Module 1 :

Machine Learning Review

Unsupervised Learning Algorithms

Géraldine Conti, August 2020

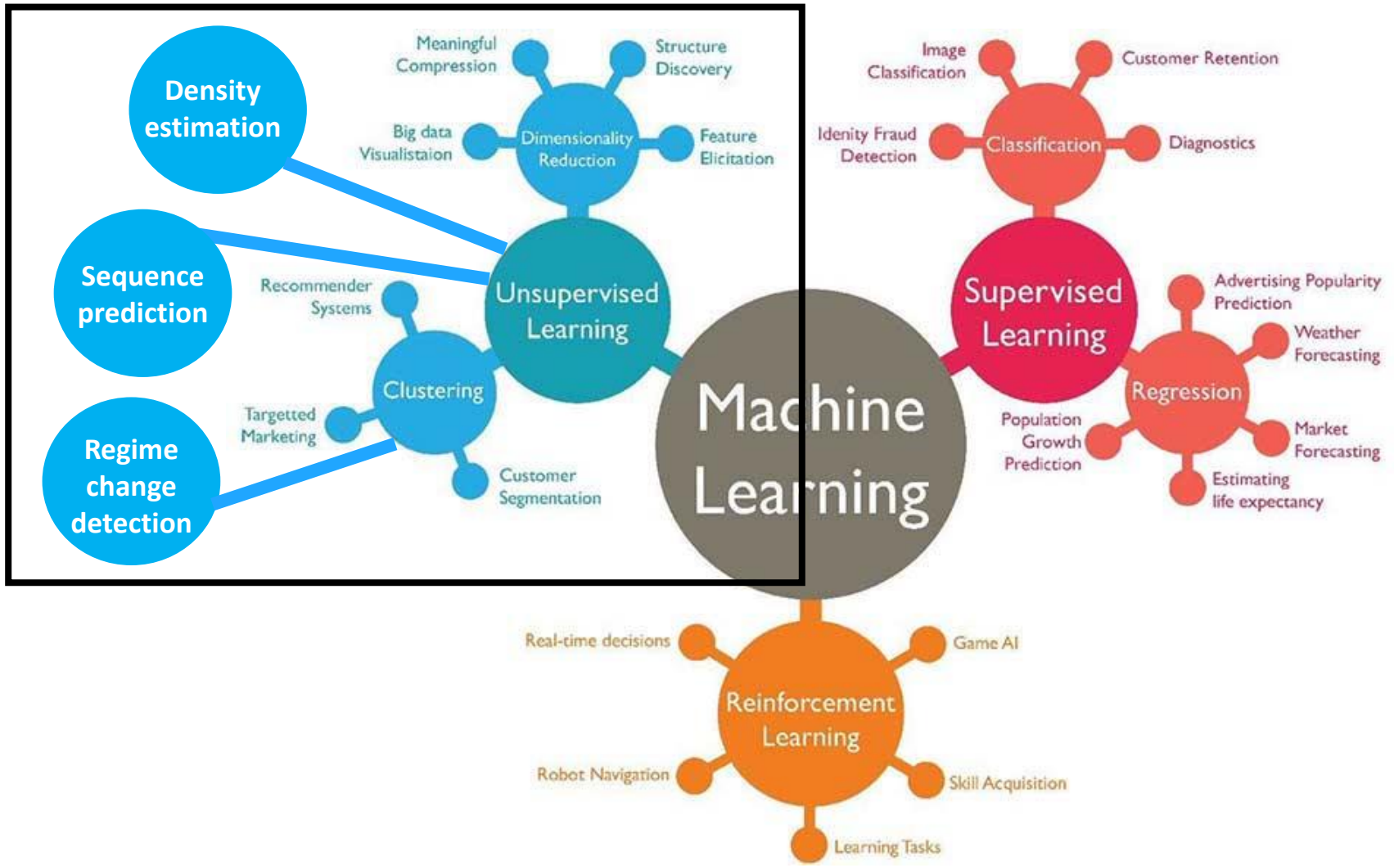
Discussion Session

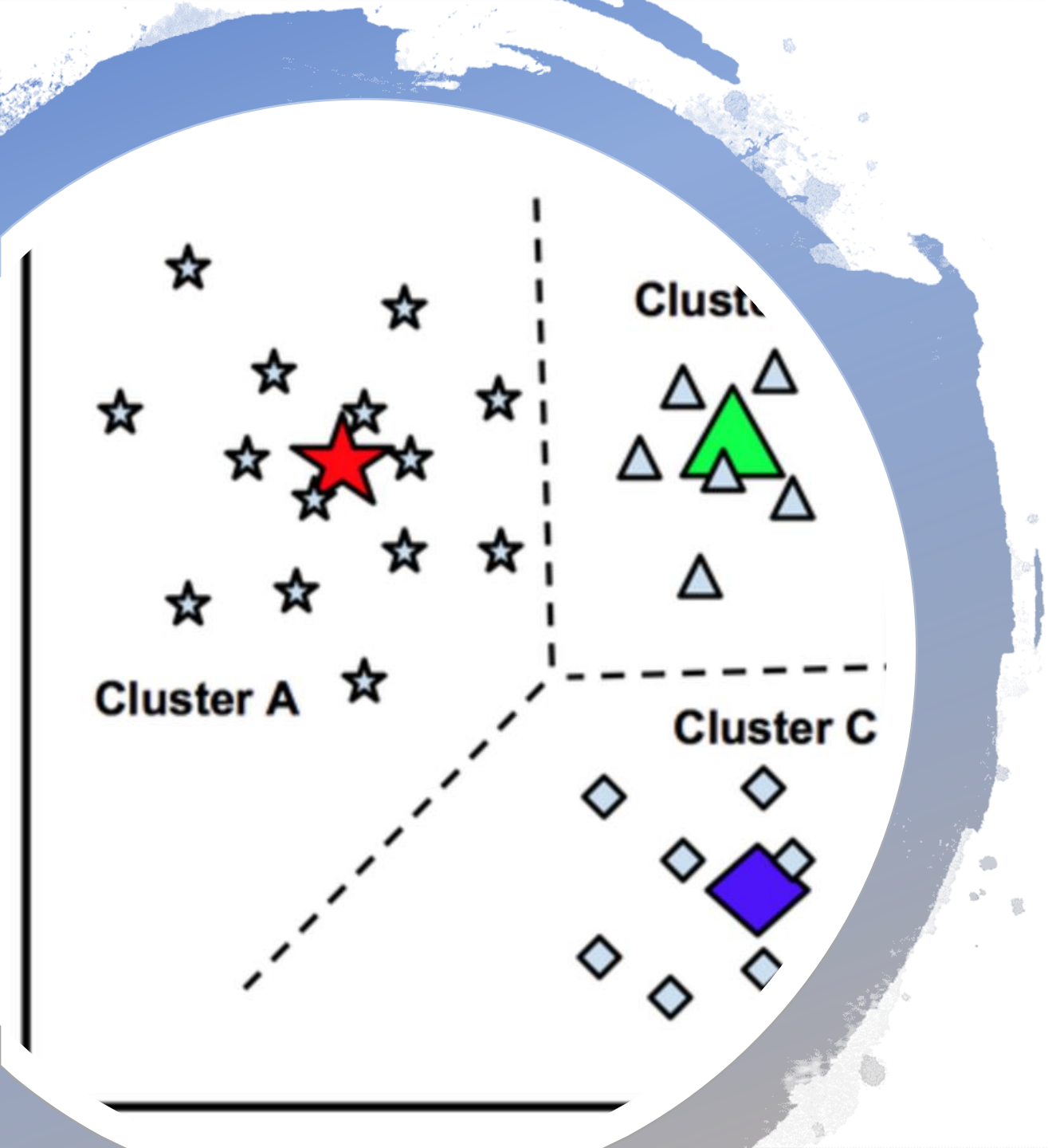
- Review of Notebook 2 (regression)
  - Pipeline for processing numerical attributes
  - Train model
  - linear regression, decision tree, random forest and SVR
  - Compare the models (MSE, MAE, cross-validation)
  - Fine-tune model (grid search, random search
  - Assess the performance on the test set (RMSE, 95% CL)

Bibliography

- Deep Learning book (Goodfellow, Bengio, Courville)
- Machine Learning @ Stanford (Prof Andrew Ng)
- Hands-On Machine Learning with Scikit-Learn & Tensorflow (Aurélien Géron)

Learning Objectives



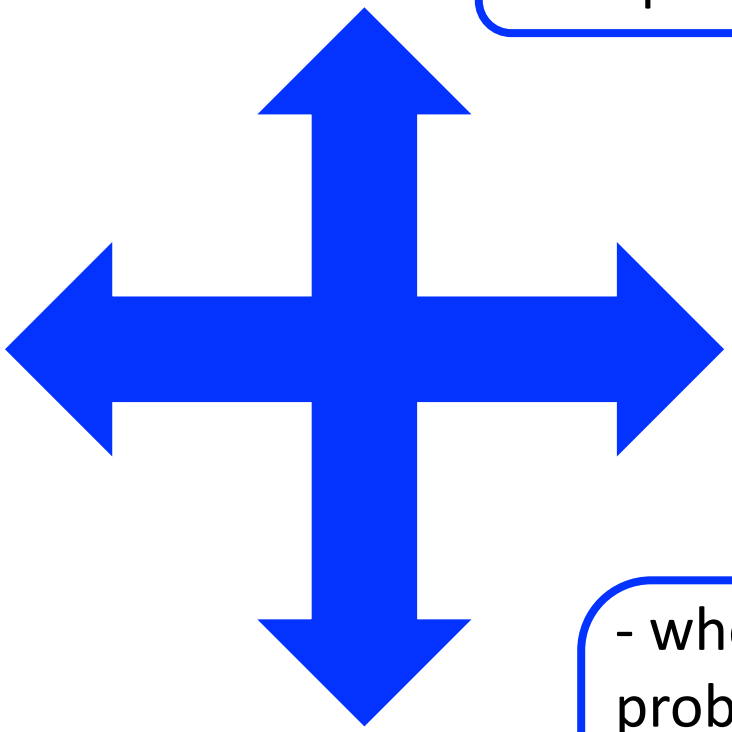


Clustering

- K-means

- Hierarchical clustering

Types of clustering



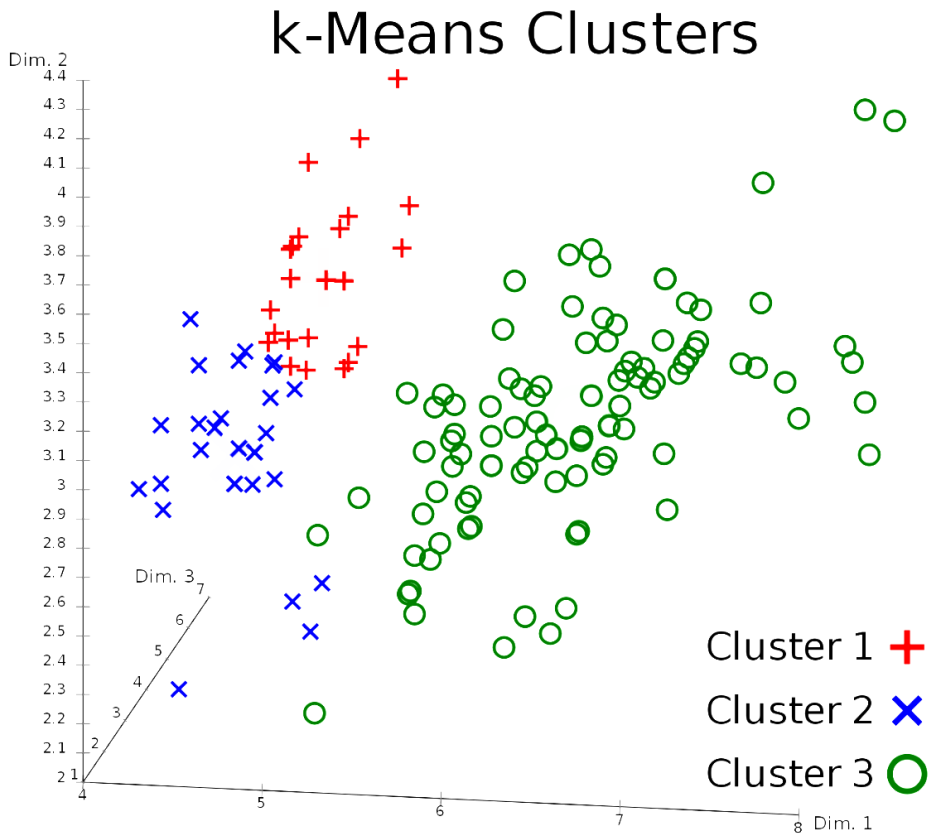
when efficiency is important

- when potential problem of flat clustering is a concern
- believed to produce better clusters

Where would you place K-means ?

