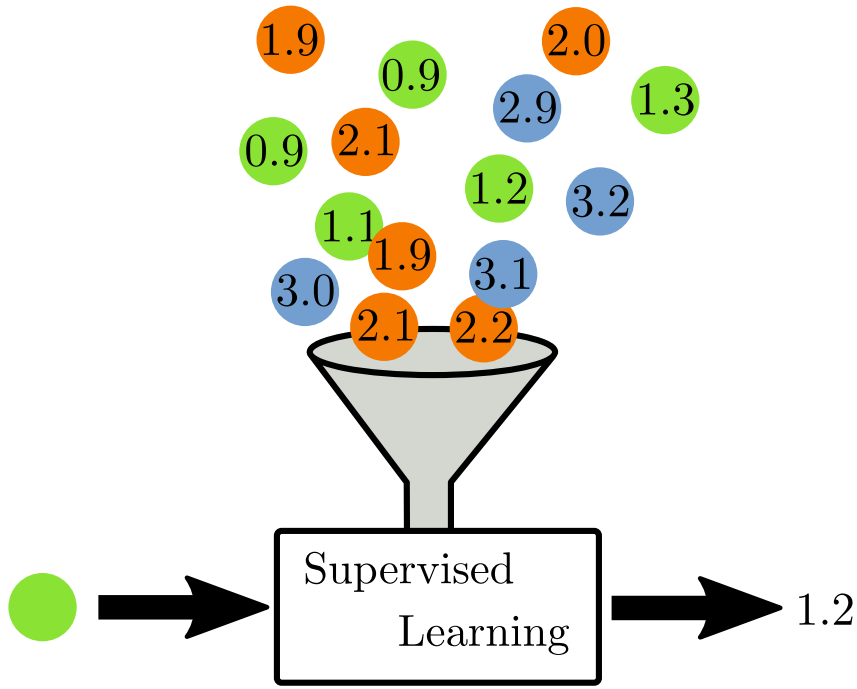
Introduction à la Data Science
Apprentissage supervisé

Benoit Gaüzère

INSA Rouen Normandie - Laboratoire LITIS

31 mars 2026

De quoi parle-t-on ?



- Apprendre à partir d'exemples
- Imiter des tâches humaines (IA ?)
- Produire des sorties à partir d'entrées (fonctions ?)

Apprentissage supervisé

Objectif

Étant donné un ensemble de données $\{(\mathbf{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}, i = 1, \ldots, N\}$, apprendre la dépendance entre $\mathcal{X}$ et $\mathcal{Y}$.

- **Exemple :** Apprendre les liens entre le risque cardiaque et les habitudes alimentaires. $\mathbf{x}_i$ décrit une personne (via d caractéristiques) ; y_i est une catégorie binaire (à risque, pas à risque).
- y_i est **essentiel** pour le processus d'apprentissage.
- Méthodes : K-Nearest Neighbors, SVM, Arbre de décision, ...

Capsule de rappel : Structure des données X et y

Comment sont organisées les données en apprentissage supervisé ?

- X : tableau des **caractéristiques (features)** des observations.
- y : vecteur des **valeurs cibles (labels)** à prédire.

Taille (cm)	Poids (kg)	Âge (ans)	y
175	70	21	1
160	55	20	0
180	80	25	1

Comment encoder les données

$\mathbf{X} \in \mathbb{R}^{n \times p}$

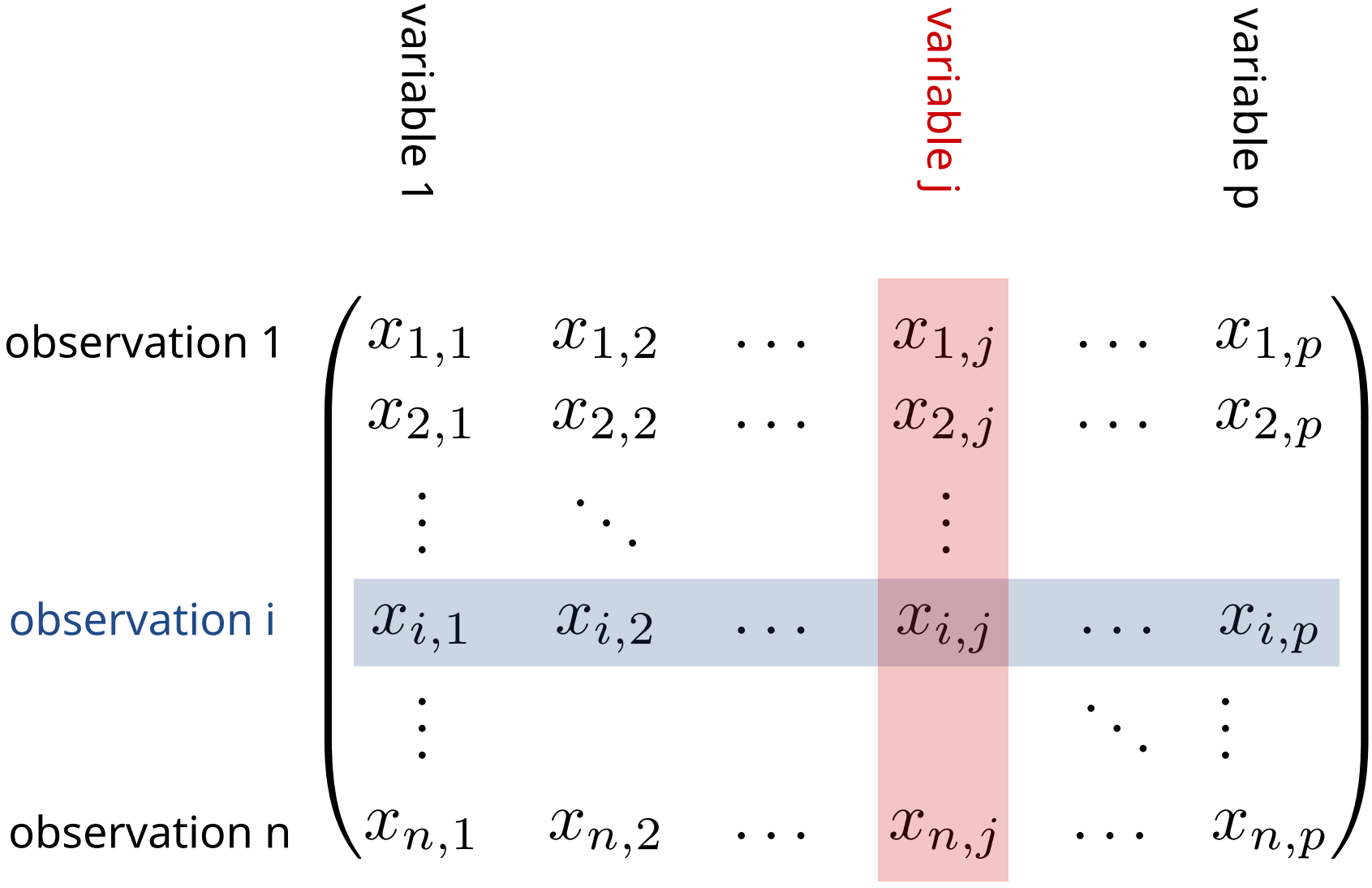
Échantillons (Samples)

- n échantillons (nombre de lignes)
- $\mathbf{X}(i,:) = \mathbf{x}_i^\top$: le i-ème échantillon
- $\mathbf{x}_i \in \mathbb{R}^p$

Caractéristiques (Features)

- p caractéristiques (nombre de colonnes)
- Chaque échantillon est décrit par p caractéristiques
- $\mathbf{X}(:,j) = \boldsymbol{x}_{.j}$: la j-ème caractéristique pour tous les échantillons

$\mathbf{X}(i,j)$: j-ème caractéristique du i-ème échantillon.


variable 1
variable j
variable p
observation 1
observation i
observation n
$\begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,j} & \cdots & x_{1,p} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,j} & \cdots & x_{2,p} \\ \vdots & \ddots & & \vdots & & \\ x_{i,1} & x_{i,2} & \cdots & x_{i,j} & \cdots & x_{i,p} \\ \vdots & & & & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,j} & \cdots & x_{n,p} \end{pmatrix}$

Modèle d'apprentissage

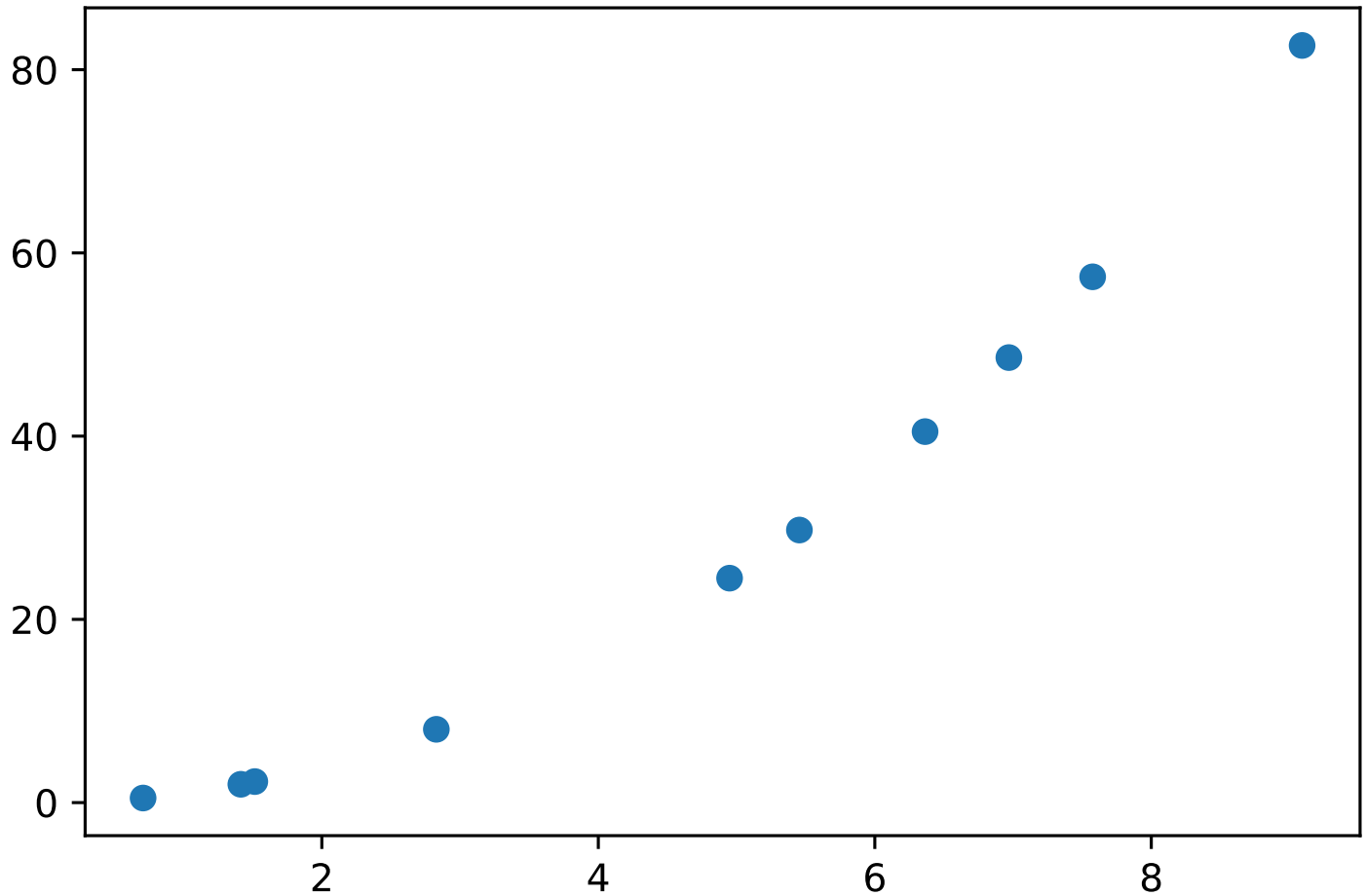
Modèle

$$f : \mathcal{X} \to \mathcal{Y} \quad ; \quad \mathbf{x}_i \mapsto \hat{y}$$

Nous voulons que

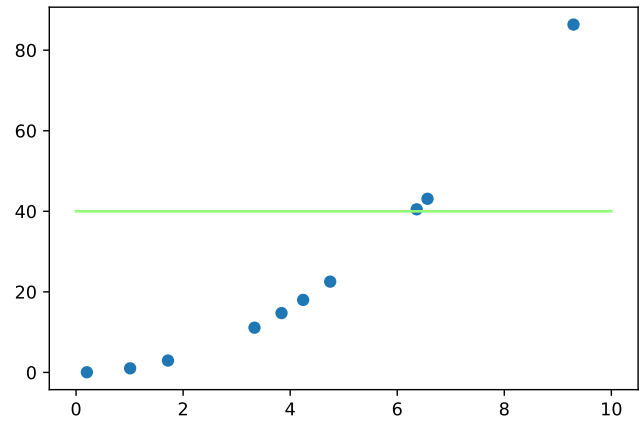
$$f(\mathbf{x}_i) \simeq y_i.$$

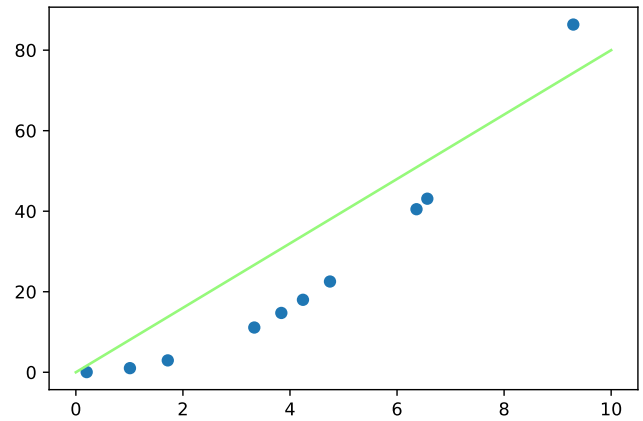
Exemple

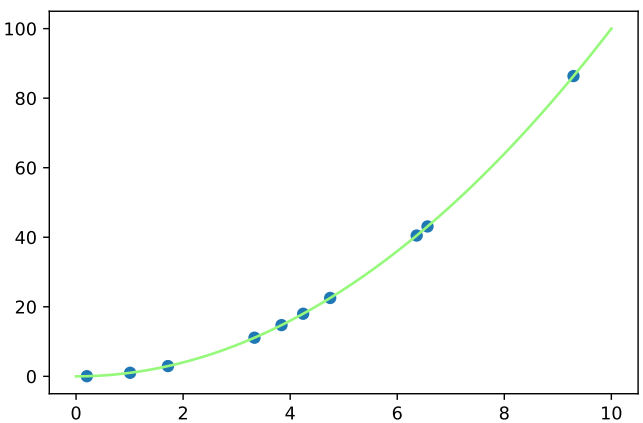


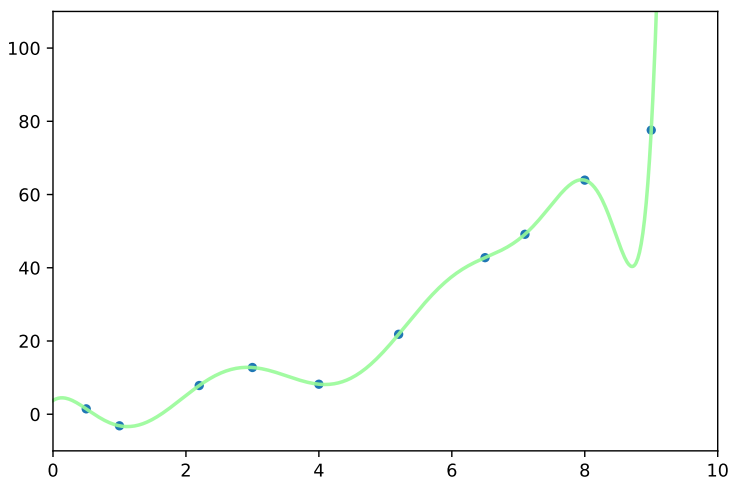
Quelle est la fonction f sous-jacente ?

