Statistical Machine Learning

UoC Stats 37700, Winter quarter

Lecture 1: Introduction. Decision Trees.

What is “machine learning”?

“Machine Learning” is traditionally classified as part of computer science.
A short history note:

- 1940’s, 1950’s : first computers are created; early on, strong belief that “one day computers will be intelligent” (e.g.: Alan Turing’s “imitation game”, a.k.a. Turing test, 1950). Mathematical formalism grounded on logic and symbolic calculus.
- 1960’s, 1970’s : development of symbolic-reasoning artificial intelligence based on formalism and rule inference. Rules are “learned” from data but the statistical analysis is almost inexistant. The results obtained in practice fall short of the initial expectations and stall.

- 1980's : development of artificial neural networks: a clear departure from symbolic-based AI (early version: Rosenblatt's perceptron 1957) that brings forth some successes. The ambitions are more modest.
- 1990-2000's : development of statistical learning methods: decision trees, kernel methods. . . The mathematical formalism of these methods is now more firmly grounded in probability, statistics, information theory and analysis (e.g. optimization).
- Now: a certain inching towards more ambitious goals. . .

A limited goal: classification

Typical machine learning problem: classification.
The task is to classify an unknown object $x \in \mathcal{X}$ into one category of a certain set $\mathcal{Y} = \{1, 2, \ldots c\}$ (labels).
Examples:

- (Handwritten) character recognition: x is a grey scale image, $\mathcal{Y}$ is the list of possible characters or digits.
- Medical diagnosis: x is a set of medical observations (numerical or categorical), $\mathcal{Y} = \{\text{benign}, \text{malignant}\}$ (for example).
- Recognition of coding/non-coding gene sequences
- Junk e-mail automatic sorting

Supervised and unsupervised learning

- The “learning” stage is to construct (in an automatic way) such a classification method from a known set of examples whose class is already known: training sample.
- “Unsupervised learning” : no labels are available from the training sample. We want to extract some relevant information, for example a separation into clusters (a kind of classification without pre-defined classes).

Some formalization

- A classifier is a function $f : \mathcal{X} \to \mathcal{Y}$.
- The "training sample" is $S = ((X_1, Y_1), \ldots, (X_n, Y_n))$.
- A learning method is a mapping $S \mapsto (\widehat{f} : \mathcal{X} \to \mathcal{Y})$.
- How can we (theoretically) assess if $\widehat{f}$ is a good classifier?
- Test it on new examples.

- Probabilistic framework: the performance of the classifier is theoretically measured by the average percentage of classification errors commited on a unknwown 'test' object (X, Y) drawn at random:

$$\mathcal{E}(\widehat{f}) = \mathbb{E}_{(X,Y)\sim P}\left[\mathbb{1}\{\widehat{f}(X) \neq Y\}\right] = \mathbb{P}\left[\widehat{f}(X) \neq Y\right].$$

  This is called the generalization error.
- The learning from examples makes sense if we assume that the sample S constains *some* information on the test objects.
- Simplest assumption: $S = ((X_i, Y_i)_{1\leq i\leq n})$ are drawn from the same distribution P, independently.

Machine learning = statistics?

Obviously, with this formalism machine learning is very close to traditional statistics:

Classification	$\rightarrow$	Regression
Unsupervised learning	$\rightarrow$	Density estimation

Emphasis of machine learning on:

- complex data: high dimensional, non-numerical, structured
- very few modelling assumptions on the distribution of the data.
- non-parametric methods coming from various sources of inspiration.

The Bayes classifier

- Assuming the probabilistic framework: (X, Y) drawn according to a distribution P, what is the best possible classifier possible?
- Represent P under the form

$$P(x, y) = P(X = x)P(Y = x|X = x) = \mu(x)\eta(Y = y|X = x)\,.$$

- For any fixed x, the best possible deterministic classification for x is to output the class having the largest conditional probability given $X = x$:

$$f^*(x) = \underset{y \in \mathcal{Y}}{\text{Arg Max}}\, \eta(Y = y|X = x)\,.$$

This is the Bayes classifier; the Bayes error is

$$L^* := \mathbb{E}\left[f^*\right] = \mathbb{E}\left[\mathbb{1}\{f^*(X) \neq Y\}\right]\,.$$

- Important: the Bayes classifier f^* is entirely determined by the function η only.

