TRAVAUX PRATIQUES

Leçon 03
Sélection de modèle & Évaluation

Durée estimée	2h30
Environnement	Python 3 + scikit-learn + Jupyter / Google Colab
Niveau	Débutant → Intermédiaire
Prérequis	Notions de base en Python et scikit-learn

1. Objectifs du TP

À la fin de cette séance, vous serez capables de :

- Séparer correctement un jeu de données en **train / validation / test**
- Utiliser la **validation croisée** (k-fold stratifiée)
- Optimiser des hyperparamètres avec **GridSearchCV** et **RandomizedSearchCV**
- Comparer plusieurs modèles de façon rigoureuse et reproductible
- Choisir et interpréter les métriques adaptées (classification et régression)
- Éviter le **data leakage** grâce aux pipelines

2. Consignes générales

- Fixez toujours `random_state=42` pour la reproductibilité.
- Utilisez un **Pipeline** (StandardScaler + modèle) pour éviter le data leakage.
- Ne touchez **jamais** au jeu de test avant l'évaluation finale.
- Commentez votre code et répondez aux questions dans le notebook.
- Livrable : un notebook Jupyter complet + conclusion personnelle (5-8 lignes).

3. Partie 1 — Préparation et séparation des données

3.1 Import des bibliothèques

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.datasets import load_breast_cancer, fetch_california_housing
from sklearn.model_selection import (
    train_test_split, cross_val_score, StratifiedKFold,
    GridSearchCV, RandomizedSearchCV
)
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression, Ridge
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import (
    accuracy_score, classification_report, confusion_matrix,
```

```
    roc_auc_score, roc_curve, mean_absolute_error,
    mean_squared_error, r2_score
)

RANDOM_STATE = 42
np.random.seed(RANDOM_STATE)
```

3.2 Chargement du jeu de données (Classification)

```
data = load_breast_cancer()
X, y = data.data, data.target

print("Shape :", X.shape)
print("Classes :", data.target_names)
print("Répartition :\n", pd.Series(y).value_counts(normalize=True).round(3))
```

Question 1 : *Les classes sont-elles équilibrées ? Quelle stratégie de validation croisée recommandez-vous ?*

3.3 Séparation Train / Validation / Test

On réserve 20 % des données pour le **test final**. Les 80 % restants sont ensuite découpés en train (60 %) et validation (20 %).

```
# 1. Garde 20 % pour le test final (jamais touché avant la fin)
X_temp, X_test, y_temp, y_test = train_test_split(
    X, y, test_size=0.20, stratify=y, random_state=RANDOM_STATE
)

# 2. Découpe des 80 % restants : 75 % train / 25 % validation
#    → proportions finales ≈ 60 % / 20 % / 20 %
X_train, X_val, y_train, y_val = train_test_split(
    X_temp, y_temp, test_size=0.25, stratify=y_temp, random_state=RANDOM_STATE
)

print(f"Train      : {X_train.shape[0]} exemples")
print(f"Validation : {X_val.shape[0]} exemples")
print(f"Test       : {X_test.shape[0]} exemples")
```

Question 2 : *Pourquoi ne doit-on jamais utiliser le jeu de test pendant la sélection de modèle ou l'optimisation des hyperparamètres ?*

Question 3 : *Que se passe-t-il si l'on choisit le meilleur modèle en regardant les performances sur le test set ?*

4. Partie 2 — Comparaison simple de modèles

On entraîne plusieurs modèles sur le train et on les évalue sur le jeu de validation (sans optimisation).

```
models = {
    "Logistic Regression": LogisticRegression(max_iter=2000, random_state=RANDOM_STATE),
    "KNN (k=5)"          : KNeighborsClassifier(n_neighbors=5),
    "Random Forest"      : RandomForestClassifier(random_state=RANDOM_STATE),
    "SVM"                : SVC(probability=True, random_state=RANDOM_STATE)
}

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled   = scaler.transform(X_val)

print("=== Performance sur le jeu de validation ===")
for name, model in models.items():
    model.fit(X_train_scaled, y_train)
    acc = accuracy_score(y_val, model.predict(X_val_scaled))
    print(f"{name:22s} : {acc:.4f}")
```

Question 4 : *Quel modèle semble le plus prometteur à ce stade ? Pourquoi la standardisation est-elle importante pour certains algorithmes ?*

5. Partie 3 — Validation croisée

La validation croisée permet une estimation plus robuste de la performance. On utilise un **Pipeline** pour que le StandardScaler soit recalculé à chaque fold (évite le data leakage).

```
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=RANDOM_STATE)

print("=== Validation croisée 5-fold (sur X_temp, y_temp) ===")
for name, model in models.items():
    pipe = Pipeline([
        ("scaler", StandardScaler()),
        ("model", model)
    ])
    scores = cross_val_score(pipe, X_temp, y_temp, cv=cv, scoring="accuracy")
    print(f"{name:22s} : {scores.mean():.4f}  ±  {scores.std():.4f}")
```

Question 5 : *Comparez les scores de validation croisée avec ceux obtenus sur le seul jeu de validation. Que constatez-vous ?*

Question 6 : *Pourquoi utilise-t-on un Pipeline plutôt que de standardiser les données une seule fois avant la CV ?*

6. Partie 4 — Optimisation d'hyperparamètres

6.1 GridSearchCV

```
pipe_rf = Pipeline([
    ("scaler", StandardScaler()),
    ("model", RandomForestClassifier(random_state=RANDOM_STATE))
])

param_grid = {
    "model__n_estimators"     : [50, 100, 200],
    "model__max_depth"        : [3, 5, 10, None],
    "model__min_samples_split": [2, 5, 10]
}

grid = GridSearchCV(
    pipe_rf, param_grid,
    cv=cv, scoring="accuracy",
    n_jobs=-1, verbose=1
)
grid.fit(X_temp, y_temp)

print("Meilleurs hyperparamètres :", grid.best_params_)
print("Meilleur score CV         :", round(grid.best_score_, 4))
```

6.2 RandomizedSearchCV (bonus)

Lorsque l'espace des hyperparamètres est grand, préférez `RandomizedSearchCV` (plus rapide).

