Decision Trees and Random Forest

Hachem Madmoun

December 16, 2024

Outline

Supervised Learning: a quick review

Setting up the objective

- **Data:** A dataset of random pairs $\{(X_i, Y_i)_{1 \leq i \leq n}\}$ over $\mathcal{X} \times \mathcal{Y}$ (typically $\mathcal{X} = \mathbb{R}^D$ and $\mathcal{Y} = \{0, 1\}$).
- The pairs $\{(X_i, Y_i)_{1 \leq i \leq n}\}$ are i.i.d. and follow an unknown distribution $\mathbb{P}$.
- A supervised algorithm is an algorithm that aims to build a predictor (i.e, a function $\Phi : \mathcal{X} \longrightarrow \mathcal{Y}$) which minimizes some risk.
- To define the risk, we need first to define a loss function (i.e, a function $l : \mathcal{Y} \times \mathcal{Y} \longrightarrow \mathbb{R}$ that measures the "distance" between the the output of the predictor and the true labels $(Y_i)_{1 \leq i \leq n}$).

- For the loss function, we typically choose for all $(y, y') \in \mathcal{Y} \times \mathcal{Y}$:

  $$l_{\text{Regression}}(y, y') = ||y - y'||_2 \quad \text{and} \quad l_{\text{Classification}}(y, y') = \delta_{y \neq y'}$$

- We then define the following risk associated to Φ

  $$\mathcal{R}_{\mathbb{P}}(\Phi) = \mathbb{E}^{\mathbb{P}}[l(Y, \Phi(X)]$$

- Our objective is then:

  $$\text{Finding} \quad \phi^*_{\mathbb{P}} = \underset{\Phi \in \mathcal{F}(\mathcal{X}, \mathcal{Y})}{\arg\min}\ \mathbb{E}^{\mathbb{P}}[l(Y, \Phi(X)] = \underset{\Phi \in \mathcal{F}(\mathcal{X}, \mathcal{Y})}{\arg\min}\ \mathcal{R}_{\mathbb{P}}(\Phi)$$

- Since $\mathbb{P}$ is unknown, we optimize the empirical risk $\mathcal{R}_n(\Phi)$:

  $$\mathcal{R}_n(\Phi) = \frac{1}{n} \sum_{i=1}^{n} l(Y_i, \Phi(X_i))$$

Bias-variance tradeoff

- In practice, we choose $\mathcal{G}$ a subset of $\mathcal{F}(\mathcal{X}, \mathcal{Y})$ and we minimise the empirical risk over $\mathcal{G}$.
- We define the $\Phi^*_{\mathbb{P},\mathcal{G}}$ the theoretical optimal predictor over the subset $\mathcal{G}$:

$$\Phi^*_{\mathbb{P},\mathcal{G}} = \arg\min_{\Phi \in \mathcal{G}} \mathbb{E}^{\mathbb{P}}[l(Y, \Phi(X)] = \arg\min_{\Phi \in \mathcal{G}} \mathcal{R}_{\mathbb{P}}(\Phi)$$

- We also define the $\hat{\Phi}^n_{\mathcal{G}}$ the empirical optimal predictor over the subset $\mathcal{G}$:

$$\hat{\Phi}^n_{\mathcal{G}} = \arg\min_{\Phi \in \mathcal{G}} \frac{1}{n} \sum_{i=1}^{n} l(Y_i, \Phi(X_i)) = \arg\min_{\Phi \in \mathcal{G}} \mathcal{R}_n(\Phi)$$

- The excess risk of $\hat{\Phi}_{\mathcal{G}}^n$ can be decomposed as follows

$$\mathcal{R}_{\mathbb{P}}(\hat{\Phi}_{\mathcal{G}}^n) - \mathcal{R}_{\mathbb{P}}(\phi_{\mathbb{P}}^*) = \underbrace{\mathcal{R}_{\mathbb{P}}(\hat{\Phi}_{\mathcal{G}}^n) - \mathcal{R}_{\mathbb{P}}(\Phi_{\mathbb{P},\mathcal{G}}^*)}_{\text{Variance}} + \underbrace{\mathcal{R}_{\mathbb{P}}(\Phi_{\mathbb{P},\mathcal{G}}^*) - \mathcal{R}_{\mathbb{P}}(\phi_{\mathbb{P}}^*)}_{\text{bias}}$$

- Bias-variance tradeoff:
  - If the "complexity" of $\mathcal{G}$ is too big, it results in high variance : **overfitting**
  - If the "complexity" of $\mathcal{G}$ is too small, it results in high bias : **underfitting**
- In the literature, the complexity of $\mathcal{G}$ is measured by the Vapnik–Chervonenkis dimension ($V_{\mathcal{G}}$) or the Rademacher complexity measure.

