Apprentissage statistique

Apprentissage non supervisé, clustering

Cours pour non-spécialistes

Aurélien Garivier

Outline

Clustering

Goal

Goal: automatically discover clusters in the data

- Model-based clustering:
  - assume that the data was generated by a model
  - try to recover the original model from the data
- Model-free clustering:
  - no assumption on the mechanism producing data
  - vector quantization
  - cluster should be *homogeneous* and *different from one another*
  - data-driven loss function to minimize

Model-based clustering: choice of distance

- Data often in $\mathbb{R}^p$ (or projected to)
- Distance sometimes natural, sometimes not
- Often: need to *normalize* first
- Default choice: Euclidian distance $d(x, x') = \sqrt{\sum (x_i - x_i')^2}$
- Other possible norms: L^1, L^∞, etc.
- Mahalanobis distance: $d(x, x') = \sqrt{(x - x')\Sigma^{-1}(x - x')}$
- Categorial data: typically χ^2 distance

Ressources

- The Elements of Statistical Learning, *T. Hastie, R. Friedman, J. Tibshirani*, Springer
- Data Mining , *S. Tuffry*, Technip
- 

P. Besse et al. `http://wikistat.fr/`
- Interesting overview at `http://scikit-learn.org/stable/modules/clustering.html`
- Interesting demos on `https://www.toptal.com/machine-learning/clustering-algorithms`

Outline

Principal Component Analysis

- Observation $x_1, \ldots, x_n$ in $\mathbb{R}^p$, *centered*
- Dimensionality reduction: replace x_i with $y_i = Wx_i$, where $W \in \mathcal{M}_{d,p}(\mathbb{R})$, $d \ll p$.
- Hopefully, we do not loose too much by replacing x_i by the y_i. Quasi-invertibility: there exists a recovering matrix $U \in \mathcal{M}_{p,d}(\mathbb{R})$ such that for all $i \in \{1, \ldots, n\}$, $\tilde{x}_i = Uy_i \approx x_i$.
- PCA = Data-driven dimensionality reduction tool (for visual clustering)
- PCA aims at finding the compression matrix W and the recovering matrix U such that the total squared distance between the original and the recovered vectors is minimal:

$$\underset{W \in \mathcal{M}_{d,p}(\mathbb{R}), U \in U \in \mathcal{M}_{p,d}(\mathbb{R})}{\arg\min} \sum_{i=1}^{n} \left\| x_i - UWx_i \right\|^2 .$$

Towards the PCA solution

Property. A solution (W, U) is such that $U^T U = I_d$ and $W = U^T$.

Proof. Let $W \in \mathcal{M}_{n,p}(\mathbb{R})$, $U \in U \in \mathcal{M}_{p,d}(\mathbb{R})$, and let $R = \left\{ UWx : x \in \mathbb{R}^p \right\}$. $\dim(R) \leq d$, and we can assume that $\dim(R) = d$. Let $V = \left(\begin{array}{c|c|c} v_1 & \dots & v_d \end{array} \right) \in \mathcal{M}_{p,d}(\mathbb{R})$ be an orthogonal basis of R, hence $V^T V = I_d$ and for every $\tilde{x} \in R^p$ there exists $y \in \mathbb{R}^d$ such that $\tilde{x} = Vy$. But for every $x \in \mathbb{R}^p$,

$$\arg\min_{\tilde{x} \in R} \|x - \tilde{x}\|^2 = V . \arg\min_{y \in \mathbb{R}^d} \|x - Vy\|^2 = V . \arg\min_{y \in \mathbb{R}^d} \|x\| + \|y\|^2 - 2y^T \left(V^T x\right) = VV^T x$$

(as can be seen easily by differentiation in y), and hence

$$\sum_{i=1}^{n} \left\| x_i - UWx_i \right\|^2 \geq \sum_{i=1}^{n} \left\| x_i - VV^T x_i \right\|^2 .$$

The PCA solution

Corollary: the optimization problem can be rewritten

$$\underset{U \in U \in \mathcal{M}_{p,d}(\mathbb{R}) : U^T U = I_d}{\arg\min} \sum_{i=1}^{n} \left\| x_i - UU^T x_i \right\|^2 .$$

Since $\left\| x_i - UU^T x_i \right\|^2 = \|x_i\|^2 - \mathrm{Tr}\left(U^T x x^T U\right)$, this is equivalent to

$$\underset{U \in U \in \mathcal{M}_{p,d}(\mathbb{R}) : U^T U = I_d}{\arg\max} \mathrm{Tr}\left(U^T \sum_{i=1}^{n} x_i x_i^T U \right) .$$

Let $A = \sum_{i=1}^{n} x_i x_i^T$, and let $A = VDV^T$ be its spectral decomposition: D is diagonal, with $D_{1,1} \geq \cdots \geq D_{p,p} \geq 0$ and $V^T V = VV^T = I_p$.

Solving PCA by SVD

Theorem Let $A = \sum_{i=1}^{n} x_i x_i^T$, and let $u_1, \ldots, u_d$ be the eigenvectors of A corresponding to the d largest eigenvalues of A. Then the solution to the PCA optimization problem is $U = \left(\begin{array}{c|c|c} u_1 & \ldots & u_d \end{array} \right)$, and $W = U^T$.

