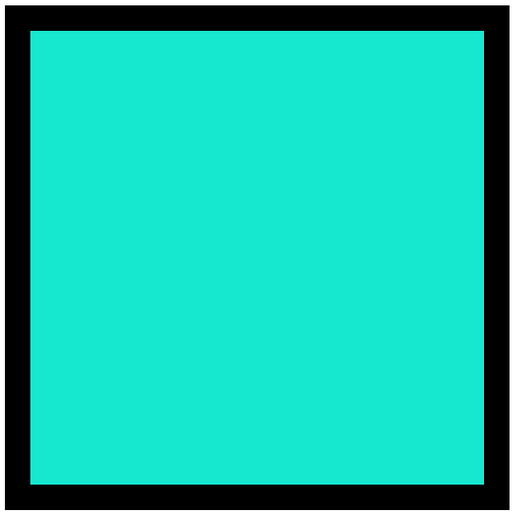
Points Near an Edge

```
import random
random.seed(999)
print("Dimensions \t Edge points out of 1000")
for dimensions in [2,3,10,30,100,300,1000,3000,10000]:
  edge_points = 0
  for p in range(1000):
    on_edge = 0
    for x in range(dimensions):
      if random.random() < 0.01:
        on_edge = 1
    edge_points += on_edge
  print(dimensions, "\t\t", edge_points)
```

```
Dimensions          Edge points out of 1000
2                   25
3                   29
10                  102
30                  250
100                 621
300                 956
1000                1000
3000                1000
10000               1000
```

Volume of a Hypercube

- Consider a 1x1 hypercube in ***d*** dimensions
- Its volume is $1^{d} = 1$
- How much of the volume is not near an edge?
- Range 0.01 through 0.99
- A hypercube 0.98 on a side
- Volume = 0.98^{d}



d	Volune
2	0.9604000000
3	0.9411920000
100	0.1326195559
300	0.0023325057
1000	0.0000000017

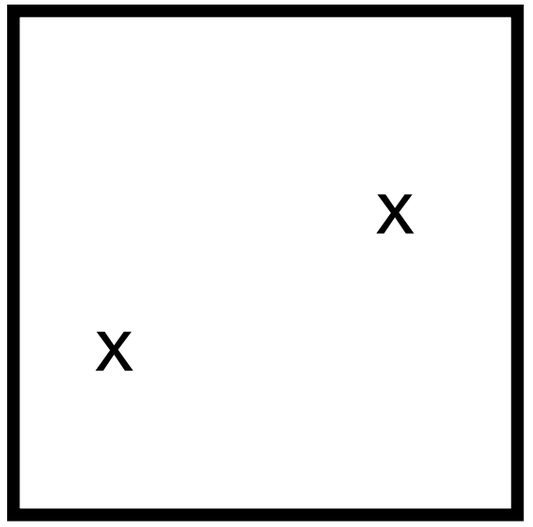
Distance Between Points

```
import random, math

random.seed(999)
print("Dimensions \t Avg Distance Between Points")
for dimensions in [2, 3, 10, 100, 1_000, 10_000, 100_000, 1_000_000]:
  total_distance = 0
  for p in range(100):
    dist2 = 0
    for x in range(dimensions):
      dist2 += (random.random() - random.random())**2
    total_distance +=  math.sqrt(dist2)
  formatted_dimensions = "{:,}".format(dimensions)
  print(formatted_dimensions.rjust(9), "\t\t", "{:.2f}".format(total_distance/100.0))
```

```
   Dimensions         Avg Distance Between Points
            2                  0.57
            3                  0.69
           10                  1.21
          100                  4.06
        1,000                 12.92
       10,000                 40.86
      100,000                129.09
    1,000,000                408.26
```

Distance Between Points



- Consider a 1x1 hypercube in ***d*** dimensions
- A typical distance between points in one dimension should intuitively be 0.5
- How far apart are such points in 3 dimensions?
  - sqrt($0.5^2 + 0.5^2 + 0.5^2$) = 0.86
- How far apart are such points in ***n*** dimensions?
  - sqrt(0.5^2 x ***n***)

d	Distance
2	0.71
3	0.87
100	5.00
300	8.66
1000	15.81
10,000	50.00
100,000	158.11
1,000,000	500.00

Sparse Data

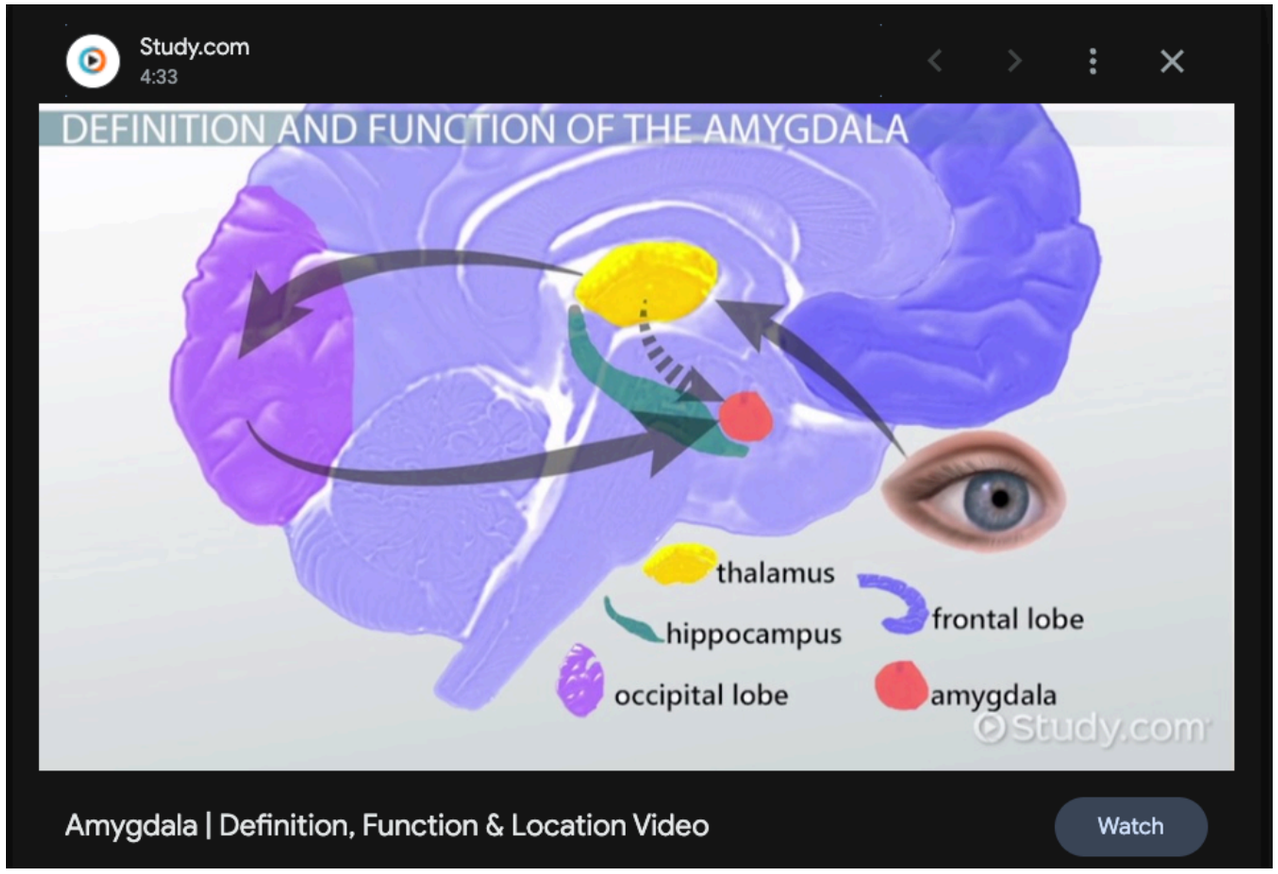
- In high dimensions
  - Points are far apart
  - A new point will be an outlier, far from the others
  - Predictions will be less reliable, since they are based on larger extrapolations
- Overfitting is very likely

Sample Size

- Consider 100 dimensional hypercube from 0 to 1
- Break it into hypercubes 0.1 on a side
- There are 10^{100} hypercubes
- You can never sample them all, or even close to it