Exemple


0
20
40
60
80
0
2
4
6
8
10


0
20
40
60
80
0
2
4
6
8
10


0
20
40
60
80
100
0
2
4
6
8
10


0
20
40
60
80
100
0
2
4
6
8
10

Comment trouver un bon f ?

$$f^\star = \arg\min_f \Big(\mathcal{L}(f(\mathbf{X}), \mathbf{y}) + \lambda\, \Omega(f) \Big).$$

- $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$
- $\lambda \in \mathbb{R}^+$
- $\Omega : (\mathcal{X} \to \mathcal{Y}) \to \mathbb{R}^+$

Terme d'ajustement aux données

$$\mathcal{L}\Big(f(\mathbf{X}), \mathbf{y}\Big)$$

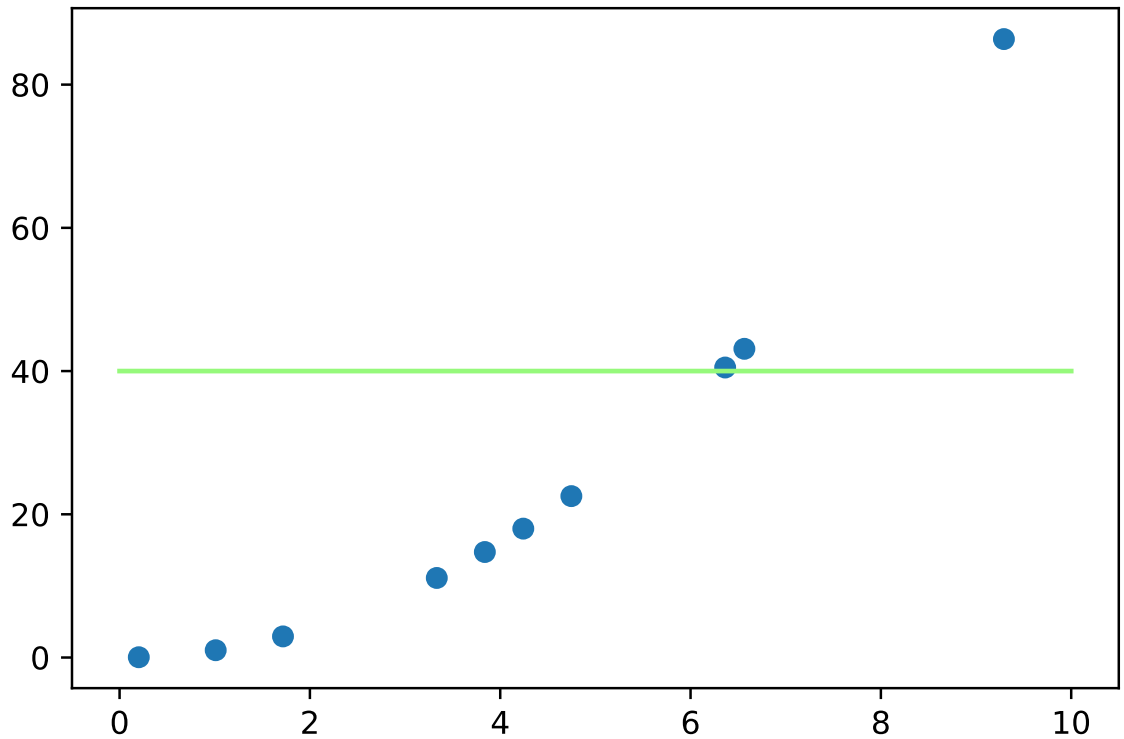
- Garantit que le modèle s'ajuste aux données
- Pénalise lorsque la valeur prédite $f(\mathbf{x}_i)$ est éloignée de y_i

Terme de régularisation

$$\Omega(f)$$

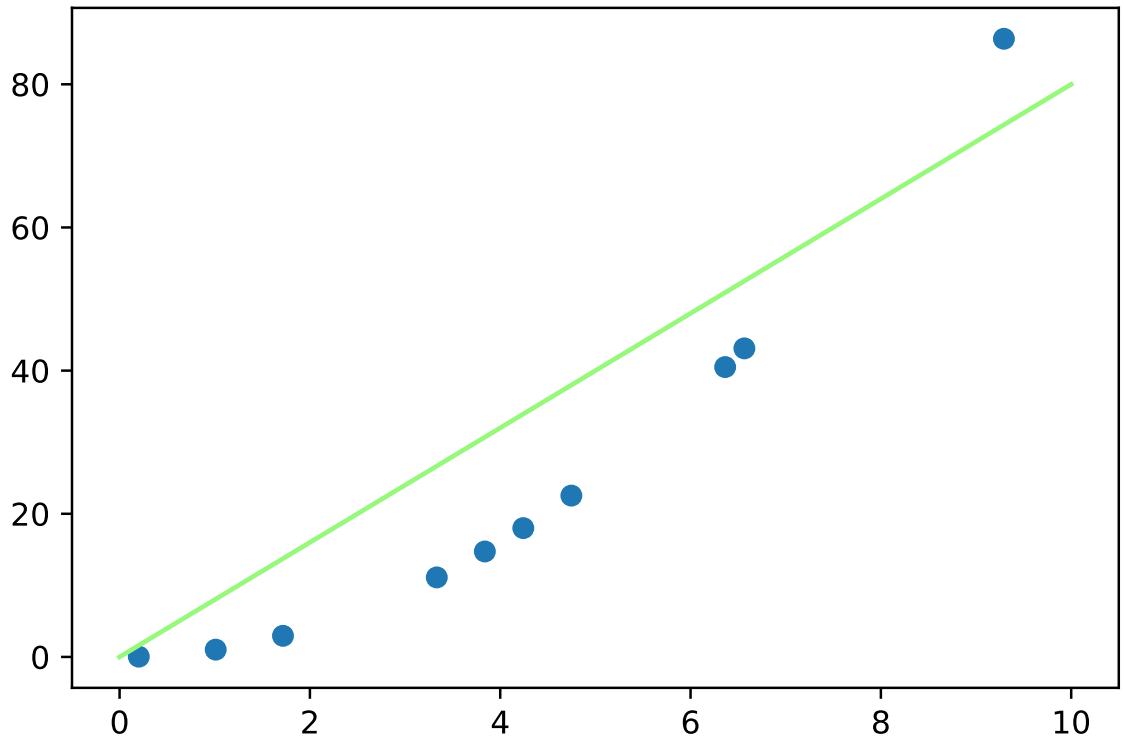
- Contrainte sur la complexité de f
- Rasoir d'Occam : plus c'est simple, mieux c'est
- λ : pondère l'équilibre entre les deux termes

Illustrations I



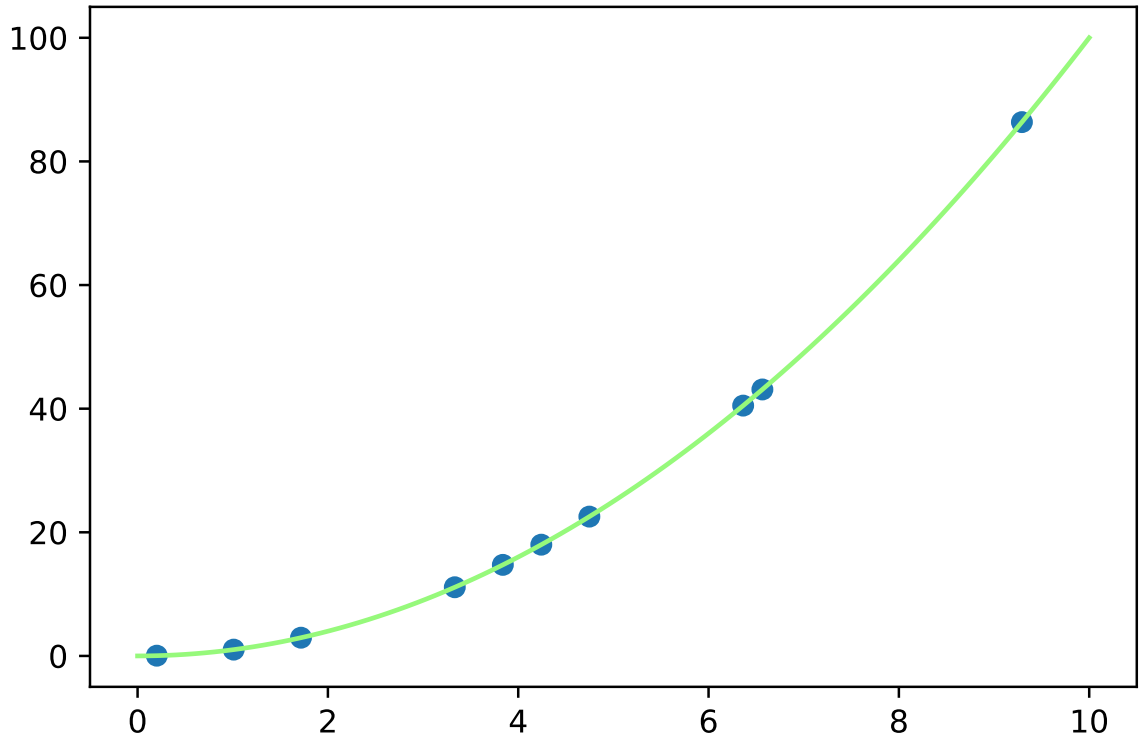
- $\mathcal{L}(f(\mathbf{X}), \mathbf{y})$ très grand
- $\Omega(f)$ très faible

Illustrations II



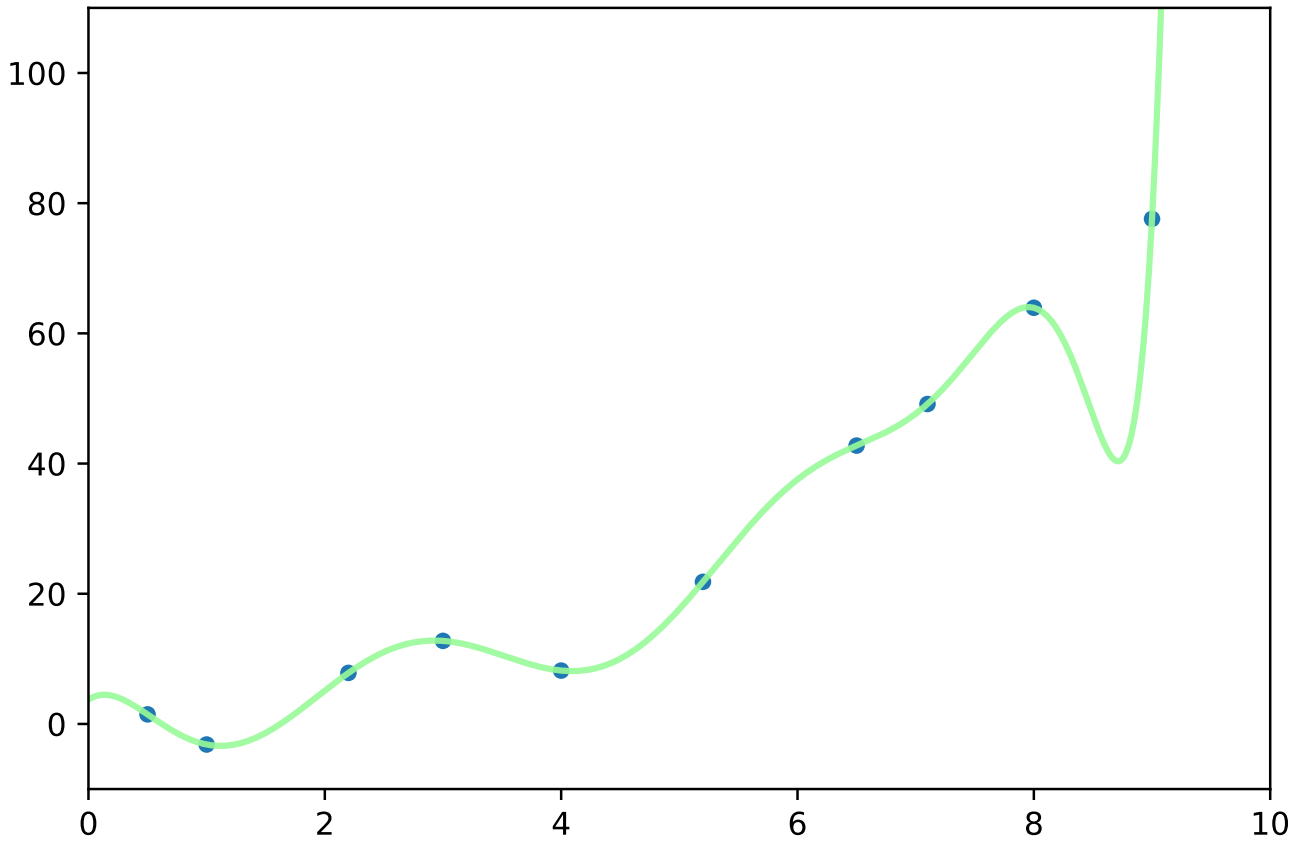
- $\mathcal{L}(f(\mathbf{X}), \mathbf{y})$ élevé
- $\Omega(f)$ faible

