جامعة سيدي محمد بن عبد الله بفاس
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵙⵉⴷⵉ ⵎⵓⵃⴰⵎⴻⴷ ⴱⵏ ⵄⴱⴷⵍⵍⴰⵀ ⴱ ⴼⴰⵙ
UNIVERSITÉ SIDI MOHAMED BEN ABDELLAH DE FES

كلية العلوم والتقنيات فاس
ⵜⴰⵖⵡⵍⵉⵜ ⵏ ⵜⵙⴽⵏⵉⵡⵉⵏ ⴷ ⵜⵓⵙⵙⵏⵉⵡⵉⵏ - ⴼⴰⵙ
FACULTÉ DES SCIENCES ET TECHNIQUES DE FÈS

TP N3: Machine Learning

LES ARBRES DE DECISION
&
FORETS ALEATOIRES

MAHDA KAOUTAR | SDSI

Démarche :

Nous allons appliquer les arbres de décision sur le jeu de données reçu par e-mail. Ensuite, nous sélectionnerons deux jeux de données sur Kaggle : l'un avec des variables qualitatives et l'autre avec des variables quantitatives. Nous appliquerons les arbres de décision et les forêts aléatoires sur ces deux jeux de données.

1. Le jeu de données Score_Sys_1.txt et Score_Sys_2.txt:

- Arbre de décisions

1.1. Chargement et exploration des données :

```
[99]: # Chargement des fichiers
      df3 = pd.read_csv('/kaggle/input/scores/Score_Sys_1.txt', sep="\s+", header=None, names=['
      df4 = pd.read_csv('/kaggle/input/scores/Score_Sys_2.txt', sep="\s+", header=None, names=['

      # Fusion des datasets
      dfglob = pd.concat([df3, df4], ignore_index=True)
```

```
[100]: print(dfglob.head())

    Class  Score
0       1  1.355
1       1  2.403
2       1  1.443
3       1  3.125
4       1  1.640
```

1.2. Séparation des données en train/test :

```
# Séparation des features et labels
X = dfglob[["Score"]]
y = dfglob["Class"]

# Division en ensemble d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

1.3. Entraînement d'un modèle DecisionTreeClassifier :

➢ CART

```
]:
# Entraînement du modèle
model = DecisionTreeClassifier(criterion='gini',max_depth=5,random_state=42)
model.fit(X_train, y_train)

# Prédictions
y_pred = model.predict(X_test)
```

➢ ID3

```
[107]:
# Entraînement du modèle
model = DecisionTreeClassifier(criterion='entropy',max_depth=5,random_state=42)
model.fit(X_train, y_train)

# Prédictions
y_pred = model.predict(X_test)
```

1.4. Évaluation du modèle avec une matrice de confusion et un score d'accuracy :

➢ CART

```
# Évaluation du modèle
conf_matrix = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)

print("Matrice de confusion:\n", conf_matrix)
print("Accuracy:", accuracy)
```

```
Matrice de confusion:
 [[4469  250]
 [ 306 2218]]
Accuracy: 0.9232362280822863
```

➢ ID3

```
[108]:
# Évaluation du modèle
conf_matrix = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)

print("Matrice de confusion:\n", conf_matrix)
print("Accuracy:", accuracy)
```

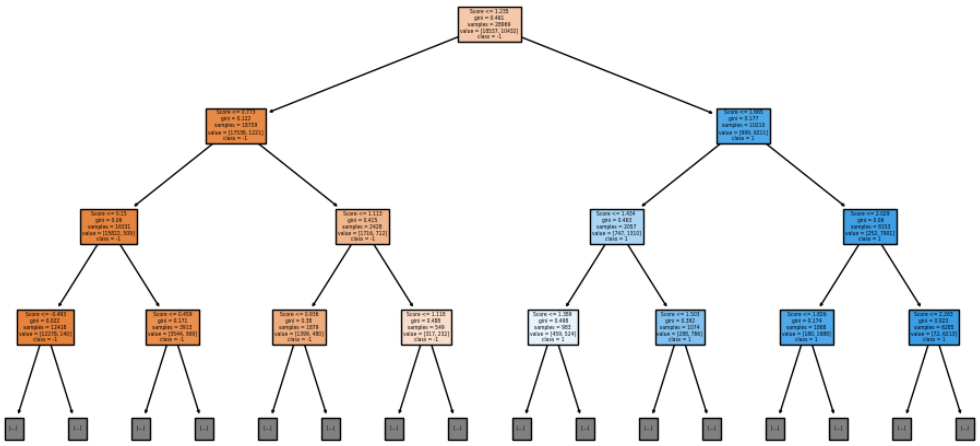
```
Matrice de confusion:
 [[4447  272]
 [ 291 2233]]
Accuracy: 0.9222697777164158
```

Après avoir évalué le module, nous pouvons conclure que les deux algorithmes donnent approximativement la même valeur de précision.

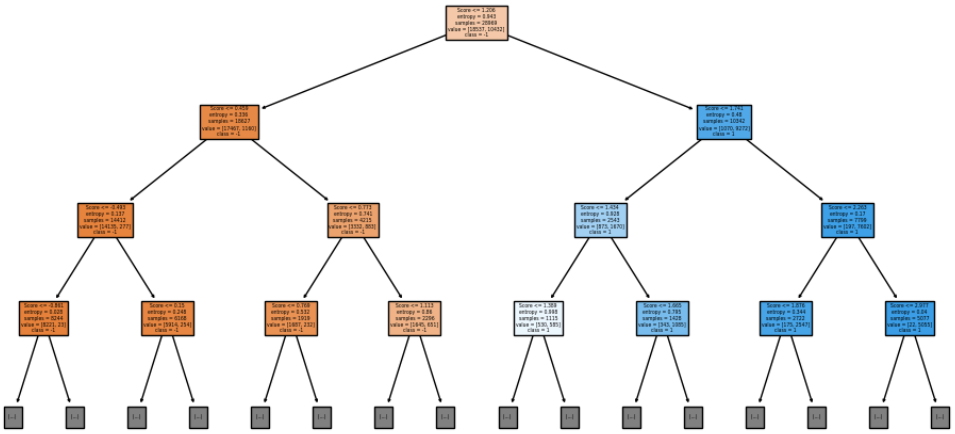
1.5. Visualisation de l'arbre de décision :

➢ CART

```
# Visualisation de l'arbre de décision
plt.figure(figsize=(12, 6))
plot_tree(model, feature_names=["Score"], class_names=["-1", "1"], filled=True, max_depth=
plt.show()
```



➢ ID3



- Forets aléatoires

➢ CART

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score

# Création du modèle de forêts aléatoires
model = RandomForestClassifier(n_estimators=30, criterion='gini', random_state=42)

# Validation croisée
cv_scores = cross_val_score(model, X, y, cv=3)

# Affichage des résultats
print(f"\nScores de validation croisée : {cv_scores}")
print(f"Moyenne des scores de validation croisée : {cv_scores.mean():.4f}")
```

```
Scores de validation croisée : [0.90928672 0.90754701 0.91077051]
Moyenne des scores de validation croisée : 0.9092
```

➢ ID3

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score

# Création du modèle de forêts aléatoires
model = RandomForestClassifier(n_estimators=30, criterion='entropy', random_state=42)

# Validation croisée
cv_scores = cross_val_score(model, X, y, cv=3)

# Affichage des résultats
print(f"\nScores de validation croisée : {cv_scores}")
print(f"Moyenne des scores de validation croisée : {cv_scores.mean():.4f}")
```

```
Scores de validation croisée : [0.90928672 0.90754701 0.91077051]
Moyenne des scores de validation croisée : 0.9092
```

Les algorithmes CART et ID3 donnent des performances très proches, ce qui suggère qu'ils fonctionnent de manière assez similaire pour ce jeu de données. ID3 et CART sont tous deux des algorithmes d'arbres de décision, mais ID3 utilise l'entropie comme critère de division des noeuds tandis que CART utilise l'indice de Gini. Le fait que leurs performances soient proches indique que ces critères de séparation fonctionnent de manière équivalente dans ce cas particulier.

2. Le jeu de données testdat et traindat:

- Arbre de décisions

2.1. Chargement et exploration des données :

[62]:
```
# Chargement des données d'entraînement et de test
trainDat = pd.read_csv("/kaggle/input/dataset/traindat.txt", delim_whitespace=True)
testDat = pd.read_csv("/kaggle/input/dataset/Testdat.txt", delim_whitespace=True)