Human Vision

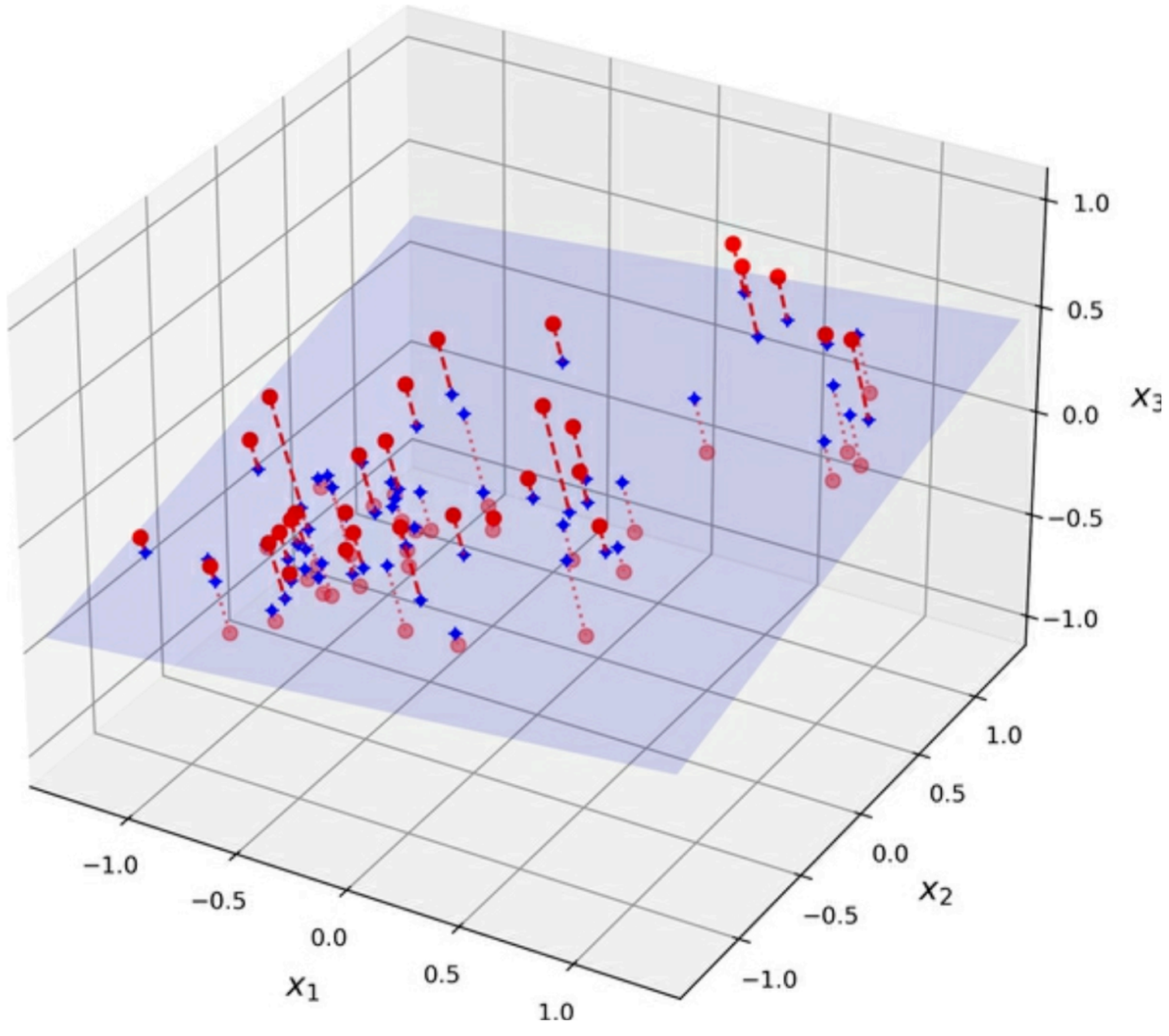
- Brain structures filter visual data
- Discarding the vast majority of it
- Paying attention only to the important data



Main Approaches for Dimensionality Reduction

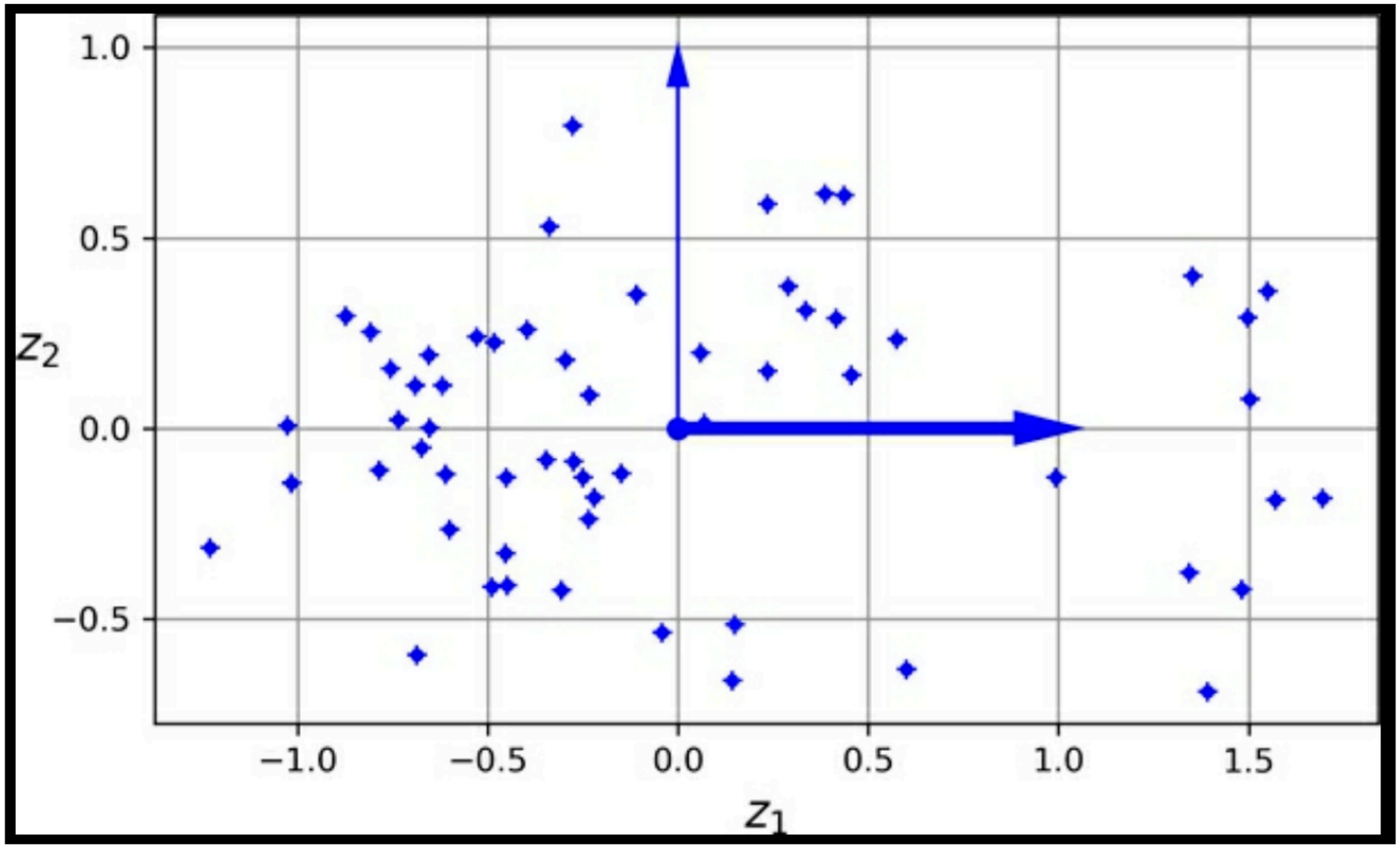
Projection

- In most real-world problems
- Samples are not uniformly dispersed
- They fall close to a lower-dimensional subspace of the high-dimensional space
- Here 3-D data lies near a 2-D plane
- **Project** each instance down to the plane



