Decision Trees and Random Forests

Taki Eddine MEKHALFA
Taki-Eddine.MEKHALFA@inria.fr

Simple strategy:

1. Split (stratify, segment) the input space into disjoint regions

Simple strategy:

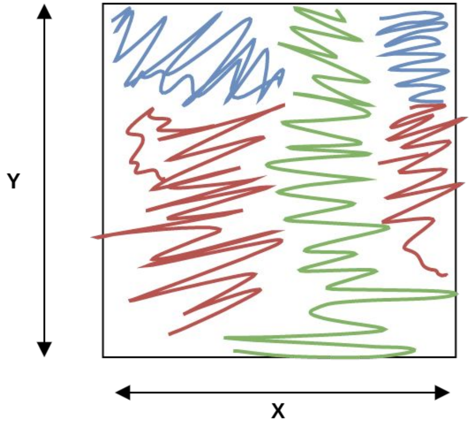
1. Split (stratify, segment) the input space into disjoint regions
2. For each input, find the region in to which it belongs

Simple strategy:

1. Split (stratify, segment) the input space into disjoint regions
2. For each input, find the region in to which it belongs
3. Return the prediction associated with this region (mean or mode for example)

Simple strategy:

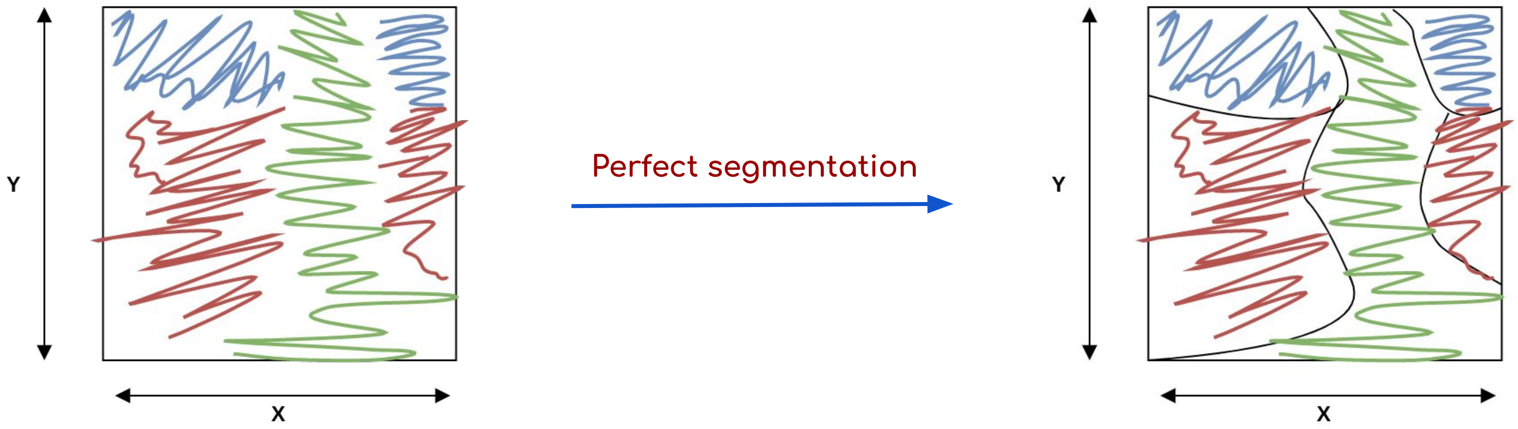
1. Split (stratify, segment) the input space into disjoint regions
2. For each input, find the region in to which it belongs
3. Return the prediction associated with this region (mean or mode for example)



Perfect segmentation

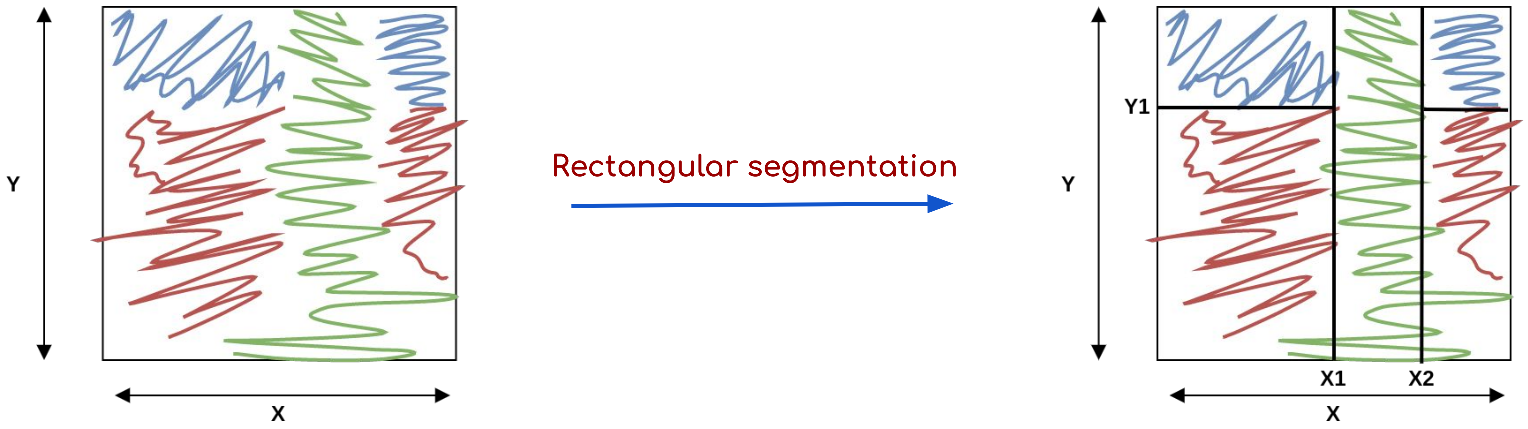
Simple strategy:

1. Split (stratify, segment) the input space into disjoint regions
2. For each input, find the region in to which it belongs
3. Return the prediction associated with this region (mean or mode for example)



Simple strategy:

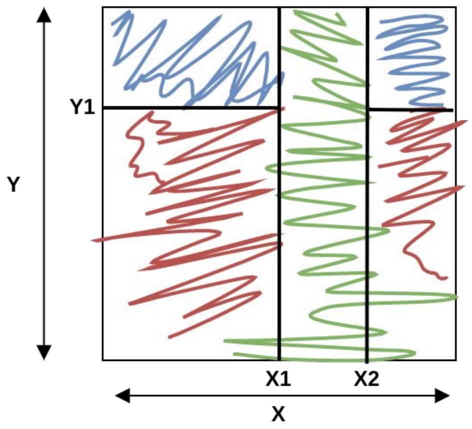
1. Split (stratify, segment) the input space into disjoint regions
2. For each input, find the region in to which it belongs
3. Return the prediction associated with this region (mean or mode for example)



