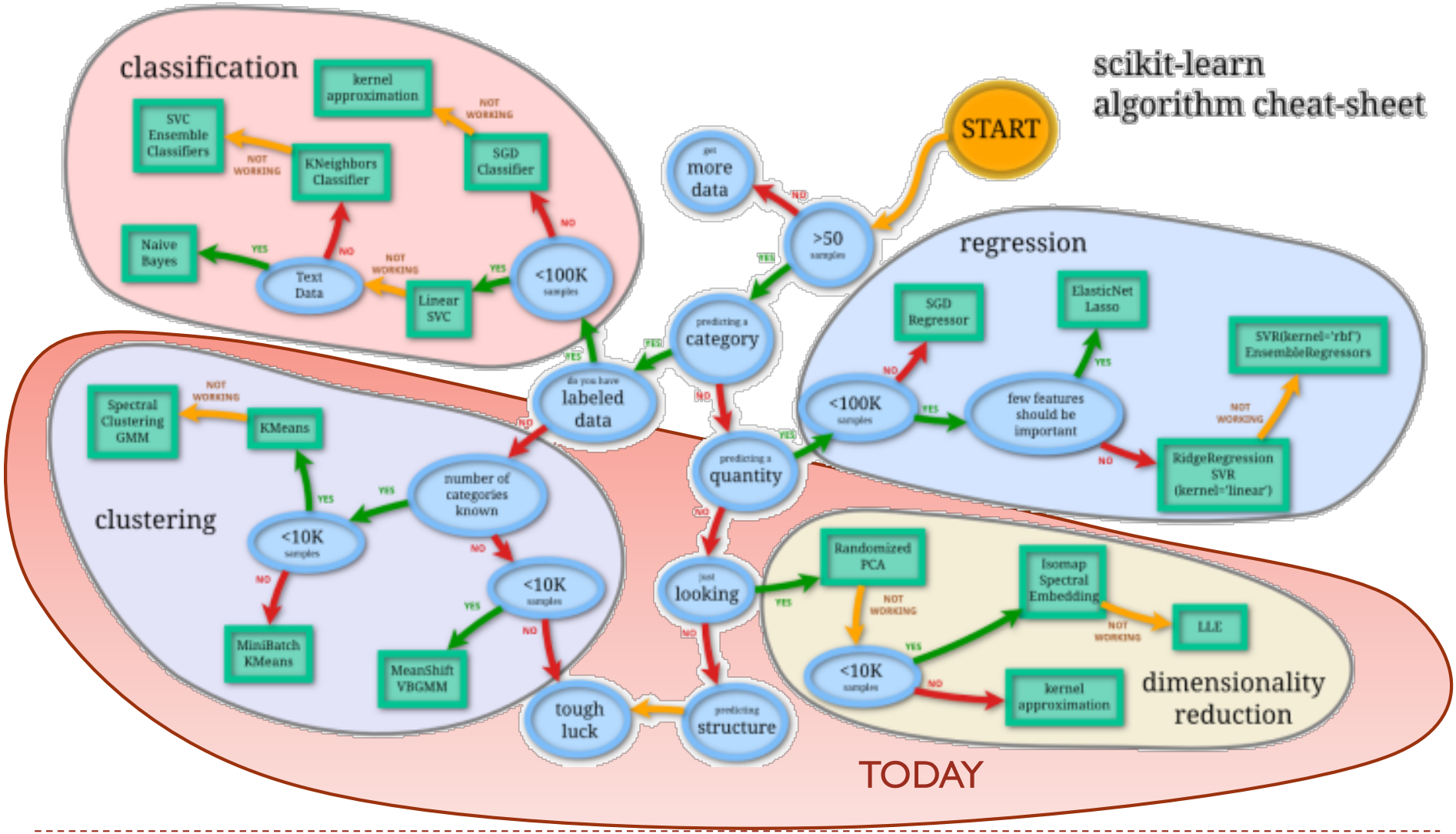
Machine Learning:
Unsupervised learning

Rodrigo Cabral, Lionel Fillatre and Michel RIVEILL
michel.riveill@univ-cotedazur.fr

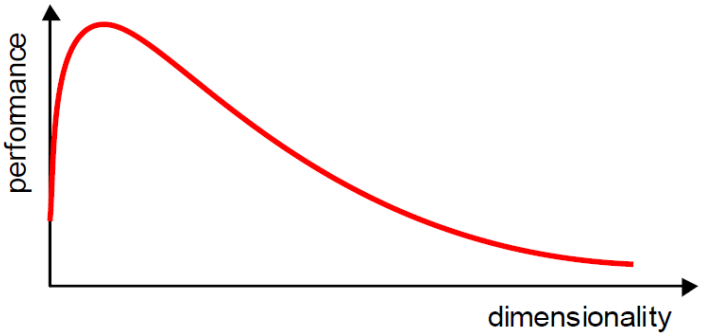
The scikit learn models



Dimension reduction

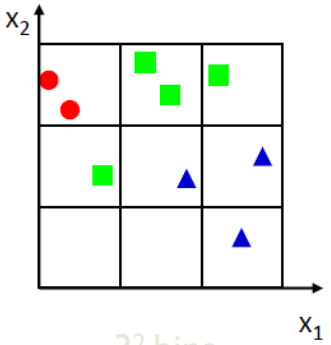
Curse of Dimensionality

- Increasing the number of features will not always improve classification accuracy.
- In practice, the inclusion of more features might actually lead to worse performance.
- The number of training examples required increases exponentially with dimensionality **d** (i.e., $k^{\mathbf{d}}$).

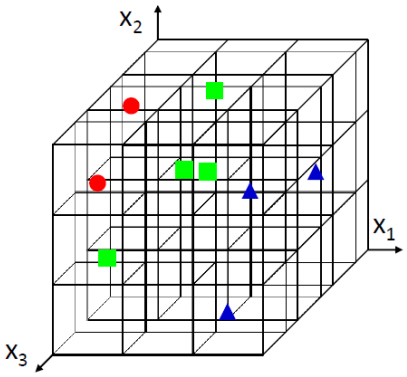


k=3

3^1 bins



3^2 bins

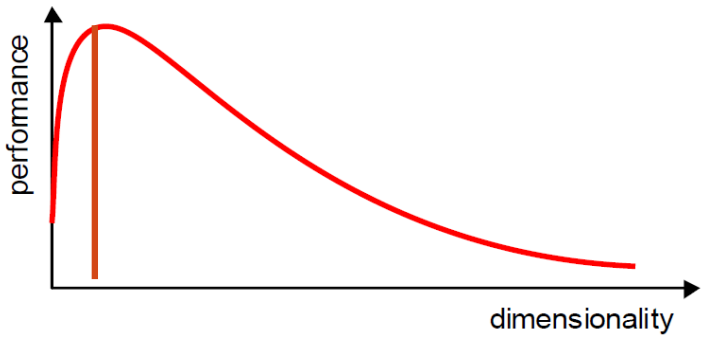


3^3 bins

k: number of bins per feature

Dimensionality Reduction

- What is the objective?
  - Choose an optimum set of features of lower dimensionality to improve classification accuracy.



- Different methods can be used to reduce dimensionality:
  - Feature extraction
  - Feature selection

Dimensionality Reduction (cont'd)

- **Feature extraction**: finds a set of new features (i.e., through some mapping f()) from the existing features.

The mapping f() could be linear or non-linear

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ x_N \end{bmatrix} \xrightarrow{f(\mathbf{x})} \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \cdot \\ \cdot \\ y_K \end{bmatrix}$$

K<<N

- **Feature selection:** chooses a subset of the original features.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ x_N \end{bmatrix} \rightarrow \mathbf{y} = \begin{bmatrix} x_{i_1} \\ x_{i_2} \\ \cdot \\ \cdot \\ x_{i_K} \end{bmatrix}$$

K<<N

Feature Extraction

- Linear combinations are particularly attractive because they are simpler to compute and analytically tractable.

- Given $x \in R^N$, find an K x N matrix T such that:

$$y = Tx \in R^K \text{ where } K<<N$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ x_N \end{bmatrix} \xrightarrow{f(\mathbf{x})} \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \cdot \\ \cdot \\ \cdot \\ y_K \end{bmatrix}$$

T

This is a projection from the N-dimensional space to a K-dimensional space.

7

Feature Extraction (cont'd)

- From a mathematical point of view, finding an optimum mapping $\mathbf{y}=f(\mathbf{x})$ is equivalent to optimizing an **objective** criterion.

- Different methods use different objective criteria, e.g.,

  - Minimize Information Loss: represent the data as accurately as possible in the lower-dimensional space.

  - Maximize Discriminatory Information: enhance the class-discriminatory information in the lower-dimensional space.

Feature Extraction (cont'd)

- Popular linear feature extraction methods:
  - Principal Components Analysis (PCA): Seeks a projection that **preserves** as much **information** in the data as possible.
  - Linear Discriminant Analysis (LDA): Seeks a projection that **best discriminates** the data.

- Many other methods:
  - Making features as independent as possible (Independent Component Analysis or ICA).
  - Retaining interesting directions (Projection Pursuit).
  - Embedding to lower dimensional manifolds (Isomap, Locally Linear Embedding or LLE).