Plug-in rules

- One way to construct a classifier is therefore to estimate the function η by $\widehat{\eta}$.
- If we know an estimator $\widehat{\eta}$ it is natural to consider the classifier

$$\widehat{f}(x) = \underset{y \in Y}{\text{Arg Max}}\, \widehat{\eta}(Y = y | X = x)$$

This is called a plug-in classifier.

Quality of plug-in rules

We can relate the performance of estimator $\widehat{\eta}$ to the peformance of the corresponding plug-in rule:

$$\begin{aligned}\mathcal{E}(\widehat{f}) - \mathcal{E}(f^*) &\leq \mathbb{E}\left[\mathbb{1}\{\widehat{f}(x) \neq f^*(x)\} \sum_y |\eta(y|x) - \widehat{\eta}(y|x)|\right] \\ &\leq \mathbb{E}\left[\sum_y |\eta(y|x) - \widehat{\eta}(y|x)|\right] .\end{aligned}$$

In the binary classification case:

$$\mathcal{E}(\widehat{f}) - \mathcal{E}(f^*) = \mathbb{E}\left[\mathbb{1}\{\widehat{f}(x) \neq f^*(x)\} |2\eta(y=1|x) - 1|\right] \leq 2 \|\eta - \widehat{\eta}\|_1 .$$

Logistic regression for η

- If $\mathcal{Y} = \{0, 1\}$ (binary classification), there is one classical way do to estimate η: logistic regression: estimate instead

$$\gamma(x) = \text{logit}(\eta(1|x)) = \log \frac{\eta(1|x)}{\eta(0|x)},$$

  the log-odds ratio.
- Advantage: $\eta \in [0, 1]$ but $\gamma \in \mathbb{R}$, therefore you can apply your favorite regression method to estimate γ.

Class density estimation

- Another classical way to go: estimate separately the density of each class

$$g_y(x)dx = dP(X = x|Y = y)$$

- Generally model-based; for example, each class is modelled by a mixture of Gaussians:

$$g_y(x) = \sum_{i=1}^{m_y} p_{y,i}\phi\left(\mu_{y,i}, \Sigma_{y,i}\right) ,$$

  which can be estimated via the EM algorithm for example .
- Then estimate the marginal probabilities of each class:

$$c_y = P(Y = y);$$

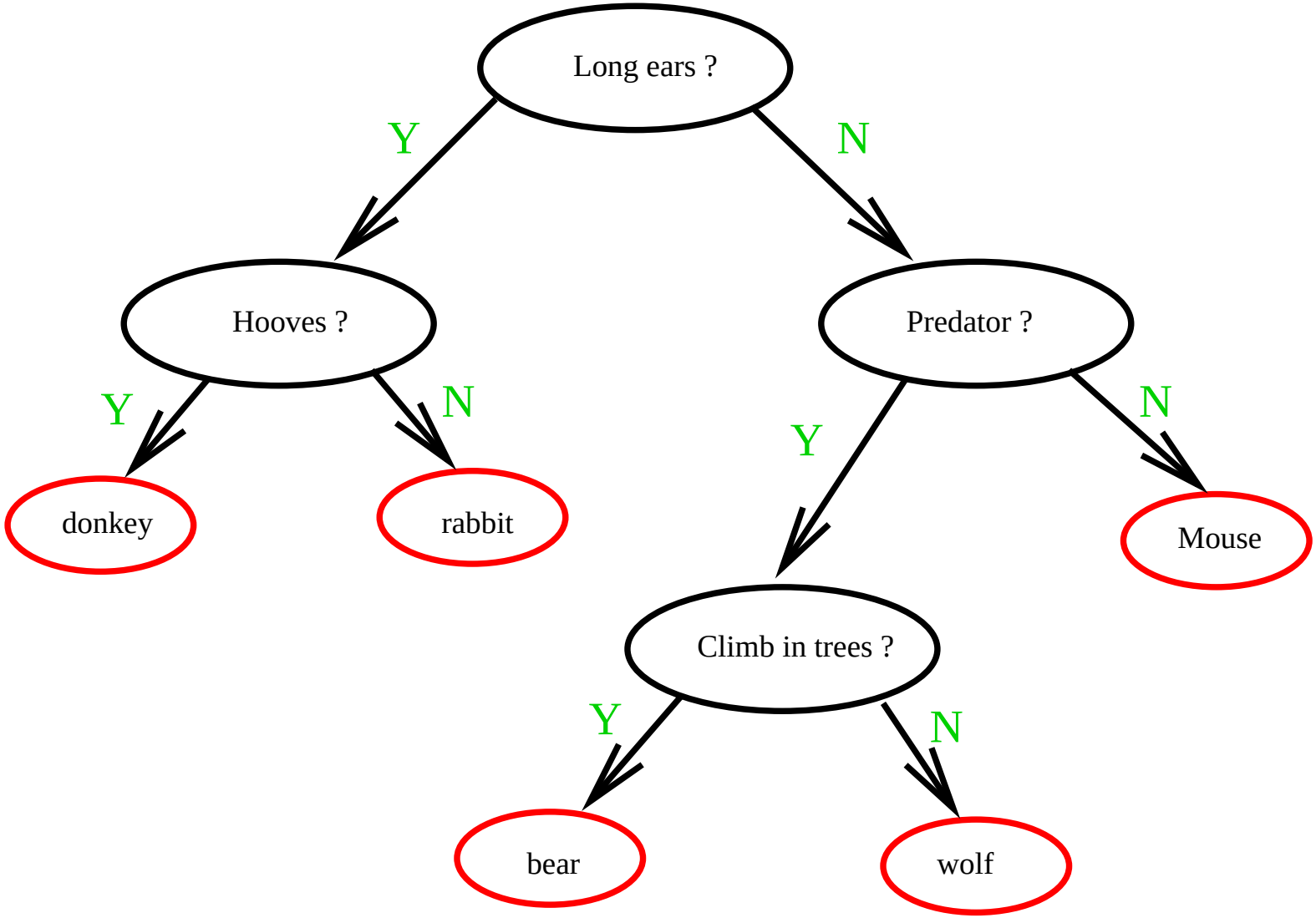
  if the above are estimated by $\widehat{g}$ and $\widehat{c}$, the plug-in rule is by definition

$$\widehat{f}(x) = \underset{y\in\mathcal{Y}}{\text{Arg Max}}\, \widehat{c}_y \widehat{g}_y(x)$$

Decision trees

- Decision trees are a way of defining classifiers that are somehow descendants of rule-based classification methods from symbolic approaches to ML.
- Can be used for classification and for regression.
- Different variants: CART (Breiman, Friedman, Olshen and Stone), C4.5, ID3 (Quinlan)
- Not the best method available nowadays in terms of generalization error, but still very popular because it provides the user with an “explanation” of the decision rule.

The shape of a decision tree: the “20 questions” game:



Formally, a decision tree is:

- a binary tree
- whose internal nodes are labeled by questions $q \in \mathcal{Q}$;
- whose terminal nodes are labeled by the decision (e.g. for classification: some $y \in \mathcal{Y}$).

- Formally, a question is a function $q : \mathcal{X} \to \{left, right\}$.
- Note: so, questions can be identified with (binary) classifiers. . . A decision tree is a way to combine "elementary" classifiers.
- Standard choice of questions: when x is a collection of numerical and/or categorical data,

$$x = (x_1, \ldots, x_k), \text{ with } x_i \in \mathbb{R} \text{ or } x_i \in \mathcal{C}_i = \left\{C_{i,1}, \ldots, C_{i,n_i}\right\},$$

consider questions of the form

$$q(x) = \mathbb{1}\{x_i > t\} \text{ if } x_i \text{ is numerical}$$

(where t is some real threshold) or

$$q(x) = \mathbb{1}\{x_i \in C\} \text{ if } x_i \text{ is categorical}$$

(where C is some subset of $\mathcal{C}_i$).

Choosing the decision when the tree structure is fixed

- Assume for now that the tree structure and the questions are fixed.
- What is the best decision to take on each of the leaves? (based on the available data)
- Let the training data “fall down the tree” and for each leaf, decide to pick the majority class among those datapoints having reached that leaf.

Growing a decision tree

Now, how can we choose the structure of the tree itself and the questions?