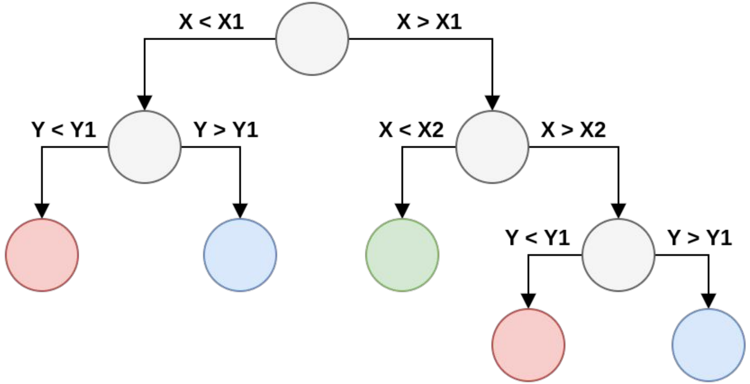
Decision Trees

<u>Simple strategy</u>:

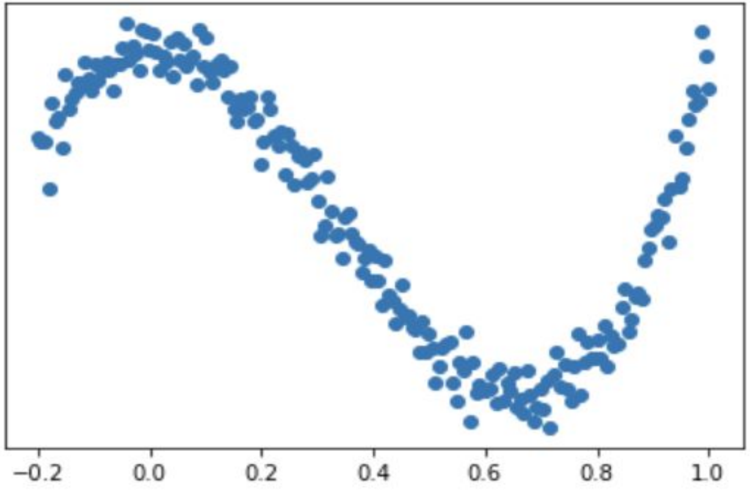
1. Split (stratify, segment) the input space into disjoint regions
2. For each input, find the region in to which it belongs
3. Return the prediction associated with this region (mean or mode for example)