9

Principle Components Analysis (PCA)

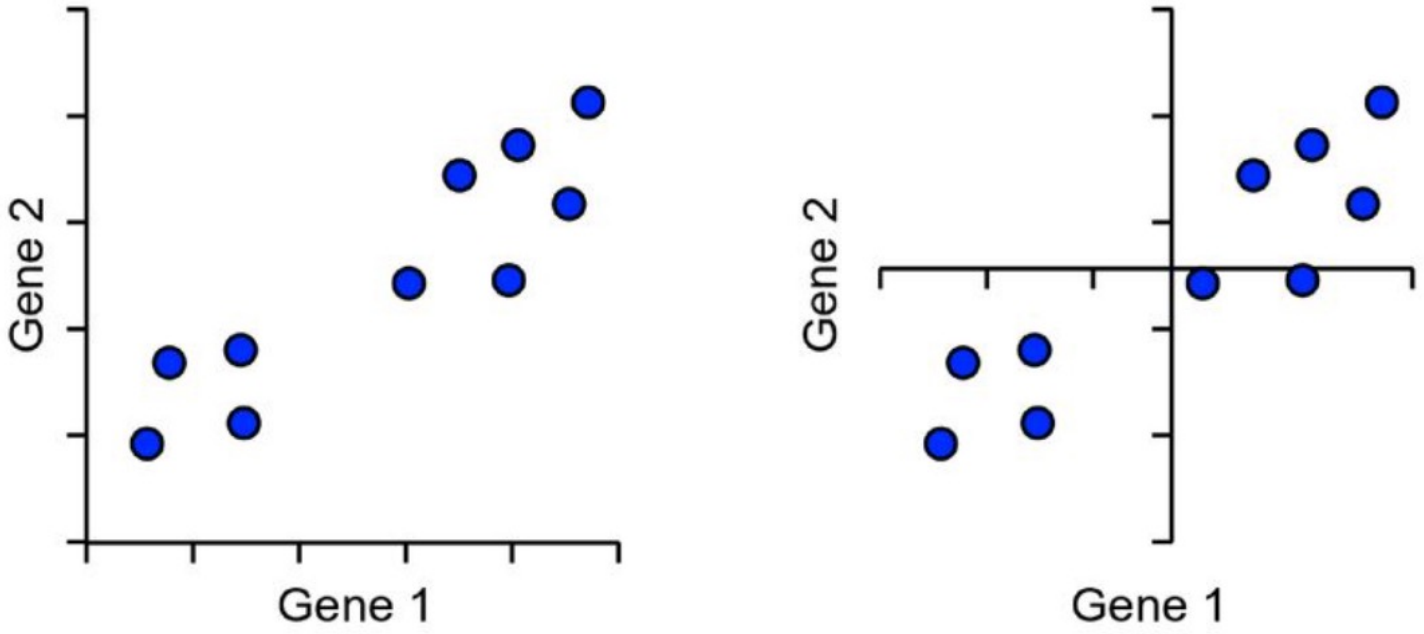
- Method to optimally summarize large multi-dimensional datasets
- Can find a smaller number of dimensions (ideally 2 for plotting) which retain most of the useful information in the data

- Builds a recipe for converting large amounts of data into a single value, called a Principle Component (PC), e.g.:

  - PC = (GeneA*10)+(GeneB*3)+(GeneC*-4)+…

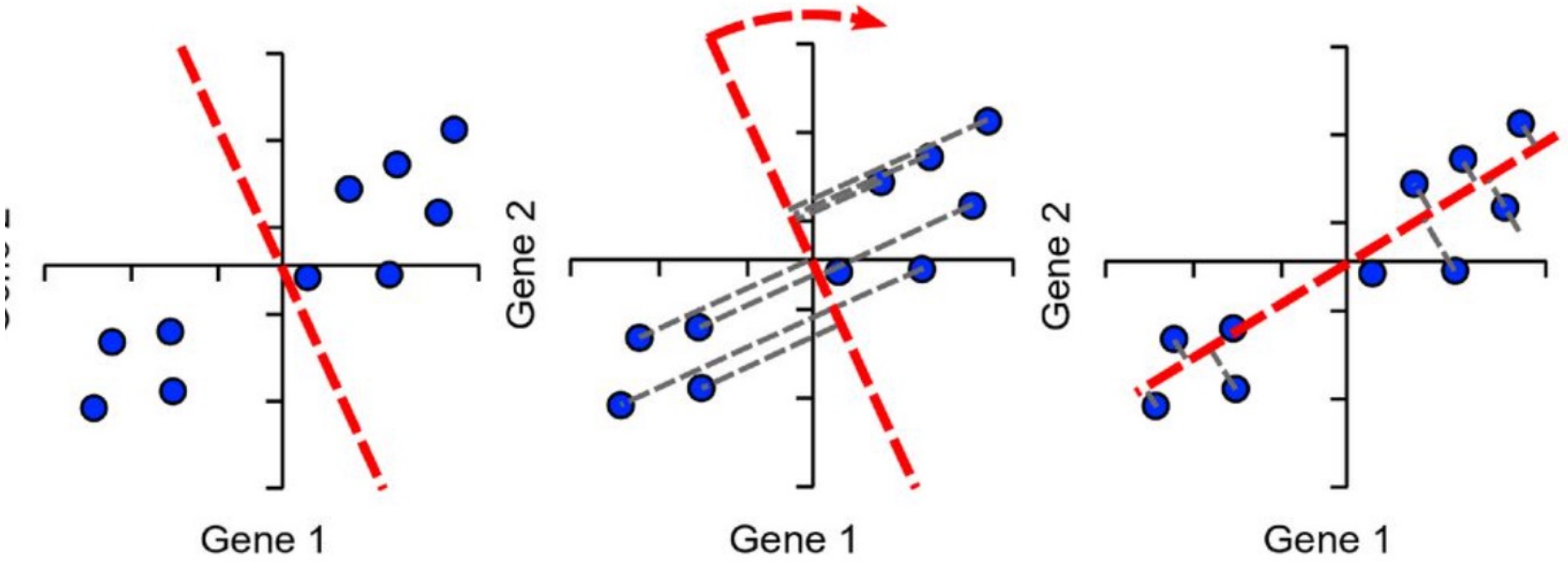
How doed PCA work?

- Simple example using 2 genes and 10 cells

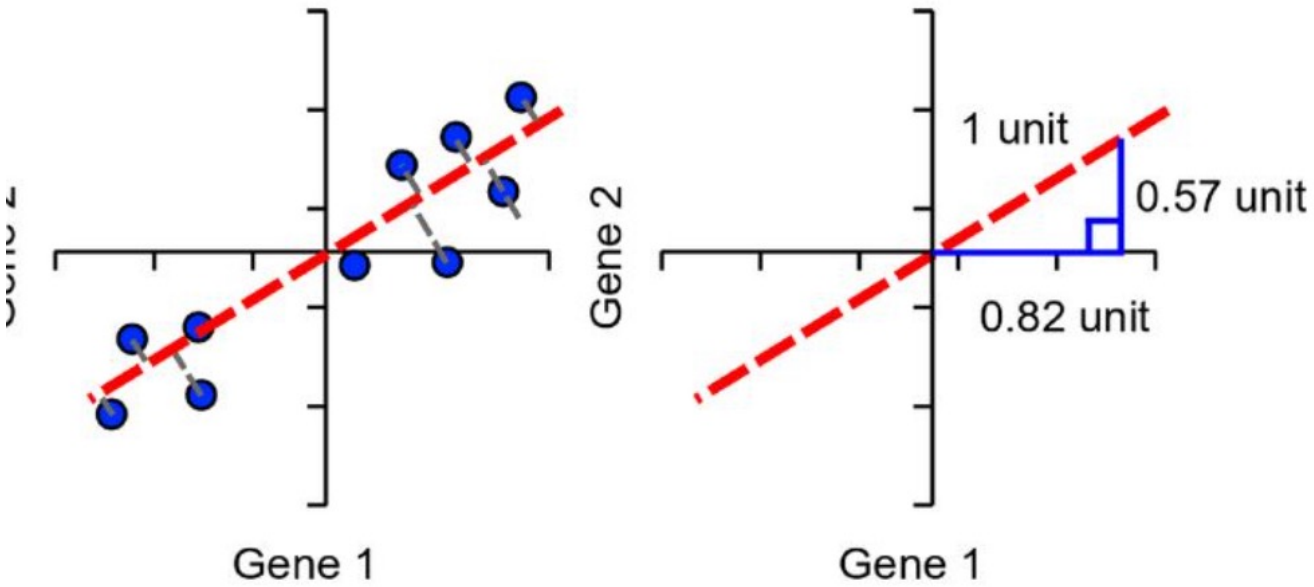


How does PCA work?

- Find line of best fit, passing through the origin



Assigning Loadings to Genes



Single Vector or
'eigenvector'

Loadings:

- Gene1 = 0.82
- Gene2 = 0.57

Higher loading equals more influence on PC

More dimensions

- The same idea extends to larger numbers of dimensions (n)
- Calculation of first PC rotates in (n-1) dimensions
  - Next PC is perpendicular to PC2, but rotated similarly (n-2)
  - Last PC is remaining perpendicular (no choice)
  - Same number of PCs as genes
- Each PC always explains some proportion of the total variance in the data.
  - Between them they explain everything
    - PC1 always explains the most
    - PC2 is the next highest
    - Etc.
  - Since we only plot 2 dimensions we'd like to know that these are a good explanation
- How do we calculate this?

PCA algorithms

1. Standardize the *d*-dimensional dataset.
2. Construct the covariance matrix.
3. Decompose the covariance matrix into its eigenvectors and eigenvalues.
4. Sort the eigenvalues by decreasing order to rank the corresponding eigenvectors.
5. Select *k* eigenvectors which correspond to the *k* largest eigenvalues, where *k* is the dimensionality of the new feature subspace ($k \leq d$).
6. Construct a projection matrix ***W*** from the "top" *k* eigenvectors.
7. Transform the *d*-dimensional input dataset ***X*** using the projection matrix ***W*** to obtain the new *k*-dimensional feature subspace.