Illustrations III



- $\mathcal{L}(f(\mathbf{X}), \mathbf{y}) = 0$
- $\Omega(f)$ élevé

Illustrations IV



- $\mathcal{L}(f(\mathbf{X}), \mathbf{y}) = 0$
- $\Omega(f)$ très élevé

Généralisation

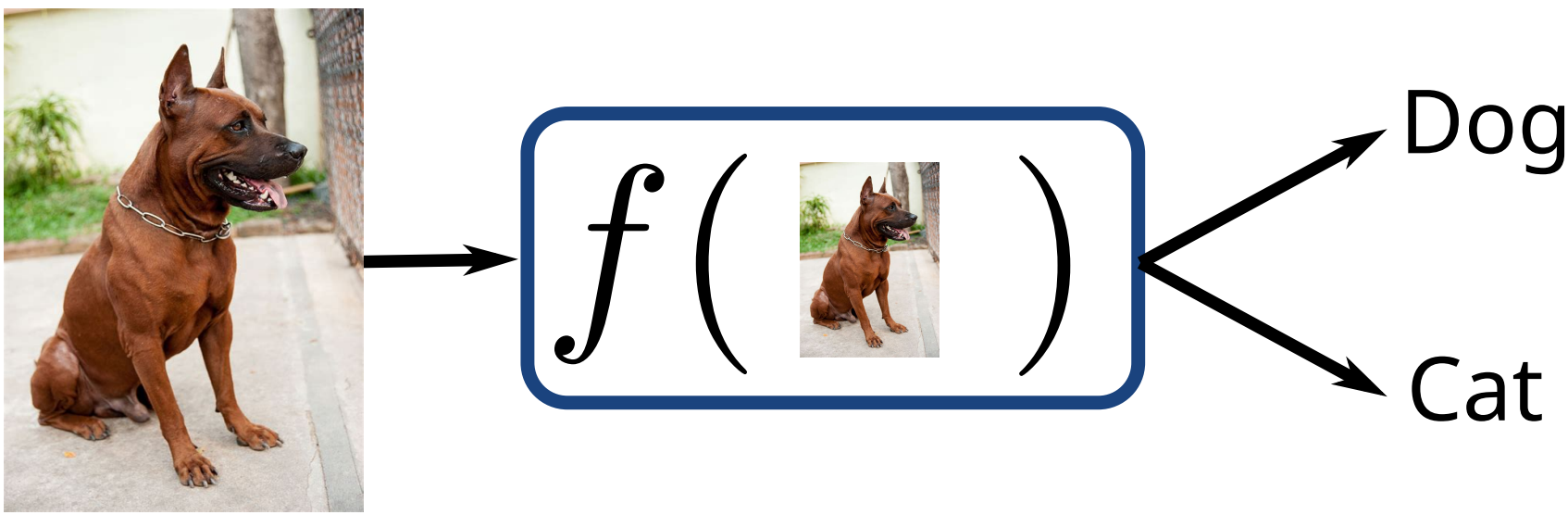
Un bon modèle généralise bien.

- Bonne généralisation : prédire correctement sur des données non vues
- Difficile à évaluer sans biais
- **Overfitting** (surapprentissage)
- Le terme de régularisation prévient le surapprentissage

Tâches en apprentissage supervisé I

Classification binaire

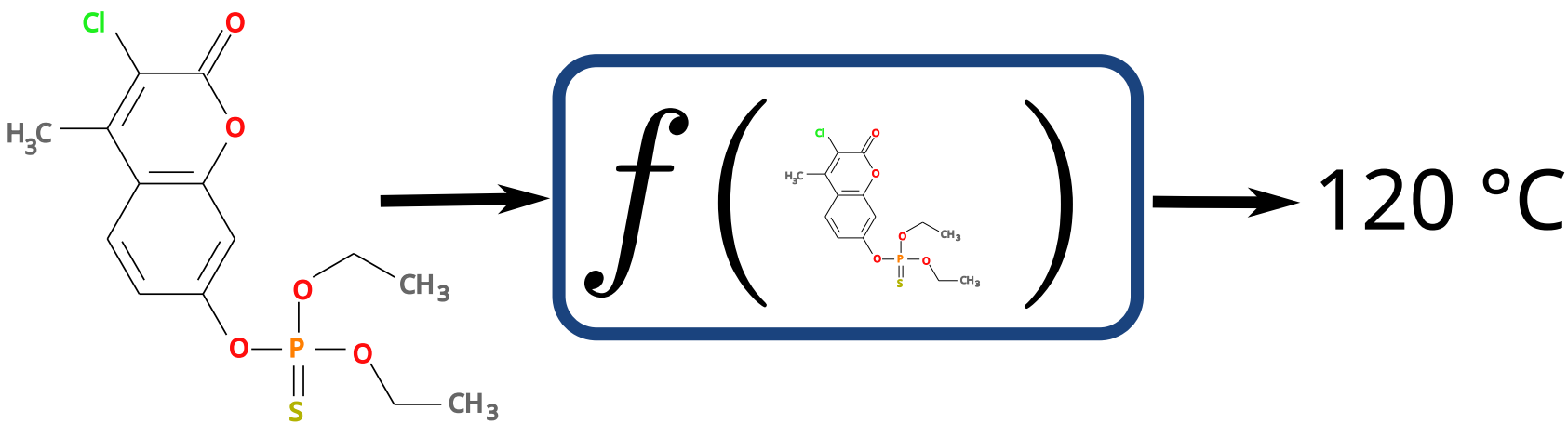
- $\mathcal{Y} = \{0, 1\}$
- Chien ou chat ? Positif ou Négatif ?
- Performances : Accuracy, rappel, précision, etc.



Tâches en apprentissage supervisé II

Régression

- $\mathcal{Y} = \mathbb{R}$
- Marché boursier, prix d'une maison, points d'ébullition de molécules, etc.
- Performances : RMSE, MSE, MAE, etc.



Deux grandes familles de tâches supervisées

Classification

- Prédire une **étiquette (catégorie)**
- Type de y : *discret, catégorique*
- Métriques : *précision, rappel, accuracy, etc.*

Régression

- Prédire une **valeur numérique continue**
- Exemples : Prix d'une maison, Température, Taux de CO_2
- Type de y : *réel, continu*
- Métriques : *MSE, RMSE, MAE, etc.*

Remarque clé

Le choix entre classification et régression dépend uniquement de la nature de la variable y que l'on veut prédire.

Méthodes de Machine Learning pour la Classification

- K-nearest neighbors
- Forêts aléatoires
- SVM & consorts
- Multi Layer Perceptron

k Nearest Neighbors

Principe

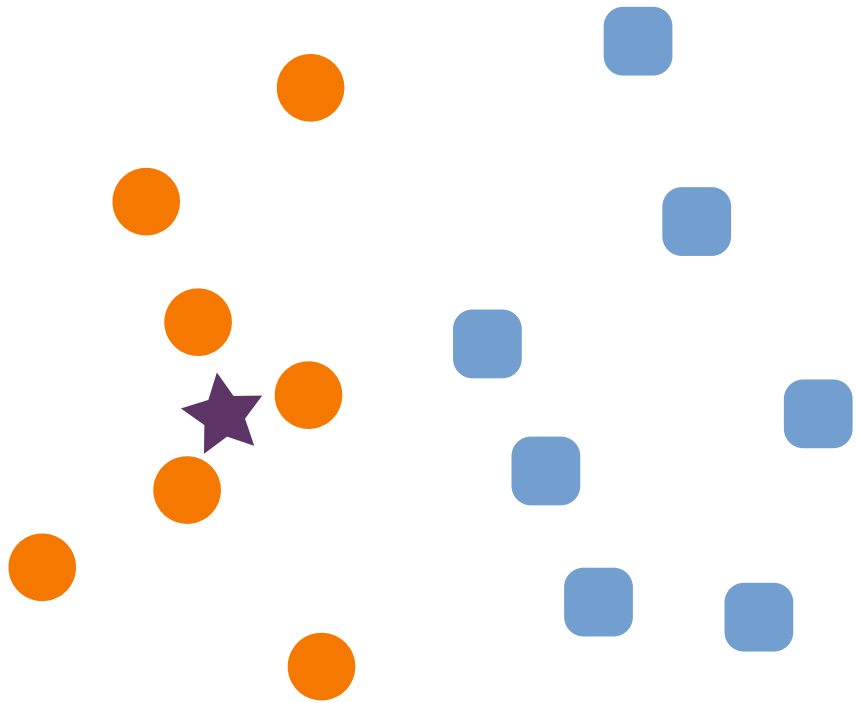
Déterminer la propriété d'un point en se basant sur les données similaires.



k Nearest Neighbors

Principe

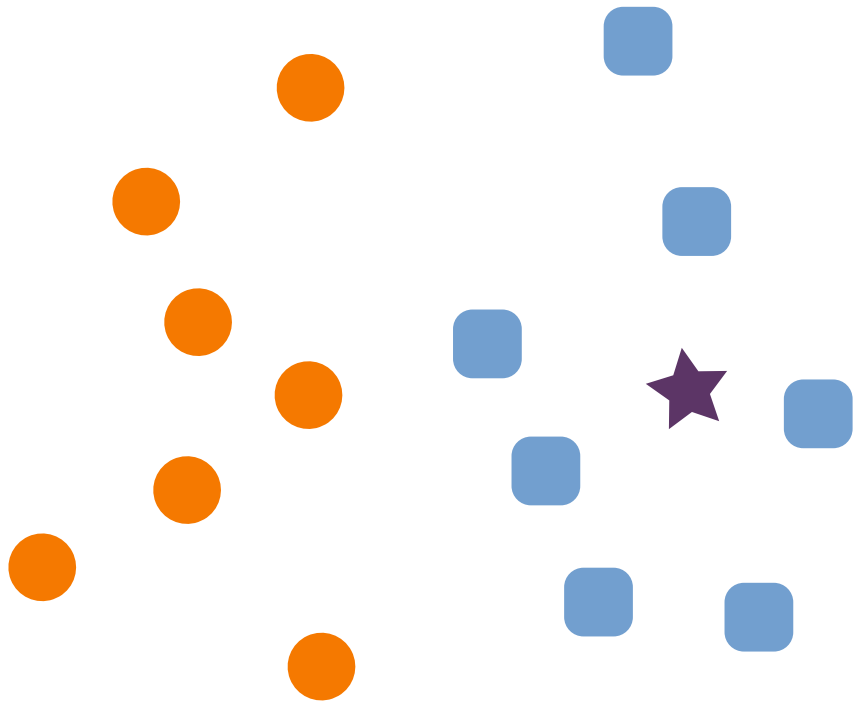
Déterminer la propriété d'un point en se basant sur les données similaires.



k Nearest Neighbors

Principe

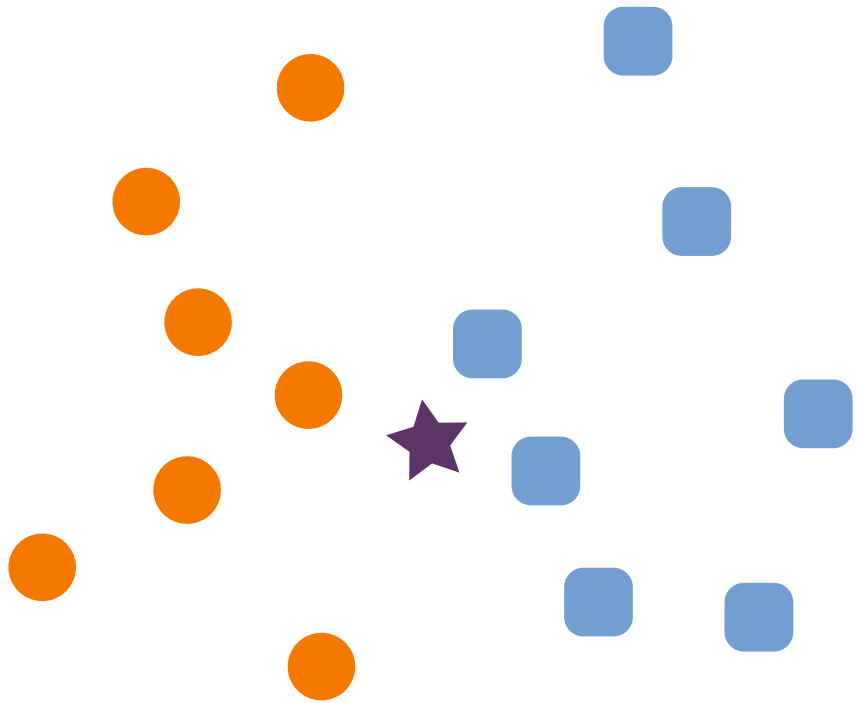
Déterminer la propriété d'un point en se basant sur les données similaires.



k Nearest Neighbors

Principe

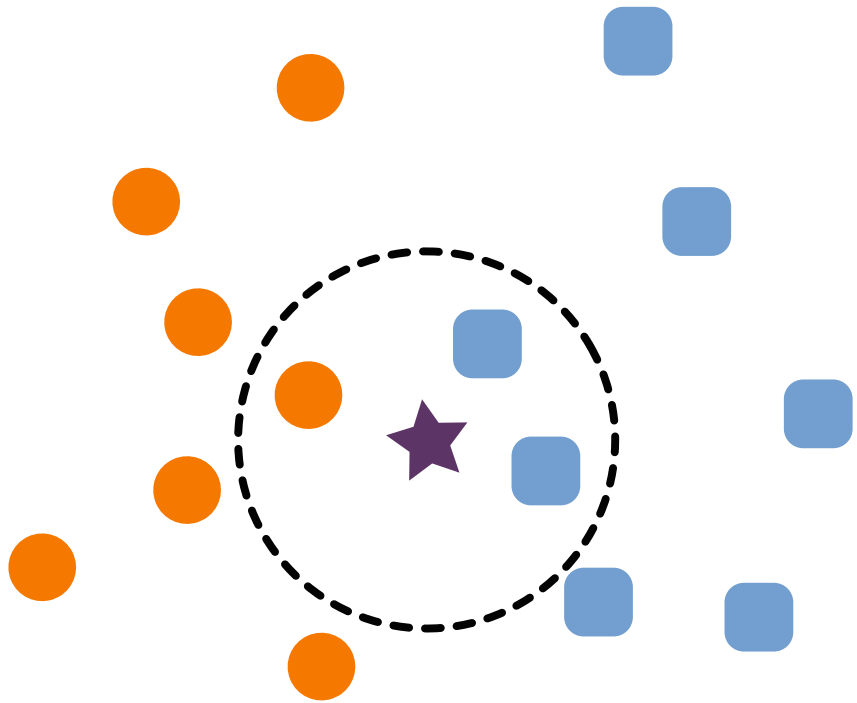
Déterminer la propriété d'un point en se basant sur les données similaires.



