Decision Trees and Random Forests
Data science Certificate

Aymeric DIEULEVEUT

May 2023

x task

1. A Reminder on Supervised Learning : x examples

x why we choose trees

2. Decision Trees

x how we build trees

x how we predict with them

Choose Hyperparameters

Learned model

3. Random Forests

Outline

A Reminder on Supervised Learning

Task: predict Y from X.

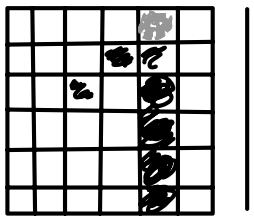
given $(x_i, y_i)_{i=1..n}$ observed

inputs / features

- Observations: $(X_i)_{i=1,\dots n}$. Each X_i has d components.
- Outputs: $(Y_i)_{i=1,\dots n}$

output

3 Examples:

- Iris Dataset: Predict species from 4 features: $\begin{cases} \text{petal length} \\ \text{width} \\ \text{sepal} \quad " \\ \quad\quad " \end{cases}$
- MNIST: X  28×28 $\quad 28^2 = 784$ pixels

  $Y = \{0, \dots 9\}$ → finite number of CLASSES → CLASSIFICATION
- House price prediction: predict the price of a house

  Y = house price → $\in \mathbb{R}_+$: REGRESSION

  $X = \{$ surface (m^2), neighborhood, garden, floor, construction year, ... $\}$

Example: Iris Dataset



Figure: Iris: setosa, versiocolor, virginica

One of the most widely used toy dataset in classification:

- 3 classes : *['setosa', 'versicolor', 'virginica']* Y.
- 50 points per class. $n = 150$
- 4 features: *['sepal length', 'sepal width', 'petal length', 'petal width']*

Example: MNIST Dataset

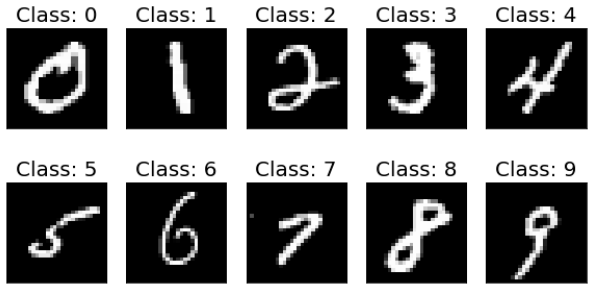


Figure:

Digit recognition from an image:

- 10 classes : $Y_i \in \{0, \dots 9\}$ classification
- 60k images, (50k train, 10k test), balanced
- 784 features.

House price Prediction

Famous kaggle.

X_1 100, 1st, Yes — 1M

X_2 20, 13th, No — 200k

X_3 200, 5th, Yes — 2M.

- Output : house value $Y \in \mathbb{R}_+$
- 1500 examples.
- 81 features: Construction year, Neighborhood, Surface, Floor, Balcony: both quantitative and qualitative, some ordinal variables.

Simplified version where $X_\cdot = \{$ Surface, Neigh, Garden $\}$.

Surface → type → quantitative

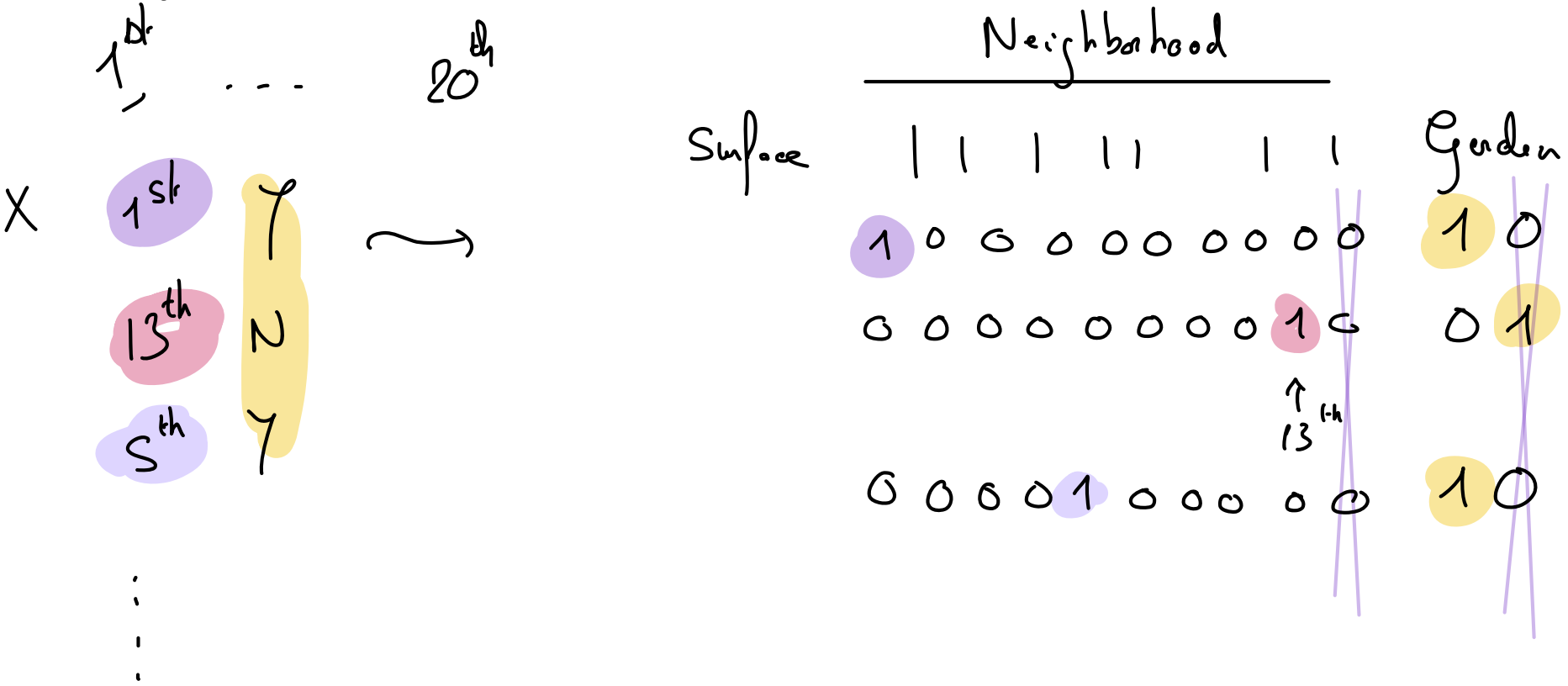
Garden → yes / no

~~Questions?~~

ONE HOT ENCODING

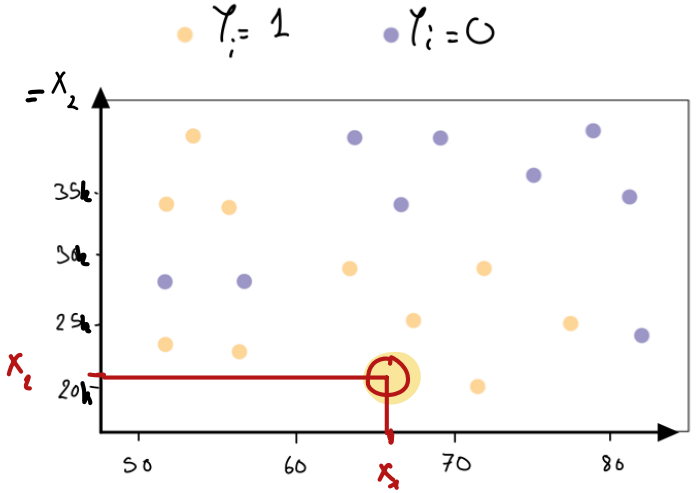
DUMMY ENCODING

Categorical variable → encoding.



Key idea : transform categorical variables with K categories into $K-1$ columns new features.

