Scikit-learn

Data science
et
Machine learning avec python

CONSULTING BSTORM

1.Introduction

CONSULTING BSTORM

1. Introduction

Comment définir la data science ?

=> Résoudre des **problèmes complexes** avec des données

E.g.:

- **Classifier** des utilisateurs pour leur proposer des publicités pertinentes
- **Identifier** des pièces défectueuses sur une ligne de production
- **Prévoir** des pannes sur une ligne de production
- **Reconnaissance** faciale
- **Prévoir** des maladies(e.g. un AVC)
- **Traducteurs** automatiques

1. Introduction

Ne pourrait-on pas le faire manuellement ?

Non, pour plusieurs raisons:

- Trop de travail
- **Trop d'informations** pour un être humain
- L'information est **trop subtile** que pour être détectée par un être humain

=> Il faut pouvoir **automatiser** ce traitement des données

1. Introduction

Comment ?

=> avec de l'intelligence artificielle:

- Des données
- Des algorithmes
- Des outils statistiques
- Des outils mathématiques

Mais avant tout c'est:

- Un esprit analytique
- Une méthode rigoureuse

1. Introduction

Il existe toute une série d'algorithmes basés sur des outils statistiques et mathématiques que l'on peut **entraîner** avec les données à disposition

Une fois entraîné, on peut l'utiliser pour faire des **prédictions sur nos données futures**

1. Introduction

Un dataset est constitué:

- Samples: les **observations**, les lignes
- Features: les **variables explicatives**, les colonnes, les **X**
- Label(target): facultatif, une **valeur cible** que l'on souhaite déterminer, **y**

Deux grandes catégories d'apprentissage:

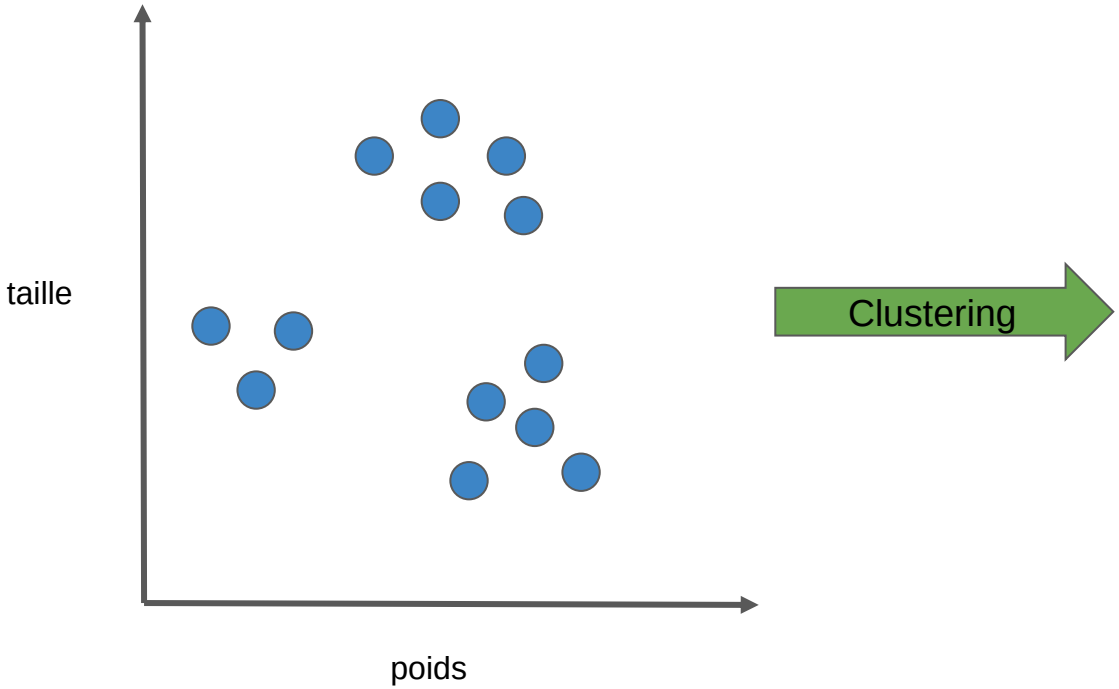
- **Non supervisé**: pas de target
- **Supervisé**: une ou plusieurs targets

1. Introduction

4 grandes catégories de problèmes:

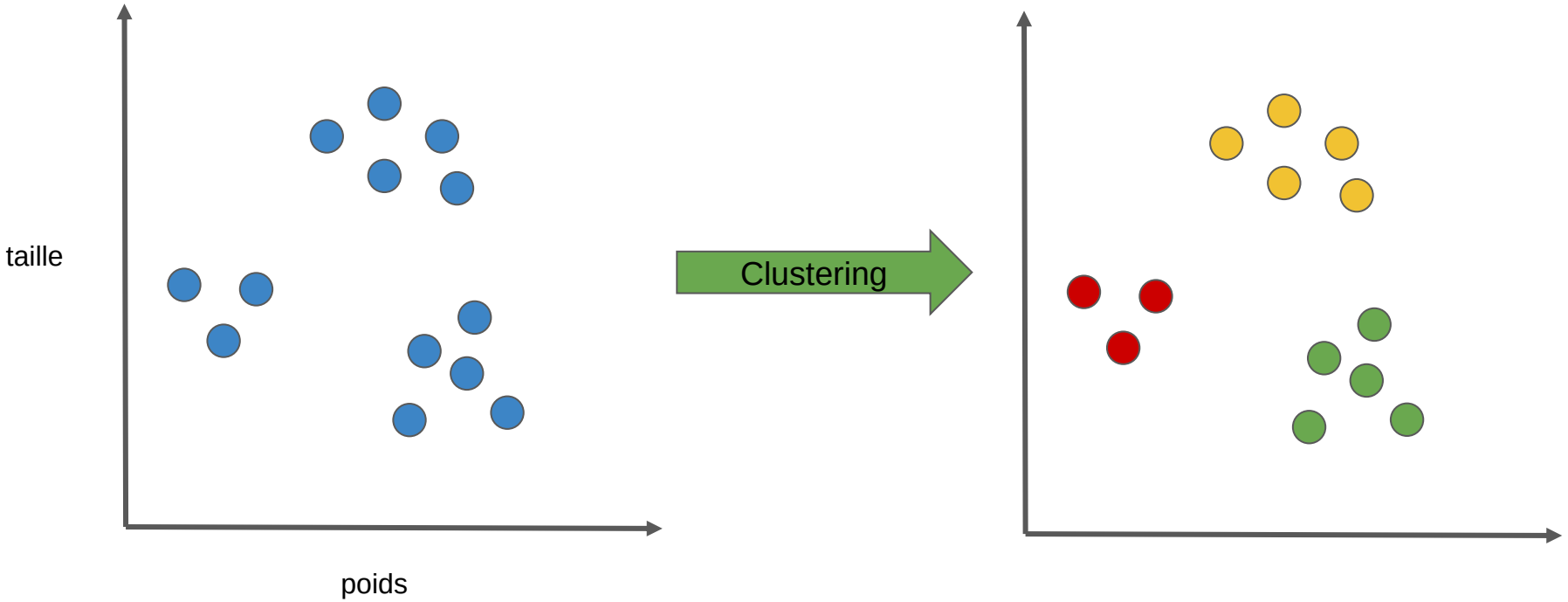
- **Clustering**: non supervisé, on regroupe les individus semblables entre eux
- **Régression**: supervisé, on cherche à déterminer la valeur d'une variable quantitative continue(e.g. le prix d'une maison)
- **Classification**: supervisé, on cherche à déterminer la valeur d'une variable discrète(e.g. chien ou chat)
- **Ranking**: supervisé, entre la régression et la classification

1. Introduction



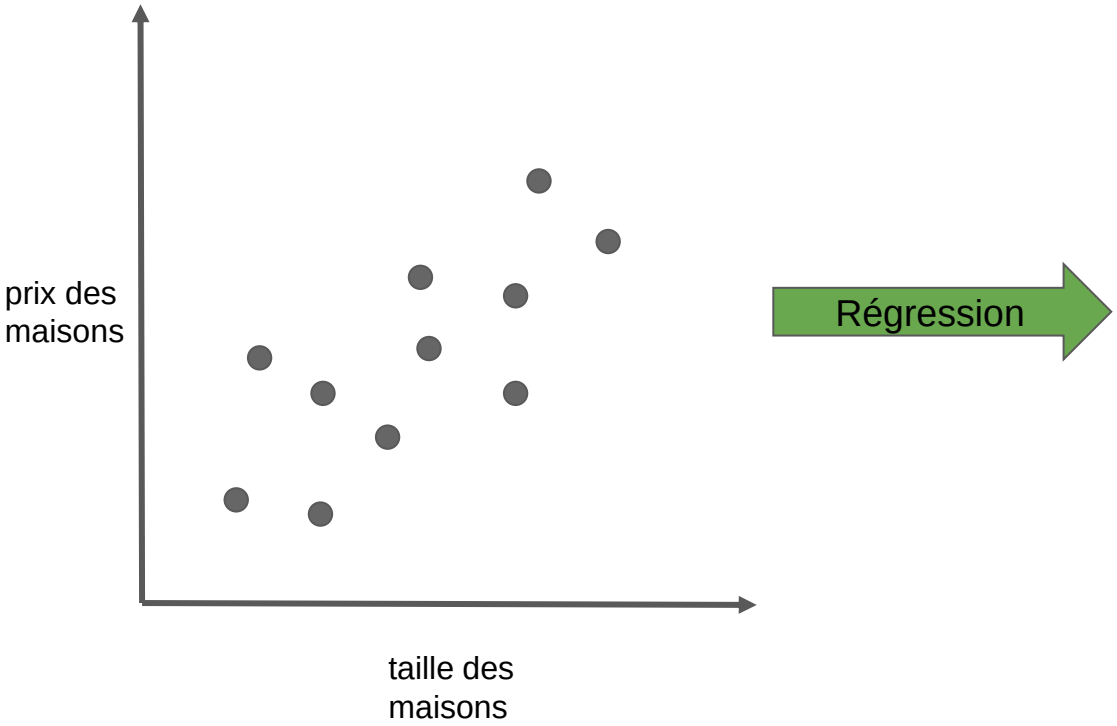
Scikit-learn

1. Introduction



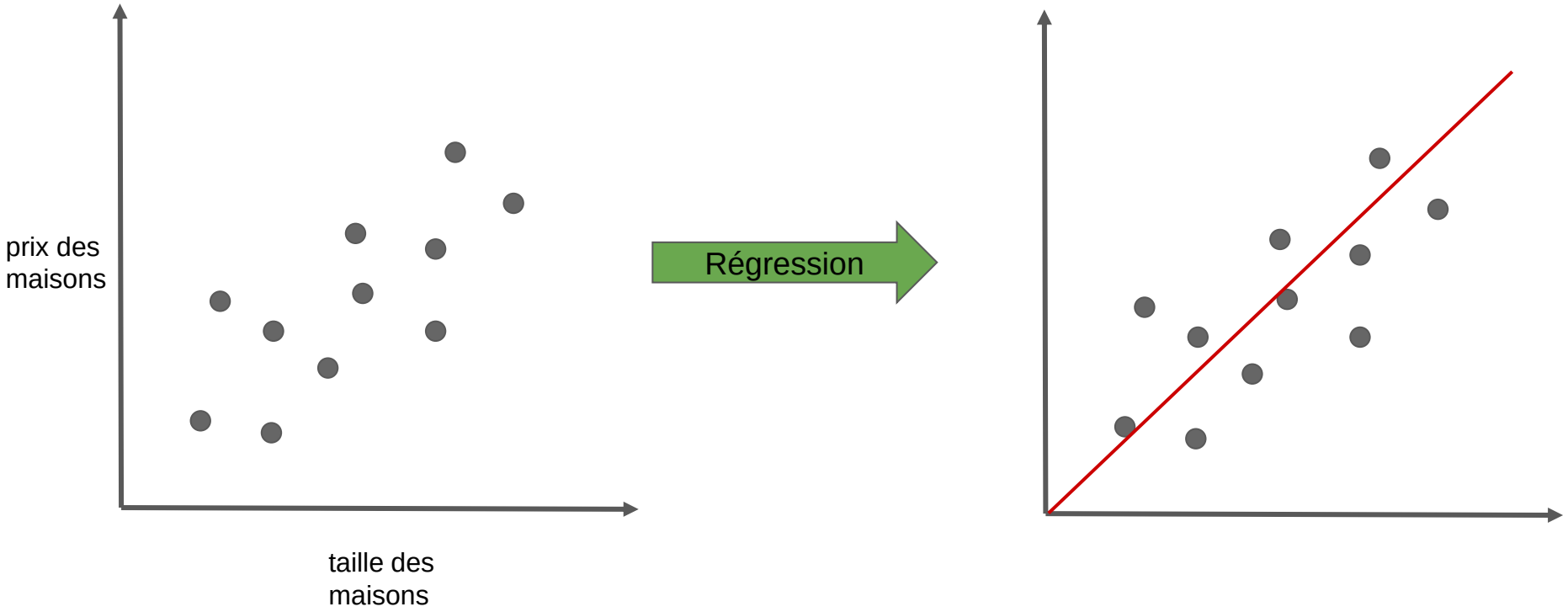
Scikit-learn

1. Introduction



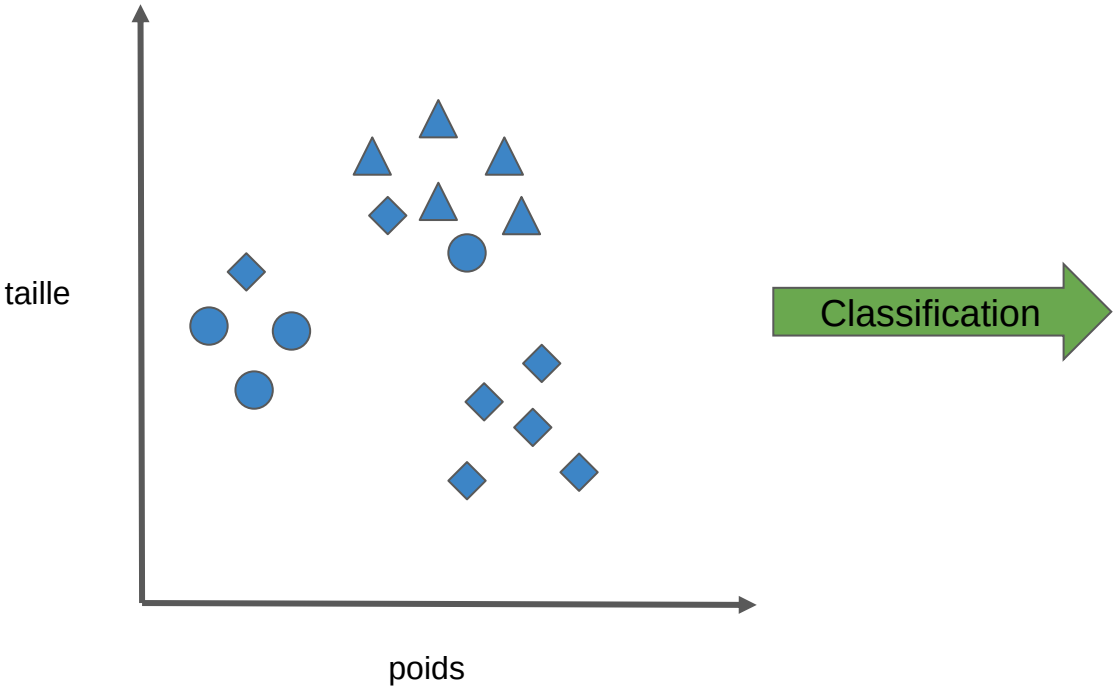
Scikit-learn

1. Introduction



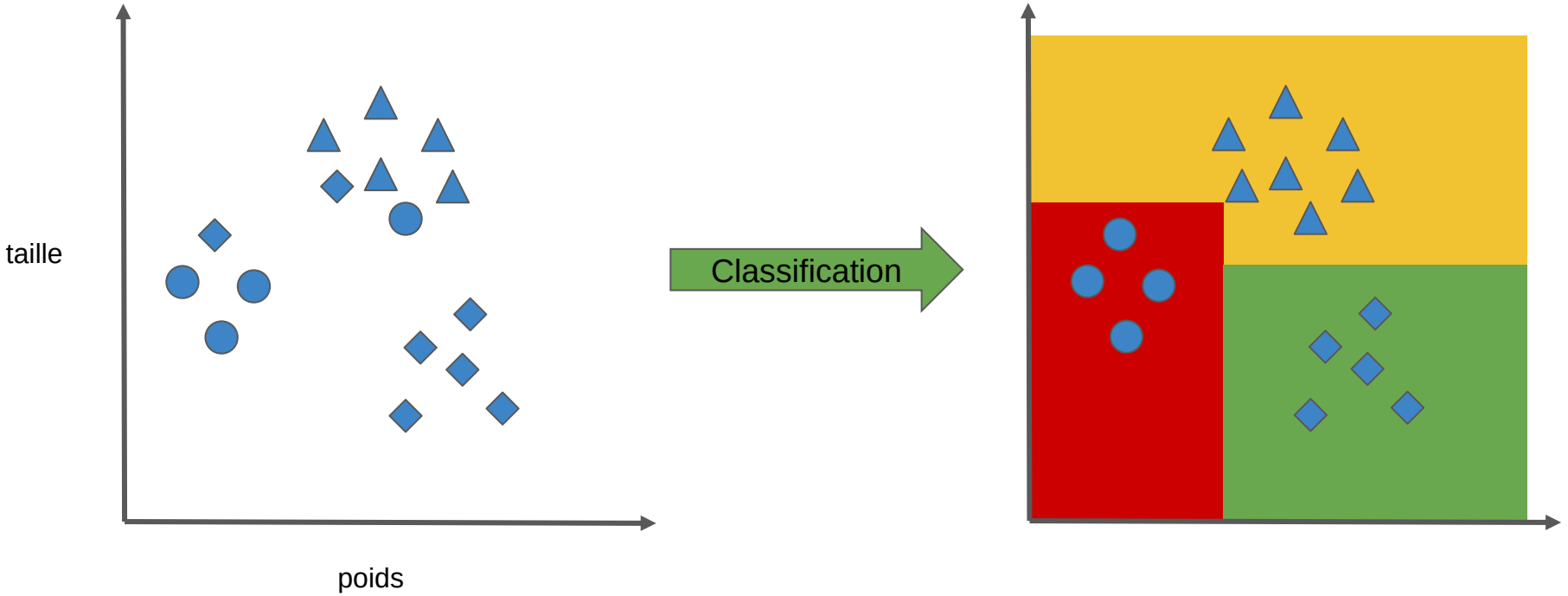
Scikit-learn

1. Introduction



Scikit-learn

1. Introduction



Scikit-learn

1. Introduction

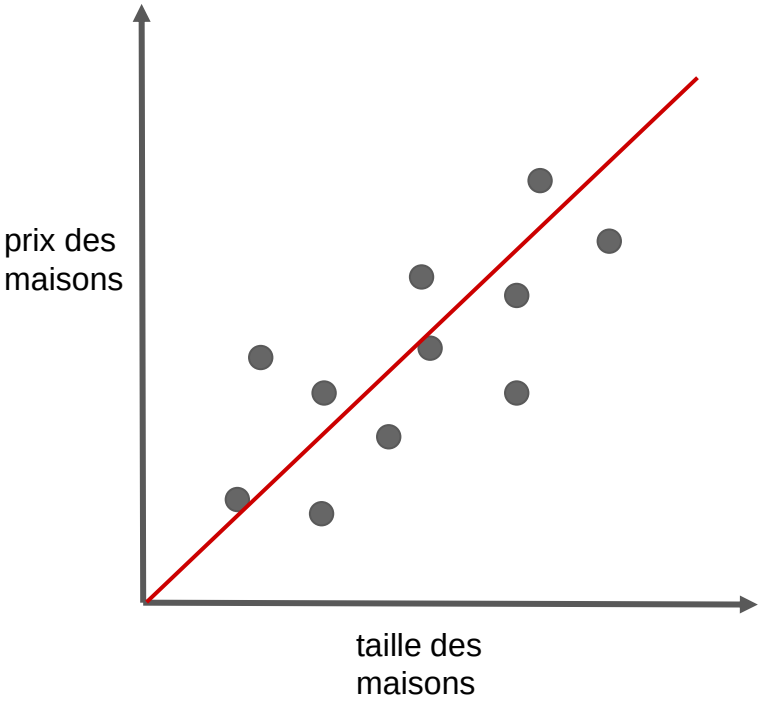
Concrètement que signifie entraîner un algorithme?

2 exemples pour mieux comprendre:

- La régression linéaire et de manière plus générale les modèles linéaires
- Les arbres de décision

1.1. la régression linéaire

1.1. La régression linéaire



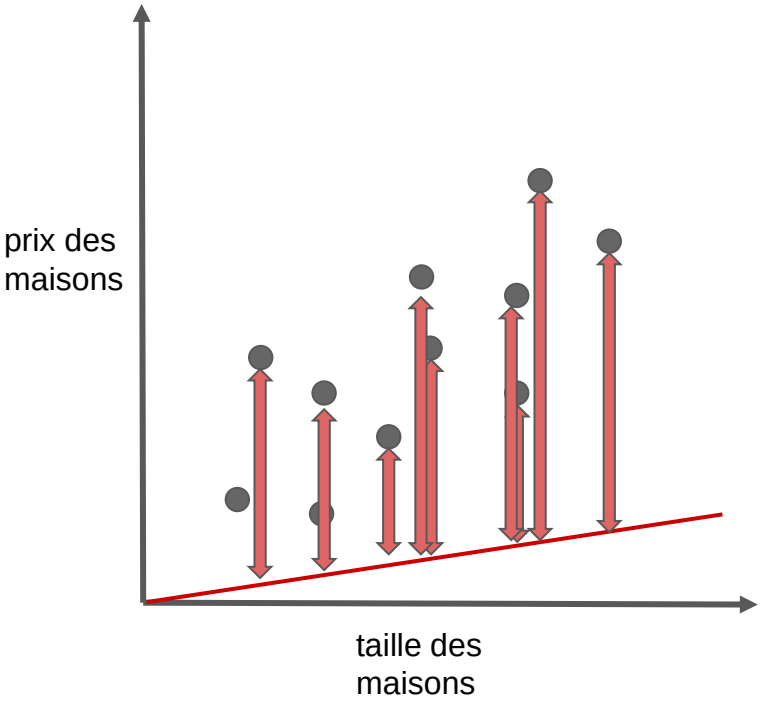
Nous avons tracé une **ligne** passant entre nos points

Rappel, une droite:

f(x) = ax+b

- a = coefficient
- b = constante
- x = variable (taille de notre maison)
- f(x) = pour faire simple, ici notre target

1.1. La régression linéaire

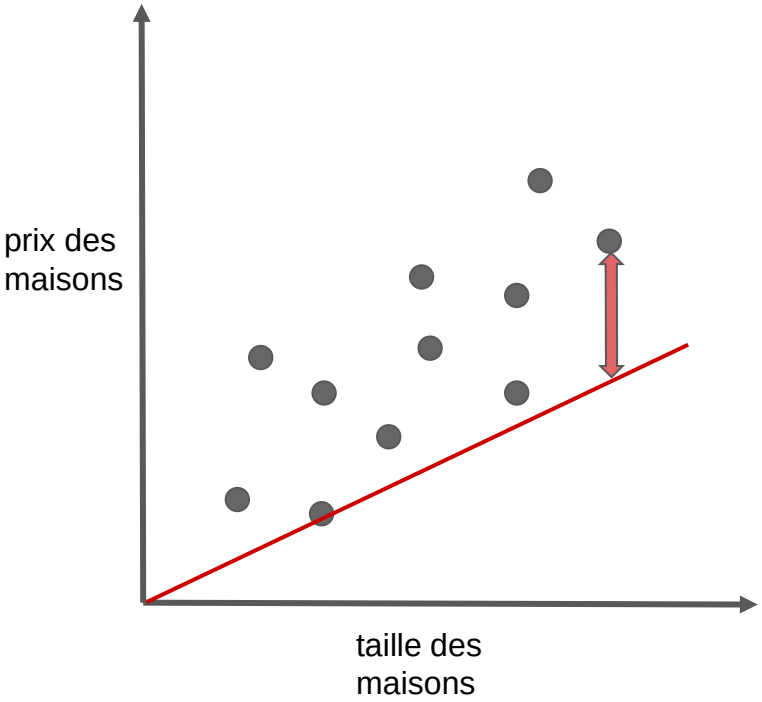


Comment trouver le bon coefficient ?

