Decision Trees and ensemble methods

Pietro Gori

Associate Professor
Equipe IMAGES - Télécom Paris
pietro.gori@telecom-paris.fr

Summary

Summary

Supervised Learning - Probabilistic framework

X: input data x_i^j, random variable in $\mathcal{X} = \mathbb{R}^p$ with $i = 1, \dots, n$ and $j = 1, \dots, p$ where n and p are the number of observations and variables respectively

Y: response (to predict), random variable in $\mathcal{Y} = \{C_1, \dots, C_K\}$ (classification with K classes) or $\mathcal{Y} = \mathbb{R}$ (regression)

P: joint probability distribution of (X, Y), fixed but unknown

$\mathcal{D}_n = \{(\mathbf{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}\}$: i.i.d.samples drawn from P

$\mathcal{F}$: collection of classifiers $f \in \mathcal{F}$

$\mathcal{L}$: loss function which measures the error of the classifier/model

- Examples (classification) : $\mathcal{L}(\mathbf{x}, y, f(\mathbf{x})) = \begin{cases} 1, & \text{si } f(\mathbf{x}) \neq y, \\ 0, & \text{sinon.} \end{cases}$
- Example (regression): $\mathcal{L}(\mathbf{x}, y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2$

Goal: estimate from $\mathcal{D}_n$ the function $f \in \mathcal{F}$ which minimizes the risk (cost) function $R(f) = \mathbb{E}_P[\mathcal{L}(X, Y, f(X))]$

Estimate a classifier

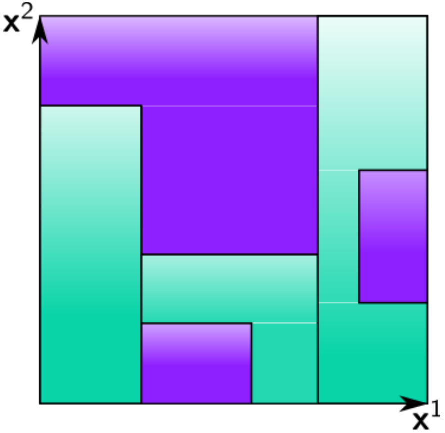
We need to define:

- **input and output data space** ($\mathcal{X}$ $\mathcal{Y}$)
- **type of classifier** ($\mathcal{F}$)
- **cost function** ($\mathcal{L}$) to minimize for finding the best f
- **minimization algorithm** for $\mathcal{L}$
- **method for model selection** to estimate hyper-parameters
- **method to evaluate performance**

Summary

Constant piecewise model/classifier

$\mathcal{F}$ belongs to the set of **constant piecewise functions**. We divide the input space $\mathcal{X}$ in M disjoint partitions $\mathcal{C}_m$: $\mathcal{X} = \mathcal{C}_1 \sqcup \dots \sqcup \mathcal{C}_M$. To simplify things, we assume that the separation lines are parallel to the coordinate axes:



Constant piecewise model/classifier

Let $\mathbf{x} = (\mathbf{x}^1, \dots, \mathbf{x}^p)$, we model the function f with a different constant value α_m in every partition $\mathcal{C}_m$: $f(\mathbf{x}) = \sum_{m=1}^{M} \alpha_m \mathbb{1}_{\mathbf{x} \in \mathcal{C}_m}$, where $\mathbb{1}_{\mathbf{x} \in \mathcal{C}_m}$ is equal to 1 if $\mathbf{x} \in \mathcal{C}_m$ and 0 otherwise.

- **Regression**: If we use the L2-norm ($\sum_i (y_i - f(\mathbf{x}_i))^2$), then the best α_m^* is simply the average of the y_i within the region $\mathcal{C}_m$: $\alpha_m^* = \frac{1}{|\mathcal{C}_m|} \sum_{\mathbf{x}_i \in \mathcal{C}_m} y_i$, where $|\mathcal{C}_m|$ is the number of elements within $\mathcal{C}_m$
- **Classification**: We define the proportion of observations belonging to class k within region $\mathcal{C}_m$ as $\rho_{mk} = \frac{1}{|\mathcal{C}_m|} \sum_{\mathbf{x}_i \in \mathcal{C}_m} \mathbb{1}_{y_i = k}$. The best α_m^* is then equal to the majority class: $\alpha_m^* = \arg\max_k \rho_{mk}$

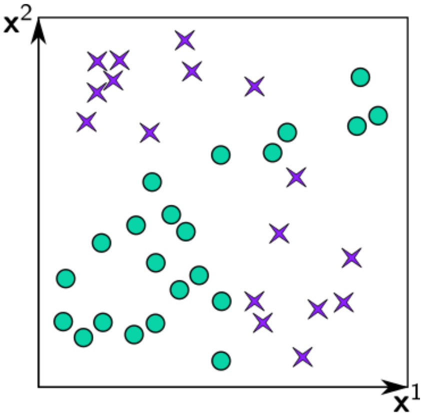
Constant piecewise model/classifier

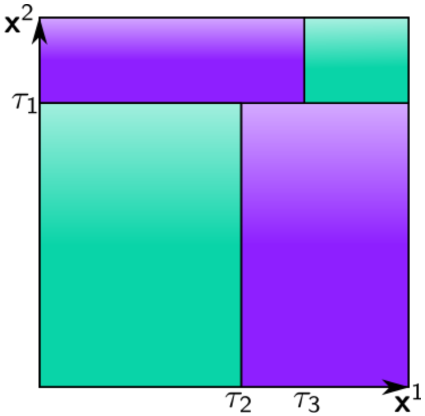
- Motivation: easy to interpret
- Limitations:
  - regions are difficult to describe
  - If the partition is fixed beforehand, many regions might end up being empty
- Possible solution: learn the partitions from the data ! How to avoid curse of dimensionality ?

Summary

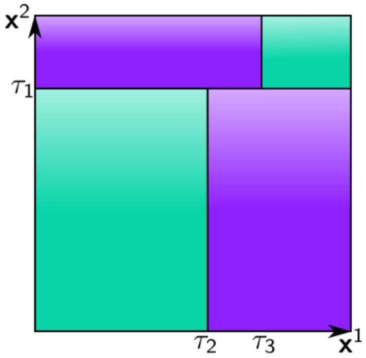
Decision trees

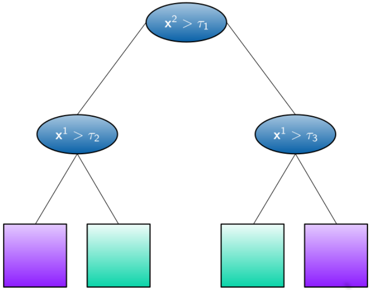
Presented by Breiman *et al.* in 1984 under the name of CART: Classification and Regression Trees.





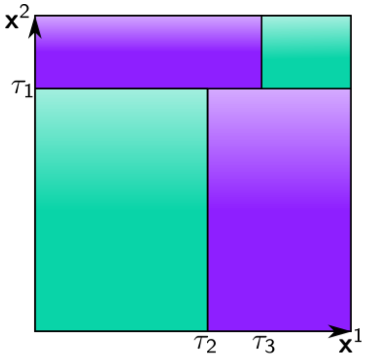
Decision trees

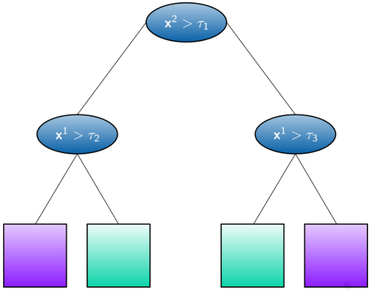




First idea:
Use several hyperplanes (and not only one) to build non linear decision boundaries.

