# Applied Machine Learning

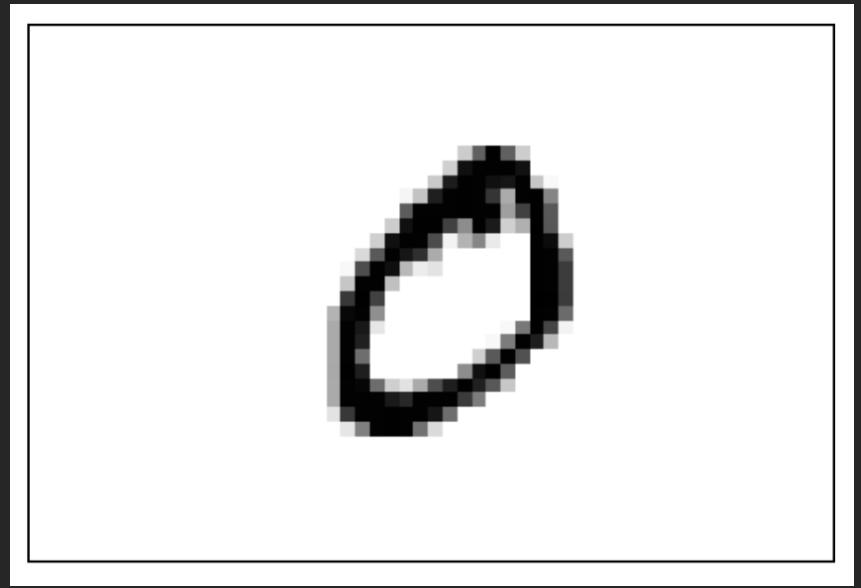
## Principle Component Analysis

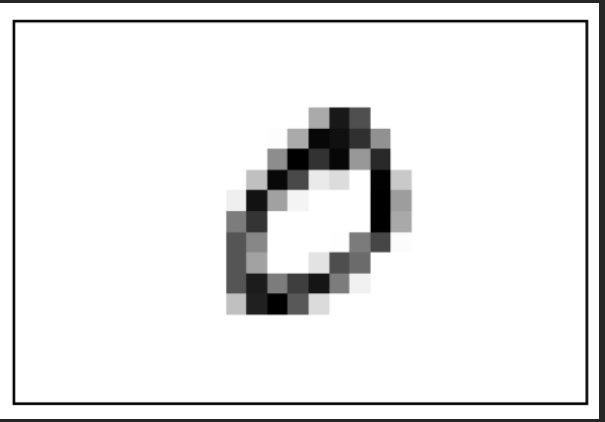
Computer Science, Fall 2022

Instructor: Xuhong Zhang

# Principal Component Analysis

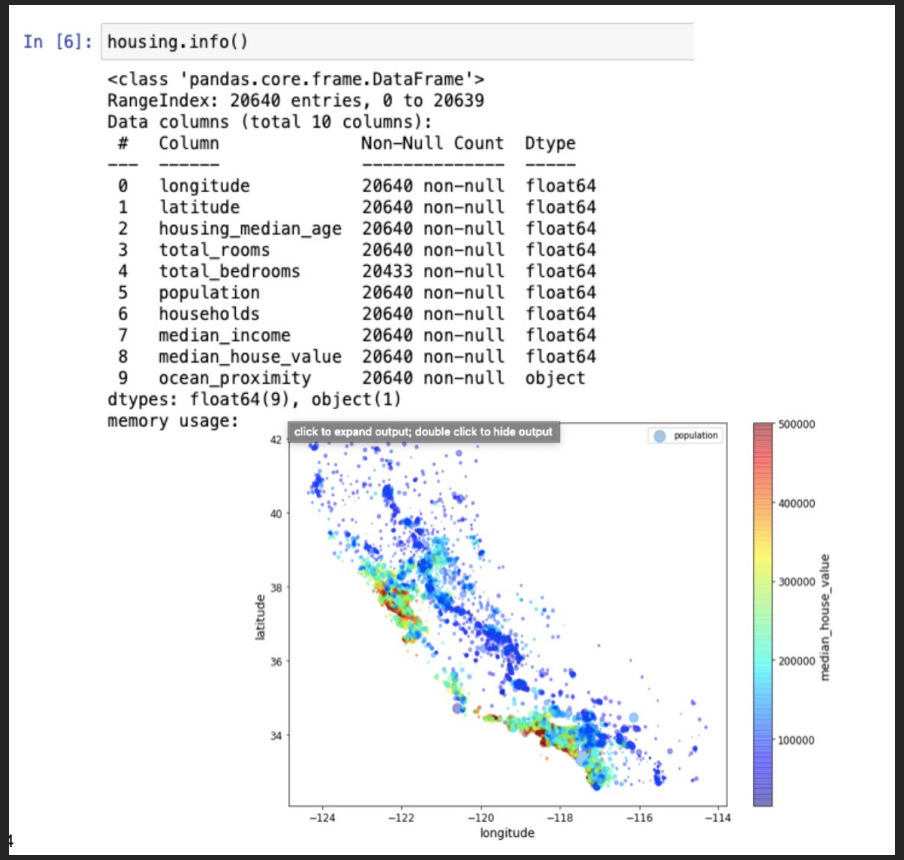
- MNIST Example : Handwritten Digits
  - Each image has $28 \times 28 = 784$ dimensions features/attributes)
  - All all the dimensions important?
    e.g. border pixels
  - Downsampling to $14 \times 14 = 196$ dimensions
    - Compute average of adjacent pixels along rows and columns
    - Can still tell that it's a zero