Proof. Let $U \in \mathcal{M}_{p,d}(\mathbb{R})$ be such that $U^T U = I_d$, and let $B = V^T U$. Then $VB = U$, and $U^T A U = B^T V^T V D V^T V B = B^T D B$, hence

$$\mathrm{Tr}\left(U^T A U\right) = \sum_{j=1}^{p} D_{j,j} \sum_{i=1}^{d} B_{j,i}^2 \ .$$

Since $B^T B = U^T V V^T U = I_d$, the colums of B are orthonormal and $\sum_{j=1}^{p} \sum_{i=1}^{d} B_{j,i}^2 = d$.

In addition, completing the columns of B to an orthonormal basis of $\mathbb{R}^p$ one gets $\tilde{B}$ such that $\tilde{B}^T \tilde{B} = I_p$, and for every j one has $\sum_{i=1}^{p} \tilde{B}_{j,i}^2 = 1$, hence $\sum_{i=1}^{d} B_{j,i}^2 \leq 1$.

Thus,

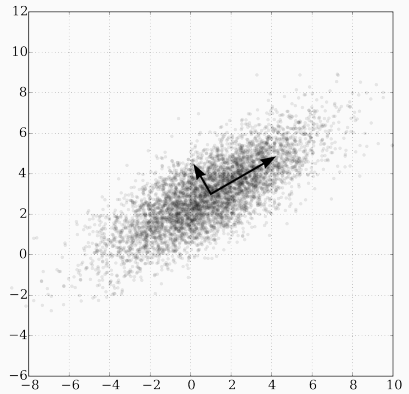
$$\mathrm{Tr}\left(U^T A U\right) \leq \max_{\beta \in [0,1]^p : \|\beta\|_1 \leq d} \sum_{j=1}^{p} D_{j,j} \beta_j = \sum_{j=1}^{d} D_{j,j} \ ,$$

which can be reached if U is made of the d leading eigenvectors of A.

PCA: comments

Interpretation: PCA aims at maximizing the projected variance.

Project onto a subspace $R^d \subset \mathbb{R}^p$ so as to save as much variance as possible



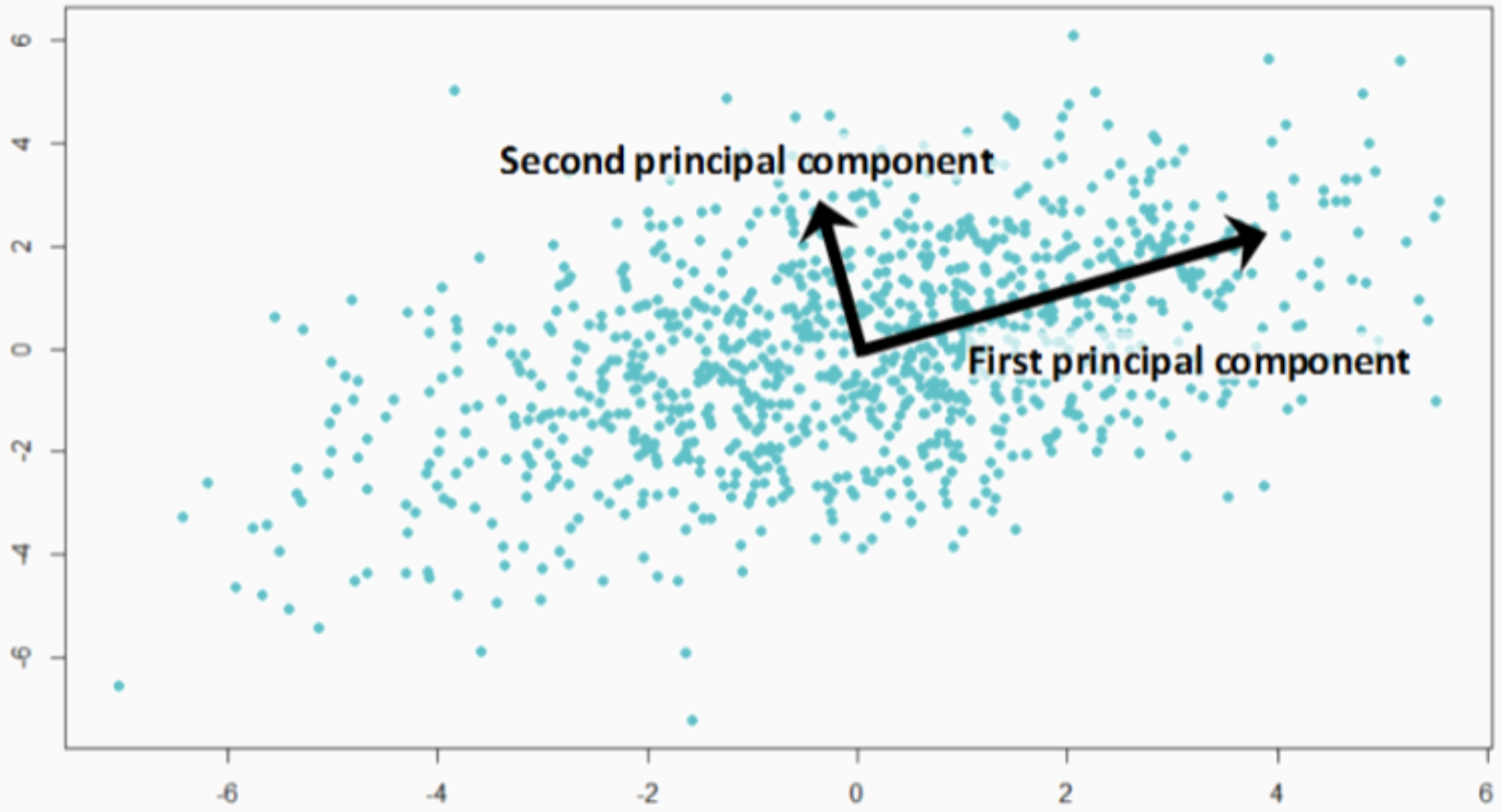
[Src: wikipedia.org]

