CENTRALE NANTES

LS2N LABORATOIRE DES SCIENCES DU NUMÉRIQUE DE NANTES

CM4 - Ensemble methods

APSTA-LEARN
MACHINE LEARNING

Diana Mateus

Table of contents

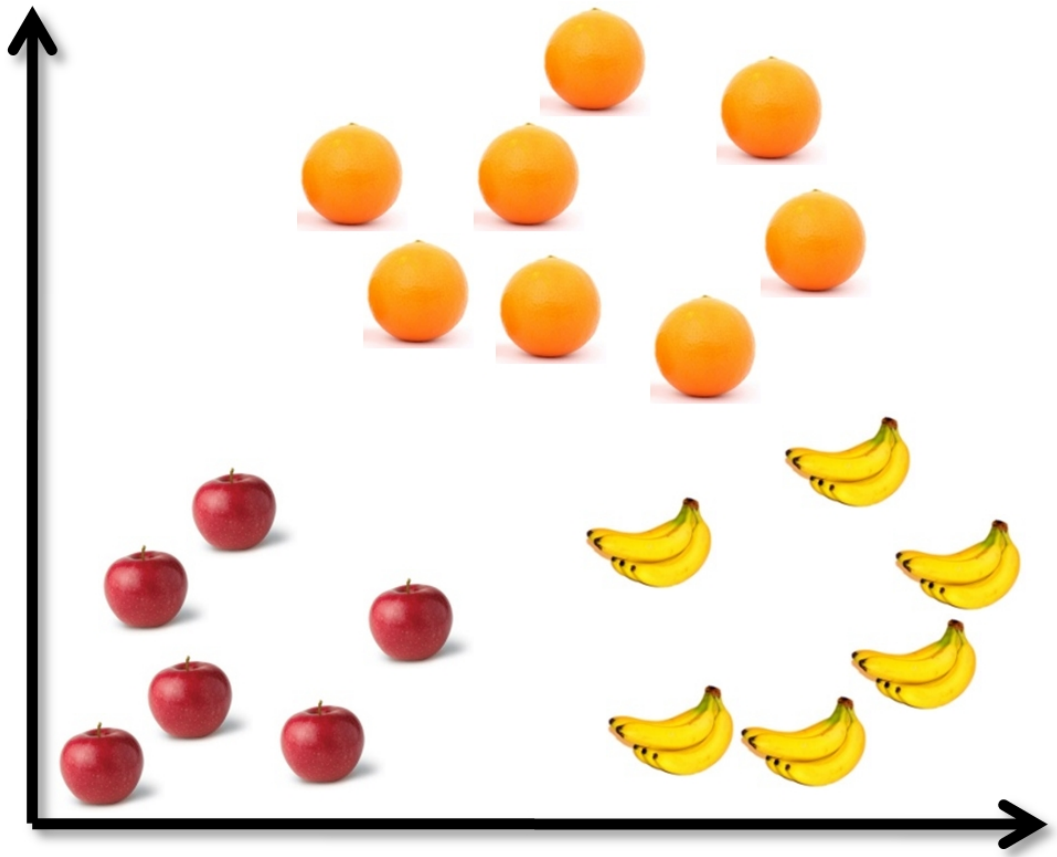
1. Motivation: multi-class classification

2. Ensemble Methods

Motivation: multi-class classification

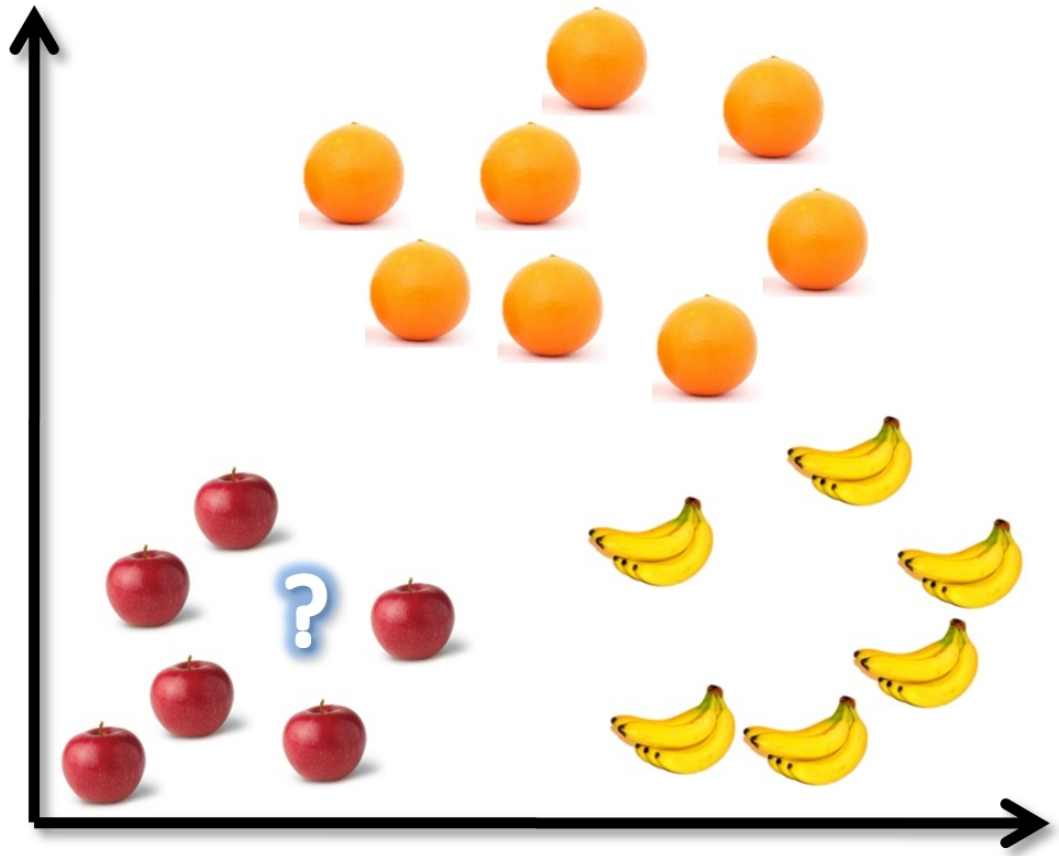
A simple example...

Let us consider 3 classes:  ,  



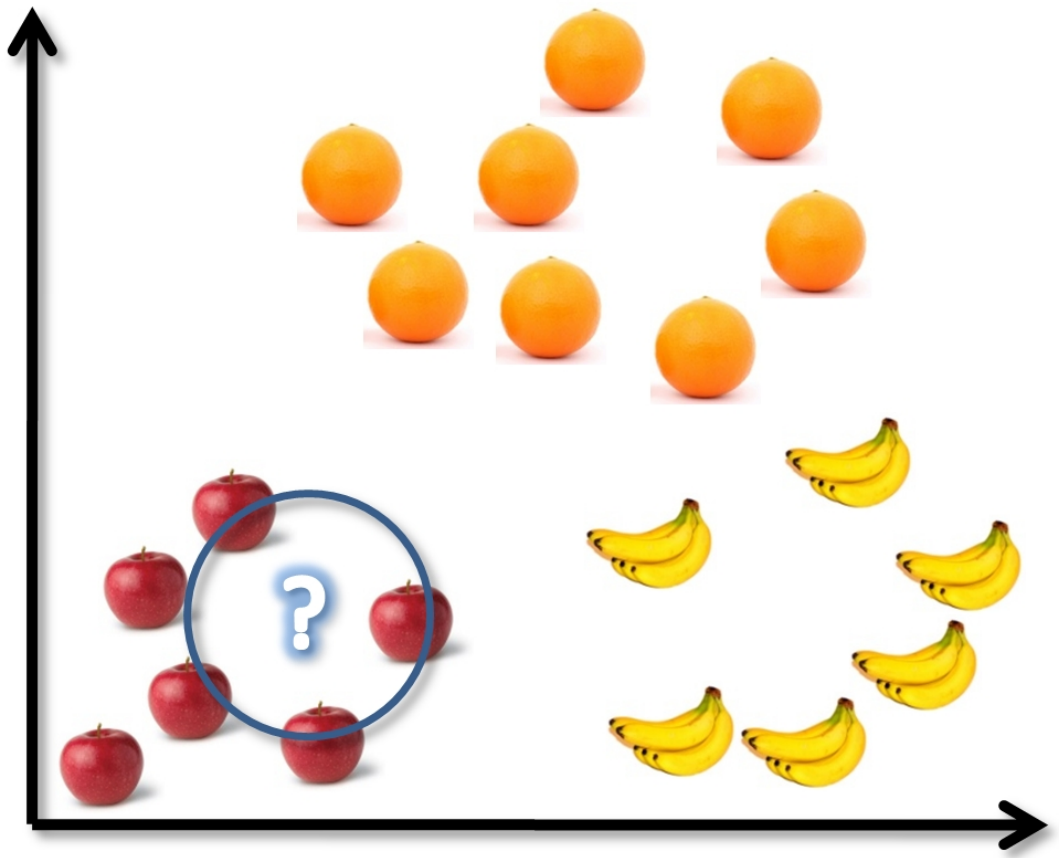
A simple example...

What is the label of this unknown point?



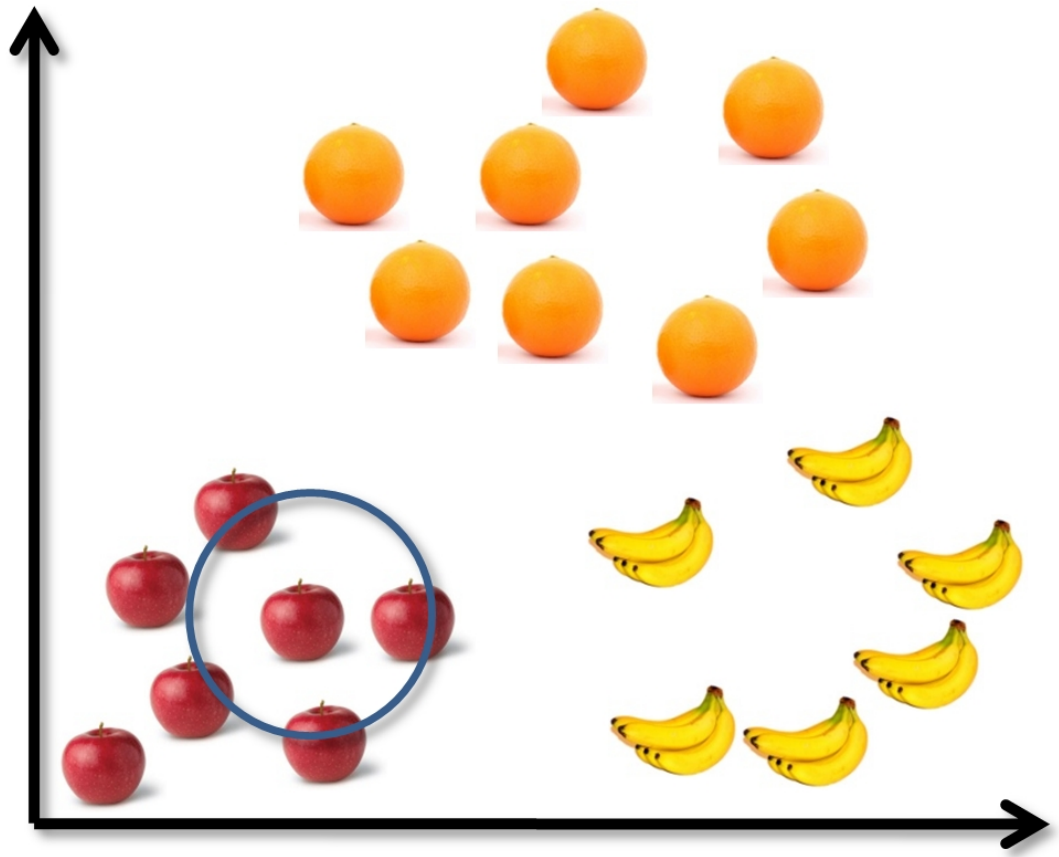
A simple example...

Intuitively, we look at its nearest neighbors...



A simple example...

And determine the right label...



A simple example...

K-nearest neighbors

Advantages

- Intuitive classifier
- Training consists only in storing training set
- Classification is performed by looking at the nearest neighbors
- Choice of distance and neighborhood K are important

Disadvantages Huge memory requirement and computationally expensive in case of large datasets!

Table of contents

1. Motivation: multi-class classification
   - K-Nearest Neighbors classifier
   - Decision Tree

2. Ensemble Methods
   - Bagging
   - Random Forests

Decision Tree

Motivation:

- To gain information on unknown observations.
- To discover patterns in data.

[Source: Wikipedia]

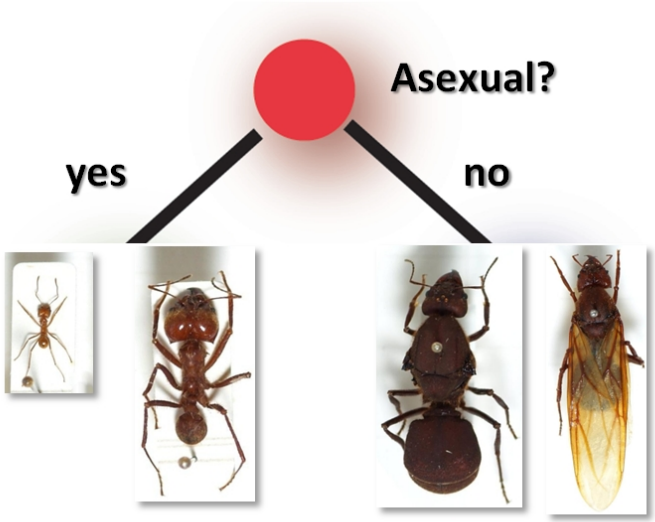
Decision Tree

Motivation:

- To gain information on unknown observations.
- To discover patterns in data.

How?

- Using simple rules ... enough?