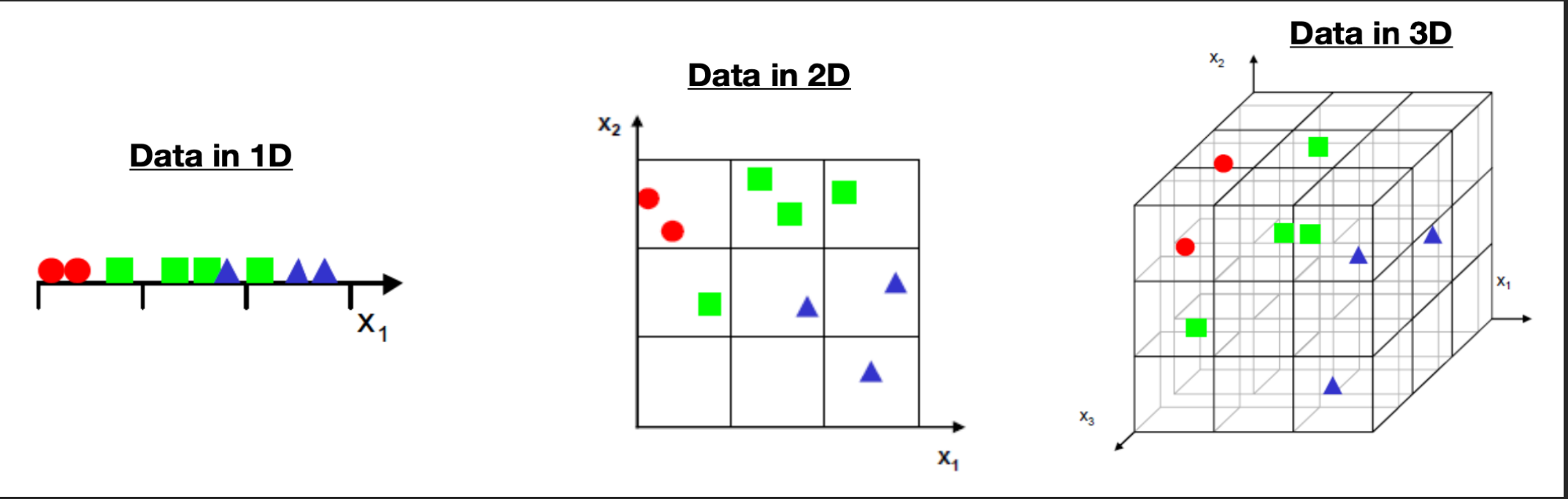
# Why Dimension Reduction?

- 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 classification
- Difficult to visualize and interpret
  - More than 3 dimensions
- Curse of dimensionality



# Curse of Dimensionality

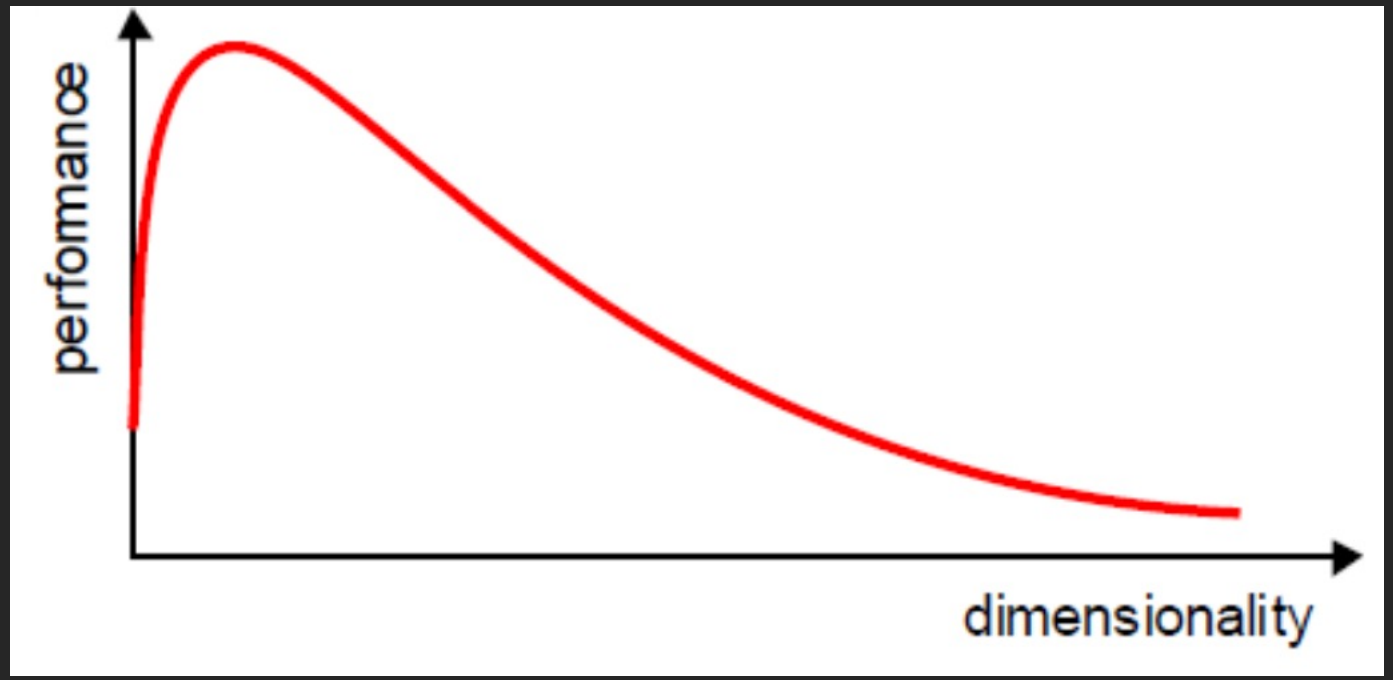
- In theory, the *curse of dimensionality* describes the phenomenon where the ***features space becomes increasingly sparse*** as the number of dimensions increases.



