

Trees, forests and boosting

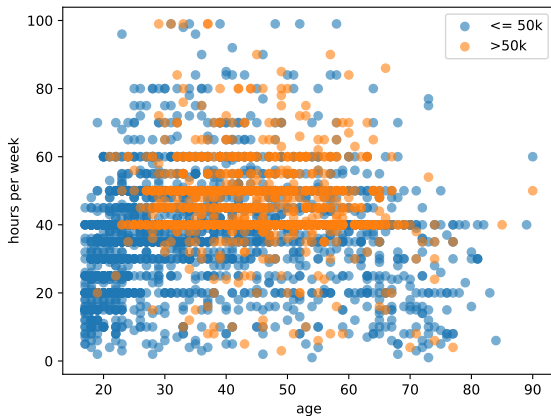
Mathurin Massias

`https://mathurinm.github.io`

Inria, OCKHAM team

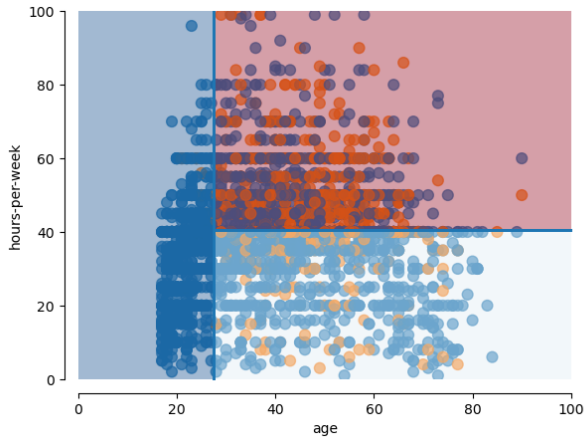
26/02/2025

Motivation: nonlinear data



How would logistic regression perform on this dataset?

A potential classifier?



Outline

Trees

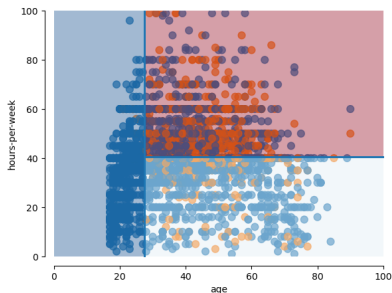
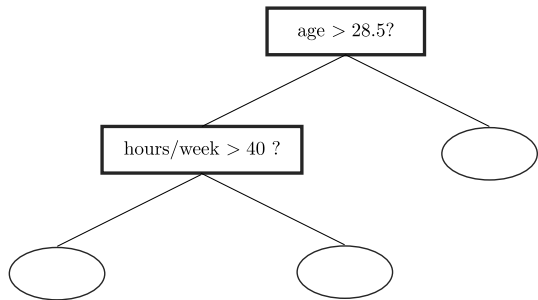
Ensemble methods: bagging and random forests

Adaboost

What is a tree?

Trees partition the whole space into rectangular cells:

- A node has exactly either zero or two children
- Each split (= question) defines two child nodes, the left and right child nodes
- A node with zero children is called a **leaf**
- We pass from a node to the left or right children by answering a question of type “Is $x_j \geq \alpha$?” for some coordinate j and threshold α



CART: Classification and Regression Trees

CART = an algorithm to build a tree out of a training set $\{(\mathbf{x}_1, y_1)\}_{i=1}^n$

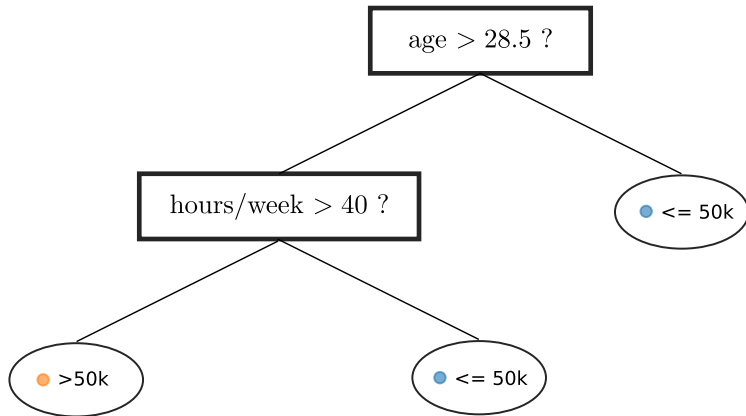
- Partition the space, use **constant prediction** over leaves
- Objective : split the space to fit training data well
- Adapted to two settings:
 - y_i qualitative with K modalities ($y_i \in \{1, \dots, K\}$): classification tree (DecisionTreeClassifier)
 - y_i quantitative, $y_i \in \mathbb{R}$: regression tree (DecisionTreeRegressor)

