Perceptron Multi-Couches (PMC) utilisé comme boîte noire - focus sur l'évaluation

Contexte important

Ce laboratoire s'adresse à des étudiants qui n'ont pas encore étudié le Machine Learning.

Le modèle (PMC) sera utilisé comme une boîte noire : l'objectif est d'apprendre à mesurer, comparer et sélectionner des modèles.

Toutes les activités demandées portent sur : métriques, courbes ROC, validation croisée, et recherche d'hyperparamètres.

1. Objectifs pédagogiques

- Mettre en place un protocole d'évaluation (train/test) reproductible.
- Calculer et interpréter : accuracy, matrice de confusion, précision, rappel (sensibilité), spécificité, F1.
- Tracer et analyser une courbe ROC, calculer l'AUC et étudier l'effet du seuil de décision.
- Estimer la performance par validation croisée (k-fold, LOO) et analyser la variance des scores.
- Optimiser des hyperparamètres via Grid Search et Randomized Search, puis sélectionner un modèle final.

2. Pré-requis et matériel

Outils : Python 3.10+, Jupyter/Colab, bibliothèques scikit-learn, numpy, pandas, matplotlib.

Le modèle PMC est fourni/paramétré dans le code.

3. Dataset réel utilisé

Nous utilisons un dataset réel de classification binaire : Breast Cancer Wisconsin (diagnostic).

Objectif : prédire si une tumeur est maligne (1) ou bénigne (0) à partir de caractéristiques numériques.

À observer dès le début :

- Nombre d'échantillons et nombre de variables (features).
- Répartition des classes (déséquilibre éventuel).
- Nécessité de normaliser/standardiser les variables (car elles n'ont pas la même échelle).

3.1 Exploration rapide du dataset (recommandé)

```
data = load_breast_cancer()
```

```
X, y = data.data, data.target

# Noms des variables
print("Exemples de features:", data.feature_names[:10])

# Statistiques rapides
df = pd.DataFrame(X, columns=data.feature_names)
print(df.describe().T[["mean", "std", "min", "max"]].head())

# Corrélation simple avec la cible (indicatif)
corr = df.assign(target=y).corr(numeric_only=True)["target"].sort_values(ascending=False)
print("Top corr positives:", corr.head(6))
print("Top corr négatives:", corr.tail(6))
```

Q3.1.1 Quelles sont les unités/échelles possibles des variables (au vu des min/max) ?

Q3.1.2 Le dataset semble-t-il standardisé au départ ? Justifiez.

Q3.1.3 Citez 2-3 variables fortement corrélées à la cible. Attention : corrélation ne signifie pas causalité.

4. Mise en place

Code 1 - Imports et constantes

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

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.dummy import DummyClassifier
from sklearn.neural_network import MLPClassifier

from sklearn.metrics import (confusion_matrix, ConfusionMatrixDisplay,
                             accuracy_score, precision_score, recall_score, f1_score,
                             roc_curve, roc_auc_score)

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

Partie A - Évaluation simple : train/test + métriques

Objectif : établir une baseline, entraîner un PMC, puis interpréter les métriques.

A.1 Chargement des données et split

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

print("X shape:", X.shape)
print("Classes (0/1) counts:", np.bincount(y))

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=RANDOM_STATE, stratify=y
)

print("Train:", X_train.shape, " Test:", X_test.shape)
```

QA.1.1 Combien d'échantillons et de variables contient le dataset ?

QA.1.2 Les classes sont-elles équilibrées ? Calculez le pourcentage de chaque classe.

QA.1.3 Pourquoi utilise-t-on stratify=y dans le split ?

A.2 Baseline (modèle naïf) vs PMC

Avant d'évaluer un modèle, on doit toujours comparer à une baseline simple.

```
# Baseline : prédire toujours la classe majoritaire
baseline = DummyClassifier(strategy="most_frequent").fit(X_train, y_train)
y_pred_base = baseline.predict(X_test)

print("Baseline accuracy:", accuracy_score(y_test, y_pred_base))
```

Ensuite, on entraîne un PMC (MLPClassifier). Vous n'avez pas besoin de comprendre l'optimisation interne.

```
pmc = Pipeline([
    ("scaler", StandardScaler()),
    ("mlp", MLPClassifier(hidden_layer_sizes=(30, 15),
                          activation="relu",
                          solver="adam",
                          alpha=1e-4,
                          max_iter=2000,
                          random_state=RANDOM_STATE))
])

pmc.fit(X_train, y_train)

y_pred_train = pmc.predict(X_train)
y_pred_test = pmc.predict(X_test)
```

```
print("PMC accuracy (train):", accuracy_score(y_train, y_pred_train))
print("PMC accuracy (test) :", accuracy_score(y_test, y_pred_test))
```

QA.2.1 Comparez l'accuracy de la baseline et du PMC sur le test. Quel est le gain absolu ?

QA.2.2 Comparez accuracy train vs test du PMC. Voyez-vous un signe de surapprentissage ?

QA.2.3 Pourquoi une baseline est-elle indispensable avant toute conclusion ?

QA.2.4 Le scaling (StandardScaler) change-t-il la performance ? Testez en supprimant StandardScaler (par curiosité). Que constatez-vous ?

A.3 Matrice de confusion et métriques dérivées

