Dimensionality Reduction: Principal Component Analysis

CSCI-P556 Applied Machine Learning
Lecture 24

D.S. Williamson

Agenda and Learning Outcomes

Today’s Topics

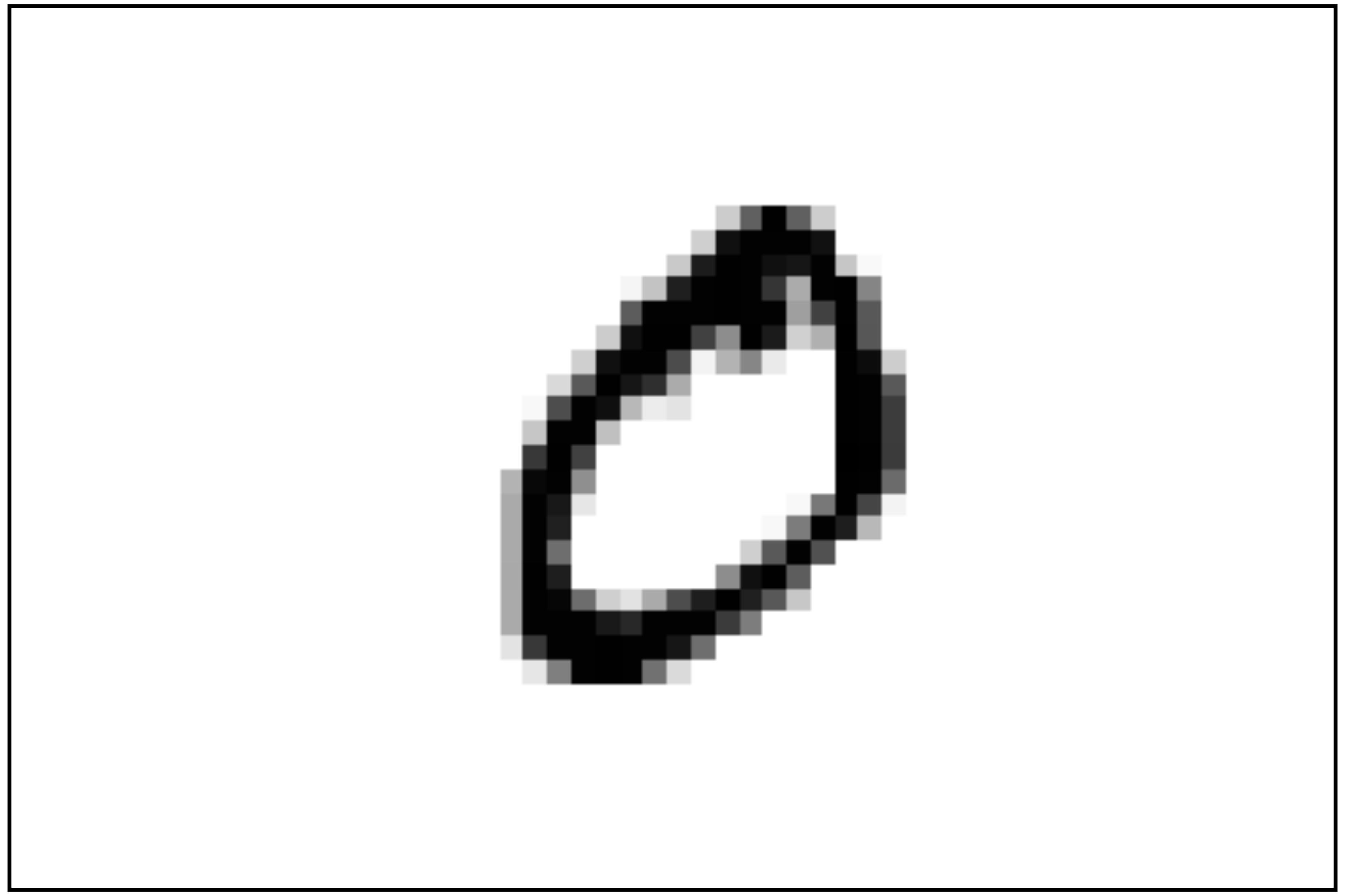
- **Topics**:
  - Dimensionality Reduction using Principal Components Analysis (PCA)
- **Announcements:**
  - HW#4, Due ?
  - No HW #5
  - Quiz #3 Next ?
  - Project presentation schedule posted shortly

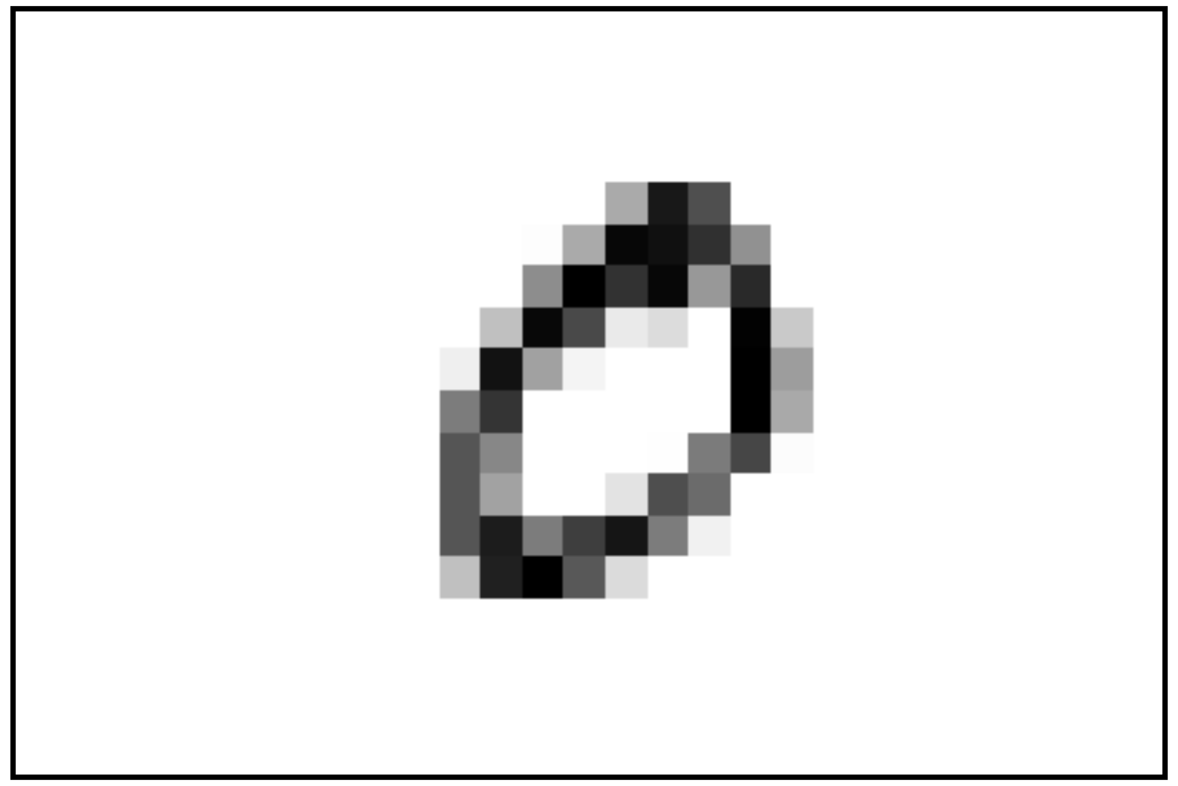
	Thursday, April 8	Gaussian Mixture Models	
13	Tuesday, April 13	Dimensionality Reduction	
	Thursday, April 15	Quiz#3 and/or Reinforcement Learning (?)	
14	Tuesday, April 20	Quiz#3 or Review or No Class (?)	
	Thursday, April 22	**No Class**	**Wellness Day**
15	Tuesday, April 27	Project Presentations I	
	Thursday, April 29	Project Presentations II	

MNIST Example

Handwritten Digits

- Each image has 28 x 28 = 784 dimensions (features/attributes)
- All the dimensions are not important (e.g. border pixels)
- Downsampling to 14 x 14 = 196 dimensions preserves information
    - Compute average of adjacent pixels along rows and columns
    - Can still tell that it's a zero