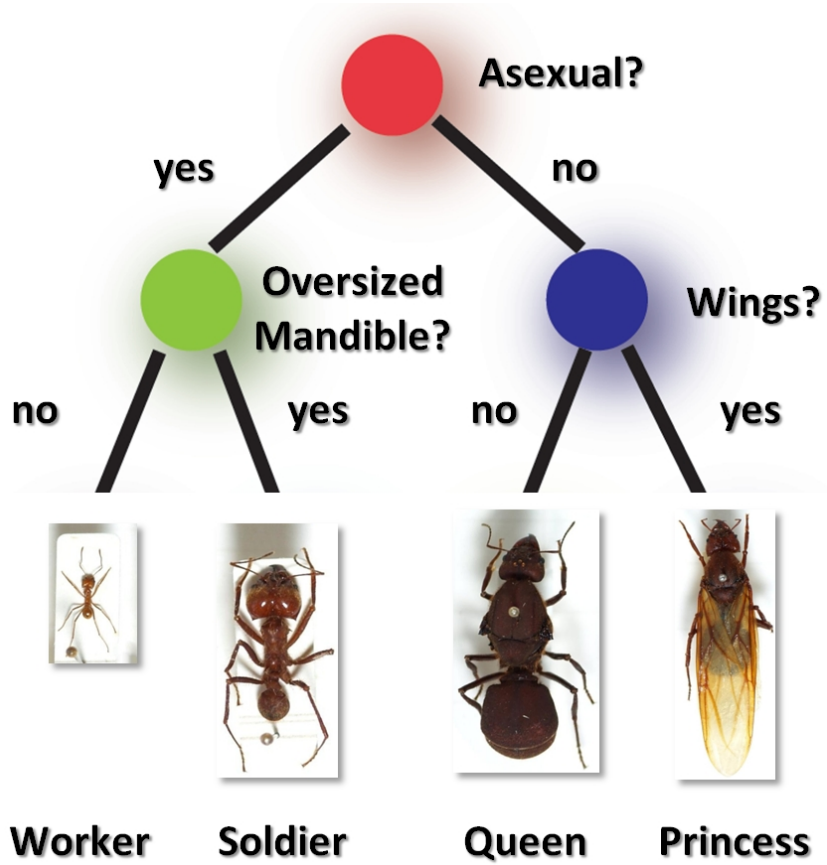
Decision Tree

Motivation:

- To gain information on unknown observations
- To discover patterns in data

How?

- Using simple rules...
- In a hierarchical fashion.



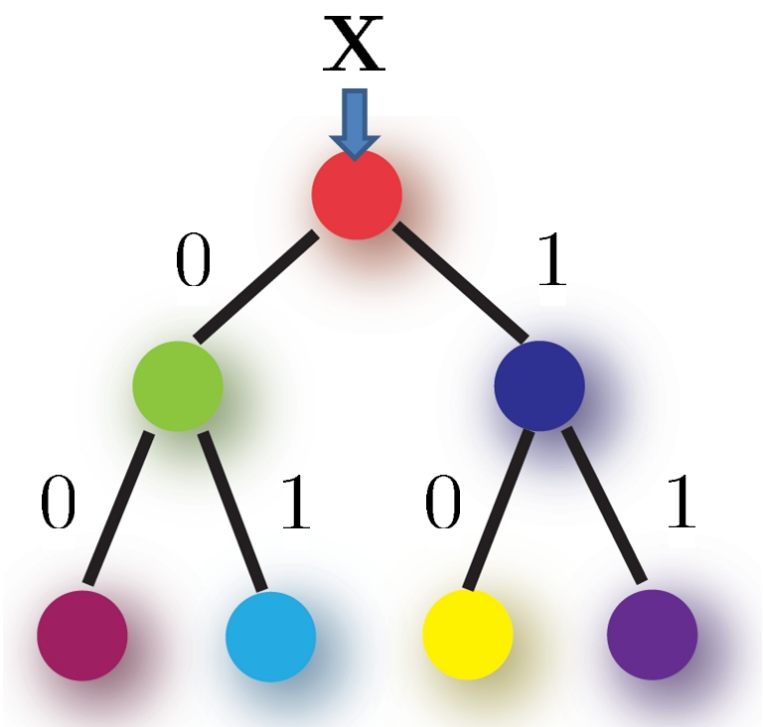
Decision Tree

Goal: Partition the feature space

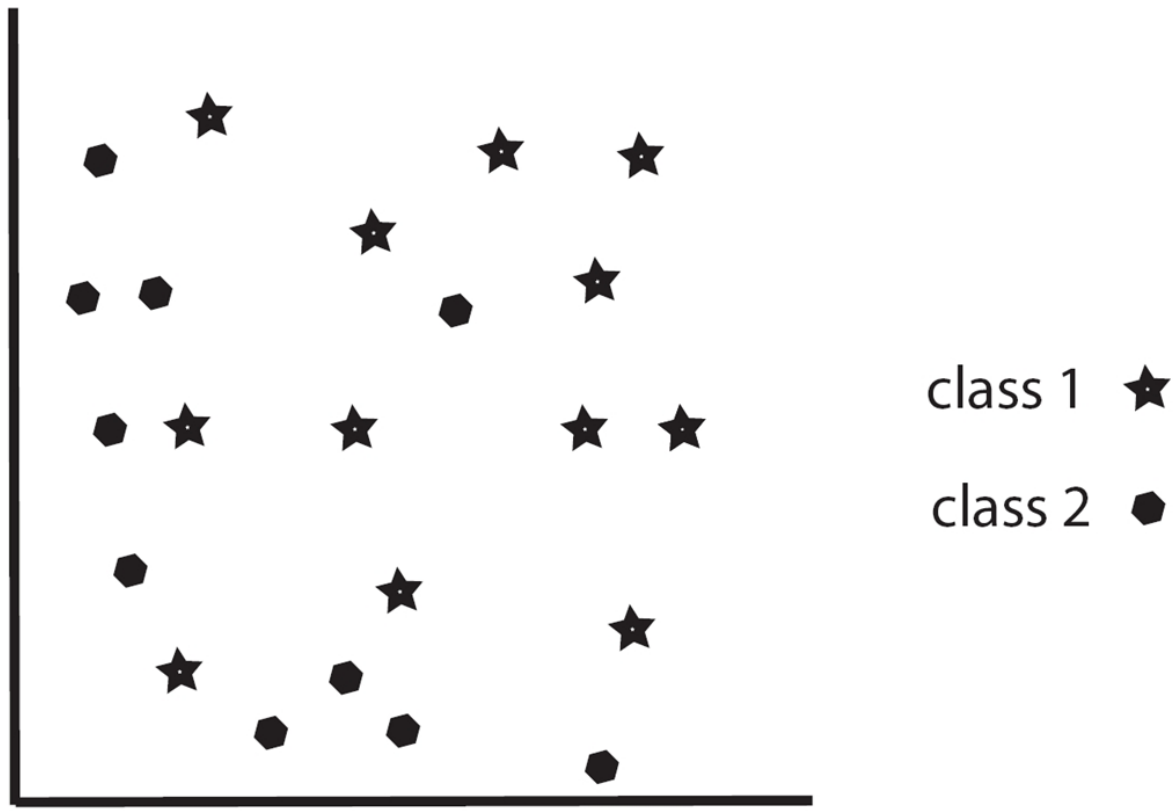
- Using simple rules...
- In a hierarchical fashion.

Tree:

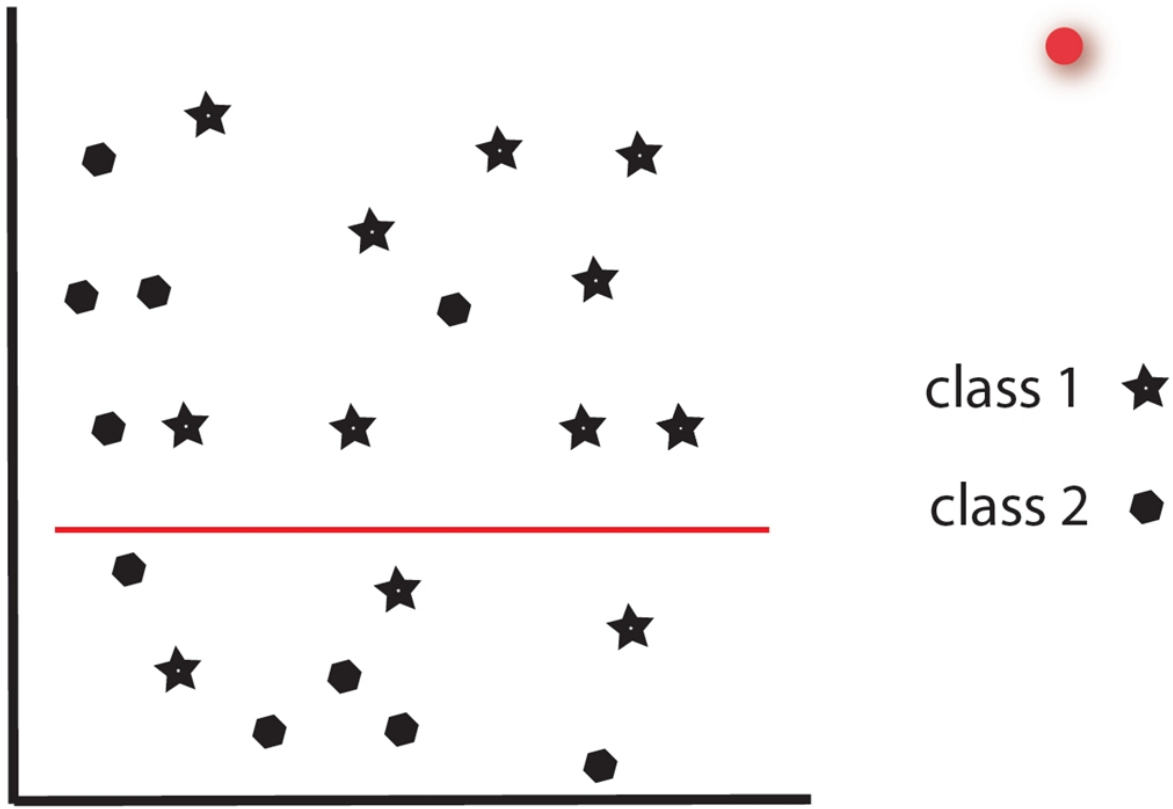
- Ensemble of nodes.
- Each node consists of a decision function and a threshold.