- PCA in action: https://www.youtube.com/watch?v=HMOI_lkzW08&t=2s

- Luckily we have libraries that do all these calculations for us and make it very easy for us to get:- the different components- the eigen values

Another approach for the same result

Singular Value Decomposition (SVD)

- Handy mathematical technique that has application to many problems
- Given any *m*×*n* matrix **X**, algorithm to find matrices **U**, **V**, and ∑ such that

$$\mathbf{X} = \mathbf{U} \sum \mathbf{V}^T$$

U is *m*×*n* and orthonormal

∑ is *n*×*n* and diagonal

V is *n*×*n* and orthonormal

$$\begin{pmatrix} \\ \mathbf{X} \\ \\ \end{pmatrix} = \begin{pmatrix} \\ \mathbf{U} \\ \\ \end{pmatrix} \begin{pmatrix} w_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & w_n \end{pmatrix} \begin{pmatrix} \\ \mathbf{V} \\ \\ \end{pmatrix}^T$$

SVD

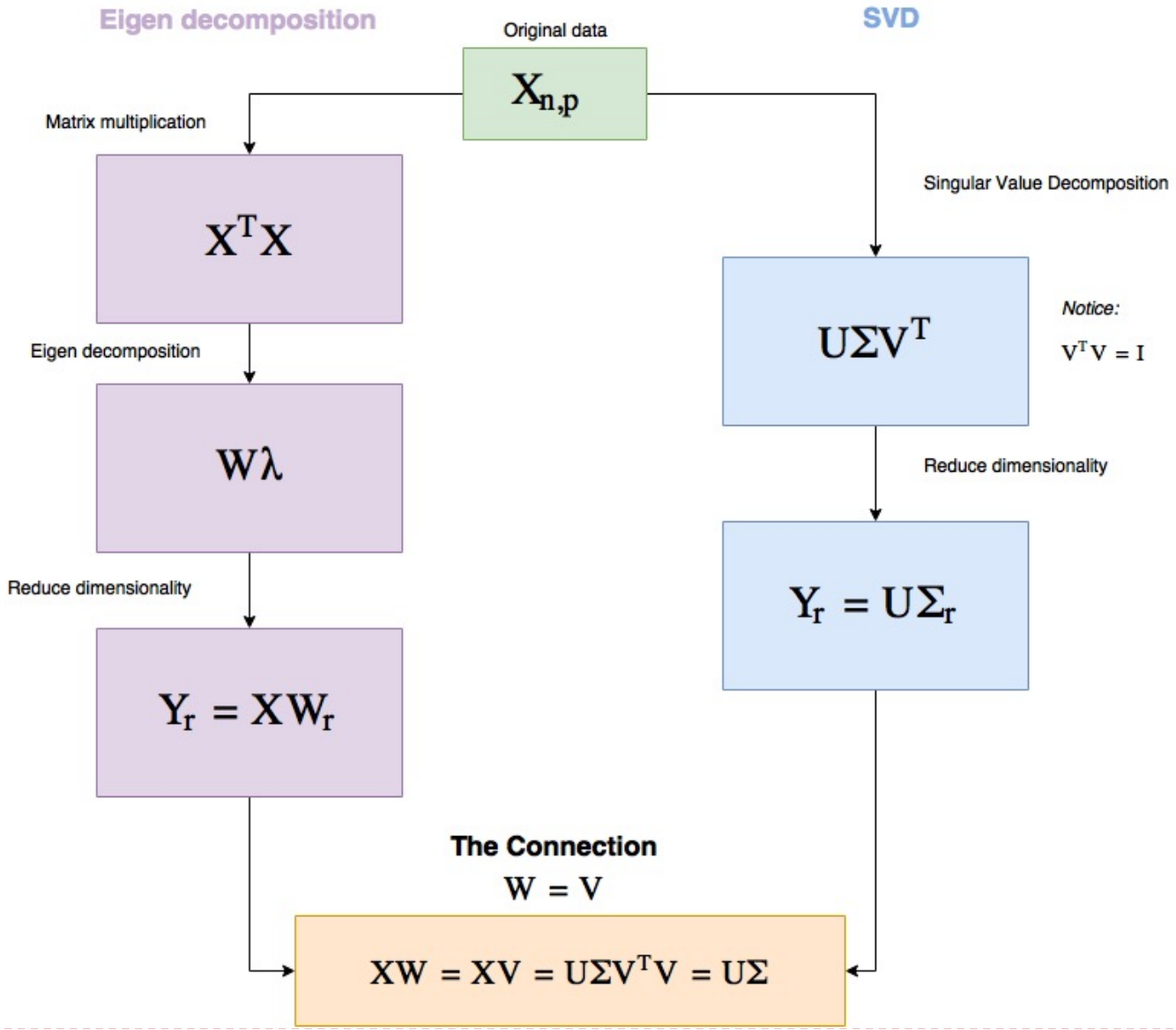
$\mathbf{X} = \mathbf{U} \sum \mathbf{V}^T$

- The σ_i are called the **singular values** of $\mathbf{X}$
  - Elements of $\sum$ are sqrt(eigenvalues)
- SVD is unique if we put the w_i descending order

- If $\mathbf{X}$ is singular, some of the w_i will be 0
- In general *rank*($\mathbf{X}$) = number of nonzero w_i

- and columns of $\mathbf{V}$ are eigenvectors of $\mathbf{X}^T\mathbf{X}$

Dimension reduction
link between PCA and SVD



Reduction dimension with PCA/SVM

- PCA or SVM
  - can be used to create isolated feature vectors which can be analyzed independently.
  - Since they both get us much smaller matrices than the original dataset, they can also be used for data compression via dimensionality reduction.

- Given the isolated feature vectors
  - Select the ones that best represent the data and its variations,
  - then re-project the data onto the isolated vectors.

The previous slide, illustrate process and connection between the 2 techniques for dimensionality reduction.

- depending on the functions available in the library you use, you can use
  - PCA = XW
  - SVD = U∑
- to reduce the size of the original data to rank k

Reduction dimension with PCA/SVM

- The problem
  - Takes a data matrix of n observations by p variables/features, which may be correlated,
  - Summarizes it by uncorrelated axes (principal components or principal axes)
    - linear combinations of the original p variables
- The first k components display as much as possible of the variation among objects.
  - k << p
  - For plotting: k=2

Step 1: Feature scaling

- feature scaling / mean normalization
  - **It's very important to normalize the range of the feature**
  - **Goal**: put all features in a similar range of values
  - Many possibilities:
    - **feature scaling:** replace each value ρby $(\rho - \rho_{min})/(\rho_{max} - \rho_{min}) \rightarrow [0, 1]$
    - **mean normalization:** replace each value ρby $(\rho - \mu_{\rho})/(\rho_{max} - \rho_{min})$
    - **Standardization:** (all feature have zero-mean) replace ρby $(\rho-\mu_{\rho})/\sigma$
      - σ is the standard deviation of all mean

- In Python
  - **sklearn.preprocessing.MinMaxScaler scale between [0, 1]**
    - **x_scale = MinMaxScaler.fit_transform(x)**
  - **sklearn.preprocessing.StandardScaler scale between [-1, 1] and** μ=0
    - **x_norm = StandardScaler().fit_transform(x)**

Step 2: sklearn.decomposition.PCA

- Main input paramaters
  - **n_components:** Number of components to keep

- Main attributes
  - **Components:**
    - Principal axes in feature space, representing the directions of maximum variance in the data.
    - Equivalently, the right singular vectors of the centered input data, parallel to its eigenvectors.
    - The components are sorted by explained_variance_.
  - **explained_variance:**
    - The amount of variance explained by each of the selected components.

- https://jakevdp.github.io/PythonDataScienceHandbook/05.09-principal-component-analysis.html

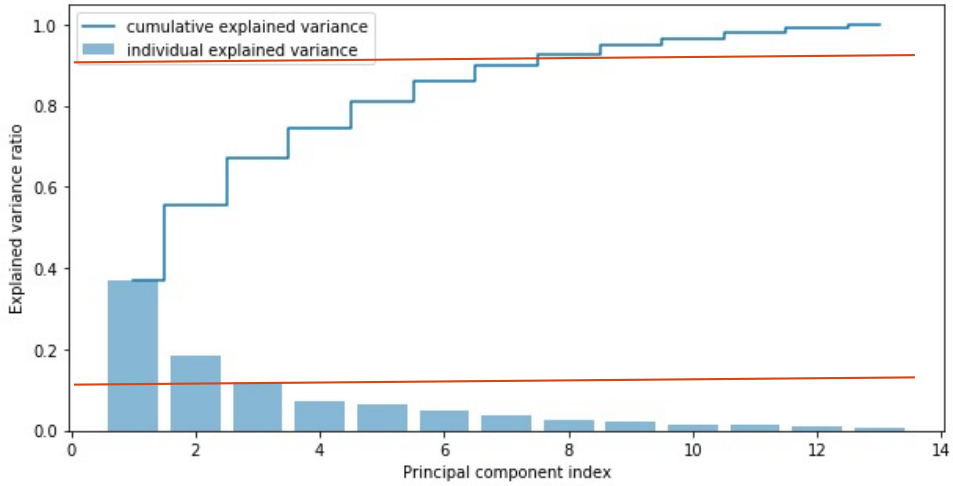
Explained variance

- pca.fit(crime_dataser)
- contrib = pd.DataFrame(pca.components_, index=df.columns)
- Contribution for the first PCA axes
  - contrib[0].sort_values(ascending=False).head(9)
    - Robbery with violence 0.568208
    - Homicides 0.532556
    - Infractions 0.338312
    - Burglaries 0.242733
    - Prison population 0.176146
    - Violent crimes 0.037280
    - Drug traffic -0.081503
    - Motor vehicle theft -0.226053
    - Police officers -0.360540

23

Choice of 'k' value (number of components)

- **pca.explained_variance_**
- **If the objective is to reduce dimension (for plotting k=2 or 3)**
  - **Plot the explained_variance_**
    - in the explained variance, you have in fact the eigen value of the SVD decomposition
  - Elbow criterion (Cattel scree-test)
    - On the eboulis of the eigen values (value of the sigma matrix), one observes a decrease (elbow) followed by a regular decrease.
    - We stop at the first inflexion
  - Kaiser criterion:
    - Eigen values that are above the mean of the singular values are retained.
    - or if the data are normed, Eigen values > 1
  - Ratio criterion
    - keep any component representing at least X% of the total variance (generally X is between 5 and 10%)
  - Explained variance
    - The Eigen values are retained until the cumulation makes it possible to explain 90% of the variance.



