

# ML, Decision Trees and Random Forests Data science Certificate

Aymeric DIEULEVEUT

May 2024

1 Introduction to Supervised Learning

2 Decision Trees

3 Random Forests

# Outline

1 Introduction to Supervised Learning

2 Decision Trees

3 Random Forests

# A Reminder on Supervised Learning

*d covariates  
features.*

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

## 3 Examples:

- Iris Dataset:
- MNIST:
- House price prediction

# Example 1: House price Prediction - Regression

**Goal:** observe the features for a house, predict its price.

House price Prediction dataset  $Y \in \mathbb{R}_+$

- Output : house value

$\rightarrow Y \in \mathbb{R} \rightarrow$  regression task

- 1500 examples

$\rightarrow n = 1500$

- 81 features: Construction year, Neighborhood, Surface, Floor, Balcony: both quantitative and qualitative, some ordinal variables.

$\rightarrow X \in \mathbb{R}^{81}$ .

$$X_{\text{Amenic-flat}} = (1970, 13^e, 60, 4, \text{Yes}, \dots)$$

SM2 goal.

\* Learn the link  $X \rightarrow Y$ .

\* From  $n$  examples

\* Predict well on a new  $\underline{X}_{\text{test}}$

## Example 2: Iris Dataset - Classification



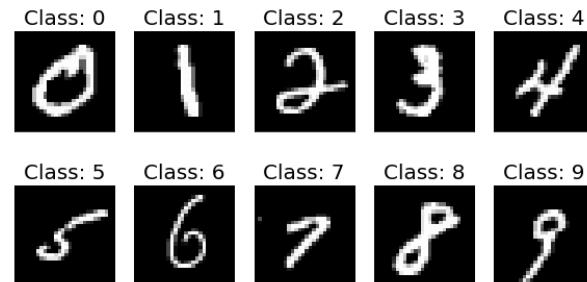
Figure: Iris: setosa, versicolor, virginica

**Goal:** identify flower species based on petal/sepal characteristics.

Iris Dataset:

- 3 classes : `['setosa', 'versicolor', 'virginica']`  
→  $Y \in \{0, 1, 2\}$  encoding each class.
- 50 points per class  
→  $n = 150$
- 4 features: `['sepal length', 'sepal width', 'petal length', 'petal width']`  
 $X \in \mathbb{R}^4$ , →  $d = 4$

# Example: MNIST Dataset



Goal: recognize an object on an image, e.g., a handwritten digit on its image.

Digit recognition from an image:

- 10 classes :

$$Y \in \{0, 1, 2, \dots, 9\}$$

- 60k images, (50k train, 10k test), balanced

$$n = 60000$$

- Observation: pixels: 28\*28 pixels per image (gray-scale).

$$X \in \mathbb{R}^{784}, \quad d = 784 \text{ features.}$$

# Summary of those three datasets

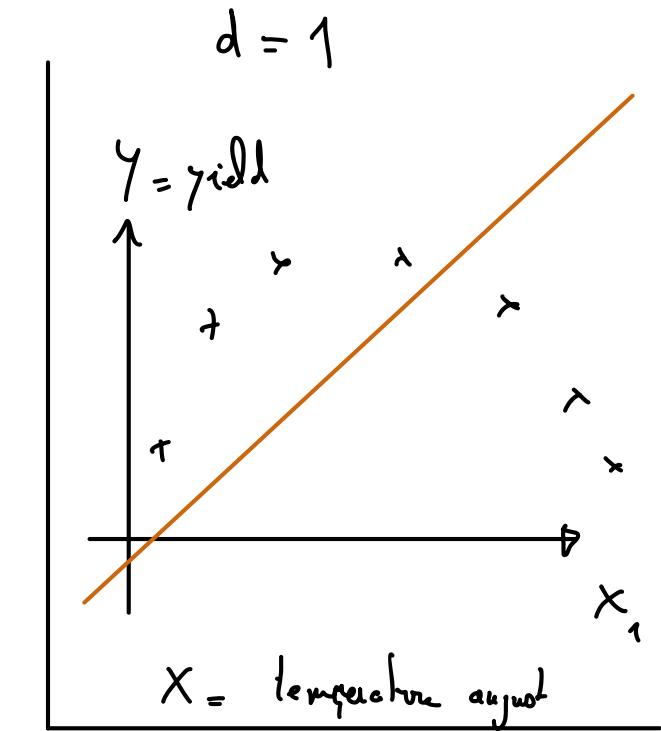
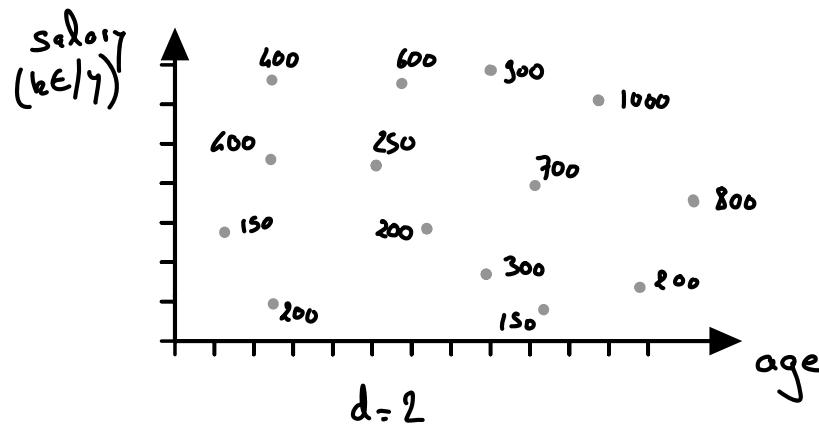
	Iris	MNIST	House price prediction
Number of examples $n$	150	60k	1500
Number of the features $d$	4	784	81
Type of the features	numerical values	numerical	categorical numerical (quantitative) ordered
Space $\mathcal{V}$	$\{v_0, v_1, v_2, \dots, v_n\}$	$\{0, \dots, 9\}$	$\mathbb{R}$
Task	C	C	R

categories → encoding.

# Toy datasets

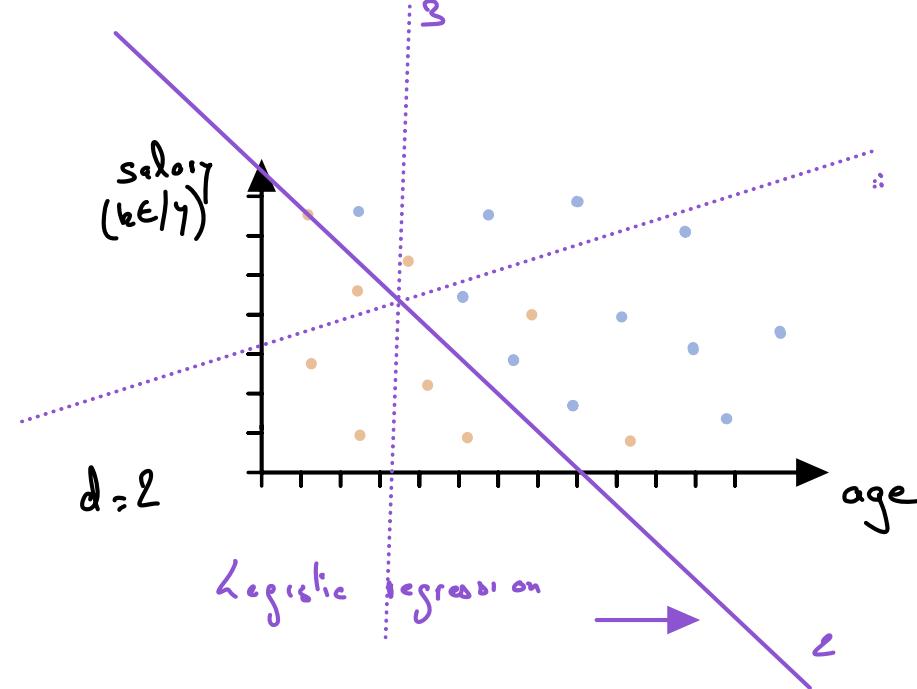
- In regression:

$$X = (\text{age}, \text{salary})$$



- In classification:

$$X = (\text{age}, \text{salary})$$



# More examples



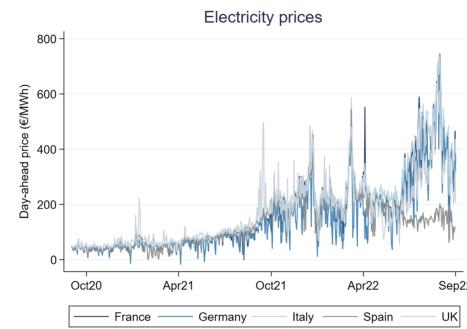
Spam detection



Semantic image segmentation



Sentiment classification



Electricity price forecasting

# Supervised Learning

## Supervised Learning

- Attributes / features / characteristics / input:

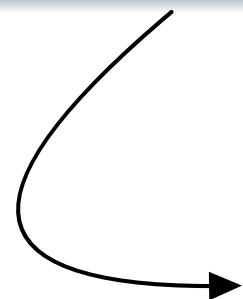
$$X \in \mathcal{X} = \mathbb{R}^d$$

- Output / Responses / label :

- $Y \in \{-1, 1\}$  - binary classification,
- $Y \in \{0, 1, 2, \dots, K-1\}$  - multi-class classification,
- $Y \in \mathbb{R}$  - regression.

- Phenomenon distribution  $(X, Y) \sim \mathbb{P}$ .



We access  **independent** **identically distributed** **iid** data

Critical assumption in ML: we learn to predict on a distrib'

# Supervised Learning

## Supervised Learning

- Attributes / features / characteristics / input:

$$X \in \mathcal{X} = \mathbb{R}^d$$

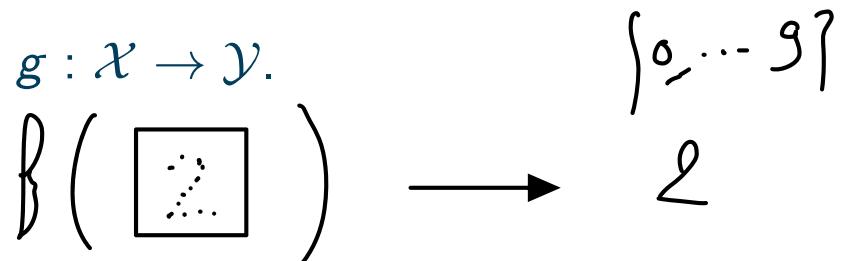
- Output / Responses / label :

- $Y \in \{-1, 1\}$  - binary classification,
  - $Y \in \{0, 1, 2, \dots, K-1\}$  - multi-class classification,
  - $Y \in \mathbb{R}$  - regression.

- Phenomenon distribution  $(X, Y) \sim \mathbb{P}$ .

- A classifier or predictor is a measurable function  $g : \mathcal{X} \rightarrow \mathcal{Y}$ .

How to **quantify the quality** of a predictor?



→ introduce a risk function.

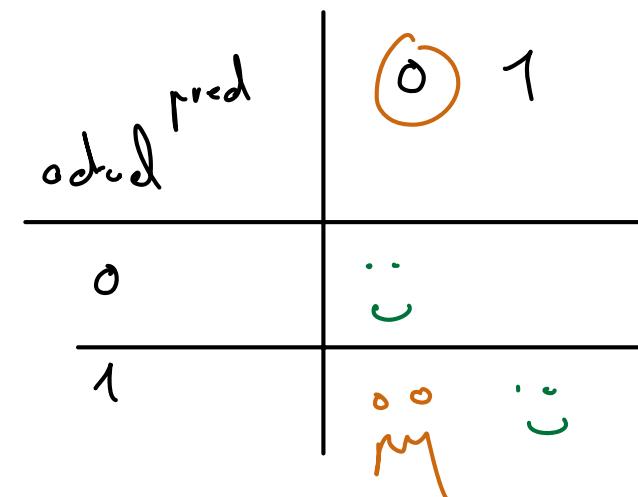
# Loss function, Generalization Risk and data-dependent predictors

## Loss Function    output predicted

- Loss function:  $\ell(y, g(x))$  quantifies the quality of the prediction  $g(x)$  of  $y$ , e.g.:
  - ▶ 0-1 Loss (classification):  $\ell(y, g(x)) = 1_{y \neq g(x)}$ ,  $y \in \mathcal{Y} = \{-1, 1\}$
  - ▶ Quadratic Loss (regression):  $\ell(y, g(x)) = (y - g(x))^2$ ,  $y \in \mathbb{R}$

choice of loss : complicated

\* bicycle example



\* what do we want ?

Alignment



# Loss function, Generalization Risk and data-dependent predictors

## Loss Function

- Loss function:  $\ell(y, g(x))$  quantifies the quality of the prediction  $g(x)$  of  $y$ , e.g.:
  - ▶ 0-1 Loss (classification):  $\ell(y, g(x)) = 1_{y \neq g(x)}$ ,  $y \in \mathcal{Y} = \{-1, 1\}$
  - ▶ Quadratic Loss (regression):  $\ell(y, g(x)) = \|y - g(x)\|^2$ ,  $y \in \mathbb{R}^d$

Risk of a Decision Rule = average loss

- Risk:  $R(g) = \mathbb{E}[\ell(Y, g(x))] = \int \ell(y, g(x)) d\mathbb{P}(x, y)$ , e.g.:
  - ▶ 0-1 Risk (classification):  $R(g) = \mathbb{P}(Y \neq g(x))$
  - ▶ Quadratic Risk (regression):  $R(g) = \mathbb{E}[|Y - g(x)|^2]$

↪  $R$  is also referred to as *Generalization* risk, or *True* risk.



$\mathbb{E}$  → average for a new point  $x, y \sim \mathbb{P}$   
 $\mathbb{P}$  → probab. that ( )  
distribution  
of the phenomenon

Generalization risk "average loss on a new point  $(x, y)$  for predicting"

# Loss function, Generalization Risk and data-dependent predictors

## Loss Function

- Loss function:  $\ell(y, g(x))$  quantifies the quality of the prediction  $g(x)$  of  $y$ , e.g.:
  - ▶ 0-1 Loss (classification):  $\ell(y, g(x)) = 1_{y \neq g(x)}$ ,  $y \in \mathcal{Y} = \{-1, 1\}$
  - ▶ Quadratic Loss (regression):  $\ell(y, g(x)) = \|y - g(x)\|^2$ ,  $y \in \mathbb{R}^d$

Risk of a Decision Rule = average loss

- Risk:  $R(g) = \mathbb{E}[\ell(Y, g(x))] = \int \ell(y, g(x))d\mathbb{P}(x, y)$ , e.g.:
    - ▶ 0-1 Risk (classification):  $R(g) = \mathbb{P}(Y \neq g(x))$
    - ▶ Quadratic Risk (regression):  $R(g) = \mathbb{E}[|Y - g(x)|^2]$
- ↪  $R$  is also referred to as *Generalization* risk, or *True* risk.

