Méthodes d'agrégation avec Python

Novembre 2025

Cédric Dangeard
cedric.dangeard@orange.com

Cours 4 : Au programme

- Optimisation des Hyperparamètres
- Grid & Random Search
- Importance de la validation croisée
- Interprétabilité des modèles
- TP : Optimisation

Optimisation des hyperparamètres

Objectif : Trouver la meilleure combinaison d'hyperparamètres pour un modèle donné.

Solutions :

Optimisation des hyperparamètres

Objectif : Trouver la meilleure combinaison d'hyperparamètres pour un modèle donné.

Solutions :

- Recherche exhaustive (Grid Search)
- Recherche aléatoire (Random Search)

Recherche exhaustive (Grid Search)

- GridSearchCV
  - Prend un modèle + un dictionnaire d'hyperparamètres à tester.
  - Évalue l'ensemble des combinaisons en validation croisée
  - Retourne la combinaison qui maximise la performance.

Exemple

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
# Definition du modèle et des hyperparamètres
rf = RandomForestClassifier()
param_grid = {
    'n_estimators': [10, 100, 1000],
    'max_features': ['sqrt', 'log2'],
    'max_depth': [None, 10, 100],
}
clf = GridSearchCV(
    estimator=rf,
    param_grid=param_grid,
    cv=5
)
# Lancement de la recherche
search = clf.fit(X,y)
# Affichage des meilleurs paramètres
search.best_params_
```

Recherche aléatoire (Random Search)

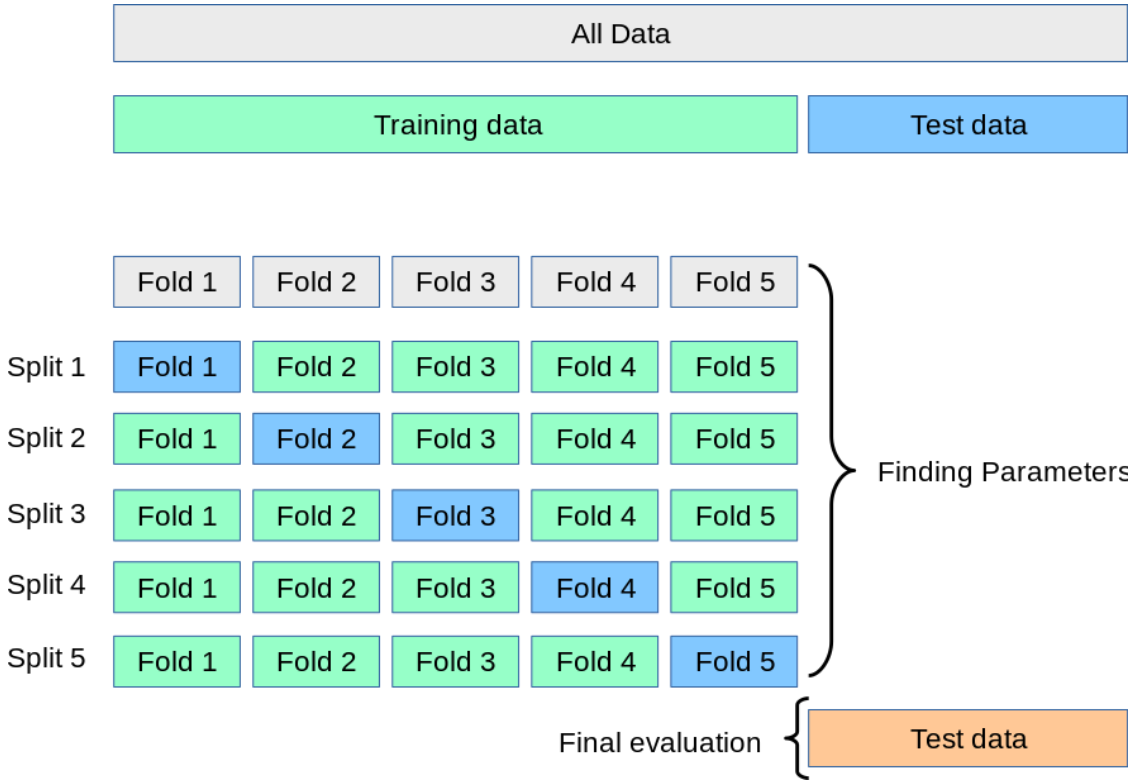
- RandomizedSearchCV
- Prend un modèle + un dictionnaire d'hyperparamètres à tester.
- Équivalent à GridSearchCV sur la plupart des points
- **n_iter** combinaisons d'hyperparamètres :
  - Maîtrise du temps d'exécution
  - Certaines combinaisons ignorées

Exemple

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
# Definition du modèle et des hyperparamètres
rf = RandomForestClassifier()
param_grid = {
    'n_estimators': [10, 100, 1000],
    'max_features': ['sqrt', 'log2'],
    'max_depth': [None, 10, 100],
}
clf = RandomizedSearchCV(
    estimator=rf,
    param_distributions=param_grid,
    n_iter=10,
    random_state=42,
    cv=5
)
# Lancement de la recherche
search = clf.fit(X,y)
# Affichage des meilleurs paramètres
search.best_params_
```