90% explained variance

Kaiser criterion

Mean of Eigen value is around 1.0

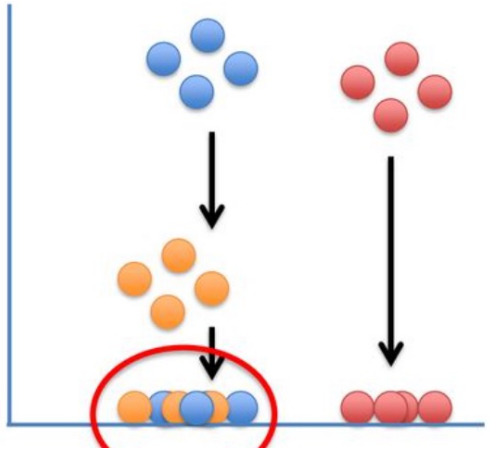
t-SNE

T-Distributed Stochastic Neighbour Emebedding (tSNE)

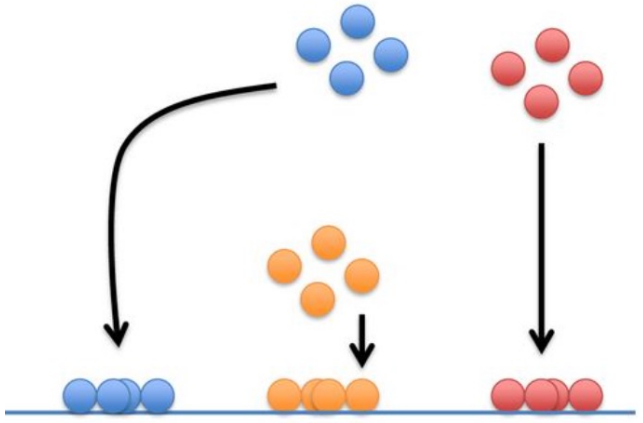
- Main drawback of PCA
  - highly influenced by outliers
  - PCA is a **linear projection**, it can't capture non-linear dependencies.

- tSNE: aims to solve the problems of PCA
  - Non-linear scaling to represent changes at different levels
  - Optimal separation in very low dimensions (1, 2 or three)

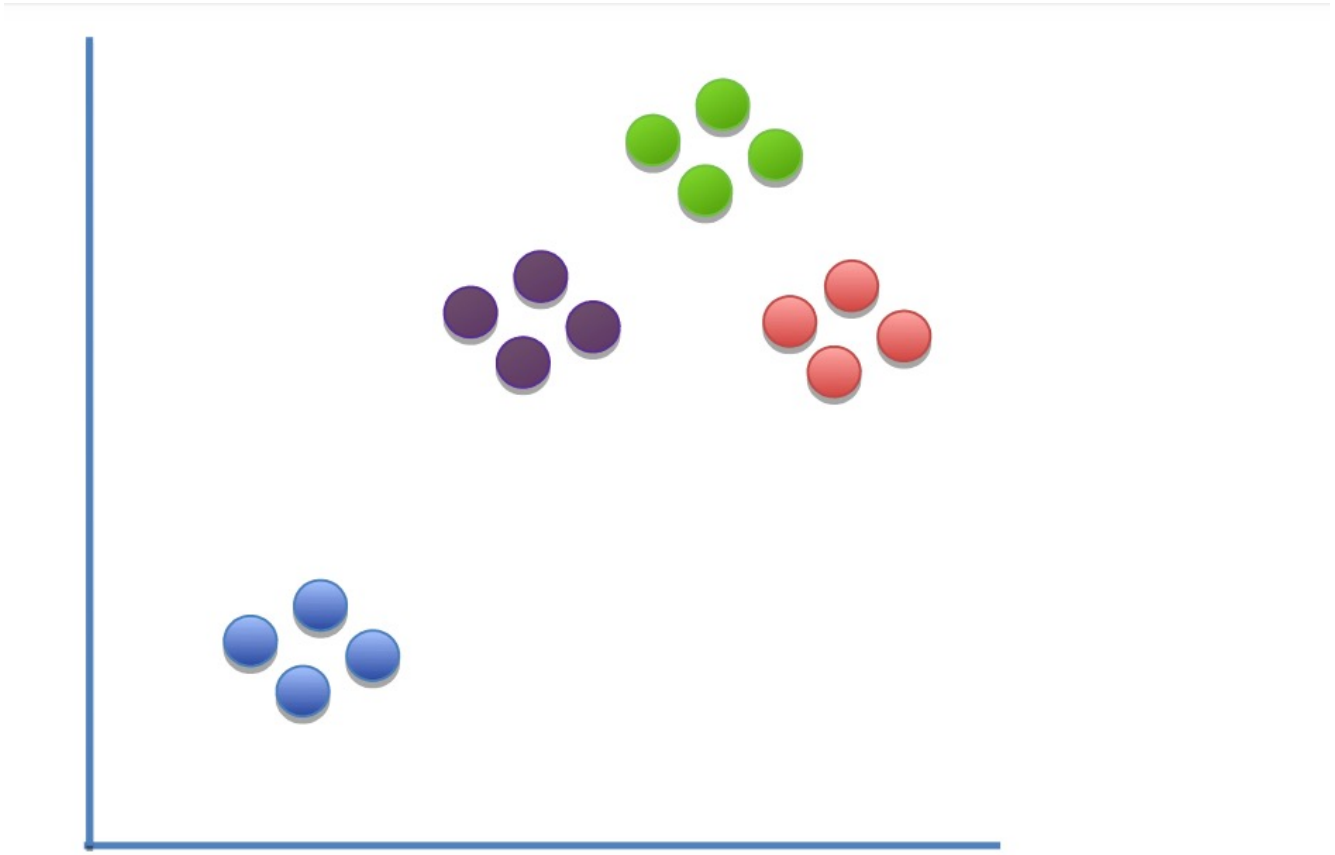
- Example: project 2D data in a 1D representation



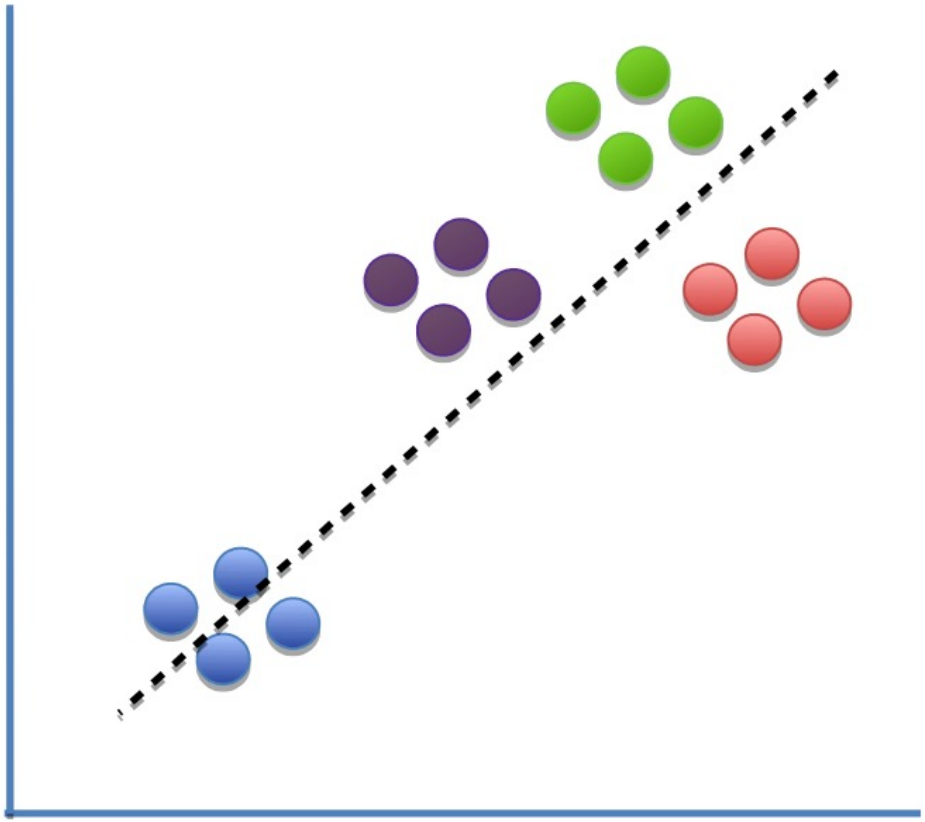
What is the better projection ?