Outline: orthogonal linear transformation such that the first component has highest variance $w_1 = \arg\max_{\|w\|=1} \sum_i (x_i \cdot w)^2$ is the eigenvector of $X^T X$ corresponding to the highest eigenvalue. Similar reasonning for the next components in the orthogonal of w_1.

Often, the quality of the result is measured by the proportion of the variance explained by the d principal components: $\frac{\sum_{i=1}^{d} D_{i,i}}{\sum_{i=1}^{p} D_{i,i}}$.

In practice: sometimes cheaper to compute svp of $B = X^T X \in \mathcal{M}_n(\mathbb{R})$, since if u is such that $Bu = \lambda u$ then for $v = X^T u / \|X^T u\|$ one has $Av = \lambda v$.

PCA: visualization



Src: [https://techannouncer.com/global-pca-unit-market-2017-adelte-airmak-industries-amss-ltd-cavotec-airport-division-ciat-effeti/]

PCA algorithm

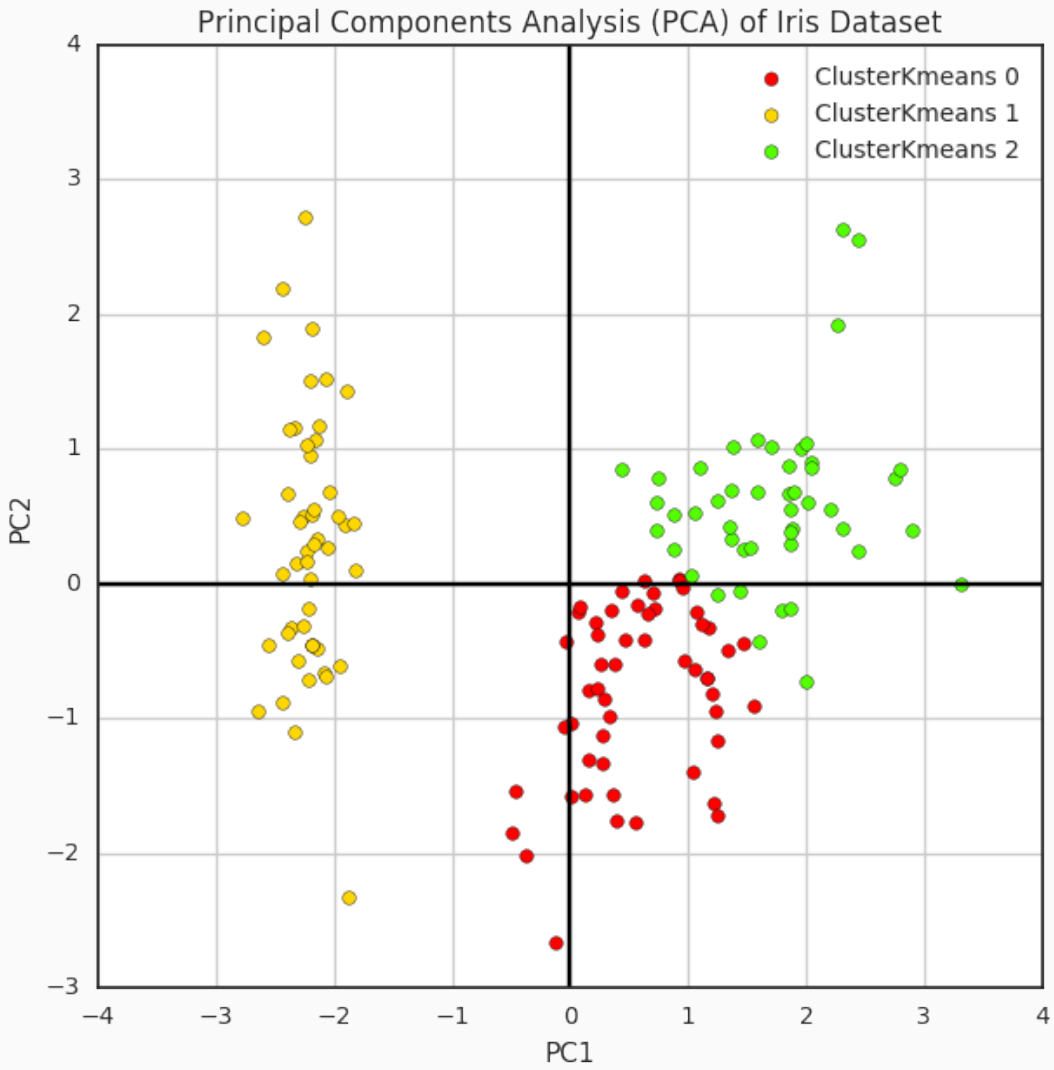
PCA

- Center all variables
- Compute the $p \times p$ empirical covariance matrix $X^T X$.
- Compute the components W_d = the d first eigenvectors of $X^T X$ in decreasing order of the eigenvalues
- Return the projection of X onto the d first components $T_d = X\, W_d$.

