# ML, Decision Trees and Random Forests
# Data science Certificate

Aymeric DIEULEVEUT

April 2025

# Outline

# A Reminder on Supervised Learning

- 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

- 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.

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

# Example 2: Iris Dataset - Classification

Figure: Iris: setosa, versiocolor, virginica

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

Iris Dataset:

- 3 classes : *['setosa', 'versicolor', 'virginica']*

  $\rightarrow$ $Y \in \{0, 1, 2\}$ encoding each class.

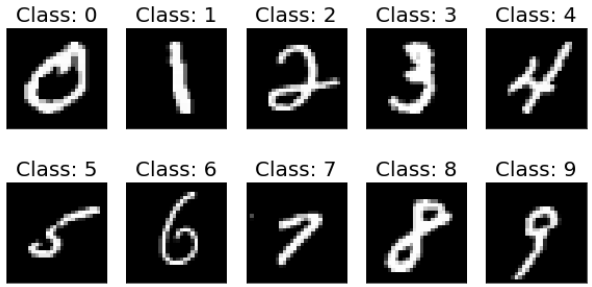
- 50 points per class

  $\rightarrow$ $n = 150$

- 4 features: *[ 'sepal length', 'sepal width', 'petal length', 'petal width']*

  $X \in \mathbb{R}^4$, $\rightarrow$ $d = 4$

# 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 :

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

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

  $n = 60000$

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

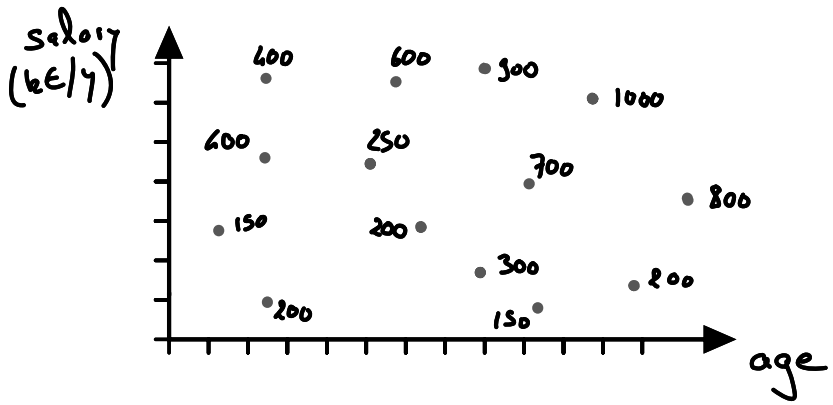
  $X \in \mathbb{R}^{784}$, $d = 784$ features.

# Summary of those three datasets

| | Iris | MNIST | House price prediction |
|---|---|---|---|
| Number of examples $n$<br><br>Number of the features $d$ | | | |
| Type of the features | | | |
| Space $\mathcal{Y}$ | | | |
| Task | | | |

# Toy datasets

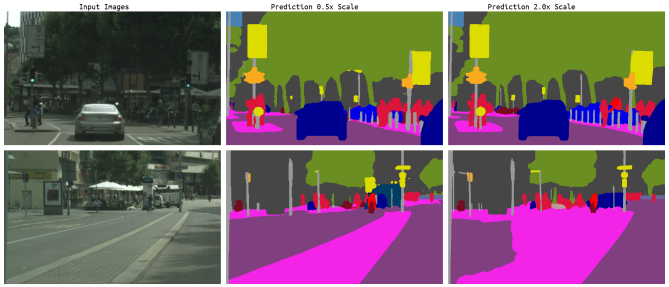
- In regression:



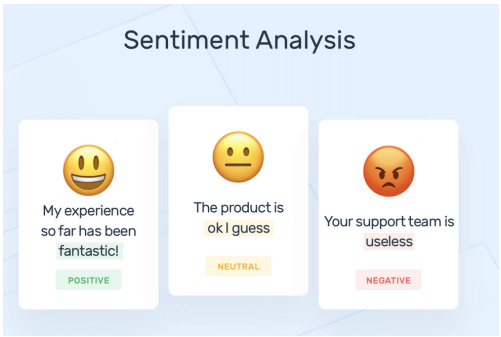

- In classification:

# More examples

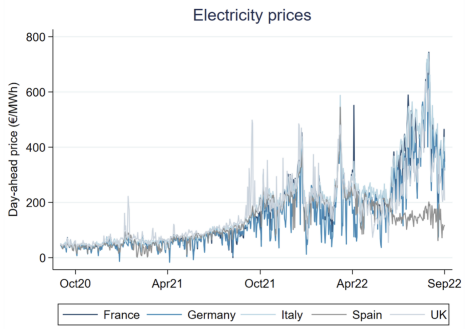
Spam detection




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,
  - $Y \in \{0, 1, 2, \ldots, K-1\}$ - multi-class classification,
  - $Y \in \mathbb{R}$ - regression.
- Phenomenon distribution $(X, Y) \sim \mathbb{P}$.

# Supervised Learning

## Supervised Learning

- Attributes / features / characteristics / input:
$$X \in \mathcal{X} = \mathbb{R}^d$$
- Output / Responses / label :
  - $Y \in \{-1, 1\}$ - binary classification,
  - $Y \in \{0, 1, 2, \ldots, K-1\}$ - multi-class classification,
  - $Y \in \mathbb{R}$ - regression.
- Phenomenon distribution $(X, Y) \sim \mathbb{P}$.

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

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

$\to$ introduce a risk function.

# 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): $\ell(y, g(x)) = 1y \neq g(x), \quad y \in \mathcal{Y} = \{-1, 1\}$
  - Quadratic Loss (regression): $\ell(y, g(x)) = (y - g(x))^2$, $y \in \mathbb{R}^d$

# 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): $\ell(y, g(x)) = 1y \neq g(x), \quad y \in \mathcal{Y} = \{-1, 1\}$
  - 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): $R(g) = \mathbb{E}[|Y - g(x)|^2]$

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

# 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): $\ell(y, g(x)) = 1y \neq g(x), \quad y \in \mathcal{Y} = \{-1, 1\}$
  - 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): $R(g) = \mathbb{E}[|Y - g(x)|^2]$

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

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

## Data-dependent predictors

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

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

## Supervised Learning: Framework Summary

- Training set: $D_n = \{(x_1, Y_1), \ldots, (x_n, Y_n)\} \quad (i.i.d. \sim \mathbb{P})$
- Decision rule (classifier, predictor): $g : \mathcal{X} \to \mathcal{Y}$
- Loss: $\ell(Y, g(x))$
- Risk: $R(g) = \mathbb{E}[\ell(Y, g(x))]$

Numerous methods:

1. In **regression**:
   - linear, polynomial, kernel regression
   - regularized methods (Lasso, Ridge)
2. In **classification**:
   - Logistic regression
   - SVM
3. For **both**
   - Nearest neighbors
   - Random forests
   - Neural networks
   - etc.