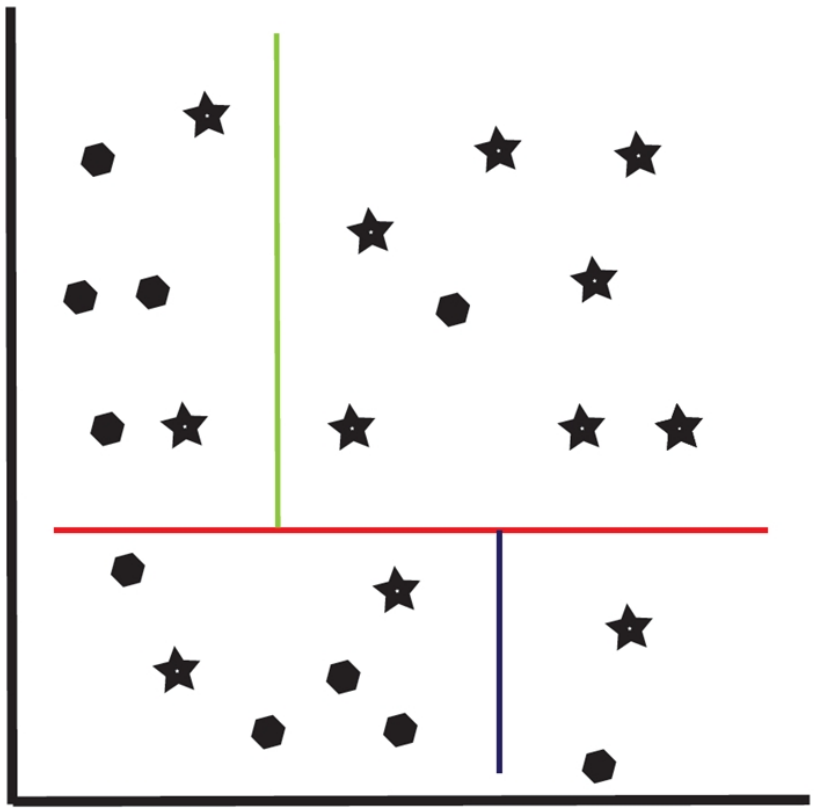
Decision Tree

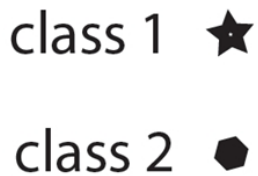


Decision Tree

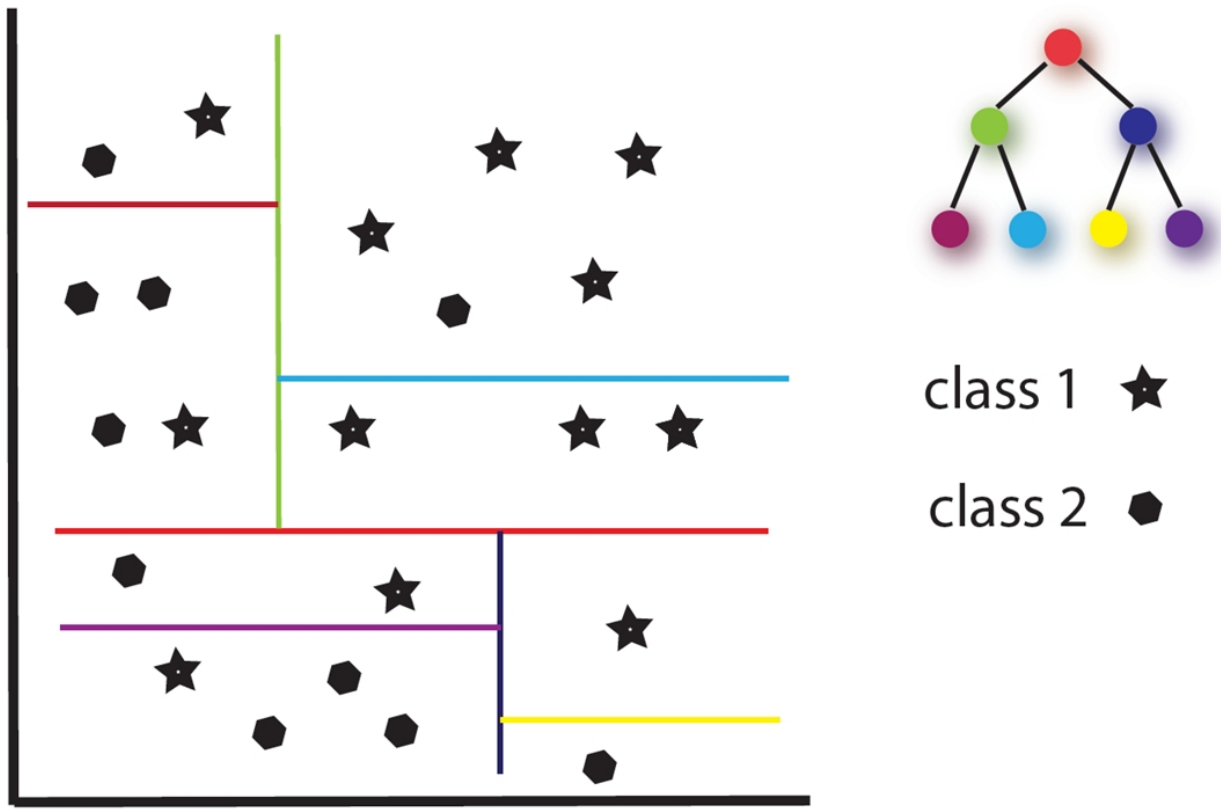


Decision Tree

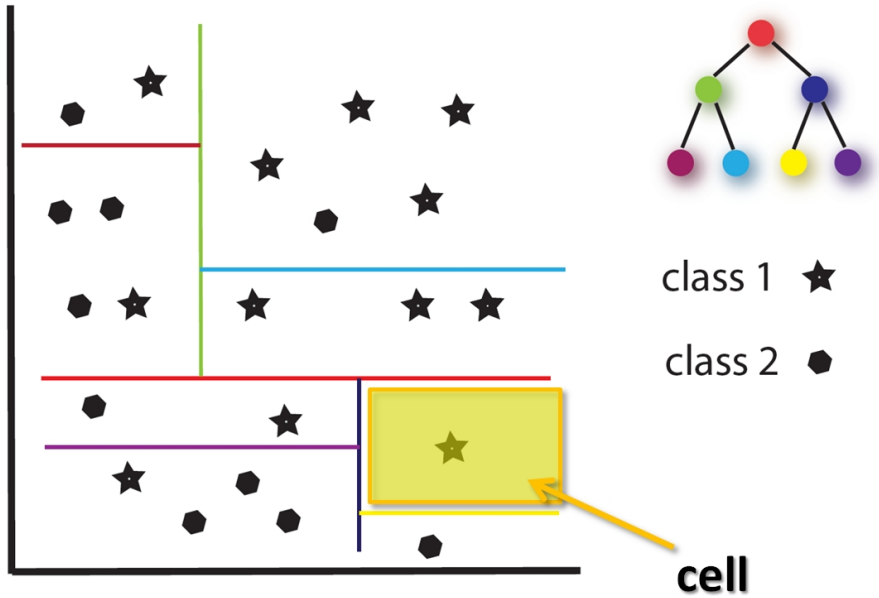




Decision Tree



Decision Tree



Tree

A tree $\mathcal{T}$ is an ensemble of Q nodes:

$$\mathcal{T} = \left\{ \mathcal{L}^{(q)} \right\}_{q=1}^{Q}$$

It induces a partition

$$\mathcal{P} = \{\mathcal{C}_z\}_{z=1}^{Z}$$

over the feature space, where $\mathcal{C}_z$ is a cell.

Decision Tree

Node

A node $\mathcal{L}^{(q)}$ consists of a decision function f_q associated to a threshold τ_q. Depending on the result of this decision function, a data point is pushed to the left or right child node.

Decision function

A decision function is (in most of the cases) a linear function f_q:

$$\begin{cases} f_q : \mathcal{X} \to \mathbb{R} \\ f_q(\mathbf{x}) = \mathbf{w}_q^\top \mathbf{x}, \text{ where } \mathbf{w}_q \in \mathbb{R}^{(D+1)} \end{cases}$$

and a threshold $\tau_q \in \mathbb{R}$:

$$\begin{cases} \text{if } f_q(\mathbf{x}) \geq \tau_n \Rightarrow \mathbf{x} \in \mathcal{L}^{(r)} \text{ right child} \\ \text{if } f_q(\mathbf{x}) < \tau_n \Rightarrow \mathbf{x} \in \mathcal{L}^{(l)} \text{ left child} \end{cases}$$

By default lines are axis aligned, that is, aligned with a single feature axis.

Decision Tree

Leaf

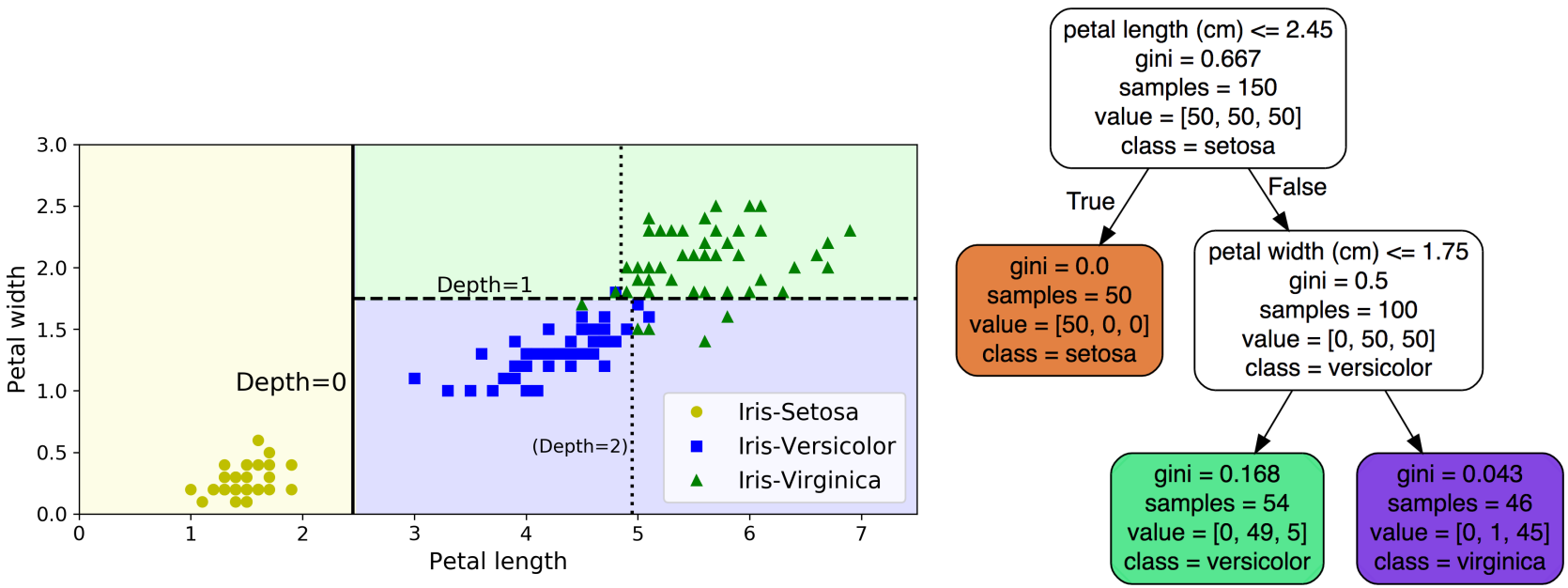
A leaf is a terminal node which corresponds to a cell $\mathcal{C}_z$ of the partition. In $\mathcal{C}_z$, the class posteriors can be evaluated:

$$p(k|\mathbf{x} \in \mathcal{C}_z) = \frac{|\mathcal{C}_z^k|}{|\mathcal{C}_z|}$$

where $|\mathcal{C}_z|$ denotes the amount of point in $\mathcal{C}_z$ and $|\mathcal{C}_z^k|$ denotes the amount of point in $\mathcal{C}_z$ belonging to class k

Decision Tree – IRIS dataset

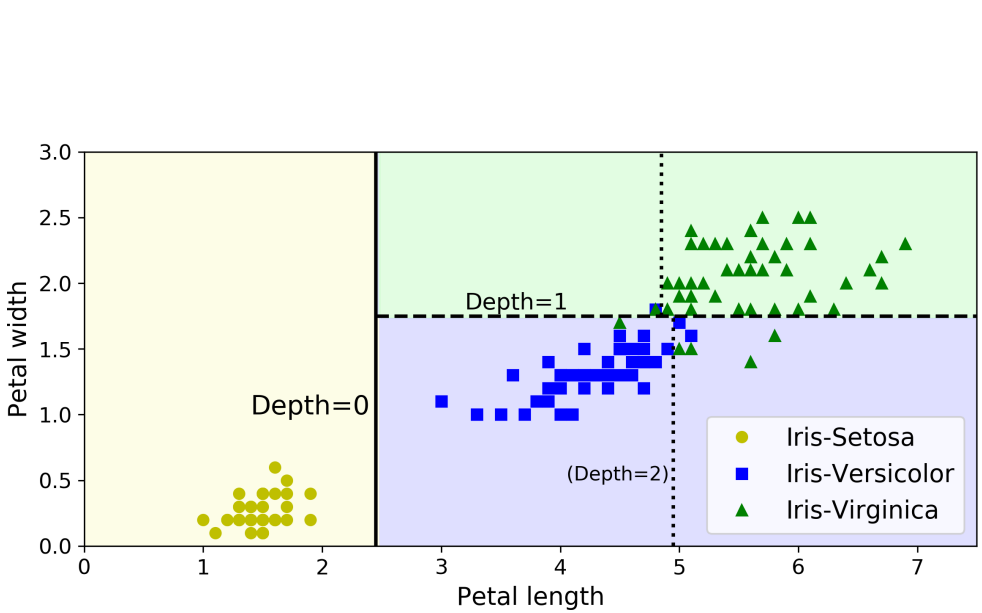
Decision tree notebook

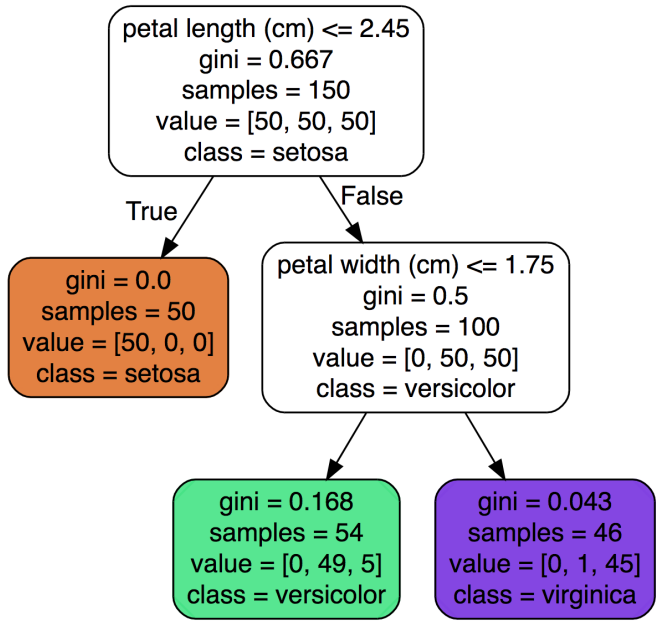


Source: Aurelien Geron. Hands-on ML with Scikit-Learn

Decision Tree – IRIS dataset

Decision tree notebook





Source: Aurelien Geron. Hands-on ML with Scikit-Learn

Train phase?, test phase?

Decision Tree – Randomized Tree Growing (CART[1])

Training key ideas:

1. **Randomized Node Optimization**:
- randomly generate several split *candidates*.
- the best splitting function is chosen according to some *quality measure*.

2. **Data splitting**:
- data points are pushed down the hierarchical tree, - at each node the points follow either the left or right child according to the split function.

4. **Leaf nodes**
- Are the terminal nodes.
- Contain the class distribution (histograms) of the points that reached that particular terminal node.

[1] Classification and Regression Tree

Decision Tree – Randomized Tree Growing (CART)[2]

procedure $\mathrm{CART}(\{\mathbf{x}_i, y_i\}_{i=1}^n)$ ▷ where $\mathbf{x}_i \in \mathbb{R}^D$
  j $\leftarrow 1$
  Initialize the tree with the root node $\mathcal{T}^{(0)} = \{L^{(0)}\}$
  while $\exists$ unsplit nodes in $\mathcal{T}^{(j)}$ **do**
    $L^{(j)} \leftarrow \mathrm{next\_node\_to\_split}(\mathcal{T}^{(j)})$ ▷ Get next node to split
    if stopping conditions met **then** ▷ Stop
      Save class distribution (histogram) of $y_i \in L^{(j)}$.
      $L^{(j)}$ becomes a leaf node (stop growing branch).
    for $q \leq \mathrm{trials\_per\_node}$ **do**
      $S_q = \{f_q, \tau_q\}$ ▷ Create a new random candidate split
      $\{L_r^{(j)}, L_l^{(j)}\} = \mathrm{split}(S_q, \{\mathbf{x}_i, y_i\}_{i=1}^n)$ ▷ split data
      if S_q best split so far **then** ▷ **randomized node opt.**
        $S_{\mathrm{best}} \leftarrow S_q$
    $\mathcal{T}^{(j+1)} = \{\mathcal{T}^{(j)}\} \cup \{L_r^{(j)}, L_l^{(j)}\}$ ▷ Add children to tree
    $j \leftarrow j + 1$
  return $\mathcal{T}^{(k+1)}$ ▷ Return the tree

A recursive version of the algorithm is commonly implemented.

[2] Classification and Regression Tree

Decision Tree – Randomized Tree Growing (CART)[2]

procedure $\mathrm{CART}(\{\mathbf{x}_i, y_i\}_{i=1}^n)$ ▷ where $\mathbf{x}_i \in \mathbb{R}^D$

  j $\leftarrow 1$

  Initialize the tree with the root node $\mathcal{T}^{(0)} = \{L^{(0)}\}$

  while $\exists$ unsplit nodes in $\mathcal{T}^{(j)}$ **do**

    $L^{(j)} \leftarrow \mathrm{next\_node\_to\_split}(\mathcal{T}^{(j)})$ ▷ Get next node to split

    if stopping conditions met **then** ▷ Stop

      Save class distribution (histogram) of $y_i \in L^{(j)}$.

      $L^{(j)}$ becomes a leaf node (stop growing branch).

    for $q \leq \mathrm{trials\_per\_node}$ **do**

      $S_q = \{f_q, \tau_q\}$ ▷ Create a new random candidate split

      $\{L_r^{(j)}, L_l^{(j)}\} = \mathrm{split}(S_q, \{\mathbf{x}_i, y_i\}_{i=1}^n)$ ▷ split data

      if S_q best split so far **then** ▷ **randomized node opt.**

        $S_{\mathrm{best}} \leftarrow S_q$

    $\mathcal{T}^{(j+1)} = \{\mathcal{T}^{(j)}\} \cup \{L_r^{(j)}, L_l^{(j)}\}$ ▷ Add children to tree

    $j \leftarrow j + 1$

  return $\mathcal{T}^{(k+1)}$ ▷ Return the tree

A recursive version of the algorithm is commonly implemented.