# Fusionner les deux ensembles de données
data = pd.concat([trainDat, testDat], axis=0).reset_index(drop=True)
```

[63]:
```
# Afficher un aperçu des données fusionnées
print(data.head())
print(data.shape)
```

```
   y    m00         mu02        mu11         mu20        mu03         mu12  \
0  a  119.0  1164.571429  -84.000000  2274.705882 -728.448980 -1026.235294
1  a  124.0  1205.870968  -30.129032  2439.120968 -703.298647 -1067.540583
2  a  123.0  1167.365854  -47.073171  2372.747967 -531.112433 -1078.021416
3  a  131.0  1288.229008  -13.320611  2523.648855 -480.553814 -1266.542218
4  a  133.0  1385.879699 -148.030075  2644.992481 -613.045395 -1565.516988

         mu21         mu30
0 -308.016807  2446.878893
1 -395.008325  2494.423127
2 -452.984335  2438.033181
3 -219.485170  2209.729619
4 -251.322856  2869.338459
(200, 9)
```

2.2. Séparation des données en train/test :

[64]:
```
# Séparer les caractéristiques (X) et la variable cible (y)
X = data.drop(['y'], axis=1)
y = data['y']
```

[65]:
```
from sklearn.model_selection import train_test_split

# Division en ensemble d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

2.3. Entraînement d'un modèle DecisionTreeClassifier :

➢ CART

[122]:
```python
# Entraînement du modèle
model = DecisionTreeClassifier(criterion='gini',max_depth=5,random_state=42)
model.fit(X_train, y_train)

# Prédictions
y_pred = model.predict(X_test)
```

➢ ID3

```python
# Entraînement du modèle
model = DecisionTreeClassifier(criterion='entropy',max_depth=5,random_state=42)
model.fit(X_train, y_train)

# Prédictions
y_pred = model.predict(X_test)
```

2.4. Évaluation du modèle avec une matrice de confusion et un score d'accuracy :

➢ CART

[123]:
```python
# Évaluation du modèle
conf_matrix = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)

print("Matrice de confusion:\n", conf_matrix)
print("Accuracy:", accuracy)
```

```
Matrice de confusion:
 [[1 0 0 0 0 0 0 0 0 1]
 [0 4 0 0 0 0 0 0 0 0]
 [1 0 2 0 0 0 0 0 0 0]
 [1 0 0 3 0 0 0 0 0 0]
 [0 0 0 0 2 0 0 0 0 0]
 [0 0 0 0 0 5 0 0 0 0]
 [0 0 0 0 0 0 8 0 0 0]
 [0 0 0 0 0 0 0 0 0 6]
 [1 0 0 0 0 0 0 0 3 0]
 [0 0 0 0 0 0 0 0 0 2]]
Accuracy: 0.75
```

➢ ID3

```python
# Évaluation du modèle
conf_matrix = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)

print("Matrice de confusion:\n", conf_matrix)
print("Accuracy:", accuracy)
```

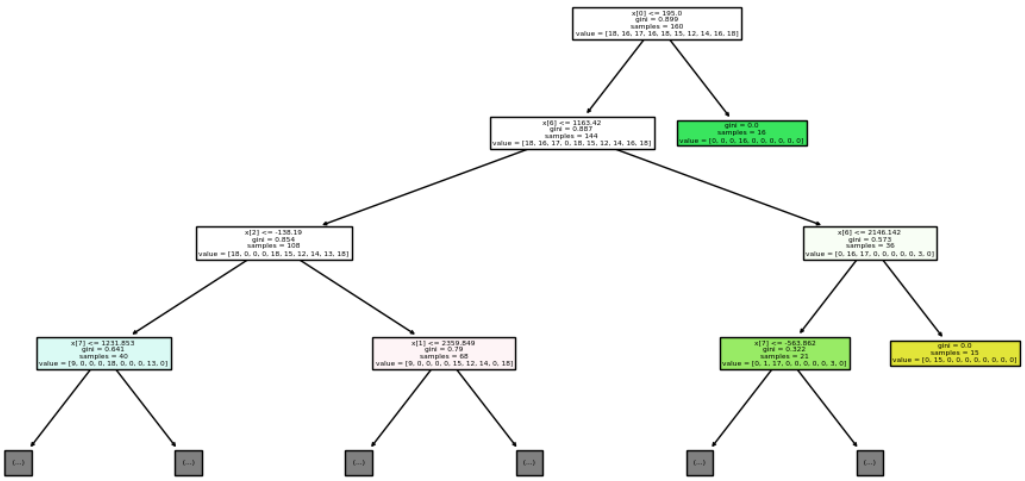
```
Matrice de confusion:
 [[2 0 0 0 0 0 0 0 0 0]
 [0 4 0 0 0 0 0 0 0 0]
 [0 0 2 0 0 0 0 0 0 1]
 [0 0 0 3 0 0 0 0 0 1]
 [0 0 0 0 2 0 0 0 0 0]
 [0 0 0 0 0 4 0 1 0 0]
 [0 0 0 0 0 0 8 0 0 0]
 [0 0 0 0 0 0 0 6 0 0]
 [0 0 0 0 0 0 0 0 3 1]
 [0 0 0 0 0 0 0 0 0 2]]
Accuracy: 0.9
```

Après avoir évalué le module, nous pouvons conclure que l'algorithme ID3 donne une précision plus grande que l'algorithme CART.

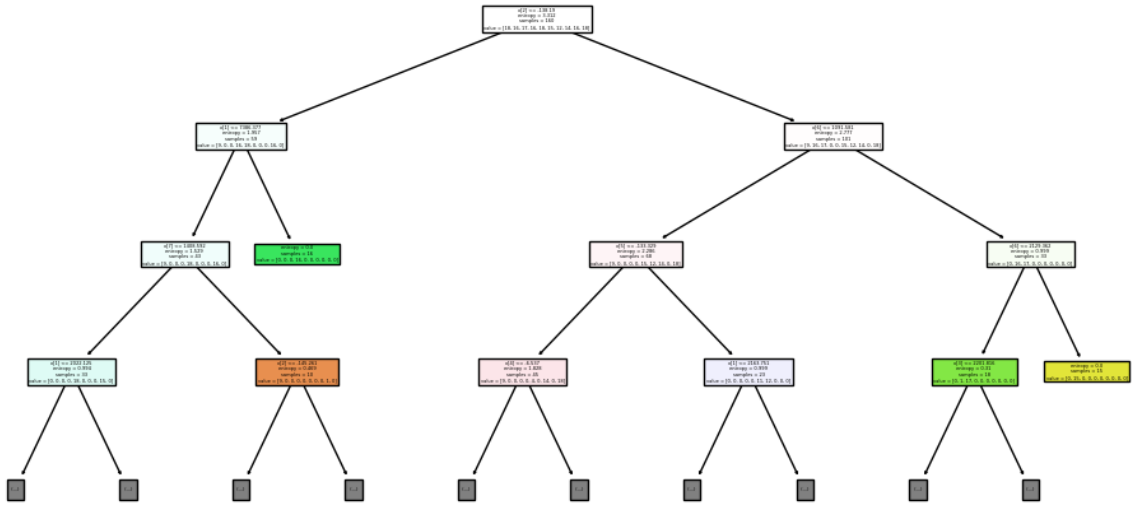
2.5. Visualisation de l'arbre de décision :

➢ CART

```python
# Visualisation de l'arbre de décision
plt.figure(figsize=(12, 6))
plot_tree(model, filled=True, max_depth=3)
plt.show()
```



➢ ID3



- Forets aléatoires :
  - ➢ CART :