```
y_pred = y_pred_test

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=data.target_names)
disp.plot(values_format="d")
plt.title("Matrice de confusion - PMC")
plt.show()

# Métriques principales
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)  # sensibilité
f1 = f1_score(y_test, y_pred)

print("Accuracy :", acc)
print("Precision:", prec)
print("Recall   :", rec)
print("F1-score :", f1)

# Spécificité = TN / (TN + FP)
tn, fp, fn, tp = cm.ravel()
spec = tn / (tn + fp)
print("Specificity:", spec)
```

QA.3.1 Recopiez la matrice de confusion et identifiez TP, TN, FP, FN.

QA.3.2 Calculez (à la main) la sensibilité (rappel) et la spécificité à partir de la matrice. Vérifiez avec le code.

QA.3.3 Dans un contexte médical, quelle erreur est la plus critique : FP ou FN ? Justifiez.

QA.3.4 Donnez une interprétation pratique de précision vs rappel dans ce contexte.

Partie B - Courbe ROC, AUC et choix du seuil

Objectif : utiliser les scores/probabilités et étudier le compromis sensibilité/spécificité.

B.1 Probabilités, ROC et AUC

```
# Probabilité d'appartenir à la classe positive (malignant)
proba_pos = pmc.predict_proba(X_test)[:, 1]

fpr, tpr, thresholds = roc_curve(y_test, proba_pos)
auc = roc_auc_score(y_test, proba_pos)

plt.figure(figsize=(6,5))
plt.plot(fpr, tpr, label=f"ROC (AUC={auc:.3f})")
plt.plot([0,1], [0,1], linestyle="--", label="Hasard")
plt.xlabel("FPR = 1 - Spécificité")
plt.ylabel("TPR = Sensibilité (Recall)")
plt.title("Courbe ROC - PMC")
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()

print("AUC:", auc)
```

QB.1.1 Que signifie une AUC = 0.5 ? Et une AUC proche de 1 ?

QB.1.2 Expliquez le rôle du seuil de décision dans la construction de la courbe ROC.

QB.1.3 Sur votre courbe, le modèle est-il nettement meilleur que le hasard ? Justifiez avec l'AUC.

QB.1.4 Bonus : pourquoi la ROC peut être trompeuse si le dataset est très déséquilibré ?

B.2 Étude du seuil : sensibilité vs spécificité

```
def metrics_at_threshold(y_true, proba, thr):
    y_hat = (proba >= thr).astype(int)
    cm = confusion_matrix(y_true, y_hat)
    tn, fp, fn, tp = cm.ravel()
    acc = (tp + tn) / (tp + tn + fp + fn)
    prec = tp / (tp + fp) if (tp + fp) > 0 else 0.0
    rec = tp / (tp + fn) if (tp + fn) > 0 else 0.0
    spec = tn / (tn + fp) if (tn + fp) > 0 else 0.0
    f1 = 2*prec*rec/(prec+rec) if (prec+rec) > 0 else 0.0
    return acc, prec, rec, spec, f1

for thr in [0.2, 0.4, 0.5, 0.6, 0.8]:
    acc, prec, rec, spec, f1 = metrics_at_threshold(y_test, proba_pos, thr)
```

```
    print(f"thr={thr:0.1f} | acc={acc:.3f} prec={prec:.3f} rec={rec:.3f} spec={spec:.3f}
f1={f1:.3f}")
```

QB.2.1 Quand on augmente le seuil, que se passe-t-il en général sur la sensibilité (recall) et la spécificité ?

QB.2.2 Choisissez un seuil qui maximise la sensibilité (priorité : ne pas rater les cas positifs). Quel coût sur la spécificité ?

QB.2.3 Choisissez un seuil qui équilibre sensibilité/spécificité (indice de Youden J = TPR - FPR). Implémentez le calcul du meilleur seuil.

QB.2.4 Concluez : quel seuil recommanderiez-vous si le coût d'un faux négatif est très élevé ?

Partie C - Validation croisée (k-fold, LOO) et stabilité

Objectif : obtenir une estimation plus robuste de la performance, et mesurer la variabilité selon les splits.

C.0 Attention à la fuite de données (data leakage)

Point clé

Standardiser les données avant de faire la validation croisée peut créer une fuite de données.

La bonne pratique est d'utiliser une Pipeline (Scaler + Modèle), comme dans ce lab, pour que la standardisation soit refaite dans chaque fold.

C.1 k-fold stratifié (k=5 puis k=10)

```
cv5 = StratifiedKFold(n_splits=5, shuffle=True, random_state=RANDOM_STATE)
cv10 = StratifiedKFold(n_splits=10, shuffle=True, random_state=RANDOM_STATE)