K-Means : Introduction

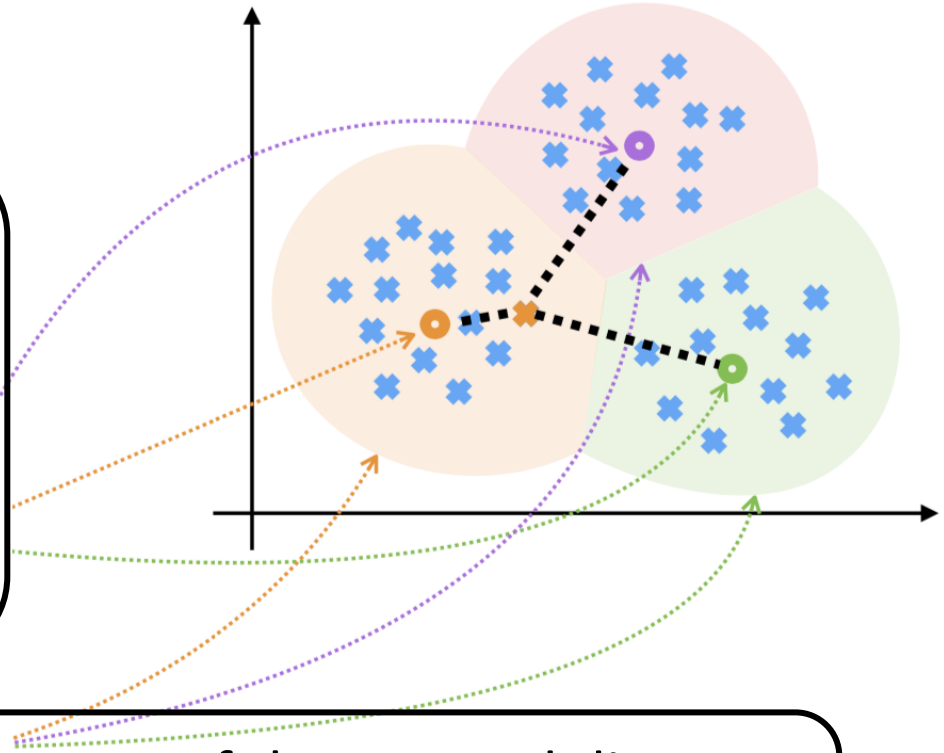
- Partition the dataset into K pre-defined non-overlapping clusters
  - "K" : *the number of clusters*
  - "means" : data is averaged (ie. finding centroids)
  - Each data point belongs to only one group



K-Means : Optimization

- Expectation-Maximization (EM) algorithm : general method for finding ML-estimates
  1) Initialization of the centroids
  2) Iteration :

2) Maximization step : Compute the centroids for the clusters by taking the average of all the data points belonging to each cluster



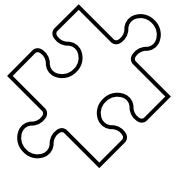
1) Expectation step : Compute the sum of the squared distance between the data points and centroids and assign each data point to the closest cluster

- Stop criteria :
  - Centroids have stabilized wrt some tolerance
  - Number of iterations has been achieved

K-Means : Performance

- Simple and flexible
- Time complexity O(n)
- Easy to interpret
- Space complexity O(n)

- No-optimal set of clusters (choose K)
- Assumptions (spherical, evenly sized clusters)
- Lack consistency with different initializations
- Sensitive to scale



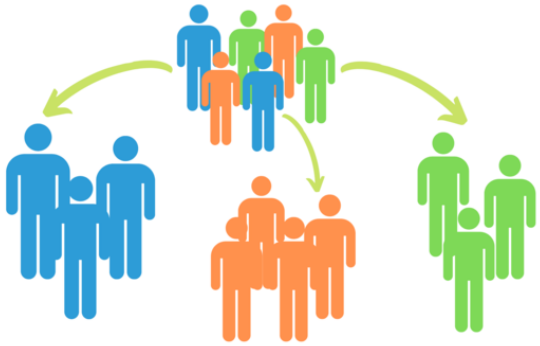
How to choose a good value for K ?

K-Means : Applications

Customer segmentation

Goal : you own a supermarket mall and you want to find who are your target customers

Data : membership card data (ID, age, gender, annual income, spending score)



Document clustering

Goal : you want to analyze Facebook/Twitter/Youtube comments of a particular event

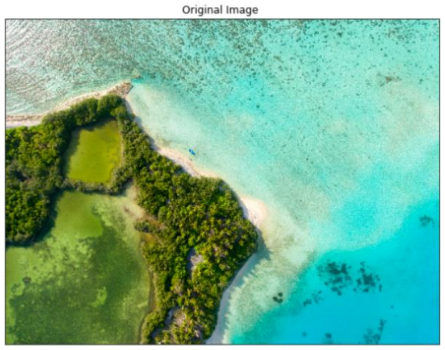
Data : social media text data

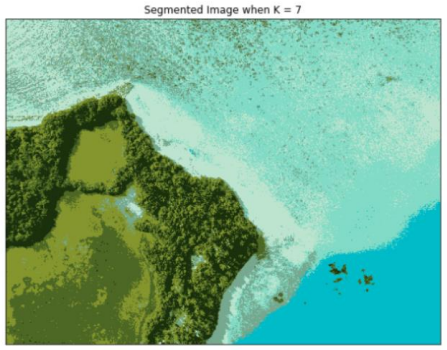
Documents	Online	Festival	Book	Flight	Delhi
D1	1	0	1	0	1
D2	2	1	2	1	1
D3	0	0	1	1	1
D4	1	2	0	2	0
D5	3	1	0	0	0
D6	0	1	1	1	2
D7	2	0	1	2	1
D8	1	1	0	1	0
D9	1	0	2	0	0
D10	0	1	1	1	1

Image compression

Goal : to compress a digital image to reduce its cost of storage or transmission

Data : colored image (pixels)





K-Means : In practice

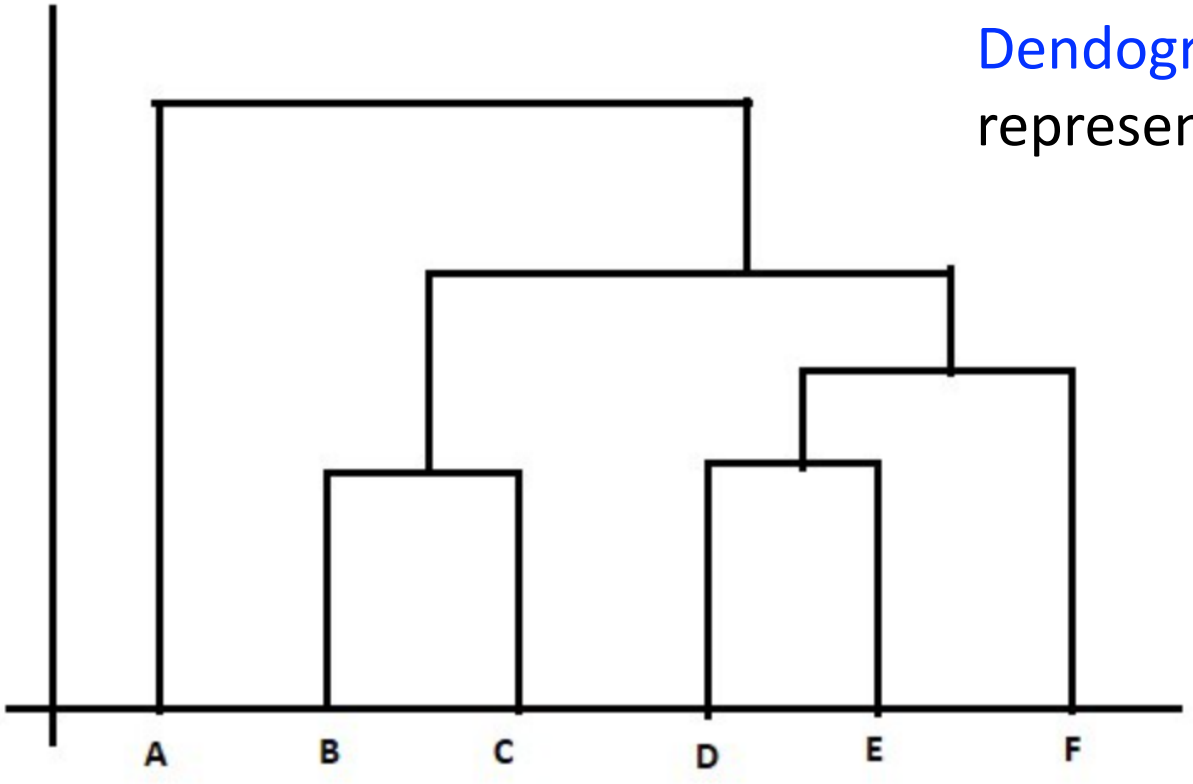
- sklearn.cluster.KMeans

```
>>> from sklearn.cluster import KMeans
>>> import numpy as np
>>> X = np.array([[1, 2], [1, 4], [1, 0],
...               [10, 2], [10, 4], [10, 0]])
>>> kmeans = KMeans(n_clusters=2, random_state=0).fit(X)
>>> kmeans.labels_
array([1, 1, 1, 0, 0, 0], dtype=int32)
>>> kmeans.predict([[0, 0], [12, 3]])
array([1, 0], dtype=int32)
>>> kmeans.cluster_centers_
array([[10.,  2.],
       [ 1.,  2.]])
```

- If large dataset (>10k samples) :
  - sklearn.cluster.MiniBatchKMeans

Hierarchical Clustering : Introduction

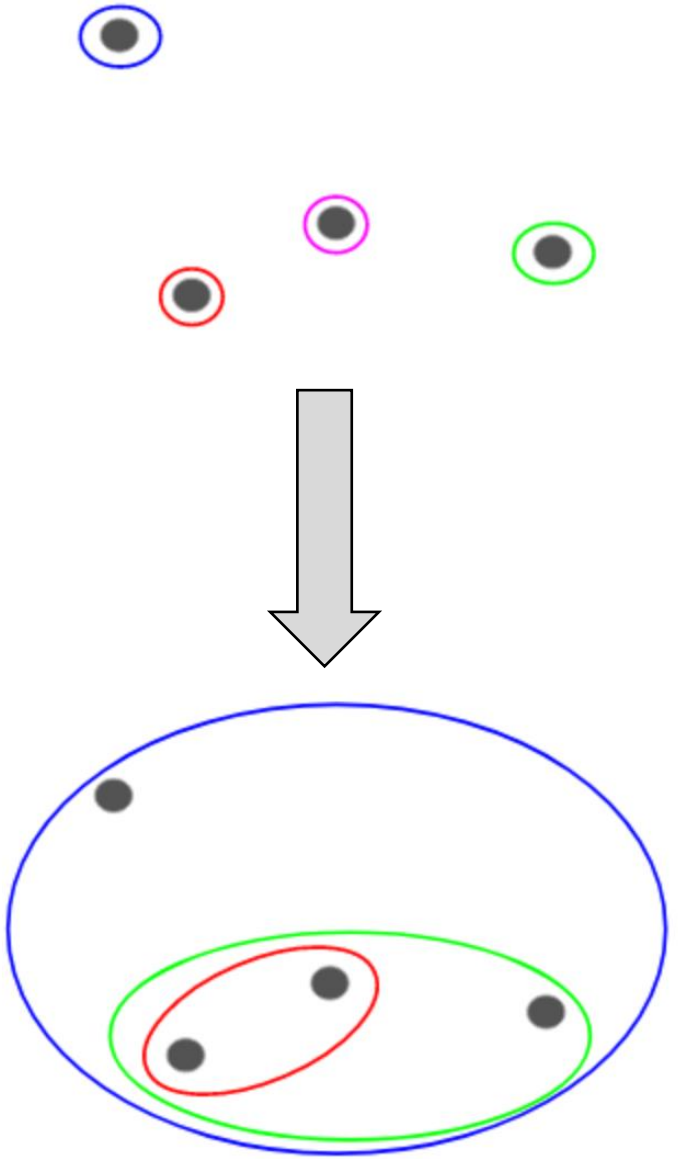
- Build a hierarchy of clusters
- Two types :
  - Agglomerative
  - Divisive (not much used)



Dendogram representation

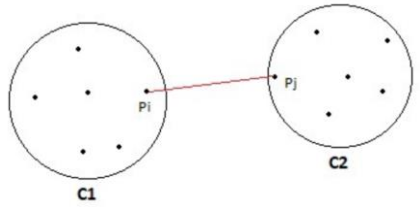
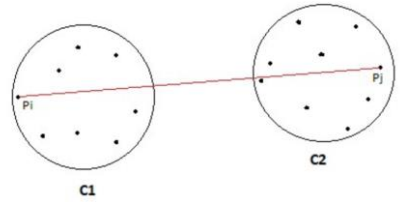
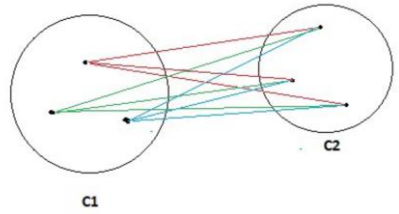
Hierarchical Clustering : Optimization

- Agglomerative algorithm: each data point is considered as an individual cluster
  1) Calculate the proximity of individual points
  2) Iteration :
     - Merge the two closest clusters
     - Update the proximity matrix
- Stop criterium :
  - a single cluster remains
- Divisive algorithm: the opposite