k Nearest Neighbors

Principe

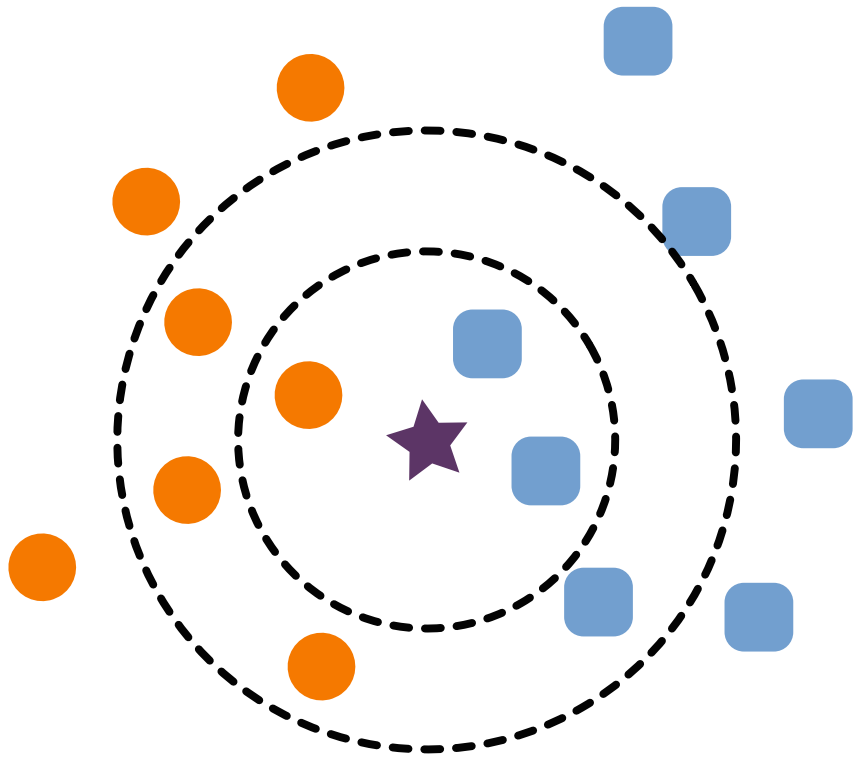
Déterminer la propriété d'un point en se basant sur les données similaires.



k Nearest Neighbors

Principe

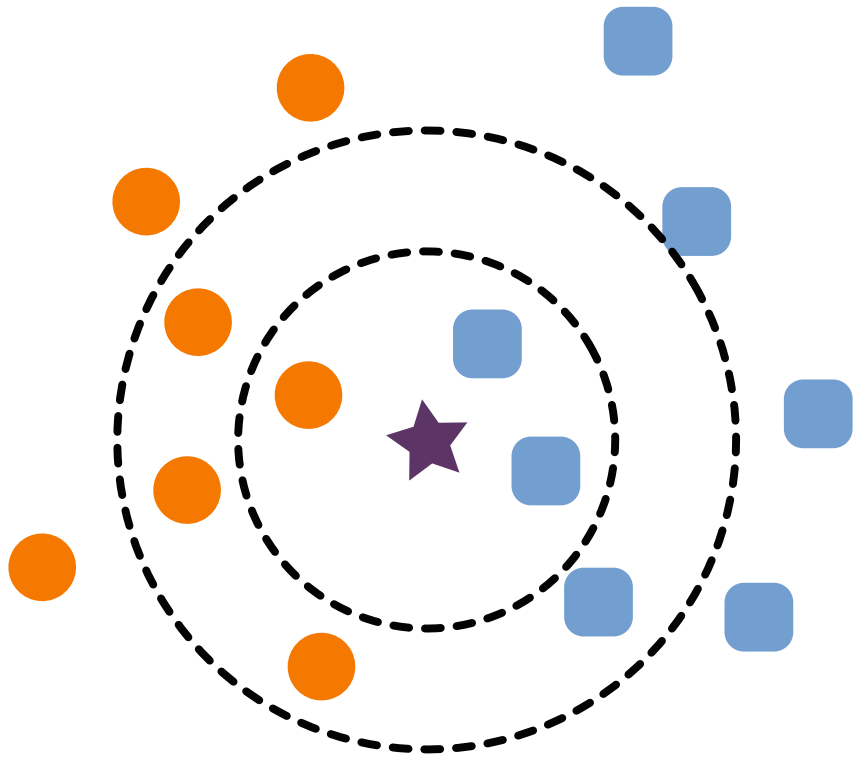
Déterminer la propriété d'un point en se basant sur les données similaires.



k Nearest Neighbors

Principe

Déterminer la propriété d'un point en se basant sur les données similaires.



Capsule de rappel : les hyperparamètres

Paramètres vs Hyperparamètres

- **Paramètres** : appris automatiquement par le modèle (ex. poids d'un réseau de neurones, coefficients d'une régression)
- **Hyperparamètres** : fixés *avant* l'apprentissage ; influencent la manière dont le modèle apprend

Pourquoi c'est important ?

- Impact direct sur la performance du modèle
- Trop de liberté $\Rightarrow$ sur-apprentissage (overfitting)
- Trop de contraintes $\Rightarrow$ sous-apprentissage (underfitting)
- Les hyperparamètres doivent être ajustés avec soin

Les hyperparamètres contrôlent le comportement du modèle et doivent être ajustés via une procédure de validation.

Hyperparamètres du k-NN

Nombre de voisins k

- Petit k : forte variabilité, précision élevée
- Grand k : prédiction plus lissée

Distance

- Détermine la similarité : dépend des données, de la structure, de la tâche
- Distances euclidienne, Manhattan, etc.

KNN : le code !

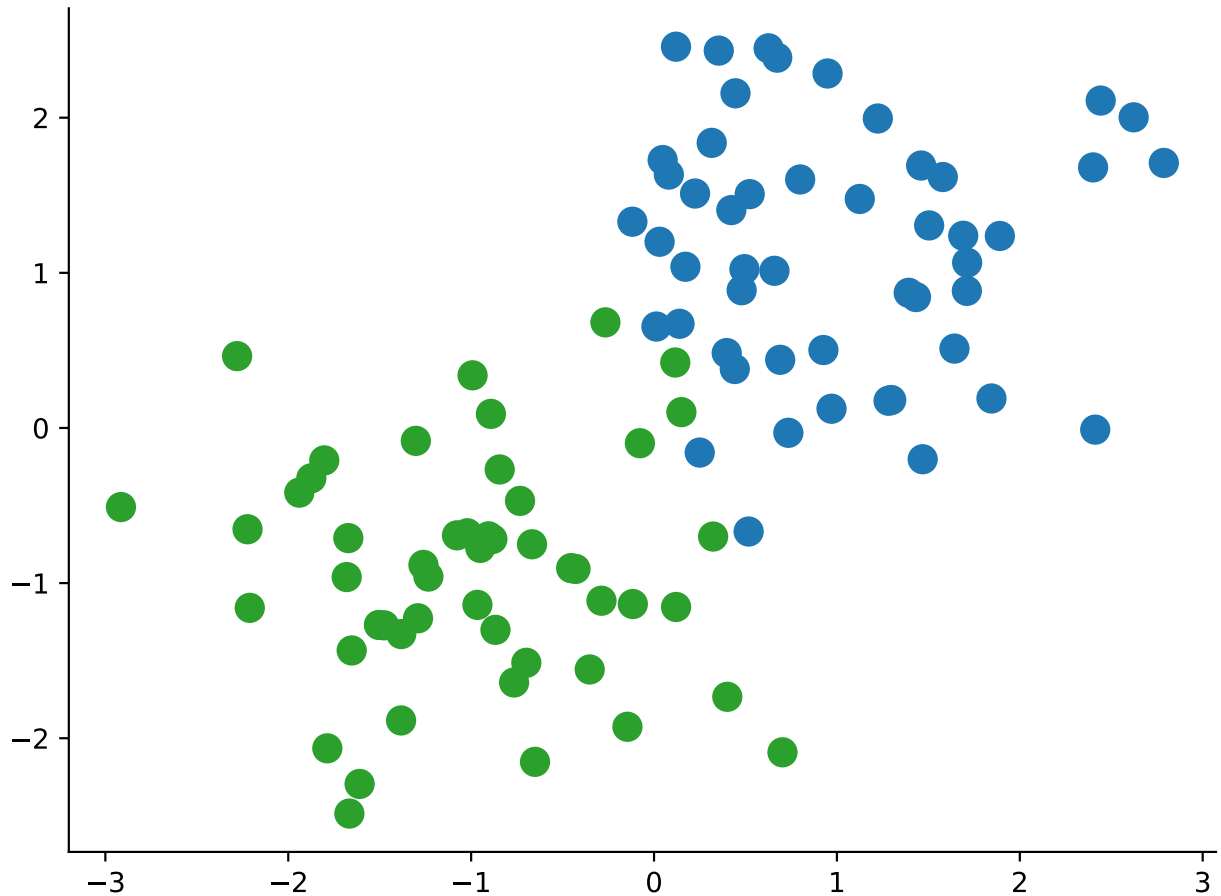
```
1    from sklearn.neighbors import KNeighborsClassifier
2    k=5
3    metric = 'manhattan'
4    knn = KNeighborsClassifier(n_neighbors=k,metric=metric)
5    knn.fit(X,y) # apprentissage
6    y_pred = knn.predict(X) # prédiction
```

- Metrics : lien
- $\Rightarrow$ Notebook
- la documentation

Arbre de décision

Principe

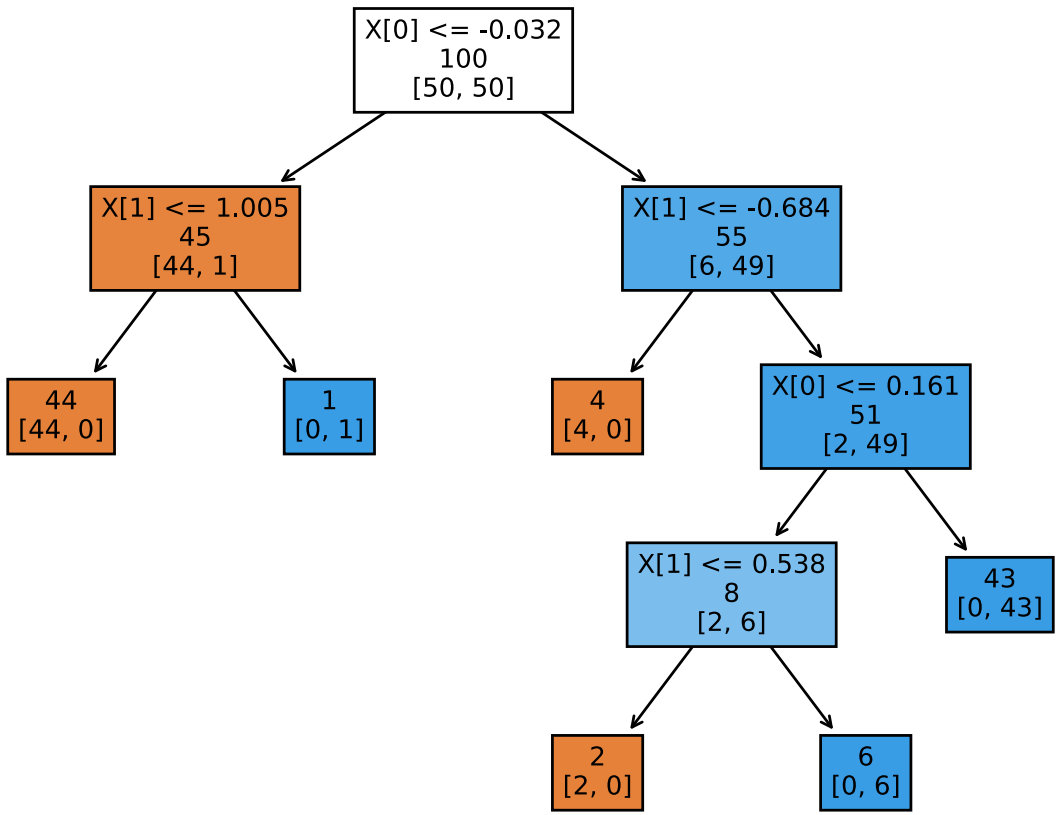
Apprendre des règles de décision pour séparer les données.



Arbre de décision

Principe

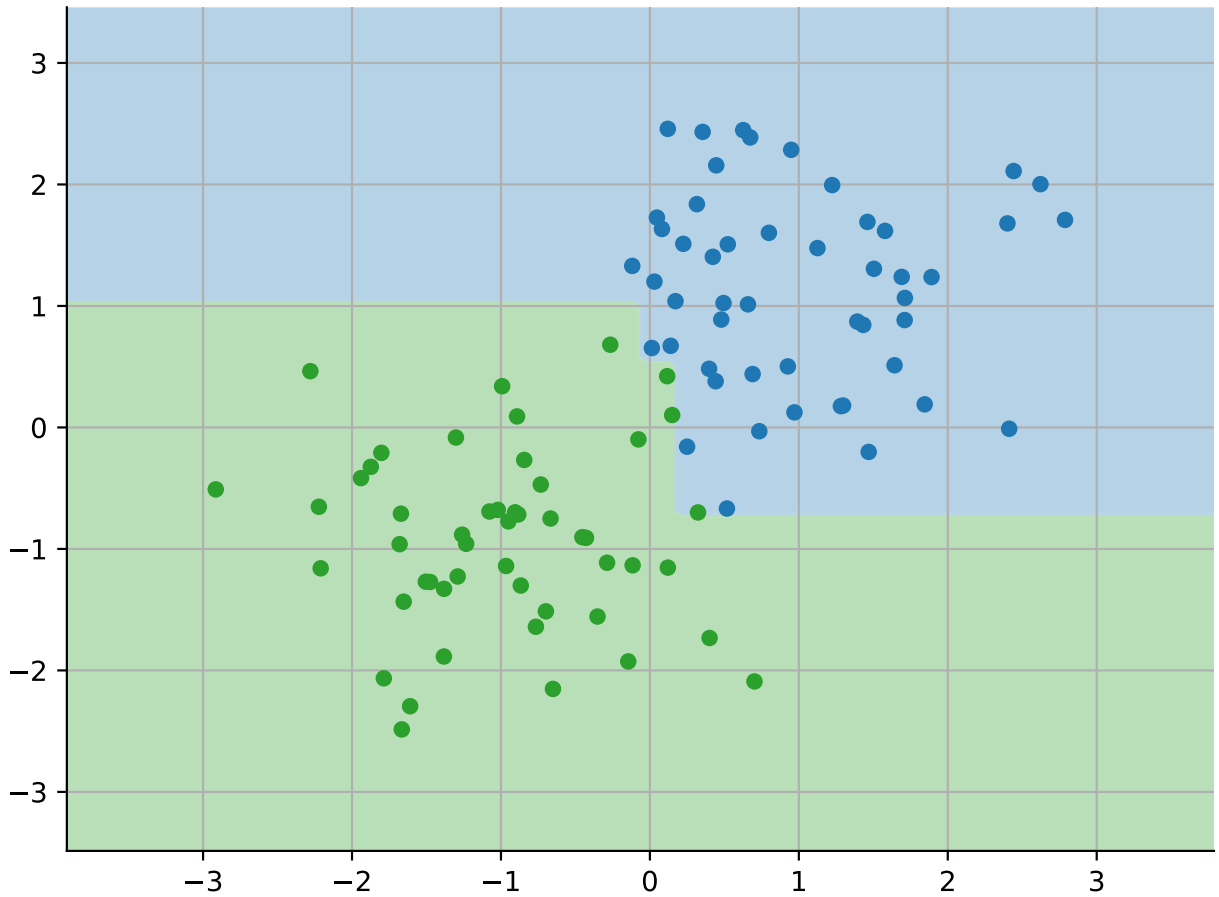
Apprendre des règles de décision pour séparer les données.



