Tree models

Andrey Filatov [1] Vladimir Bocharnikov [2]

[1]MIPT [2]DESY

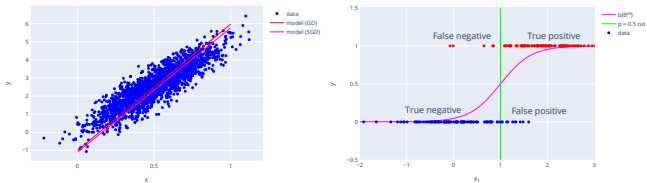
Outline

- Decision tree
- Random forest
- Gradient boosting

Decision tree

Decision tree

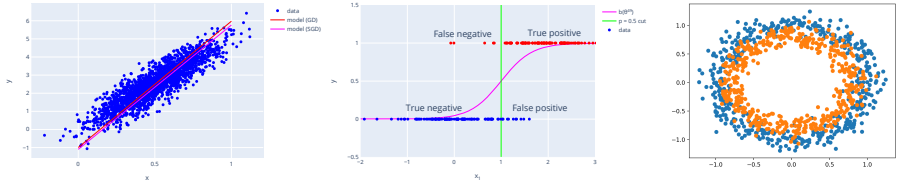
motivation



- In the previous lecture we explored a family of linear models:
  - → nicely describe linear correlations
  - → fast and easy to train
  - → performing well with large number of samples/features

Decision tree

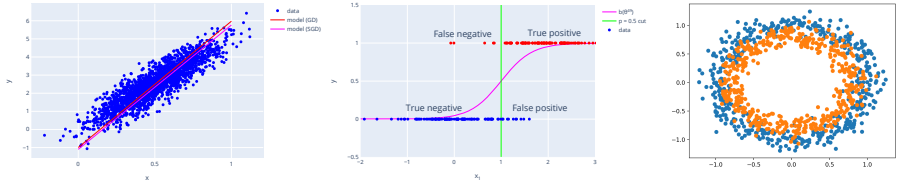
motivation



- In the previous lecture we explored a family of linear models:
  - → nicely describe linear correlations
  - → fast and easy to train
  - → performing well with large number of samples/features
- However, they are poor in modelling non-linearities
- Additional heuristics might be applied (e.g. polynomial features) but are ad-hoc

Decision tree

motivation



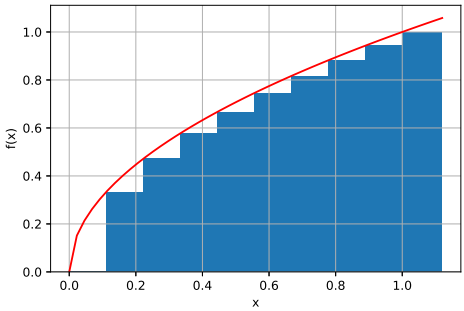
- In the previous lecture we explored a family of linear models:
  - → nicely describe linear correlations
  - → fast and easy to train
  - → performing well with large number of samples/features
- However, they are poor in modelling non-linearities
- Additional heuristics might be applied (e.g. polynomial features) but are ad-hoc

Is there a more general way to describe non-linearities?

Decision tree

motivation

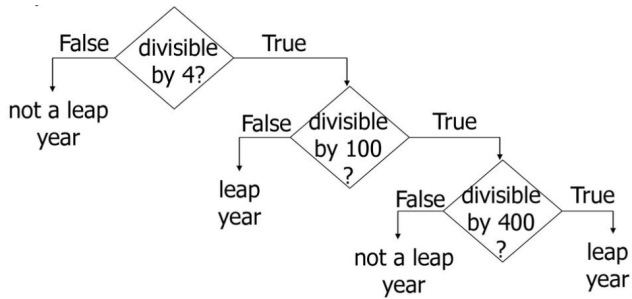
- What is the simplest way to approximate a function?
- Using **piecewise linear function**



$$f(x) = \begin{cases} 0, & x < 0.12 \\ 0.35, & 0.12 \le x < 0.22 \\ 0.47, & 0.22 \le x < 0.36 \\ \dots \end{cases}$$

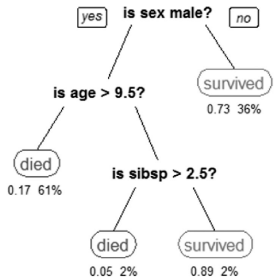
Decision tree — intuition

- What is the simplest way to categorize things?
- Ask **yes/no questions**



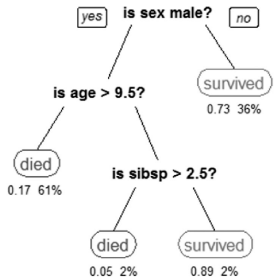
Decision tree — intuition

- What is the simplest way to categorize things?
- Ask **yes/no questions**



Decision tree — intuition

- What is the simplest way to categorize things?
- Ask **yes/no questions**



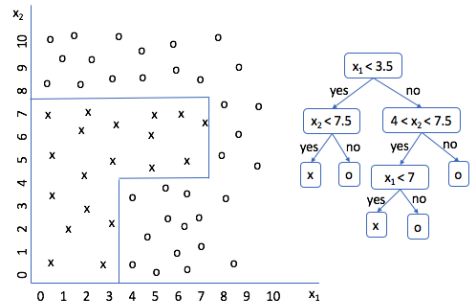
Can we mathematically formulate this?

Decision tree — algorithm

Algorithm 1: Decision tree

1: Initialize: hyperparameters, leaf set = $\{X_{\text{train}}\}$

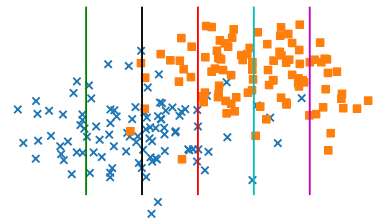
2: **while** not stopping criteria **do**
3: leaf to split = Choose(leaf set)
4: left leaf, right leaf = Split(leaf to split)
5: Add left leaf, right leaf to leaf set
6: Remove leaf to split from leaf set
7: **end while**



Decision tree

split criteria

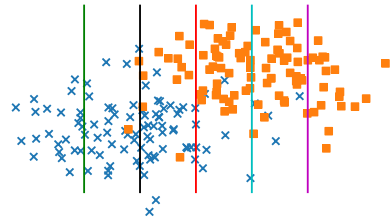
- Suppose there is a classification problem
- So we want to separate one class from the other by constructing **decision boundary**
- And using decision tree algorithm described earlier
- To do that we need to start splitting on features
- Which **split** is the best?



Decision tree — split criteria

- Suppose there is a classification problem
- So we want to separate one class from the other by constructing **decision boundary**
- And using decision tree algorithm described earlier
- To do that we need to start splitting on features
- Which **split** is the best?

→ We need some criteria



Decision tree — split criteria

more about IG

- Suppose we have n features $x^{(1)}, x^{(2)}, ..., x^{(n)}$ in our dataset X
- If we want to split on some feature $x^{(j)}$ at threshold t then we propose to maximize **Information Gain** (IG):

$$\mathrm{IG}(\mathrm{X}, \mathrm{a}) = \mathrm{H}(\mathrm{X}) - \mathrm{H}(\mathrm{X} \mid \mathrm{a}) \to \max_{\mathrm{a}}$$

$$\mathrm{H}(\mathrm{X} \mid \mathrm{a}) = \sum_{v \in vals(\mathrm{a})} \frac{|S_{\mathrm{a}}(v)|}{|\mathrm{X}|} \cdot \mathrm{H}\left(S_{\mathrm{a}}(v)\right)$$

- In case of a binary tree with L(R) elements in the left (right) split we've got:

$$\mathrm{IG}(j, t) = \frac{|\mathrm{L}|}{|\mathrm{X}|}\mathrm{H}(\mathrm{L}) + \frac{|\mathrm{R}|}{|\mathrm{X}|}\mathrm{H}(\mathrm{R}) \to \min_{j,t}$$

Decision tree — split criteria

more about IG

- Suppose we have n features $x^{(1)}, x^{(2)}, ..., x^{(n)}$ in our dataset X
- If we want to split on some feature $x^{(j)}$ at threshold t then we propose to maximize **Information Gain** (IG):

$$\mathrm{IG}(\mathrm{X}, \mathrm{a}) = \mathrm{H}(\mathrm{X}) - \mathrm{H}(\mathrm{X} \mid \mathrm{a}) \to \max_{\mathrm{a}}$$

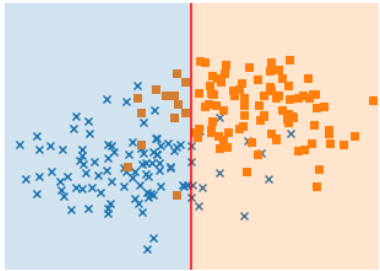
$$\mathrm{H}(\mathrm{X} \mid \mathrm{a}) = \sum_{v \in vals(\mathrm{a})} \frac{|S_{\mathrm{a}}(v)|}{|\mathrm{X}|} \cdot \mathrm{H}\left(S_{\mathrm{a}}(v)\right)$$

- In case of a binary tree with L(R) elements in the left (right) split we've got:

$$\mathrm{IG}(j, t) = \frac{|\mathrm{L}|}{|\mathrm{X}|}\mathrm{H}(\mathrm{L}) + \frac{|\mathrm{R}|}{|\mathrm{X}|}\mathrm{H}(\mathrm{R}) \to \min_{j,t}$$

How to choose H(X)?

Decision tree — split criteria



The ”best” split is a central one because it introduces **purity** in the best way. And we have some functions to measure the purity!

Decision tree — split criteria

- Define H(R) in a leaf as a goodness of approximation with the ″best″ constant c

$$\mathrm{H}(\mathrm{R}) = \min_{c \in \mathbb{Y}} \frac{1}{|N|} \sum_{(x_i, y_i) \in \mathrm{R}} \mathcal{L}(y_i, c)$$

Decision tree — split criteria

- Define H(R) in a leaf as a goodness of approximation with the "best" constant c

$$\mathrm{H}(\mathrm{R}) = \min_{c \in \mathbb{Y}} \frac{1}{|N|} \sum_{(x_i, y_i) \in \mathrm{R}} \mathcal{L}(y_i, c)$$

- or c_k with $\sum_k c_k = 1$, if tree predicts class probabilities

Decision tree — split criteria

- Define H(R) in a leaf as a goodness of approximation with the "best" constant c

$$\mathrm{H(R)} = \min_{c \in \mathbb{Y}} \frac{1}{|N|} \sum_{(x_i, y_i) \in \mathrm{R}} \mathcal{L}(y_i, c)$$

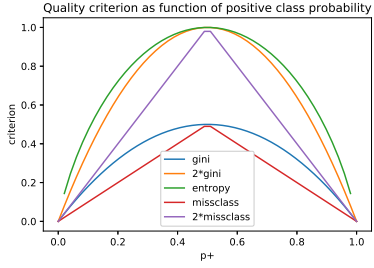
- or c_k with $\sum_k c_k = 1$, if tree predicts class probabilities
- Define a fraction of class k in a leaf as p_k

Decision tree — split criteria

- Define H(R) in a leaf as a goodness of approximation with the "best" constant c

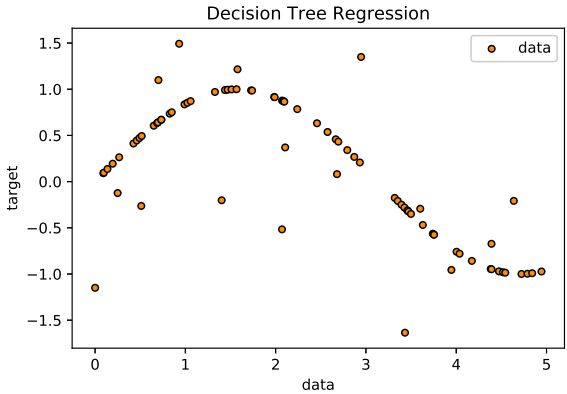
$$\mathrm{H(R)} = \min_{c \in \mathbb{Y}} \frac{1}{|N|} \sum_{(x_i, y_i) \in \mathrm{R}} \mathcal{L}(y_i, c)$$

- or c_k with $\sum_k c_k = 1$, if tree predicts class probabilities
- Define a fraction of class k in a leaf as p_k
- Then one can show:
  - **Misclassification criteria**
    $\mathcal{L}(y, c) = [y \neq c] \Rightarrow \mathrm{H(R)} = 1 - \max_k \{p_k\}$
  - **Gini Impurity**
    $\mathcal{L}(y, c) = \sum_k (c_k - [y = k])^2 \Rightarrow \mathrm{H(R)} = 1 - \sum_k (p_k)^2$
  - **Entropy criteria**
    $\mathcal{L}(y, c) = -\sum_k [y = k] \log c_k \Rightarrow \mathrm{H(R)} = -\sum_k p_k \log_2 p_k$



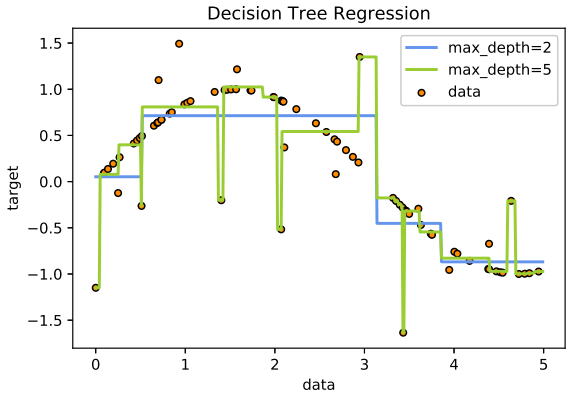
Lower diversity (higher purity)
$\Rightarrow$ **lower impurity** criterion H(R)

Decision tree — regression



Can we use decision tree approach for regression problem?

Decision tree — regression



Can we use decision tree approach for regression problem? Yes

Decision tree — regression

- How should we modify decision criteria?

$$\mathrm{IG}(j,t) = \frac{|\mathrm{L}|}{|\mathrm{Q}|}\mathrm{H}(\mathrm{L}) + \frac{|\mathrm{R}|}{|\mathrm{Q}|}\mathrm{H}(\mathrm{R})$$

→ Mean squared error: $\mathcal{L}(y,c) = (y-c)^2 \Rightarrow \mathrm{H}(\mathrm{R}) = \min_c \frac{1}{|\mathrm{R}|}\sum_{(x_i,y_i)\in \mathrm{R}} (y_i - c)^2$

→ Mean absolute error: $\mathcal{L}(y,c) = |y-c| \Rightarrow \mathrm{H}(\mathrm{R}) = \min_c \frac{1}{|\mathrm{R}|}\sum_{(x_i,y_i)\in \mathrm{R}} |y_i - c|$

Decision tree — regression

- How should we modify decision criteria?

$$\mathrm{IG}(j,t) = \frac{|\mathrm{L}|}{|\mathrm{Q}|}\mathrm{H}(\mathrm{L}) + \frac{|\mathrm{R}|}{|\mathrm{Q}|}\mathrm{H}(\mathrm{R})$$

→ Mean squared error: $\mathcal{L}(y,c) = (y-c)^2 \Rightarrow \mathrm{H}(\mathrm{R}) = \min_c \frac{1}{|\mathrm{R}|}\sum_{(x_i,y_i)\in \mathrm{R}} (y_i - c)^2$

→ Mean absolute error: $\mathcal{L}(y,c) = |y-c| \Rightarrow \mathrm{H}(\mathrm{R}) = \min_c \frac{1}{|\mathrm{R}|}\sum_{(x_i,y_i)\in \mathrm{R}} |y_i - c|$

- One can further show that:

$$c^* = \frac{1}{|\mathrm{R}|}\sum_{y_i \in \mathrm{R}} y_i$$

$$c^* = \mathrm{median}(y_i \in \mathrm{R})$$

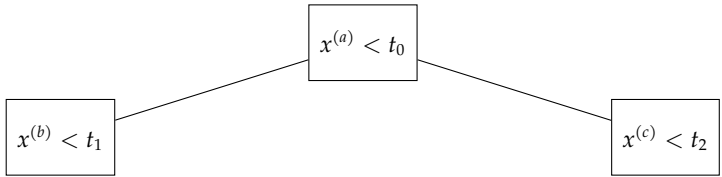
Decision tree — growing

What about tree construction?

$$x^{(a)} < t_0$$

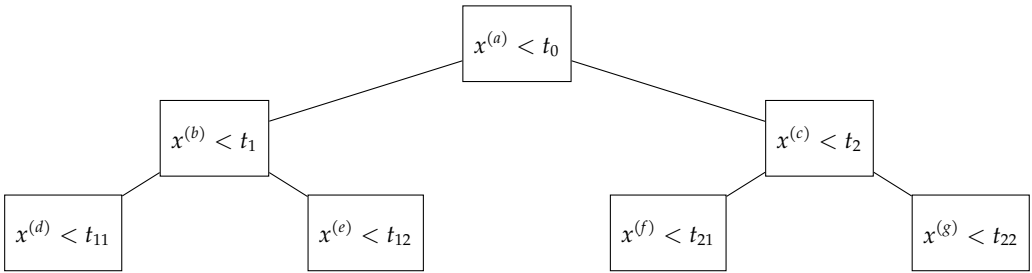
Decision tree — growing

What about tree construction? Do it recursively by **greedy** algorithm



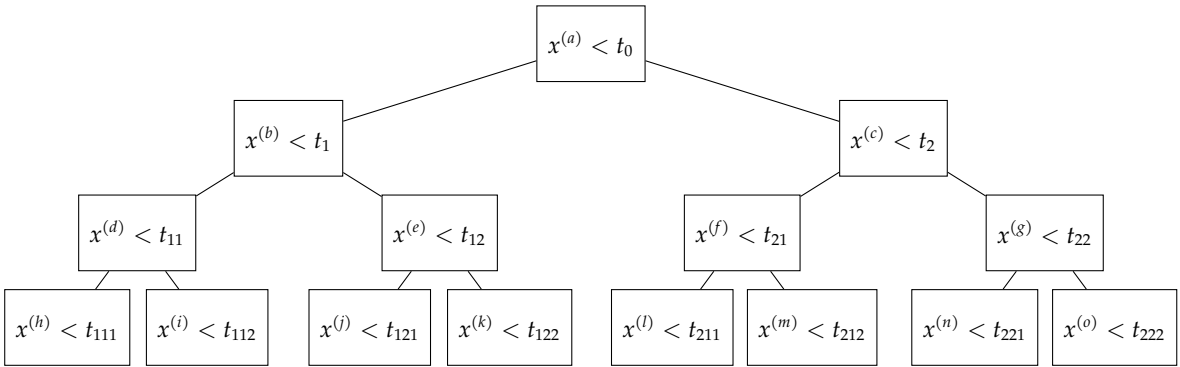
Decision tree — growing

What about tree construction? Do it recursively by **greedy** algorithm



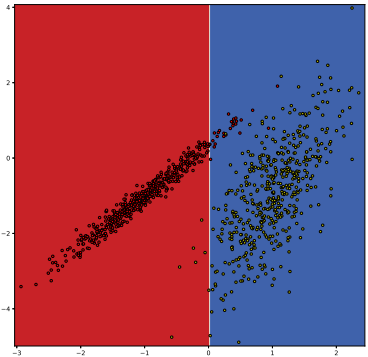
Decision tree — growing

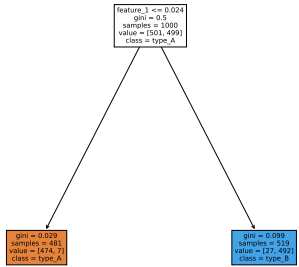
What about tree construction? Do it recursively by **greedy** algorithm



Decision tree — growing

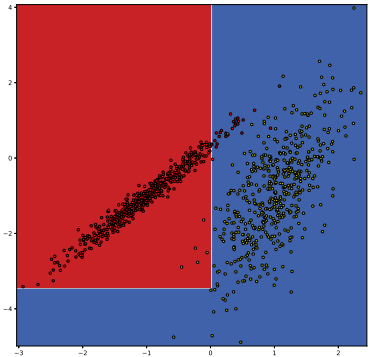
check out also these visuals

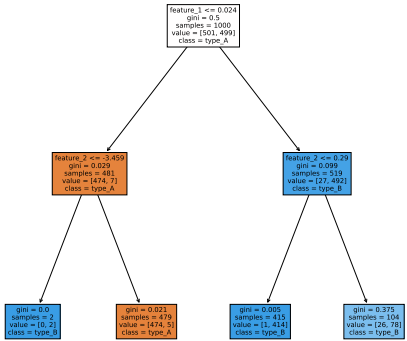




Decision tree — growing

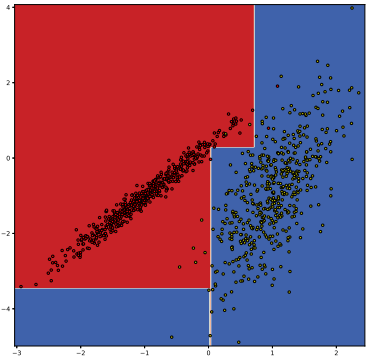
check out also these visuals

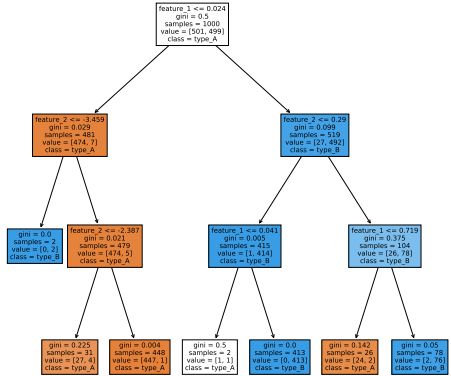




Decision tree — growing

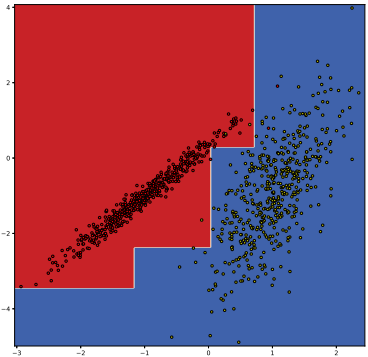
check out also these visuals

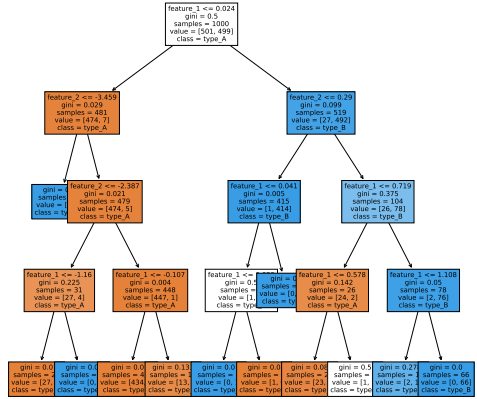




Decision tree — growing

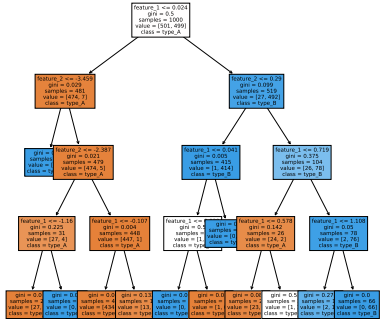
check out also these visuals





Decision tree — stopping criteria

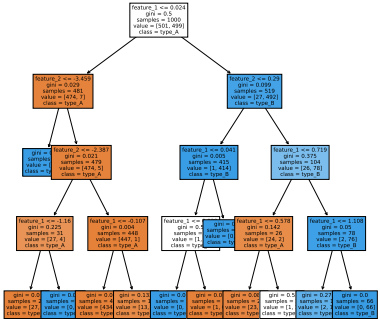
- What about stopping criteria?
  1. Split until one element in leaf
     (which problems can we meet?)
  2. Limit max depth/max number of leaves
  3. Limit amount of elements in leaf/split
  4. All elements in a leaf belong to a single class
  5. IG(j, t) isn't improving by x%
  6. Combination of limitations



Decision tree — stopping criteria

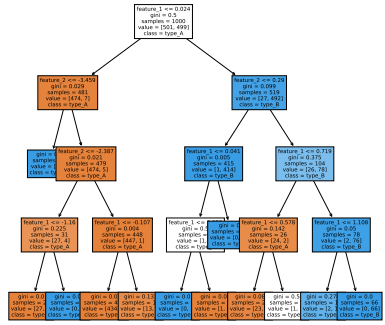
- What about stopping criteria?
  1. Split until one element in leaf
     (which problems can we meet?)
  2. Limit max depth/max number of leaves
  3. Limit amount of elements in leaf/split
  4. All elements in a leaf belong to a single class
  5. IG(j, t) isn't improving by x%
  6. Combination of limitations

→ The choice is up to you
(aka hyperparameter)



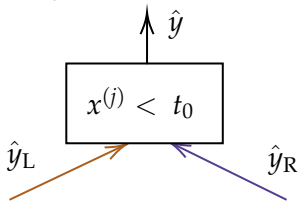
Decision tree ⟶ prediction

1. Once tree is built, look at the training samples in each leaf
2. Assign to every leaf a value according to:
   - Classification: majority vote
     ➔ assign the most frequent class
   - Classification: class probabilities
     ➔ approximate with class fractions
   - Regression: mean, median over the values in a leaf
3. During inference step, propagate every sample through the tree
4. Assign a value of the leaf where the sample ended up



Can already trained decision tree handle **missing values** on inference step?

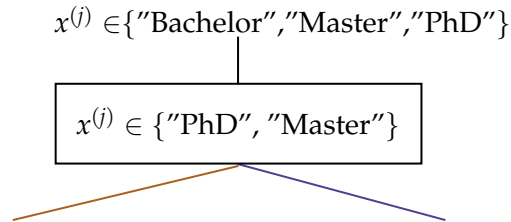
Can already trained decision tree handle **missing values** on inference step? Yes!



$$\hat{y} = \frac{|L|}{|X|}\hat{y}_L + \frac{|R|}{|X|}\hat{y}_R$$

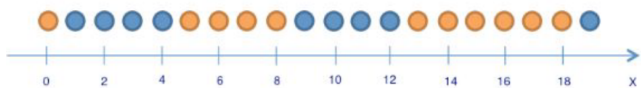
Categorical features can also easily be treated by tree:

- If categorical feature is selected at particular node
- Loop through all combination of its values
- Pick those set of values which result in best IG
- Split the leaf as before



Decision tree

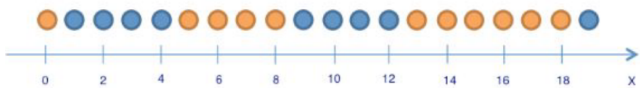
check your understanding



- What is the entropy of the distribution on the figure?
- What task do we solve to make a split?
- How many hyperparameters of decision tree do we know?
- How is the tree construsted?
- Which stopping criteria can we use?

Decision tree

check your understanding



- $\frac{11}{20} \log \frac{11}{20} + \frac{9}{20} \log \frac{9}{20}$
- Maximization of Information Gain
- Stopping criteria hyperparameters, Splitting criteria
- Recursively by greedy algorithm
- Max depth, Min elements in leaf, Min elements to make split, Combination

Random forest

Random forest — theoretical insight

more in this lecture

- Consider dataset X containing M objects $\boldsymbol{x_m}$ with targets y_m:

$$X = \{(\boldsymbol{x_m}, y_m),\ m = 1 \dots M\}$$

Random forest

theoretical insight

more in this lecture

- Consider dataset X containing M objects $\boldsymbol{x_m}$ with targets y_m:

$$X = \{(\boldsymbol{x_m}, y_m),\ m = 1 \dots M\}$$

- Pick M objects with return from X and repeat in N times

Random forest

theoretical insight

more in this lecture

- Consider dataset X containing M objects $\boldsymbol{x_m}$ with targets y_m:

$$X = \{(\boldsymbol{x_m}, y_m),\ m = 1 \dots M\}$$

- Pick M objects with return from X and repeat in N times
- This results in N datasets $\{X_i,\ i = 1, \dots N\}$ ➔ **bootstrapping**

Random forest

theoretical insight

more in this lecture

- Consider dataset X containing M objects $\boldsymbol{x_m}$ with targets y_m:

$$X = \{(\boldsymbol{x_m}, y_m),\ m = 1 \dots M\}$$

- Pick M objects with return from X and repeat in N times
- This results in N datasets $\{X_i,\ i = 1, \dots N\}$ ➔ **bootstrapping**
- Error of base algorithm $b_i(\boldsymbol{x})$ trained on X_i:

$$\varepsilon_i(\boldsymbol{x}) \equiv b_i(\boldsymbol{x}) - y(\boldsymbol{x}), \quad i = 1, \dots, N$$

Random forest

theoretical insight

more in this lecture

- Consider dataset X containing M objects $\boldsymbol{x_m}$ with targets y_m:

$$X = \{(\boldsymbol{x_m}, y_m),\ m = 1 \dots M\}$$

- Pick M objects with return from X and repeat in N times
- This results in N datasets $\{X_i,\ i = 1, \dots N\}$ ➔ **bootstrapping**
- Error of base algorithm $b_i(\boldsymbol{x})$ trained on X_i:

$$\varepsilon_i(\boldsymbol{x}) \equiv b_i(\boldsymbol{x}) - y(\boldsymbol{x}), \quad i = 1, \dots, N$$

- Expected mean squared error (MSE) for $b_i(\boldsymbol{x})$ on dataset X_i:

$$\mathbb{E}_{\boldsymbol{x}} \left(b_i(\boldsymbol{x}) - y(\boldsymbol{x})\right)^2 = \mathbb{E}_{\boldsymbol{x}} \varepsilon_i^2(\boldsymbol{x})$$

Random forest

theoretical insight

more in this lecture

- Consider dataset X containing M objects $\boldsymbol{x_m}$ with targets y_m:

$$X = \{(\boldsymbol{x_m}, y_m),\ m = 1 \dots M\}$$

- Pick M objects with return from X and repeat in N times
- This results in N datasets $\{X_i,\ i = 1, \dots N\}$ ➔ **bootstrapping**
- Error of base algorithm $b_i(\boldsymbol{x})$ trained on X_i:

$$\varepsilon_i(\boldsymbol{x}) \equiv b_i(\boldsymbol{x}) - y(\boldsymbol{x}), \quad i = 1, \dots, N$$

- Expected mean squared error (MSE) for $b_i(\boldsymbol{x})$ on dataset X_i:

$$\mathbb{E}_{\boldsymbol{x}} \left(b_i(\boldsymbol{x}) - y(\boldsymbol{x})\right)^2 = \mathbb{E}_{\boldsymbol{x}} \varepsilon_i^2(\boldsymbol{x})$$

- Average error of base algorithms $b_i(\boldsymbol{x})$:

$$\mathcal{E} \equiv \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{\boldsymbol{x}} \varepsilon_i^2(\boldsymbol{x})$$

Random forest — theoretical insight

- Assume errors to be unbiased and uncorrelated:

$$\mathbb{E}_x \varepsilon_i(\boldsymbol{x}) = 0, \quad \forall i$$

$$\mathbb{E}_{\boldsymbol{x}}[\varepsilon_i(\boldsymbol{x})\varepsilon_j(\boldsymbol{x})] = 0, \quad i \neq j$$

- Let's construct an algorithm so that it averages predictions of base algorithms:

$$a(\boldsymbol{x}) = \frac{1}{N}\sum_{i=1}^{N} b_i(\boldsymbol{x})$$

Random forest — theoretical insight

but there's also bias-variance trade-off

- Then the error of this algorithm:

$$\mathcal{E}_N \equiv \mathbb{E}_{\boldsymbol{x}} \left(a(\boldsymbol{x}) - y(\boldsymbol{x})\right)^2 = \mathbb{E}_{\boldsymbol{x}} \left(\frac{1}{N}\sum_{i=1}^{N} b_i(\boldsymbol{x}) - y(\boldsymbol{x})\right)^2 =$$

$$= \mathbb{E}_{\boldsymbol{x}} \left(\frac{1}{N}\sum_{i=1}^{N} \varepsilon_i(\boldsymbol{x})\right)^2 =$$

$$= \frac{1}{N^2}\mathbb{E}_{\boldsymbol{x}} \left(\sum_{i=1}^{N} \varepsilon_i^2(\boldsymbol{x}) + \underbrace{\sum_{i\neq j} \varepsilon_i(\boldsymbol{x})\varepsilon_j(\boldsymbol{x})}_{=0}\right) = \frac{1}{N}\frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{\boldsymbol{x}}\varepsilon_i^2(\boldsymbol{x}) = \frac{\mathcal{E}}{N}$$

→ Composition of N algorithms has N times smaller error!

Random forest

how to achieve these assumptions?

$\mathbb{E}_x \varepsilon_i(\boldsymbol{x}) = 0$

➔ Strong models provide low bias $\Rightarrow$ train *overfitted* models $\Rightarrow$ **decision trees**

Random forest

how to achieve these assumptions?

$\mathbb{E}_x \varepsilon_i(\boldsymbol{x}) = 0$

- Strong models provide low bias $\Rightarrow$ train *overfitted* models $\Rightarrow$ **decision trees**

$\mathbb{E}_x[\varepsilon_i(\boldsymbol{x})\varepsilon_j(\boldsymbol{x})] = 0, \quad i \neq j$ $\leftarrow$ models need to be decorrelated

- Use different features *at each split* — **random subspace method** (RSM)
- Use different datasets — **bootstrapping**

Random forest

how to achieve these assumptions?

$\mathbb{E}_x \varepsilon_i(\boldsymbol{x}) = 0$

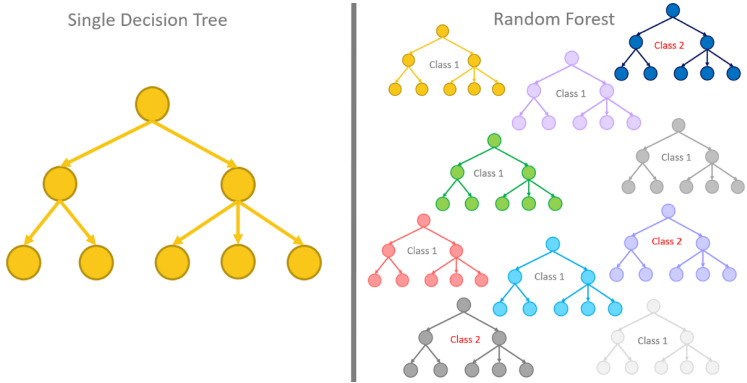
- Strong models provide low bias $\Rightarrow$ train *overfitted* models $\Rightarrow$ **decision trees**

$\mathbb{E}_x[\varepsilon_i(\boldsymbol{x})\varepsilon_j(\boldsymbol{x})] = 0, \quad i \neq j$ $\leftarrow$ models need to be decorrelated

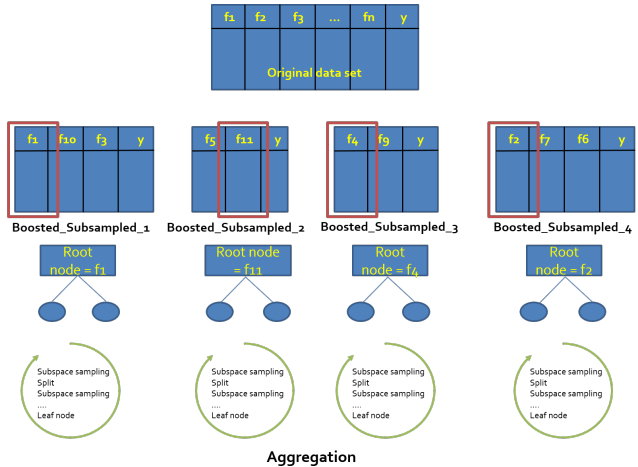
- Use different features *at each split* — **random subspace method** (RSM)
- Use different datasets — **bootstrapping**

Random forest = low-biased decision trees + RSM + $\underbrace{\textbf{b}\text{ootstrap} + \textbf{agg}\text{regat}\textbf{ing}}_{\textbf{bagging}}$

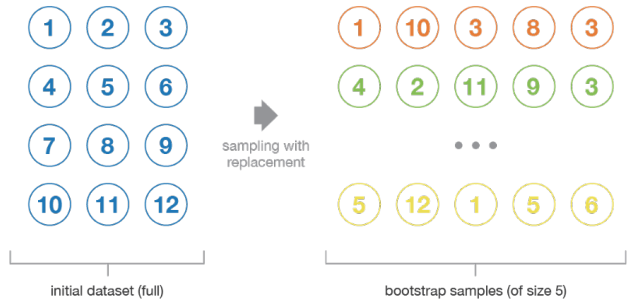
Random forest — trees



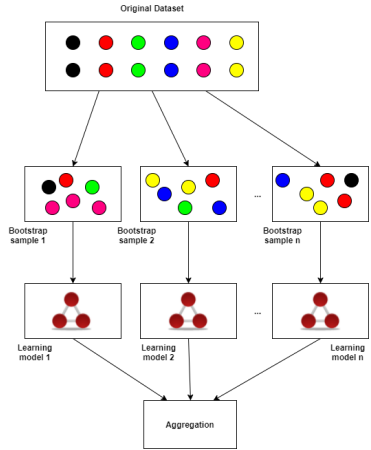
Random forest — RSM



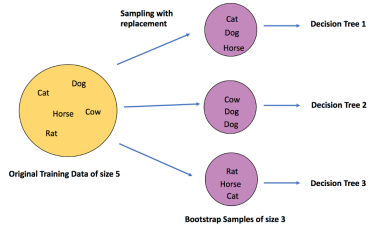
Random forest — bootstrap



Random forest — aggregating



Random forest — insights



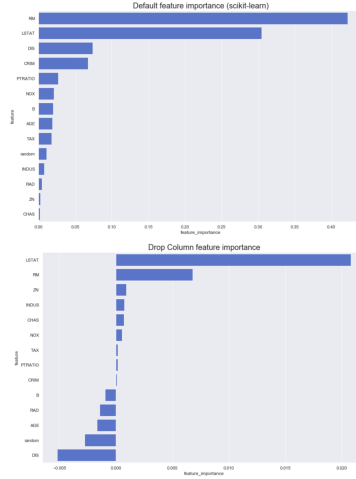
- As we use bagging, some data ($\approx 37\%$) is left unused during training
- Therefore, we can validate model on it $\Rightarrow$ **out-of-bag score**

$$\text{OOB} = \sum_{i=1}^{\ell} L\left(y_i, \frac{1}{\sum_{n=1}^{N}[x_i \notin X_n]} \sum_{n=1}^{N} [x_i \notin X_n]\, b_n(x_i)\right)$$

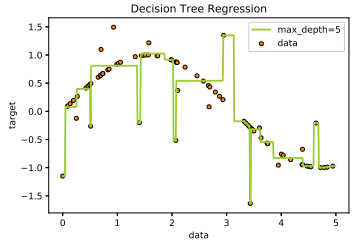
Random forest — insights

- IGs at each split can be used to calculate **feature importance**
- The higher (normalized) total reduction of IG brought by that feature – the better
- Note: this logic can be used in any tree-based model
- Caveat: inflates the importance of features features with high cardinality (many unique values)
- Other methods such as permutation importance exist
- And you might want to look at SHAP values too

more here



Random forest — check your understanding



- Can Random Forest be applied to regression task?
- What's the motivation for using Random Forest: decrease bias or variance?
- Given 1000 observations, minimum observation required to split a node: 200, minimum leaf size: 300 – what's the possible maximum depth of a Decision Tree?
- How would you propose to make the model on the figure better? Does it correspond to RF or Decision Tree?

- Yes
- Decrease variance
- 2
- Weaken the model by changing stopping criteria: low max depth, increase min elements in leaf, etc.

Gradient boosting

Gradient boosting

motivation

- Ensembling models improves performance comparing to a single model
- However, simply training them independently sounds not quite optimal (although is easy to parallelize)
- Maybe, we can add models to ensemble in a cleverer way?

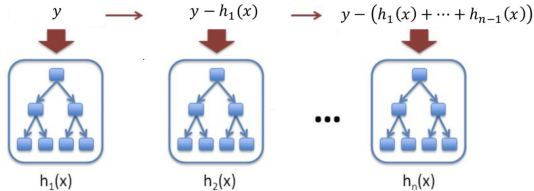
Gradient boosting

motivation

- Ensembling models improves performance comparing to a single model
- However, simply training them independently sounds not quite optimal (although is easy to parallelize)
- Maybe, we can add models to ensemble in a cleverer way?

➔ e.g., iteratively by correcting upon mistakes of previous models?

$$a_n(x) = h_1(x) + \cdots + h_n(x)$$



Can it happen that we won't be able to fit directly to residual errors?

Can it happen that we won't be able to fit directly to residual errors?
In case of classification, if we predict label "1" when true label is "0", on the next stage we get "-1" label

Can it happen that we won't be able to fit directly to residual errors?
In case of classification, if we predict label "1" when true label is "0", on the next stage we get "-1" label

Let's dive into theory

Gradient boosting — theory

- Suppose we have dataset $\{(\boldsymbol{x}_i, y_i)\,,\ i = 1, \ldots, N\}$, model $a(\boldsymbol{x})$ to be found and loss function $L(y, a(\boldsymbol{x}))$
- Optimal model:
$$\hat{a}(\boldsymbol{x}) = \arg\min_{a(\boldsymbol{x})} \mathbb{E}_{x,y}[L(y, a(\boldsymbol{x}))]$$
- Let it be from parametric family:
$$\hat{a}(\boldsymbol{x}) = a(\boldsymbol{x}, \hat{\boldsymbol{\theta}})$$
$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{x,y}[L(y, a(\boldsymbol{x}, \boldsymbol{\theta}))]$$
- Here $L(y, z)$ encodes information about the problem type:
  - classification: $\log\left(1 + e^{-yz}\right)$
  - regression: $(y - z)^2$ or $|y - z|$

Gradient boosting

theory

- Let's iteratively construct $\hat{a}(\boldsymbol{x})$ as a sum of base algorithms:

$$\hat{a}(\boldsymbol{x}) = \sum_i \hat{\rho}_i \cdot h(\boldsymbol{x}, \hat{\theta}_i)$$

Gradient boosting — theory

- Let's iteratively construct $\hat{a}(\boldsymbol{x})$ as a sum of base algorithms:

$$\hat{a}(\boldsymbol{x}) = \sum_i \hat{\rho}_i \cdot h(\boldsymbol{x}, \hat{\theta}_i)$$

- Suppose we know how it looks like at $(t-1)$-th iteration

$$\hat{a}_{t-1}(\boldsymbol{x}) = \sum_{i=0}^{t-1} \hat{\rho}_i \cdot h(\boldsymbol{x}, \hat{\theta}_i)$$

Gradient boosting — theory

- Let's iteratively construct $\hat{a}(\boldsymbol{x})$ as a sum of base algorithms:

$$\hat{a}(\boldsymbol{x}) = \sum_i \hat{\rho}_i \cdot h(\boldsymbol{x}, \hat{\theta}_i)$$

- Suppose we know how it looks like at $(t-1)$-th iteration

$$\hat{a}_{t-1}(\boldsymbol{x}) = \sum_{i=0}^{t-1} \hat{\rho}_i \cdot h(\boldsymbol{x}, \hat{\theta}_i)$$

- At t-th iteration we want to improve the current model by adding a new base algorithm:

$$\hat{a}_t(\boldsymbol{x}) = \hat{a}_{t-1}(\boldsymbol{x}) + \hat{\rho}_t \cdot h(\boldsymbol{x}, \hat{\theta}_t)$$

$$(\hat{\rho}_t, \hat{\boldsymbol{\theta}}_t) = \arg\min_{\rho, \boldsymbol{\theta}} \mathbb{E}_{x,y}[L(y, \hat{a}_{t-1}(\boldsymbol{x}) + \rho \cdot h(\boldsymbol{x}, \boldsymbol{\theta}))],$$

Gradient boosting — theory

- Let's iteratively construct $\hat{a}(\boldsymbol{x})$ as a sum of base algorithms:

$$\hat{a}(\boldsymbol{x}) = \sum_i \hat{\rho}_i \cdot h(\boldsymbol{x}, \hat{\theta}_i)$$

- Suppose we know how it looks like at $(t-1)$-th iteration

$$\hat{a}_{t-1}(\boldsymbol{x}) = \sum_{i=0}^{t-1} \hat{\rho}_i \cdot h(\boldsymbol{x}, \hat{\theta}_i)$$

- At t-th iteration we want to improve the current model by adding a new base algorithm:

$$\hat{a}_t(\boldsymbol{x}) = \hat{a}_{t-1}(\boldsymbol{x}) + \hat{\rho}_t \cdot h(\boldsymbol{x}, \hat{\theta}_t)$$

$$(\hat{\rho}_t, \hat{\boldsymbol{\theta}}_t) = \arg\min_{\rho, \boldsymbol{\theta}} \mathbb{E}_{x,y}[L(y, \hat{a}_{t-1}(\boldsymbol{x}) + \rho \cdot h(\boldsymbol{x}, \boldsymbol{\theta}))],$$

- In fact, it looks pretty much like a gradient descent in a model space $\Rightarrow$

- So let's train each base learner to predict this gradient:

$$\boldsymbol{r}_{it} = -\left[\frac{\partial L\left(y_i,\, a\left(\boldsymbol{x}_i\right)\right)}{\partial a\left(\boldsymbol{x}_i\right)}\right]_{a(\boldsymbol{x}_i)=\hat{a}(\boldsymbol{x}_i)}, \quad \text{for } i = 1, \ldots, N$$

$$\hat{\theta}_t = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \left(\boldsymbol{r}_{it} - h\left(\boldsymbol{x}_i, \boldsymbol{\theta}\right)\right)^2,$$

Gradient boosting — theory

- So let's train each base learner to predict this gradient:

$$\boldsymbol{r}_{it} = -\left[\frac{\partial L\left(y_i,\, a\left(\boldsymbol{x}_i\right)\right)}{\partial a\left(\boldsymbol{x}_i\right)}\right]_{a(\boldsymbol{x}_i)=\hat{a}(\boldsymbol{x}_i)}, \quad \text{for } i = 1, \ldots, N \quad \leftarrow \text{consider } L = (y - a(\boldsymbol{x}))^2 \Rightarrow \Rightarrow \boldsymbol{r} \sim (y - a(\boldsymbol{x})) \Rightarrow \Rightarrow \text{our residual errors}$$

$$\hat{\theta}_t = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \left(\boldsymbol{r}_{it} - h\left(\boldsymbol{x}_i, \boldsymbol{\theta}\right)\right)^2,$$

- So let's train each base learner to predict this gradient:

$$\boldsymbol{r}_{it} = -\left[\frac{\partial L\left(y_i,\, a\left(\boldsymbol{x}_i\right)\right)}{\partial a\left(\boldsymbol{x}_i\right)}\right]_{a(\boldsymbol{x}_i)=\hat{a}(\boldsymbol{x}_i)}, \quad \text{for } i = 1, \ldots, N \quad \leftarrow \text{consider } L = (y - a(\boldsymbol{x}))^2 \Rightarrow$$
$$\Rightarrow \boldsymbol{r} \sim (y - a(\boldsymbol{x})) \Rightarrow$$
$$\Rightarrow \text{our residual errors}$$

$$\hat{\theta}_t = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \left(\boldsymbol{r}_{it} - h\left(\boldsymbol{x}_i, \boldsymbol{\theta}\right)\right)^2,$$

- And learning the optimal learning rate (but can be skipped and set to constant):

$$\hat{\rho}_t = \arg\min_{\rho} \sum_{i=1}^{N} L\left(y_i,\, \hat{a}_{t-1}\left(\boldsymbol{x}_i\right) + \rho \cdot h\left(\boldsymbol{x}_i, \hat{\theta}_t\right)\right)$$

Gradient boosting — theory

- So let's train each base learner to predict this gradient:

$$\boldsymbol{r}_{it} = -\left[\frac{\partial L\left(y_i,\, a\left(\boldsymbol{x}_i\right)\right)}{\partial a\left(\boldsymbol{x}_i\right)}\right]_{a(\boldsymbol{x}_i)=\hat{a}(\boldsymbol{x}_i)}, \quad \text{for } i = 1, \ldots, N \quad \begin{array}{l}\leftarrow \text{consider } L = (y - a(\boldsymbol{x}))^2 \Rightarrow \\ \Rightarrow \boldsymbol{r} \sim (y - a(\boldsymbol{x})) \Rightarrow \\ \Rightarrow \text{our residual errors}\end{array}$$

$$\hat{\theta}_t = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \left(\boldsymbol{r}_{it} - h\left(\boldsymbol{x}_i, \boldsymbol{\theta}\right)\right)^2,$$

- And learning the optimal learning rate (but can be skipped and set to constant):

$$\hat{\rho}_t = \arg\min_{\rho} \sum_{i=1}^{N} L\left(y_i,\, \hat{a}_{t-1}\left(\boldsymbol{x}_i\right) + \rho \cdot h\left(\boldsymbol{x}_i, \hat{\theta}_t\right)\right)$$

- Finally:

$$\hat{a}_t(\boldsymbol{x}) = \hat{a}_{t-1}(\boldsymbol{x}) + \hat{\rho}_t \cdot h(\boldsymbol{x}, \hat{\theta}_t)$$

Gradient boosting

theory

- In theory, you can boost whatever algorithm you want
- But people opted for trees $\rightarrow$ aka **Boosted Decision Trees** (BDT) in HEP vocabulary
- What type of models we should use: weak or strong?

Gradient boosting

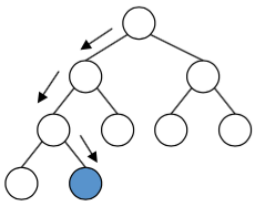
theory

- In theory, you can boost whatever algorithm you want
- But people opted for trees → aka **Boosted Decision Trees** (BDT) in HEP vocabulary
- What type of models we should use: weak or strong?
  - speed
  - accuracy
  - overfitting

Gradient boosting

theory

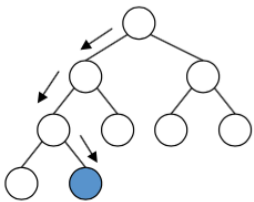
- In theory, you can boost whatever algorithm you want
- But people opted for trees → aka **Boosted Decision Trees** (BDT) in HEP vocabulary
- What type of models we should use: weak or strong?
  - speed
  - accuracy
  - overfitting

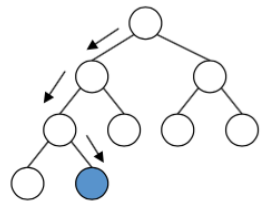


Gradient boosting

theory

- In theory, you can boost whatever algorithm you want
- But people opted for trees → aka **Boosted Decision Trees** (BDT) in HEP vocabulary
- What type of models we should use: weak or strong?
  - speed
  - accuracy
  - overfitting

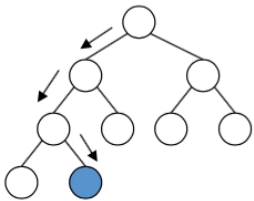


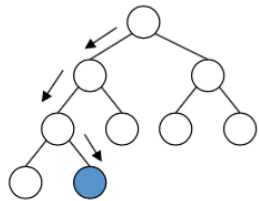


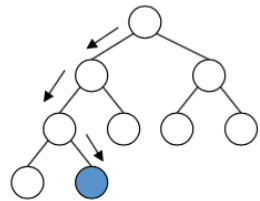
Gradient boosting

theory

- In theory, you can boost whatever algorithm you want
- But people opted for trees → aka **Boosted Decision Trees** (BDT) in HEP vocabulary
- What type of models we should use: weak or strong?
  - speed
  - accuracy
  - overfitting







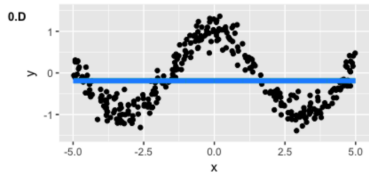
- Data: toy dataset $y = \cos(x) + \epsilon, \epsilon \sim \mathcal{N}\left(0, \frac{1}{5}\right), x \in [-5, 5]$
- Loss function: MSE
- Family of algorithms: decision trees, depth $= 2$
- Number of iterations: M $= 3$
- Initialial base algorithm: constant with mean value

Gradient boosting

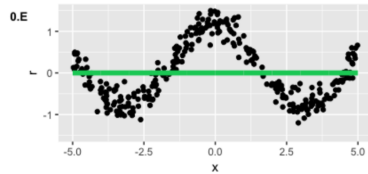
examples

playgrounds click, click

Dataset and Current ensemble



Residuals and Last model in ensemble

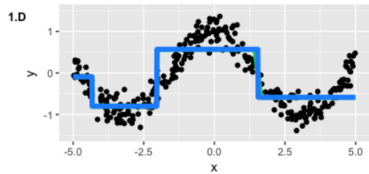


Gradient boosting

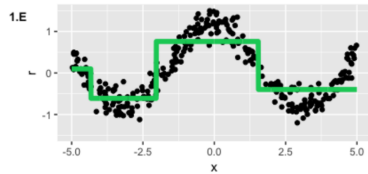
examples

playgrounds click, click

Dataset and Current ensemble



Residuals and Last model in ensemble

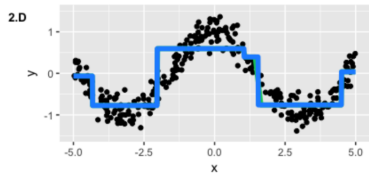


Gradient boosting

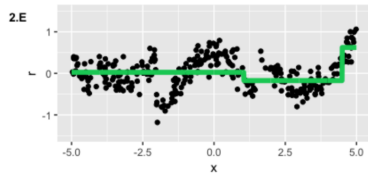
examples

playgrounds click, click

Dataset and Current ensemble



Residuals and Last model in ensemble

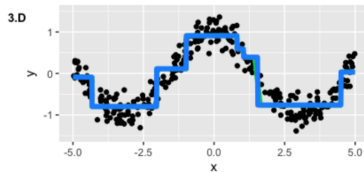


Gradient boosting

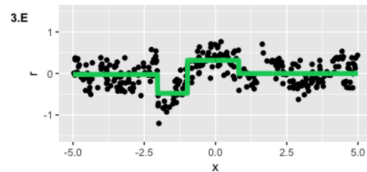
examples

playgrounds click, click

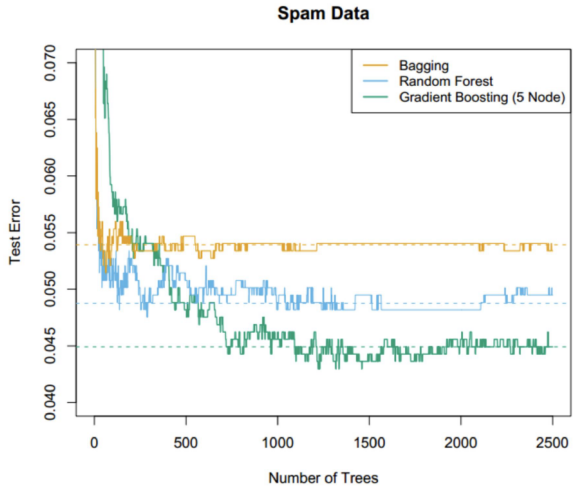
Dataset and Current ensemble



Residuals and Last model in ensemble



Gradient boosting — examples



Gradient boosting

check your understanding

- Are base models in boosted desicion trees independent of each other?
- Does boosting method improve performance because of aggregation of weak learners predictions?
- Can we use Random forests as base models in Gradient Boosting?
- What will happen if we remove first tree from Boosting?
- What will happen if we remove first tree from Random Forest?

Gradient boosting

check your understanding

- No
- Yes
- You shouldn't do that
- Model will break
- Almost nothing

Summary

- **Decision tree**
  - binary decisions to model non-linearities
  - grow greedily leaf by leaf
  - split leaves to improve Information Gain
  - until stopping criteria is reached
  - predict with constant in a leaf after propagating through
- **Random forest**
  - combine trees into ensemble for smaller error
  - each tree is independent of others
  - low-biased trees + RSM + bagging
- **Gradient boosting**
  - combine algorithms into ensemble for even smaller error
  - each next algorithm improves predictions of previous ones