- **Vapnik-Chervonenkis Theorem** If $\mathcal{Y} = \{0, 1\}$ and $l : \mathcal{Y} \times \mathcal{Y} \longrightarrow \{0, 1\}$ is the classification loss. Then, for all $\epsilon > 0$, with probability at least $1 - \epsilon$:

$$\mathcal{R}_{\mathbb{P}}(\hat{\Phi}^n_{\mathcal{G}}) - \mathcal{R}_{\mathbb{P}}(\Phi^*_{\mathbb{P},\mathcal{G}}) \leq 2\sqrt{\frac{2V_{\mathcal{G}} \ln[4(2n+1)\epsilon^{-1}]}{n}}$$

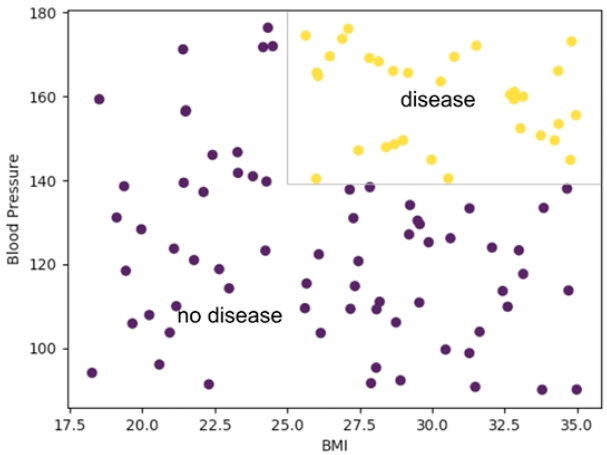
- The proof of the above stated theorem remains the same if $\mathcal{G}$ depends on the dataset.
- The next section will present the Decision Tree algorithm, which is one way of defining $\mathcal{G}$ based on the dataset.

Decision Tree Model

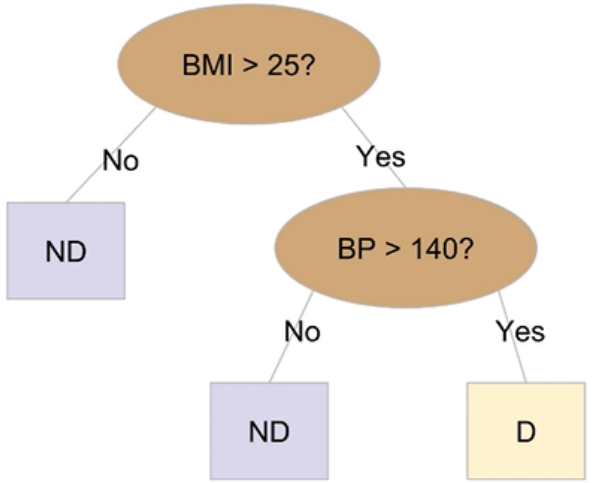
Introduction

Intuition

- The **Decision Tree** (DT) algorithm is an attractive model if we care about interpretability.
- As the name decision tree suggests, we can think of this model as breaking down our data by making a decision based on asking a series of questions.



(a) Features and Labels



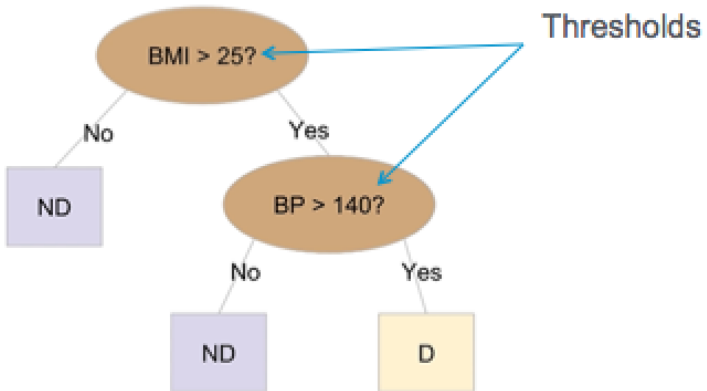
(b) Decision Tree graph

A high level description of the algorithm

- The DT algorithm is basically just a bunch of nested if-statements on the input features (also called **attributes**) in the training dataset.
- The decision algorithm:
  - We start at the tree root (with the whole dataset)
  - Then we split the dataset on the attribute that results in the largest **Information Gain** (IG).
  - We iterate the splitting procedure at each child node until the leaves are pure (which means that the samples at the leaves belong to the same class)
  - A very deep tree is prone to overfitting. To avoid that, we set a limit for the maximal depth of the tree.

Building Decision Trees

- First, we need to define an objective function that we want to optimize (Information Gain).
- Then, at each iteration, two challenges arise when trying to choose the best split.
  - How do we choose the best attribute responsible for the split ?
  - How do we choose the threshold when splitting based on the "best attribute" ?



Information Theory

Entropy - Definition -

- Let Y be a random variable taking values in the finite set $\mathcal{Y}$.
- Let's denote $p(y) = \mathbb{P}(Y = y)$
- In informarion theory, the quantity

$$I(y) = \log_2(\frac{1}{p(y)})$$

  can be interpreted as a quantity of information carried by the occurence of y (sometimes called *self-information*)
- So, the **entropy** is defined as the expected amount of information of the random variable Y:

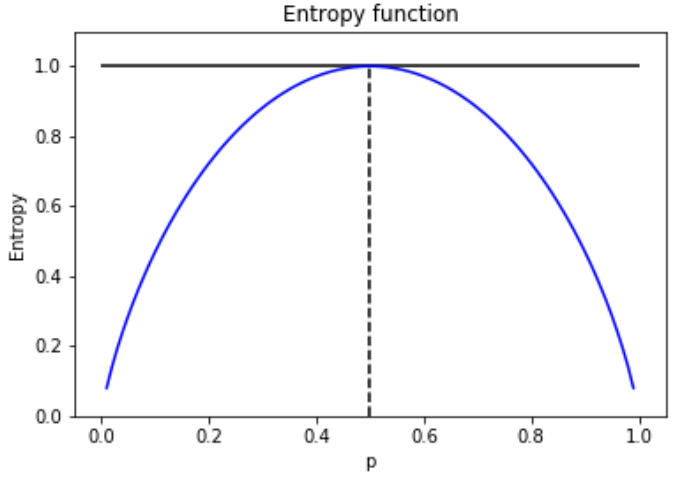
$$H(Y) = \mathbb{E}_{p(y)}[I(Y)] = -\sum_{y \in \mathcal{Y}} p(y) \log_2(p(y))$$

Entropy - Example -

- Let's take an example of a Bernoulli distribution $Y \sim \mathcal{B}(p)$
- The entropy of Y as a function of p is :

$$H(p) = -p \log_2(p) + (1-p) \log_2(1-p)$$

- $p = 0.5$ yields maximum entropy.
- So, the entropy is a measure of how much information we get from finding out the value of the random variable.