Par essai-erreur:

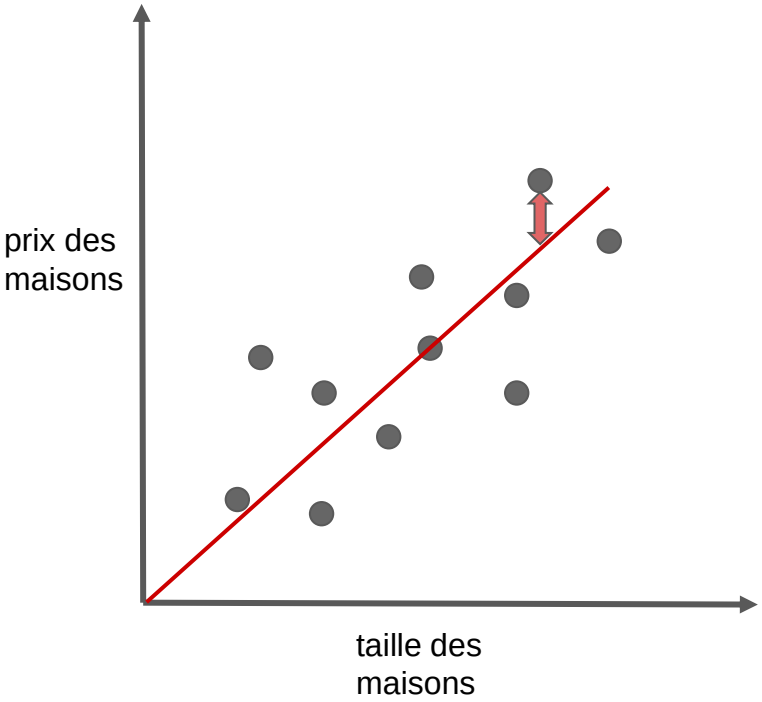
- on choisit la première fois un coefficient aléatoirement
- on regarde la quantité d'erreur
- on modifie notre coefficient

1.1. La régression linéaire

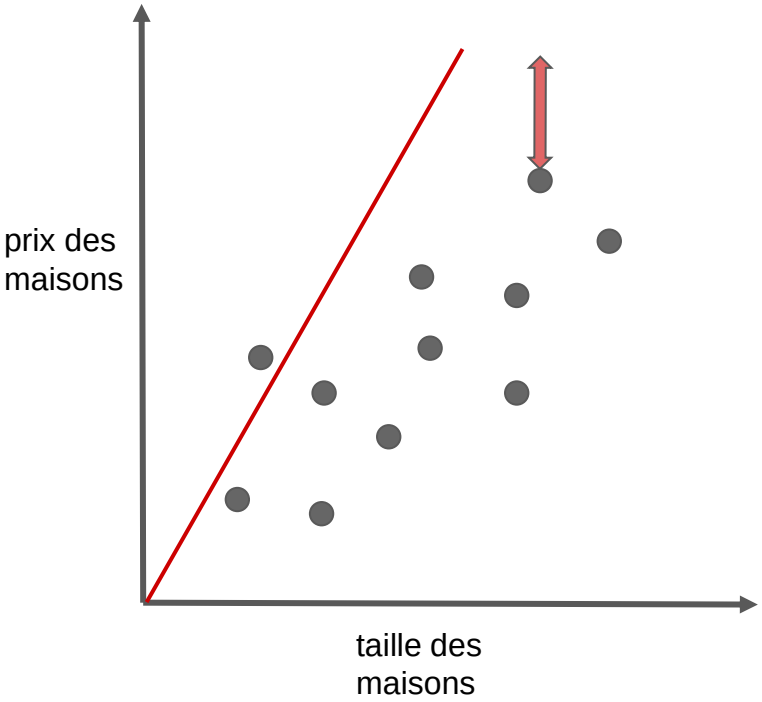


A chaque itération, on calcule nos erreurs et on modifie notre coefficient

1.1. La régression linéaire



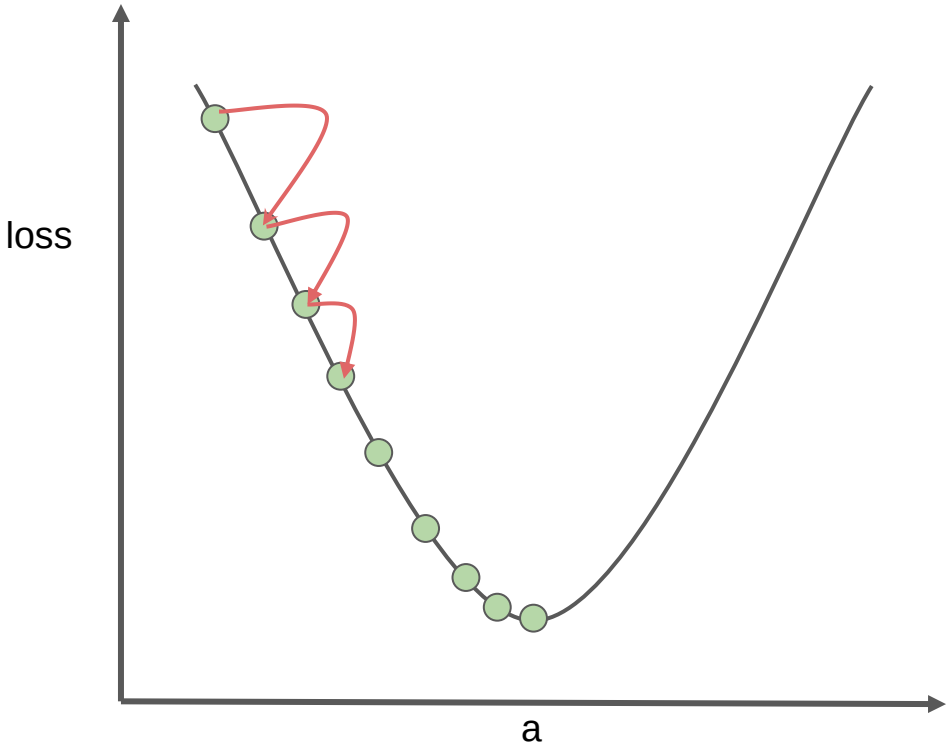
1.1. La régression linéaire



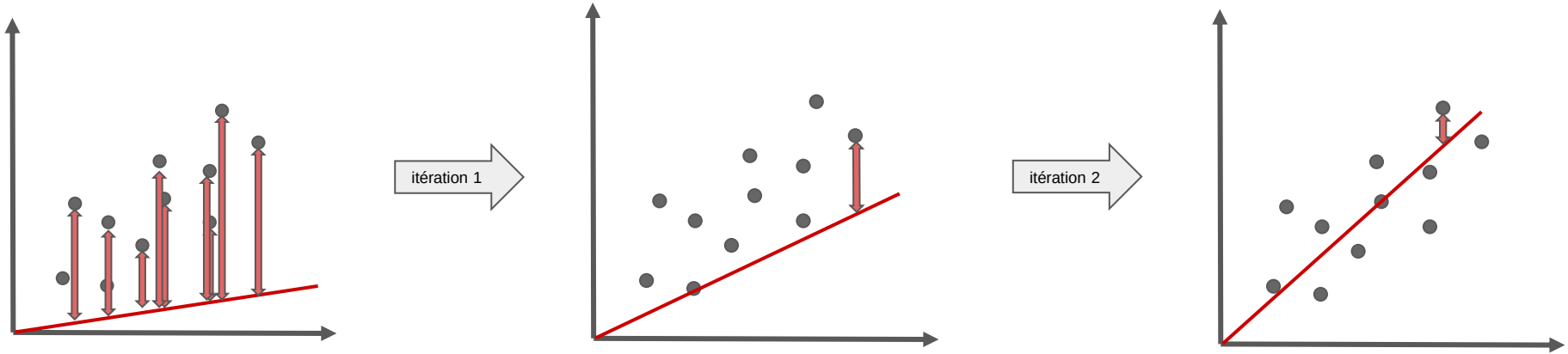
Nos erreurs remontent, nous sommes allés trop loin

L'entraînement est terminé

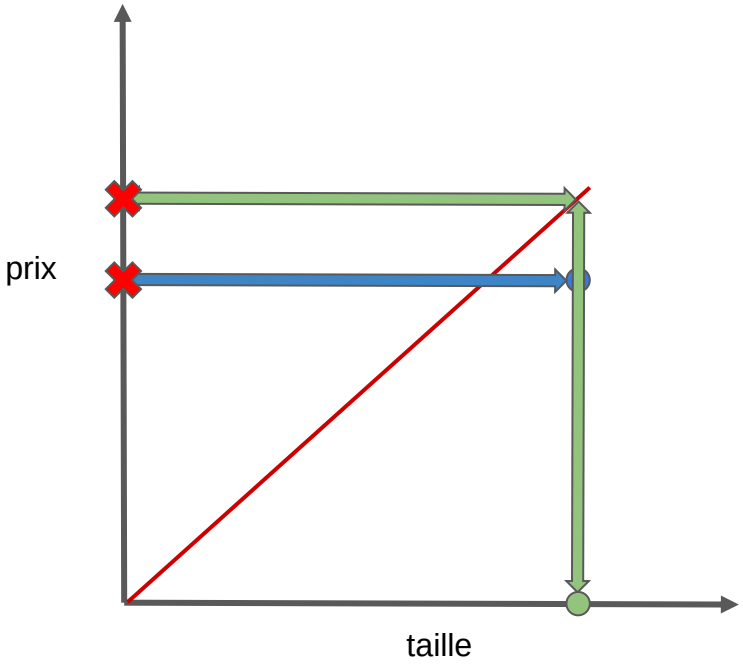
1.1. La régression linéaire



1.1. La régression linéaire



1.1. La régression linéaire



Désormais nous pouvons prédire le prix d'une maison

Avec en bleu la valeur réelle et en vert la prédiction de notre modèle

1.1. La régression linéaire

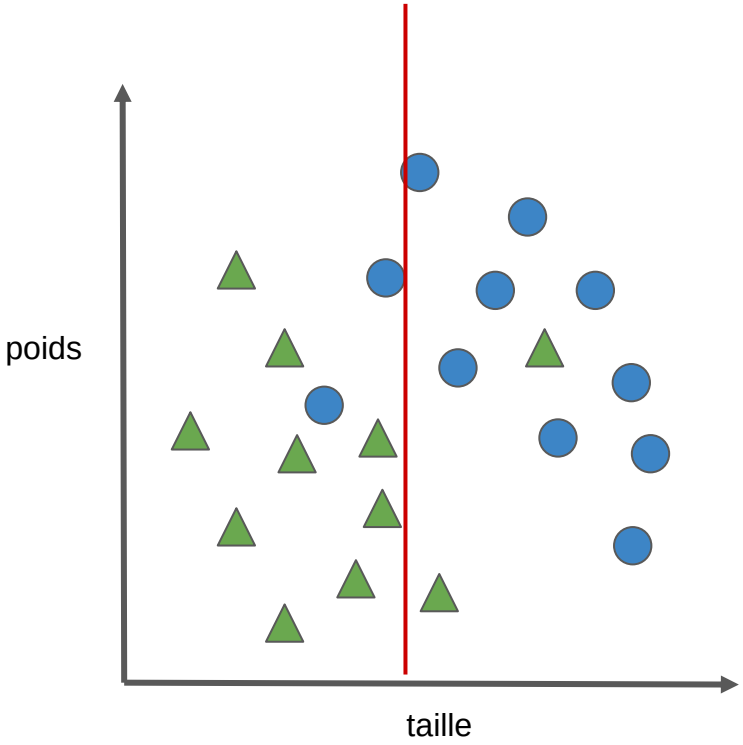
Si on a plus qu'une variable?

$f(x) = a_1x_1+a_2x_2+b$

prix = a_1*taille maison+a_2* taille jardin+b

1.2. La régression logistique

1.2. La régression logistique



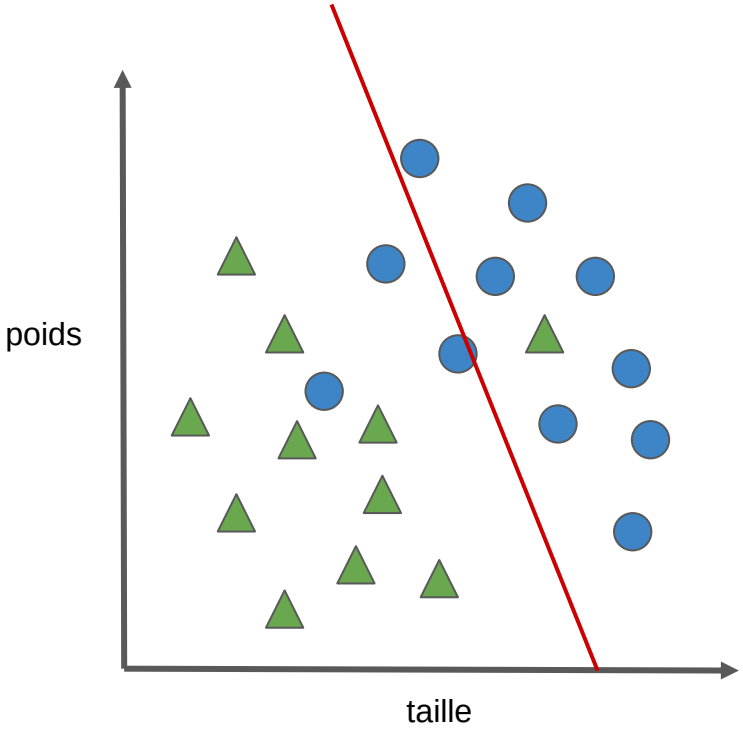
=> Même principe que la régression linéaire

=> **Classification**

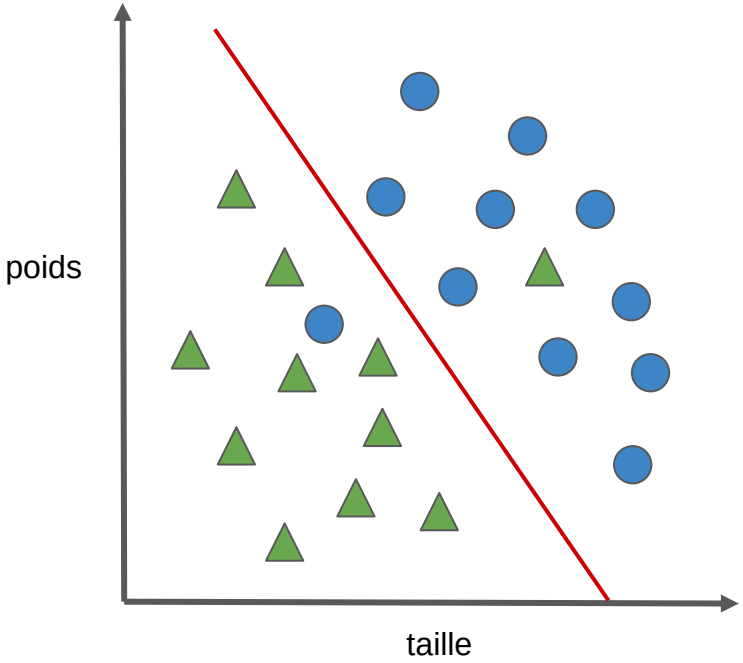
Avec la régression linéaire on **minimise les erreurs**

Avec la régression logistique on cherche à **maximiser la distance** séparant nos classes

1.2. La régression logistique



1.2. La régression logistique



1.3. Les arbres de décision

1.3. Les arbres de décision

=> Pour la régression ou la classification

=> Non linéaire

=> Très simple à interpréter

1.3. Les arbres de décision

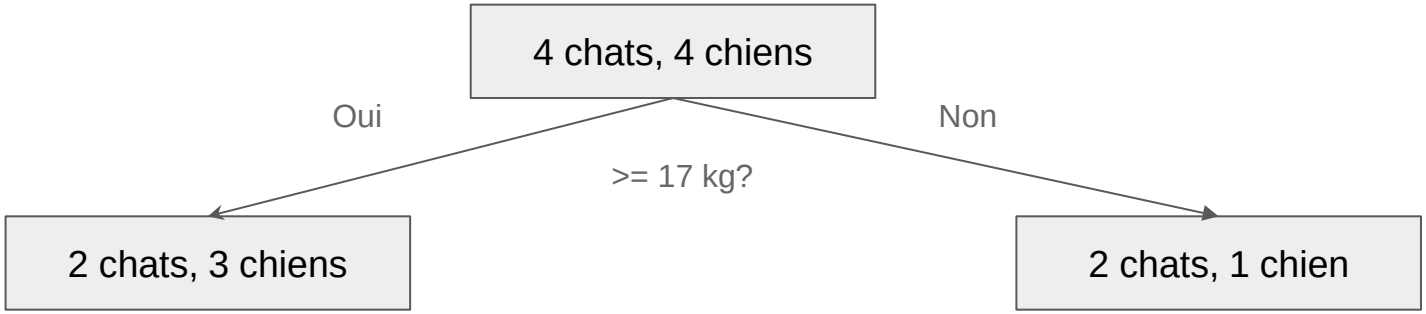
poids(kg)	target(classe)
13	chat
17	chat
26	chien
32	chien
18	chien
8	chat
15	chien
20	chat

Nous avons deux classes possibles et une seule feature pour les discriminer

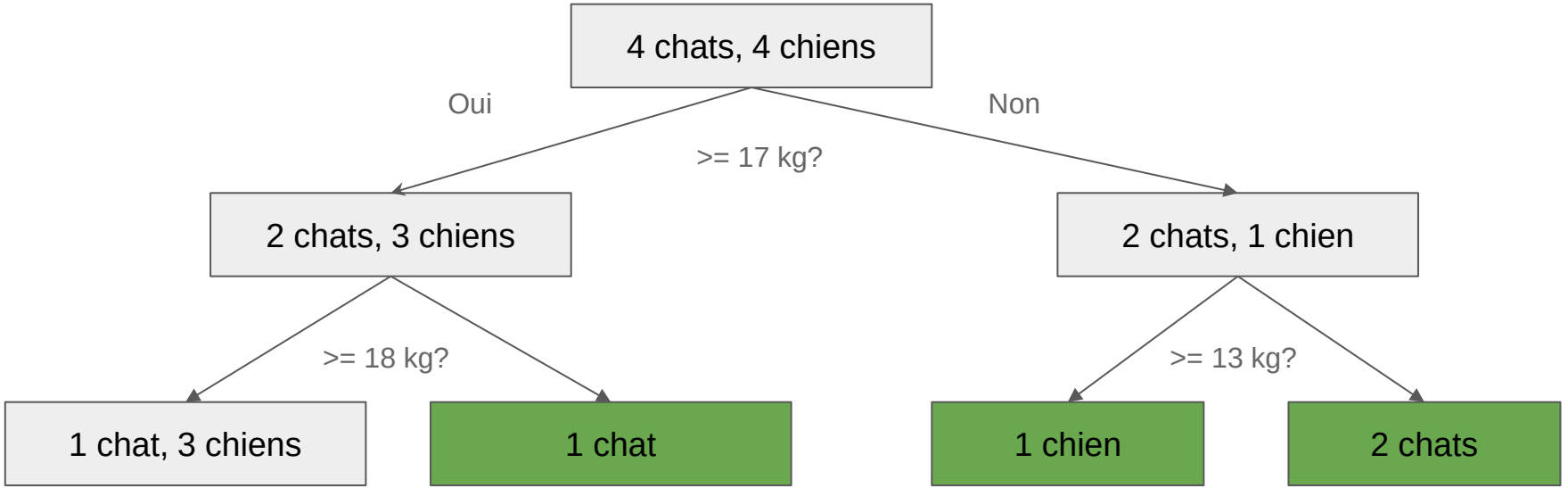
=> Les arbres de décision se construisent en se posant **une succession de questions**

