COMP 432 Machine Learning

Decision Trees

Computer Science & Software Engineering
Concordia University, Fall 2024

Summary of the last episode....

What we have seen the **last lecture:**

- Non-Linear Support Vector Machines

What we are going to **learn today:**

- Decision Trees

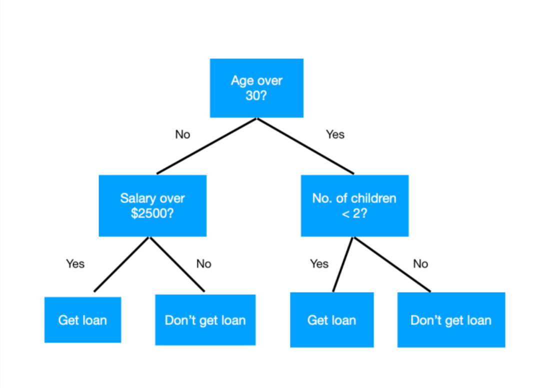
Decision Trees

- We already have seen in detail popular machine learning algorithms such as:

  - Linear Models (linear regression, logistic regression, multi-class logistic regression)
  - Neural Networks (MLPs, CNNs, RNNs)
  - SVM (linear and non-linear).

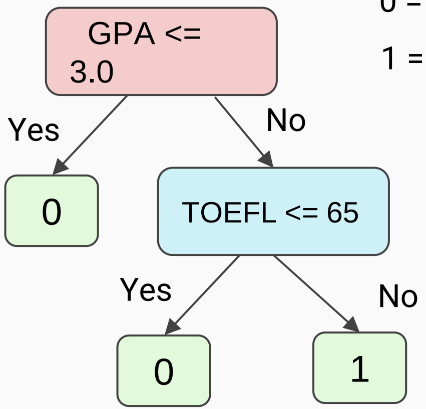
Now, we are going to give an overview of other popular supervised machine learning algorithms.

Let's start with **decision trees**!



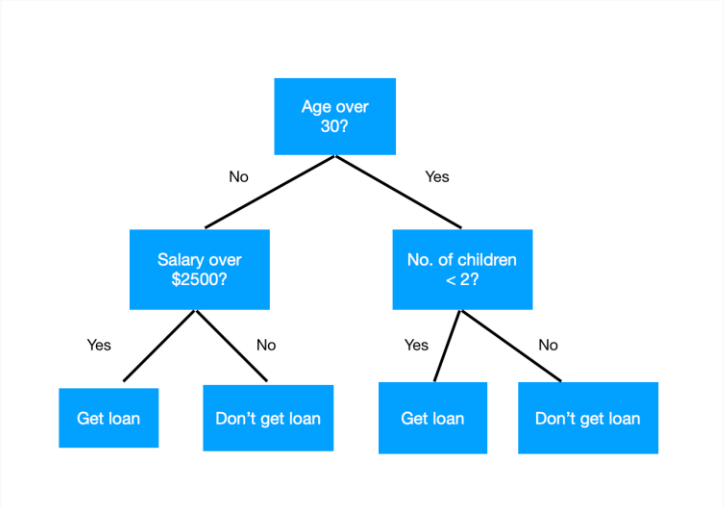
Decision Trees

- *What is a decision tree?*



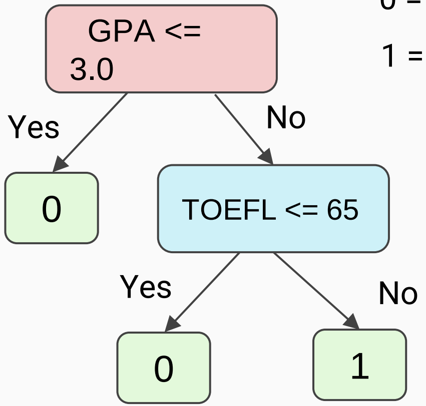
0 = “Not to hire”

1 = “To Hire”



- The decision tree poses **questions** about one **specific feature.**
- If answers are YES/NO, we have a **binary tree** (object of this lecture)
- The questions can be of any type (e.g., involving numbers, attributes, categories, etc.)

Decision Trees



0 = “Not to hire”

1 = “To Hire”

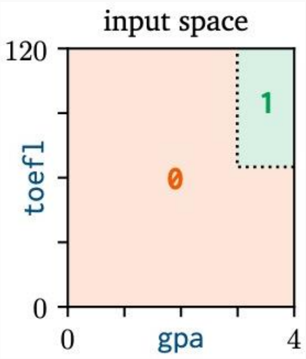
- Binary numerical questions often take this form:

$$\boxed{x_j \leq \tau\ ?}$$

Where:

- x_j is one of the features of the input $\mathbf{x}=[x_1,x_1,\ldots, x_D]^T$
- τ is the threshold value.

- The last nodes (leaf) corresponds to a particular class.

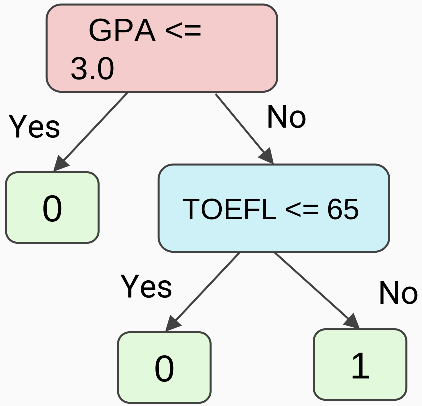



```
int accept(float gpa, int toefl) {
    if (gpa < 3.0)
        return 0;           // REJECT
    else if (toefl < 65)
        return 0;           // REJECT
    else
        return 1;           // ACCEPT
}
```

- The decision tree partitions the input space using **cuboid regions** (e.g. rectangles in a 2D feature space),

- A decision tree is a program comprising nested **if-else** and **returns** statements.

