

IFT870/BIN710

Forage de données

Thème 4 : Fonctions prédictives

Aida Ouangraoua

Département d'informatique



Université de
Sherbrooke

Prétraitement

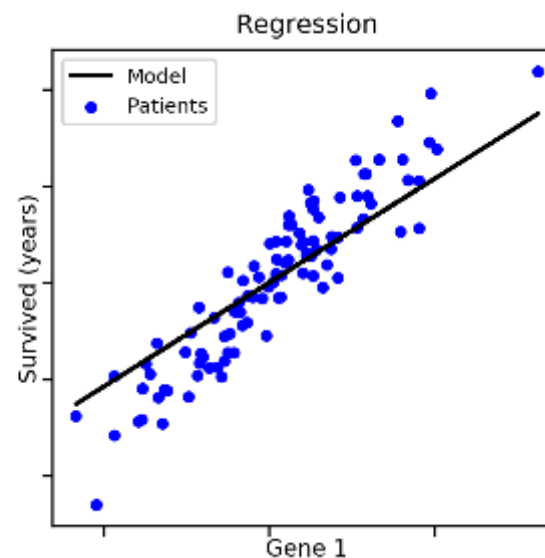
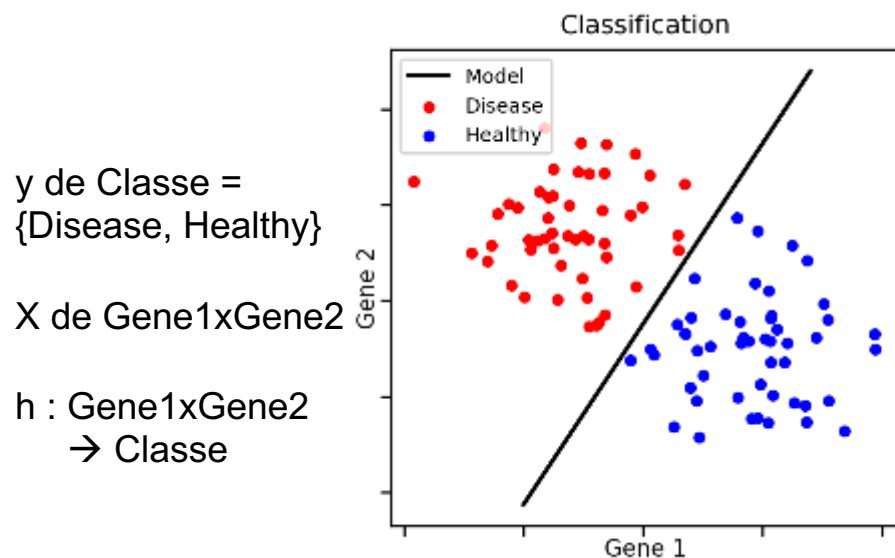
- ❑ **Rappel : fonctions prédictives**
- ❑ **Évaluation et amélioration de modèles**
- ❑ **Métriques d'évaluation de la performance**

Fonctions prédictives

□ Pour un ensemble de données observées sous la forme (X,y) où X est un vecteur de valeurs d'attributs et y la valeur d'un attribut cible correspondant, trouver un modèle h qui estime y avec précision étant donné un nouveau vecteur X , i.e. $y \cong h(X)$.

❖ **Classification** : attribut cible à valeurs discrètes (catégoriques)

❖ **Régression** : attribut cible à valeurs continues (numériques)