Nous prenons aléatoirement une valeur pour séparer nos données en deux

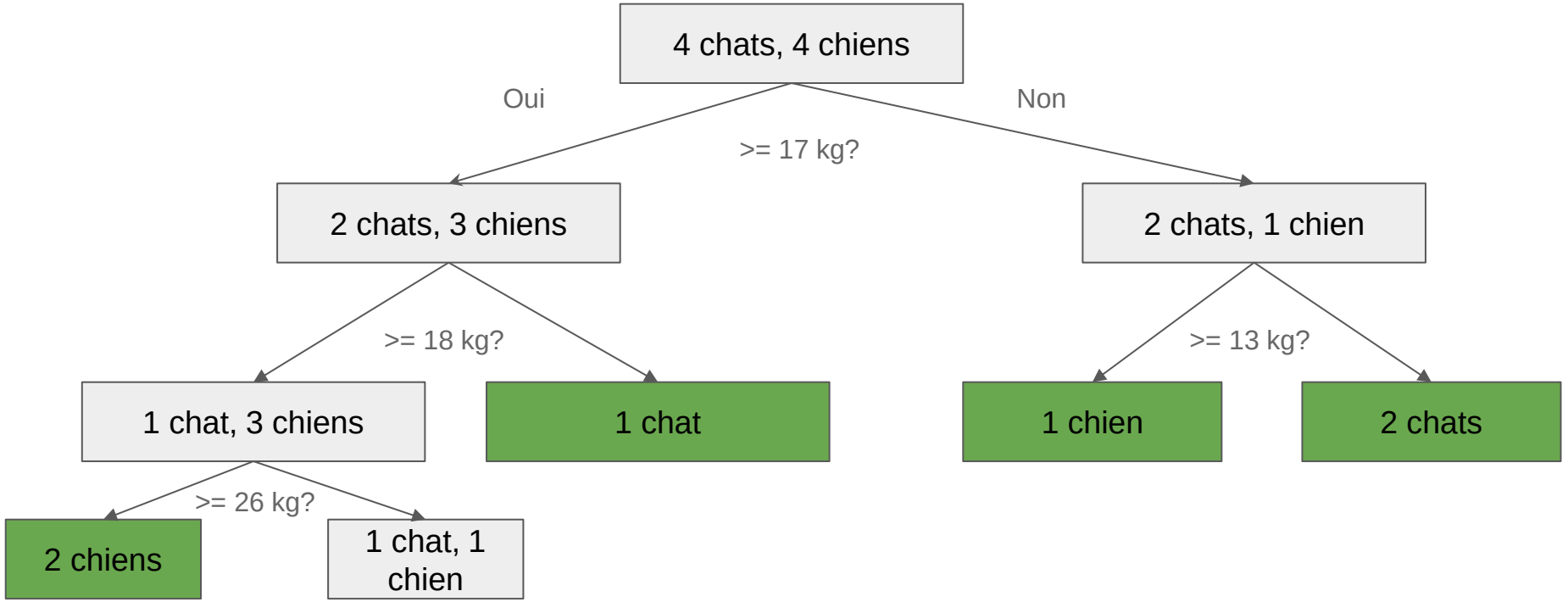
1.3. Les arbres de décision



1.3. Les arbres de décision

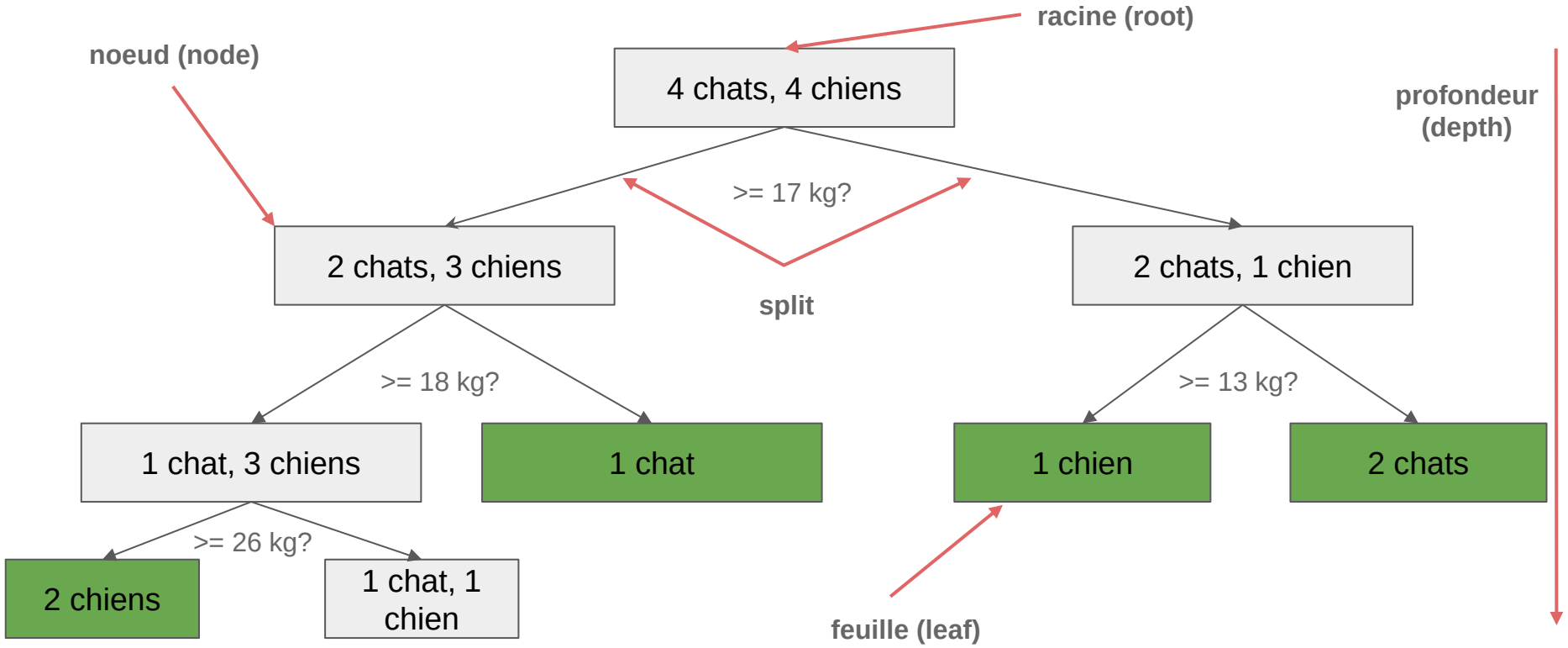


1.3. Les arbres de décision



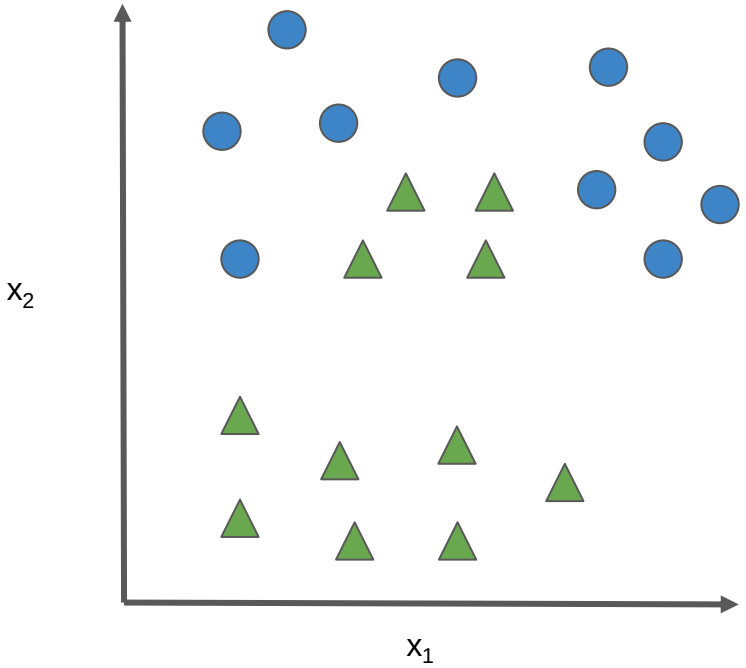
Scikit-learn

1.3. Les arbres de décision

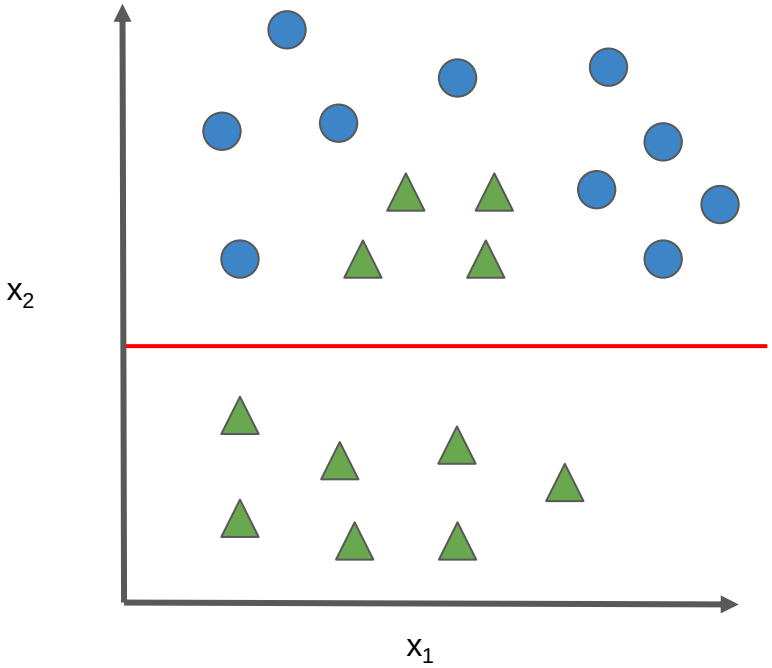


Scikit-learn

1.3. Les arbres de décision



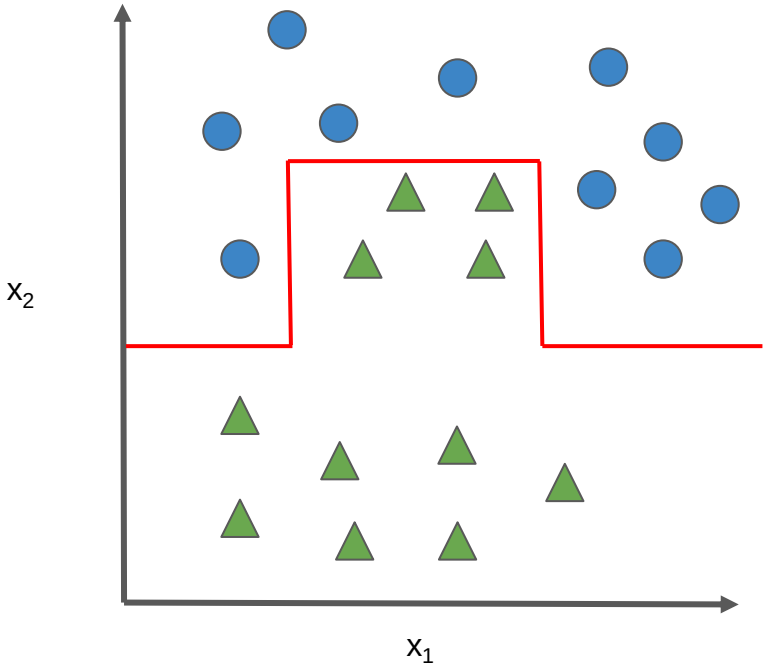
1.3. Les arbres de décision



Régression logistique

Scikit-learn

1.3. Les arbres de décision



Arbre de décision

1.3. Les arbres de décision

Dans la pratique on utilise un critère pour sélectionner notre valeur:

- indice de Gini
- entropie

S’il y a plusieurs features:

- sélection avec test statistique

1.3. Les arbres de décision

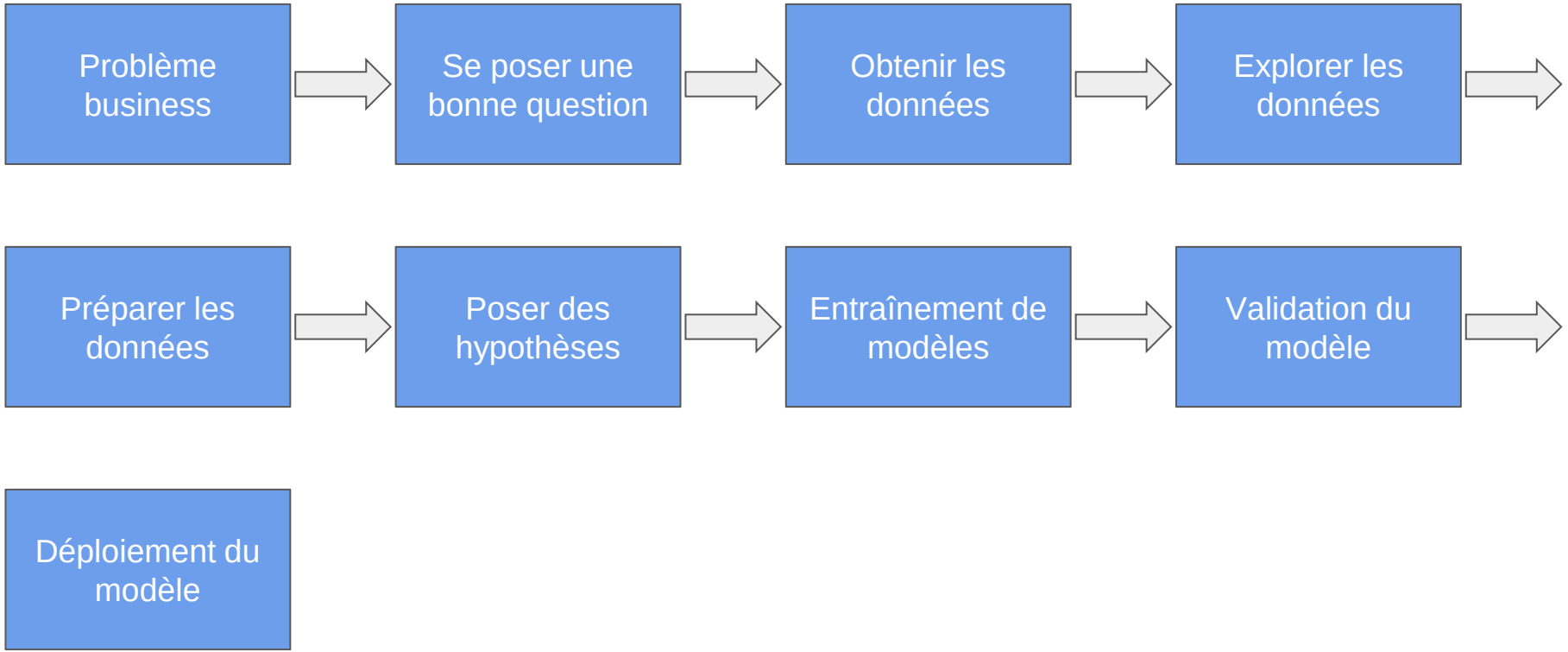
Le machine learning:

- Sur des datasets de taille raisonnable (< 1000000 points)
- Sur des données très structurées

Sklearn, librairie très complète et Open source pour le machine learning

1.4. Etapes de travail

1.4. Etapes de travail



2. Les méthodes

CONSULTING BSTORM

2. Les 3 grandes méthodes

L’objet le plus répandu dans sklearn est l’**estimateur**

C’est un objet que l’on entraîne avec la méthode **fit** sur des données avec ses features et sa target afin de pouvoir prédire la target sur des données futures

Ensuite, on retrouve des ***transformers*** que l’on entraîne(**fit**) afin de modifier nos données

2. Les 3 grandes méthodes

Index	poids	prix
0	1	20
1	4	80
2	10	200

Prenons un cas simple:
Le prix d'un objet selon son poids
Comment prédire le prix d'un objet que l'on pas encore vendu ?

```
X = data['poids'].values.reshape(-1,1)
y = data['prix'].values.reshape(-1,1)
```

X = les features, les caractéristiques
y = la target

```
estimator = LinearRegression()
estimator.fit(X, y)
```

On utilise une régression linéaire comme estimateur
Ensuite on l'entraîne

```
new_data = np.array([2,5]).reshape(-1,1)
y_pred = estimator.predict(new_data)
```

Nous avons de nouvelles données,
On peut réutiliser l'estimateur que l'on a déjà entraîné

Scikit-learn

2. Les 3 grandes méthodes

En ce qui concerne les *transformers*, on utilisera la méthode **transform** à la place de la méthode **predict**

Régulièrement on utilise la méthode ***fit_transform*** afin de gagner en clarté dans le code

Certains *transformers* possèdent une méthode **inverse_transform** qui permet de retrouver les données originales

Exception: *LinearDiscriminantAnalysis*(LDA) qui possède une méthode fit, predict **ET** transform

3. Exploratory data analysis

EDA

CONSULTING BSTORM

3. EDA

=> Il faut comprendre vos données:

- D’un point de vue business
- D’un point de vue statistique

cfr. Jupyter Notebook

4. Preprocessing

CONSULTING BSTORM

4. Preprocessing

C’est l’étape qui consiste à préparer toutes nos données

Les algorithmes de machine learning ne peuvent comprendre que des chiffres

Comment gérer les variables catégorielles, l’absence de données, *etc*. ?

Index	poids	taille	couleur	prix
0	1	100	rouge	20
1	4	nan	rouge	80
2	10	40	vert	200

Scikit-learn

4. Preprocessing

sklearn.preprocessing

preprocessing.Binarizer(*[, threshold, copy])	Binarize data (set feature values to 0 or 1) according to a threshold.
preprocessing.FunctionTransformer([func, ...])	Constructs a transformer from an arbitrary callable.
preprocessing.KBinsDiscretizer([n_bins, ...])	Bin continuous data into intervals.
preprocessing.KernelCenterer()	Center a kernel matrix.
preprocessing.LabelBinarizer(*[, neg_label, ...])	Binarize labels in a one-vs-all fashion.
preprocessing.LabelEncoder()	Encode target labels with value between 0 and n_classes-1.
preprocessing.MultiLabelBinarizer(*[, ...])	Transform between iterable of iterables and a multilabel format.
preprocessing.MaxAbsScaler(*[, copy])	Scale each feature by its maximum absolute value.
preprocessing.MinMaxScaler([feature_range, ...])	Transform features by scaling each feature to a given range.
preprocessing.Normalizer([norm, copy])	Normalize samples individually to unit norm.
preprocessing.OneHotEncoder(*[, categories, ...])	Encode categorical features as a one-hot numeric array.
preprocessing.OrdinalEncoder(*[, ...])	Encode categorical features as an integer array.
preprocessing.PolynomialFeatures([degree, ...])	Generate polynomial and interaction features.
preprocessing.PowerTransformer([method, ...])	Apply a power transform featurewise to make data more Gaussian-like.
preprocessing.QuantileTransformer(*[, ...])	Transform features using quantiles information.
preprocessing.RobustScaler(*[, ...])	Scale features using statistics that are robust to outliers.
preprocessing.StandardScaler(*[, copy, ...])	Standardize features by removing the mean and scaling to unit variance

4. Preprocessing

1. sklearn.preprocessing.OrdinalEncoder

Permet d'encoder des variables catégorielles

Index	poids	taille	couleur	prix
0	1	100	0	20
1	4	nan	0	80
2	10	40	1	200

Le problème c'est que nous induisons un ordre. En effet, maintenant rouge = 0 et est donc plus petit que vert = 1

4. Preprocessing

2. **sklearn.preprocessing.OneHotEncoder**

Index	poids	taille	couleur	prix	rouge	vert
0	1	100	rouge	20	1	0
1	4	nan	rouge	80	1	0
2	10	40	vert	200	0	1

On crée autant de colonnes qu'il y a de catégories et on indique si cette catégorie est présente ou non dans notre ligne.

4. Preprocessing

En général, on supprime une des catégories, pourquoi ?

Pour éviter le phénomène de **multicolinéarité**:

- Ce qui arrive quand **deux features expliquent** la même chose(redondance)
- Ou autrement dit quand on peut **déduire** le résultat d'une feature sur base d'une autre

pandas.get_dummies, alternative à sklearn

4. Preprocessing

3. **sklearn.preprocessing.LabelEncoder**

Même principe de fonctionnement que l'*OrdinalEncoder* mais est destiné à modifier **uniquement** la target

Dans la pratique, sklearn gère en interne le *LabelEncoding* mais il utile si vous faites du subclassing

4. Preprocessing

4. **sklearn.preprocessing.StandardScaler**

Permet de modifier l'échelle d'une **distribution normale** afin d'obtenir une moyenne de 0 (centrer) et un écart type de 1:

Xs = (X-u)/s

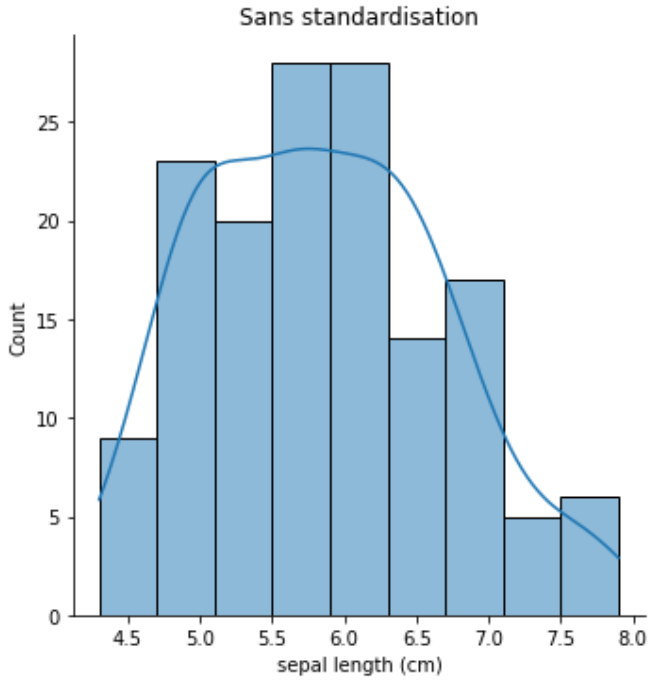
- X = la valeur d'un sample
- u = moyenne
- s = écart type

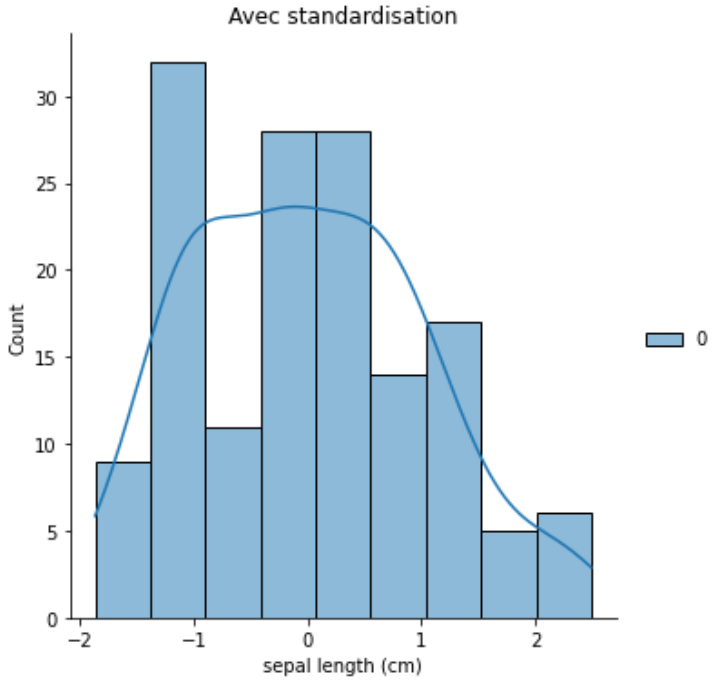
4. Preprocessing

Exemple:

Index	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5	3.6	1.4	0.2	0
5	5.4	3.9	1.7	0.4	0
6	4.6	3.4	1.4	0.3	0
7	5	3.4	1.5	0.2	0

4. Preprocessing





Scikit-learn

4. Preprocessing

Si la distribution n'est pas parfaitement normale, l'écart type ne vaudra pas 1

Pourquoi standardiser ?

=> permet d'avoir des features avec la même magnitude

=> Beaucoup d'algorithmes sont construits sur des **hypothèses**:

- Facilite les calculs
- Augmente la robustesse

Dans la pratique, ces hypothèses sont presque **toujours** violées mais cela ne signifie pas pour autant que les résultats seront mauvais. Tout dépend à quel point les hypothèses sont violées

4. Preprocessing

sklearn.preprocessing.RobustScaler

Même procédé que pour le StandardScaler mais on utilise d'autres statistiques:

- la médiane
- L'écart interquartile (Q3-Q1)

Contrairement à la moyenne et à l'écart type, ces deux statistiques ne sont pas sensibles aux **outliers**

4. Preprocessing

sklearn.preprocessing.MinMaxScaler

Alternative au StandardScaler permettant d’avoir toutes les données comprises entre un minimum et un maximum définis. En général entre 0 et 1.