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score

# Création du modèle de forêts aléatoires
model = RandomForestClassifier(n_estimators=30, criterion='gini', random_state=42)

# Validation croisée
cv_scores = cross_val_score(model, X, y, cv=3)

# Affichage des résultats
print(f"\nScores de validation croisée : {cv_scores}")
print(f"Moyenne des scores de validation croisée : {cv_scores.mean():.4f}")
```

```
Scores de validation croisée : [1.         0.97014925 0.86363636]
Moyenne des scores de validation croisée : 0.9446
```

  - ➢ ID3

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score

# Création du modèle de forêts aléatoires
model = RandomForestClassifier(n_estimators=30, criterion='entropy', random_state=42)

# Validation croisée
cv_scores = cross_val_score(model, X, y, cv=3)

# Affichage des résultats
print(f"\nScores de validation croisée : {cv_scores}")
print(f"Moyenne des scores de validation croisée : {cv_scores.mean():.4f}")
```

```
Scores de validation croisée : [0.95522388 0.94029851 0.90909091]
Moyenne des scores de validation croisée : 0.9349
```

3. Le jeu de données qualitatif :

Voici le lien du dataset "Mushroom Classification" de Kaggle
https://www.kaggle.com/datasets/uciml/mushroom-classification

Ce dataset contient des caractéristiques de champignons et vise à prédire s'ils sont comestibles ou toxiques. Il contient 23 colonnes : 22 pour les caractéristiques (comme la couleur du chapeau, l'odeur, la texture...) et 1 colonne cible qui indique si le champignon est comestible (e) ou toxique (p). Toutes les variables sont catégoriques, ce qui est important pour l'entraînement du modèle.

- Arbre de décisions :

2.1. Chargement et exploration des données :

[108]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

[109]:
```
# Charger le dataset
df=pd.read_csv('/kaggle/input/mushroom-classification/mushrooms.csv')
```

[110]:
```
# Afficher les premières lignes pour comprendre la structure des données
print(df.head())
```

```
  class cap-shape cap-surface cap-color bruises odor gill-attachment  \
0     p         x           s         n       t    p               f
1     e         x           s         y       t    a               f
2     e         b           s         w       t    l               f
3     p         x           y         w       t    p               f
4     e         x           s         g       f    n               f
```

```
  gill-spacing gill-size gill-color  ... stalk-surface-below-ring  \
0            c         n          k  ...                        s
1            c         b          k  ...                        s
2            c         b          n  ...                        s
3            c         n          n  ...                        s
4            w         b          k  ...                        s

  stalk-color-above-ring stalk-color-below-ring veil-type veil-color  \
0                      w                      w         p          w
1                      w                      w         p          w
2                      w                      w         p          w
3                      w                      w         p          w
4                      w                      w         p          w

  ring-number ring-type spore-print-color population habitat
0           o         p                 k          s       u
1           o         p                 n          n       g
2           o         p                 n          n       m
3           o         p                 k          s       u
4           o         e                 n          a       g

[5 rows x 23 columns]
```

[112]:
```
print(df.columns)
```

```
Index(['class', 'cap-shape', 'cap-surface', 'cap-color', 'bruises', 'odor',
       'gill-attachment', 'gill-spacing', 'gill-size', 'gill-color',
       'stalk-shape', 'stalk-root', 'stalk-surface-above-ring',
       'stalk-surface-below-ring', 'stalk-color-above-ring',
       'stalk-color-below-ring', 'veil-type', 'veil-color', 'ring-number',
       'ring-type', 'spore-print-color', 'population', 'habitat'],
      dtype='object')
```

[114]:
```
# Vérifier les valeurs manquantes
print("\nValeurs manquantes par colonne:")
print(df.isnull().sum())
```

```
Valeurs manquantes par colonne:
class                       0
cap-shape                   0
cap-surface                 0
cap-color                   0
bruises                     0
odor                        0
gill-attachment             0
gill-spacing                0
gill-size                   0
gill-color                  0
stalk-shape                 0
stalk-root                  0
stalk-surface-above-ring    0
stalk-surface-below-ring    0
stalk-color-above-ring      0
stalk-color-below-ring      0
veil-type                   0
veil-color                  0
```

```
ring-number               0
ring-type                 0
spore-print-color         0
population                0
habitat                   0
dtype: int64
```

3.1. Prétraitement des variables catégorielles :

[115]:
```python
# Encodage des variables catégorielles en numériques
label_encoders = {}
for col in df.columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])  # Convertir chaque colonne catégorielle en numérique
    label_encoders[col] = le  # Stocker le label encoder pour une éventuelle future conversion inverse
```

3.2. Séparation des données en train/test :

[116]:
```python
# Séparation des features (X) et de la cible (y)
X = df.drop(columns=['class'])  # Toutes les colonnes sauf la colonne cible
y = df['class']  # La colonne cible indiquant si le champignon est comestible ou toxique

# Séparation des données en ensemble d'entraînement et de test (70% - 30%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

3.3. Entraînement d'un modèle DecisionTreeClassifier :

➢ CART :

[117]:
```python
# Création et entraînement du modèle d'arbre de décision
model1 = DecisionTreeClassifier(criterion='gini', max_depth=5, random_state=42)

model1.fit(X_train, y_train)

# Prédiction sur l'ensemble de test
y_pred1 = model1.predict(X_test)
```

➢ ID3 :

[118]:
```python
# Création et entraînement du modèle d'arbre de décision en utilisant ID3
model2 = DecisionTreeClassifier(criterion='entropy', max_depth=5, random_state=42)
model2.fit(X_train, y_train)

# Prédiction sur l'ensemble de test
y_pred2 = model2.predict(X_test)
```

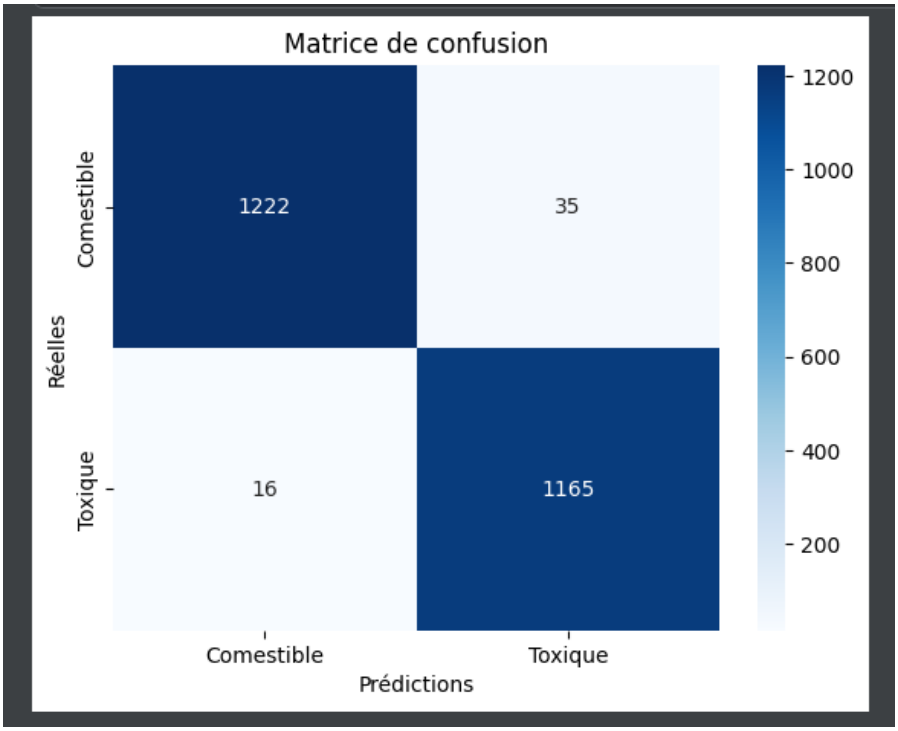
3.4. Évaluation du modèle avec une matrice de confusion et un score d'accuracy :

➢ CART :

```
# Évaluation du premier modele
accuracy = accuracy_score(y_test, y_pred1)
print(f"\nAccuracy du premier modèle : {accuracy:.4f}")
```