Decision Trees



- *How can we train a decision tree?*

We have a dataset D composed of input features and labels:

$$D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\} \longrightarrow \text{Dataset}$$

$$y_i \in \{1, \ldots, K\} \longrightarrow \text{Label (K categories)}$$

$$\mathbf{x}_i \in \mathbb{R}^D \longrightarrow \text{Input (D features)}$$

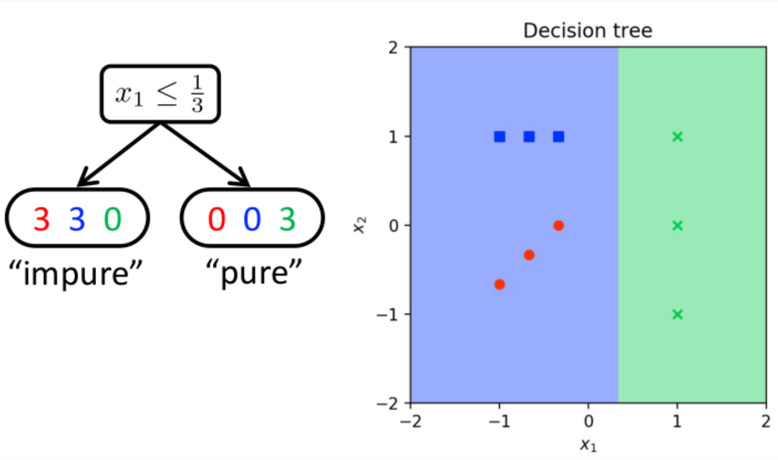
Given D, we want to learn a decision tree.

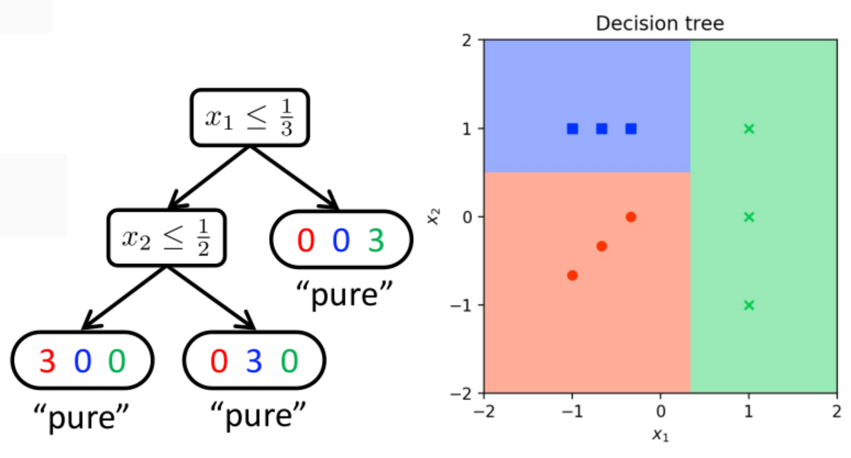
- *Which features should we test first? With which threshold τ?*
- *When should we stop splitting and declare a leaf?*

Notion of Impurity

- To train a decision tree, we need to introduce the notion of **impurity.**

- If all training samples that arrive at a leaf are the same class, the leaf is "**pure**."

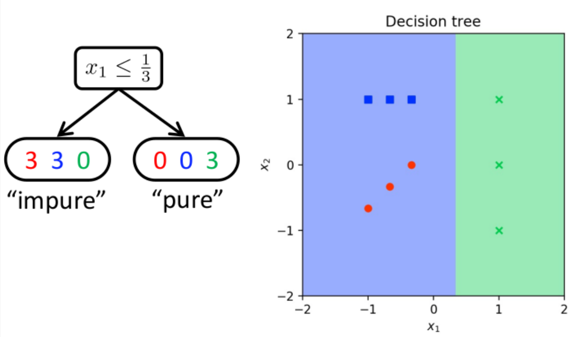


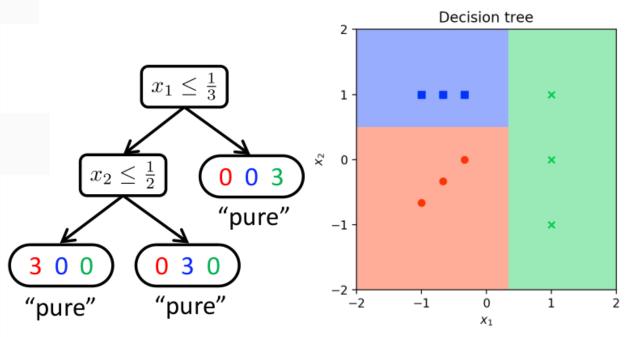


<u>Intuitively:</u>

- If a leaf is "impure" then we may decide to split.
- If we do split, we guarantee the subtrees have leaves that are strictly "more pure."

Notion of Impurity



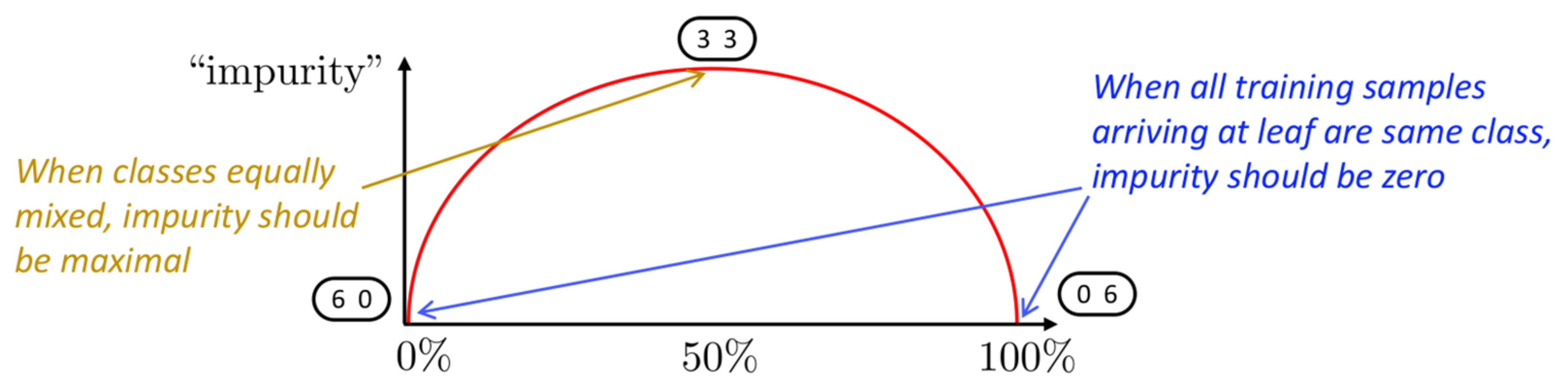


<u>Intuitively:</u>

- If a leaf is "impure" then we may decide to split.
- If we do split, we guarantee the subtrees have leaves that are strictly "more pure."
- The training algorithm should find splits that give the lowest class impurity (uncertainty)!