4. Preprocessing

Comment gérer l’absence de données ?

Index	poids	taille	couleur
0	1	100	rouge
1	4	nan	rouge
2	10	40	nan

En les retirant: **pandas.DataFrame.dropna**

=> perte de données, dans notre exemple on perd ⅔ de nos données

4. Preprocessing

En les remplaçant **pandas.DataFrame.fillna**

Par quoi ?

Par la moyenne, la médiane, le mode, moyenne conditionnelle, une valeur arbitraire, *etc.*

```
data = data.fillna(data.mean())
data['couleur'] = data['couleur'].fillna(data['couleur'].mode().iloc[0])
```

Index	poids	taille	couleur
0	1	100	rouge
1	4	70	rouge
2	10	40	rouge

Scikit-learn

4. Preprocessing

impute.SimpleImputer(*[, missing_values, ...])	Imputation transformer for completing missing values.
impute.IterativeImputer([estimator, ...])	Multivariate imputer that estimates each feature from all the others.
impute.MissingIndicator(*[, missing_values, ...])	Binary indicators for missing values.
impute.KNNImputer(*[, missing_values, ...])	Imputation for completing missing values using k-Nearest Neighbors.

sklearn.impute.SimpleImputer:

- Même principe que **pandas.DataFrame.fillna**

sklearn.impute.KNNImputer:

- Remplacement par la moyenne de ses plus proches voisins

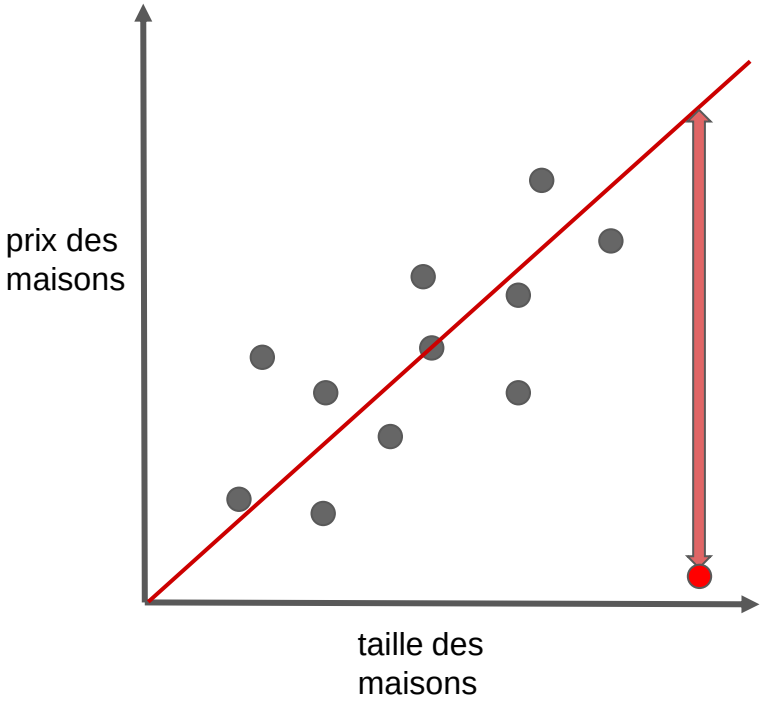
4. Preprocessing

Pourquoi et comment se débarrasser des outliers ?

=> ils ont tendance à fausser l’apprentissage

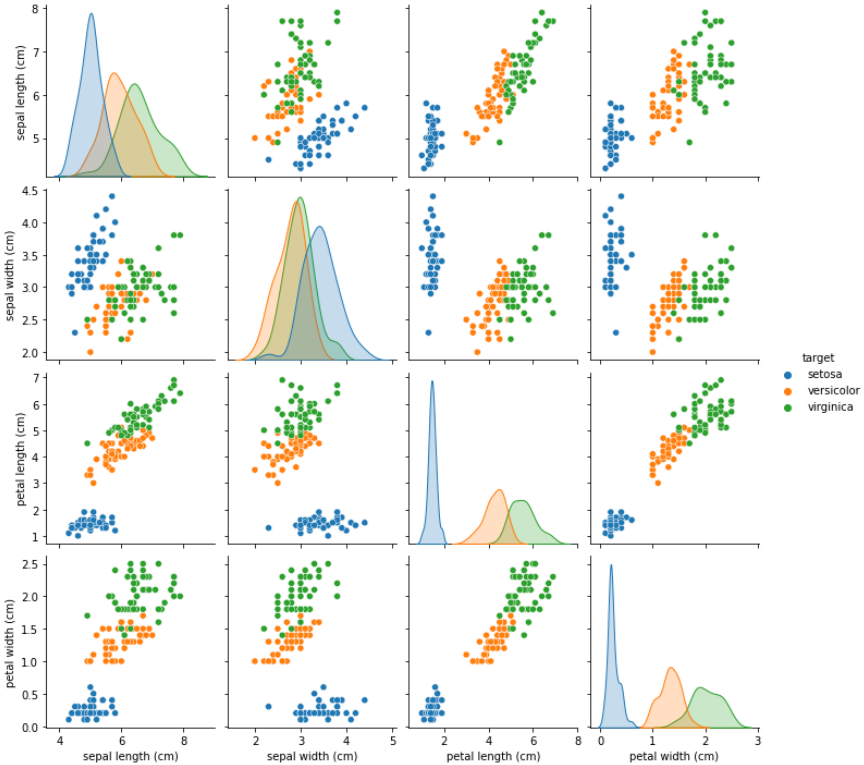
Si les outliers sont peu nombreux il est envisageable de les retirer manuellement

4. Preprocessing



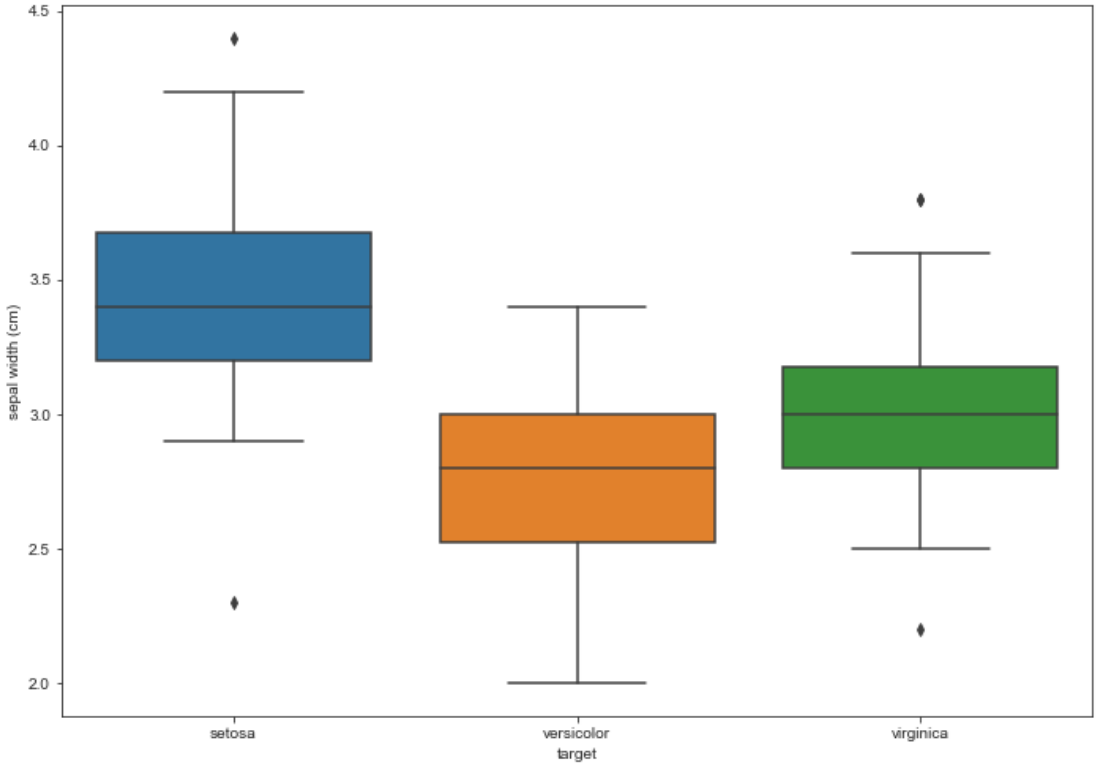
Scikit-learn

4. Preprocessing



Scikit-learn

4. Preprocessing



Scikit-learn

4. Preprocessing

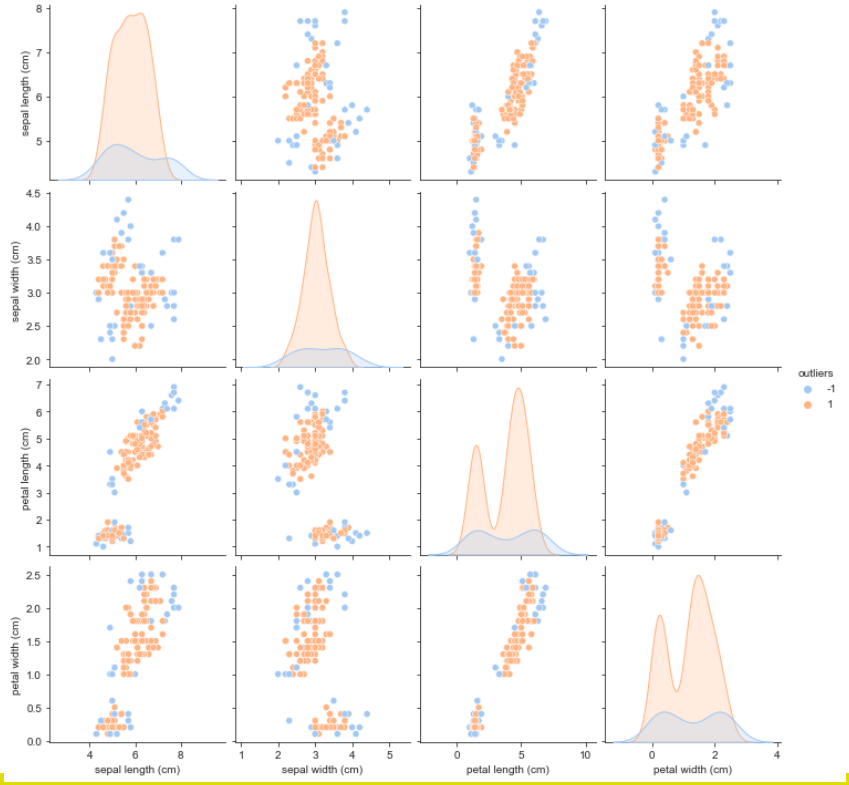
sklearn.ensemble.IsolationForest

```
isolator = IsolationForest()
data['outliers'] = isolator.fit_predict(data.drop(columns='target'))
```

renvoie -1 pour les outliers

103	6.3	2.9	5.6	1.8	virginica	1
104	6.5	3	5.8	2.2	virginica	1
105	7.6	3	6.6	2.1	virginica	-1
106	4.9	2.5	4.5	1.7	virginica	-1
107	7.3	2.9	6.3	1.8	virginica	-1
108	6.7	2.5	5.8	1.8	virginica	-1
109	7.2	3.6	6.1	2.5	virginica	-1
110	6.5	3.2	5.1	2	virginica	1

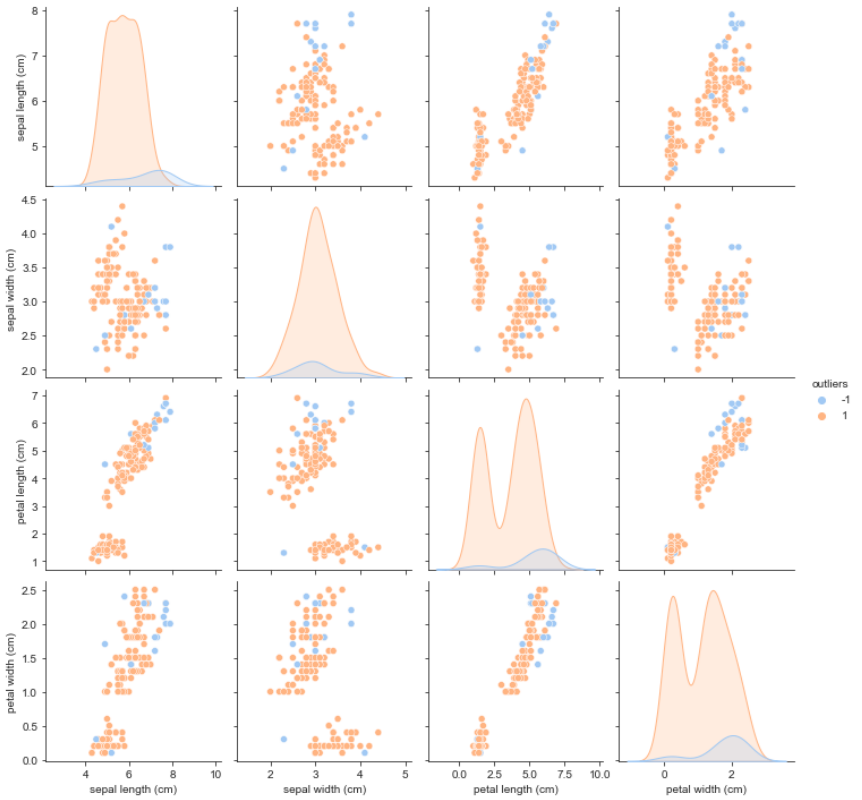
4. Preprocessing



4. Preprocessing

sklearn.covariance.EllipticEnvelope

Sert à spécifiquement détecter des outliers dans une **distribution normale**



4. Preprocessing

Comment identifier des features qui ne seraient potentiellement pas utiles ?

Prenons un exemple:

- imaginez trois voitures rouges de trois marques différentes
- Comment pourriez-vous les discriminer sur la seule base de leur couleur ?

=> Vous ne pouvez pas parce que la variance de cette feature couleur est nulle

4. Preprocessing

feature_selection.GenericUnivariateSelect([...])	Univariate feature selector with configurable strategy.
feature_selection.SelectPercentile([...])	Select features according to a percentile of the highest scores.
feature_selection.SelectKBest([score_func, k])	Select features according to the k highest scores.
feature_selection.SelectFpr([score_func, alpha])	Filter: Select the pvalues below alpha based on a FPR test.
feature_selection.SelectFdr([score_func, alpha])	Filter: Select the p-values for an estimated false discovery rate
feature_selection.SelectFromModel(estimator, *)	Meta-transformer for selecting features based on importance weights.
feature_selection.SelectFwe([score_func, alpha])	Filter: Select the p-values corresponding to Family-wise error rate
feature_selection.SequentialFeatureSelector(...)	Transformer that performs Sequential Feature Selection.
feature_selection.RFE(estimator, *[, ...])	Feature ranking with recursive feature elimination.
feature_selection.RFECV(estimator, *[, ...])	Feature ranking with recursive feature elimination and cross-validated selection of the best number of features.
feature_selection.VarianceThreshold([threshold])	Feature selector that removes all low-variance features.

sklearn.feature_selection.VarianceThreshold:

- Permet de retirer les features dont la variance ne dépasse pas un seuil fixé

sklearn.feature_selection.SelectKBest:

- Ne garde que les K meilleures features. Le K est à définir

4. Preprocessing

Comment savoir si mon preprocessing est bon ?

Proposition d'une méthode:

- Choisissez un **modèle simple**
- Entraînez-le
- Evaluez sa performance avant toute transformation, elle servira de **référence**
- A chaque fois que vous traitez vos données, analyser l'évolution de la performance

5. Sélection de modèle

CONSULTING BSTORM

5. Sélection de modèle

Définir un modèle:

=> Au **minimum** c'est un algorithme de classification, régression, etc.

=> En général c'est une succession de processus comprenant un algorithme:

- Des transformations de données
- Un entraînement de notre algorithme
- Un résultat

5. Sélection de modèle

KNN, Logistic Regression, Linear Discriminant Analysis, Decision Tree, SVM, Ridge, Lasso, Naive Bayes, etc.

Comment choisir un modèle ?

=> **Identifier** son problème(Clustering, régression, classification)

=> **Empirisme**, méthode de l'essai-erreur

=> cheat sheet

=> Avec un peu d'expérience, on sait plus ou moins vers quels modèles se tourner

5. Sélection de modèle

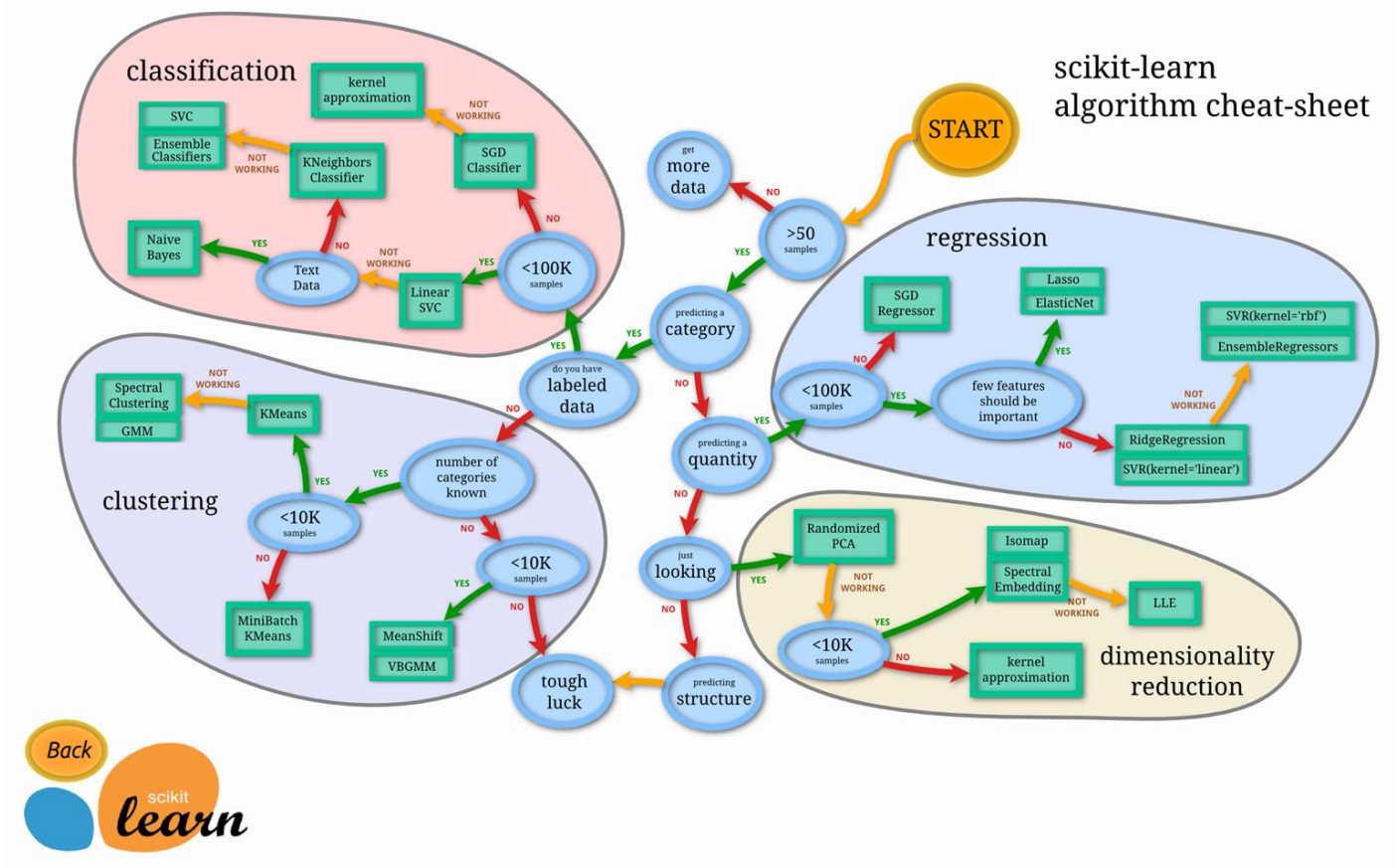
Splitter Classes	
model_selection.GroupKFold([n_splits])	K-fold iterator variant with non-overlapping groups.
model_selection.GroupShuffleSplit([...])	Shuffle-Group(s)-Out cross-validation iterator
model_selection.KFold([n_splits, shuffle, ...])	K-Folds cross-validator
model_selection.LeaveOneGroupOut()	Leave One Group Out cross-validator
model_selection.LeavePGroupsOut(n_groups)	Leave P Group(s) Out cross-validator
model_selection.LeaveOneOut()	Leave-One-Out cross-validator
model_selection.LeavePOut(p)	Leave-P-Out cross-validator
model_selection.PredefinedSplit(test_fold)	Predefined split cross-validator
model_selection.RepeatedKFold(*[, n_splits, ...])	Repeated K-Fold cross validator.
model_selection.RepeatedStratifiedKFold(*[, ...])	Repeated Stratified K-Fold cross validator.
model_selection.ShuffleSplit([n_splits, ...])	Random permutation cross-validator
model_selection.StratifiedKFold([n_splits, ...])	Stratified K-Folds cross-validator.
model_selection.StratifiedShuffleSplit([...])	Stratified ShuffleSplit cross-validator
model_selection.TimeSeriesSplit([n_splits, ...])	Time Series cross-validator

5. Sélection de modèle

Splitter Functions	
model_selection.check_cv([cv, y, classifier])	Input checker utility for building a cross-validator
model_selection.train_test_split(*arrays[, ...])	Split arrays or matrices into random train and test subsets

Hyper-parameter optimizers	
model_selection.GridSearchCV(estimator, ...)	Exhaustive search over specified parameter values for an estimator.
model_selection.HalvingGridSearchCV(...[, ...])	Search over specified parameter values with successive halving.
model_selection.ParameterGrid(param_grid)	Grid of parameters with a discrete number of values for each.
model_selection.ParameterSampler(...[, ...])	Generator on parameters sampled from given distributions.
model_selection.RandomizedSearchCV(...[, ...])	Randomized search on hyper parameters.
model_selection.HalvingRandomSearchCV(...[, ...])	Randomized search on hyper parameters.

Model validation	
model_selection.cross_validate(estimator, X)	Evaluate metric(s) by cross-validation and also record fit/score times.
model_selection.cross_val_predict(estimator, X)	Generate cross-validated estimates for each input data point
model_selection.cross_val_score(estimator, X)	Evaluate a score by cross-validation
model_selection.learning_curve(estimator, X, ...)	Learning curve.



