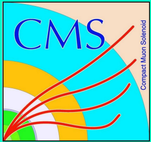
BDT



Outline

Basics

- Decision Trees – Two classes of problems
- Boosting
- Flowchart of implementing your own training
- How XGBoost implements trees, boosting, loss function

Intermediate (Linked to our specific problems)

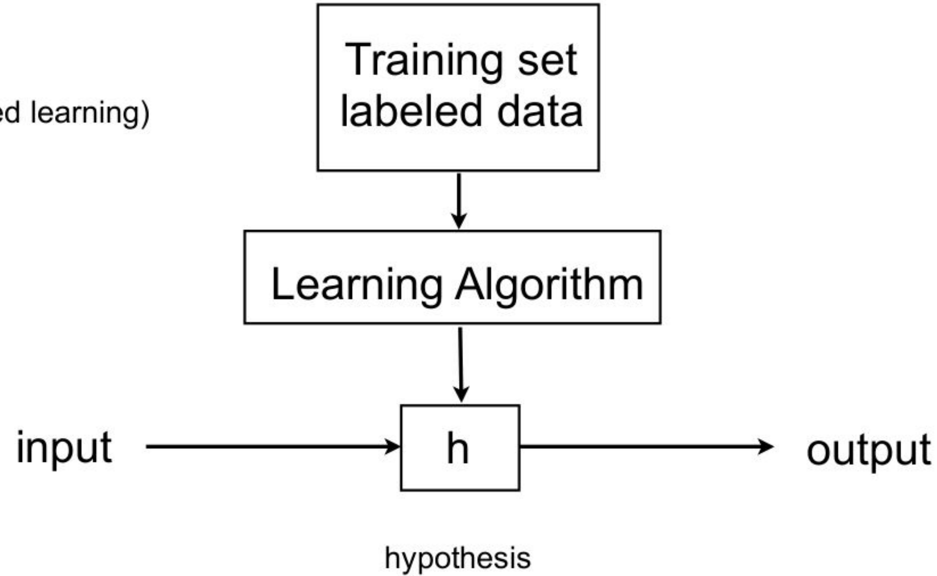
- Overtraining how to deal with it
- Correlated input features
- Unbalanced training data set, weights, negative weights!
- Is the trained model representative of data

Advanced+ Hands on

- Hyperparameters in XGBClassifier and Hyperparameter optimization in a systematic way
- Comparison between XGBoost and GBR, and within a model different loss functions
- Multiclassification
- Semi-parametric regression

tifr

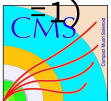
Supervised learning



Two classes of problems :

classification (e.g. separate sig/bkg: output 1 for sign 0 for bkg)

regression (e.g. energy corrections:output will be a weigh such that (output x Erec)/Egen =1)

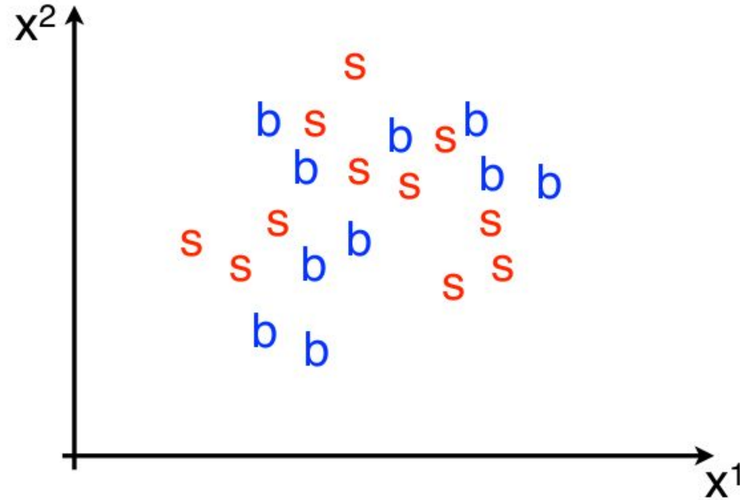


tifr

Decision trees : classification

Ex: Training sample $\in \Re^2$

	classes
$(x^1{}_1, x^2{}_1)$	
...	
$(x^1{}_i, x^2{}_i)$	b
...	
$(x^1{}_n, x^2{}_n)$	s



In this case it's difficult to have a good separation with a single linear cut.
Introduce non linearities

For the DT the idea is to separate the classes placing several simple cuts
(i.e. binary splits of the data $x^i <$ value or $x^i >$ value)

Decision trees : classification

Training sample $\in \Re^2$

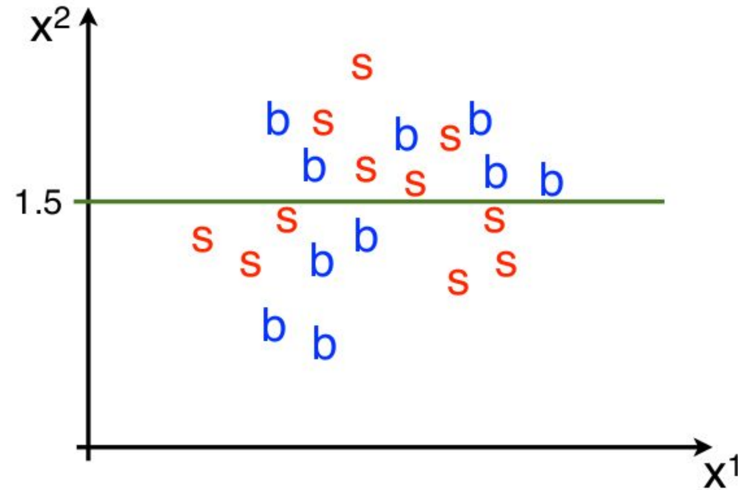
$(x^1{}_1, x^2{}_1)$ classes

...

$(x^1{}_i, x^2{}_i)$ b

...

$(x^1{}_n, x^2{}_n)$ s

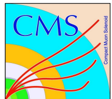


Where to put the cuts ?
Strategy is to minimize the misclassification at each step

You choose the variable that provides the greatest increase in the separation measure (e.g., Gini index) in the two daughter nodes relative to the parent. (The same variable may be used at several nodes or ignored)

Define a metric for the separation:
the "Gini index" Gini = P(1-P) Where P =purity:

$$P = \frac{\sum_{\text{signal}} w_i}{\sum_{\text{signal}} w_i + \sum_{\text{background}} w_i}$$



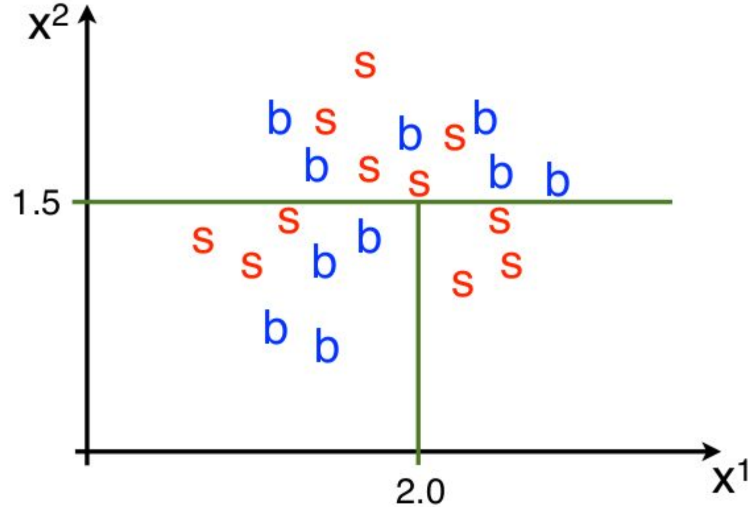
This is maximum for P = 0.5 (no separation / random guess) and zero for P = 0 or 1.
(having purity of 0 or 1 is the same, you always have max separation)

tifr

Decision trees : classification

Training sample $\in \Re^2$

	classes
(x^1_1, x^2_1)	
...	
(x^1_i, x^2_i)	b
...	
(x^1_n, x^2_n)	s

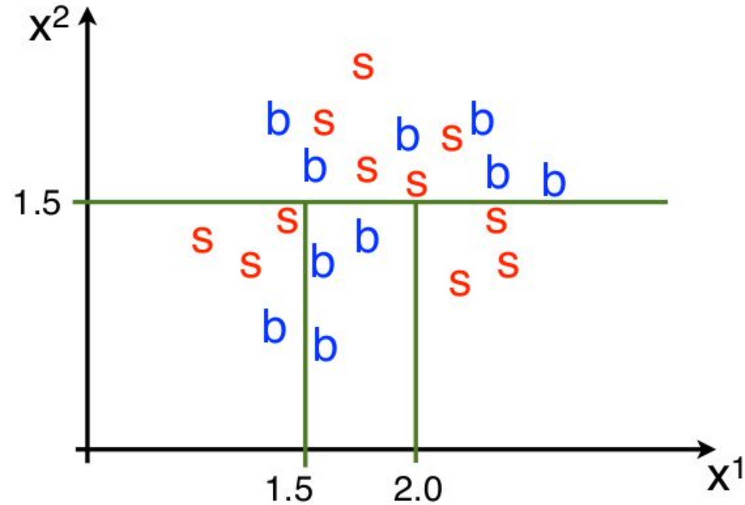


Strategy is to minimize the misclassification at each leaf

Decision trees : classification

Training sample $\in \Re^2$

	classes
(x^1_1, x^2_1)	
...	
(x^1_i, x^2_i)	b
...	
(x^1_n, x^2_n)	s

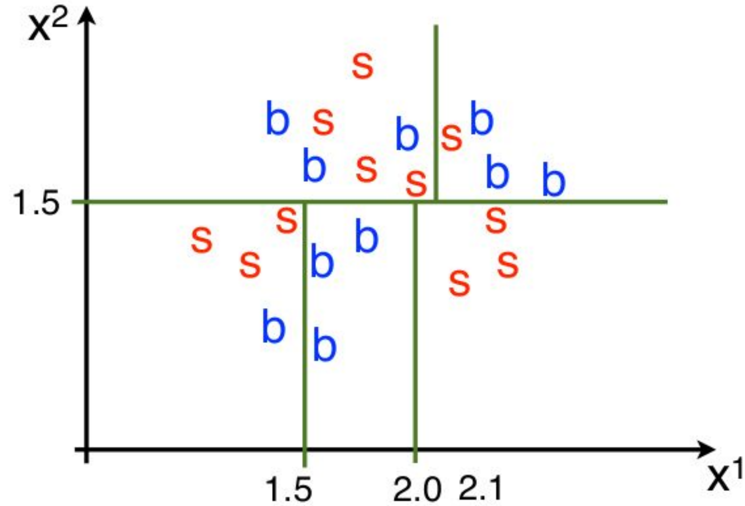


Strategy is to minimize the misclassification at each leaf

Decision trees : classification

Training sample $\in \Re^2$

	classes
$(x^1{}_1, x^2{}_1)$	
...	
$(x^1{}_i, x^2{}_i)$	b
...	
$(x^1{}_n, x^2{}_n)$	s



Strategy is to minimize the misclassification at each leaf

Decision trees : classification

Training sample $\in \Re^2$

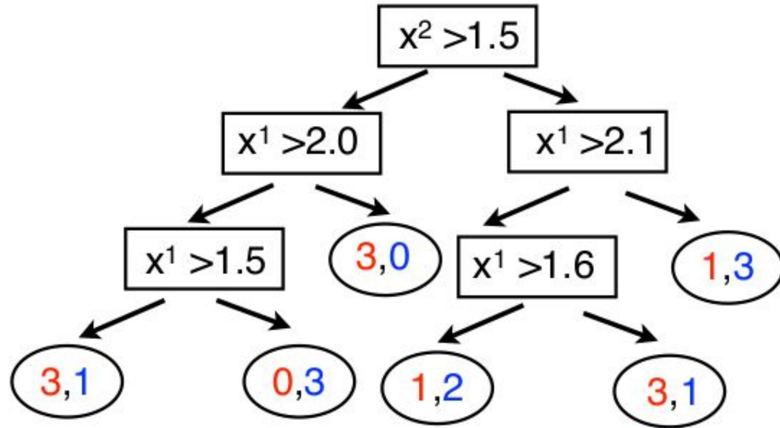
(x^1_1, x^2_1) classes

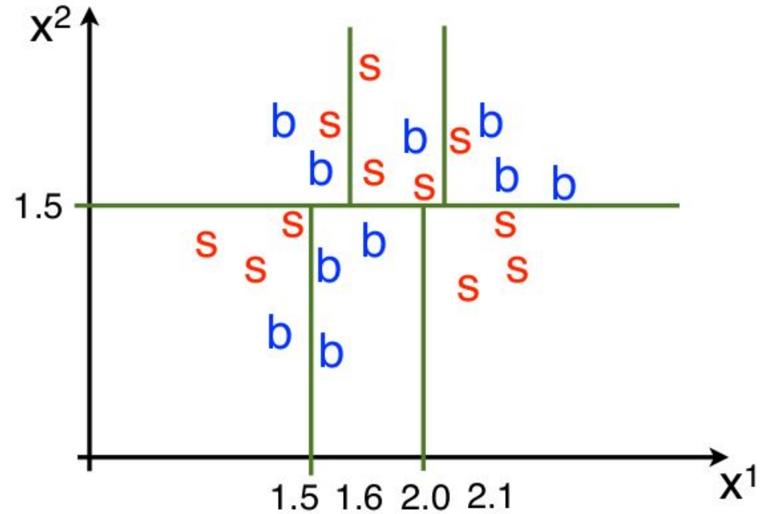
...

(x^1_i, x^2_i) b

...

(x^1_n, x^2_n) s





Strategy is to minimize the misclassification at each leaf

Decision trees : classification

Training sample $\in \Re^2$

(x^1_1, x^2_1) classes

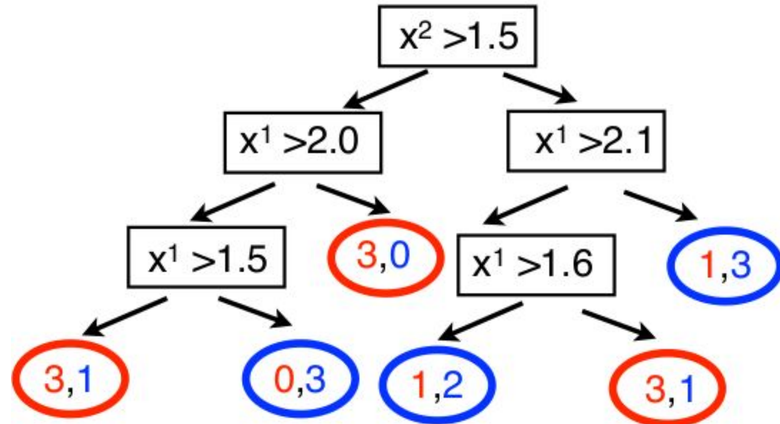
...

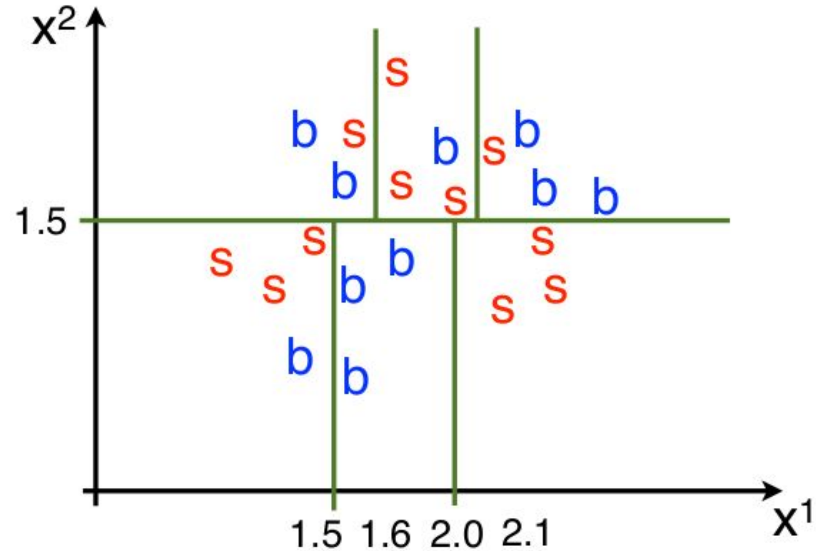
(x^1_i, x^2_i) b

...

(x^1_n, x^2_n) s

Build a binary tree:





Strategy is to minimize the misclassification at each leaf

Decision trees : classification

Training sample $\in \Re^2$

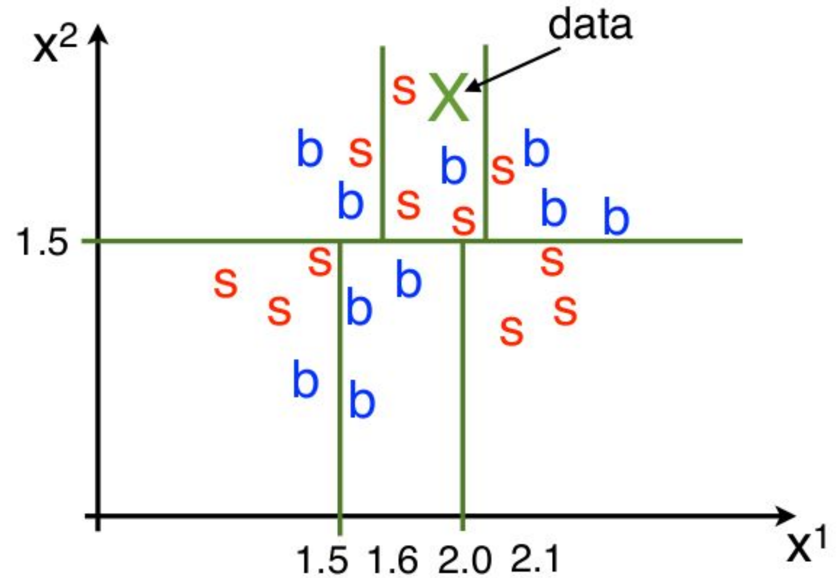
$(x^1{}_1, x^2{}_1)$ classes

...

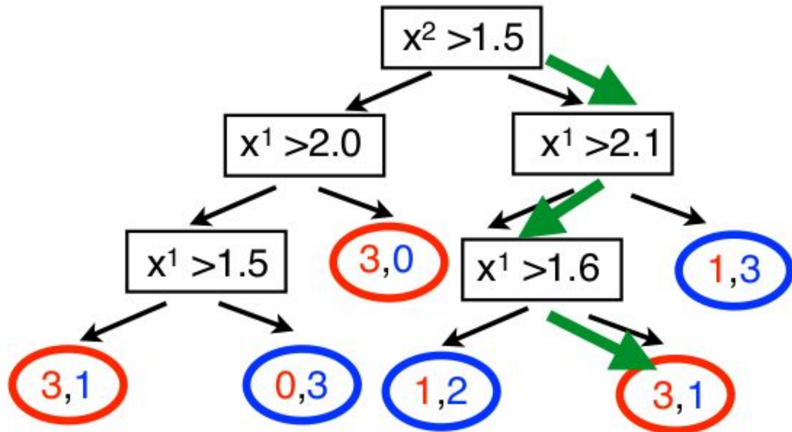
$(x^1{}_i, x^2{}_i)$ b

...

$(x^1{}_n, x^2{}_n)$ s



Build a binary tree:



Strategy is to minimize the misclassification at each leaf

Decision trees : regression

Training sample $\in \Re$

(x_1, y_1)

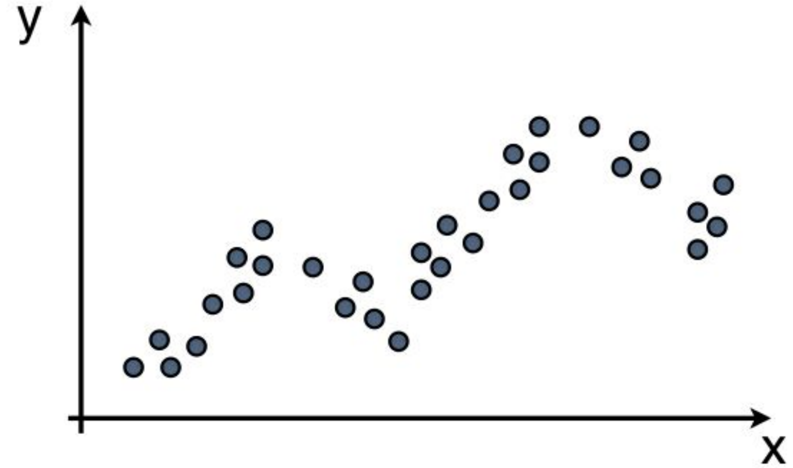
...

(x_i, y_i)

...

(x_n, y_n)

continuous target



Decision trees : regression

Training sample $\in \Re$

(x_1, y_1)

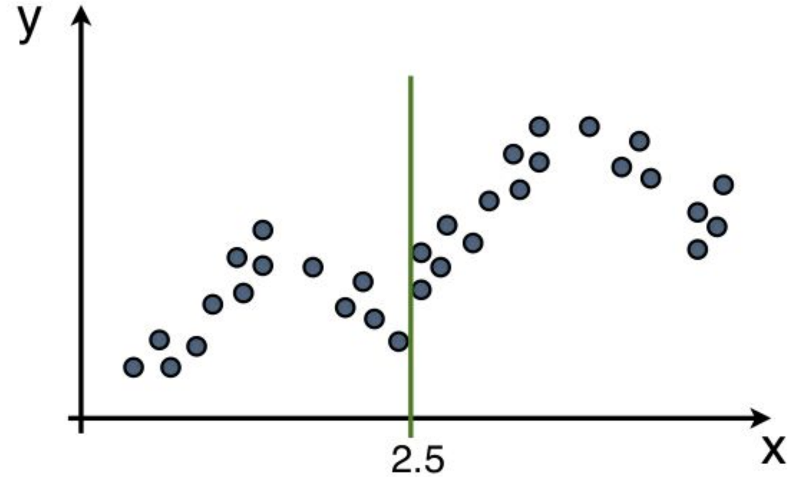
...

(x_i, y_i)

...

(x_n, y_n)

continuous target



Strategy is to minimize the error at each leaf

Decision trees : regression

Training sample $\in \Re$

(x_1,y_1)

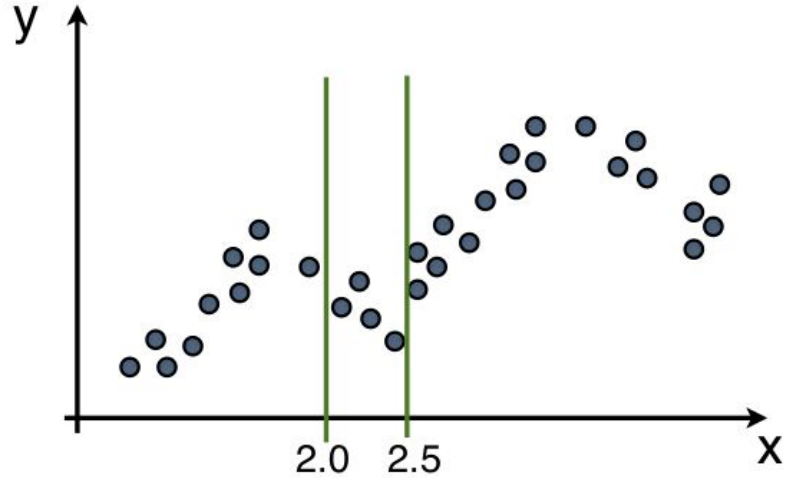
...

(x_i,y_i)

...

(x_n,y_n)

continuous target



Strategy is to minimize the error at each leaf

CMS

tifr

Decision trees : regression

Training sample $\in \Re$

(x_1, y_1)

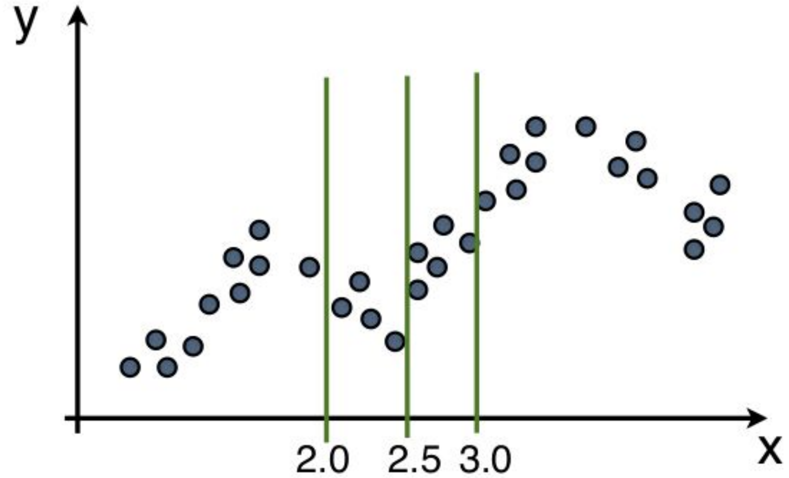
...

(x_i, y_i)

...

(x_n, y_n)

continuous target



Strategy is to minimize the error at each leaf

CMS

tifr

Decision trees : regression

Training sample $\in \Re$

(x_1, y_1)

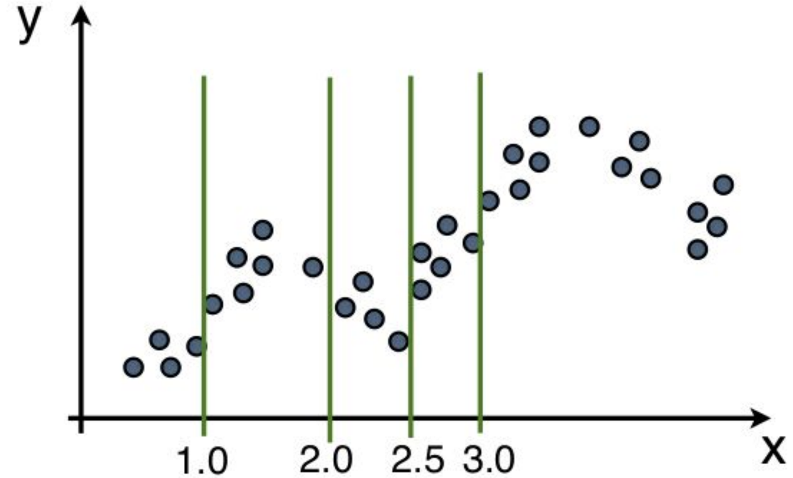
...

(x_i, y_i)

...

(x_n, y_n)

continuous target

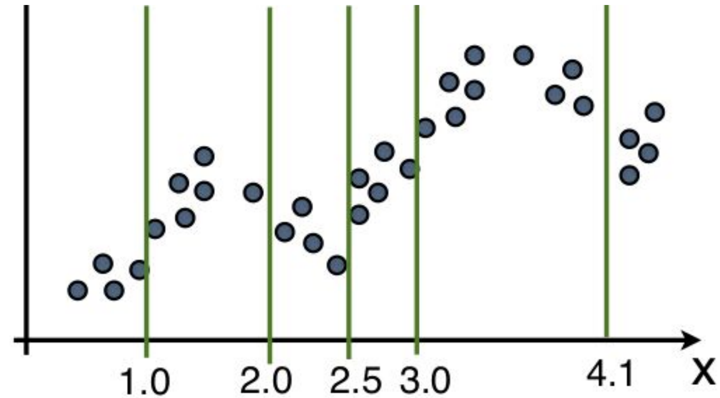


Strategy is to minimize the error at each leaf

Decision trees : regression

(x_1, y_1)
...
(x_i, y_i)
...
(x_n, y_n)

continuous target



Strategy is to minimize the error at each leaf

Decision trees : regression

Training sample $\in \Re$

(x_1, y_1)

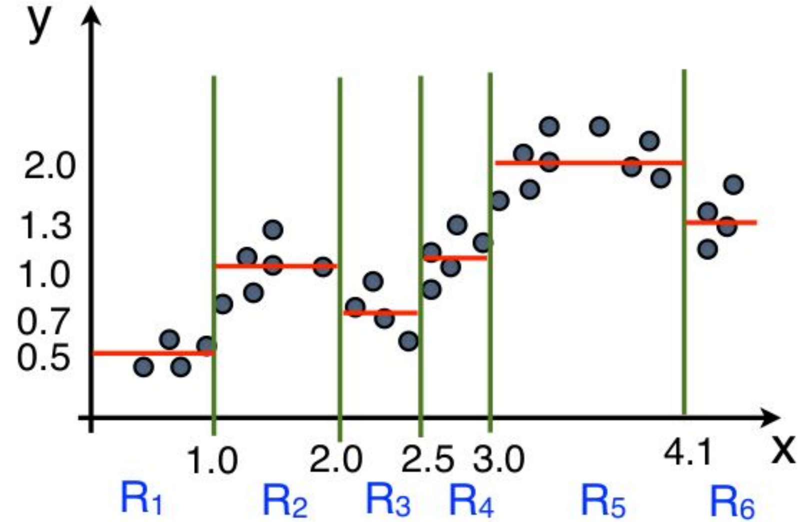
...

(x_i, y_i)

...

(x_n, y_n)

continuous target

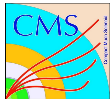


Strategy is to minimize the error at each leaf

Repeat until every region (leaf) contains a "minimum" number of points

Average of the points in each region

i.e. given x predict y



tifr

Decision trees : regression

Training sample $\in \Re$

(x_1, y_1)

...

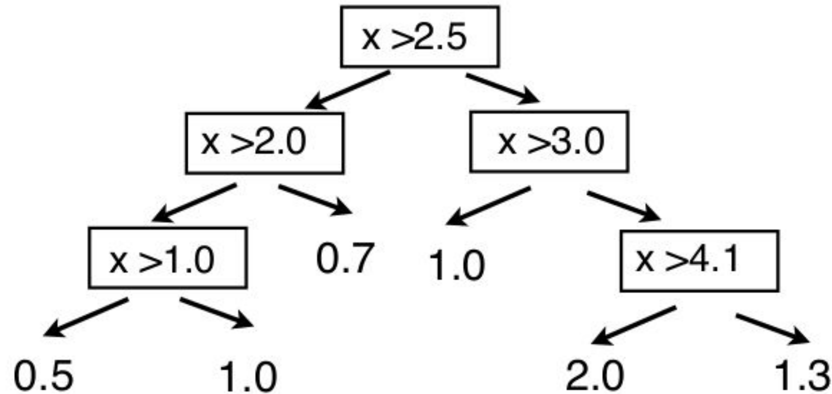
(x_i, y_i)

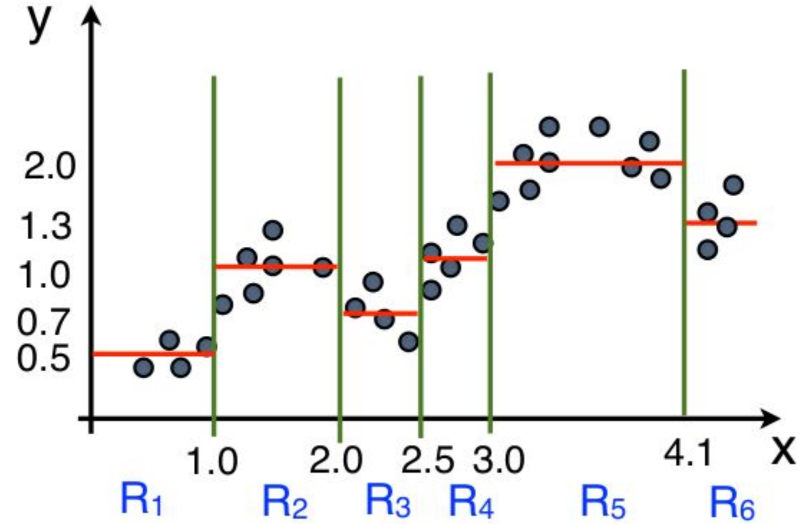
...

(x_n, y_n)

continuous target

Build a binary tree





Strategy is to minimize the error at each leaf

Average of the points in each region

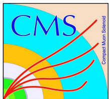
Issues with DTs

The variables and the order are chosen on the base of separation. So if you change the training sample you might get different trees.

- Whatever variable is the most discriminating it will influence the rest of the tree
- Decision trees tend to be very sensitive to statistical fluctuations of the training sample (noise).
- Decision trees are too unstable to be used safely.

Several aggregation techniques have been developed to improve the performance of the DT. (aggregating copies of the same tree)

- The most commonly used is BOOSTING (BDT).
- These techniques can be applied to classification and regression



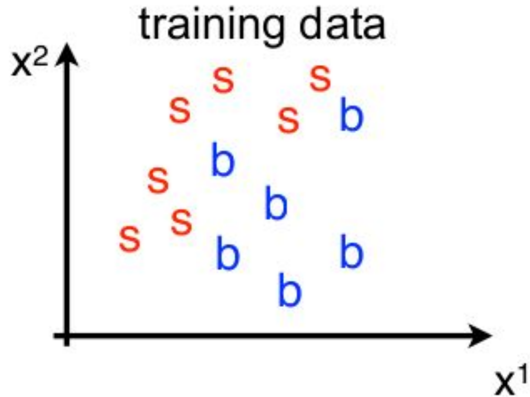
tifr

Boosting

Sequentially training a model learning from the errors of the previous ones.

The idea is to create modifications that give smaller error rates than those of the preceding classifiers.

(for graphical reasons I use 2 variables and a single level DT, i.e. one cut)



CMS

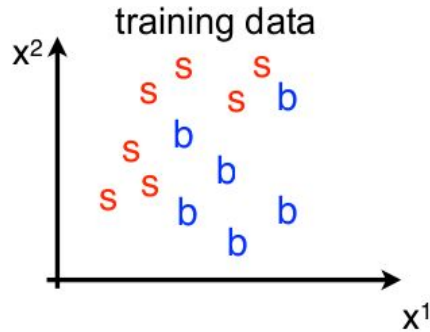
tifr

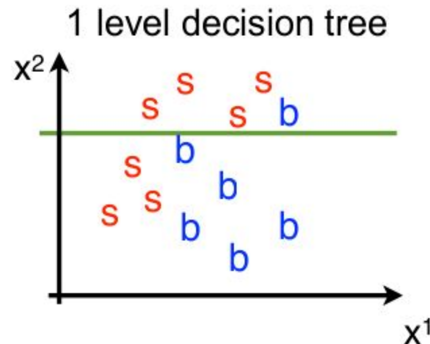
Boosting

Sequentially training a model learning from the errors of the previous ones.

The idea is to create modifications that give smaller error rates than those of the preceding classifiers.

(for graphical reasons I use 2 variables and a single level DT, i.e. one cut)



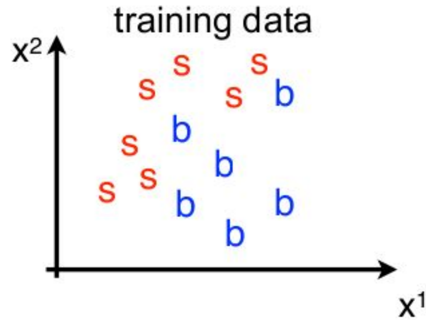


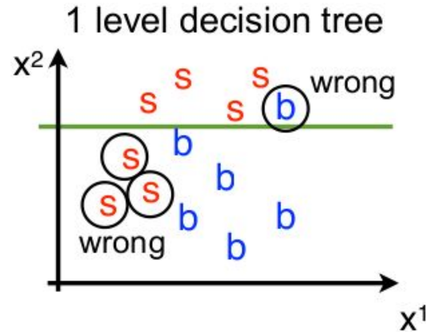
Boosting

Sequentially training a model learning from the errors of the previous ones.

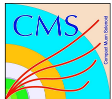
The idea is to create modifications that give smaller error rates than those of the preceding classifiers.

(for graphical reasons I use 2 variables and a single level DT, i.e. one cut)



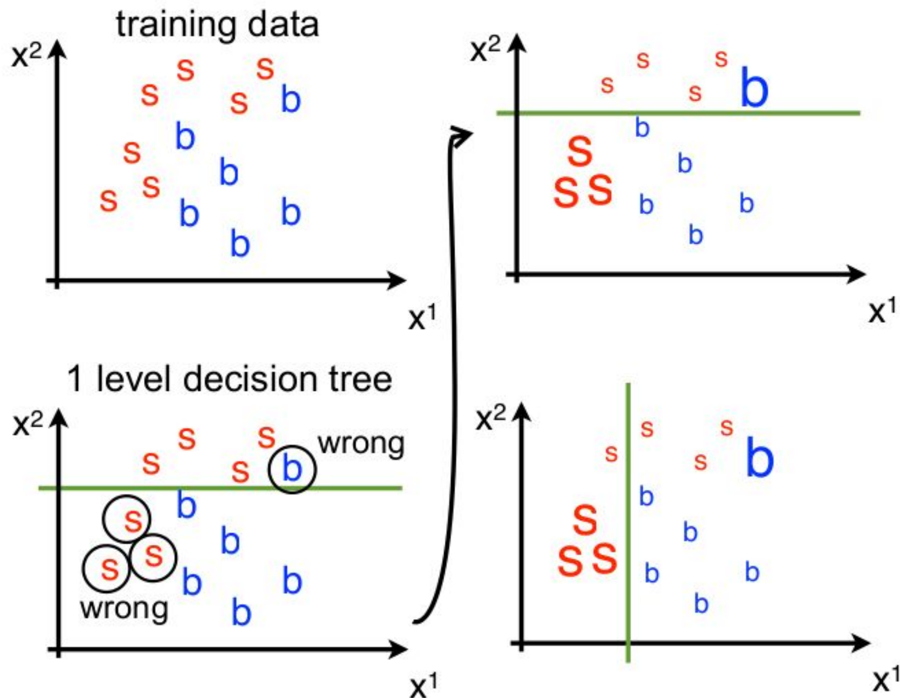


Focus on the 4 wrong ones



Boosting

Sequentially training a model learning from the errors of the previous ones. The idea is to create modifications that give smaller error rates than those of the preceding classifiers.



Right classification:
decrease weight

Wrong classification:
increase weight

it is more important to make this three “S” right than the other wrong

Boosting

Sequentially training a model learning from the errors of the previous ones. The idea is to create modifications that give smaller error rates than those of the preceding classifiers.

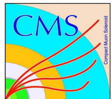
training data

1 level decision tree

wrong

wrong

It's essentially like keeping refitting the residuals



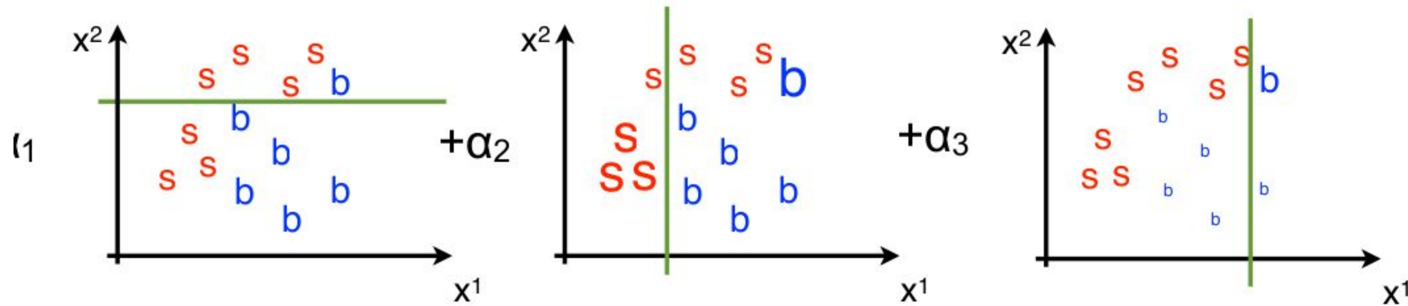
tifr

Boosting

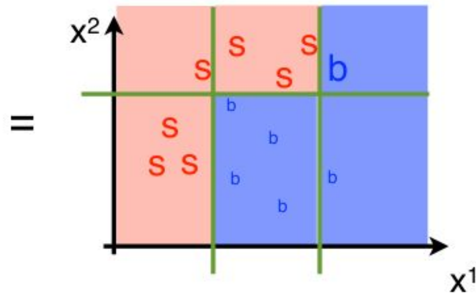
In practice:

assign numerical values to the two classes: b = +1 s =-1

assign a coefficient/"weight α_i" to each of the trees and sum them

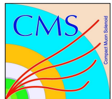


=



boosted tree

Each of these trees is a simple cut, the boosted version makes a non linear boundary decision



tifr

A flowchart of one possible implementation

For i = 1.. N_{boost}

{

c(i) = train $(\vec{x}, \vec{y}, \vec{w})$

$\hat{y}$ = predict $(c(i), \vec{x})$

Compute error

$e = \vec{w} * (\vec{y} == \hat{y})$

Set $\alpha_i = \frac{1}{2} \log \left(\frac{1-e}{e} \right)$

$$\vec{w} = \vec{w} e^{-\alpha_i (\vec{y}_i \cdot \hat{\vec{y}}_i)}$$

$$\vec{w} = \vec{w} / \sum (\vec{w})$$

}

"Sequential training"

c(i) = classifier/tree (i)
$\vec{x}$ = vector of variables in
$\vec{y}$ = vector of class/target out: two classes = +1, -1
$\vec{w}$ = vector of weights

initially set all weights to 1, then evolve them

e = scalar error = vector of weights * vector of 0s and 1s
correct/wrong

coefficient α at the step i: this is the coefficient for c(i)

Update the weights

(true,predict)

correct (s,s) or (b,b) ⇒ "+ sign"down-weighted
wrong (s,b) or (b,s) ⇒ "- sign" up-weighted

normalize by the sum of all weights

final classifier/tree = $\sum_i \alpha_i \, predict(c(i), x)$

tifr

CMS
Compact Muon Solenoid

tifr

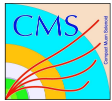
Objective function

- The objective of training is to find the best parameter θ_{ij} that best fits the training data x_i and predicts y_i
- **$obj(\theta) = L(\theta) + \Omega(\theta)$**
- $L(\theta)$: Loss function; $\Omega(\theta)$: Regularization term
- Choices of 'L' :
  - Mean squared error (MSE)
  - Binary logistic
- MSE is used for regression where y_i is continuous
- Binary logistic is used for classifier where y_i is discrete
- Regularisation term controls the complexity of the model; helps to avoid overfitting.

tifr

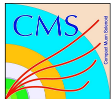
Difference between random forest & gradient boosting

- Difference between Random Forest & Gradient Boosting:
  - The main difference between random forest and gradient boosting is how the decision trees are created and aggregated.
  - Random forest builds a decision tree independently. Each decision tree is a predictor and their results are aggregated into a single result.
  - The decision trees in gradient boosting are built additively; one after another. Each tree is built to make up the deficiency of previous tree.
  - Another difference is : in random forest the result of the decision trees are aggregated at the end of the process. Gradient boosting aggregates the results of each decision tree to calculate the final result.
  - Gradient boosting generally performs better than random forest, but there may be an overfitting issue with gradient boosting.



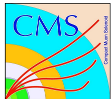
tifr

XGBOOST



Supervised learning

- We use training data x_i to predict target variable y_i
- Main mathematical form of supervised learning:
  - $\mathbf{y_i = \Sigma\, \theta_{ij}\, x_j}$
  - A linear combination of the weighted input features
  - θ_{ij} is called the objective function
- Objective function consists of two terms
  - Loss function
  - Regularization term



tifr

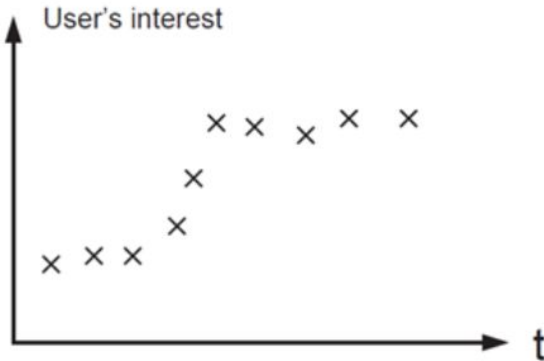
Decision tree ensembles

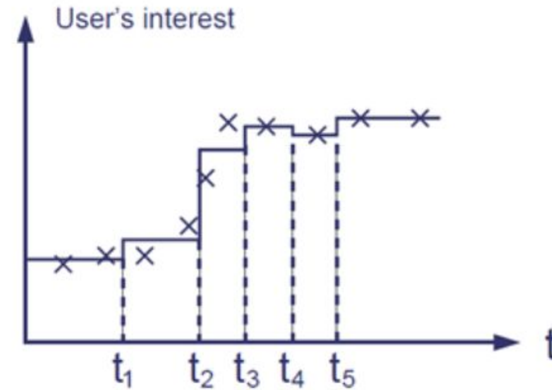
- Model choice of xgboost: decision tree ensembles
  - **Decision tree ensembles --> set of classification and regression trees (CART)**
- Tree boosting
  - The main thing of supervised learning is : define an objective function and optimize it

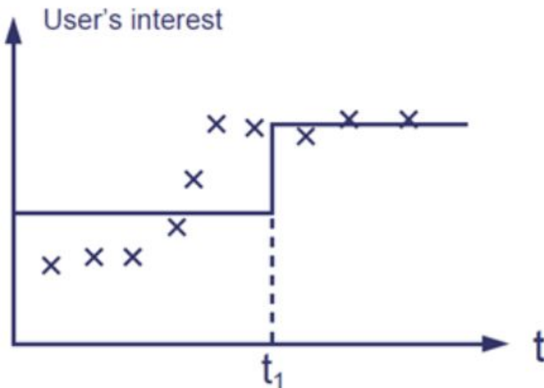
$$\text{obj} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \omega(f_i)$$

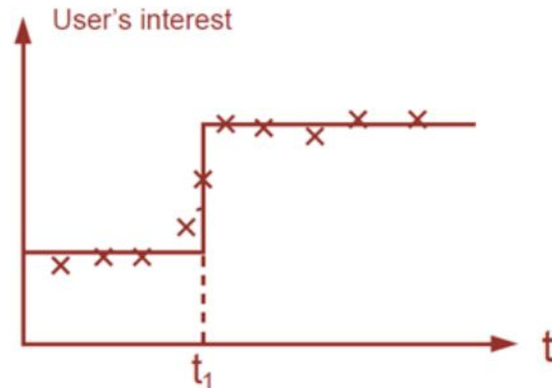
  - We use an additive strategy here; fix what we have learned and add one new tree at a time.

bias - variance tradeoff





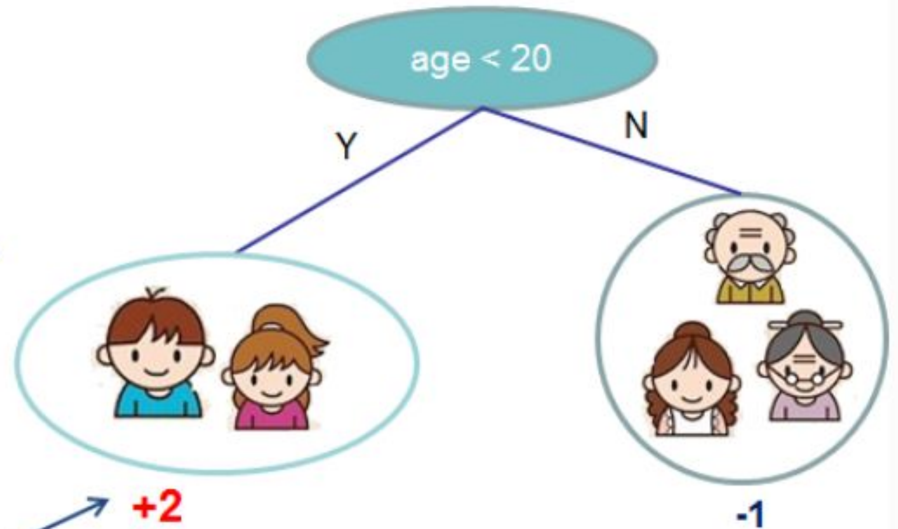




Decision tree ensemble

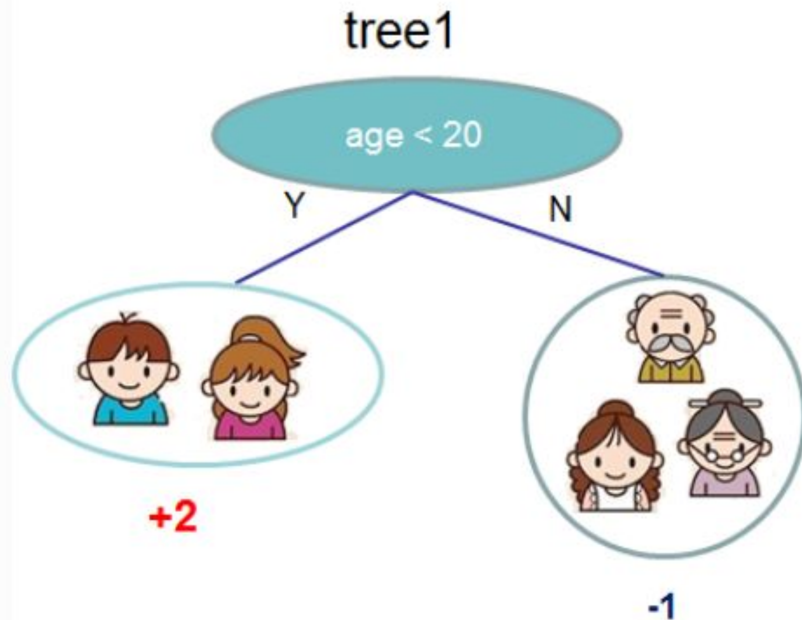
Input: age, gender, occupation, …

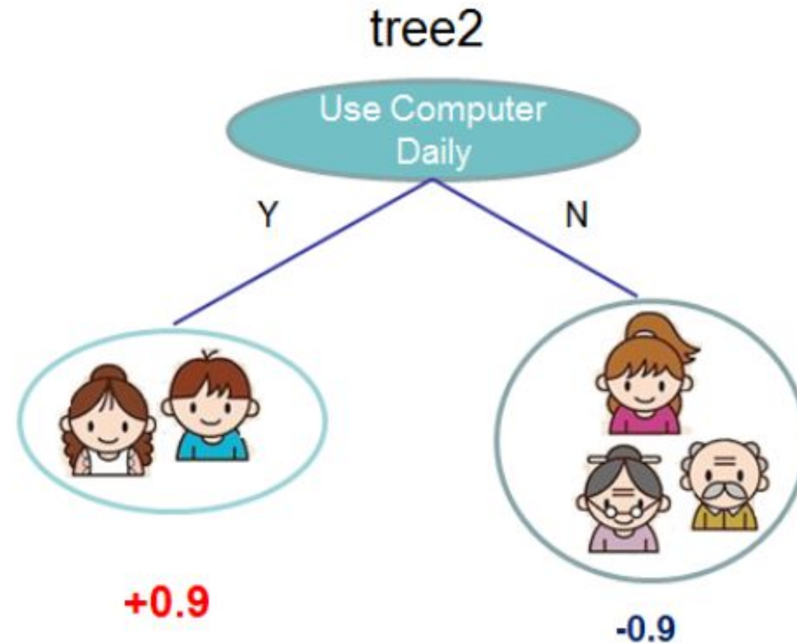
Like the computer game X



prediction score in each leaf

Decision tree ensemble





f( ) = 2 + 0.9= 2.9 f( )= -1 - 0.9= -1.9

Additive strategy of xgboost

$$
\begin{aligned}
\hat{y}_i^{(0)} &= 0 \\
\hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\
\hat{y}_i^{(2)} &= f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \\
&\dots \\
\hat{y}_i^{(t)} &= \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)
\end{aligned}
$$

- Our objective function is :
  -
$$
\begin{aligned}
\text{obj}^{(t)} &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \omega(f_i) \\
&= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \omega(f_t) + \text{constant}
\end{aligned}
$$

tifr

Objective function

- If we consider mean square error (MSE) as loss function:

$$\text{obj}^{(t)} = \sum_{i=1}^{n}(y_i - (\hat{y}_i^{(t-1)} + f_t(x_i)))^2 + \sum_{i=1}^{t}\omega(f_i)$$
$$= \sum_{i=1}^{n}[2(\hat{y}_i^{(t-1)} - y_i)f_t(x_i) + f_t(x_i)^2] + \omega(f_t) + \text{constant}$$

- If we consider logistic loss, then we generally expand it in Taylor series upto second order:

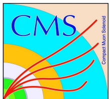
$$\text{obj}^{(t)} = \sum_{i=1}^{n}[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)] + \omega(f_t) + \text{constant}$$

where the g_i and h_i are defined as

$$g_i = \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$$
$$h_i = \partial^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$$

After we remove all the constants, the specific objective at step t becomes

$$\sum_{i=1}^{n}[g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)] + \omega(f_t)$$



tifr

Model Complexity

We have introduced the training step, but wait, there is one important thing, the **regularization term**! We need to define the complexity of the tree $\omega(f)$. In order to do so, let us first refine the definition of the tree $f(x)$ as

$$f_t(x) = w_{q(x)}, w \in R^T, q : R^d \to \{1, 2, \cdots, T\}.$$

Here w is the vector of scores on leaves, q is a function assigning each data point to the corresponding leaf, and T is the number of leaves. In XGBoost, we define the complexity as

$$\omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$