- We have the following inequalities:

$$H(Y) \geq 0 \quad \text{with equality if Y is constant a.s}$$
$$H(Y) \leq \log_2(\text{Card}(\mathcal{Y}))$$

- **Proof of the first point:**
  - Since $\forall y \in \mathcal{Y} \quad p(y) \leq 1$ then:

$$H(Y) = \sum_{y \in \mathcal{Y}} \underbrace{-p(y)\log_2(p(y))}_{\geq 0} \geq 0$$

  - With equality iff $\forall y \in \mathcal{Y} \quad p(y)\log_2(y) = 0$, which proves the first point.

Kullback-Leibler divergence - Definition -

- To prove the second point, we need to introduce the Kullback-Leibler divergence.
- Let p and q be two finite distributions on $\mathcal{Y}$.
- The **Kullback-Leibler divergence Kullback-Leibler divergence** between p and q is defined as follows:

$$
\begin{aligned}
D_{\mathrm{KL}}\left(p||q\right) &= \mathbb{E}_{Y\sim p}\left[\log\frac{p(Y)}{q(Y)})\right] \\
&= \sum_{y\in\mathcal{Y}} p(y)\log\frac{p(y)}{q(y)} \\
&= \sum_{y\in\mathcal{Y}} \frac{p(y)}{q(y)}\left(\log\frac{p(y)}{q(y)}\right)q(y) \\
&= \mathbb{E}_{Y\sim q}\left[\frac{p(Y)}{q(Y)}\log\frac{p(Y)}{q(Y)})\right]
\end{aligned}
$$

- **Proposition:**

$$D_{\mathrm{KL}}\left(p||q\right) \geq 0 \quad \text{and equality holds iff} \quad p = q$$

- **Proof:**

$$\begin{aligned} D_{\mathrm{KL}}\left(p||q\right) &= \mathbb{E}_{Y\sim q}\left[\frac{p(Y)}{q(Y)}\log\frac{p(Y)}{q(Y)})\right] \quad \text{(definition)} \\ &\geq \mathbb{E}_{Y\sim q}\left[\frac{p(Y)}{q(Y)}\right]\log\mathbb{E}\left[\frac{p(Y)}{q(Y)})\right] \quad \text{(Jensen Inequality)} \\ &= 0 \quad \left(\text{since} \quad \mathbb{E}_{Y\sim q}\left[\frac{p(Y)}{q(Y)}\right] = 1\right) \end{aligned}$$

- Furthermore, $D_{\mathrm{KL}}\left(p||q\right) = 0$ iff there is an equality in Jensen's inequality, which means $p = q$.

Back to Entropy - Proposition 2 -

- We want to prove the proposition $H(Y) \leq \log_2(\text{Card}(\mathcal{Y}))$
- We have for all distributions p and q:

$$D_{\text{KL}}(p||q) = \sum_{y \in \mathcal{Y}} p(y) \log \frac{p(y)}{q(y)} \quad \text{(definition)}$$

$$= -\sum_{y \in \mathcal{Y}} p(y) \log(q(y)) - \left(-\sum_{y \in \mathcal{Y}} p(y) \log(p(y)) \right)$$

$$= -\sum_{y \in \mathcal{Y}} p(y) \log(q(y)) - H(Y)$$

- Hence, by choosing $\forall y \in \mathcal{Y} \quad q^*(y) = \frac{1}{\text{Card}(\mathcal{Y})}$:

$$H(Y) = \log(\text{Card}(\mathcal{Y})) - \underbrace{D_{\text{KL}}(p||q^*)}_{\geq 0} \leq \log_2(\text{Card}(\mathcal{Y}))$$

Information Gain

Information Gain - Definition -

- We define the **information gain** at a split on the attribute c as follows:

$$IG(D_p, c) = I(D_p) - \frac{N_{\text{left}}}{N_p} I(D_{\text{left}}) - \frac{N_{\text{right}}}{N_p} I(D_{\text{right}})$$

  - D_p refers to the dataset of the parent.
  - D_{left} and D_{right} are the datasets of the left and right child nodes. (For simplicity, most libraries only implement binary decision trees).
  - I is the **impurity** measure.
  - N_p is the total number of samples at the parent node.
  - N_{left} and N_{right} are the total number of samples at the right and left child nodes.