Notion of Impurity

- *How can we measure impurity?*
- From an impurity measure we want something like this:



Example for 2 classes and 6 data points

Notion of Impurity

- There are many measures of impurity.

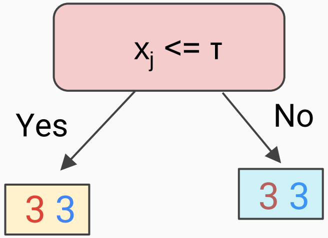
- The most used one is called **Gini impurity** (or **Gini index**):

$$I_{gini} = \frac{1}{2}\left(1 - \sum_{k=1}^{K} \underbrace{P(y=k)}^{2}\right)$$

Probability that a training sample $\mathbf{x}_i$ that arrives at this node will have class y_i=k

This can be approximated with the fraction of training samples arriving at this node with y_i=k

Example



$$I_L = \frac{1}{2}\left(1 - \left(\frac{3}{6}\right)^2 - \left(\frac{3}{6}\right)^2\right) = 0.25$$

$$I_R = \frac{1}{2}\left(1 - \left(\frac{3}{6}\right)^2 - \left(\frac{3}{6}\right)^2\right) = 0.25$$

Notion of Impurity

- There are many measures of impurity.

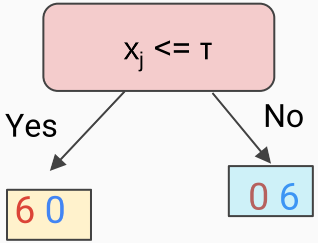
- The most used one is called **Gini impurity** (or **Gini index**):

$$I_{gini} = \frac{1}{2}\left(1 - \sum_{k=1}^{K} P(y=k)^2\right)$$

Probability that a training sample $\mathbf{x}_i$ that arrives at this node will have class y_i=k

This can be approximated with the fraction of training samples arriving at this node with y_i=k

Example



$$I_L = \frac{1}{2}\left(1 - \left(\frac{0}{6}\right)^2 - \left(\frac{6}{6}\right)^2\right) = 0.0$$

$$I_R = \frac{1}{2}\left(1 - \left(\frac{6}{6}\right)^2 - \left(\frac{0}{6}\right)^2\right) = 0.0$$

Notion of Impurity

- There are many measures of impurity.

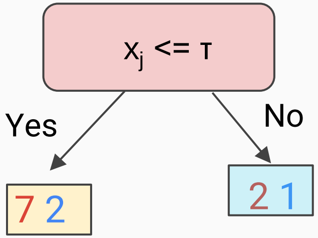
- The most used one is called **Gini impurity** (or **Gini index**):

$$I_{gini} = \frac{1}{2}\left(1 - \sum_{k=1}^{K} \underbrace{P(y=k)}^{2}\right)$$

Probability that a training sample $\mathbf{x}_i$ that arrives at this node will have class y_i=k

This can be approximated with the fraction of training samples arriving at this node with y_i=k

Example



I_L = $\frac{1}{2}\left(1 - \left(\frac{7}{9}\right)^2 - \left(\frac{2}{9}\right)^2\right) = 0.17$

I_R = $\frac{1}{2}\left(1 - \left(\frac{2}{3}\right)^2 - \left(\frac{1}{3}\right)^2\right) = 0.22$

Notion of Impurity

- To measure how good is a split, we need a measure of impurity that considers both the left and the right nodes:

$$I = \underbrace{P(x_i \leq \tau)}_{\text{Fraction of samples going to the left}} I_L + \underbrace{(1 - P(x_i \leq \tau))}_{\text{Fraction of samples going to the right}} I_R$$

This is the best split because it has the lowest I

$x_i \leq \tau$		$x_i \leq \tau$		$x_i \leq \tau$	
Yes	No	Yes	No	Yes	No
3 3	3 3	7 2	2 1	6 0	0 6

$$I_L = \frac{1}{2}\left(1 - \left(\frac{3}{6}\right)^2 - \left(\frac{3}{6}\right)^2\right) = 0.25$$

$$I_R = \frac{1}{2}\left(1 - \left(\frac{3}{6}\right)^2 - \left(\frac{3}{6}\right)^2\right) = 0.25$$

$$I = \frac{6}{12} \cdot 0.25 + \frac{6}{12} \cdot 0.25 = 0.25$$

$$I_L = \frac{1}{2}\left(1 - \left(\frac{7}{9}\right)^2 - \left(\frac{2}{9}\right)^2\right) = 0.17$$

$$I_R = \frac{1}{2}\left(1 - \left(\frac{2}{3}\right)^2 - \left(\frac{1}{3}\right)^2\right) = 0.22$$

$$I = \frac{9}{12} \cdot 0.17 + \frac{3}{12} \cdot 0.22 = 0.18$$

$$I_L = \frac{1}{2}\left(1 - \left(\frac{0}{6}\right)^2 - \left(\frac{6}{6}\right)^2\right) = 0.0$$

$$I_R = \frac{1}{2}\left(1 - \left(\frac{6}{6}\right)^2 - \left(\frac{0}{6}\right)^2\right) = 0.0$$

$$\boxed{I = \frac{6}{12} \cdot 0.0 + \frac{6}{12} \cdot 0.0 = 0.0}$$

Splitting threshold

- *Given a feature x_i, how can we choose the value for the threshold τ?*
- We can just try different τ and choose the one that minimizes I:

$$\boxed{\tau^* = \underset{\tau}{\operatorname{argmin}}\ I(\tau)}$$

Can we solve this problem easily?

  - τ is a real number but only a few thresholds really make sense.

GPA (x_1)	To Hire (y)
2.5	0
3.0	0
3.5	1
4.0	0
4.0	1

Which thresholds does it make sense to try?

We can test:
- $x_1 <= 2.0$ → Non meaningful: all elements partitioned into one leaf or zero in the other.
- $x_1 <= 2.5$
- $x_1 <= 2.8$
- $x_1 <= 3.0$
- $x_1 <= 3.5$
- $x_1 <= 4.0$

Testing thresholds resulting in 0 elements in a leaf is meaningless!