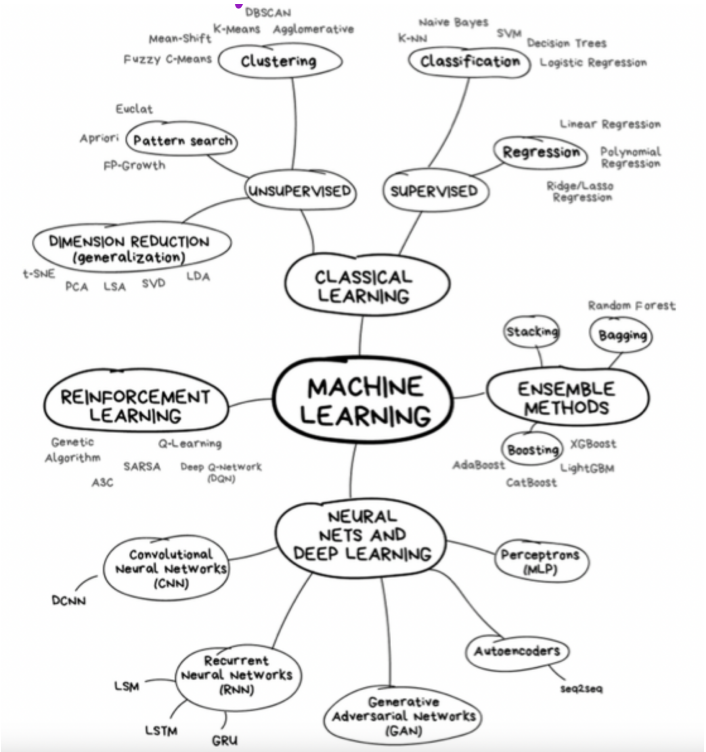



Figure:

# Roadmap towards Understanding machine learning!

# Training Pipeline

1. Fitting

   💡 "Learn" the dependency between input and output. Each model (LR, Logistic regression, Neural networks, Decision Trees) $\rightarrow$ obtain the model !

# Fitting a method

Demo url (Lab): https://colab.research.google.com/github/adieulev/Certificate-DS-2025/blob/main/Labs/MWE_ML_init_student_version.ipynb

```python
# Fetching the MNIST dataset
mnist = fetch_openml('mnist_784', version=1)
X, y = mnist["data"], mnist["target"]

# Setting aside 10% of the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.9, random_state=42)
```

Figure: Importing a dataset

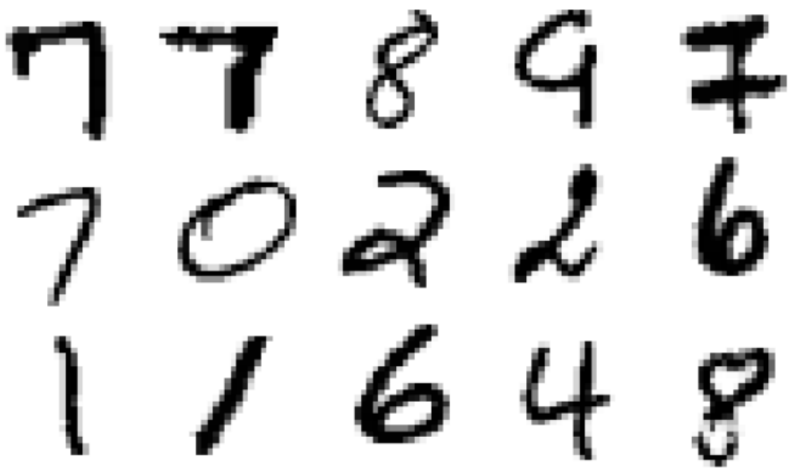


Figure: Digits

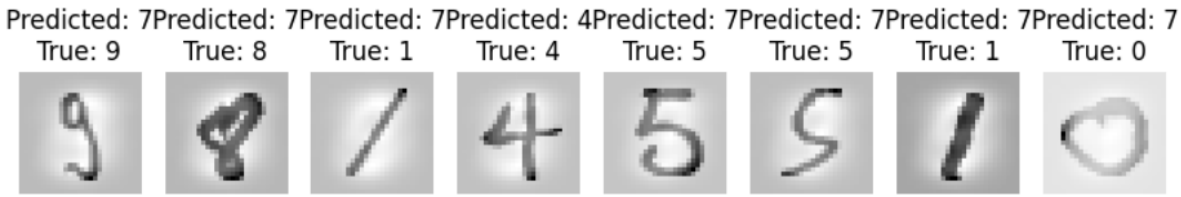
# Fitting 3 methods

```
X_fit, X_val, y_fit, y_val = train_test_split(X_train,  y_train,
                                              test_size=0.2, random_state=42)

from sklearn.tree import DecisionTreeClassifier

dt_clf = DecisionTreeClassifier()
dt_clf.fit(X_fit, y_fit)
```

▾ DecisionTreeClassifier ❶ ❷

```
DecisionTreeClassifier()
```

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




```
from sklearn.ensemble import RandomForestClassifier

rf_clf = RandomForestClassifier(n_estimators=10,random_state=42)
rf_clf.fit(X_fit, y_fit)
```

▾ RandomForestClassifier ❶ ❷

```
RandomForestClassifier(n_estimators=10, random_state=42)
```

```
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression(multi_class="multinomial", solver="lbfgs", max_iter=100, random_state=42)
log_reg.fit(X_fit, y_fit)
```

▾ LogisticRegression ❶ ❷

```
LogisticRegression(multi_class='multinomial', random_state=42)
```

# Looking at the accuracy

## Obtain Accuracies on dataset to fit (train accuracy)

```
[25] print(
         f"On the dataset used to fit, the decision tree has "
         f"{100 * dt_clf.score(X_fit, y_fit):.1f}% accuracy"
     )
     print(
         f"On the dataset used to fit, the random forest has "
         f"{100 * rf_clf.score(X_fit, y_fit):.1f}% accuracy"
     )
     print(
         f"On the dataset used to fit, the logistic regresssion has "
         f"{100 * log_reg.score(X_fit, y_fit):.2f}% accuracy"
     )
```

```
On the dataset used to fit, the decision tree has 100.0% accuracy
On the dataset used to fit, the random forest has 99.8% accuracy
On the dataset used to fit, the logistic regresssion has 100.00% accuracy
```

## Obtain Validation Accuracies

```
[26] print(
         f"On the dataset left aside for the fitting part, the decision tree has "
         f"{100 * dt_clf.score(X_val, y_val):.1f}% accuracy"
     )
     print(
         f"On the dataset left aside for the fitting part, the random forest has "
         f"{100 * rf_clf.score(X_val, y_val):.1f}% accuracy"
     )
     print(
         f"On the dataset left aside for the fitting part, the logistic regresssion has "
         f"{100 * log_reg.score(X_val, y_val):.2f}% accuracy"
     )
```

```
On the dataset left aside for the fitting part, the decision tree has 77.6% accuracy
On the dataset left aside for the fitting part, the random forest has 88.8% accuracy
On the dataset left aside for the fitting part, the logistic regresssion has 87.57% accuracy
```