Arbre de décision

Principe

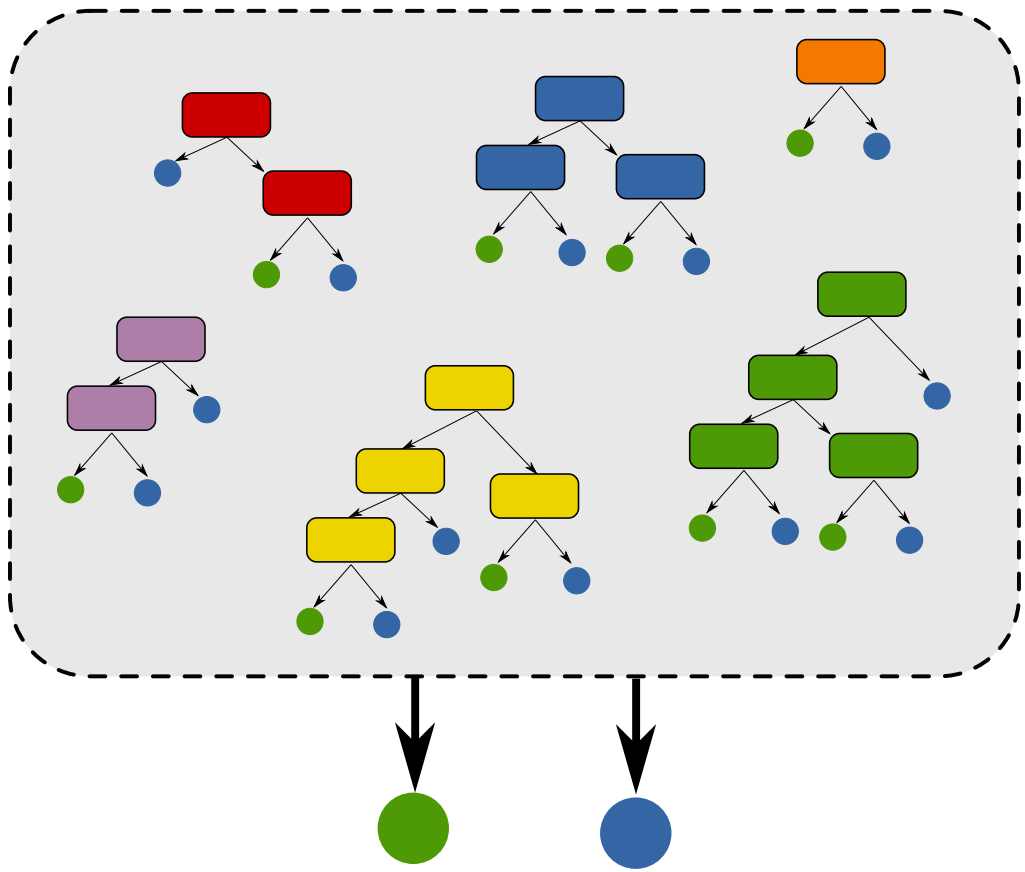
Apprendre des règles de décision pour séparer les données.



Forêts aléatoires (Random Forests)

Principe

- Combiner de nombreux arbres de décision pour apprendre des fonctions complexes
- Méthodes d'ensemble, vote majoritaire



Hyperparamètres des forêts aléatoires

Nombre d'arbres

- Petit nombre : rapide à calculer, mais moins précis
- Nombre élevé : plus lent, plus précis jusqu'à un certain point

Nombre de features

- Définit le nombre de caractéristiques utilisées à chaque division
- Voir les indications de `scikit-learn`

Profondeur de l'arbre

- Spécifie la profondeur maximale de l'arbre
- Plus la profondeur est grande, plus les règles sont spécialisées
- Mais risque de moins bonne généralisation

Forêts aléatoires : le code !

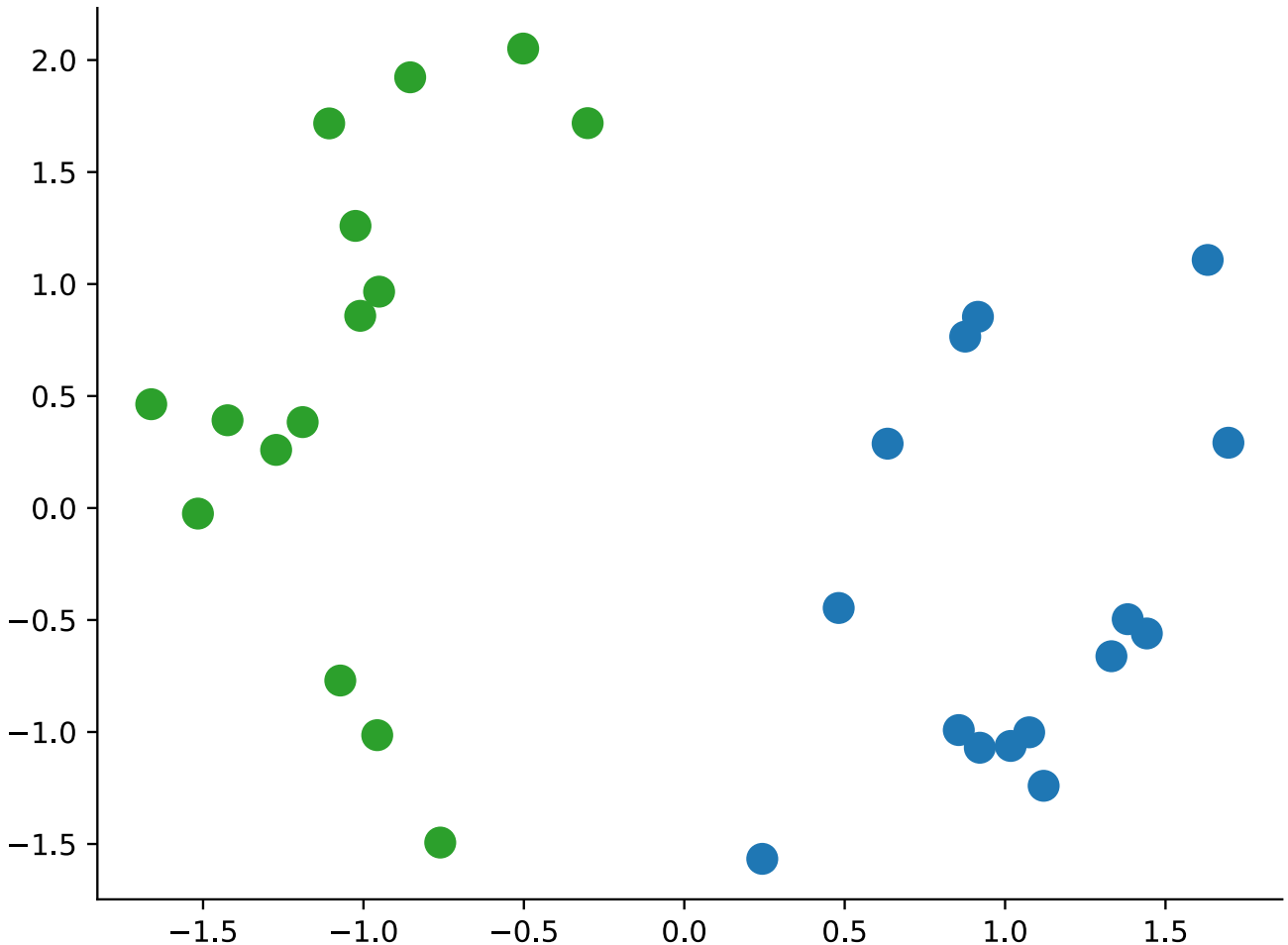
```
1    from sklearn.ensemble import RandomForestClassifier
2    n_estimators = 20 # nombre d'arbres
3    max_depth = None  # expansion maximale
4    max_features = "sqrt" # RTFM
5    clf = RandomForestClassifier(n_estimators=n_estimators,
6    max_depth=max_depth,
7    max_features=max_features)
8    clf.fit(X,y)
9    ypred = clf.predict(X)
```

- Guide utilisateur sur les hyperparamètres : lien
- $\Rightarrow$ Notebook
- la documentation

Support Vector Machines

Principe

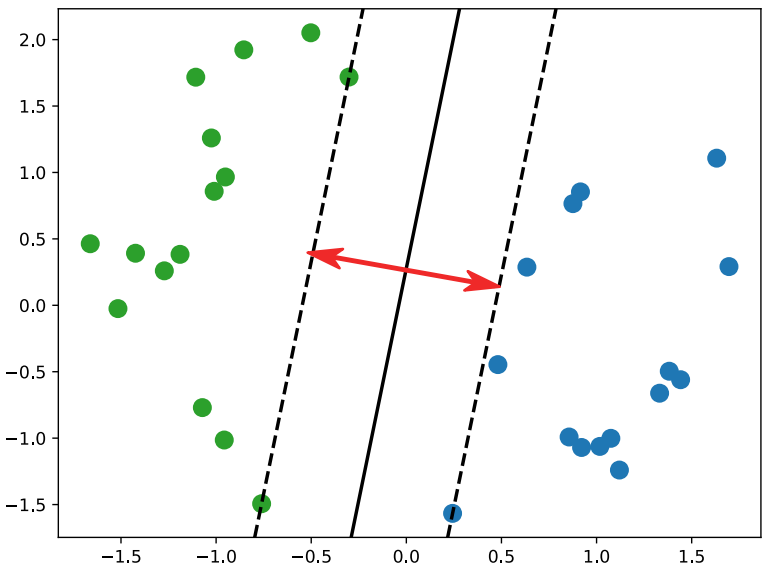
Trouver la meilleure frontière (droite ou hyperplan) pour séparer les données

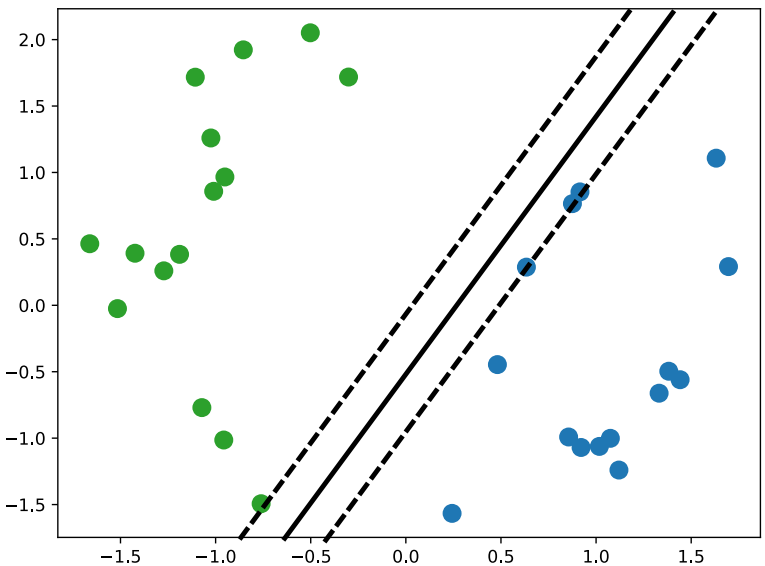


Support Vector Machines

Principe

Trouver la meilleure frontière (droite ou hyperplan) pour séparer les données



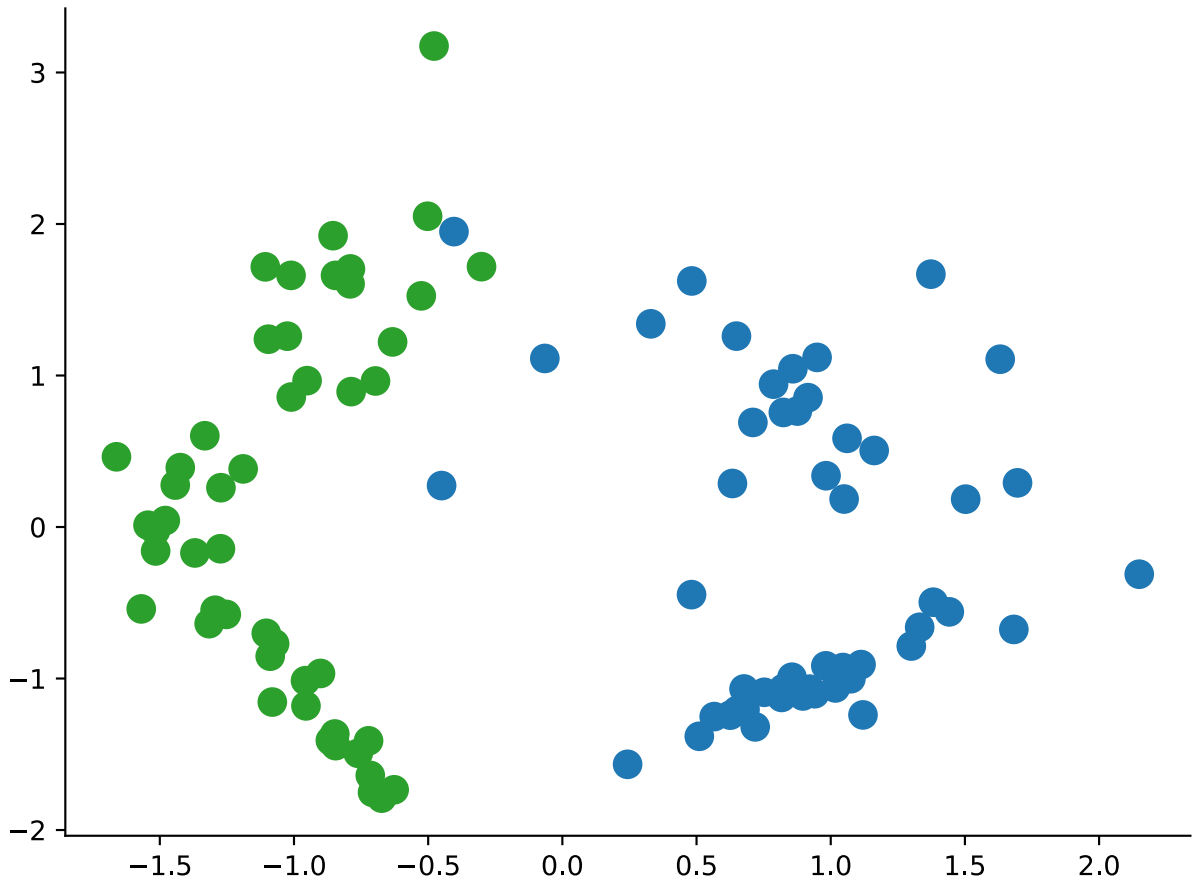


- Meilleure séparation $\Rightarrow$ points les plus éloignés de la frontière
- Points situés sur la marge : *support vectors*