Splitting threshold

- *Given a feature x_j, how can we choose the value for the threshold τ?*
- We can just try different τ and choose the one that minimizes I:

$$\boxed{\tau^* = \underset{\tau}{\operatorname{argmin}}\ I(\tau)}$$

Can we solve this problem easily?

  - τ is a real number but only a few thresholds really make sense.

GPA (x_1)	To Hire (y)
2.5	0
3.0	0
3.5	1
4.0	0
4.0	1

Which thresholds does it make sense to try?

We can test:

$$\left\{\begin{array}{l} x_1 <= 2.0 \ ✗ \\ x_1 <= 2.5 \ ✓ \\ x_1 <= 2.8 \ ✗ \\ x_1 <= 3.0 \\ x_1 <= 3.5 \\ x_1 <= 4.0 \end{array}\right.$$

Non meaningful: this leads to the same split as for x_1 <= 2.5

⚠ Testing thresholds between two feature values is meaningless!

Splitting threshold

- *Given a feature x_i, how can we choose the value for the threshold τ?*
- We can just try different τ and choose the one that minimizes I:

$$\tau^* = \underset{\tau}{\operatorname{argmin}}\ I(\tau)$$

Can we solve this problem easily?

  - τ is a real number but only a few thresholds really make sense.

GPA (x_1)	To Hire (y)
2.5	0
3.0	0
3.5	1
4.0	0
4.0	1

Which thresholds does it make sense to try?

We can test:

$x_1 <= 2.0$ ✗

$x_1 <= 2.5$ ✓

$x_1 <= 2.8$ ✗

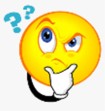
$x_1 <= 3.0$ ✓

$x_1 <= 3.5$ ✓

$x_1 <= 4.0$ ✗

Non meaningful: all elements partitioned into one leaf or zero in the other.

Splitting threshold



How can we identify the thresholds to test?

GPA (x_1)	To Hire (y)
2.5	0
3.0	0
3.5	1
4.0	0
4.0	1

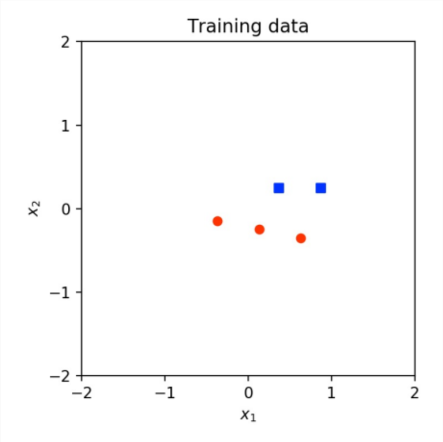
1. Sort feature values in **ascending order** (remove duplicates):
   $\tau=\{2.5, 3.0, 3.5, 4.0\}$
2. Remove the **last element**:
   $\tau=\{2.5, 3.0, 3.5\}$
3. Get the **splits** with the valid threshold:
   $x_1 <= 2.5$ $\quad$ $x_1 <= 3.0$ $\quad$ $x_1 <= 3.5$
4. Compute the **impurity** for each valid threshold:
   $I(\tau=2.5)=0.20$ $\quad$ $I(\tau=3.0)=0.13$ $\quad$ $I(\tau=3.5)=0.23$
5. Choose the threshold with the **minimum impurity**:
   $I(\tau=2.5)=0.20$ $\quad$ $\boxed{I(\tau=3.0)=0.13}$ $\quad$ $I(\tau=3.5)=0.23$
6. Set final threshold **halfway** between the best and the subsequent threshold.
   $\boxed{\tau = (3.0+3.5/2)=3.25}$

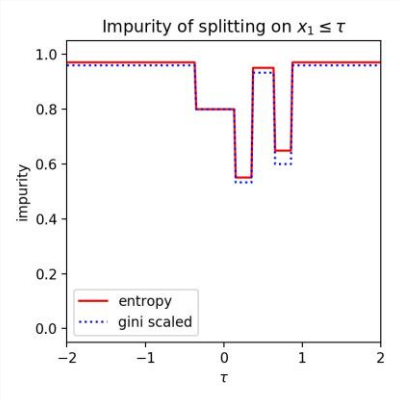
Choosing the feature

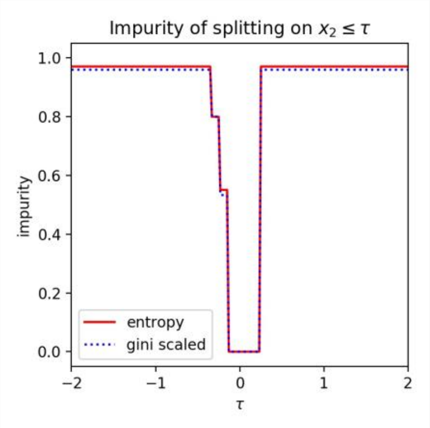
- *How can we choose the feature x_j?*

There are a few options:

1. **Random**: Choose a $j \in \{1, .., D\}$ at random.
2. **Best**: Choose the j that allows for the smallest impurity I(τ). This might be expensive…..
3. **Best from a random subset**. A compromise.







When to stop splitting?

- **Case 1**: When the leaf is **pure**, stop splitting.
- **Case 2**: When a leaf is impure but I(τ) **cannot be decreased** for any choice of τ, stop splitting.
- However, using only the above criteria will lead to complex decision trees that likely **overfit** the data.
- So, we may also want to ‘**regularize**’ by imposing:

Max depth: If the height of the leaf in the tree is already at some maximum depth, stop splitting.

Min samples: If fewer than some number of training samples can arrive at the leaf, stop splitting

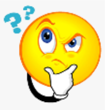
When to stop splitting?

- We can regularize the decision trees with a few heuristics such as:

Max depth: If the height of the leaf in the tree is already at some maximum depth, stop splitting.

Min samples: If fewer than some number of training samples can arrive at the leaf, stop splitting

When we regularize the tree, we might end up with **impure end nodes**.



What can be the classification class for non-pure end nodes?

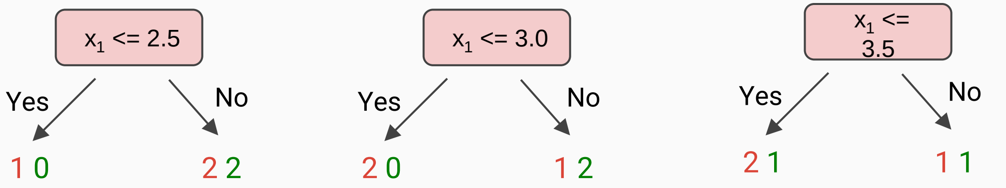
We can just vote for the class that gets the majority of votes in the leaf node.