Pros of trees:

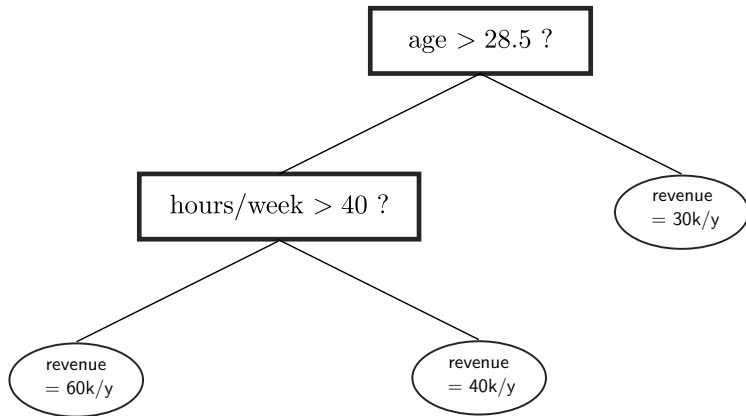
Easy to interpret

Nonparametric model: no assumption on the data distribution.

Example of classification tree

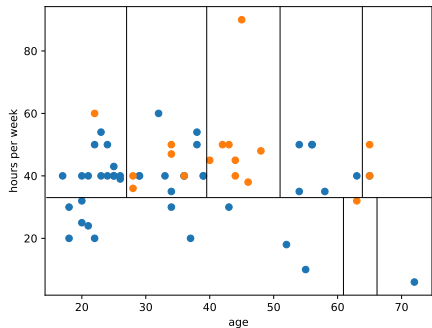
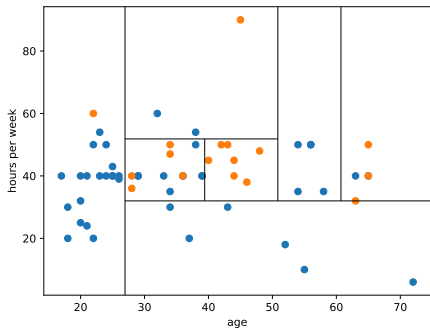


Example of regression tree



Building a tree = partitioning the space

Building a tree aims at finding a partition of the input space into a set of rectangles that separates blue points from orange points



How to classify new samples?

Classification → A simple majority vote to predict class probabilities

- select $C(\mathbf{x})$ the cell containing \mathbf{x}
- predict majority class inside $C(\mathbf{x})$:

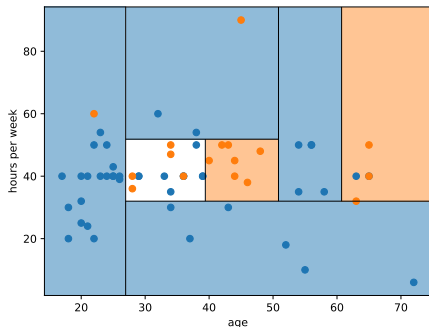
$$\hat{y}(\mathbf{x}) = \begin{cases} +1 & \text{if } \sum_{i:\mathbf{x}_i \in C(\mathbf{x})} \mathbb{1}_{y_i=1} > \sum_{\mathbf{x}_i \in C(\mathbf{x})} \mathbb{1}_{y_i=-1} \\ -1 & \text{otherwise.} \end{cases}$$

How to classify new samples?