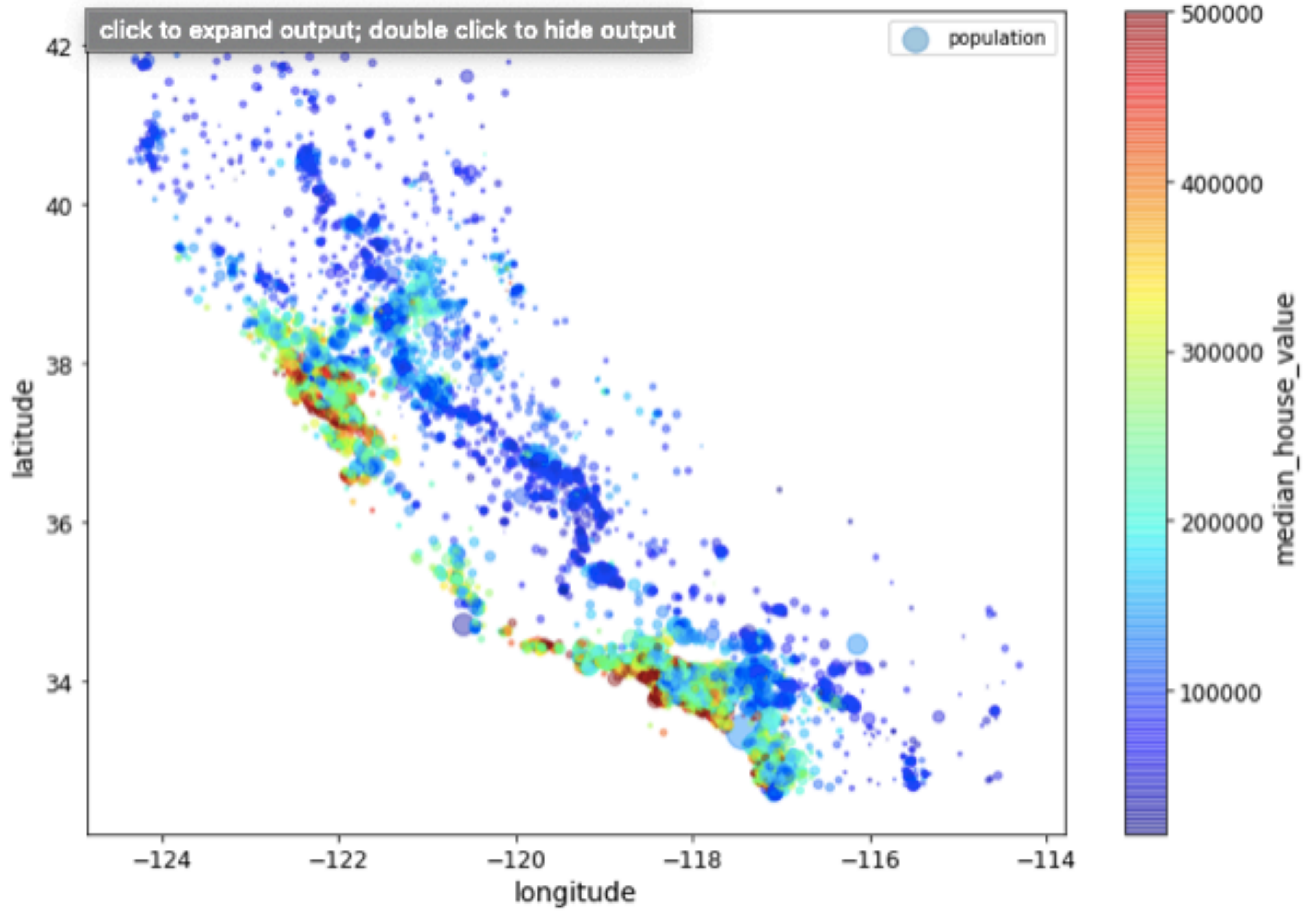




Why Dimension Reduction?

- **High Dimensionality**: Large number of features/ attributes
  - Documents represented by thousands of words
  - Images represented by hundreds (or thousands) of pixels
- **Redundant and irrelevant features** (not all words are relevant for classifying/clustering documents)
- **Difficult to interpret and visualize**. How do you visually compare samples that have more than 3 dimensions? Think housing data that had 10 dimensions.
- **Curse of dimensionality**

```
In [6]: housing.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object
dtypes: float64(9), object(1)
memory usage:
```



