Machine Learning

Projet de TP:

Détection d'adulération des huiles d'olive à l'aide de classificateurs d'apprentissage automatique: Optimisation des coûts et du temps

IIA4

Option ISI

Elaboré par :

DJOBBI Amer (Groupe 2)

KHECHINE Elyes (Groupe 1)

2022-2023

I. Introduction & But du projet:

Ce projet vise à développer un système de détection d'adulération des huiles d'olive en utilisant des classificateurs d'apprentissage automatique. L'objectif principal est de réduire à la fois les coûts et le temps associés à la détection de l'adultération, qui est actuellement réalisée à l'aide d'un test chimique coûteux et chronophage.

Pour atteindre cet objectif, nous avons collecté des données brutes à partir d'huiles d'olive. Notre projet consiste à entraîner et évaluer différents classificateurs sur ces données, qui sont sous la forme de séries temporelles de mesures d'absorbance pour 4 régions différentes.

L'objectif spécifique de ce projet est de prédire la région correspondante à chaque série temporelle d'absorbance en utilisant les différentes mesures disponibles. Pour ce faire, nous avons utilisé une variété de classificateurs d'apprentissage automatique tels que Naive Bayes, K-Nearest Neighbors, Linear Discriminant Analysis, Decision Tree, Artificial Neural Networks, SVM, Gaussian Process, Random Forest, AdaBoost et Quadratic Discriminant Analysis.

Chaque classificateur a été entraîné sur les données d'entraînement et sa précision a été évaluée sur les données de test. En utilisant les résultats, nous avons comparé les performances de chaque classificateur en termes de précision de prédiction. Un graphe a été généré pour visualiser les labels prédits par rapport aux labels réels pour chaque classe.

Enfin, en se basant sur les performances obtenues, notre algorithme sélectionne le classificateur ayant atteint la précision maximale pour la prédiction de la région d'origine de l'huile d'olive.

II. Classificateurs utilisés

Naive Bayes : il s'agit d'un classificateur probabiliste basé sur le théorème de Bayes, qui suppose que toutes les variables sont indépendantes les unes des autres. Il est souvent utilisé pour la classification de texte ou pour des problèmes où il y a peu de variables prédictives.

K-Nearest Neighbors (K-NN) : c'est un classificateur non paramétrique qui utilise les k échantillons les plus proches dans l'espace de caractéristiques pour prédire la classe d'un nouvel échantillon. Il est souvent utilisé pour les problèmes de classification où les données sont bien séparées.

Linear Discriminant Analysis (LDA) : c'est un classificateur linéaire qui cherche à séparer les classes en utilisant des combinaisons linéaires de variables prédictives. Il suppose que les variables sont distribuées normalement et que les variances sont égales.

Decision Tree : c'est un classificateur basé sur un arbre de décision, qui divise récursivement les données en fonction des caractéristiques pour atteindre la décision finale. Il est souvent utilisé pour des problèmes de classification avec des données discrètes.

Artificial Neural Networks (ANN) : il s'agit d'un classificateur basé sur un réseau de neurones artificiels, qui cherche à apprendre les relations non linéaires entre les variables prédictives et la variable de sortie. Il est souvent utilisé pour des problèmes de classification avec des données continues ou des images.

SVM : c'est un classificateur qui cherche à trouver l'hyperplan qui sépare les classes avec la plus grande marge possible. Il est souvent utilisé pour les problèmes de classification où les données ne sont pas linéairement séparables.

Gaussian Process : c'est un classificateur qui utilise un processus gaussien pour modéliser la distribution des données. Il est souvent utilisé pour les problèmes de classification avec des données continues.

Random Forest : c'est un classificateur basé sur un ensemble d'arbres de décision, qui cherche à réduire le surapprentissage en utilisant des sous-ensembles aléatoires des données pour construire chaque arbre. Il est souvent utilisé pour des problèmes de classification avec des données discrètes.

AdaBoost : c'est un classificateur basé sur un ensemble de classificateurs faibles, qui sont pondérés en fonction de leur précision pour construire un classificateur fort. Il est souvent utilisé pour des problèmes de classification avec des données discrètes.

Quadratic Discriminant Analysis (QDA) : c'est un classificateur qui cherche à séparer les classes en utilisant des combinaisons quadratiques de variables prédictives. Il suppose que les variables sont distribuées normalement mais les variances peuvent être différentes. Il est souvent utilisé pour des problèmes de classification avec des données continues.