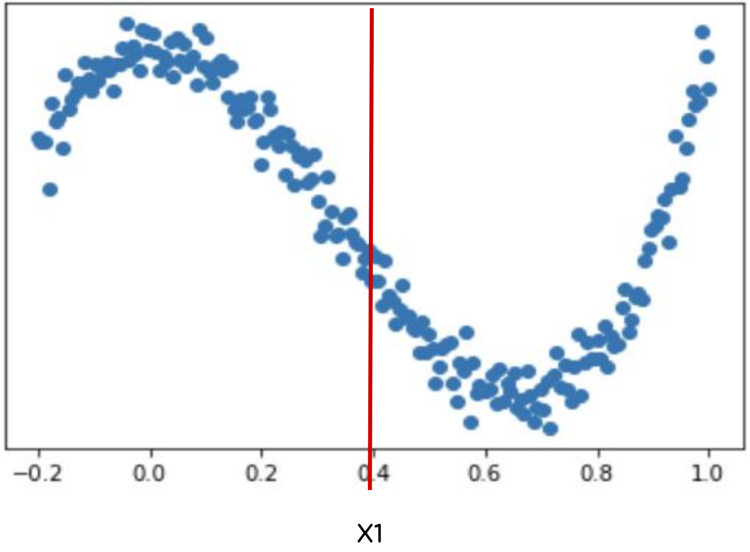
Equivalent Decision Tree



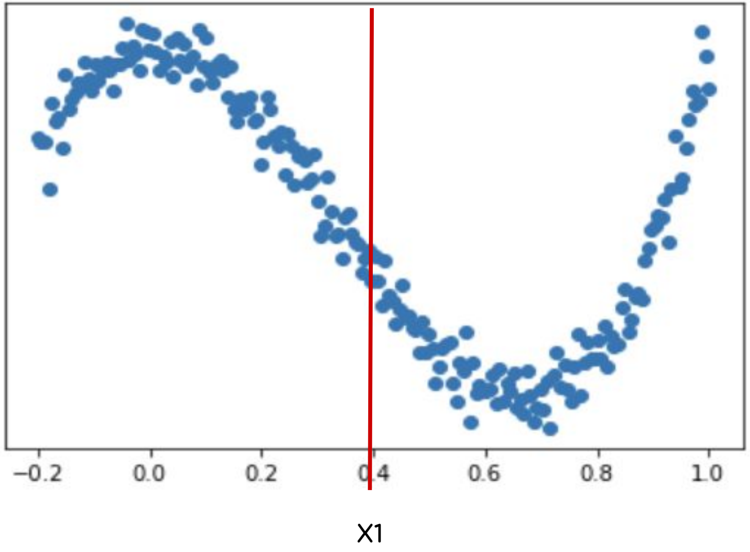
Regression example

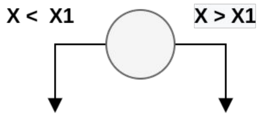


Regression example

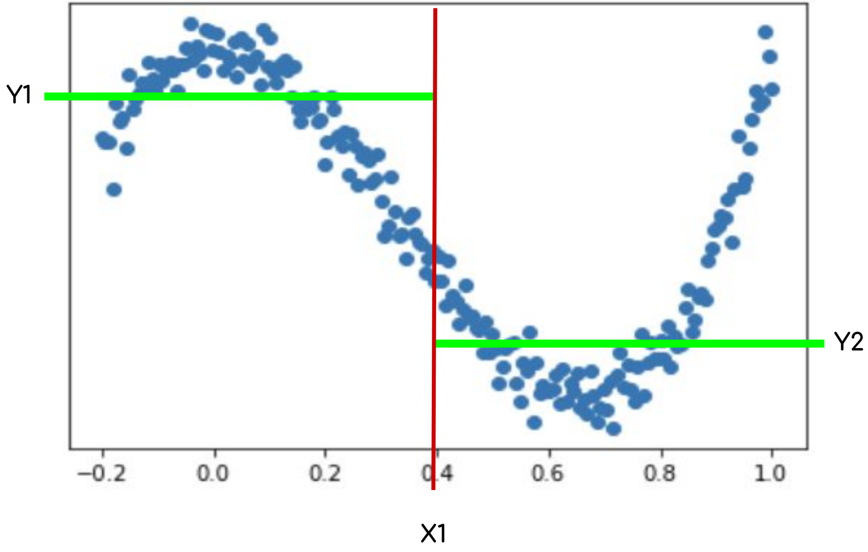


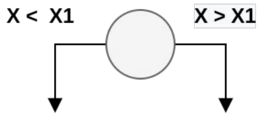
Regression example



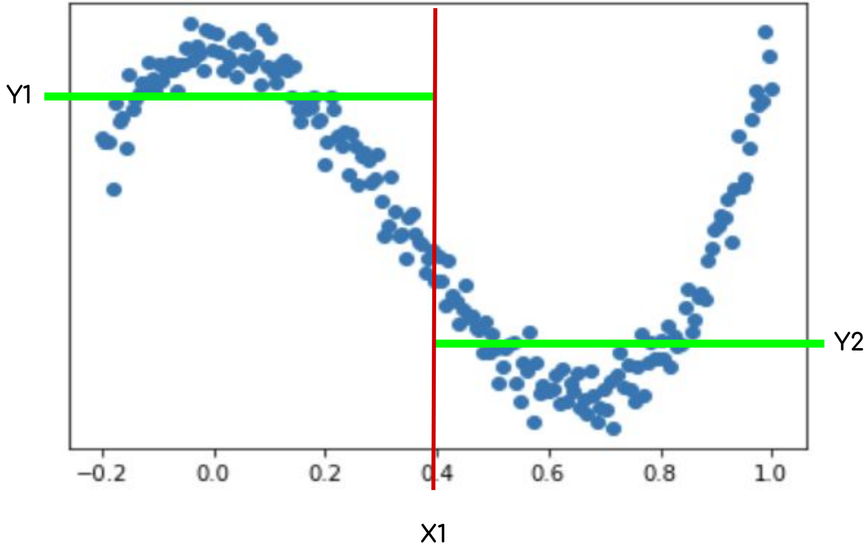


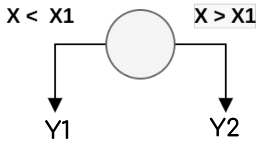
Regression example



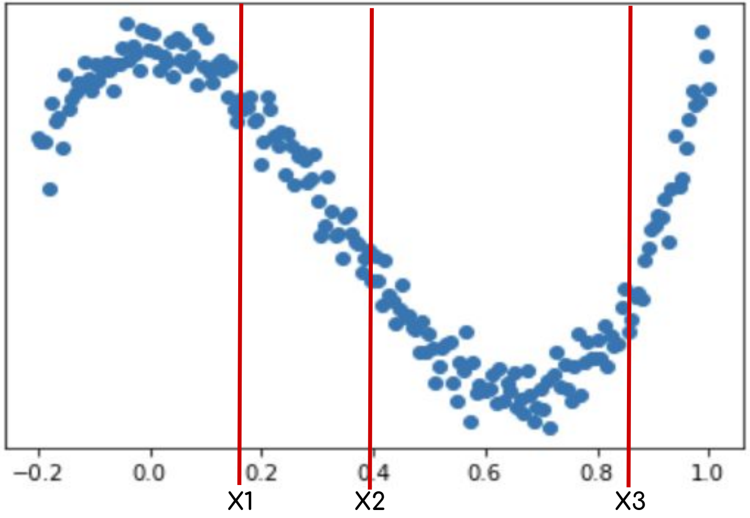


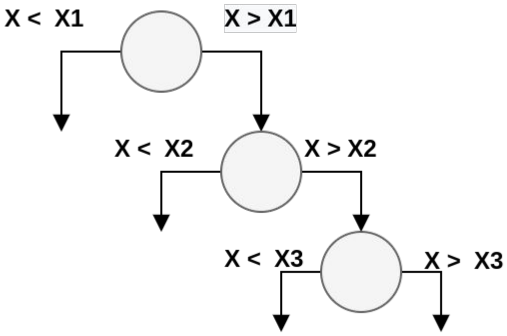
Regression example



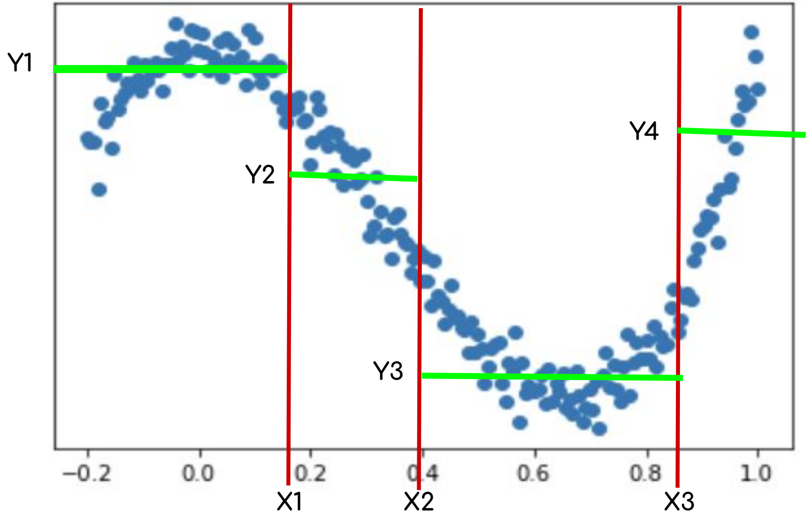


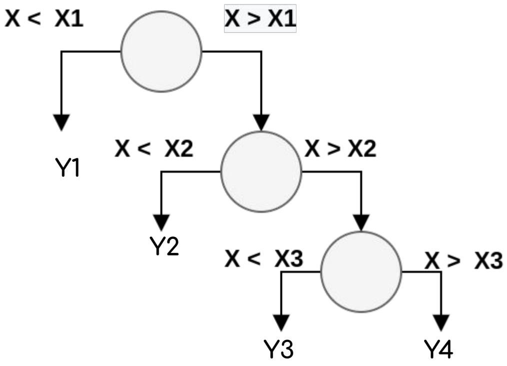
Regression example



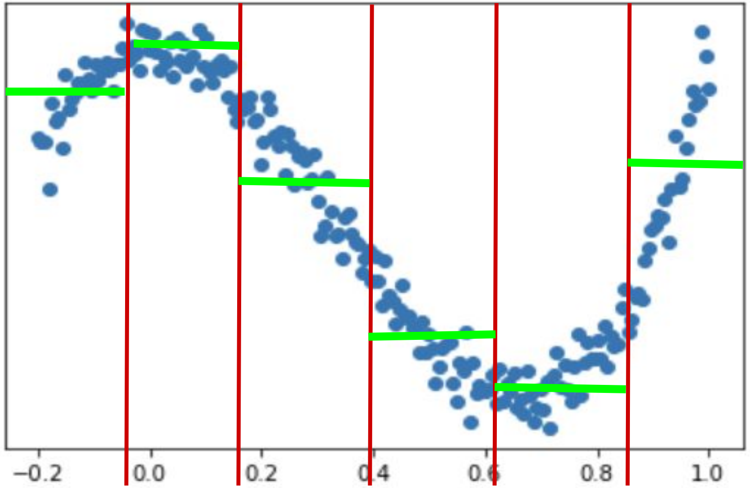


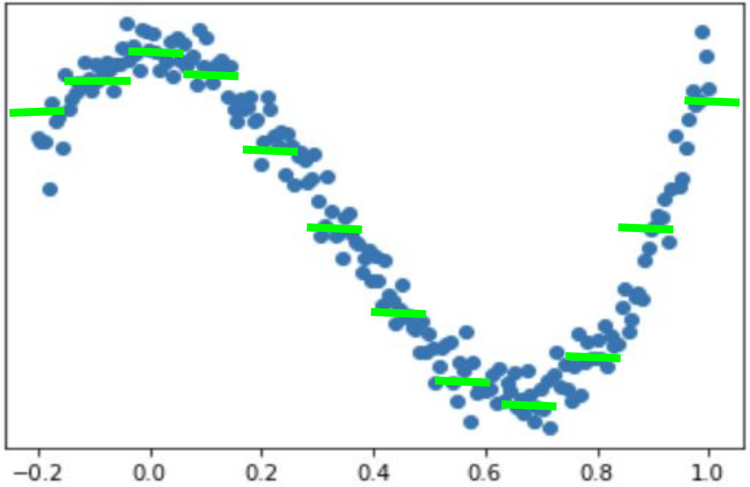
Regression example



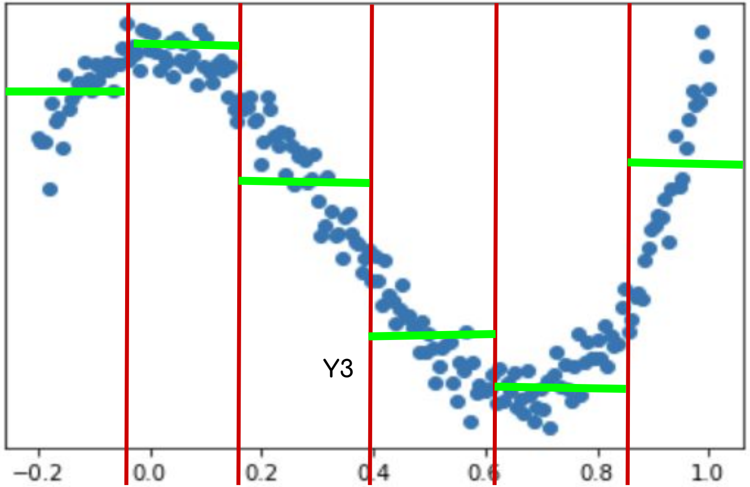


Regression example

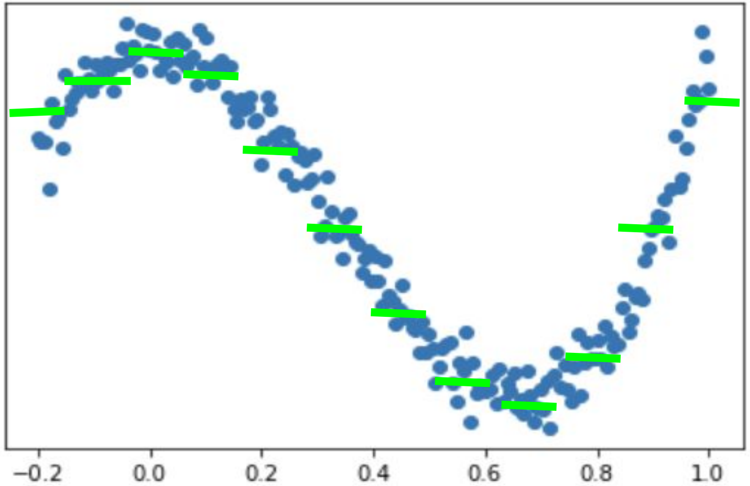




Regression example



1. Same thing for **high dimensional data** (you can segment your space using different **features**)
