Machine learning 6

More optimization, Unsupervised learning, Clustering

Stefano Rovetta

A.y. 2016-2017

What is learning?

- We want to optimize some cost function $R = R(\mathbf{w})$

- Learning as optimization
- Learning as **stochastic optimization**

Stochastic optimization

- Optimize a cost that is a **random variable**
- Types of randomness:
- Measurement plus noise: $R + \nu$
- Multiple effects mixed together (we might use a **mixture model**)
- Unknown statistical properties

Monte Carlo integration

- True distribution $p_f(\xi) \rightarrow$
- Expectation of f:

$$\mathbb{E}\{f\} = \int \xi \, p_x(\xi) \, d\xi$$

- **Empirical distribution** $P_x(\xi) = \frac{1}{n} \sum_{l=1}^{n} \delta(\xi - x_l) \rightarrow$
- Expectation of f:

$$\mathbb{E}\{f\} \approx \int \xi \, P_x(\xi) \, d\xi = \frac{1}{n} \sum_{l=1}^{n} f(x_l)$$

- This is a **Monte Carlo integral**

- Suppose that R is the classification risk in a learning task.
- We want to optimize the true risk (expected loss):

$$R(\mathbf{w}) = \int R(\alpha(\mathbf{x}), \mathbf{w})\, p(\mathbf{x}) d\mathbf{x}.$$

- This is a function of $\mathbf{w}$
  (the weights identify one specific learner)
- It is also a function of the data distribution $p(\mathbf{x})$
  (the performance is estimated on the data)

- When training a learner we don't have $p(\mathbf{x})$, but only X
- From X we have the empirical distribution

$$P_x(\xi) = \frac{1}{n} \sum_{l=1}^{n} \delta(\xi - x_l)$$

- so we can compute a Monte Carlo estimate of the expected loss

$$\hat{R}(\mathbf{w}, X) = \frac{1}{n_p} \sum_{l=1}^{n_p} R(\alpha(\mathbf{x}_l), \mathbf{w})$$

this is the **empirical risk**.

Training by epoch

- is computing R (and the optimal learner's parameters)
- on the basis of a Monte Carlo estimate $\hat{R}$ of risk

- Finds the optimal value of an **approximate cost function**

Stochastic gradient descent

- is looking for the optimal R
- by applying **gradient descent** to $\hat{R}$

This approach in general is called Empirical Risk Minimization

Stochastic approximation

- A special kind of stochastic optimization
- R is estimated at each input pattern **using that pattern alone**
- Extremely unreliable estimation – but it converges in probability!
- Robbins and Monro, 1951; Kiefer and Wolfowitz, 1952

- Convergence in probability:

$$\lim_{n\to\infty} \Pr\left(|\hat{R}_n - R| \geq \varepsilon\right) = 0$$

- $\hat{R}_n$ is the estimate of R on a training set of size n

Stochastic approximation

- Given:
- A function R whose gradient ∇R we want to minimize (but we can't measure)
- A sequence $G_1, G_2, \ldots, G_l, \ldots$ of random samples of ∇R, affected by random noise
- A decreasing sequence $\eta_1, \eta_2, ldots, \eta_l, \ldots$ of step size coefficients
- Basic iteration:

$$\mathbf{w}(l+1) = \mathbf{w}(l) - \eta_l \, G_l$$

Stochastic approximation: The intuition

- Each sample gives a noisy (stochastic) estimate of the gradient
- $\Rightarrow \nabla R$ + noise
- By averaging over time, noise cancels out
- **Random variations also make it possible to escape local minima**

Results on convergence of stochastic approximation

- If R is twice differentiable and convex,
  then stochastic approximation converges with a rate of $O\left(\frac{1}{l}\right)$
- A condition of convergence (not optimal rate of convergence):

$$0 < \sum_{l} \eta_l^2 = A < \infty$$

- Usually the hypotheses are not met (complex cost landscape) and we don't have guarantees.

Training by pattern

- is computing $\hat{R}$ (and the ΔW)
- on the basis of an estimate of risk on a **single point**
- An extreme Monte Carlo estimate on a training set of one observation only

- Finds the **approximate optimal value** of an **approximate cost function**

Implementation of training

- By epoch: estimation loop, then update
- By pattern: estimation + update loop
- By pattern on a training set: l = random
- Learning rate $\eta \rightarrow$ By pattern: keep it **low**
- $\rightarrow$ By epoch: make it **adaptive**

What do we have up to now?