Support Vector Machines

Principe général

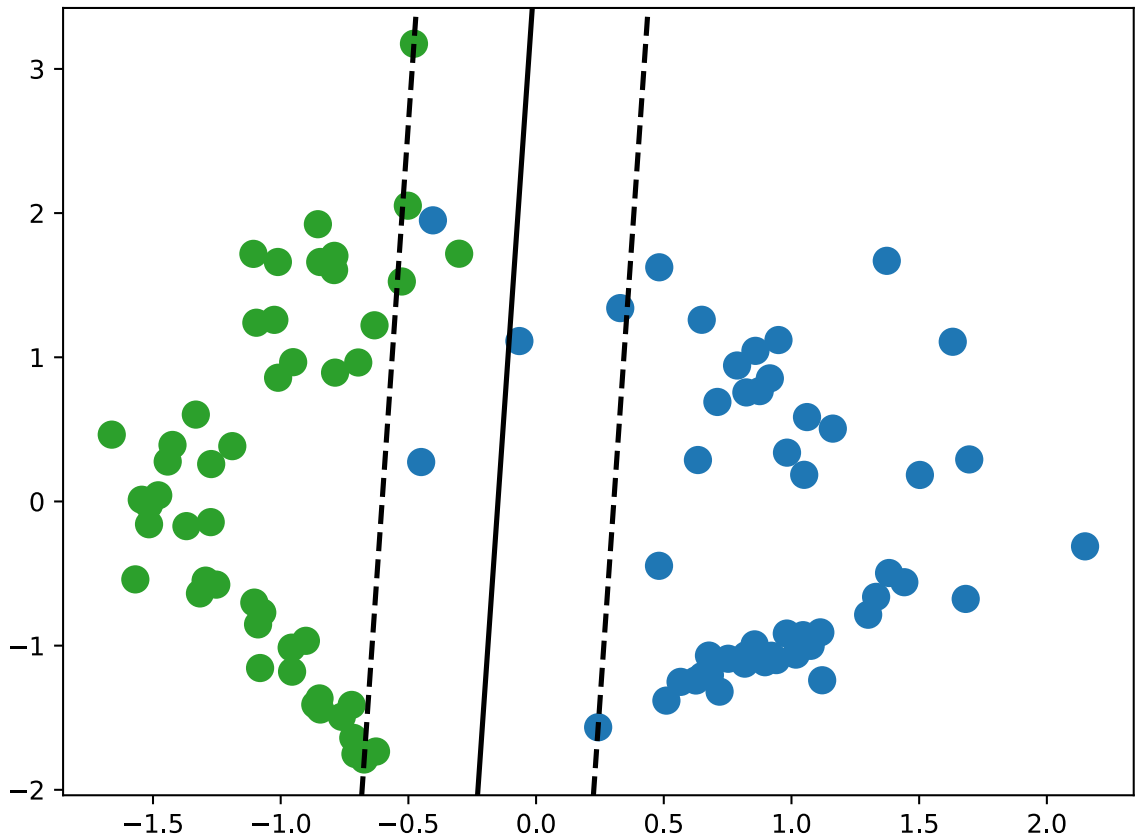
Que se passe-t-il quand il n'existe pas de séparation linéaire ?



Support Vector Machines

Principe général

Que se passe-t-il quand il n'existe pas de séparation linéaire ?

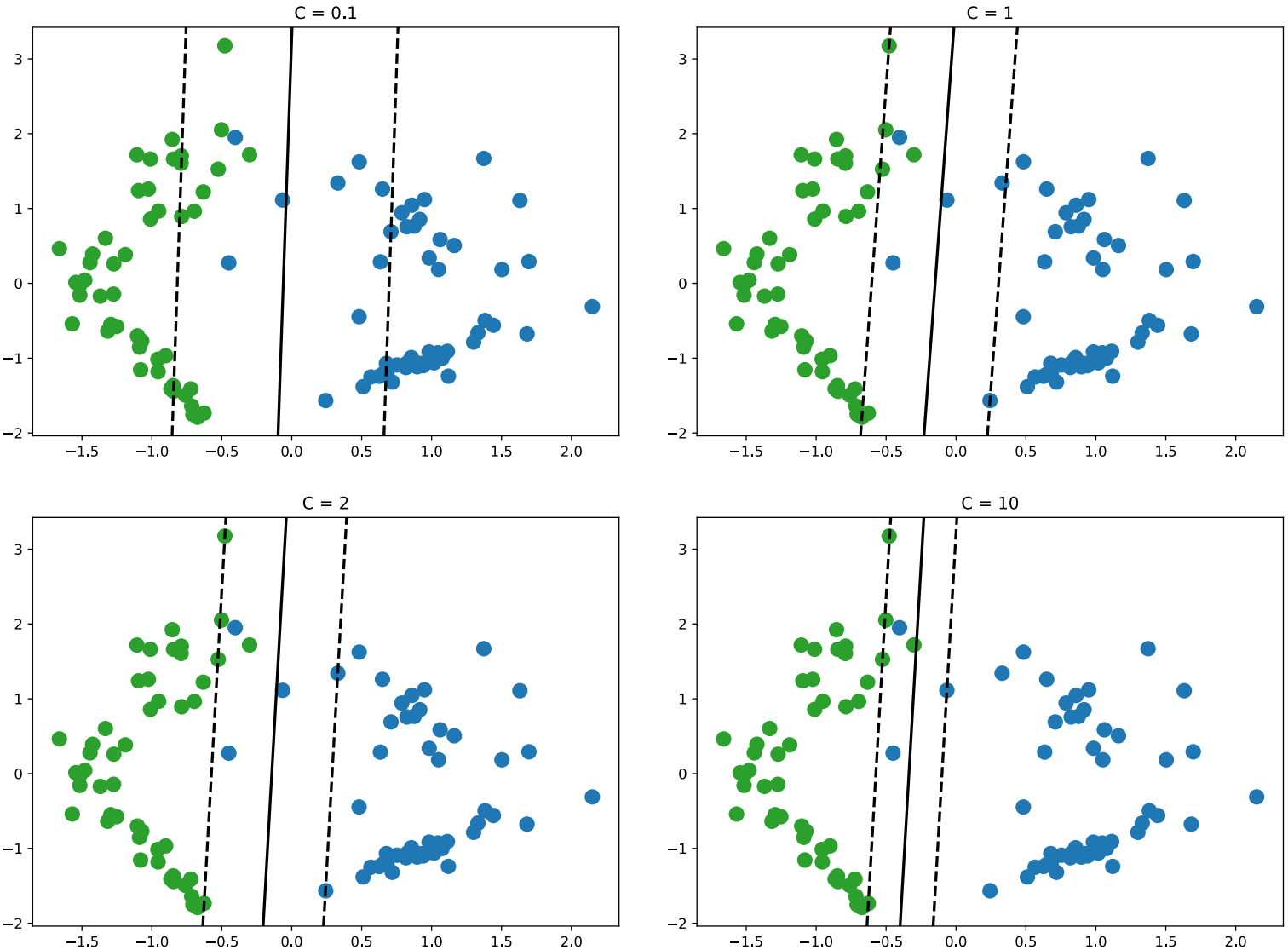


- $\Rightarrow$ Nous autorisons des erreurs !

Support Vector Machines

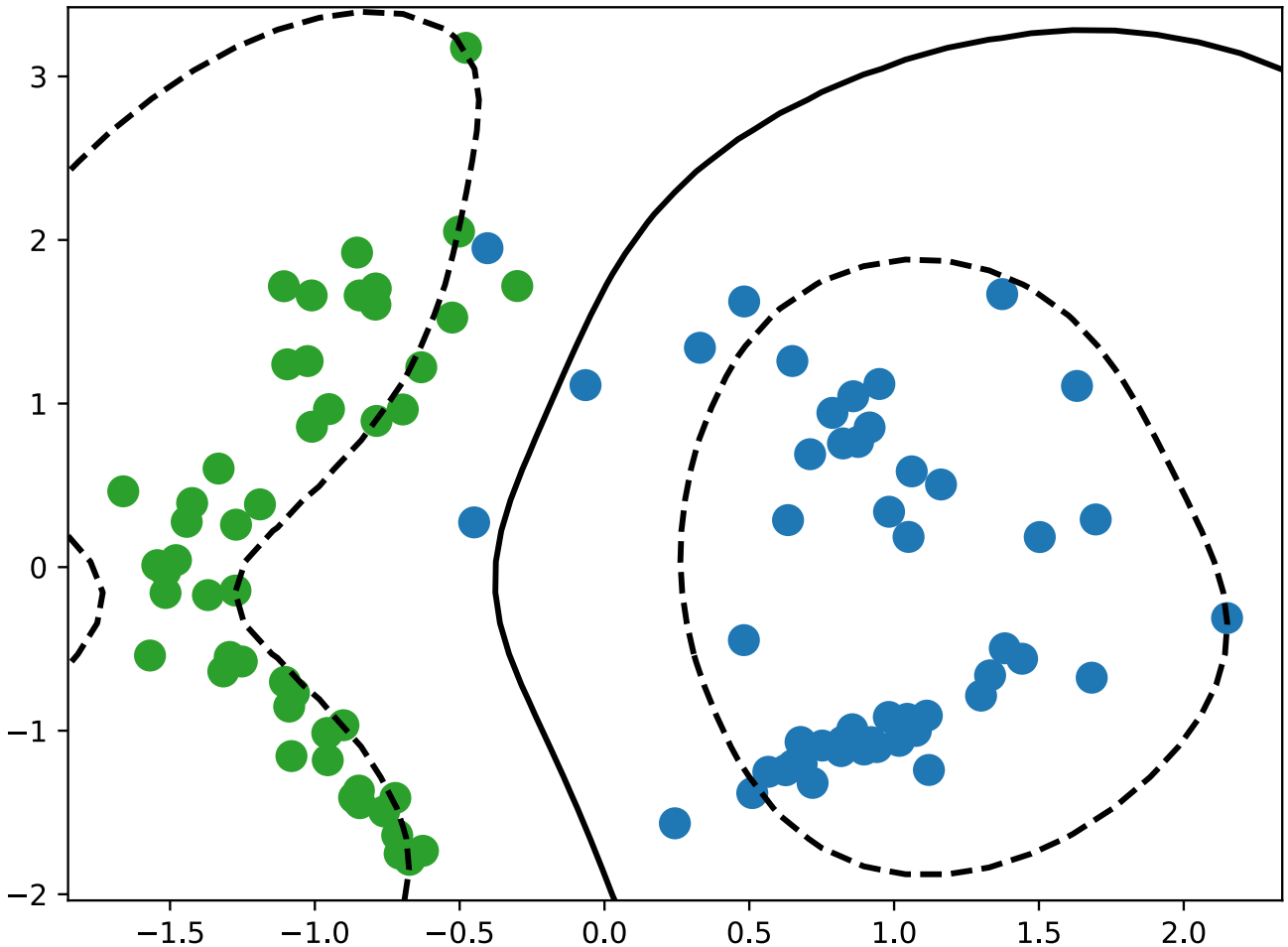
Contrôle des erreurs

C contrôle l'équilibre entre les erreurs et la taille de la séparation



Extension à la séparation non linéaire

Grâce au *kernel trick*, SVM peut calculer toutes sortes de frontières de décision.



- Cela dépend du noyau (kernel)

Hyperparamètres du SVM

C

- Faible C : calcul rapide, séparation plus simple, plus d'erreurs
- Grand C : plus lent, moins d'erreurs, séparation potentiellement trop complexe

kernel

- `linear` : frontière linéaire
- `poly`, `rbf`, `sigmoid` : frontières plus complexes, `rbf` étant un choix courant
- `precomputed` : matrice de similarité fournie (plus compliqué)

SVM : le code !