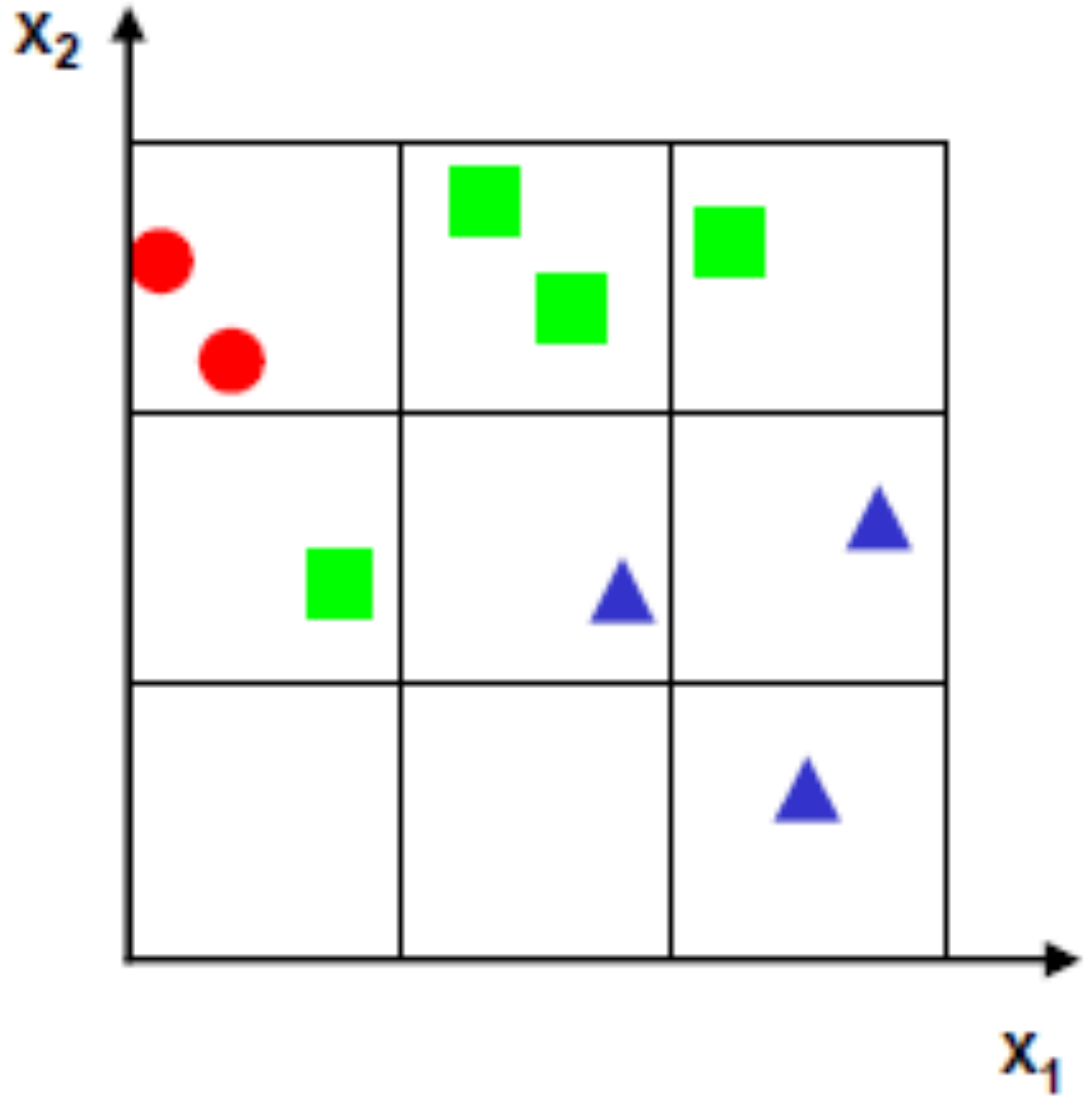
Curse of Dimensionality

Data comparison and amount of data becomes more challenging

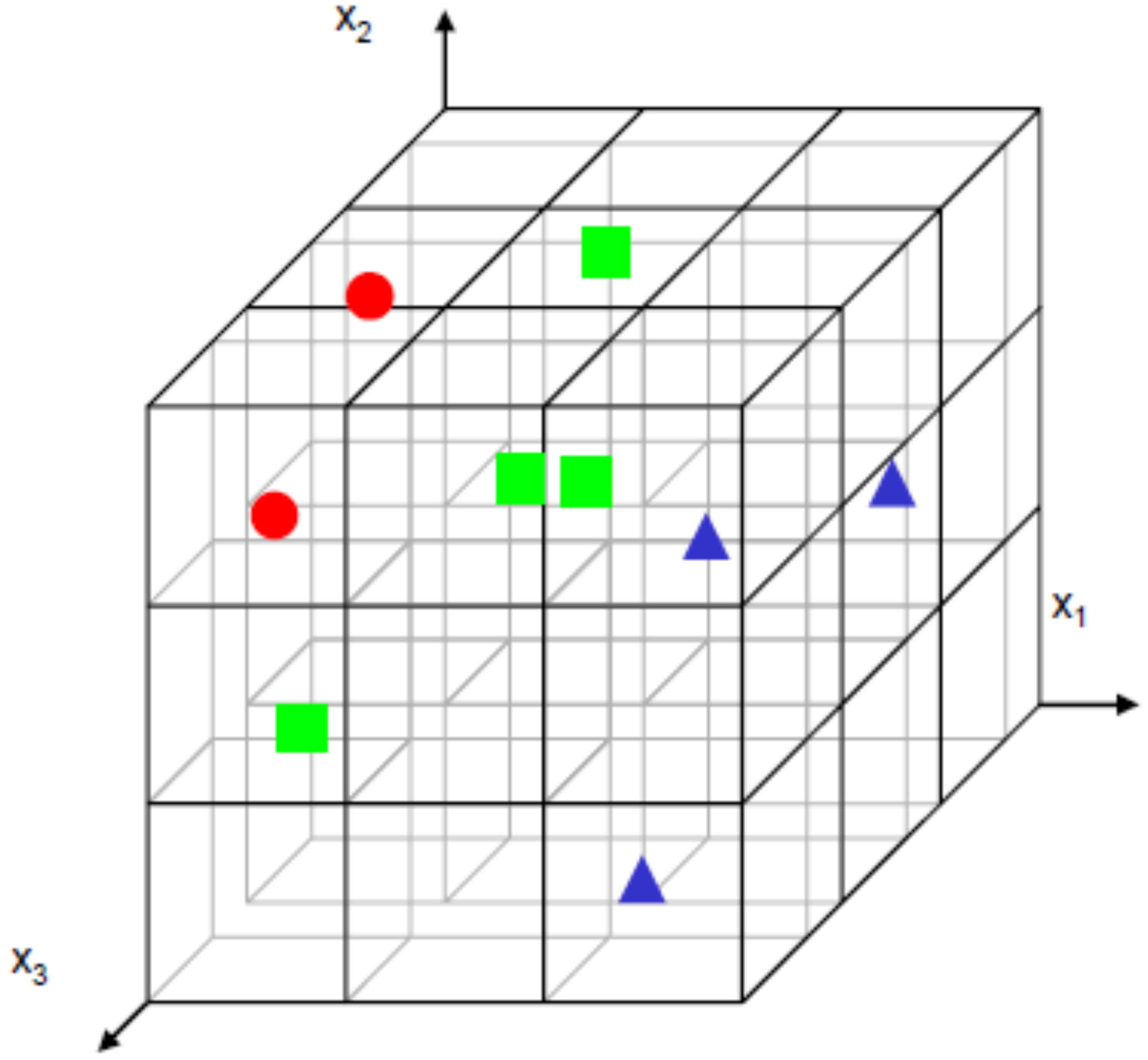
- In theory, the ***curse of dimensionality*** describes the phenomenon where the ***features space becomes increasingly sparse*** as the number of dimensions increases.

Data in 1D

Data in 2D

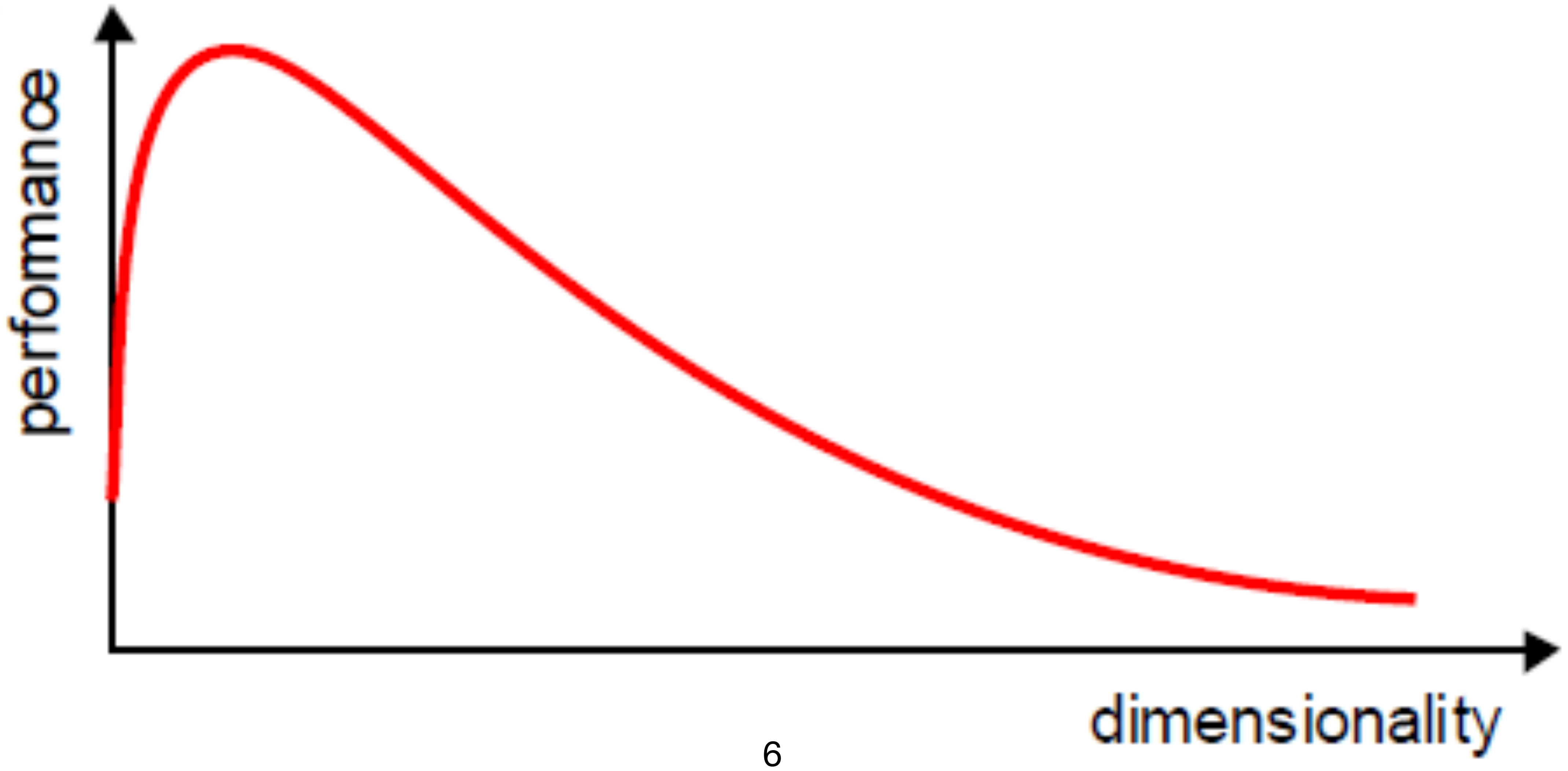


Data in 3D



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 additional information that is lost by discarding some features is (more than) compensated by a more accurate mapping in the lower dimensional space



Curse of Dimensionality

- **How do we beat the curse of dimensionality?**
  - By incorporating prior knowledge (e.g., expert opinion)
  - By providing increasing smoothness of the target function
  - By reducing the dimensionality

Unsupervised Dimensionality Reduction

- Consider a collection of data points in a high-dimensional feature space (e.g., 500-d)
  - Try to find a more compact data representation
  - Create new features defined as functions over all of the original features
- **Why?**
  - **Visualization**: need to display low-dimensional version of a data set for visual inspection
  - **Preprocessing**: learning algorithms (supervised and unsupervised) often work better with smaller numbers of features, both in terms of runtime and accuracy (why?)

Principal Component Analysis (PCA)

How to preserve information?

- **Suppose we have two features, and we can only keep one:**
  - For one of the features, most examples have similar value
  - For the other, most examples differ from each other
- **Which one should we keep?**
    - The second, because it retains information about the data items

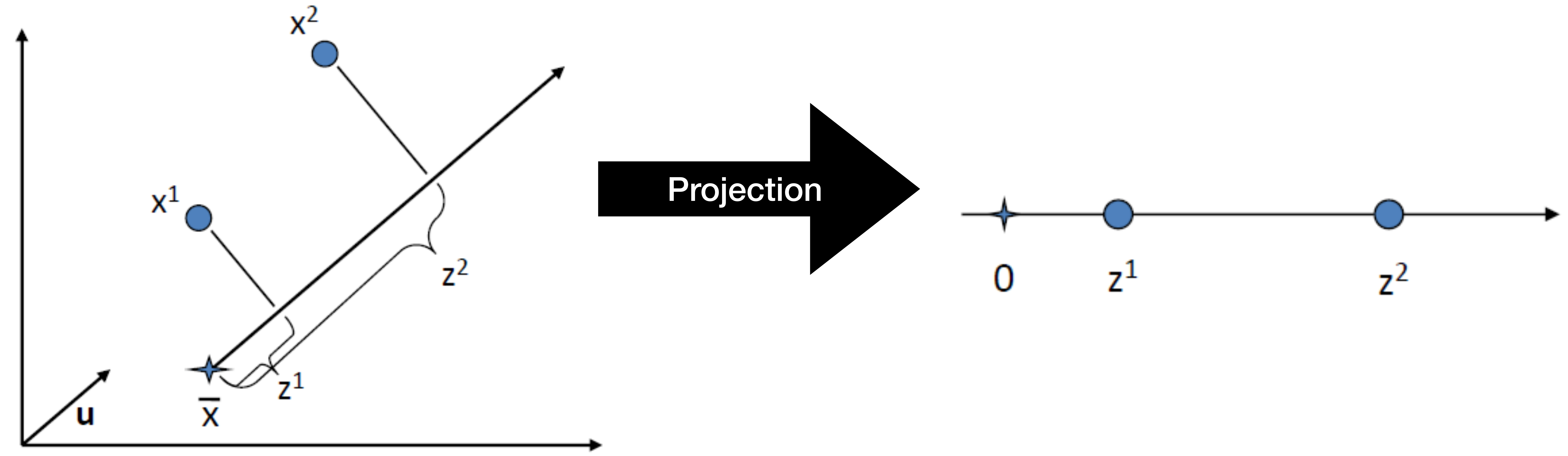
Principal Component Analysis (PCA)