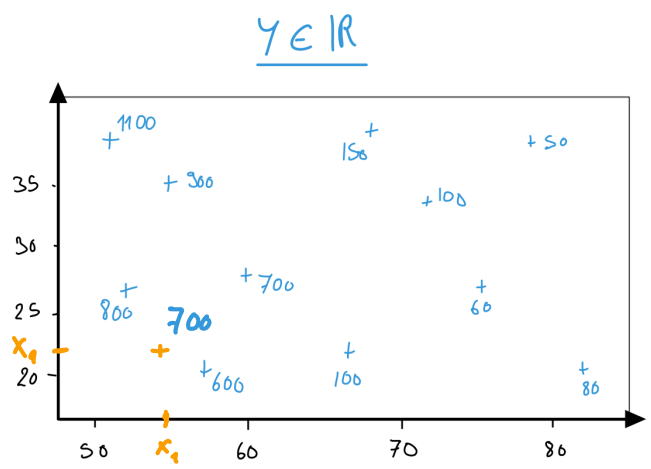
Toy dataset



$d = 2$

$n = 20$

$Y = \{0, 1\}$ Classification



$d = 2$

$n = 11$

$Y = \mathbb{R}$ Regression

Summary of those three datasets

	Iris	MNIST	House price prediction	Toy
Number of examples n	150	50k	1500	20-11
Number of the features d	4	784	81 - 3	2
Type of the features	quantitativ	quantitativ	qualit + quant	quant.
Space $\mathcal{Y}$	{setosa, va, virg}	{0, ... 9}	$\mathbb{R}_+$	$\{0,1\}/\mathbb{R}_+$
Task	Classif		Regressia	

Learning Pipeline

1. Training phase

Training set: given a training set

Goal: predict well! → on the train set (how) → most importantly: on the test set

Choose: method + its hyper-parameters

Learn: a prediction function = model: $\hat{f}$ such that $\hat{f}(X) \simeq Y$

2. Validation phase

× Class of functions ?

× How to find a good one ?

} ≠ methods

Choosing hyperparameters is hard → try many, evaluate the validation score for each. → take the best hyperparameter → use those!

3. Test phase (X_{test}, Y_{test}) both observed

Compute your test accuracy on never-seen data. → only number trusted

4. Deployment X_{new} — Y is really unknown

Reminder on Linear and Logistic regression

Linear method

input point X.

learn weights w (Beta before)

biases b

$\langle X, \beta \rangle$

$\langle X, w \rangle + b$

new notation

Take away:

	Lin. reg	& Logistic regress°
Task	Regress°	Classif
Result	Linear combinat° of features	
Learn	Weights	

→ + simple fast

→ − not enough

Limits of Linear and logistic regression

House price $\simeq$ $8k$ $\times$ Surface $+$ $100k \times$ Is_garden $+ \dots$

$+ \ 80k \times$ nb rooms $+ \ 40k$ open_kitchen

$+ \ \dots$

Outline

Decision Tree

Goals of Decision Trees

1. Supervised Learning: solve the classification or regression task
2. Provide some understanding: what on this example allows me to classify it? What are the important features ?

(3) go beyond linear functions ⟶ learn complex dependencies .

Decision Tree

Goals of Decision Trees

1. Supervised Learning: solve the classification or regression task
2. Provide some understanding: what on this example allows me to classify it? What are the important features ?

Two questions:

- How to train a decision tree? $\rightarrow$ **training phase (learning algorithm)**
- How to predict with a given decision tree? $\rightarrow$ **inference phase**

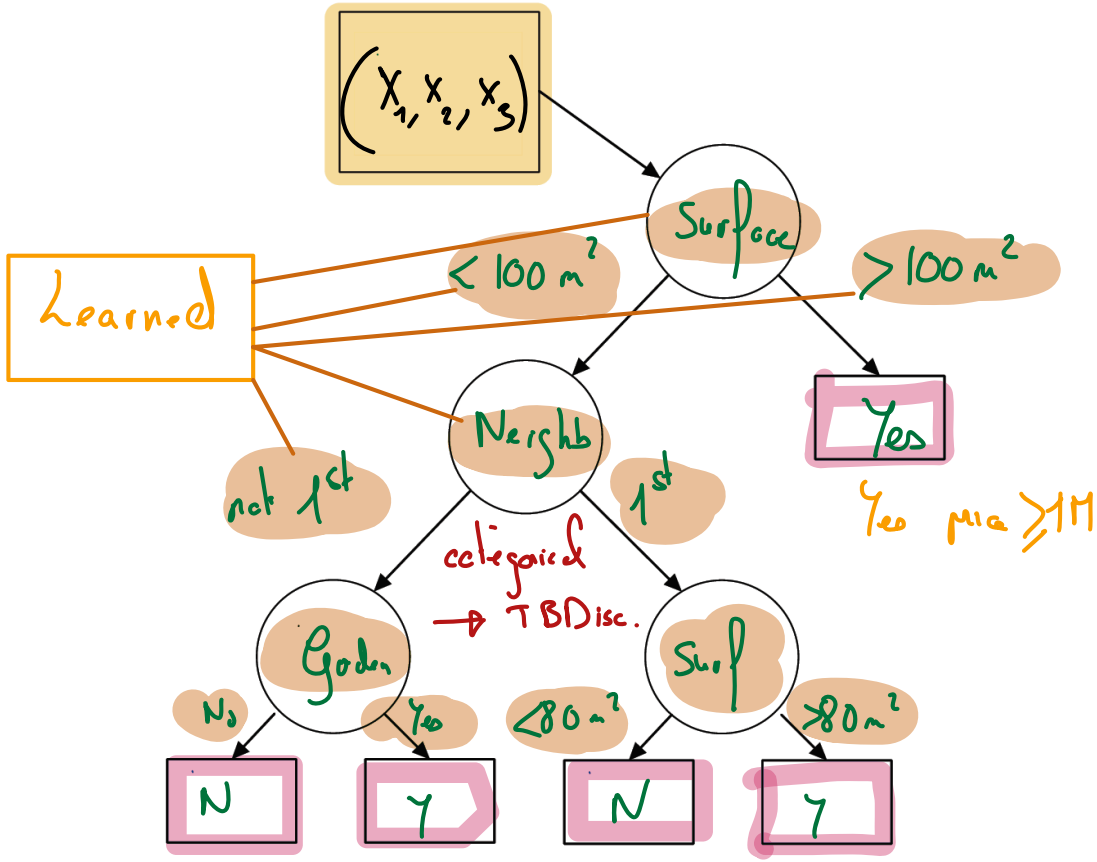
Inference Phase

house price. $Y = \mathbb{R}$

$Y = \mathbb{1}_{price > 1M}$

X_1, X_2, X_3 = Surface, Neighbor, Garden

- Root = observation = input
- Nodes = tests — binary answer
  → only one feature per node
- Branches = possible outcomes
- Leaves = final decision = output