- Assume we want to build a decision tree of size k (the size is the number of leaves)
- One standard way to choose a classifier belonging to a certain set $\mathcal{F}$ (here trees of size k) is to find the minimizer of the training (or empirical) classification error:

$$\widehat{f} = \operatorname*{Arg\,Min}_{f \in \mathcal{F}} \widehat{\mathcal{E}}(S, f) = \operatorname*{Arg\,Min}_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}\{f(X_i) \neq Y_i\}$$

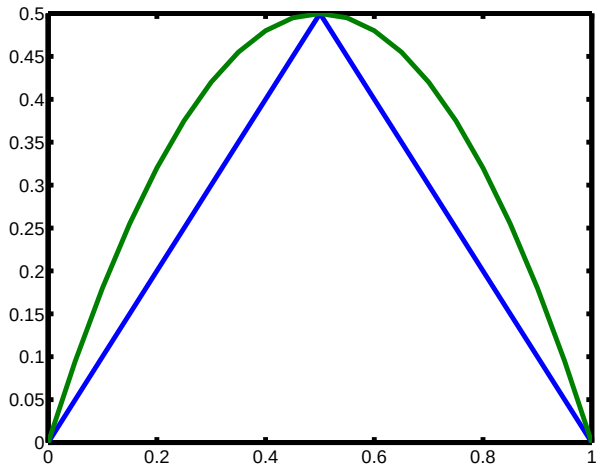
  This is known as Empirical Risk Minimization (ERM).
- Unfortunately, in the case of trees this is intractable from a practical point of view.

Greedy growing

- An alternative to global optimization: “greedy” construction:
  - Start with a tree reduced to its root (a constant classifier!)
  - Choose the question that results in the largest reduction in the empirical risk when splitting the root into two sub-nodes.
  - Iterate this procedure for splitting the sub-nodes in turn.
- Unfortunately, in the case of classification, there arises a new problem: it can happen that no available question leads to any “local” improvement.

Impurity functions

- As a function of the (estimated) class probabilities (p_i), the (training) error of a locally constant classifier is piecewise linear; this is the source of the latter problem.
- Idea: replace this by a strictly convex "impurity function" $I((p_i))$:



- Once an impurity function I has been chosen, the "greedy" criterion to choose a question is to find the minimum of

$$N_{left} I((p_{i,left})) + N_{right} I((p_{i,right}))\,;$$

then strict convexity of I implies that this is always strictly smaller than

$$N_{tot} I((p_{i,tot}))$$

whenever $(p_{i,left}) \neq (p_{i,right})$.