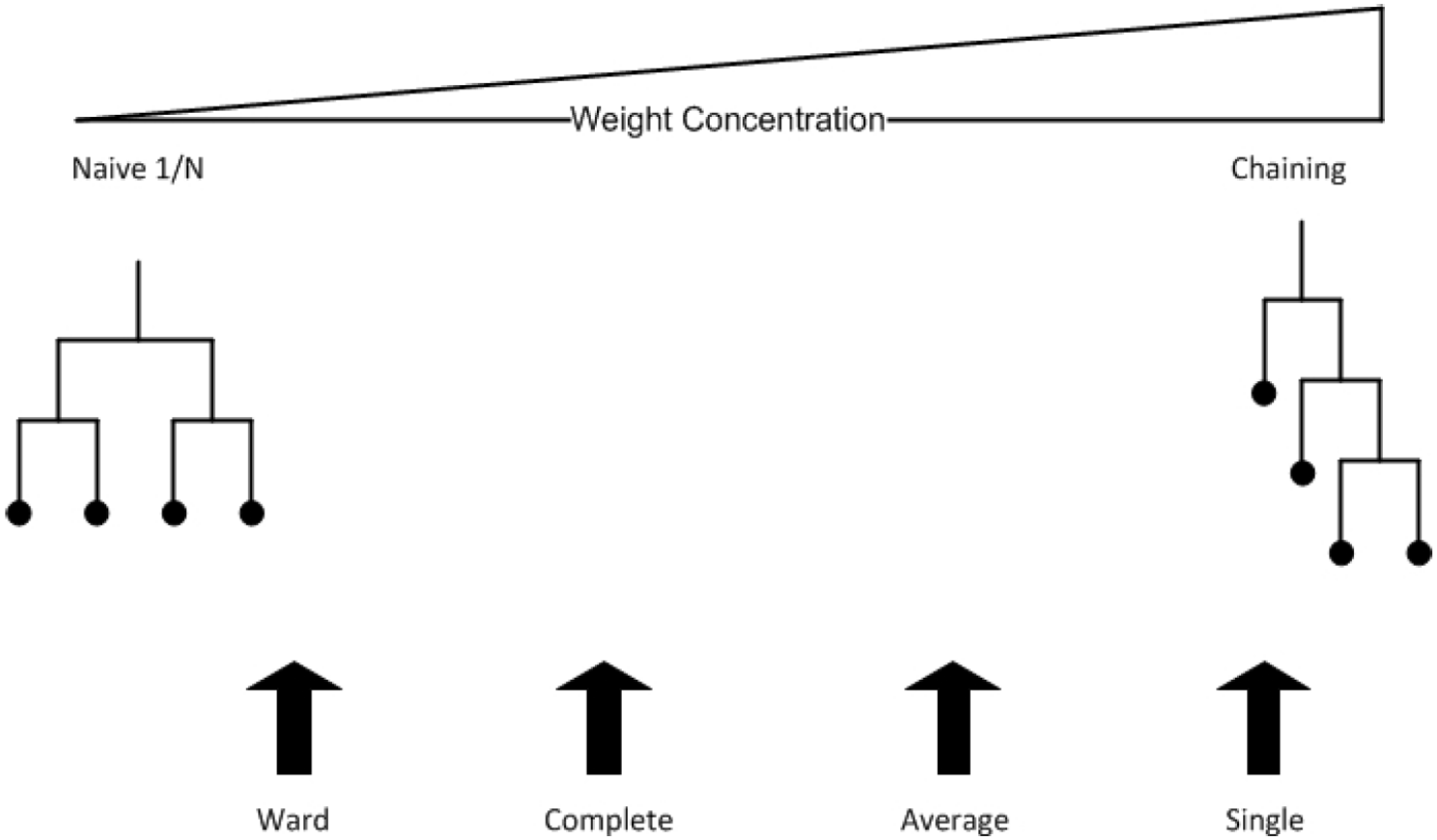
How can the proximity be calculated ?

Hierarchical Clustering : Proximity (Linkage)

	Single linkage algorithm (MIN)	Complete linkage algorithm (MAX)	Group Average *
Proximity			
Pros	separate non-elliptical shapes if gap between two clusters is not small	Separates well clusters even if there is noise between clusters	Separates well clusters even if there is noise between clusters
Cons	cannot separate clusters properly if there is noise between clusters	Bias towards globular clusters Tends to break large clusters	Bias towards globular clusters

* Ward's method is similar, but it calculates the sum of the squares of the distances

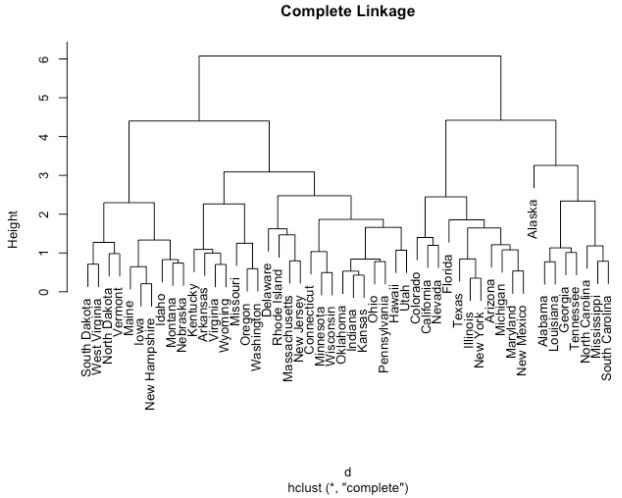
Proximity Comparison

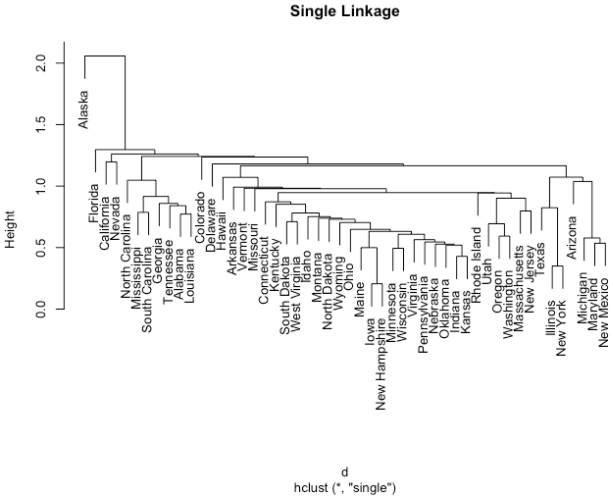


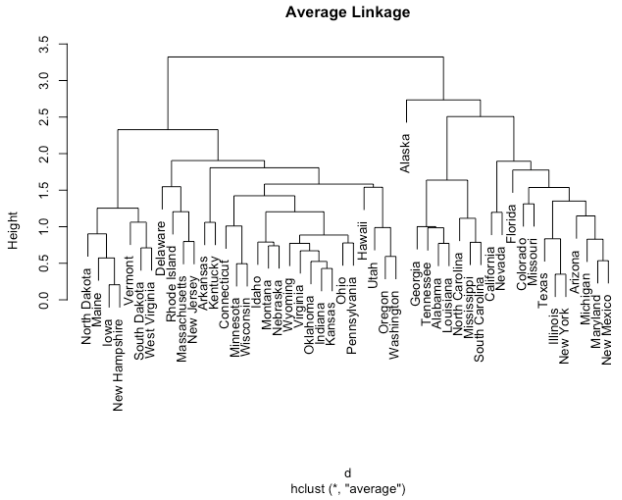
Hierarchical Clustering : Performance

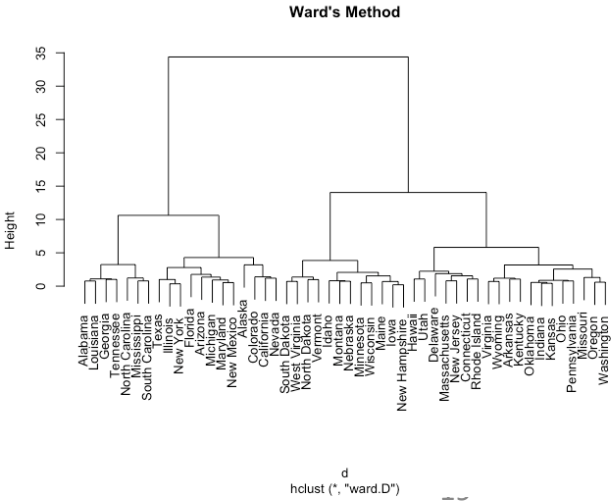
- Deterministic

- Choice of similarity
- Space complexity $O(n^2)$
- Time complexity $O(n^3)$







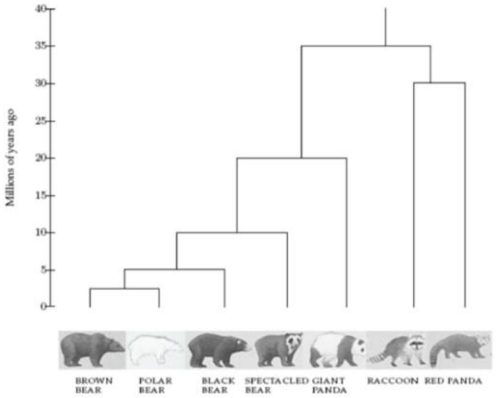


Hierarchical Clustering: Applications

Animal evolution

Goal : find the phylogenetic tree relating different species together

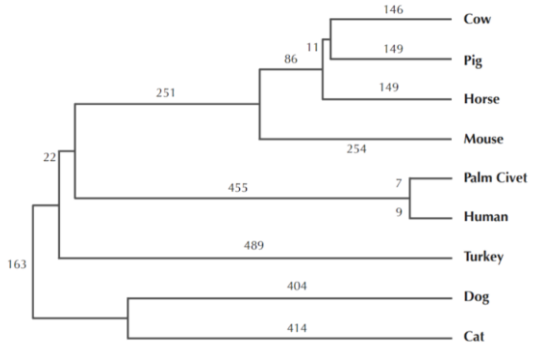
Data : DNA sequences



Tracking viruses

Goal : track viral outbreaks and their sources (SARS virus example)

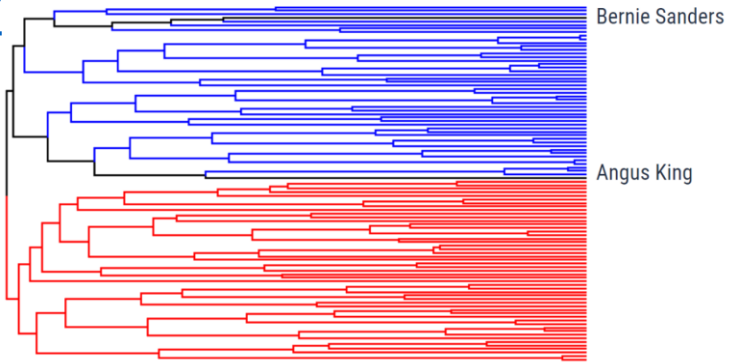
Data : DNA sequences



Find party lines through Twitter

Goal : Cluster US senators into their respective parties

Data : Twitter data



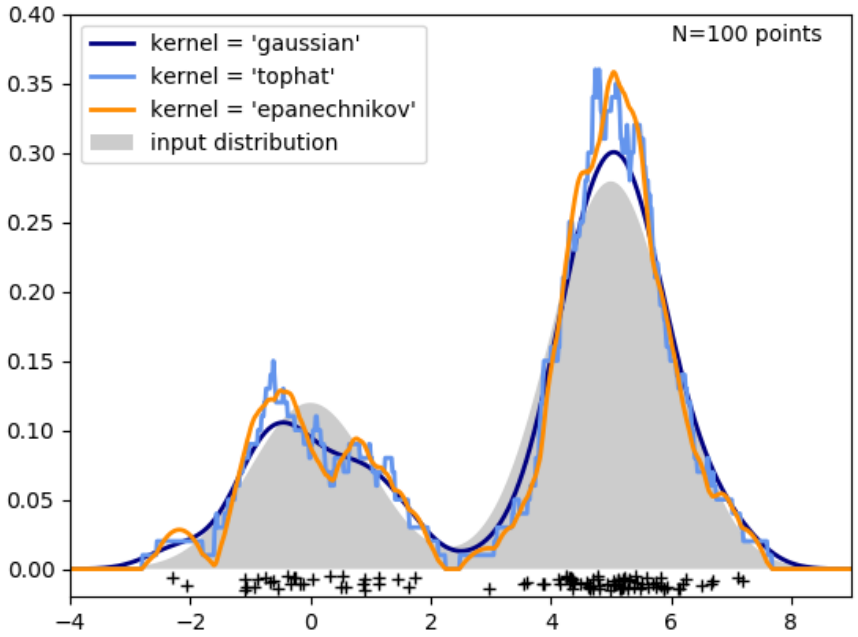
Hierarchical Clustering : In Practice

- sklearn.cluster.AgglomerativeClustering

```
#----------------------------------------------------------------------
# 2D embedding of the digits dataset
print("Computing embedding")
X_red = manifold.SpectralEmbedding(n_components=2).fit_transform(X)
print("Done.")

from sklearn.cluster import AgglomerativeClustering

for linkage in ('ward', 'average', 'complete', 'single'):
    clustering = AgglomerativeClustering(linkage=linkage, n_clusters=10)
    t0 = time()
    clustering.fit(X_red)
    print("%s :\t%.2fs" % (linkage, time() - t0))
```