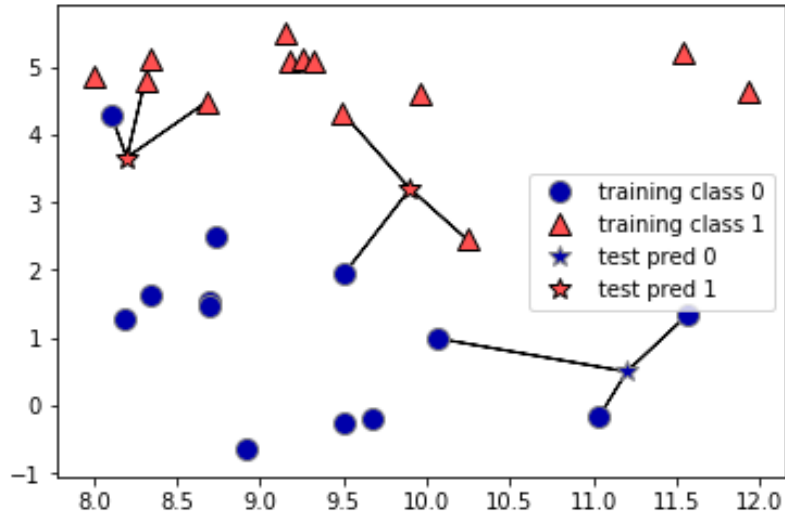
y de AnneesSurvie = $\mathbb{R}^+$

X de Gene2

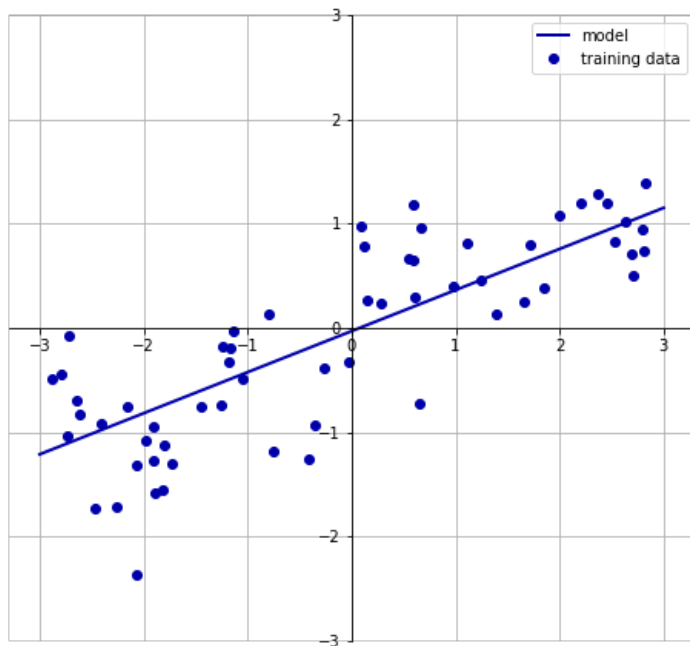
h : Gene2 → AnneesSurvie

Rappel : quelques algorithmes

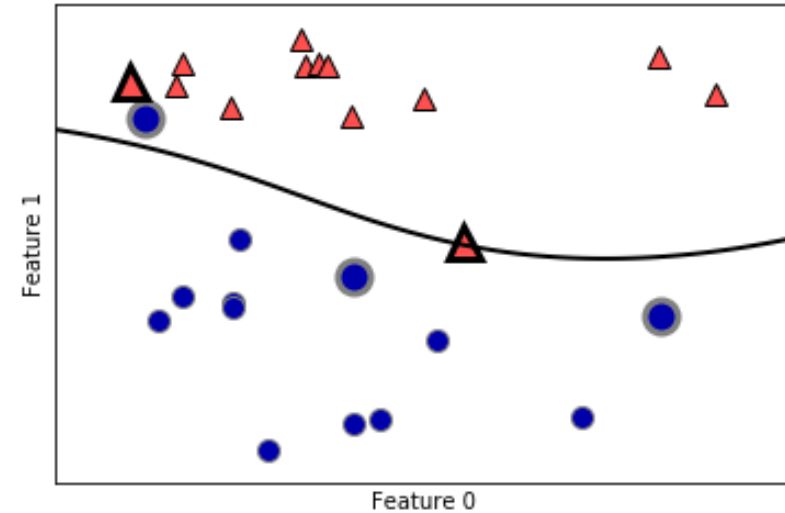
❑ K plus proches voisins



❑ Modèles linéaires



❑ Machines à vecteur de support

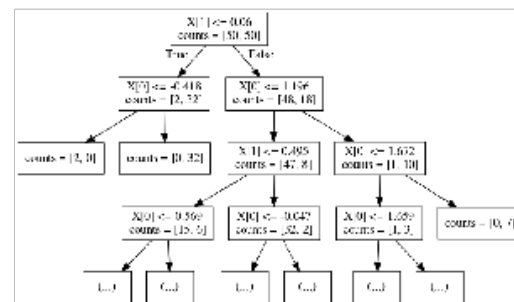
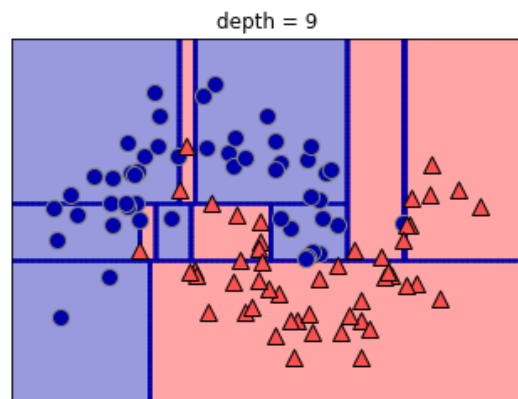


❑ Classificateur de Bayes Naïf

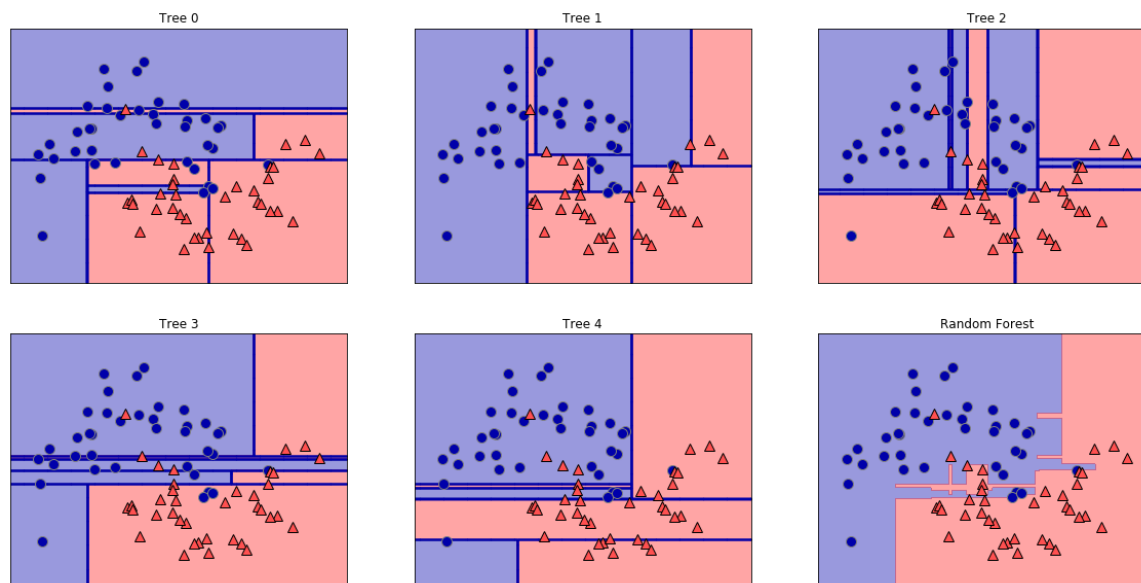
Frequency Table					Likelihood Table				
		Play Golf					Play Golf		
		Yes	No				Yes	No	
Outlook	Sunny	3	2	⇒		Outlook	Sunny	3/9	2/5
	Overcast	4	0				Overcast	4/9	0/5
	Rainy	2	3				Rainy	2/9	3/5
		Play Golf					Play Golf		
		Yes	No				Yes	No	
Humidity	High	3	4	⇒		Humidity	High	3/9	4/5
	Normal	6	1				Normal	6/9	1/5
		Play Golf					Play Golf		
		Yes	No				Yes	No	
Temp.	Hot	2	2	⇒		Temp.	Hot	2/9	2/5
	Mild	4	2				Mild	4/9	2/5
	Cool	3	1				Cool	3/9	1/5
		Play Golf					Play Golf		
		Yes	No				Yes	No	
Windy	False	6	2	⇒		Windy	False	6/9	2/5
	True	3	3				True	3/9	3/5

Rappel : quelques algorithmes

□ Arbres de décision



□ Ensemble d'arbres de décision

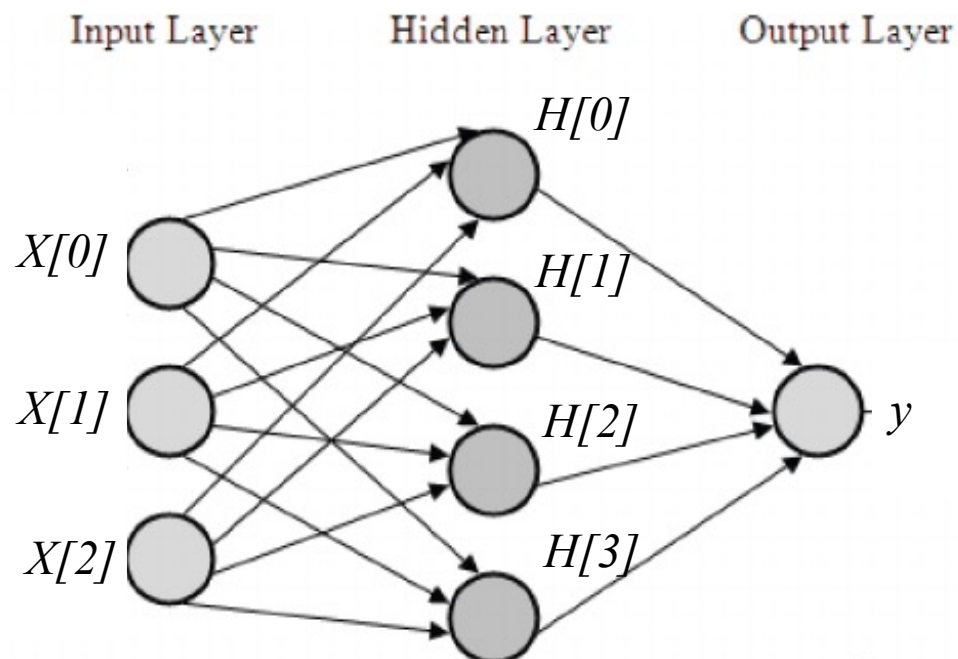


Rappel : quelques algorithmes

□ Réseaux de neurones

Application d'une fonction non-linéaire aux unités cachées:

- rectified linear unit (relu) (par défaut)
- tangens hyperbolicus (tanh)



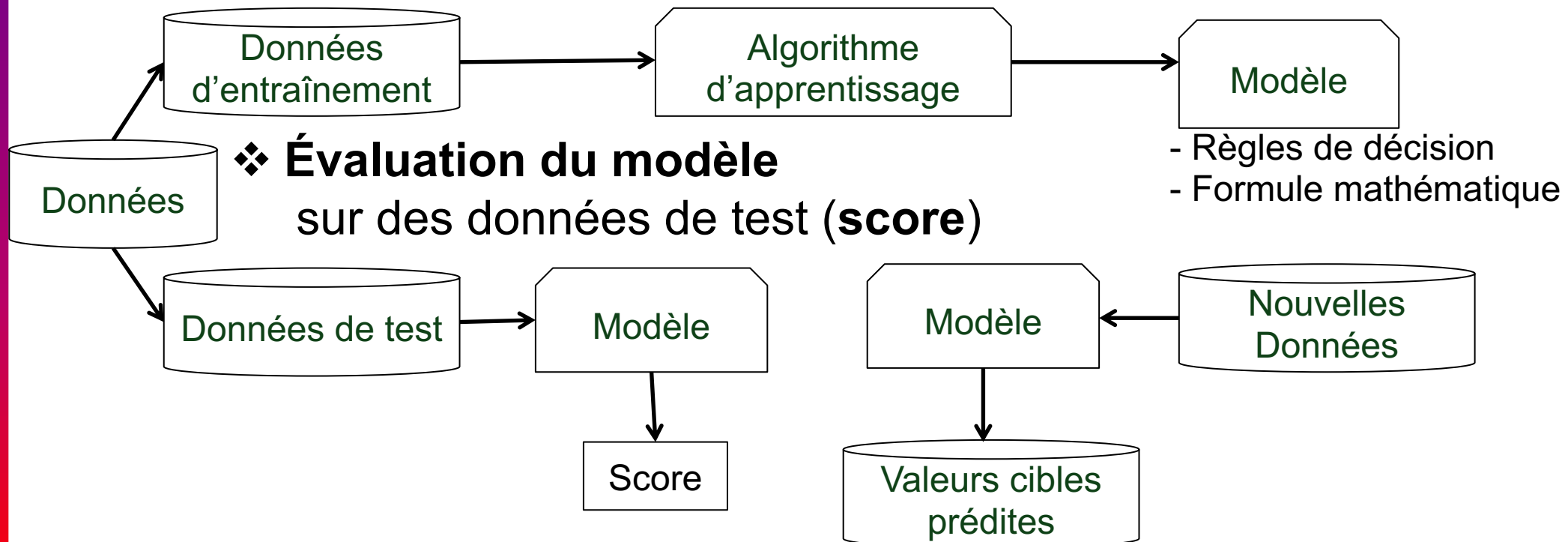
Prétraitement

- ❑ **Rappel : fonctions prédictives**
- ❑ **Évaluation et amélioration de modèles**
- ❑ **Métriques d'évaluation de la performance**

Évaluation et amélioration de modèles

- ❑ **Processus à deux étapes** après division des données en un ensemble d'entraînement et un ensemble de test (**train_test_split**)

❖ **Construction du modèle**
sur des données d'entraînement (**fit**)



- ❑ **Objectif** : mesurer la façon dont le modèle se généralise à de nouvelles données

Évaluation et amélioration de modèles

□ Évaluation de la performance de généralisation

❖ Méthode robuste pour évaluer la performance d'un modèle

- **Échantillonnage aléatoire** : répéter train_test_split k fois, et calculer la moyenne des scores obtenus
- **Bootstrap** (pour petit jeu de données de taille n)
Exemple: la méthode de « bootstrap .632 » : tirer aléatoirement 1 donnée n fois avec remise. Environ 63.2% des données sont tirés pour l'entraînement, et les 36.8% de données non-tirées sont gardés pour le test ($(1 - 1/d)^d \approx 0.368$).
Répéter la procédure d'échantillonnage k fois et calculer la moyenne des scores obtenus

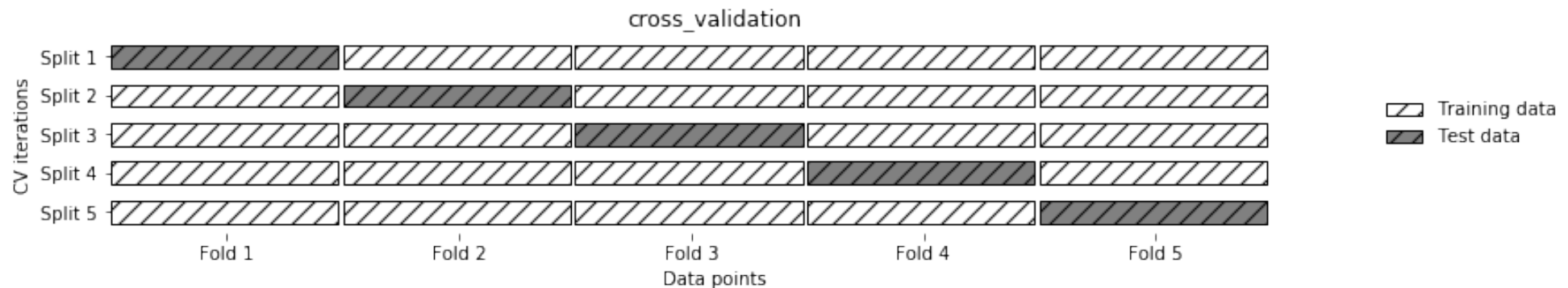
Évaluation et amélioration de modèles

❑ Évaluation de la performance de généralisation

❖ Méthode robuste pour évaluer la performance d'un modèle

- **Cross-validation**

Exemple: k-fold cross-validation (en général $k=5$ ou 10) : partitionner les données en k sous-ensembles de taille similaire (fold_1, fold_2, ... fold_k). Entraîner une séquence de modèles: à la i ème itération, utiliser fold_i comme données de test, et les autres comme données d'entraînement.



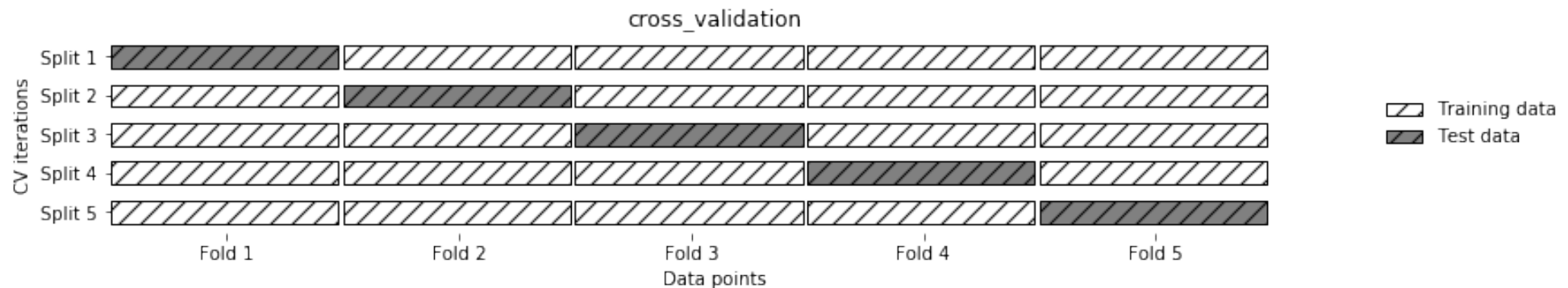
Évaluation et amélioration de modèles

❑ Évaluation de la performance de généralisation

❖ Méthode robuste pour évaluer la performance d'un modèle

- **K-fold cross-validation**

```
from sklearn.model_selection import cross_val_score  
model = Model()  
X,y = donnees.data, donnees.target  
scores = cross_val_score(model, X, y, cv=5)
```



```
scores : [0.99, 0.9, 0.95, 0.93, 1.0]
```