- Geometric representation of data as points/vectors
- Classification theory, when probabilities are known (no need for data)
- The concept of loss function and risk
- Parameter estimation, when probabilities are unknown but their functional form is known: the maximum likelihood criterion
- Uncertainty in estimation: bias, variance
- Some possible indexes to evaluate classifier quality
- Some ideas on how to optimize classifier quality; gradient descent

→ Now a digression (but not so much after all)

Unsupervised learning

GOAL:

Find hidden structure in unlabeled data.

No error or reward measure to evaluate a potential solution.

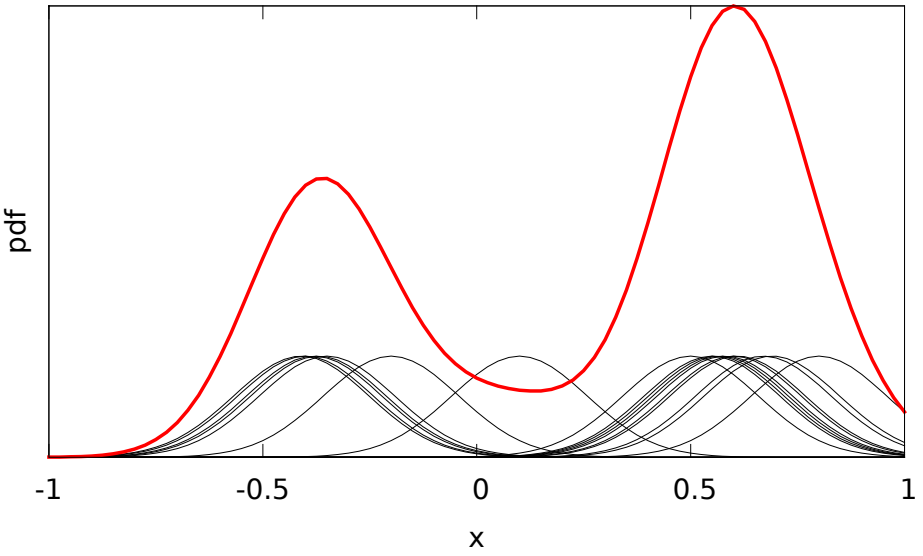
Cost functions based on data distribution, pairwise relationships, distances from reference points...

Why is unsupervised learning important?

- Most learning in nature occurs without a target (in general)
- There are computational models of how (in particular) some brain and retina functions are learned by simple unsupervised learning rules
- Class labels are often **very expensive** to obtain
- Some tasks are intrinsically unsupervised.
  For instance: "Features" in data describe data structure, not categories. Feature extraction (or **feature learning**) is normally an unsupervised task

Unsupervised tasks

Probability density approximation



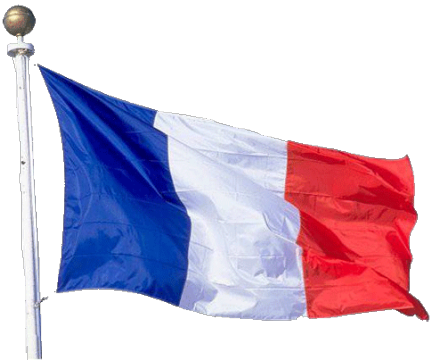
Unsupervised tasks

Concept formation (learning natural classes in data): **clustering**

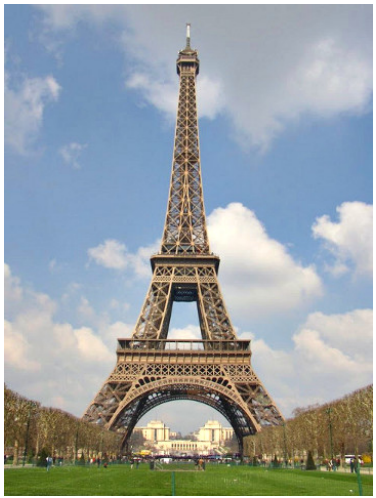
Unsupervised tasks

Associative memory

Heteroassociative:



→



Unsupervised tasks

Associative memory

Autoassociative:

→

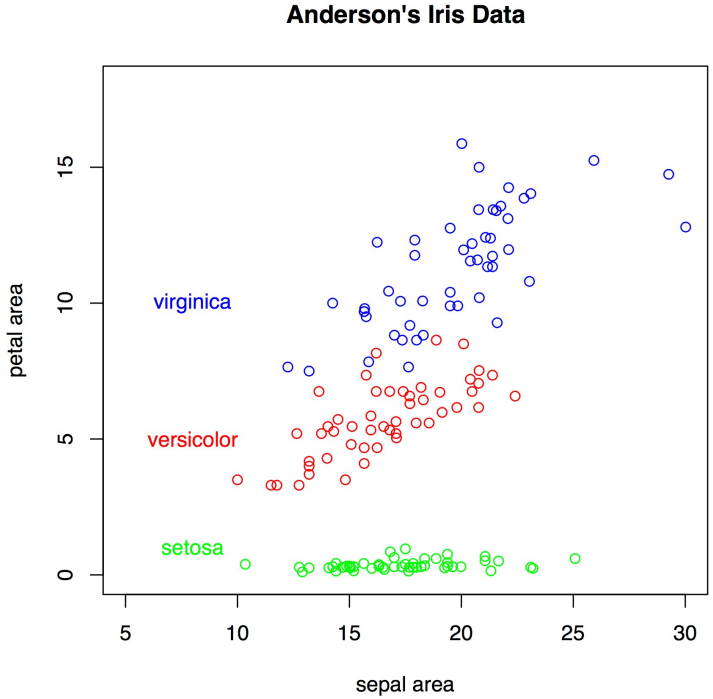
An unsupervised learning problem: Clustering

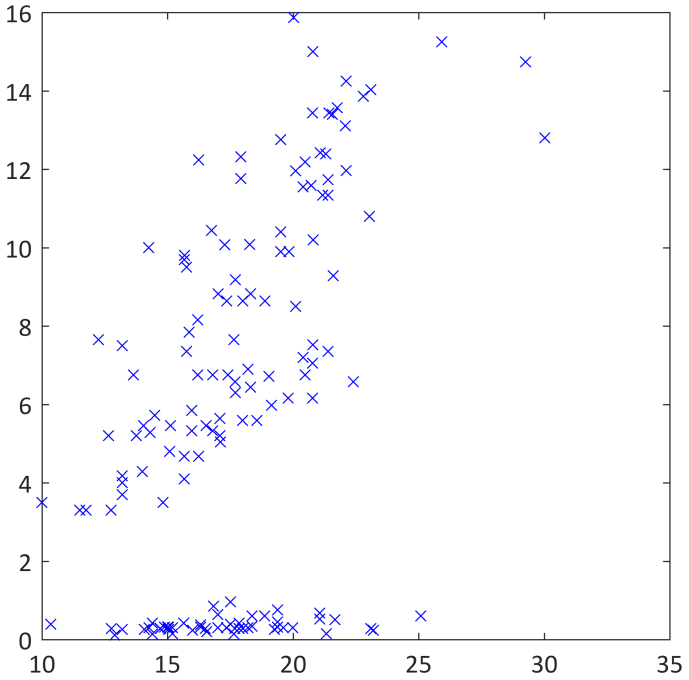
Recall:

We have previously (ml1) put the **clustering** problem in the category

Describing the data (from input to a more compact representation of the input itself)

Clustering = "Finding groups in data"


Anderson's Iris Data
petal area
sepal area
virginica
versicolor
setosa
0
5
10
15
5
10
15
20
25
30


16
14
12
10
8
6
4
2
0
10
15
20
25
30
35

Clusters

What is a group is not uniquely defined

- Sets of mutually similar data (pairwise)
- Sets of locally similar data
- Regions where data are dense separated by regions of low density
- Regions where data are close to a given location (clouds)
- Regions that are distributed according to a given model, e.g., statistical distribution or geometrical shape

A **clustering** is a set of clusters

Crisp (hard) clustering
Each data item belongs 100% to a single cluster

Fuzzy (soft) clustering
Data items belong to different clusters in different proportions (usually $\sum = 100\%$)

Sometimes handy to use **indicator vectors**:

Crisp: `[ 0 0 0 1 0 ]`
Fuzzy: `[0.10 0.20 0.05 .60 0.05 ]`

A little warning

Don't underestimate the difficulty of the clustering task, just because...