⚠ The distribution  $\mathbb{P}$  is unknown.

## Data-dependent predictors

- Training set:  $\mathcal{D}_n = \{(X_1, Y_1), \dots, (X_n, Y_n)\}$  (i.i.d.  $\sim \mathbb{P}$ )
- Goal: build a **data dependent predictor / classifier**  $\hat{g}_n$  based on the training data
- That has a **minimal generalization risk**  $R(\hat{g}_n)$ .

$R(\hat{g}_n)$  = average loss on a “new point”  $(X, Y) \sim \mathbb{P}$ , independent from  $D_n$

↪ Our goal is to predict well on inputs/outputs pairs that we have not seen in the dataset, i.e. *generalize* well.



# Supervised Learning: Summary

## Levels of understanding:

- # Supervised Learning: Framework Summary

- Training set:  $D_n = \{(x_1, Y_1), \dots, (x_n, Y_n)\}$  (*i.i.d.*  $\sim \mathbb{P}$ )
  - Decision rule (classifier, predictor):  $g : \mathcal{X} \rightarrow \mathcal{Y}$  ~~measurable~~
  - Loss:  $\ell(Y, g(x))$
  - Risk:  $R(g) = \mathbb{E}[\ell(Y, g(x))] = \int \ell(y, x) d\mathbb{P}(x, y)$

- \* generic pipeline  
(algos are Black box)
- \* opening the block box  
for each algo.

## Numerous methods:

M. libra

## ① In regression:

- ▶ linear, polynomial, (kernel regression)
  - ▶ (regularized methods (Lasso, Ridge))

② In classification:  $\hookleftarrow$  anti over fitting

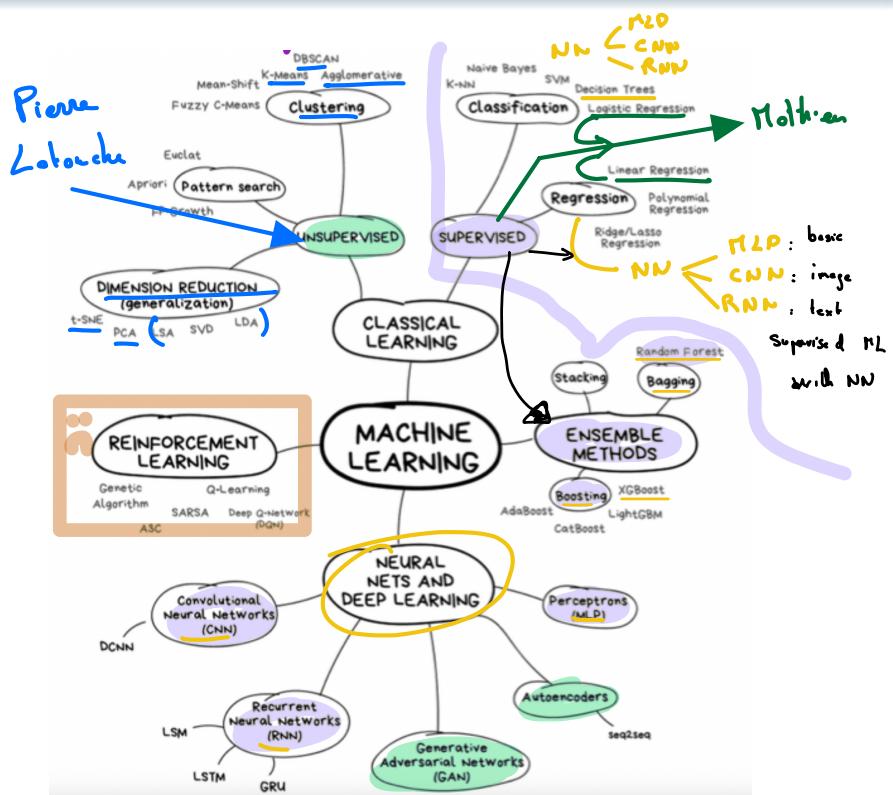
- Logistic regression
  - SVM → also linear classifiers

### ③ For both

- ▶ Nearest neighbors
  - ▶ Random forests
  - ▶ Neural networks
  - ▶ etc.

Super  
unsuper

Reinhardt



## Figure:

2 levels of understanding:

- 1 \* generic pipeline  
(algs are Black box)
- 2 opening the block box  
for each algo.



how to do ML without knowing  
exactly what algo do?

Learn parameters: = automatically | computed | optimized | by the

computer to perform well on the Train set

→ User does nothing

Discussion knowing methods (obj 2)

enables to know which ones (angle)

- \* Speed
- \* Interpretability
- \* Performance

The ordering  $S/I/P$  depends on the task.

High freq loadings       $S \cdot P$

Treatment predictn ;       $P - I$

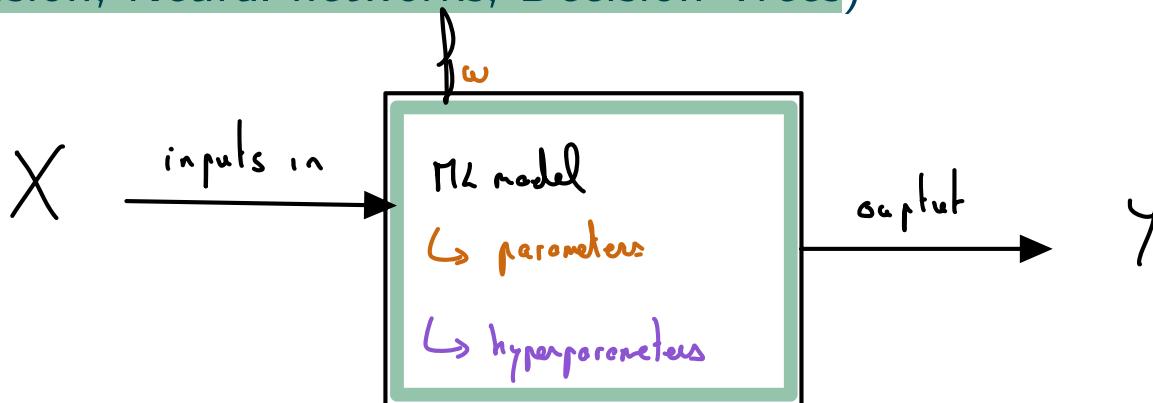
credit assignt (regulated env)       $P - I$

# Learning Pipeline

1

## Training

Learn the dependency between input and output. Each model (LR, Logistic regression, Neural networks, Decision Trees)



parameters :

linear regression

House price predict

$$X = (\text{surf}, \text{surf bdc})$$

$$f_{\omega}(X) = X_1 \underbrace{\omega_1}_{\substack{10 \text{€/m}^2 \\ S_1}} + X_2 \underbrace{\omega_2}_{\substack{3 \text{€/m}^2 \\ \text{parameter}}} \quad \text{on parameter dependence}$$

Fitting phase: give Train dataset  $D_{Tr}$   
"fitting phase"  
→ min risk  $\int_{\omega} \text{on } D_{Tr}$   
computer finds the "best/good" parameters  $\omega$  w.r.t. Train set

hyperparameters  
are chosen by  
the user:

- \* nb of trees in forest
- \* architecture
- \* model type

Solution:  
\* expert knowledge  
\* "cross validated"

"try many pick  
the best"

# Fitting a method

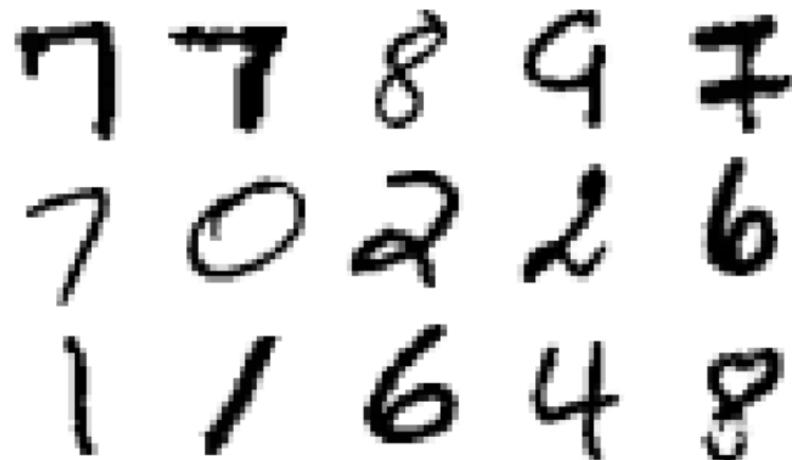
Demo url (Lab): [https://colab.research.google.com/github/adieulev/Certificate-DS-2023/blob/main/Labs/MWE\\_ML\\_init\\_student\\_version.ipynb](https://colab.research.google.com/github/adieulev/Certificate-DS-2023/blob/main/Labs/MWE_ML_init_student_version.ipynb)

```
1 import numpy as np
2 from sklearn.datasets import fetch_openml
3 from sklearn.model_selection import train_test_split
4 from sklearn.preprocessing import StandardScaler
5 from sklearn.metrics import accuracy_score
6
7 # Fetching the MNIST dataset
8 mnist = fetch_openml('mnist_784', version=1)
9 X, y = mnist["data"], mnist["target"]
10
11 # Setting aside 10% of the data for test size 10%
12 X_sample, _, y_sample, _ = train_test_split(X, y, test_size=0.9, random_state=42)
13
14 X_train, X_test, y_train, y_test = train_test_split(X_sample, y_sample, test_size=0.2, random_state=42)
15
16 # Scaling the data
17 scaler = StandardScaler()
18 X_train_scaled = scaler.fit_transform(X_train.astype(np.float64))
19 X_test_scaled = scaler.transform(X_test.astype(np.float64))
20
21 import warnings
22 warnings.filterwarnings('ignore') not recommended
```

Annotations on the code:

- Line 11: A red bracket groups lines 12-13 with the text "Setting aside 10% of the data ~~for test~~ ~~size 10%~~".
- Line 16: A red bracket groups lines 17-19 with the text "# Scaling the data".
- Line 22: The text "not recommended" is written in orange at the end of the line.

Figure: Importing data



# Fitting

## Fitting a method

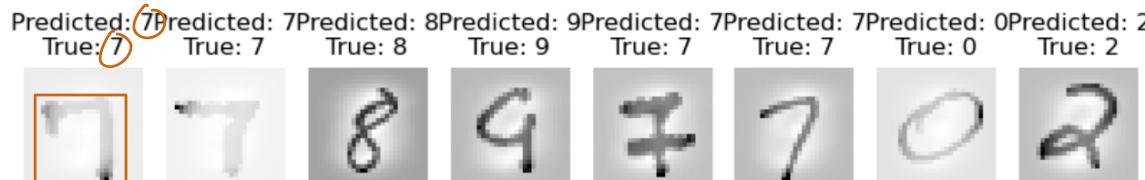
```
Entrée [37]: 1 from sklearn.tree import DecisionTreeClassifier  
2  
3 dt_clf = DecisionTreeClassifier()  
4 dt_clf.fit(X_train_scaled, y_train)
```

choose by parameters !!  
here : None !

```
Out[37]: ▾ DecisionTreeClassifier  
DecisionTreeClassifier(random_state=42)
```

"train on train data"  
= "fit" = "learn w"

```
Entrée [38]: 1 predictions = dt_clf.predict(X_train_scaled[:16])  
2 fig, axes = plt.subplots(1, 8, figsize=(10, 2)) # Adjust the figure size as needed  
3 for i, ax in enumerate(axes):  
4     ax.imshow(X_train_scaled[i].reshape(28, 28), cmap=plt.cm.gray_r, interpolation='nearest')  
5     ax.set_title(f"Predicted: {predictions[i]}\nTrue: {y_train.iloc[i]}")  
6     ax.axis('off')  
7 plt.show()
```



```
Entrée [39]: 1 from sklearn.ensemble import RandomForestClassifier  
2  
3 rf_clf = RandomForestClassifier(n_estimators=10, random_state=42)  
4 rf_clf.fit(X_train_scaled, y_train)
```

```
Out[39]: ▾ RandomForestClassifier  
RandomForestClassifier(n_estimators=10, random_state=42)
```

hyperparam : nb of trees.

```
Entrée [40]: 1 from sklearn.linear_model import LogisticRegression  
2  
3 log_reg = LogisticRegression(multi_class="multinomial", solver="lbfgs", max_iter=100, random_state=42)  
4 log_reg.fit(X_train_scaled, y_train)
```

```
Out[40]: ▾ LogisticRegression  
LogisticRegression(multi_class='multinomial', random_state=42)
```

loss solver iter

Figure: Fitting 3 methods and predicting

What is true about SML

① Supervised Machine Learning does not have inputs X

② Supervised Machine Learning can be combined with unsupervised in some contexts ✓

③ Supervised Machine Learning does not have outputs X

④ The goal is to predict well on new unseen points ✓✓

⑤ It is necessary to perform well on the train set for that

→ most of the time, yes ✓ (99%)