# 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.
  - However, it may not lead to a simpler solution, depending on the dataset.

# Curse of Dimensionality




How do we beat the curse of dimensionality?

- Incorporating prior
- Exponentially increasing the training set size
- Reducing the dimension

# 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 the original features

- Why?
  - **Visualization**: need to display low-dimensional version of a data set for visual inspection
  - **Preprocessing**: algorithms often work better with smaller numbers of features, both in terms of runtime and accuracy

# 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 (e.g. low variance)
  - For the other, most examples differ from each other (high variance)
- Which one should we keep?
  - The second, because it retains more information about the data items

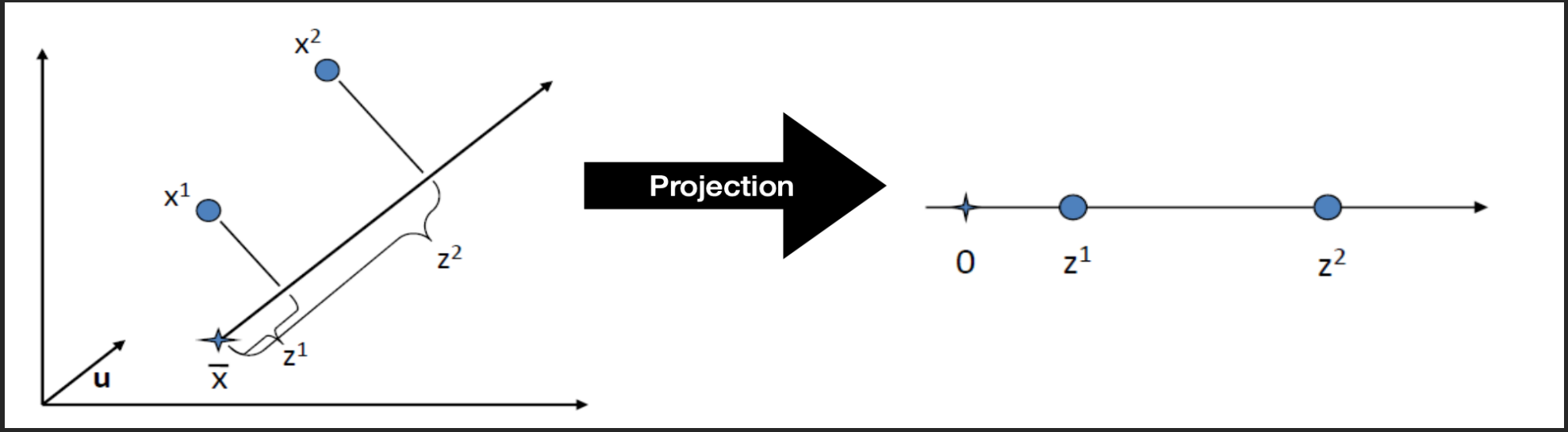
# Principle Component Analysis (PCA)

- 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

# Linear Projection

- A linear projection defines a new axis which is the result of rotating existing ones

- It is often used with the operation of translation — moving the origin of the coordination system.

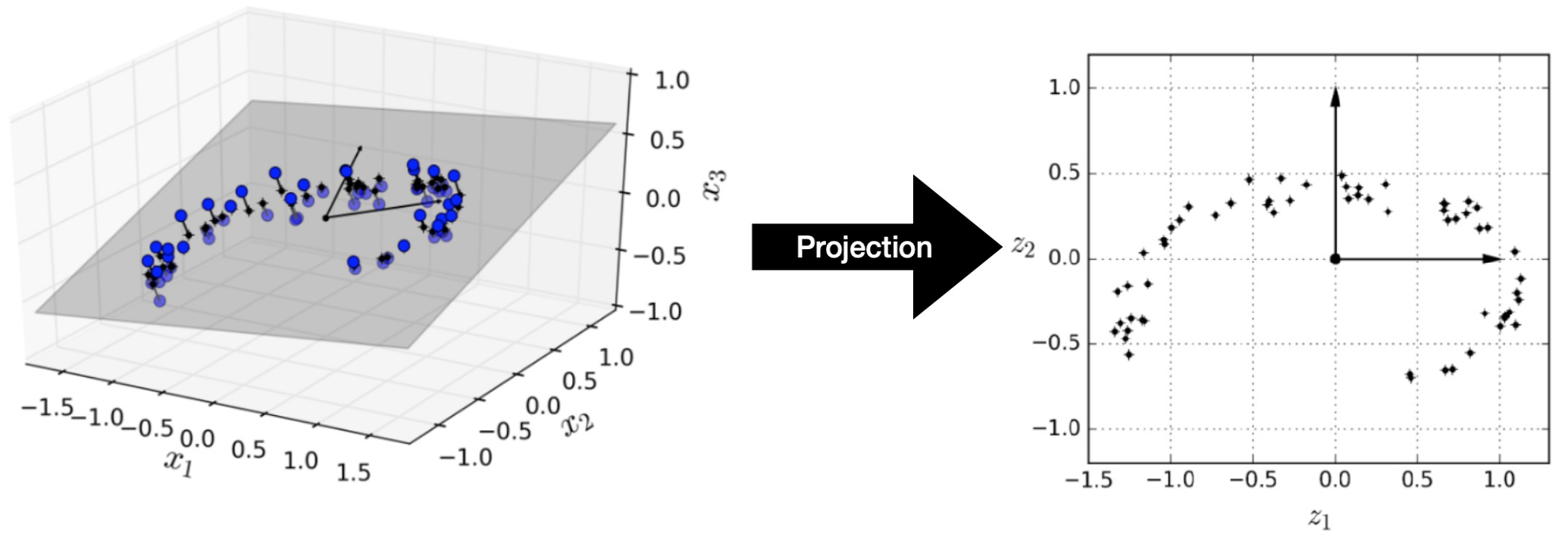


# Projection

- 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 low-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

