TP : Overfitting et Élagage des Arbres de Décision

Objectifs pédagogiques

- Comprendre le phénomène d'overfitting
- Observer l'impact de la complexité d'un arbre sur sa généralisation
- Appliquer l'élagage pour améliorer les performances
- Interpréter les métriques de performance

Introduction

Dans ce TP, vous allez travailler sur un jeu de données concernant la réussite d'étudiants. Vous allez :

1. Créer un arbre de décision qui **surappprend** (overfitting)
2. Appliquer des techniques d'**élagage** pour améliorer la généralisation

---

Matériel nécessaire

Fichier de données

- **Téléchargez** le fichier `dataset_etudiants.csv`
- Placez-le dans le même dossier que votre script Python

Description du dataset

Le fichier contient 100 observations avec 3 colonnes :

- `heures_etude` : nombre d'heures d'étude par semaine
- `presence` : taux de présence aux cours (en %)
- `reussite` : 1 si l'étudiant a réussi, 0 sinon

PARTIE 1 : Observer l'Overfitting

Exercice 1.1 : Chargement et séparation des données

Consignes :

1. Chargez le fichier CSV
2. Affichez les 10 premières lignes

3. Séparez les variables explicatives (X) et la variable cible (y)
4. Divisez les données en ensemble d'entraînement (70%) et de test (30%)

```
import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt

# 1. Charger les données
data = pd.read_csv('dataset_etudiants.csv')

# 2. Afficher les premières lignes
print("Aperçu des données :")
print(data.head(10))
print(f"\nNombre total d'observations : {len(data)}")

# 3. Séparer X et y
# À COMPLÉTER
X = # Variables explicatives (heures_etude, presence)
y = # Variable cible (reussite)

# 4. Diviser en train/test (70/30)
# À COMPLÉTER
X_train, X_test, y_train, y_test = train_test_split(
    # À COMPLÉTER
    random_state=42
)

print(f"\nTaille ensemble d'entraînement : {len(X_train)}")
print(f"Taille ensemble de test : {len(X_test)}")
```

Questions :

- Combien d'observations avez-vous dans l'ensemble d'entraînement ?
- Combien dans l'ensemble de test ?
- Pourquoi est-il important de séparer les données ?

Exercice 1.2 : Créer un arbre sans limitation (overfitting)

Consignes :

1. Créez un arbre de décision **SANS aucune limitation** de complexité
2. Entraînez-le sur les données d'entraînement
3. Calculez l'accuracy sur train ET test
4. Affichez les caractéristiques de l'arbre (profondeur, nombre de feuilles)

```
# 1. Créer l'arbre SANS limitation
# À COMPLÉTER
arbre_overfit = DecisionTreeClassifier(
    random_state=42
    # NE RIEN AJOUTER - laisser l'arbre grandir librement
)

# 2. Entraîner l'arbre
# À COMPLÉTER


# 3. Faire des prédictions
# À COMPLÉTER
y_pred_train =
y_pred_test =

# 4. Calculer les performances
# À COMPLÉTER
```

```
acc_train =
acc_test =

print("=" * 50)
print("ARBRE SANS LIMITATION (OVERFITTING)")
print("=" * 50)
print(f"Accuracy Train : {acc_train:.3f}")
print(f"Accuracy Test  : {acc_test:.3f}")
print(f"Différence     : {acc_train - acc_test:.3f}")
print(f"\nProfondeur de l'arbre : {arbre_overfit.get_depth()}")
print(f"Nombre de feuilles    : {arbre_overfit.get_n_leaves()}")
print("=" * 50)
```

Questions 1.2 :

1. Quelle est l'accuracy sur l'ensemble d'entraînement ?
2. Quelle est l'accuracy sur l'ensemble de test ?
3. Calculez la différence entre les deux. Que constatez-vous ?
4. Quelle est la profondeur de l'arbre ? Est-ce raisonnable ?
5. D'après ces résultats, pensez-vous que l'arbre généralise bien ? Pourquoi ?