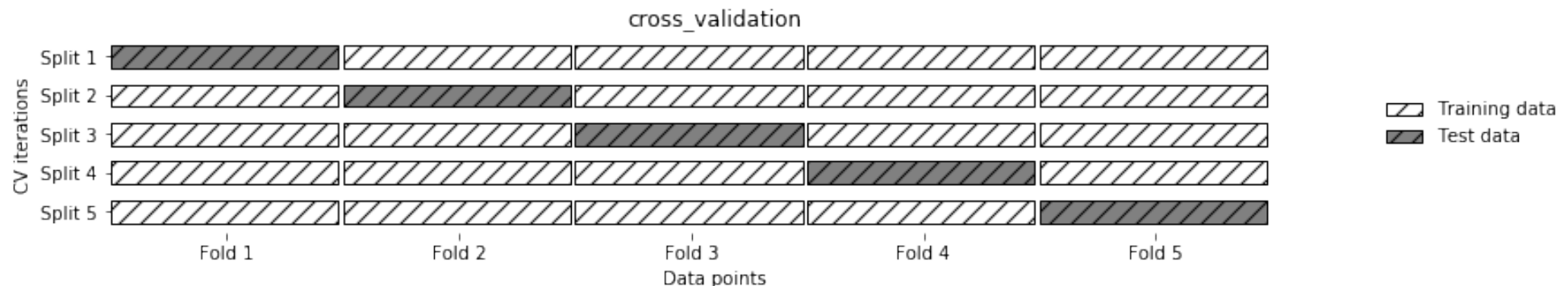
Évaluation et amélioration de modèles

❑ Évaluation de la performance de généralisation

❖ Méthode robuste pour évaluer la performance d'un modèle

- K-fold cross-validation

```
from sklearn.model_selection import cross_validate  
result = cross_validate(model, X, y, cv=5, return_train_score=True)
```



```
result : fit_time, score_time, test_score, train_score for all iterations
```

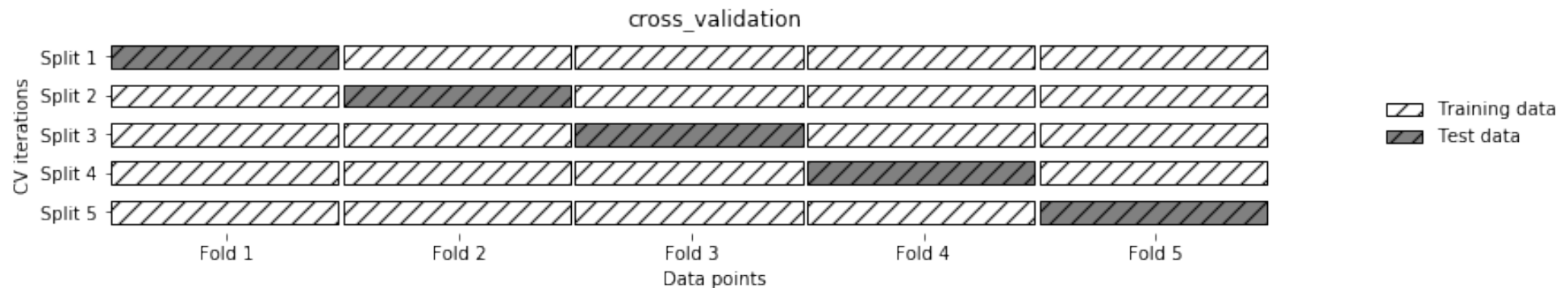
Évaluation et amélioration de modèles

□ Évaluation de la performance de généralisation

❖ Méthode robuste pour évaluer la performance d'un modèle

- **K-fold cross-validation**

- Assure que chaque donnée est utilisée une fois comme test
- Évalue la **robustesse** du modèle aux changements de données d'entraînement



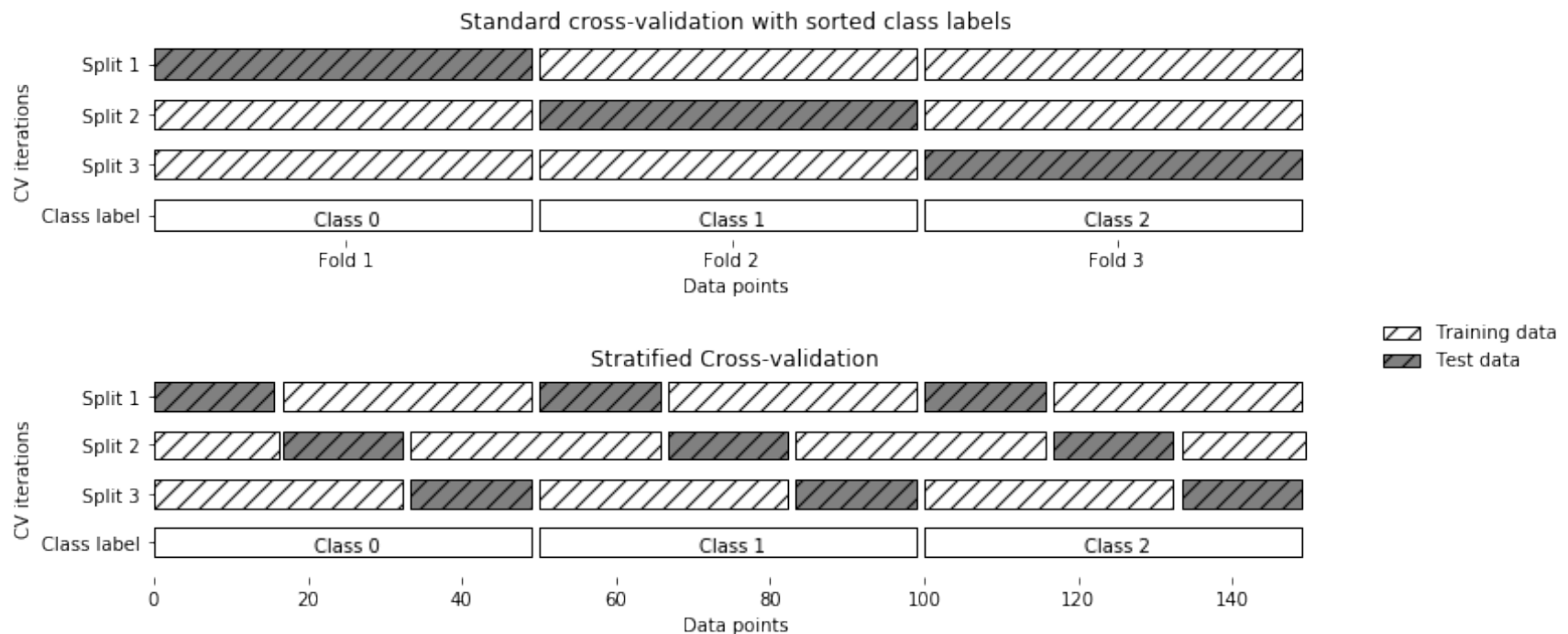
Évaluation et amélioration de modèles

❑ Évaluation de la performance de généralisation

❖ Méthode robuste pour évaluer la performance d'un modèle

- **K-fold stratifié**

Stratifier les parties de telle sorte que les distributions des classes dans les parties sont approximativement les mêmes.