2. Same thing for **classification** (use **majority class** instead of the **mean**)



Building decision trees

- Decision trees are generally built only to a maximum depth and not until there is only one data point inside leaf nodes. We prefer moderately deep trees than very deep trees. Why?

Building decision trees

- Decision trees are generally built only to a maximum depth and not until there is only one data point inside leaf nodes. We prefer moderately deep trees than very deep trees. Why?
    - We don't build very deep trees to avoid overfitting

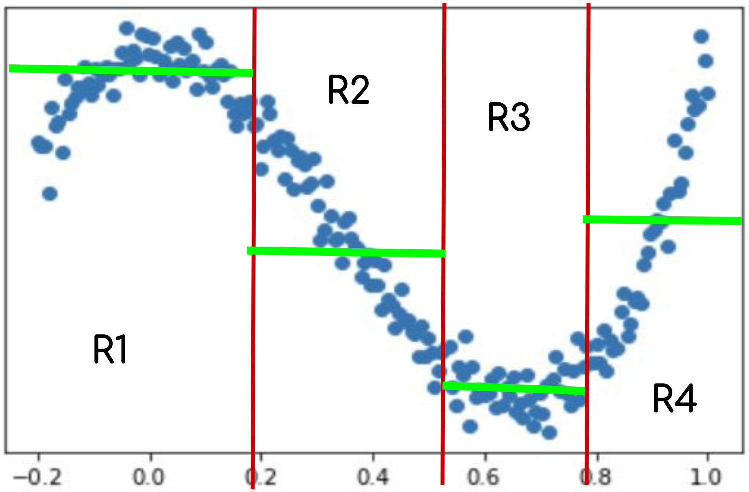
Building decision trees

- Decision trees are generally built only to a maximum depth and not until there is only one data point inside leaf nodes. We prefer moderately deep trees than very deep trees. Why?
    - We don't build very deep trees to avoid overfitting

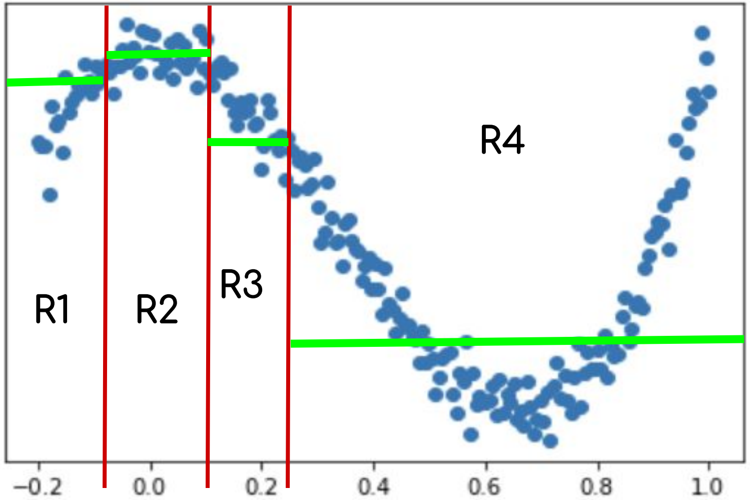
- We have a limited number of splits. What features we choose to split on? And in what order? How to split a given feature?