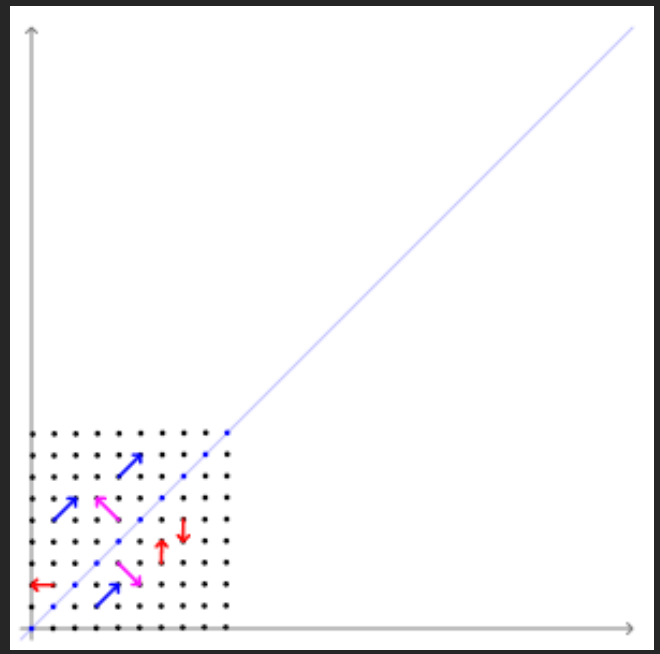


# PCA: the most popular approach

- Suppose we have a dataset $X$, with one row vector per data point
- Standardization: substracting the mean and dividing by the standard deviation (why)
- Covariance matrix computation: a symmetric matrix
  - Positive: two variables increase or decrease together
  - Negative: one increases when the other decreases
- Find eigenvectors and eigenvalues of the covariance matrix
- Principle 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.
  - 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.