Example

- Let's derive the decision tree for this case

GPA (x_1)	TOEFL (x_2)	To Hire (y)
2.5	70	0
3.0	65	0
3.5	80	1
4.0	60	0
4.0	70	1

1. Let's start from GPA (random choice)



I_L=0, I_R=0.25

I = 0.20

I_L=0, I_R=0.22

I = 0.132

I_L=0.22, I_R=0.25

I = 0.232

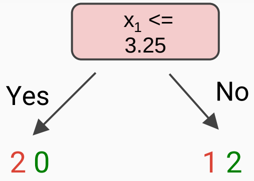
We can choose a threshold between **3.0** and **3.5** (we choose **3.25**)

Example

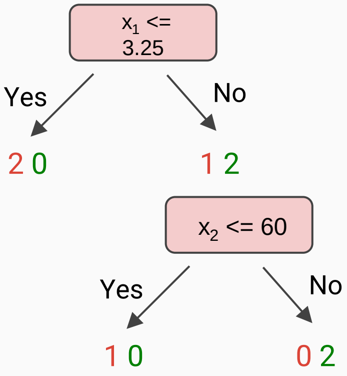
- Let's derive the decision tree for this case

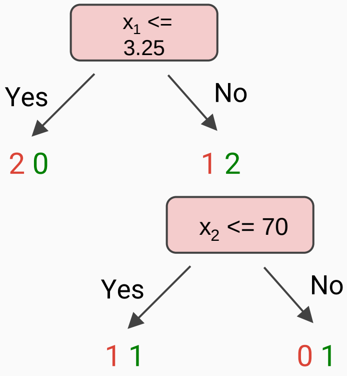
GPA (x_1)	**TOEFL (x_2)**	**To Hire (y)**
2.5	70	0
3.0	65	0
3.5	80	1
4.0	60	0
4.0	70	1

Pure node, stop! ←



1. Let's now use TOEFL





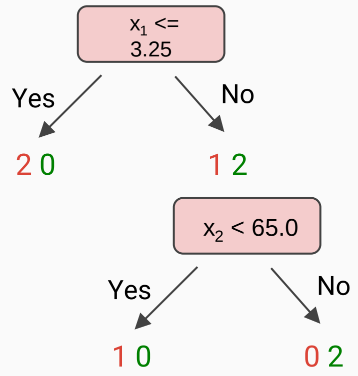
I = 0.0

Pure node, stop! → Threshold at 65.0

Example

- Let's derive the decision tree for this case

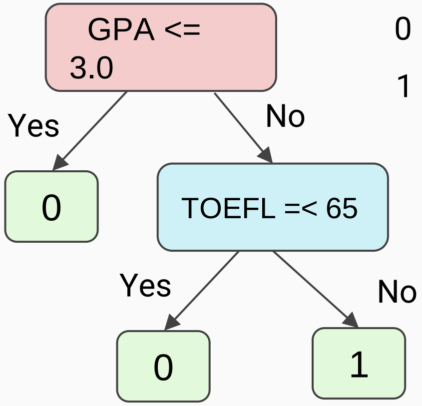
GPA (x_1)	TOEFL (x_2)	To Hire (y)
2.5	70	0
3.0	65	0
3.5	80	1
4.0	60	0
4.0	70	1



Random Forest

Decision Trees

- We have seen that Decision Trees are structured in this way:



0 = “Not to hire”

1 = “To Hire”

$$D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_N, y_N)\}$$

$y_i \in \{1, ..., K\}$ Label (K categories)

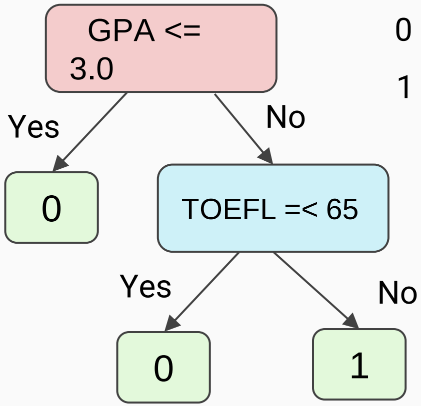
$\mathbf{x}_i \in \mathbb{R}^D$ Input (D features)

- We test (through a question) a specific feature at a time.
- Each leaf node corresponds to a predicted class.
- An **impurity measure** (e.g. Gini Index) helps us quantify “how good” is a split.

One way to train a decision tree is the following:

1. Select one random feature to test.
2. Choose the threshold that minimizes the impurity.
3. Repeat 1-2 until all the nodes are leaves.

Decision Trees



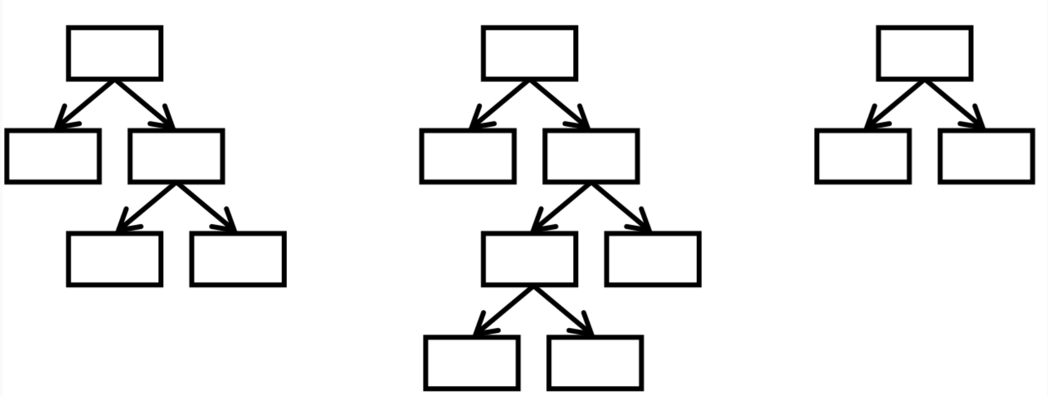
0 = “Not to hire”

1 = “To Hire”

- **Observation**: There is a lot of “arbitrariness” in the way decision trees are built.
  - **Different splits** early on can lead to **completely different trees** and completely **different decision regions**.
  - The tree is highly sensitive to the **training dataset** (e.g, even individual training points can potentially affect a split).