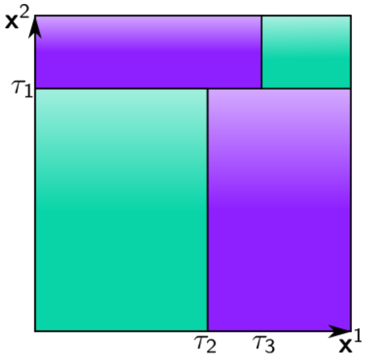
Decision trees

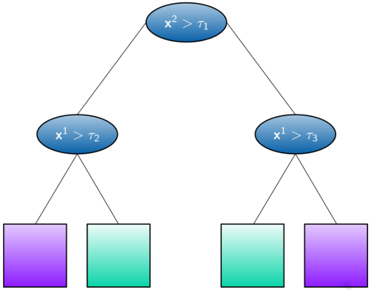




Secod idea:
Use separation lines orthogonal to the coordinate axis, *i.e.*, hyperplanes $\{\mathbf{x} \in \mathcal{X} : \mathbf{x}^j = \tau\}$ to ease interpretation.

Decision trees





Third idea:
Use recursive binary decision trees: The full data-set sits at the top of the tree. Every node (junction) is associated to a separating hyperplane $\{\mathbf{x} \in \mathcal{X} : \mathbf{x}^j = \tau\}$. The terminal nodes correspond to the regions.

Linear partition orthogonal to the axes

- We use the same model as before: $f(\mathbf{x}) = \sum_{m=1}^{M} \alpha_m \mathbb{1}_{\mathbf{x} \in \mathcal{C}_m}$, namely we assign a constant value at every region
- Let $\mathbf{x} = (\mathbf{x}^1, \ldots, \mathbf{x}^p)$ with p variables. We define the split $t_{j,\tau}(\mathbf{x})$ along the direction $\mathbf{x}^j$ with threshold τ as:

$$t_{j,\tau}(\mathbf{x}) = \text{sign}(\mathbf{x}^j - \tau) = \begin{cases} +1, & \text{si } \mathbf{x}^j > \tau \\ -1, & \text{si } \mathbf{x}^j < \tau \end{cases} \tag{1}$$

- We assume for now that we have already defined a *loss function* $\mathcal{L}$ and a *stopping criteria*

Efficient recursive algorithm

For a binary tree:

1. Given $\mathcal{D}_n$, the entire data-set is the root node
2. Look for the best separator $t_{j,\tau}$ on $\mathcal{D}_n$ such that the local cost function $\mathcal{L}(t_{j,\tau}, \mathcal{D}_n)$ is minimal. This means looking for the "best" direction j and threshold τ that splits $\mathcal{D}_n$ into $\mathcal{D}_n^d$ and $\mathcal{D}_n^g$.
3. Be careful: the splitting values τ are not infinite ! They depend on the data $\mathcal{D}_n$. Hence, we can scan all inputs $\mathbf{x}$ and quickly find the best j and τ
4. Split $\mathcal{D}_n$ to $\mathcal{D}_n^d$ and $\mathcal{D}_n^g$ using the estimated separator hyperplane.
5. It results two nodes, a left ($\mathcal{D}_n^g$) and a right ($\mathcal{D}_n^d$) one
6. Evaluate the stopping criteria for the right node, if it is verified, the nodes becomes a terminal node, otherwise go to 3 using $\mathcal{D}_n^d$ as input space
7. Evaluate the stopping criteria for the left node, if it is verified, the nodes becomes a terminal node, otherwise go to 3 using $\mathcal{D}_n^g$ as input space

Examples

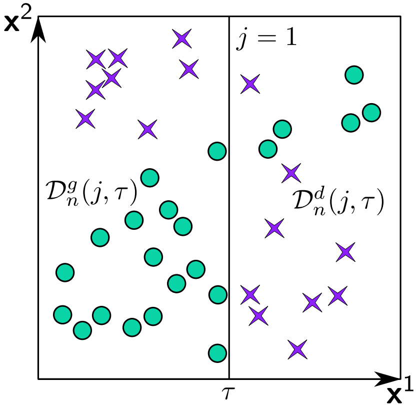
Given the input data-set $\mathcal{D}_n$ and a binary separator $t_{j,\tau}$, we have

$$\mathcal{D}_n^d(j,\tau) = \{(\mathbf{x}, y) \in \mathcal{D}_n, t_{j,\tau}(\mathbf{x}) > 0\}$$
$$\mathcal{D}_n^g(j,\tau) = \{(\mathbf{x}, y) \in \mathcal{D}_n, t_{j,\tau}(\mathbf{x}) \leq 0\}$$

Examples

Given the input data-set $\mathcal{D}_n$ and a binary separator $t_{j,\tau}$, we have

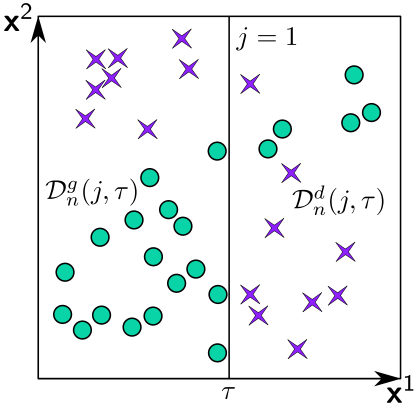
$$\mathcal{D}_n^d(j,\tau) = \{(\mathbf{x},y) \in \mathcal{D}_n, t_{j,\tau}(\mathbf{x}) > 0\}$$
$$\mathcal{D}_n^g(j,\tau) = \{(\mathbf{x},y) \in \mathcal{D}_n, t_{j,\tau}(\mathbf{x}) \leq 0\}$$

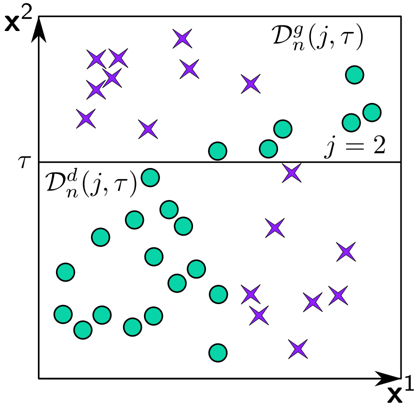


Examples

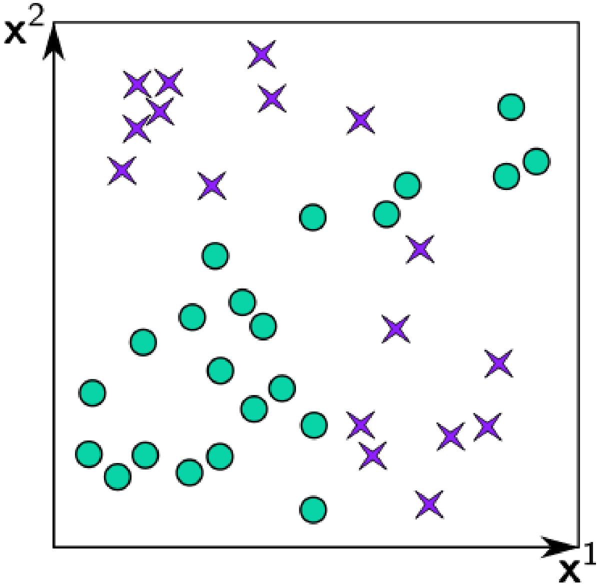
Given the input data-set $\mathcal{D}_n$ and a binary separator $t_{j,\tau}$, we have

$$\mathcal{D}_n^d(j,\tau) = \{(\mathbf{x}, y) \in \mathcal{D}_n, t_{j,\tau}(\mathbf{x}) > 0\}$$
$$\mathcal{D}_n^g(j,\tau) = \{(\mathbf{x}, y) \in \mathcal{D}_n, t_{j,\tau}(\mathbf{x}) \leq 0\}$$