A Classic Dimensionality Reduction Technique

- It linearly projects n-dimensional data onto a k-dimensional space while preserving information (k < n)
    - e.g., project space of 10k words onto a 3d space
- **Basic idea for PCA**:
    - Find a linear projection that retains the most **variance** in data
        - i.e. the projection that retains most amount of information
    - Find a line such that when the data is projected onto that line, it has the maximum variance

First, what is a linear projection

1. A linear projection defines a new axis which is the result of rotating existing ones
2. It is often used with the operation of translation — moving the origin of the coordination system.

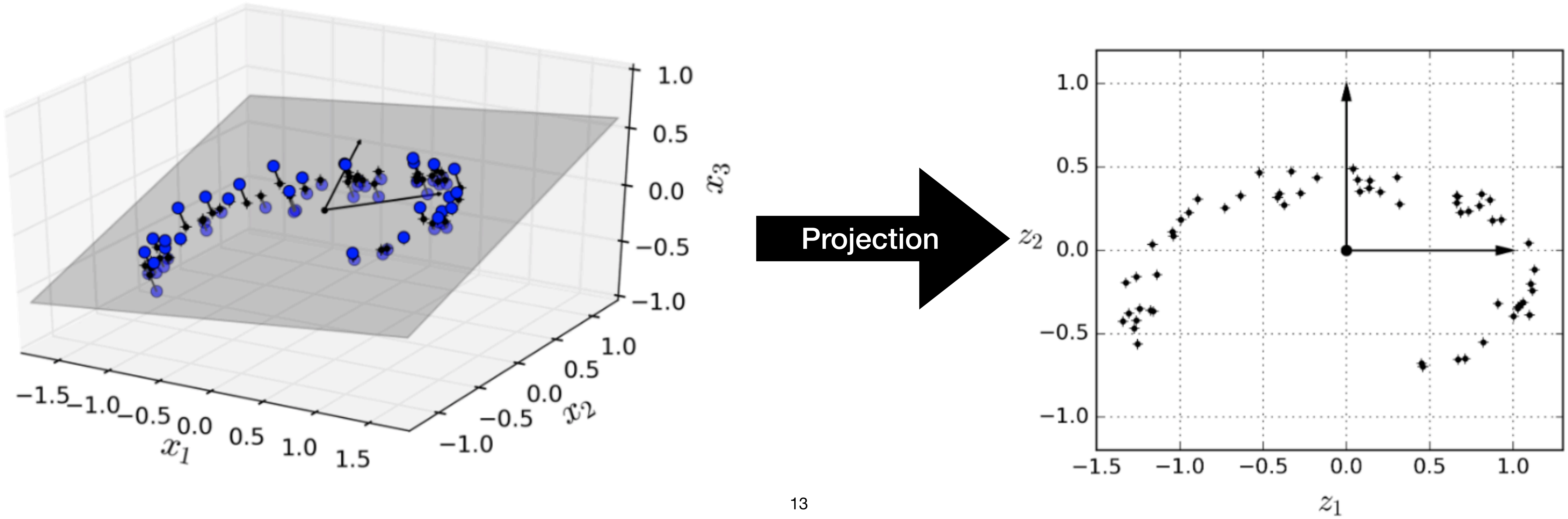


Projection

- **Why Project?**
  - Training samples are not uniform across all dimensions (e.g. they bunch)
  - Many features are nearly constant or highly correlated with others
  - Hence, training examples are bunched in a lower-dimensional subspace, within the higher-dimensional space

Projection from 3D to 2D: A Visual Example

- Three-Dimensional data actually (mostly) lies within a 2-D subspace
- We can project every example down to the 2-D subspace
- Hence, dimensionality reduction



PCA Algorithm

The Most Popular Approach

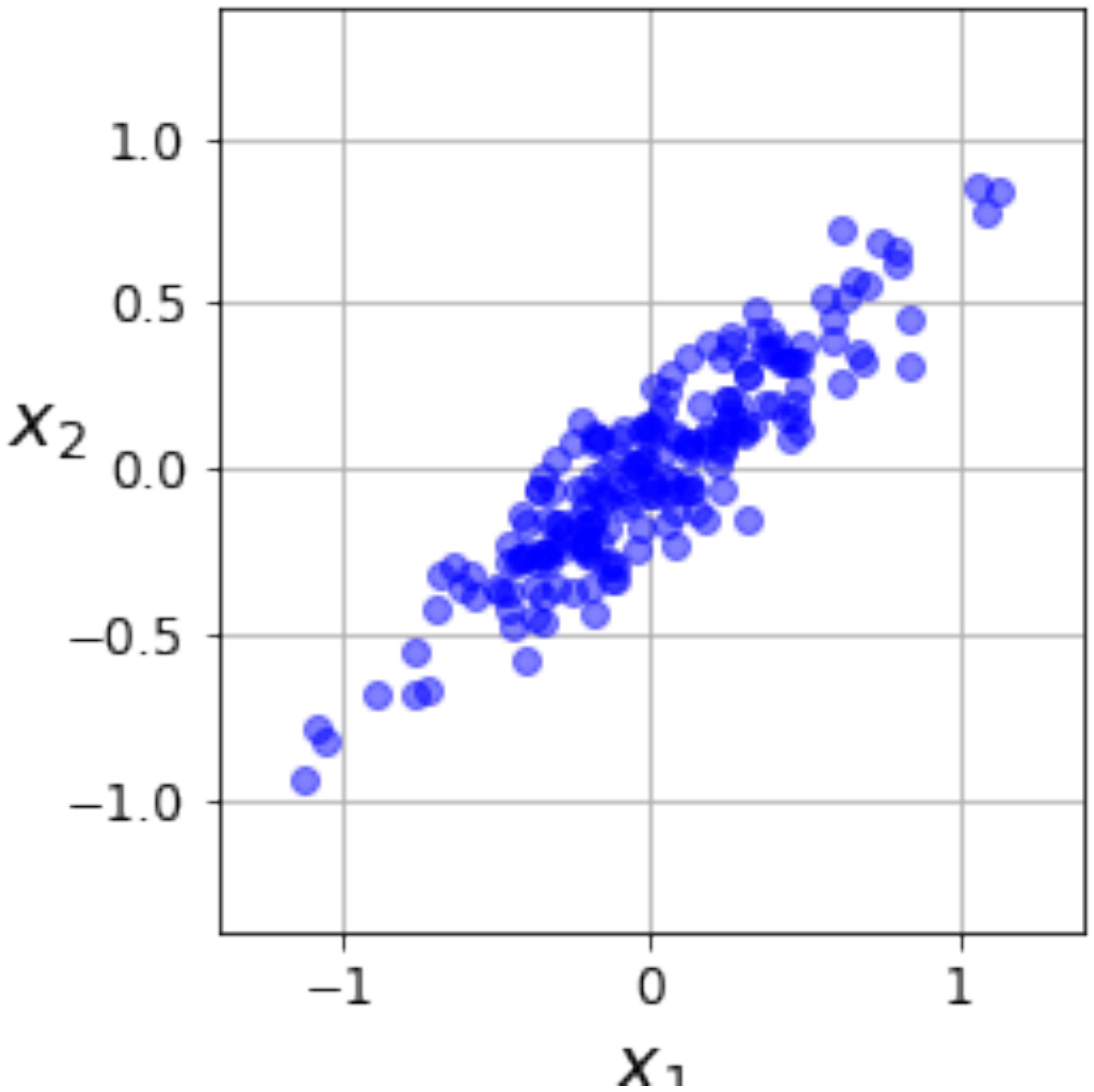
- Create data matrix, X, with one row vector x_n per data point
- Compute and subtract mean $\mu_{\mathbf{X}}$ from each row vector x_n in X
- Find eigenvectors and eigenvalues of X
- Principal Components (PCs) are the M eigenvectors with largest eigenvalues