∴ It is sufficient to perform well on the train X

⑥ The goal is to predict well on the train points X

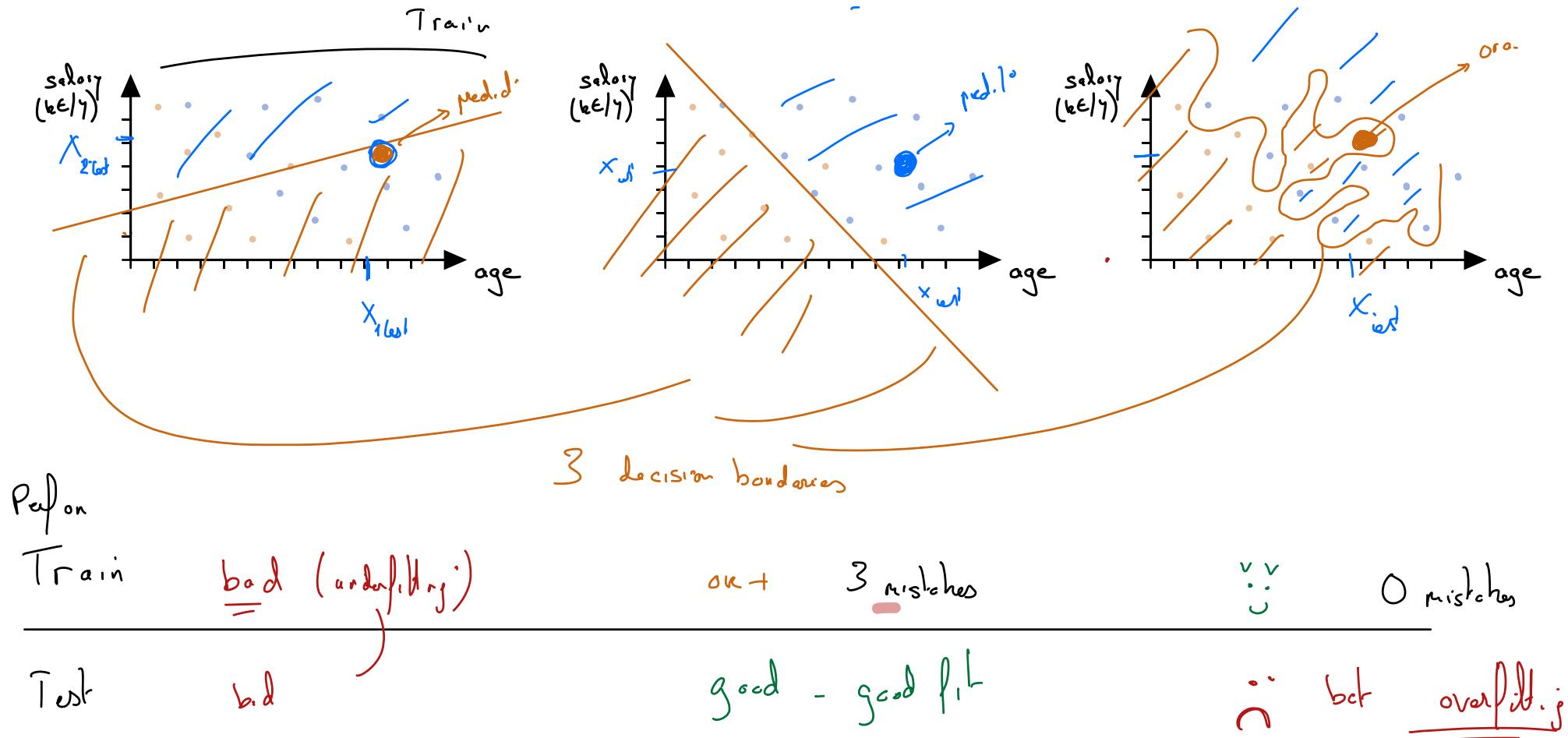
⑦ The balance interpretability/speed/performance depends on the method ✓

⑧ I assume my training data to be representative (iid) of a phenomenon with distribution P ✓

## Caveat: Overfitting - Toy datasets

⚠ The performance on the training set may not reflect the ability to predict on a future point !!

💡 Our Goal is to perform well on unseen outputs!!

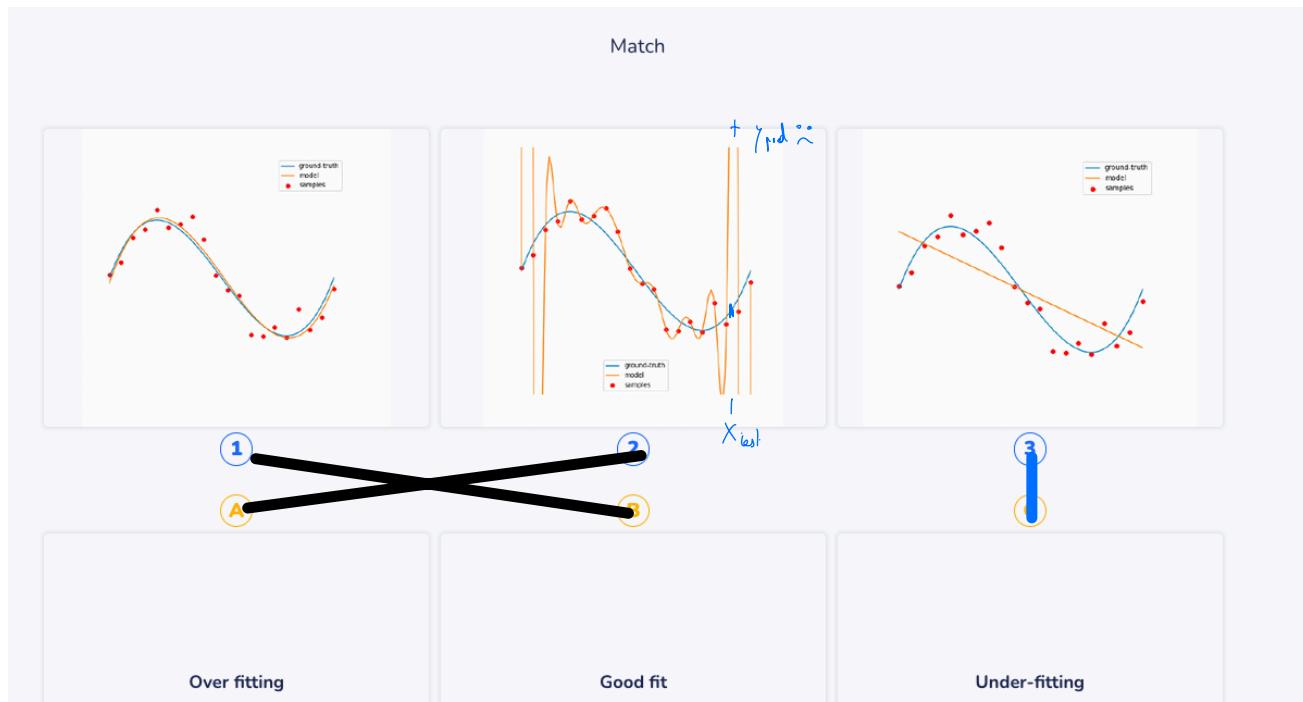


💡 Need to focus on validation/test performance

## Caveat: Overfitting - Toy datasets

⚠ The performance on the training set may not reflect the ability to predict on a future point !!

💡 Our Goal is to perform well on unseen outputs!!



overfitting

My performance on  
the test is much  
worse than on  
the train

to choose a method, evaluate "test era"  
appropriate

💡 Need to focus on validation/test performance

Underfitting : I do not even learn  
on the train

# Underfitting and Overfitting depending on the choice of $\mathcal{C}$

---

$\mathcal{C} =$

$\hat{R}_n(\theta_n^*)$

$R(\theta_n^*)$

Problem

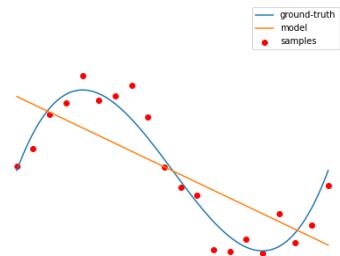
---

Dim. of  $\Theta$

---

# Underfitting and Overfitting depending on the choice of $\mathcal{C}$

---



---

$\mathcal{C} =$  Affine functions

$\hat{R}_n(\theta_n^*)$  high

$R(\theta_n^*)$  high

Problem Underfitting

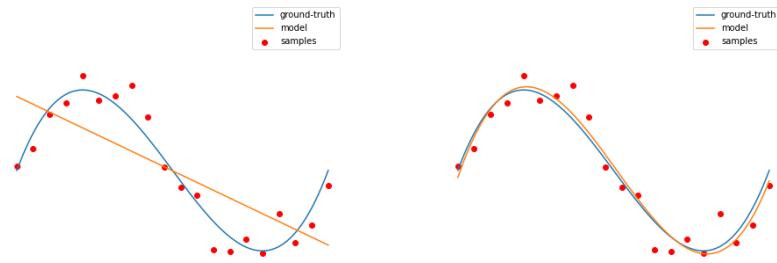
---

Dim. of  $\Theta$  2

---

# Underfitting and Overfitting depending on the choice of $\mathcal{C}$

---



---

$\mathcal{C} =$	Affine functions	$\mathbb{R}_3[X]$
-----------------	------------------	-------------------

$\hat{R}_n(\theta_n^*)$	high	low
-------------------------	------	-----

$R(\theta_n^*)$	high	low
-----------------	------	-----

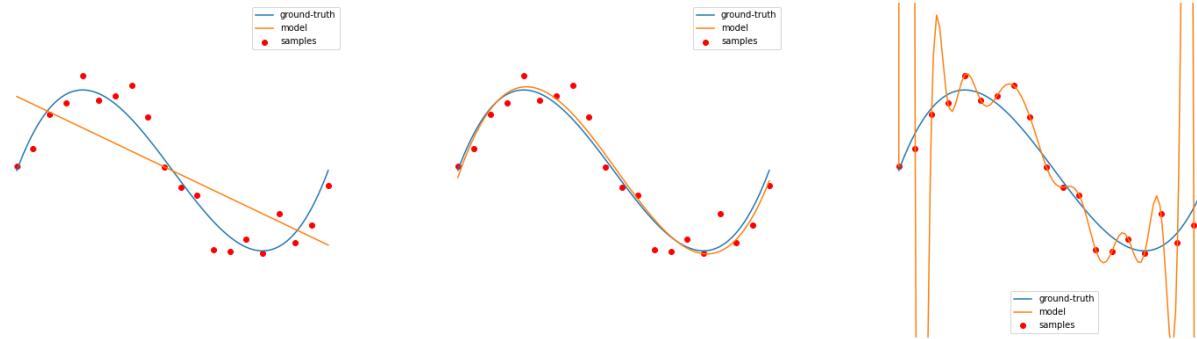
Problem	Underfitting	All good
---------	--------------	----------

---

Dim. of $\Theta$	2	4
------------------	---	---

# Underfitting and Overfitting depending on the choice of $\mathcal{C}$

---

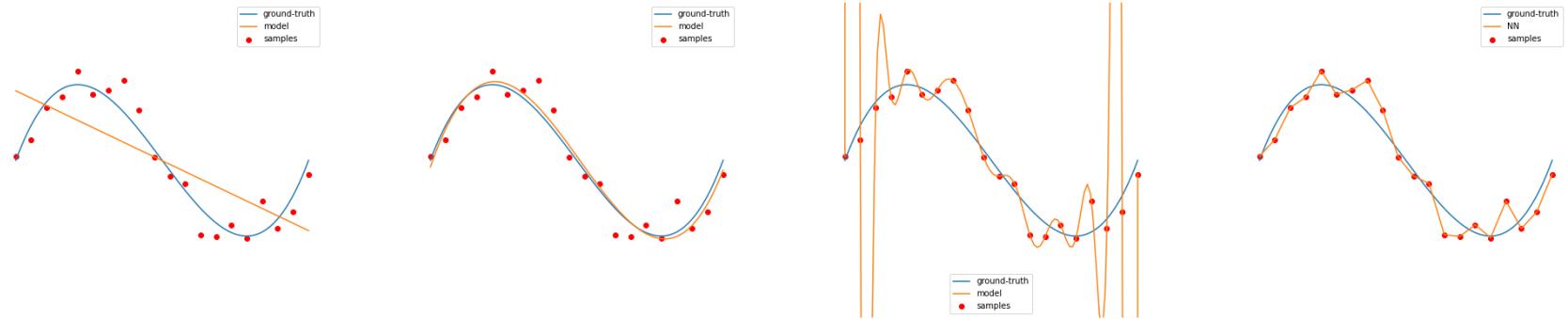


$\mathcal{C} =$	Affine functions	$\mathbb{R}_3[X]$	$\mathbb{R}_{19}[X]$
$\hat{R}_n(\theta_n^*)$	high	low	zero
$R(\theta_n^*)$	high	low	high
Problem	Underfitting	All good	Overfitting
Dim. of $\Theta$	2	4	20

---

# Underfitting and Overfitting depending on the choice of $\mathcal{C}$

---



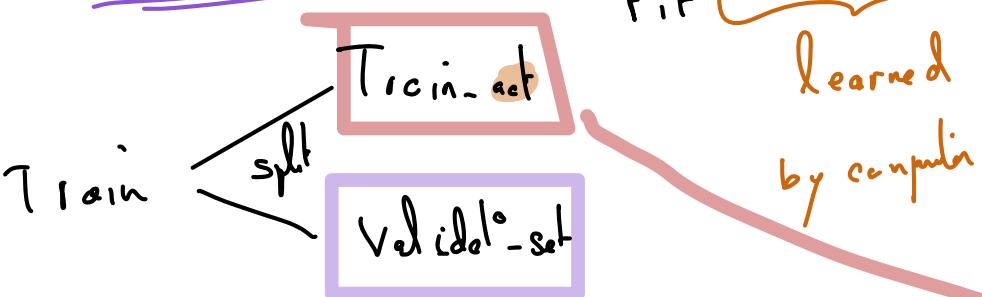
$\mathcal{C} =$	Affine functions	$\mathbb{R}_3[X]$	$\mathbb{R}_{19}[X]$	FCNN
$\hat{R}_n(\theta_n^*)$	high	low	zero	zero
$R(\theta_n^*)$	high	low	high	mid low
Problem	Underfitting	All good	Overfitting	
Dim. of $\Theta$	2	4	20	17375

Demo Colab

# Learning Pipeline

Split data as Tr + Test

- ① Training phase  
For all hyperparameters / models :



Learn  $\omega$  for a given model & hyperparameters on Tr.  
chosen / proposed by the user

- ② Validation phase

→ compare methods in a fair way  
evaluate the predictor  $f(\hat{\omega})$  on Validation-set  
→ Fair estimate of the performance for all methods & hyperparameters

- ③ Test phase

Test

Fair estimate of the model.