Different impurity measures

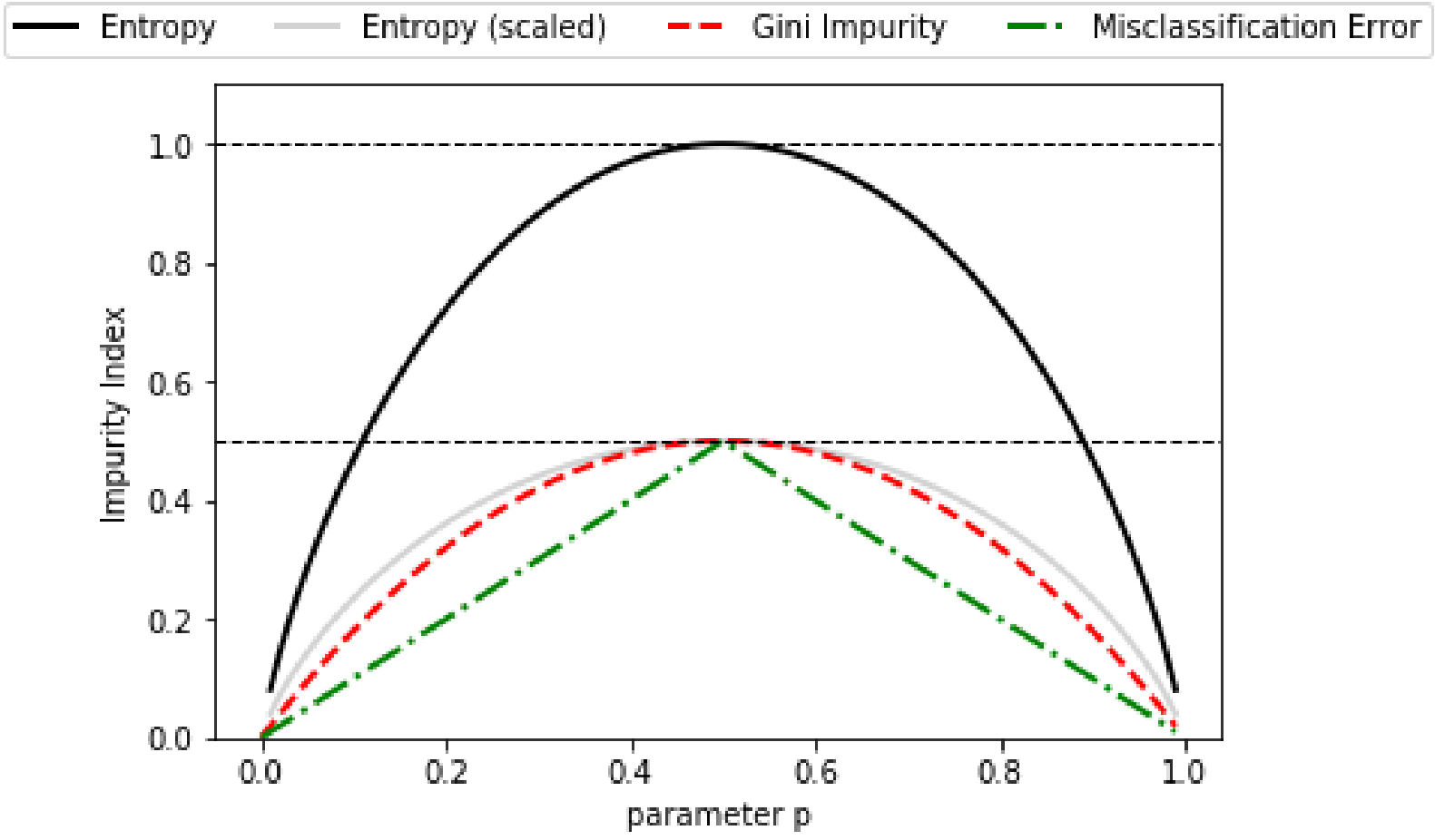
- The three impurity measures or splitting criteria that are commonly used in binary decision trees are **Gini Impurity** (I_G), **Entropy** (I_H), and the **Classification Error** (I_E).
- Let's denote p_k^m the proportion of the samples that belong to the class $m \in \{1, \ldots, M\}$ for a particular node k. We define the above stated impurities as follows:

$$I_G(k) = -\sum_{m=1}^{M} p_k^m (1 - p_k^m)$$

$$I_H(k) = -\sum_{m=1}^{M} p_k^m \log_2(p_k^m)$$

$$I_E(k) = 1 - \max_{1 \leq m \leq M} p_k^m$$

Different impurity measures - Figure -



A simple example - Description -

Let's consider the following example

- We want to predict whether a person is going to the good place ($Y = 1$) or to the bad place ($Y = 0$) based on 4 features:
- $X_1 \in \{0, 1\}$ is whether the person has ever used the expression "au jour d'aujourd'hui" ($X_1 = 0$ if it's the case)
- X_2 is a consequentialist score (in $[0, 100]$) reflecting the number of good things the person has done on earth.
- X_3 is the number of times the person has lied (in millions).
- $X_4 \in \{0, 1\}$ is whether the person is from CERMICS or not. ($X_4 = 1$ if it's the case)

A simple example - Dataset -

- We end up with the following small dataset of 10 samples.
- Let's use the entropy as the impurity measure.
- We have at the root:

$$I(D_p) = H(Y) = -\frac{1}{2}\log_2(\frac{1}{2}) - \frac{1}{2}\log_2(\frac{1}{2}) = 1$$

Forbidden_expression	Score	Lies	CERMICS	Y
0	98	0	0	0
0	87	1	0	0
1	93	9	0	1
1	97	4	1	1
1	81	7	0	1
1	42	8	1	0
1	28	3	0	0
1	12	1	0	1
1	8	0	0	1
1	2	9	0	0

A simple example - split on CERMICS column -

Let's split on the attribute "CERMICS".