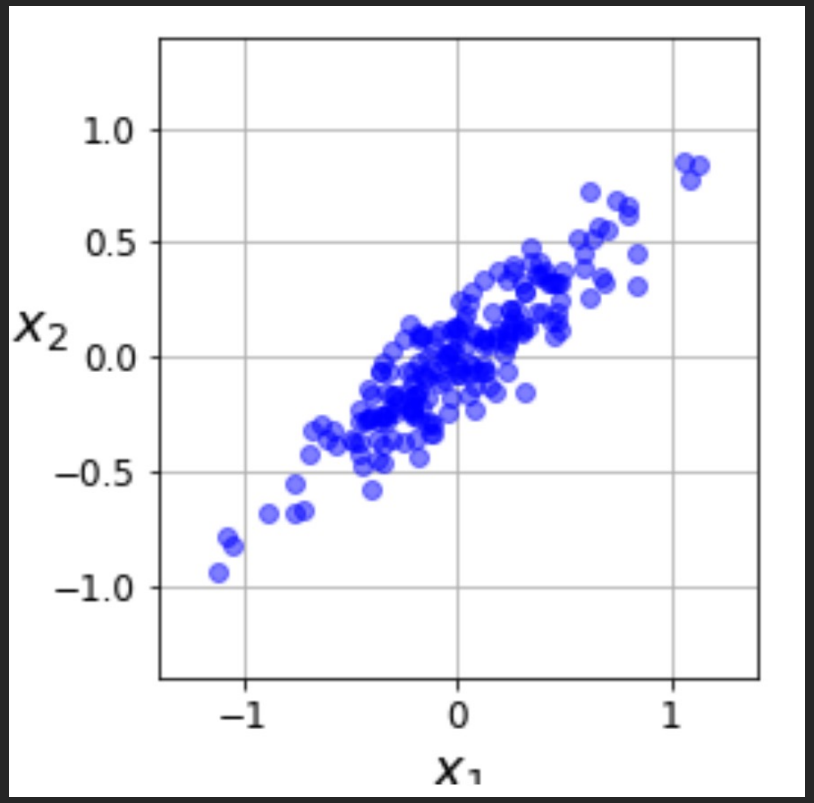
Animation from wiki

# PCA

- ***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

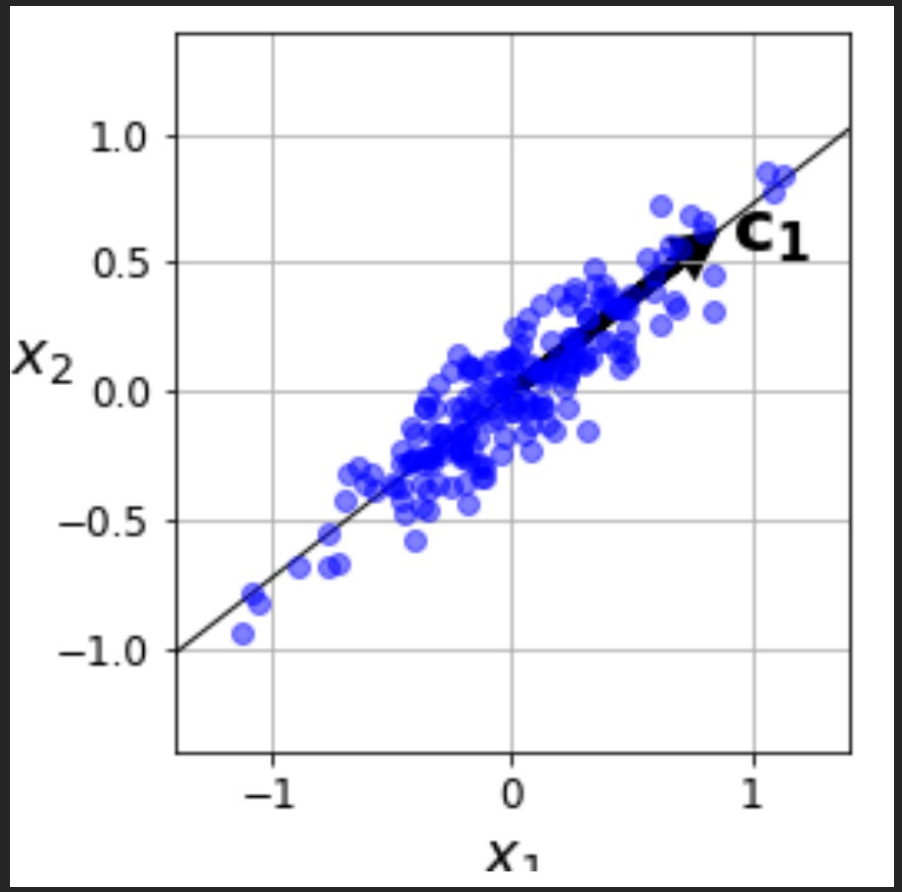
# 2D Data: Finding Principal Components

- Find axis (vector) along data with highest variance



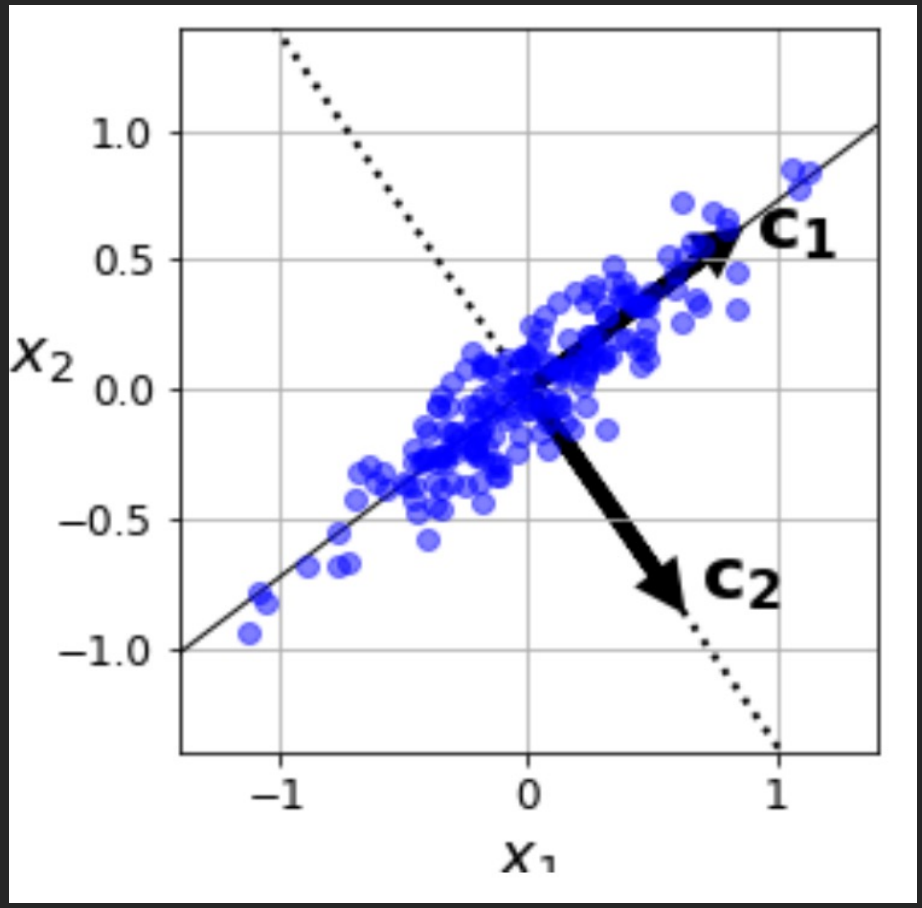
# 2D Data: Finding Principal Components

- Find axis (vector) along data with highest variance
  - Axis (vector), C1, is the axis with the largest amount of variance