- ④ Deployment

If test error is acceptable, move to deployment



# One algorithm to rule SML: Cross validation

 Enables to select the best method within multiple methods

```
Entrée [7]: 1 models = {
2     "Decision Tree": DecisionTreeClassifier(random_state=42),
3     "Random Forest": RandomForestClassifier(n_estimators=10,random_state=42),
4     "Logistic Regression": LogisticRegression(multi_class="multinomial", solver="lbfgs", max_iter=100, random_st
5     "MLP Classifier": MLPClassifier(random_state=42),
6     "SVM": SVC()
7 }
```

```
Entrée [12]: 1 cv_results = {}  
2  
3 for name, model in models.items():  
4     kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)  
5     cv_score = cross_val_score(model, X_train, y_train, cv=kfold, scoring="accuracy")  
6     cv_results[name] = cv_score  
7 print(f"\n{name}: Mean CV accuracy={np.mean(cv_score)}, Std Deviation={np.std(cv_score)}")
```

Decision Tree: Mean CV accuracy=0.77125, Std Deviation=0.013163512495701512  
 Random Forest: Mean CV accuracy=0.8826785714285714, Std Deviation=0.008957097288560329  
 Logistic Regression: Mean CV accuracy=0.875, Std Deviation=0.007804204779790029  
 MLP Classifier: Mean CV accuracy=0.8842857142857143, Std Deviation=0.0031237242293813998  
 SVM: Mean CV accuracy=0.9505357142857143, Std Deviation=0.005463235193135154

```
Entrée [18]: 1 # Convert results to DataFrame for better visualization  
2 results_df = pd.DataFrame(cv_results)  
3 results_df
```

	Decision Tree	Random Forest	Logistic Regression	MLP Classifier	SVM
0	0.775893	0.885714	0.880357	0.885714	0.953571
1	0.752679	0.875000	0.866071	0.882143	0.942857
2	0.776786	0.898214	0.883929	0.885714	0.958929
3	0.790179	0.881250	0.865179	0.888393	0.950000
4	0.760714	0.873214	0.879464	0.879464	0.947321

→ best average validation performance

```
Entrée [17]: 1 # Descriptive statistics can also be helpful  
2 results_df.describe()
```

	Decision Tree	Random Forest	Logistic Regression	MLP Classifier	SVM
count	5.000000	5.000000	5.000000	5.000000	5.000000
mean	0.771250	0.882679	0.875000	0.884286	0.950536
std	0.014717	0.010014	0.008725	0.003492	0.006108
min	0.752679	0.873214	0.865179	0.879464	0.942857
25%	0.760714	0.875000	0.866071	0.882143	0.947321
50%	0.775893	0.881250	0.879464	0.885714	0.950000
75%	0.776786	0.885714	0.880357	0.888393	0.953571
max	0.790179	0.898214	0.883929	0.888393	0.958929

# Cross Validation

Goal: find a pair estimate of the general risk of the model

Data



Train

Test

For all models / hyperparam :

for i = 1 ... n\_splits  
 ↳ repeat the process  
 \* split Train



→ fit model on Fit  
 → compute error on Val



:

Figure: Cross Validation

# One algorithm to rule SML: Cross validation

💡 Enables to select the best hyperparameters for a given methods

## Leveraging grid search CV

```
Entrée [ ]: 1 from sklearn.model_selection import GridSearchCV
2
3 # Example for SVM
4 param_grid = {
5     'C': [1, 10, 100],
6     'gamma': [0.001, 0.01, 0.1],
7 }
8 grid_search = GridSearchCV(SVC(), param_grid, cv=3, verbose=2)
9 grid_search.fit(X_train_scaled[:10000], y_train[:10000]) # Reduced data size for quicker execution
10
11 print("Best parameters:", grid_search.best_params_)
12 print("Best cross-validation accuracy:", grid_search.best_score_)
13 best_model = grid_search.best_estimator_
14 final_predictions = best_model.predict(X_test_scaled)
15 print("Test set accuracy:", accuracy_score(y_test, final_predictions))
```

```
Entrée [43]: 1 pd.DataFrame(grid_search.cv_results_)
```

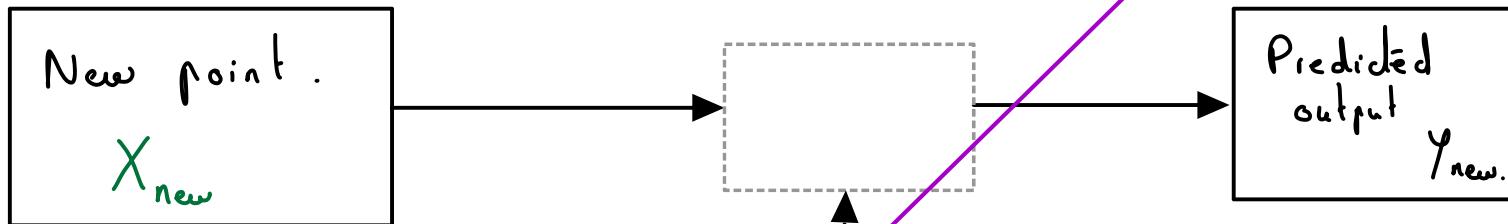
Out[43]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_gamma	params	split0_test_score	split1_test_score	split2_test_score	mean
0	2.354261	0.049712	1.981289	0.027614	1	0.001	{'C': 1, 'gamma': 0.001}	0.918050	0.930370	0.927653	
1	7.614626	0.022478	2.995119	0.006781	1	0.01	{'C': 1, 'gamma': 0.01}	0.693626	0.718800	0.724009	
2	8.525636	0.034385	3.381043	0.004120	1	0.1	{'C': 1, 'gamma': 0.1}	0.175147	0.187467	0.181136	
3	1.957532	0.022052	1.767812	0.001655	10	0.001	{'C': 10, 'gamma': 0.001}	0.928227	0.933048	0.937835	
4	7.731749	0.122276	3.002695	0.004842	10	0.01	{'C': 10, 'gamma': 0.01}	0.708623	0.735404	0.746517	
5	8.540356	0.023448	3.378721	0.004322	10	0.1	{'C': 10, 'gamma': 0.1}	0.176219	0.187467	0.184887	
6	1.959067	0.025085	1.774580	0.014967	100	0.001	{'C': 100, 'gamma': 0.001}	0.928763	0.933048	0.937835	
7	7.861557	0.143017	3.017998	0.025798	100	0.01	{'C': 100, 'gamma': 0.01}	0.708623	0.735404	0.746517	
8	8.649604	0.021181	3.455342	0.009529	100	0.1	{'C': 100, 'gamma': 0.1}	0.176219	0.187467	0.184887	

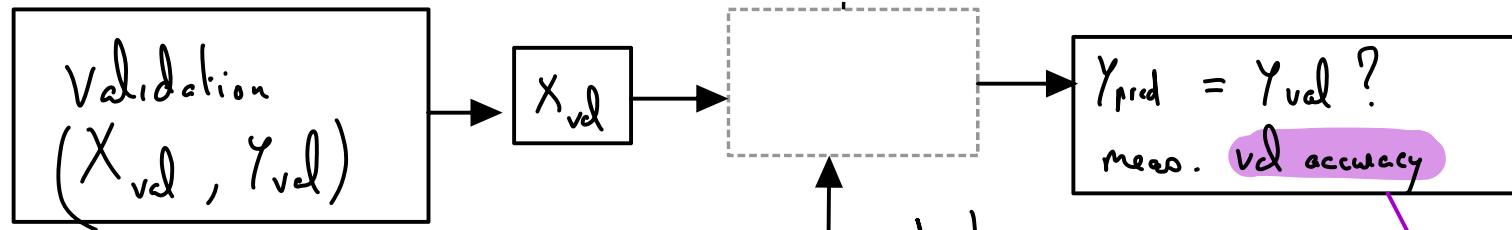
Figure: Grid search CV

# Supervised Machine Learning Pipeline

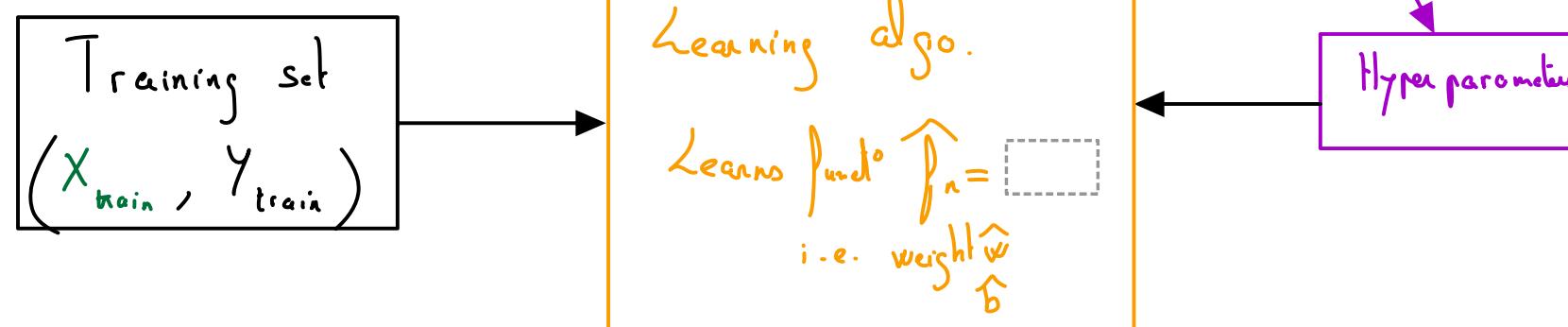
T  
E  
S  
T



V  
A  
L  
I  
D



F  
I  
T  
T  
R  
A  
I  
N  
I  
N  
G





# SML -Summary

## ① Key concepts

- ▶ Tasks
- ▶ Methods
- ▶ Overfitting and cross validation

# SML -Summary

## ① Key concepts

- ▶ Tasks
- ▶ Methods
- ▶ Overfitting and cr

## ② Black box solution:

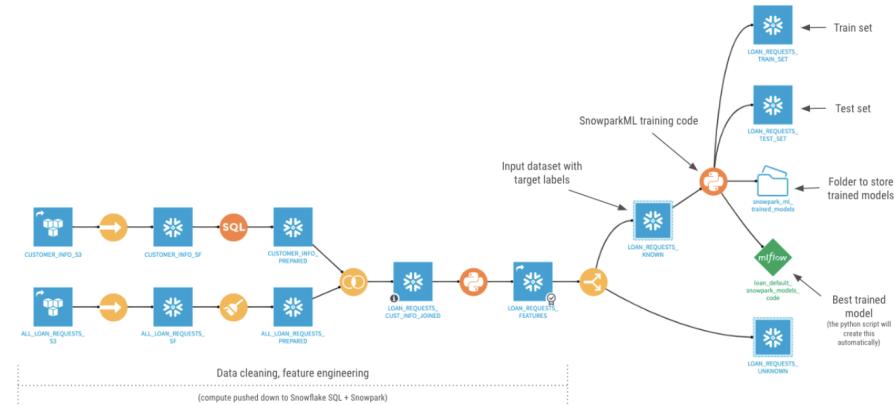


Figure: Dataiku

<https://blog.dataiku.com/training-ml-models-with-dataiku-and-snowpark-ml-a-code-approach>

# SML -Summary

## ① Key concepts

- ▶ Tasks
- ▶ Methods
- ▶ Overfitting and cr

## ② Black box solution:

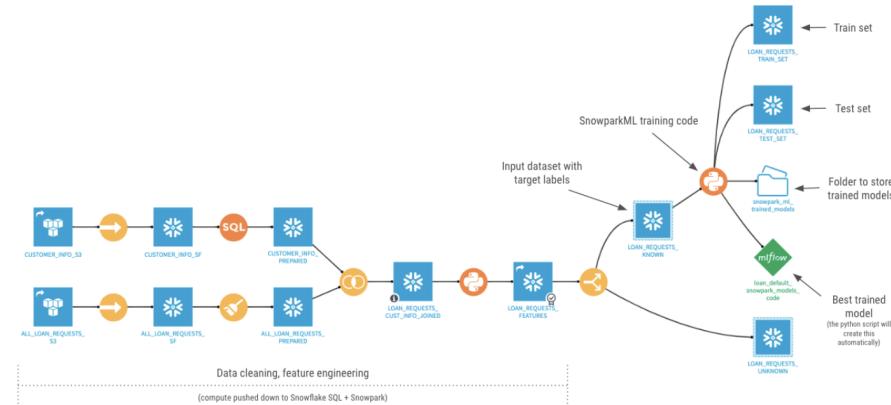


Figure: Dataiku

<https://blog.dataiku.com/training-ml-models-with-dataiku-and-snowpark-ml-a-code-approach>

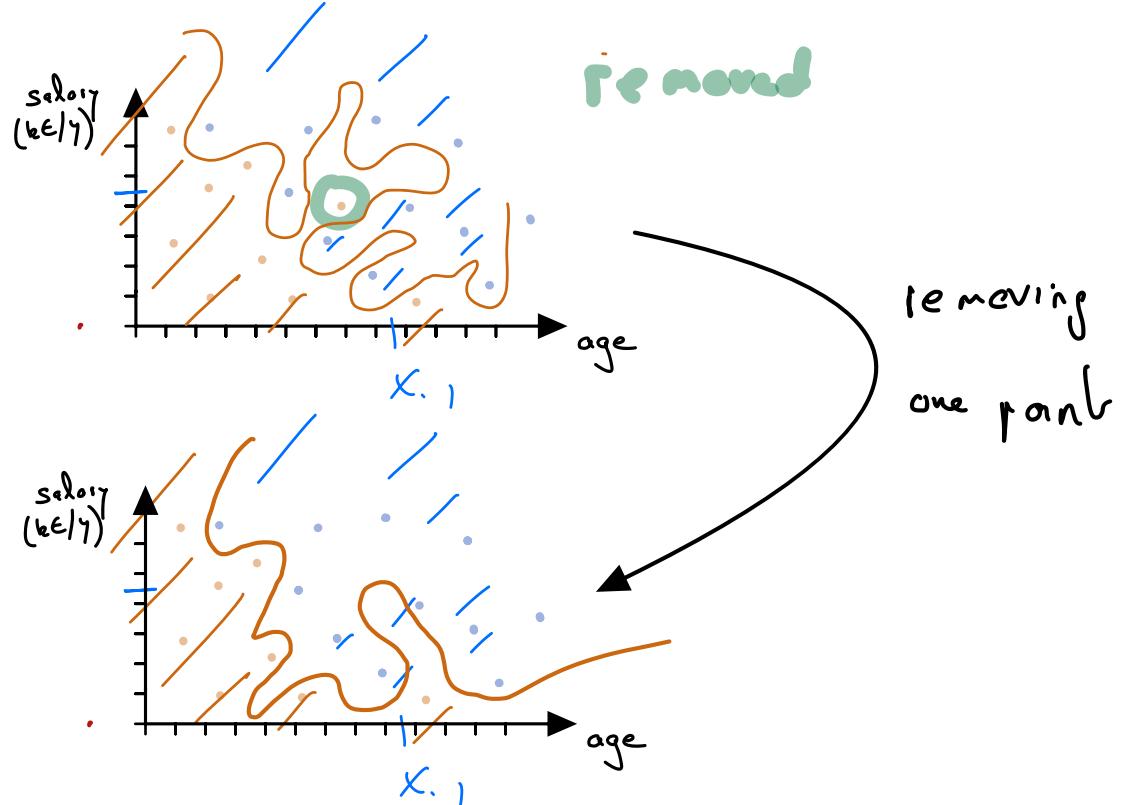
## ③💡 Our goal for today and tomorrow: **Opening the black box**