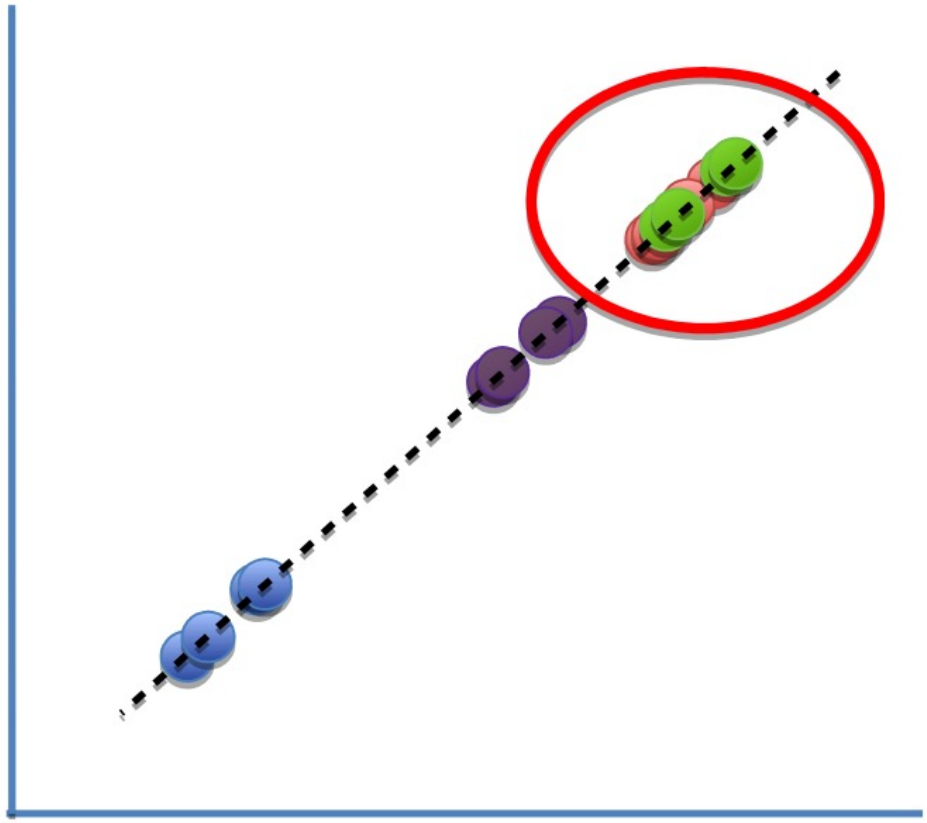
PCA projection from 2D to 1D



PCA projection from 2D to 1D



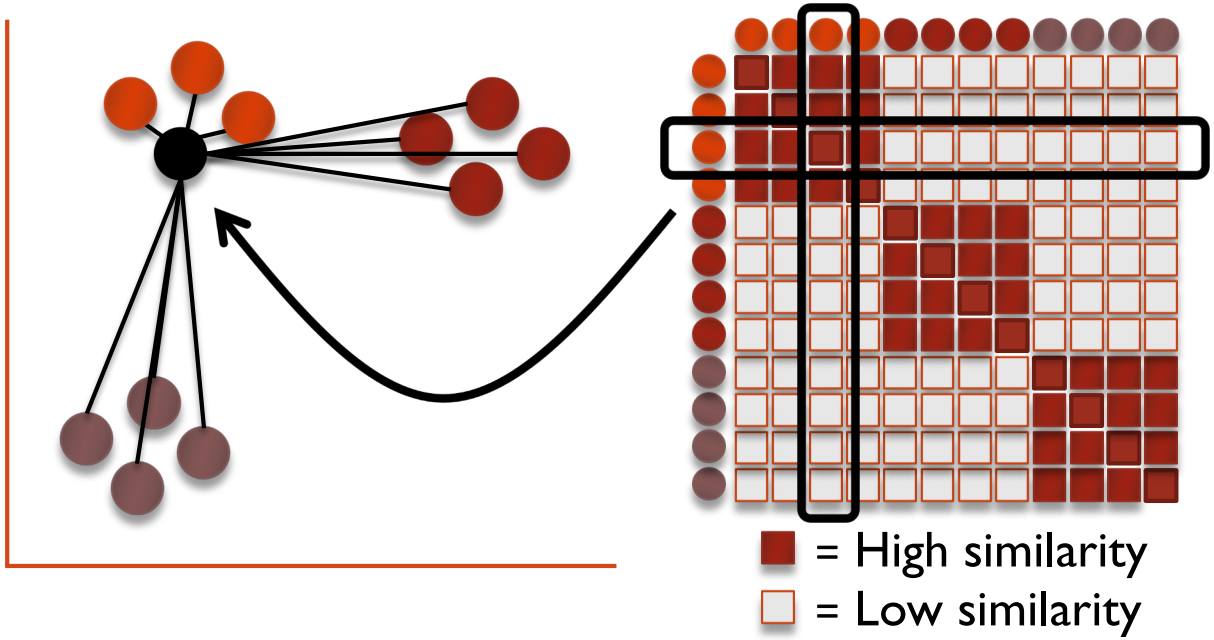
PCA projection from 2D to 1D



No too good

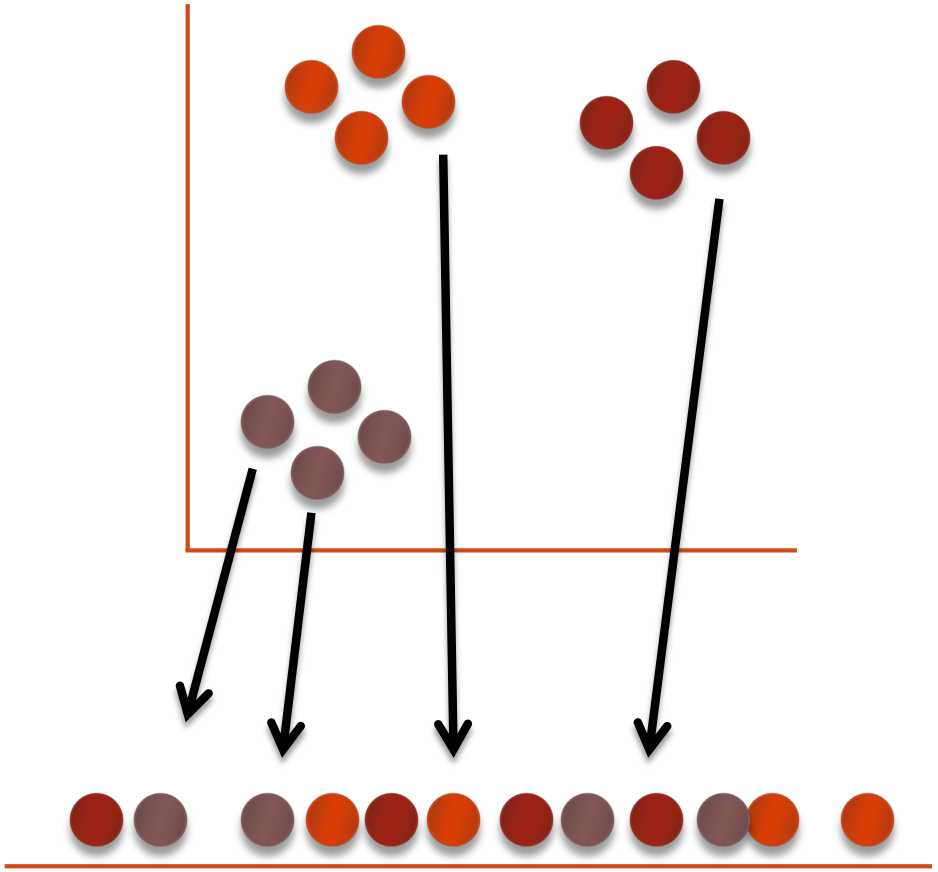
tSNE

- Step1: calculate similarities between all point



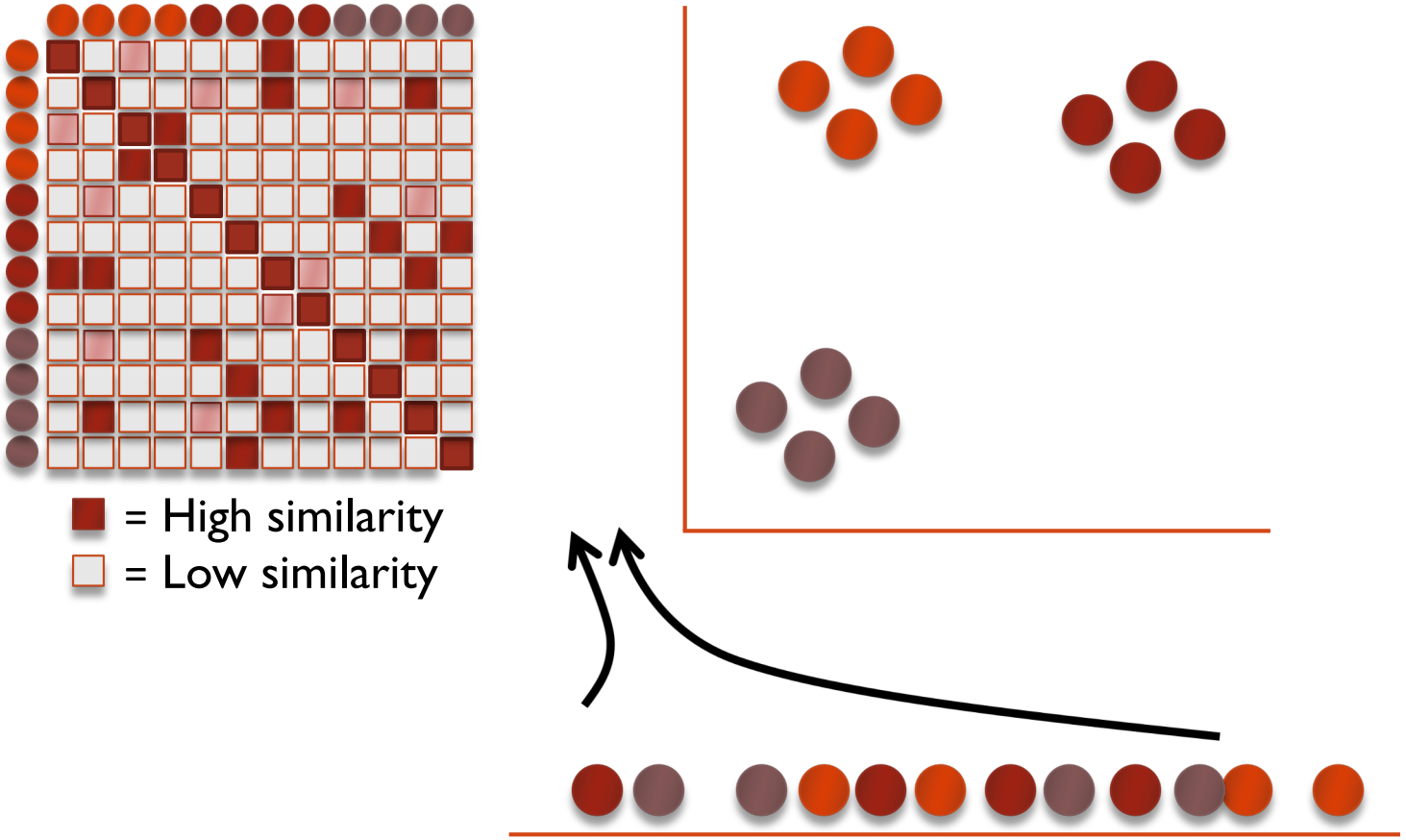
tSNE

- Step1: calculate similarities between all point
- Step2: randomly project the data onto the desired number of axis



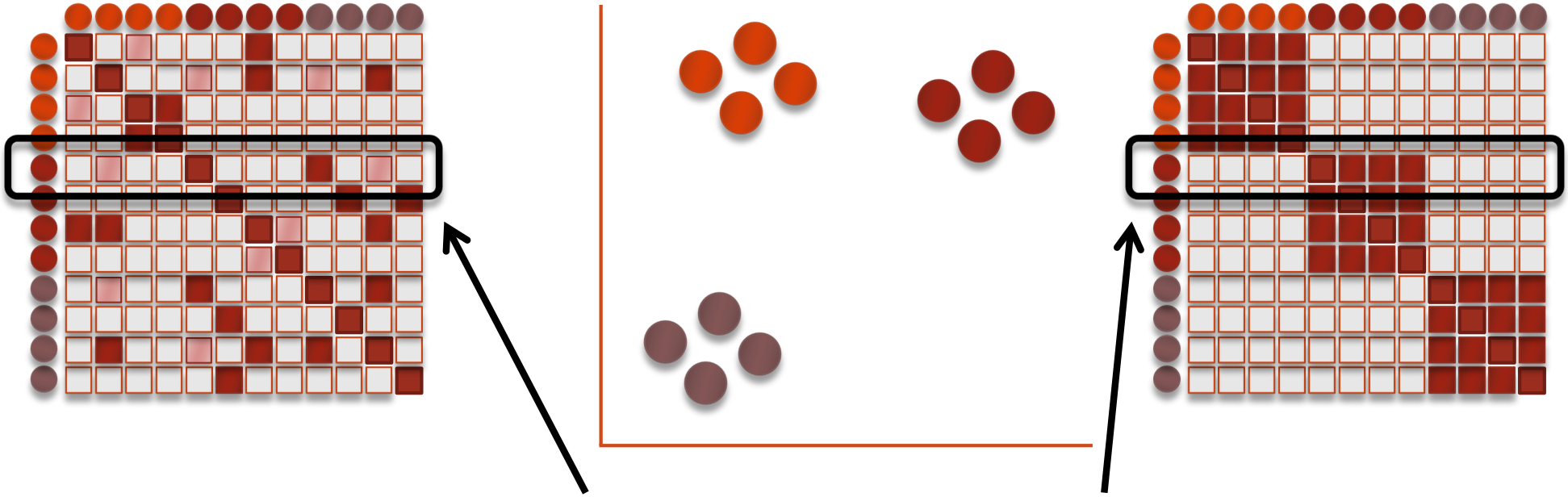
tSNE

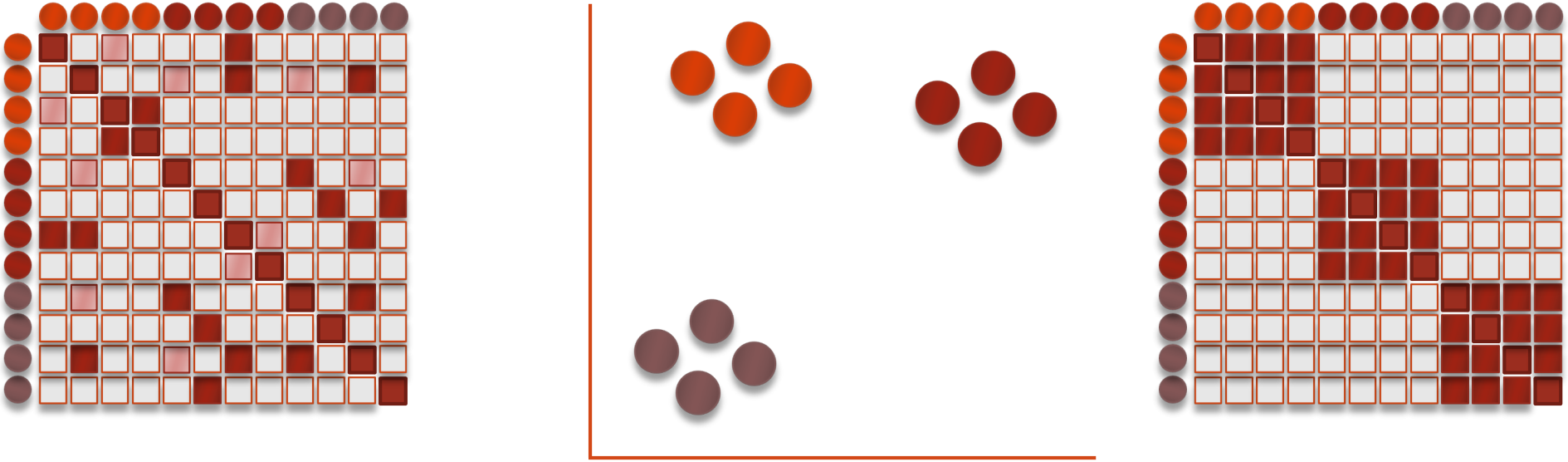