Validation K-Fold

- Coupe du set en k blocs
- Entraînement sur $k-1$ blocs
- Evaluation sur la partie non utilisée
- Répétition de l'opération k fois
- Score: moyenne des k scores
- Cas Particulier : Leave one Out
  - (K-Fold avec $k = n$)



source : scikit-learn.org

En pratique

```
from sklearn import metrics
scores = cross_val_score(model, X, y, cv=5, scoring='f1_macro')
```

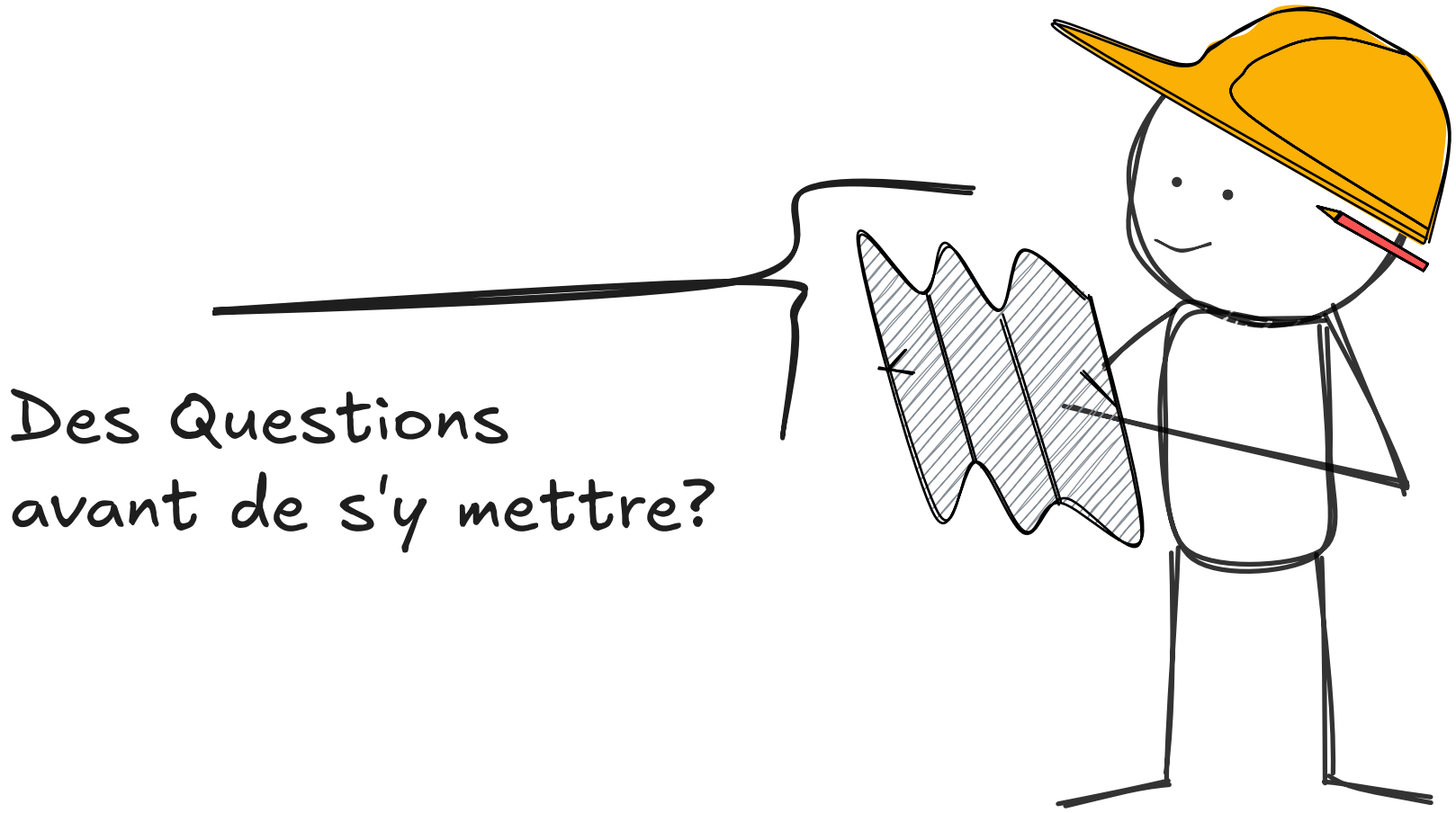
- Plus il y a de blocs, meilleure est l'approximation
- Mais plus le temps de traitement est long
- k=10 en général