For a given DT, prediction inference is straightforward interpretable

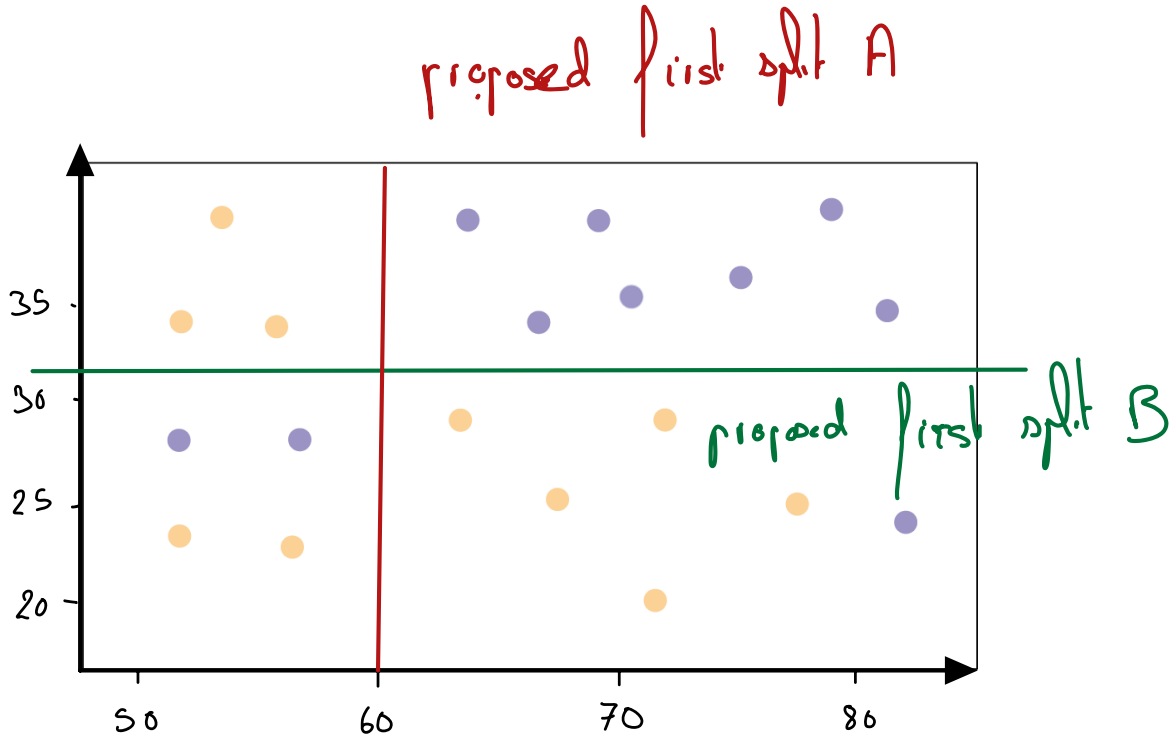
Training: how to build a tree.

Key question !

- What is a good tree?
- What do I need to choose?
- How to choose?

What do you think?

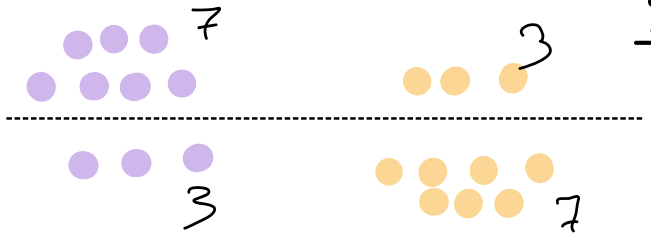
Building trees : CART



Algorithm at each step:

- try all possible values for all possible features
- compute some heterogeneity of the resulting groups
- pick the split that minimizes heterog.

prop B

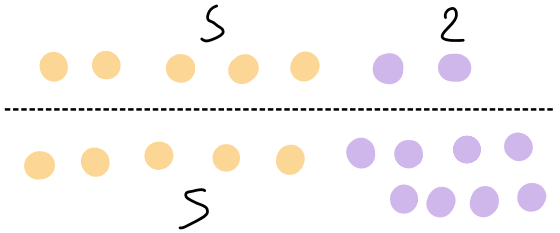


$$\frac{3}{10}\left(1-\frac{3}{10}\right) = \frac{21}{100}$$

→ heterogeneity ?

→ often several splits

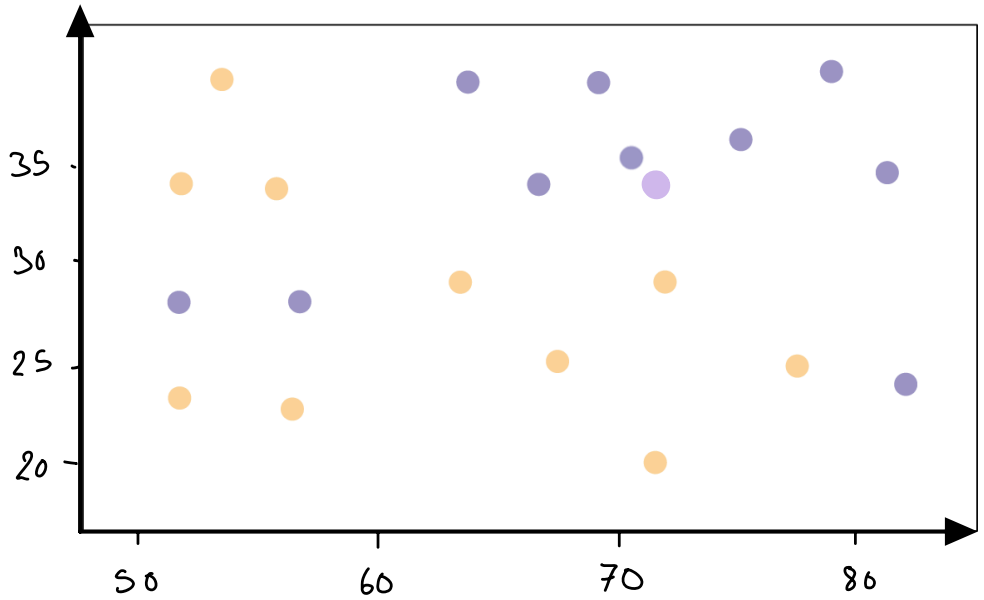
prop A



$$\rightarrow \frac{2}{7}\times\left(1-\frac{2}{7}\right)$$

$$\frac{5}{13}\left(1-\frac{5}{13}\right) = p_i(1-p_i)$$

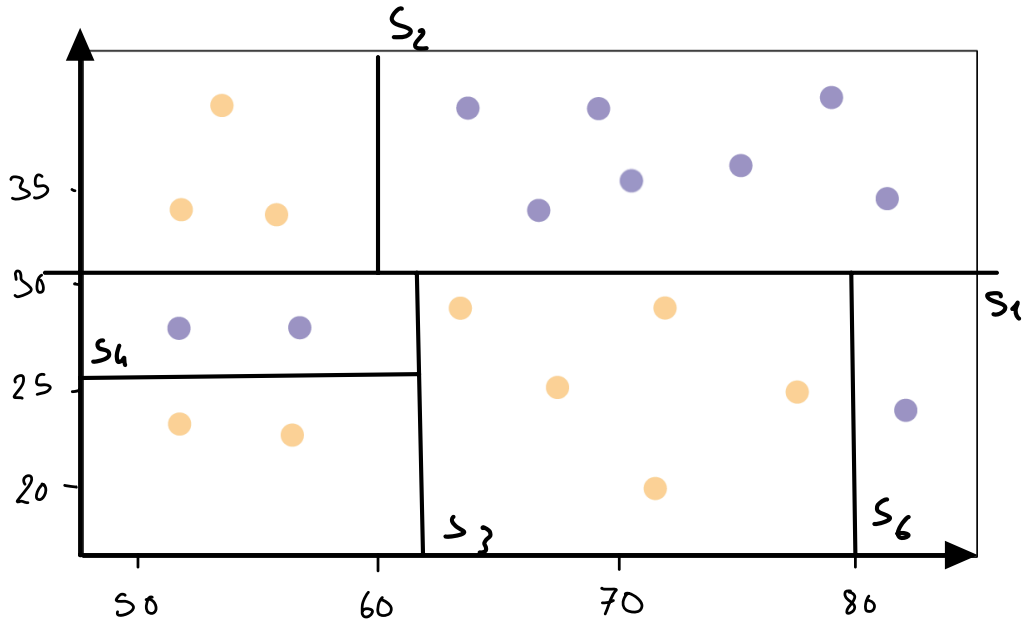
Building trees : CART



How to chose the best cut ?

- e.g., Given a possible cut, measure the reduction of "impurity":
  - in regression: mean-squared-error $\sum_{i \in C}(Y_i - \bar{Y}_C)^2$
  - in classification: Gini's impurity.
- Find dimension and threshold with optimal impurity reduction.
- Iterate until stopping criterion is met.

Gini's impurity



$$\frac{3}{10} \times \frac{7}{10}$$

$p_i \quad (1 - p_i)$

Gini impurity measures **how often a randomly chosen element from the set** would be **incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset**.

$$I_G(p) = \sum_{i=1}^{J} p_i(1 - p_i) = 1 - \sum_{i=1}^{J} {p_i}^2$$

For two classes, related to the variance if classified as Bernoulli r.v..