Scikit-learn

5. Sélection de modèle

sklearn.model_selection.train_test_split

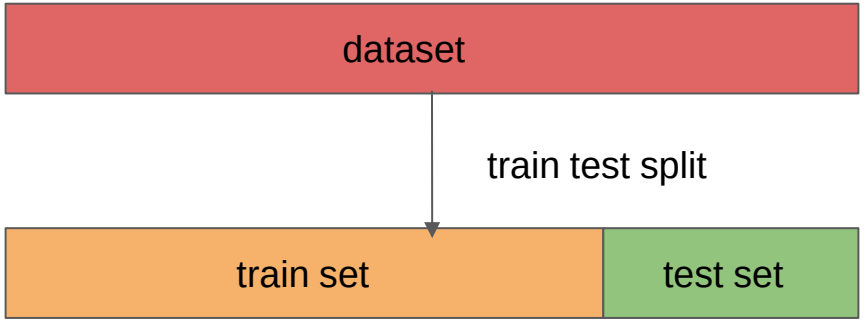
En machine learning on sépare **systématiquement** notre dataset en deux sous datasets:

- le **train set**: sert de base à l'entraînement d'un modèle
- le **test set**: sert à évaluer notre modèle **APRÈS** l'avoir correctement entraîné

Ceci permet de s'assurer que notre modèle parvient à **bien généraliser** et n'a pas appris **par coeur** le trainset

=> Mélanger(**shuffle**) son dataset avant la découpe

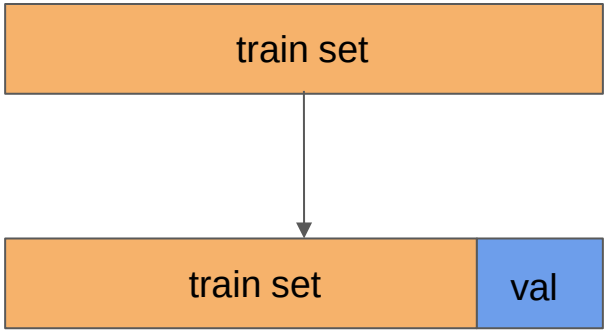
5. Sélection de modèle



5. Sélection de modèle

Comment s'assurer que notre modèle est bien entraîné avant de le valider sur le test set?

=> Le **validation set**



5. Sélection de modèle

Résumons:

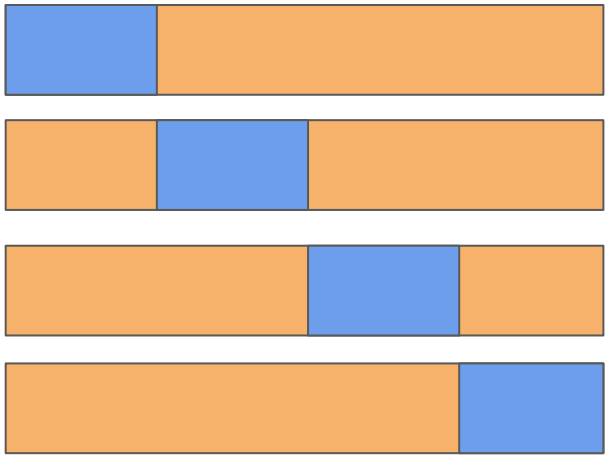
=> **Entraîner** son modèle sur le **train set** jusqu'à avoir de **bonnes performances** sur le **validation set**

=> Confirmer le modèle sur le **test set**

5. Sélection de modèle

Oui **MAIS** si mon validation set n'est **pas représentatif** ? Ce qui peut arriver quand on a peu de données.

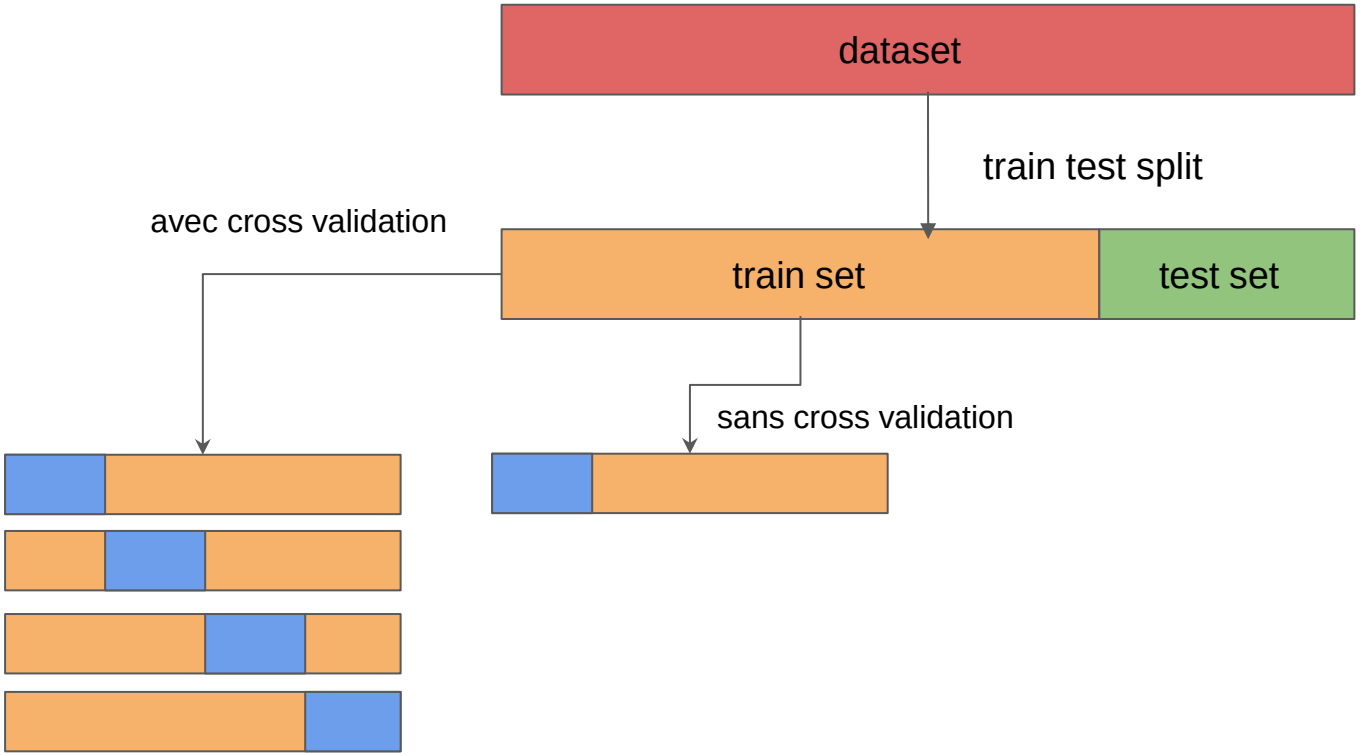
=> **Cross Validation**



4 cross validation

Peu probable de tomber 4 fois sur un validation set non représentatif

5. Sélection de modèle



Scikit-learn

5. Sélection de modèle

sklearn.model_selection.train_test_split

```
from sklearn.model_selection import train_test_split, KFold
from sklearn.datasets import load_iris
# ======================================================================
#
# ======================================================================
data = load_iris(as_frame=True)['data']
data['target'] = load_iris()['target']

X = data.drop(columns='target').values.reshape(-1,4)
y = data['target'].values.reshape(-1,1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3, random_state=1)
```

On sépare X et y

On split entre train et test set

5. Sélection de modèle

sklearn.model_selection.KFold

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3, random_state=1)

kfold = KFold(n_splits=5, shuffle=True, random_state=1)
for train_index, val_index in kfold.split(X_train, y_train):
    X_train_cross, y_train_cross = X_train[train_index], y_train[train_index]
    X_val, y_val = X_train[val_index], y_train[val_index]
```

On crée 5 splits

donc on aura 5 cross validation

train_index	Array of int32	(84,)	[0 2 3 ... 102 103 104]
val_index	Array of int32	(21,)	[1 5 6 ... 76 79 84]

kfold.split(X_train, y_train) renvoie les index

sklearn.model_selection.StratifiedKFold

Permet de prendre en compte la fréquence de vos classes

Si un trainset a 2 classes comme target (chien:70%, chat:30%), le validation set gardera ces fréquences

5. Sélection de modèle

sklearn.model_selection.GridSearchCV

sklearn.model_selection.RandomizedSearchCV

Comment fixer les **hyper paramètres(HP)** d'un modèle ?

GridSearchCV vous permet de tester un modèle avec plusieurs valeurs pour les HP

RandomizedSearchCV fonctionne de la même manière mais ne teste pas toutes les possibilités

5. Sélection de modèle

```
model = RandomForestClassifier()
params = {'n_estimators':[20,50,100,150],
          'max_depth':[5,30,50,100]}

grid = GridSearchCV(model, param_grid=params, cv=5)
grid.fit(X_train, y_train)

best_params = grid.best_params_
best_model = grid.best_estimator_
```

best_params:

Key	Type	Size	
max_depth	int	1	5
n_estimators	int	1	20

On définit un modèle à essayer

On crée un dictionnaire avec comme clefs le nom des HP et comme valeurs toutes possibilités que l'on veut tester

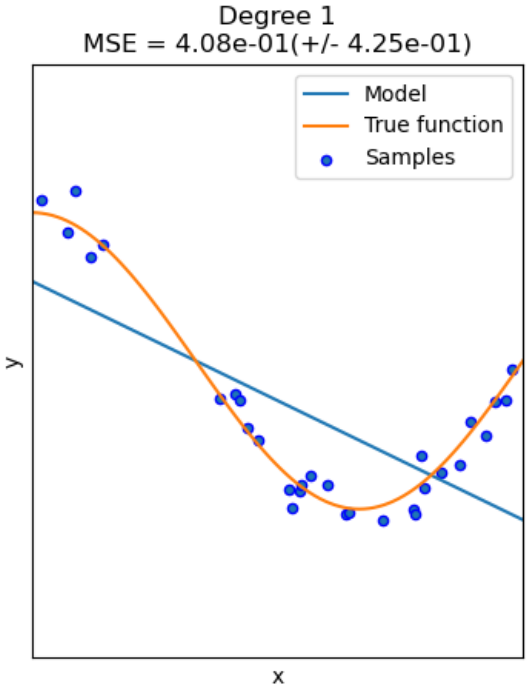
Après entraînement, on peut récupérer les meilleurs HP ainsi que le meilleur modèle

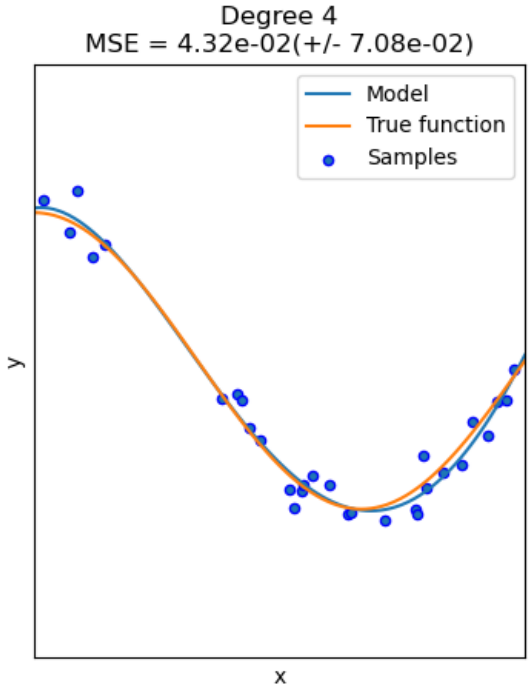
5. Sélection de modèle

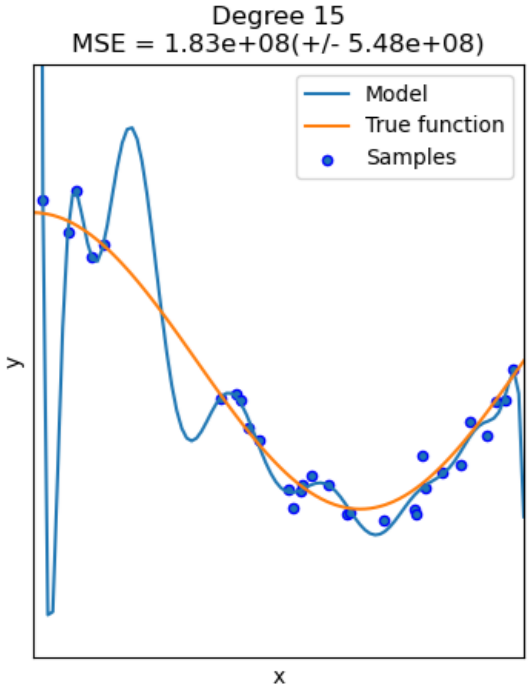
Il existe deux cas de figure où notre modèle ne généralise pas bien:

- Overfitting: modèle avec de très **bonnes** performances sur le **train set** mais de **mauvaises** sur le **val set**
- Underfitting: mauvaise performance sur le **train set et val set**

5. Sélection de modèle







Scikit-learn

5. Sélection de modèle

sklearn.model_selection.learning_curve

Permet de répondre à deux questions:

- Suis-je en état d'over/under fitting ?
- Est-ce que plus de données améliorerait les performances ?

=> On entraîne notre modèle avec 20% du train set puis 40% jusqu'à 100% et on observe les performances de notre modèle

5. Sélection de modèle

```
best_params = grid.best_params_
best_model = grid.best_estimator_

n, train_score, val_score = learning_curve(best_model, X_train, y_train, cv=5)

plt.plot(n, train_score.mean(axis=1), label='train score')
plt.plot(n, val_score.mean(axis=1), label='val score')
plt.title('Learning curve')
plt.xlabel('n')
plt.ylabel('accuracy')
plt.legend()
plt.show()
```

n	Array of int32	(5,)	[8 27 46 65 84]

Il suffit de passer votre modèle ainsi que X_train et y_train

n correspond au nombre d'observations utilisées

Pourquoi faire la moyenne sur le train et val score ?

5. Sélection de modèle

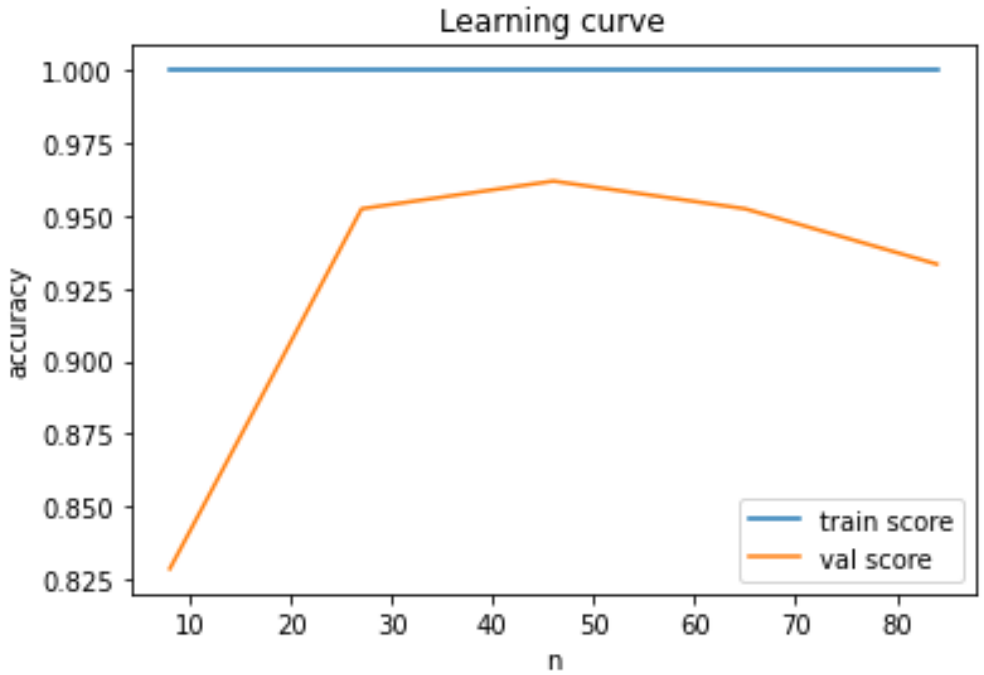
val_score:

	0	1	2	3	4
0	0.714286	0.857143	0.809524	0.857143	0.904762
1	1	0.952381	0.904762	0.952381	0.952381
2	1	1	1	0.952381	0.857143
3	1	1	1	0.904762	0.857143
4	1	0.904762	1	0.904762	0.857143

Les colonnes représentent les cross validations

les lignes représentent **n**

5. Sélection de modèle



On peut répondre à nos question de départ

=> Je n'ai pas besoin de plus que 50 observations pour l'entraînement

=> à n=50, le modèle est bon

avant ou après, **léger** overfitting

6. Les métriques

CONSULTING BSTORM

6. Les métriques

Jusqu'ici nous parlions de "performance" d'un modèle mais de quoi s'agit-il précisément ?

Ca dépend:

- du problème(*i.e.* classification, régression, *etc*.)
- de la métrique choisie
- du type de métrique(score vs loss)
  - Maximiser les scores
  - minimiser les pertes

Le choix de la métrique est **crucial** et se fait avant même le choix du modèle

6. Les métriques

pour la classification

CONSULTING BSTORM

6. Les métriques

Classification metrics

See the Classification metrics section of the user guide for further details.

metrics.accuracy_score(y_true, y_pred, *[, ...])	Accuracy classification score.
metrics.auc(x, y)	Compute Area Under the Curve (AUC) using the trapezoidal rule.
metrics.average_precision_score(y_true, ...)	Compute average precision (AP) from prediction scores.
metrics.balanced_accuracy_score(y_true, ...)	Compute the balanced accuracy.
metrics.brier_score_loss(y_true, y_prob, *)	Compute the Brier score loss.
metrics.classification_report(y_true, y_pred, *)	Build a text report showing the main classification metrics.
metrics.cohen_kappa_score(y1, y2, *[, ...])	Cohen's kappa: a statistic that measures inter-annotator agreement.
metrics.confusion_matrix(y_true, y_pred, *)	Compute confusion matrix to evaluate the accuracy of a classification.
metrics.dcg_score(y_true, y_score, *[, k, ...])	Compute Discounted Cumulative Gain.
metrics.det_curve(y_true, y_score[, ...])	Compute error rates for different probability thresholds.
metrics.f1_score(y_true, y_pred, *[, ...])	Compute the F1 score, also known as balanced F-score or F-measure.
metrics.fbeta_score(y_true, y_pred, *, beta)	Compute the F-beta score.
metrics.hamming_loss(y_true, y_pred, *[, ...])	Compute the average Hamming loss.
metrics.hinge_loss(y_true, pred_decision, *)	Average hinge loss (non-regularized).
metrics.jaccard_score(y_true, y_pred, *[, ...])	Jaccard similarity coefficient score.
metrics.log_loss(y_true, y_pred, *[, eps, ...])	Log loss, aka logistic loss or cross-entropy loss.
metrics.matthews_corrcoef(y_true, y_pred, *)	Compute the Matthews correlation coefficient (MCC).
metrics.multilabel_confusion_matrix(y_true, ...)	Compute a confusion matrix for each class or sample.
metrics.ndcg_score(y_true, y_score, *[, k, ...])	Compute Normalized Discounted Cumulative Gain.
metrics.precision_recall_curve(y_true, ...)	Compute precision-recall pairs for different probability thresholds.
metrics.precision_recall_fscore_support(...)	Compute precision, recall, F-measure and support for each class.
metrics.precision_score(y_true, y_pred, *[, ...])	Compute the precision.
metrics.recall_score(y_true, y_pred, *[, ...])	Compute the recall.
metrics.roc_auc_score(y_true, y_score, *[, ...])	Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scores.
metrics.roc_curve(y_true, y_score, *[, ...])	Compute Receiver operating characteristic (ROC).
metrics.top_k_accuracy_score(y_true, y_score, *)	Top-k Accuracy classification score.
metrics.zero_one_loss(y_true, y_pred, *[, ...])	Zero-one classification loss.

Scikit-learn

6. Les métriques

sklearn.metrics.accuracy_score

Score compris entre 0 et 1

Nombre total de bonnes prédictions/Nombre d'observations

$$\texttt{accuracy}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} 1(\hat{y}_i = y_i)$$

Métrique très **macro**

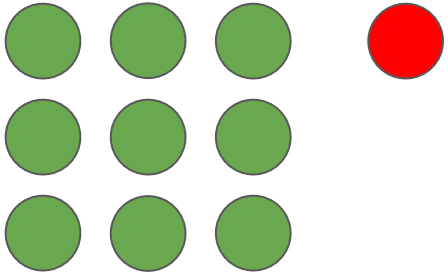
6. Les métriques

L’accuracy peut être trompeuse

Imaginons que vous deviez élaborer un test de dépistage du cancer

Sur un échantillon de 100 individus vous savez que 10 sont atteints du cancer

réalité



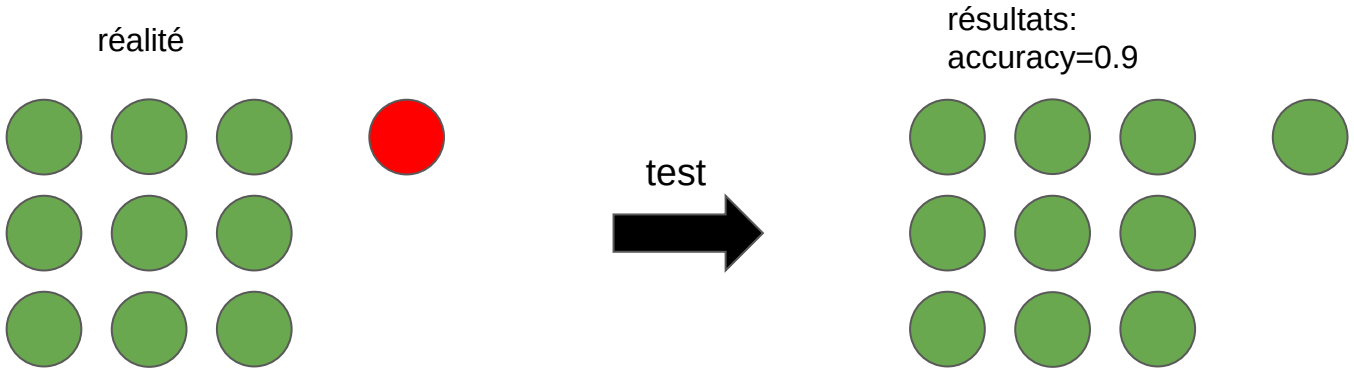
Quel est le potentiel problème avec l’accuracy ?

6. Les métriques

L’accuracy peut être trompeuse

Imaginons que vous deviez élaborer un test de dépistage du cancer

Sur un échantillon de 100 individus vous savez que 10 sont atteints du cancer

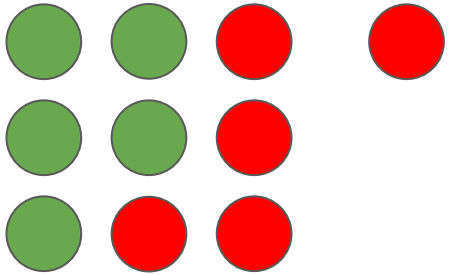


6. Les métriques

Est-ce un bon test ?

Imaginons le test suivant:

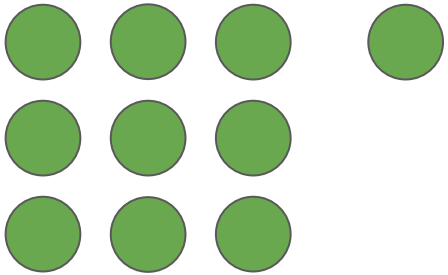
résultats:
accuracy=0.6



Est-ce un meilleur test ?

6. Les métriques

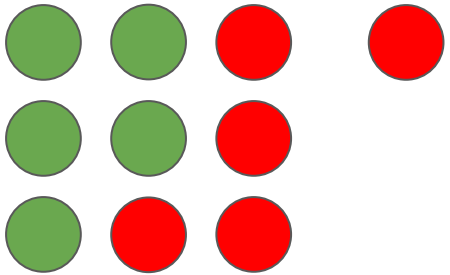
résultats 1:
accuracy=0.9



- Le test passe à côté de tous les malades

- Pour évacuer tous les doutes vous devez effectuer **100 radiographies**

résultats 2:
accuracy=0.6



- Le test détecte bien tous les malades + quelques non malades

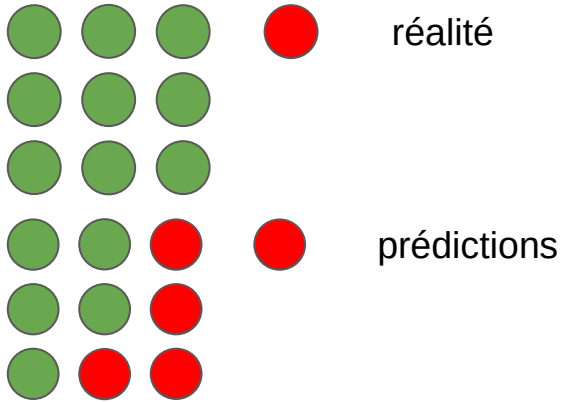
- Pour évacuer tous les doutes vous n'avez plus qu'à effectuer **50 radiographies**

6. Les métriques

Matrice de confusion

Permet de comparer les prédictions à la réalité

Pred \ Actual	sain	cancer
sain		
cancer		

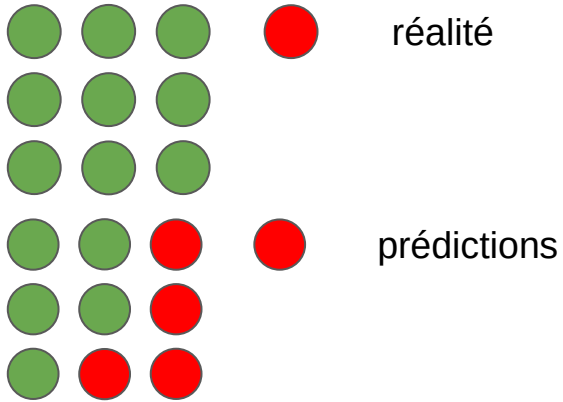


6. Les métriques

Matrice de confusion

Permet de comparer les prédictions à la réalité

Pred \ Actual	sain	cancer
sain	5	0
cancer	4	1



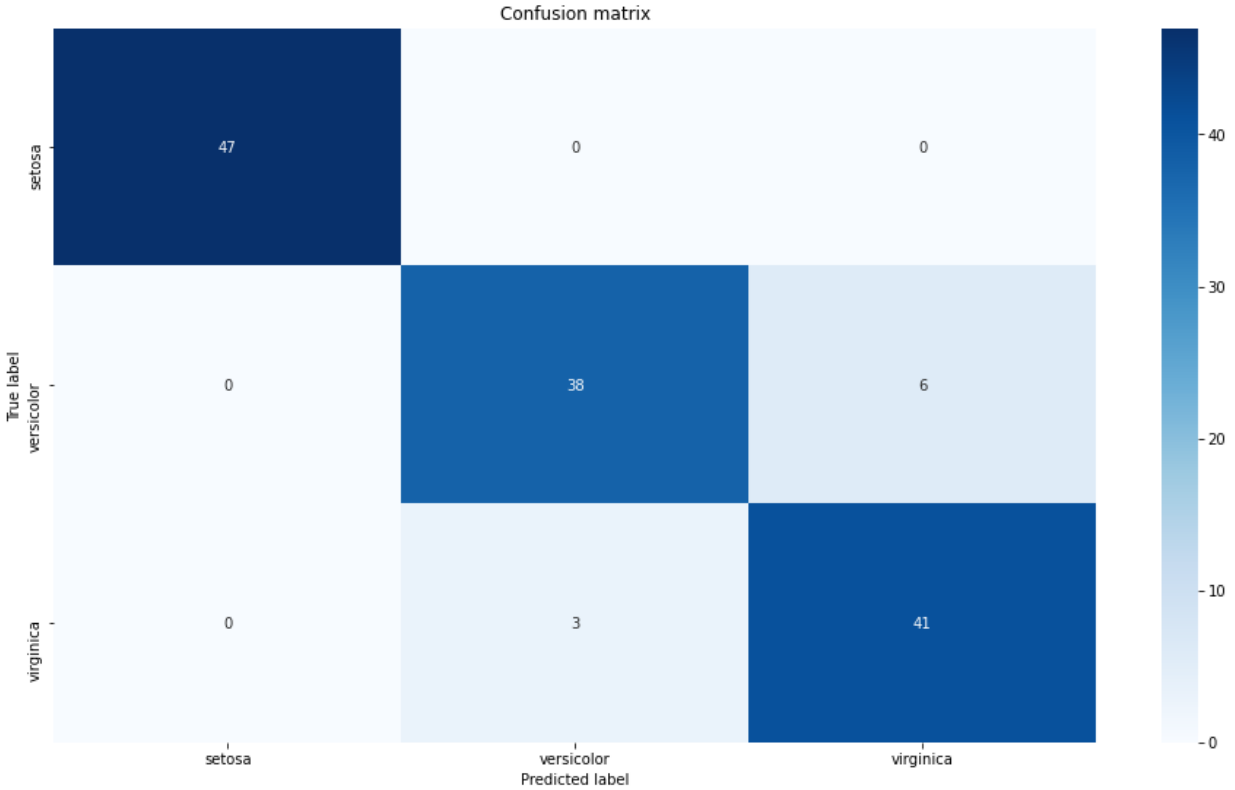
6. Les métriques

sklearn.metrics.confusion_matrix