Exercice 1.3 : Visualiser l'arbre en overfitting

Consignes : Visualisez l'arbre créé précédemment.

```
# Visualisation de l'arbre
plt.figure(figsize=(25, 15))
plot_tree(arbre_overfit,
          feature_names=['heures_etude', 'presence'],
          class_names=['Échec', 'Réussite'],
          filled=True,
          fontsize=10)
plt.title("Arbre en Overfitting (sans limitation)", fontsize=16)
plt.savefig('arbre_overfit.png', dpi=300, bbox_inches='tight')
plt.show()
```

Questions 1.3 :

1. L'arbre est-il facile à lire et à interpréter ?
2. Observez-vous beaucoup de branches ?
3. Certaines feuilles contiennent-elles très peu d'exemples ?
4. Pensez-vous que cet arbre a "mémorisé" les données plutôt que d'apprendre des règles générales ?

PARTIE 2 : Appliquer l'Élagage

Exercice 2.1 : Tester différentes profondeurs maximales

Consignes :

1. Testez des arbres avec différentes profondeurs maximales (2, 3, 4, 5, 6, 8, 10)
2. Pour chaque profondeur, calculez l'accuracy sur train et test
3. Stockez les résultats dans un tableau

```
# Tester différentes profondeurs
profondeurs = [2, 3, 4, 5, 6, 8, 10]
resultats = []
```

```
for depth in profondeurs:
    # À COMPLÉTER
    # 1. Créer un arbre avec max_depth=depth
    arbre =

    # 2. Entraîner l'arbre


    # 3. Calculer les performances
    acc_train =
    acc_test =

    # 4. Stocker les résultats
    resultats.append({
        'profondeur': depth,
        'acc_train': acc_train,
        'acc_test': acc_test,
        'nb_feuilles': arbre.get_n_leaves(),
        'difference': acc_train - acc_test
    })

# Afficher les résultats
df_resultats = pd.DataFrame(resultats)
print("\n" + "=" * 80)
print("RÉSULTATS POUR DIFFÉRENTES PROFONDEURS")
print("=" * 80)
print(df_resultats.to_string(index=False))
print("=" * 80)
```

Questions 2.1 :

1. Quelle profondeur donne la **meilleure accuracy sur test** ?
2. Que se passe-t-il quand la profondeur augmente ?
3. À partir de quelle profondeur observez-vous de l'overfitting ?
4. Complétez le tableau suivant :

Profondeur	Acc Train	Acc Test	Overfitting ?
2			
3			
4			
5			

Exercice 2.2 : Visualiser les courbes d'apprentissage

Consignes : Créez un graphique montrant l'évolution de l'accuracy selon la profondeur.

```
# Visualisation des courbes
plt.figure(figsize=(12, 7))

# À COMPLÉTER
# Tracer acc_train en fonction de la profondeur
plt.plot()

# Tracer acc_test en fonction de la profondeur
plt.plot()

plt.xlabel('Profondeur maximale', fontsize=12)
plt.ylabel('Accuracy', fontsize=12)
plt.title('Évolution de l\'Accuracy selon la Profondeur de l\'Arbre', fontsize=14)
plt.legend(fontsize=11)
plt.grid(True, alpha=0.3)
plt.xticks(profondeurs)
plt.savefig('courbes_apprentissage.png', dpi=300, bbox_inches='tight')
plt.show()
```

Questions 2.2 :

1. À quel moment les deux courbes commencent-elles à diverger ?
2. Que représente cette divergence ?
3. Pourquoi l'accuracy train continue-t-elle à augmenter alors que l'accuracy test stagne ou diminue ?

Exercice 2.3 : Créer l'arbre optimal

Consignes :

1. Identifiez la profondeur optimale d'après vos résultats
2. Créez un arbre avec cette profondeur
3. Comparez-le à l'arbre sans limitation

```
# 1. Définir la meilleure profondeur
# À COMPLÉTER (basé sur vos résultats)
meilleure_profondeur =

# 2. Créer l'arbre optimal
# À COMPLÉTER
arbre_optimal = DecisionTreeClassifier(

)


# 3. Calculer les performances
acc_train_optimal = accuracy_score(y_train, arbre_optimal.predict(X_train))
acc_test_optimal = accuracy_score(y_test, arbre_optimal.predict(X_test))

print("\n" + "=" * 50)
print("COMPARAISON : OVERFITTING vs ÉLAGAGE")
print("=" * 50)
print(f"{'Modèle':&lt;20} {'Acc Train':&lt;12} {'Acc Test':&lt;12} {'Feuilles':&lt;10}")
print("-" * 50)
print(f"{'Sans limitation':&lt;20} {acc_train:.3f}{'':&lt;8} {acc_test:.3f}{'':&lt;8} {arbre_overfit.get_
print(f"{'Avec élagage':&lt;20} {acc_train_optimal:.3f}{'':&lt;8} {acc_test_optimal:.3f}{'':&lt;8} {arbre
print("=" * 50)
```