Eigenvectors and Eigenvalues

- An ***eigenvector*** is a vector that is ***scaled by a linear transformation***, but not moved
  - Think of it as an arrow whose magnitude is changed, but it direction is not changed
  - It may stretch (or shrink) as it is transformed, but it points in the same direction
- The ***scaling factor*** of an eigenvector is called its ***eigenvalue***.
- **Key idea**: Transform in such a way that the first PC has as high variance as possible and successively find PCs with highest variance such that the orthogonal constraint is satisfied

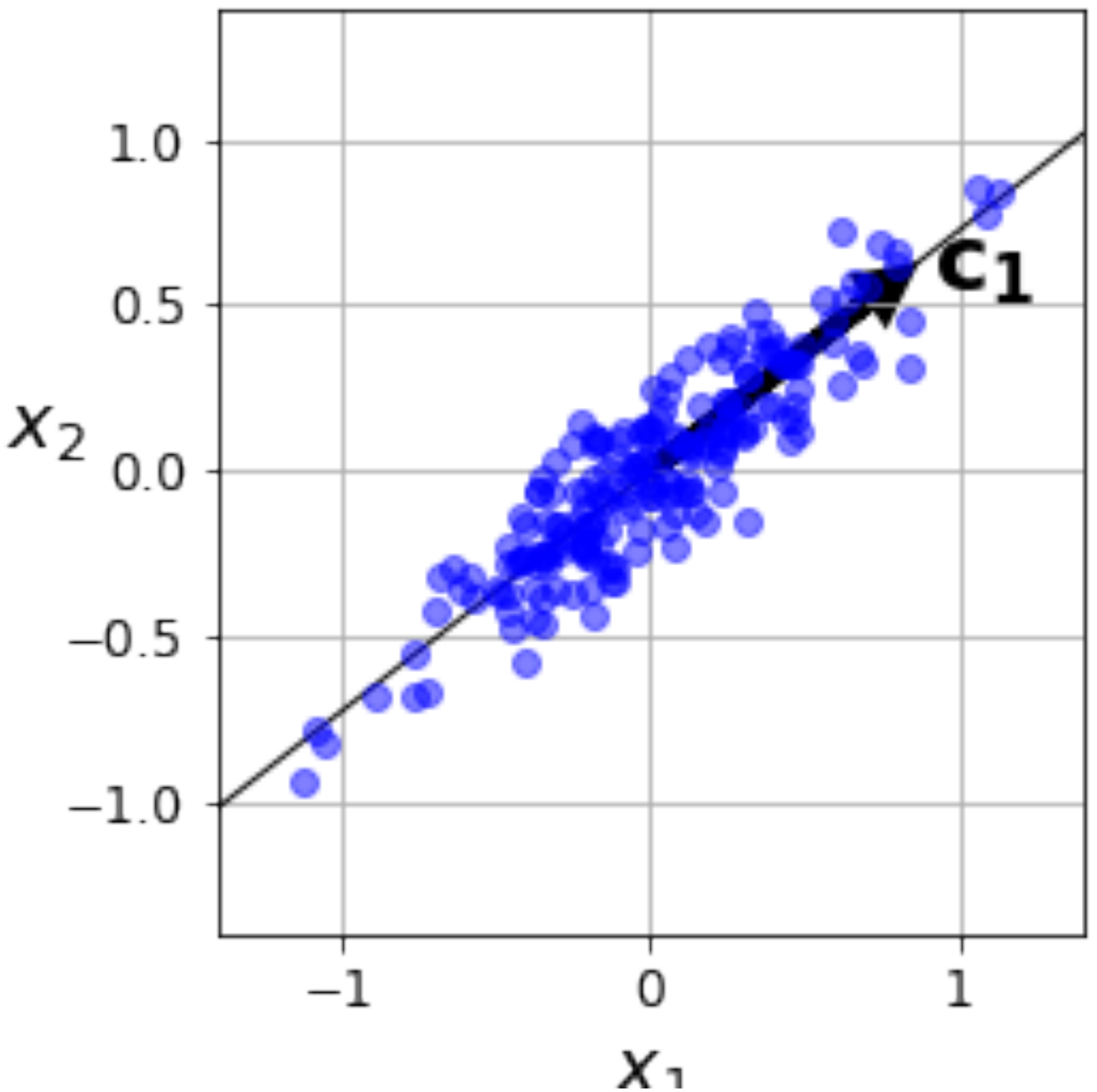
2d Data: Finding Principal Components

- Find vectors along data with highest variance



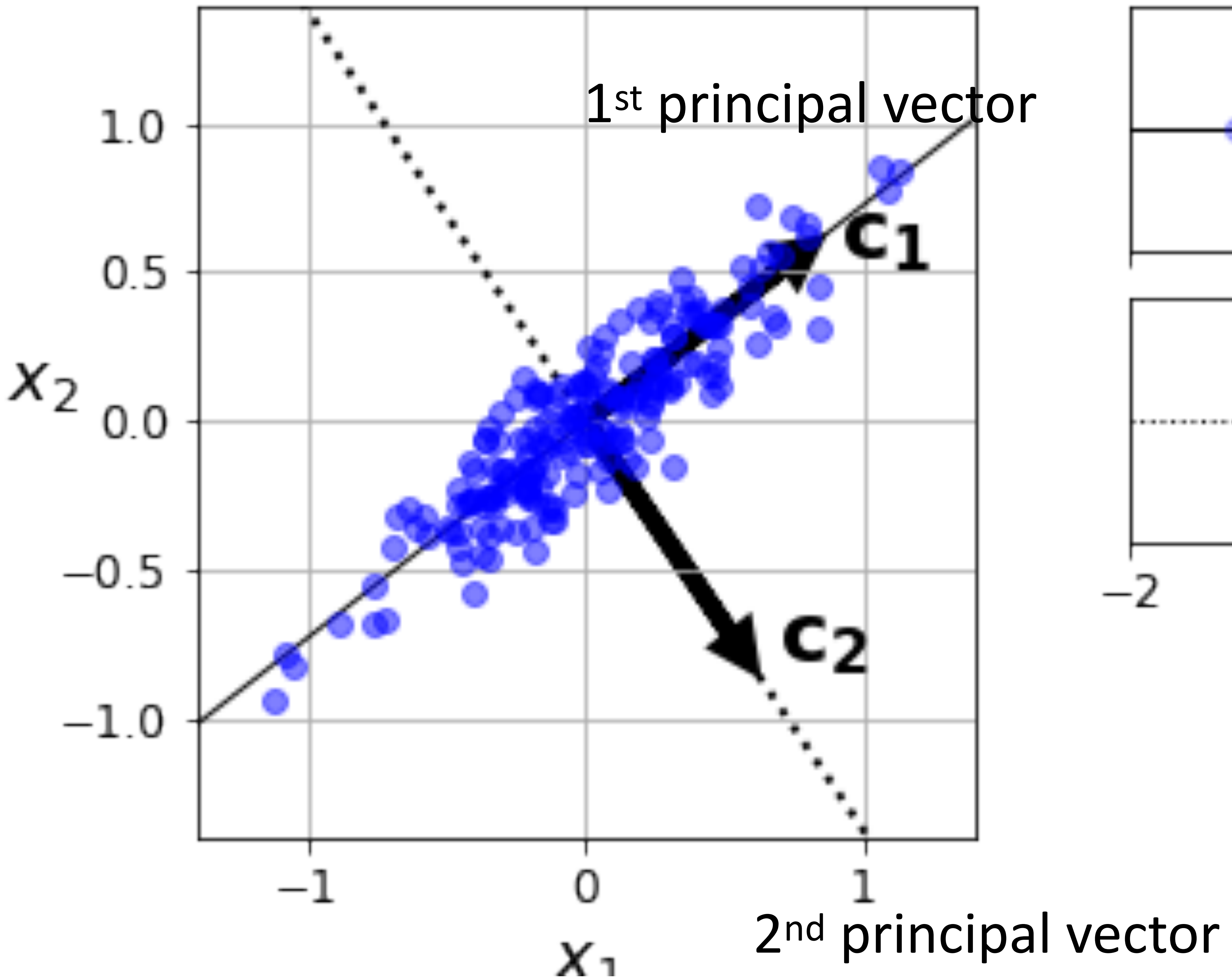
2d Data: Finding Principal Components

- Find vectors along data with highest variance
  - Vector, C1, is the axis with the largest amount of variance