Temps de calcul

Attention : Le temps de calcul augmente rapidement avec le nombre de combinaisons d'hyperparamètres et le nombre de plis dans la validation croisée.
Ex : pour 3 hyperparamètres avec 4 valeurs chacun et une validation croisée à 5 plis :

- Nombre de combinaisons : 4 x 4 x 4 = 64
- Nombre total d'entraînements : 64 x 5 = **320**

TP : Optimisation



Interprétabilité des modèles

Contrairement aux modèles linéaires, les modèles d'agrégation sont souvent considérés comme des "boîtes noires".

Il existe cependant des techniques pour interpréter ces modèles et comprendre l'importance des différentes caractéristiques.

- Que peut-on utiliser pour déterminer l'importance des caractéristiques dans un arbre ou un modèle d'agrégation ?

Random Forest Feature Importance

- Chaque arbre dans la forêt est construit en utilisant un sous-ensemble aléatoire des caractéristiques.
- Lors de la construction de chaque arbre, les caractéristiques qui contribuent le plus à la réduction
de l'impureté (par exemple, l'impureté de Gini ou l'entropie) sont utilisées pour diviser les nœuds.
- La "feature importance" est calculée en agrégeant la réduction de l'impureté
apportée par chaque caractéristique à travers tous les arbres de la forêt.

Random Forest Feature Importance

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
model.fit(X_train, y_train)
importances = model.feature_importances_
```

- Cette méthode est rapide, mais peut être biaisée
  en faveur des caractéristiques avec plus de niveaux ou de valeurs uniques.

Permutation Importance

- Pour calculer la permutation importance :
  - On mesure la performance du modèle sur un ensemble de test.
  - On permute aléatoirement les valeurs d'une caractéristique spécifique dans l'ensemble de test.
  - On mesure à nouveau la performance du modèle sur cet ensemble modifié.
  - La diminution de la performance du modèle due à la permutation de cette caractéristique est utilisée comme mesure de son importance.

Permutation Importance

```
from sklearn.inspection import permutation_importance
result = permutation_importance(model, X_test, y_test, n_repeats=10, random_state=42)
importances = result.importances_mean
```

- Cette méthode est plus fiable que la feature importance intégrée, mais elle est également plus coûteuse en termes de calcul.

Boosting Feature Importance

- Les modèles de boosting, comme XGBoost ou LightGBM, fournissent également des mesures d'importance des caractéristiques similaires à celles des forêts aléatoires.
- Ils calculent l'importance en fonction de la contribution de chaque caractéristique
  à la réduction de l'erreur lors de la construction des arbres.

Boosting Feature Importance

```
import xgboost as xgb
model = xgb.XGBClassifier()
model.fit(X_train, y_train)
importances = model.feature_importances_
```

- Comme pour les forêts aléatoires, cette méthode peut être biaisée en faveur des caractéristiques avec plus de niveaux ou de valeurs uniques.

TP : Feature Importance