# Wooclap time!

1 Allez sur wooclap.com

2 Entrez le code d'événement dans le bandeau supérieur

Code d'événement
TAMXAU

1 Envoyez @TAMXAU au 06 44 60 96 62

2 Vous pouvez participer

Désactiver les réponses par SMS

## What is true about SML

1. Supervised Machine Learning does not have inputs
2. Supervised Machine Learning can be combined with unsupervised in some contexts
3. Supervised Machine Learning does not have outputs
4. The goal is to predict well on new unseen points
5. It is necessary to perform well on the train (fit) set for that
6. The goal is to predict well on the train (fit) points
7. The balance interpretability/speed/ performance depends on the method
8. I assume my training data (fit + validation + test) to be representative (iid) of a phenomenon with distribution P

## What is the role of fitting, e.g., in

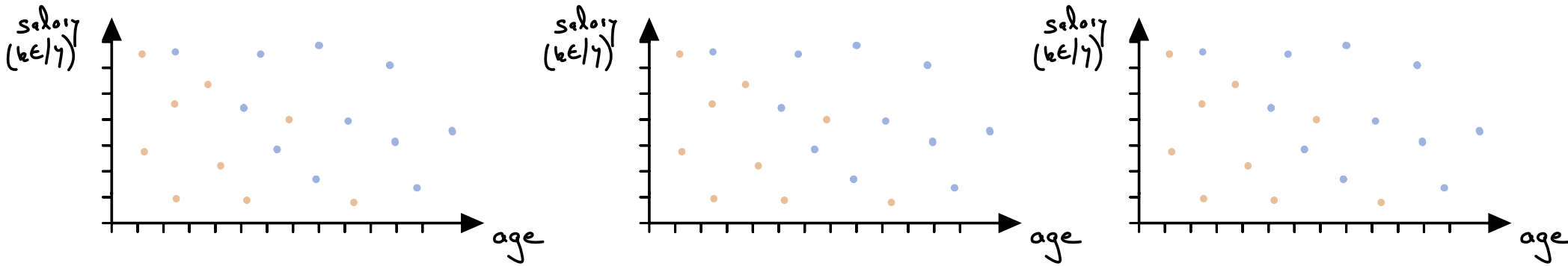
```
rf_clf = RandomForestClassifier(n_estimators=10,random_state=4...
```

1. Finding the best parameters of the Forest/Treess
2. Finding the best hyperparameters (n_estimators)
3. Predicting on a test point

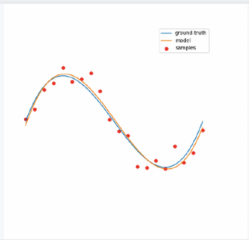
# Caveat: Overfitting - Toy datasets

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

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



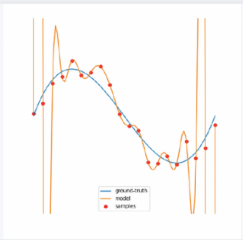

💡 Need to focus on validation/test performance



1

A

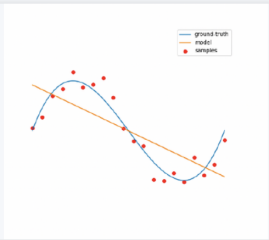
Good fit



2

B

Under-fitting



3

C

Over fitting

# Caveat: Overfitting - Toy datasets

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

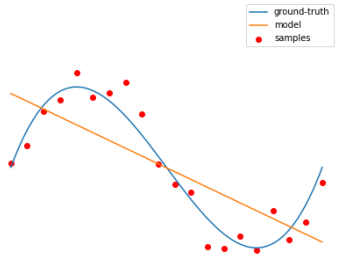
💡 Our Goal is to perform well on unseen outputs!!


💡 Need to focus on validation/test performance

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

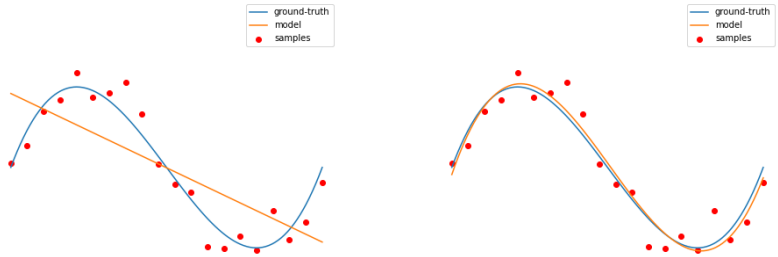
| $\mathcal{C} =$ |
| --- |
| $\hat{R}_n(\theta_n^*)$ |
| $R(\theta_n^*)$ |
| Problem |
| Dim. of $\Theta$ |

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



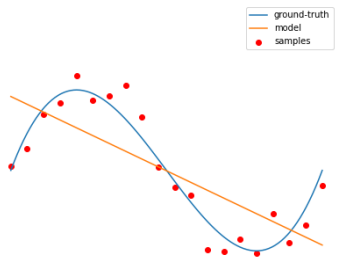

| | |
|---|---|
| $\mathcal{C} =$ | Affine functions |
| $\hat{R}_n(\theta_n^*)$ | high |
| $R(\theta_n^*)$ | high |
| Problem | Underfitting |
| Dim. of $\Theta$ | 2 |

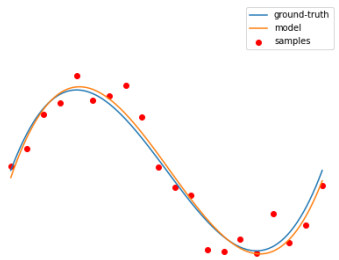
# Underfitting and Overfitting depending on the choice of $\mathcal{C}$

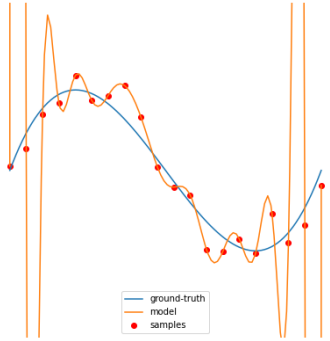



| $\mathcal{C} =$ | Affine functions | $\mathbb{R}_3[X]$ |
|---|---|---|
| $\hat{R}_n(\theta_n^*)$ | high | low |
| $R(\theta_n^*)$ | high | low |
| Problem | Underfitting | All good |
| Dim. of $\Theta$ | 2 | 4 |