Questions 2.3 :

1. L'arbre élagué a-t-il une meilleure accuracy sur test ?
2. Combien de feuilles possède l'arbre élagué par rapport à l'arbre complet ?
3. Quel modèle choisirez-vous pour prédire la réussite de nouveaux étudiants ? Justifiez.

Exercice 2.4 : Validation croisée pour choisir la profondeur optimale

Jusqu'ici, nous avons utilisé un seul découpage train/test. Mais ce découpage est-il représentatif ? La **validation croisée** permet d'obtenir une estimation plus robuste des performances.

Principe : Diviser les données en K parties (folds), entraîner K fois en utilisant K-1 parties pour l'entraînement et 1 partie pour la validation, puis faire la moyenne.

Consignes :

1. Utilisez la validation croisée à 5 folds (cv=5)
2. Testez les profondeurs de 2 à 10
3. Comparez les résultats avec et sans validation croisée

```
from sklearn.model_selection import cross_val_score

# Tester différentes profondeurs avec validation croisée
profondeurs_cv = range(2, 11)
```

```
resultats_cv = []

for depth in profondeurs_cv:
    arbre = DecisionTreeClassifier(max_depth=depth, random_state=42)

    # Validation croisée à 5 folds
    # À COMPLÉTER
    scores_cv = cross_val_score(arbre, X_train, y_train, cv=5, scoring='accuracy')

    # Calculer la moyenne et l'écart-type
    # À COMPLÉTER
    mean_cv = scores_cv.mean()
    std_cv = scores_cv.std()

    resultats_cv.append({
        'profondeur': depth,
        'mean_cv_score': mean_cv,
        'std_cv_score': std_cv
    })

df_cv = pd.DataFrame(resultats_cv)
print("\n" + "=" * 70)
print("RÉSULTATS AVEC VALIDATION CROISÉE (5 folds)")
print("=" * 70)
print(df_cv.to_string(index=False))
print("=" * 70)

# Trouver la meilleure profondeur selon la validation croisée
# À COMPLÉTER
meilleure_profondeur_cv = df_cv.loc[df_cv['mean_cv_score'].idxmax(), 'profondeur']
print(f"\nMeilleure profondeur selon validation croisée : {int(meilleure_profondeur_cv)}")
```

Visualisation comparative :

```
plt.figure(figsize=(12, 6))

# Tracer les scores de validation croisée avec barres d'erreur
plt.errorbar(df_cv['profondeur'], df_cv['mean_cv_score'],
             yerr=df_cv['std_cv_score'],
             marker='o', label='Validation Croisée (5-fold)',
             linewidth=2, capsize=5)

plt.xlabel('Profondeur maximale', fontsize=12)
plt.ylabel('Accuracy moyenne', fontsize=12)
plt.title('Validation Croisée : Choix de la Profondeur Optimale', fontsize=14)
plt.legend(fontsize=11)
plt.grid(True, alpha=0.3)
plt.xticks(profondeurs_cv)
plt.savefig('validation_croisee.png', dpi=300, bbox_inches='tight')
plt.show()
```

Questions 2.4 :

1. Quelle profondeur donne le meilleur score moyen en validation croisée ?
2. Est-ce la même que celle trouvée avec un simple découpage train/test ?
3. Qu'indique l'écart-type (std) ? Une grande valeur est-elle préférable ?
4. Pourquoi la validation croisée est-elle plus fiable qu'un seul découpage ?