Building trees: example

Split on some feature: What boundary?



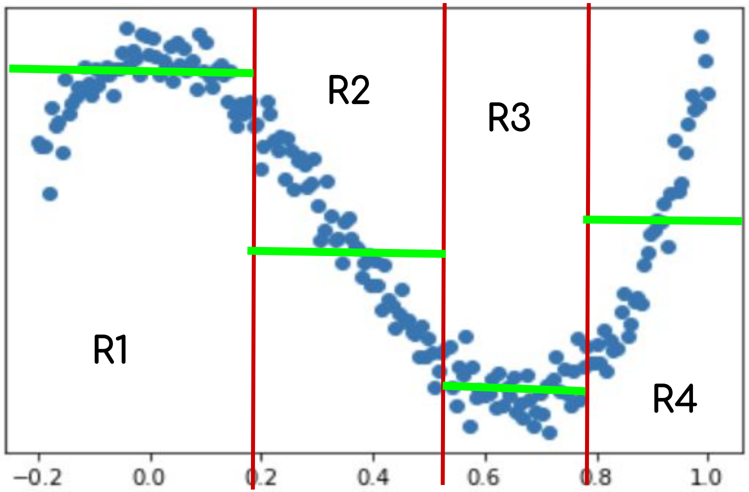
Tree 1



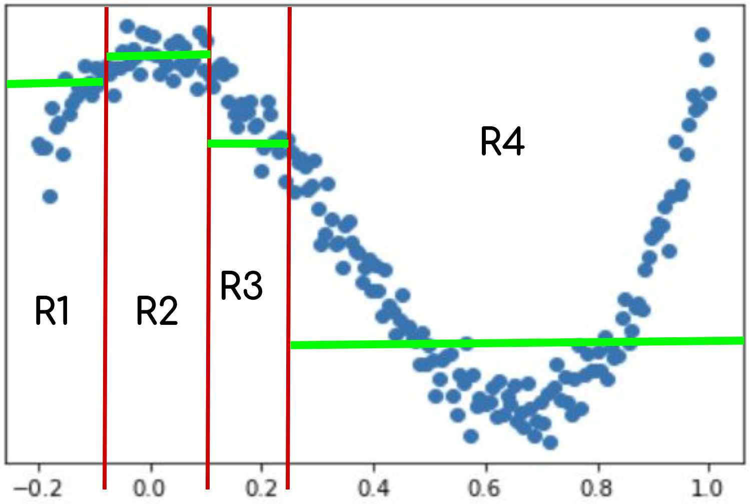
Tree 2

Building trees: example

Split on some feature: What boundary?



Tree 1

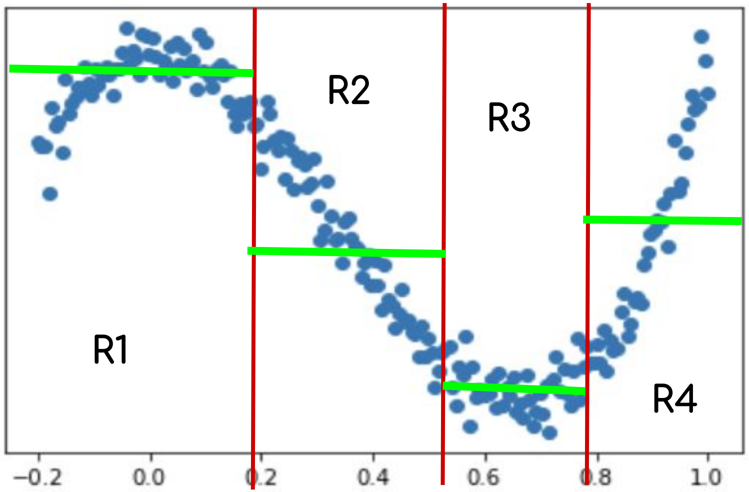


Tree 2

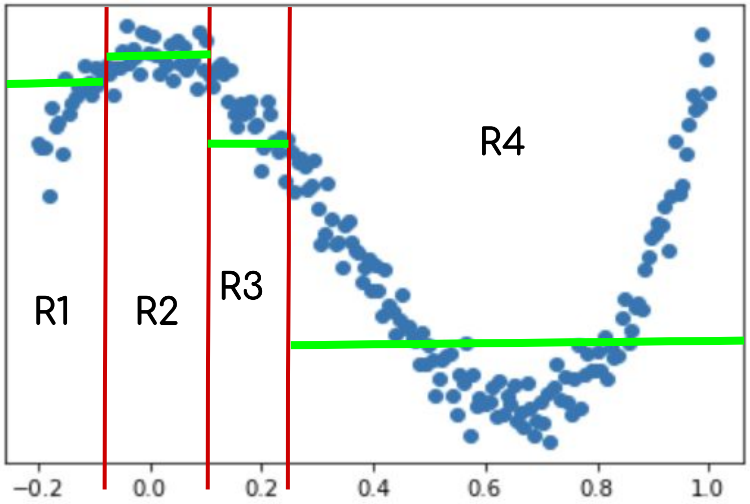
Cost(Tree) = $\sum$Cost(Region^i) s.t. Cost(Region^i) = MSE(Region^i) = MSE(y^j-m^i) where m^i is the mean of Region^i

Building trees: example

Split on some feature: What boundary?