Classification → A simple majority vote to predict class probabilities

- select $C(\mathbf{x})$ the cell containing \mathbf{x}
- predict majority class inside $C(\mathbf{x})$:

$$\hat{y}(\mathbf{x}) = \begin{cases} +1 & \text{if } \sum_{i:\mathbf{x}_i \in C(\mathbf{x})} \mathbb{1}_{y_i=1} > \sum_{\mathbf{x}_i \in C(\mathbf{x})} \mathbb{1}_{y_i=-1} \\ -1 & \text{otherwise.} \end{cases}$$



How to classify new samples?

Regression → A simple average in each leaf to predict a value (in each region the predicted value is constant)

- select $C(\mathbf{x})$ the cell containing \mathbf{x}
- predict mean of target of training points inside $C(\mathbf{x})$:

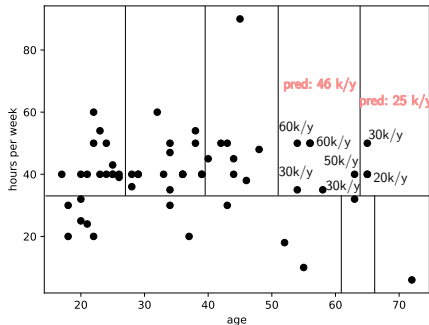
$$\hat{y}(\mathbf{x}) = \frac{\sum_{i:\mathbf{x}_i \in C(\mathbf{x})} y_i}{|C(\mathbf{x})|}$$

How to classify new samples?

Regression → A simple average in each leaf to predict a value (in each region the predicted value is constant)

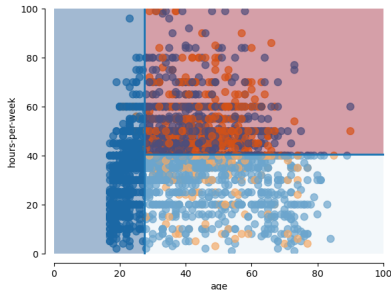
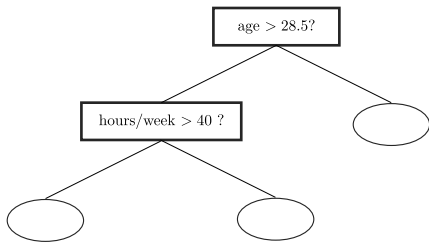
- select $C(\mathbf{x})$ the cell containing \mathbf{x}
- predict mean of target of training points inside $C(\mathbf{x})$:

$$\hat{y}(\mathbf{x}) = \frac{\sum_{i:\mathbf{x}_i \in C(\mathbf{x})} y_i}{|C(\mathbf{x})|}$$



An algorithm to build a tree: CART

- The CART algorithm builds the partition recursively (split after split)
- At each step, the method splits an existing cell into two regions according to a split variable (j in x_j) and a threshold point (t): the question is: " $x_j > t$?"



This is the best tree found by `scikit-learn`. How were the splits found?

Penguin time

01-grow_a_tree.ipynb

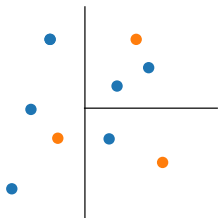
How to build the optimal tree? Recap after notebook

Iteratively (split after split)

- We want to split a node N into a left child node N_L and a right child node N_R
- The children depend on the cut = the (feature, threshold) pair denoted by (j, t)

$$N_L(j, t) = \{x \in N : x_j < t\} \quad N_R(j, t) = \{x \in N : x_j \geq t\}.$$

- a measure of cell “purity” is used; take split that maximizes gain in purity
- for all current leaves N , all possible feature/threshold pairs, compute purity gain if we used this split. Pick split with maximal gain.



How many splits to try?

How to split?

Heuristic: greedy algorithm

At each cut, choose a split so that the two new regions are as homogeneous possible

Homogeneous nodes

For **classification**: class proportions should be as close as possible to $(0, 1)$ or $(1, 0)$

For **regression**: labels should be very concentrated around their mean in a node/cells

How to quantify homogeneity?