- ...*clusters are evident by inspection*
  (they are not! only in 2D, and not always!)

- ...*it is a straightforward problem*
  (it is not! it is ill-defined and there are literally hundreds of possible problem statements)

- ...*it has been studied for decades, so it is a solved problem*
  (new applications with peculiar requirements arise constantly: clustering complex objects –like documents, semantic clustering –like images by content, biclustering, overlapping clusters, big data clustering, non-stationary data...)

Mixture models

Context: **we assume the parametric hypothesis**

Mixture model

$$p(\mathbf{x}) = \sum_{j=1}^{k} p(\mathbf{x}|C_j)P(C_j)$$

C_j j-th mixture component

$p(\mathbf{x}|C_j)$ j-th component density, with its parameters

$P(C_j)$ j-th mixing coefficient

Mixture models

Example of mixture component densities: Gaussians

$$p(\mathbf{x}|C_j) = \frac{1}{\sqrt{(2\pi)^d}|\Sigma_j|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x}-\mathbf{m}_j)^T \Sigma_j^{-1}(\mathbf{x}-\mathbf{m}_j)\right]$$

with $\theta_j = [\mathbf{m}_j, \Sigma_j, P(C_j)]$

k means

- Clusters are represented by **prototypes** or **codevectors** or **centroids**= representative points
- Each centroid has the "centroid property": it is the mean of all points in its cluster (barycenter)
- The number k of clusters (and therefore of centroids) must be specified in advance

Hence the name: k means.

k means algorithm

- Step 0: centroids are initialized
- Step 1: All data points are put in a cluster.
  Criterion: **nearest centroid**
- Step 2: Centroids are adjusted to actually be in the **barycenter (mean) of the cluster they represent**
- Repeat steps 1 and 2 until nothing changes anymore.

k means algorithm

Input: Training set X, number of clusters k

0 Initialize k centroids $\mathbf{y}_1, \ldots, \mathbf{y}_k$, for instance randomly.
Create an indicator vector $\mathbf{u}_l$ for each data point.

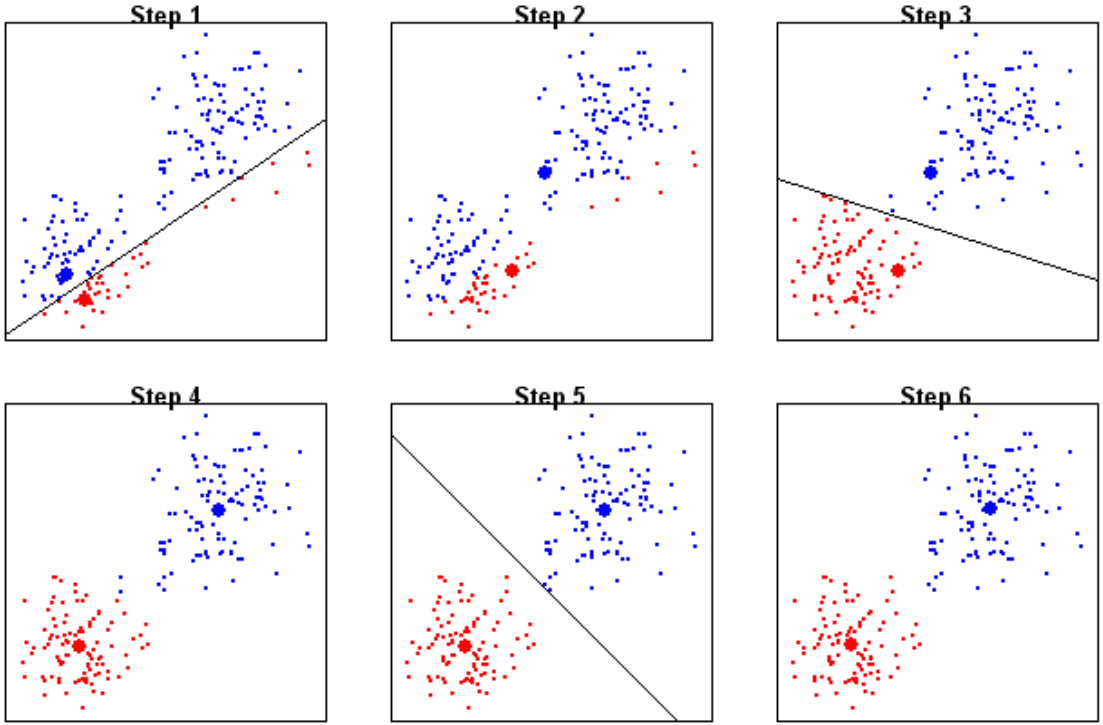
1 For each data point $\mathbf{x}_l$:
  1.1 Compute the distance $d_{lj} = \|\mathbf{x}_l - \mathbf{y}_j\|$ to each centroid $\mathbf{y}_j$
  1.2 Select $j^* = \arg\min_j d_{lj}$
  1.3 Set