Tree 1

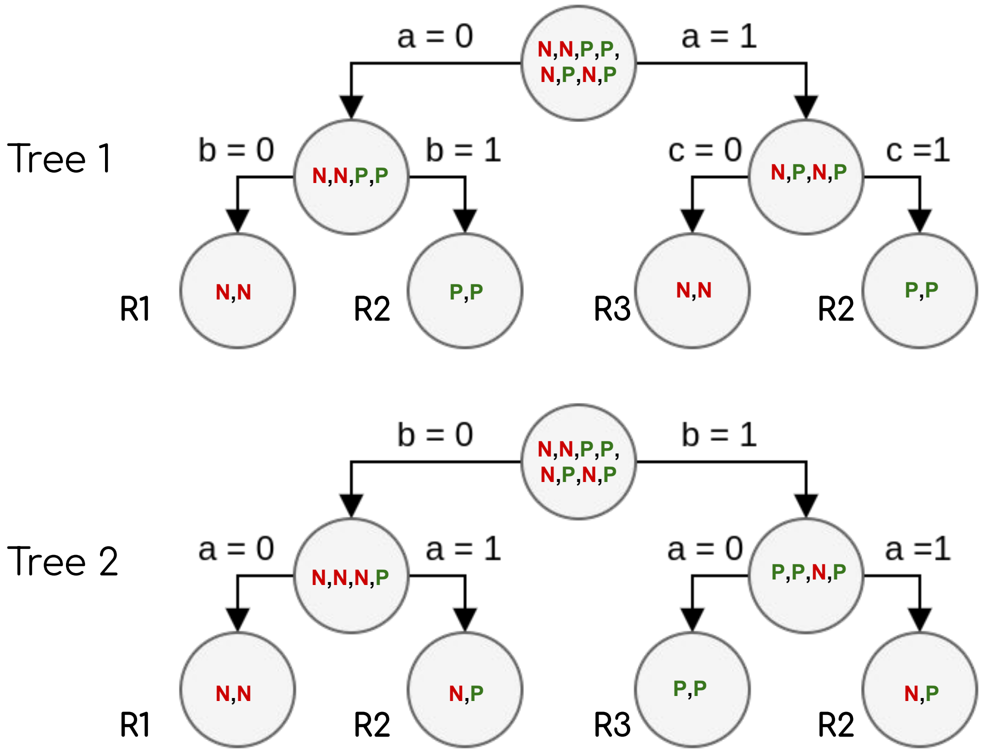


Tree 2

Cost(Tree) = $\sum$Cost(Region^i) s.t. Cost(Region^i) = MSE(Region^i) = MSE(y^j-m^i) where m^i is the mean of Region^i

Multiple features:In what order?

a	b	c	CLASS
0	0	0	NEG
0	0	1	NEG
0	1	0	POS
0	1	1	POS
1	0	0	NEG
1	0	1	POS
1	1	0	NEG
1	1	1	POS



Cost($Region^i$) = Impurity($Region^i$)

Building trees: example

Multiple features:In what order?

a	b	c	CLASS
0	0	0	NEG
0	0	1	NEG
0	1	0	POS
0	1	1	POS
1	0	0	NEG
1	0	1	POS
1	1	0	NEG
1	1	1	POS

a = 0

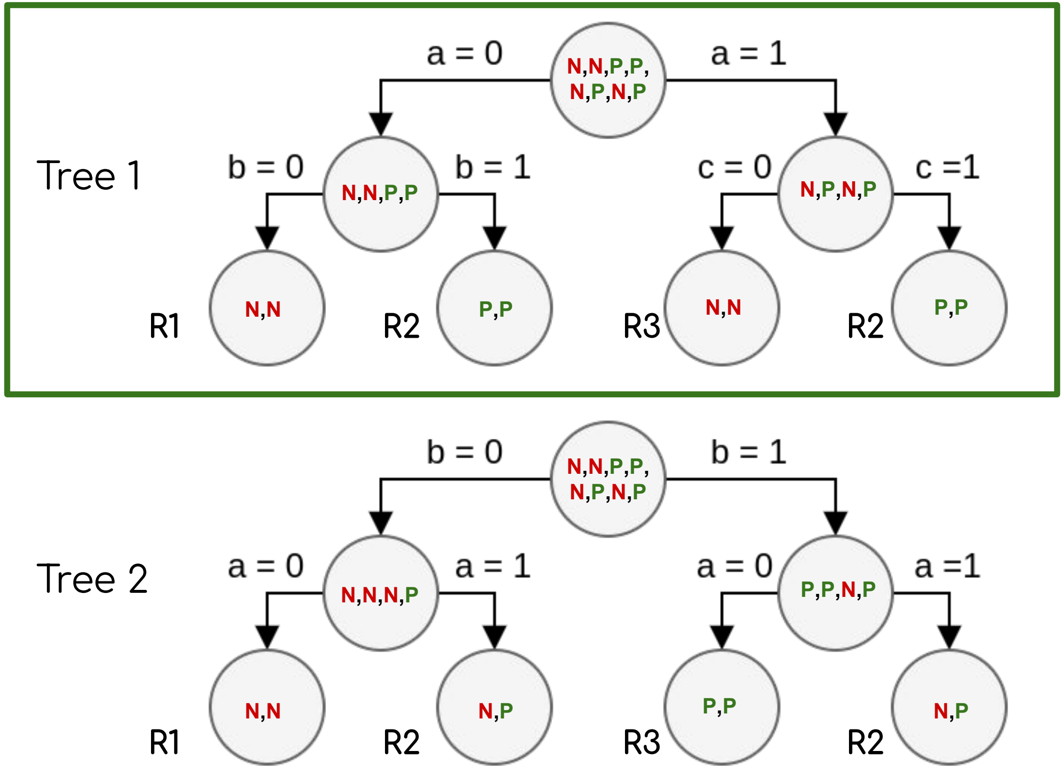
b = 0

b = 1

c = 0

c = 1

a = 1



Cost(Region^i) = Impurity(Region^i)

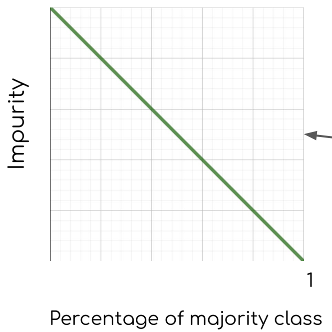
Building trees: example

Cost(Region^i) = Impurity(Region^i)

P,P,P,P Is purer than P,P,P,N Is purer than P,P,N,N

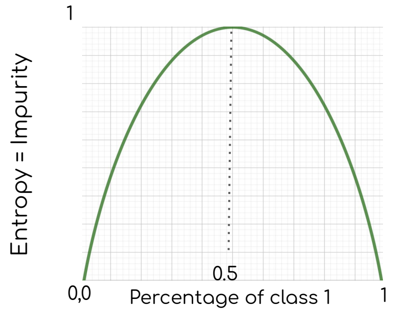
How to quantify Impurity?

First Idea: 1 - percentage of the majority class



The more the majority class is major the smaller the impurity

Second Idea: Entropy



The more the classes are equally probable, the more entropy (impurity) is high and vice versa

Entropy = -∑log(percentage(C^j)) x percentage(C^j)

Building trees: Algorithm

- **Inputs:**
  - Depth of the tree: K
  - Learning data points

A. **Algorithm:**
   a. For all trees of depth K choose the best tree with respect to the defined loss

Building trees: Algorithm

- **Inputs:**
  - Depth of the tree: K
  - Learning data points

A. **Algorithm:**
   a. For all trees of depth K choose the best tree with respect to the defined loss

Non feasible (actually NP-Complete).
There is an intractable number of **K-depth** trees

- **Inputs:**
  - Depth of the tree: K
  - Learning data points

A. **Algorithm:**
   a. For all trees of depth K choose the best tree with respect to the defined loss

What can we do?