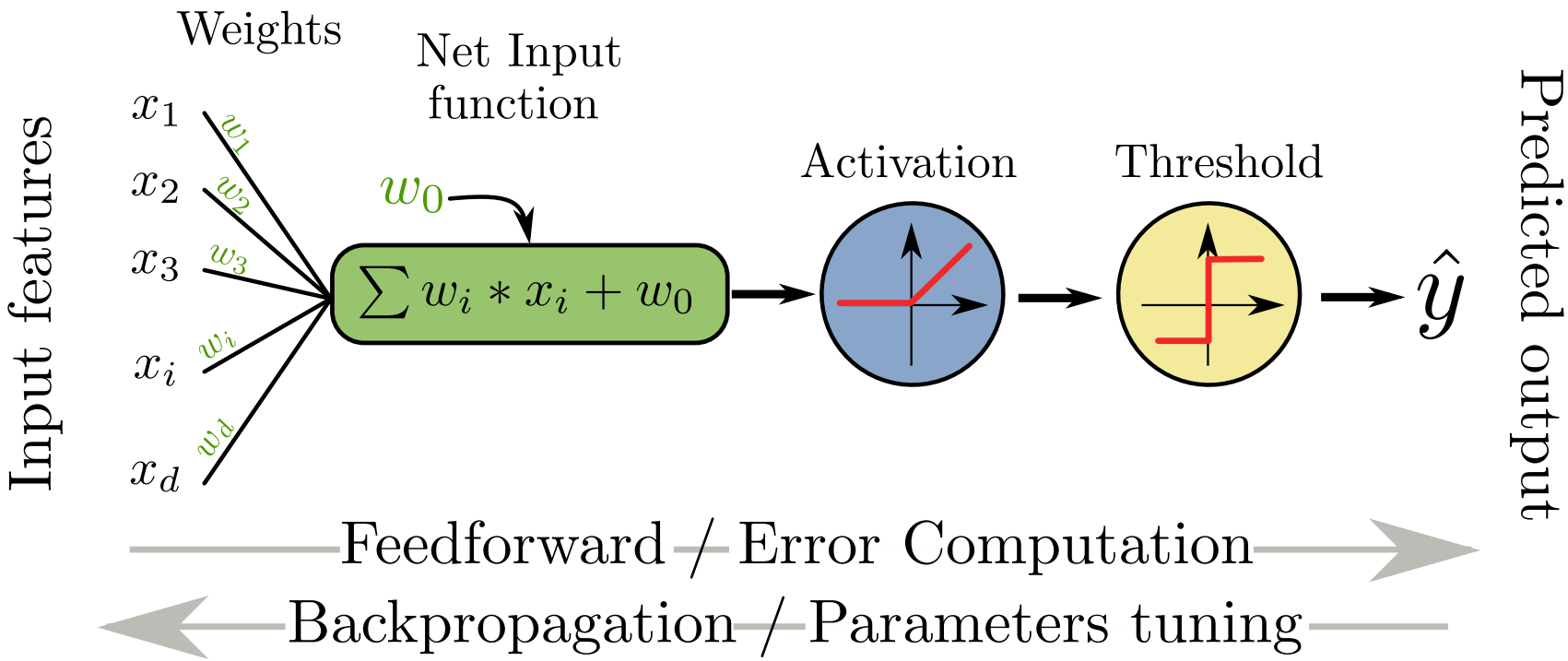
```
1    from sklearn import svm
2    C = 1
3    kernel = 'rbf'
4    clf = svm.SVC(C=C, kernel=kernel)
5    clf.fit(X, y)
6    ypred = clf.predict(X)
```

- Guide utilisateur pour les hyperparamètres : lien
- $\Rightarrow$ Notebook
- la documentation

Multi Layer Perceptron

Principe

Apprendre la meilleure représentation des données



- Les poids $\boldsymbol{w}$ sont optimisés par descente de gradient
- Enchaînement de couches

Hyperparamètres MLP

Couches cachées (hidden layers)

- Définissent l'architecture de votre MLP
- Nombre de couches : beaucoup de couches $\Rightarrow$ réseau profond
- Nombre de neurones par couche : plus de neurones $\Rightarrow$ réseau large

Plus il y a de neurones, plus le modèle est complexe.

Fonction d'activation

- `identity` : modèle linéaire
- `tanh`, `relu`, `logistic` : non linéaires (ReLU est un choix très populaire)

MLP : le code !

```
1  from sklearn.neural_network import MLPClassifier
2  activation = 'relu' # par défaut
3  layers = [32,64,128,64,32] # 5 couches de tailles différentes
4  clf = MLPClassifier(hidden_layer_sizes=layers,max_iter=500)
5  clf.fit(X,y)
6  ypred = clf.predict(X)
```

- Guide utilisateur : lien
- $\Rightarrow$ Notebook
- la documentation

Pratique

Comment apprendre un ≪ bon ≫ modèle ?

- Bonne performance
- Le plus simple possible
- Capable de prédire des données non vues

Risque empirique

Erreur sur l'ensemble d'apprentissage

- Risque empirique :

$$R_{\text{emp}}(f) \;=\; \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}\big(f(\mathbf{x}_i), y_i\big)$$

- $\mathcal{L}$ évalue la performance de prédiction $f(\mathbf{x}_i)$
- L'erreur est calculée sur l'ensemble d'apprentissage
- Le modèle peut être trop spécialisé à ce jeu de données

Généralisation

Tentative de définition

- Capacité du modèle à bien prédire sur des données non vues
- Difficile à évaluer
- Objectif réel d'un modèle

Régularisation

- Contrôle la complexité du modèle
- Équilibre entre le risque empirique et la capacité de généralisation
- Besoin de régler ce compromis (λ)

Comment évaluer la capacité de généralisation ?

Évaluer sur des données non vues

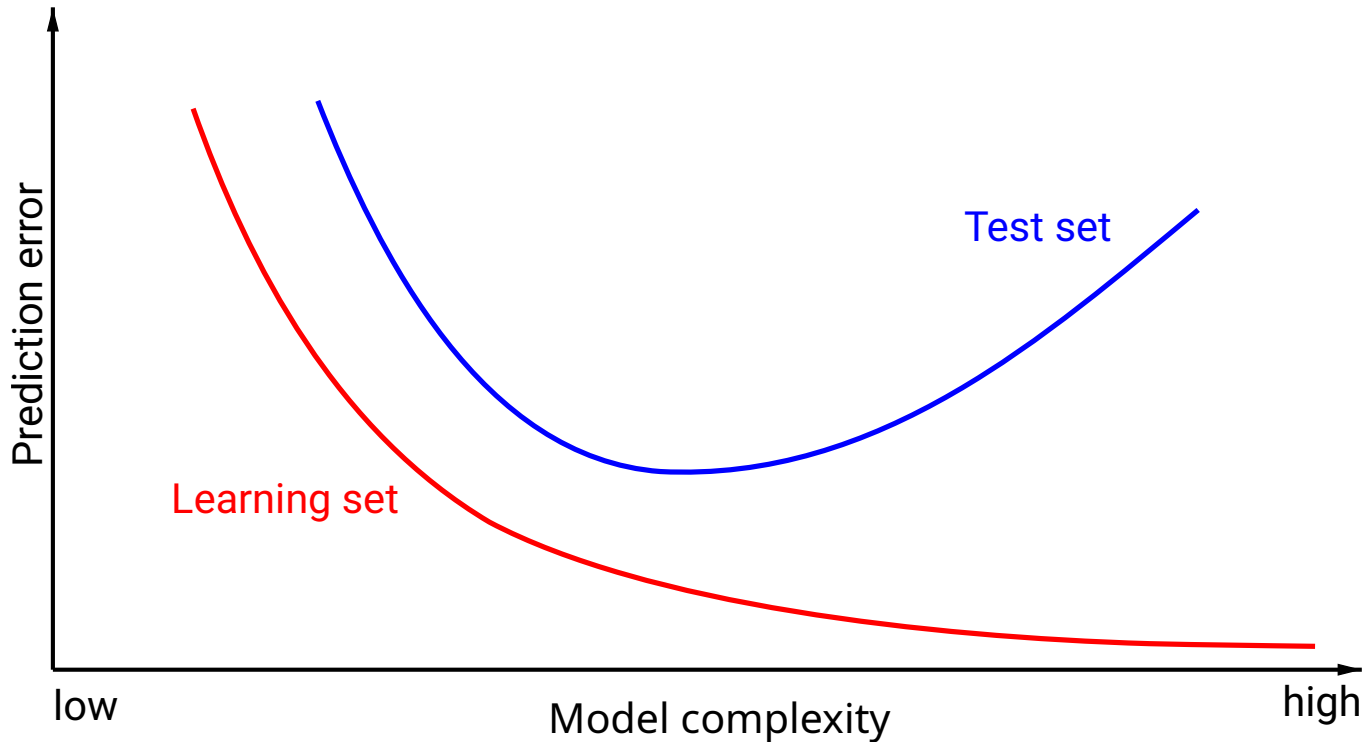
- Définir et isoler un ensemble de test
- Évaluer sur l'ensemble de test

Biais

- Éviter d'utiliser les mêmes données pour l'entraînement et le test
- L'ensemble de test doit être complètement isolé

Surapprentissage vs Sous-apprentissage

- Surapprentissage (overfitting) : R_{emp} bas, erreur de généralisation élevée
- Sous-apprentissage (underfitting) : R_{emp} élevé, erreur de généralisation moyenne



Hyperparamètres

Paramètres hors du modèle

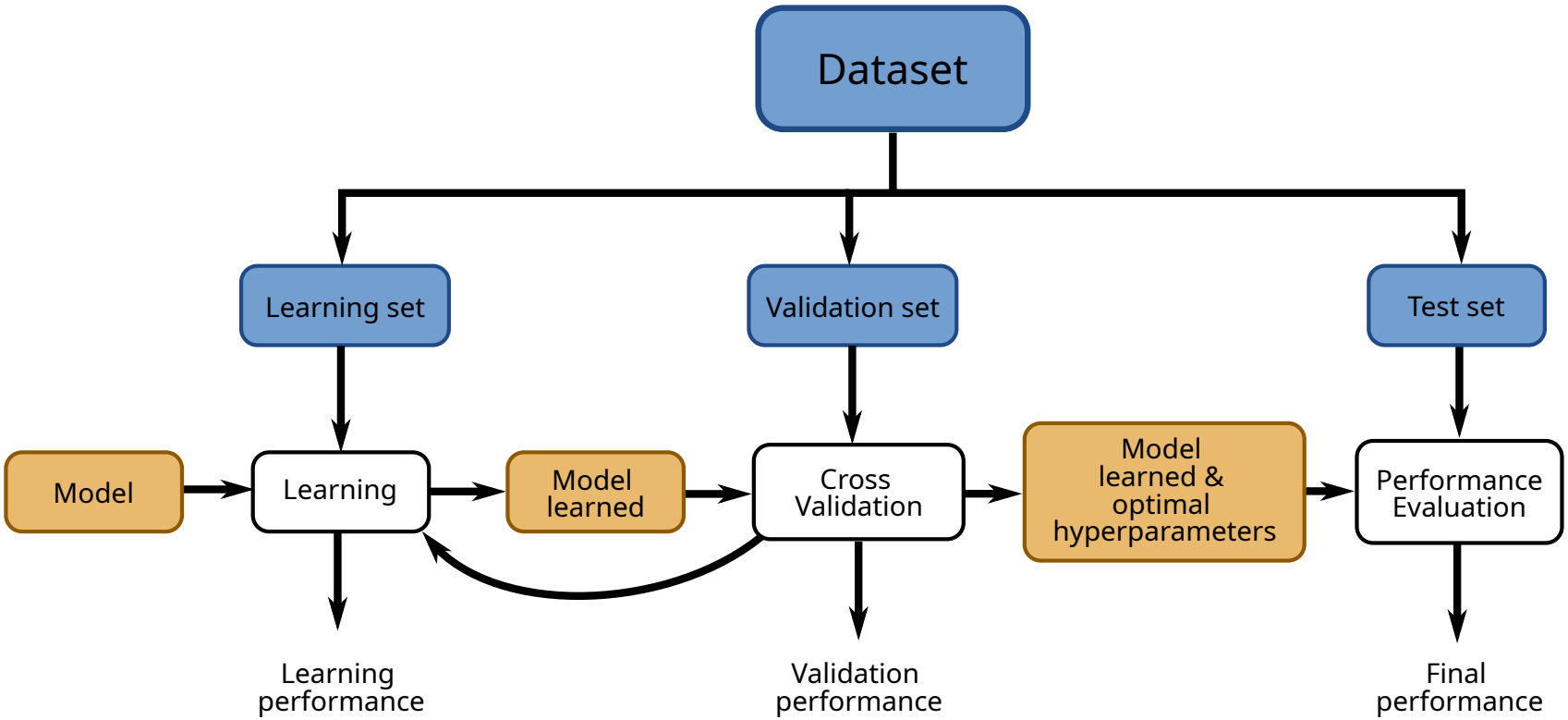
- Certains paramètres ne sont pas appris
- Ce sont des hyperparamètres, qui doivent être ajustés
- **À ne pas** ajuster sur l'ensemble de test
- Exemple : λ dans la régression Ridge

Comment ajuster les hyperparamètres ?

Validation set

- Diviser l'ensemble d'apprentissage en *apprentissage* et *validation*
- Apprendre les paramètres du modèle sur l'ensemble d'apprentissage
- Évaluer la performance sur l'ensemble de validation
- L'ensemble de validation simule des données non vues

Cadre général



Pseudo-code de validation croisée

```
1:  function EVAL(X, y, model, h_params)
2:      best_perf ← 0
3:      X_train, y_train, X_test, y_test ← split(X, y)
4:      for h_param ∈ h_params do
5:          perf ← 0
6:          for i ∈ 1...10 do
7:              X_t, y_t, X_v, y_v ← cv_split(X_train, y_train, i)
8:              fitted_model ← train(model, X_t, y_t, h_param)
9:              y_pred ← predict(fitted_model, X_v)
10:             perf ← perf + score(y_pred, y_v)
11:         end for
12:         if perf < best_perf then
13:             best_perf ← perf
14:             best_param ← h_param
15:         end if
16:     end for
17:     fitted_model ← train(model, X_train, y_train, best_param)
18:     y_pred ← predict(fitted_model, X_test)
19:     return score(y_pred, y_test)
20: end function
```

Toujours étudié dans les conférences de haut niveau

F. Errica, M. Podda, D. Bacciu, and A. Micheli, 'A Fair Comparison of Graph Neural Networks for Graph Classification'. ICLR 2020.