```
model = LinearDiscriminantAnalysis()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cmap='Blues', xticklabels=labels, yticklabels=labels)
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.title('Confusion matrix')
plt.show()
```

Petite précision dans sklearn, les colonnes représentent les prédictions et non pas les valeurs réelles

6. Les métriques



Scikit-learn

6. Les métriques

sklearn.metrics.precision_score

La précision (à ne pas confondre avec accuracy) est une mesure d'**exactitude**. Il s'agit d'une mesure plus micro, un precision_score par classe

Autrement dit quand mon test m'affirme telle classe, quelle est la probabilité que cela soit réellement le cas

Pred \ Actual	sain	cancer
sain	5	0
cancer	4	1

precision_score classe sain = ?

precision_score classe cancer = ?

6. Les métriques

precision_score classe sain = 1

- 5/5 = 1
- Le test a prédit 5 personnes saines et ces 5 personnes le sont réellement

precision_score classe cancer = 0.2

- 1/5 = 0.2
- Le test a prédit 5 cancers mais seulement une personne a réellement un cancer

6. Les métriques

sklearn.metrics.recall_score

Le recall, le rappel, la sensibilité est une mesure d'**exhaustivité.** Probabilité de retrouver tous les individus d'une **même** classe. Autrement dit, la probabilité de ne pas passer à côté des individus d'une classe

Pred \ Actual	sain	cancer
sain	5	0
cancer	4	1

recall_score classe sain = ?

recall_score classe cancer = ?

6. Les métriques

recall_score classe sain = 0.56

- 5/9 = 0.56
- Le test a retrouvé 5 des 9 personnes réellement saines

recall_score classe cancer = 1

- 1/1 = 1
- Le test a retrouvé 1 des 1 personne réellement cancéreuse

6. Les métriques

sklearn.metrics.f1_score

Il s'agit d'une mesure qui permet de résumer la précision et le rappel. En terme plus statistique il s'agit d'une moyenne harmonique:

f1_score = 2*(precision*recall)/(precision+recall)

f1_score classe sain = 2*(1*0.56)/(1+0.56) = 0.71

f1_score classe cancer = 2*(0.2*1)/(1+0.2) = 0.33

6. Les métriques

sklearn.metrics.classification_report

Permet de résumer toutes les métriques dans un seul tableau

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	47
versicolor	0.93	0.86	0.89	44
virginica	0.87	0.93	0.90	44
accuracy			0.93	135
macro avg	0.93	0.93	0.93	135
weighted avg	0.93	0.93	0.93	135

6. Les métriques

Maintenant on comprend pourquoi le test 2 était meilleur que le test 1

Quand on teste une personne pour une maladie grave ou très contagieuse, on ne veut surtout pas passer à côté(recall = 1 pour la classe cancer)

C’est prendre le risque de ne pas traiter à temps un cancer ou de laisser dans la nature une personne contagieuse

6. Les métriques

pour la régression

CONSULTING BSTORM

6. Les métriques

Regression metrics

See the Regression metrics section of the user guide for further details.

`metrics.explained_variance_score`(y_true, ...)	Explained variance regression score function.
`metrics.max_error`(y_true, y_pred)	max_error metric calculates the maximum residual error.
`metrics.mean_absolute_error`(y_true, y_pred, *)	Mean absolute error regression loss.
`metrics.mean_squared_error`(y_true, y_pred, *)	Mean squared error regression loss.
`metrics.mean_squared_log_error`(y_true, y_pred, *)	Mean squared logarithmic error regression loss.
`metrics.median_absolute_error`(y_true, y_pred, *)	Median absolute error regression loss.
`metrics.mean_absolute_percentage_error`(...)	Mean absolute percentage error regression loss.
`metrics.r2_score`(y_true, y_pred, *[, ...])	R^2 (coefficient of determination) regression score function.
`metrics.mean_poisson_deviance`(y_true, y_pred, *)	Mean Poisson deviance regression loss.
`metrics.mean_gamma_deviance`(y_true, y_pred, *)	Mean Gamma deviance regression loss.
`metrics.mean_tweedie_deviance`(y_true, y_pred, *)	Mean Tweedie deviance regression loss.

Scikit-learn

6. Les métriques

sklearn.metrics.mean_squared_error (MSE)

En Français: erreur quadratique moyenne

=> Différence entre la réalité et la prédiction au carré divisé par le nombre d'observations

$$\mathrm{MSE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2.$$

6. Les métriques

y_test = [1,5,6,10,11]

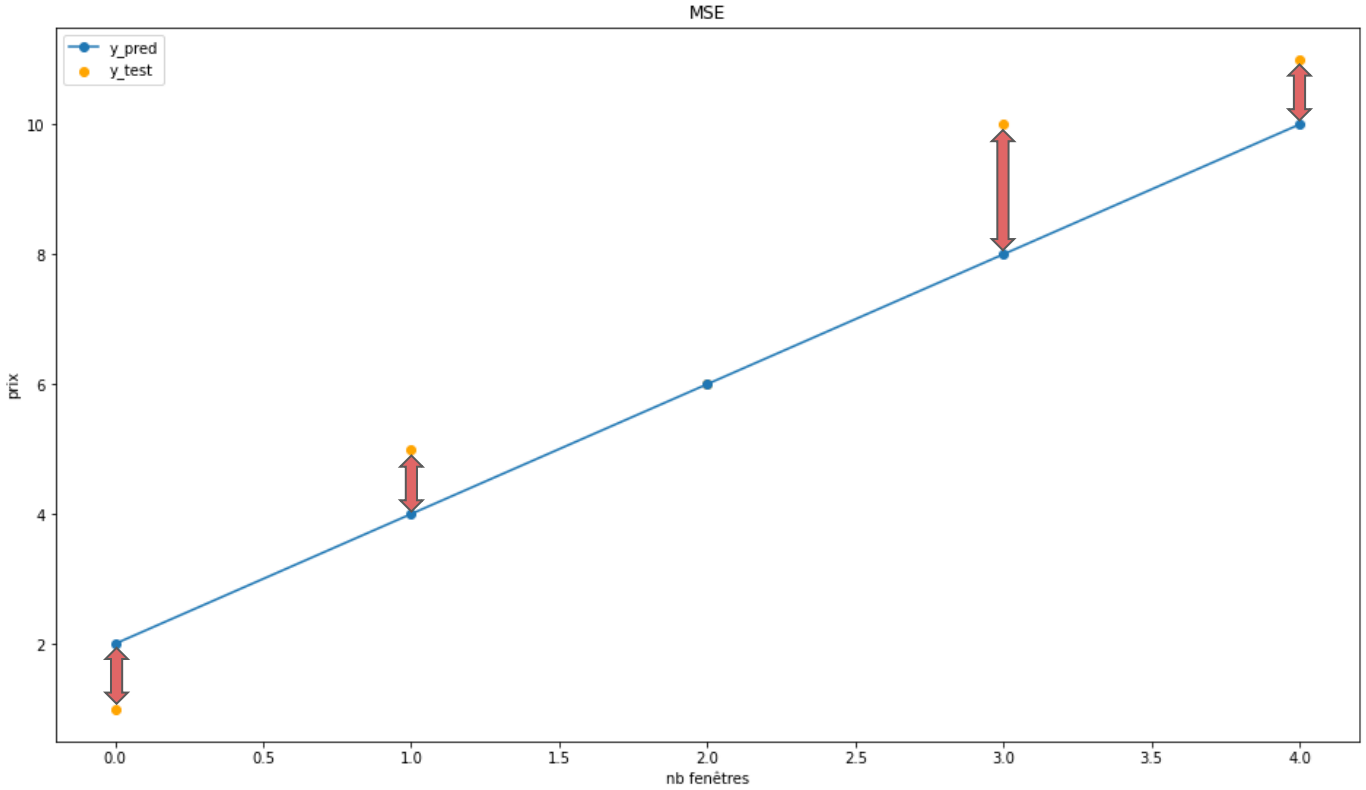
y_pred = [2,4,6,8,10]

MSE = $((1-2)^2+(5-4)^2+(6-6)^2+(10-8)^2+(11-10)^2)/5 = 1.4$

il suffit de prendre la racine carrée pour retrouver l'échelle originale

Root Mean Squared Error (RMSE)= 1.18

6. Les métriques



6. Les métriques

sklearn.metrics.mean_absolute_error

$$\text{MAE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} |y_i - \hat{y}_i|$$

y_test = [1,5,6,10,11]

y_pred = [2,4,6,8,10]

MAE = (|1-2|+|5-4|+|6-6|+|10-8|+|11-10|)/5 = 1

6. Les métriques

MAE ou **MSE** ?

Est-ce grave de faire des grosses erreurs ?

- Si oui **MSE**: comme on met au carré, les grosses erreurs auront plus de poids. Attention que cela rend la **MSE** plus sensible aux outliers
- Si Non **MAE**: l'importance d'une erreur est linéaire à son amplitude. La **MAE** est moins sensible aux outliers

E.g.: imaginons que vous développiez un vaccin et que vous cherchez à déterminer la quantité d'un composant à ajouter pour améliorer l'efficacité

6. Les métriques

	erreur 1	erreur 2	erreur 3	**MAE**	**MSE**
modèle 1	15	0	0	5	75
modèle 2	10	7	2	6	51
modèle 3	20	10	0	10	166

- Si vous avez de la marge, que vous pouvez mettre beaucoup de ce composant sans compromettre l'efficacité du vaccin alors le modèle 1 est mieux
- Par contre si cette marge est petite alors le modèle 2 est meilleur

Scikit-learn

6. Les métriques

sklearn.metrics.median_absolute_error

$$\text{MedAE}(y, \hat{y}) = \text{median}(| y_1 - \hat{y}_1 |, \ldots, | y_n - \hat{y}_n |).$$

Au lieu de prendre la moyenne, on utilise la **médiane**. Mesure très robuste aux outliers

Imaginons que notre modèle fasse 1 très grosse erreur

y_test = [1,5,6,10,11,**2**]

y_pred = [2,4,6,8,10,**1000**]

6. Les métriques

MAE	MSE	Median AE
167	166001	1

Attention que si faire une très grosse erreur a de très grosses conséquences, n'utilisez pas la **Median AE**

Quand choisir quoi ?

- Le choix de la métrique dépend du contexte, c'est à vous de la choisir
- Vous pouvez utiliser plusieurs métriques
- Vous pouvez utiliser les métriques pour construire des intervalles de confiance

6. Les métriques

sklearn.metrics.r2_score

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

On retrouve:

- au numérateur l'erreur quadratique
- au dénominateur la variance

6. Les métriques

Au plus votre erreur quadratique est petite par rapport à la variance au plus le **R2** est proche de 1

r2_score = 1-(2/20) = 0.9

=> Mon modèle explique 90% des variations au sein de mon dataset

=> Au plus ce **R2** est élevé, au plus la différence entre vos prédictions est liée à la variance et non pas aux erreurs de votre modèle

=> Est-ce que la différence entre mes mesures est liées aux erreurs de mon modèle ou à la variance de ma distribution ?

7. Dimensionality curse

7. Dimensionality curse

Ajouter de l'information peut améliorer les performances mais cela peut aussi les diminuer

=> Problème de multicolinéarité, redondance

=> Augmente les temps de calcul

=> trop d'information tue l'information

Nous avons déjà vu dans la partie preprocessing des méthodes pour sélectionner des variables (SelectKBest, VarianceThreshold)

7. Dimensionality curse

sklearn.feature_selection.SelectKBest

Par défaut il s'agit d'une one way ANOVA(Valable que pour la classification)

- Univarié signifie ici que l'on analyse feature par feature
- Permet de déduire si les moyennes conditionnelles sont significativement différentes les unes des autres

=> Il s'agit d'un test d'**indépendance**

7. Dimensionality curse

```
X = load_iris()['data']
y = load_iris()['target']