$$I(D_{\text{left}}) = H(Y|\text{CERMICS} = 0) = 1$$

Forbidden_expression	Score	Lies	CERMICS	Y
0	98	0	0	0
0	87	1	0	0
1	93	9	0	1
1	81	7	0	1
1	28	3	0	0
1	12	1	0	1
1	8	0	0	1
1	2	9	0	0

$$I(D_{\text{right}}) = H(Y|\text{CERMICS} = 1) = 1$$

Forbidden_expression	Score	Lies	CERMICS	Y
1	97	4	1	1
1	42	8	1	0

A simple example - split on CERMICS column -

- If we split on the attribute : "CERMICS", we obtain the following information gain :

$$\begin{aligned} IG(D_p, \text{"}CERMICS\text{"}) &= I(D_p) - \frac{N_{\text{left}}}{N_p} I(D_{\text{left}}) - \frac{N_{\text{right}}}{N_p} I(D_{\text{right}}) \\ &= 1 - \frac{8}{10} \times 1 - \frac{2}{10} \times 1 \\ &= 0 \end{aligned}$$

- We conclude that there is absolutely nothing to gain from splitting on the "CERMICS" attribute.

A simple example - split on FE column -

- If we split on the attribute : "Forbidden Expression" (FE), we obtain the following information gain :

$$IG(D_p, "FE") = 1 - \frac{2}{10} \times 0 - \frac{8}{10} \times 0.95 = 0.24$$

Forbidden_expression	Score	Lies	CERMICS	Y
0	98	0	0	0
0	87	1	0	0
1	93	9	0	1
1	97	4	1	1
1	81	7	0	1
1	42	8	1	0
1	28	3	0	0
1	12	1	0	1
1	8	0	0	1
1	2	9	0	0

Finding the best split for a particular column

- In the last example, we made it easy : "CERMICS" and "FE" columns have only two possible values each.
- For continuous data, we can find some rules that lead to a smaller set of possible values.
- So to find the best split for a continuous attribute X:
  - We sort the values of the attribute in order, and sort the target Y in the corresponding way.
  - We find all the boundary points (i.e, where Y changes from one value to an other).
  - We calculate the information gain when splitting at each boundary.
  - We keep the split which gives the highest information gain.

Example - Split on "Lies" -

- We can see that the best split (with the highest information gain) is still very low.
- For the first split (at the root), it is better to do it on FE column as the information gain is 0.24.

Thresholds	Forbidden_expression	Score	Lies	CERMICS	Y	IG
	0	98	0	0	0	
0.5 →	1	8	0	0	1	
	1	12	1	0	1	
	0	87	1	0	0	
2 →	1	28	3	0	0	
3.5 →	1	97	4	1	1	← 0.03
5.5 →	1	81	7	0	1	
7.5 →	1	42	8	1	0	← 0.04
8.5 →	1	2	9	0	0	
	1	93	9	0	1	

Example - Split on "Lies" -

- After some iterations, splitting on "Lies" becomes interesting.

Thresholds	Forbidden_expression	Score	Lies	CERMICS	Y	IG
	1	8	0	0	1	
2 →	1	12	1	0	1	← 0.97
	1	28	3	0	0	
5.5 →	1	42	8	1	0	
8.5 →	1	2	9	0	0	

Leaf Nodes

When do we stop splitting ?

- If a node is pure (all the samples of this node belong to the same category), we make it a leaf node and predict the class of the samples.
- If the maximum information gain = 0, we gain nothing from splitting, we make the node a leaf node and predict the most likely class (majority vote).
- To avoid overfitting, we set a limit to the depth of the tree.

Pseudocode

Training Algorithm (rough outline)

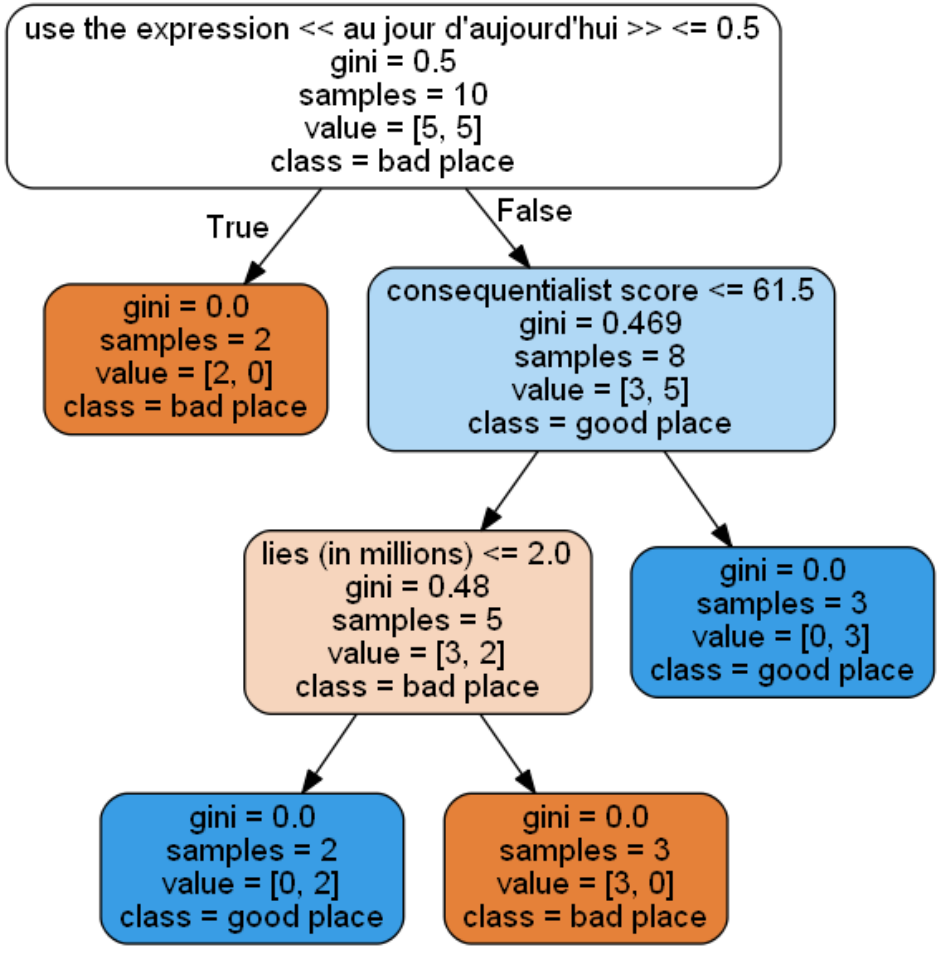
```
1  def fit(X,Y):
2          max_IG = 0
3          Best_attribute = None
4          for c in columns:
5              condition = find_split(X, Y, c) # later
6              Y_left = Y[X[c] meets condition]
7              Y_right = Y[X[c] does not meet condition]
8              IG = Entropy(Y) - p(left)*Entropy(Y_left)
                   - p(right)*Entropy(Y_right)
9              If IG > max_IG:
10                 max_IG = IG
11                 Best_attribute = c
12     ...
```