[2] Classification and Regression Tree

Decision Tree – Randomized Tree Growing (CART)[2]

procedure $\mathrm{CART}(\{\mathbf{x}_i, y_i\}_{i=1}^n)$ ▷ where $\mathbf{x}_i \in \mathbb{R}^D$
  j $\leftarrow 1$
  Initialize the tree with the root node $\mathcal{T}^{(0)} = \{L^{(0)}\}$
  while $\exists$ unsplit nodes in $\mathcal{T}^{(j)}$ **do**
    $L^{(j)} \leftarrow \mathrm{next\_node\_to\_split}(\mathcal{T}^{(j)})$ ▷ Get next node to split
    if stopping conditions met **then** ▷ Stop
      Save class distribution (histogram) of $y_i \in L^{(j)}$.
      $L^{(j)}$ becomes a leaf node (stop growing branch).
    for $q \leq \mathrm{trials\_per\_node}$ **do**
      $S_q = \{f_q, \tau_q\}$ ▷ Create a new random candidate split
      $\{L_r^{(j)}, L_l^{(j)}\} = \mathrm{split}(S_q, \{\mathbf{x}_i, y_i\}_{i=1}^n)$ ▷ split data
      if S_q best split so far **then** ▷ **randomized node opt.**
        $S_{\mathrm{best}} \leftarrow S_q$
    $\mathcal{T}^{(j+1)} = \{\mathcal{T}^{(j)}\} \cup \{L_r^{(j)}, L_l^{(j)}\}$ ▷ Add children to tree
    $j \leftarrow j + 1$
  return $\mathcal{T}^{(k+1)}$ ▷ Return the tree

A recursive version of the algorithm is commonly implemented.

[2] Classification and Regression Tree

Decision Tree – Randomized Tree Growing (CART)[2]

procedure $\mathrm{CART}(\{\mathbf{x}_i, y_i\}_{i=1}^n)$ ▷ where $\mathbf{x}_i \in \mathbb{R}^D$
  j $\leftarrow 1$
  Initialize the tree with the root node $\mathcal{T}^{(0)} = \{L^{(0)}\}$
  while $\exists$ unsplit nodes in $\mathcal{T}^{(j)}$ **do**
    $L^{(j)} \leftarrow \mathrm{next\_node\_to\_split}(\mathcal{T}^{(j)})$ ▷ Get next node to split
    if stopping conditions met **then** ▷ Stop
      Save class distribution (histogram) of $y_i \in L^{(j)}$.
      $L^{(j)}$ becomes a leaf node (stop growing branch).
    for $q \leq \mathrm{trials\_per\_node}$ **do**
      $S_q = \{f_q, \tau_q\}$ ▷ Create a new random candidate split
      $\{L_r^{(j)}, L_l^{(j)}\} = \mathrm{split}(S_q, \{\mathbf{x}_i, y_i\}_{i=1}^n)$ ▷ split data
      if S_q best split so far **then** ▷ **randomized node opt.**
        $S_{\mathrm{best}} \leftarrow S_q$
    $\mathcal{T}^{(j+1)} = \{\mathcal{T}^{(j)}\} \cup \{L_r^{(j)}, L_l^{(j)}\}$ ▷ Add children to tree
    $j \leftarrow j + 1$
  return $\mathcal{T}^{(k+1)}$ ▷ Return the tree

A recursive version of the algorithm is commonly implemented.

[2] Classification and Regression Tree

Decision Tree – Randomized Tree Growing (CART)

Stopping criteria:

- Max depth.
- Force a minimum number of data points in the current node/leaf.
- Quality measure.

Decision Tree – Quality measures

Choose the best split candidates according to a quality measure: gini index, information gain, ...

Out-of-bag error (OOB)

Minimize error rate after splitting using a test set

$$E_{\text{OOB}} = \frac{1}{n}\sum_{i=1}^{n}(\hat{\mathbf{y}}_i \neq \mathbf{y}_i)$$

Decision Tree – Node Quality measures

Choose the best split candidates according to a quality measure: gini index, information gain, ...

Gini (Impurity) Index

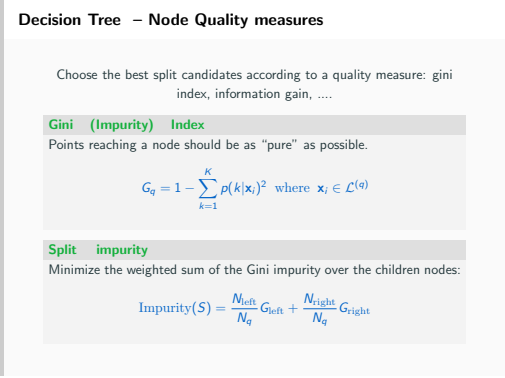
Points reaching a node should be as "pure" as possible.

$$G_q = 1 - \sum_{k=1}^{K} p(k|\mathbf{x}_i)^2 \text{ where } \mathbf{x}_i \in \mathcal{L}^{(q)}$$

Split impurity

Minimize the weighted sum of the Gini impurity over the children nodes:

$$\text{Impurity}(S) = \frac{N_{\text{left}}}{N_q} G_{\text{left}} + \frac{N_{\text{right}}}{N_q} G_{\text{right}}$$



Gini Index[Wikipedia]

Used by the CART (classification and regression tree) algorithm for classification trees, Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labelled if it was randomly labelled according to the distribution of labels in the subset. The Gini impurity can be computed by summing the probability $p(k|\mathbf{x}_i)$ of an item with label k (denoted for simplicity p_k) being chosen, times the probability $\sum_{l \neq k}^{K} p(l|\mathbf{x}_i) = 1 - p(k|\mathbf{x}_i)$ of making a mistake. It reaches its minimum (zero) when all cases in the node fall into a single target category.

$$G_q = \sum_{k=1}^{K} p_k \sum_{l \neq k} p_l = \sum_{k=1}^{K} p_k(1 - p_k) = \sum_{k=1}^{K} (p_k - {p_k}^2) = \sum_{k=1}^{K} p_k - \sum_{k=1}^{K} {p_k}^2 = 1 - \sum_{k=1}^{K} {p_k}^2$$

Decision Tree – Node Quality measures

Choose the best split candidates according to a quality measure: gini index, information gain, ...

Entropy

Instead of the Gini index use the Entropy

$$H_q = -\sum_{k=1}^{K} p(k|\{\mathbf{x}_i\}) \log\left(p(k|\{\mathbf{x}_i\})\right) \quad \forall \mathbf{x}_i \in \mathcal{L}^{(q)}$$

Entropy is zero when all elements belong to the same class

Entropy is 1 when all the classes are equally likely

Information gain

Maximize the information gain after splitting.

$$\text{InformationGain}(S) = H_q - \frac{N_{\text{left}}}{N_q} H_{\text{left}} - \frac{N_{\text{right}}}{N_q} H_{\text{right}}$$

Does the entropy grow when the split is applied?

Try your own implementation of node optimization

Decision Tree – Prediction

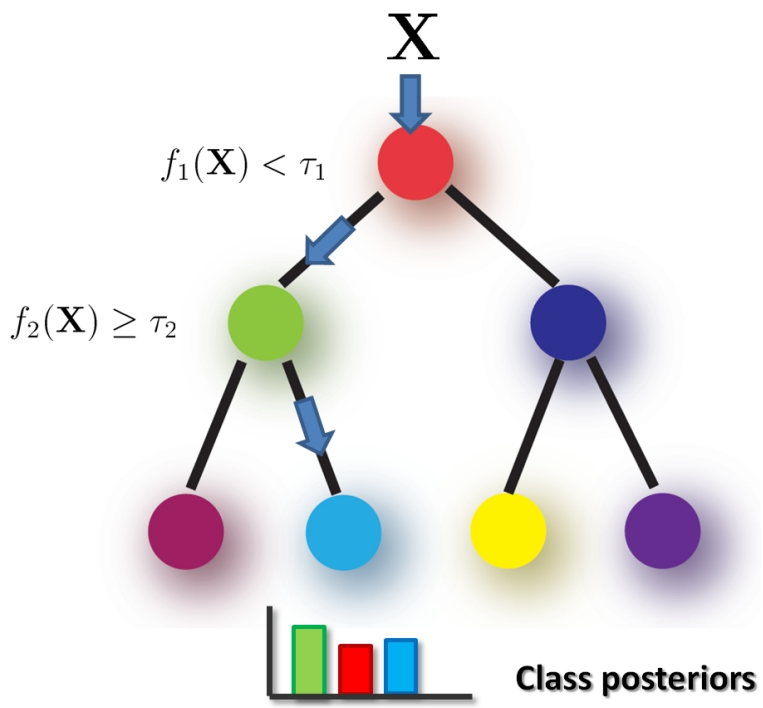
Unknown **test point x** pushed through the tree until it reaches a leaf.

Class posteriors

Stored in this leaf are used for prediction:

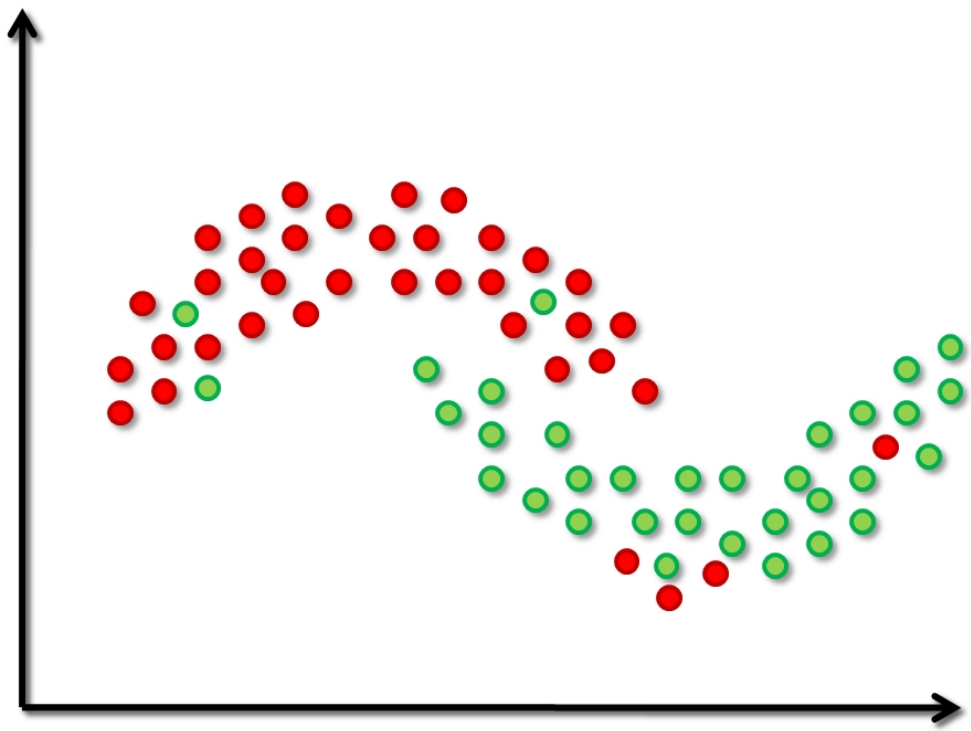
Prediction

$$\hat{\mathbf{y}} = \underset{k \in \mathcal{K}}{\operatorname{argmax}} \left(p(k|\mathbf{x}) \right)$$



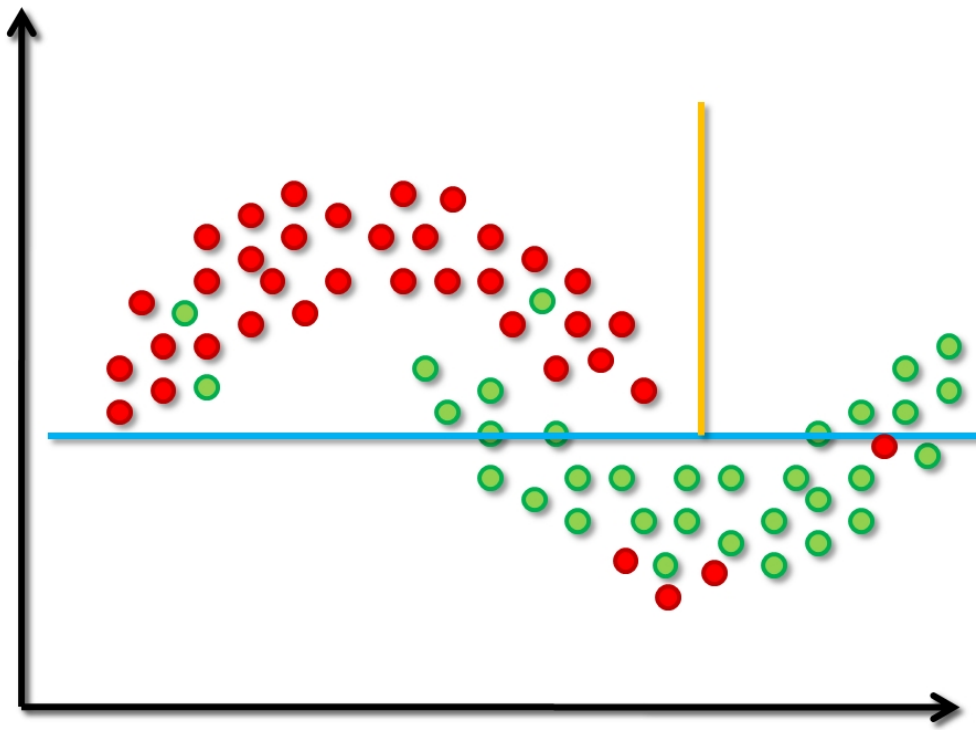
Decision Tree – Influence of tree depth

- Tree depth must be chosen carefully
- Small trees may underfit the data!
- Deep trees may overfit the data!