Then:

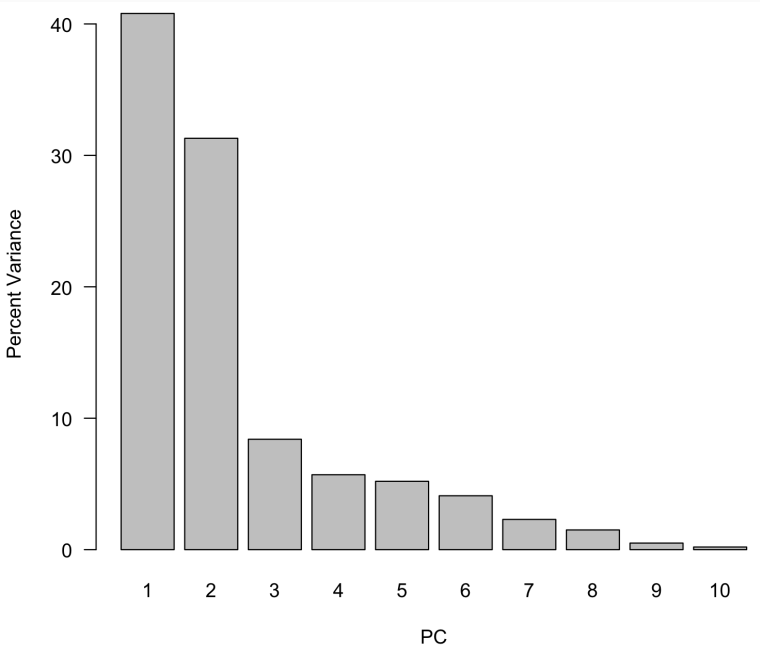
- either vizualize clusters (2d or 3d plots)
- or use another clustering algorithm on the lower-dimensionnal data T_d (*dimension reduction*)

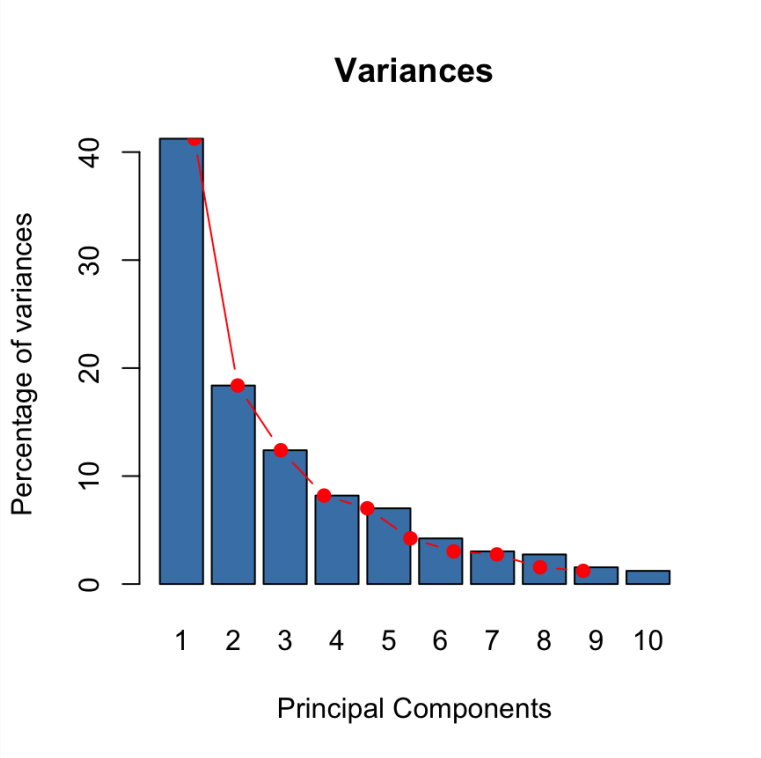
Example: IRIS



[Src: https://www.kaggle.com/bburns/iris-exploration-pca-k-means-and-gmm-clustering]

PCA scree plot





Src: http://strata.uga.edu/8370/lecturenotes/principalComponents.html and

http://www.sthda.com/english/wiki/print.php?id=207

Independent Component Analysis (ICA)

- similar idea, but search for *independent* (instead of uncorrelated) components
- computationally more demanding, but iterative (entropy-based) algorithm exists
- often used for blind source separation
- see also Non-negative Matrix Factorization (NMF)
- for vizualization, see also t-SNE: *t*-distributed Stochastic Neighbor Embedding

Commands

scikitlearn:

```
class sklearn.decomposition.PCA(n_components=None,
copy=True, whiten=False, svd_solver=?auto?, tol=0.0,
iterated_power=?auto?, random_state=None)
```

ICA: `sklearn.decomposition.FastICA`

R: package stats

```
prcomp() et princomp() [fonction de base, package stats],
PCA() [package FactoMineR],
dudi.pca() [package ade4],
epPCA() [package ExPosition]
```

ICA: `ica`

Outline

Model-based approach: Gaussian Mixtures

- Observations $X_1, \ldots, X_n$ in $\mathbb{R}^p$
- K cluster centers: $\mu_1, \ldots, \mu_K$
- Cluster k has probability p_k
- Points in cluster k have law $\mathcal{N}(\mu_k, \Sigma)$

- Given a sample $(X_1, \ldots, X_n)$, how to estimate the cluster centers $(\mu_k)_k$ and how identify the clusters?