- Classical choices for I: (1) Entropy

$$H((p_i)) = -\sum_i p_i \log p_i$$

(2) Gini criterion

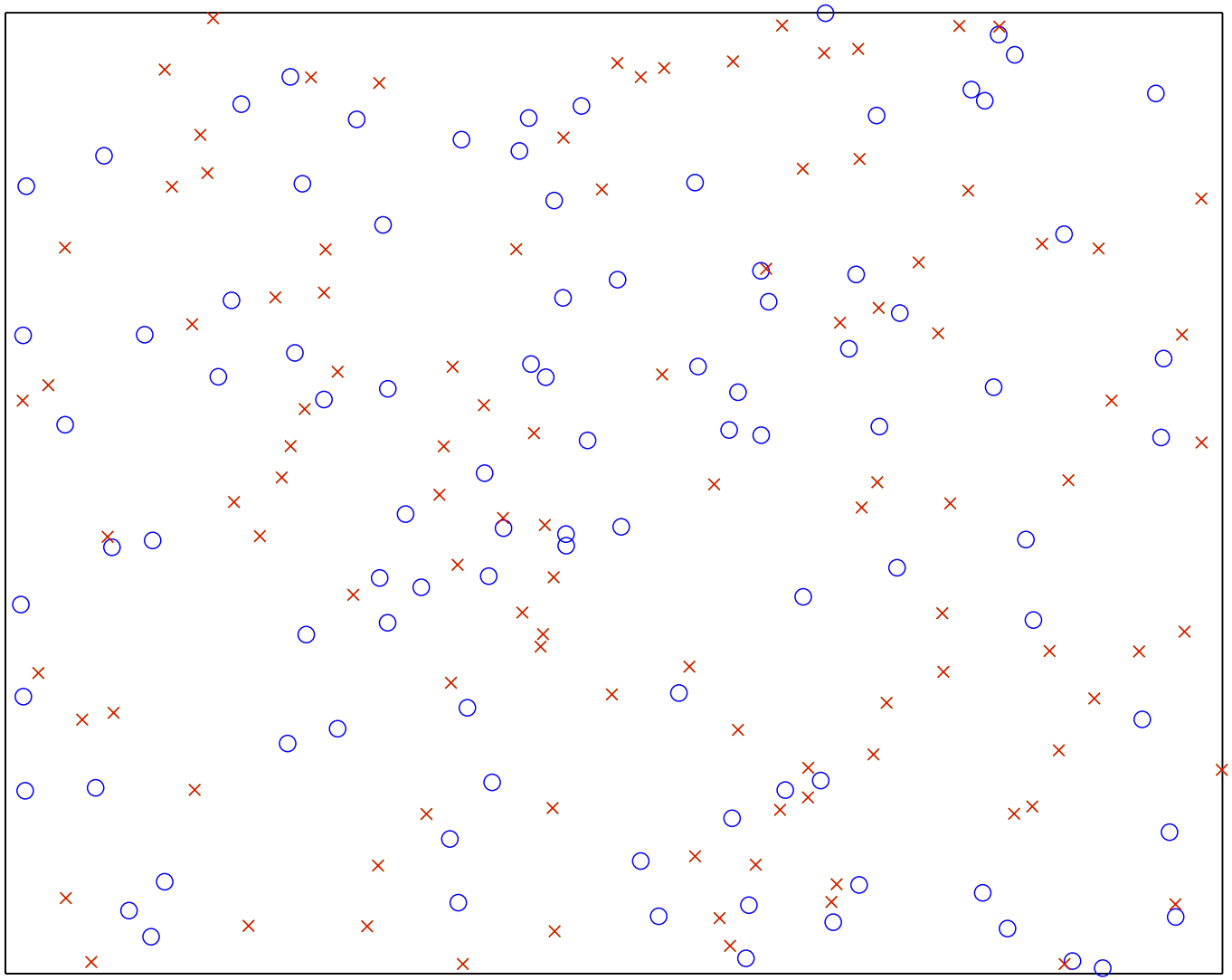
$$G((p_i)) = -\sum_i p_i^2$$

- Note: a classification tree can be seen as a plug-in rule wherein the function η is estimated by a constant function on the leaves of the trees.
- To this regard the entropy criterion is the natural cost function when estimating η on such a model via Maximum Likelihood ("greedy" maximum likelihood).
- Similarly considering the Gini criterion is equivalent to locally minimizing a kind of least square error:

$$\ell(\eta, x, y) = \sum_{y' \in \mathcal{Y}} (\mathbb{1}\{y = y'\} - \eta(x, y'))^2 .$$