# 2D Data: Finding Principal Components

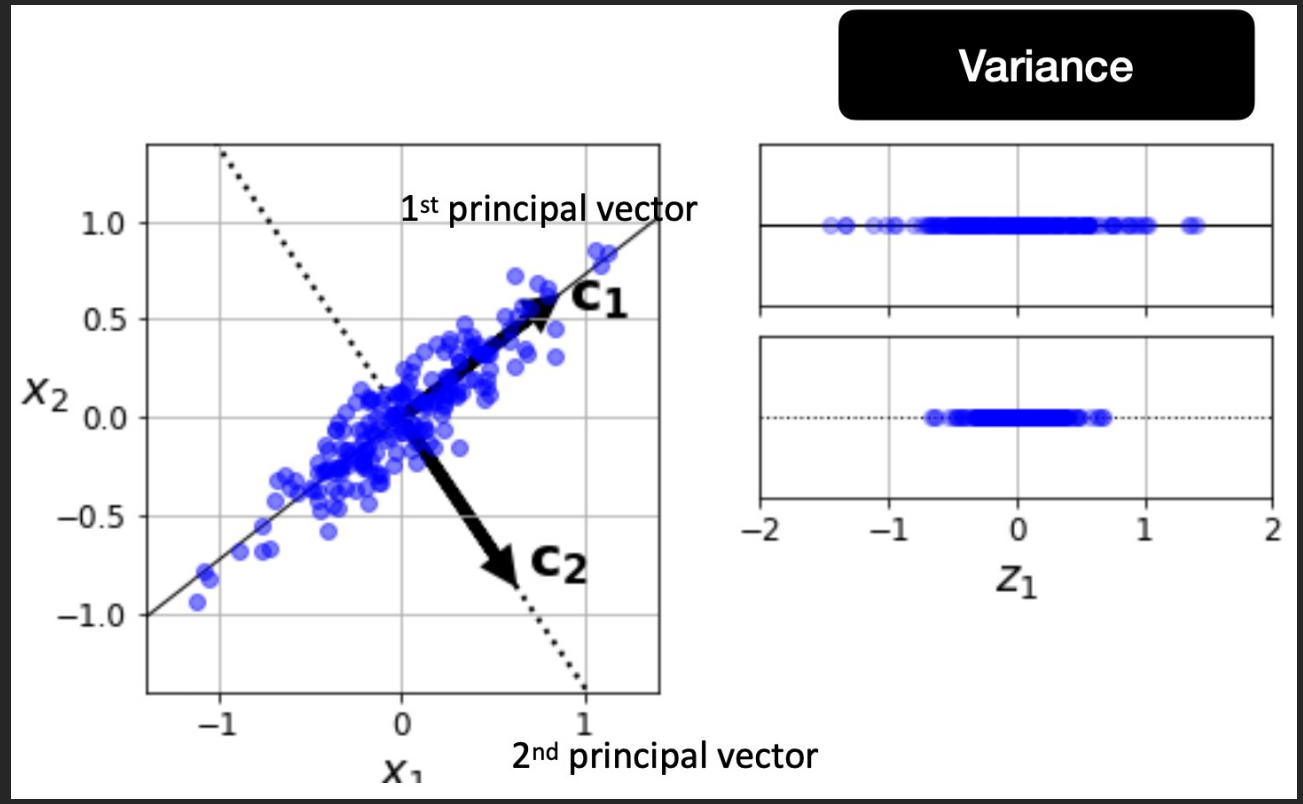
- Find axis (vector) along data with highest variance
  - Axis (vector), C1, is the axis with the largest amount of variance
  - Axis (vector), C2, is another option, but its variance is smaller