Expectation-Maximization Algorithm

- Likelihood: the parameter $\theta = (p, \mu)$ has likelihood

$$L(\theta) \propto \sum_{z_1,\ldots,z_n \in \{1,\ldots,K\}^n} \prod_{i=1}^{n} p_{z_i} e^{-\frac{1}{2}\Sigma^{-1}(X_i - \mu_{z_i})\Sigma^{-1}}$$

  $\Longrightarrow$ non-convex, very hard to maximize

- *Approximate* iterative optimization
- can also optimize on the covariance matrix Σ if unknown

Expectation-Maximization Algorithm

Given an estimate $\theta^j = (p^j, \mu^j)$, compute

- *membership weights*

$$w_{i,k}^j = P_{\theta^j}(z_i = k|X_i) = \frac{p_k^j \, e^{-\frac{1}{2}\Sigma^{-1}(X_i - \mu_k^j)\Sigma^{-1}}}{\sum_{\ell=1}^K p_\ell^j \, e^{-\frac{1}{2}\Sigma^{-1}(X_i - \mu_\ell^j)\Sigma^{-1}}}$$

- updated cluster weights:

$$p_k^{j+1} = \frac{\sum_{i=1}^n w_{i,k}^j}{n}$$

- updated cluster means:

$$\mu_k^{j+1} = \frac{\sum_{i=1}^n w_{i,k}^j \, X_i}{\sum_{i=1}^n w_{i,k}^j}$$

Convergence of the Expectation-Maximization Algorithm

Theorem

The likelihood of the iterates are increasing:

$$L\left(\theta^{j+1}\right) \geq L\left(\theta^{j}\right)$$

Good: converges

Bad: local optimum

EM Algorithm

- randomly initialize θ_0
- compute EM iterations until convergence:
  - *membership weights* $(w_{i,k}^{j})_{i,k}$
  - *updated cluster weights* $(p_k^{j+1})_k$
  - *updated cluster means* $(\mu_k^{j+1})_k$
- start again (a few times) to look for a better local optimum

Commands

scikitlearn:

```
class sklearn.mixture.GaussianMixture(n_components=1,
covariance_type=?full?, tol=0.001, reg_covar=1e-06,
max_iter=100, n_init=1, init_params=?kmeans?,
weights_init=None, means_init=None, precisions_init=None,
random_state=None, warm_start=False, verbose=0,
verbose_interval=10)
```

R:

```
MClust() [package MASS],
GuassianMixtures() [package sBIC]
```

Outline

Model-free clustering

- Observations $X_1, \ldots, X_n$ in $\mathbb{R}^p$;
- Objective function: for candidate cluster centers $\mu = (\mu_1, \ldots, \mu_K)$ and cluster assignations $z = (z_1, \ldots, z_n)$:

$$L(\mu, z) = \sum_{k=1}^{K} \sum_{i:z_i=k} \|X_i - \mu_k\|^2 = \sum_{i=1}^{n} \sum_{k=1}^{k} \mathbb{1}\{z_i = k\} \|X_i - \mu_k\|^2$$

- If $S_k = \{i : z_i = k\}$,

$$L(\mu, z) = \sum_{k=1}^{K} |S_k| \, \mathbb{V}ar[S_k]$$

- Minimizing L is equivalent to minimizing pairwise deviations in the clusters:

$$\arg\min_{\mu,z} L(\mu, z) = \arg\min_{\mu,z} \sum_{k=1}^{K} \frac{1}{|S_k|} \sum_{i,j \in S_k} \|X_i - X_j\|^2$$

Lloyd's algorithm

- For a fixed μ, optimizing in z is easy: choose $z_i = \arg\min_k \|X_i - \mu_k\|$
- BUT optimizing in μ is NP-hard!

k-means

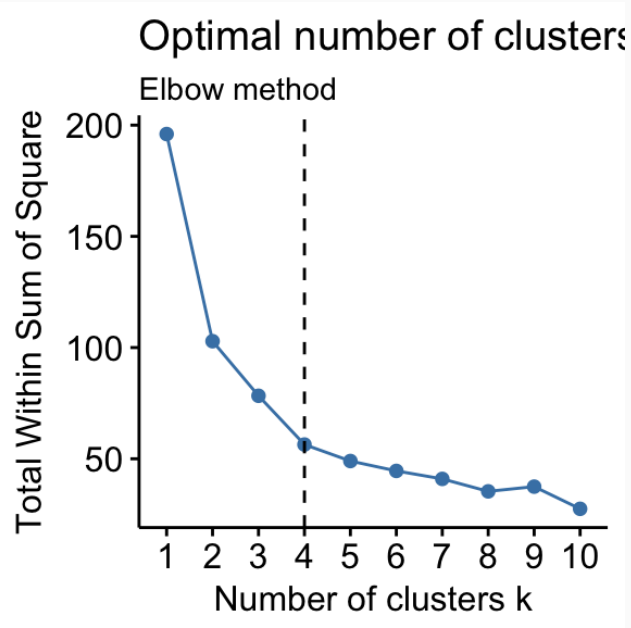
- randomly initialize θ_0
- compute Lloyd's iterations until convergence:
  - *membership variables* $z_i^j = \arg\min_k \|X_i - \mu_k^j\|$
  - *updated cluster weights* $N_k^j = \sum_{i=1}^n \mathbb{1}\{z_i^j = k\}$
  - *updated cluster means* $\mu_k^{j+1} = \frac{\sum_{i:z_i^j=k} X_i}{N_k^j}$
- start again (a few times) to look for a better local optimum