What about if we train multiple decision trees (a.k.a, a forest) and average their predictions?

Random Forest



Tree 1 Tree 2 Tree 3

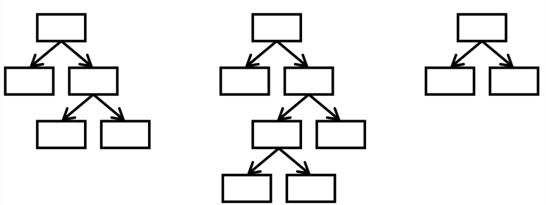
This is not the first time we hear about combining predictions from different models...right?

The random forest algorithm is a **bagging** algorithm.

This has a powerful **regularization** effect.

We indeed "*integrate out*" the arbitrariness that occurs when training a decision tree we get some kind of "*expectation*" over all the decision trees.

Random Forest



To train a “forest” we must introduce “**variation**” into the decision trees.

- One way is to **randomize** the **training algorithm**.

  We can choose a random set of features and find the one that minimizes the impurity.

- One way is to **randomize** the **training dataset**.

  We can take a random subset of the training data (subsample M data points).

  Reasonable! But each new training set is smaller than N.

To have a dataset of size N, we can randomly sample N data samples. Some data will get duplicated, but no parameters to choose. Random Forest does this (in addition to randomizing the training algorithm).

Random Forest

GPA (x_1)	TOEFL (x_2)	To Hire (y)
2.5	70	0
3.0	65	0
3.5	80	1
4.0	60	0
4.0	70	1

Original Dataset

GPA (x_1)	TOEFL (x_2)	To Hire (y)
3.0	65	0
4.0	60	0
2.5	70	0
2.5	70	0
4.0	70	1

Bootstrapped Dataset 1

GPA (x_1)	TOEFL (x_2)	To Hire (y)
4.0	60	0
4.0	60	0
2.5	70	0
3.5	80	1
3.0	65	0

Bootstrapped Dataset 2

GPA (x_1)	TOEFL (x_2)	To Hire (y)
2.5	70	0
3.5	80	1
2.5	70	0
2.5	70	0
4.0	70	1

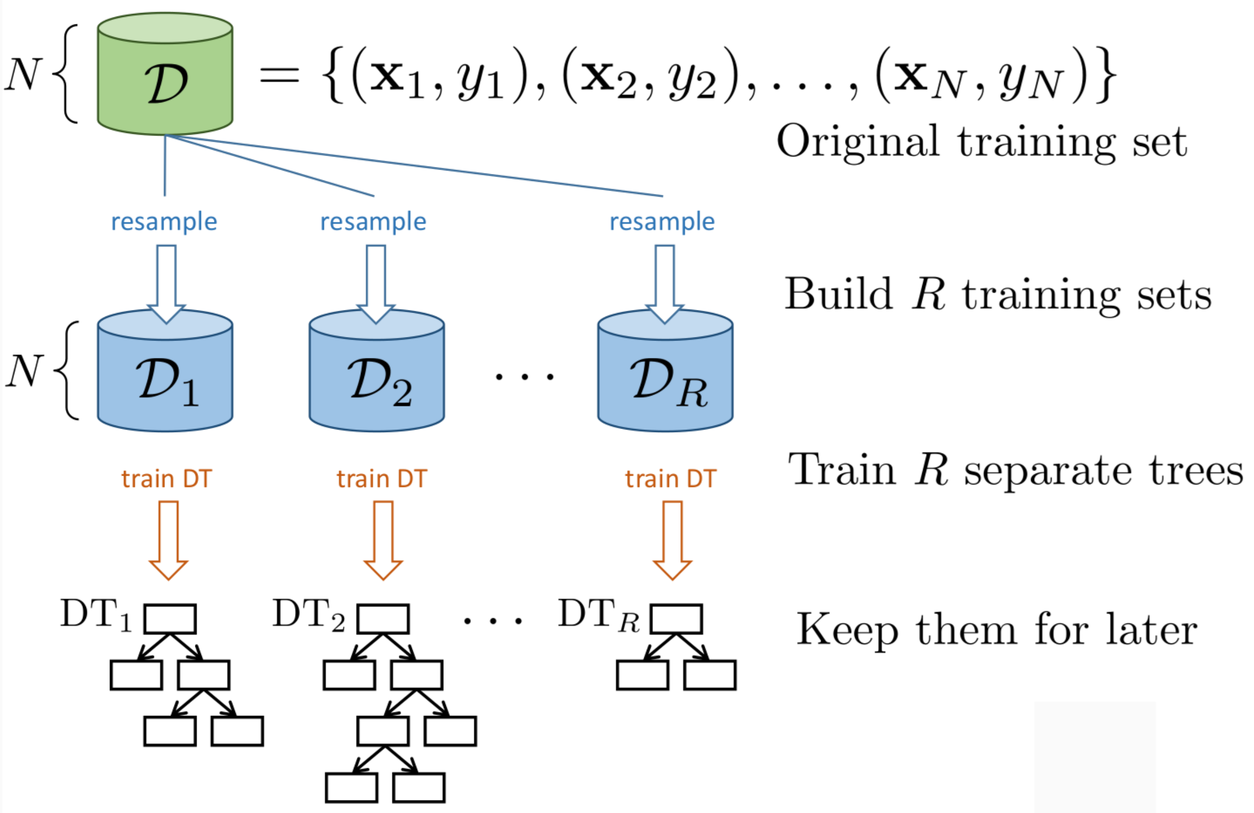
Bootstrapped Dataset 3

....

We select the same data point more than one time!