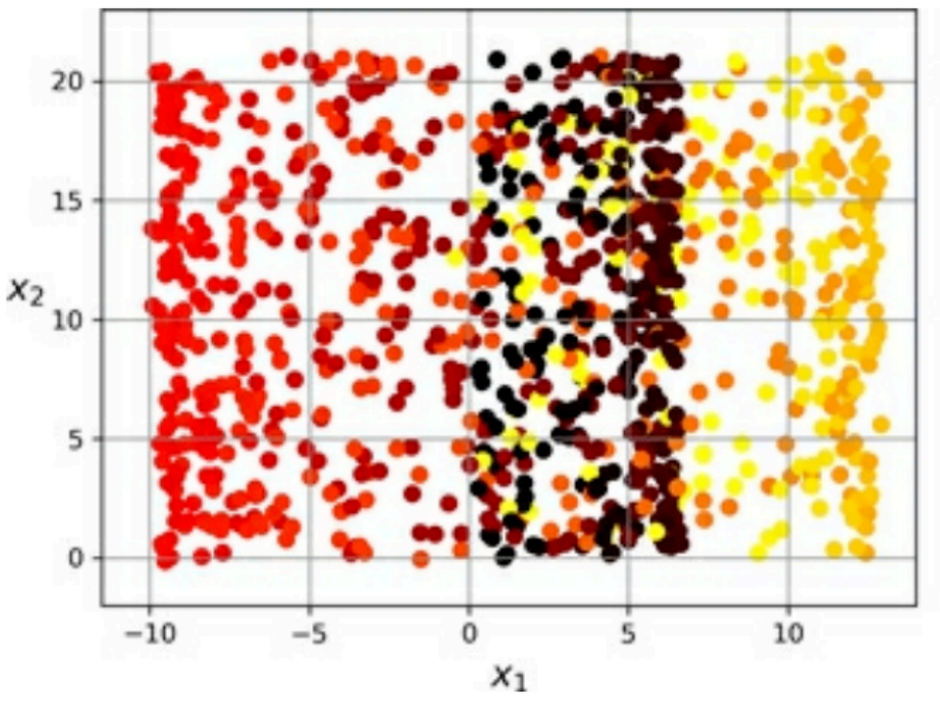
After Projection

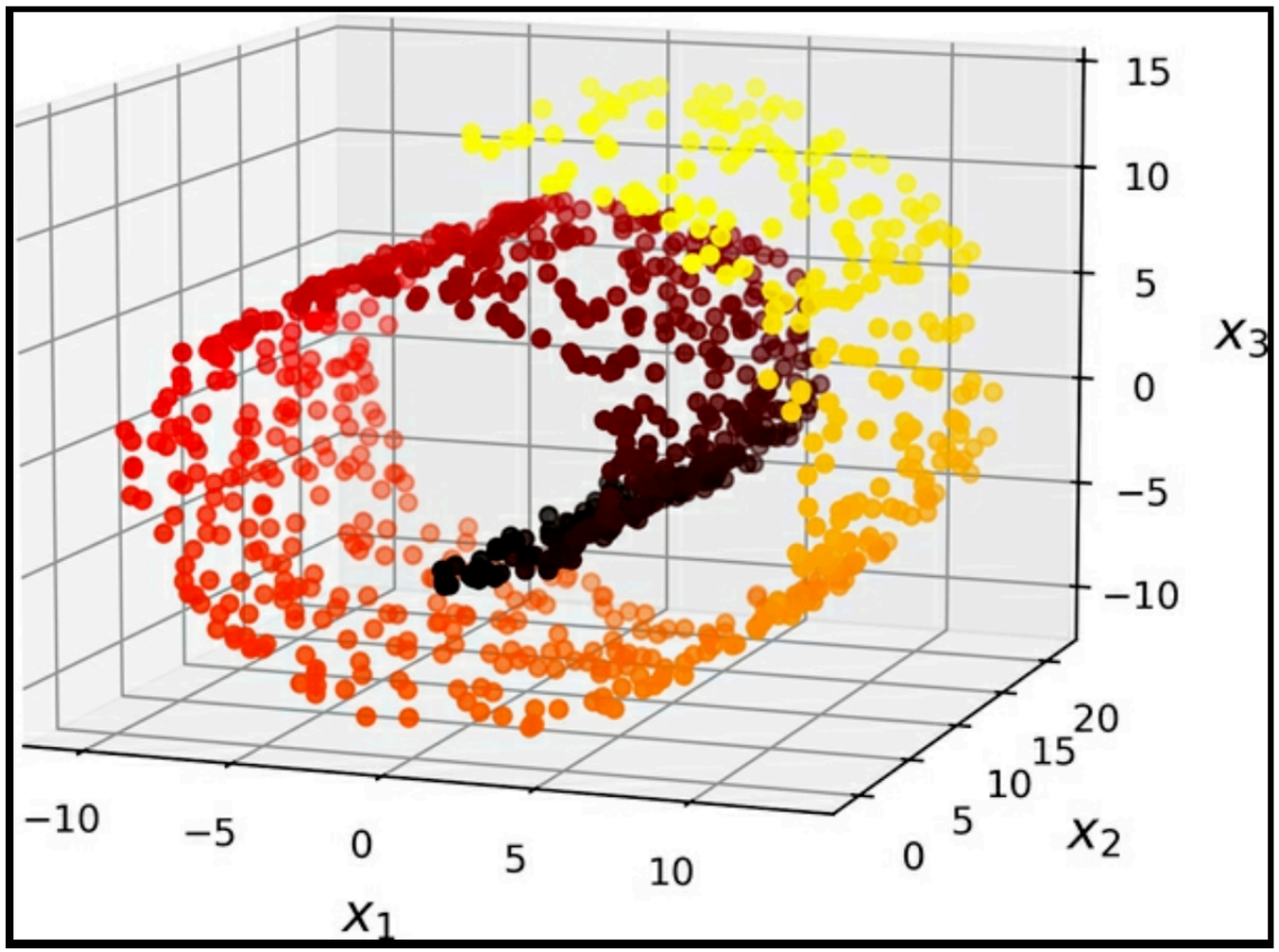
- 3-D is reduced to 2-D



Manifold Learning

- The subspace isn't flat
- **Projecting** it onto a plane mixes the colors together