Comments on k-means

- k-means is a "hard" version of EM for mixtures! (when variances tends to 0)
- Complexity: linear in n and k (fast)
- Requires only a dissimilarity measure (not necessarily a distance)
- Quality of solution found depends on (random) initialization
- Not robust to outliers

- Problem: how to choose k?

[Src: http://www.sthda.com/english/articles/29-cluster-validation-essentials/]



Improved Initialization: k-means++

Idea: enforce distant cluster centers from the start

k-means++

- Choose first center μ_1^1 at random
- for $j = 2$ to K, repeat:
  - compute $D_i^j = \min_{\ell<j} \|X_i - \mu_\ell^1\|$
  - choose $\mu_j^1 = X_i$ with probability proportionnal to D_i^j

- theoretical guarantee of $O(\log k)$ approximation from the start
- still linear complexity
- often a dramatic improvement in practice

k-medoids: robustness vs computation time

k-medoids

- randomly initialize θ_0 with K points of the dataset
- iterate until convergence:
  - *membership variables* $z_i^j = \arg\min_k \|X_i - \mu_k^j\|$
  - *updated cluster medoids* $\mu_k^{j+1} = \arg\min_{i \in S_k} \sum_{\ell \in S_k} \|X_\ell - X_i\|$
- start again (a few times) to look for a better local optimum

- Robust to outliers (cf median versus mean)
- BUT computation time is quadratic in n
- k-means is k-medoids with $\|X_\ell - X_i\|^2$ instead of $\|X_\ell - X_i\|$

Commands

scikitlearn:

```
class sklearn.cluster.KMeans(
n_clusters=8, init=?k-means++?, n_init=10, max_iter=300,
tol=0.0001, precompute_distances='auto', verbose=0,
random_state=None, copy_x=True, n_jobs=1, algorithm='auto'
```

R: package stats

```
kmeans(x, centers, iter.max = 10, nstart = 1,
algorithm = c("Hartigan-Wong", "Lloyd", "Forgy",
"MacQueen"), trace=FALSE)
```

Variants to be found in various packages: `ClusterR`, `kmed`, etc.

Outline

Agglomerative clustering

- greedy bottom-up algorithm
- requires a distance (idssimilarity) between observations $\|x - x'\|$
- choice of *distance between clusters:*
  - complete linkage: $d(A, B) = \max\{\|x - x'\| : x \in A,\, x' \in B\}$
  - single linkage: $d(A, B) = \min\{\|x - x'\| : x \in A,\, x' \in B\}$
  - average linkage distance: $d(A, B) = \dfrac{1}{|A|\,|B|} \displaystyle\sum_{x \in A} \sum_{x' \in B} \|x - x'\|$
  - Ward distance for Euclidian mean: $d(A, B) = \dfrac{|A|\,|B|}{n\big(|A| + |B|\big)} \big\|\bar{A} - \bar{B}\big\|$
  - sum of intra-cluster variance
  - etc.