Évaluation et amélioration de modèles

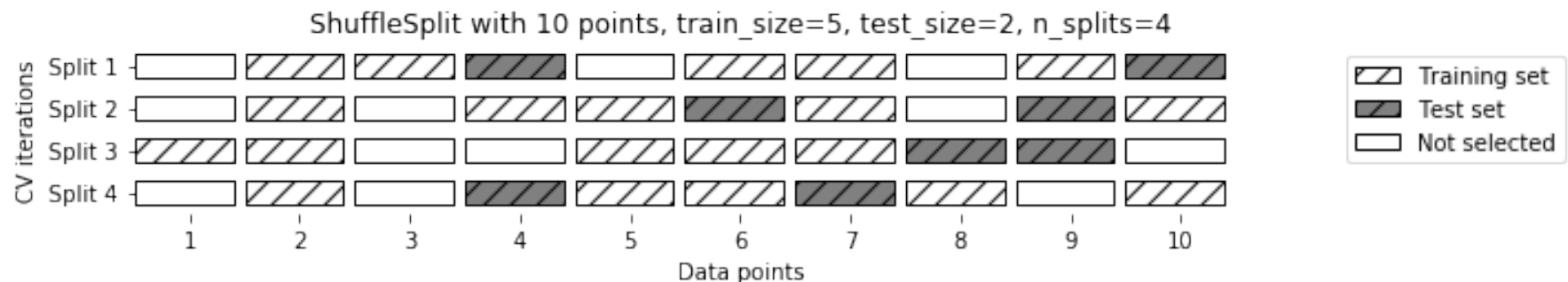
❑ Évaluation de la performance de généralisation

❖ Méthode robuste pour évaluer la performance d'un modèle

- **K-fold : spécification de la stratégie de séparation**

KFold, ShuffleSplit, GroupKFold, LeaveOneOut, StratifiedKFold, StratifiedShuffleSplit

```
from sklearn.model_selection import KFold  
kfold = KFold(n_splits=5, shuffle = True)  
score = cross_val_score(model, X, y, cv=kfold)
```



Évaluation et amélioration de modèles

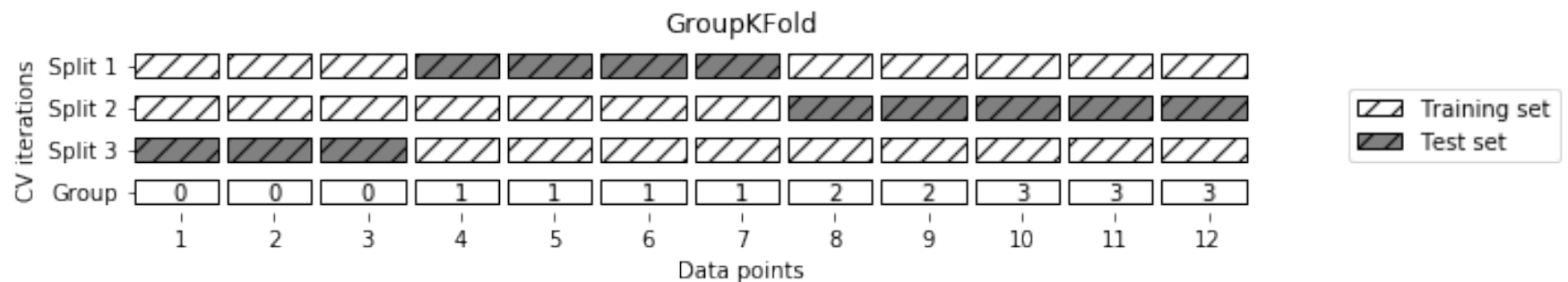
❑ Évaluation de la performance de généralisation

❖ Méthode robuste pour évaluer la performance d'un modèle

- **K-fold** : spécification de la stratégie de séparation

KFold, ShuffleSplit, GroupKFold, LeaveOneOut, StratifiedKFold, StratifiedShuffleSplit

```
from sklearn.model_selection import KFold  
kfold = KFold(n_splits=5, shuffle = True)  
score = cross_val_score(model, X, y, cv=kfold)
```



Évaluation et amélioration de modèles

□ Amélioration de la performance de généralisation

❖ Méthodes pour comparer/améliorer des modèles

- **Grid search**

Essayer des combinaisons de paramètres pour un modèle afin de choisir la meilleure combinaison