Group

$\sum_{i=1}^{J} p_i(1-p_i)$

$50\%\,(1-50\%) = \frac{1}{4} = 25\%$

$\frac{7}{10} \times \frac{3}{10} = 21\%$

$\frac{3}{10} \times \frac{1}{10} = 9\%$

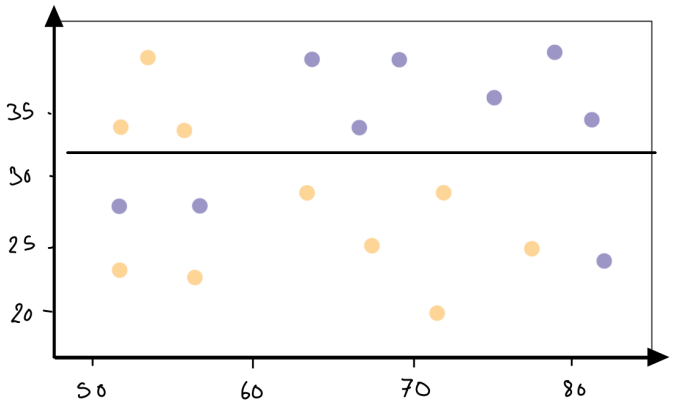
0%

$\times 2 \left(\bullet + \bullet \right)$

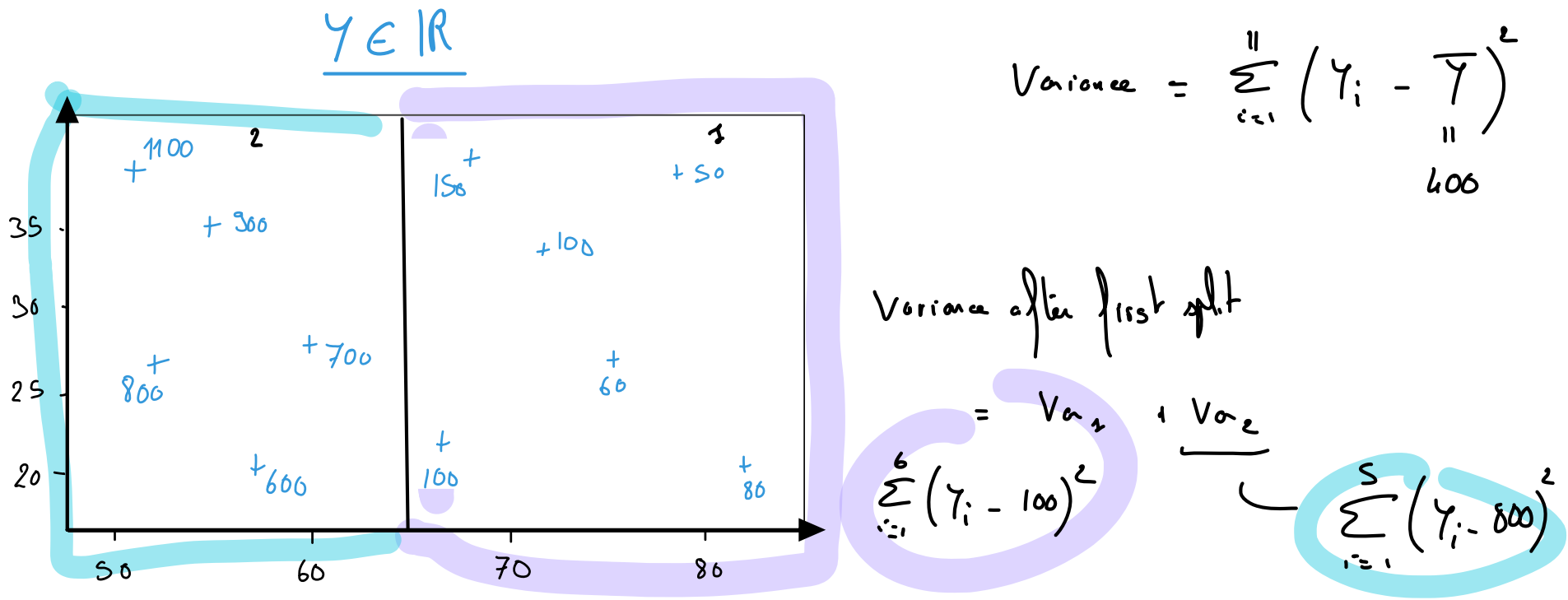
$3/10 \times 7/10$

$+ \; 3/10 \times 7/10$

$+ \; 6/10 \times 6/10 =$



Building trees : CART



How to chose the best cut ?

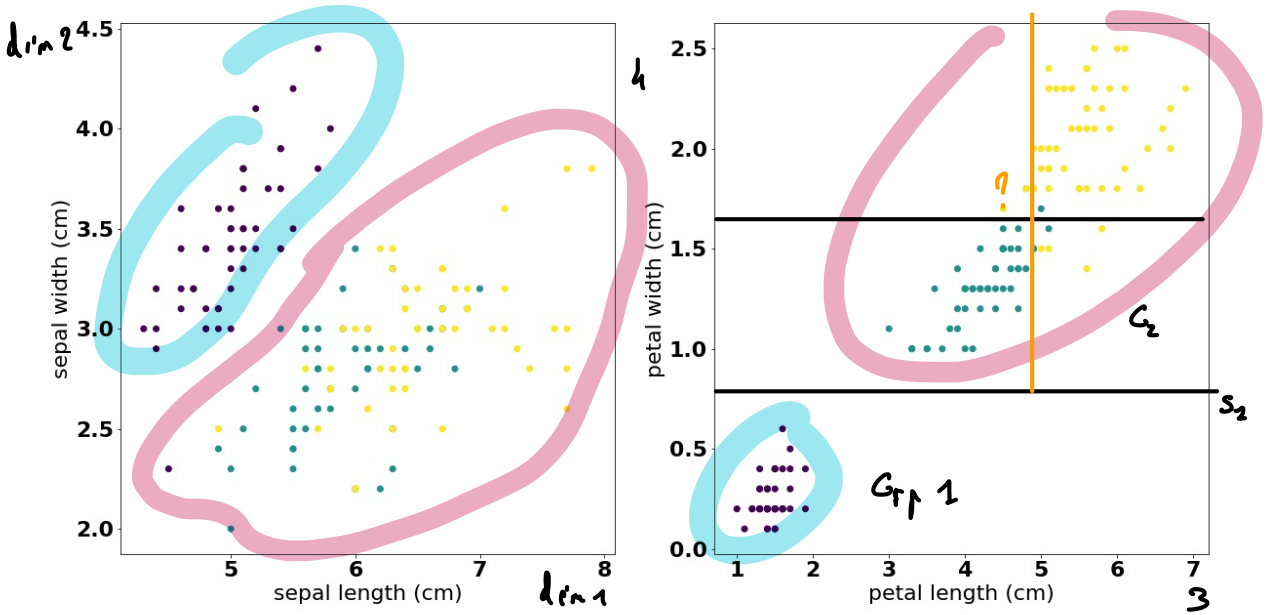
- e.g., Given a possible cut, measure the reduction of impurity :
  - in regression: mean-squared-error $\sum_{i \in C}(Y_i - \bar{Y}_C)^2$
  - in classification: Gini's impurity.
- Find dimension and threshold with optimal impurity reduction.
- Iterate until stopping criterion is met.