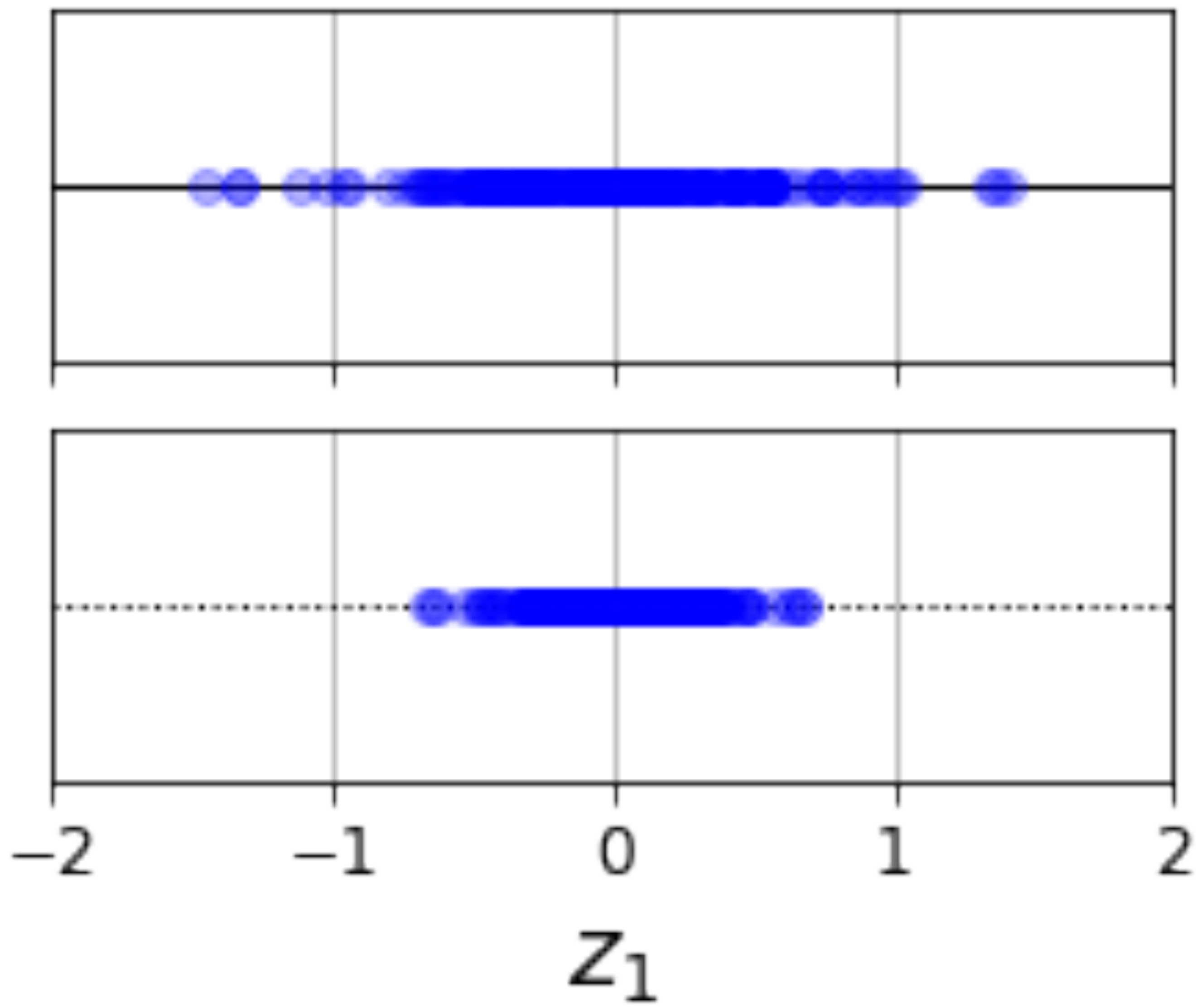


2d Data: Finding Principal Components

- Find vectors along data with highest variance
  - Vector, C1, is the axis with the largest amount of variance.
  - Vector, C2, is another option, but its variance is smaller.
- Principal vectors are orthogonal



Variance



Computing Eigenvectors and Eigenvalues

- **Singular Value Decomposition** is one way to find Eigenvectors and Eigenvalues. It **is a matrix factorization approach**
    - E.g. Factor a given matrix as the product of multiple matrices
- For instance, a given matrix ***S***, can be factored into $\mathbf{U\Sigma V}^T$ (e.g. $\mathbf{S} \approx \mathbf{U\Sigma V}^T$)
    - The columns of $\mathbf{V}$ contain the Eigenvectors (e.g. principal components) that we are interested in
    - The scalars along the diagonal of $\mathbf{\Sigma}$ contain the Eigenvalues
    - $\mathbf{U}$ also has Eigenvectors, but we aren’t interested in these.

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

SVD in Python

- The data must be centered around 0, before performing SVD

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

PCA for Dimensionality Reduction

- After principal components (e.g. eigenvectors) have been identified, **dimensionality reduction is accomplished as follows**:
  - Assume the desired new dimension is d, where d < N (the original dimension)
  - Project original dataset onto the hyperplane using the first d principal components
    - Compute the matrix multiplication of the dataset and a Eigenvector matrix that only contains the first d columns, $\mathbf{W}_d$
    - $\mathbf{X}_{d-proj} = \mathbf{X}\mathbf{W}_d$

$$\mathbf{W}_d = \begin{bmatrix} | & | & \cdots & | \\ \mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_d \\ | & | & \cdots & | \end{bmatrix}$$