selector = SelectKBest(k=X.shape[1])
selector.fit(X, y)
scores = selector.scores_
```

	0
0	119.265
1	49.16
2	1180.16
3	960.007

Après entraînement, vous pouvez accéder aux scores

Au plus cette valeur est élevée, au plus les moyennes conditionnelles sont différentes

On ne garde que les K meilleur scores

scipy.stats.f_oneway donnera exactement les même scores

7. Dimensionality curse

Attention aux méthodes univariées:

- 2 variables identiques auront toutes les deux le même score et peuvent donc être potentiellement toutes les deux sélectionnées
- Toutes les variables pourraient avoir de très bons scores et on exclurait de l'information importante

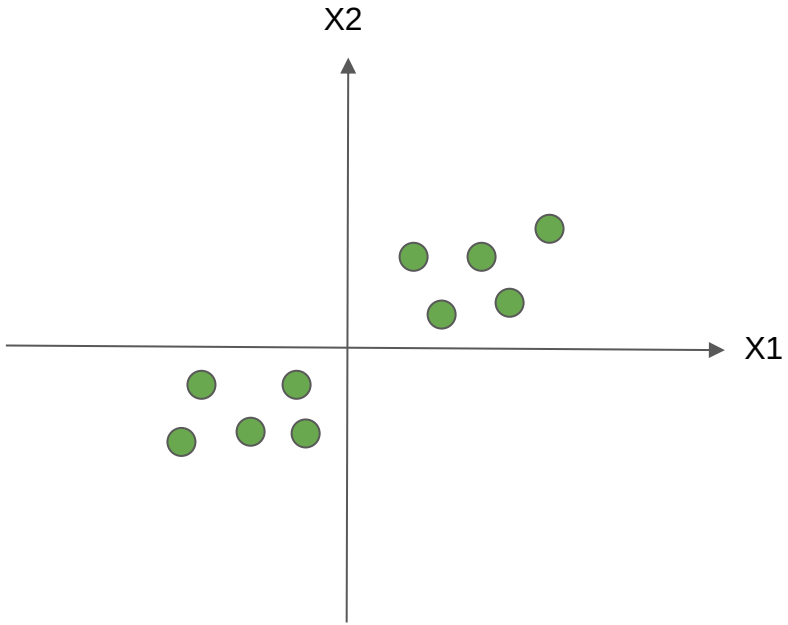
7. Dimensionality curse

sklearn.decomposition.PCA

Principal Component Analysis

La PCA permet de réduire le nombre de feature ET de réduire la redondance

Imaginons le dataset suivant avec 6 observations et 2 features



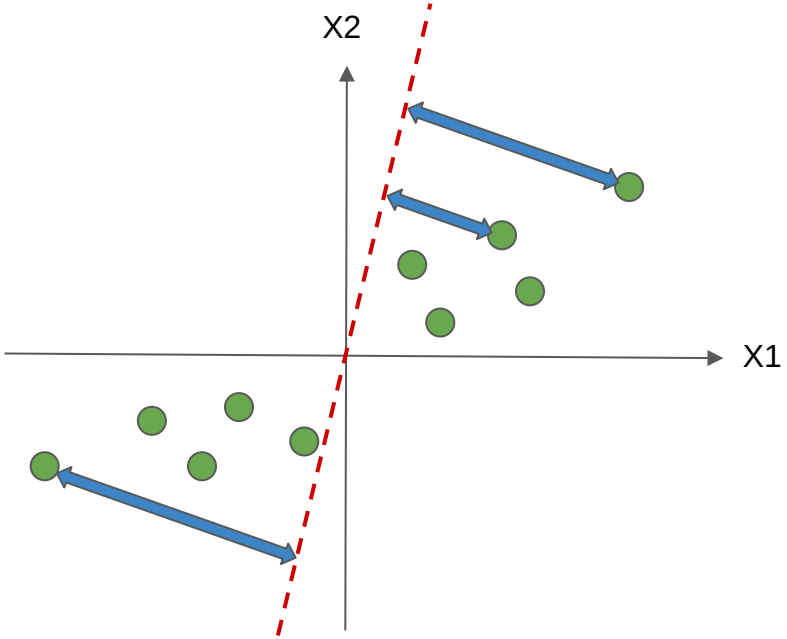
7. Dimensionality reduction

L’idée est de tracer une ligne passant par 0.

Ensuite on calcule la distance séparant chaque point de cette ligne, on met au carré (pour que les négatifs soient positifs) et on additionne tout.

Au plus cette valeur est petite au mieux c’est.

P.S.: c’est la même chose que la **MSE** sauf qu’on ne fait pas de moyenne

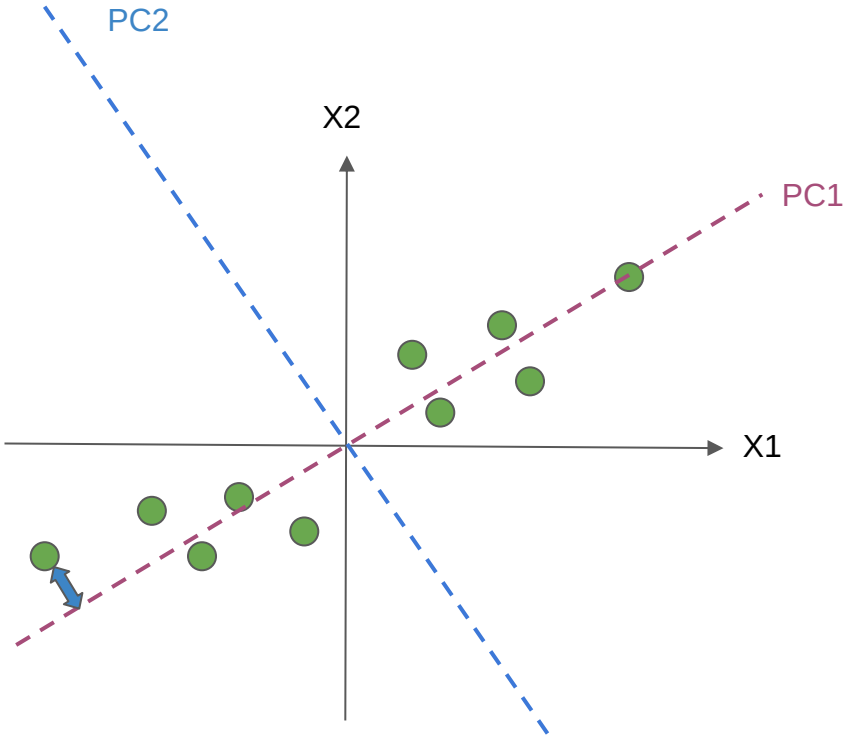


7. Dimensionality reduction

Une fois cette distance minimisée, on obtient notre première composante principale(PC1)

Ensuite la seconde composante est nécessairement perpendiculaire à la première

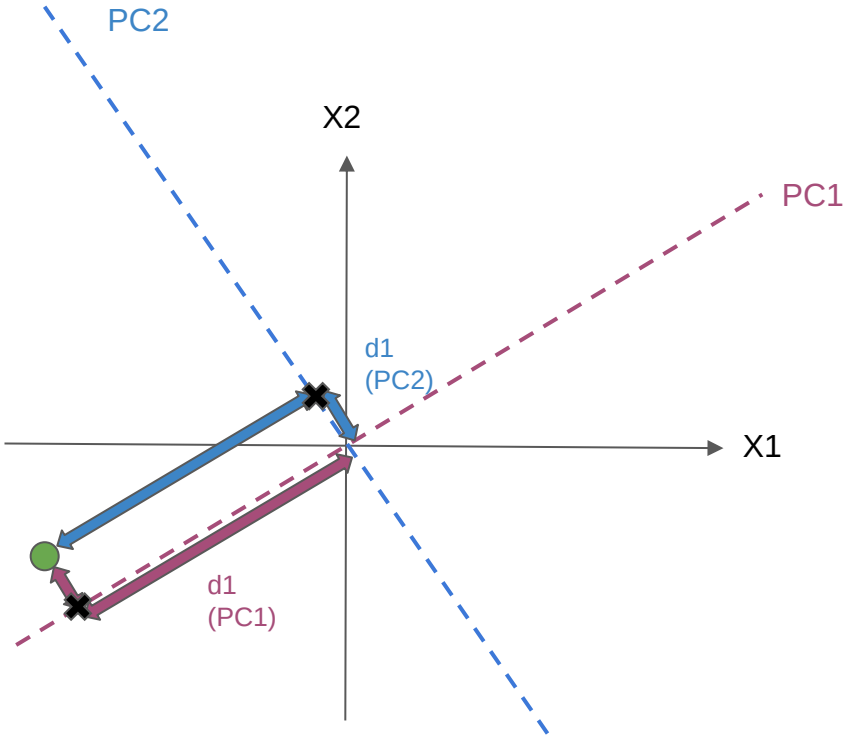
Et il y a autant de composantes qu'il y a de features



7. Dimensionality reduction

On peut projeter chaque point sur une composante et prendre la distance qui le sépare de l'origine.

Si on calcule cette distance pour chaque point, que l'on met au carré et qu'on l'on somme le tout, on obtient ce que l'on appelle l'**eigenvalue(Singular Value)** pour la PC1



eigenvalue = $d1^2+...+dn^2$

=> **Sommes des carrés (SS)** des distances par rapport à l'origine

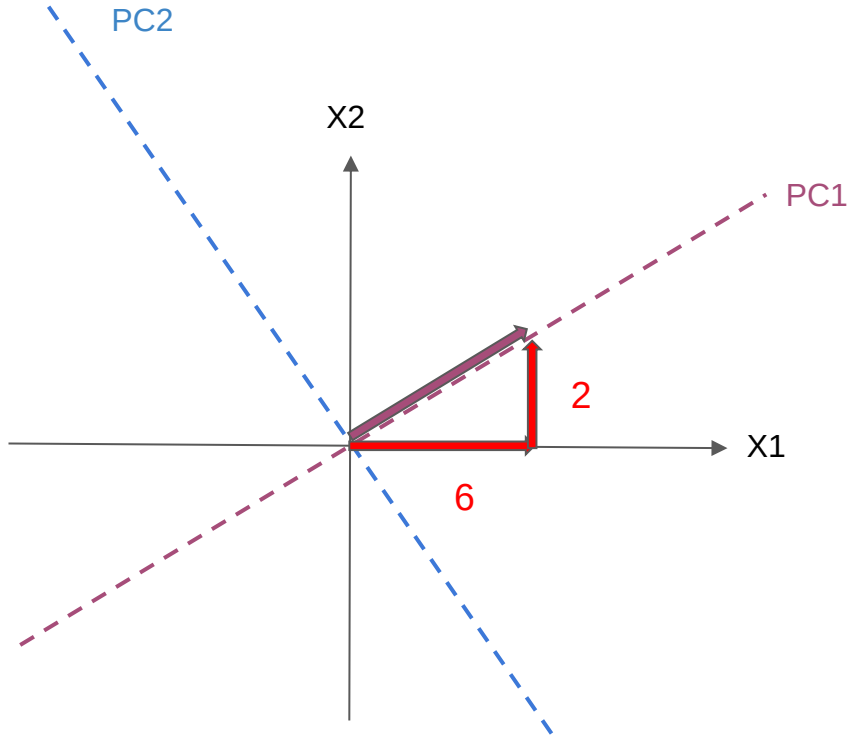
7. Dimensionality reduction

Il est possible de décomposer la PC1 en 2 vecteurs:

- Je vais 6 à droite sur X1
- Je vais 2 en hauteur sur X2

=> C'est pareil que quand vous faites un verre de grenadine, 6 unités d'eau pour 2 de concentré

Pour simplifier, on préfère avoir un vecteur PC1 d'une unité



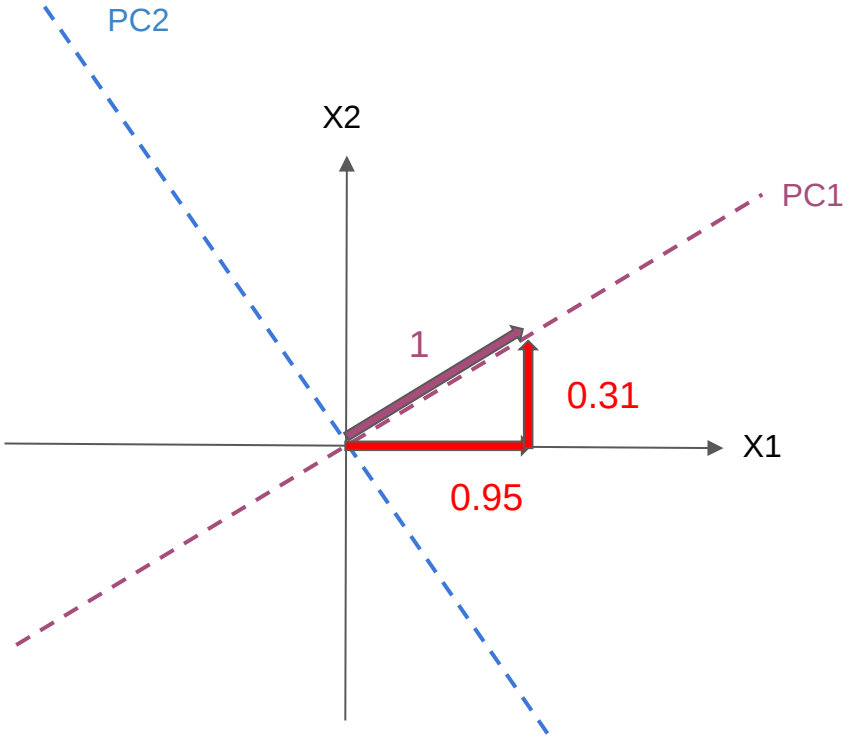
7. Dimensionality reduction

On passe les étapes mathématiques mais il s'agit du théorème de pythagore

Pour que notre vecteur = 1:

- X1 = 0.95
- X2 = 0.31

On appelle ce vecteur de 1 obtenu un **vecteur propre(eigenvector)** pour la PC1



7. Dimensionality reduction

Comment réduire notre nombre de dimensions? Avec les **eingenvalues**

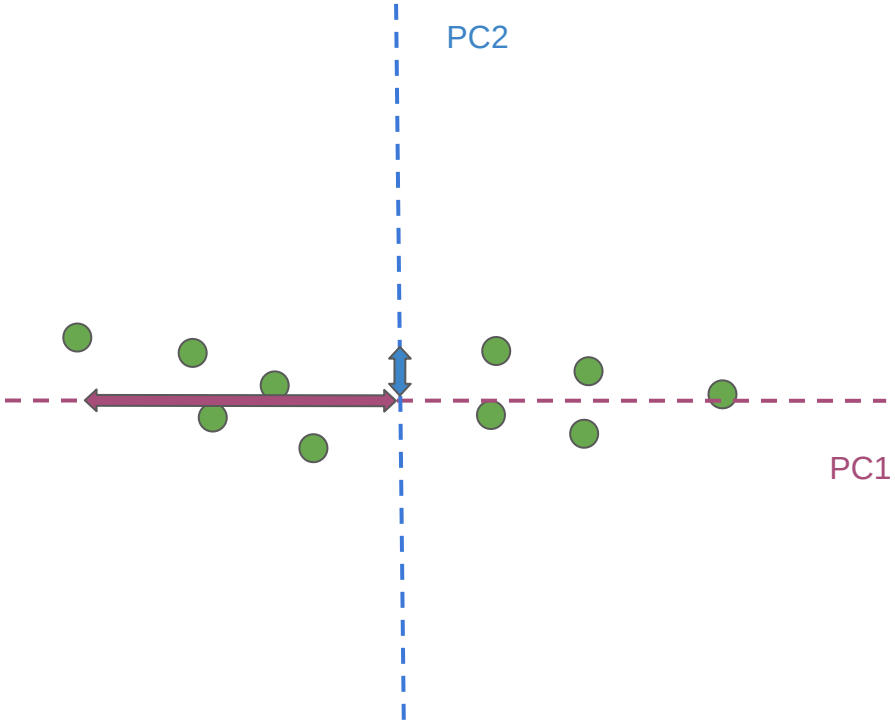
On peut transformer ces valeurs en **variance** en les divisant par **n-1**

eigenvalue(PC1)/n-1 = **Variance (PC1) = 9**

eigenvalue(PC2)/n-1 = **Variance(PC2) = 1**

Notre PC1 explique 90% de la variance et PC2 seulement 10%

=> On peut retirer notre PC2



Scikit-learn

7. Dimensionality reduction

```
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
```

X	Array of float64	(150, 4)	[[-0.90068117 1.01900435 -1.34022653 -1.3154443] [-1.14301691 -0.13 ...
X_pca	Array of float64	(150, 2)	[[-2.26470281 0.4800266] [-2.08096115 -0.67413356]

sklearn.decomposition.PCA

=> Standardiser vos données

Fonctionne comme n'importe quel transformer

n_components = le nombre de dimensions à garder

7. Dimensionality reduction

```
n_samples, n_features = X.shape
pca = PCA(n_components=n_features)
pca.fit(X)

explained_variance = pca.explained_variance_ratio_

plt.plot(range(n_features), explained_variance)
plt.title("Explained Variance")
plt.show()
```

explained_variance_ratio_:

```
0.729624
0.228508
0.0366892
0.00517871
```

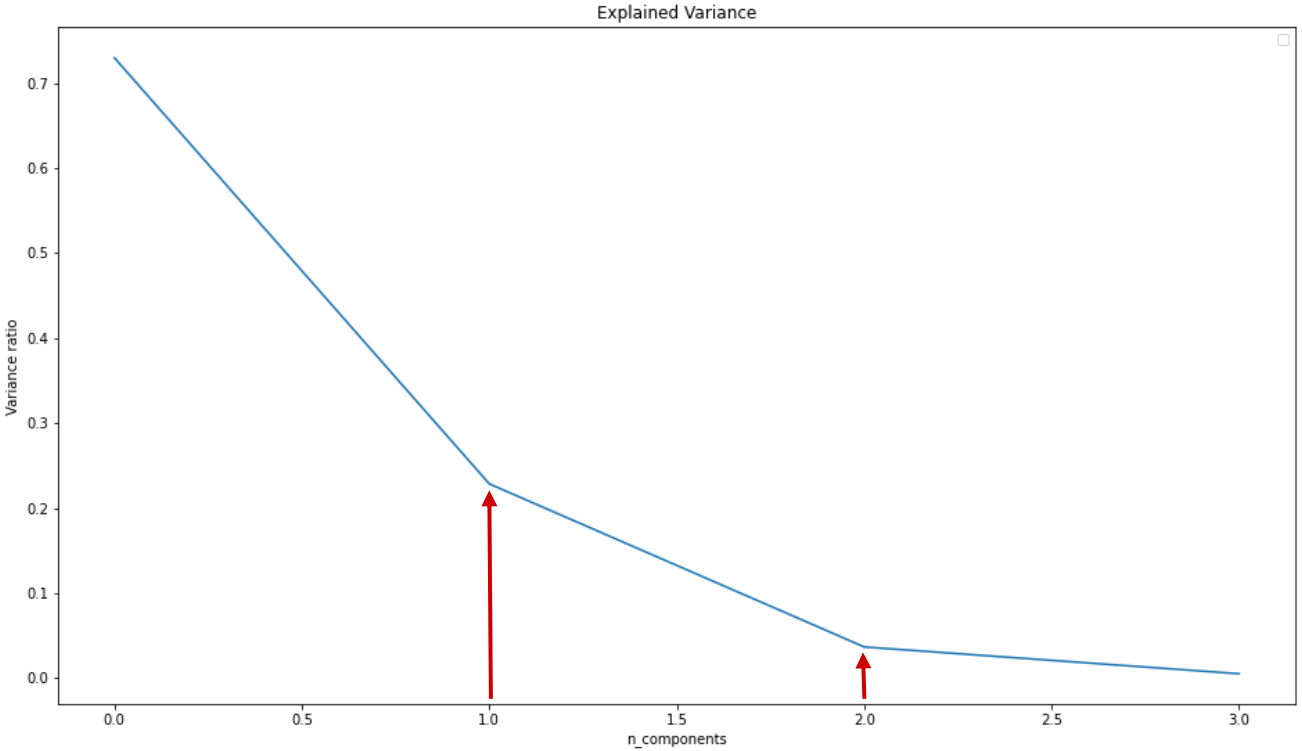
Comment choisir le nombre de composante à garder ?

=> méthode du coude

Fitter votre PCA en gardant autant de composantes que de features

Plotter l'**explained_variance_ratio_**

7. Dimensionality reduction



Scikit-learn

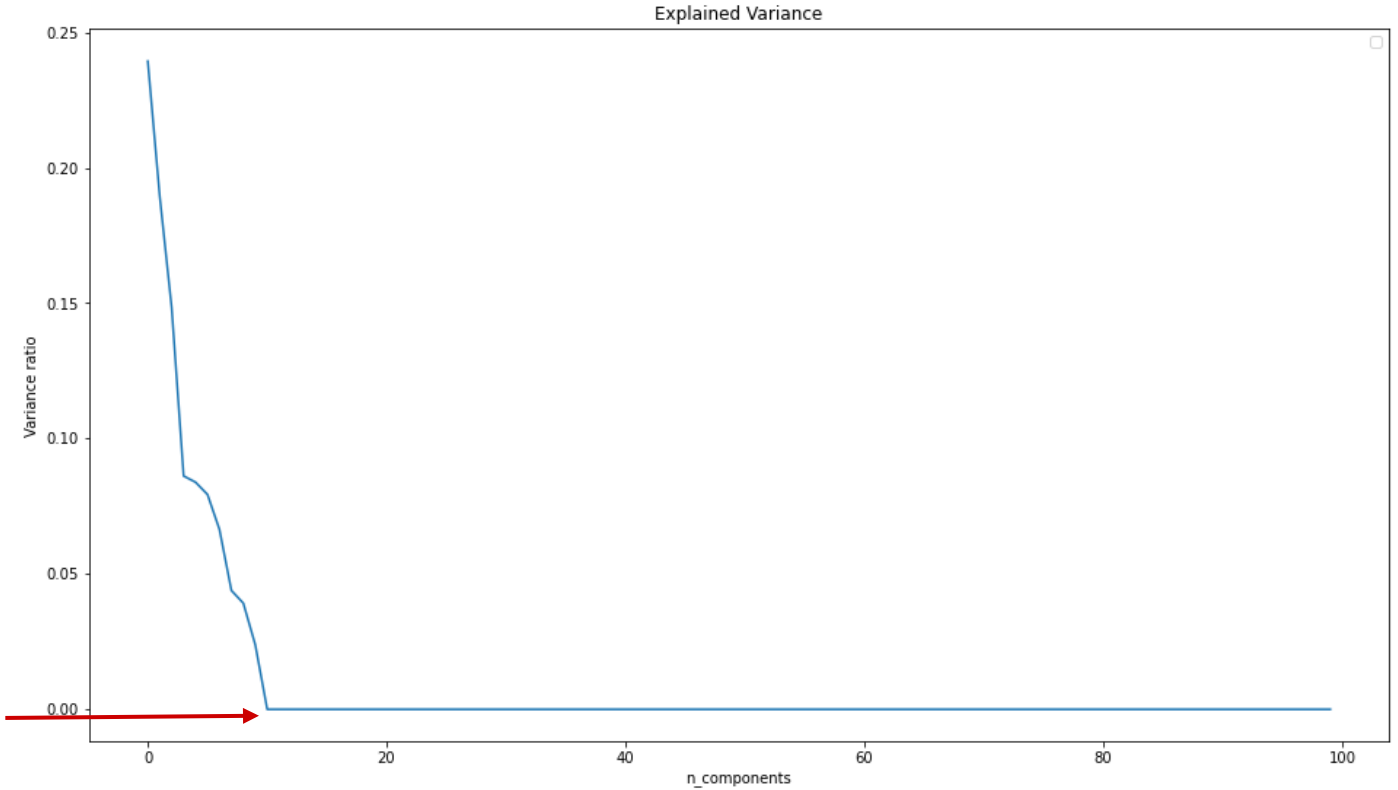
7. Dimensionality reduction

```
X, y = make_classification(n_samples=100000, n_features=100, n_informative=10, n_redundant=90)

n_samples, n_features = X.shape
pca = PCA(n_components=n_features)
pca.fit(X)