```
Accuracy du premier modèle : 0.9791
```

```
[120]: # Matrice de confusion du premier modele
conf_matrix = confusion_matrix(y_test, y_pred1)
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Comestible', 'Toxique'], yticklabel
plt.xlabel('Prédictions')
plt.ylabel('Réelles')
plt.title('Matrice de confusion')
plt.show()
```



[121]:
```
# Rapport de classification du premier modele
print("\nRapport de classification:")
print(classification_report(y_test, y_pred1))
```

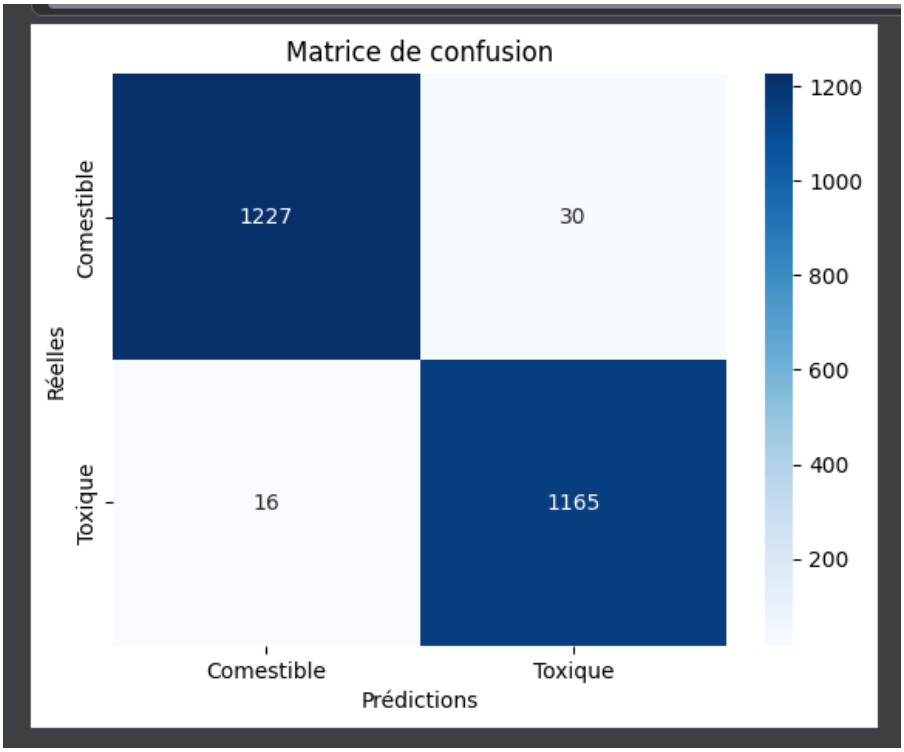
```
Rapport de classification:
              precision    recall  f1-score   support

           0       0.99      0.97      0.98      1257
           1       0.97      0.99      0.98      1181

    accuracy                           0.98      2438
   macro avg       0.98      0.98      0.98      2438
weighted avg       0.98      0.98      0.98      2438
```

➢ ID3 :

[122]:
```
# Évaluation du deuxiemme modèle
accuracy = accuracy_score(y_test, y_pred2)
print(f"\nAccuracy du deuxiemme modèle : {accuracy:.4f}")
```

```
Accuracy du deuxiemme modèle : 0.9811
```

[123]:
```
# Matrice de confusion du deuxiemme modele
conf_matrix = confusion_matrix(y_test, y_pred2)
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Comestible', 'Toxique'], yticklabel
plt.xlabel('Prédictions')
plt.ylabel('Réelles')
plt.title('Matrice de confusion')
plt.show()
```



[124]:
```
# Rapport de classification du deuxiemme modele
print("\nRapport de classification:")
print(classification_report(y_test, y_pred2))
```

```
Rapport de classification:
              precision    recall  f1-score   support

           0       0.99      0.98      0.98      1257
           1       0.97      0.99      0.98      1181

    accuracy                           0.98      2438
   macro avg       0.98      0.98      0.98      2438
weighted avg       0.98      0.98      0.98      2438
```

Une **accuracy élevée (>97%)** indique que **CART et ID3** sont bien adaptés à votre jeu de données qualitatif.

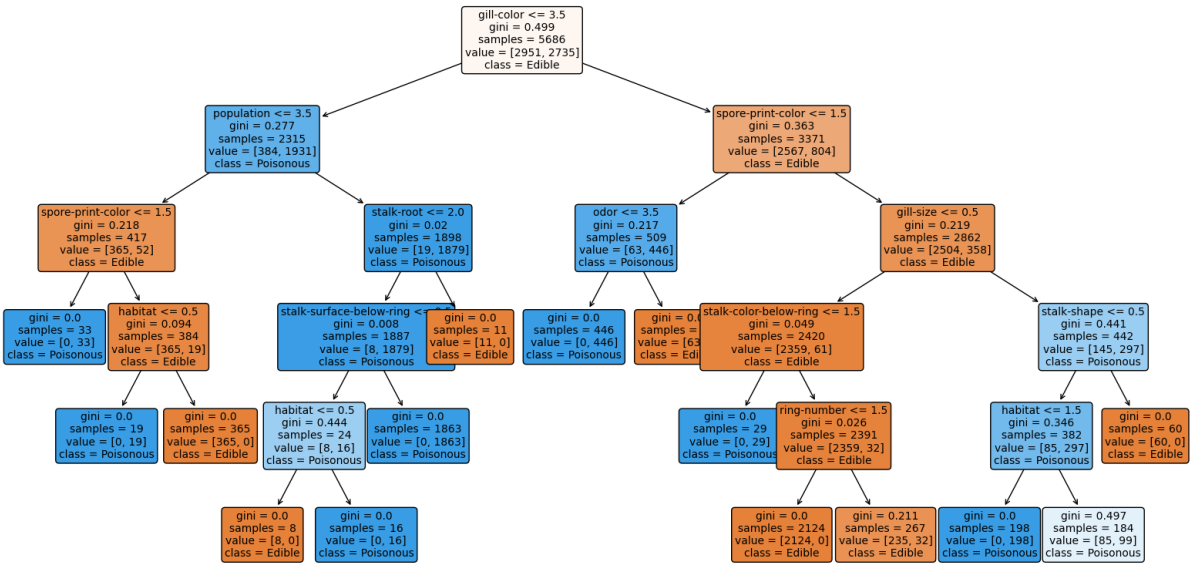
Cela signifie que les **variables catégorielles sont bien discriminantes** et permettent de classer correctement les observations.

Légère différence : **0.9811 (ID3) - 0.9791 (CART) = 0.002**
Puisque on travaille avec **des variables catégorielles**, ID3 est souvent un bon choix car il fonctionne uniquement avec ces types de données, tandis que CART est plus flexible et peut traiter à la fois **des données numériques et qualitatives**.

3.5. Visualisation de l'arbre de décision :

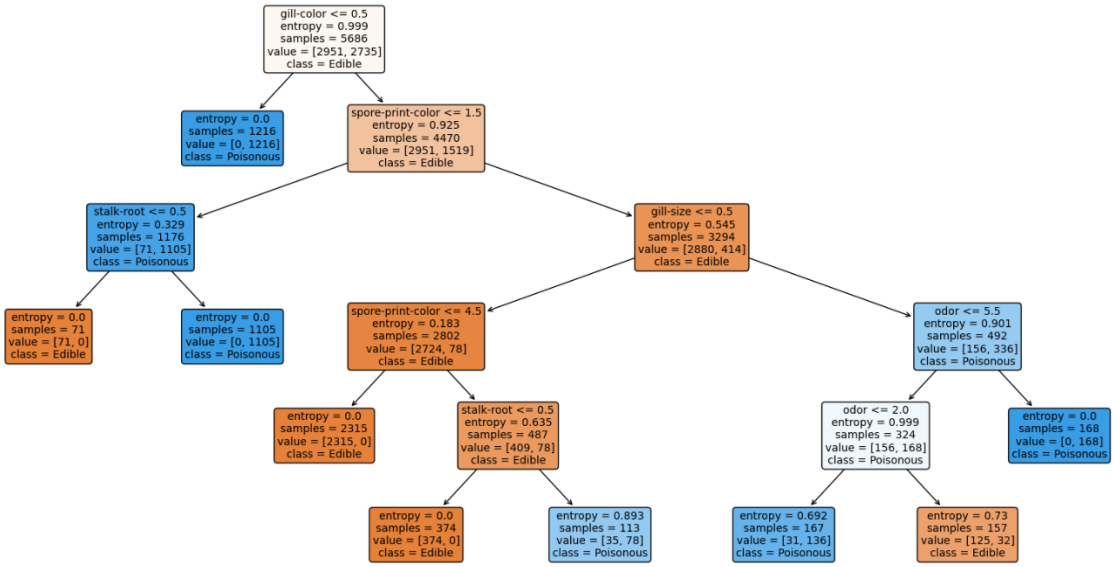
➢ CART :

[125]:
```
# Visualisation de l'arbre de décision du premier modele
plt.figure(figsize=(20,10))
plot_tree(model1, feature_names=X.columns, class_names=['Edible', 'Poisonous'], filled=True, rounded=True, fo
plt.show()
```



➢ ID3 :