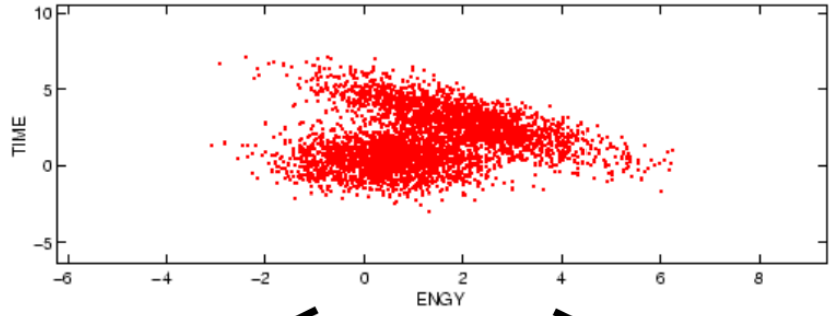


Density estimation

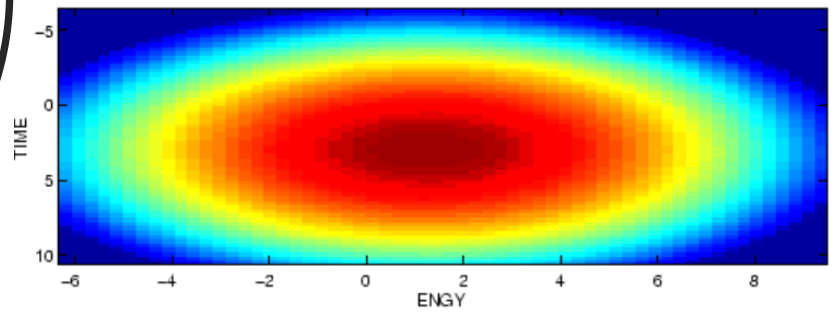
Observed data from a complex but unknown probability distribution

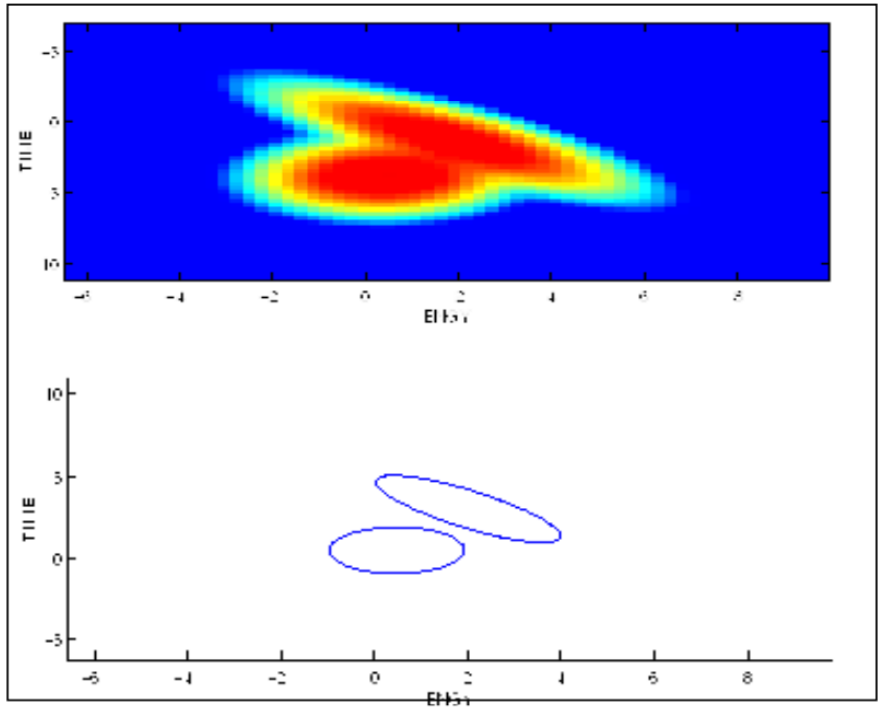
- Gaussian Mixture Model (GMM)

GMM: Introduction



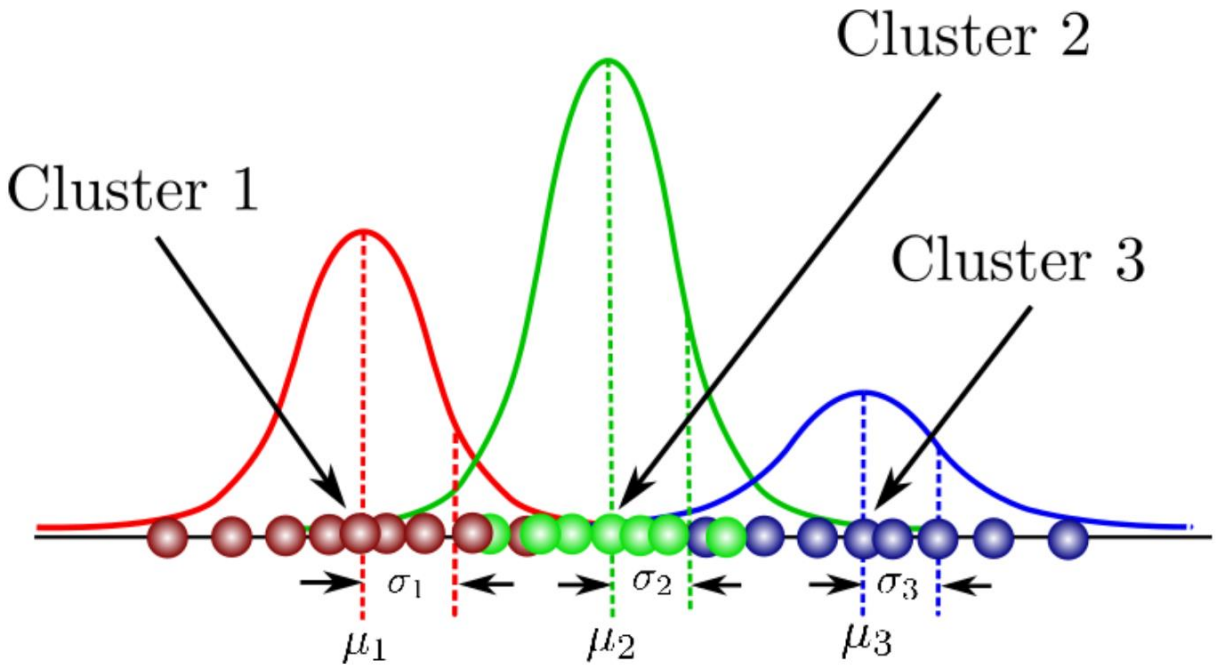
single Gaussian ? ? GMM





GMM: Definition

- Function of several Gaussians $k \in \{1,\ldots, K\}$, where K is the number of clusters of our dataset



- Parameters :
  - mean μ
  - covariance Σ that defines its width
  - mixing probability c that defines how big the Gaussian will be

GMM: Formalism

- Mixture models are linear combinations of densities :

$$p(x \mid \Theta) = \sum_{i=1}^{K} c_i \, p(x \mid \Theta_i)$$

$$\text{with} \quad \sum_{i=1}^{K} c_i = 1 \quad , \quad \int_x p(x \mid \Theta_i) dx = 1$$

- For Gaussian mixtures :

$$\Theta_i = \{\mu_i, \Sigma_i\}, \quad \Rightarrow \quad p(x \mid \Theta_i) \equiv N(\mu_i, \Sigma_i)$$

- To find the parameters, minimize the –log-likelihood :

$$L(\Theta) = \ln p(X; \Theta)$$

$$\Theta^* = \arg\max_{\Theta} L(\Theta)$$

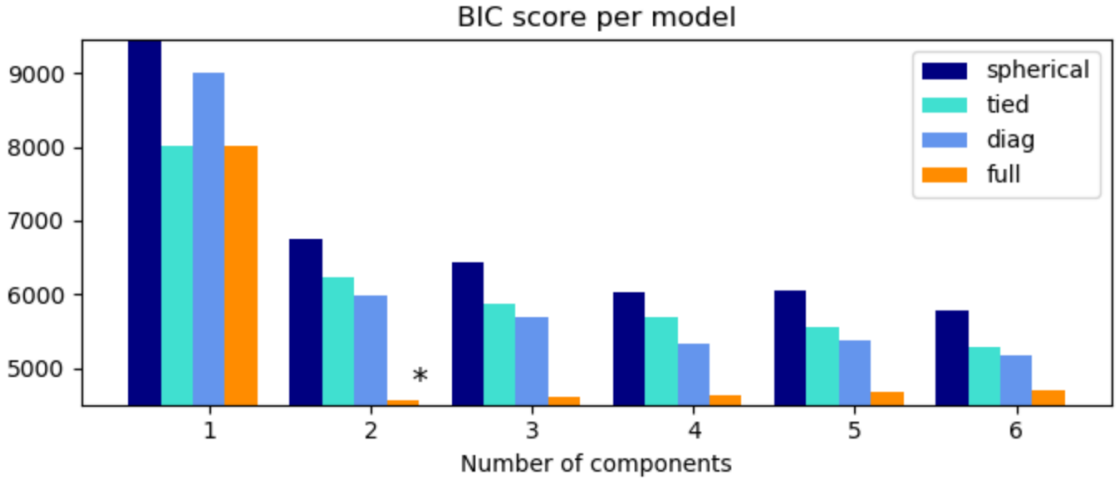
No analytical expression → EM-algorithm

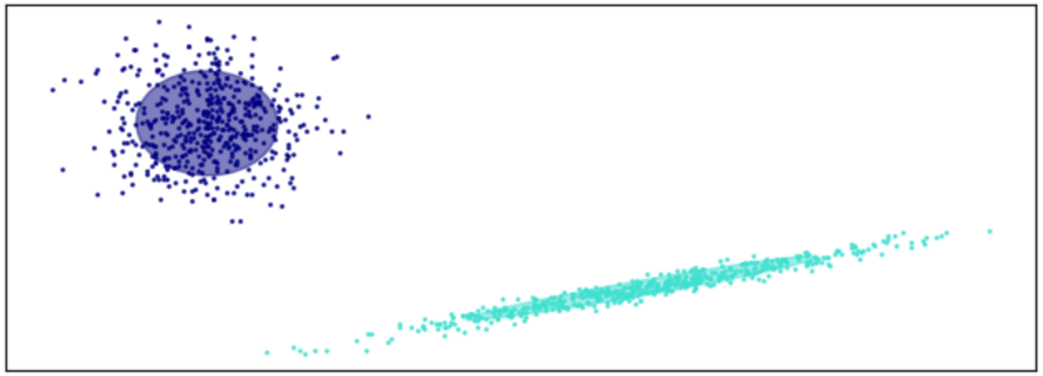
Gaussian Mixture : Performance

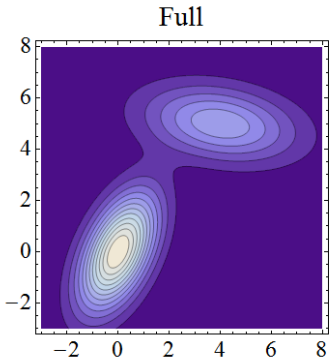
- Speed (fastest algorithm for mixture models)
- Agnostic (no biases of the means towards zero/of the cluster sizes to have specific structures that might (or not) apply)

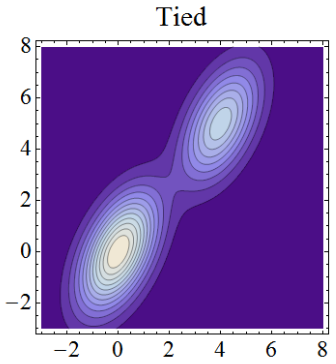
- Singularities
- Number of components

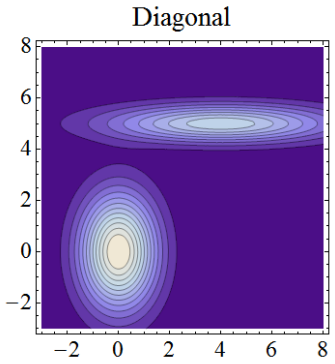
Bayesian Information Criterion (BIC)

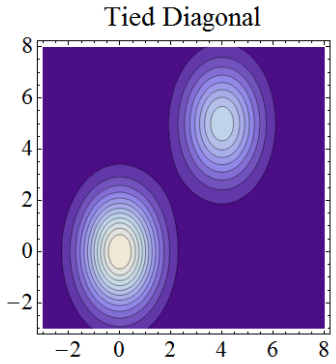


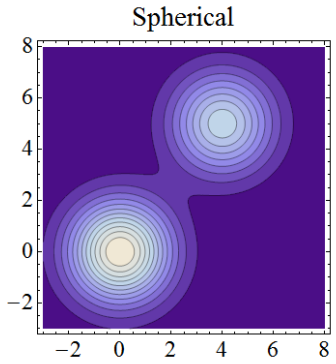










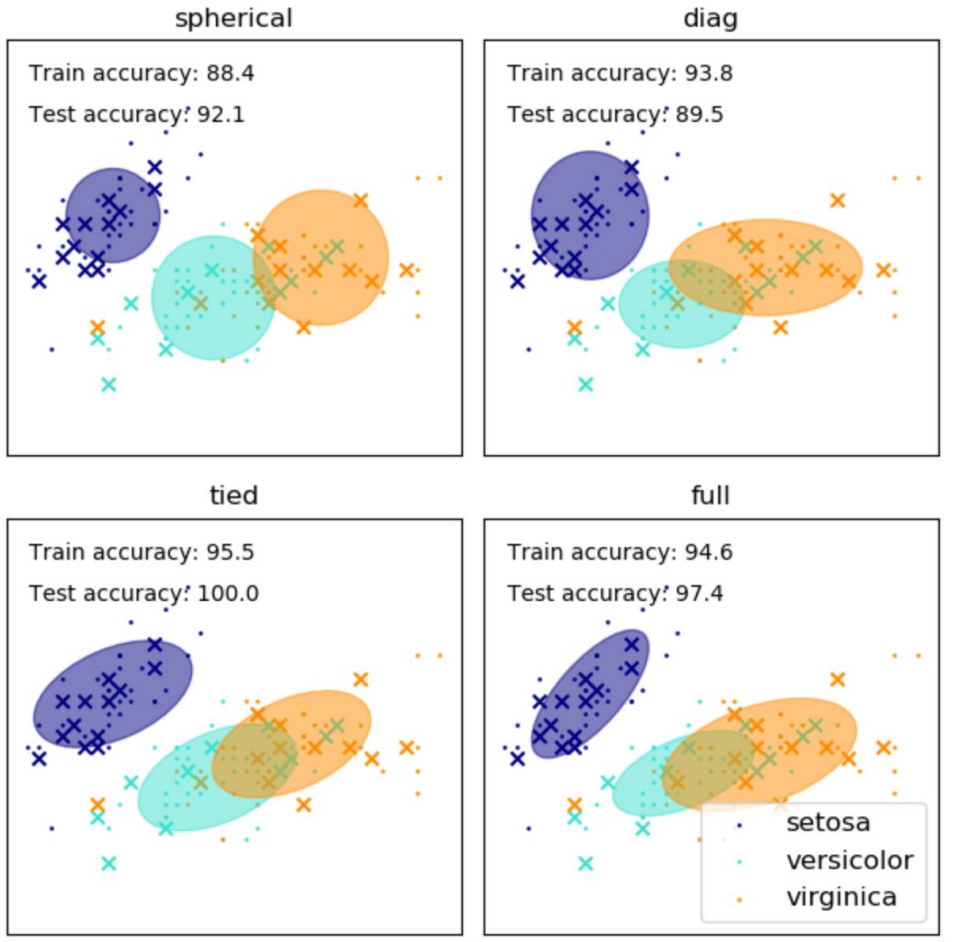


Gaussian Mixture : In practice

- sklearn.mixture.GaussianMixture

```
# Try GMMs using different types of covariances.
estimators = {cov_type: GaussianMixture(n_components=n_classes,
              covariance_type=cov_type, max_iter=20, random_state=0)
              for cov_type in ['spherical', 'diag', 'tied', 'full']}
```