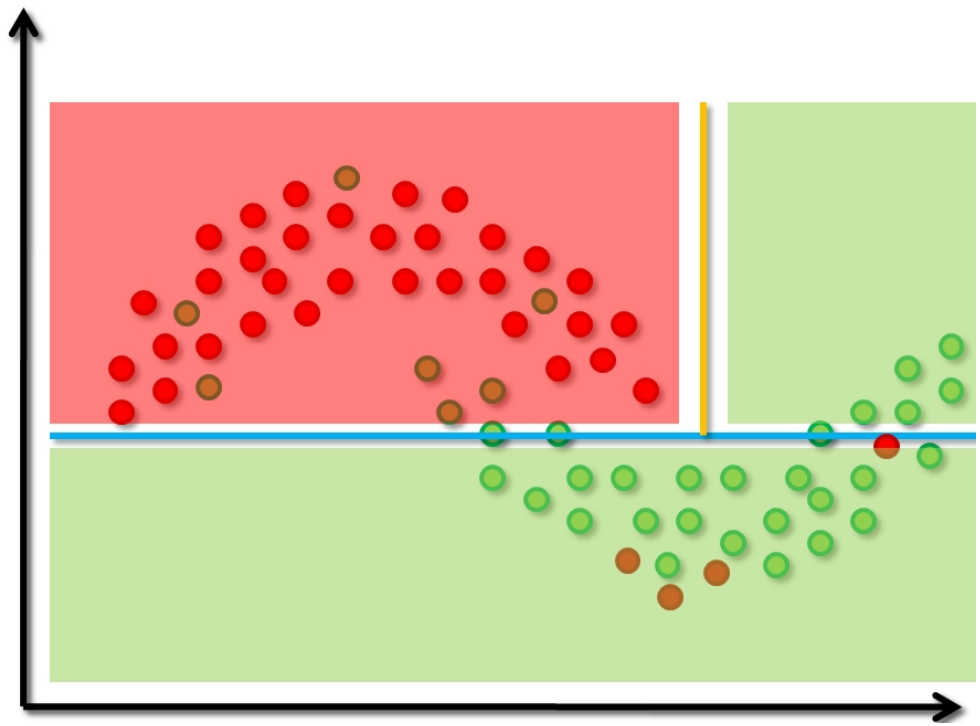
Decision Tree – Influence of tree depth

- Tree depth must be chosen carefully
- Small trees may underfit the data!
- Deep trees may overfit the data!



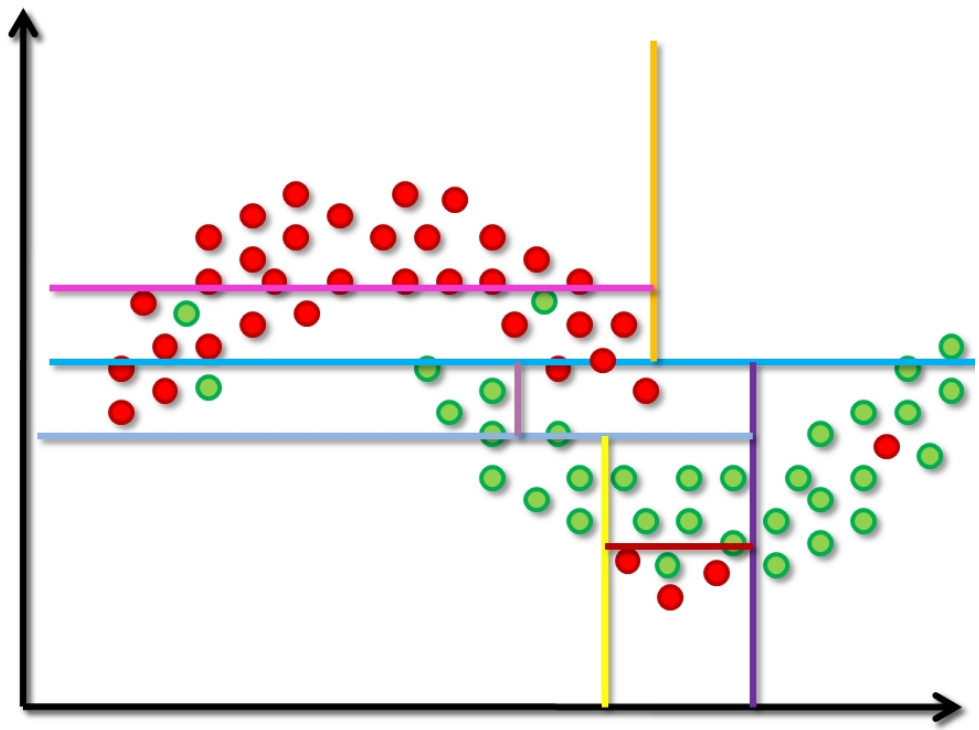
Decision Tree – Influence of tree depth

- Tree depth must be chosen carefully
- Small trees may underfit the data!
- Deep trees may overfit the data!



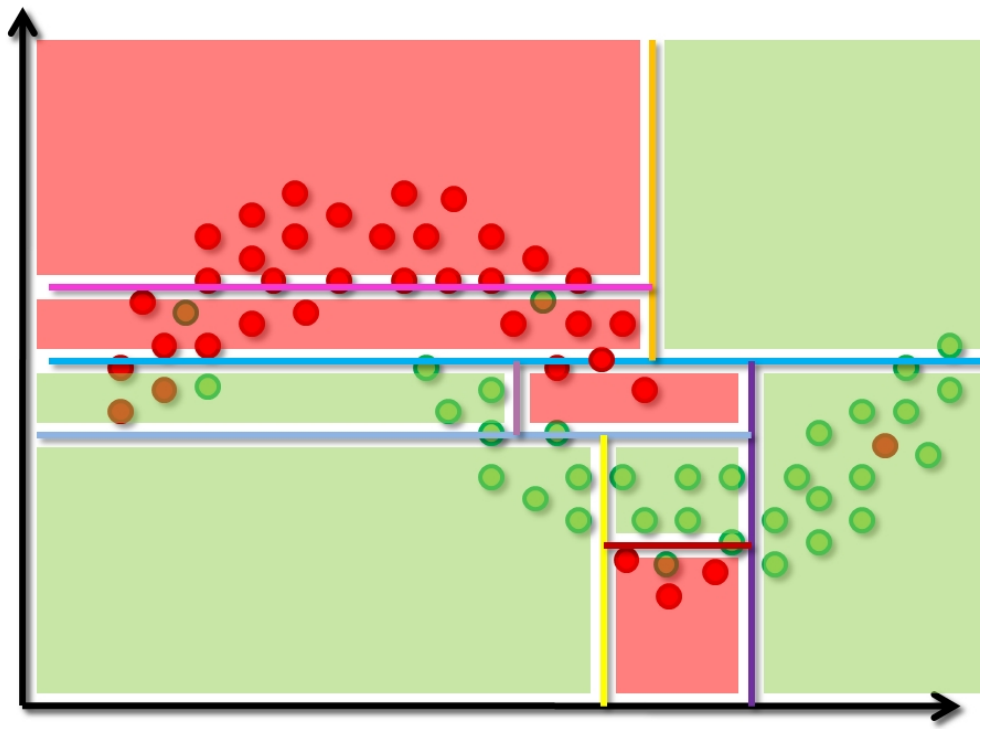
Decision Tree – Influence of tree depth

- Tree depth must be chosen carefully
- Small trees may underfit the data!
- Deep trees may overfit the data!



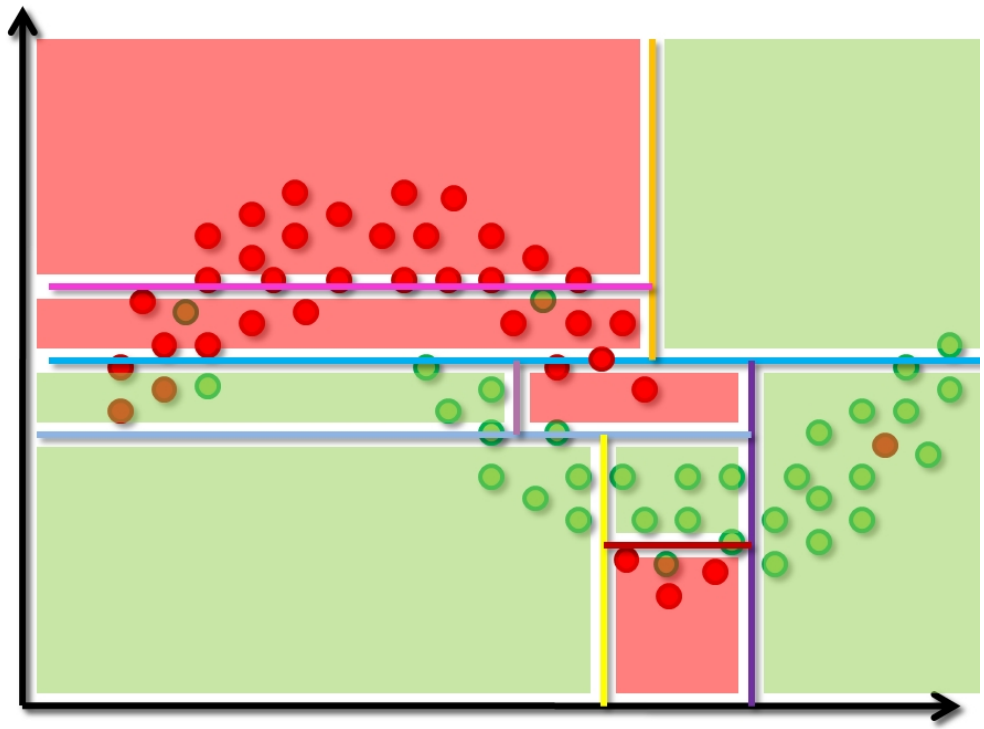
Decision Tree – Influence of tree depth

- Tree depth must be chosen carefully
- Small trees may underfit the data!
- Deep trees may overfit the data!



Decision Tree – Influence of tree depth

- Tree depth must be chosen carefully
- Small trees may underfit the data!
- Deep trees may overfit the data!



Two moons notebook

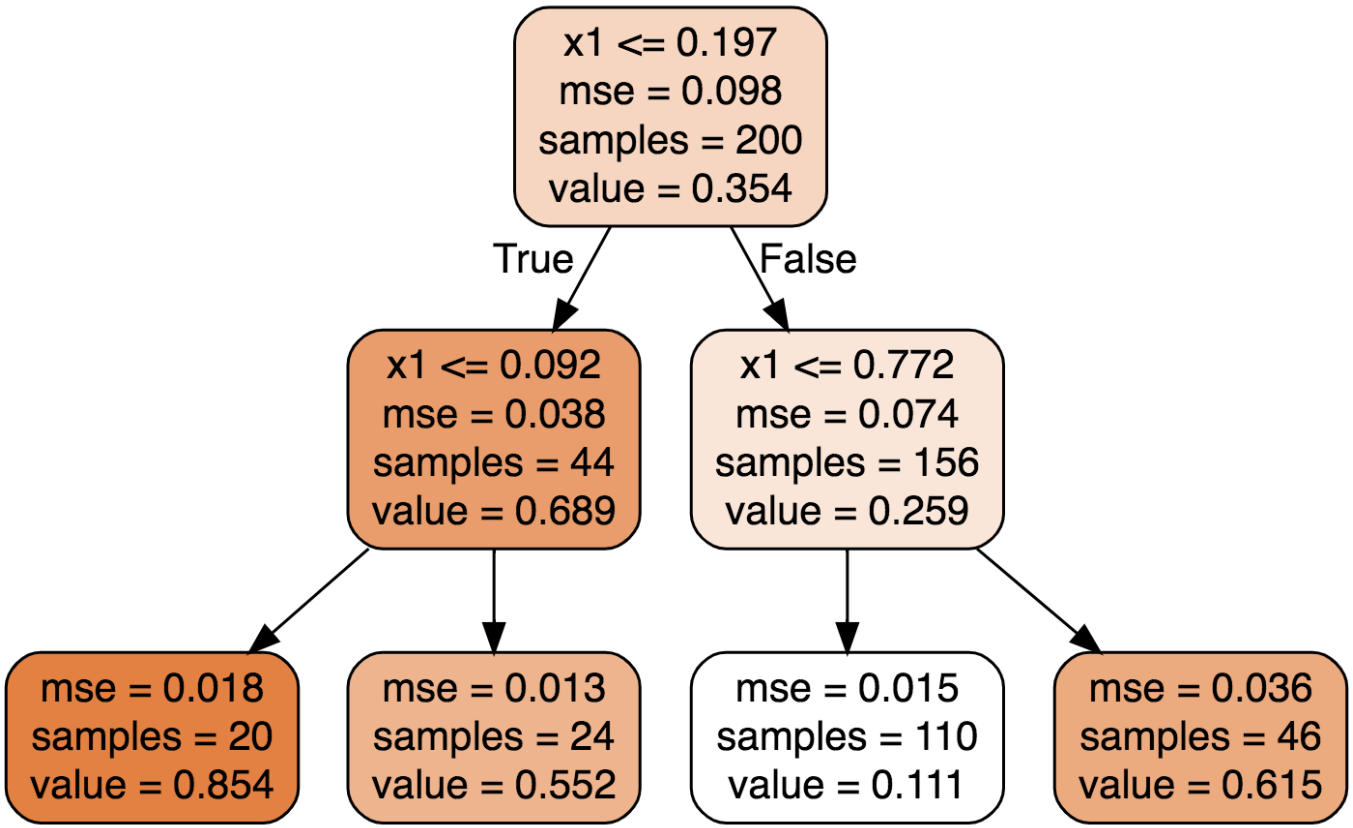
Decision Tree – Regression

How can we use decision trees for regression?