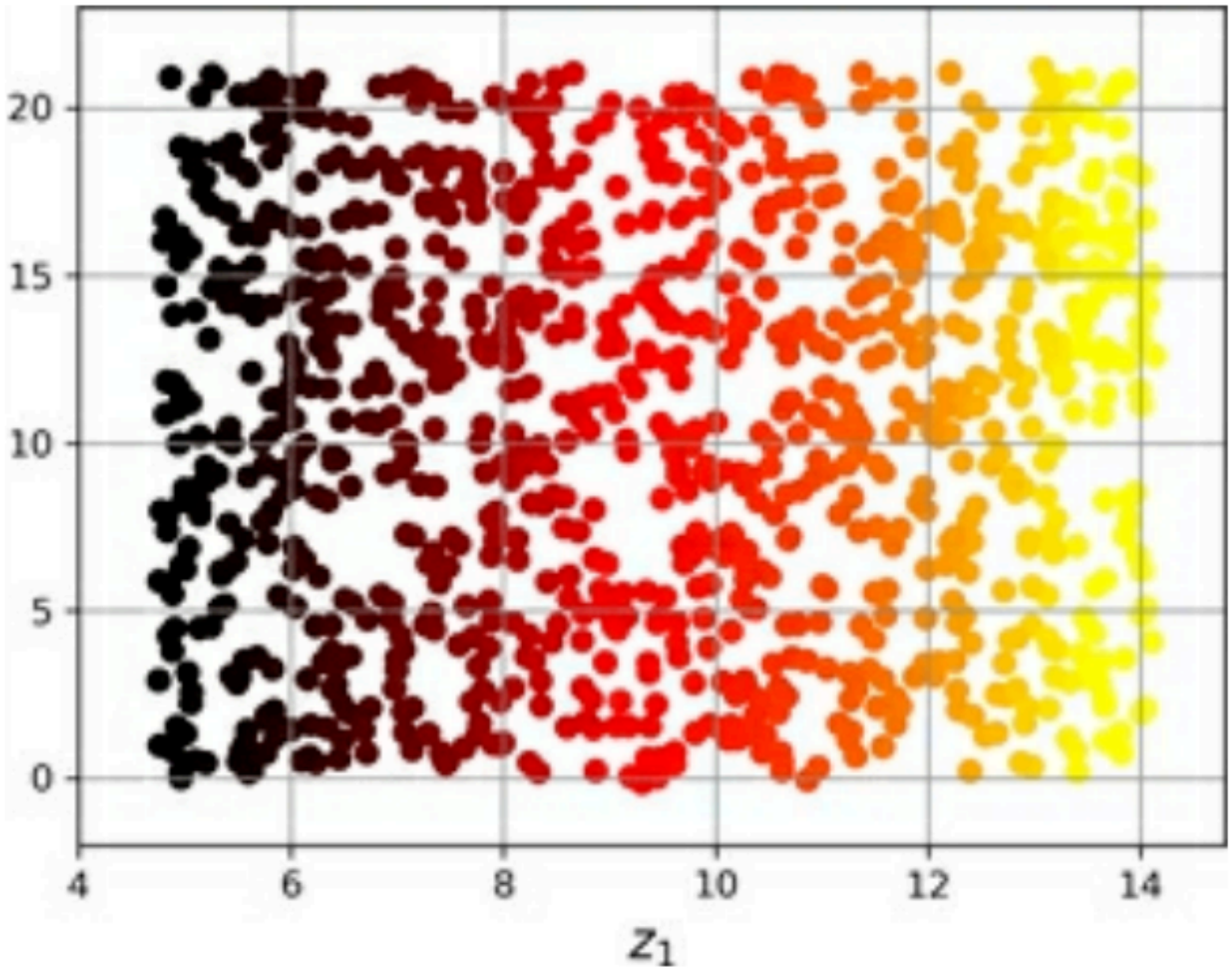




Manifold Learning

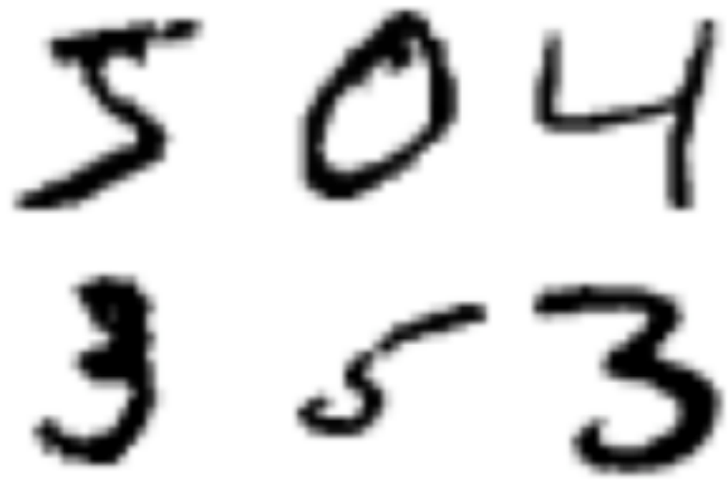
- A 2-D manifold is a 2-D shape that can be bent and twisted in 3-D space

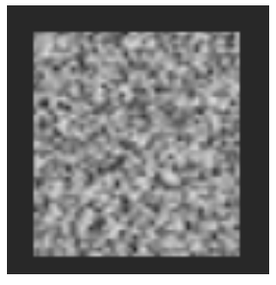
- Unroll the manifold to get the figure to the right



MNIST Data

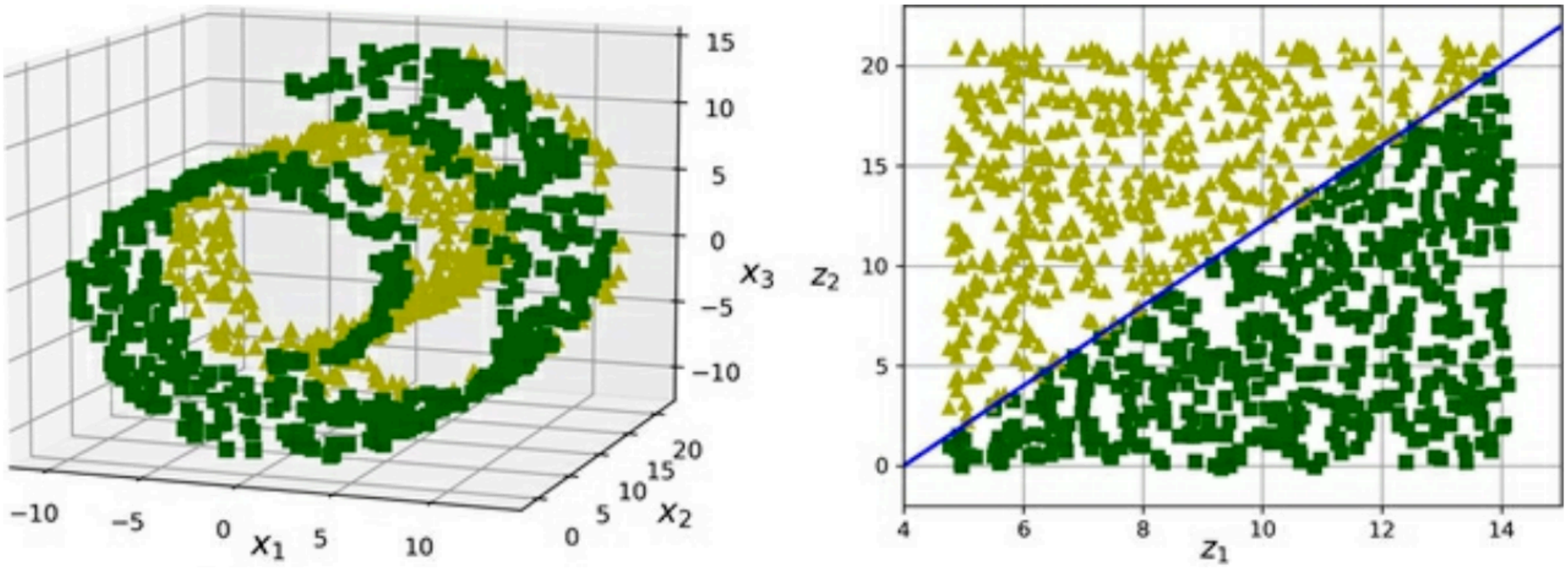
- Instances are handwritten digits
  - 28x28 pixels grayscale
  - 784 dimensions
- But random pixels look very different
- The data has far less variation than 784 dimensions