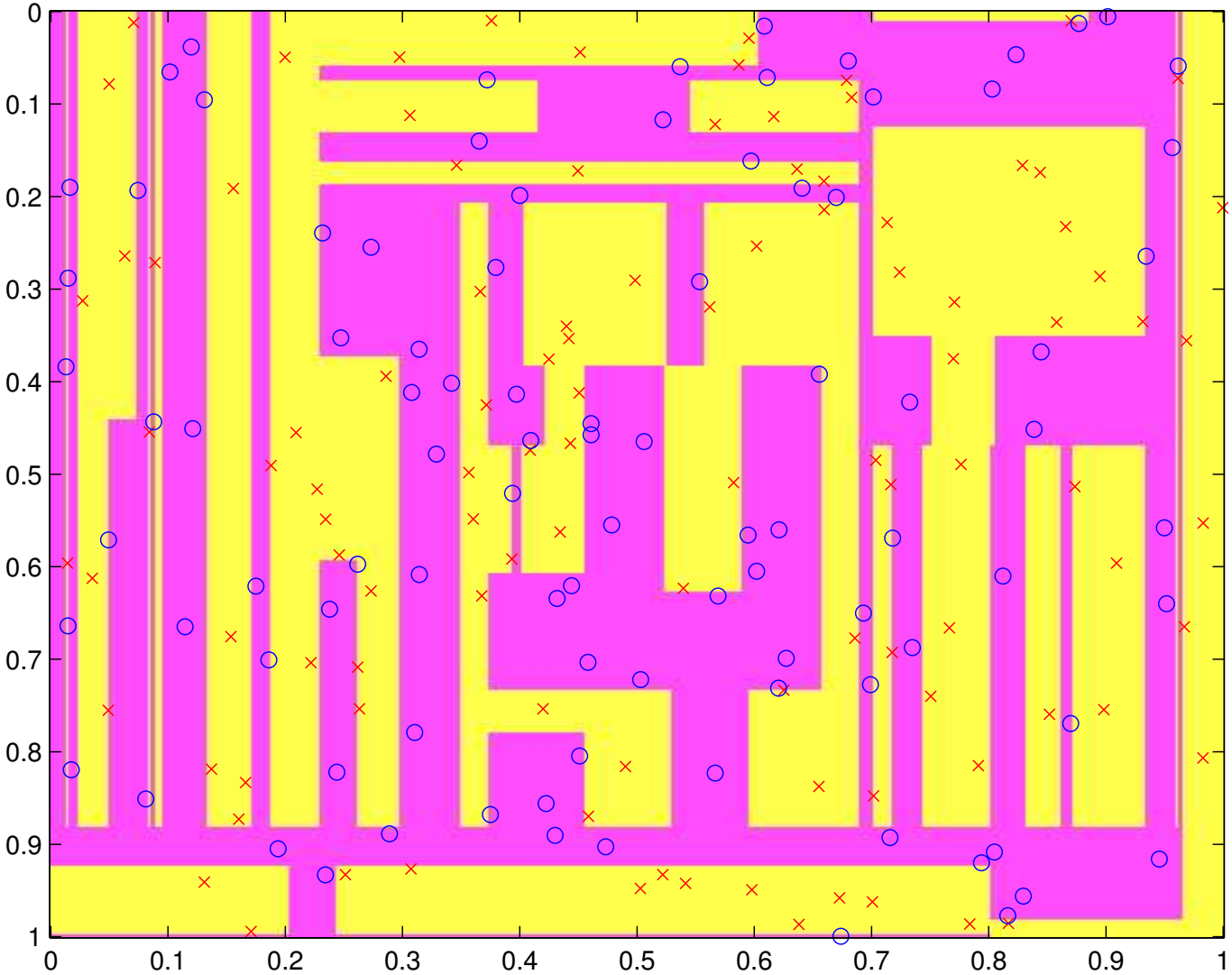
- We now have a reasonable way to construct a tree structure recursively.
- But when should we stop growing the tree?
- We could keep growing the tree until each leaf only contains data points of one single class. . .
- Unfortunately this is not a very good idea: why?

Overfitting



A classification problem (here totally random data).