Gini index, Entropy (**classification**)

Variance (**regression**)

Regression: split measure

- **Impurity** is the variance of the target y inside the node N :

$$V(N) = \sum_{i:\mathbf{x}_i \in N} (y_i - \bar{y}_N)^2 \quad \bar{y}_N = \frac{1}{|N|} \sum_{i:\mathbf{x}_i \in N} y_i \quad |N| = |\{i : \mathbf{x}_i \in N\}|$$

- **Information gain** is given by

$$IG(j, t) = V(N) - \frac{|N_L(j, t)|}{|N|} V(N_L(j, t)) - \frac{|N_R(j, t)|}{|N|} V(N_R(j, t))$$

Regression: Finding the best split

Maximize the information gain:

$$\max_{j,t} \left[\min_{\bar{y}_L} \sum_{i:\mathbf{x}_i \in N_L(j,t)} (y_i - \bar{y}_L)^2 + \min_{\bar{y}_R} \sum_{i:\mathbf{x}_i \in N_R(j,t)} (y_i - \bar{y}_R)^2 \right]$$

For any (j, t) the inner minimization is solved by

$$\bar{y}_L = \frac{\sum_{i:\mathbf{x}_i \in N_L(j,t)} y_i}{|N_L(j,t)|}; \quad \bar{y}_R = \frac{\sum_{i:\mathbf{x}_i \in N_R(j,t)} y_i}{|N_R(j,t)|}$$

For each j , finding t can be done quickly \rightarrow determination of the best (j, t) is feasible!

Classification: split measure

Given the classes distribution inside cell N , $p_N = (p_{N,1}, \dots, p_{N,K})$, with

$$p_{N,k} = \frac{|\{i: \mathbf{x}_i \in N, y_i = k\}|}{|N|}$$

(if two classes only: $p_N = (p, 1 - p)$)

- Two possible **impurity** measures:

$$G(N) = G(p_N) = \sum_{k=1}^K p_{N,k}(1 - p_{N,k}) \quad \text{Gini index}$$

$$H(N) = H(p_N) = \sum_{k=1}^K p_{N,k} \log_2(p_{N,k}) \quad \text{Entropy index}$$

basically: maximal when $P_N = (1/K, \dots, 1/K)$ (cell is not pure!)

- Information gain** is given by ($I=G$ or $I=H$):

$$IG(j, t) = I(N) - \frac{|N_L(j, t)|}{|N|} I(N_L(j, t)) - \frac{|N_R(j, t)|}{|N|} I(N_R(j, t))$$

- best split: enumerate all possible splits...

CART algorithm

CART builds the partition **iteratively**. At each iteration:

- Find the best (node, feature, threshold) triplet (N, j, t) that maximizes $IG(N, j, t)$
- Create the two new children of the leaf
- Stop if some stopping criterion is met
- Otherwise continue

Some **stopping criteria**?

CART algorithm

CART builds the partition **iteratively**. At each iteration:

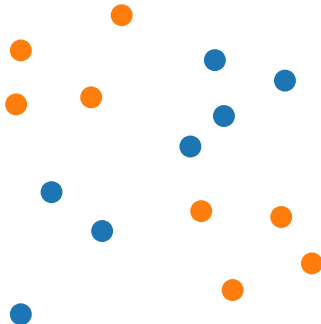
- Find the best (node, feature, threshold) triplet (N, j, t) that maximizes $IG(N, j, t)$
- Create the two new children of the leaf
- Stop if some stopping criterion is met
- Otherwise continue

Some **stopping criteria**?

- Maximum depth of the tree
- All leafs have less then a chosen number of samples
- Impurity in all leafs is small enough
- Testing error is increasing

Post processing: pruning

What should we do here?



Final remarks on trees

- 😊 leads to nice interpretable results
- 😊 insensitive to datascaling (no preprocessing needed!)
- 😞 usually overfit
- 😞 usually not the best for prediction
- 😊 but the basis of more powerful techniques: random forests, boosting (ensemble methods)

Ensemble methods

Basic idea

Aggregate multiple models or “weak learners” trained for the same problem. Weak models combined rightly give accurate model

- **Bagging**: multiple weak models of the same type that learn from different data sets in parallel and are combined to decrease the variance on the prediction
- **Boosting**: weak models learn sequentially and adaptively to improve model predictions of a learning algorithm.