Exercice 2.5 : Créer et évaluer l'arbre final

Consignes :

1. Utilisez la profondeur optimale trouvée par validation croisée
2. Entraînez l'arbre sur **tout** l'ensemble d'entraînement
3. Évaluez sur l'ensemble de test (qui n'a JAMAIS été utilisé)

```
# 1. Créer l'arbre avec la profondeur optimale (validation croisée)
```

```
# À COMPLÉTER
arbre_final = DecisionTreeClassifier(
    max_depth=int(meilleure_profondeur_cv),
    random_state=42
)

# 2. Entraîner sur TOUT le train
arbre_final.fit(X_train, y_train)

# 3. Évaluer sur test
acc_train_final = accuracy_score(y_train, arbre_final.predict(X_train))
acc_test_final = accuracy_score(y_test, arbre_final.predict(X_test))

print("\n" + "=" * 70)
print("MODÈLE FINAL (choisi par validation croisée)")
print("=" * 70)
print(f"Profondeur optimale : {int(meilleure_profondeur_cv)}")
print(f"Accuracy Train : {acc_train_final:.3f}")
print(f"Accuracy Test  : {acc_test_final:.3f}")
print(f"Nombre de feuilles : {arbre_final.get_n_leaves()}")
print("=" * 70)
```

Visualisation de l'arbre final :

```
plt.figure(figsize=(18, 10))
plot_tree(arbre_final,
          feature_names=['heures_etude', 'presence'],
          class_names=['Échec', 'Réussite'],
          filled=True,
          fontsize=12,
          rounded=True)
plt.title(f"Arbre Final (profondeur = {int(meilleure_profondeur_cv)}, choisi par CV)", fontsize=16)
plt.savefig('arbre_final.png', dpi=300, bbox_inches='tight')
plt.show()
```

Questions 2.5 :

1. Cet arbre est-il plus facile à interpréter que l'arbre sans limitation ?
2. Pouvez-vous identifier les règles principales de classification ?
3. Donnez un exemple de règle extraite de l'arbre (ex: "Si heures_etude > X ET presence > Y alors réussite")
4. Quelle est la performance finale sur le test ? Est-elle satisfaisante ?

PARTIE 3 : Questions de Synthèse

Question 3.1 : Définitions

Expliquez avec vos propres mots :

- **Overfitting** :
- **Élagage** :
- **Généralisation** :

Question 3.2 : Compromis biais-variance

1. Un arbre très profond a-t-il un biais élevé ou faible ?
2. Un arbre très profond a-t-il une variance élevée ou faible ?
3. Quel est le risque d'un arbre trop simple (profondeur = 1) ?

Question 3.3 : Méthodologie complète

Vous devez créer un système de prédiction pour votre université. Décrivez les étapes que vous suivriez :

1. Comment diviseriez-vous les données ?
2. Comment choisiriez-vous les hyperparamètres ?
3. Comment évalueriez-vous le modèle final ?
4. Quel modèle déployeriez-vous en production ?

Question 3.4 : Validation croisée

1. Pourquoi la validation croisée est-elle plus fiable qu'un simple train/test split ?
2. Quel est le rôle de l'ensemble de test dans cette méthodologie ?
3. Pourquoi ne doit-on JAMAIS utiliser l'ensemble de test pour choisir les hyperparamètres ?

BONUS : Autres méthodes d'élagage

Bonus 1 : Tester min_samples_leaf

```
# Tester différentes valeurs de min_samples_leaf
min_samples_values = [1, 5, 10, 15, 20]
resultats_bonus = []

for min_samples in min_samples_values:
    arbre = DecisionTreeClassifier(min_samples_leaf=min_samples, random_state=42)
    arbre.fit(X_train, y_train)

    acc_train = accuracy_score(y_train, arbre.predict(X_train))
    acc_test = accuracy_score(y_test, arbre.predict(X_test))

    resultats_bonus.append({
        'min_samples_leaf': min_samples,
        'acc_train': acc_train,
        'acc_test': acc_test,
        'profondeur': arbre.get_depth(),
        'nb_feuilles': arbre.get_n_leaves()
    })

df_bonus = pd.DataFrame(resultats_bonus)
print(df_bonus)
```

Question Bonus :
Quel est l'effet de `min_samples_leaf` sur l'arbre ?

Aide-mémoire

Commandes principales

```
# Créer un arbre
```

```
arbre = DecisionTreeClassifier(max_depth=5, random_state=42)

# Entraîner
arbre.fit(X_train, y_train)

# Prédire
predictions = arbre.predict(X_test)

# Évaluer
accuracy = accuracy_score(y_test, predictions)

# Visualiser
plot_tree(arbre, feature_names=['var1', 'var2'], filled=True)
```

Bon travail !