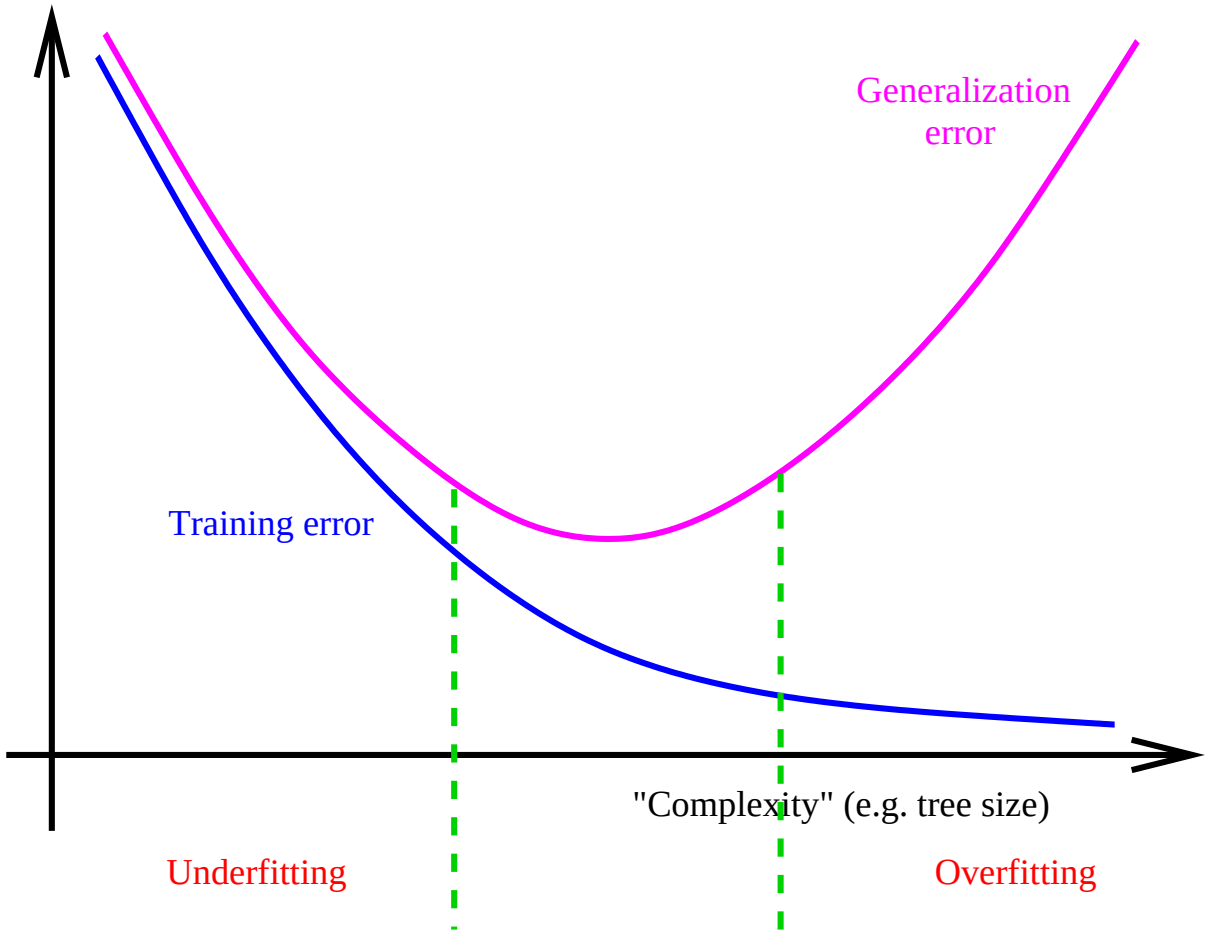
Overfitting



Output of maximally grown decision tree.

Underfitting and overfitting

Informally, there is a tradeoff to be found between the “complexity” of a classifier and the amount of data available.



- One first idea: stop if a split leads to a leafs containing less than ℓ datapoints.
- This might not be the best idea.
- More interesting idea: complexity regularization: find a tradeoff between the empirical risk $\widehat{\mathcal{E}}(S, f)$ and the "complexity" (tree size) of f .
- Grow a tree $\mathcal{T}$ of maximal size using the greedy procedure and select a sub-tree $T \subset \mathcal{T}$ optimizing the following regularized error:

$$\operatorname*{Arg\,Min}_{T \subset \mathcal{T}} \widehat{\mathcal{E}}(S, T) + \lambda |T| := R_\lambda(T)\,;$$

  this is called pruning.
- Note: λ has to be chosen, too! But let us assume for now that it is fixed.

- Interestingly, when the "maximal" tree $\mathcal{T}$ is fixed the problem of finding the optimal subtree minimizing the previous regularize criterion is tractable.
- If $\widehat{T}_\lambda$ denotes the pruned tree for a fixed λ, we have

$$R_\lambda(\widehat{T_\lambda}) = \min(R_\lambda(T_{root}), R_\lambda(\widehat{T}_{left,\lambda}) + R_\lambda(\widehat{T}_{right,\lambda}))\,,$$

  then use recursive principle.
- Furthermore, then

$$\lambda_1 \geq \lambda_2 \Rightarrow \widehat{T}_{\lambda_1} \subset \widehat{T}_{\lambda_2}\,.$$

- Hence, as λ grows from 0 to $+\infty$, we have a decreasing sequence of pruned trees

$$\mathcal{T} = \widehat{T}_0 \supset \widehat{T}_{\lambda_1} \supset \ldots \supset T_{root}\,,$$

  which is easily computable.