- Understanding methods enables to better anticipate the best choice, limits, potential, etc.
- Better choices for hyper-parameters to be chosen

# Afternoon Woodlop.

What is the problem of overfitting

- ① I will have a bad error on the train set X
- ② I will have a bad error on the test set ✓
- ③ My predictor may change a lot if I change the training set



What is the role of cross validation

- ① Find the best model in terms of validation error ✓
- ② Find the best parameters in terms of training error X (although we do this, we devide pts with cv)
- ③ Find the best hyperparameters in terms of validation error ✓
- ④ Fit the model X ("")
- ⑤ Find the best parameters in terms of validation error X

```

Entrée [7]: 1 models = {
2     "Decision Tree": DecisionTreeClassifier(random_state=42),
3     "Random Forest": RandomForestClassifier(n_estimators=10,random_state=42),
4     "Logistic Regression": LogisticRegression(multi_class="multinomial", solver="lbfgs", max_iter=100, random_st
5     "MLP Classifier": MLPClassifier(random_state=42),
6     "SVM": SVC()
7 }

Entrée [12]: 1 cv_results = {}
2
3 for name, model in models.items():
4     kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
5     cv_score = cross_val_score(model, X_train, y_train, cv=kfold, scoring="accuracy")
6     cv_results[name] = cv_score
7 print(f"\n{name}: Mean CV accuracy={np.mean(cv_score)}, Std Deviation={np.std(cv_score)}")

Decision Tree: Mean CV accuracy=0.77125, Std Deviation=0.013163512495701512
Random Forest: Mean CV accuracy=0.8826785714285714, Std Deviation=0.008957097288560329
Logistic Regression: Mean CV accuracy=0.875, Std Deviation=0.007804204779790029
MLP Classifier: Mean CV accuracy=0.884285142857143, Std Deviation=0.0031237242293813998
SVM: Mean CV accuracy=0.9505357142857143, Std Deviation=0.005463235193135154

Entrée [18]: 1 # Convert results to DataFrame for better visualization
2 results_df = pd.DataFrame(cv_results)
3 results_df

```

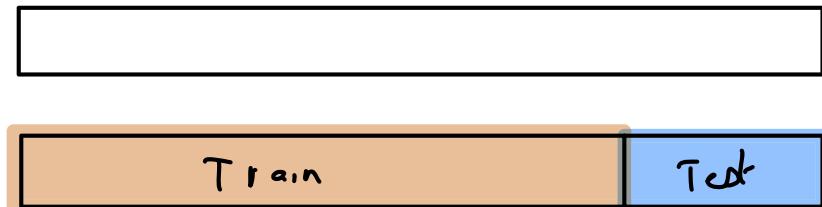
	Decision Tree	Random Forest	Logistic Regression	MLP Classifier	SVM
0	0.775893	0.885714	0.880357	0.885714	0.953571
1	0.752679	0.875000	0.866071	0.882143	0.942857
2	0.776786	0.898214	0.883929	0.882714	0.958929
3	0.790179	0.881250	0.885179	0.888393	0.950000
4	0.760714	0.873214	0.879464	0.879464	0.947321

What does this code do

- 1 It enables to compare the five methods given above ✓
- 2 It computes Train score for each method ✓
- 3 It computes Validation score for each method ✓
- 4 It computes 5 scores for each method, splitting the test data  
↳ never used
- 5 It computes 5 scores for each method, splitting the train data as val, fit ✓
- 6 The best method is SVM ✓
- 7 About 95% of the digits are well classified on the calibration set ✓
- 8 Random forest outperform Decision Trees  
88% val. acc vs 77% val. acc
- 9 25 fits were performed overall ✓ (expensive)
- 10 5 fits were performed overall X

finding a fair estimate of the generalization risk of the model

## Data



For all models / hyperparam :

for  $i = 1 \dots n\text{-splits}$   
↳ repeat the process

split Train

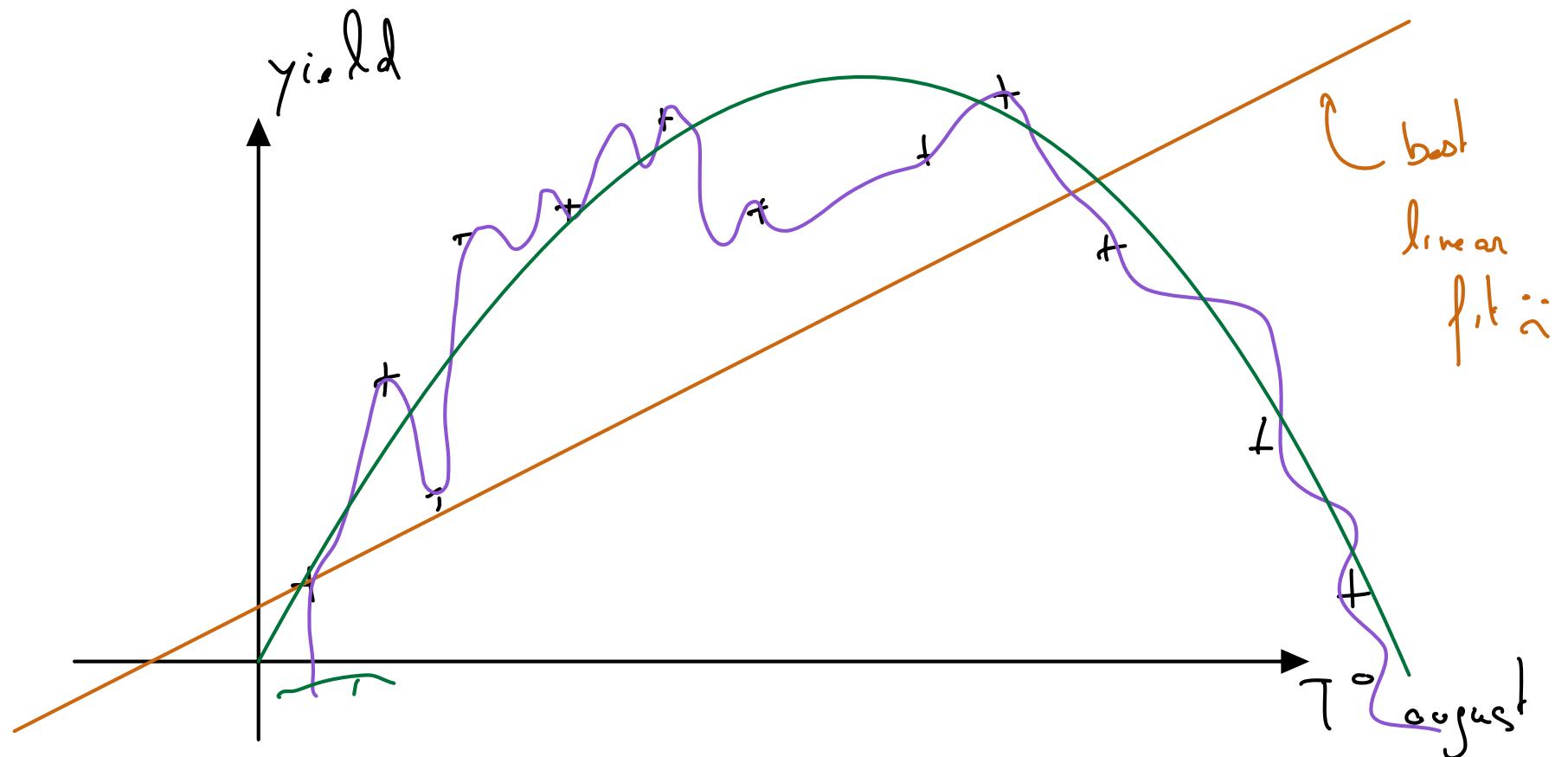


→ fit model on Fit  
→ compute error on Val



:

# Reminder on Linear and Logistic regression - towards DT



Tradeoff

Few possible final  $\theta$

Many possible  $\theta$

Interpretability

Speed (close form formula for fit)

Performance

Stable

good fit

Overfitting

Less Interp.

Slow, possible more perf.

# Limits of Linear and logistic regression - towards DT

# Outline

1 Introduction to Supervised Learning

2 Decision Trees

3 Random Forests

# 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 —→ allow 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 ?
- ③ Go beyond linear functions → allow complex dependencies.

Two questions:

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

## Inference Phase

Can get Covid shot example

(age, precondit., doctor)  
(a, p, d)

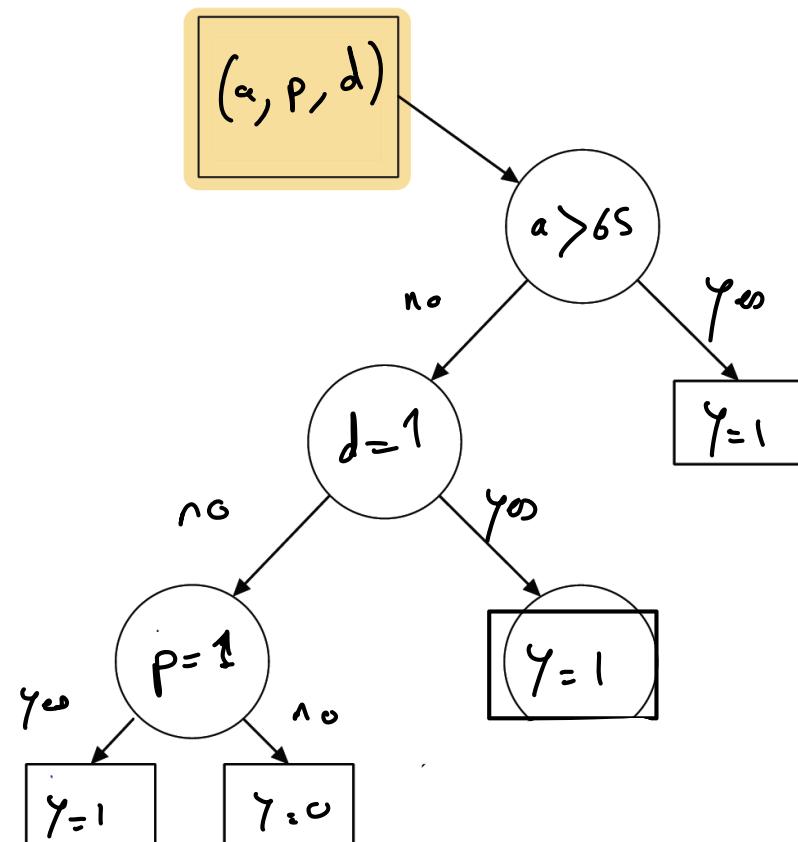
- Root = observation = input

- Nodes = tests

→ based on a single variable

- Branches = possible outcomes

- Leaves = final decision = output



Each node = 1 leaf = 

1 feature
1 threshold.

Training: how to build a tree.