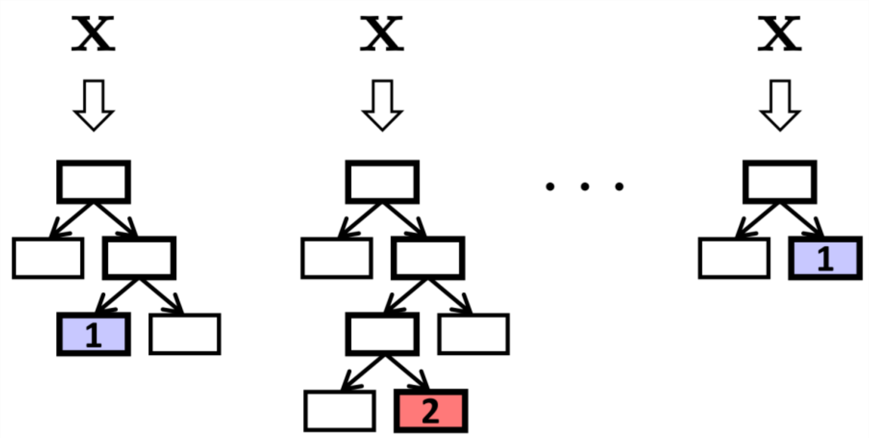
Random Forest

- Once we have the bootstrapped dataset, we can **train** the random forest in this way:

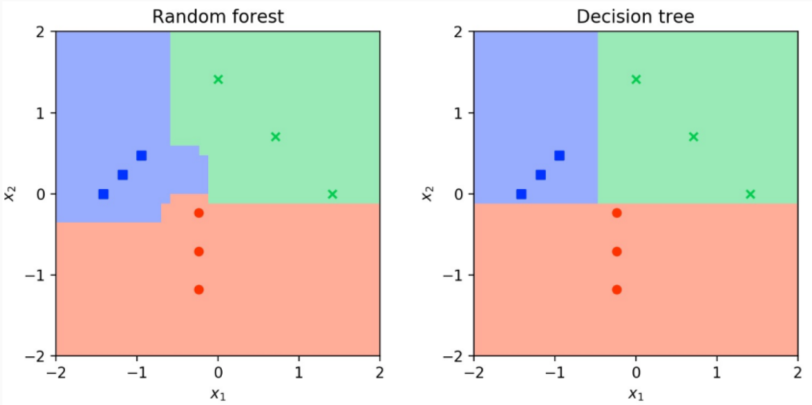


Random Forest

- *How can we do a prediction?*



1. Given a new data point **x**, run it through each tree.
2. Predict the class with the largest share of R ‘votes’.



Regression Trees

- So far, we have seen regression trees for classification.
- *What about regression?*
- Decision trees (and random forest) can be extended to **regression problems**.
- The main difference is that we use the **Mean Squared Error** (MSE) as an impurity measure:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(\hat{y} - y_i)^2$$

N: Number of data points in each node.

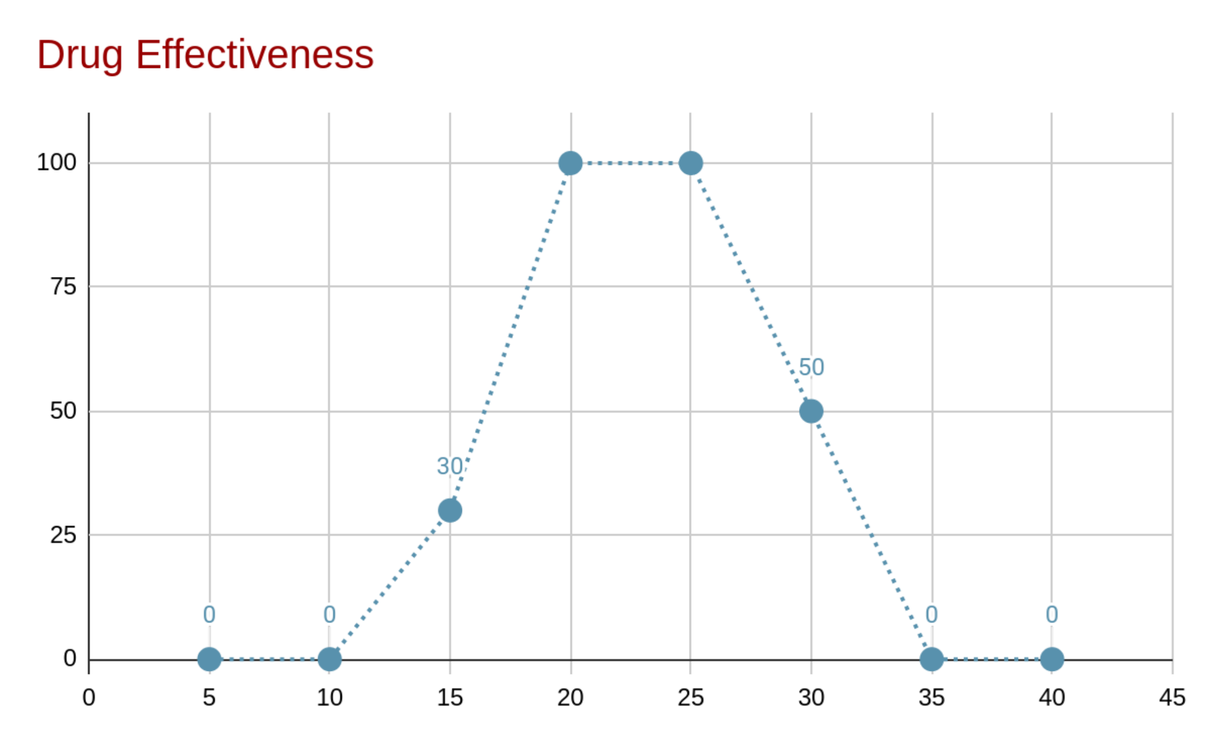
Average value of the labels in the node

Label

Regression Trees (Example)

- Let’s see an example

Dosage (mg)	Drug Effectiveness (%)
5	0
10	0
15	30
20	100
25	100
30	50
35	0
40	0



Regression Trees (Example)

- Let's see an example

Dosage (mg)	Drug Effectiveness (%)
5	0
10	0
15	30
20	100
25	100
30	50
35	0
40	0

Dosage <= 5

Yes: 0

No: 0 30 100 100 50 0 0

Average: y = 0

MSE_L: 0

Average: y = 40

MSE_R: [(0-40)^2 + (30-40)^2 + …]/7 = 1742

MSE = ⅛ * 0 + ⅞ * 1742 = **1524**

Regression Trees (Example)