Non feasible (actually NP-Complete).
There is an intractable number of **K-depth** trees

Building trees: Algorithm

- **Inputs:**
  - Depth of the tree: K
  - Learning data points

A. **Algorithm:**
   a. For all trees of depth K choose the best tree with respect to the defined loss

Non feasible (actually NP-Complete).
There is an intractable number of K-depth trees

What can we do?

Optimize greedily!

Building trees: Algorithm

- **Inputs:**
  - Depth of the tree: K
  - Learning data points

A. **Algorithm:**
   a. For each internal node with depth = $d < K$:
      i. Choose the feature f s.t when splitted it decreases the loss the most.
      ii. Split on f and create two binary nodes and increment their depth to $d + 1$

Feasible

Building trees: Algorithm

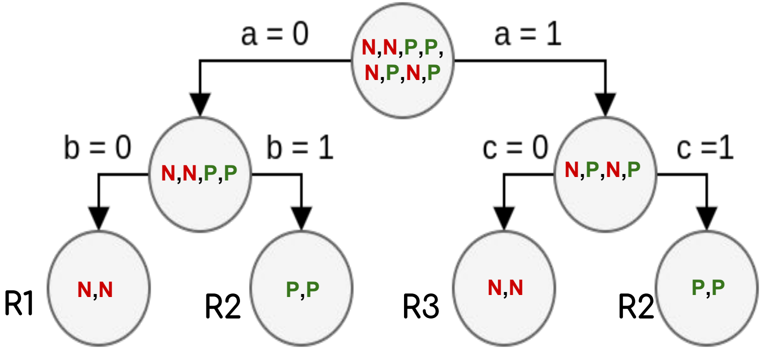
Greedy algorithm:

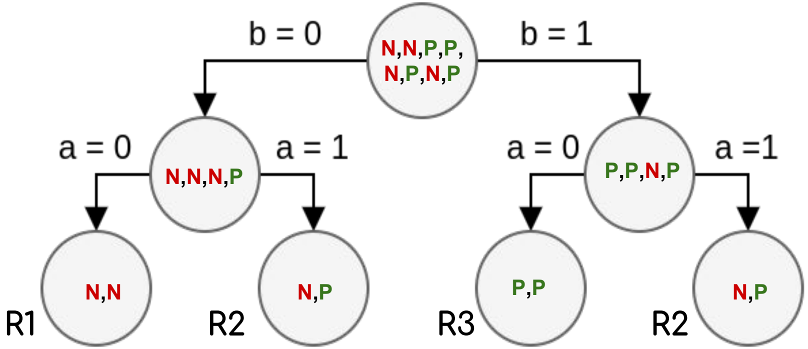
1. Simple and efficient
2. BUT we are not guaranteed to build the optimal tree
3. Nonetheless works well in practice.

Building trees: Algorithm

Greedy algorithm:

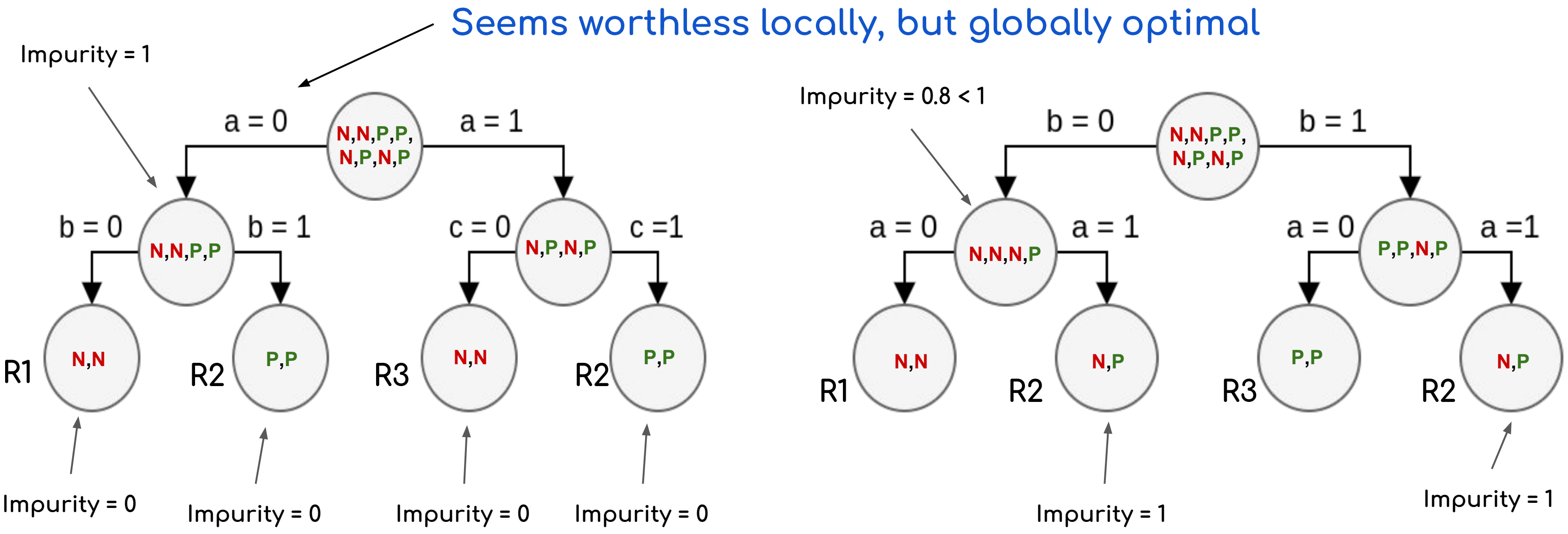
1. Simple and efficient
2. BUT we are not guaranteed to build the optimal tree
3. Nonetheless works well in practise.





Greedy algorithm:

1. Simple and efficient
2. BUT we are not guaranteed to build the optimal tree
3. Nonetheless works well in practise.



Take away points:

- Decision trees are simple
- Decision trees are interpretable (their behavior can be explained using splitting rules) and mimic rule based human decision making (trees can be easily visualized)
- Decision trees are built using a greedy algorithm which usually works well in practice.
- Decision trees are a flexible model: Small bias (fits non linear relationships) but a big variance (a small change in data might result in a large change of the final tree i.e overfit easily)

Decision trees are overfit easily (high variance)! How to remediate this problem?

Decision trees are overfit easily (high variance)! How to remediate this problem?

Suppose you have n independent variables $\{Z_1, Z_2, ..., Z_n\}$ where $var(Z_i) = \sigma^2$,

then $var(\overline{Z_n}) = \sigma^2/n$ i.e averaging reduces the variance.

Decision trees are overfit easily (high variance)! How to remediate this problem?

Suppose you have n independent variables $\{Z_1, Z_2, ..., Z_n\}$ where $var(Z_i) = \sigma^2$,

then $var(\overline{Z_n}) = \sigma^2/n$ i.e averaging reduces the variance.