"diagonal" situation with circular contours



contour axes are oriented along the coordinate axes

they have the same shape, but the shape may be anything

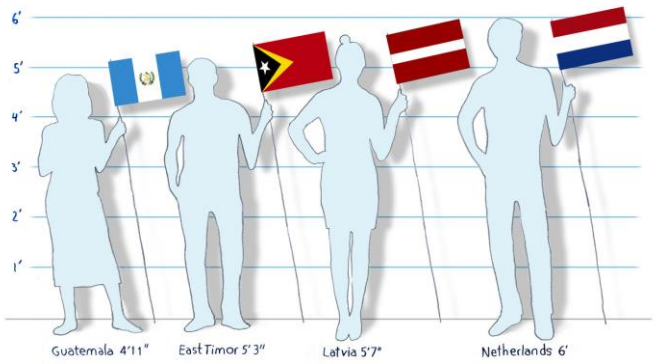
any position and shape

Gaussian Mixture: Applications

Demographic Statistics

Goal : compute the average height of people of different ethnicities (African-American, Caucasian, Asian and Latino)

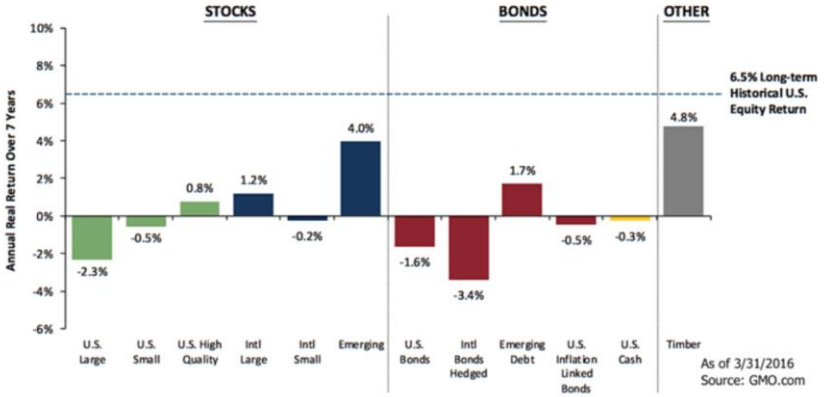
Data : People's heights



Finance

Goal : forecast asset returns

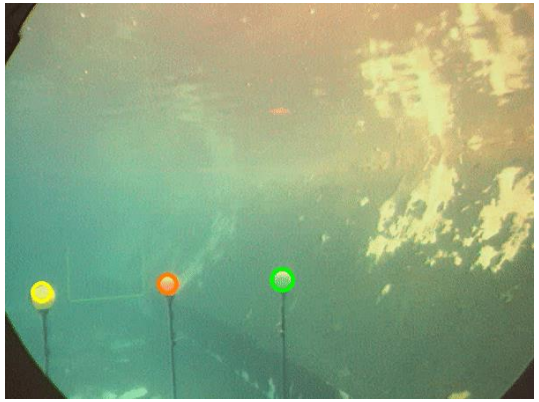
Data : timeseries of asset prices

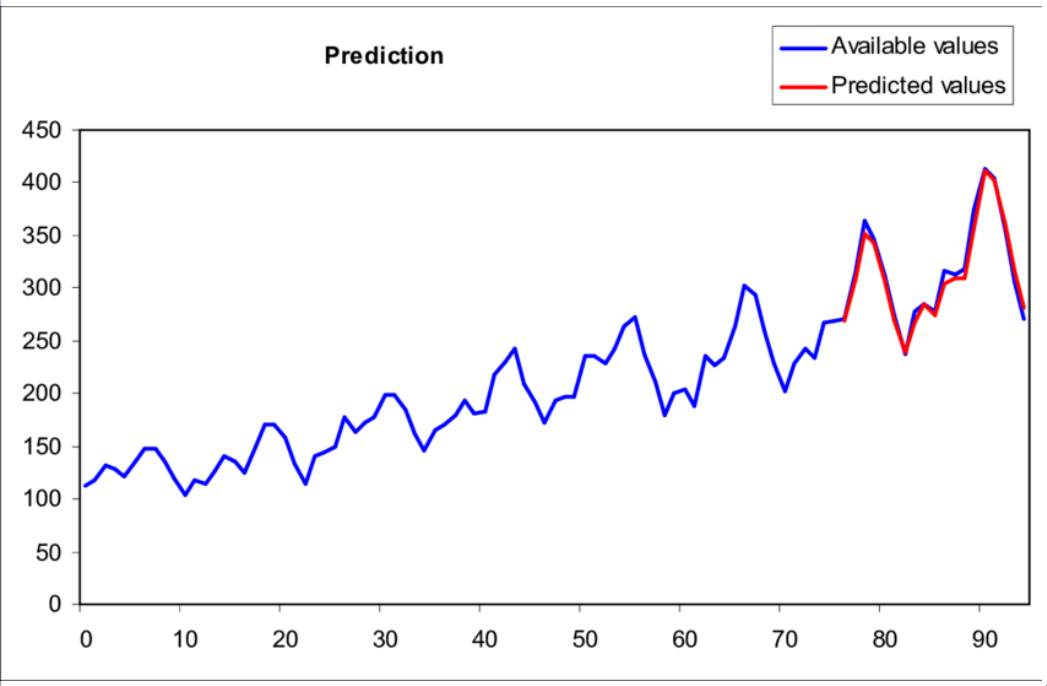


Detect objects in challenging environments

Goal : detect coloured buoys underwater

Data : video sequence





Sequence Prediction

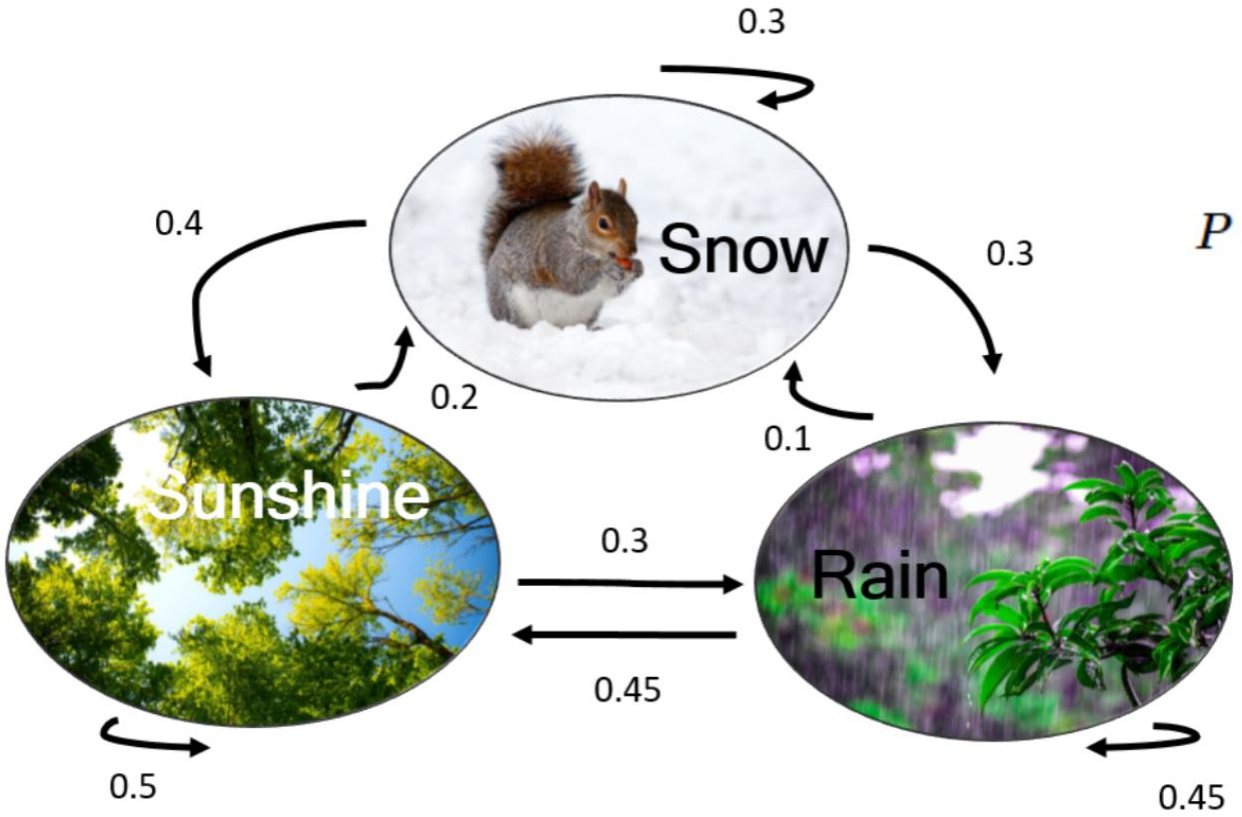
- HMM

- RNN (module 2)

- LSTM (module 2)

Hidden Markov Model : Introduction

- Markov Chain:
  - Type of random process, where the probability of being in state j depends *only* on the previous state
  - Often described by a graph with transition probabilities



$$P = \begin{bmatrix} 0.3 & 0.3 & 0.4 \\ 0.1 & 0.45 & 0.45 \\ 0.2 & 0.3 & 0.5 \end{bmatrix}$$

Hidden Markov Model (HMM)

- Invisible Markov chain that we cannot observe. Each state generates a random one out of k observations
- Number of states m to be decided

observations

states

$$p(x_1,..,x_m,s_1,...,s_m;\theta) = \pi(s_1)\prod_{j=2}^{m} t(s_j \mid s_{j-1})\prod_{j=1}^{m} e(x_j \mid s_j)$$

Initial probabilities π of the states

Transition probabilities t from a state to another

Emission probabilities e of an observation given a state

HMM Tasks

- 3 main tasks :

Task1 : Probability of an observation sequence (state estimation)

Calculate the probability that we feel cold for two consecutive days

Task 2 : Most likely explanation (inference/decoding)

- find the most probable hidden states from a series of observations → Viterbi algorithm

In a sequence of 14 days, we know if it was hot or cold. From it, we want to find the most likely weather forecast (snow, rain, sunshine) for these two weeks