Exemple: Modèle SVC avec les paramètres γ et C

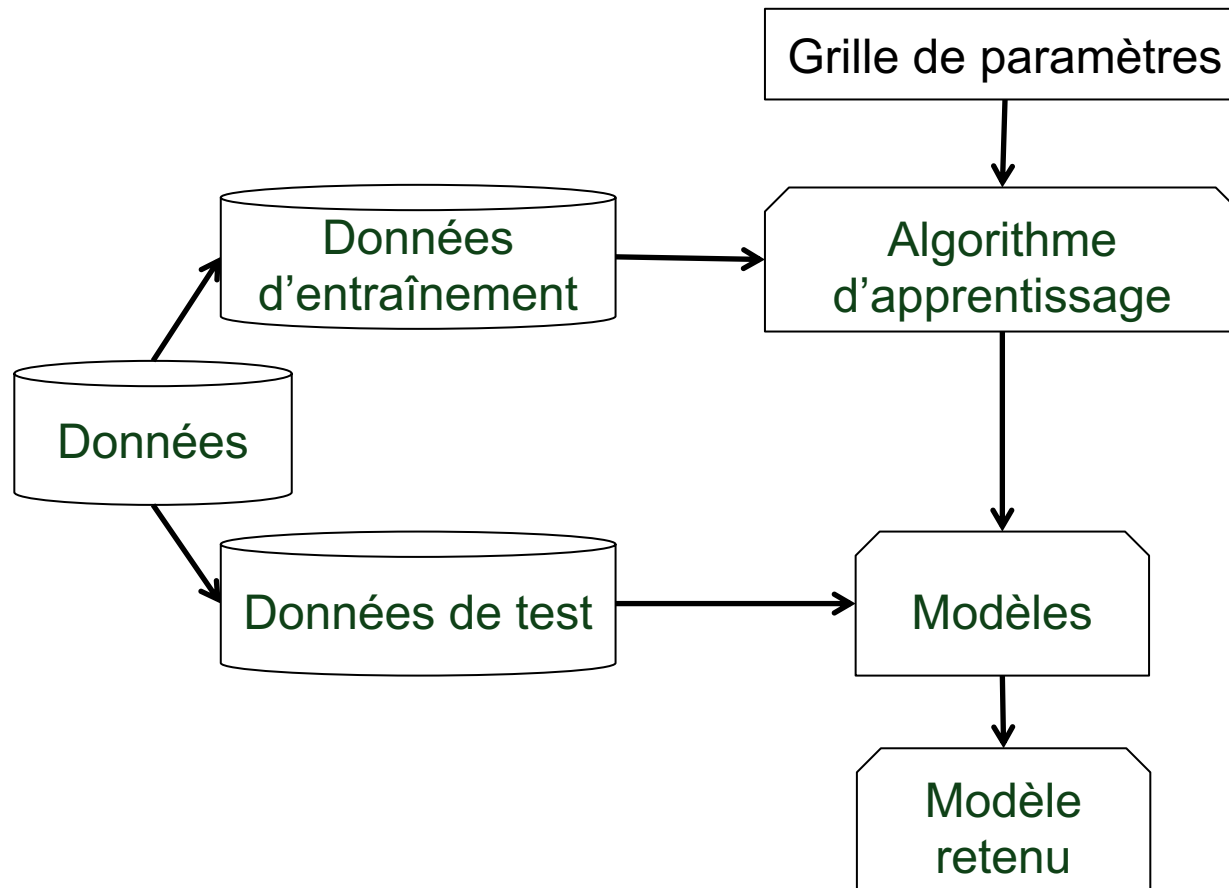
	$C=0.001$	$C=0.01$	$C=0.1$	$C=1.0$	$C=10.0$
$\gamma=0.001$	SVC($\gamma=0.001$, $C=0.001$)	SVC($\gamma=0.001$, $C=0.01$)	SVC($\gamma=0.001$, $C=0.1$)	SVC($\gamma=0.001$, $C=1.0$)	SVC($\gamma=0.001$, $C=10.0$)
$\gamma=0.01$	SVC($\gamma=0.01$, $C=0.001$)	SVC($\gamma=0.01$, $C=0.01$)	SVC($\gamma=0.01$, $C=0.1$)	SVC($\gamma=0.01$, $C=1.0$)	SVC($\gamma=0.01$, $C=10.0$)
$\gamma=0.1$	SVC($\gamma=0.1$, $C=0.001$)	SVC($\gamma=0.1$, $C=0.01$)	SVC($\gamma=0.1$, $C=0.1$)	SVC($\gamma=0.1$, $C=1.0$)	SVC($\gamma=0.1$, $C=10.0$)
$\gamma=1.0$	SVC($\gamma=1$, $C=0.001$)	SVC($\gamma=1$, $C=0.01$)	SVC($\gamma=1$, $C=0.1$)	SVC($\gamma=1$, $C=1.0$)	SVC($\gamma=1$, $C=10.0$)
$\gamma=10.0$	SVC($\gamma=10$, $C=0.001$)	SVC($\gamma=10$, $C=0.01$)	SVC($\gamma=10$, $C=0.1$)	SVC($\gamma=10$, $C=1.0$)	SVC($\gamma=10$, $C=10.0$)

Évaluation et amélioration de modèles

❑ Amélioration de la performance de généralisation

❖ Méthodes pour comparer/améliorer des modèles

- **Grid search** : approche naïve



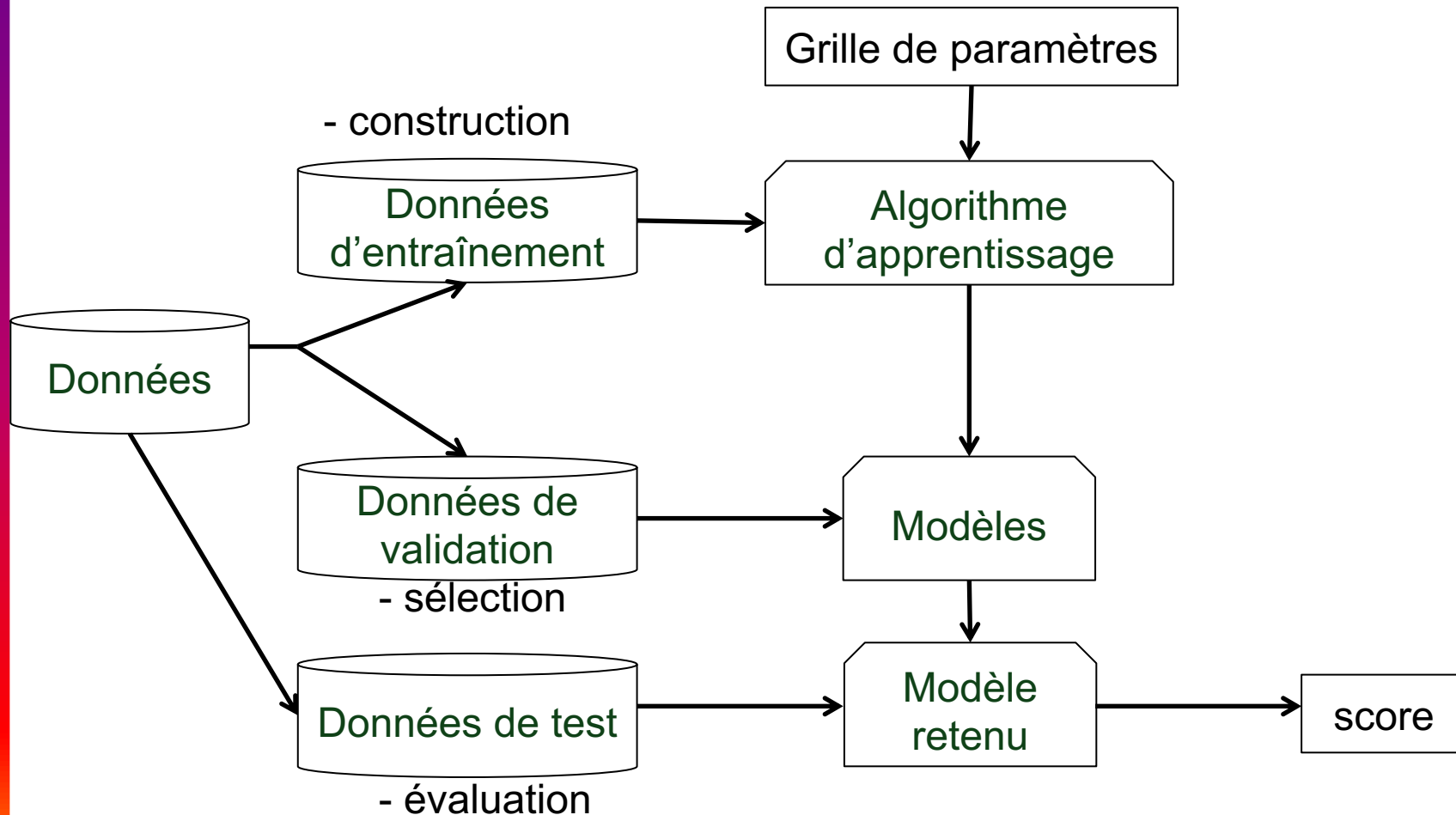
Possibilité de sur-apprentissage !