Source: Aurelien Geron. Hands-on ML with Scikit-Learn

Decision Tree – Regression

Instead of predicting the class in each node, predict a value

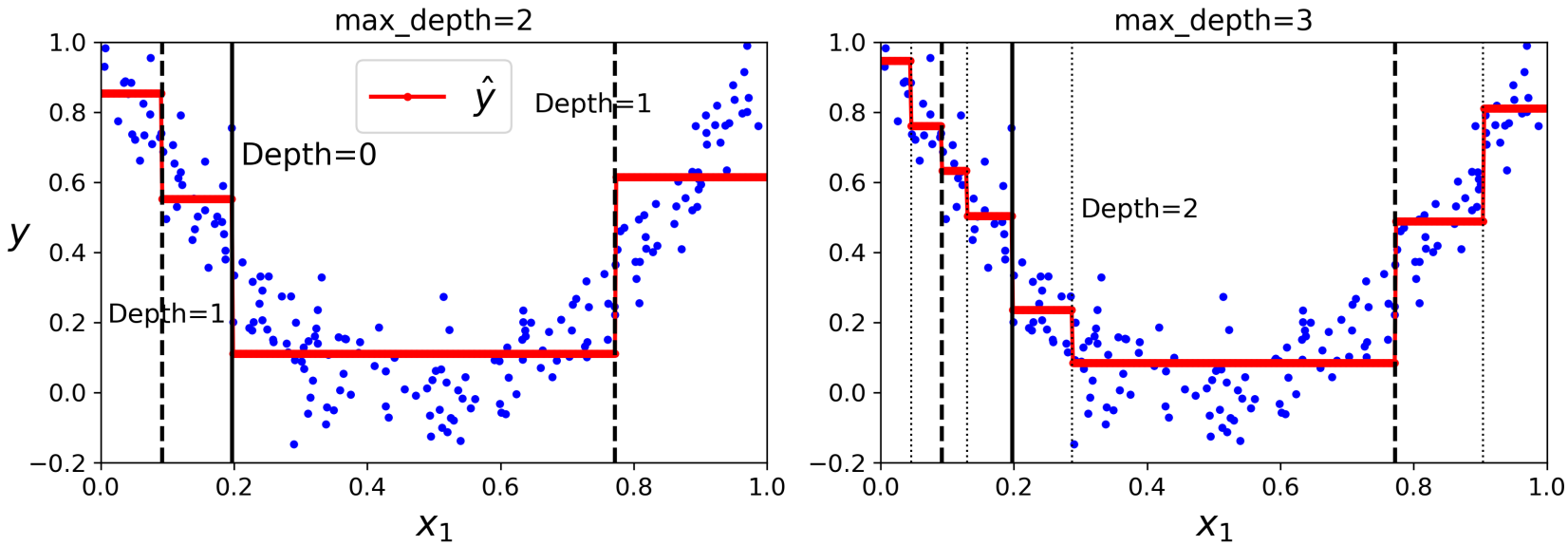


Source: Aurelien Geron. Hands-on ML with Scikit-Learn

What info to store in the leaves?

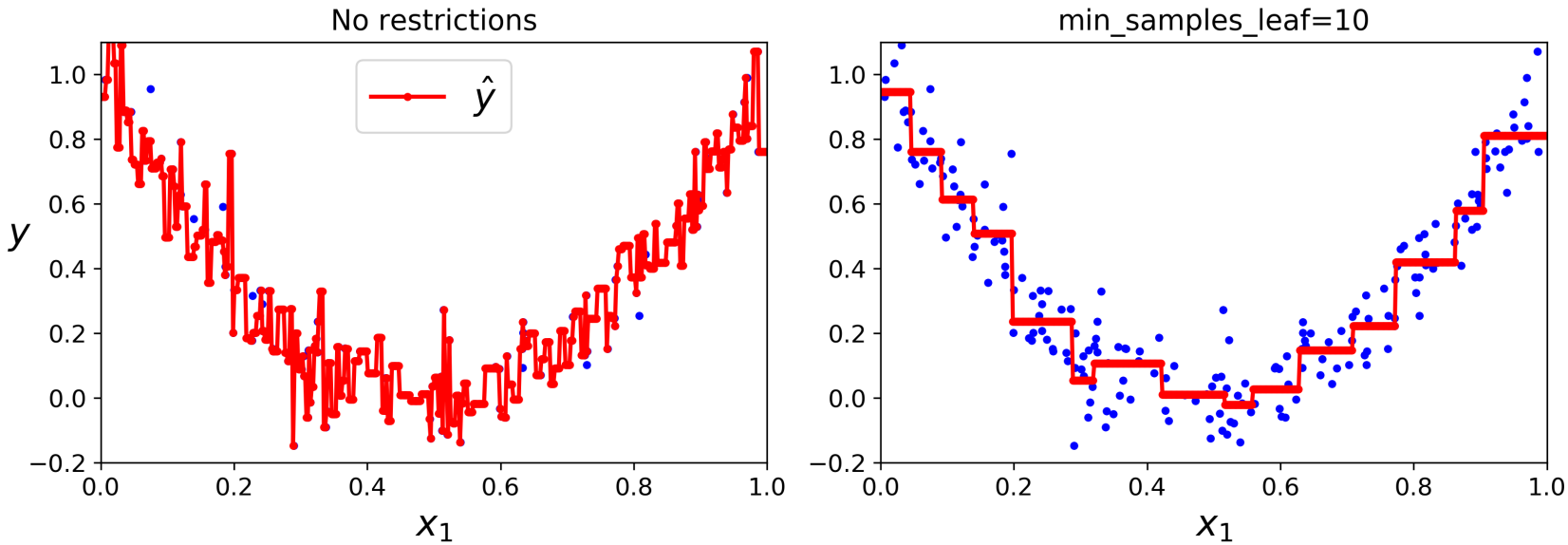
What is the best split?

Decision Tree – Regression

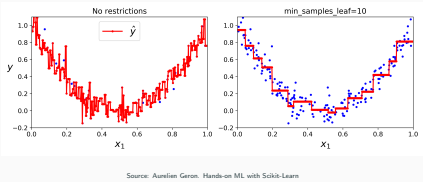


Source: Aurelien Geron. Hands-on ML with Scikit-Learn

Decision Tree – Regression



Source: Aurelien Geron. Hands-on ML with Scikit-Learn



Source: Aurelien Geron. Hands-on ML with Scikit-Learn

The predicted value for a region/cell is the average target value of the instances in that region.

$$\hat{y} = \frac{1}{N_{\text{node}}} \sum_{i \in \text{node}} y_i$$

During training split regions such that most training instances are as close as possible to the predicted value. Minimize the joint MSE

$$\text{regressionSplitQuality}(S) = \frac{N_{\text{left}}}{N_q} \text{MSE}_{\text{left}} + \frac{N_{\text{right}}}{N_q} \text{MSE}_{\text{right}}$$

$$MSE_{\text{node}} = \sum_{i \in \text{node}} (\hat{y} - y_i)$$

Decision Tree – Conclusion

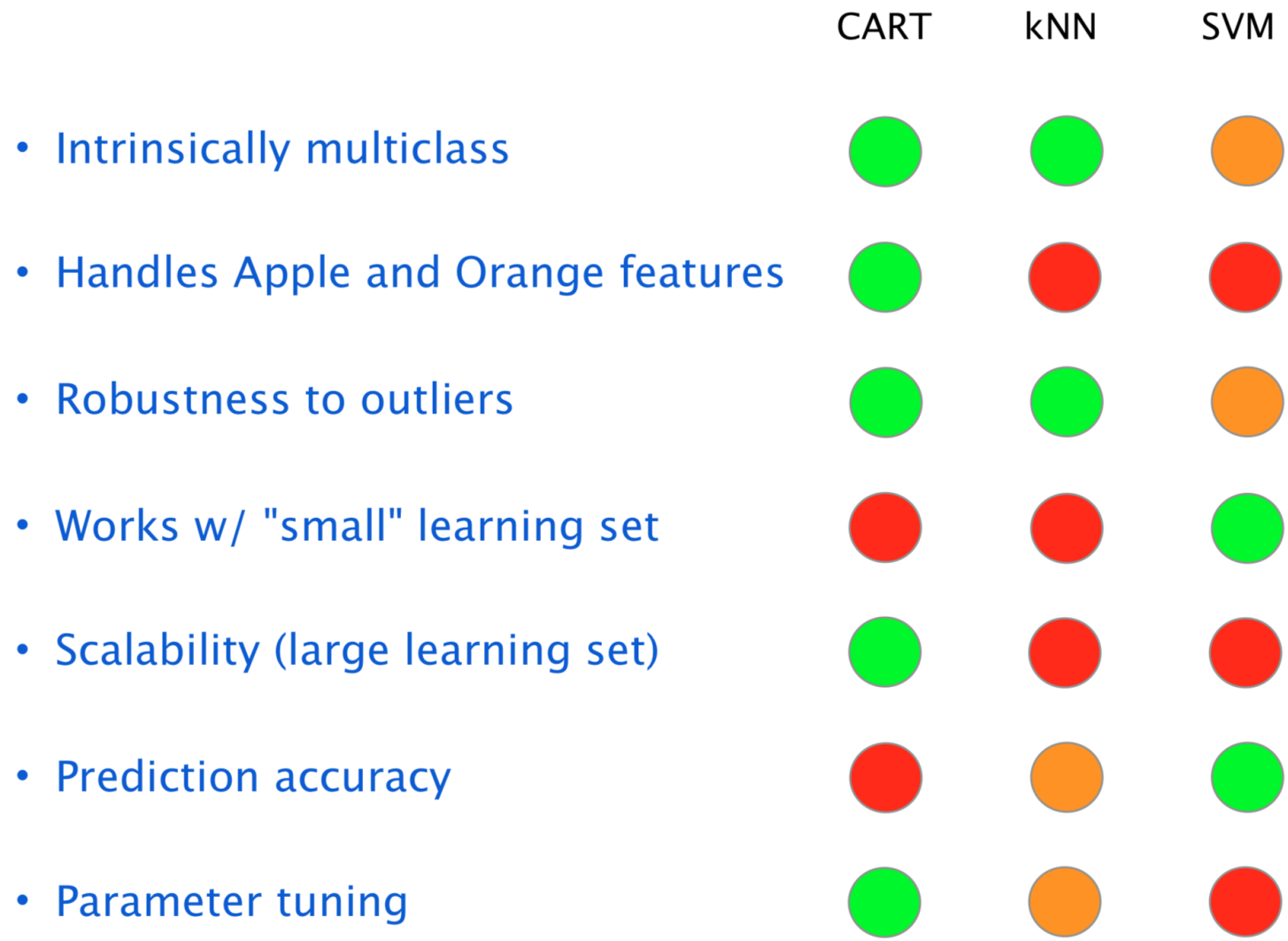
Advantages

- Induces a **partition** of the feature space.
- Real **multi-class** solution. Also useful for **regression**
- **Fast** training and evaluation
- Memory friendly.

Disadvantages

- **Variance** in the performance! Be careful! deep trees may overfit the data!
- Choice of tree depth and number of candidates per node is important .

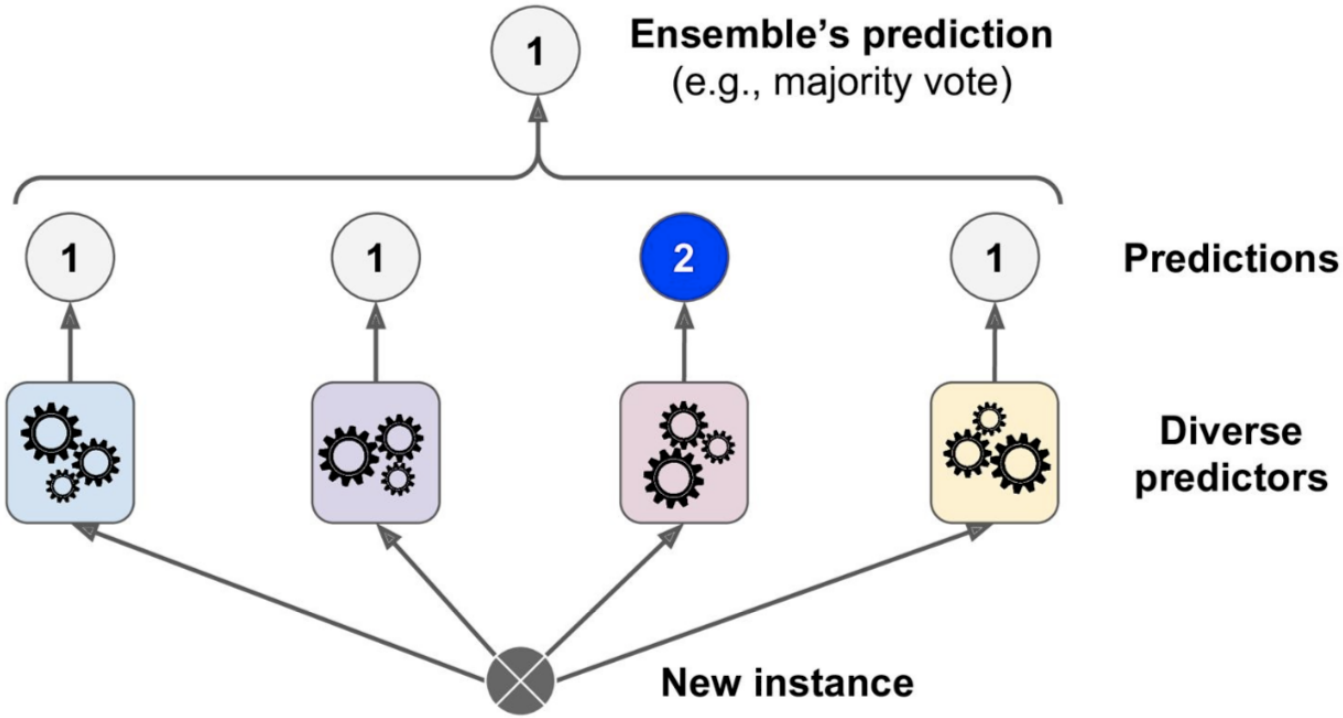
Comparison



Source: Eric Debreuve

Ensemble Methods

Ensemble Methods– Motivation



Source: Aurelien Geron. Hands-on ML with Scikit-Learn

- An **ensemble** of simple or **weak learners** instead of one single complex learner.
- Different strategies for **generating and combining** the weak models

Weak learners: decision stumps, random trees, SVM, ...

Ensemble Methods – Motivation

Ensemble methods are meta-algorithms that combine several machine learning techniques into one predictive model in order to

- reduce variance (preventing overfitting)
- Average out (reduce) bias
- Improve prediction

Ensemble Methods – Motivation

Ensemble methods are meta-algorithms that combine several machine learning techniques into one predictive model in order to

- reduce variance (preventing overfitting) $\Rightarrow$ **Bagging**
- Average out (reduce) bias $\Rightarrow$ **Boosting**
- Improve prediction $\Rightarrow$ **Stacking**

Ensemble learning algorithms

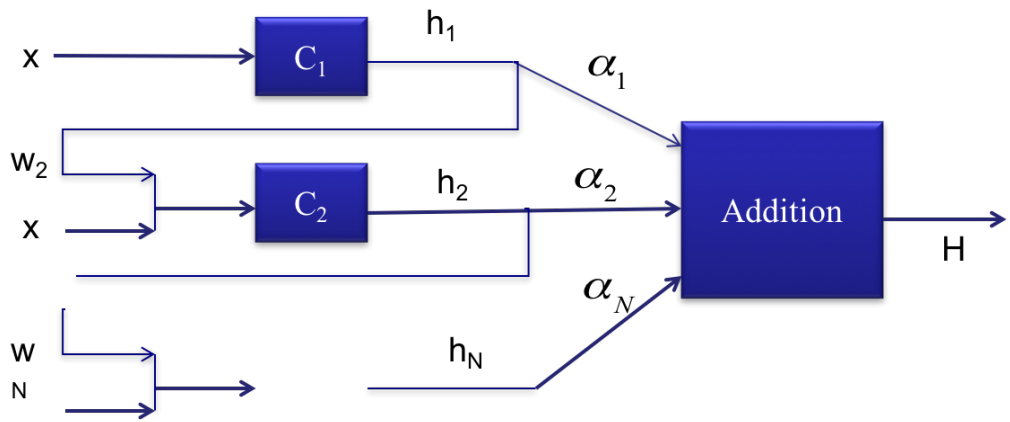
Bagging (Bootstrap aggregating)

- **Bootstraps** of the training set are created by random sampling with replacement.
- Weak classifiers are trained using the different bootstraps.
- All weak classifiers are combined into a strong classifier.
- Predictions are **aggregated**.