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

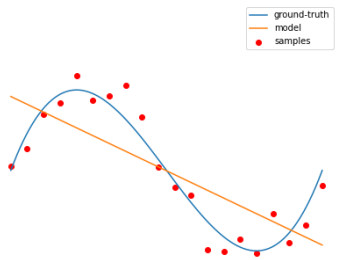


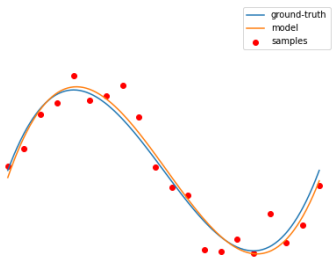



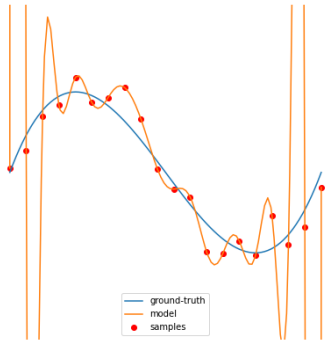



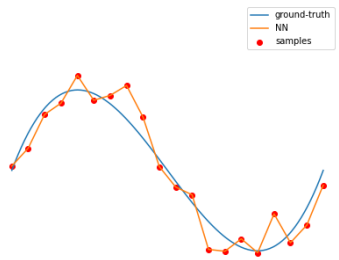

| $\mathcal{C} =$ | Affine functions | $\mathbb{R}_3[X]$ | $\mathbb{R}_{19}[X]$ |
|---|---|---|---|
| $\hat{R}_n(\theta_n^*)$ | high | low | zero |
| $R(\theta_n^*)$ | high | low | high |
| Problem | Underfitting | All good | Overfitting |
| Dim. of $\Theta$ | 2 | 4 | 20 |

# 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 |

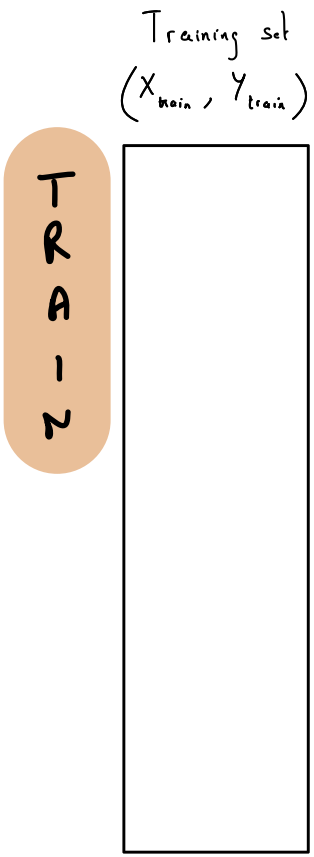
Demo Colab

# Learning Pipeline

1. Training phase

# Learning Pipeline

1. Training phase
   1. Fitting Phase




FIT

Goal

   2. Validation phase

VALID

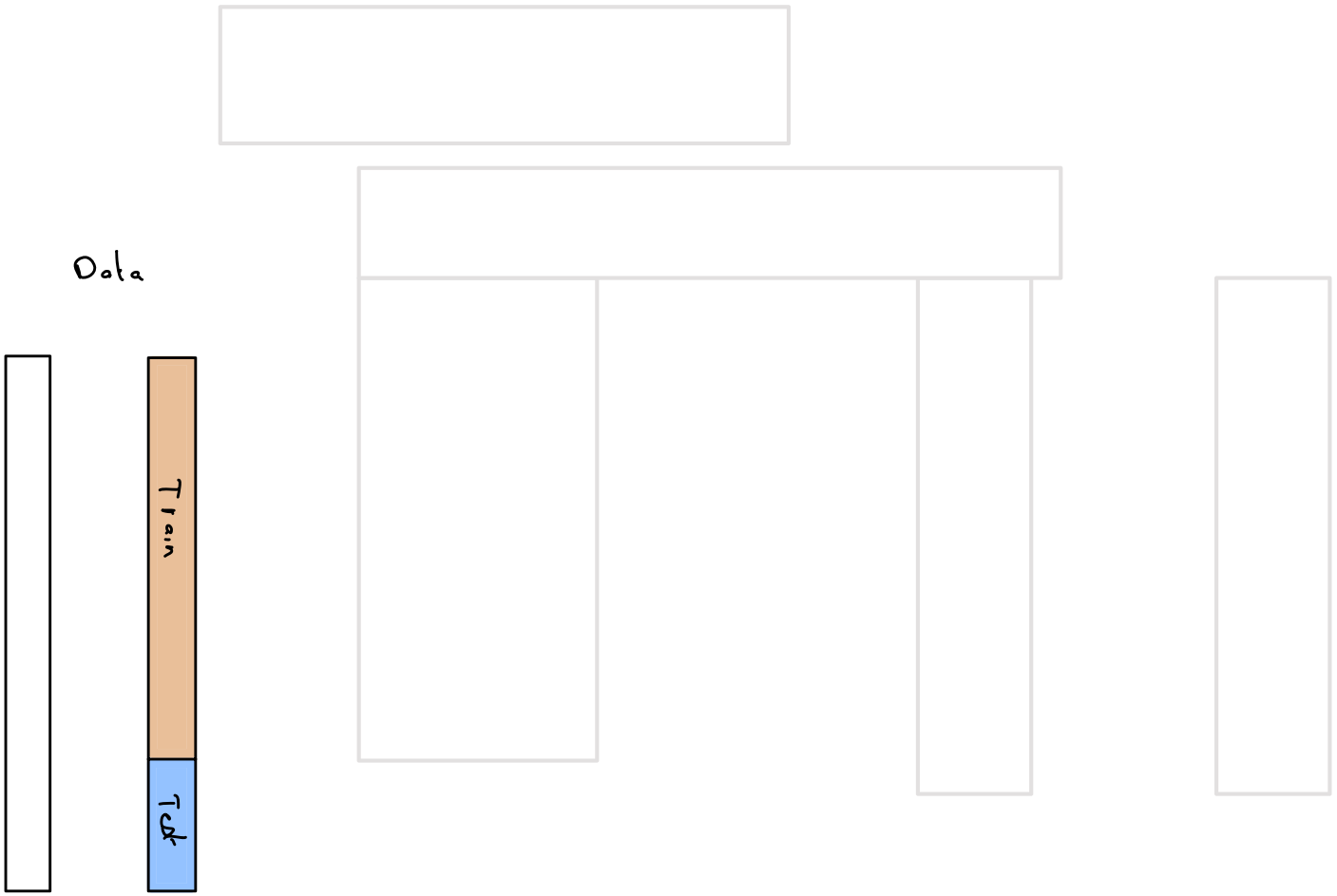
Possible models & Hyperparameters

Goal

2. Test phase
3. Deployment

# Cross Validation

Goal:



# One algorithm to rule SML: Cross validation

## Enables to select the best method within mutliple methods

Entrée [7]:

```
models = {
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(n_estimators=10,random_state=42),
    "Logistic Regression": LogisticRegression(multi_class="multinomial", solver="lbfgs", max_iter=100, random_st
    "MLP Classifier": MLPClassifier(random_state=42),
    "SVM": SVC()
}
```

Entrée [12]:

```
cv_results = {}

for name, model in models.items():
    kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
    cv_score = cross_val_score(model, X_train, y_train, cv=kfold, scoring="accuracy")
    cv_results[name] = cv_score
    print(f"{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]:

```
# Convert results to DataFrame for better visualization
results_df = pd.DataFrame(cv_results)
results_df
```

Out[18]:

| | 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 |

Entrée [17]:

```
# Descriptive statistics can also be helpful
results_df.describe()
```

Out[17]:

| | 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 | 0.885714 | 0.953571 |
| **max** | 0.790179 | 0.898214 | 0.883929 | 0.888393 | 0.958929 |

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 [ ]:

```python
from sklearn.model_selection import GridSearchCV

# Example for SVM
param_grid = {
    'C': [1, 10, 100],
    'gamma': [0.001, 0.01, 0.1],
}
grid_search = GridSearchCV(SVC(), param_grid, cv=3, verbose=2)
grid_search.fit(X_train_scaled[:10000], y_train[:10000])  # Reduced data size for quicker execution

print("Best parameters:", grid_search.best_params_)
print("Best cross-validation accuracy:", grid_search.best_score_)
best_model = grid_search.best_estimator_
final_predictions = best_model.predict(X_test_scaled)
print("Test set accuracy:", accuracy_score(y_test, final_predictions))
```

Entrée [43]:

```python
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 | mea |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 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

# 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:

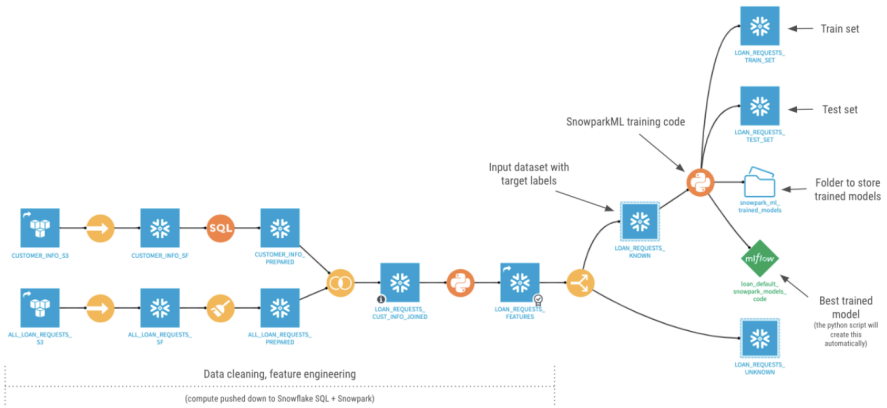



Figure: Dataiku

`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:

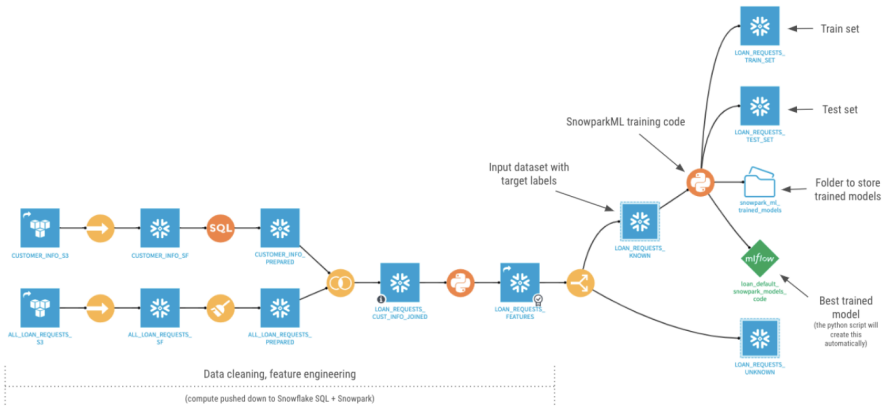



Figure: Dataiku

`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.
   → Which method is suitable depending on Performance/Speed/Interpretability trade-offs
   → Better choices for hyper-parameters to be chosen

# Wooclap time!

1 Allez sur wooclap.com

2 Entrez le code d'événement dans le bandeau supérieur

Code d'événement
TAMXAU

1 Envoyez @TAMXAU au 06 44 60 96 62

2 Vous pouvez participer

Désactiver les réponses par SMS

## What is the problem of overfitting

1. I will have a  bad error on the train set
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
3. Find the best hyperparameters in terms of validation error
4. Fit the model
5. Find the best parameters in terms of validation error

Entrée [7]:

```
models = {
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(n_estimators=10,random_state=42),
    "Logistic Regression": LogisticRegression(multi_class="multinomial", solver="lbfgs", max_iter=100, random_st
    "MLP Classifier": MLPClassifier(random_state=42),
    "SVM": SVC()
}
```

Entrée [12]:

```
cv_results = {}

for name, model in models.items():
    kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
    cv_score = cross_val_score(model, X_train, y_train, cv=kfold, scoring="accuracy")
    cv_results[name] = cv_score
    print(f"{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]:

```
# Convert results to DataFrame for better visualization
results_df = pd.DataFrame(cv_results)
results_df
```

Out[18]:

| | 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 |

## 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
5. It computes 5 scores for each method, splittin the train data
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
9. 25 fits were performed overall
10. 5 fits were performed overall

# Reminder on Linear and Logistic regression - towards DT

# Limits of Linear and logistic regression - towards DT

# 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 $\longrightarrow$ allow complex dependencies.

# Decision Tree

**Goals of Decision Trees**

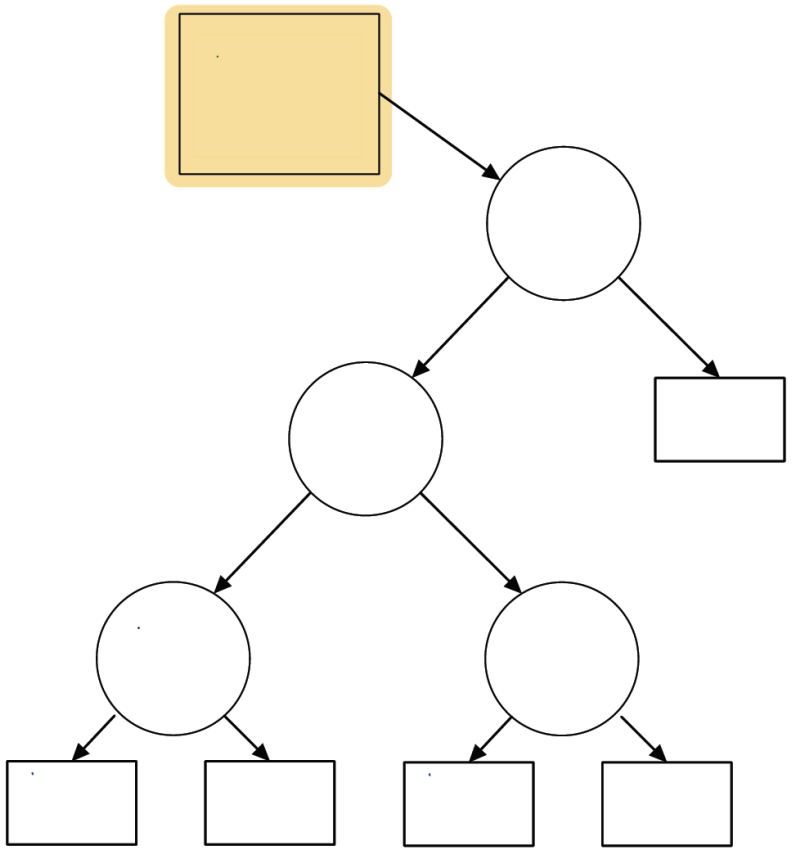
1. Supervised Learning: solve the classification or regression task
2. Provide some understanding: what on this example allows me to classify it? What are the important features ?
3. Go beyond linear functions $\longrightarrow$ allow complex dependencies.

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

- Root = observation = input
- Nodes = tests
- Branches = possible outcomes
- Leaves = final decision = output



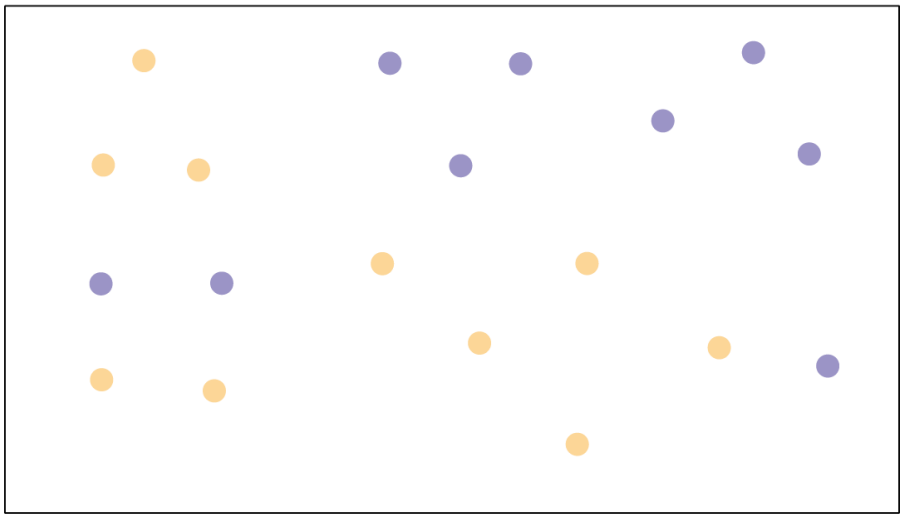
# Training: how to build a tree.

**Key question !**

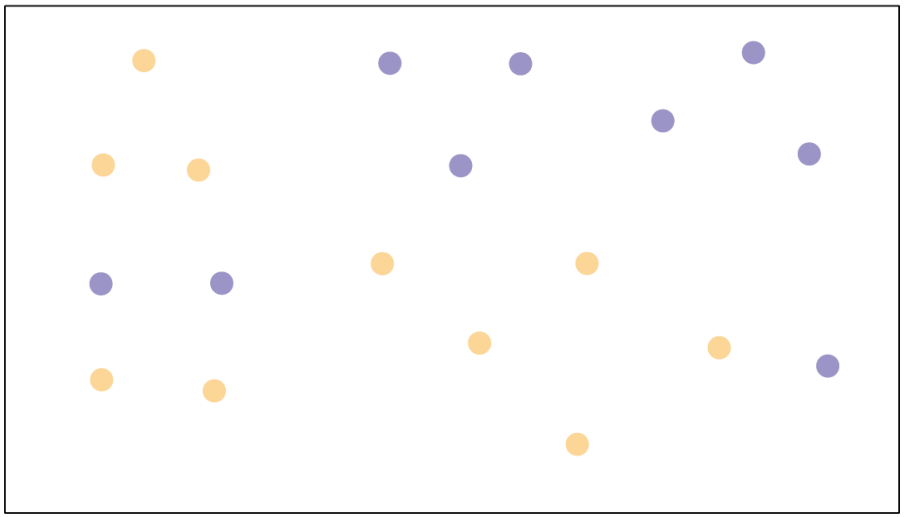
- What is a good tree?
- What do I need to choose?
- How to choose?

What do you think?

# Building trees : CART



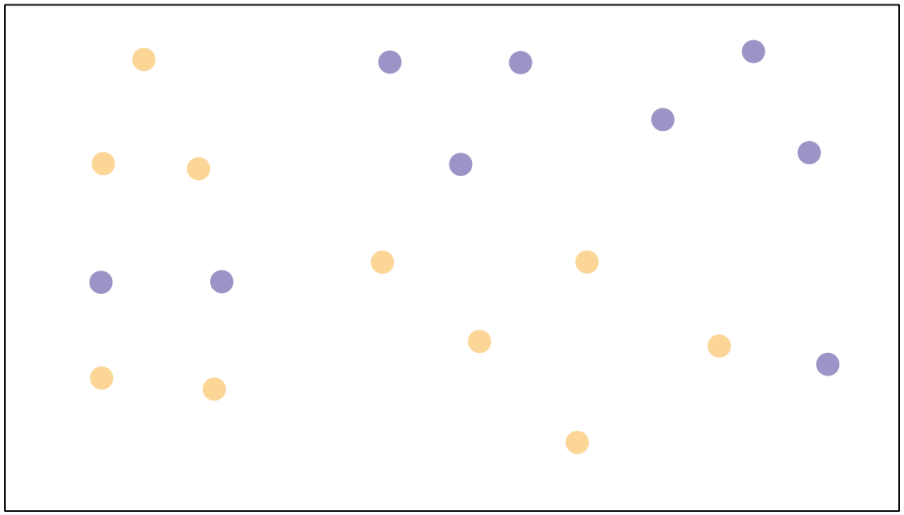
# Building trees : CART



**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**.

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

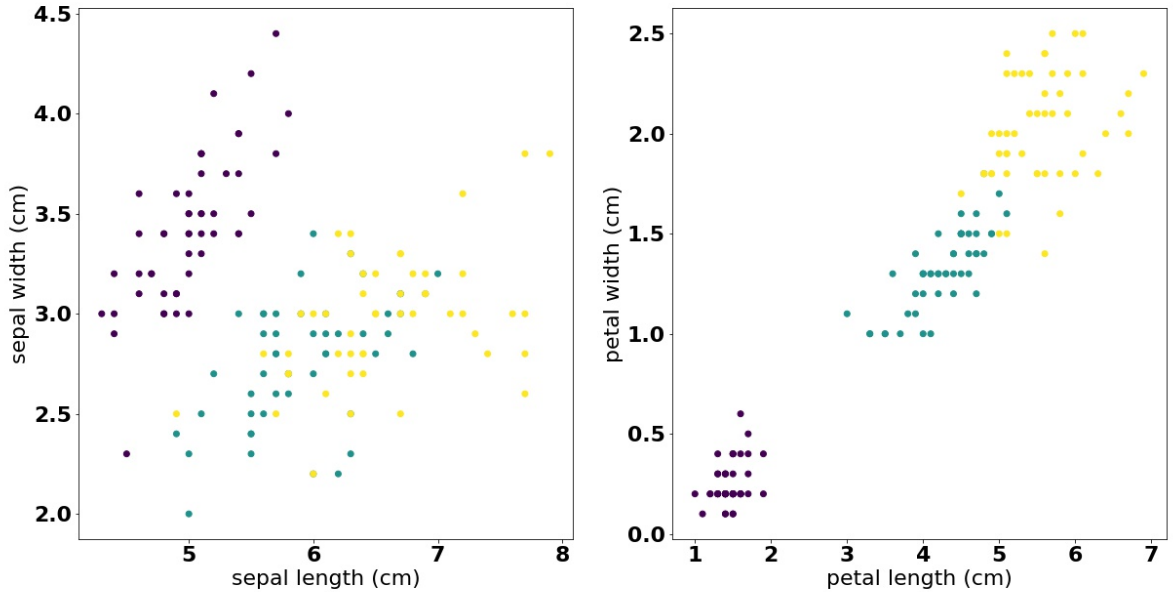
For two classes, related to the variance if classified as Bernoulli r.v..

# Example: Iris Dataset

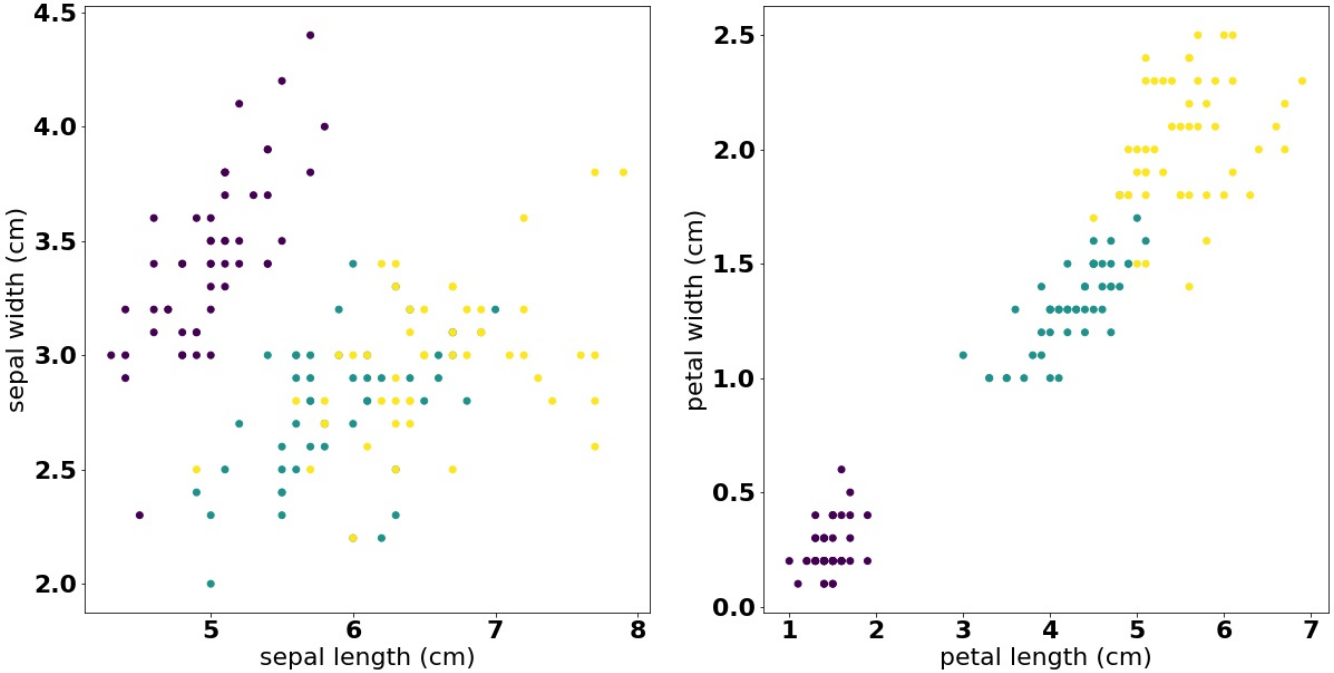
Figure: Iris: setosa, versiocolor, virginica

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

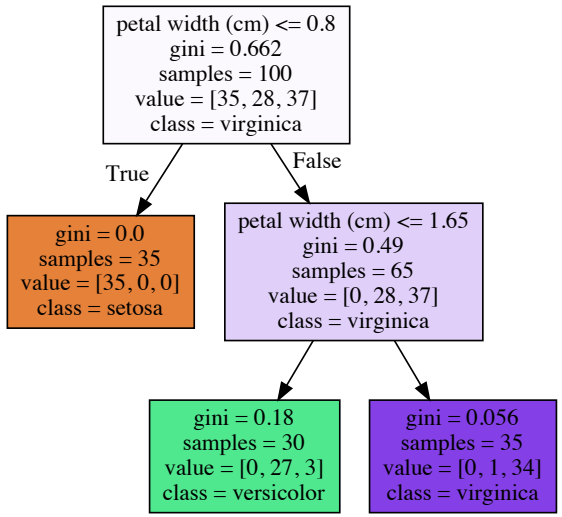
`https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html`

# Example: Iris Dataset

## What do we need to specify ?