Example: Iris Dataset

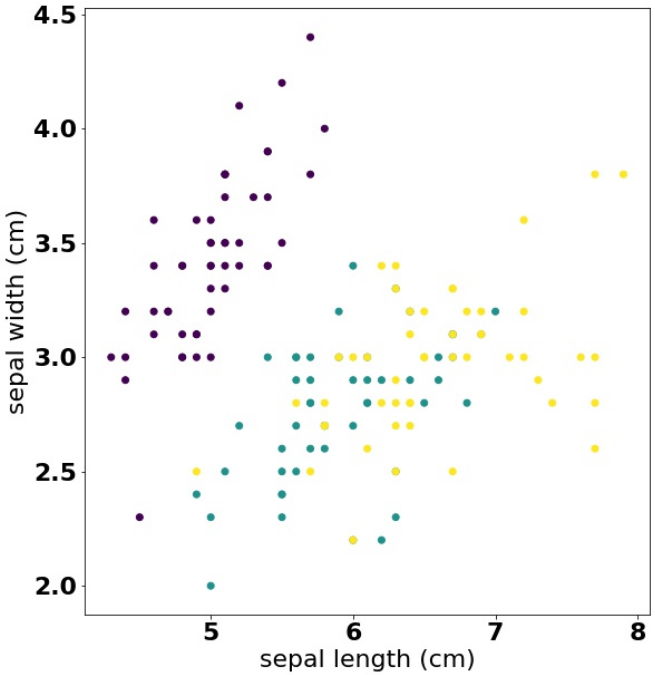
Figure: Iris: setosa, versiocolor, virginica

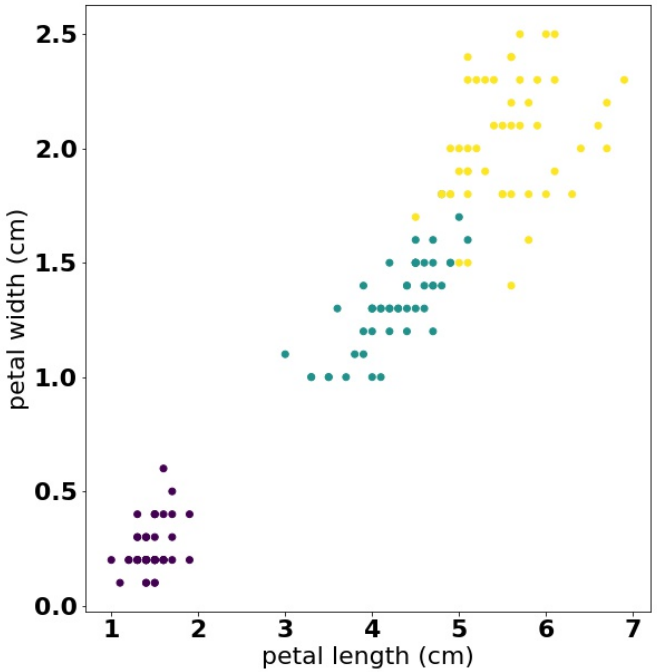
One of the most widely used toy dataset in classification:

- 3 classes : *['setosa', 'versicolor', 'virginica']*
- 50 points per class.
- 4 features: *['sepal length', 'sepal width', 'petal length', 'petal width']*



Exercise: Guess Classification tree for Iris Dataset





In pratice : Scikit Learn

Universal Python library for Machine Learning:

- Very easy to use
- Very well documented
- Open Source

`https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html`

Example: Iris Dataset

What do we need to specify ?

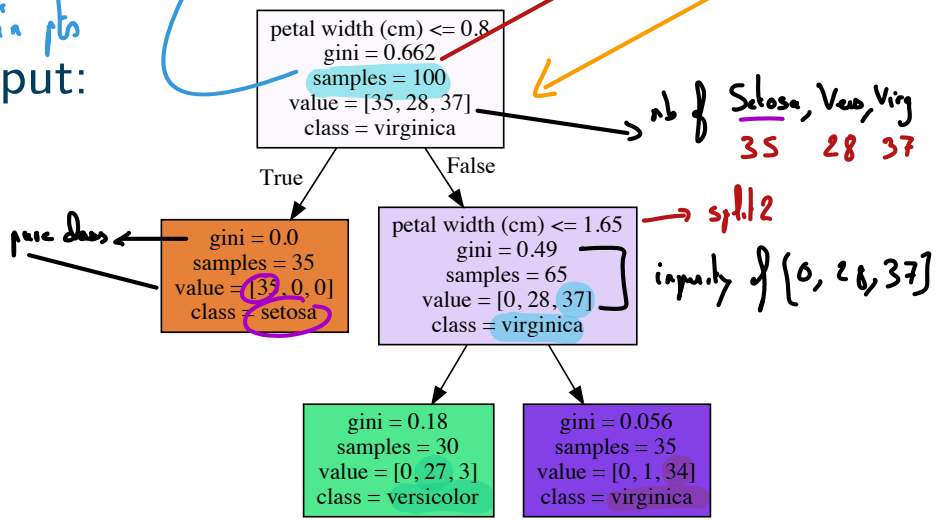
Hyperparameters : - depth

→ avoid overfitting

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split

# Choose the Learning algorithm
clf = DecisionTreeClassifier(max_depth = 2,
                             random_state=0)

#Load the dataset
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(
    iris.data, iris.target, test_size=0.33, random_state=43)

#Check accuracy
cross_val_score(clf, iris.data, iris.target, cv=10)

# Fit the model
clf.fit(X_train,y_train)

# Plot tree
plot_tree(clf, feature_names = fn,
                class_names=cn,
                filled = True)
plt.show()
```

66% × 150 pts = 100 train pts

Output:



before split gini impurity of 100 train pts

nb of Setosa, Vers, Virg 35 28 37

split 2

impurity of [0, 28, 37]

pure class

Example:

```python
# Check
X_test[0,:], iris.target_names[y_test[0]]
```

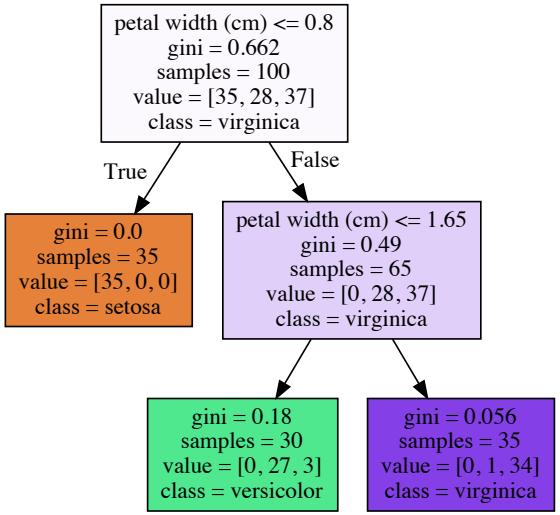
Returns:

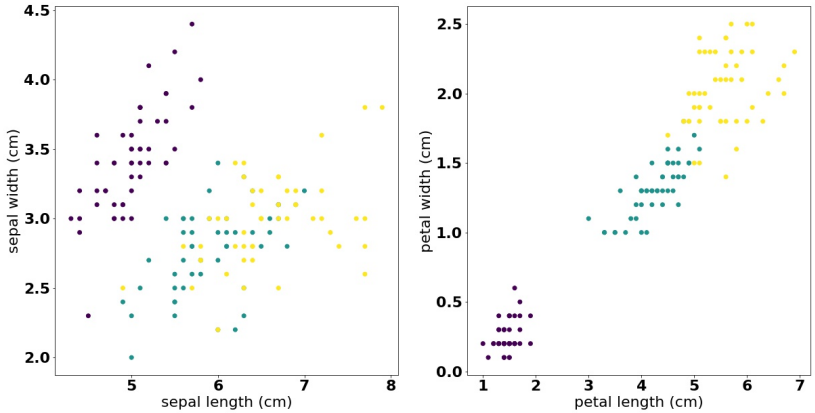
(array([4.8, 3.1, 1.6, 0.2]), 'setosa')

Code: https://colab.research.google.com/drive/1dXYIjacNAeASgl54vIWyWjMQlfgigSVA?usp=sharing#scrollTo=VeJHkJd7RsDw

Example: Iris Dataset

Output:





Example:

```
# Check
X_test[0,:], iris.target_names[y_test[0]]
```

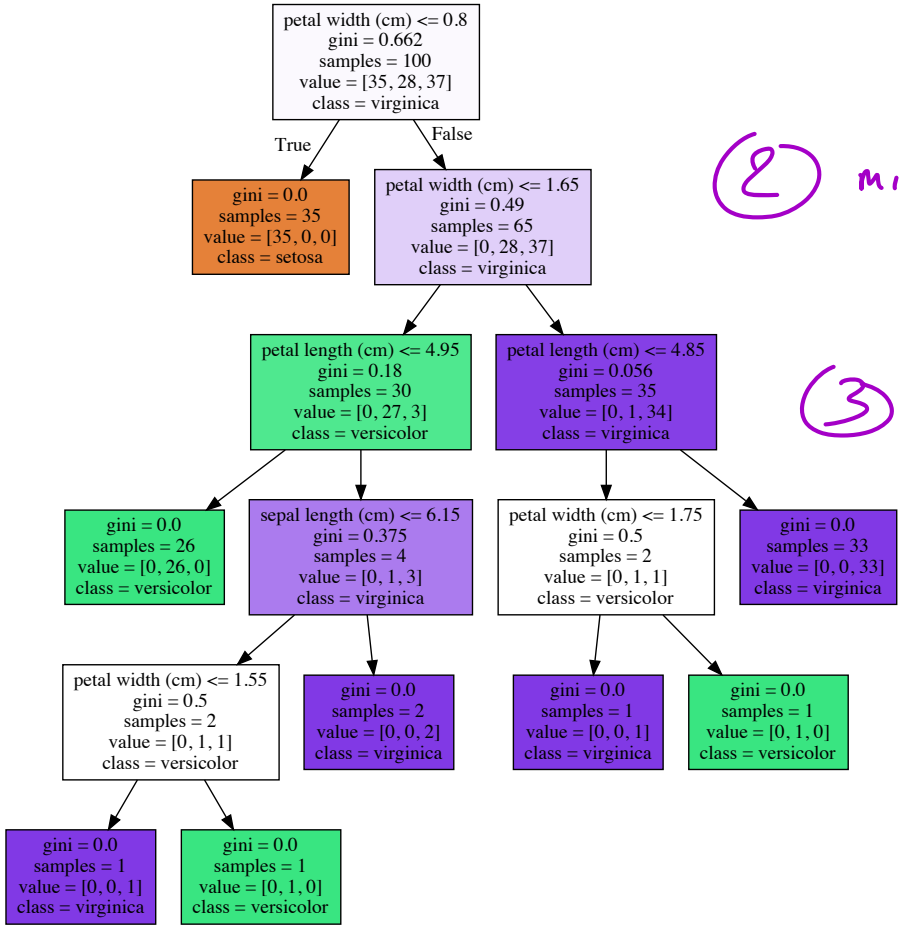
Returns:
(array([4.8, 3.1, 1.6, 0.2]), 'setosa')

Code: `https://colab.research.google.com/drive/1dXYIjacNAeASgl54vIWyWjMQlfgigSVA?usp=sharing#scrollTo=VeJHkJd7RsDw`

What if I use a deeper tree ? Or if I do not specify the depth in the model definition ?

Deeper trees