Training Algorithm (Recursiveness)

```
1  def fit(X,Y):
2      # We find the best attribute
3          max_IG = ...
4          Best_attribute = ...
5      # Call fit recursively
6          X_left, Y_left, X_right, Y_right = split by
               best attribute
7          self.left_node = TreeNode()
8          self.left_node.fit(X_left, Y_left)
9          self.right_node = TreeNode()
10         self.right_node.fit(X_right, Y_right)
```

Basic example after training

By applying the training algorithm on the basic example, we obtain the following tree:

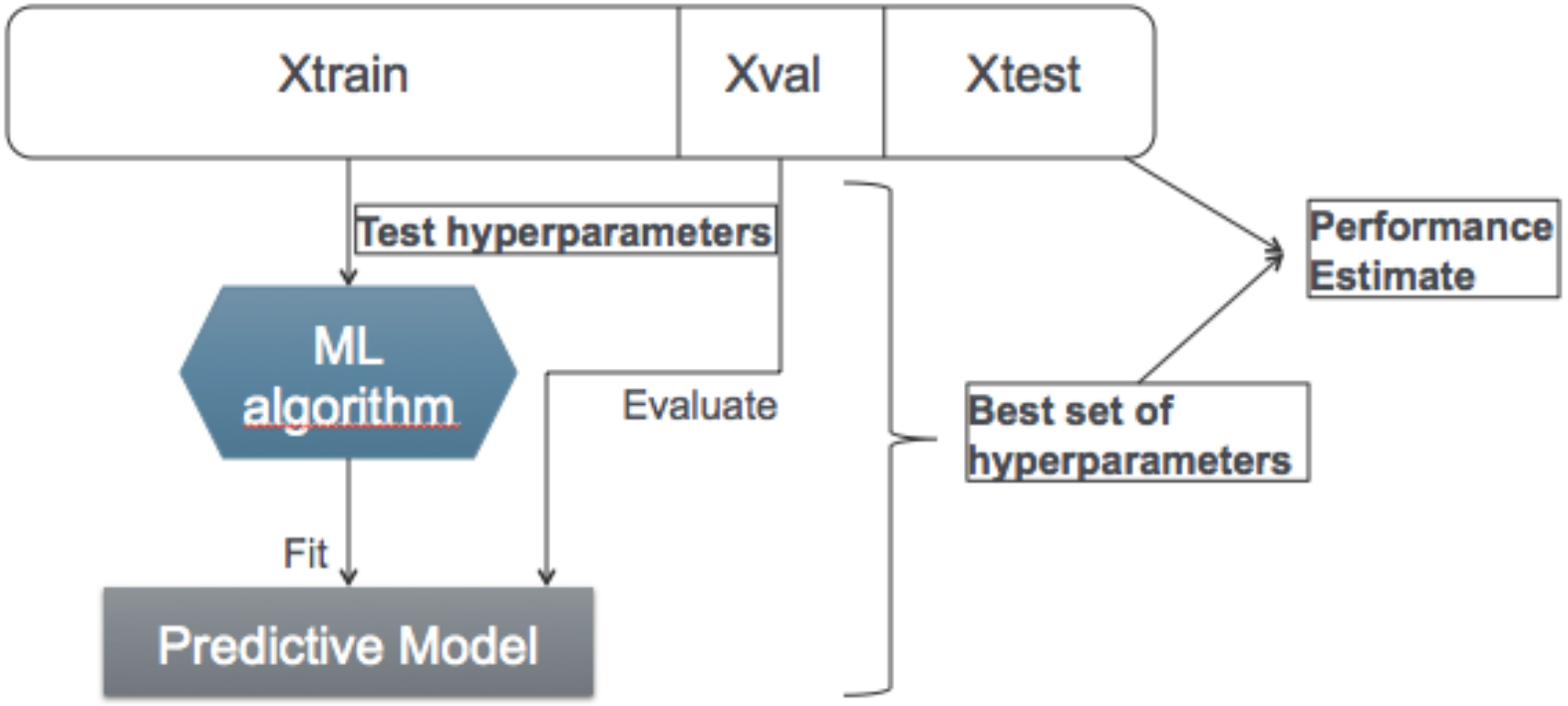


Model Evaluation and Model Selection