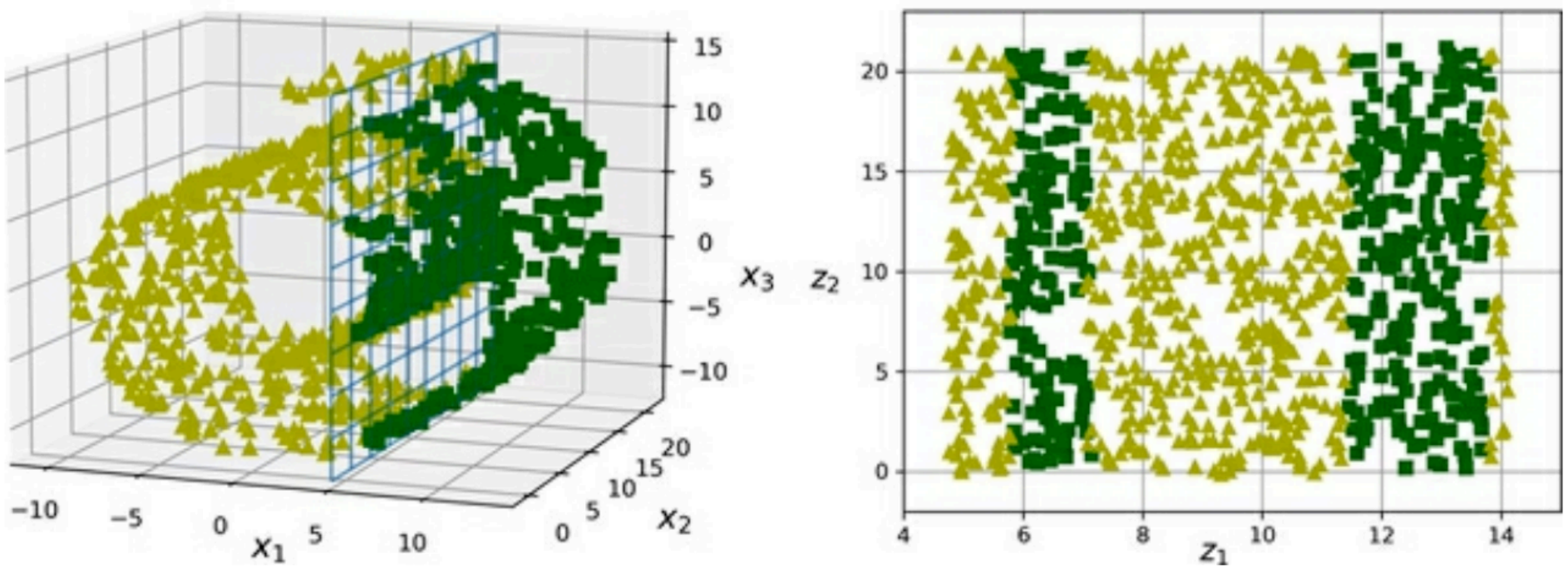
Decision Boundary

- Here, the decision boundary is simpler in 2-D than in 3-D



Decision Boundary

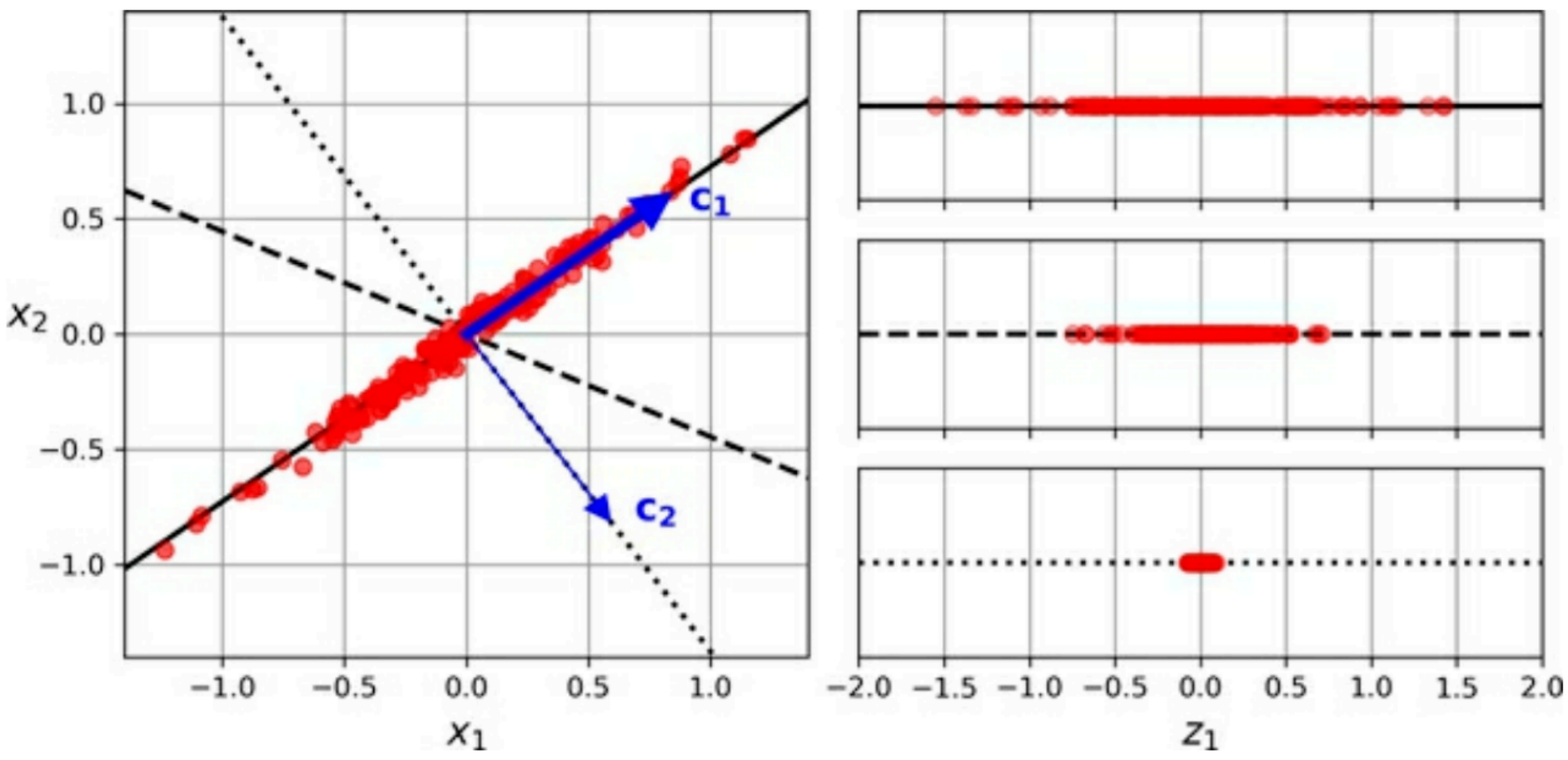
- Here, a decision boundary that is simple in 3-D becomes more complex in 2-D
- Reducing dimensionality speeds up training
  - But may not lead to a better or simpler solution



PCA

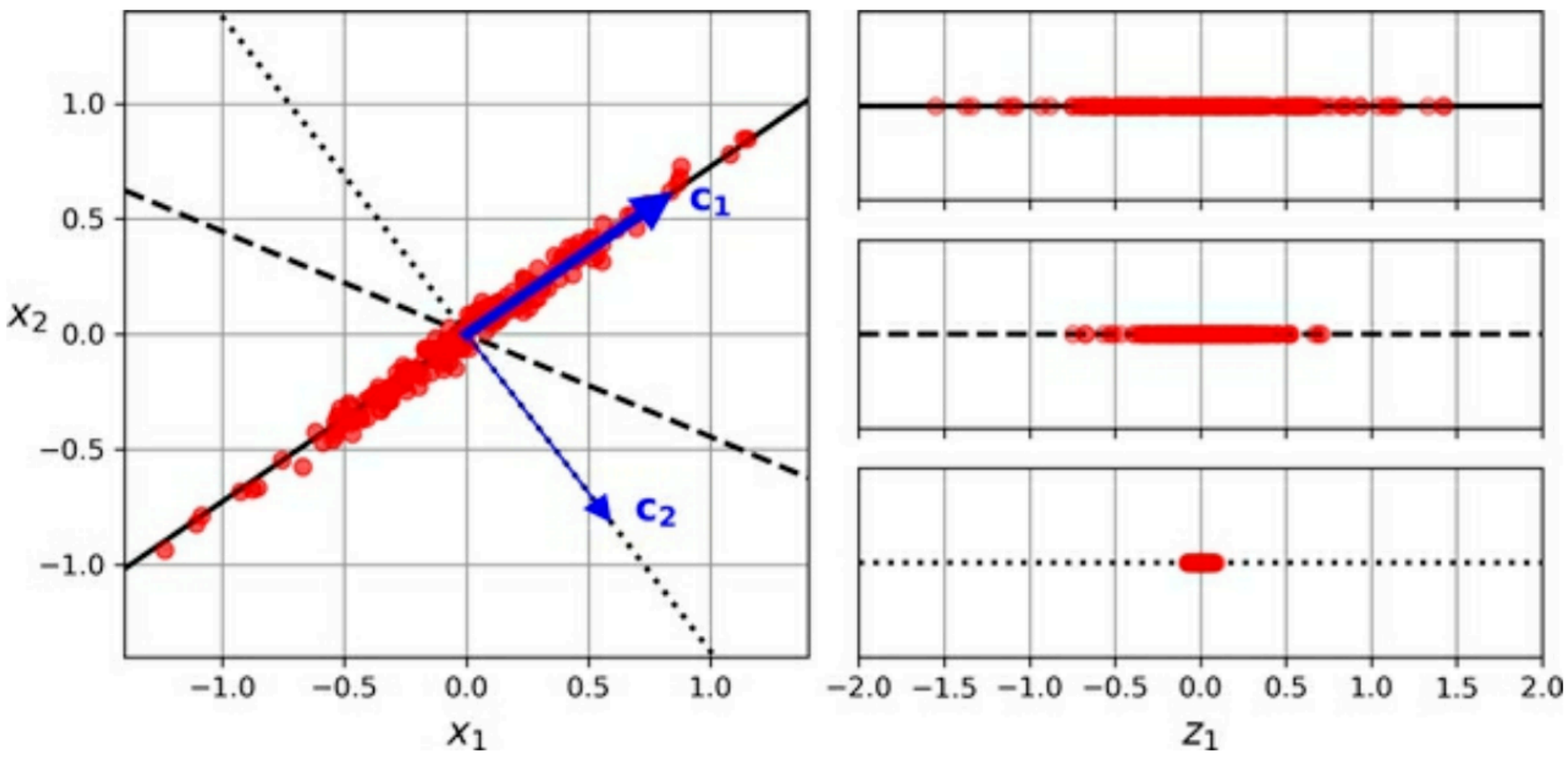
Principal Component Analysis (PCA)