Ensemble learning algorithms

AdaBoost (Adaptive boosting)

- Based on an additive model.
- Iteratively modify the training dataset to focus misclassified samples.
- Assign a weight to each classifier according to its performance.



AdaBoost (Adaptive boosting)

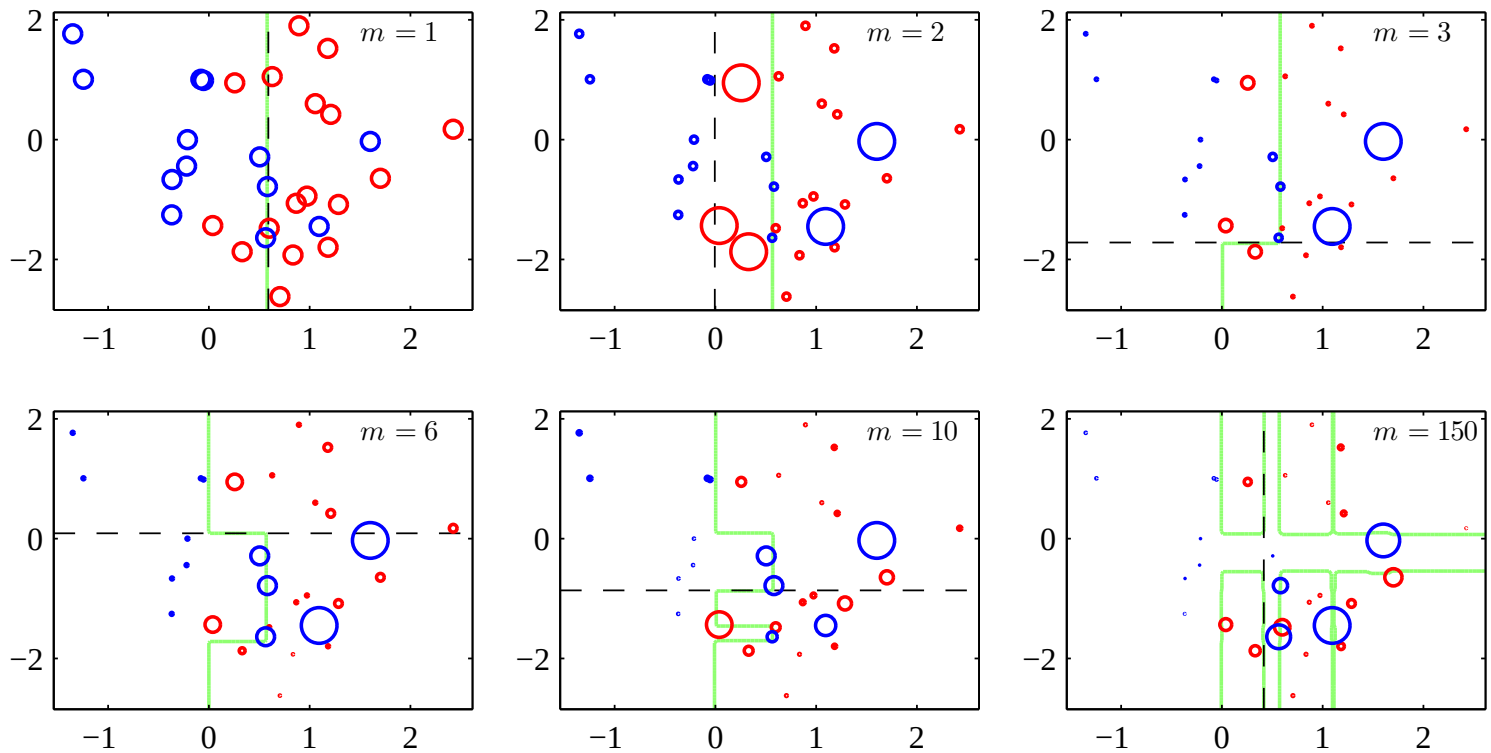


Illustration of boosting in which the base learners consist of simple thresholds applied to one or other of the axes. Each figure shows the number m of base learners trained so far, along with the decision boundary of the most recent base learner (dashed black line) and the combined decision boundary of the ensemble (solid green line). Each data point is depicted by a circle whose radius indicates the weight assigned to that data point when training the most recently added base learner. Thus, for instance, we see that points that are misclassified by the m = 1 base learner are given greater weight when training the m = 2 base learner. Source: Bishop 2006

Ensemble learning algorithms

Stacking

- train a second-level metalearner to find the optimal combination of the base learners.
- Unlike bagging and boosting, the goal in stacking is to ensemble stronger, diverse sets of learners together.

Table of contents

1. Motivation: multi-class classification
   - K-Nearest Neighbors classifier
   - Decision Tree

2. Ensemble Methods
   - Bagging
   - Random Forests

Random forests

Tree

A tree $\mathcal{T}$ is an ensemble of Q nodes:

$$\mathcal{T} = \left\{ \mathcal{L}^{(q)} \right\}_{q=1}^{Q}$$

inducing a partition over the feature space:

$$\mathcal{P} = \{\mathcal{C}_z\}_{z=1}^{Z}$$

where $\mathcal{C}_z$ is a cell.

Random forest

A random forest $\mathcal{F}$ is an ensemble of T random trees :

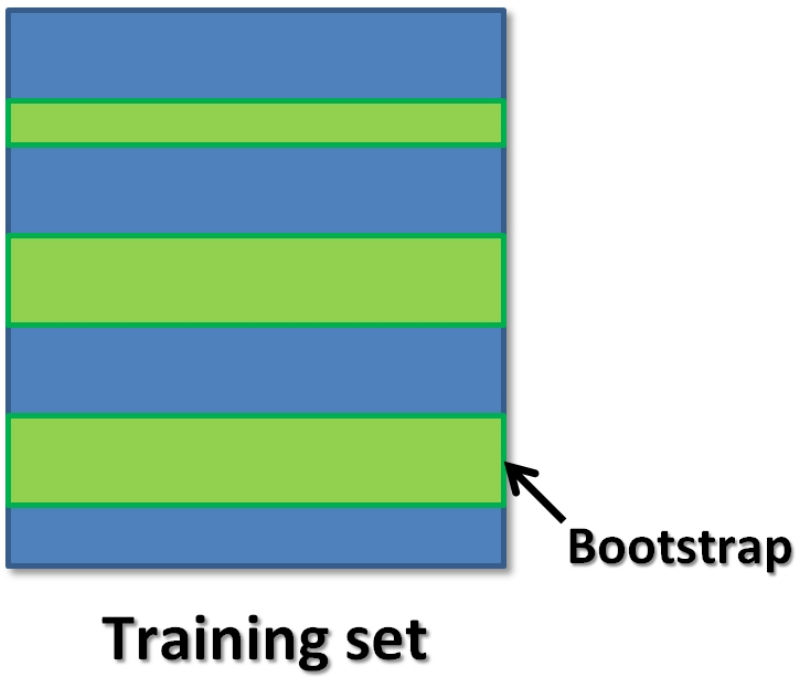
$$\mathcal{F} = \left\{ \mathcal{T}^{(t)} \right\}_{t=1}^{T}$$

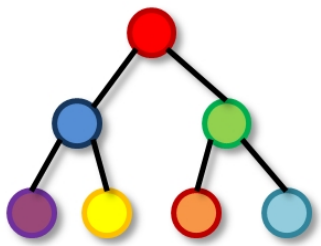
A Forest induces **a set** of random partitions over the feature space:

$$\mathbf{P} = \{\mathcal{P}_t\}_{t=1}^{T}$$

Random forests

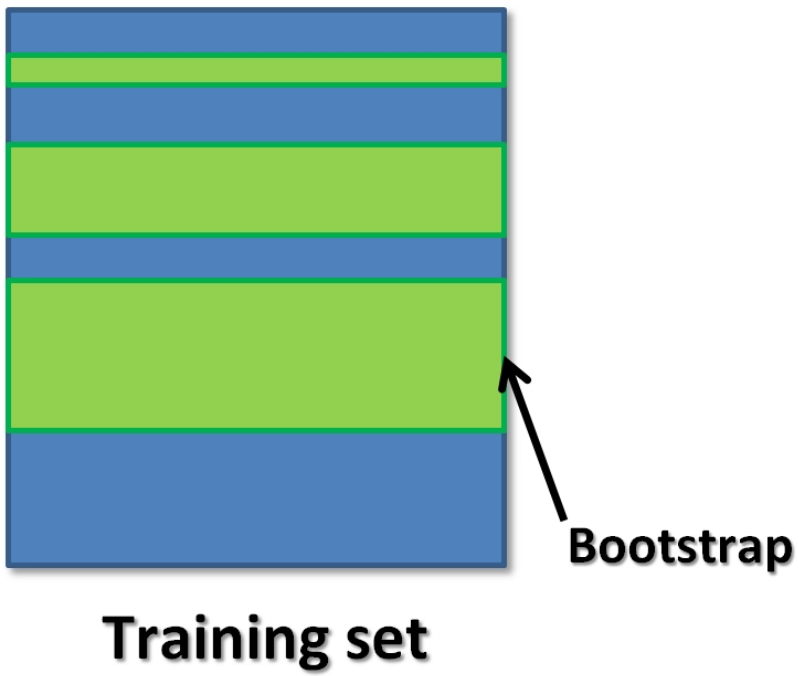
Training: Performed using Bagging (Bootstrap aggregating)
Each tree is trained using a different **bootstrap** of the training set.

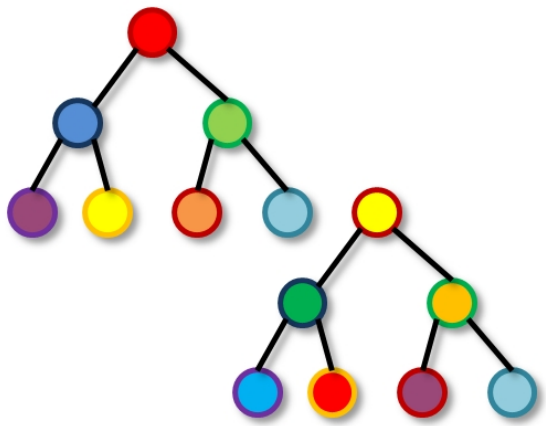




Random forests

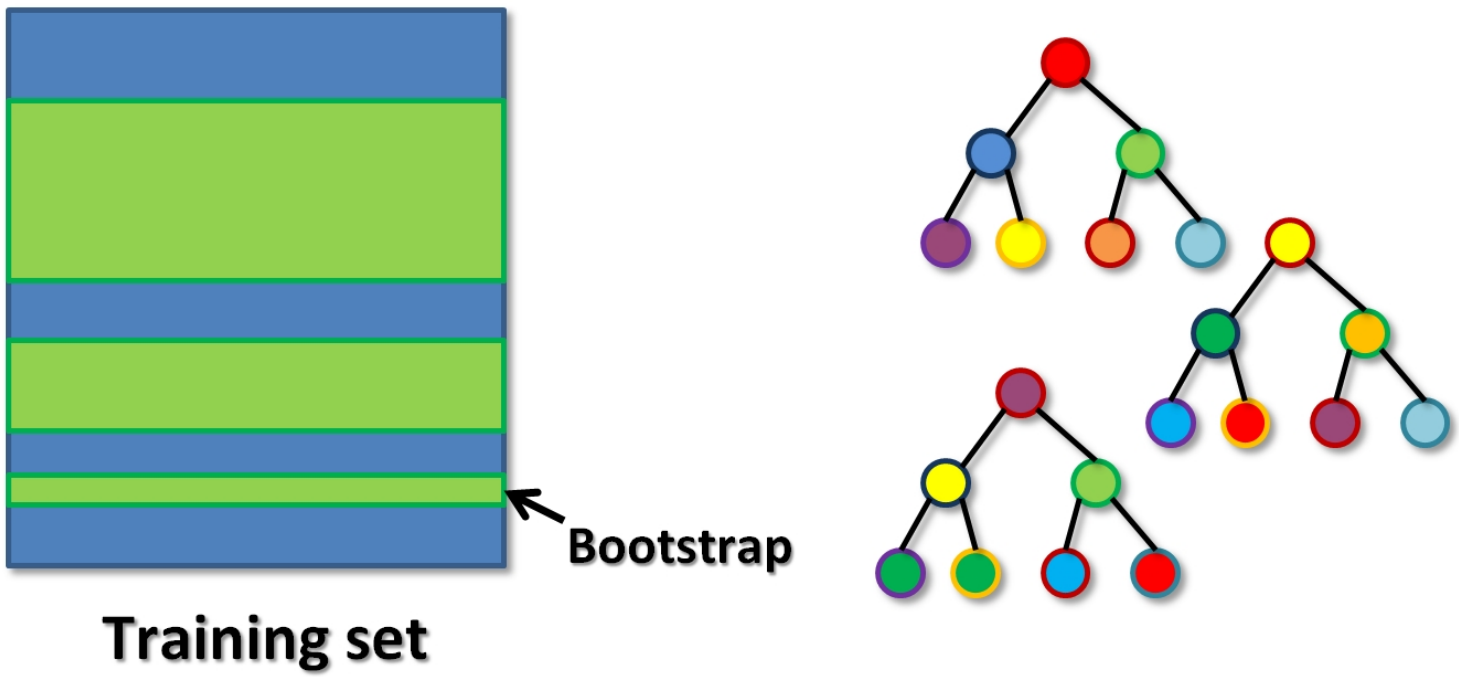
Training: Performed using Bagging (Bootstrap aggregating)
Each tree is trained using a different **bootstrap** of the training set.





Random forests

Training: Performed using Bagging (Bootstrap aggregating)
Each tree is trained using a different **bootstrap** of the training set.