```
from scipy.stats import randint

param_dist = {
    "model__n_estimators"     : randint(50, 300),
    "model__max_depth"        : [3, 5, 10, 15, None],
    "model__min_samples_split": randint(2, 20)
}

random_search = RandomizedSearchCV(
    pipe_rf, param_dist,
    n_iter=20, cv=cv, scoring="accuracy",
    random_state=RANDOM_STATE, n_jobs=-1
)
random_search.fit(X_temp, y_temp)
print("Meilleurs params (Random) :", random_search.best_params_)
print("Meilleur score CV         :", round(random_search.best_score_, 4))
```

7. Partie 5 — Évaluation finale sur le Test set

C'est **la seule fois** que l'on utilise le jeu de test. On prend le meilleur modèle trouvé par GridSearchCV.

```
best_model = grid.best_estimator_

y_pred  = best_model.predict(X_test)
y_proba = best_model.predict_proba(X_test)[:, 1]
```

```
print("=== Performance FINALE sur le Test set ===")
print(f"Accuracy : {accuracy_score(y_test, y_pred):.4f}")
print("\nRapport de classification :")
print(classification_report(y_test, y_pred, target_names=data.target_names))

# Matrice de confusion
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=data.target_names,
            yticklabels=data.target_names)
plt.xlabel("Prédit")
plt.ylabel("Réel")
plt.title("Matrice de confusion (Test set)")
plt.tight_layout()
plt.show()

# Courbe ROC
fpr, tpr, _ = roc_curve(y_test, y_proba)
auc = roc_auc_score(y_test, y_proba)
plt.figure(figsize=(5, 4))
plt.plot(fpr, tpr, label=f"AUC = {auc:.3f}")
plt.plot([0, 1], [0, 1], "k--")
plt.xlabel("Taux de faux positifs")
plt.ylabel("Taux de vrais positifs")
plt.title("Courbe ROC")
plt.legend()
plt.tight_layout()
plt.show()
```

8. Partie 6 — Régression (exercice complémentaire)

Appliquez la même méthodologie sur un problème de régression avec le dataset California Housing.

```
housing = fetch_california_housing()
X_reg, y_reg = housing.data, housing.target

X_tr, X_te, y_tr, y_te = train_test_split(
    X_reg, y_reg, test_size=0.2, random_state=RANDOM_STATE
)

pipe_reg = Pipeline([
    ("scaler", StandardScaler()),
    ("model", RandomForestRegressor(random_state=RANDOM_STATE))
])

param_grid_reg = {
    "model__n_estimators": [50, 100, 150],
    "model__max_depth"   : [5, 10, 15, None]
}

grid_reg = GridSearchCV(
    pipe_reg, param_grid_reg,
    cv=5, scoring="neg_mean_squared_error", n_jobs=-1
)
grid_reg.fit(X_tr, y_tr)

best_reg = grid_reg.best_estimator_
y_pred_r = best_reg.predict(X_te)

print(f"MAE  : {mean_absolute_error(y_te, y_pred_r):.3f}")
print(f"RMSE : {np.sqrt(mean_squared_error(y_te, y_pred_r)):.3f}")
print(f"R²   : {r2_score(y_te, y_pred_r):.3f}")
```

9. Questions de synthèse

Q1. *Pourquoi utilise-t-on un* ***Pipeline*** *(StandardScaler + modèle) à l'intérieur de la validation croisée ?*

Q2. *Quelle est la différence entre le score de validation croisée et le score final obtenu sur le test set ?*

Q3. *Dans quels cas préfère-t-on le* ***F1-score*** *à l'accuracy ? Donnez un exemple concret.*

Q4. *Que se passe-t-il concrètement si l'on optimise les hyperparamètres en utilisant le jeu de test ?*

Q5. *Proposez une amélioration possible de ce protocole expérimental (ex. : nested CV, autre métrique…).*

10. Livrable attendu

Un notebook Jupyter (ou fichier .py commenté) contenant :

- Tout le code exécuté et commenté
- Les graphiques (matrice de confusion + courbe ROC)
- Les réponses aux questions (Parties 1 à 5 + synthèse)
- Une **conclusion personnelle** (5 à 8 lignes) : ce que vous avez retenu de la sélection de modèle et de l'évaluation

Critères d'évaluation : respect du protocole (séparation train/val/test), utilisation correcte de la CV et des pipelines, qualité des analyses et de la conclusion, clarté du notebook.

Rappels importants

- **Ne jamais toucher au test set** : avant l'évaluation finale du modèle choisi.
- **Toujours utiliser un Pipeline** : quand on combine preprocessing + modèle dans une CV ou un GridSearch.
- **StratifiedKFold** : est préférable en classification (préserve les proportions de classes).
- **Le score de CV** : sert à sélectionner le modèle ; le score test sert à estimer la performance en généralisation.
- **Documentez vos choix** : pourquoi telle métrique ? pourquoi tel modèle ?

Bon courage et n'hésitez pas à poser vos questions pendant la séance !