$$\mathbf{u}_l = \begin{bmatrix} 0 & 0 & \ldots & 1 & \ldots & 0 & 0 \\ 1 & 2 & \ldots & j^* & \ldots & k-1 & k \end{bmatrix}$$

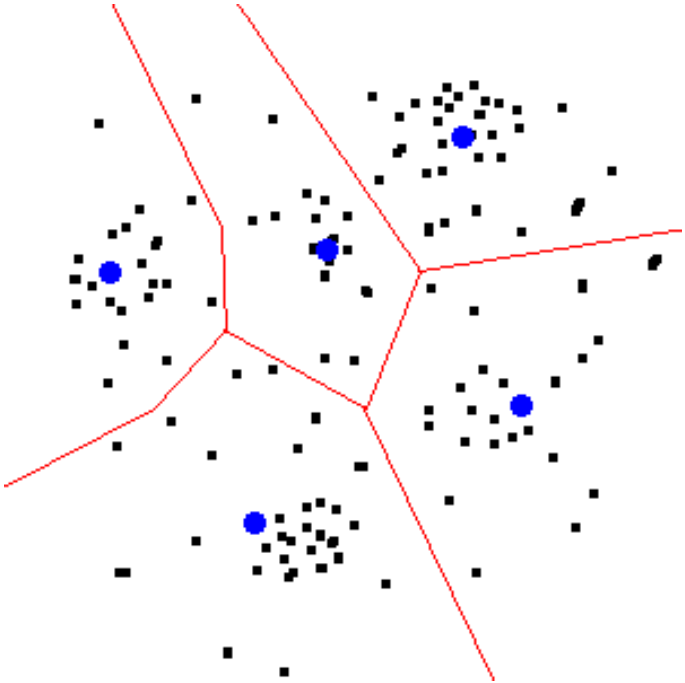
2 For each centroid $\mathbf{y}_j$:
  2.1 Select all data points that have $\mathbf{u}_{l,j} = 1$
  2.2 Move centroid $\mathbf{y}_j$ to their geometric mean

3 If centroids have changed w.r.t. last iteration, goto 1

k means example



Voronoi tessellation



k means objective

$$J = \sum_{l=1}^{n} \|\mathbf{x}_l - \mathbf{y}_{j^*}\|^2$$

where $\mathbf{y}_{j^*}$ is the **nearest centroid** to $\mathbf{x}_l$.

Using indicator vectors

$$J = \sum_{l=1}^{n} \sum_{j=1}^{k} u_{lj} \|\mathbf{x}_l - \mathbf{y}_j\|^2$$

and (in step 2) the centroids can be computed as:

$$\mathbf{y}_j = \frac{\sum_{l=1}^{n} u_{lj} \mathbf{x}_l}{\sum_{i=1}^{n} u_{ij}}$$

Pros/cons of k means

Pros:

- Very simple
- Guaranteed not to diverge; usually it converges quickly
- Works with high-dimensional data

Cons:

- Local optimization only: J is not convex, many local minima
- Always finds k clusters (even if they are not there)