```
[126]: # Visualisation de l'arbre de décision du deuxiemme modele
plt.figure(figsize=(20,10))
plot_tree(model2, feature_names=X.columns, class_names=['Edible', 'Poisonous'], filled=True, rounded=True, fo
plt.show()
```



Structure	ID3	CART
Type de division	Multi-division (chaque nœud peut avoir plusieurs branches).	Binaire (chaque nœud a **seulement** deux branches).
Complexité	Peut produire un arbre plus profond et complexe.	Produit souvent un arbre plus compact et équilibré.

- Forets aléatoires :
  - ➢ CART :

```python
from sklearn.ensemble import RandomForestClassifier

# Création et entraînement du modèle de forêts aléatoires avec le critère 'entropy'
model3 = RandomForestClassifier(n_estimators=100, criterion='gini', random_state=42)
model3.fit(X_train, y_train)

# Prédiction sur l'ensemble de test
y_pred3 = model3.predict(X_test)

# Évaluation du modèle
accuracy = accuracy_score(y_test, y_pred3)
print(f"\nAccuracy du modèle : {accuracy:.4f}")

# Rapport de classification
print("\nRapport de classification:")
print(classification_report(y_test, y_pred3))

train_accuracy = accuracy_score(y_train, model3.predict(X_train))
test_accuracy = accuracy_score(y_test, y_pred3)
print(f"Accuracy entraînement : {train_accuracy:.4f}")
print(f"Accuracy test : {test_accuracy:.4f}")
```

```
Accuracy du modèle : 0.9274

Rapport de classification:
                    precision    recall  f1-score   support

Insufficient_Weight       0.95      0.93      0.94        86
      Normal_Weight       0.77      0.89      0.83        93
     Obesity_Type_I       0.97      0.94      0.96       102
    Obesity_Type_II       0.97      0.99      0.98        88
   Obesity_Type_III       1.00      0.99      0.99        98
 Overweight_Level_I       0.93      0.84      0.88        88
Overweight_Level_II       0.93      0.90      0.92        79

           accuracy                           0.93       634
          macro avg       0.93      0.93      0.93       634
       weighted avg       0.93      0.93      0.93       634

Accuracy entraînement : 1.0000
Accuracy test : 0.9274
```

Le modèle sur-apprend (overfitting), car il obtient 100% de précision sur les données d'entraînement. Cela signifie qu'il a mémorisé les données au lieu de généraliser.

Solution:

La validation croisée (cross-validation) permet de tester le modèle sur différents sous-ensembles des données pour s'assurer qu'il se généralise bien. Cela peut aider à éviter un surapprentissage lié à la sélection aléatoire d'un ensemble d'entraînement.

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score

# Création du modèle de forêts aléatoires
model3 = RandomForestClassifier(n_estimators=100, criterion='gini', random_state=42)

# Validation croisée (en supposant que X et y sont bien définis)
cv_scores = cross_val_score(model3, X, y, cv=3)

# Affichage des résultats
print(f"\nScores de validation croisée : {cv_scores}")
print(f"Moyenne des scores de validation croisée : {cv_scores.mean():.4f}")
```

```
Scores de validation croisée : [0.90546529 0.99113737 0.71935007]
Moyenne des scores de validation croisée : 0.8720
```

➢ ID3 :

```
[129]:
```

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score

# Création du modèle de forêts aléatoires
model4 = RandomForestClassifier(n_estimators=100, criterion='entropy', random_state=42)

# Validation croisée (en supposant que X et y sont bien définis)
cv_scores = cross_val_score(model4, X, y, cv=3)

# Affichage des résultats
print(f"\nScores de validation croisée : {cv_scores}")
print(f"Moyenne des scores de validation croisée : {cv_scores.mean():.4f}")
```

```
Scores de validation croisée : [0.90546529 0.99187592 0.73412112]
Moyenne des scores de validation croisée : 0.8772
```

Les algorithmes CART et ID3 donnent des performances très proches.

4. Le jeu de données quantitatif :

Voici le lien du dataset " Obesity Prediction Dataset " de Kaggle

https://www.kaggle.com/datasets/adeniranstephen/obesity-prediction-dataset

Le Obesity Prediction Dataset disponible sur Kaggle est conçu pour estimer les niveaux d'obésité en fonction des habitudes alimentaires, des antécédents familiaux et de la condition physique des individus. Ce jeu de données comprend des informations provenant de personnes originaires du Mexique, du Pérou et de la Colombie.

Description du jeu de données : Le jeu de données contient 17 attributs et 2111 enregistrements. Les enregistrements sont étiquetés avec la variable de classe "NObesity" (niveau d'obésité), qui permet de classer les données selon les valeurs suivantes :

Insufficient Weight Normal Weight Overweight Level I Overweight Level II Obesity Type I Obesity Type II Obesity Type III

Les attributs incluent des informations sur les habitudes alimentaires, l'activité physique, les antécédents familiaux et d'autres facteurs liés à la santé.

- Arbre de décisions

4.1. Chargement et exploration des données :

[130]:
```
# Charger le dataset
df2=pd.read_csv('/kaggle/input/obesity-prediction-dataset/ObesityDataSet_raw_and_data_sinthetic.csv')
```

[131]:
```
# Afficher les premières lignes pour comprendre la structure des données
print(df2.head())
```

```
   Gender  Age  Height  Weight family_history_with_overweight FAVC  FCVC  NCP  \
0  Female   21    1.62    64.0                            yes   no   2.0  3.0
1  Female   21    1.52    56.0                            yes   no   3.0  3.0
2    Male   23    1.80    77.0                            yes   no   2.0  3.0
3    Male   27    1.80    87.0                             no   no   3.0  3.0
4    Male   22    1.78    89.8                             no   no   2.0  1.0

        CAEC SMOKE  CH2O  SCC  FAF  TUE        CALC                MTRANS  \
0  Sometimes    no   2.0   no  0.0  1.0          no  Public_Transportation
1  Sometimes   yes   3.0  yes  3.0  0.0   Sometimes  Public_Transportation
2  Sometimes    no   2.0   no  2.0  1.0  Frequently  Public_Transportation
3  Sometimes    no   2.0   no  2.0  0.0  Frequently                Walking
4  Sometimes    no   2.0   no  0.0  0.0   Sometimes  Public_Transportation

           NObeyesdad
0       Normal_Weight
1       Normal_Weight
2       Normal_Weight
3  Overweight_Level_I
4 Overweight_Level_II
```

[133]:
```
print(df2.columns)
```

```
Index(['Gender', 'Age', 'Height', 'Weight', 'family_history_with_overweight',
       'FAVC', 'FCVC', 'NCP', 'CAEC', 'SMOKE', 'CH2O', 'SCC', 'FAF', 'TUE',
       'CALC', 'MTRANS', 'NObeyesdad'],
      dtype='object')
```

[135]:
```
# Vérifier les valeurs manquantes
print("\nValeurs manquantes par colonne:")
print(df2.isnull().sum())
```