K-fold cross-validation for assessing model performance

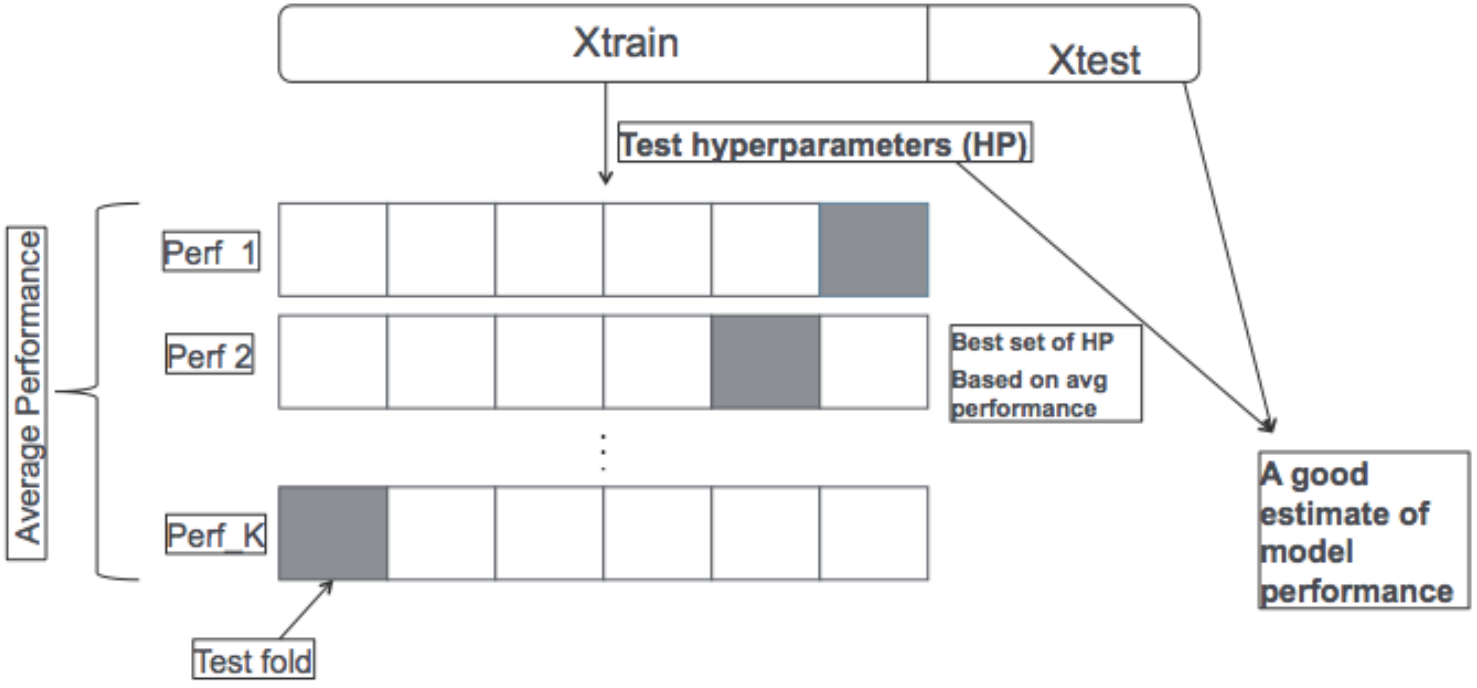
The classic holdout method

A popular approach for estimating the generalization performance of machine learning models is the **holdout cross-validation**



K-fold cross-validation

As the performance of the holdout method is sensitive to how we partition the dataset. A more robust technique for performance estimation is the **K-fold cross validation**



Decision Tree on Iris dataset

- The data set consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor), so 150 total samples.
- Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters.