- Step1: calculate similarities between all point
- Step2: randomly project the data onto the desired number of axis
- Step3: calculate similarities between all point in the latent space



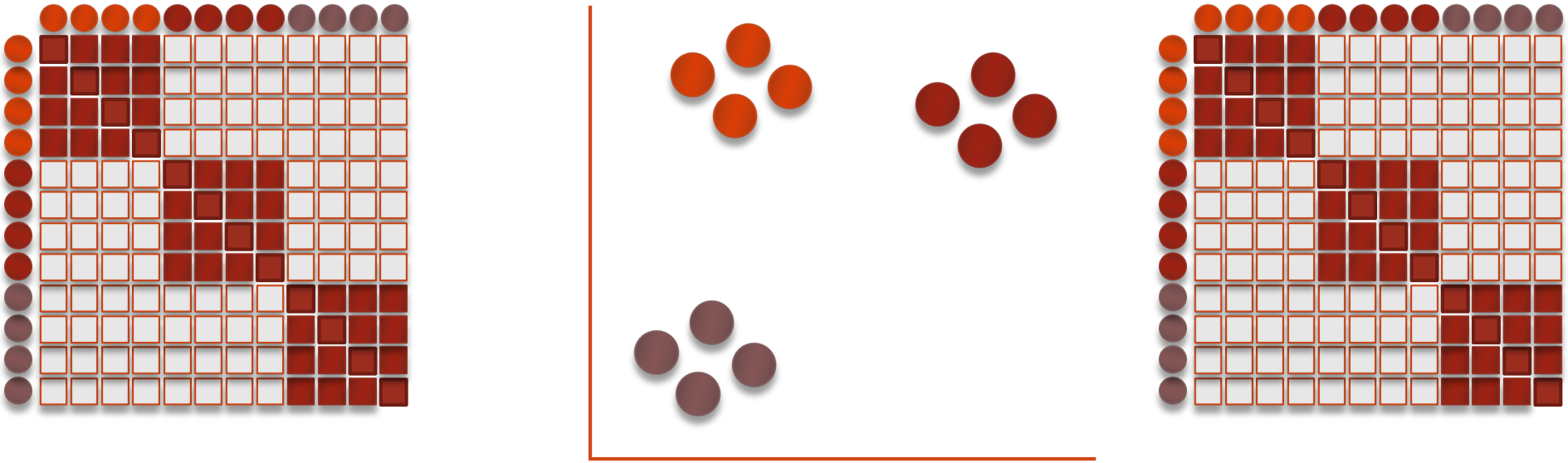
tSNE

- Step1: calculate similarities between all point
- Step2: randomly project the data onto the desired number of axis
- Step3: calculate similarities between all point in the latent space
- Step4: Move step by step the points in the latent space so that the similarity matrix in this space looks like the one in the original space