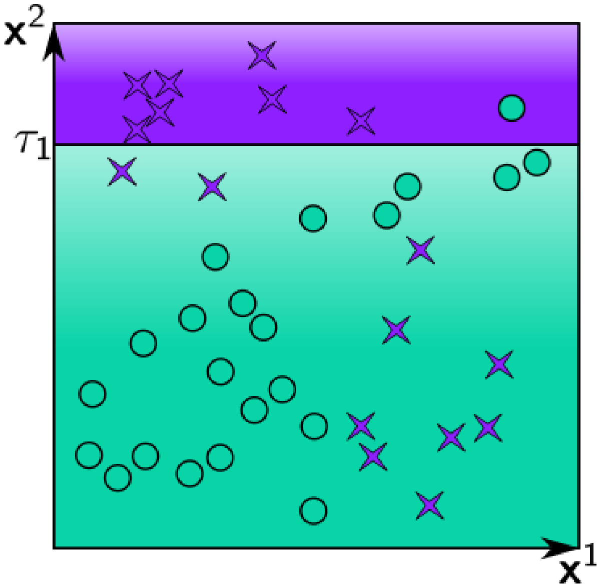




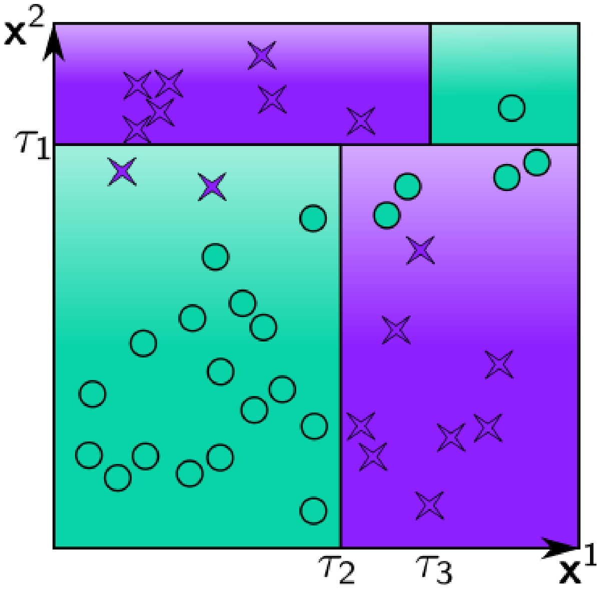
Example



Example



Example



Greedy method

- The presented algorithm (called CART) is greedy
- We do not optimize a global criterion. Instead, we locally look for an optimal separator (with respect to L), which means at every direction j independently.
- Why not using a global criterion ? Why not looking for the best sequence of separators ?

Greedy method

- The presented algorithm (called CART) is greedy
- We do not optimize a global criterion. Instead, we locally look for an optimal separator (with respect to L), which means at every direction j independently.
- Why not using a global criterion ? Why not looking for the best sequence of separators ? Because it is a NP-hard problem !

Summary

Cost function - Classification

- We recall that for a classification tree, given the input data $\mathcal{D}_n$ divided in K classes, we define the proportion of observations belonging to class k as: $\rho_k(\mathcal{D}_n) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}(y_i = k)$
- Note that $\rho(\mathcal{D}_n) = (\rho_1(\mathcal{D}_n), \dots, \rho_K(\mathcal{D}_n))^\top \in \Delta_{K-1}$ where $\Delta_{K-1} := \left\{ \rho_k \in \mathbb{R}^K : \sum_{k=1}^{K} \rho_k = 1 \text{ and } \forall k \in [\![1, K]\!], \rho_k \geq 0 \right\}$ is the (K-1)-simplex. A simplex can be seen as the smallest convex set containing the given vertices, or the convex hull of the K vertices. *Example*: a 2-simplex is a triangle.

Cost function - Classification

Among all parameters $(j, \tau) \in \{1, \dots, p\} \times \{\tau_1, \dots, \tau_m\}$, we look for j^* and τ^* which minimizes the following cost function:

$$L(t_{j,\tau}, \mathcal{D}_n) = \frac{n_g}{n} H\left(\rho(\mathcal{D}_n^g(j,\tau))\right) + \frac{n_d}{n} H\left(\rho(\mathcal{D}_n^d(j,\tau))\right)$$
$$\text{avec} \quad n_g = |\mathcal{D}_n^g(j,\tau)| \quad \text{et} \quad n_d = |\mathcal{D}_n^d(j,\tau)|$$

- H is an "impurity" function that evaluate the splitting. Pure means a node with observations from the same class. We want to **minimize** H in order to have pure nodes.
- The total cost is the sum of the impurity of each child node ($\mathcal{D}_n^g$ and $\mathcal{D}_n^d$) weighted by the proportion of its observations (n_g and n_d)
- We evaluate a finite number of thresholds (max n)

Summary

Impurity function

Definition: impurity function

An impurity function $H : [0:1]^K \rightarrow \mathbb{R}$ is a function defined on Δ_{K-1} for which the following properties hold:

1. H becomes maximum at points $(\frac{1}{K}, \dots, \frac{1}{K})^\top$, *i.e.,* all ρ_k are equal
2. H becomes minimum at points $(1, 0, \dots, 0)^\top, (0, 1, 0, \dots, 0)^\top, \dots, (0, \dots, 0, 1)^\top$, *i.e.,* the probability of being in a certain class is 1 and 0 for all other classes. These are the vertices of Δ_{K-1}.
3. H is symmetric with respect to its arguments $\rho_1, \dots, \rho_K$, *i.e.,* even if we permute ρ_j, H does not change

Impurity function: binary case ($K = 2$)

When we have only two classes ($K = 2$):

- Δ_{K-1} is the line segment joining $(1, 0)$ and $(0, 1)$ in $\mathbb{R}^2$
- H becomes maximum at $(\frac{1}{2}, \frac{1}{2})$
- H becomes minimum at $(0, 1)$ or $(1, 0)$, which means when all observations in one region belong to the same class.