```
Valeurs manquantes par colonne:
Gender                            0
Age                               0
Height                            0
Weight                            0
family_history_with_overweight    0
FAVC                              0
FCVC                              0
NCP                               0
CAEC                              0
SMOKE                             0
CH2O                              0
SCC                               0
FAF                               0
TUE                               0
CALC                              0
MTRANS                            0
NObeyesdad                        0
dtype: int64
```

4.2. Prétraitement des variables catégorielles :

[136]:
```
print(df2.dtypes)
```

```
Gender                            object
Age                                int64
Height                           float64
Weight                           float64
family_history_with_overweight    object
FAVC                              object
FCVC                             float64
NCP                              float64
CAEC                              object
SMOKE                             object
CH2O                             float64
SCC                               object
FAF                              float64
TUE                              float64
CALC                              object
MTRANS                            object
NObeyesdad                        object
dtype: object
```

[137]:
```
from sklearn.preprocessing import OneHotEncoder

# Sélection des colonnes catégorielles
categorical_columns = ['Gender', 'family_history_with_overweight', 'FAVC', 'CAEC', 'SMOKE', 'SCC', 'CALC', 'MT

# Appliquer One-Hot Encoding
data = pd.get_dummies(df2, columns=categorical_columns, drop_first=True)

print(data.head())
```

```
   Age  Height  Weight  FCVC  NCP  CH2O  FAF  TUE           NObeyesdad  \
0   21    1.62    64.0   2.0  3.0   2.0  0.0  1.0        Normal_Weight
1   21    1.52    56.0   3.0  3.0   3.0  3.0  0.0        Normal_Weight
2   23    1.80    77.0   2.0  3.0   2.0  2.0  1.0        Normal_Weight
3   27    1.80    87.0   3.0  3.0   2.0  2.0  0.0   Overweight_Level_I
4   22    1.78    89.8   2.0  1.0   2.0  0.0  0.0  Overweight_Level_II

   Gender_Male  ...  CAEC_no  SMOKE_yes  SCC_yes  CALC_Frequently  \
0        False  ...    False      False    False            False
1        False  ...    False       True     True            False
2         True  ...    False      False    False             True
3         True  ...    False      False    False             True
4         True  ...    False      False    False            False

   CALC_Sometimes  CALC_no  MTRANS_Bike  MTRANS_Motorbike  \
0           False     True        False             False
1            True    False        False             False
2           False    False        False             False
3           False    False        False             False
4            True    False        False             False

   MTRANS_Public_Transportation  MTRANS_Walking
0                          True           False
1                          True           False
2                          True           False
3                         False            True
```

Certaines variables peuvent avoir des échelles différentes, donc on normalise

```
[138]: from sklearn.preprocessing import StandardScaler

       # Sélection des colonnes numériques
       numerical_columns = ['Age', 'Height', 'Weight', 'FCVC', 'NCP', 'CH2O', 'FAF', 'TUE']

       # Appliquer la normalisation
       scaler = StandardScaler()
       data[numerical_columns] = scaler.fit_transform(data[numerical_columns])

       print(data.head())
```

```
        Age    Height    Weight      FCVC       NCP      CH2O       FAF  \
0 -0.521741 -0.874380 -0.862558 -0.784810  0.404102 -0.013141 -1.188028
1 -0.521741 -1.945660 -1.168077  1.088307  0.404102  1.618701  2.339676
2 -0.207057  1.053924 -0.366089 -0.784810  0.404102 -0.013141  1.163774
3  0.422312  1.053924  0.015809  1.088307  0.404102 -0.013141  1.163774
4 -0.364399  0.839668  0.122741 -0.784810 -2.166941 -0.013141 -1.188028
```

```
        TUE           NObeyesdad  Gender_Male  ...  CAEC_no  SMOKE_yes  \
0  0.562005        Normal_Weight        False  ...    False      False
1 -1.080619        Normal_Weight        False  ...    False       True
2  0.562005        Normal_Weight         True  ...    False      False
3 -1.080619   Overweight_Level_I         True  ...    False      False
4 -1.080619  Overweight_Level_II         True  ...    False      False

   SCC_yes  CALC_Frequently  CALC_Sometimes  CALC_no  MTRANS_Bike  \
0    False           False           False     True        False
1     True           False            True    False        False
2    False            True           False    False        False
3    False            True           False    False        False
4    False           False            True    False        False

   MTRANS_Motorbike  MTRANS_Public_Transportation  MTRANS_Walking
0             False                          True           False
1             False                          True           False
2             False                          True           False
3             False                         False            True
4             False                          True           False

[5 rows x 24 columns]
```

4.3. Séparation des données en train/test :

[139]:
```python
# Séparation des attributs et de la variable cible
X = data.drop('NObeyesdad', axis=1)
y = data['NObeyesdad']

# Division en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

4.4. Entraînement d'un modèle DecisionTreeClassifier :

➢ CART

[140]:
```python
# Initialisation et entraînement de l'arbre de décision
clf = DecisionTreeClassifier(criterion='gini',max_depth=5, random_state=42)
clf.fit(X_train, y_train)

# Prédictions
y_pred1 = clf.predict(X_test)
```

➢ ID3

[141]:
```python
# Initialisation et entraînement de l'arbre de décision
clf2 = DecisionTreeClassifier(criterion='entropy',max_depth=5, random_state=42)
clf2.fit(X_train, y_train)

# Prédictions
y_pred2 = clf2.predict(X_test)
```

4.5. Évaluation du modèle avec une matrice de confusion et un score d'accuracy :