explained_variance = pca.explained_variance_ratio_

plt.figure(figsize=(16,9))
plt.plot(range(n_features), explained_variance)
plt.ylabel('Variance ratio')
plt.xlabel('n_components')
plt.title('Explained Variance')
plt.legend()
plt.show()
```

Est-ce que la PCA permet vraiment d'éliminer la redondance ?

Nous avons créé un faux dataset avec 100 features dont 90 sont redondantes

7. Dimensionality reduction



Scikit-learn

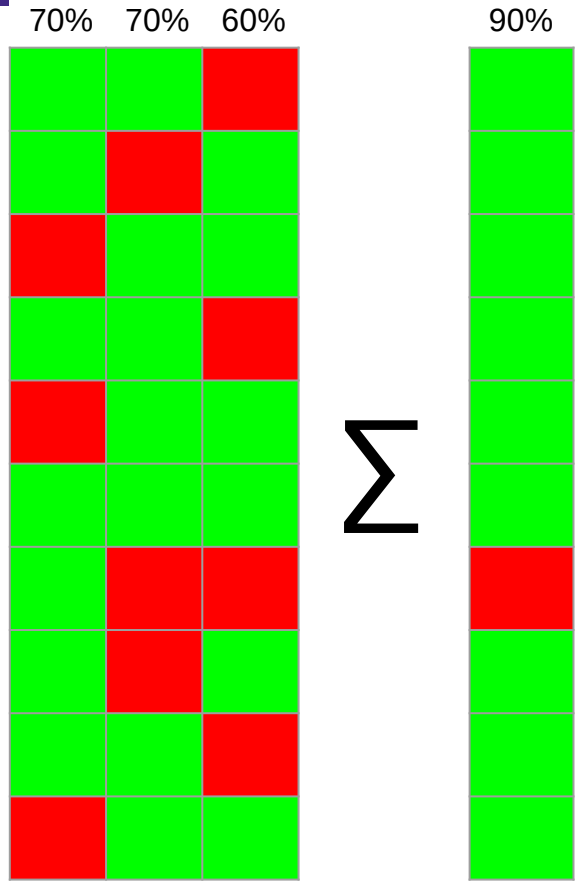
8. Wisdom of the crowd

CONSULTING BSTORM

8. Wisdom of the crowd

ensemble.AdaBoostClassifier([...])	An AdaBoost classifier.
ensemble.AdaBoostRegressor([base_estimator, ...])	An AdaBoost regressor.
ensemble.BaggingClassifier([base_estimator, ...])	A Bagging classifier.
ensemble.BaggingRegressor([base_estimator, ...])	A Bagging regressor.
ensemble.ExtraTreesClassifier([...])	An extra-trees classifier.
ensemble.ExtraTreesRegressor([n_estimators, ...])	An extra-trees regressor.
ensemble.GradientBoostingClassifier(*[, ...])	Gradient Boosting for classification.
ensemble.GradientBoostingRegressor(*[, ...])	Gradient Boosting for regression.
ensemble.IsolationForest(*[, n_estimators, ...])	Isolation Forest Algorithm.
ensemble.RandomForestClassifier([...])	A random forest classifier.
ensemble.RandomForestRegressor([...])	A random forest regressor.
ensemble.RandomTreesEmbedding([...])	An ensemble of totally random trees.
ensemble.StackingClassifier(estimators[, ...])	Stack of estimators with a final classifier.
ensemble.StackingRegressor(estimators[, ...])	Stack of estimators with a final regressor.
ensemble.VotingClassifier(estimators, *[, ...])	Soft Voting/Majority Rule classifier for unfitted estimators.
ensemble.VotingRegressor(estimators, *[, ...])	Prediction voting regressor for unfitted estimators.
ensemble.HistGradientBoostingRegressor([...])	Histogram-based Gradient Boosting Regression Tree.
ensemble.HistGradientBoostingClassifier([...])	Histogram-based Gradient Boosting Classification Tree.

Scikit-learn



Aucun des 3 modèles ne dépassent les 70% de bonnes classifications

Si on combine les résultats on passe à 90%

Wisdom of the crowd - sagesse de la foule

Une foule de personnes éclairées aura plus souvent raison qu'un seul expert

Applicable en machine learning

Attention: la performance de chaque modèle doit au moins être égal à 51%

Sinon convergence vers 0

Scikit-learn

8. Wisdom of the crowd

sklearn.ensemble.Bagging

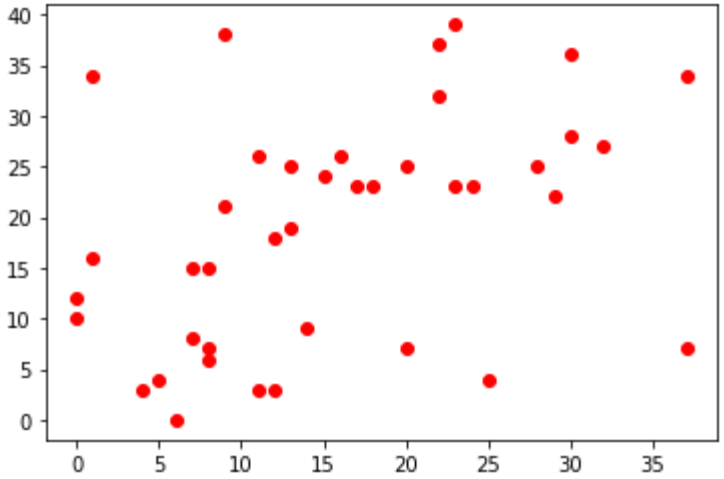
Bootstrap **Agg**regat**ing**

L’idée ici est d’entraîner une foule d’un même modèle mais en utilisant le **bootstrapping**

Bootstrapping: méthode d'échantillonnage où l’on tire au hasard des individus et que l’on replace après

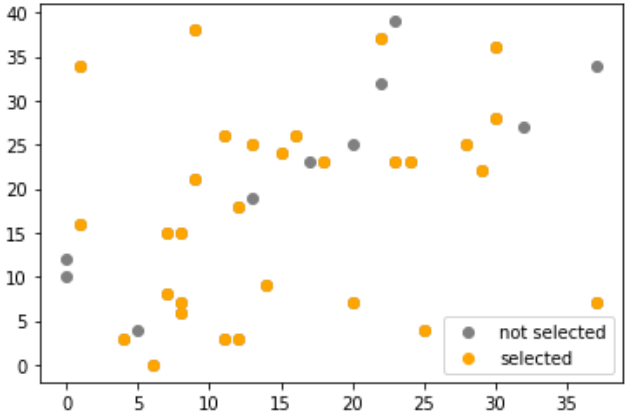
8. Wisdom of the crowd

Dataset original

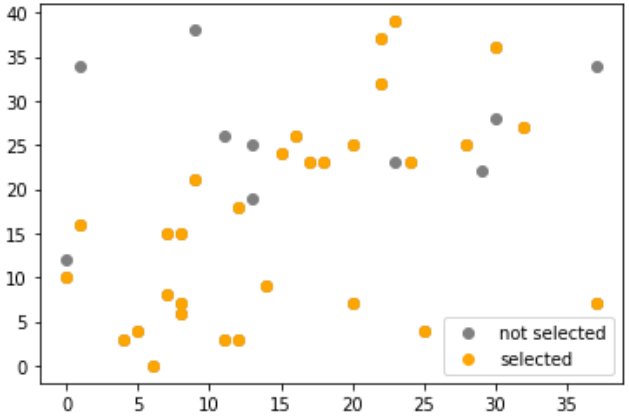


Bootstrapping

modèle 1



modèle 2



Scikit-learn

8. Wisdom of the crowd

Les modèles partagent donc une partie de l'information du dataset

L'idée est de "**moyenner**" le résultat de tous les modèles

Un des plus connus: **sklearn.ensemble.RandomForest**

- L'idée est d'entraîner un ensemble d'arbres de décision

8. Wisdom of the crowd

sklearn.ensemble.Boosting

sklearn.ensemble.AdaBoost

On entraîne une succession de modèle faible

Sur base des erreurs du précédent on modifie le poids des individus

Plus de poids donné aux erreurs

8. Wisdom of the crowd

Star du boosting: XGBoost (e**X**treme **G**radient **B**oosting):

- Beaucoup plus optimisé que les arbres de sklearn
- Compatible sklearn
- Calcul sur GPU

8. Wisdom of the crowd

Bagging:

- entraînement en **parallèle**
- foule d'**experts**
- pris individuellement, les modèles sont en **overfitting**
- la foule permet de réduire l'**overfitting/variance**

Boosting:

- entraînement **successif**
- foule de modèle **faible**
- pris individuellement, les modèles sont en **underfitting**
- permet de réduire l'**underfitting/biais**

8. Wisdom of the crowd

sklearn.ensemble.stacking

On entraîne des modèles **différents**

Ensuite, on entraîne **par dessus** ces modèles un modèle qui doit déterminer qui a raison

Les modèles doivent être différents et ne pas faire les même erreurs

9. Multi-class classification

CONSULTING BSTORM

9. Multi-class classification

La multi-class classification décrit le cas où on a **plus que deux classes à prédire**

Pourquoi est-ce un problème ?

=> Certains algorithmes ne peuvent traiter que des cas de classification **binaire**

E.g.: la régression logistique, SVM, etc.

=> Nous devons subdiviser notre problème en plusieurs problèmes de classification binaire

9. Multi-class classification

sklearn.multiclass.OneVsRestClassifier

Imaginons un problème avec 3 classes: [‘chien’,’chat’,’tortue’]

Une première stratégie consiste à opposer 1 classe versus toutes les autres:

- chien VS reste
- chat VS reste
- tortue VS reste

=> Dans ce cas précis, en réalité nous entraînons **3** régressions logistiques

=> One Versus All/Rest (**OVA/OVR**)

9. Multi-class classification

La prédiction correspondra au modèle avec la plus haute probabilité

=> Il faut entraîner autant de modèle que l’on a de classes

9. Multi-class classification

sklearn.multiclass.OneVsOneClassifier

Approche différente qui consiste à opposer les deux à deux:

- chien vs chat
- chien vs tortue
- chat vs tortue

=> One Vs One (**OVO**)

L’attribution suit la même règle que pour la stratégie OVR

9. Multi-class classification

Les deux semblent similaires mais en réalité si le nombre de modèles à entraîner augmente de manière linéaire avec l'approche **OVR**, ce n'est pas le cas avec l'approche **OVO**

Imaginons 10 classes:

OVR = **10 modèles**

OVO = (n_classes*(n_classes-1))/2 = **45 modèles**

Ceci dit, l'approche OVO ne garde qu'une **partie du dataset** pour l'entraînement contrairement à l'approche OVR qui garde **l'entièreté**

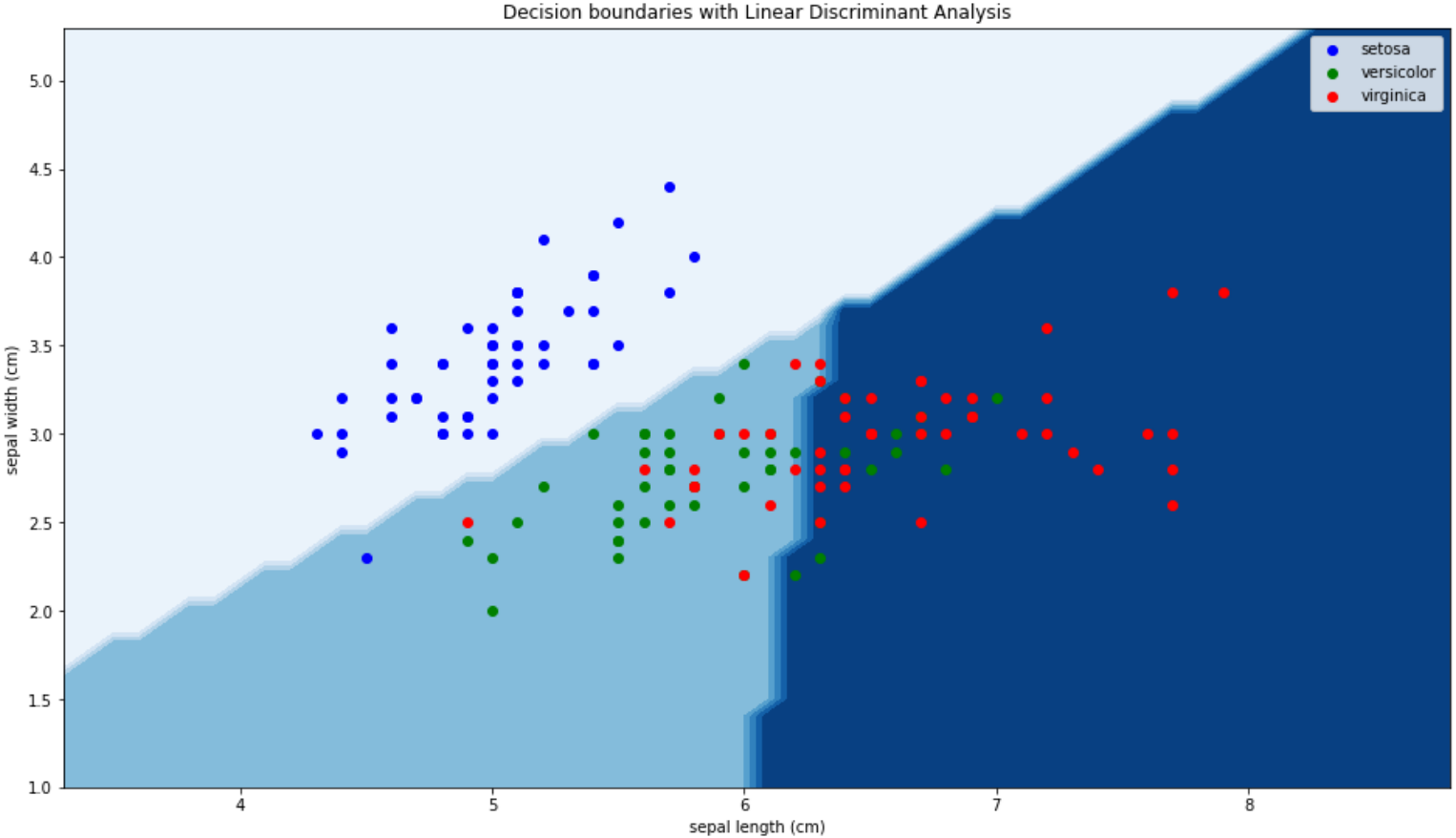
9. Multi-class classification

Prenons l'exemple du dataset des fleurs d'iris.

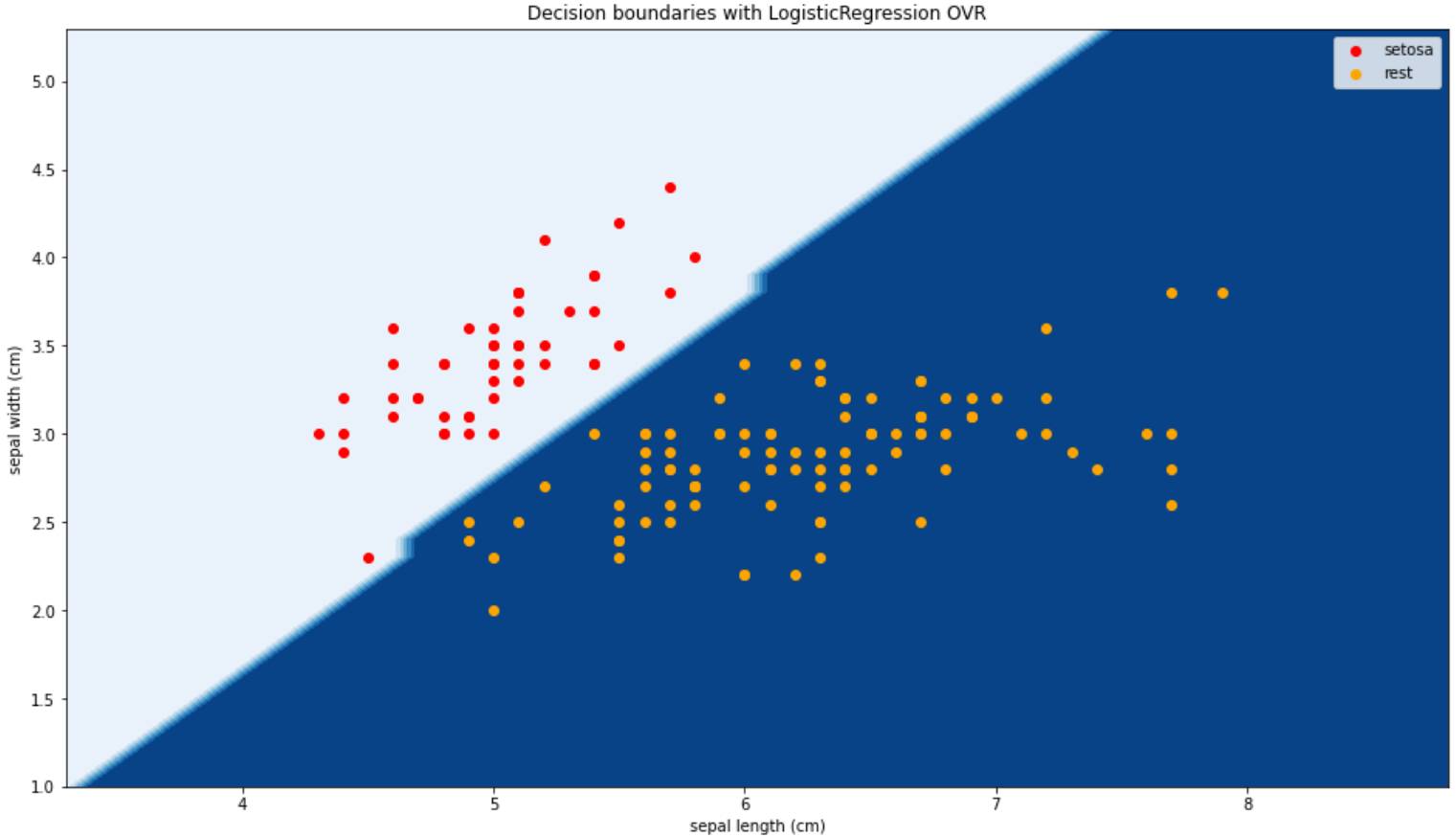
Il est composé de **trois classes** (setosa, versicolor, virginica)

D'abord prenons le cas d'une **Analyse discriminante linéaire** qui peut nativement être utilisée pour un problème de multi-class classification

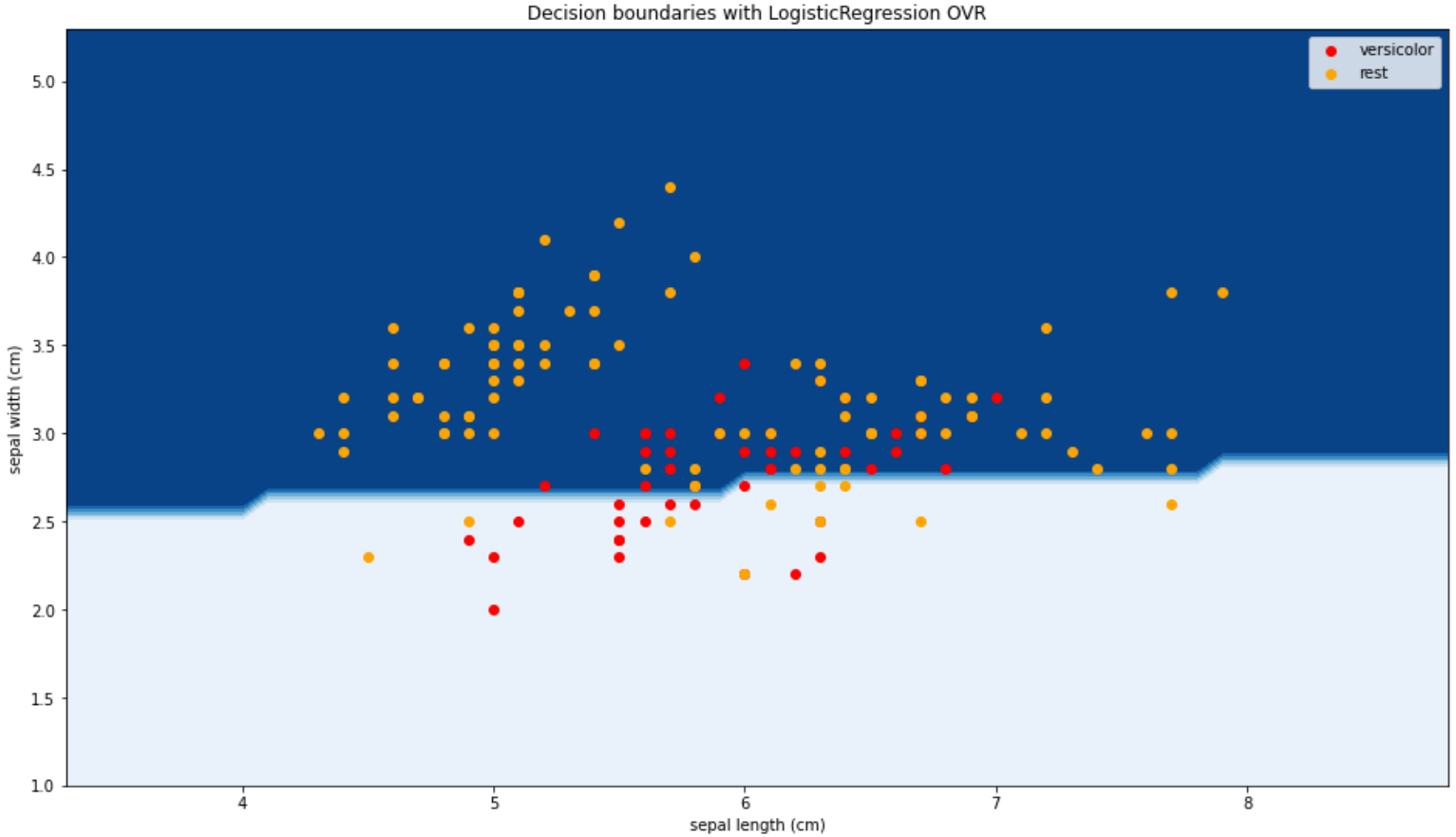
Ensuite nous illustrerons l'approche OVR avec la **régression logistique**



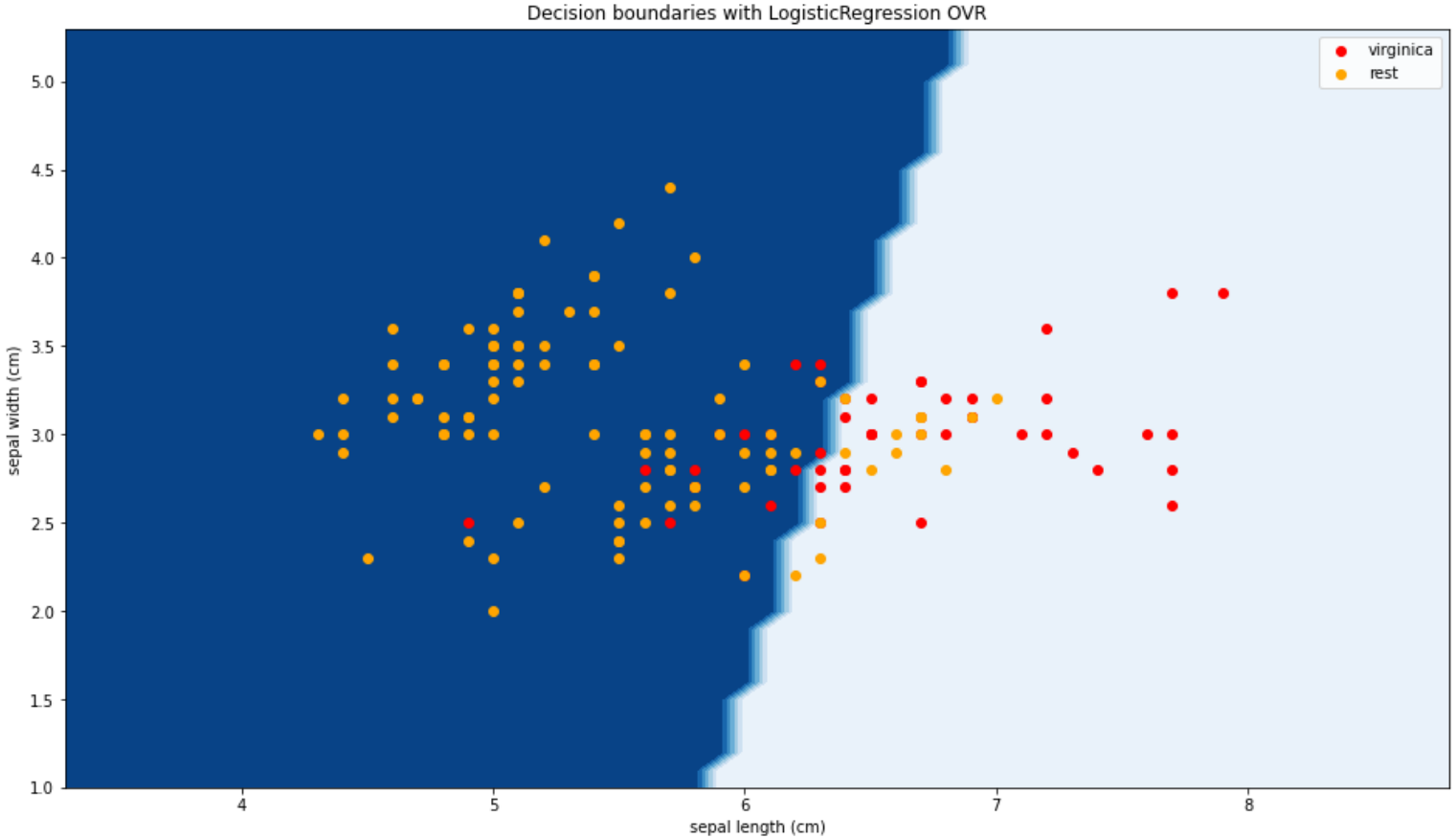
Scikit-learn



Scikit-learn



Scikit-learn



Scikit-learn

9. Multi-class classification

Chaque régression logistique nous renvoie les probabilités pour 1 classe précise vs les autres

	0	1
0	0.89272	0.10728
1	0.771046	0.228954
2	0.925862	0.0741382
3	0.927383	0.0726168
4	0.941261	0.058739
5	0.914367	0.0856335
6	0.970589	0.0294111
7	0.894845	0.105155
8	0.93034	0.0696599

	0	1
0	0.117729	0.882271
1	0.30009	0.69991
2	0.205972	0.794028
3	0.245352	0.754648
4	0.0940318	0.905968
5	0.050685	0.949315
6	0.13715	0.86285
7	0.143281	0.856719
8	0.338067	0.661933

	0	1
0	0.0477808	0.952219
1	0.0399172	0.960083
2	0.0234961	0.976504
3	0.0198779	0.980122
4	0.036769	0.963231
5	0.0737362	0.926264
6	0.0170924	0.982908
7	0.0405783	0.959422
8	0.0142041	0.985796

Scikit-learn

9. Multi-class classification

	0	1	2
0	0.89272	0.117729	0.0477808
1	0.771046	0.30009	0.0399172
2	0.925862	0.205972	0.0234961
3	0.927383	0.245352	0.0198779
4	0.941261	0.0940318	0.036769
5	0.914367	0.050685	0.0737362
6	0.970589	0.13715	0.0170924
7	0.894845	0.143281	0.0405783
8	0.93034	0.338067	0.0142041

On ne garde que la première colonne de chaque régression

La règle d'affectation est simple puisqu'il s'agit de la colonne avec la plus haute probabilité