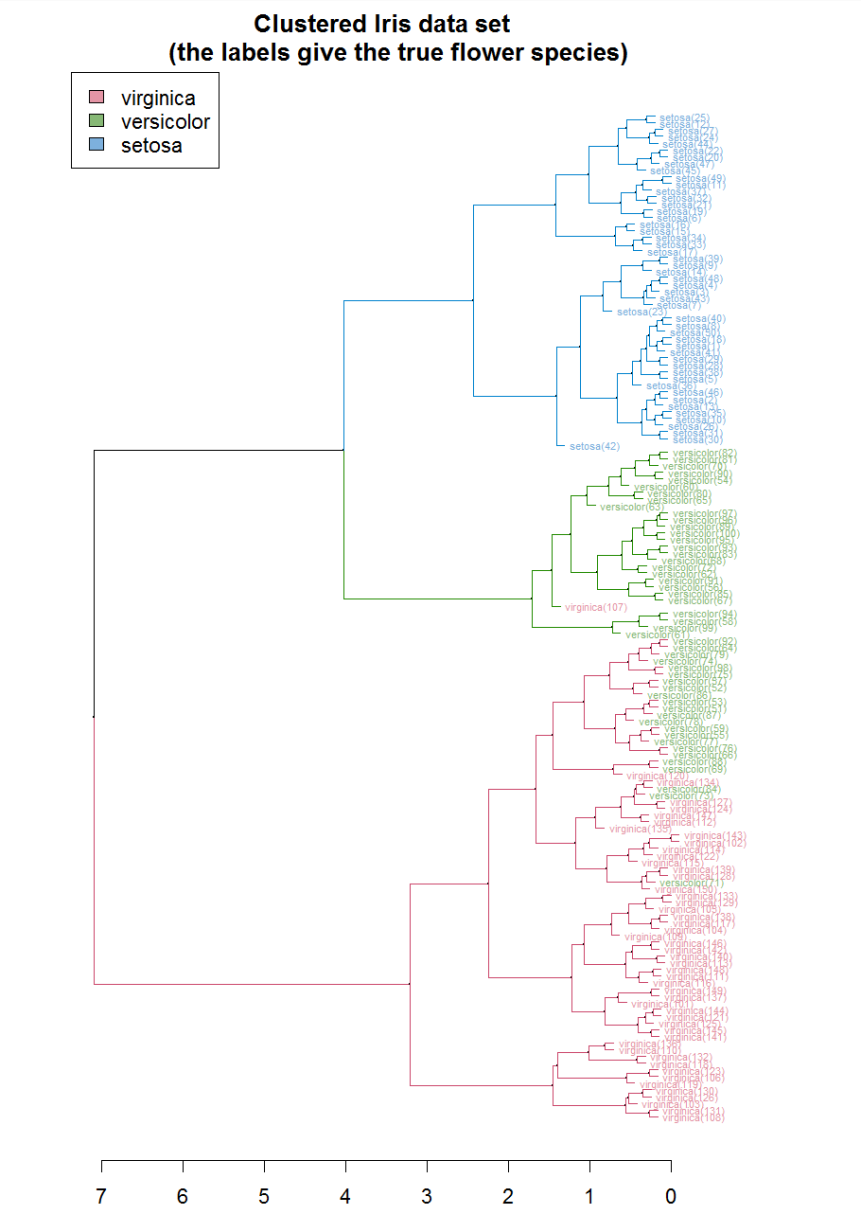
HCA algorithm

HCA

- Initialization: all observations are clusters $\{X_1\}, \ldots, \{X_n\}$
- As long as there are at least two clusters:
  - add a link between two clusters with smallest distance
  - merge them for the next iterations
- Return the *dendrogram* = hierarchy of clusters

Property of Ward for Euclidian distance: interclass variance decreasing with the number of classes

Dendrogram



Author: Talgalili

https://commons.wikimedia.org/wiki/File:Iris_dendrogram.png

Pros and Cons

- No need to specify the number of clusters in advance
- A relevant choice can be deduced from the observation of the dendrogram (and practical needs)
- Computational complexity in $O(n^2)$
- Does not find an optimal solution

Commands

scikitlearn:

```
class sklearn.cluster.AgglomerativeClustering(n_clusters=2,
affinity=?euclidean?, memory=None, connectivity=None,
compute_full_tree=?auto?, linkage=?ward?,
pooling_func=<function mean>)
```

R:

```
hclust(d, method = "complete", members = NULL)
```

Outline

Affinity Propagation

- Message-passing algorithm
- affinity function $s(x, x')$, for example $s(x, x') = -\|x - x'\|^2$
- $s(x, x) =$ input preference : the lower, the higher the chances to be an exemplar
- responsibility matrix R: $r(i, k) =$ how well-suited X_k can serve as an exemplar for X_i (wrt other candidate exemplars)
- availability matrix A: $a(i, k) =$ how appropriate it is for X_i to pick X_k as an examplar, taking into account other points preferences for X_k as an exemplar

Affinity Propagation

- Initialize R and A with 0
- Repeat until convergence:
  - update responsibilities: $r(i,k) \leftarrow s(i,k) - \max_{k' \neq k} a(i,k') + s(i,k')$
  - update availabilities:

$$a(i,k) \leftarrow \min \left\{ 0, r(k,k) + \sum_{i' \notin \{i,k\}} \max(0, r(i',k)) \right\} \text{ for } i \neq k \text{ and}$$

$$a(k,k) \leftarrow \sum_{i' \neq k} \max\left(0, r(i',k)\right)$$

- Pick exemplars as maximizers of $r(i,i) + a(i,i)$

Affinity Propagation: pros/cons

- No need to specify the number of clusters
- ... but a parameter plays the same role
- quadratic time complexity
- some improvement over k-means in some cases

Commands

scikitlearn:

```
class sklearn.cluster.AffinityPropagation(damping=0.5,
max_iter=200, convergence_iter=15, copy=True,
preference=None, affinity=?euclidean?, verbose=False)
```

R: package APCluster

```
apcluster(s, x, p=NA, q=NA, maxits=1000,
convits=100, lam=0.9, includeSim=FALSE, details=FALSE,
nonoise=FALSE, seed=NA)
```

Spectral Clustering

- Similarity matrix $S_{i,j}$, for example $S_{i,j} = -\|X_i - X_j\|^2$
- Idea: use standard clustering method on eigenvectors of the (normalized) Laplacian matrix

$$L = Id - D^{-1/2} S D^{-1/2}$$

  where D is diagonal with $D_{i,i} = \sum_j S_{i,j}$
- Intuition: if S is diagonal by blocks, the eigenvectors are the indicators of the blocks

Commands

scikitlearn:

```
class sklearn.cluster.SpectralClustering(n_clusters=8,
eigen_solver=None, random_state=None, n_init=10, gamma=1.0,
affinity=?rbf?, n_neighbors=10, eigen_tol=0.0,
assign_labels=?kmeans?, degree=3, coef0=1,
kernel_params=None, n_jobs=1)
```

