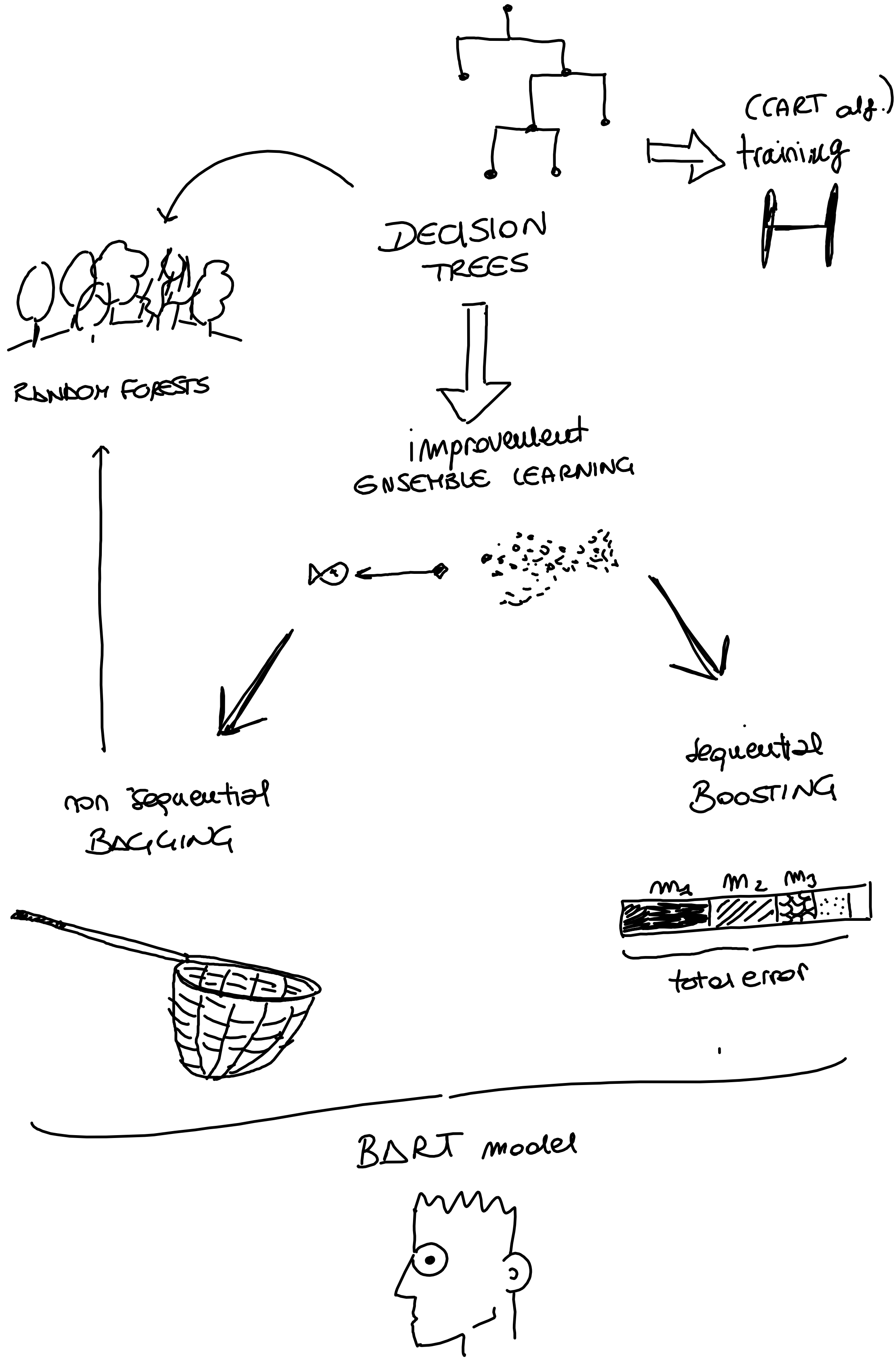
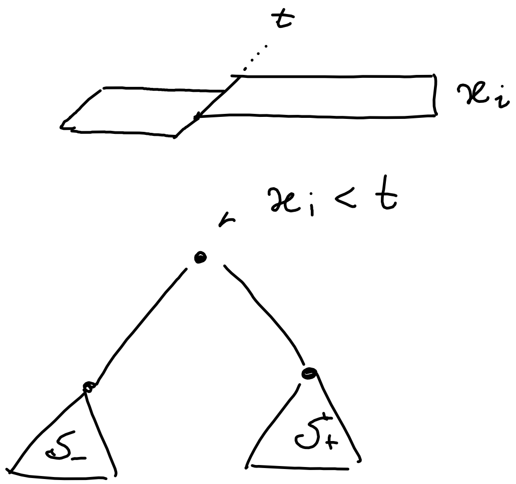

(CART alg.)
training
DECISION TREES
RANDOM FORESTS
improvement
ENSEMBLE LEARNING
non sequential
BAGGING
sequential
BOOSTING
m1 m2 m3
total error
BART model