# Score : AUC (plus informatif qu'accuracy en cas de déséquilibre)
scores5 = cross_val_score(pmc, X, y, cv=cv5, scoring="roc_auc")
scores10 = cross_val_score(pmc, X, y, cv=cv10, scoring="roc_auc")

print("CV-5 AUC  : mean=", scores5.mean(), " std=", scores5.std())
print("CV-10 AUC : mean=", scores10.mean(), " std=", scores10.std())
```

QC.1.1 Comparez la moyenne et l'écart-type (std) entre CV-5 et CV-10. Que remarquez-vous ?

QC.1.2 Pourquoi la validation croisée donne-t-elle souvent une estimation plus fiable qu'un seul split train/test ?

QC.1.3 Sur quel score vous baseriez-vous pour comparer 2 modèles : la moyenne seule, ou moyenne + écart-type ? Pourquoi ?

QC.1.4 Bonus : testez cross_val_score avec scoring='accuracy'. Les conclusions changent-elles ?

C.2 Leave-One-Out (LOO) sur un sous-ensemble

Le LOO est coûteux (autant d'entraînements que d'exemples). Pour rester dans une durée raisonnable, on l'applique sur un sous-ensemble.

```
from sklearn.model_selection import LeaveOneOut

# Sous-ensemble pour LOO
X_small, y_small = X[:80], y[:80]  # ajustez si nécessaire

loo = LeaveOneOut()
scores_loo = cross_val_score(pmc, X_small, y_small, cv=loo, scoring="roc_auc")

print("LOO AUC : mean=", scores_loo.mean(), " std=", scores_loo.std(), " n=",
len(scores_loo))
```

QC.2.1 Comparez (sur ce sous-ensemble) la moyenne LOO avec une CV-5 sur le même sous-ensemble. Est-ce proche ?

QC.2.2 Pourquoi le LOO peut avoir une variance élevée malgré un entraînement "maximal" à chaque itération ?

QC.2.3 Dans quel cas le LOO est-il intéressant, et dans quel cas il devient impraticable ?

Partie D - Sélection et optimisation : Grid Search et Randomized Search

Objectif : chercher de meilleurs hyperparamètres et éviter les conclusions basées sur un seul split.

D.1 GridSearchCV (recherche exhaustive)

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    "mlp__hidden_layer_sizes": [(20,), (40,), (30, 15), (60, 30)],
    "mlp__alpha": [1e-5, 1e-4, 1e-3],
    "mlp__learning_rate_init": [1e-3, 3e-4]
}

grid = GridSearchCV(
    estimator=pmc,
    param_grid=param_grid,
    scoring="roc_auc",
    cv=cv5,
```

```
    n_jobs=-1
)

grid.fit(X_train, y_train)

print("Best CV AUC:", grid.best_score_)
print("Best params:", grid.best_params_)
```

D.2 Évaluer le meilleur modèle sur le test (hold-out)

```
best_model = grid.best_estimator_
proba_test = best_model.predict_proba(X_test)[:, 1]

auc_test = roc_auc_score(y_test, proba_test)
print("Test AUC (hold-out):", auc_test)
```

QD.2.1 Le meilleur score CV (grid.best_score_) est-il proche du score test ? Si non, proposez une explication.

QD.2.2 Quels hyperparamètres semblent les plus influents ? Justifiez à partir des résultats.

QD.2.3 Pourquoi utiliser scoring='roc_auc' plutôt que accuracy ici ?

QD.2.4 Quel risque existe si on utilise le test pour choisir les hyperparamètres ?

D.3 RandomizedSearchCV (recherche aléatoire)

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import loguniform

param_dist = {
    "mlp__hidden_layer_sizes": [(20,), (40,), (60,), (30, 15), (60, 30)],
    "mlp__alpha": loguniform(1e-6, 1e-2),
    "mlp__learning_rate_init": loguniform(1e-4, 3e-3)
}

rnd = RandomizedSearchCV(
    estimator=pmc,
    param_distributions=param_dist,
    n_iter=20,
    scoring="roc_auc",
    cv=cv5,
    random_state=RANDOM_STATE,
    n_jobs=-1
)

rnd.fit(X_train, y_train)
```

```
print("Best CV AUC:", rnd.best_score_)
print("Best params:", rnd.best_params_)
```

QD.3.1 Comparez Grid Search vs Randomized Search : temps d'exécution, meilleure AUC, meilleure configuration.

QD.3.2 Pourquoi Randomized Search peut-il être plus efficace quand l'espace des hyperparamètres est grand ?

QD.3.3 Reprenez le meilleur modèle randomisé et calculez son AUC sur le test. Comparez à Grid Search.

5. Rendu attendu

Rendre un notebook (ou script) + une courte synthèse format word/PDF (2 à 3 pages) contenant :

- Tableau récapitulatif : baseline vs PMC (accuracy, precision, recall, specificity, F1) + commentaire.
- Courbe ROC + AUC (sur le test) et discussion sur le choix du seuil (avec justification).
- Résultats de validation croisée (CV-5 et CV-10 : moyenne + std) + interprétation.
- Résultats d'optimisation (GridSearch et RandomizedSearch) + comparaison sur le test.
- Une section "Analyse d'erreurs" : 5 à 10 exemples (ou types d'exemples) mal prédits et hypothèses explicatives.