Task 3 : Learning the HMM parameters → Baum-Welch algorithm

- Transition probability matrix
- Observation likelihood matrix

HMM : Applications

Word recognition

Goal : recognize a word by looking at the characters separately

Data : images of characters

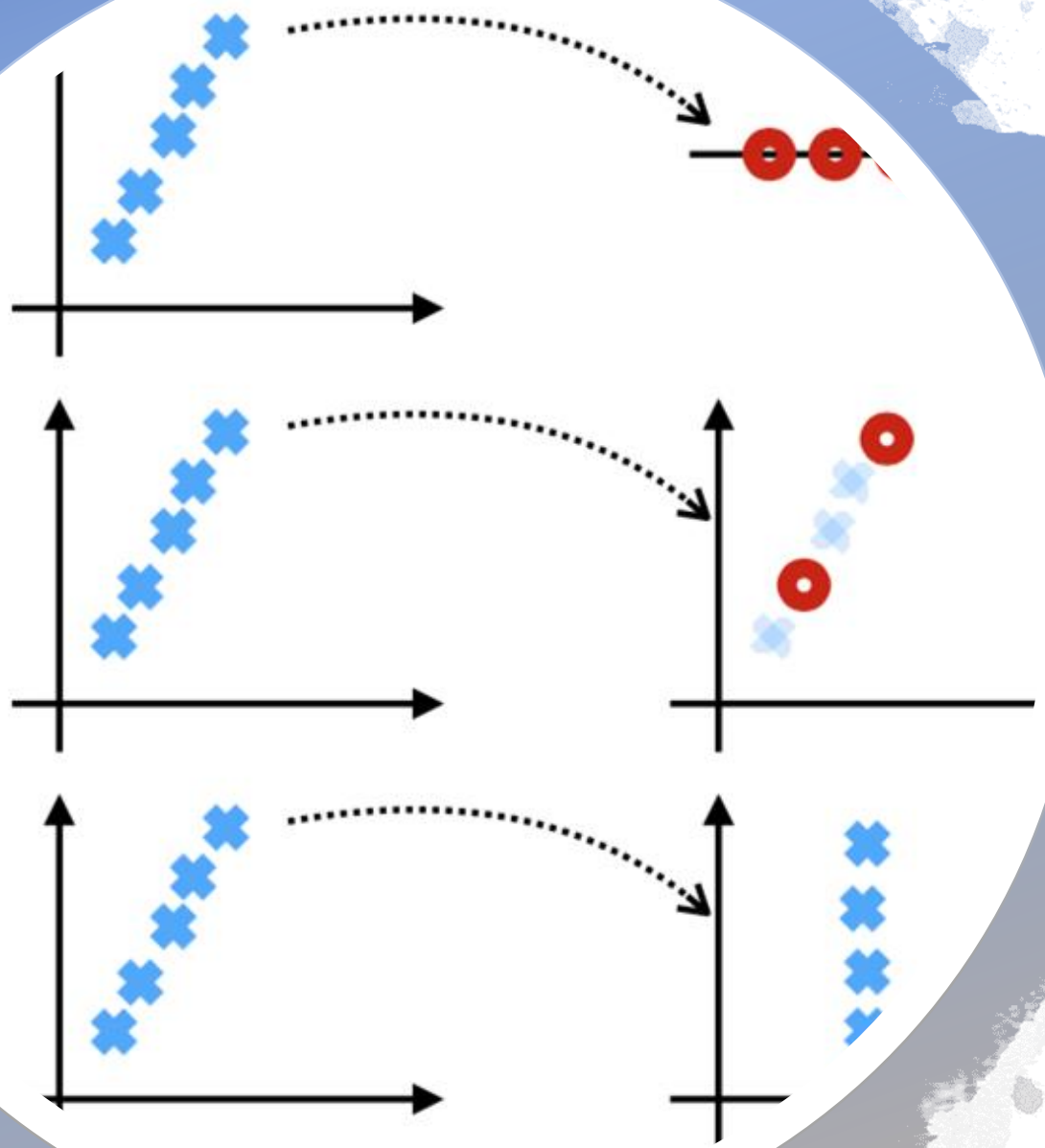
Stock trading

Goal : predict monthly closing prices of the S&P 500 index

Data : monthly prices

HMM: In practice

- No implementation in sklearn
- https://pystruct.github.io/
- https://larsmans.github.io/seqlearn/
  - seqlearn.hmm.MultinomialHMM
- https://hmmlearn.readthedocs.io/en/latest/index.html



Feature Extraction

- PCA

- Kernel PCA

- Manifold Learning

The Curse of Dimensionality

- Many ML problems involve thousands of features for each training instance
  - Extremely slow, much harder to find a good solution
- Pick two points randomly, average distance between them :

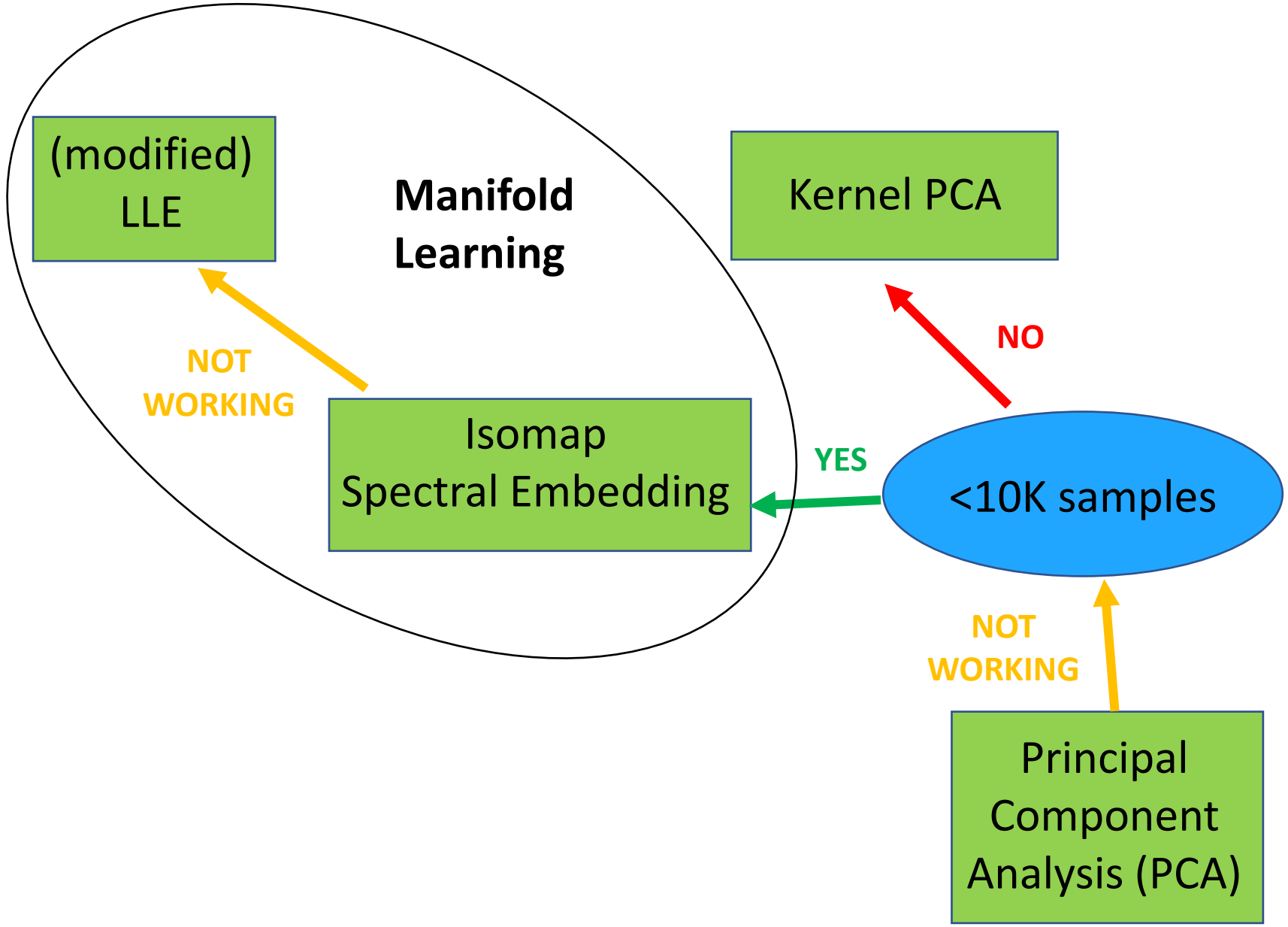
Unit square	Unit 3D cube	Unit 1mio-dim hypercube
0.52	0.66	408.25

  - High-dimensional datasets are at risk of being very sparse!
  - Any new instance will likely be far away from any training instance, making predictions much less reliable

The Curse of Dimensionality: Solutions

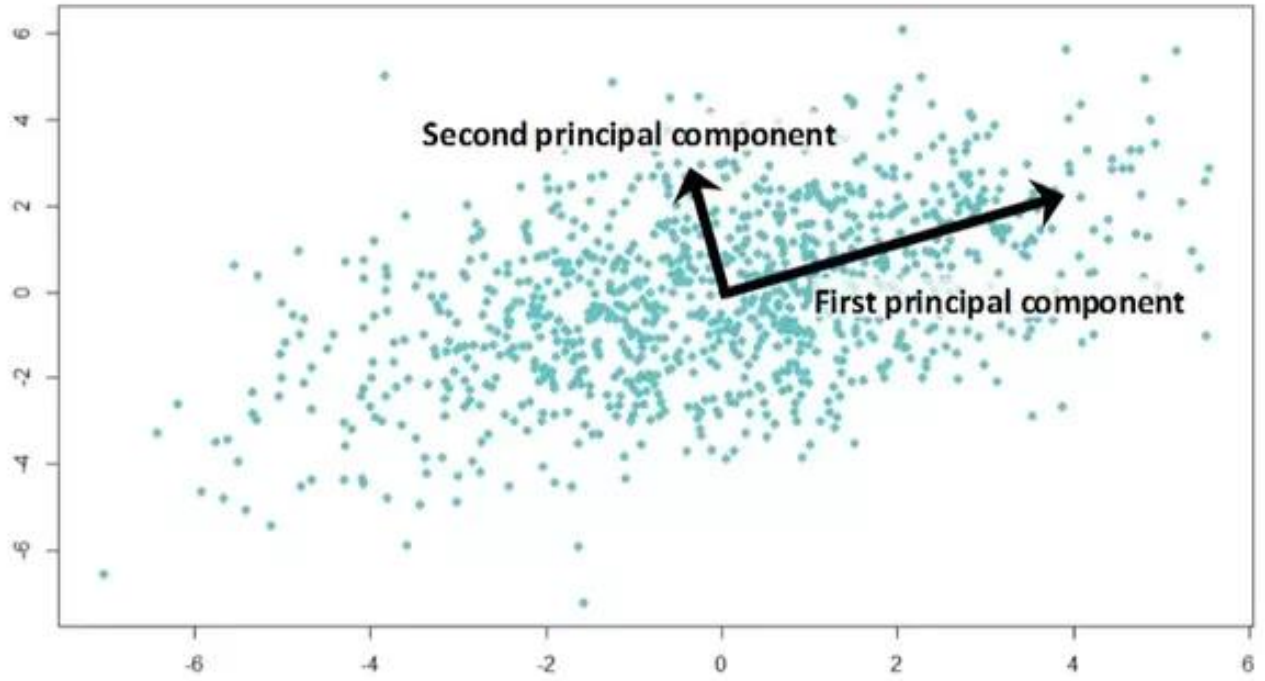
- Increase the size of the training set
  - Required size grows exponentially with number of dimensions
- Reduce the number of features considerably
    - extremely useful for data visualization as well

Feature Extraction Algorithms



Principal Component Analysis (PCA) : Introduction

- Find a new set of orthogonal dimensions and ranked according to the variance of data along them.
  - more important principal axis occurs first
  - the new dimensions allow to predict/reconstruct the original dimensions



Singular Value Decomposition (SVD)

- Eigen-decomposition :
  1) Calculate the covariance matrix *X* of data points.

$$C_x = \frac{1}{n-1}(X - \bar{X})(X - \bar{X})^T \qquad X^T = Transpose\ of\ X$$

  2) Calculate eigenvectors and corresponding eigenvalues

$$[Covariance\ \ matrix] \cdot [Eigenvector] = [eigenvalue] \cdot [Eigenvector]$$

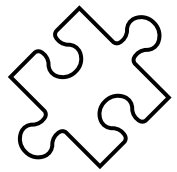
  3) Sort the eigenvectors according to their eigenvalues in decreasing order
  4) Choose first k eigenvectors and that will be the new k dimensions
  5) Transform the original n dimensional data points into k dimensions

- Stop criterium :
  - The reconstruction/projection error should be minimized

PCA : Performance

- Performs well
- Versatile method

- Deals with linear correlations only
- New principal components not interpretable
- Choose the number of components to keep



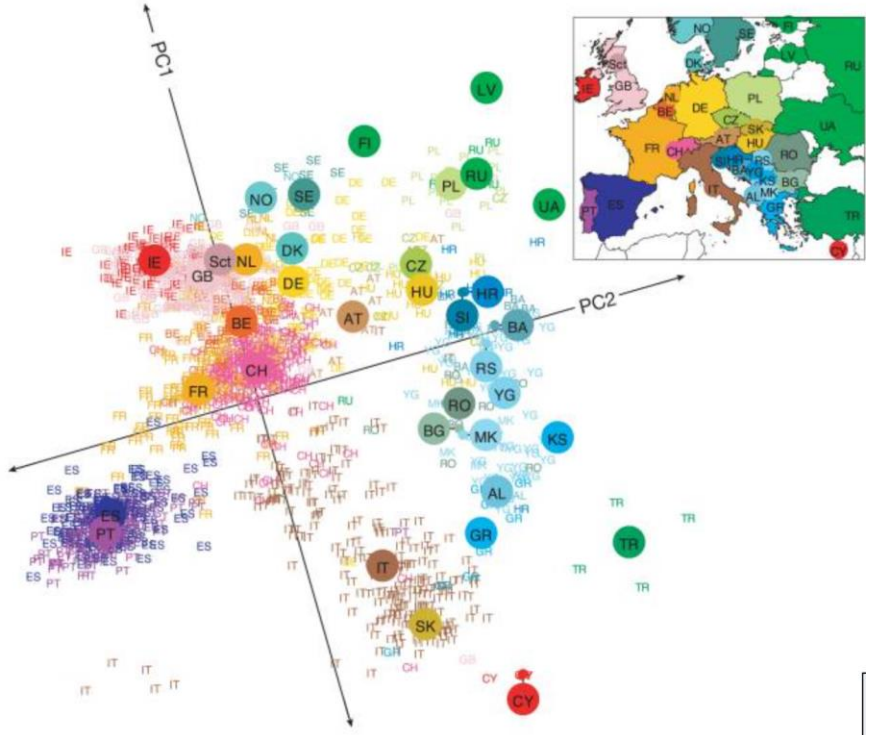
How many components to retain ?