- By far, the most popular dimensionality reduction algorithm
- First, find the hyperplane that lies closest to the data
- Then project the data onto it



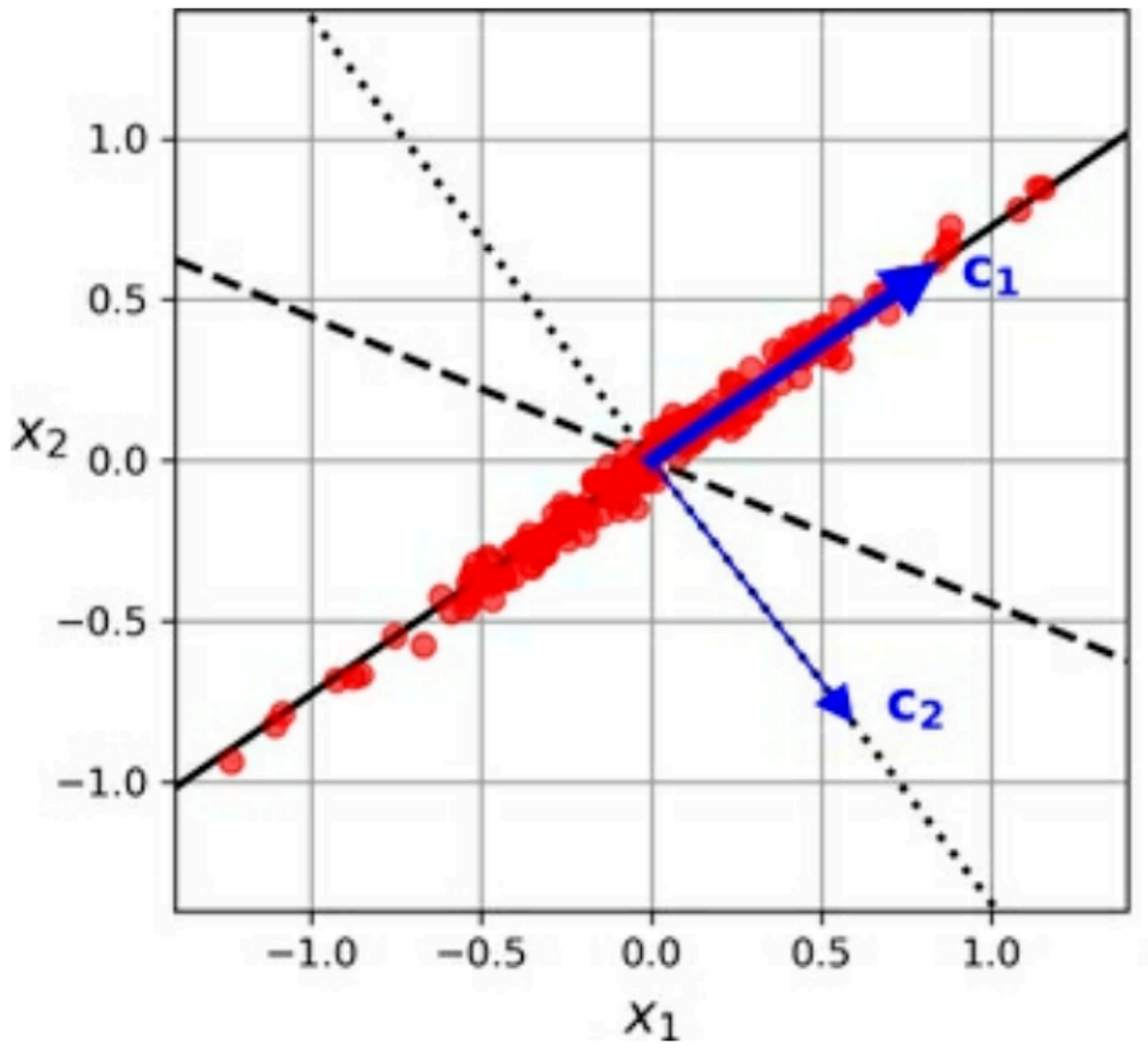
Preserving Variance

- Use the plane that preserves the most variance
- Or, equivalently, minimize the mean squared distance from the samples and the plane



Principal Components

- Use the plane $\mathbf{C_1}$ that preserves the most variance
- Then find a second axis $\mathbf{C_2}$
  - Orthogonal to the first one
  - That captures most of the remaining variance $\mathbf{C_1}$



Singular Value Decomposition

- A standard matrix factorization technique
- Decomposes the training set matrix **X**
  - Into the multiplication of three matrices
  - One of them is **V** -- unit vectors that define the principal components

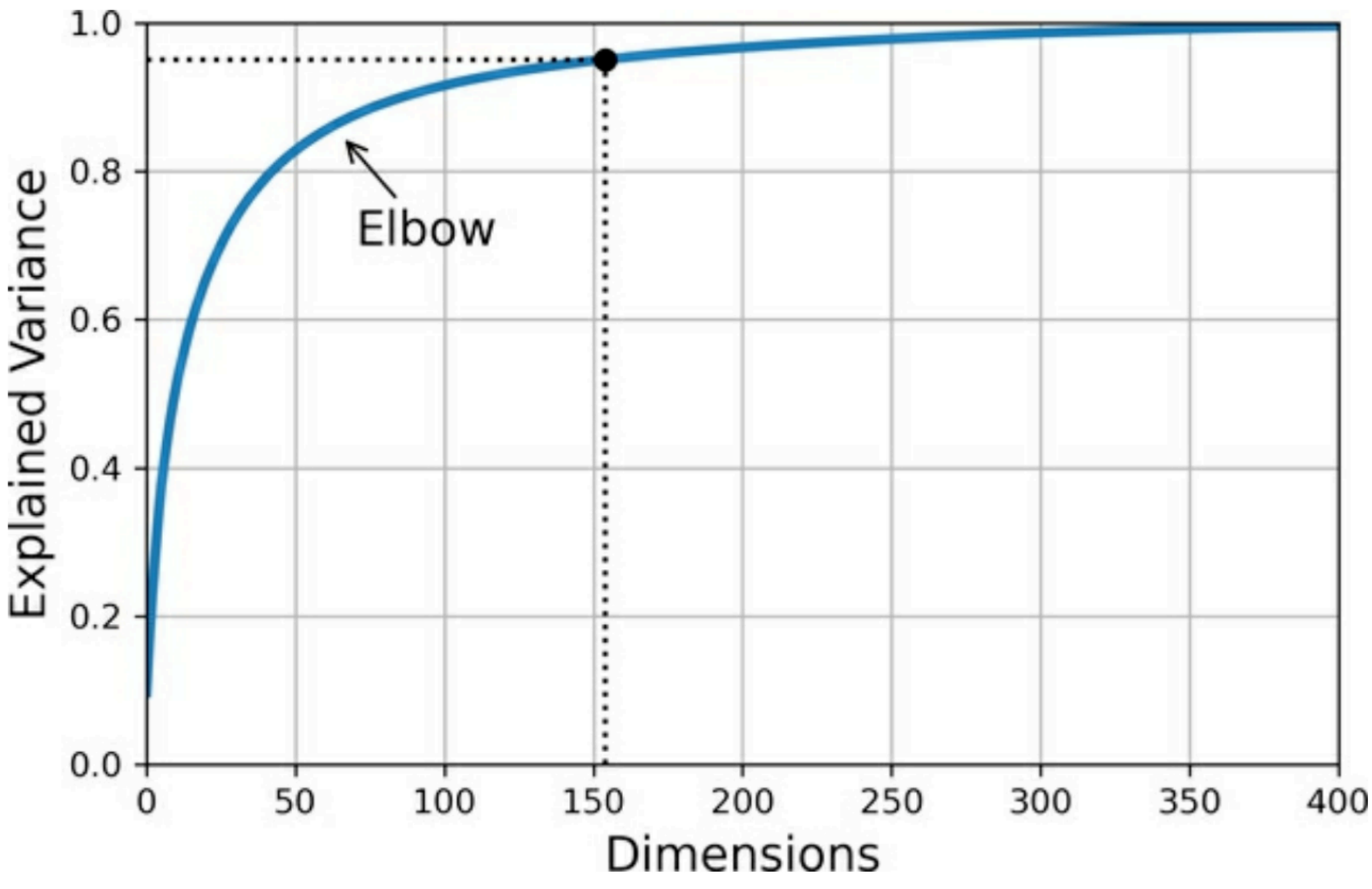
$$\mathbf{V} = \begin{pmatrix} | & | & & | \\ \mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_n \\ | & | & & | \end{pmatrix}$$