Misclassification error

Given the data of a node $\mathcal{D}_n$ (it might be the root node or a child node), we assign the observations in $\mathcal{D}_n$ to the majority class k^*:

$$k^* = \arg\max_{k=1,\dots,K} \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}_n} \mathbb{1}(y_i = k)$$

Then we define:

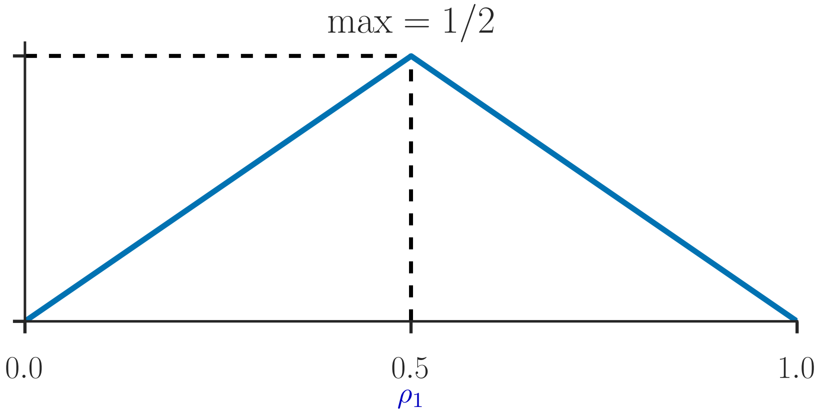
Misclassification error: $H_{\text{mis}}(\mathcal{D}_n) = 1 - \rho_{k^*}(\mathcal{D}_n)$,

This is the error that we commit by assigning the observations to the class k^*. Remember that $\sum_{k=1}^{K} \rho_k = 1$.

Misclassification error

When the number of classes K is 2

$$H_{\text{mis}}(\mathcal{D}_n) = 1 - \max_{k=1,2} \rho_k(\mathcal{D}_n) = \min(\rho_1(\mathcal{D}_n), 1 - \rho_1(\mathcal{D}_n))$$

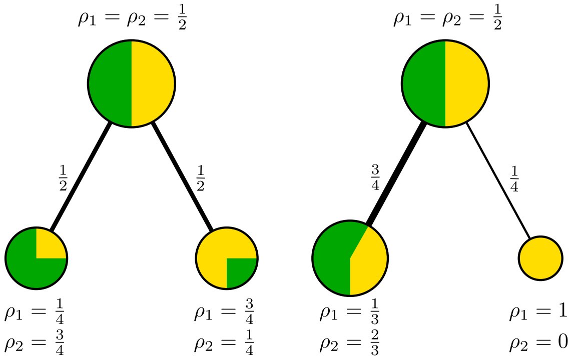


Limitations of the misclassification error

- Remember that the cost of a split is:

$$L(t_{j,\tau},\mathcal{D}_n) = \frac{n_g}{n} H\left(\rho(\mathcal{D}_n^g(j,\tau))\right) + \frac{n_d}{n} H\left(\rho(\mathcal{D}_n^d(j,\tau))\right)$$

- For both splits we have $L_{\text{mis}} = \frac{1}{2}\cdot\frac{1}{4} + \frac{1}{2}\cdot\frac{1}{4} = \frac{1}{4} = \frac{3}{4}\cdot\frac{1}{3} + \frac{1}{4}\cdot 0 = \frac{1}{4}$. However it seems that the second one is definitely better !

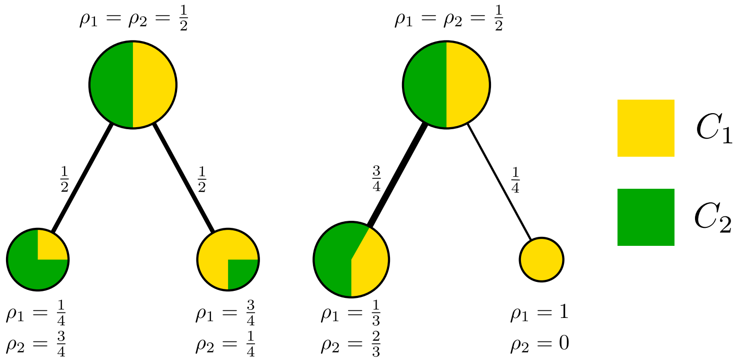


🟨 C_1

🟩 C_2

Limitations of the misclassification error

- For a partition where a class has a clear majority, we might not find a split which reduces L
- The function is not differentiable (optimization is harder)
- It might underestimate pure nodes:



Strict impurity function

Definition: strict impurity function

Let $H : [0:1]^K \to \mathbb{R}$ be an impurity function, ρ, ρ' two distributions in Δ_{K-1} with $\rho \neq \rho'$ and $\alpha \in]0,1[$. Then H is called strict, if it is strictly concave:

$$H(\alpha\rho + (1-\alpha)\rho') > \alpha H(\rho) + (1-\alpha) H(\rho')$$

If H is strict then it follows that

$$L(t_{j,\tau}, \mathcal{D}_n) = \frac{n_g}{n} H\left(\rho(\mathcal{D}_n^g(j,\tau))\right) + \frac{n_d}{n} H\left(\rho(\mathcal{D}_n^d(j,\tau))\right) \leq H\left(\rho(\mathcal{D}_n)\right)$$

$$n_g = |\mathcal{D}_n^g(j,\tau)| \quad \text{et} \quad n_d = |\mathcal{D}_n^d(j,\tau)|$$

the equality is given iff $\rho_k(\mathcal{D}_n) = \rho_k(\mathcal{D}_n^g) = \rho_k(\mathcal{D}_n^d)$ for all k

Remark: The impurity function of the misclassification error is concave, but it is not strictly concave. L might be equal for all possible splittings.

Entropy

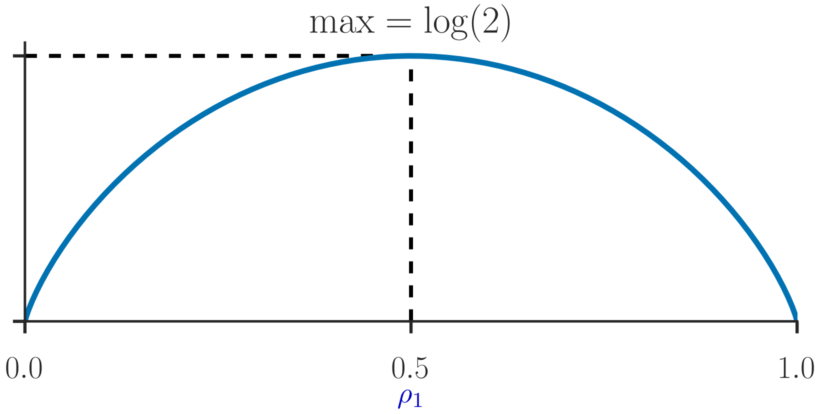
Entropy: $$H_{\text{ent}}(\mathcal{D}_n) = -\sum_{k=1}^{K} \rho_k(\mathcal{D}_n) \log \rho_k(\mathcal{D}_n)$$

- if we use $\log_2$, it is called Shannon entropy
- $-\log \rho(\mathcal{D}_n)$ is the information content of $\mathcal{D}_n$
- Entropy is defined as the expected value of the information content (average amount of information). It measures the randomness
- when an event is certain, entropy is 0
- information gain is defined as reduction in entropy
- it is differentiable