Outline

Trees

Ensemble methods: bagging and random forests

Adaboost

Bagging: training models on bootstrap samples

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} \quad \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{pmatrix}$$

$\in \mathbb{R}^{n \times p}$ $\in \mathbb{R}^n$

Bootstrap copies:
(sampling w. replacement)

$$\begin{pmatrix} x_1 \\ x_3 \\ x_3 \\ \vdots \\ x_{n-3} \end{pmatrix} \quad \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{pmatrix}$$

$\in \mathbb{R}^{n \times p}$ $\in \mathbb{R}^n$

train model 1

$$\begin{pmatrix} x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_n \end{pmatrix} \quad \begin{pmatrix} y_2 \\ y_3 \\ y_4 \\ \vdots \\ y_n \end{pmatrix}$$

$\in \mathbb{R}^{n \times p}$ $\in \mathbb{R}^n$

train model 2

...

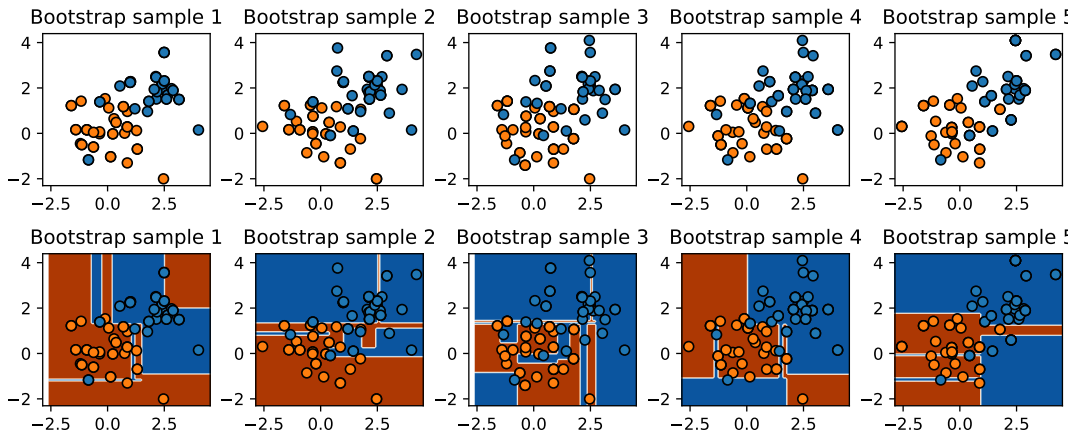
$$\begin{pmatrix} x_1 \\ x_1 \\ x_2 \\ \vdots \\ x_{n-1} \end{pmatrix} \quad \begin{pmatrix} y_1 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

$\in \mathbb{R}^{n \times p}$ $\in \mathbb{R}^n$

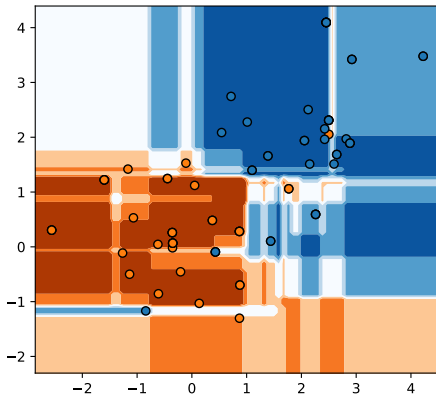
train model N

Final model = aggregation of N models (majority vote or averaging)

Bagging



Bagging: averaging the 5 models



overfits way less than individual models!

A refinement of bagging: Random Forests

- for more robustness, in bagging we can also subsample **features** for each bootstrap copy
- see `sklearn.ensemble.BaggingClassifier`
- Random forest can subsample features **at each split** (no need for absolute best split)
- see `sklearn.ensemble.RandomForestClassifier`
- check more ensemble classifiers in the `sklearn.ensemble` submodule

Pros and cons of bagging and RFs?