Idea => Construct B decision trees on B independent data sets and average their results at prediction time (or use majority vote in case of classification for example)

Decision trees are overfit easily (high variance)! How to remediate this problem?

Suppose you have n independent variables $\{Z_1, Z_2, ..., Z_n\}$ where $var(Z_i) = \sigma^2$,

then $var(\overline{Z_n}) = \sigma^2/n$ i.e averaging reduces the variance.

Idea => Construct B decision trees on B independent data sets and average their results at prediction time (or use majority vote in case of classification for example)

Problem => We don't have B data sets!

Decision trees are overfit easily (high variance)! How to remediate this problem?

Suppose you have n independent variables $\{Z_1, Z_2, ..., Z_n\}$ where $var(Z_i) = \sigma^2$,

then $var(\overline{Z_n}) = \sigma^2/n$ i.e averaging reduces the variance.

Idea => Construct B decision trees on B independent data sets and average their results at prediction time (or use majority vote in case of classification for example)

Problem => We don't have B data sets!
Solution => Use bootstrapping!

Decision trees are overfit easily (high variance)! How to remediate this problem?

Suppose you have n independent variables $\{Z_1, Z_2, ..., Z_n\}$ where $var(Z_i) = \sigma^2$,

then $var(\overline{Z_n}) = \sigma^2/n$ i.e averaging reduces the variance.

Idea => Construct B decision trees on B independent data sets and average their results at prediction time (or use majority vote in case of classification for example)

Problem => We don't have B data sets!
Solution => Use bootstrapping!

Bootstrapping: A sampling technique where for each data set with n points, sample n points uniformly with replacement (i.e a given data point might be sampled multiple times)

Thoeretically, Bagging (Bootstrap aggregating), can be used on any prediction model.
We show here a pseudo algorithm on bagging with regression decision trees:

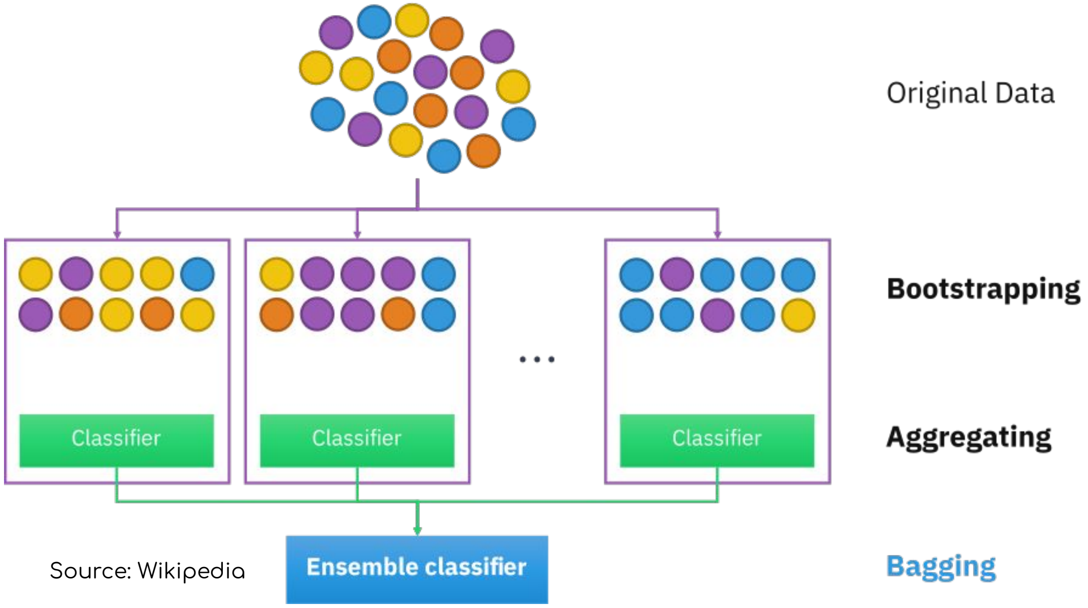
- Inputs:
  - Number of trees B
  - Depth of a tree

A. Algorithm:
   a. For b in range(B):
      i. Bootstrap a data set D^b of size n
      ii. Fit a decision tree T^b on D^b
      iii. Add T^b to the already built trees

B. Prediction:
   a. Take a input *i*
   b. For each b compute $T^b(i)$
   c. Return the average of all predictions

Bagging and Random Forests

Thoeretically, Bagging (Bootstrap aggregating), can be used on any prediction model.
We show here a pseudo algorithm on begging with regression decision trees:



Source: Wikipedia

Out-of-Bag error estimation:

=> Because of bootstrapping, each tree is constructed using only a subset of the total data points (one can show that a tree is constructed with only ~ 2/3 of data)

Out-of-Bag error estimation:

=> Because of bootstrapping, each tree is constructed using only a subset of the total data points (one can show that a tree is constructed with only ~ 2/3 of data)

=> random 1/3 of data is not used to construct a given tree i.e each data point is not used in 1/3 of trees

Out-of-Bag error estimation:

=> Because of bootstrapping, each tree is constructed using only a subset of the total data points (one can show that a tree is constructed with only ~ 2/3 of data)

=> random 1/3 of data is not used to construct each tree i.e each data point is not used in 1/3 of trees

=> This remaining data can be used as a validation subset to approximate test errors
=> No need for cross validation

Take away points:

- Increasing the total number of trees will not cause overfitting, therefore one can increase the number of trees until the error settles

Take away points:

- Increasing the total number of trees will not cause overfitting, therefore one can increase the number of trees until the error settles

- Individual trees can be built to deep levels, we don't care about individual overfitting since this effect is overridden by averaging

Take away points:

- Increasing the total number of trees will not cause overfitting, therefore one can increase the number of trees until the error settles

- Individual trees can be built to deep levels, we don’t care about individual overfitting since this effect is overridden by averaging

- If there are strong features, all trees will use them in first levels => Trees will have similar behavior (correlated) => We don’t gain much by averaging => Use Random Forests

How can we **decorrelate** different trees ?

How can we decorrelate different trees ?

=> for each tree, use a subset of features only (not all features)

How can we decorrelate different trees ?

=> for each tree, use a subset of features only (not all features)

=> Generally we build a tree using m randomly chosen features (where $m < p$ the total number of features)

=> In practice we use $m = \sqrt{p}$

How can we decorrelate different trees ?

=> for each tree, use a subset of features only (not all features)

=> Generally we build a tree using m randomly chosen features (where $m < p$ the total number of features)

=> In practice we use $m = \sqrt{p}$

=> When stronger features are not chosen, other features has more chances to be explored => trees are different and uncorrelated

Take away points:

- Random forests is just a bagging of decision trees but trees are built on a subset of features only.
- When $m = p$, random forests = bagging