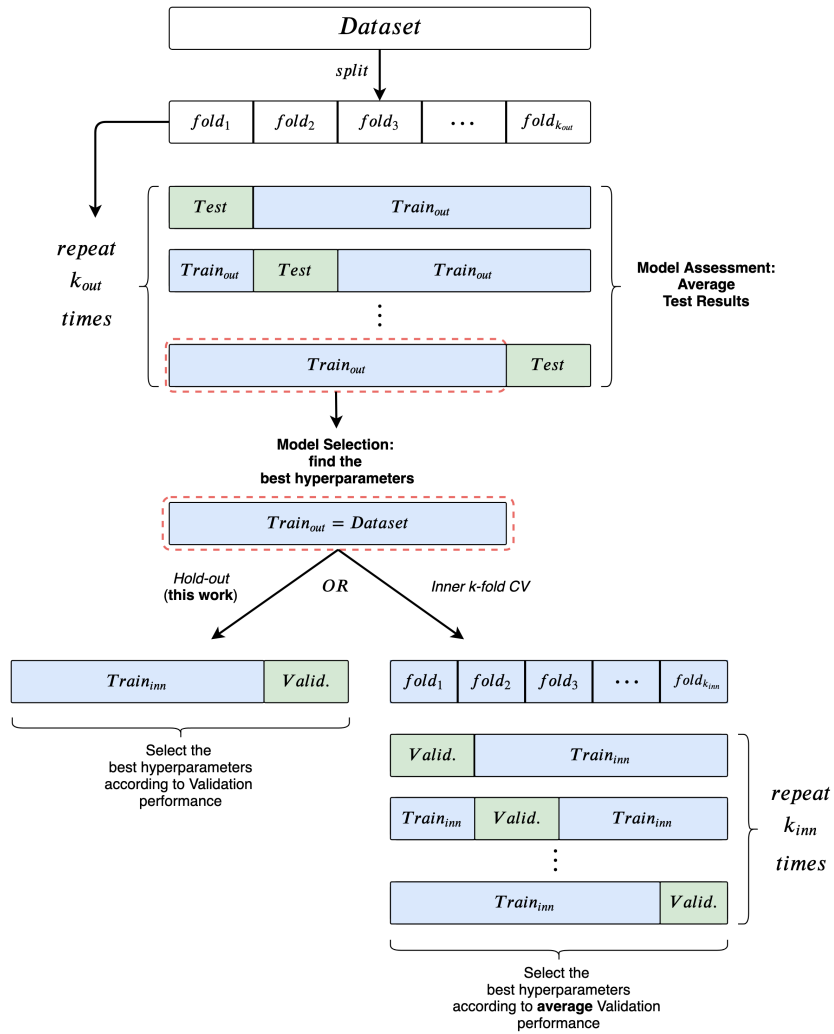
http://arxiv.org/abs/1912.09893

Algorithm 1 Model Assessment (k-fold CV)

Input: Dataset $\mathcal{D}$, set of configurations Θ
Split $\mathcal{D}$ into k folds $F_1, \ldots, F_k$
for $i \leftarrow 1, \ldots, k$ **do**
    $\text{train}_k, \text{test}_k \leftarrow \left(\bigcup_{j \neq i} F_j\right), F_i$
    $\text{best}_k \leftarrow \text{Select}(\text{train}_k, \Theta)$
    for $r \leftarrow 1, \ldots, R$ **do**
        $\text{model}_r \leftarrow \text{Train}(\text{train}_k, \text{best}_k)$
        $\text{p}_r \leftarrow \text{Eval}(\text{model}_k, \text{test}_k)$
    end for
    $\text{perf}_k \leftarrow \sum_{r=1}^{R} \text{p}_r / R$
end for
return $\sum_{i=1}^{k} \text{perf}_i / k$

Algorithm 2 Model Selection

Input: train_k, Θ
Split train_k into *train* and *valid*
$\text{p}_\theta = \emptyset$
for each $\theta \in \Theta$ **do**
    $\text{model} \leftarrow \text{Train}(\text{train}_k, \theta)$
    $\text{p}_\theta \leftarrow \text{p}_\theta \cup \text{Eval}(\text{model}, \textit{valid})$
end for
$\text{best}_\theta \leftarrow \text{argmax}_\theta \text{p}_\theta$
return best_θ



Stratégies de validation

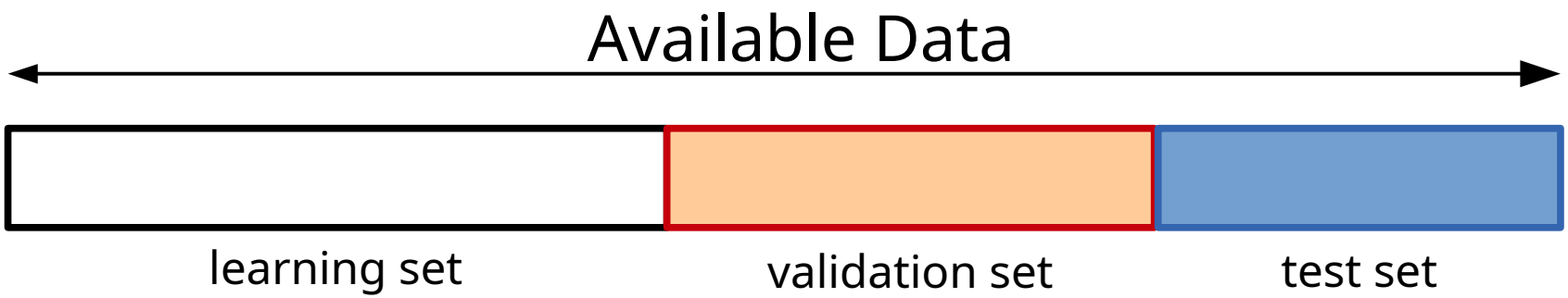
Comment séparer apprentissage/validation ?

- Besoin d'une méthode pour séparer l'ensemble d'apprentissage en « apprentissage » et « validation »
- L'ensemble « apprentissage » sert à ajuster les paramètres
- L'ensemble « validation » sert à évaluer le modèle selon les hyperparamètres

Apprentissage/Validation/Test

Découpage unique (Single split)

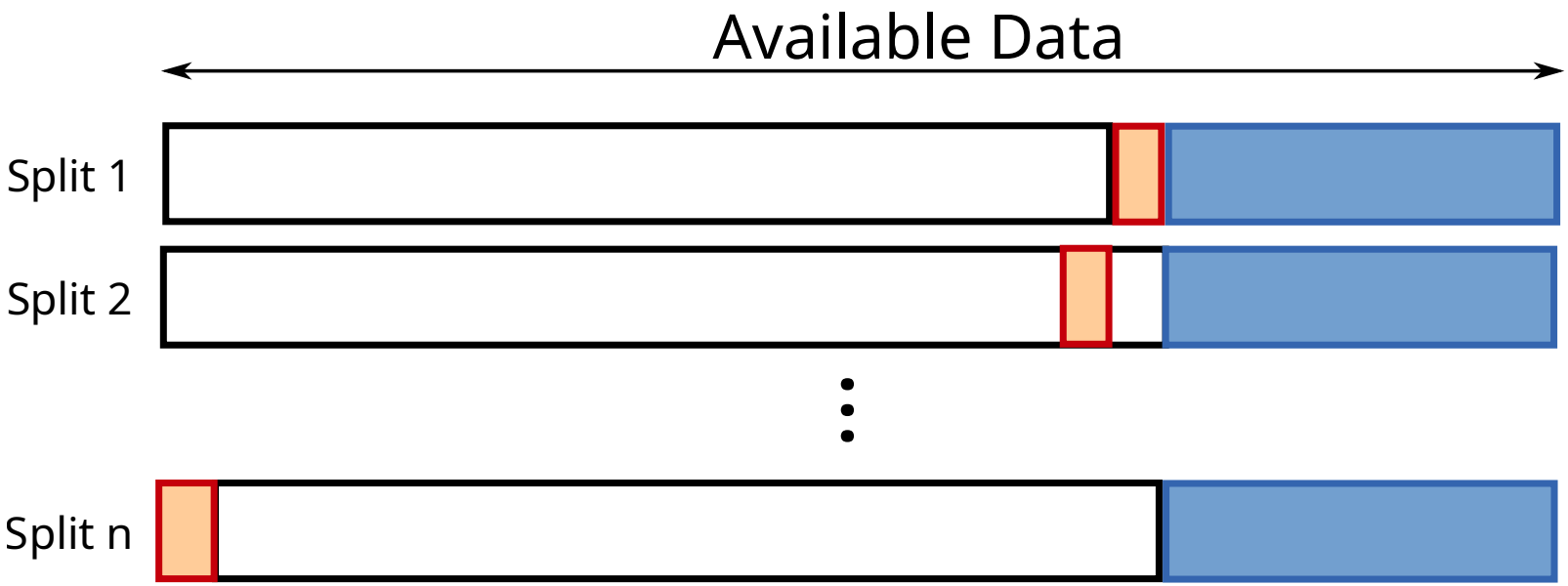
+ Un seul modèle à apprendre
– Risque de biais de découpage
– Une seule évaluation de performance



Leave one out

N splits

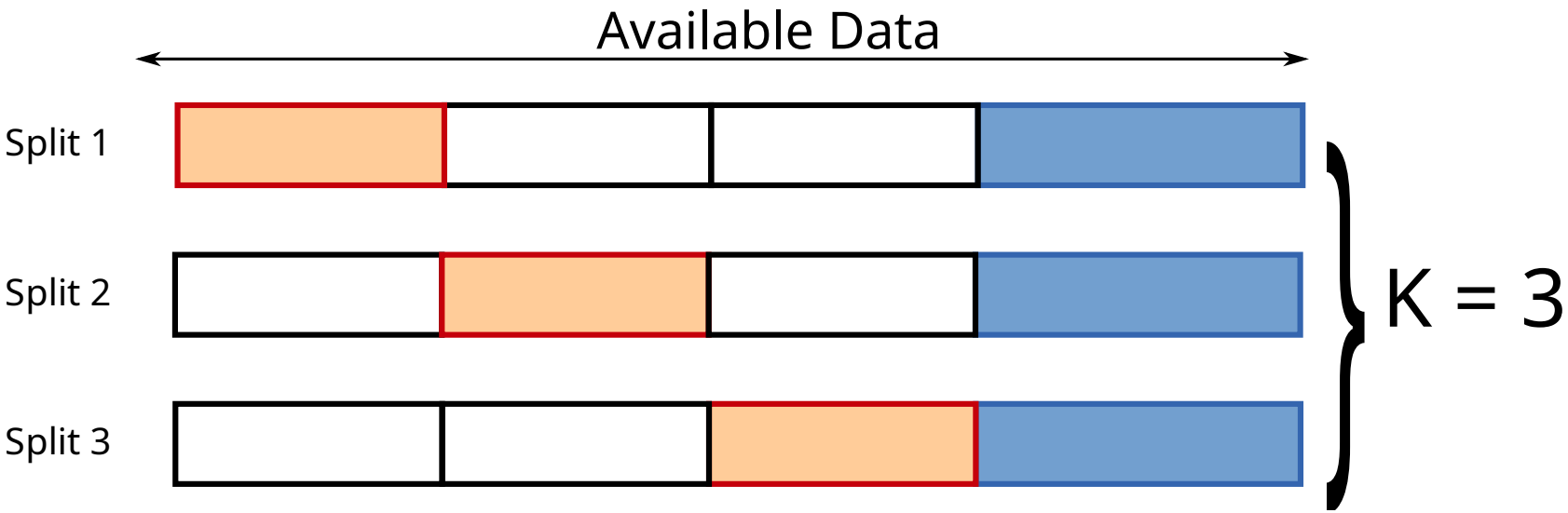
- N modèles à entraîner
- L'erreur de validation est évaluée sur 1 donnée à chaque fois



Validation croisée KFold

K splits

- \+ K modèles à entraîner
- ▶ L'erreur de validation est évaluée sur N/K données
- – Certains splits peuvent être biaisés



Shuffle Split Cross Validation

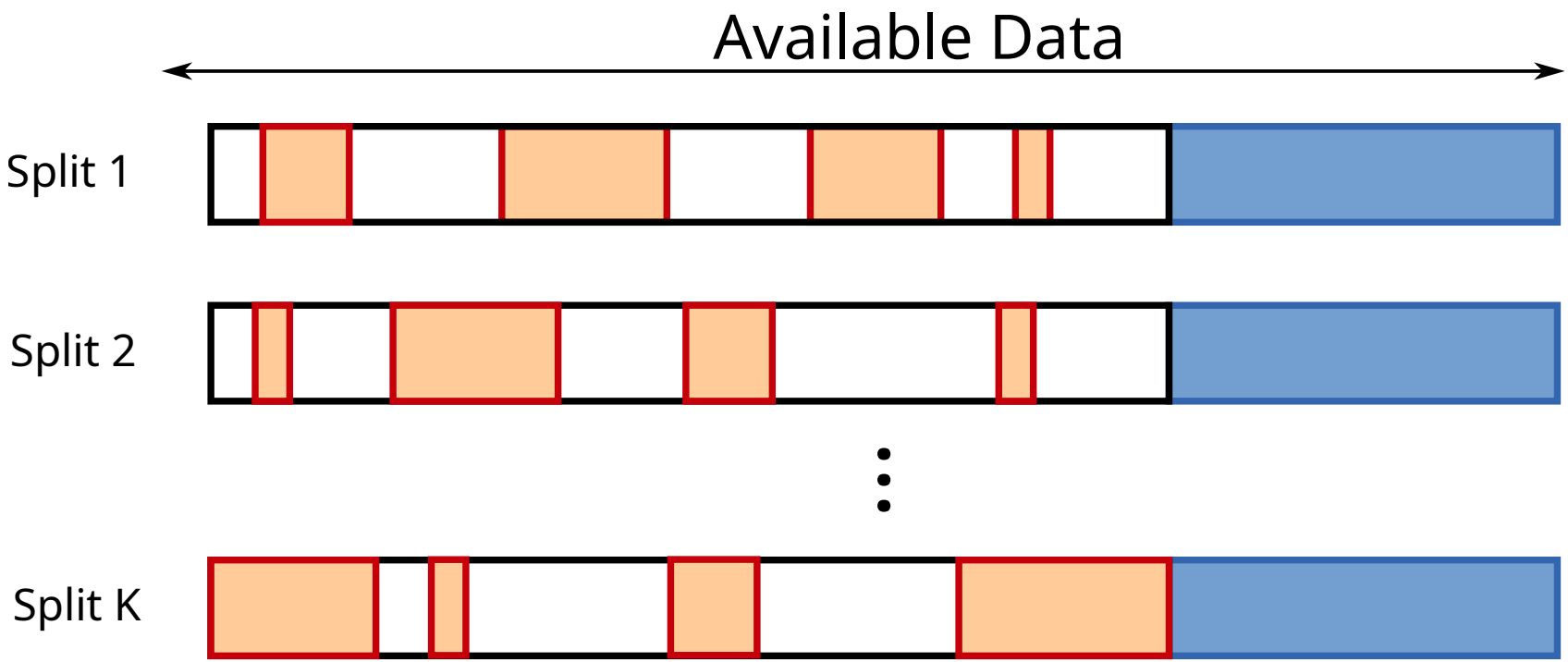
K splits

- Les ensembles Apprentissage/Validation sont choisis aléatoirement

\+ K modèles à entraîner

\+ Limite le biais

– Certaines données peuvent ne jamais être évaluées



Avec scikit-learn

- `sklearn.model_selection.train_test_split`
- `sklearn.model_selection.KFold`
- `sklearn.model_selection.ShuffleSplit`
- `sklearn.model_selection.GridSearchCV`

Recommandation

Taille des splits

- Combien de splits ?
- Combien d'exemples par split ?
- Dépend du nombre de données
- Compromis entre apprentissage et généralisation

Splits stratifiés

- Un découpage naïf peut créer des ensembles déséquilibrés
- Vérifier que la distribution de y est similaire dans chaque ensemble

Conclusion

- Un protocole correct évite les biais
- L'ensemble de test n'est jamais utilisé pour régler les (hyper)paramètres
- Il n'existe pas de protocole parfait