- Let’s see an example

Dosage (mg)	Drug Effectiveness (%)
5	0
10	0
15	30
20	100
25	100
30	50
35	0
40	0

Threshold at 12.5

Dosage <= 5
MSE = **1524**

Dosage <= 10
MSE = **1291**

Dosage <= 15
MSE = **1325**

Dosage <= 20
MSE = **1693**

Dosage <= 25
MSE = **1498**

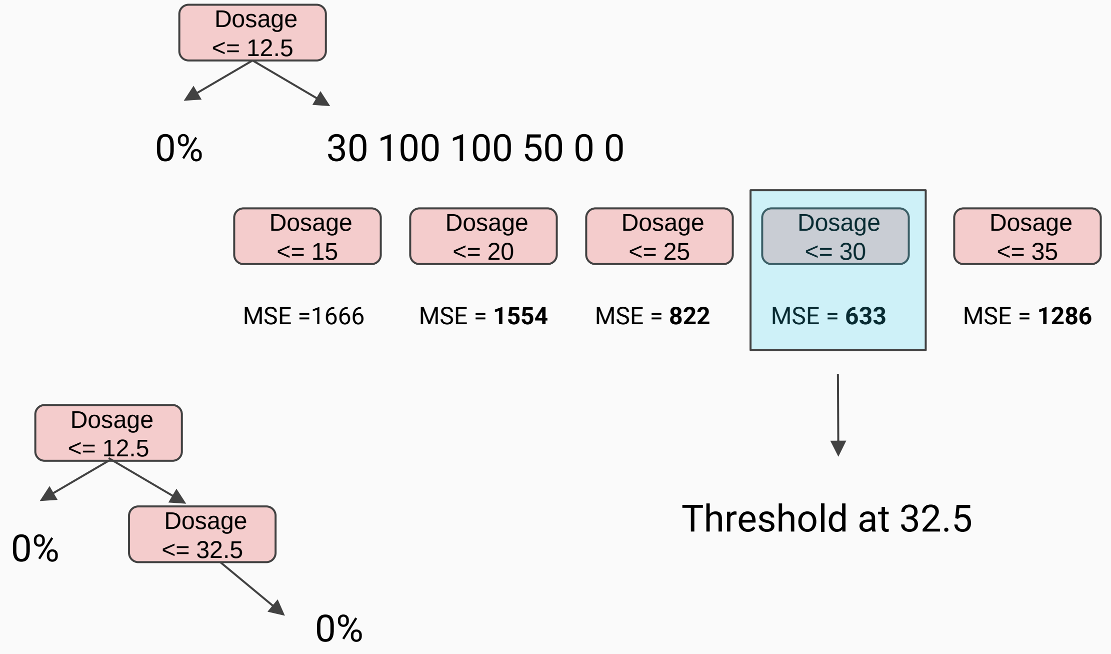
Dosage <= 30
MSE = **1291**

Dosage <= 35
MSE = **1525**

Regression Trees (Example)

- Let’s see an example

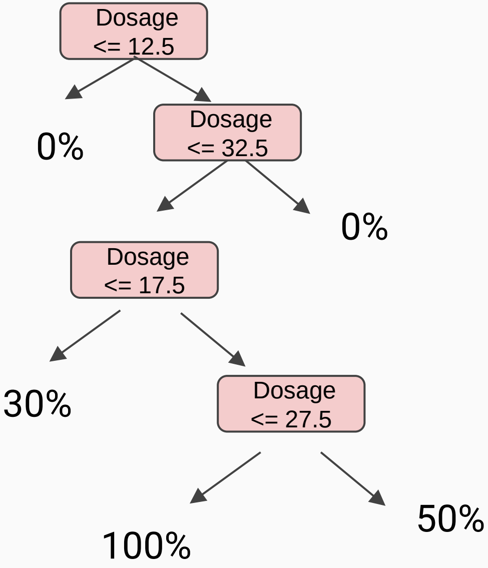
Dosage (mg)	**Drug Effectiveness (%)**
5	0
10	0
15	30
20	100
25	100
30	50
35	0
40	0

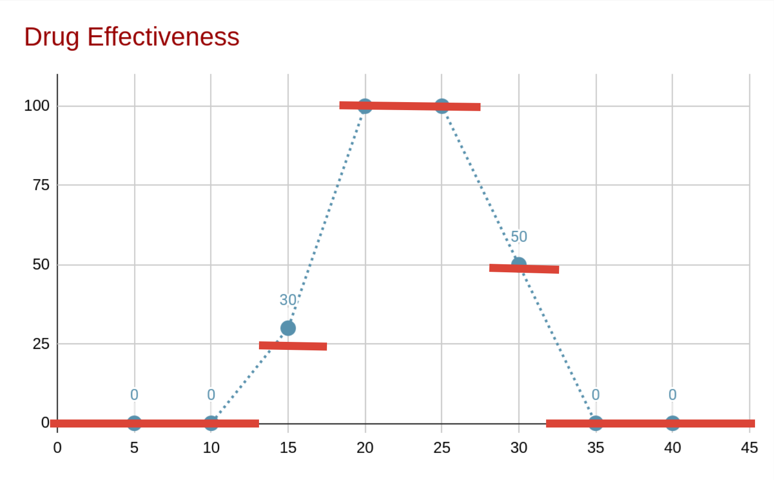


Regression Trees (Example)

- Let's see an example

Dosage (mg)	**Drug Effectiveness (%)**
5	0
10	0
15	30
20	100
25	100
30	50
35	0
40	0





The output is a piecewise constant functions!

Regression Trees

- You may see “Classification and Regression Trees” (CART) mentioned. CART refers to the general framework for growing decision trees for either classification or regression.

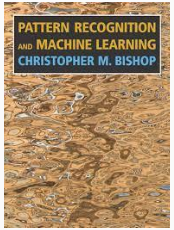
- We can apply **random forest** on top of regression trees as well!

- The only difference with classification is that we **average the predictions** performed by the trees instead of using the voting scheme adopted for classification.

Decision Trees vs Random Forest

	Decision Trees	Random Forest
Number of Trees		
Generalization		
Performance		
Computational Complexity		
Interpretability		
Sensitivity to feature Scale		

Additional Material



14.4 Tree-based models

StatQuest with Josh Starmer

https://youtu.be/efR1C6CvhmE

Lab Session

- During the weekly lab session, we will do:

colab Implementing Decision Trees (From Scratch)

Boosting

9th Lab Assignment This Week