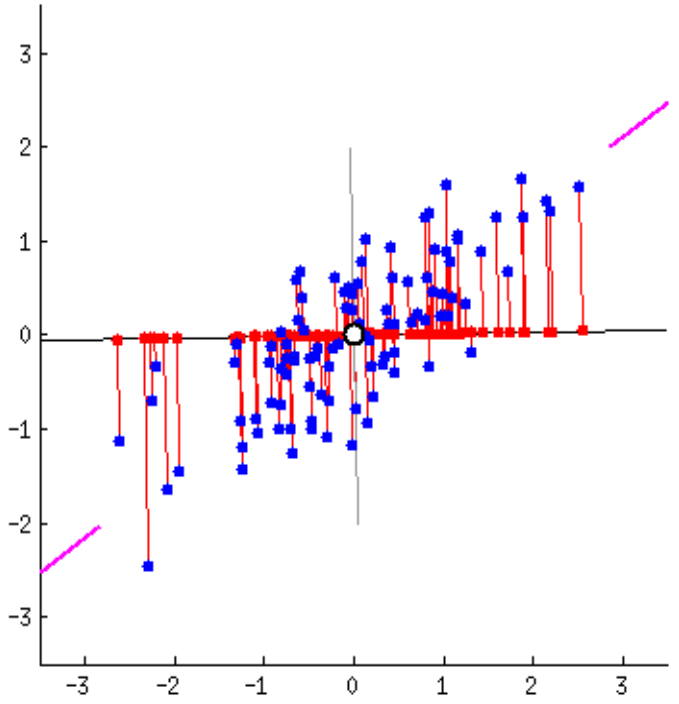


# 2D Data: Finding Principal Components

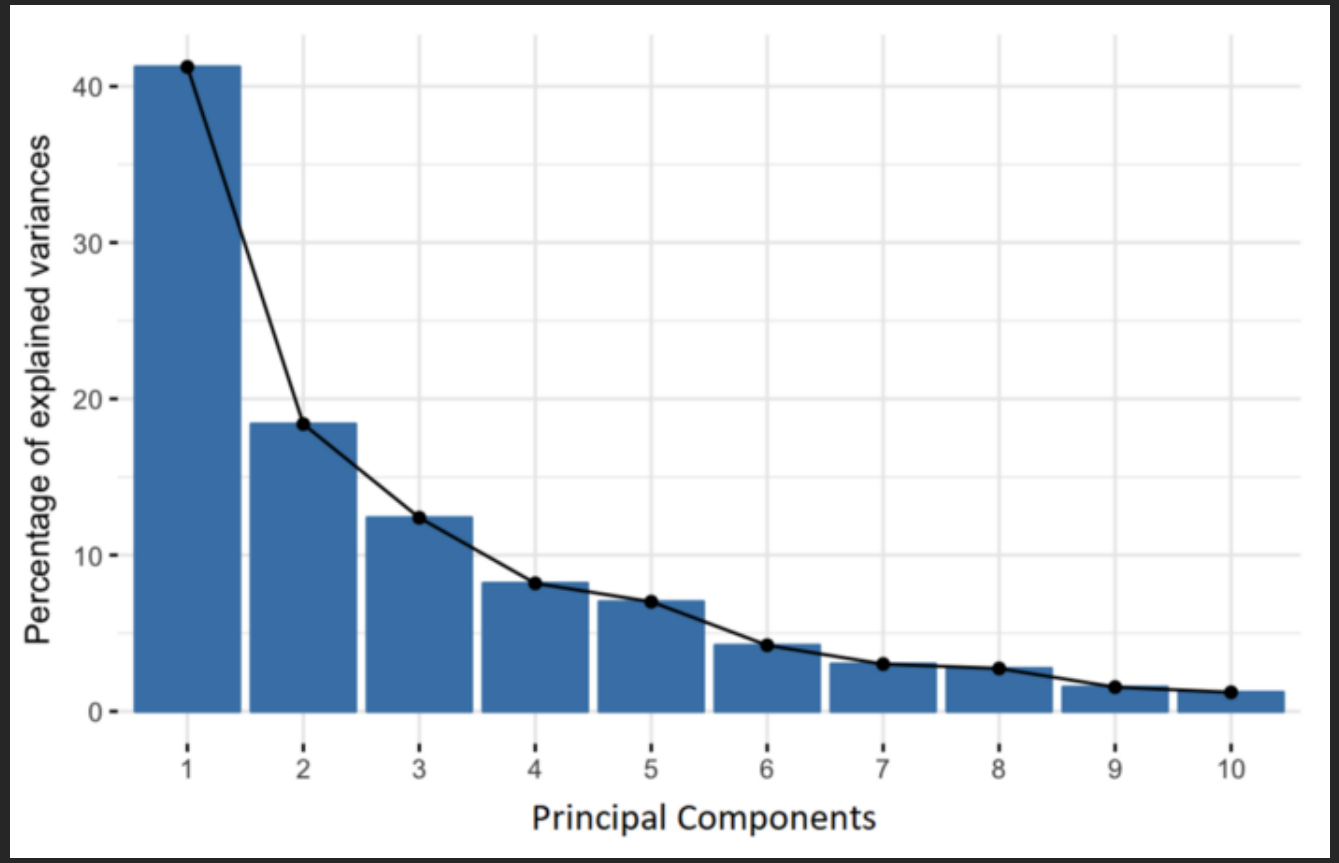
- The vectors serve as principle vectors (or components)

- Principle vectors are ***orthogonal to each other***




3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3

# 2D Data: Finding Principal Components



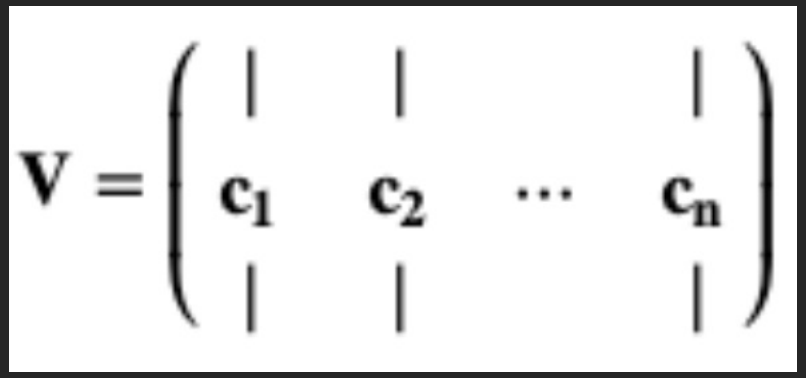

The first few, e.g. 3, PC Can account for large mount of data variance.

# 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

# Computing Eigenvectors and Eigenvalues

- For instance, a given matrix $\boldsymbol{S}$, can be factored into $\boldsymbol{U\Sigma V^T}$
  - $\boldsymbol{S \approx U\Sigma V^T}$
  - The columns of $V$ contain the Eigenvectors
  - (e.g. principal components) that we are interested in
  - The scalars along the diagonal of $\Sigma$ contain the Eigenvalues
  - $U$ also has Eigenvectors, but we aren't interested in these



# PCA for Dimensionality Reduction

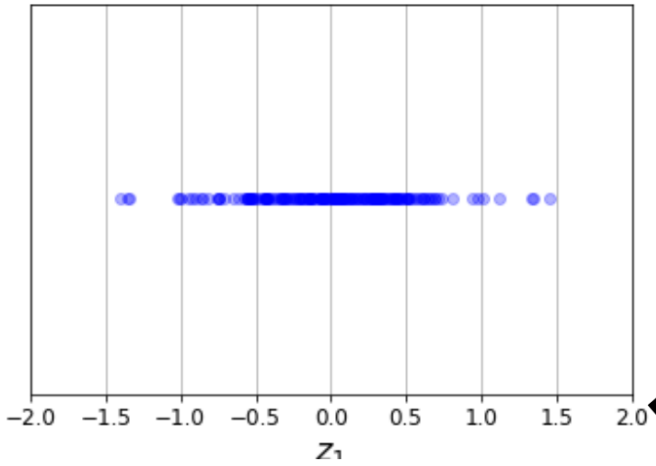
- After principal components (e.g. eigenvectors) have been identified, dimensionality reduction can be accomplished as follows:
  - Assume the desired new dimension is $d$ and $d < D$ (the original)
  - Project original dataset onto the hyperplane using the d principal components
    - Compute the matrix multiplication of the dataset and an Eigenvector matrix that only contains d columns $W_d$
    - $X_{d-proj=}XW_d$