Explained Variance Ratio

- The proportion of the dataset's variance that lies along each principal component
- **Choosing the Right Number of Dimensions**
  - Use enough dimensions to capture enough variance
    - Such as 95%
- Or reduce to 2 or 3 dimensions just so you can visualize the data

Elbow in the Curve

- The explained variance stops growing fast at the elbow
- A logical place to stop



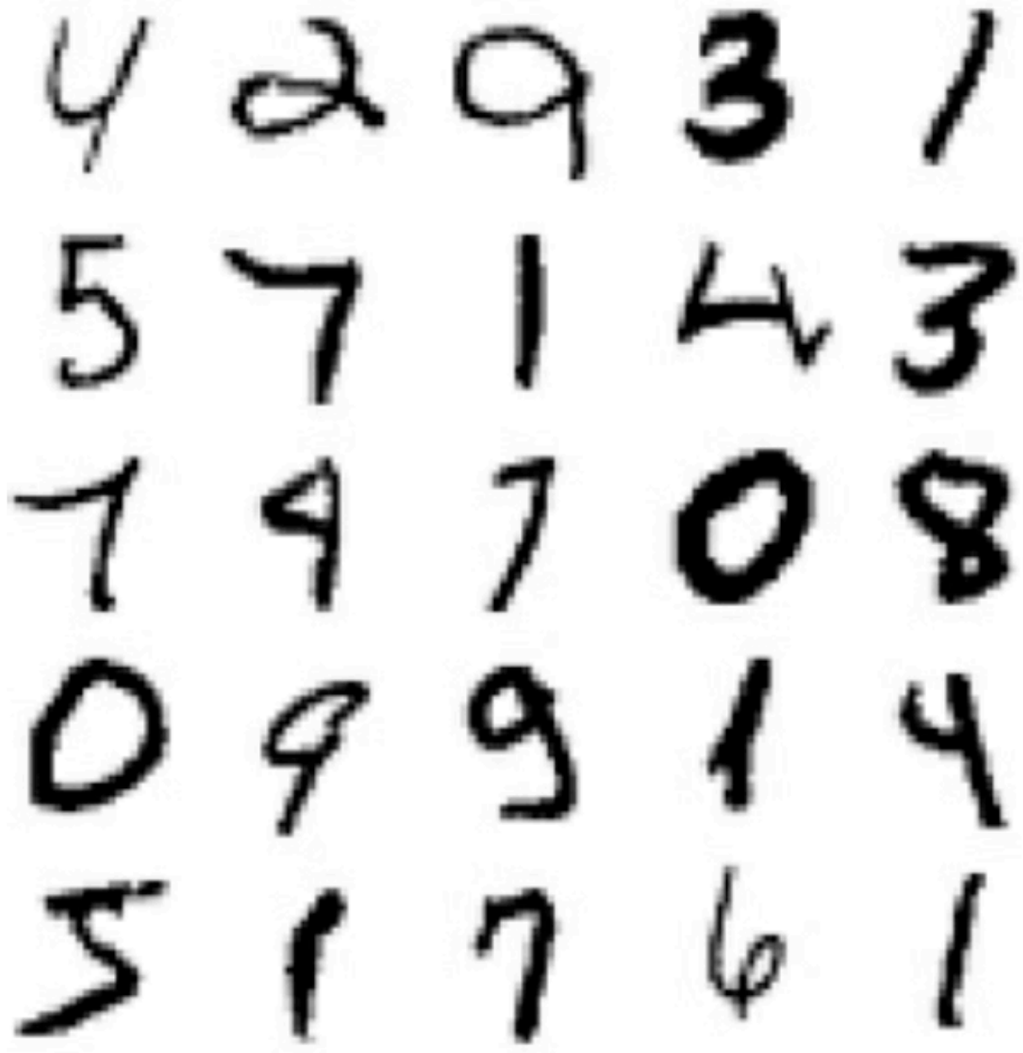
Hyperparameter Fitting

- Treat the number of dimensions as a hyperparameter like any other
- Perform a randomized search to find good values for all the hyperparameters together

PCA for Compression

- Applying PCA to the MNIST dataset (784 dimensions)
  - Captures 95% of the variance with 154 features
  - This **compresses** the dataset to less than 20% of its original size
- You can **decompress** the reduced dataset back to 784 dimensions
  - Some information has been lost
    - ***reconstruction error***

Original



Compressed

Randomized PCA

- Uses a stochastic algorithm to quickly find an approximation of the first ***d*** principal components
  - Much faster
  - Sci-kit automatically uses this method if the problem is too large

Incremental PCA

- Useful if the training set is too large to fit into memory
- Splits the training set into mini-batches
  - Feeds them in one at a time

Ch 8a

Random Projection

Random Projection

- Use a random linear projection to a lower-dimensional space
- Distances are very likely to be preserved
  - Similar instances remain similar
  - Different instances remain different
- No training required
- The data itself is not used at all to choose the projection axes

Random Projection

- For dataset containing ***m*** instances with ***n*** features
- And a tolerance for squared distance change of ***ε***
- Project the data down to ***d*** dimensions

$$d \geq 4 \log(m) / (\frac{1}{2} \varepsilon^2 - \frac{1}{3} \varepsilon^3)$$

- With 5,000 instances, 20,000 features, and 10% tolerance
  - ***d*** is 7,300 dimensions