(c) Setosa

(d) Versicolor

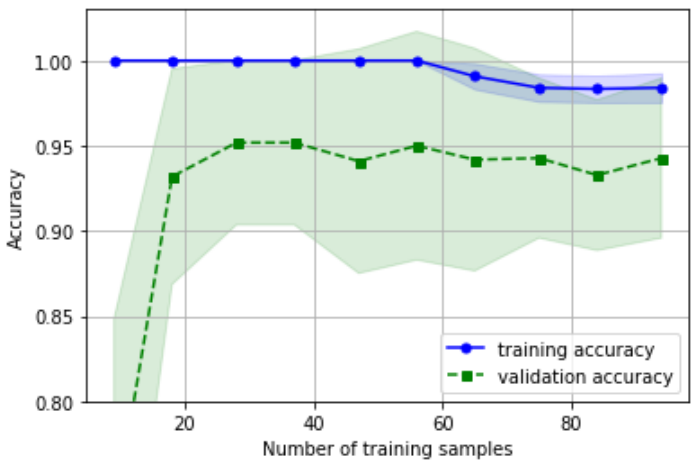
(e) Virginica

DT on Iris Dataset - Model Selection -

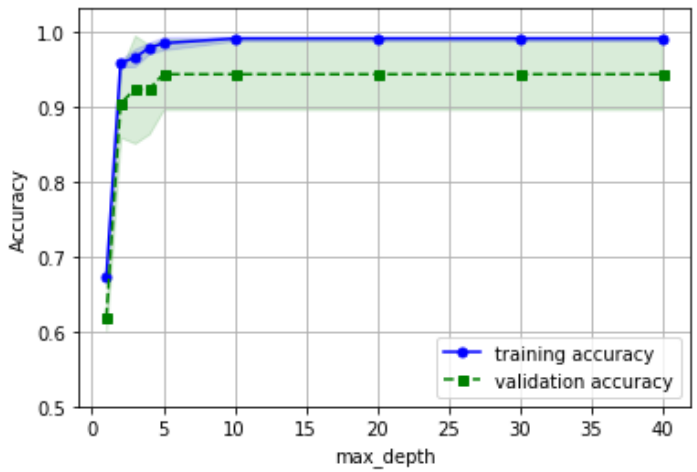
- To find the optimal set of hyperparameters, we use **grid search**, which is a brute-force exhaustive search paradigm where we specify a list of values for different hyperparameters, and the algorithm evaluate the model performance (via cross validation) for each combination of hyperparameters.
- For decision tree algorithm, we tuned the following two hyperparameters:
  - **Impurity measure** with the two possibilities : "gini" and "entropy"
  - **The depth of the tree** with values in $[1, 2, 3, 4, 5, 10, 20, 30, 40]$
- As a result, we obtain the following best parameters : best impurity : 'gini' and best depth : 5

DT on Iris Dataset - Bias and Variance Problems -

By plotting the model training and validation accuracies as functions of the training set size, we can detect whether the model suffers from high variance or high bias, and whether the collection of more data could help address this problem.



(f) Accuracy vs training set size



(g) Accuracy vs max depth

Random Forest

Combining multiple decision trees via Random Forest

The Random Forest Algorithm

The Random Forest algorithm aims to average multiple decision trees that individually suffer from overfitting, in order to build a model with better generalization performance. It consists in the following steps:

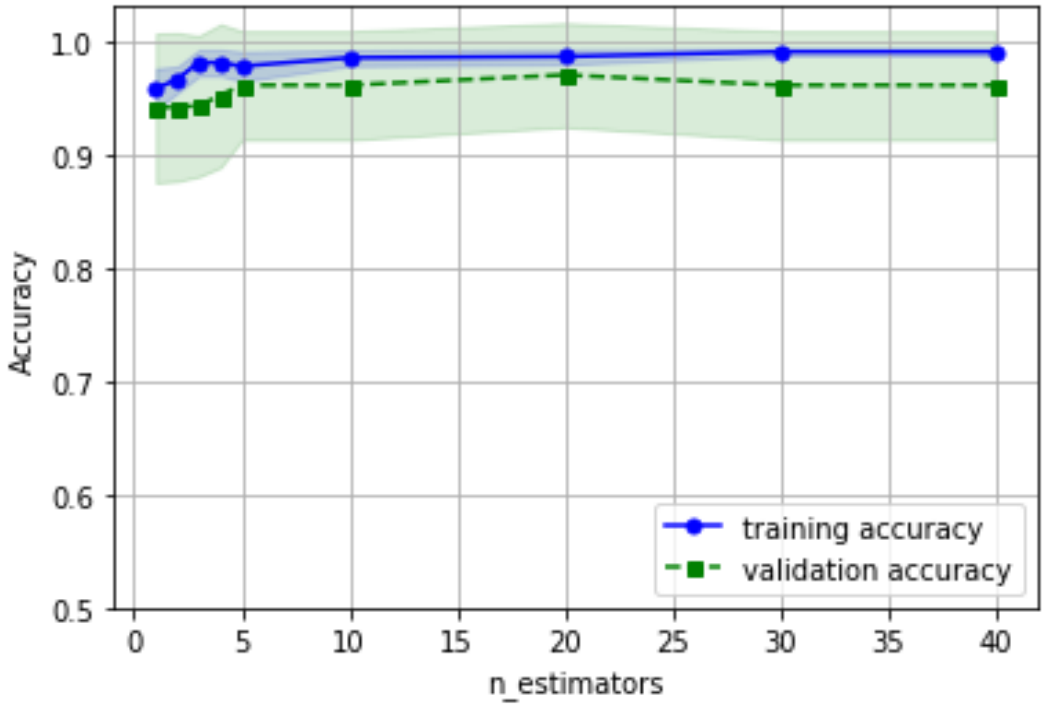
1. Randomly choose n samples from the training set **with replacement** (bootstrap sample).
2. Grow a decision tree from the bootstrap sample. And at each node:
   - Randomly select d attributes without replacement.
   - Split the node using the attribute that provides the best split according to maximizing the information gain.
3. Repeat the firs two steps K times.
4. Use the K trees to assign the class label by **majority voting**.

Random Forest Training

```
1  def fit(X,Y):
2      models = []
3      for k in range(K):
4          X_k, Y_k = sample_with_replacement(X, Y)
5          model = DecisionTree()
6          while not at terminal node and not reached
               max_depth:
7              select randomly d attributes
8              choose best split from the d attributes
9              add split to model
10         models.append(model)
```

- The sample size n of the bootstrap controls the **bias-variance** tradeoff of the Random Forest.
- In most libraries (including scikit-learn), the size of the bootstrap sample is chosen to be equal to the number of samples in the training dataset.
- For the number of attributes d at each split, a reasonable default value (used in scikit-learn) is $d = \sqrt{D}$ where D is the number of attributes in the dataset.
- So, one big advantage of Random Forest is that it requires very little tuning: The main hyperparameter to tune is the number of trees.

- By tuning the number of trees as a hyperparameter for the Iris dataset, we get 20 as the best value, which is confirmed by the following curve.



Thank you for your attention ☺