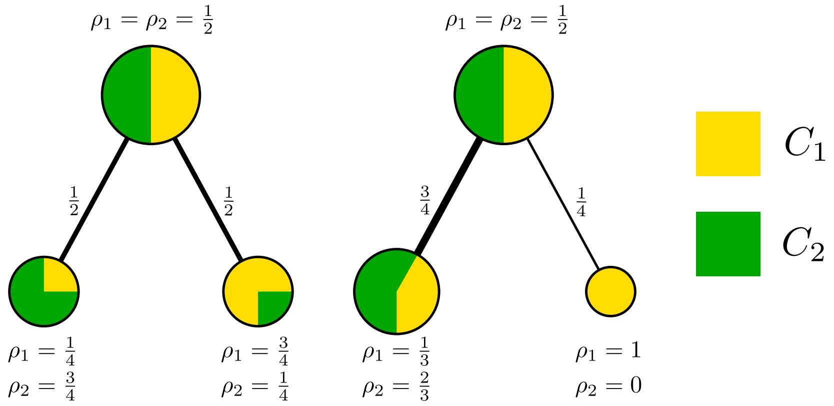
Entropy

When the number of classes K is 2

$$H_{\text{ent}}(\mathcal{D}_n) = -\rho_1(\mathcal{D}_n)\log\left(\rho_1(\mathcal{D}_n)\right) - \left(1-\rho_1(\mathcal{D}_n)\right)\log\left(1-\rho_1(\mathcal{D}_n)\right)$$



Example



Question: Compute L_{ent} associated to H_{ent}. Which split is better ?

Gini index

$$H_{\text{Gini}}(\mathcal{D}_n) = \sum_{k=1}^{K} \rho_k(\mathcal{D}_n)(1-\rho_k(\mathcal{D}_n)) = \sum_{k=1}^{K} \sum_{\substack{k'=1 \\ k'\neq k}}^{K} \rho_k(\mathcal{D}_n)\rho_{k'}(\mathcal{D}_n)$$

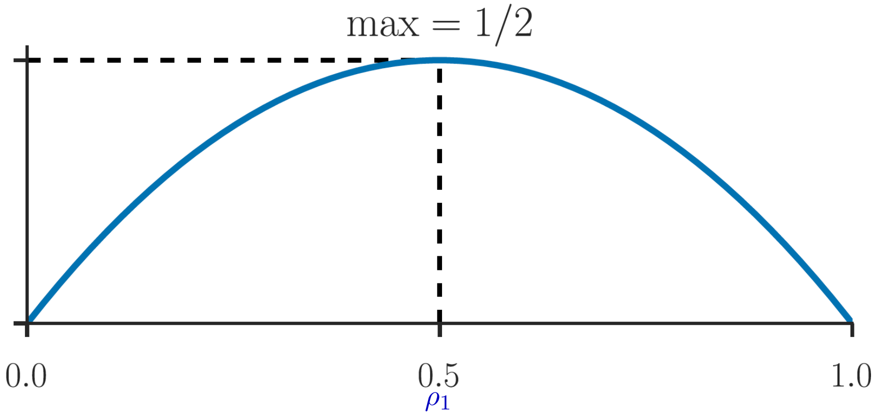
Two different interpretations:

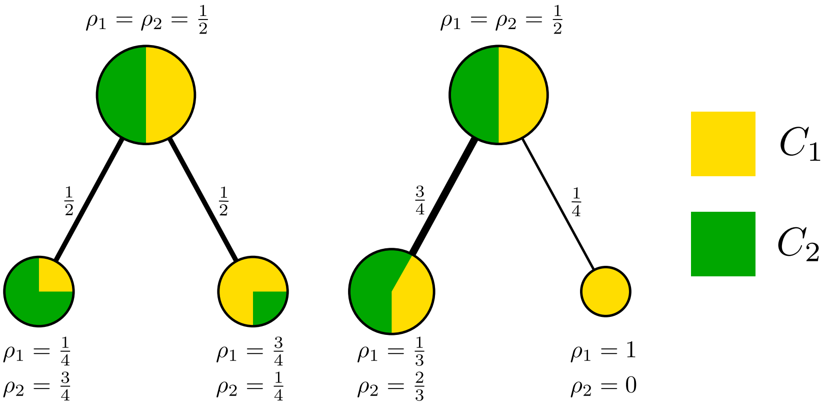
- If we code each observation as 1 for class k and zero otherwise (bernoulli variable) the variance is $\rho_k(\mathcal{D}_n)(1-\rho_k(\mathcal{D}_n))$. The Gini index is the sum of the variances of the "binarized" classes
- We do not assign observations to the majority class (as for H_{mis}) but we classify them to class k with probability $\rho_k(\mathcal{D}_n)$. The training error rate of this rule is $\sum_{\substack{k'=1 \\ k'\neq k}}^{K} \rho_k(\mathcal{D}_n)\rho_{k'}(\mathcal{D}_n)$. The Gini index is the sum over all classes.

Gini index

When the number of classes K is 2

$$H_{\text{Gini}}(\mathcal{D}_n) = 2\rho_1(\mathcal{D}_n)\left(1 - \rho_1(\mathcal{D}_n)\right)$$





Question: Compute L_{Gini} associated to H_{Gini}. Which split is better ?

Summary

Stopping criteria

Without stopping criteria, we could grow a tree until a situation where each observation represents a terminal node. This would be computationally expensive, difficult to interpret and prone to over-fitting. Instead, we could use one or more of the following stopping criteria:

- maximal depth
- maximum number of terminal nodes
- a node becomes a terminal node when it reaches a maximum number of observations
- degree of purity of a node (*i.e.*, threshold on $\rho_k(\mathcal{D}_n)$)

Categorical variables

- For a binary tree : if we have a categorical variable x which can take up to M values, we transform it into M binary variables
- Warning: The partitioning algorithm tends to favor categorical variables with many values since the number of possible partitions grows exponentially with M. This means that we have more choices to find a good partition. This can lead to over-fitting ! Try to avoid such variables.

Loss matrix

In some cases, the consequences of misclassifying observations can be very serious (*i.e.*, medicine). To account for that, we introduce a loss matrix $C \in \mathbb{R}^{K \times K}$, with $C_{k,k'}$ being the loss incurred for classifying observations of class k as belonging to class k'

$$C_{k,k'} = 0 \text{ si } k = k' \qquad C_{k,k'} \geq 0 \text{ si } k \neq k'$$

We can then modify the Gini index as follows:

$$\text{Gini index:} \quad \sum_{k=1}^{K} \sum_{\substack{k'=1 \\ k' \neq k}}^{K} C_{k,k'} \rho_k(\mathcal{D}_n) \rho_{k'}(\mathcal{D}_n)$$

Note: This works for $K > 2$ but it has no effect in the binary case (Why?).

Regression trees

For regression the process is almost identical, we only change the impurity function. We use the squared error (or variance):

$$H(\mathcal{D}_n) = \frac{1}{|\mathcal{D}_n|} \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}_n} (y_i - \bar{y}_n)^2$$

where

$$\bar{y}_n = \frac{1}{|\mathcal{D}_n|} \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}_n} y_i$$

and we minimize as before

$$L(t_{j,\tau}, \mathcal{D}_n) = \frac{n_g}{n} H(\mathcal{D}_n^g(j,\tau)) + \frac{n_d}{n} H(\mathcal{D}_n^d(j,\tau))$$

Note: as before we want to maximize the homogeneity (purity) of the terminal nodes

Summary

Model selection (1)

We can compute one (or more) of the following hyper-parameters instead than fixing them as stopping criteria:

- maximal depth of the tree
- maximum number of terminal nodes
- maximum number of observations in a node to become a terminal node

$\rightarrow$ we could use cross validation

Pruning (2)

What's the optimal size of a tree ? A large tree might overfit the data, while a small tree might not capture important structures.