```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
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

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

#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)

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

# Plot tree
plt.figure(figsize=(5,4))
plot_tree(clf, feature_names = iris.feature_names,
               class_names=iris.target_names,
               filled = True)
plt.show()
```

## Output:




## Example:

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

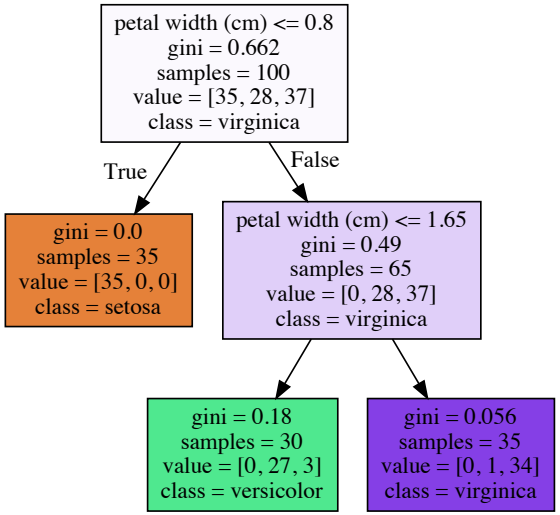
## Returns:

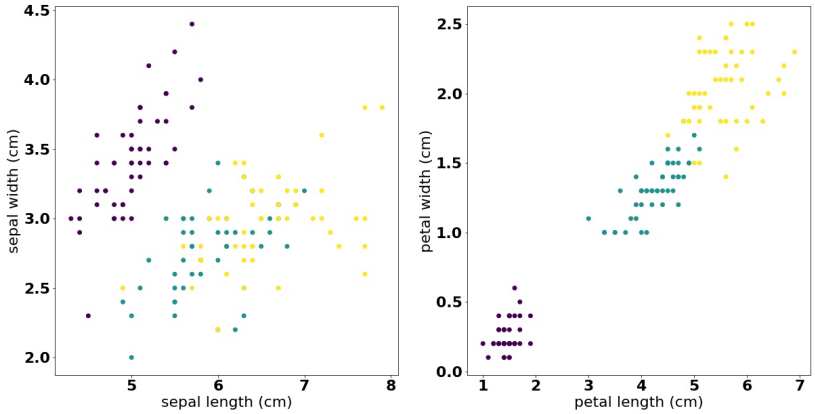
(array([4.8, 3.1, 1.6, 0.2]), 'setosa')

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

# Example: Iris Dataset

Output:




## Example:

```
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 ?