Sparse Random Projection

- Uses a random sparse matrix
  - Most cells are zero
  - Uses much less memory

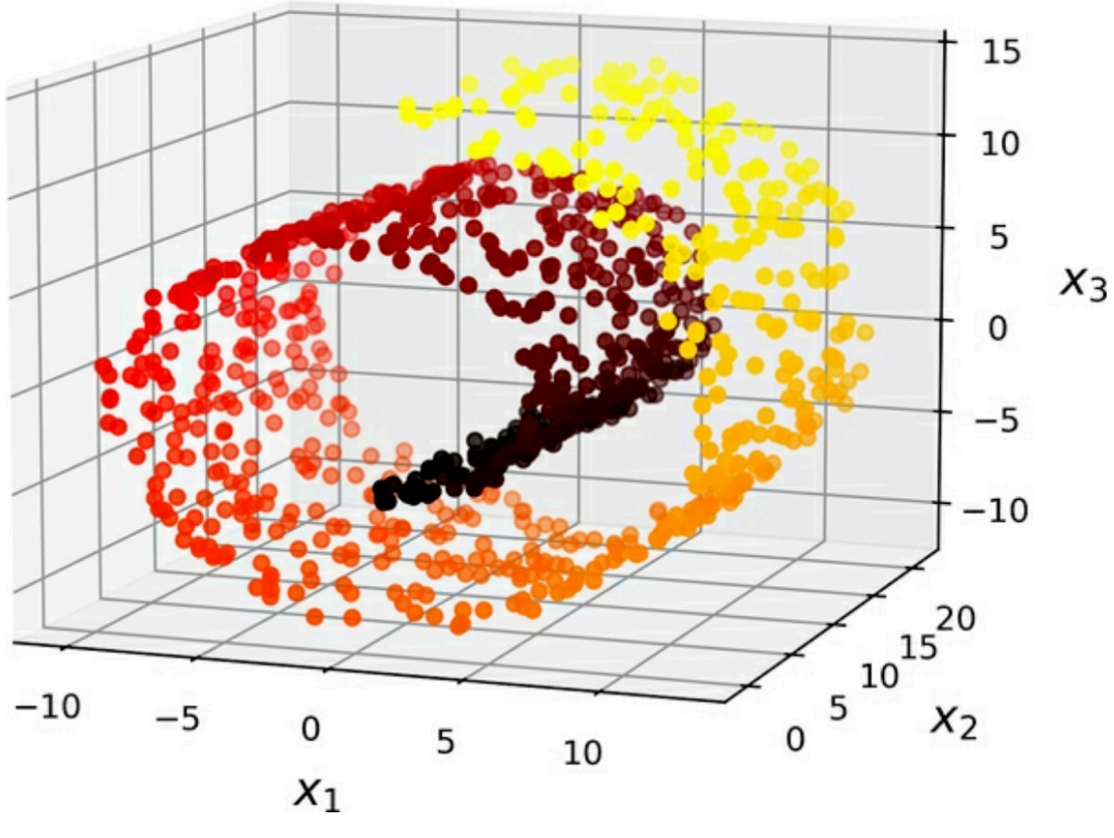
LLE

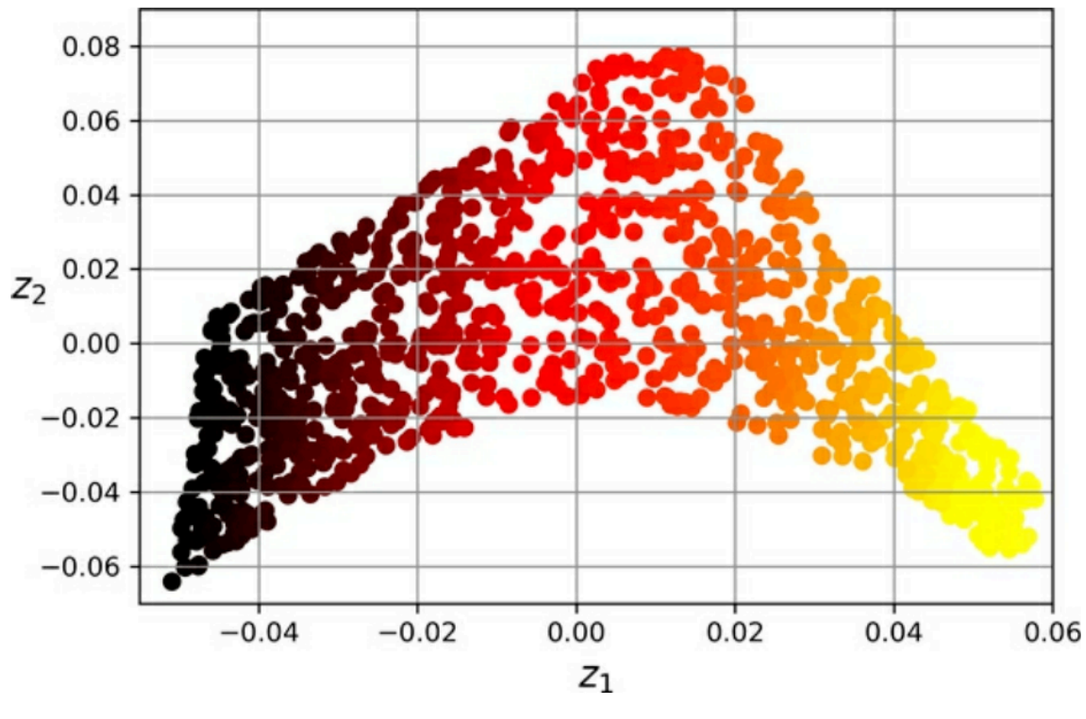
Locally Linear Embedding (LLE)

- A ***nonlinear dimensionality reduction*** (NLDR) technique
- Compates each training instance to its nearest neighbors
- Looks for a low-dimensional representation that preserves those local relationships
- Good for unrolling twisted manifolds
    - Especially if there's not too much noise

Locally Linear Embedding (LLE)

- The 3-D rolled data on the left becomes the 2-D data on the right
- Long-range distances are not preserved
  - So the rectangle gets stretched and twisted





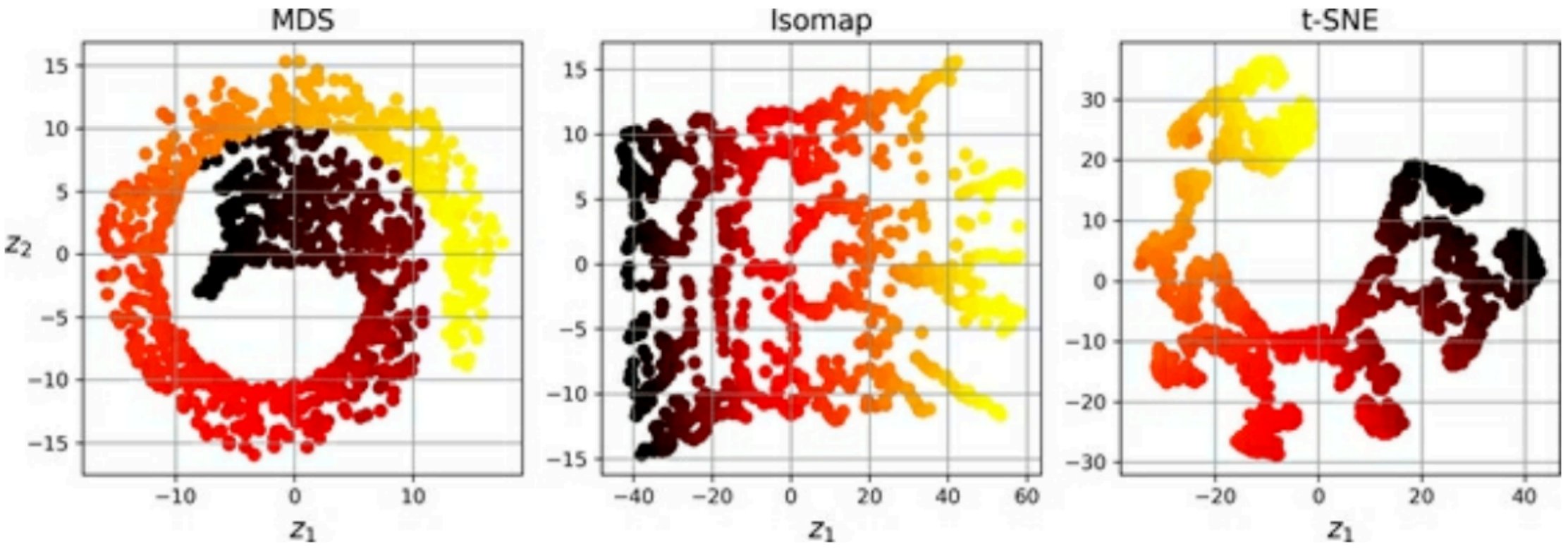
Other Dimensionality Reduction Techniques

MDS and Isomap

- ***Multidimensional scaling*** (MDS)
  - Reduces dimensionality while preserving distance between the instances
  - Random projection does that, but only works well for high-dimensional data
- ***Isomap***
  - Creates a graph connecting each instance to its nearest neighbors
  - Reduces dimensionality while trying to preserve the ***geodesic distances*** between the instances
    - The number of nodes on the shortest path between nodes

t-SNE and LDA

- ***t-distributed stochastic neighbor enbedding*** (t-SNE)
  - Reduces dimensionality while trying to keep similar instances close and dissimilar instances apart
  - Mostly used for visualization
- ***Linear discriminant analysis*** (LDA)
  - Linear classification algorithm
  - Learns the most discriminative axes between the classes
  - Those axes define a hyperplane upon which to project the data
  - This projection will keep the classes as far apart as possible


MDS
Isomap
t-SNE
z_2
z_1
z_1
z_1

Ch 8b