R: package kernlab

```
specc(x, data = NULL, na.action = na.omit, ...)
```

DBscan

- DBscan = density-based spatial clustering of applications with noise
- parameters: radius ϵ and minimal cluster size *minSize*

DBscan

Repeat as long as at least one point has not been visited:

- pick an unvisited point X_i at random
- if it has less than *minSize* ϵ-neighbors, mark it as outlier
- other, form the cluster of all points that can be reached by jumps of at most ϵ starting from X_i

DBscan: comments

- simple and fast
- no need to specify number of clusters
- Problem: sensitive to the choice of parameters
- choosing the right parameters ϵ and *minSize* propertly is hard
- choice of ϵ: such that the proportion of outliers is at most 10% (say)
- choice of *minSize*: such that at least 90% have at least *minSize* neighbors
- unable to handle clusters with very different densities

Commands

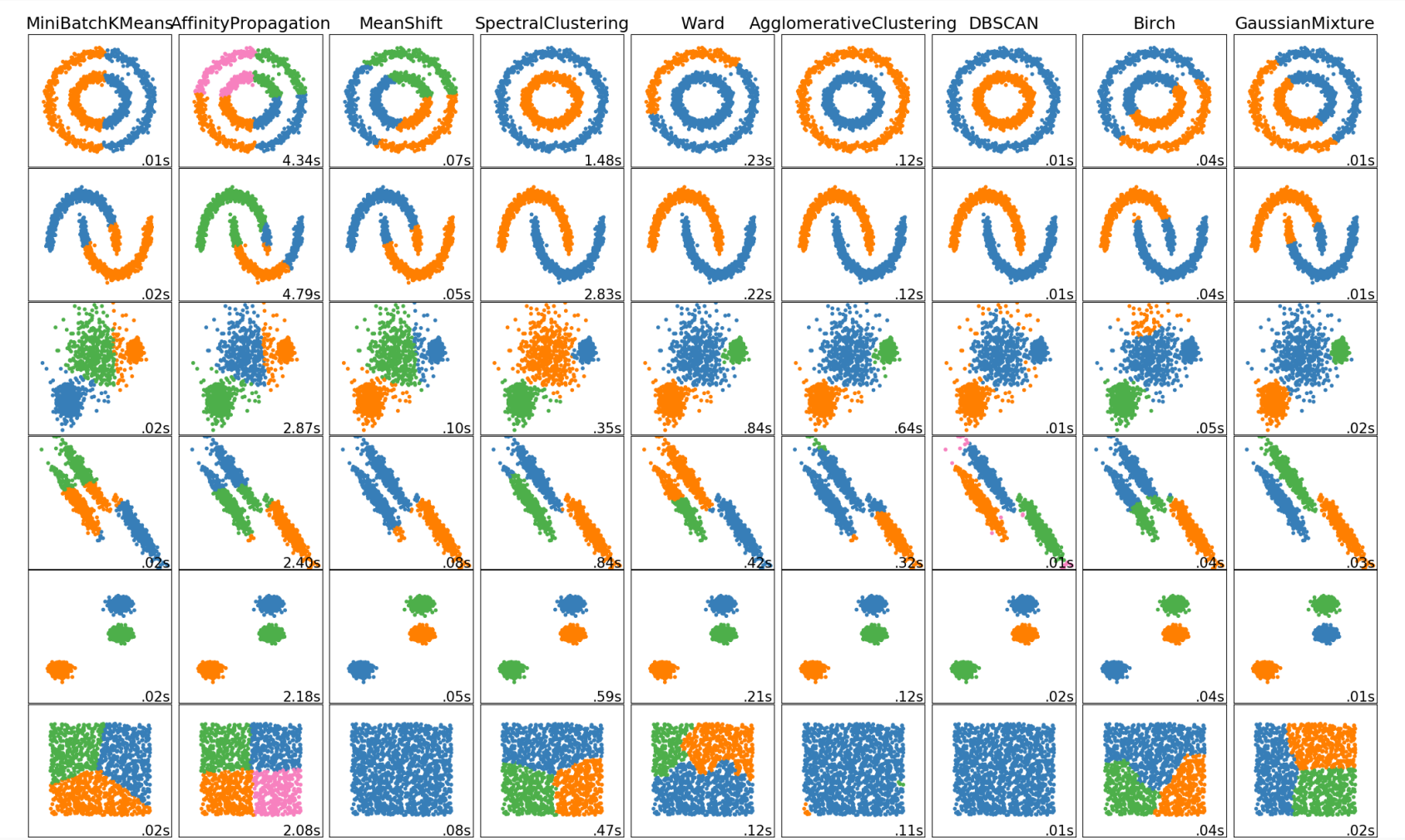
scikitlearn:

```
class sklearn.cluster.DBSCAN(eps=0.5, min_samples=5,
metric='euclidean', metric_params=None, algorithm='auto',
leaf_size=30, p=None, n_jobs=1)
```

R: package dbscan

```
dbscan(x, eps, minPts = 5, weights = NULL,
borderPoints = TRUE, ...)
```

Which algorithm to choose?



Src: [http://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html]

Clustering time series

Features:

- mean
- trend
- auto-correlation coefficients
- inter-series correlation
- etc.

$\Longrightarrow$ it depends on the nature of the problem and on the goal of the clustering!