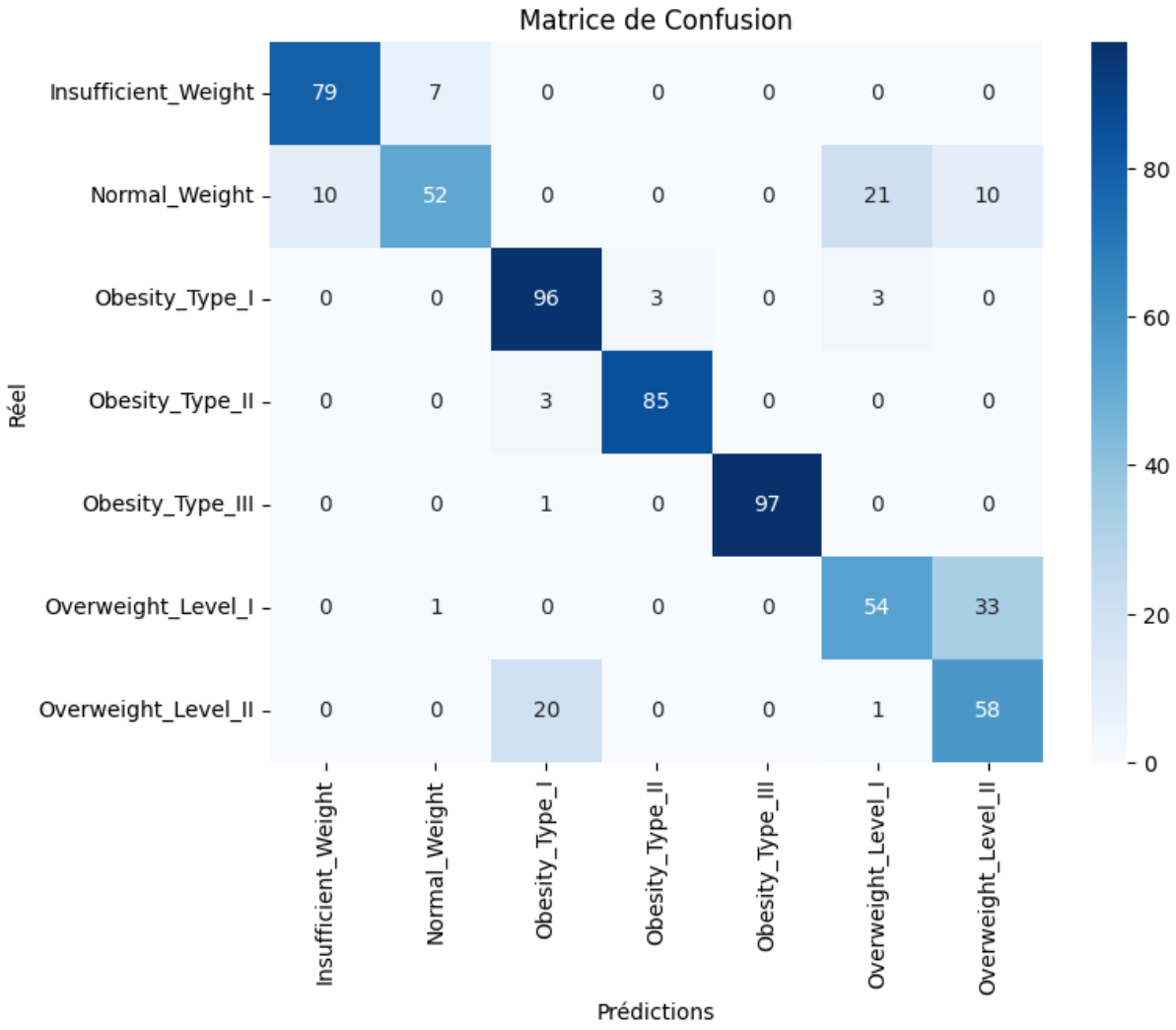
➢ CART

[142]:
```python
# Évaluation du premier modele
accuracy = accuracy_score(y_test, y_pred1)
print(f"\nAccuracy du premier modèle : {accuracy:.4f}")
```

```
Accuracy du premier modèle : 0.8218
```

[143]:

```python
# Créer la matrice de confusion
cm = confusion_matrix(y_test, y_pred1)

# Affichage avec Seaborn
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=clf.classes_, yticklabels=clf.classes_)
plt.xlabel("Prédictions")
plt.ylabel("Réel")
plt.title("Matrice de Confusion")
plt.show()
```



```python
# Rapport de classification du premier modele
print("\nRapport de classification:")
print(classification_report(y_test, y_pred1))
```

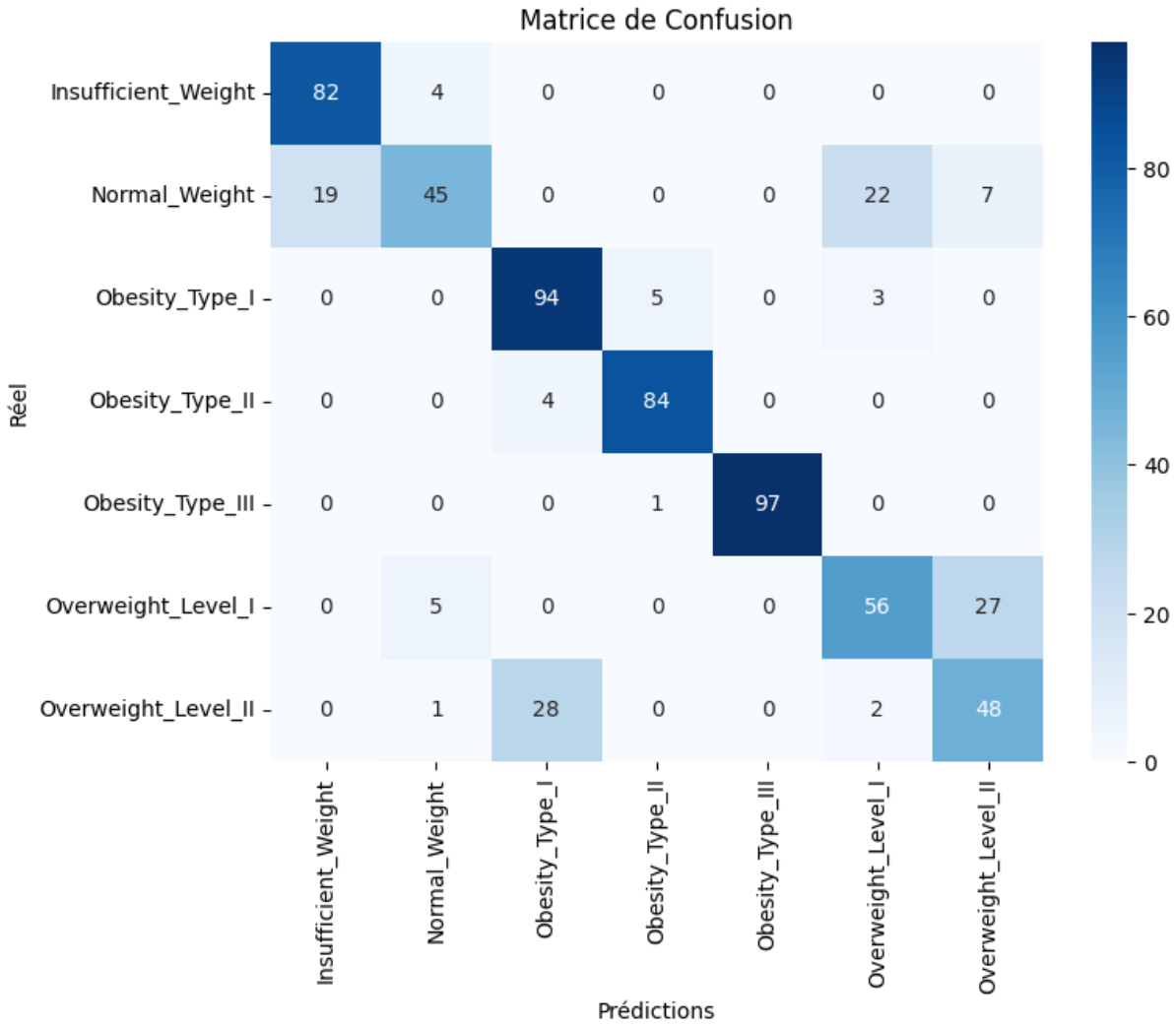
```
Rapport de classification:
                    precision    recall  f1-score   support

Insufficient_Weight       0.89      0.92      0.90        86
      Normal_Weight       0.87      0.56      0.68        93
     Obesity_Type_I       0.80      0.94      0.86       102
    Obesity_Type_II       0.97      0.97      0.97        88
   Obesity_Type_III       1.00      0.99      0.99        98
 Overweight_Level_I       0.68      0.61      0.65        88
Overweight_Level_II       0.57      0.73      0.64        79

           accuracy                           0.82       634
          macro avg       0.83      0.82      0.81       634
       weighted avg       0.83      0.82      0.82       634
```

➤ ID3

```python
# Évaluation du premier modele
accuracy = accuracy_score(y_test, y_pred2)
print(f"\nAccuracy du premier modèle : {accuracy:.4f}")
```

```
Accuracy du premier modèle : 0.7981
```

```python
# Créer la matrice de confusion
cm = confusion_matrix(y_test, y_pred2)

# Affichage avec Seaborn
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=clf.classes_, yticklabels=clf.classes_)
plt.xlabel("Prédictions")
plt.ylabel("Réel")
plt.title("Matrice de Confusion")
plt.show()
```




```
[147]: # Rapport de classification du premier modele
       print("\nRapport de classification:")
       print(classification_report(y_test, y_pred2))

Rapport de classification:
                     precision    recall  f1-score   support

Insufficient_Weight       0.81      0.95      0.88        86
      Normal_Weight       0.82      0.48      0.61        93
     Obesity_Type_I       0.75      0.92      0.82       102
    Obesity_Type_II       0.93      0.95      0.94        88
   Obesity_Type_III       1.00      0.99      0.99        98
 Overweight_Level_I       0.67      0.64      0.65        88
Overweight_Level_II       0.59      0.61      0.60        79

           accuracy                           0.80       634
          macro avg       0.80      0.79      0.79       634
       weighted avg       0.80      0.80      0.79       634
```

ID3 ne peut **pas** gérer directement les variables numériques, il doit d'abord **les convertir en catégories**. Cela **perd de l'information** et réduit la précision.