Random forests training

Input:
- Training data $(\mathbf{X}, \mathbf{Y}) = \{\mathbf{x}_i, y_i\}_{i \in \{1, \cdots, n\}}$
- Nb of iterations T
- Cardinality of the bootstrap M

FOR $t = 1, \cdots, T$

1. Generate a bootstrap $(\mathbf{x}_t, \mathbf{Y}_t)$ by selecting randomly M points and their associated labels from $(\mathbf{X}, \mathbf{Y})$.
2. Train random tree $\mathcal{T}^{(t)}$ using bootstrap $(\mathbf{X}_t, \mathbf{Y}_t)$
3. Add $\mathcal{T}^{(t)}$ to the forest $\mathcal{F}$

END

Random forests testing

Input:
- Random forest $\mathcal{F} = \left\{\mathcal{T}^{(t)}\right\}_{t=1}^{T}$
- Unkown test data point $\mathbf{x}$
- Aggregation rule Φ

FOR $t = 1, \cdots, T$

1. Push $\mathbf{x}$ through random tree $\mathcal{T}^{(t)}$
2. Gather class posteriors $\left\{p(k|\mathbf{x}, \mathcal{T}^{(t)})\right\}_{k \in \mathcal{K}}$
3. Store all tree predictions in $\left\{p(k|\mathbf{x}, \mathcal{T}^{(t)})\right\}_{k \in \mathcal{K}}^{t=1,\cdots,T}$

END

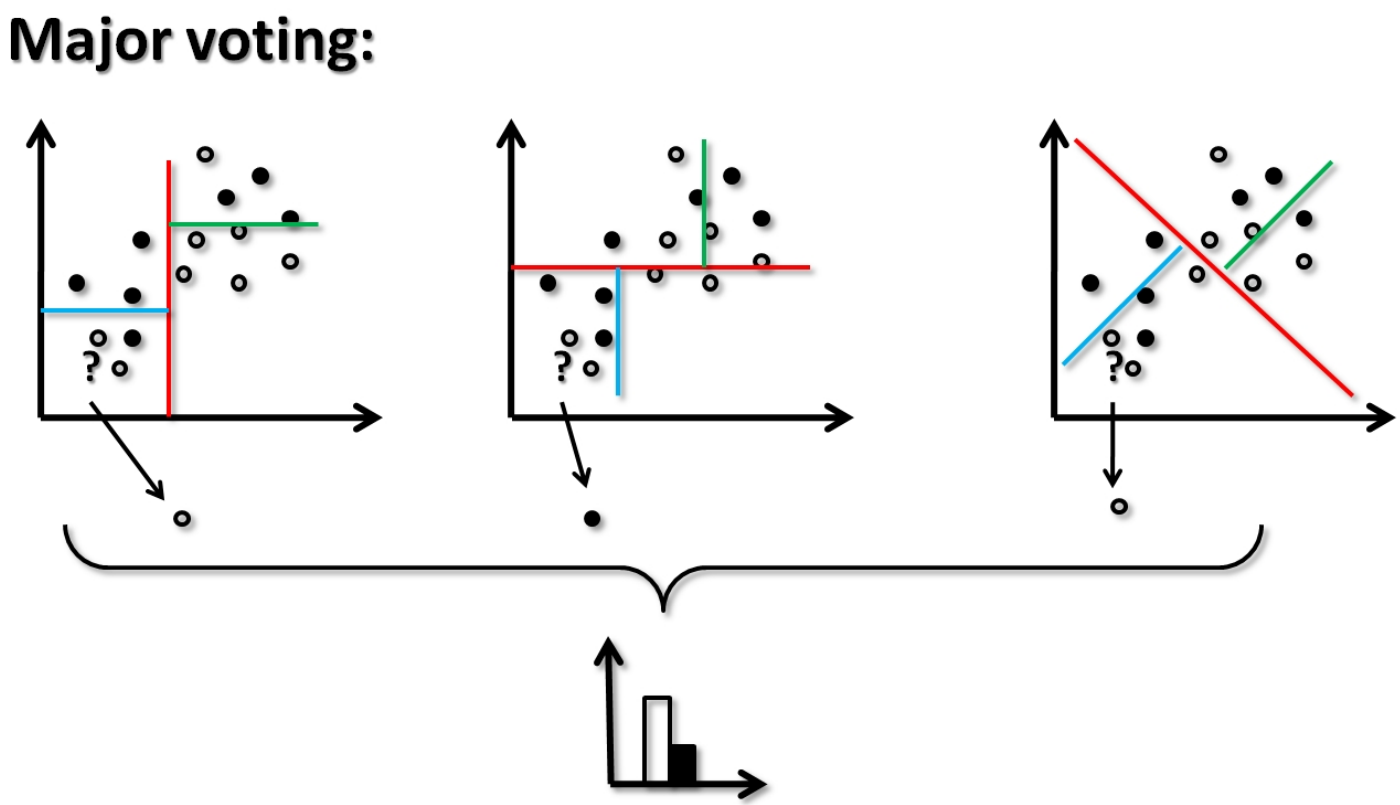
Aggregate predictions: $\hat{k} = \Phi\left(\left\{p(k|\mathbf{x}, \mathcal{T}^{(t)})\right\}_{k \in \mathcal{K}}^{t=1,\cdots,T}\right)$

Aggregation rules

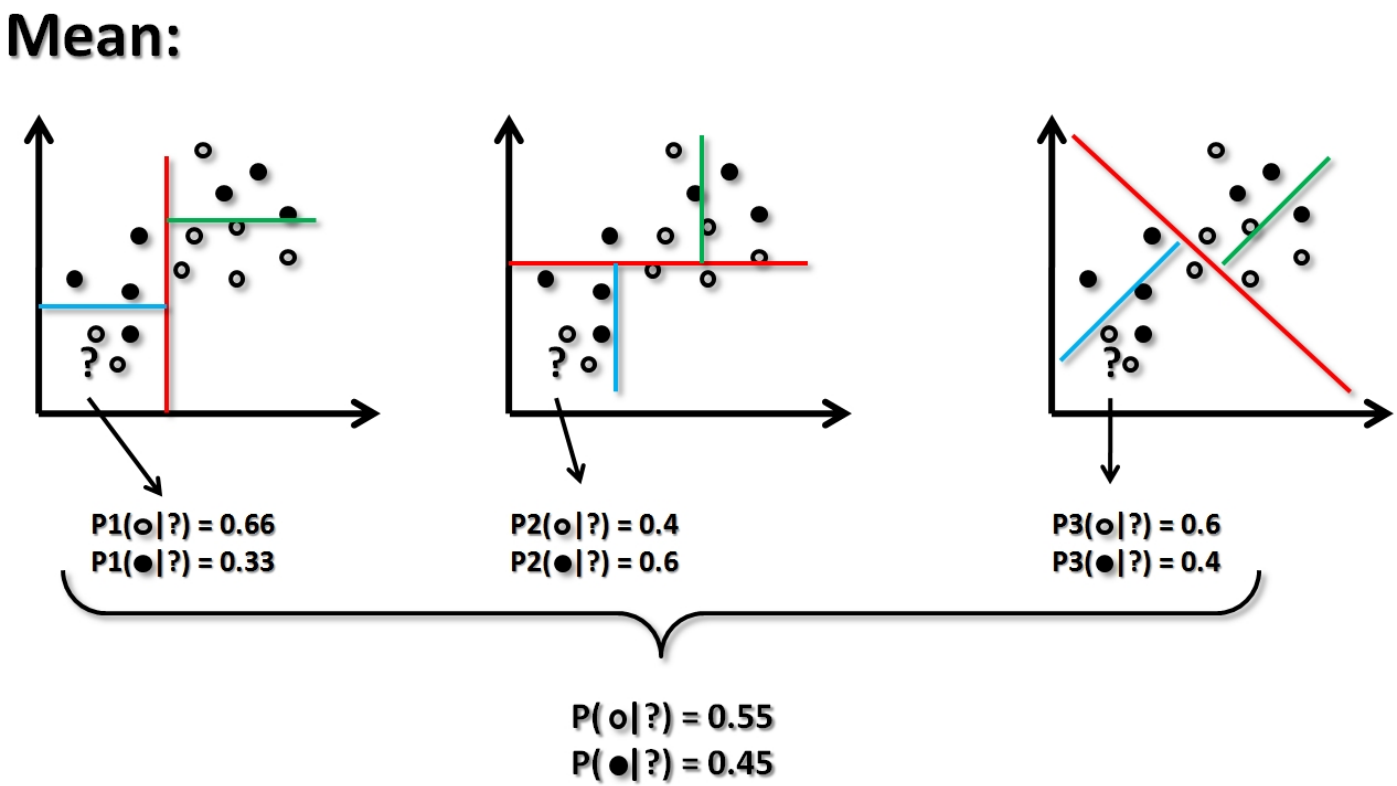
Several possibilities for the choice of Φ:

- **Major voting**
  $\hat{k} = \text{argmax}_{k \in \mathcal{K}} \left(\sum_{t=1}^{T} \left(k == \text{argmax}_{k \in \mathcal{K}} \, p(k|\mathbf{x}, \mathcal{T}^{(t)}) \right) \right)$
- **Sum rule**
  $\hat{k} = \text{argmax}_{k \in \mathcal{K}} \left(\sum_{t=1}^{T} p(k|\mathbf{x}, \mathcal{T}^{(t)}) \right)$
- **Mean rule**
  $\hat{k} = \text{argmax}_{k \in \mathcal{K}} \left(\frac{1}{T} \sum_{t=1}^{T} p(k|\mathbf{x}, \mathcal{T}^{(t)}) \right)$
- **Product rule**
  $\hat{k} = \text{argmax}_{k \in \mathcal{K}} \left(\prod_{t=1}^{T} p(k|\mathbf{x}, \mathcal{T}^{(t)}) \right)$

Aggregation rules



Aggregation rules

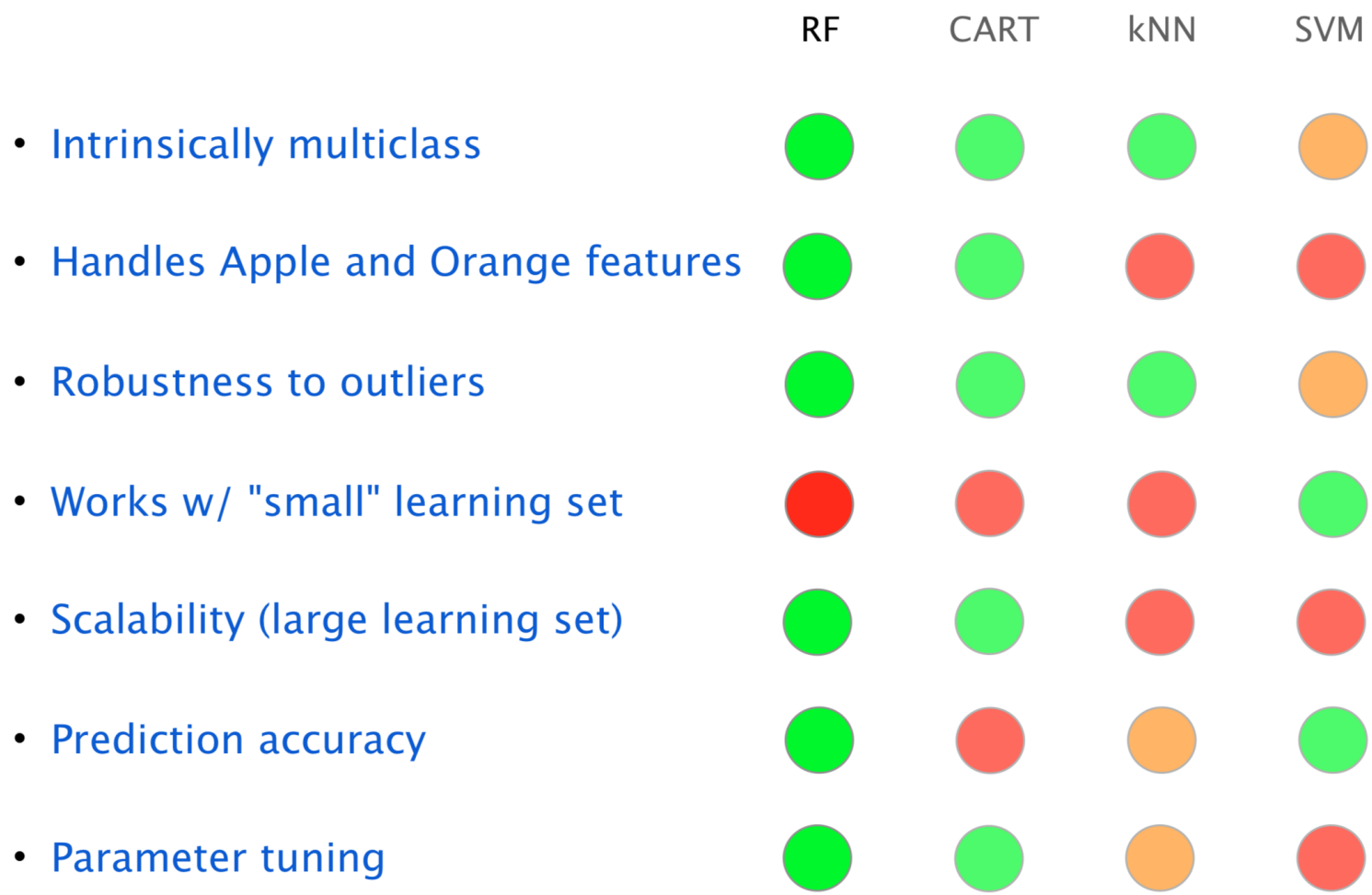


Conclusion

Random forest

- Real multi-class solution.
- Fast training and testing (can be parallelized!).
- Choice of number of trees and tree depth is very important.
- Achieves state-of-the-art performance: good generalization through randomization during training.

Comparison



Source: Eric Debreuve

Summary

Motivation: multi-class classification

- K-Nearest Neighbors classifier
- Decision Tree

Ensemble Methods

- Bagging
- Random Forests

References

- Bishop C. : Pattern Recognition and Machine Learning. Springer, 2006.
- Kevin P. Murphy, Machine Learning MIT Press, 2013.
- Andrew Ng video lectures for Machine Learning.
- Zhi-Hua Zhou, Ensemble Methods: Foundations and Algorithms, CRC Press, 2012
- L. Kuncheva, Combining Pattern Classifiers: Methods and Algorithms, Wiley, 2004
- Kaggle Ensembling Guide
- Scikit Learn Ensemble Guide