PCA: Applications

Genetic cartography

Goal : characterize genetic variation in European indiviuals

Data : Individual's genotypes



Movie recommendation

Goal : build a system that can make personalized movie recommendations to users

Data : historic user-community ratings

	Movie #1 Toy Story	Movie #2 Monsters Inc.	Movie #3 Saw	Movie #4 Ring	Movie #5 Hitch
User #1	4	5	1	NULL	4
User #2	5	NULL	1	1	NULL
User #3	5	4	3	NULL	3
User #4	5	4	1	1	NULL
User #5	5	5	NULL	NULL	3

PCA : In practice

- sklearn.decomposition.PCA

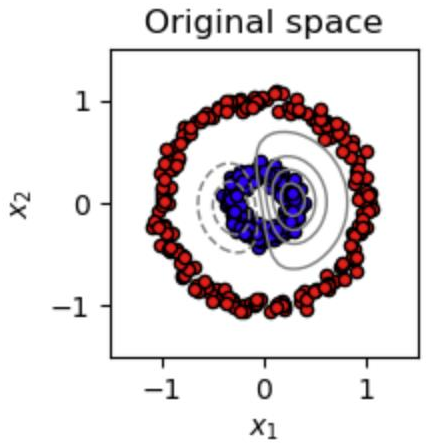
```
>>> import numpy as np
>>> from sklearn.decomposition import PCA
>>> X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
>>> pca = PCA(n_components=2)
>>> pca.fit(X)
PCA(n_components=2)
>>> print(pca.explained_variance_ratio_)
[0.9924... 0.0075...]
>>> print(pca.singular_values_)
[6.30061... 0.54980...]
```

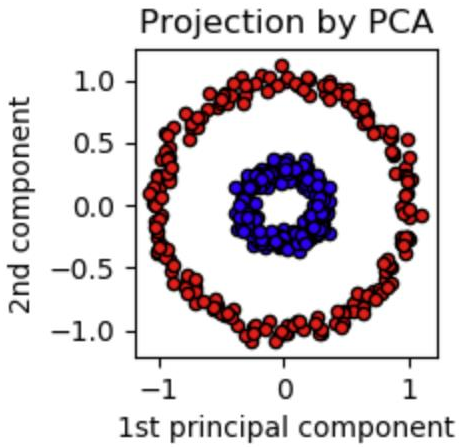
Kernel PCA

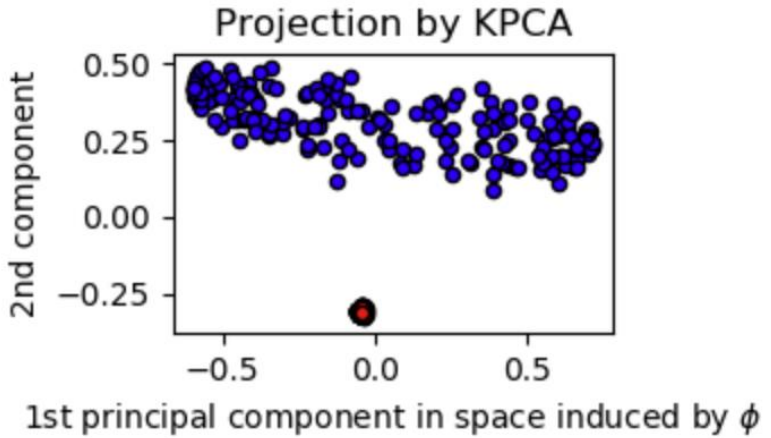
- Non-linear extension of PCA
- Data points mapped into a higher-dim feature space before applying standard PCA in this space

$$x_i \rightarrow \varphi(xi)$$

- Kernel trick : the vectors $\varphi(xi)$ appear only within scalar products (no mapping needed)
  - kernel function k(x,y) replaces the scalar product $(\varphi(x) \cdot \varphi(y))$





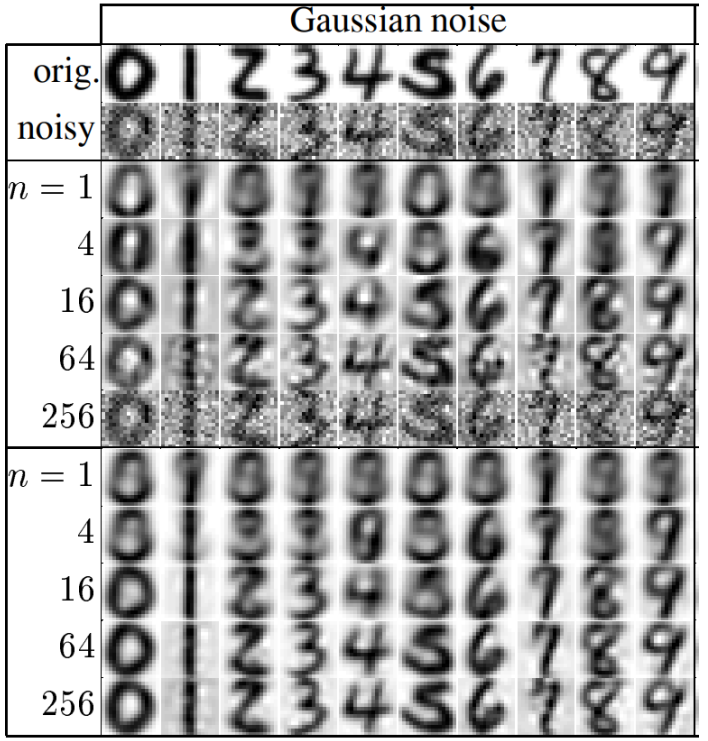


Kernel PCA: Applications

Image denoising

Goal : denoise images

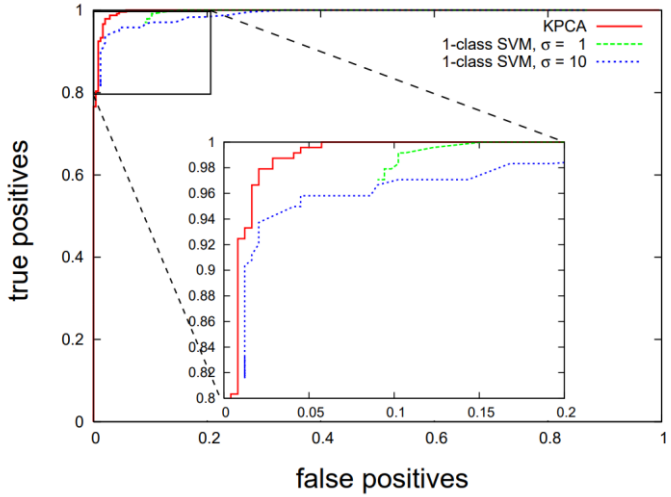
Data : 256-dim handwritten digits



Novelty Detection

Goal : detect novel patterns in new data

Data : breast-cancer data

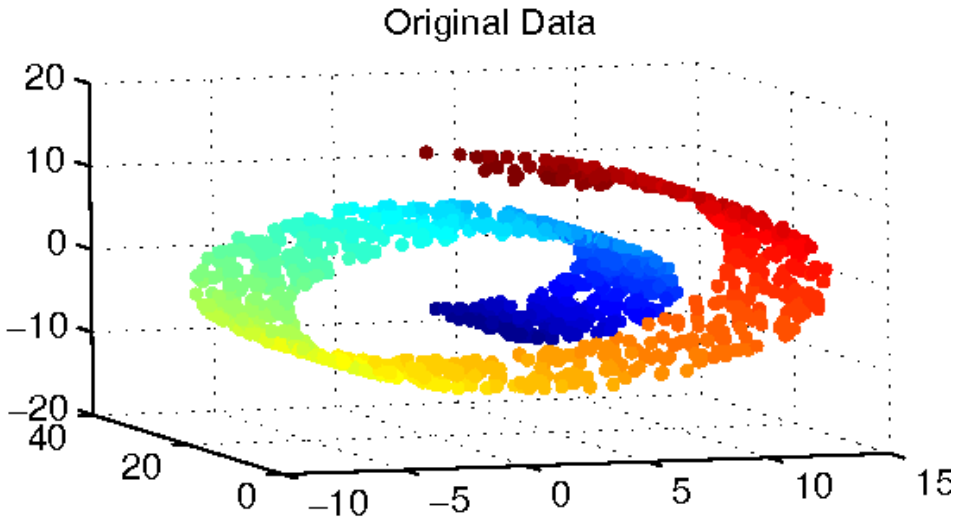


Kernel PCA : In Practice

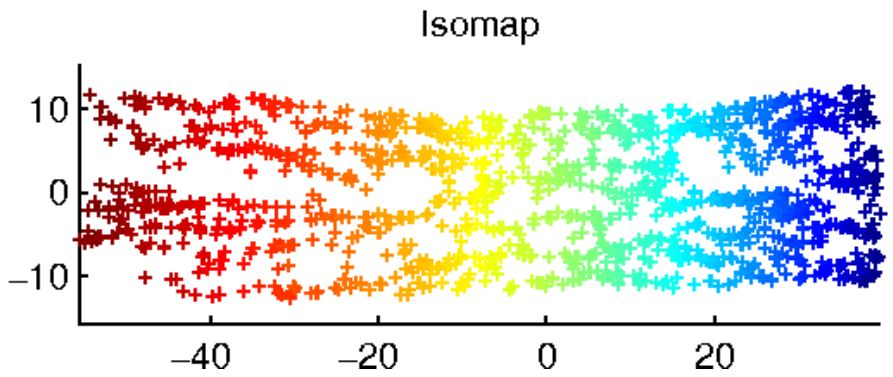
```
>>> from sklearn.datasets import load_digits
>>> from sklearn.decomposition import KernelPCA
>>> X, _ = load_digits(return_X_y=True)
>>> transformer = KernelPCA(n_components=7, kernel='linear')
>>> X_transformed = transformer.fit_transform(X)
>>> X_transformed.shape
(1797, 7)
```

Manifold Learning : Introduction

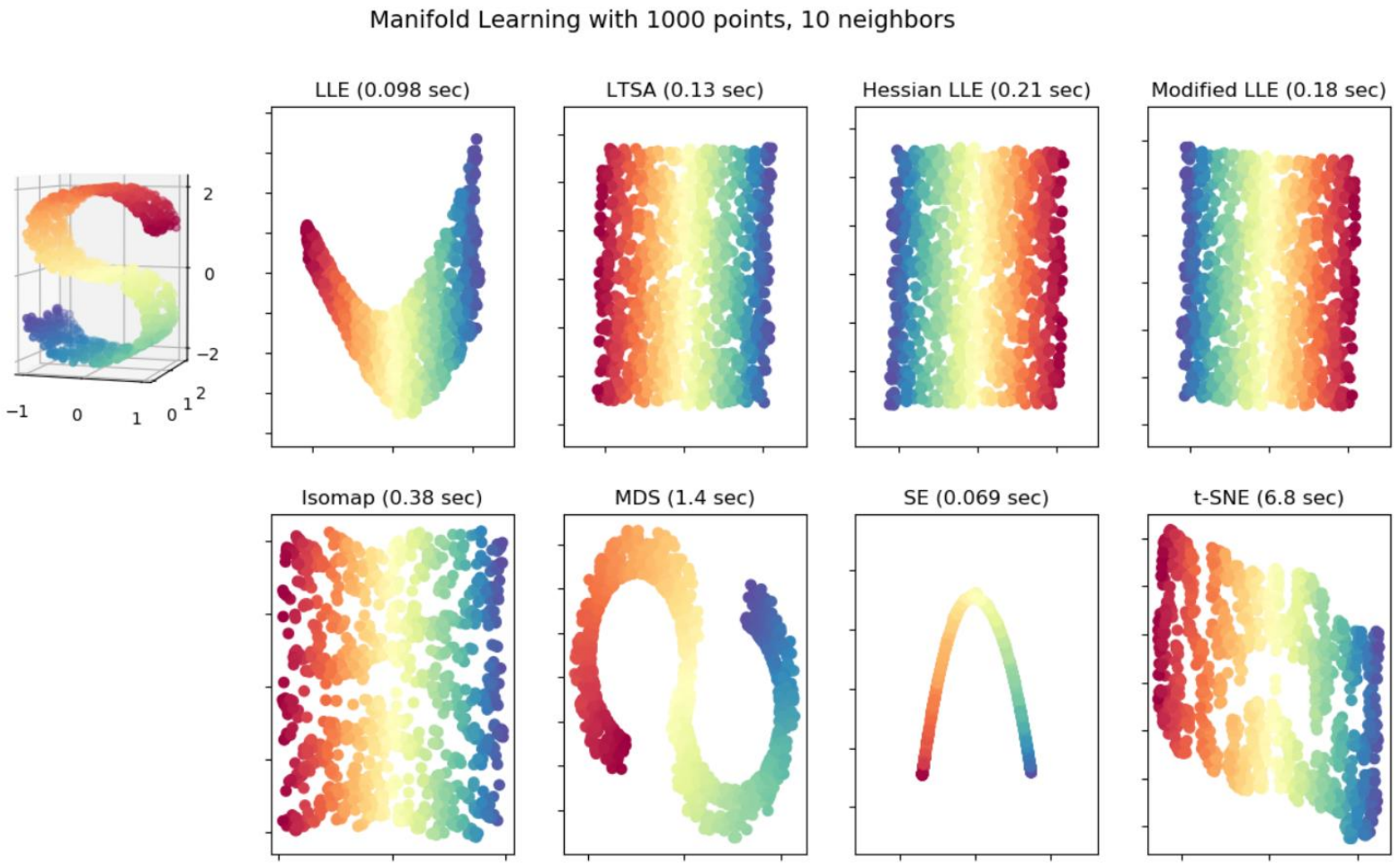
- Non-linear method to visualize the structure of a high-dimensional dataset by reducing its dimension.
  - N = number of training data points
  - D = input dimension