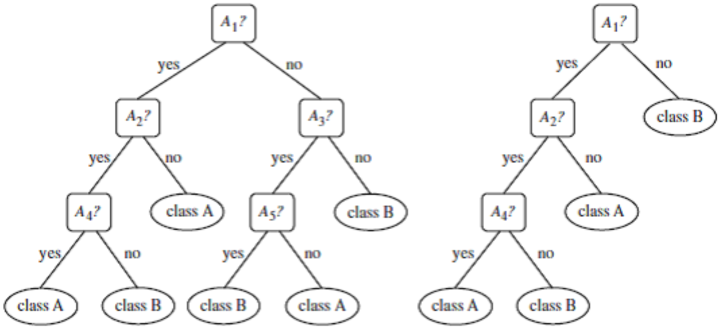
A possible solution is to grow a large tree in a training set stopping the splitting only when the terminal nodes have reached a minimum number of observations or a certain purity. Then, we produce several new trees by pruning the original tree at different nodes. When we want to prune with respect to node t, we delete all successor nodes of t in the original tree.

The new trees are then tested in a validation set. We select the tree that gives the best performance.

Trying all possible trees might be computationally unfeasible. Several greedy techniques exist. See Hastie *et al.* (2009) for more details.

Note: Only *Minimal Cost-Complexity Pruning* is currently supported in `sklearn`

Example of pruning



Advantages and drawbacks of decision trees

Advantages

- Build a non-linear and intepretable decision function
- Invariant under scaling and other linear transformations of the input data X
- Robust to the inclusion of (few) irrelevant features x
- It works for multi-class
- Computationally efficient: $O(\log F)$, where F is the number of terminal nodes
- It works for continuous and categorical variables (even if not optimal)

Advantages and drawbacks of decision trees

Drawbacks

- Low bias but very high variance. A small change in (all) input data can bring to a completely different tree (noisy) ! This instability is due to the hierarchical nature of the process. $\rightarrow$ averaging trees reduces the variance (bagging, random forests)
- No global optimization (NP-complete) we use a greedy algorithm where locally optimal decisions are made
- Separation hyperplanes are aligned with the feature axes $\rightarrow$ it might entail a sub-optimal solution
- Splits are hard. This creates piecewise-constant predictions with discontinuities at the split boundaries $\rightarrow$ prediction function is not smooth !

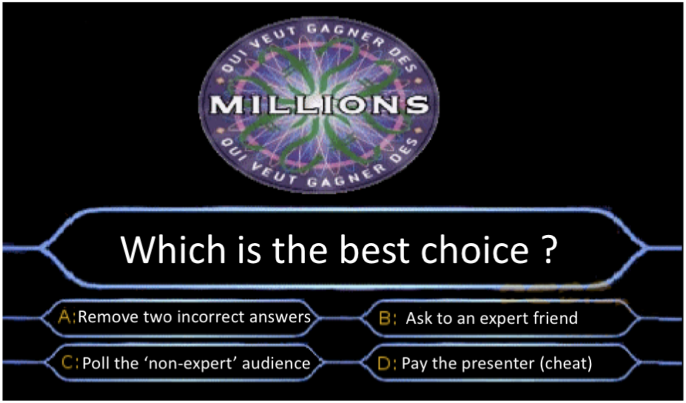
Summary

Trees limitations

- We know that Trees suffer from high variance
- We also know that averaging reduce variance $\rightarrow$ given M iid observations, each with variance σ^2, the variance of the average is $\frac{\sigma^2}{M}$
- For any statistical model (specially trees) we could take many different training sets, build a separate model in each training set and average the resulting predictions for regression or using the majority vote for classification
- Problem: we usually have only one training set !! $\rightarrow$ **Ensemble methods**

Qui veut gagner des millions ?

Which answer is the best choice ?



Which is the best choice ?

A: Remove two incorrect answers

B: Ask to an expert friend

C: Poll the 'non-expert' audience

D: Pay the presenter (cheat)

Qui veut gagner des millions ?

- Assume that you first use the option *Remove two incorrect answers* removing answer A) and D) $\rightarrow$ binary classification problem !
- Assume that:
  1. The number of people in the audience N is **odd** (to have only one winner...)
  2. The probability of choosing the correct answer is p for every person
  3. The votes of the audience are **independent**
- The probability of getting exactly k correct answers among the N voters is : $\binom{N}{k} p^k (1-p)^{N-k}$ (binomial distribution)
- The probability of ensemble success, namely when the majority $(N/2)+1$ makes the correct decision, is : $\sum_{k=(N/2)+1}^{N} \binom{N}{k} p^k (1-p)^{N-k}$
- This sum approaches 1 as $N \rightarrow \infty$ only when $p > 0.5$, otherwise it tends towards 0

Qui veut gagner des millions ?

Condorcet's jury Theorem

Given a group of N independent voters who choose the correct answer with probability $p > 0.5$ then, as $N \to \infty$, the probability of choosing the correct answer by majority vote tends to 1

- Assumptions about independence and uniform probability p not always realistic
- It can be extended beyond binary classification problem [1]
- It shows that **combining several weak classifiers may be better than using a single-expert classifier**

Ensemble learning

Ensemble learning - Definition

Combine several simple and moderately inaccurate classifiers to create a very accurate one

Rationale

It may be easier and more accurate to train and combine several simple classifiers than to learn a single complex classifier

Three famous ensemble methods

1. **Bagging**, 1994, Breiman
2. **Random Forests**, 1995, Tin Kam Ho
3. **Boosting**, 1990-1997, Schapire and Freund (AdaBoost won the Godel Prize in 2003)

Summary

Bagging (Bootstrap aggregating)

- Aggregate the prediction of several weak classifiers fitted on randomly generated and (almost) independent training sets (*model averaging*) in order to reduce variance and avoid overfitting
- Given the training set $\mathcal{D}_n$, we generate M new training sets $\{\mathcal{D}_k\}_1^M$, each of size k, by randomly sampling with replacement k samples from $\mathcal{D}_n$. This is called **bootstrap** and it means that an observation $\{x_i, y_i\}$ may appear multiple times in the same bootstrap training set $\mathcal{D}_k$.
- **Hypotheses**
  1. *Representativity*: if the size of $\mathcal{D}_n$ is large enough to capture the underlying actual distribution of the data, then a bootstrap set $\mathcal{D}_k$ should be a good approximation
  2. *Independence*: if k is sufficiently smaller than n, then the bootstrap sets $\mathcal{D}_k$ should not be too much correlated among them

Bagging (Bootstrap aggregating)

- Just like averaging i.i.d. random variables preserves the expected value and reduce variance, aggregating the prediction of several weak learners trained on (almost) independent data-sets does not change the expected outcome but reduces its variance
- Once fitted M weak models f_i, we can aggregate their predictions using:
  - A simple average for regression problems $\frac{1}{M}\sum_{i=1}^{M} f_i$
  - A majority vote for classification problems
- **Possible limitations**:
  - If n is not big enough and $k < n$, the bootstrap sets will not be a good approximation of the actual underlying distribution of the data
  - if $k \sim n$ the bootstrap sets might not be independent (correlation between predictions $\rightarrow$ Random Forests)