Pros and cons of RFs

- many models to train
- many models to evaluate for prediction
- model can be trained in parallel (`n_jobs` parameter)
- base models are usually simple/small
- aggregating models gives more expressivity (vs linear models)
- RF is the go to estimator to try on real data: fast to train, easy to tune

Outline

Trees

Ensemble methods: bagging and random forests

Adaboost

Adaboost

- Freund & Schapire 1995 (Gödel prize 2003)
- setup: binary classification
- (very) weak learners $h^{(t)}$ learnt on **weighted** training points
- iteratively give more weight to misclassified points
- Final classifier: $H(\mathbf{x}) = \text{sign}(\sum_{t=1}^T \alpha^{(t)} h^{(t)}(\mathbf{x}))$

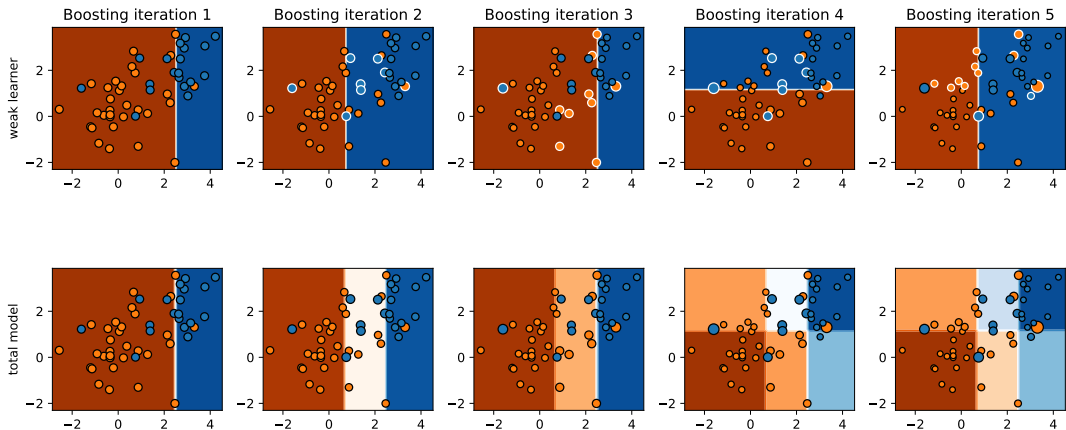
Adaboost

For binary classification

ADABOOST($D_n = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, BASE(\cdot, \cdot), T)

```
1   $\mathbf{w}^{(1)} \leftarrow (1/n, \dots, 1/n)$   $\triangleright$  initial weights
2  for  $t \leftarrow 1$  to  $T$ 
3       $h^{(t)} \leftarrow \text{BASE}(D_n, \mathbf{w}^{(t)})$   $\triangleright$  calling the base learner
4       $\gamma^{(t)} \leftarrow \sum_{i=1}^n w_i^{(t)} h^{(t)}(\mathbf{x}_i) y_i$   $\triangleright$  edge =  $1 - 2 \times \text{error}$ 
5       $\alpha^{(t)} \leftarrow \frac{1}{2} \ln \left( \frac{1 + \gamma^{(t)}}{1 - \gamma^{(t)}} \right)$   $\triangleright$  coefficient of  $h^{(t)}$ 
6      for  $i \leftarrow 1$  to  $n$   $\triangleright$  re-weighting the points
7          if  $h^{(t)}(\mathbf{x}_i) \neq y_i$  then
8               $w_i^{(t+1)} \leftarrow w_i^{(t)} \frac{1}{1 - \gamma^{(t)}}$ 
9          else
10              $w_i^{(t+1)} \leftarrow w_i^{(t)} \frac{1}{1 + \gamma^{(t)}}$ 
11 return  $f^{(T)}(\cdot) = \sum_{t=1}^T \alpha^{(t)} h^{(t)}(\cdot)$ 
```

Adaboost



- point size is proportional to weights
- white points are points which were misclassified by the weak learner at previous iteration

Adaboost generalization

- Gradient boosting (any loss, beyond binary)
- many implementations: Xgboost (apple), lightGBM (MS), CatBoost (yandex)
- supports feature binning (sklearn `HistGradientBoostingClassifier`), robust to preprocessing
- tree-based models outperform deep learning on tabular data Why do tree-based models still outperform deep learning on tabular data?
- almost always involved in winning submission on Kaggle