Évaluation et amélioration de modèles

❑ Amélioration de la performance de généralisation

❖ Méthodes pour comparer/améliorer des modèles

- **Grid search** : nécessaire de partitionner les données en 3

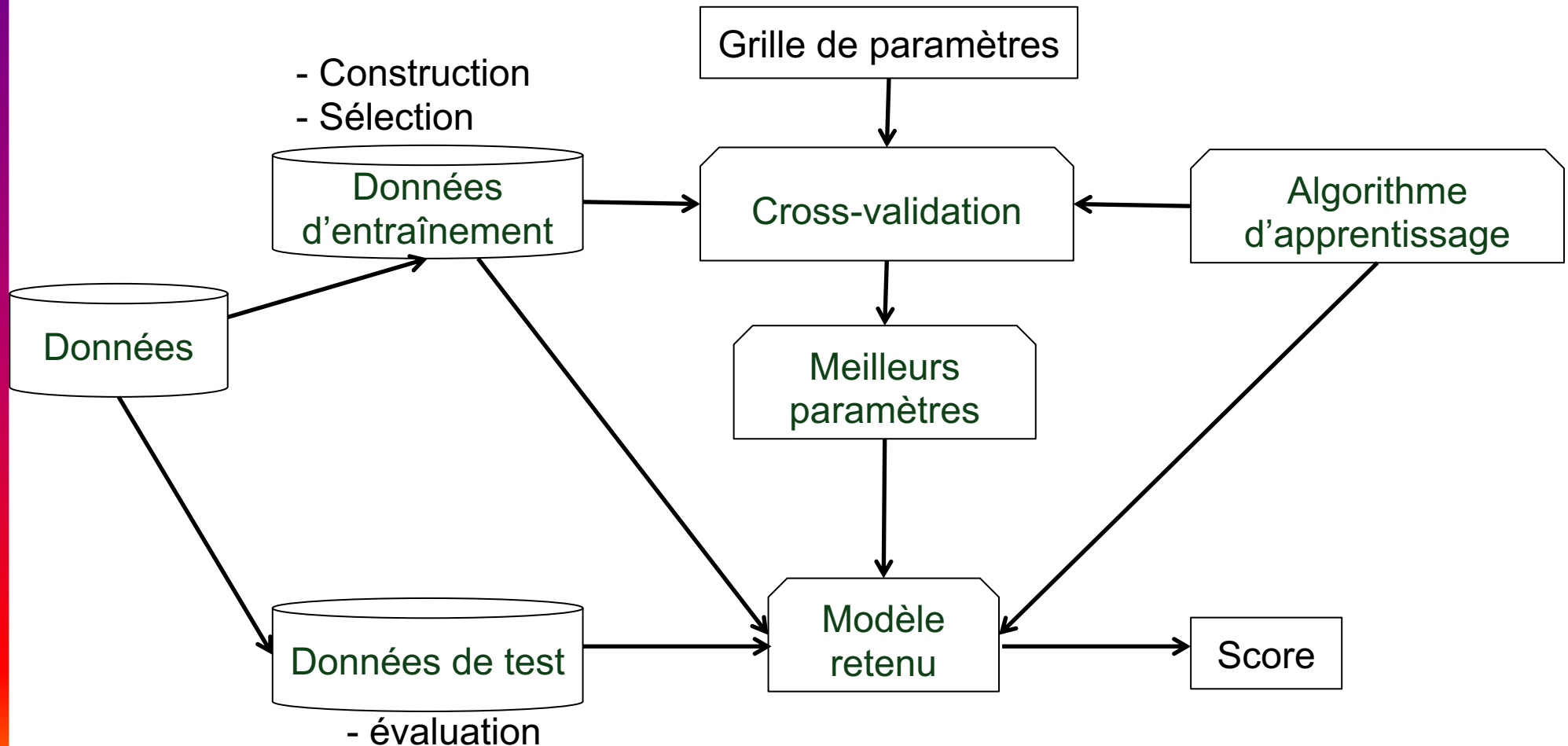


Évaluation et amélioration de modèles

❑ Amélioration de la performance de généralisation

❖ Méthodes pour comparer/améliorer des modèles

- Grid search avec validation croisée



Évaluation et amélioration de modèles

❑ Amélioration de la performance de généralisation

❖ Méthodes pour comparer/améliorer des modèles

- Grid search avec validation croisée

```
for gamma in [0.001, 0.01, 0.1, 1, 10]:  
    for C in [0.001, 0.01, 0.1, 1, 10]:  
        svm = SVC(gamma=gamma, C=C)  
        scores = cross_val_score(svm, X_train, y_train, cv=5)  
        score = np.mean(scores)  
        if score > best_score:  
            best_score = score  
            best_parameters = {'C': C, 'gamma': gamma}  
svm = SVC(**best_parameters)  
svm.fit(X_train, y_train)
```

Évaluation et amélioration de modèles

❑ Amélioration de la performance de généralisation

❖ Méthodes pour comparer/améliorer des modèles

- Grid search avec validation croisée

```
grille_param = {'C': [0.001, 0.01, 0.1, 1, 10, 100],  
                'gamma': [0.001, 0.01, 0.1, 1, 10, 100]}  
  
from sklearn.model_selection import GridSearchCV  
from sklearn.svm import SVC  
  
grid_search = GridSearchCV(SVC(), grille_param, cv=5,  
                           return_train_score=True)  
  
grid_search.fit(X_train, y_train)
```

Évaluation et amélioration de modèles

□ Amélioration de la performance de généralisation

❖ Méthodes pour comparer/améliorer des modèles

- Grid search avec plusieurs grilles de paramètres

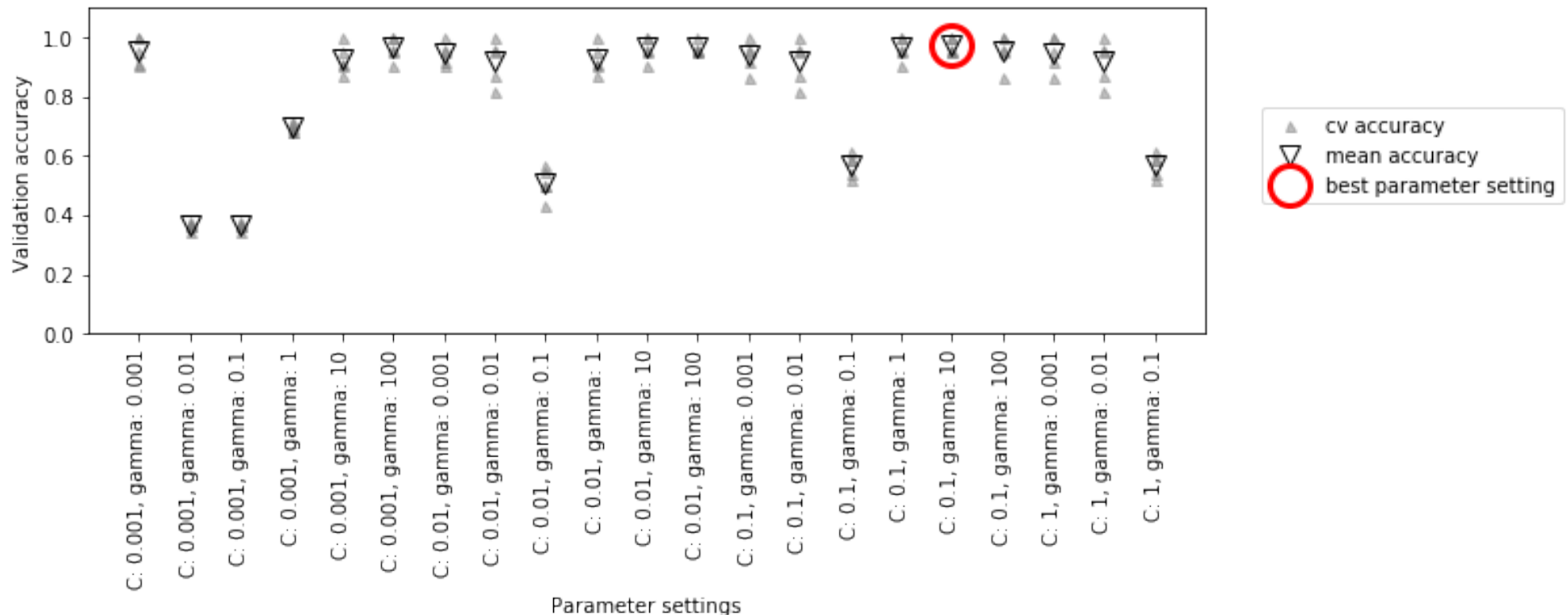
```
grille_param = [{'kernel': ['rbf'],  
                  'C': [0.001, 0.01, 0.1, 1, 10, 100],  
                  'gamma': [0.001, 0.01, 0.1, 1, 10, 100]}],  
               [{'kernel': ['linear'],  
                  'C': [0.001, 0.01, 0.1, 1, 10, 100]}] from  
grid_search = GridSearchCV(SVC(), grille_param, cv=5,  
                           return_train_score=True)à  
grid_search.fit(X_train, y_train)
```