Summary

Random Forests

- When $k \sim n$, bagged trees might be very similar to each other if there is one or more strong predictors (i.e. features) $\rightarrow$ all trees will use these features in the top split
- If trees are similar their prediction might be highly correlated ! $\rightarrow$ Average of M identically distributed variables, each with variance σ^2 and positive pairwise correlation ρ, has variance:

$$\rho\sigma^2 + \frac{1-\rho}{M}\sigma^2 \tag{2}$$

- We should create **many uncorrelated** bagged trees in order to reduce both terms $\rightarrow$ **Random Forests** !
- The idea of Random Forests is to consider only a subsample of the p features at each split $\rightarrow$ in many splits strong predictors will not be considered and thus bagged trees should be less correlated

Random Forests - Algorithm

- For each one of the M tree T_m:
  1. Draw a bootstrap sample from $\mathcal{D}_n$
  2. Until every terminal node has less than n_{min} observations (typical values are 1 for classification and 5 for regression), repeat the following steps for each terminal node:
     I. Select at random q variables from the p of $\mathbf{x}$ $(q \leq p)$. Typical values are $\sqrt{p}$ for classification and $p/3$ for regression
     II. Estimate the best split for every selected variable
     III. Pick the best (variable,split) among the q
     IV. Split the node into left and right daughter nodes
- To predict the output of a new test observation $\mathbf{x}$:
  - *Regression*: $f^*_{RF} = \frac{1}{M} \sum_{m=1}^{M} T_m(\mathbf{x})$
  - *Classification*: We use the majority vote among the classed predicted by each tree $T_m(\mathbf{x})$
- The hyperparameters here are M, q and n_{min} $\rightarrow$ Cross-validation
- Intuitively, reducing q will also decrease ρ between pair of trees $\rightarrow$ true only if the fraction of relevant variables is not small !

Summary

Boosting

- Boosting is another general approach whose goal is to boost a weak learning algorithm, that works just slightly better than random guess, into a strong one
- There are many boosting algorithms and among them two of the most famous ones are:
  1. Adaptive boosting (AdaBoost)
  2. Gradient boosting (three of the most famous tree-based implementations are XGBoost, CatBoost and LightGBM)
- The first one (AdaBoost) was proposed by Schapire in 1990 and it consists of applying the same weak algorithm (e.g. Trees) several times, each time to a *modified version of the data*, producing a sequence of weak classifiers which are *finally combined*

AdaBoost

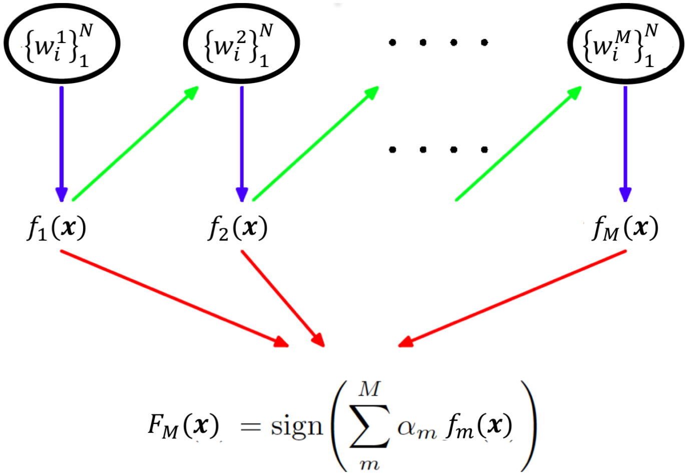


Figure 1: Schematic illustration of the AdaBoost algorithm. Each weak classifier f_m is trained on a weighted variation $w_n^{(m)}$ of the data. Classifiers are then combined into F_M via a weighted sum (α_m). Image modified from [2].

AdaBoost

Input: $\mathcal{D}_N = \{(x_i, y_i)\}$, type of classifier $f(x)$, number of classifiers M

- Initialize the weight of the data for the first classifier as $1/N$, namely all data points $\{x_i, y_i\}$ have equal weight $w_i^1 = 1/N$
- For $m = 1, ..., M$
  1. Fit classifier f_m to the training data $\mathcal{D}_N$ weighted with $\{w_i^m\}$, namely minimize $\sum_{i=1}^N w_i^m I(y_i \neq f_m(x_i))$
  2. Define $\epsilon_m = \frac{\sum_{i=1}^N w_i^m I(y_i \neq f_m(x_i))}{\sum_{i=1}^N w_i^m}$
  3. Use ϵ_m to evaluate the weight α_m of the classifier f_m $\rightarrow$ The more accurate f_m, the greater its weight α_m
  4. Use ϵ_m to update the weights of the data w_i^{m+1} $\rightarrow$ higher weights are given to the data which were incorrectly predicted

---

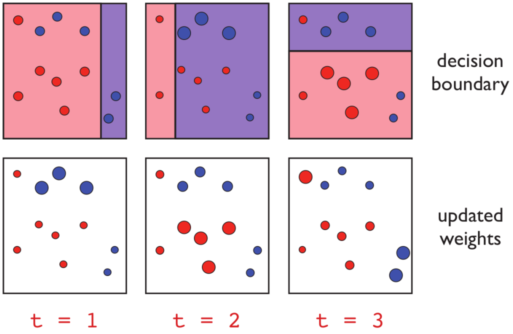
Output: $F_M(x) = \text{sign}\left(\sum_{m=1}^M \alpha_m f_m(x)\right)$

AdaBoost

How are defined the weights from a mathematical point of view ?

- The weights of the classifiers: $\alpha_m = \ln\left(\frac{1-\epsilon_m}{\epsilon_m}\right)$. Thus α_m takes value between 0 (classifier is equal to random guess) and $+\infty$ (classifier is correct $\forall i$)
- For the weights of the data we have $w_i^{m+1} = w_i^m \exp\left(\alpha_m\right)$ if $y_i \neq f_m(x_i)$ and $w_i^{m+1} = w_i^m$ if $y_i = f_m(x_i)$. Thus, since $\alpha_m \in \mathcal{R}_+$, w_i^{m+1} will be equal to w_i^m if the classifier is correct and **bigger** if the classifier is wrong
- Please note that we could also choose to reduce the weights w_i^{m+1} for the observations correctly classified by defining $w_i^{m+1} = w_i^m \exp\left(-\alpha_m\right)$ if $y_i = f_m(x_i)$

AdaBoost - binary classification



(a)

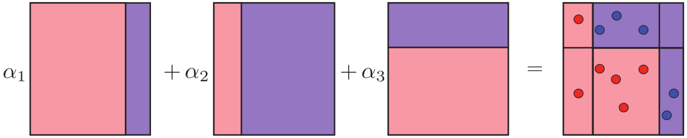


Figure 2: Axis-aligned hyperplanes as base learners. Image taken from [7].