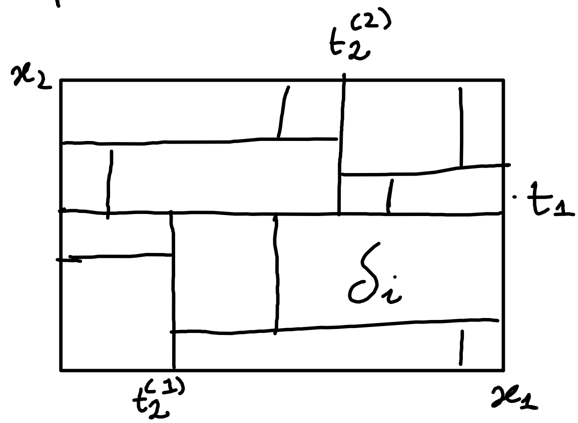
DECISION TREES

$x_1, \dots, x_m$ features $\qquad y = \{C_1, \dots, C_m)$. class

Recursive Definition



Each split corresponds to a subdivision of the feature space:

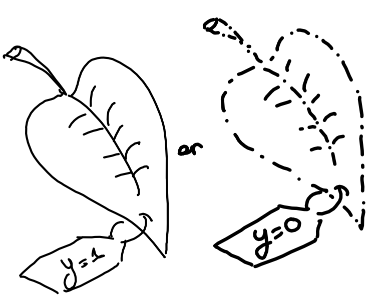


At each level we try to find best pair of $(x_i; t)$ to divide the data in order to:

- minimize RSS (REGRESSION)
- maximize IG (CLASSIFICATION)

Prediction formula: In every hyperplane S_i of the feature space the same formula for prediction is used.

final leaves



$$f_S := \begin{cases} \frac{1}{|S|}\sum_{i \in S} y_i = \bar{y}_S & \text{in Regression} \\ \max_k P(y \in C_k \mid \underline{x} \in S) & \text{in Classif.} \end{cases}$$

LIKE: infected tree.

DECISION TREES (REGRESSION)

$$f_S = \bar{y}_S = \frac{1}{|S|} \sum_{y \in S} y$$ (prediction $\forall S$ hyp. in feature space)

GOAL FUNCTION:

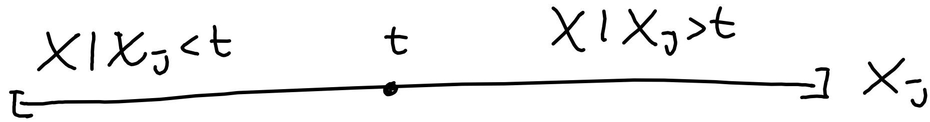
$$\min_J \underbrace{\sum_{j=1}^{J} \sum_{i \in S_j} (y_i - \bar{y}_{S_j})^2}_{RSS}$$

J = # of subsest S we find.

The goal is finding the best set of $S_1, \dots, S_J$ to divide the feature space, so that the predictions we make minimize the goal function.

SOLUTION:

Top-down building of the tree.
At each level we seek the best pair $(x_j; t)$ such that splitting the feature space into:



leads to biggest reduction in RSS.

DECISION TREES (CLASSIFICATION)

mathematical form of the optimization problem.

the threshold t is found as the value that, each split, maximizes the information gain:

$$IG_t(S) = H(S) - \sum_{i=1}^{2} \frac{|S_i|}{|S|} H(S_i)$$

This means that at every split we seek the t that divides the training set in the most balanced way:

$$\min IG_t(S_r) = \max \sum_{i=1}^{2} \frac{|S_i|}{|S|} H(S_i)$$

where $S_r := \{$record in the root$\}$

Eventually the IG we can achieve by splitting the node will be very small.

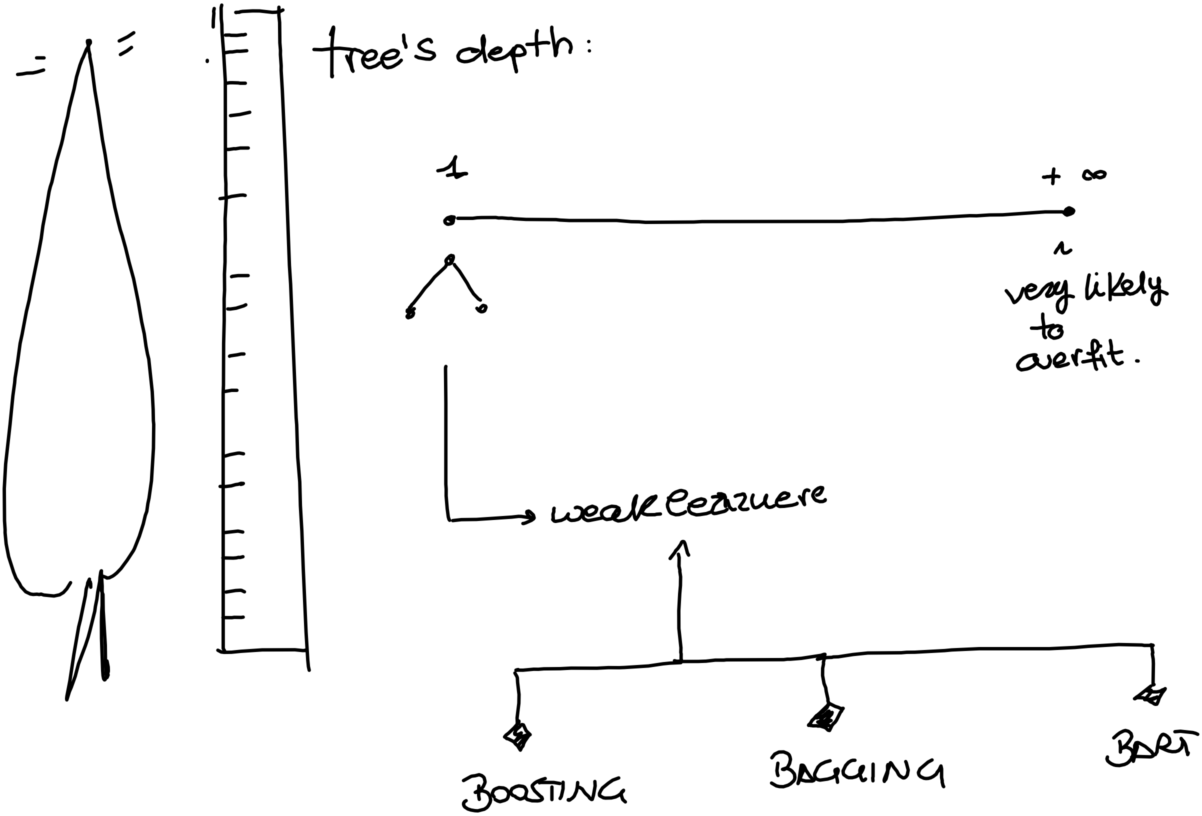
$$|IG| < \varepsilon$$

This means that the leaf we arrived at during the splitting process cannot be divided into 2 balanced subsets of records. So the leaf is already very imbalanced: its records belong mostly to the same class. So:

$$H(S_r) = -\sum_{i=1}^{m} f_i \ln f_i \simeq -1 \ln 1 \simeq \emptyset$$

DECISION TREES

hyperparameters



<u>NOTE</u>

For how they work, decision trees do <u>NOT</u> demand any kind of feature scaling. Infact features (x) are analyzed singularly and there's no direct comparison between them of any kind..
This means that relative scales don't matter.

DECISION TREES (PRUNING)

PROBLEM: decision trees tend to overfit the data. They become too complex and the feat. space (X) subd. too specific.

Pruning is the operation of undoing the splits (rejoining 2 subspaces of the feature space) that grant the less advancement wrt the optimization problem

PRO:
- reducing overfitting tendency.

DEAD BRANCH: → not $\downarrow\downarrow$ RSS (Regression).
→ not $\uparrow\uparrow$ IG (classif.)

DECISION TREES
in Sklearn

The algorithm that is used to train the model in Python Sklearn is called Cart. It works both for classification and regression trees, by changing the cost function.

At every split considered we try to find the best pair (k, t_k) that minimizes a problem tailored cost function.

CLASSIFICATION (cost: Gini impurity ; $f_i(x) = \max_{C_j} P_i(y \in C_j \mid x \in S_i)$)

$$\min_{k,t} J(k, t_k) = \min_{k,t} \frac{m_{left}}{m} G_1 + \frac{m_{right}}{m} G_2$$

REGRESSION (cost: MSE ; $f_i(x) = \frac{1}{|S_i|} \sum_{y \in S_i} y$)

$$\min_{k,t} J(k, t_k) = \min_{k,t} \frac{m_{left}}{m} MSE_{left} + \frac{m_{right}}{m} MSE_{right}$$

where
- G_1, G_2 measures the impurity of the left, right subset
- $m_{left/right}$ is the n° of instances in the left/right subset.
- k is a feature (from x) and t_k the splitting threshold.

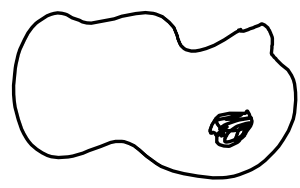
① DECISION TREES

Gini impurity or entropy?

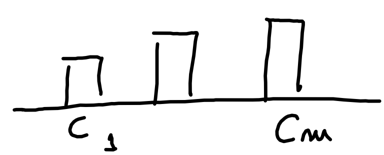
Let's assume we have: m different classes to choose from: $C_1, \dots, C_m$

Which is the best cost function for training a classification tree?

$$G_{S_i} = 1 - \sum_{k=1}^{m} p_{i,k}^2$$



$$H(S_i) = - \sum_{\substack{k=1 \\ p_{i,k} \neq 0}}^{n} p_{i,k} \log_2 (p_{i,k})$$



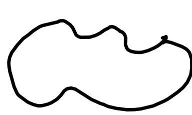
where: $$p_{i,k} = \frac{\# (x,y) \in S_i \text{ so that } y = C_k}{\# (x,y) \in S_i}$$

ANSWER: there's no big difference in the result. Gini impurity is a little more computationally efficient.

$G_{S_i} = 0$

$H(S_i) = 0$

$\Longleftrightarrow$

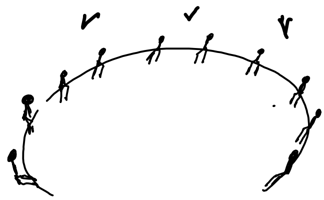


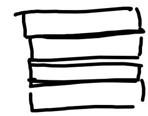
S_i All instances in S_i have the same label.

ENSEMBLE LEARNING

Class of tecniques that uses the wisdom of the crod effect

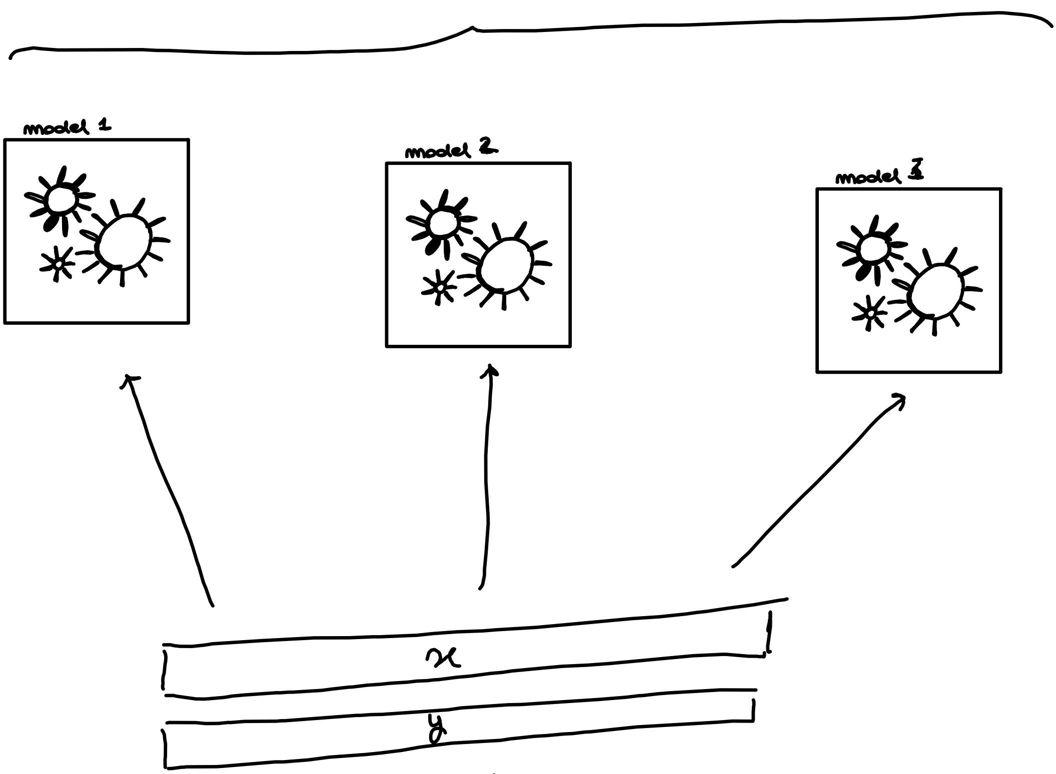
y ← STACKING





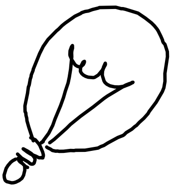
(ex: mean, mode, majority vote...)

ENSEMBLE PREDICTION



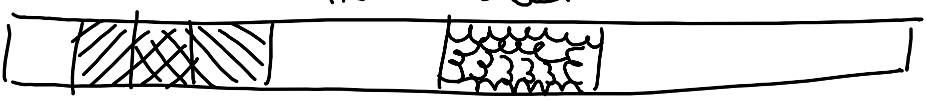
TRAINING DATA

BAGGING..

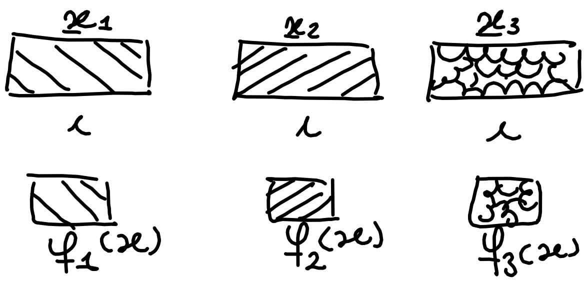


model that are good will tend to agree on the same prediction, while bad model tend to disagree on different predictions

TRAINING SET

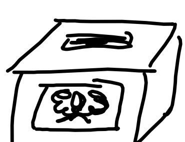


B bootstrap samples from the training data



train a weak model on each one of them

prediction happens through votation of weak model



$$y \leftarrow f(x) = \frac{1}{B} \sum_{i=1}^{B} f_i(x)$$

BAGGING
theory.

Bagging is a method used for reducing the variance of a statistical learning method.

$$f(x) = \frac{1}{B} \sum_{i=1}^{B} f_i(x)$$

The model prediction is computed as average output of B other models, trained on Bootstrap sets.

Infatt Bagging stands for Bootstrap Aggregation.
Since we output an avg., from the CLT:

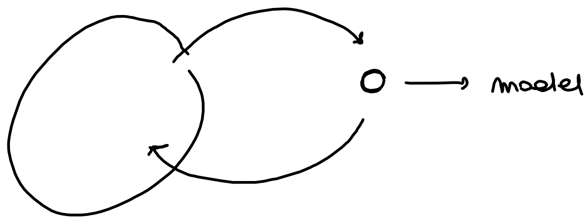
$$f(x) \overset{d}{\sim} \mathcal{N}\left(\bar{f}(x) = y ; \frac{\mathbb{V}(f(x))}{B}\right)$$

As B increases the final model variance decreases.
This is why we operate bagging with high variance learning methods.

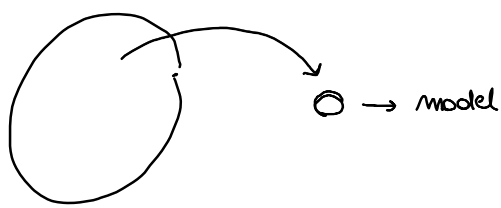
Bagging improves prediction accuracy at the expenses of readibility.

BAGGING and PASTING and Out of bag evaluation (OOb)

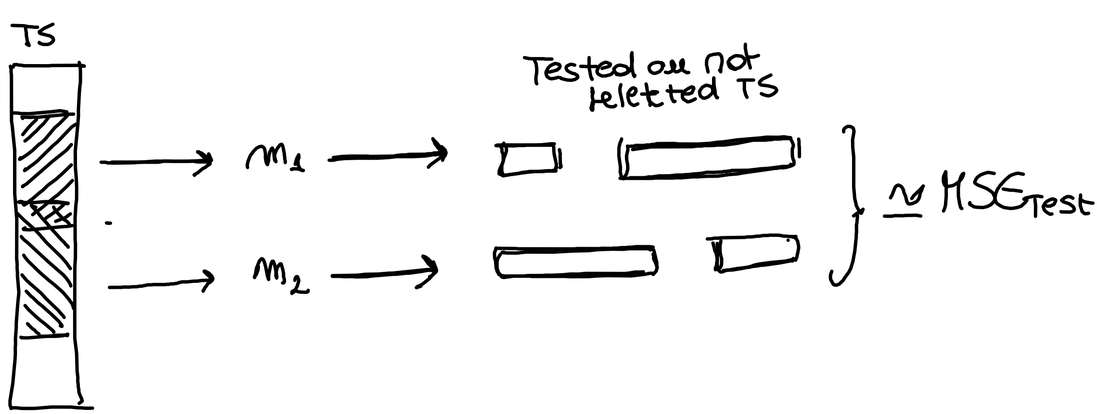
BAGGING: performs sampling with replacement



PASTING: performs sampling without replacement

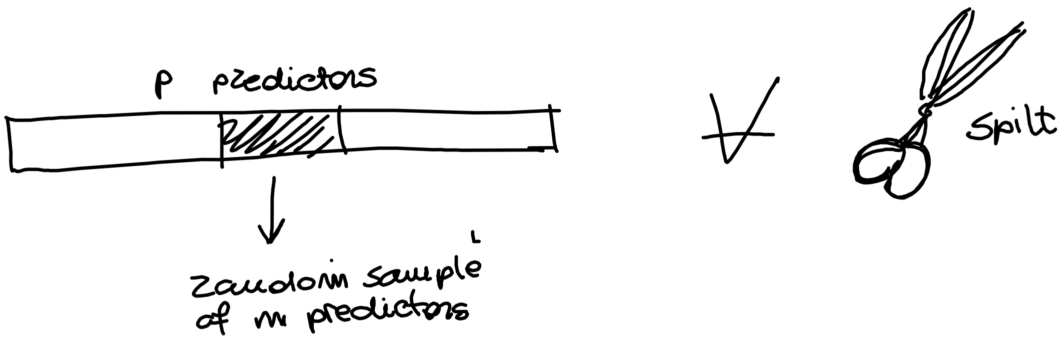


Out of BAG evaluation: since every model is trained on a random subset of the training data, the training instances not used for training can be used as test instances:



RANDOM FORESTS

We apply a bagging procedure to a decision tree. Each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates.



Avoiding collinearity between bagged trees

RANDOM FOREST

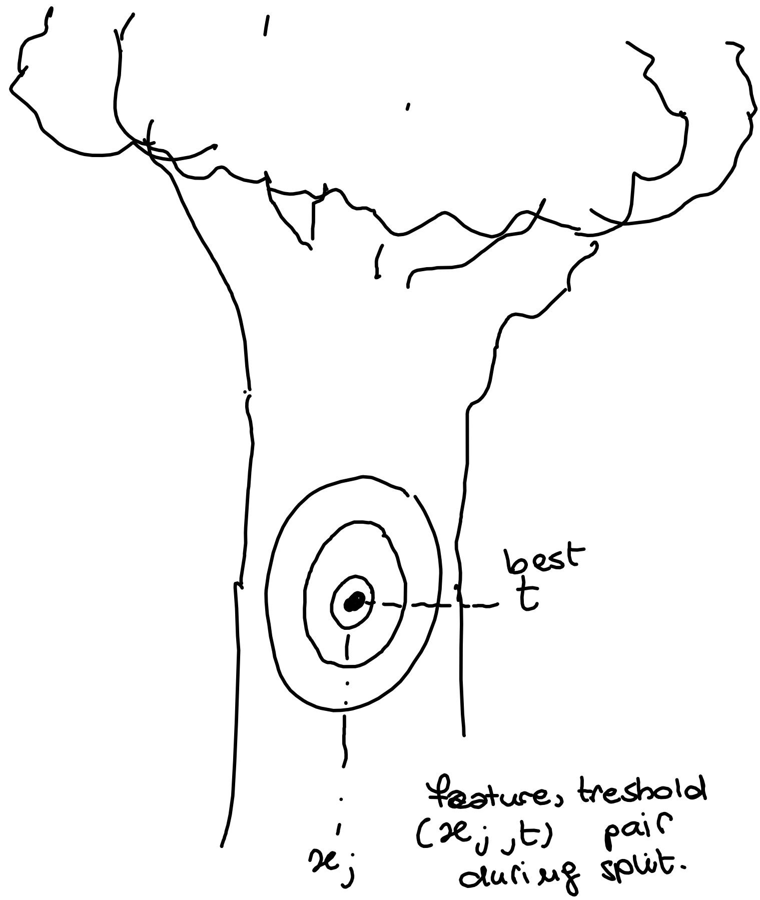
Having correlated trees doesn't lead to a reduction in variance as having many un-correlated trees. For this reason RANDOM FORESTS represent a quick improvement of bagged trees

NOTE: usually $m \simeq \sqrt{p}$.

EXTREMELY RANDOMIZED TREES

Like random forests, but also the threshold t at each split during training is randomly selected

random t selected



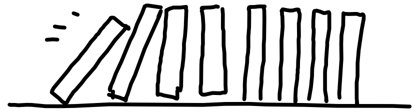
PRO: • training is faster the Random forests.

There's no way of knowing if ERT will fit the data better than Random forests beforehand.

BOOSTING

overview

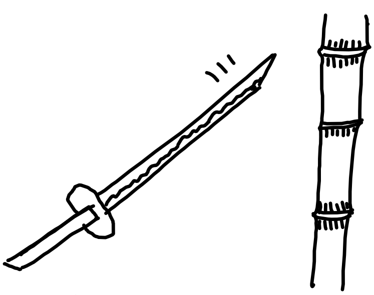
sequential training



BOOSTING

training on residuals

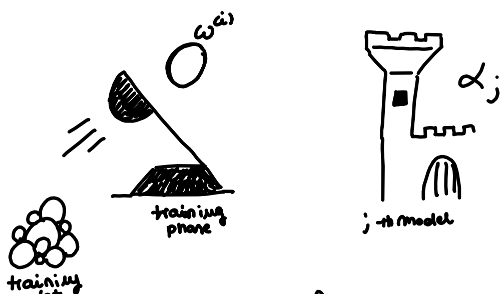
GRADIENT BOOSTING



$$f(x) = \alpha^T \underline{f}(x)$$

t. on weighted training set

ADA-BOOST

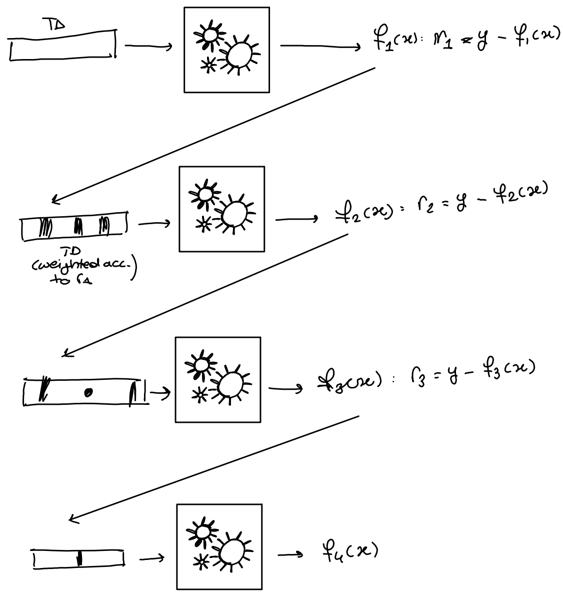


$$f(x) = h(\alpha_j)$$

model weights.

BOOSTING: Gradient Boosting

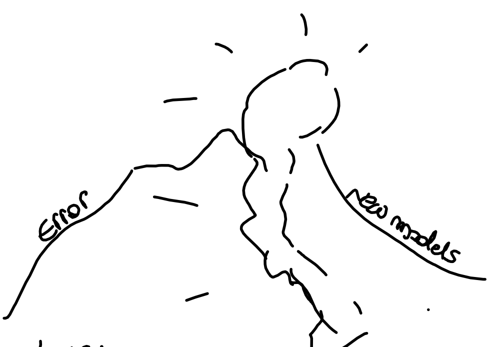
Create sequential models to predict the errors the other ones make.



boosted model: $$f(\underline{x}) = \sum_{i=1}^{n} \alpha\, f_i(x) = \alpha^T \underline{f}(x)$$

Each new model is trained having as labels the error of the previous ones:

$$y^{(k)} = y - \sum_{i=1}^{k-1} f_i(\underline{x})$$



LIKE: EROSION

NOTE: Similarity to Grad. Desc.
At each step the overall error is implicitly reduced by training a new mod. on the past error.

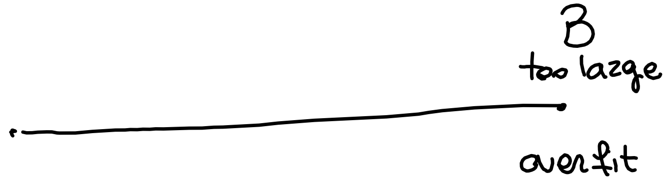
STEP UPD.) $x_{k+1} = x_k - \alpha \nabla f(x_k)$

RESID. UPD.) $y_{k+1} = y_k - \alpha f_k(x)$

For this reason, Boosting is also referred as Gradient Boosting.

BOOSTING: Gradient Boosting hyperparameters

1) B := n° of models to train

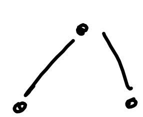


chosen through cross validation

2) α := learning rate (step of GD)

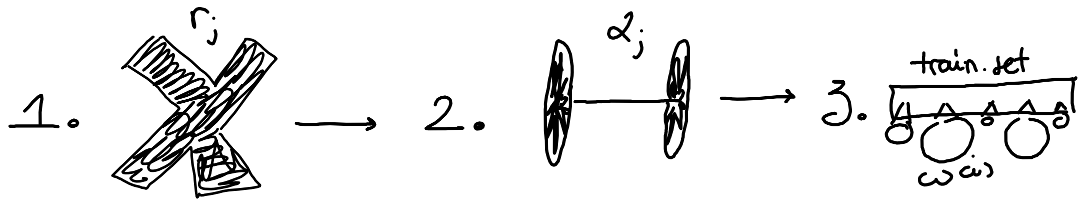
Typical values: $\alpha = 10^{-2}, 10^{-3}$

3) d := n° of splits in each tree
this controls the complexity of the boosted ensemble.
Often: $d = 1$



BOOSTING

AdaBoost basic operations. Models are sequentially giving each time more importance to « hard » training instance



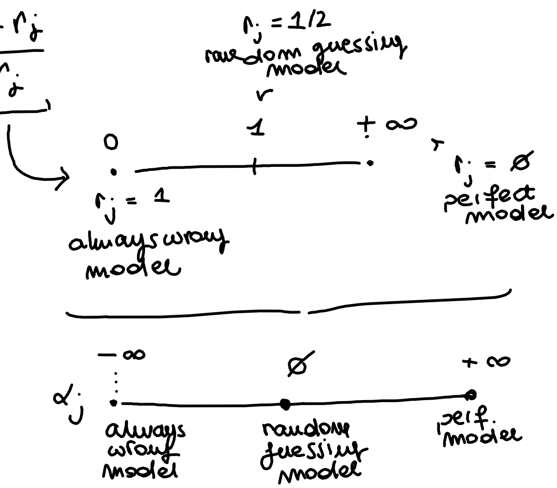
$\forall$ model sequentially trained:

1. weighted error rate of the j-th model

$$r_j := \frac{\sum\limits_{\substack{i=1 \\ \hat{y}_j^{(i)} \neq y^{(i)}}}^{m} \omega_i}{\sum\limits_{i=1}^{m} \omega_i}$$

2. generating weight of the j-th model

$$\alpha_j := \eta \log \underbrace{\frac{1 - r_j}{r_j}}$$



3. weight updating

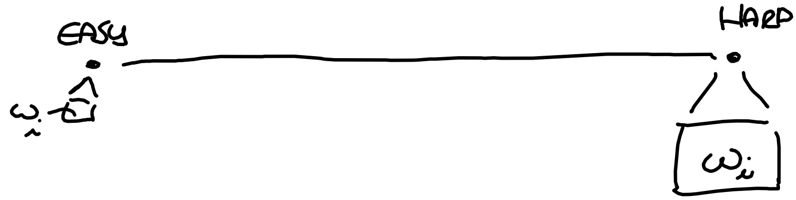
$$\omega^{(i)} = \begin{cases} \omega^{(i)} & \text{correct prediction: } \hat{y}_j^{(i)} = y^{(i)} \\ \omega^{(i)} e^{\alpha} & \text{wrong pred.: } \hat{y}_j^{(i)} \neq y^{(i)} \end{cases}$$

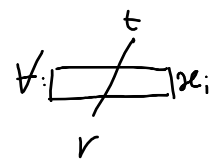
BOOSTING: AdaBoost

example algorithm: each new model is trained on data that are weighted by how hard they [were] hard for the past models to get right.

Training data are weighted:

NOTE: $y_i = \begin{cases} 1 \\ -1 \end{cases}$





AdaBoost:

normalize weights.

FOR $t = 1, \dots, T$:

- choose weak classifier:
  - find weak learner $h_t(x)$ that minimize the weighted sum of errors: $\varepsilon_t = \sum_i \omega_{i,t} \leq 0{,}5$
  - choose $\alpha_t := \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$

- ensemble:
  - $F_t(x) = F_{t-1}(x) + h_t(x) \,] = h_1(x) + h_2(x) + \dots$
- update weights:
  - $\omega_{i,t+1} = (\omega_{i,t}) e^{-y_i \alpha_t h_t(x_i)} \quad \forall i$
  - Renormalize weights.

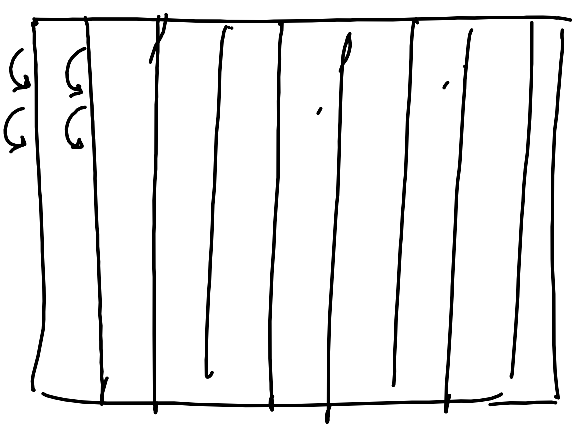
$$-y_i \alpha_t h_t(x_i) = \begin{cases} >0 & h_t(x_i) \neq y_i \rightarrow \uparrow \omega_i \quad \text{WRONG PRED.} \\ <0 & h_t(x_i) = y_i \rightarrow \downarrow \omega_i \quad \text{RIGHT PRED.} \end{cases}$$

BART: visualization and influences.

from BAGGING.

parallel training of m trees

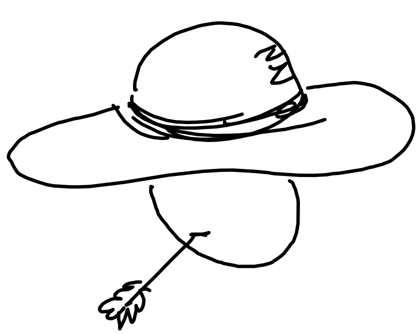
random. modif → from previous models



from GRAD BOOSTING

eg. training on residuals

BART



BART (for zegression)

Bayesian Additive Regression trees

K = # regression trees

B = # iterations

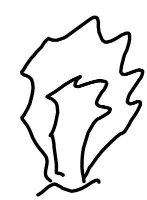
$f_k^{(b)}(x)$ = prediction at x of the k-th zegression tree at iteration b

n = # train. records

1. $f_1^{(1)}(x) = \dots = f_K^{(1)}(x) = \frac{1}{nK} \sum_{i=1}^{n} y_i$
2. $f^{(1)}(x) = \frac{1}{n} \sum_{i=1}^{n} f_i^{(1)}(x)$

like bagging

3. for $b = 2, \dots, B$
   a) for $K = 1, \dots, K$:
      I) for $i = 1, \dots, n$ compute partial residual
      $$r_i = y_i - \sum_{k' \neq K} f_k^{(b)}(x)$$
      like boosting
      II) Fit a new tree $f_K^{(b)}(x)$ by rand. perturbating $f_K^{(b-1)}(x)$.
   b) compute $f^{(b)}(x) = \sum_{K=1}^{K} f_K^{(b)}(x)$
4. Compute the mean after L burn-in samples
$$f(x) = \frac{1}{B-L} \sum_{b=L+1}^{B} f^{(b)}(x)$$



In words:

1, 2: initialization of K regression trees. $f^{(1)}(x)$ is computed as avg. output of the K trees

3.: B iterations. At each iteration everyone of the K trees is updated from its previous version, by a random modification

4.: the model is an avg. of the not-burnt prediction models.

BART (for regression)

key insights

3. At this step we do not fit a NEW tree to the current partial residual error but a modification of the previous one

- AVOID overfitting: improving the older tree prevents from fitting «hard» the data at each iteration

4. The burn-in period (the first L iterations) corresponds to the number of model we do not consider in the final version model. Generally the first models found $(f^{(1)}(x), \dots, f^{(L)}(x))$ tend to not provide very good results.

4. As in bagging the final model is an average of the best models we have found:

$$f(x) = \frac{1}{B-L} \sum_{i=L+1}^{B} f^{(i)}(x)$$

$$= \frac{1}{B-L} \sum_{i=L+1}^{B} \sum_{k=1}^{K} f_k^{(i)}(x)$$

Having 2 avg in the final model, maximally leverages the variance reduction effect of the mean.

As CLT states:

$$\bar{X} \overset{d}{\sim} \mathcal{N}\left(E[\tilde{X}]; \frac{V[\tilde{X}]}{n}\right)$$

BART
hyperparameters setting

For how it is thought: the BART:

- prevents overfitting
- is good out of the box

Standard value for its hyperparameters are:

$K = 200$ (n° of trees)
$B = 1000$ (n° of iterations)
$L = 100$ (burn-in period)

STACKING.

Instead of using a voting aggregation function like the mean in bagging or boosting, we can train a model to aggregate votes.