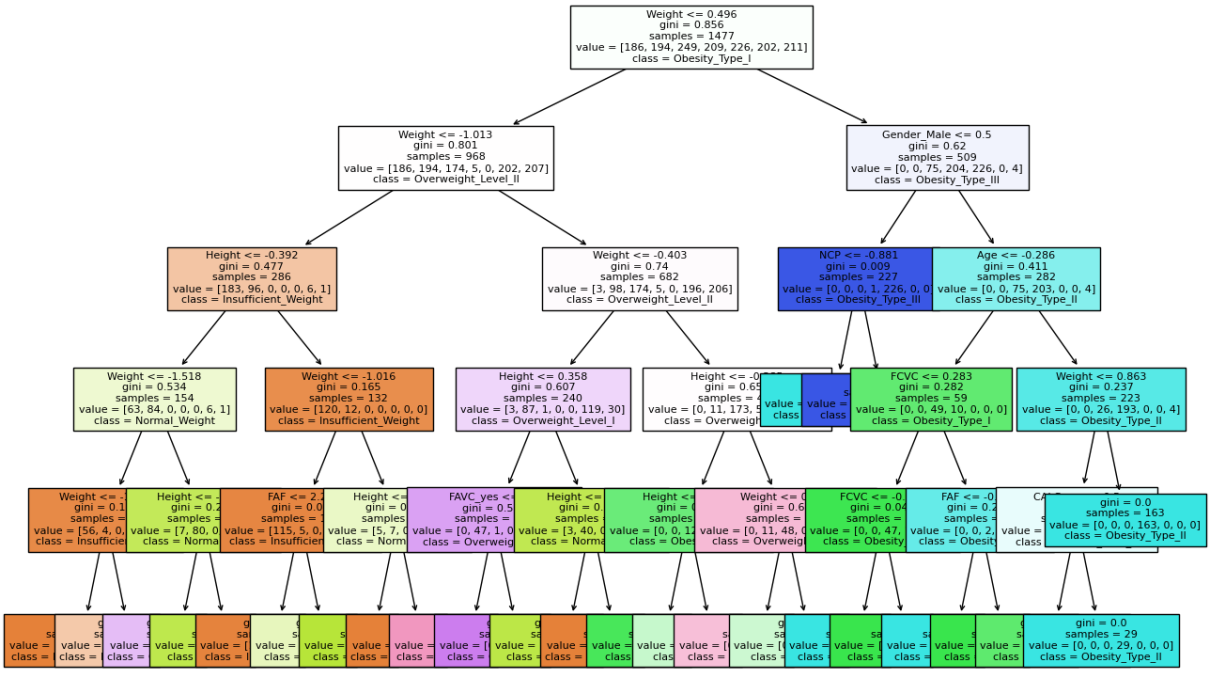
CART, en revanche, peut directement travailler avec les valeurs continues et trouver les seuils optimaux.

Conclusion principale : **CART est plus adapté aux données quantitatives**, car il peut gérer directement les valeurs continues, choisir les seuils optimaux. **ID3 fonctionne mieux avec des variables qualitatives** mais est moins efficace pour les données numériques.

4.6. Visualisation de l'arbre de décision :

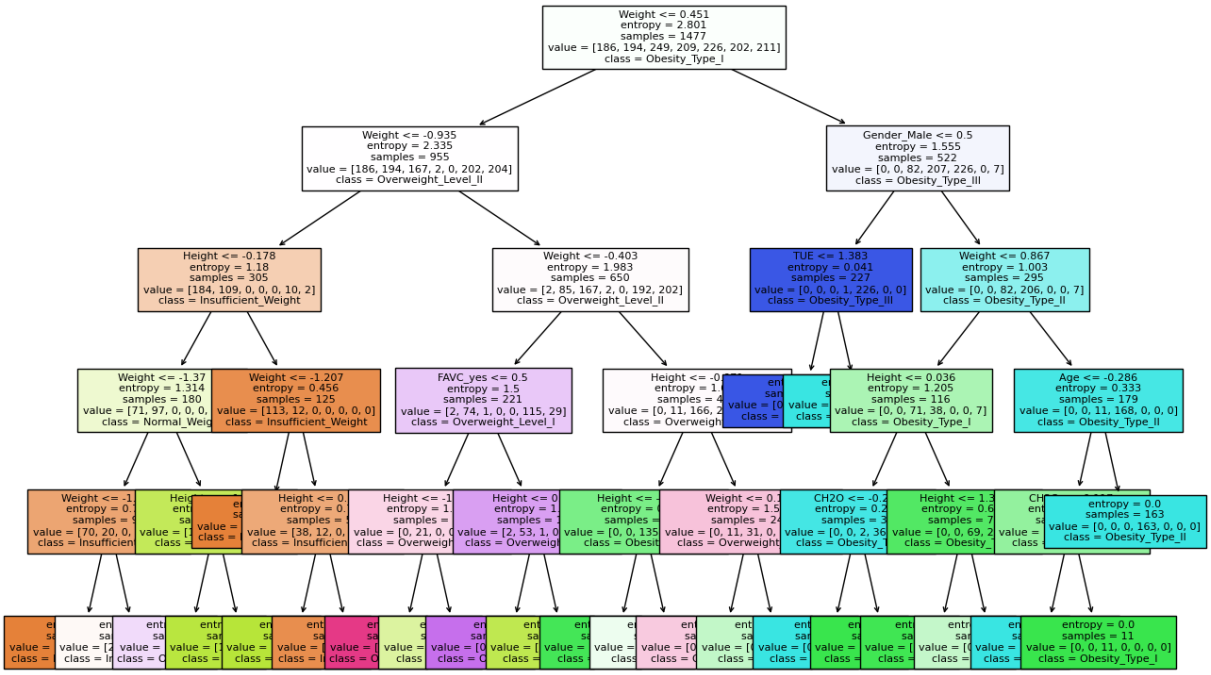
➢ CART

```
[148]: # Affichage de l'arbre de décision
       plt.figure(figsize=(15, 10))
       plot_tree(clf, feature_names=X.columns, class_names=clf.classes_, filled=True, fontsize=8)
       plt.show()
```



➢ ID3

```
[149]: # Affichage de l'arbre de décision
       plt.figure(figsize=(15, 10))
       plot_tree(clf2, feature_names=X.columns, class_names=clf2.classes_, filled=True, fontsize=8)
       plt.show()
```



- Forets aléatoires
  - ➢ CART

```
[150]: from sklearn.ensemble import RandomForestClassifier
       from sklearn.model_selection import cross_val_score

       # Création du modèle de forêts aléatoires
       clf3 = RandomForestClassifier(n_estimators=100, criterion='gini', random_state=42)

       # Validation croisée (en supposant que X et y sont bien définis)
       cv_scores = cross_val_score(clf3, X, y, cv=3)

       # Affichage des résultats
       print(f"\nScores de validation croisée : {cv_scores}")
       print(f"Moyenne des scores de validation croisée : {cv_scores.mean():.4f}")

Scores de validation croisée : [0.80539773 0.97585227 0.98150782]
Moyenne des scores de validation croisée : 0.9209
```

  - ➢ ID3

```
[151]: from sklearn.ensemble import RandomForestClassifier
       from sklearn.model_selection import cross_val_score

       # Création du modèle de forêts aléatoires
       clf4 = RandomForestClassifier(n_estimators=100, criterion='entropy', random_state=42)

       # Validation croisée (en supposant que X et y sont bien définis)
       cv_scores = cross_val_score(clf4, X, y, cv=3)

       # Affichage des résultats
       print(f"\nScores de validation croisée : {cv_scores}")
       print(f"Moyenne des scores de validation croisée : {cv_scores.mean():.4f}")

Scores de validation croisée : [0.82954545 0.98011364 0.98008535]
Moyenne des scores de validation croisée : 0.9299
```