AdaBoost - binary classification

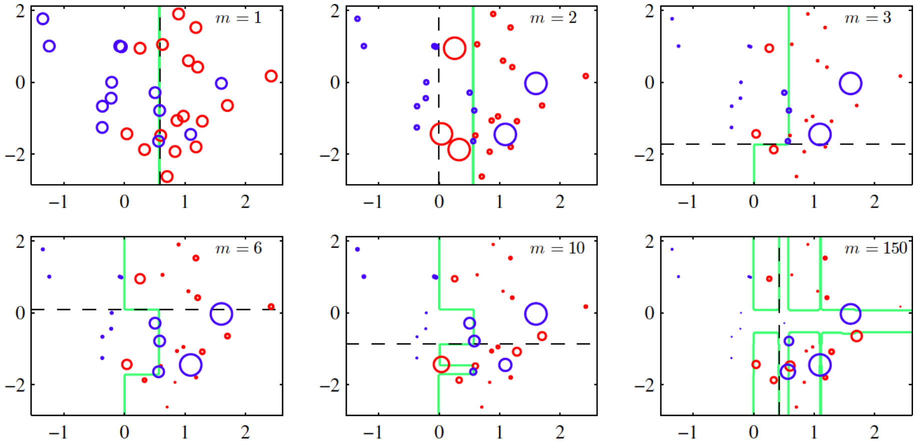


Figure 3: Black line: base learner at current iteration. Green line: ensemble decision boundary. Image taken from [2].

Gradient Boosting

- To introduce gradient boosting it is useful to rewrite the AdaBoost algorithm in terms of *Forward Stagewise Additive Modeling*
- We can notice that the idea of Boosting is to add at each iteration a new weak classifier (or model) $f(x;\beta_m)$ without adjusting the parameters β and coefficients α of the previous models $(1,..,m-1)$ $\rightarrow$ greedy solution

$$h(x) = \frac{1}{2} \sum_{m=1}^{M} \alpha_m f(x;\beta_m) \tag{3}$$

- Please note that in this example f is a tree model, β_m the parameters of the m model (split variables, split points, predictions of terminal nodes) and α_m are the coefficients of the m model

Gradient Boosting

- At each iteration m one looks for the optimal model $f_m = f(x;\beta_m)$ and corresponding coefficient α_m to add to the current expansion $h_{m-1}(x)$
- Given $\mathcal{D}_N = \{(x_i, y_i)\}$, number of models M and a loss function $\mathcal{L}$
- The algorithm for the Forward Stagewise Additive Modeling is:

- Initialize $h_0(x) = 0$
- For $m = 1, ..., M$
  1. Compute

$$(\alpha_m, \beta_m) = \arg\min_{\alpha,\beta} \sum_{i=1}^{N} \mathcal{L}(y_i, h_{m-1}(x_i) + \frac{1}{2}\alpha f(x_i;\beta)) \tag{4}$$

  2. Set $h_m(x) = h_{m-1} + \frac{1}{2}\alpha_m f(x;\beta_m)$

Gradient Boosting

- This can be seen as a generic formulation for Boosting $\rightarrow$ change the loss $\mathcal{L}$ and you have different algorithms !
- Exponential loss ($\mathcal{L}(y, f(x)) = \exp(-yf(x))$) $\rightarrow$ AdaBoost
- L2-norm, L1-norm, M-estimators $\rightarrow$ Regression
- Cross-entropy $\rightarrow$ Multi-class classification
- Cox model $\rightarrow$ Survival models

AdaBoost - Details

- Please notice that in AdaBoost, we are not minimizing the training error ϵ_m given by the 0-1 loss but an upper bound, namely: $\frac{1}{N}\sum_{i=1}^{N} I(F_M(x_i) \neq y_i) \leq \frac{1}{N}\sum_{i=1}^{N} \exp\left(-y_i h(x_i)\right)$ where $h(x_i) = \sum_m \alpha_m f_m(x_i)$
- Since the 0-1 loss is non-smooth and non-convex, it is common in statistical learning to use surrogate losses such as the exponential loss

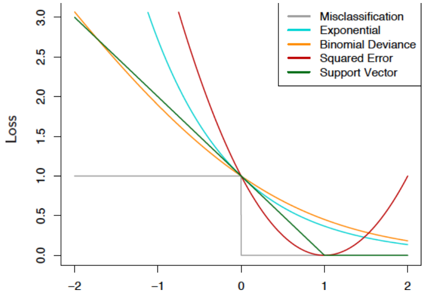


Figure 4: Figure taken from [5].

Gradient Boosting

- To increase generalization capability, we can also use Regularization techniques such as:
  - **Sampling** the training data (with or without replacement) using only a small part of the entire training set at each iteration $\rightarrow$ it introduces randomness (models less similar/correlated), effective when data-set is very large
  - **Shrinking** the impact of the additional models. Simplest way is to introduce a hyper-parameter $\gamma > 0$ in the update: $h_m(x) = h_{m-1}(x) + \frac{1}{2}\gamma\alpha_m f(x; \beta_m) \rightarrow$ to reduce model complexity and avoid overfitting
  - **Early stopping**, we choose the number of models f_m that corresponds to the minimum validation error

XGBoost

- XGBoost is a recent and very effective implementation of gradient boosting with some peculiarities:
  - It adds two regularization terms on $f_m(x)$: a L1-norm on the number of leaves, and a L2-norm on the leaf weights
  - it uses a second-order approximation of the cost function which gives a new impurity function
  - they also use shrinking (see slide before) and feature sub-sampling as in random forests
  - fast algorithm to find the best split (even for large data-sets)
  - can handle missing (or sparse) values
  - efficient implementation using a block structure and a cache-aware prefetching algorithm
  - Freely-available here: https://xgboost.readthedocs.io/en/latest/build.html

LightGBM

- LightGBM has been implemented to tackle data-sets with many features and large feature dimension
- Instead than scanning all observations for each feature to estimate the best split points, authors propose a technique to use only few observations (the ones with the greater gradients)
- Accuracy is preserved, with respect to other gradient boosting algorithms such as XGBoost but computational time is definitely lower
- Freely available here: `https://lightgbm.readthedocs.io/en/latest/Python-Intro.html`

Other implementations

- **CatBoost**: efficient and fast implementation with support for categorical variables
- **pGBRT**: fast implementation especially used for ranking

Take home message: all of these implementations have many different hyper-parameters (for trees, regularization and optimization) that need to be set by the user.. not so simple ! It needs time and experience
In scikit-learn, you can find the implementation of AdaBoost and LightGBM (called Histogram-Based Gradient Boosting)

Summary

Stacking

- Differently from Bagging and Boosting, Stacking considers **heterogeneous weak learners** (different models like KNN and SVM)
- Differently from Bagging and Boosting, Stacking aggregates the heterogeneous weak learners using an **external meta-model** (like a neural network) and not an deterministic strategy
- Depending on the problem, the user needs therefore to select the appropriate heterogeneous weak learners and the meta-model that combines their predictions

References

1. Hastie, Tibshirani and Friedman (2009). The Element of Statistical Learning. Springer.
2. Bishop (2006). Pattern Recognition and Machine Learning. Springer.
3. L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone (1984). Classification and regression trees. Wadsworth Statistics/Probability Series.
4. Breiman, L. (2001). Random forests, Machine Learning 45: 5–32.
5. R. Schapire and Y. Freund (2012). Boosting: Foundations and Algorithms. MIT Press.
6. Zhi-Hua Zhou (2012). Ensemble Methods: Foundations and Algorithms. Chapman & Hall/CRC