```
# Choose the Learning algorithm
clf = DecisionTreeClassifier(random_state=0)
```



(1) max_depth = 2

For any pb, at most 2 questions

(2) min_samples_leaf.

(3) min_impurity_decrease

- When does the algorithm stop then ?
- What do you think about it ?

Stopping / Pruning

Trees that achieve perfect classification may suffer from over-fitting. (see example in the lab)

To avoid this problem, we can reduce the complexity of the trees:

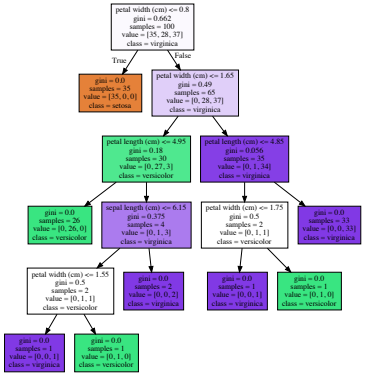
1. Stopping rules
   - Set a minimum number of samples inside each leaf.
   - Set a maximum depth.
2. Pruning
   - Reduced error pruning.
   - Cost complexity pruning.

} "taille" prune the complete tree.

→ other solution

Pruning : example of Cost complexity pruning.

We create a sequence of Trees by changing one subtree at each step into a leaf, until we get only the root: the subtree is chosen to minimize the error made.[1] Then the test error is evaluated along the sequence, to choose the optimal pruning.



[1] *more details here*

Cart: pros and cons

Pros:

- Simple to understand, interprete, visualize
- Implicitly perform features/variable selection
- can handle both categorical and numerical data*
- little effort for data preparation
- Non linear relationships do not affect performance
- Can work with large number of observations.

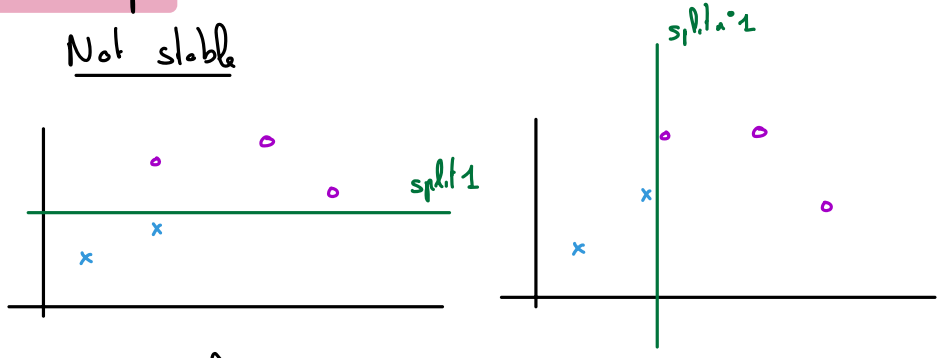
Cons:

1. Can create over-complex trees: overfitting
2. Unstability: small variations of data can generate completely different trees
3. Cannot guarantee global optimal tree
4. Biased if some classes dominate

Solution : Using Random Forests !

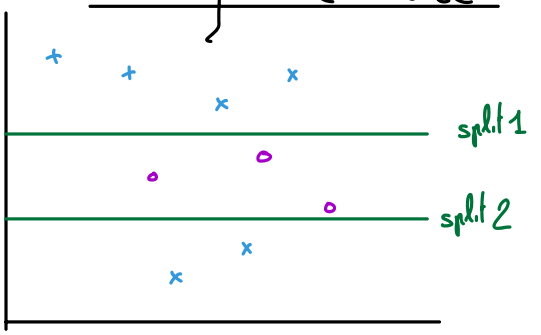
Wooclap.

Not stable



changing one data point may drastically change the decision tree.

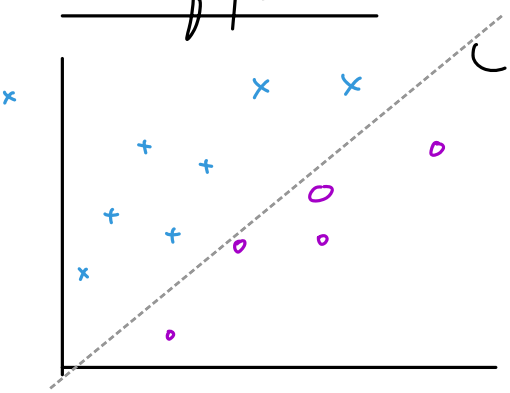
Same feature twice



} both use feature X_2 ✓

Rk: non linear separation!

Pairs of features

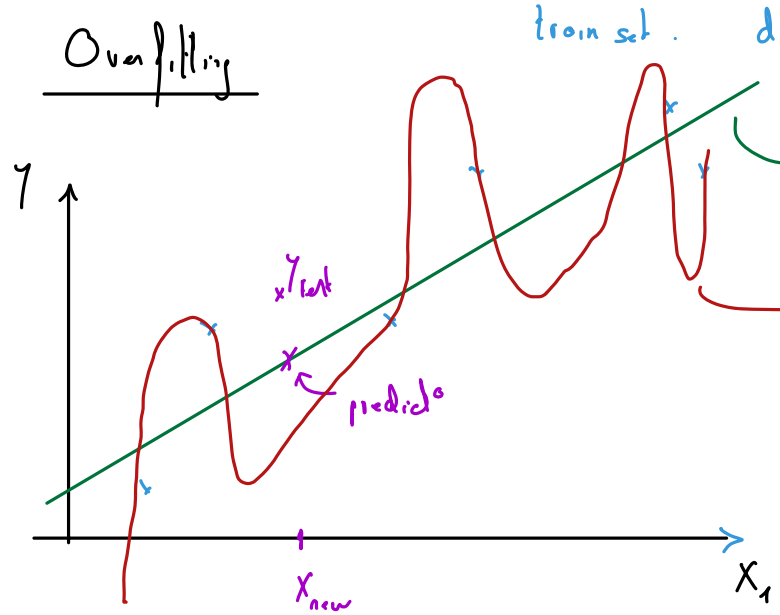


↳ logistic regression

in DT, we cannot use pairs of features!

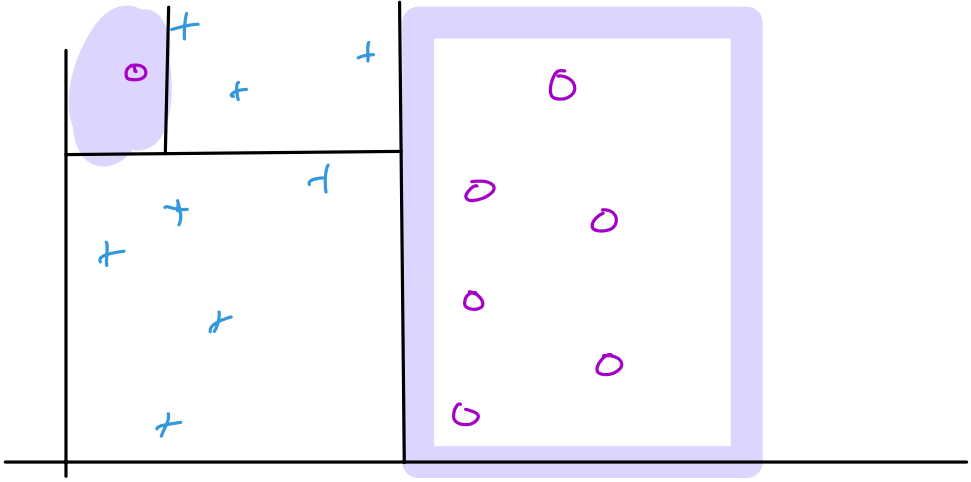
(complexity reasons)

Overfitting



no overfitting: ok on train ok on test

overfitting: perfect on train bad on test



Train / Validation / Test

Avoid overfitting

Outline

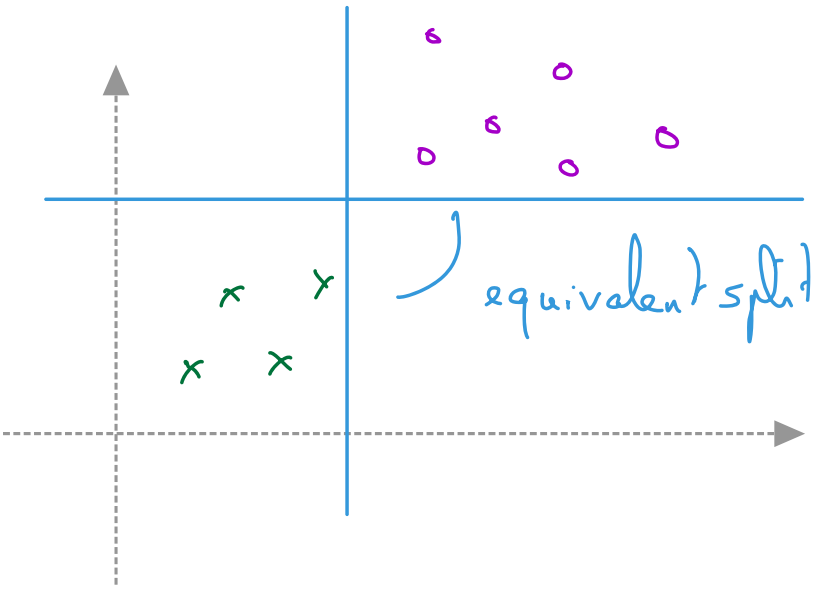
Random Forests

To put it in simple words: " random forests builds multiple decision trees and merge them together to get a more accurate and stable predictions" .

↬key idea: create variable trees + aggregate them

How to create variability ?

① At each step Test only some features



Tree 1 : spl 1, only uses X_1

Tree n°2 spl n°1 only uses X_2.

② Bootstraping :

each tree use a part of data to learn

Random Forests

To put it in simple words: random forests builds multiple decision trees and merge them together to get a more accurate and stable predictions .
↳key idea: create variable trees + aggregate them

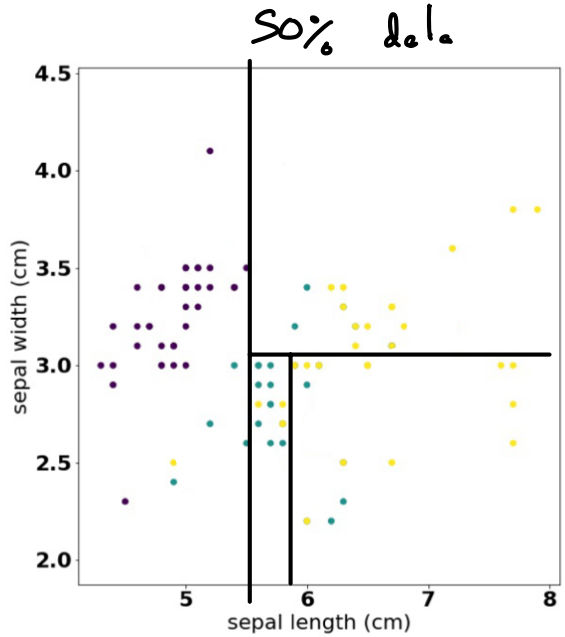
How to create variability ?

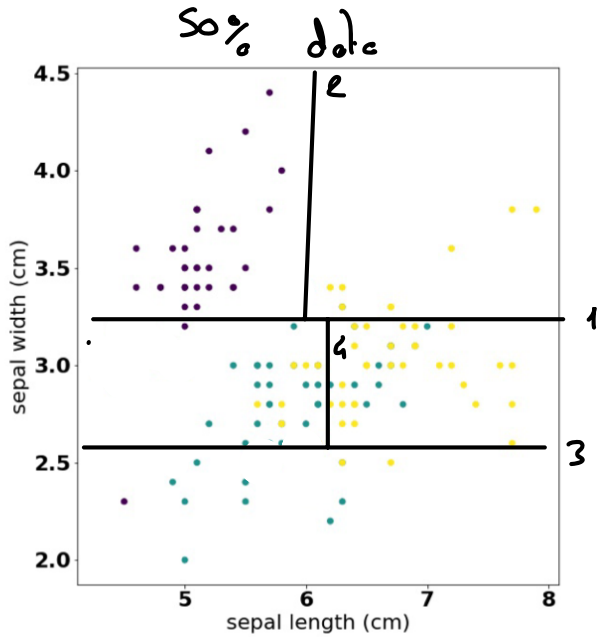
1. Instead of the most important features, search the best feature among a random subset (~~ntry~~). max_feature
2. Work on subsets of data (bootstrap=True).
   - More `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html`