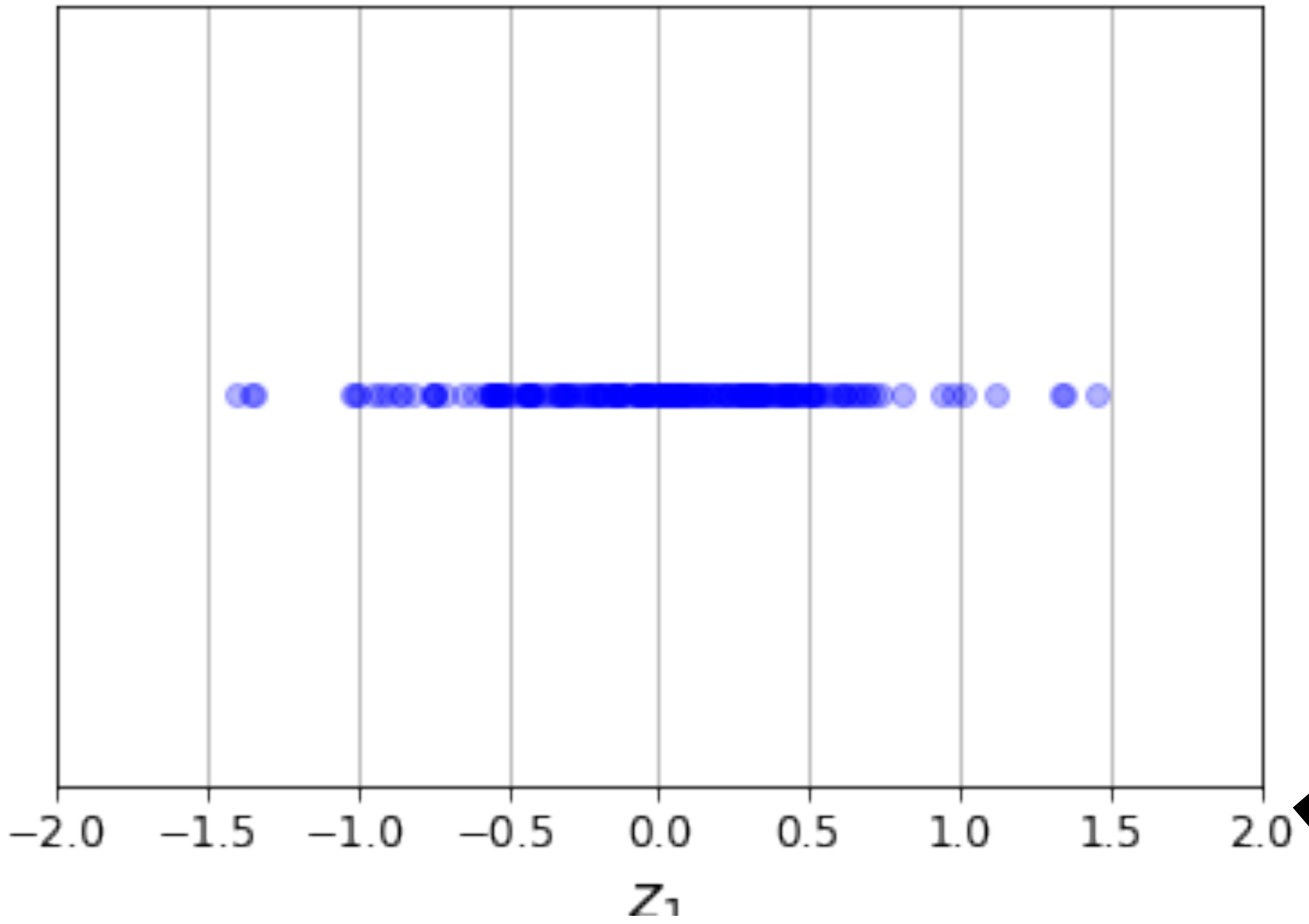
```python
from sklearn.decomposition import PCA

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

PCA in Scikit-Learn. Do NOT need to center data first.

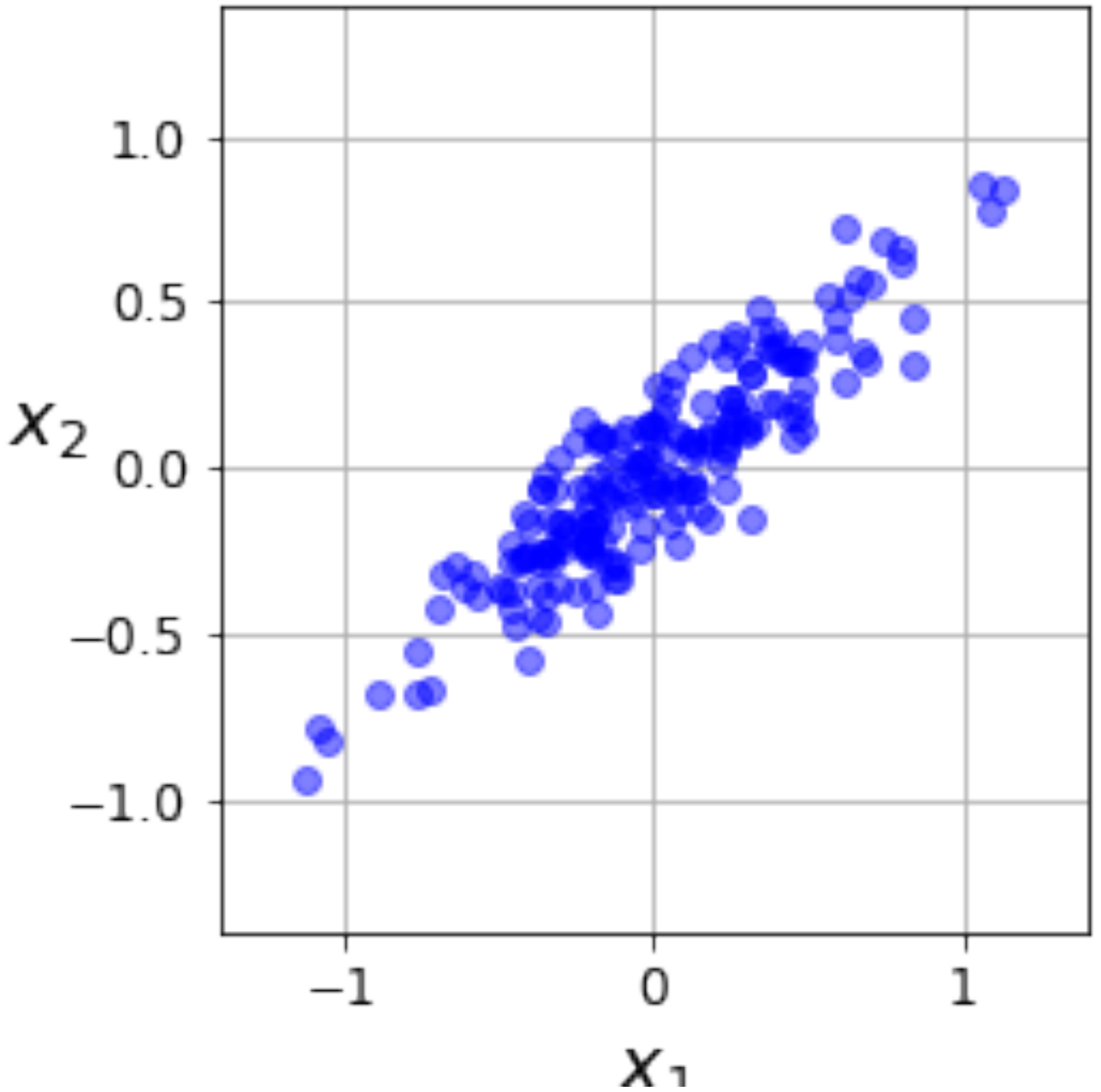
Example

$$\mathbf{X}_{reduced} = (\mathbf{X} - \mu_{\mathbf{X}})\mathbf{W}_d$$

1D Data after PCA, $\mathbf{X}_{reduced}$

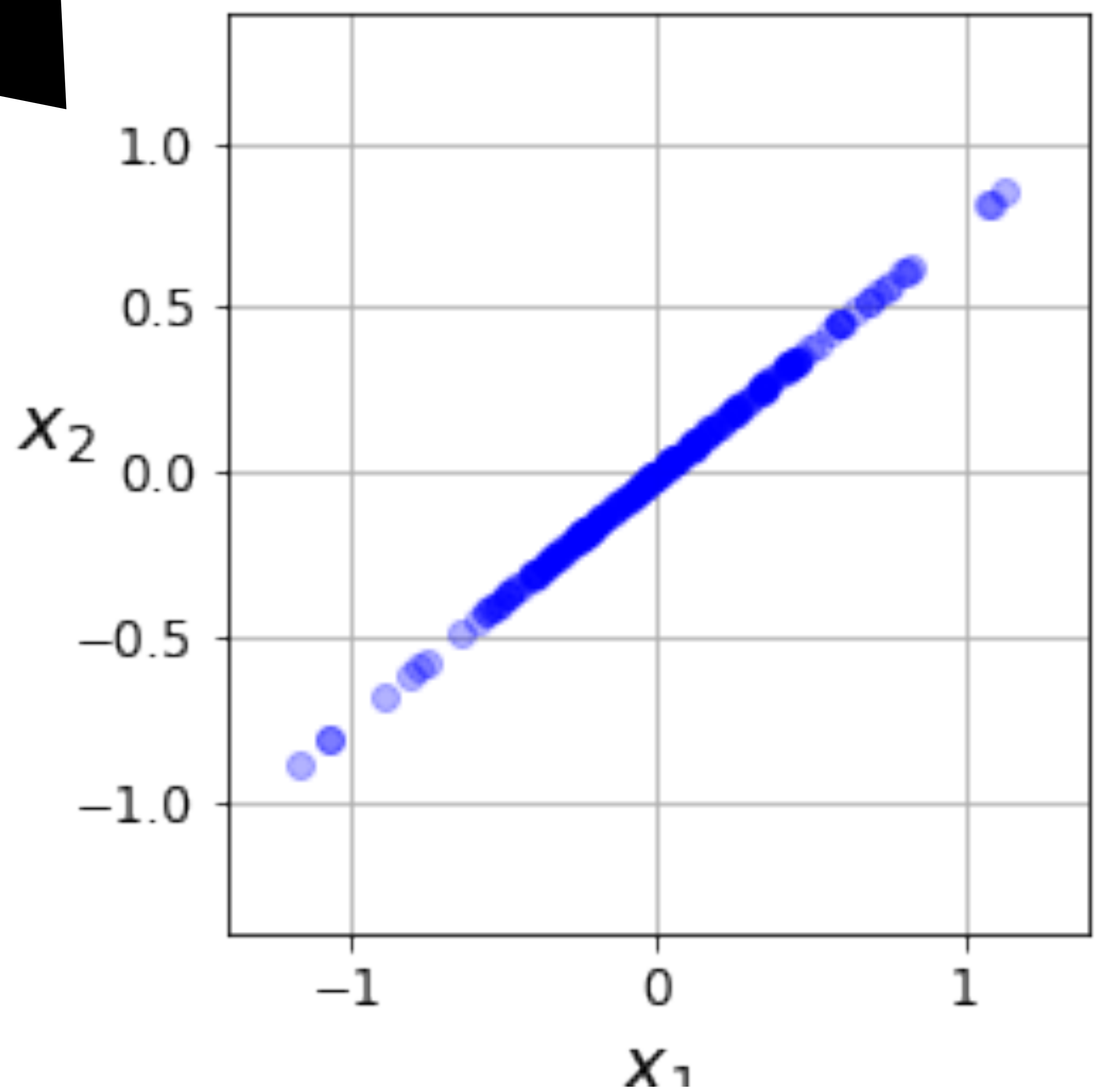


Original Data, $\mathbf{X}$



$$\mathbf{X}_{recovered} = \mathbf{X}_{reduced}\mathbf{W}_d^T$$

Data Recovered Back to 2D, $\mathbf{X}_{recovered}$



Information is Lost!

PCA: Things to Consider

- **PC is sensitive to noise**. Direction of new PCs may change if training set is perturbed by noise.
- **Number of dimensions, d**
    - Choose value for d, so that a large portion of variance is maintained (see PCA in scikit-learn; set n_components to value between 0 and 1 — desired explained variance ratio).
    - Or, reduce down to 2 or 3 dimensions for visualization
- **Stochastic and Incremental versions of PCA exist.** Hence, do not require the entire dataset at once. Computationally more efficient
- **Kernel PCA for nonlinear projections.**

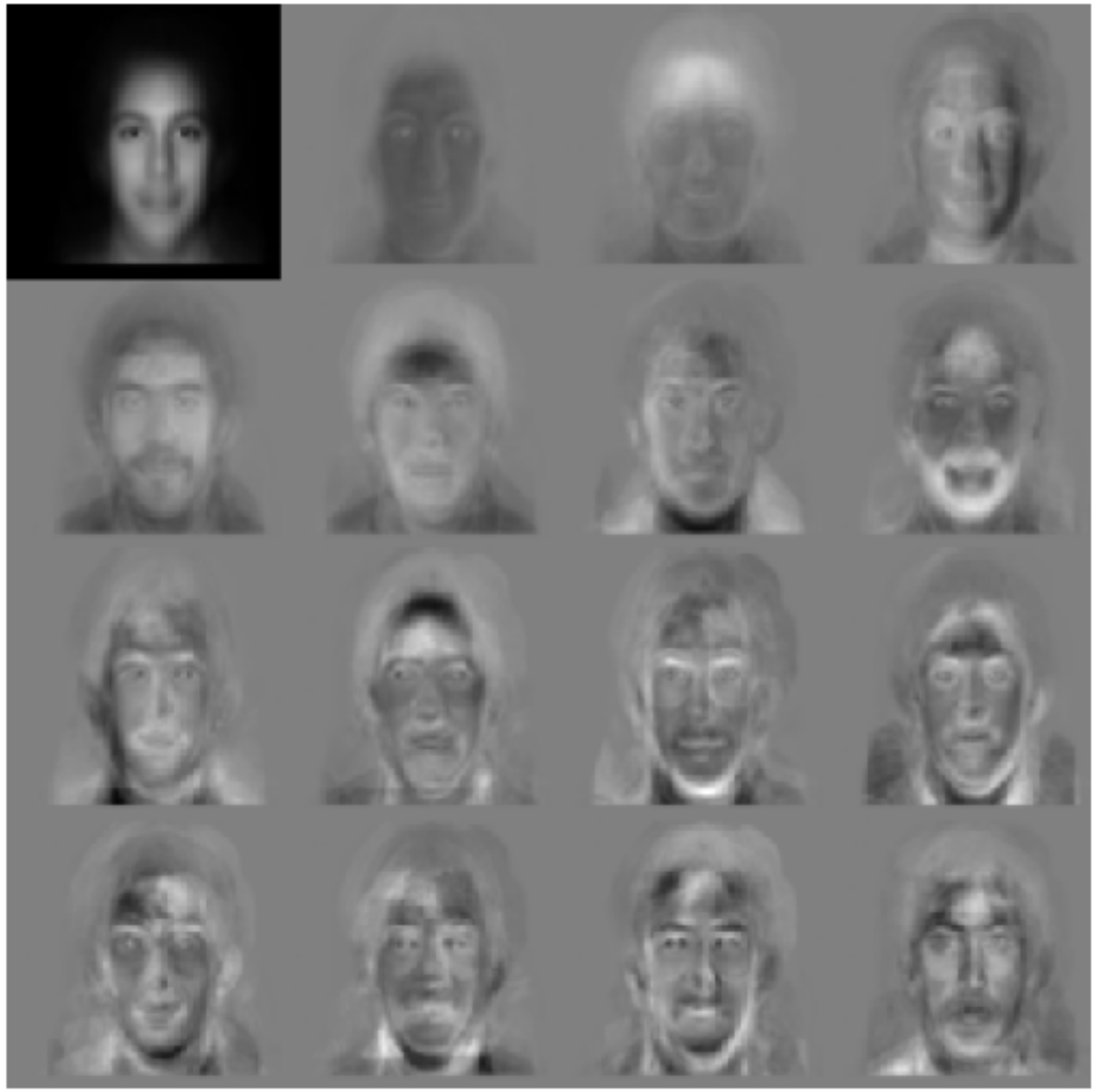
Example: Face Recognition

PCA for Supervised Learning