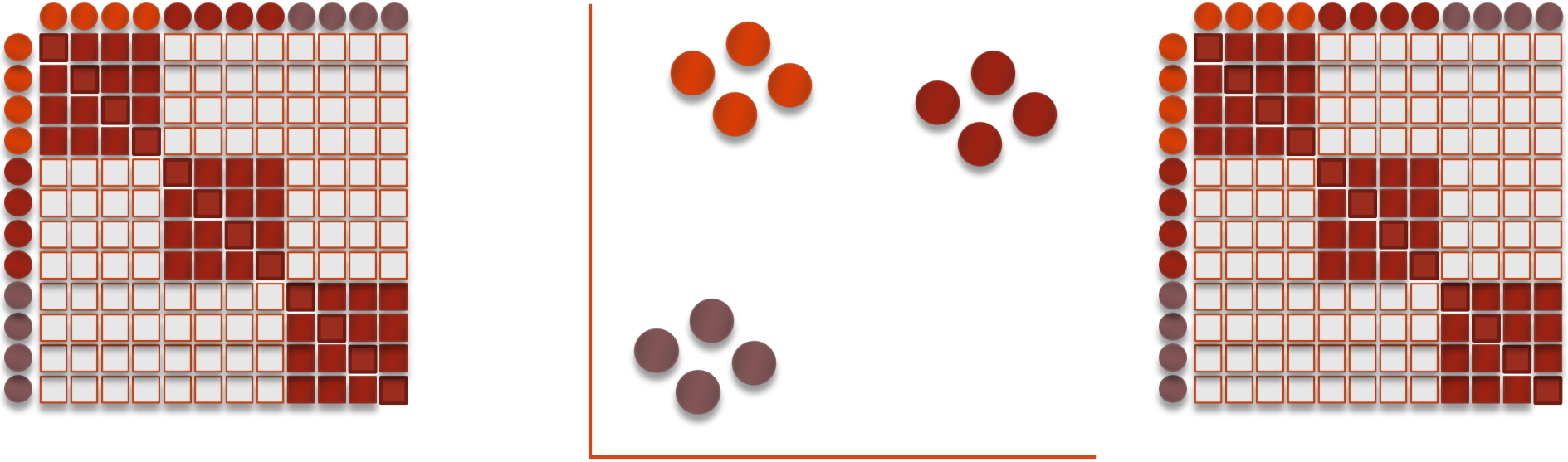




t-SNE moves the points a little bit at a time, and each step it chooses a direction that makes the matrix on the left more like the matrix on the right.



t-SNE moves the points a little bit at a time, and each step it chooses a direction that makes the matrix on the left more like the matrix on the right.



t-SNE moves the points a little bit at a time, and each step it chooses a direction that makes the matrix on the left more like the matrix on the right.

It uses small steps, because it's a little bit like a chess game and can't be solved all at once. Instead, it goes one move at at time.

tSNE projection

- Normally 2D, but can be any number of dimensions

- Axis don't mean anythink (unlike PCA)
- Distance don't mean anything (unlike PCA)

- Close proximity is highly informative
- Distant proximity isn't very interesting

- tSNE in action: https://www.youtube.com/watch?v=NEaUSP4YerM&t=185s

PCA + tSNE

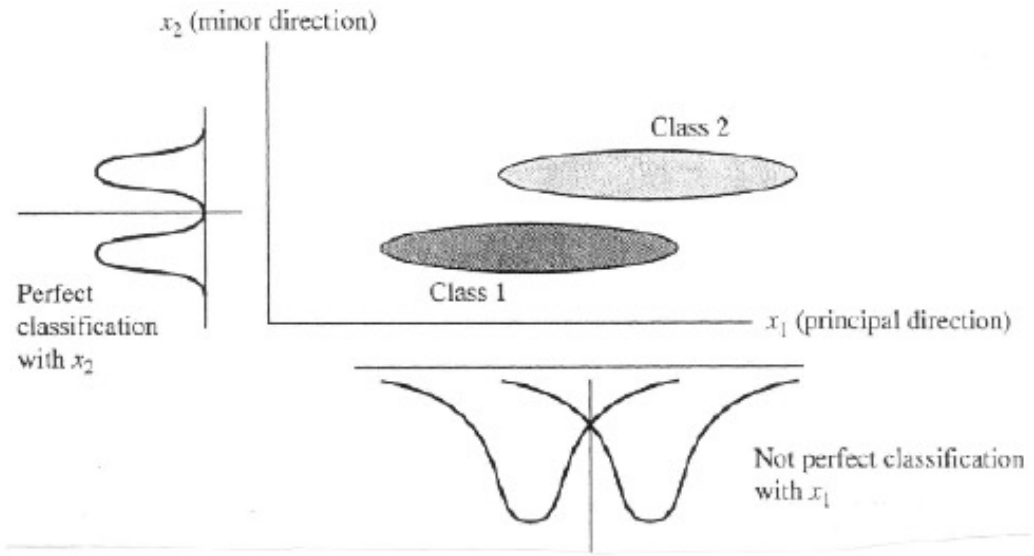
- PCA
  - Extracts the signal from the noise
  - Extracts informative dimensions → reduce dimensionality (but not to 2)

- tSNR
  - Works well with non-linear features → scale distances from PCA projections
  - Excellent for 2D projection → scale distances and project int 2-dimensions

LDA

Limitations (cont'd)

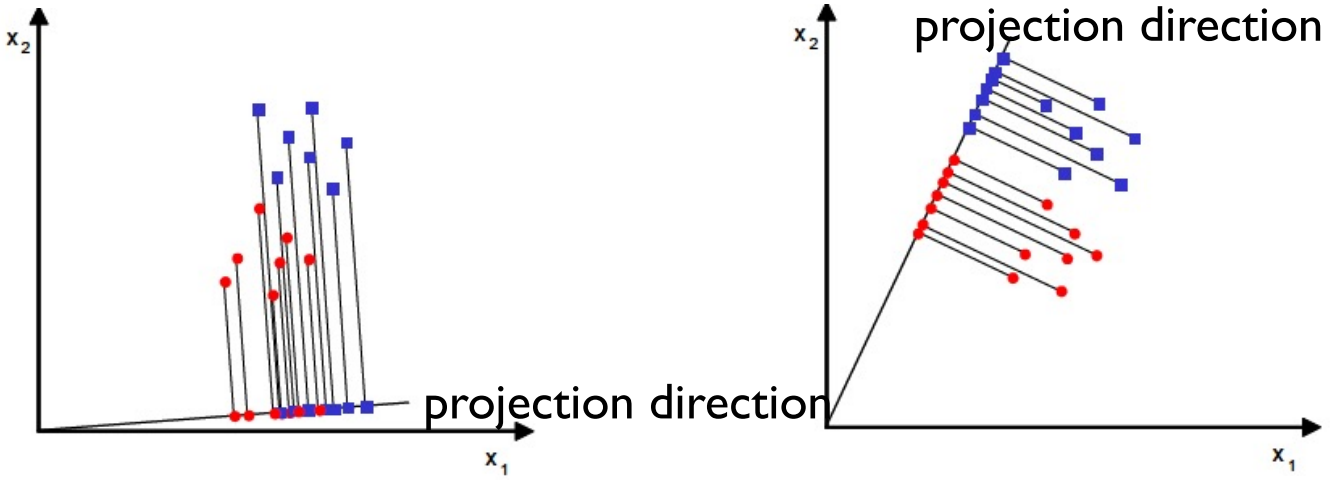
- PCA is not always an optimal dimensionality-reduction technique for classification purposes.



42

Linear Discriminant Analysis (LDA)

- What is the goal of LDA?
  - Seeks to find directions along which the classes are best separated (i.e., increase discriminatory information).
  - It takes into consideration the scatter (i.e., variance) within-classes and between-classes.
  - This is not an unsupervised projection



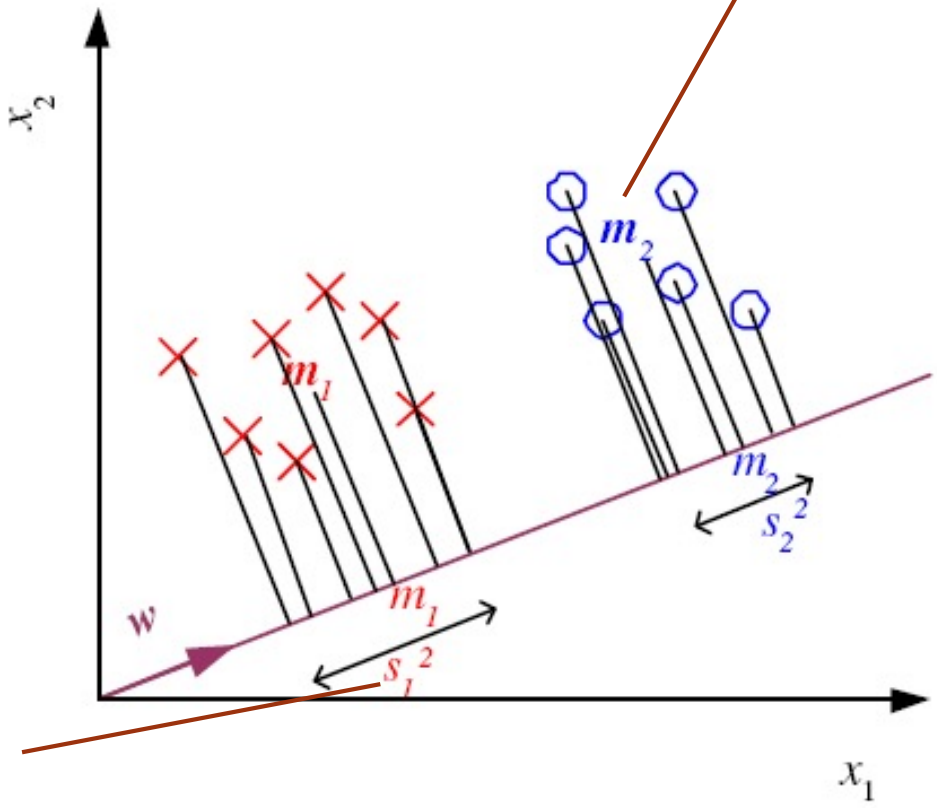
Bad separability

Good separability

43

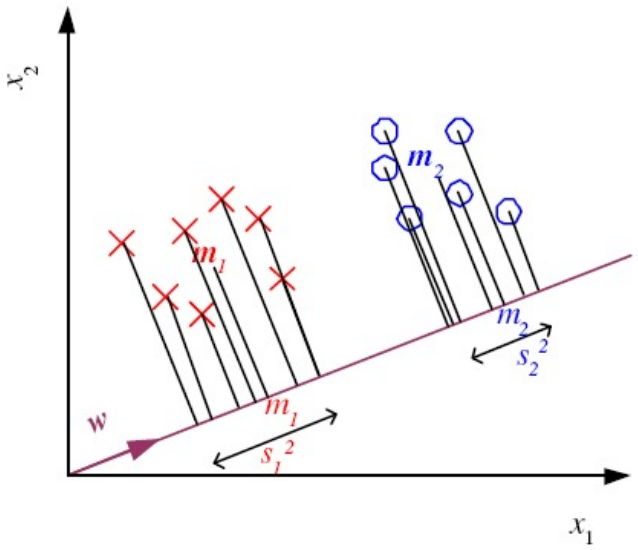
Linear Discriminant Analysis

- Find a low-dimensional space such that when $\boldsymbol{x}$ is projected, classes are well-separated