```
# Creating and fitting a Random Forest
from sklearn.ensemble import RandomForestClassifier

plt.figure(figsize=(20,10))
clf = RandomForestClassifier(n_estimators = 100,
                             max_depth=3, bootstrap = True,
                             random_state =43)

clf.fit(X_train,y_train)
```

How to aggregate ? How to interpret a Random Forest ?

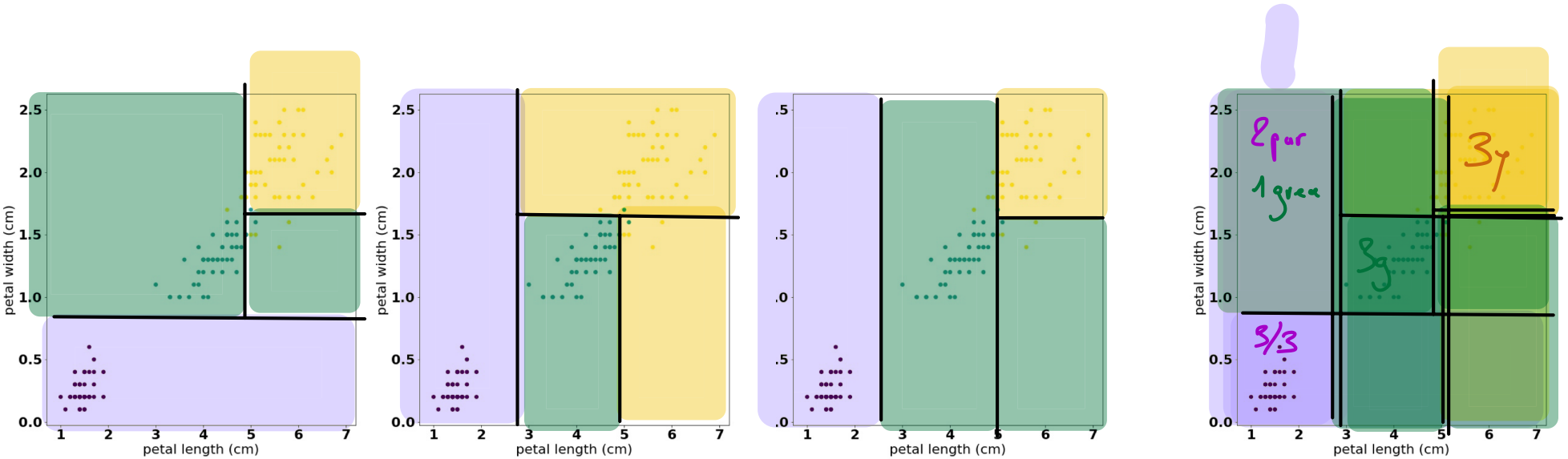
Bootstrap.


50% data
sepal width (cm)
sepal length (cm)
4.5
4.0
3.5
3.0
2.5
2.0
5
6
7
8


50% data
2
1
4
3
sepal width (cm)
sepal length (cm)
4.5
4.0
3.5
3.0
2.5
2.0
5
6
7
8

Aggregation

We using a voting or averaging process !



Feature importance

- how much tree nodes using a particular feature reduce impurity along the trees.
- suggests feature selection rule: drop out features with low importance to avoid overfitting.
- Attribute **feature_importance_** in RandomForestRegressor of Sklearn.

good enough

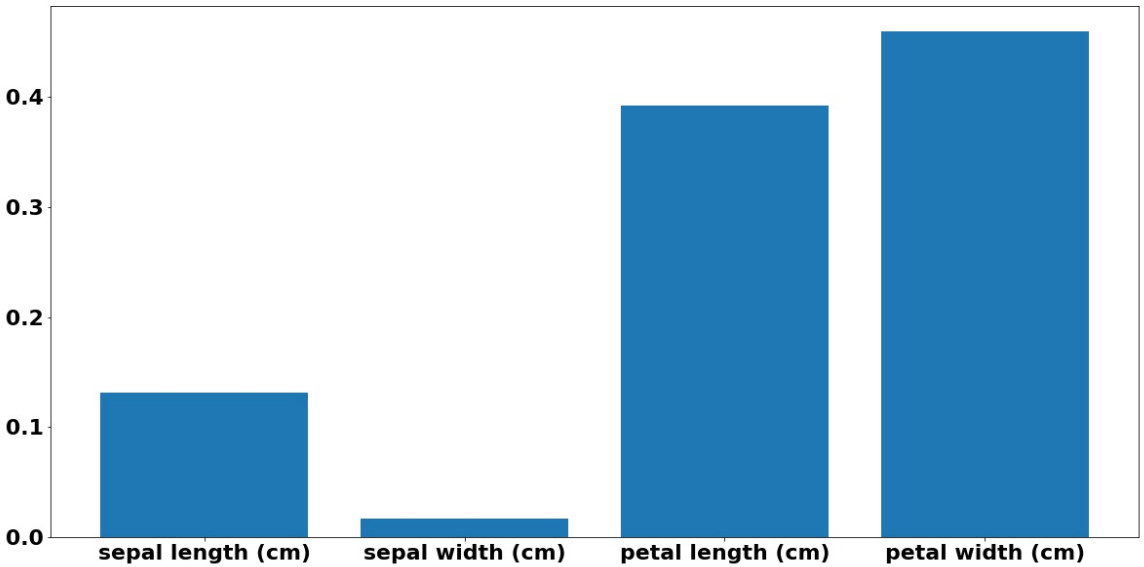


Figure: Feature importane output for a Random Forest Classifier in Python

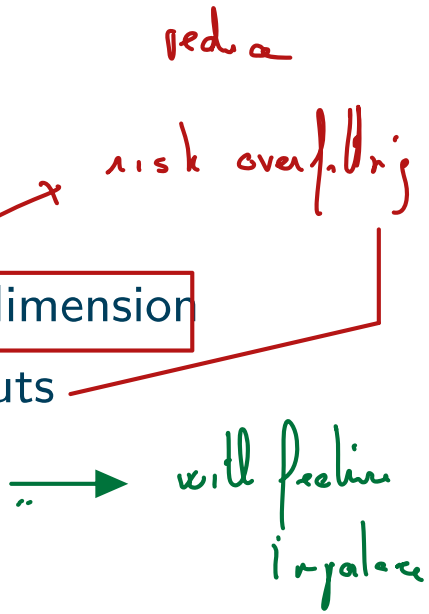
Comparison with decision trees

Cons:

- Less interpretable
- Slower to run

Pros:

- Better in higher dimension
- More stable outputs
- Feature selection



Improving predictions - more arguments.

1. **n_estimators** : number of trees in the forest
   - improves prediction / stability
   - slows down the algorithm
2. **max_features** : maximum number of features considered to split a node
3. **min_sample_leaf** : minimum number of samples that should remain in a leaf node.

RF pros and cons

RF Pros:

1. works for classification and regression
2. default hyperparameters often produce reasonable prediction -> easy to use.
3. avoid overfitting if some good trees in the forest and easy feature selection -> high dimensional pb.
4. hard to beat in performance.
5. Bonus : supports multiple outputs!

RF Cons:

1. ~~Fast to train~~ but slow to provide new predictions -> ineffective for real-time predictions ..
2. Good for prediction but bad for description.

What about Regression ? all the same!

Questions?

Boosting

Another import technique (very powerful !)

Main idea:

- Give more importance to difficult point iteratively ++
- Incrementally building an ensemble by training each new instance to emphasize the training instances previously mis-modeled.

Example: AdaBoost

⊖ we can't distribute computation as much (dependency / order between trees)

See: `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html`

Bonus: Multiple Outputs

- **Goal:** predict several outputs simultaneously
- **Solution:**
  - each leaf contains a value for each output
  - to chose the splits, we use a (weighted) average of the impurity of each output.

Bonus: complexity

Total complexity for one decision tree:

Worst case:

$$O(n^2 d)$$

Balanced case:

$$O(nd)$$

Tips

- Decision trees tend to overfit: limit the depth or use `min_samples_leaf` (5 is a good initial pick)
- Visualize the tree with a small depth first
- Balance your dataset: trees tend to be biased towards dominating class.

Conclusion

1. Trees are one of the simplest method (the most intuitive / human like)
2. Random Forests give excellent results in many applications.

Lab :

- Example on a synthetic dataset of time series
- Application to inflation prediction in Brazil.