f<sub>i</sub>

to a dot-set

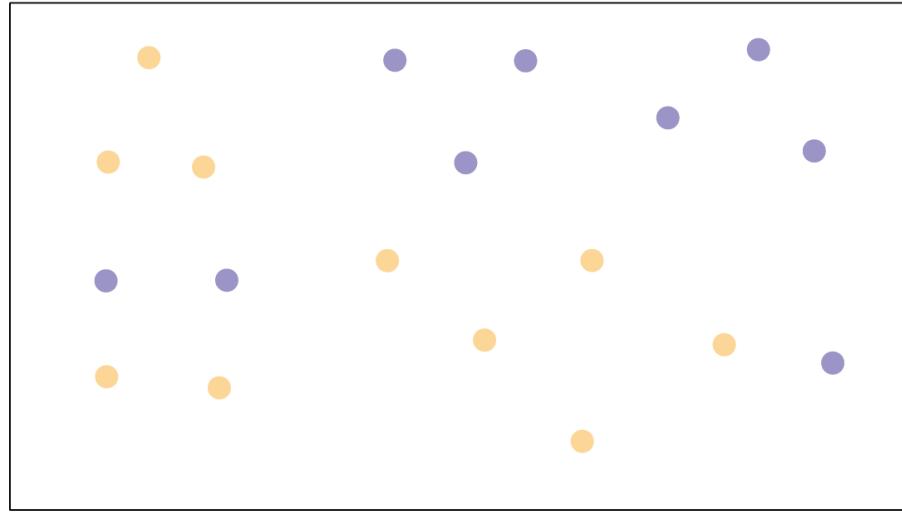
x<sub>f,i</sub>, y<sub>f,i</sub>

## Key question !

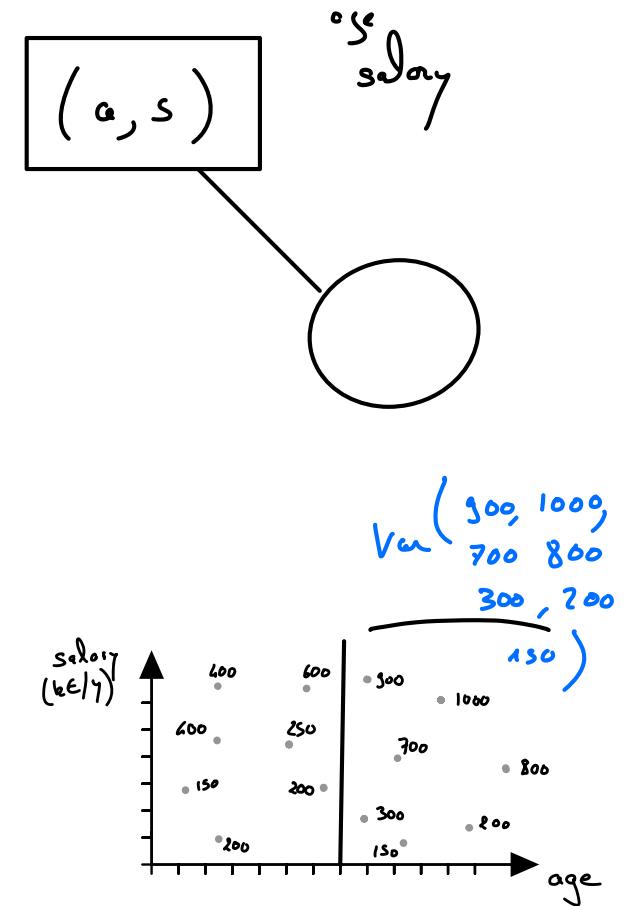
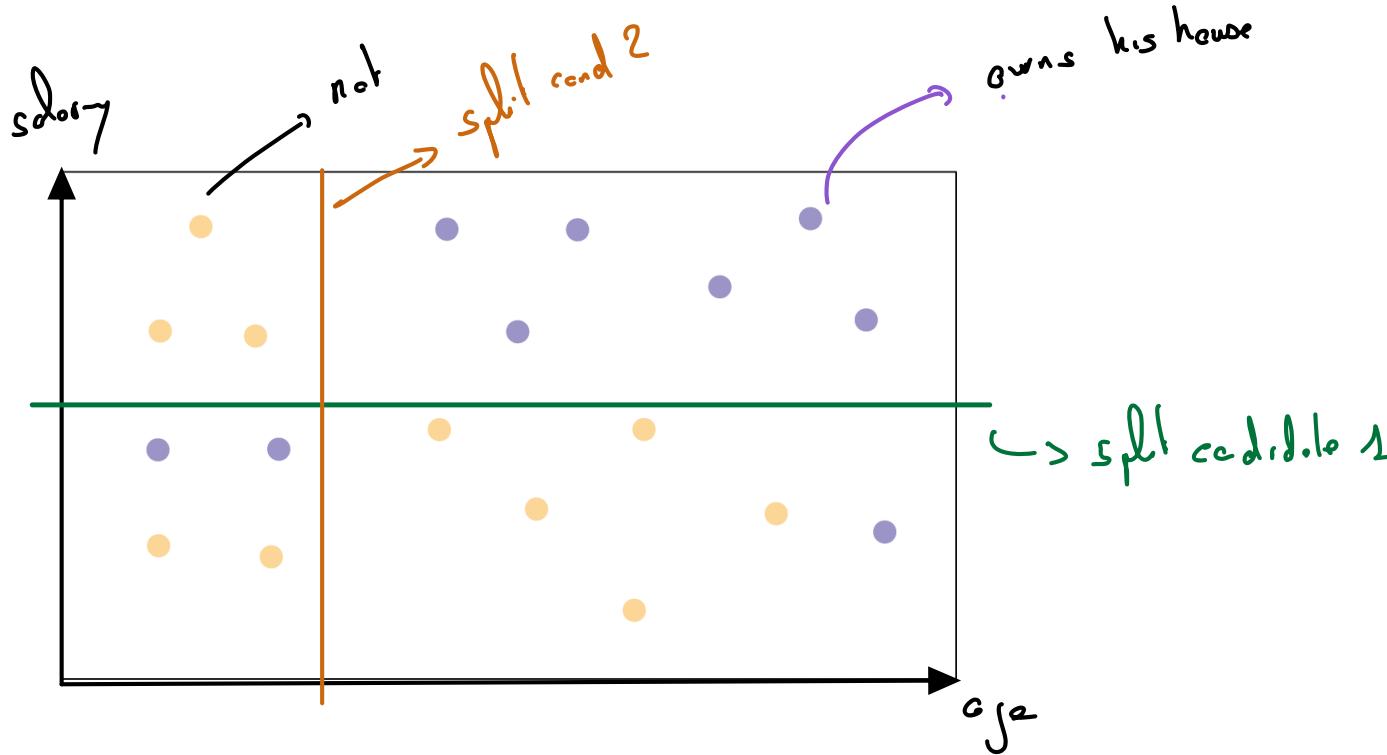
- What is a good tree?
- What do I need to choose?
- How to choose?

What do you think?

# Building trees : CART



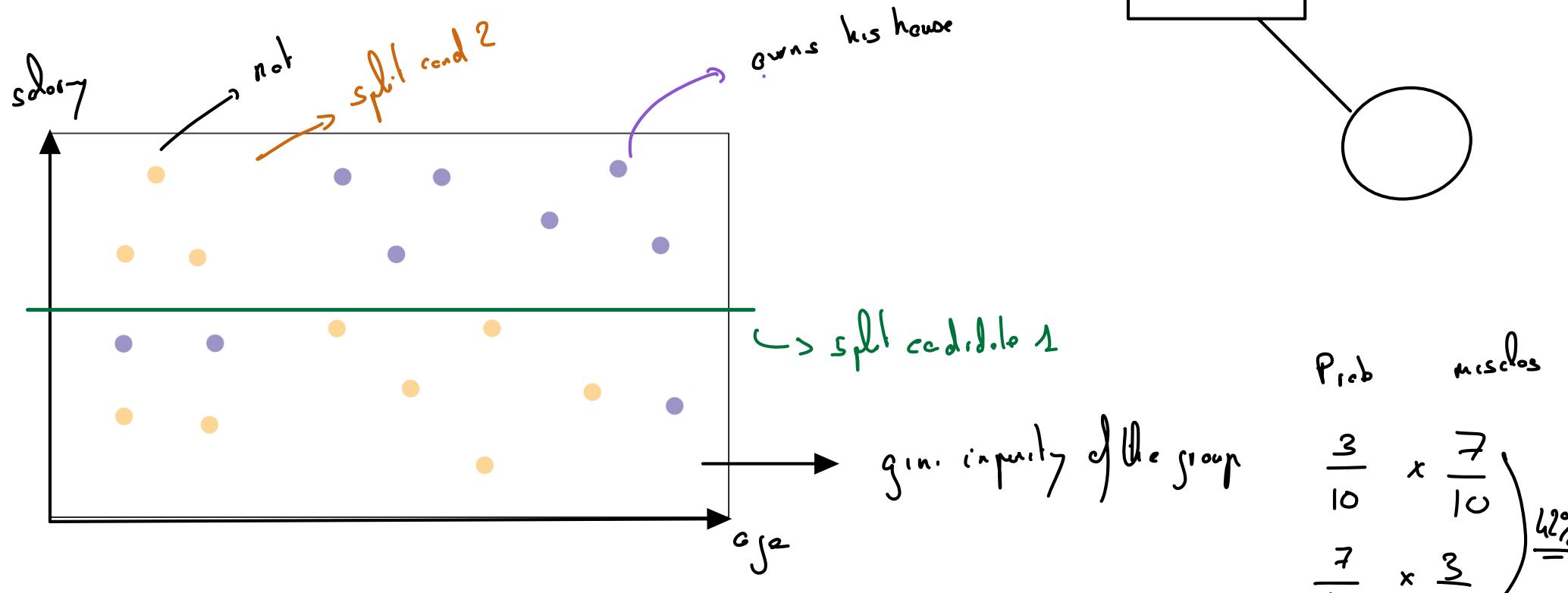
# Building trees : CART



## 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



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

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



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

$$x 2 ( \text{yellow} + \text{purple} ) = 50\%$$



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

48%



$$\frac{9}{10} \times \frac{1}{10} \approx 9\%$$

(almost 100%)



0%

18%



$$\begin{aligned} \frac{3}{10} &\times \frac{7}{10} & 21 \\ \frac{4}{10} &\times \frac{6}{10} & 24 \\ \frac{3}{10} &\times \frac{7}{10} = & 21 \\ && \hline && 66 \end{aligned}$$

0%  
"pure"

Ide.: a good split is one that makes both parts homogeneous in terms of output values.

# Example: Iris Dataset

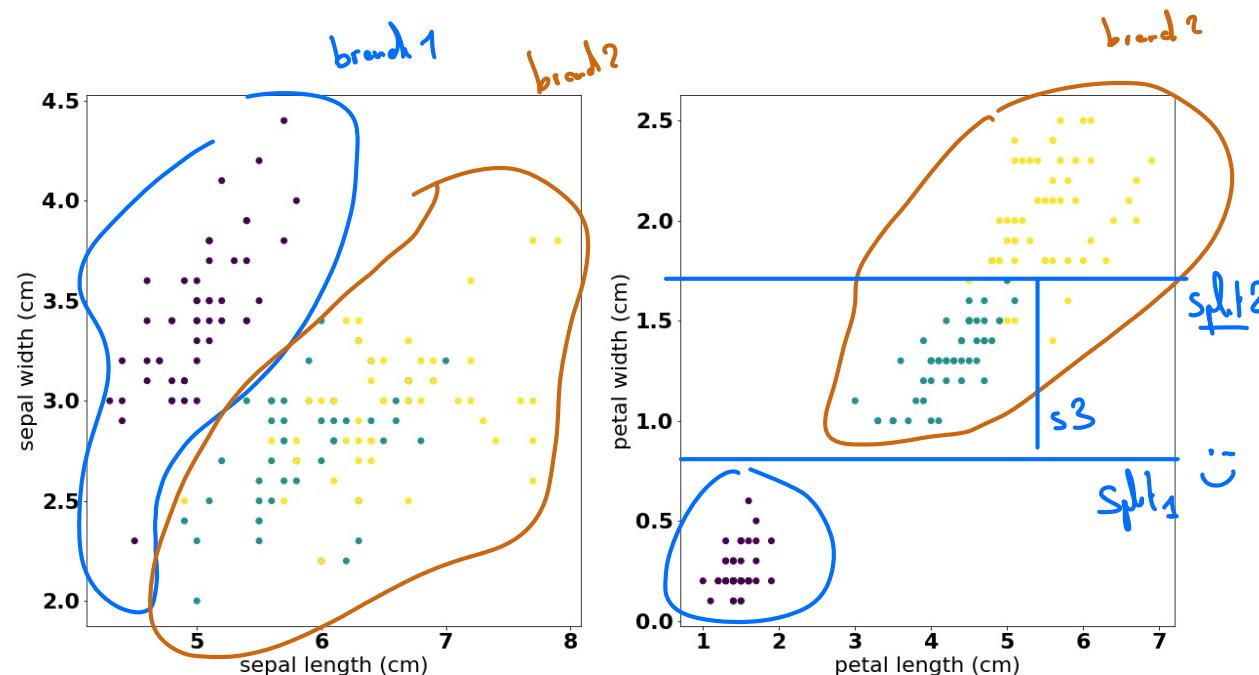
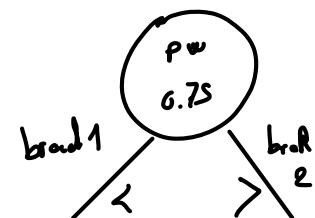


Figure: Iris: setosa, versicolor, virginica

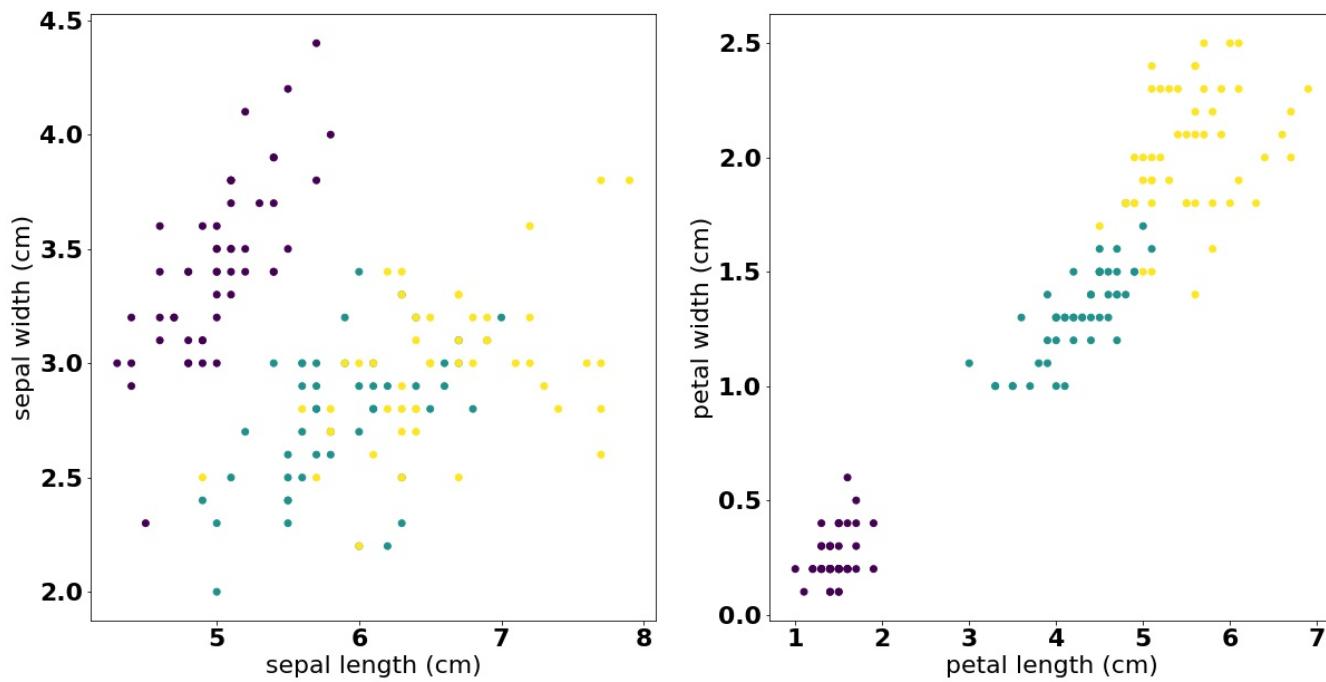


One of the most widely used toy dataset in classification:

- 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 practice : 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 ?