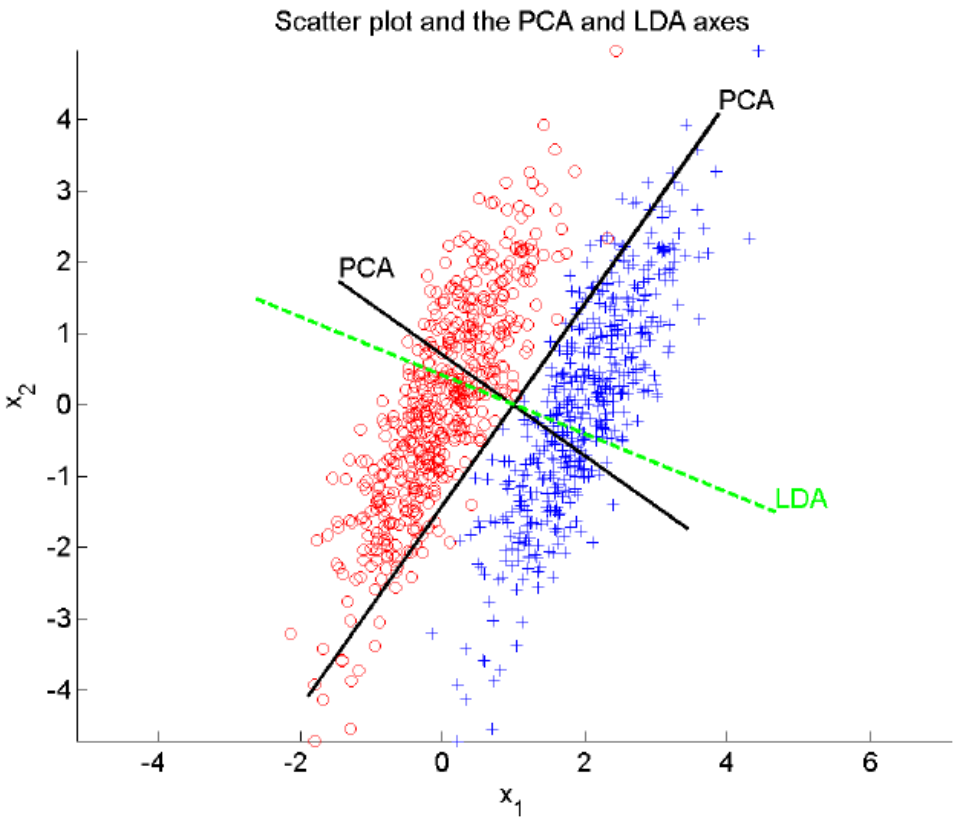
Good Projection

- Means are far away as possible
- Scatter is small as possible

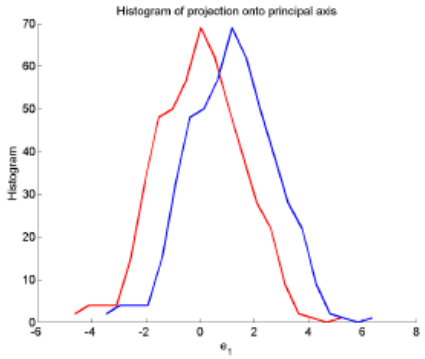


$$J(\mathbf{w}) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2}$$

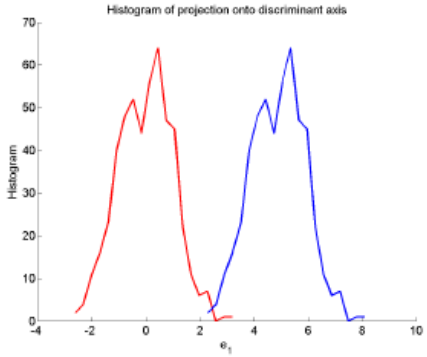
Example



(a) Scatter plot.



(b) Projection onto the first PCA axis.



(c) Projection onto the first LDA axis.

49

Linear Discriminant Analysis (LDA) (cont'd)

- For plotting, PCA could be applied first:

1) First, apply PCA to reduce data dimensionality:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ x_N \end{bmatrix} \xrightarrow{PCA} \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ y_M \end{bmatrix}$$

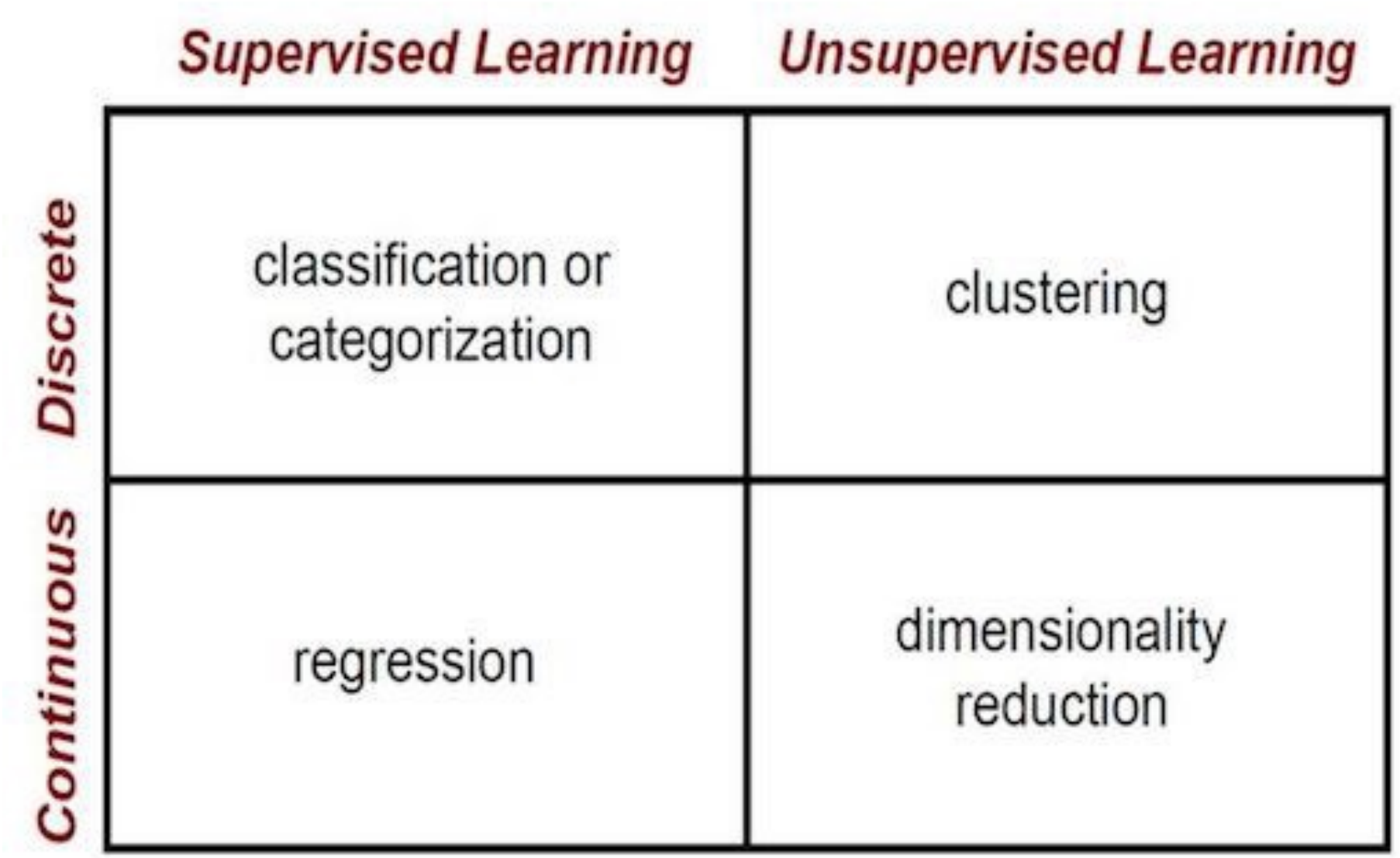
2) Then, apply LDA to find the most discriminative directions:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ y_M \end{bmatrix} \xrightarrow{LDA} \mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ . \\ . \\ z_K \end{bmatrix}$$

51

UMAP

Summary


Supervised Learning
Unsupervised Learning
Discrete
Continuous
classification or categorization
clustering
regression
dimensionality reduction