Évaluation et amélioration de modèles

□ Amélioration de la performance de généralisation

❖ Méthodes pour comparer/améliorer des modèles

- Grid search : analyse des résultats (bien choisir l'espace de recherche)

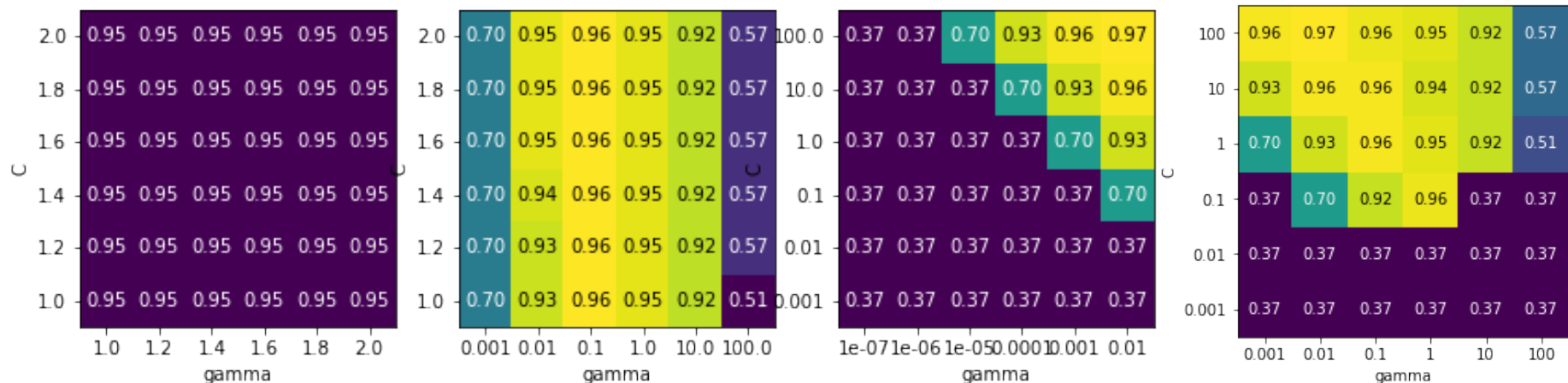


Évaluation et amélioration de modèles

❑ Amélioration de la performance de généralisation

❖ Méthodes pour comparer/améliorer des modèles

- Grid search : analyse des résultats (bien choisir l'espace de recherche)



Prétraitement

- ❑ **Rappel : fonctions prédictives**
- ❑ **Évaluation et amélioration de modèles**
- ❑ **Métriques d'évaluation de la performance**

Métrique d'évaluation de la performance

- ❑ **Choix de la métrique en fonction du type d'analyse et des objectifs**
 - ❖ Classification binaire, multi-classes, régression ?
 - ❖ Jeu de données équilibré, déséquilibré ?

- ❑ **Exemple**: pour la classification, la correctitude (accuracy) n'est pas la meilleure métrique si l'une des classes est beaucoup plus fréquentes que les autres (ex: 90%)
Correctitude : pourcentage de données correctement classées

Métrique d'évaluation de la performance

❑ Métrique pour classification binaire

❖ Matrice de confusion

Une des classes dite « positive », l'autre « négative »

Classe réelle\prédite	C1	C2
C1	True Positives (TP)	False Negatives (FN)
C2	False Positives (FP)	True Negatives (TN)

Métrique d'évaluation de la performance

❑ Métrique pour classification binaire

❖ Matrice de confusion

Une des classes dite « positive », l'autre « négative »

Classe réelle\prédite	C1	C2
C1	True Positives (TP)	False Negatives (FN)
C2	False Positives (FP)	True Negatives (TN)

Correctitude (Accuracy): pourcentage de données classées correctement

$$\text{Accuracy} = (TP + TN) / (TP + FP + FN + TN)$$

Taux d'erreur (Error rate):

$$\text{Error rate} = 1 - \text{Accuracy} = (FP + FN) / (TP + FP + FN + TN)$$

Métrique d'évaluation de la performance

❑ Métrique pour classification binaire

❖ Matrice de confusion

Une des classes dite « positive », l'autre « négative »

Classe réelle\prédite	C1	C2
C1	True Positives (TP)	False Negatives (FN)
C2	False Positives (FP)	True Negatives (TN)

Si le jeu de données est déséquilibré (Exemple Pos << Neg):

Sensibilité (Sensitivity): pourcentage de positifs classés correctement : **Sensitivity = TP / (TP+FN)**

Spécificité (Specificity): pourcentage de négatifs classés correctement : **Specificity = TN / (FP+TN)**

Métrique d'évaluation de la performance

❑ Métrique pour classification binaire

❖ Matrice de confusion

Une des classes dite « positive », l'autre « négative »

Classe réelle\prédite	C1	C2
C1	True Positives (TP)	False Negatives (FN)
C2	False Positives (FP)	True Negatives (TN)

Autre résumé de la matrice de confusion:

Exactitude (Precision): pourcentage de prédictions positifs qui sont réellement positifs : **Precision = $TP / (TP+FP)$**

Complétude (Recall): pourcentage de positifs qui sont effectivement prédits positifs : **Recall = $TP / (TP+FN)$**

Métrique d'évaluation de la performance

❑ Métrique pour classification binaire

❖ Matrice de confusion

Une des classes dite « positive », l'autre « négative »

Classe réelle\prédite	C1	C2
C1	True Positives (TP)	False Negatives (FN)
C2	False Positives (FP)	True Negatives (TN)

F-score : moyenne harmonique de l'exactitude et de la complétude

$$\text{F-score} = 2 * (\text{precision} * \text{recall}) / (\text{precision} + \text{recall})$$