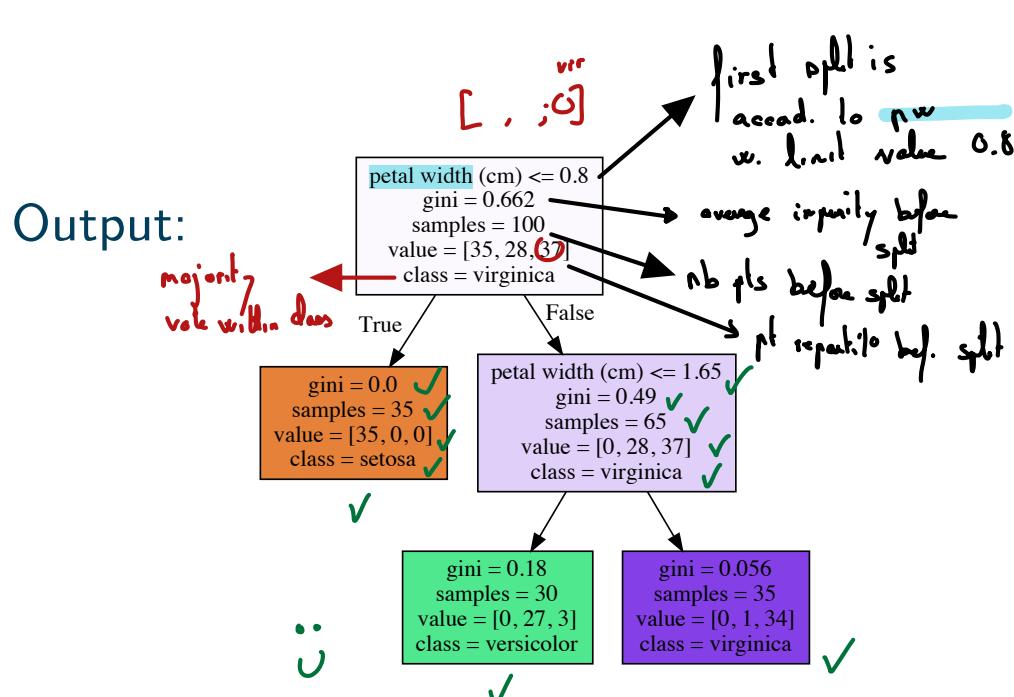
hyperparameters! as always

regularization techniques

→ avoid overfitting

max\_depth \* min-impurity decrease

samples - leaf pruning



Output:

```
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, ← hyperparams
| | | | | | | 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 X_tr, Y_tr.
cross_val_score(clf, [REDACTED], [REDACTED], cv=10) → given score

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

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

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/1dXYIjacNAeASgl54vIWjMQlfgigSVA?usp=sharing#scrollTo=VeJHkJd7RsDw>

# Woodap.

Decision trees and Supervised Learning

- ① The depth is "learned" in DT  X
- ② The depth is a hyperparameter  ✓
- ③ I use the validation set to find hyperparameters  ✓
- ④ There is no train set for Decision Trees  X

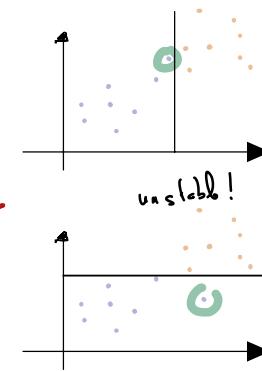
Which of the following statements are true about the construction of a decision tree?

- ① The same feature cannot be used twice consecutively"  X
- ② The same feature cannot be used twice overall"  X
- ③ I can use a pair of features if it gives me a better split  ✓
- ④ I want to find the split that has minimal impurity  ✓ obj pa fit
- ⑤ To define a new split, I only use one example  X

Which of the following statements are true about Decision Trees?

- ① They are very stable  X
- ② They provide an interpretable approach  ✓
- ③ They are able to approach the "optimal function" for nearly any data distribution (e.g. beyond linearly separable data)  ✓
- ④ They work well for a high number of observations  ✓
- ⑤ They do not work for Regression  X they do

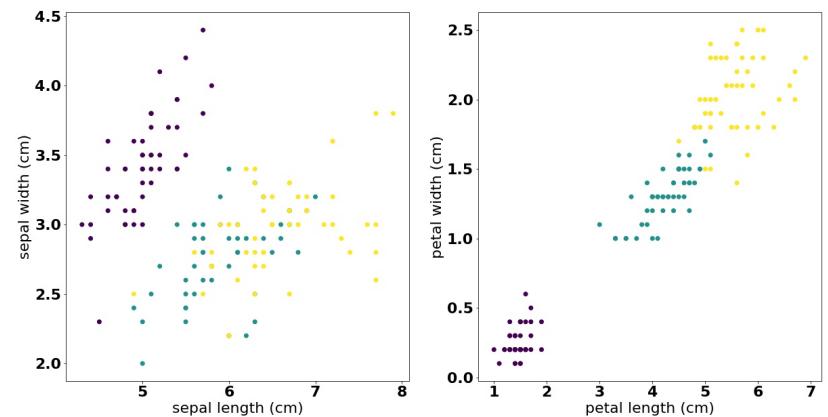
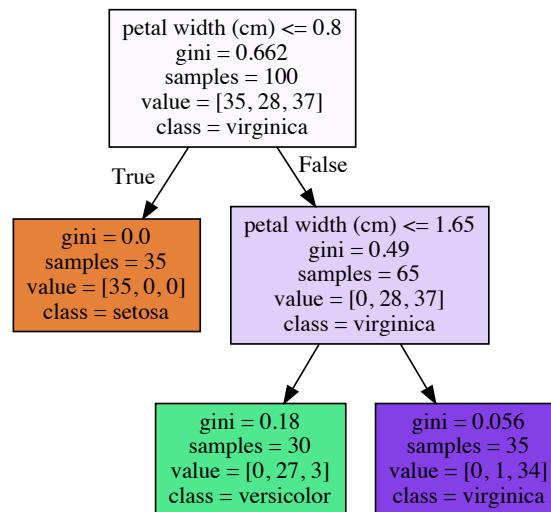
large d → large root ::



rich less  
( $\neq$  linear regression) → risk of overfitting

# Example: Iris Dataset

Output:



Example:

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

Returns:

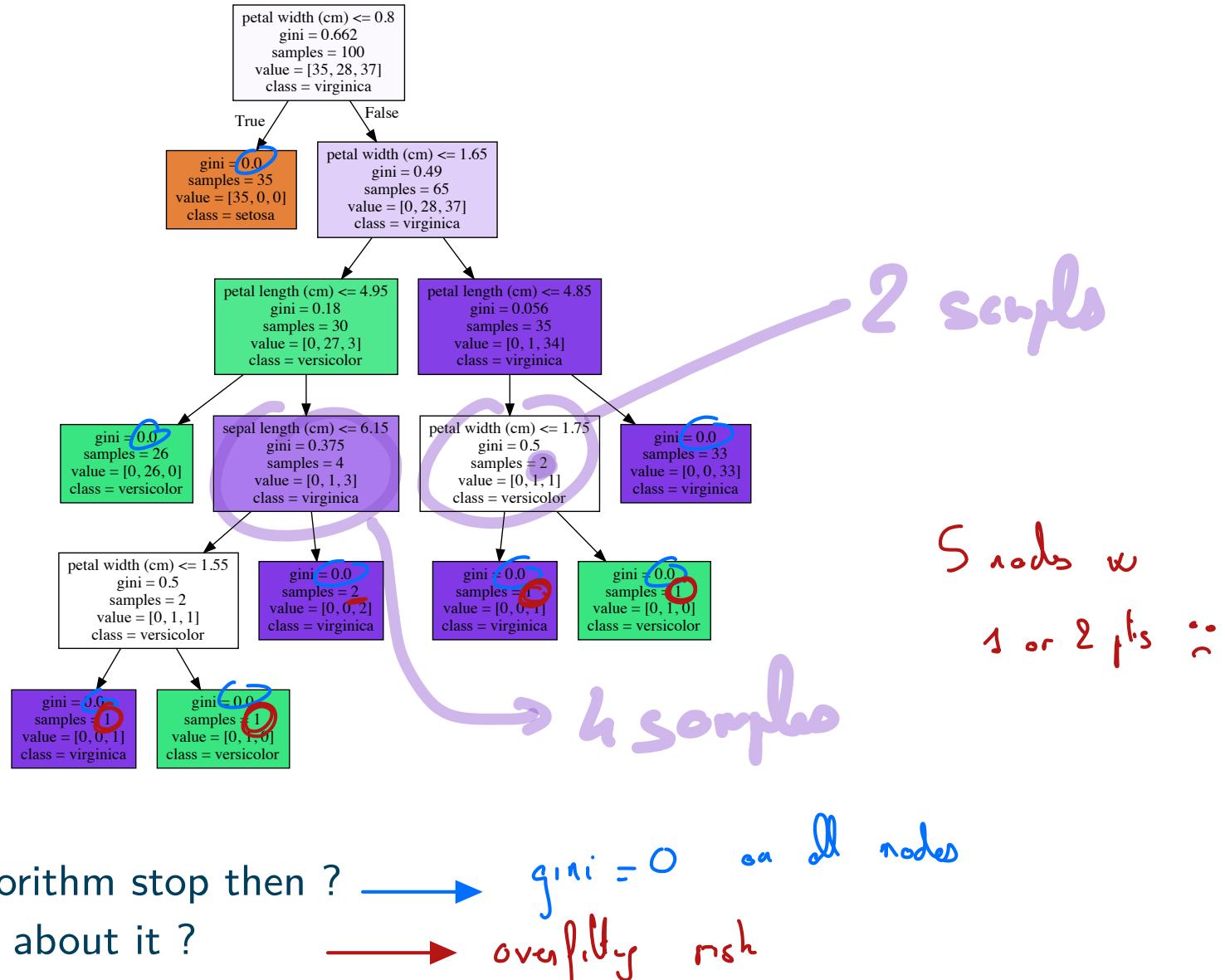
```
(array([4.8, 3.1, 1.6, 0.2]), 'setosa')
```

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 ?

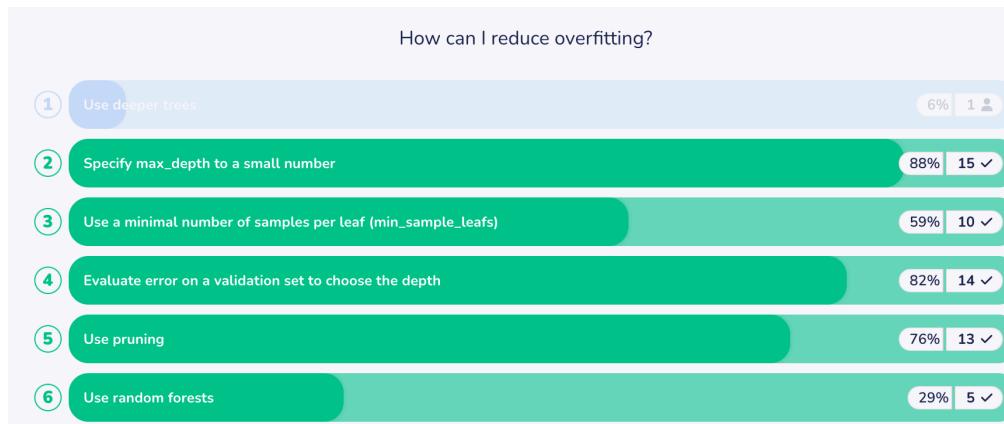
# Deeper trees

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



# 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.

✓ e.g. 5

## 2 Pruning

- Reduced error pruning.
- Cost complexity pruning.

True

# 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.<sup>1</sup> Then the test error is evaluated along the sequence, to choose the optimal pruning.



<sup>1</sup> more details here

# Cart: pros and cons

## Pros:

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

## 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 ✓

Summary,

- \* how to predict w. a tree ✓
  - \* has to fit a tree.
    - choose a criterion (Gini, Var)
  - iteratively find the best split
- cross val
- planning ; max depth,  
min - sample - ; etc ✓

Solution : Using Random Forests !

# Outline

1 Introduction to Supervised Learning

2 Decision Trees

3 Random Forests

# Random Forests

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

→ key idea: create variable trees + aggregate them

How to create variability ?

→ use only a random subset of features of each split

→ each tree works on a random subset of points

# Random Forests

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

How to create variability ?

- Instead of the most important features, search the best feature among a random subset (ntry). /
- 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

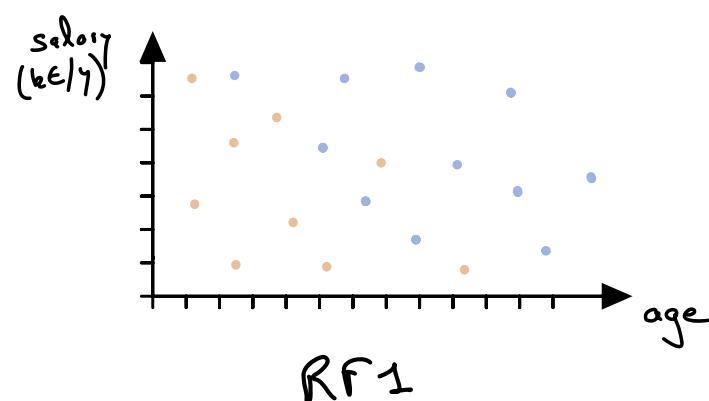
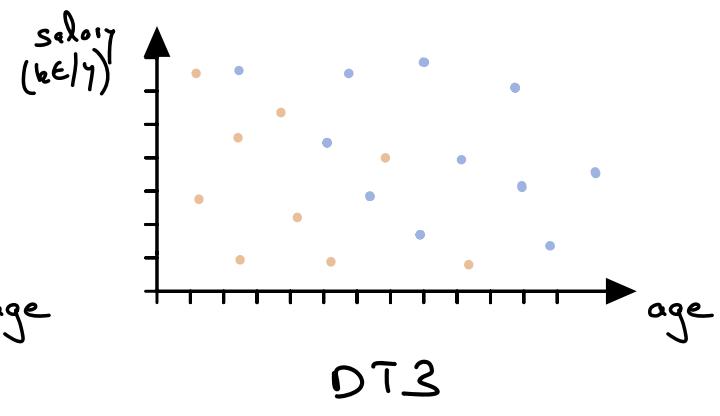
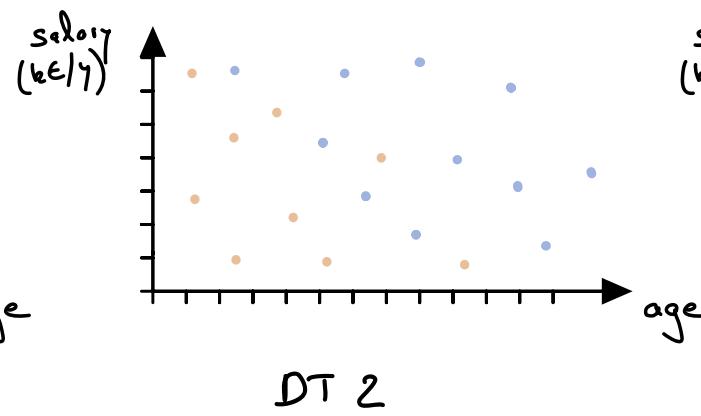
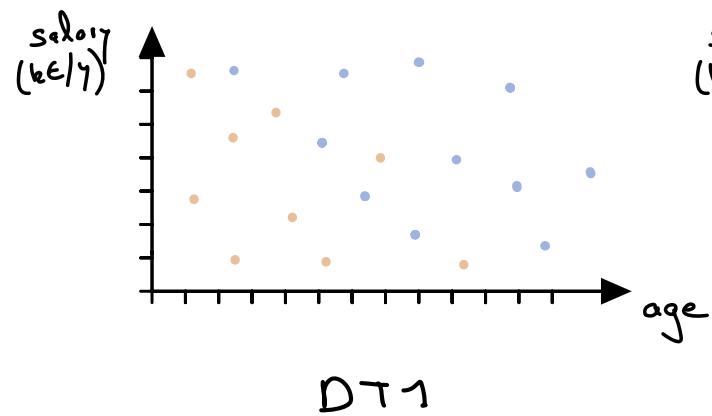
plt.figure(figsize=(20,10))
clf = RandomForestClassifier(n_estimators = 100,
                             max_depth=3, bootstrap = True,
                             random_state =43)

clf.fit(X_train,y_train)
```

How to aggregate ? How to interpret a Random Forest ?