# Wooclap time!

1 Allez sur wooclap.com

2 Entrez le code d'événement dans le bandeau supérieur

Code d'événement
TAMXAU

1 Envoyez @TAMXAU au 06 44 60 96 62

2 Vous pouvez participer

Désactiver les réponses par SMS

## Decision trees and Supervised Learning

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

## Which of the following statements are true about Decision Trees?

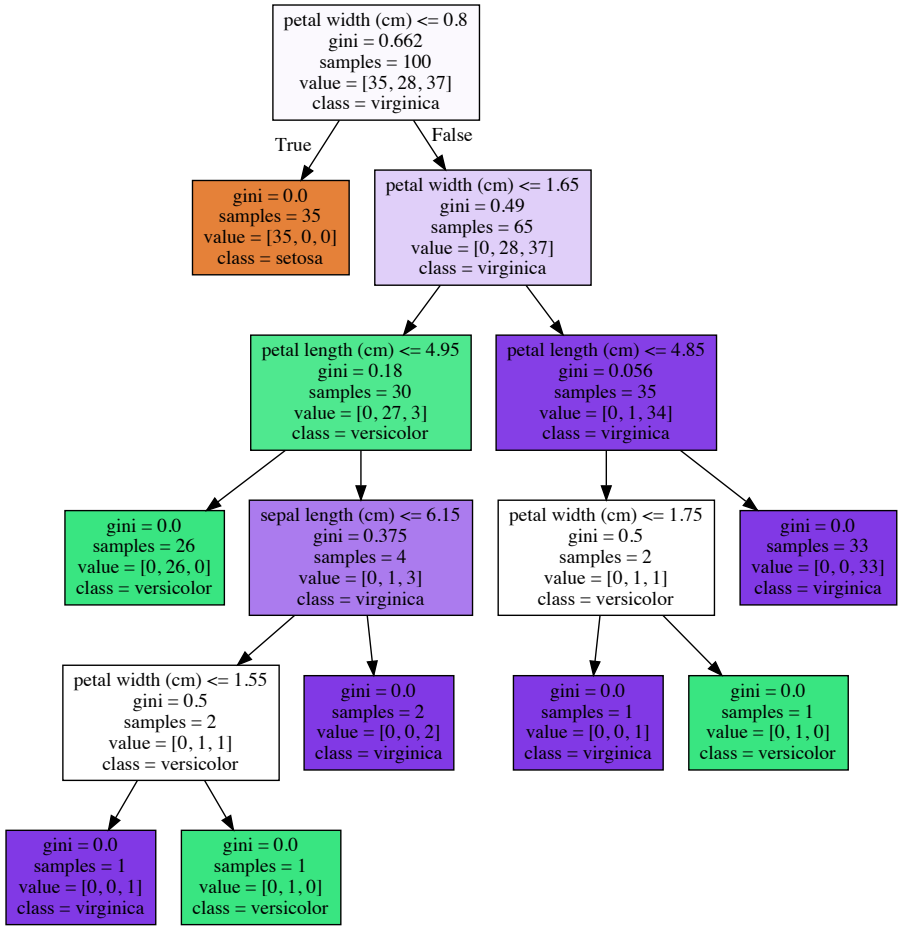
1. They are very stable
2. They provide an interpretable approach
3. They are able to approach the "optimal function" for nearly any data distribution (e.g. beyond linearly separable data)

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

1. The same feature cannot be used twice consecutively"
2. The same feature cannot be used twice overall"
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
5. To define a new split, I only use one example

# Deeper trees

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




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

# Stopping / Pruning

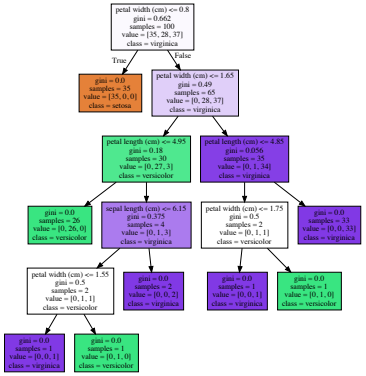
Trees that achieve perfect classification may suffer from over-fitting. (see example in the lab)

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.

# 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 !

# Outline

# 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 ?

# 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 !

# 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.

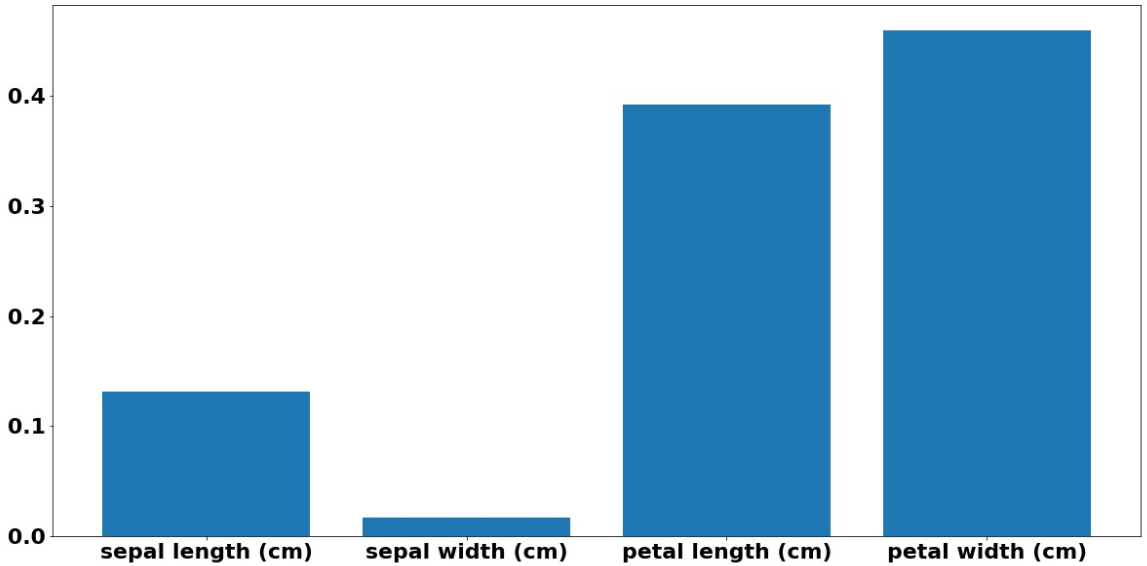



Figure: Feature importane output for a Random Forest Classifier in Python

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

What about Regression ?

# 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.

# 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.