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

# Example

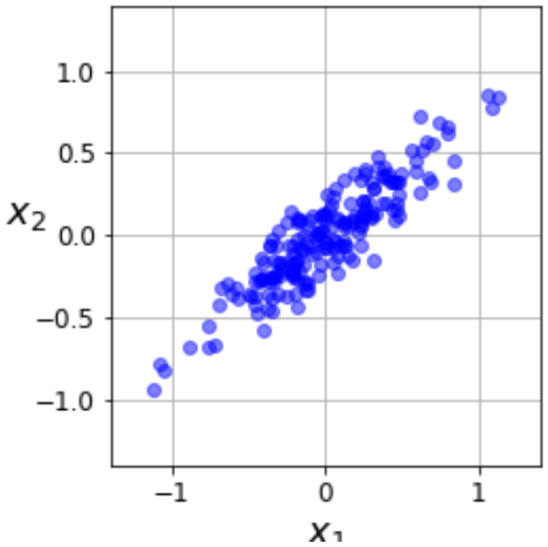
$$\mathbf{X}_{reduced} = (\mathbf{X} - \mu_{\mathbf{X}})\mathbf{W}_d$$

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

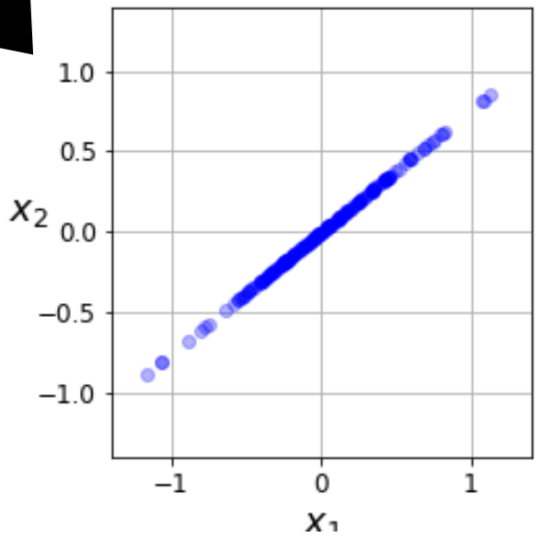



$$\mathbf{X}_{recovered} = \mathbf{X}_{reduced}\mathbf{W}_d^T$$

**Original Data, $\mathbf{X}$**




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




**Information is Lost!**

# PCA Properties

- *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
  - 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*

# Face Recognition: supervised learning

- A typical image of size $256\times128$ is described by $n = 256\times128 = 32768$ 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

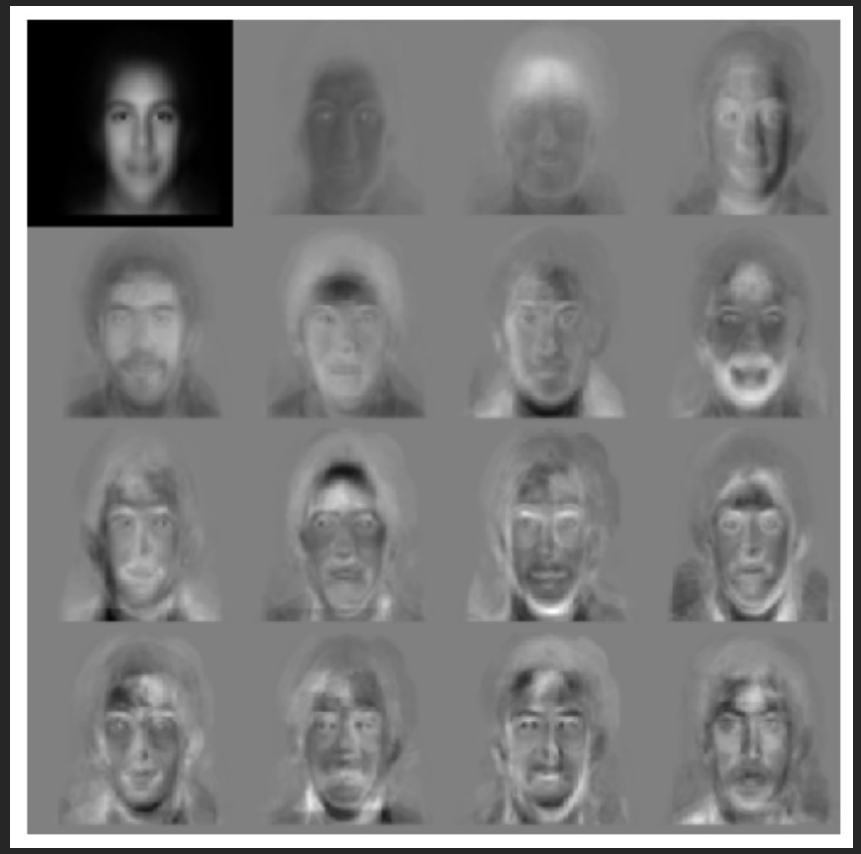
# Eigen-faces

- 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 (Turk and Pentland 1991)

- 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

- Goal:
  - Given one image of a person, find the other images of this person
  - Different amounts of variations were separately introduced to the data (lighting, orientation, size,…)

- Nearest Neighbor classifier in the eigenface space

# Face Recognition (Turk and Pentland 1991)

- Result 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