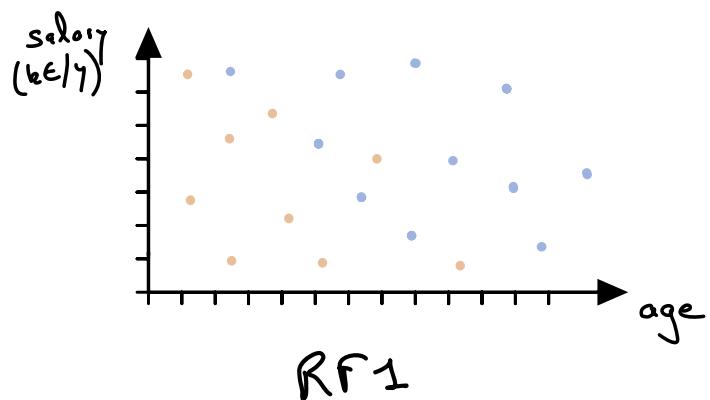
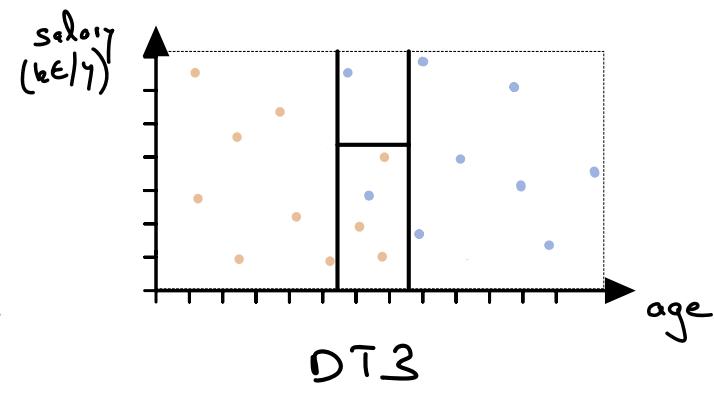
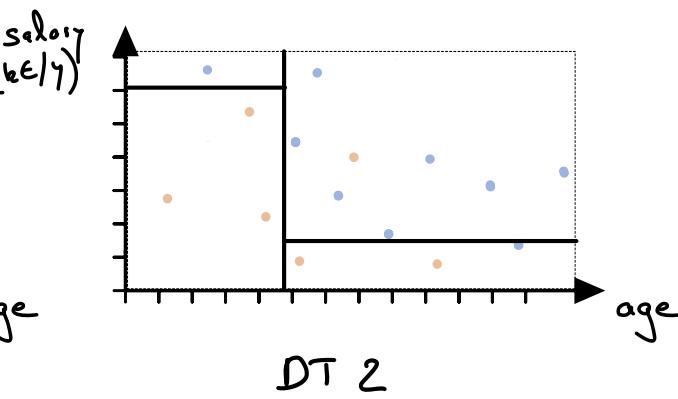
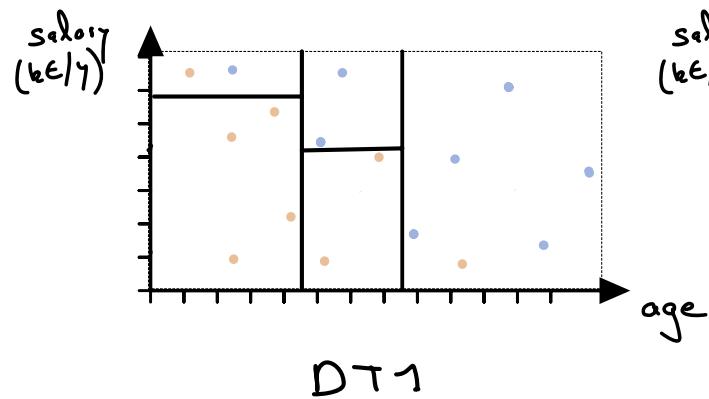
# Aggregation

We using a voting or averaging process !



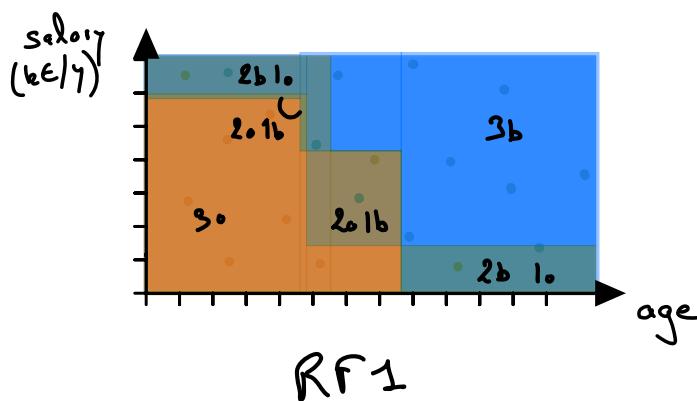
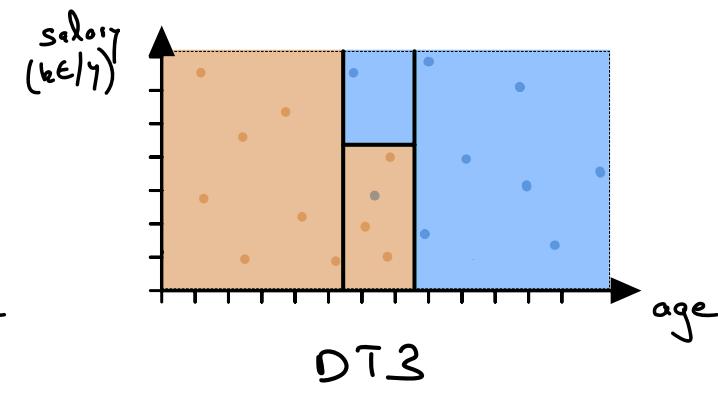
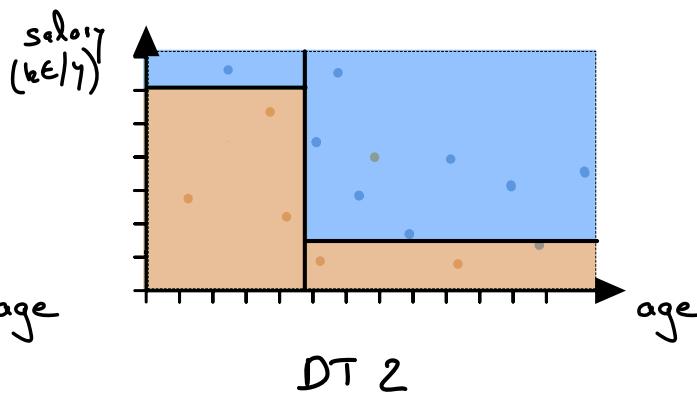
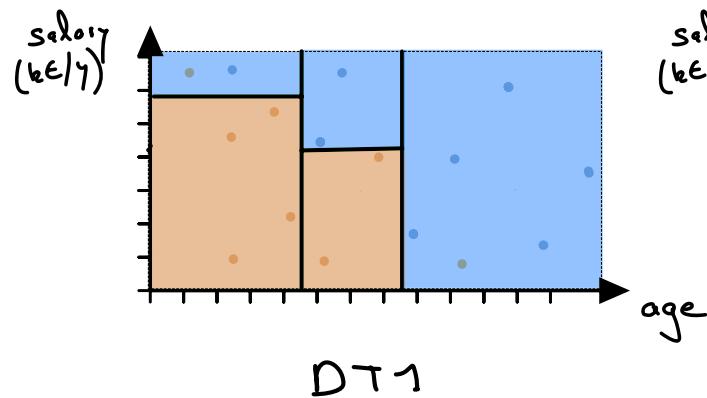
# Aggregation

We using a voting or averaging process !



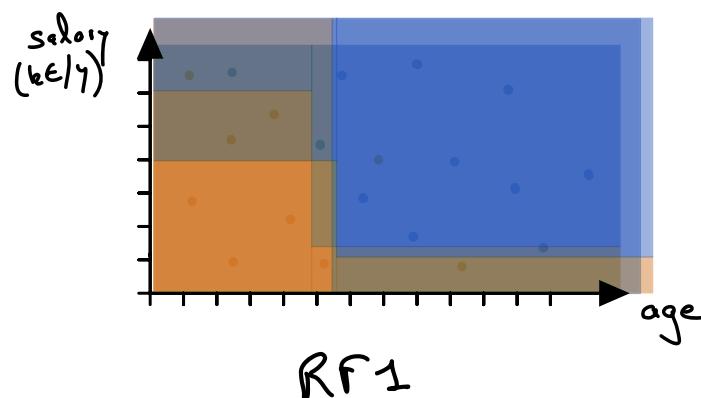
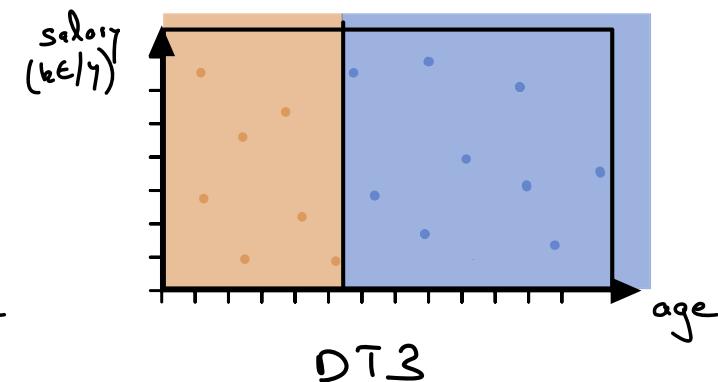
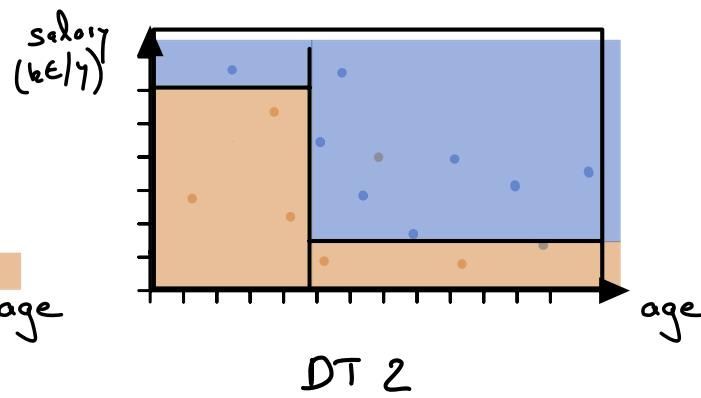
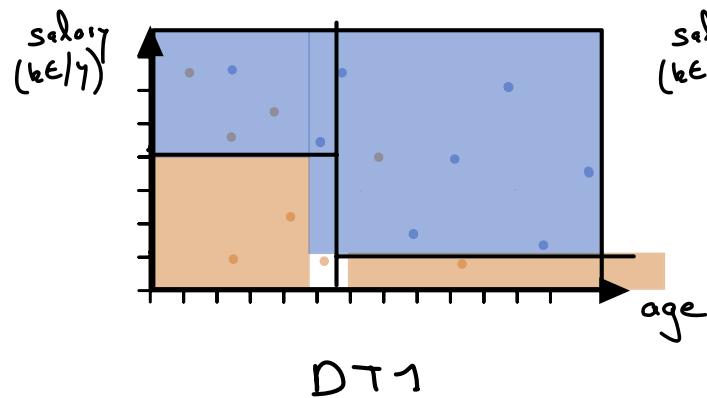
# Aggregation

We using a voting or averaging process !



# 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.

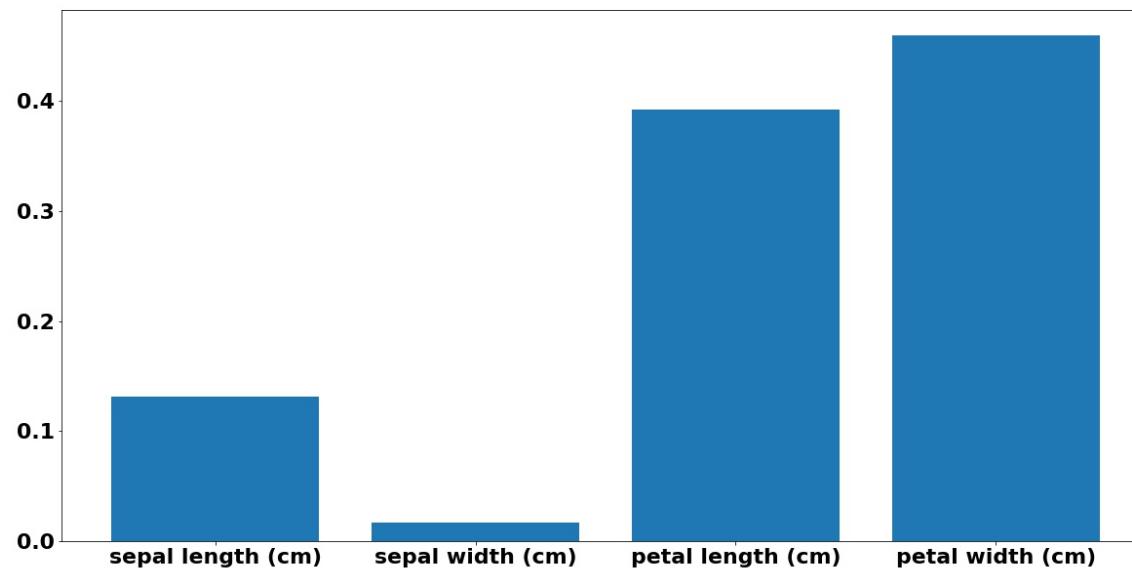


Figure: Feature importance output for a Random Forest Classifier in Python

# 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 Slow to provide new predictions -> ineffective for « real-time predictions ».
- ② Good for prediction but bad for description.

What about Regression ?

writes the same with *Regression Trees*

# 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

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.

Face completion with multi-output estimators



## 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.