- A typical image of size 256 x 128 is described by n = 256x128 = 32768 dimensions — each dimension described by a grayscale value
- Each face image lies somewhere in this high-dimensional space
- **Images of faces are generally similar in overall configuration**, thus
  - They should be randomly distributed in this space
  - We should be able to describe them in a much lower dimensional space

PCA for Face Images: Eigen-faces

- Database of 128 carefully-aligned faces.
- Here are the mean and the first 15 eigenvectors.
- Each eigenvector (32768-d vector) can be shown as an image — each element is a pixel on the image
- These images are face-like, thus called **Eigen-faces**



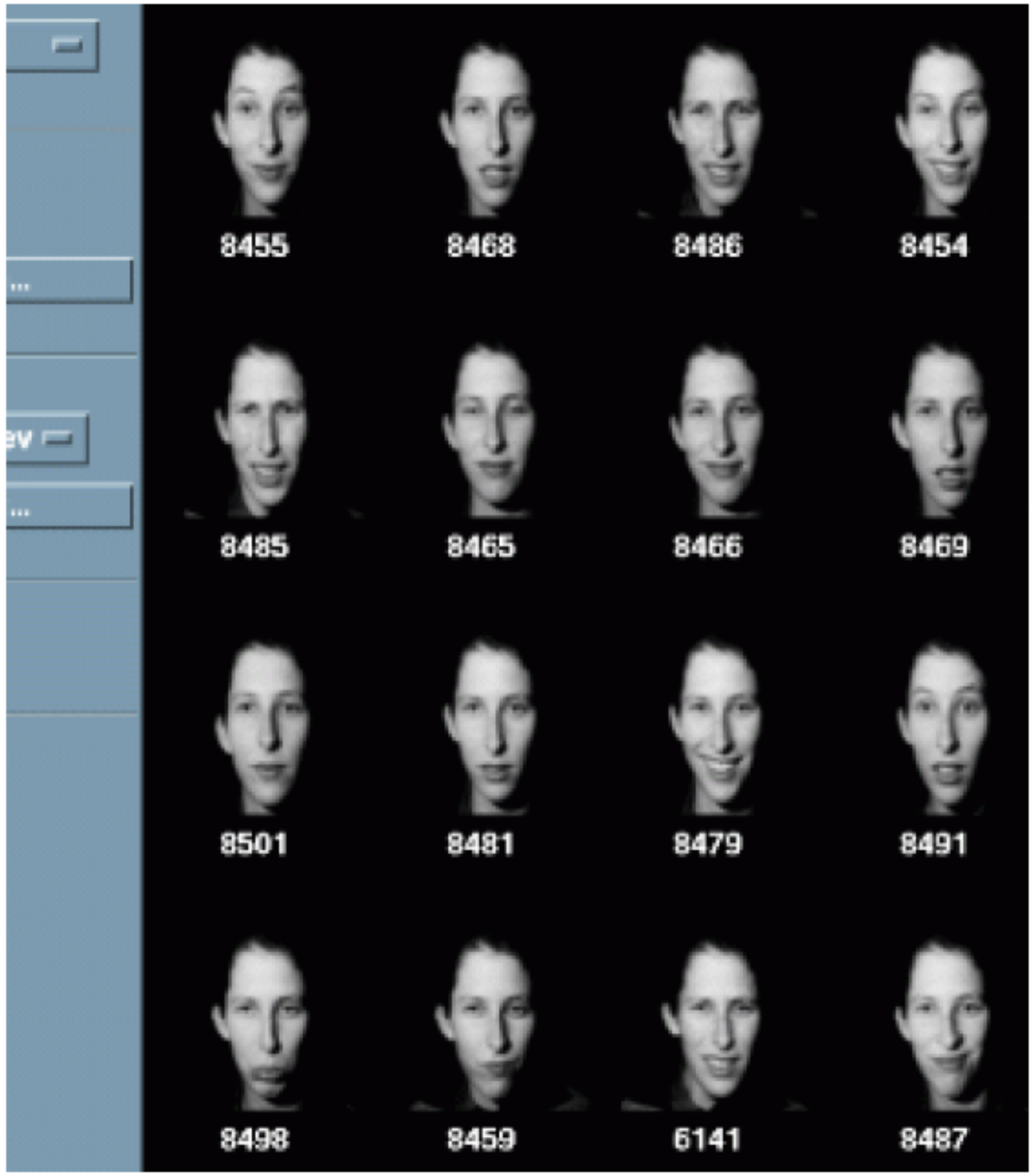
Face Recognition in Eigenface Space

Turk and Pentland 1991

- Nearest Neighbor classifier in the eigenface space
- Training set always contains 16 face images of 16 people, all taken under the same set of conditions of lighting, head orientation and image size
- **Accuracy**:
    - Variation in lighting: 96%
    - Variation in orientation: 85%
    - Variation in image size: 64%

Face Image Retrieval

- Left-top image is the query image
- Return 15 nearest neighbors in the eigenface space
- **Able to find the same person despite:**
  - Different expressions
  - Variations such as glasses



Next Class: