Tree models

Vladimir Bocharnikov [2] Andrey Filatov [1] Oleg Filatov [2]
Stepan Zakharov [3]

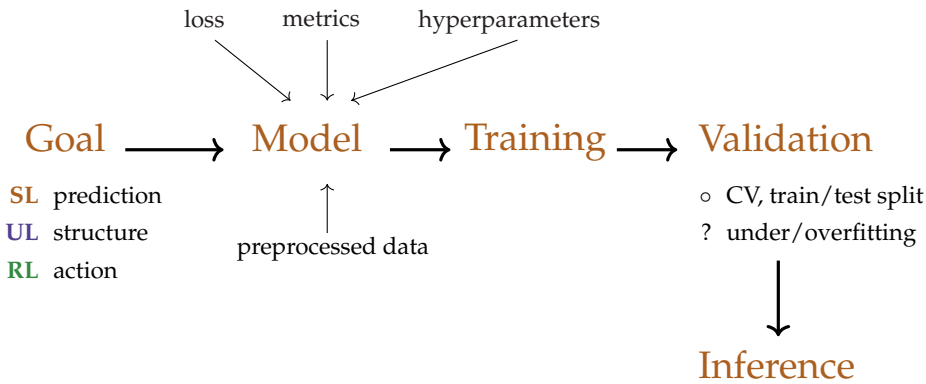
[1]MIPT [2]DESY [3]BINP

Outline

- Decision tree
  - From linear models to decision trees
  - Information entropy
  - Making a good split
  - Some insights
- Random forest
  - Random subspace method
  - Bootstrapping
  - Aggregating
- Gradient boosting
  - Some motivation
  - Gradient descent in a model space
  - BDT example

Recap

ML Problems — General pipeline



Linear models

Classification (Log. Regression)

- $Y = \{0, 1\}$
- N objects with K numeric features: $D = \mathbb{R}^K$
- $a(\boldsymbol{x}, \boldsymbol{\theta}) = \sum_{j=1}^{K} f_j(\boldsymbol{x}) \cdot \boldsymbol{\theta_j} = \langle \boldsymbol{x}, \boldsymbol{\theta} \rangle \to \sigma\big[\langle x, \theta \rangle\big]$
- $\mathcal{L}\big(a(\boldsymbol{x_i}), y_i\big) = \big[a(\boldsymbol{x}) \neq y_i\big] \to$ log. likelihood
- $Q(a, X, Y) = \sum_{i=1}^{N} \mathcal{L}\big(a(\boldsymbol{x_i}), y_i\big)$

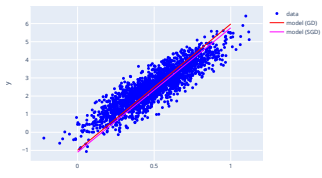
Regression

- $Y = \mathbb{R}$
- N objects with K numeric features: $D = \mathbb{R}^K$
- $a(\boldsymbol{x}, \boldsymbol{\theta}) = \langle \boldsymbol{x}, \boldsymbol{\theta} \rangle$
- $\mathcal{L}\big(a(\boldsymbol{x_i}), y_i\big) = \big(a(\boldsymbol{x}) - y_i\big)^2$
- $Q(a, X, Y) = \sum_{i=1}^{N} \mathcal{L}\big(a(\boldsymbol{x_i}), y_i\big)$

$$\hat{a} = \arg\min_{a \in A} Q(a, X, Y) \;\Leftarrow \textbf{Training}$$

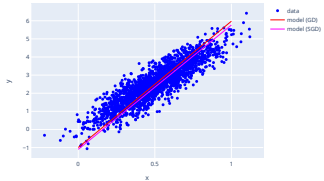
Linear models

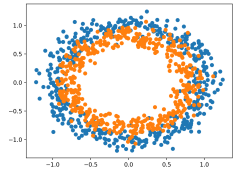
Pros & cons



- Linear models:
  - ➔ nicely describe linear correlations
  - ➔ fast and easy to train
  - ➔ performing well with large number of samples/features

Linear models — Pros & cons

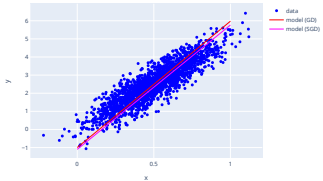


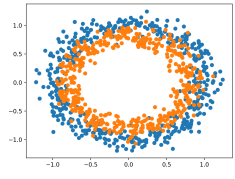


- Linear models:
  - ➔ nicely describe linear correlations
  - ➔ fast and easy to train
  - ➔ performing well with large number of samples/features
- However, they are poor in modelling non-linearities
- Additional heuristics might be applied (e.g. polynomial features) but are ad-hoc

Linear models

Pros & cons



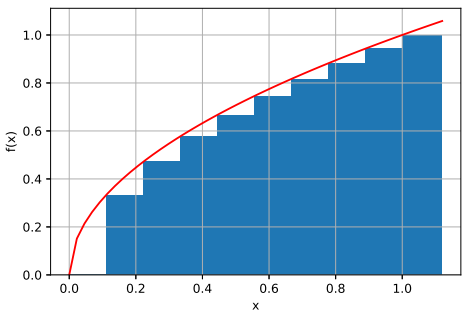


- Linear models:
  - ➔ nicely describe linear correlations
  - ➔ fast and easy to train
  - ➔ performing well with large number of samples/features
- However, they are poor in modelling non-linearities
- Additional heuristics might be applied (e.g. polynomial features) but are ad-hoc

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

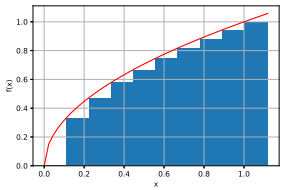
What is the simplest way to approximate a function?

Use **piecewise linear function**

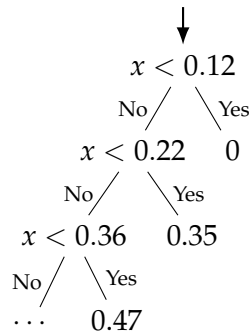


$$f(x) = \begin{cases} 0, & x < 0.12 \\ 0.35, & 0.12 \leq x < 0.22 \\ 0.47, & 0.22 \leq x < 0.36 \\ \dots \end{cases}$$

We can define this function by introducing a set of **yes/no questions**



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

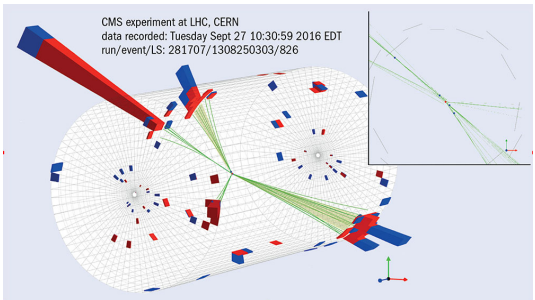


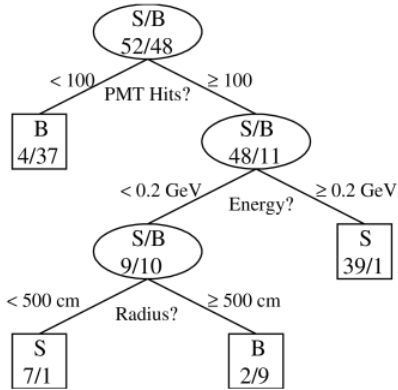
Well this looks not a tree but **decision list**

Desision tree

Intuition

This approach also works for classification problems



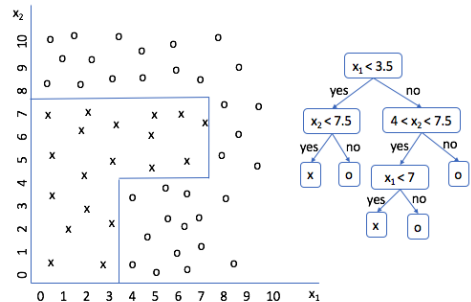


Desision tree — Algorithm

Algorithm 1: Decision tree

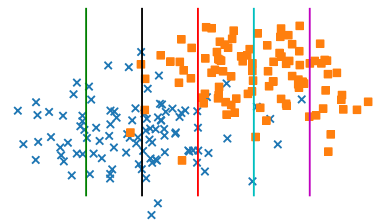
1: Initialize: hyperparameters, leaf set = $\{X_{\text{train}}\}$

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**



Desision tree — Split criteria

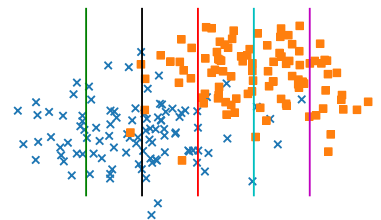
- Suppose there is a classification problem
- So we want to separate one class from the other by constructing **decision boundary**
- And using decision tree algorithm described earlier
- To do that we need to start splitting on features
- Which **split** is the best?



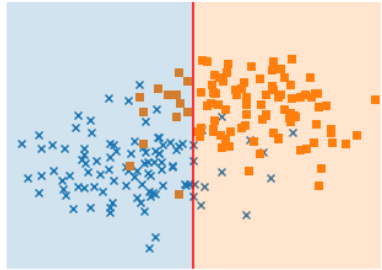
Desision tree — Split criteria

- 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

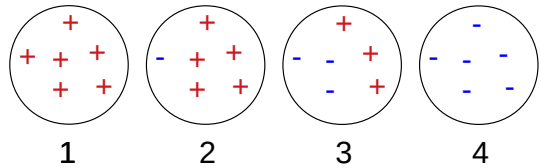


Desision tree — Split criteria

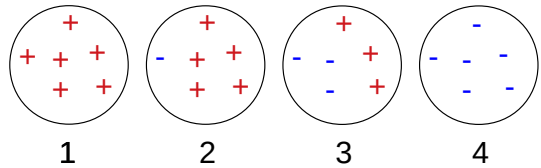


The ”best” split is a central one because it introduces **purity** in the best way. Let’s try to bring up some math to this topic.

- Consider four datasets:
  1&4 are pure
  2 is almost pure
  3 is the most impure

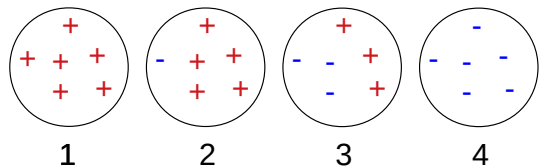


- Consider four datasets:
  1&4 are pure
  2 is almost pure
  3 is the most impure
- How to numerically describe impurity?



- Consider four datasets:
  1&4 are pure
  2 is almost pure
  3 is the most impure
- How to numerically describe impurity?
- Introduce **information entropy:**

$$H(X) = -\sum_i p_i \log_2 p_i, \quad p_i = \frac{N_i}{X}$$



- $i = [+, -]$

1: $H(X) = -6/6 \log_2(6/6) = 0$

2: $H(X) = -5/6 \log_2(5/6) - 1/6 \log_2(5/6) \approx 0.65$

3: $H(X) = -3/6 \log_2(3/6) - 3/6 \log_2(3/6) = 1$

4: $H(X) = -6/6 \log_2(6/6) = 0$

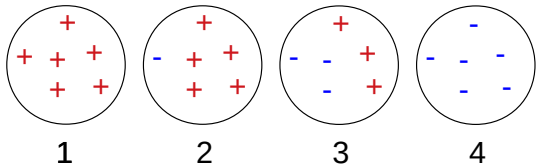
Desision tree

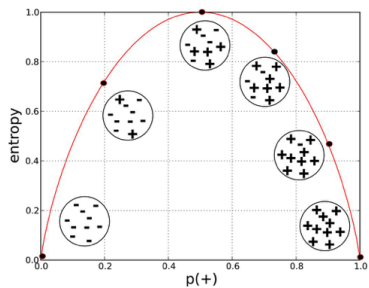
Concept of information entropy

- Consider four datasets:
  1&4 are pure
  2 is almost pure
  3 is the most impure
- How to numerically describe impurity?
- Introduce **information entropy:**

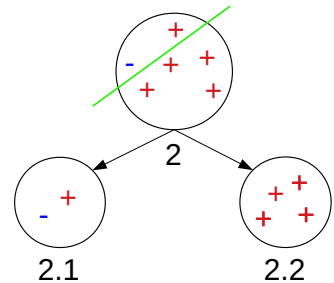
$$H(X) = -\sum_i p_i \log_2 p_i, \quad p_i = \frac{N_i}{X}$$

- impurity $\propto H(X)$



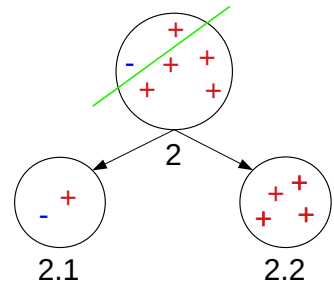


- impurity $\propto H(X)$
- What happens with H when we make a split?



- impurity $\propto H(X)$
- What happens with H when we make a split?

2: $H(X) \approx 0.65$
2.1: $H(L) = -2 \cdot 1/2 \log_2(1/2) = 1$
2.2: $H(R) = 0$



- impurity $\propto H(X)$
- What happens with H when we make a split?

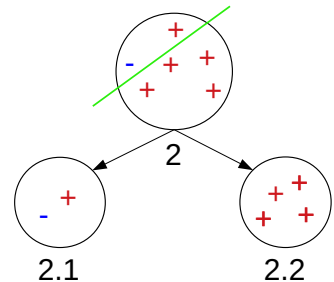
  2: $H(X) \approx 0.65$

  2.1: $H(L) = -2 \cdot 1/2 \log_2(1/2) = 1$

  2.2: $H(R) = 0$

  Let's weight the results to compare with the initial case:

$$H(X|a) = \frac{L}{X}H(L) + \frac{R}{X}H(R) = \frac{2}{6} \cdot 1 + \frac{4}{6} \cdot 0 \approx 0.33$$



- impurity $\propto H(X)$
- What happens with H when we make a split?

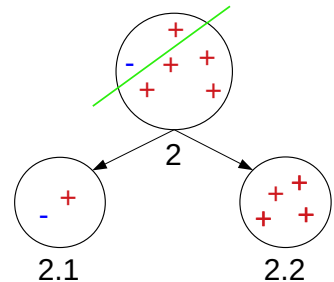
  2: $H(X) \approx 0.65$

  2.1: $H(L) = -2 \cdot 1/2 \log_2(1/2) = 1$

  2.2: $H(R) = 0$

  Let's weight the results to compare with the initial case:

$$H(X|a) = \frac{L}{X}H(L) + \frac{R}{X}H(R) = \frac{2}{6} \cdot 1 + \frac{4}{6} \cdot 0 \approx 0.33$$



$$H(X|a) < H(X) \Rightarrow \text{profit!}$$

more about IG

- To measure the profit by a classification algorithm a we define **Information Gain** (IG):

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

- In case of binary split:

$$\mathrm{IG}(\mathrm{X}, a) = \mathrm{H}(\mathrm{X}) - \left[\frac{L}{X} H(L) + \frac{R}{X} H(R)\right]$$

- The best classifier gives maximal IG:

$$\hat{a} = \arg\min_{a \in A} \mathrm{IG}(\mathrm{X}, \mathrm{a})$$

Two questions:

1. How to find optimal split criteria?
2. How to grow a tree?

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

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

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

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

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

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

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

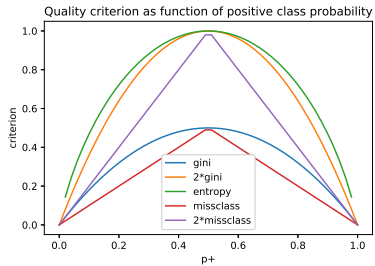
- or c_k with $\sum_k c_k = 1$, if tree predicts class probabilities
- If p_k is a fraction of class k events in a leaf

Desision tree — Split criteria

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

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

- or c_k with $\sum_k c_k = 1$, if tree predicts class probabilities
- If p_k is a fraction of class k events in a leaf
- 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**
    $\mathcal{L}(y, c) = -\sum_k [y = k] \log c_k \Rightarrow \mathrm{H(R)} = -\sum_k p_k \log_2 p_k$

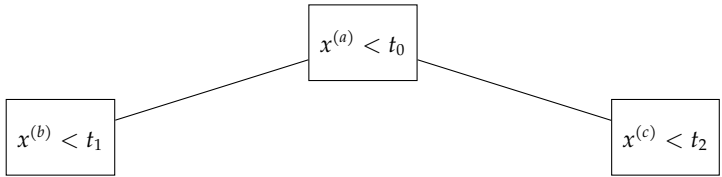


Lower diversity (higher purity)
$\Rightarrow$ **lower impurity** criterion H(R)

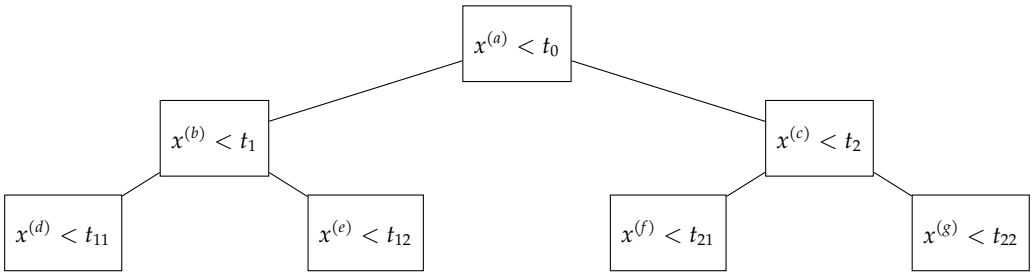
What about tree construction?

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

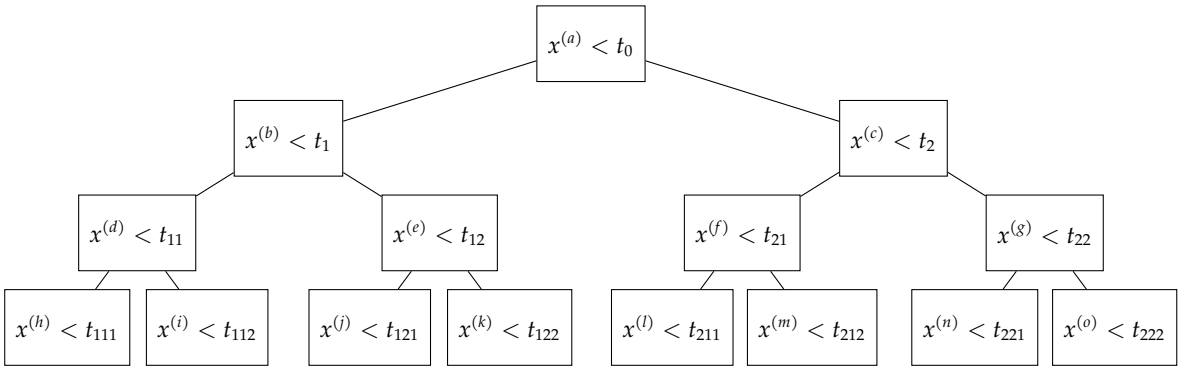
What about tree construction? Do it recursively by **greedy** algorithm



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

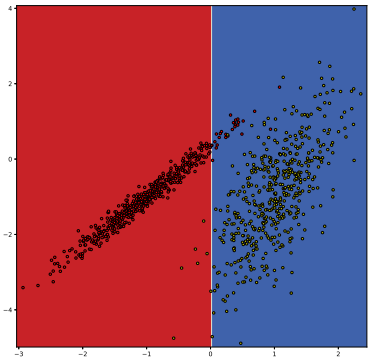


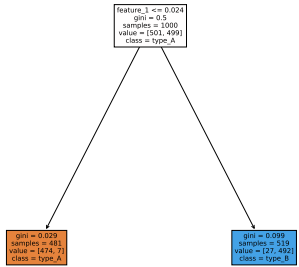
What about tree construction? Do it recursively by **greedy** algorithm



Desision tree — Growing

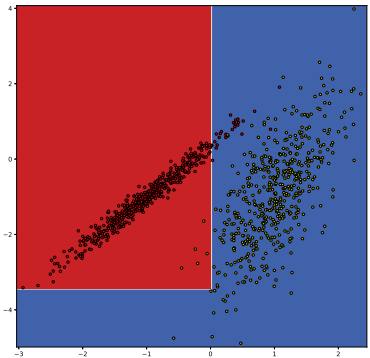
check out also these visuals

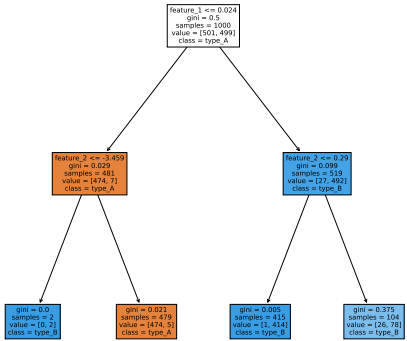




Desision tree — Growing

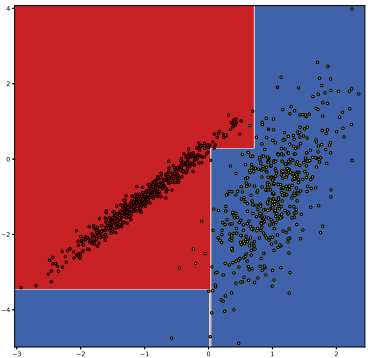
check out also these visuals

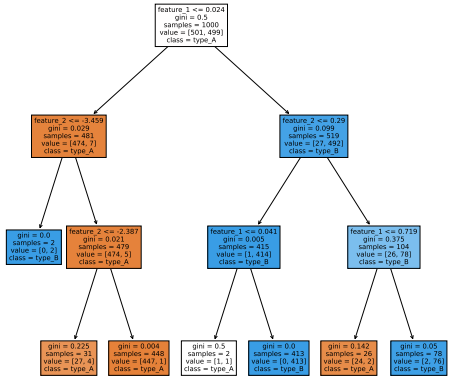




Desision tree — Growing

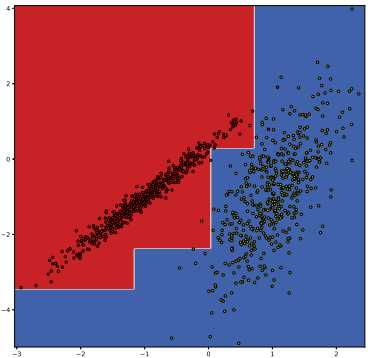
check out also these visuals

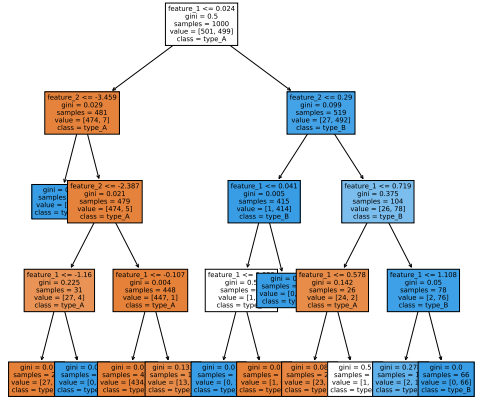




Desision tree — Growing

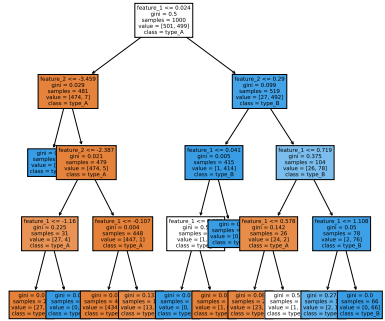
check out also these visuals





Desision tree — Stopping criteria

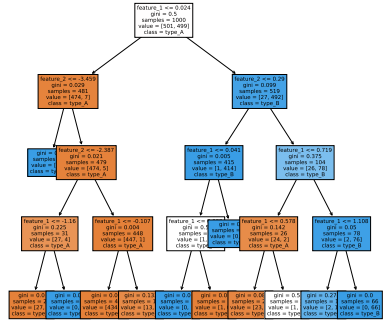
- What about stopping criteria?
  1. Split until one element in leaf
     (which problems can we meet?)
  2. Limit max depth/max number of leaves
  3. Limit amount of elements in leaf/split
  4. All elements in a leaf belong to a single class
  5. IG(j, t) isn't improving by x%
  6. Combination of limitations



Desision tree — Stopping criteria

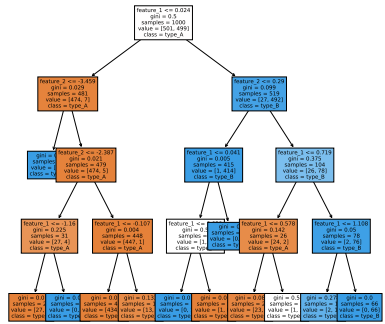
- What about stopping criteria?
  1. Split until one element in leaf
     (which problems can we meet?)
  2. Limit max depth/max number of leaves
  3. Limit amount of elements in leaf/split
  4. All elements in a leaf belong to a single class
  5. IG(j, t) isn't improving by x%
  6. Combination of limitations

➔ The choice is up to you
(aka hyperparameter)



Desision 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



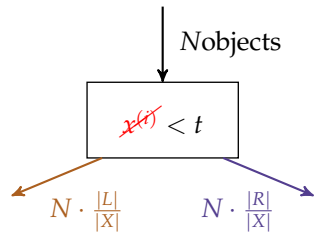
Can already trained decision tree handle **missing values** on inference step?

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

- While growing a tree, we know **split probabilities**:

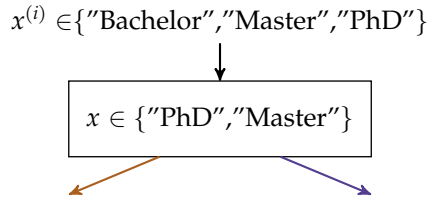
$$\hat{y} = \frac{|\mathrm{L}|}{|\mathrm{X}|}\hat{y}_{\mathrm{L}} + \frac{|\mathrm{R}|}{|\mathrm{X}|}\hat{y}_{\mathrm{R}}$$

- If at some node feature $x^{(i)}$ is missing, there still exists a way to use the tree
- Split all subset according to probabilities obtained during the training

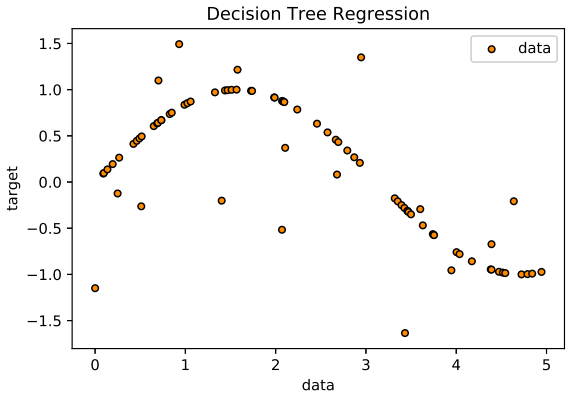


Categorical features can also easily be treated by tree:

- Loop through all combinations of its values
- Pick a set of values which result in the best IG
- Make a split using this criteria

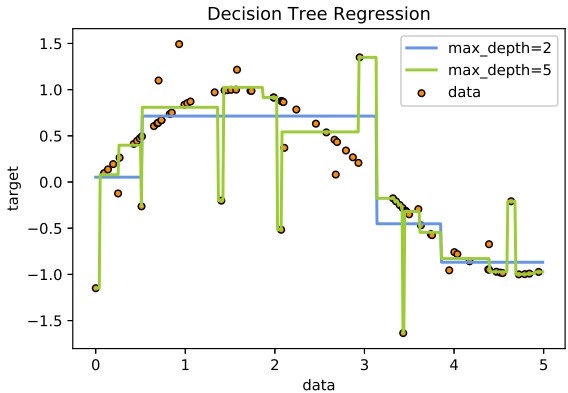


Desision tree — Regression



Can we use decision tree approach for regression problem?

Desision tree — Regression



Can we use decision tree approach for regression problem? Yes

- 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: $\mathcal{L}(y,c) = (y-c)^2 \Rightarrow \mathrm{H}(\mathrm{X}) = \min_c \frac{1}{|\mathrm{X}|}\sum_i (y_i - c)^2$

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

- How should we modify decision criteria?

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

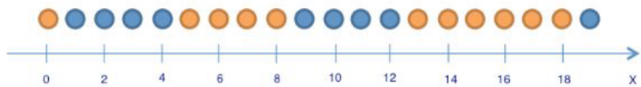
→ Mean squared error: $\mathcal{L}(y,c) = (y-c)^2 \Rightarrow \mathrm{H}(\mathrm{X}) = \min_c \frac{1}{|\mathrm{X}|}\sum_i (y_i - c)^2$

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

- One can further show that:

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

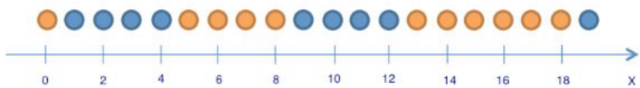
$$c^* = \mathrm{median}(y_i \in \mathrm{R})$$



- 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?
- What algorithm constructs a tree?
- Which stopping criteria can we use?

Desision 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 K 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 K 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 K 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 K 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:

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

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 K 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 $b_i(\boldsymbol{x})$:

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

- Assume errors to be unbiased and uncorrelated:

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

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

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

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

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?

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

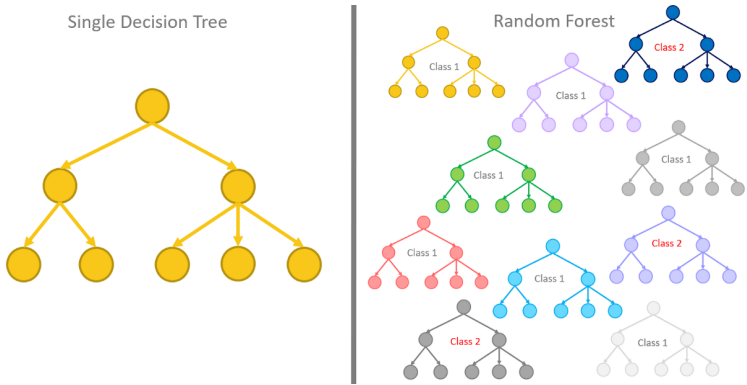
- ➔ Strong models provide low bias ⇒ train *overfitted* models ⇒ **decision trees**

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

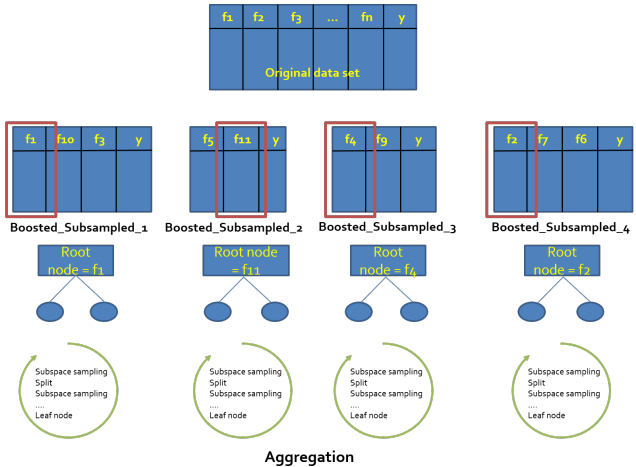
- ➔ Use different features *at each split* — **random subspace method** (RSM)
- ➔ Use different datasets — **bootstrapping**

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

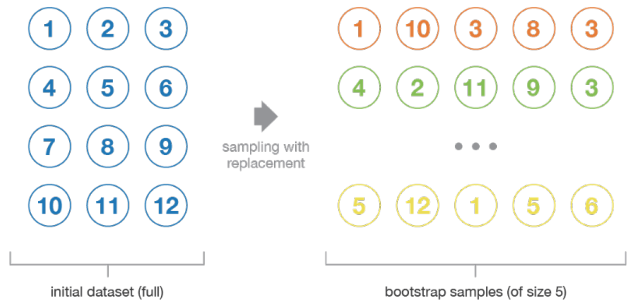
Random forest — Trees



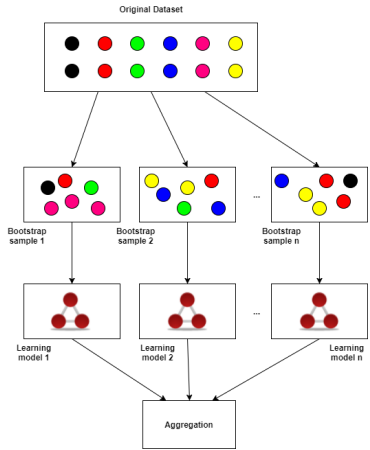
Random forest — RSM

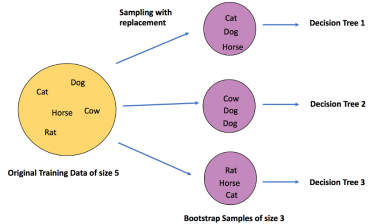


Random forest — Bootstrap



Random forest — Aggregating





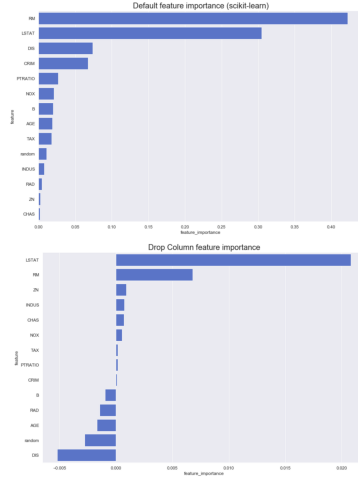
- As we use bagging, some data ($\approx 37\%$) is left unused during training
- Therefore, we can validate model on it $\Rightarrow$ **out-of-bag score**

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

Random forest — Insights

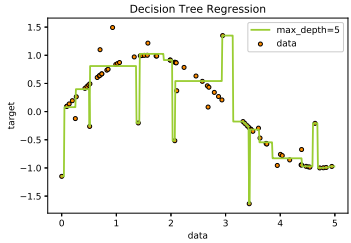
- IGs at each split can be used to calculate **feature importance**
- The higher (normalized) total reduction of IG brought by that feature – the better
- Note: this logic can be used in any tree-based model
- Caveat: might inflate the importance of numerical features and categorical features with high cardinality (many unique values)

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?

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

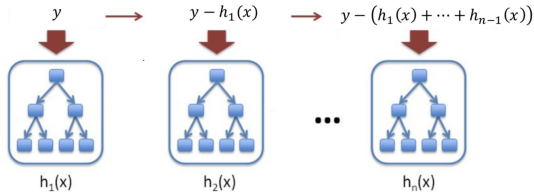
Gradient boosting

- 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?

- 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

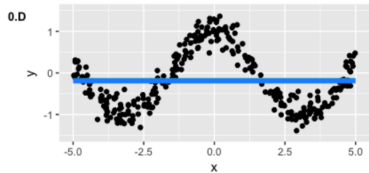
Can it happen that we won't be able to fit directly to residual errors?
In case of classification, if we predict label "1" when true label is "0", on the next stage we get "-1" label

Let's look at the example and dive into theory

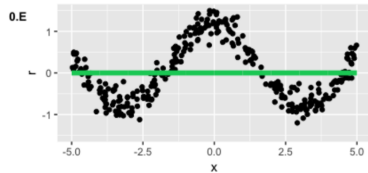
- Data: toy dataset $y = \cos(x) + \epsilon, \epsilon \sim \mathcal{N}\left(0, \frac{1}{5}\right), x \in [-5, 5]$
- Loss function: MSE
- Family of algorithms: decision trees, depth $= 2$
- Number of iterations: M $= 3$
- Initialial base algorithm: constant with mean value

playgrounds click, click

Dataset and Current ensemble

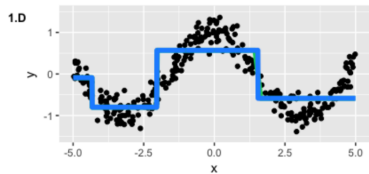


Residuals and Last model in ensemble

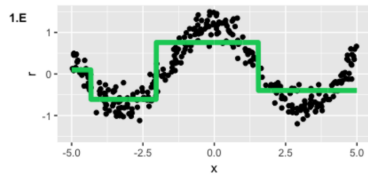


playgrounds click, click

Dataset and Current ensemble

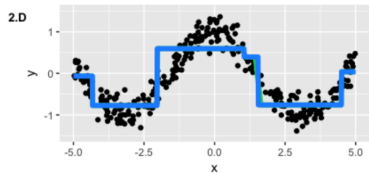


Residuals and Last model in ensemble

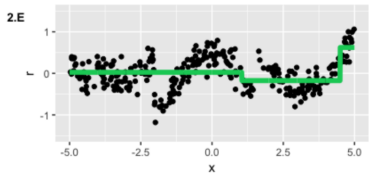


playgrounds click, click

Dataset and Current ensemble

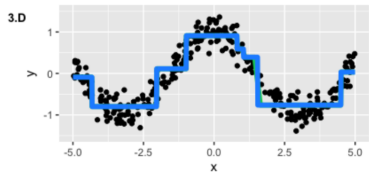


Residuals and Last model in ensemble

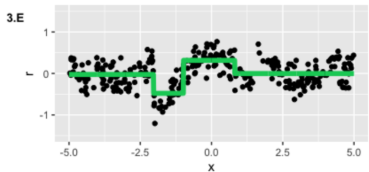


playgrounds click, click

Dataset and Current ensemble



Residuals and Last model in ensemble



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 $L(y, a(\boldsymbol{x}))$ encodes information about the problem type:
  - classification: $\log\left(1 + e^{-y \cdot a(\boldsymbol{x})}\right)$
  - regression: $\left(y - a(\boldsymbol{x})\right)^2$ or $\left|y - a(\boldsymbol{x})\right|$

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:

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

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

- 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):

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

- 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):

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

- 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

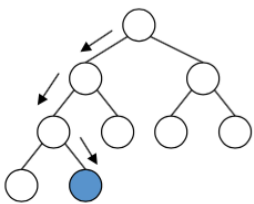
Theory

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

Gradient boosting

Theory

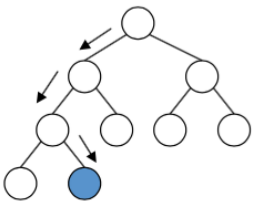
- In theory, you can boost whatever algorithm you want
- But people opted for trees $\rightarrow$ aka **Boosted Decision Trees** (BDT) in HEP vocabulary
- What type of models we should use: weak or strong?
  - speed
  - accuracy

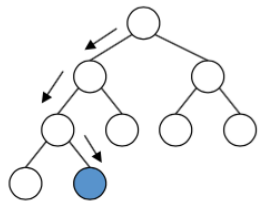


Gradient boosting

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?
  - speed
  - accuracy

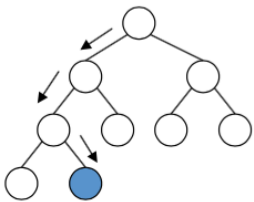


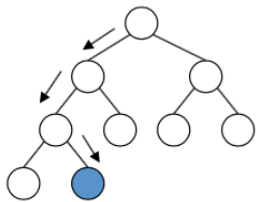


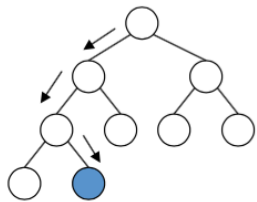
Gradient boosting

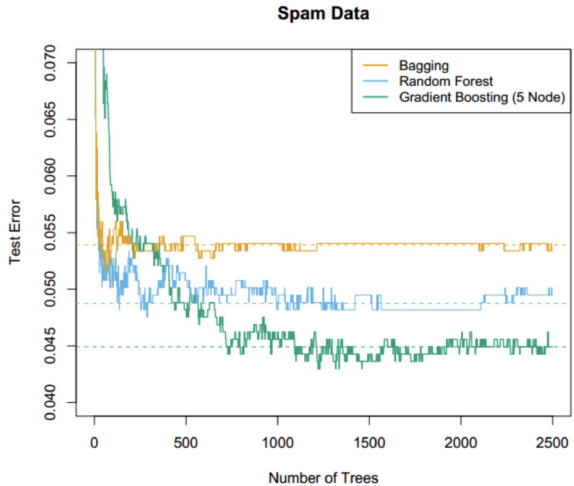
Theory

- 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?
  - speed
  - accuracy








Spam Data
Bagging
Random Forest
Gradient Boosting (5 Node)
Test Error
0.040
0.045
0.050
0.055
0.060
0.065
0.070
0
500
1000
1500
2000
2500
Number of Trees

- 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?

- 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