- Use cases :
  - Clustering
  - Dimensionality Reduction
  - Semi-Supervised (labels)

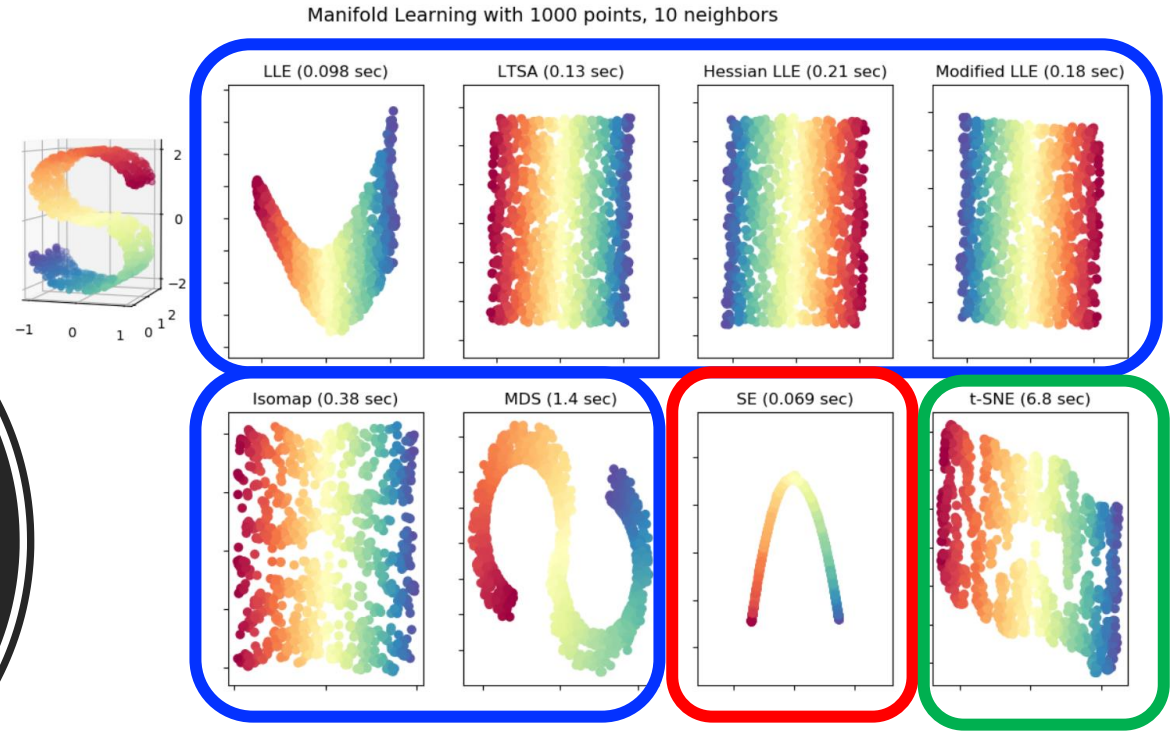


Manifold Learning : Algorithms



- Locally Linear Embedding (LLE), Modified LLE (MLLE), Hessian LLE
- Local Tangent Space Alignment (LTSA)
- Multi-Dimensional Scaling (MDS), Isomap (extension of MDS)
- Spectral Embedding (SE)
- t-distributed Stochastic Neighbor Embedding (t-SNE)

Step 1



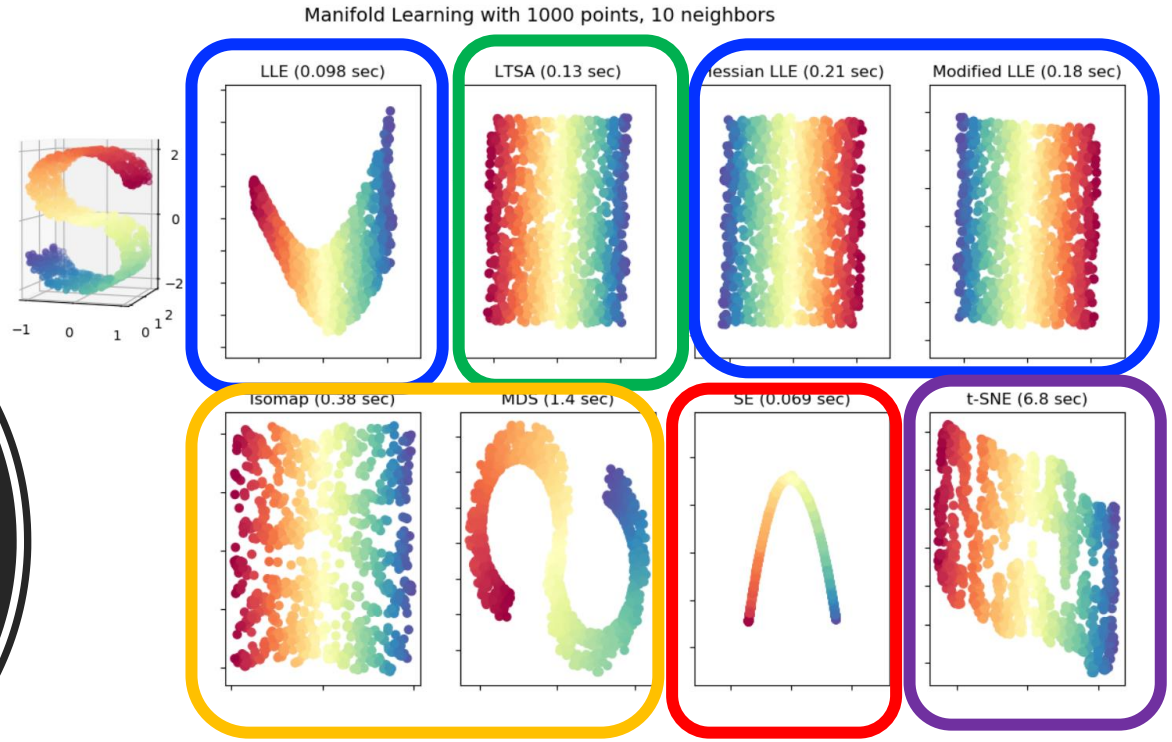
N : number of training data points
D : input dimension
k : number of nearest neighbors
d : output dimension

Nearest Neighbors Search : identify k closest neighbors of training instance x^i

Weighted graph construction : transform the raw input data into graph representation (data points are treated as nodes of a graph)

Similarity measurement : between points in the high dimensional space (Gaussian distribution)

Step 2



LLE : regularization problem if k>D (MLLE, Hessian LLE)

Weight matrix construction :

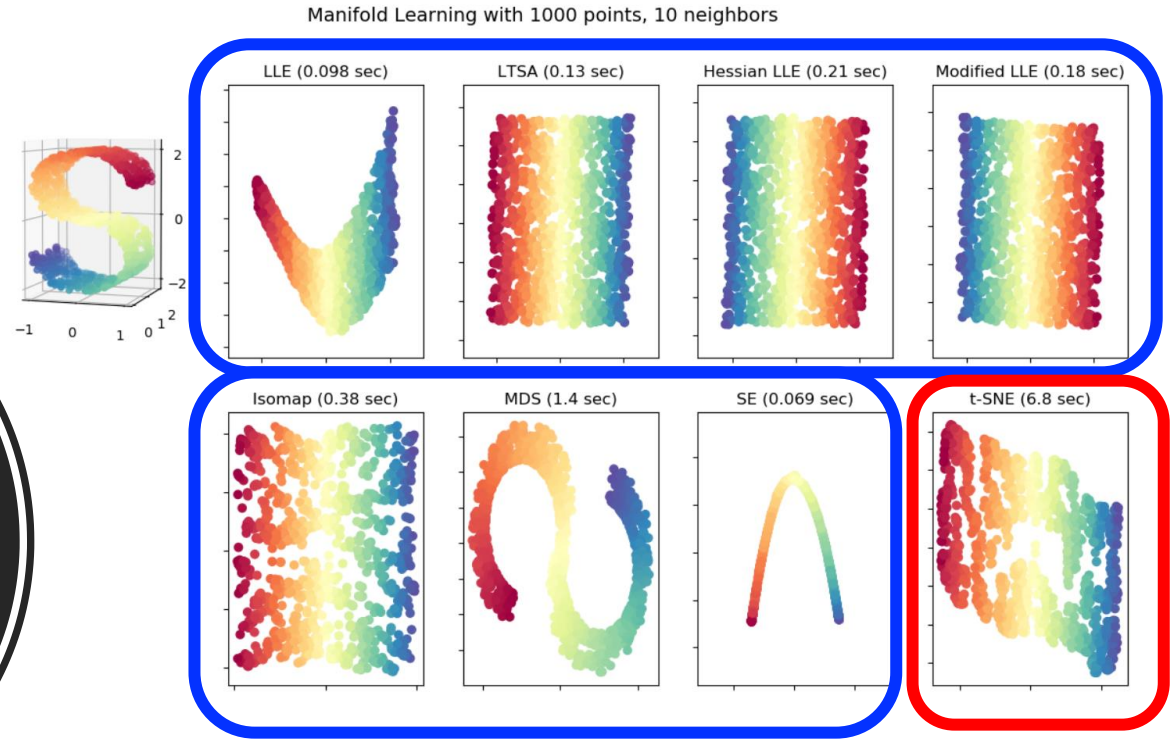
- **preserve neighborhood distances**
- characterize the **local geometry at each neighborhood via its tangent space**

Shortest-path graph search (preserve **geodesic distance for Isomap**)

Graph Laplacian construction

keep similar instances close and dissimilar instances apart (step 2 similar to step 1, except that we use a Student t-distribution in the low dim space)

Step 3

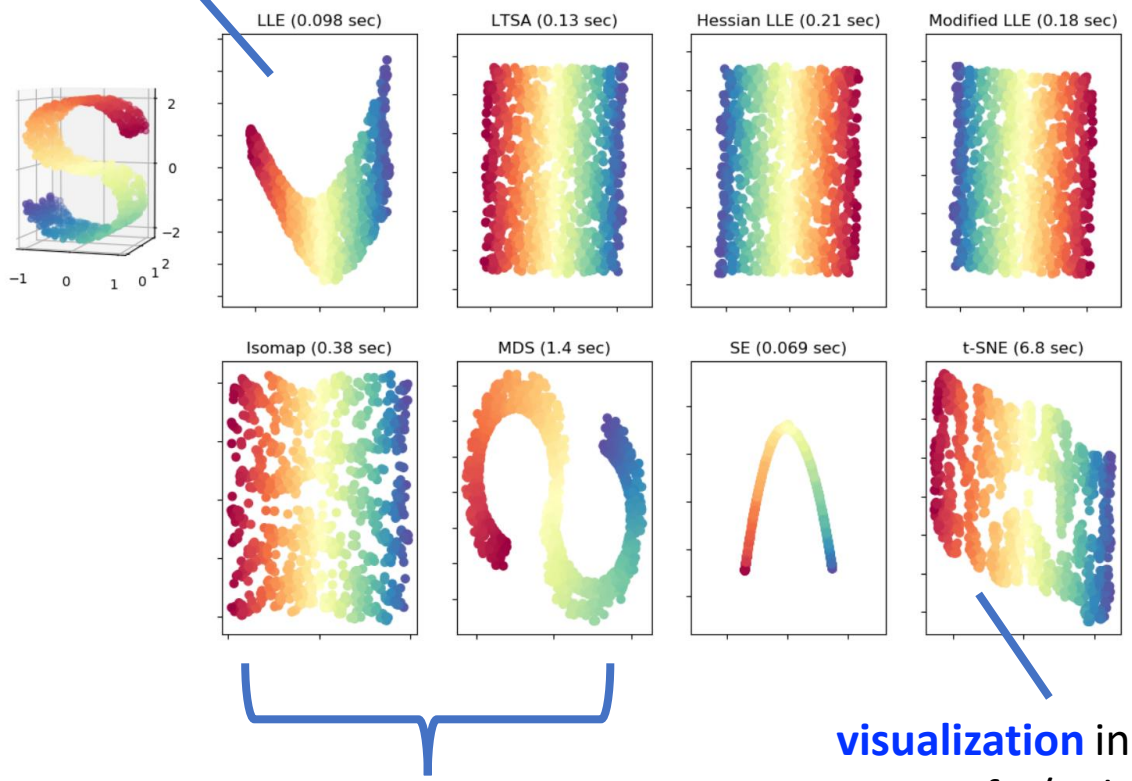


Partial eigenvalue decomposition to map the training instances to a d-dimensional space

Map probabilities of the high and low dimensionality spaces as best as possible (KL divergence)

Use Cases

unroll twisted manifolds when there is **not too much noise**



visualise **similarity** or **dissimilarity** between samples

visualization in a low-dimensional space of 2/3 dimensions

Quiz

https://b.socrative.com/login/student/

Room : CONTI6128