III. L'algorithme:

```
!pip install liac-arff
```

```
# Importation des bibliothèques nécessaires
import urllib.request
import numpy as np
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.gaussian_process.kernels import RBF
import arff
```

```
# Téléchargement du fichier OliveOil_TRAIN.arff depuis le site http://www.timeseriesclassification.com/
# et enregistrement sous le nom OliveOil.zip

urllib.request.urlretrieve("http://www.timeseriesclassification.com/Downloads/OliveOil.zip", "OliveOil.zip")

# Décompression du fichier OliveOil.zip
!unzip OliveOil.zip
```

```python
# Chargement du fichier ARFF contenant les données d'entraînement
with open('OliveOil_TRAIN.arff') as f:
    dataset = arff.load(f)

# Extraction des données et des métadonnées
data = np.array(dataset['data'], dtype=np.float64)
meta = dataset['attributes']

# Séparation des données en features (X) et labels (y)
X = data[:,:-1]
y = data[:,-1]
```

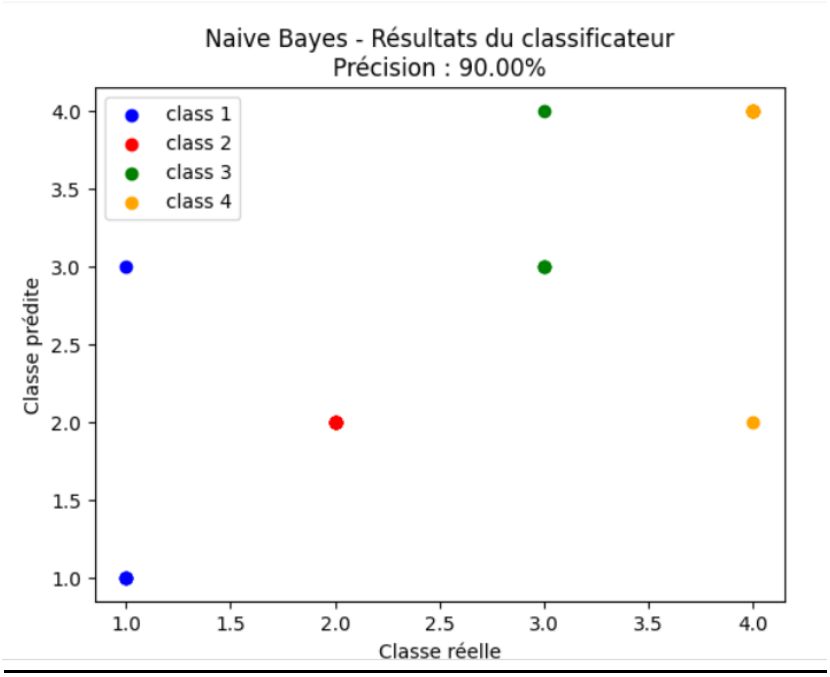
```python
# Définition des différents classificateurs à tester
classifiers = [
    ("Naive Bayes", GaussianNB()),

    ("K-Nearest Neighbors (DEFAULT K=5)", KNeighborsClassifier()),
    ("K-Nearest Neighbors (K=3)", KNeighborsClassifier(3)),

    ("Linear Discriminant Analysis", LinearDiscriminantAnalysis()),

    ("Decision Tree", DecisionTreeClassifier()),
    ("Decision Tree (max_depth=4)", DecisionTreeClassifier(max_depth=4)),

    ("Artificial Neural Networks (DEFAULT: alpha=0.0001 max_iter=200) ", MLPClassifier()),
    ("Artificial Neural Networks (alpha=0.0000000001, max_iter=1000)", MLPClassifier(alpha=0.0000000001, max_iter=1000)),
```

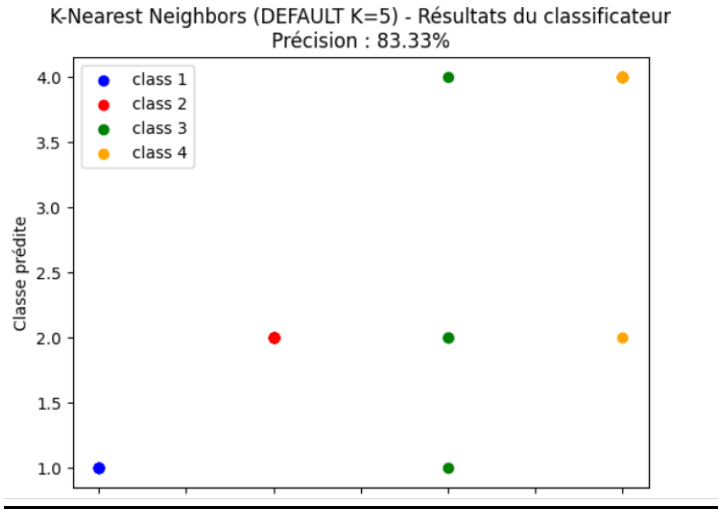
```python
    ("RBF SVM (DEFAULT gamma='scale', C=1)", SVC()),
    ("RBF SVM (gamma=15, C=4)", SVC(gamma=15, C=3)),

    ("Linear SVM (DEFAULT gamma='scale', C=1)", SVC(kernel="linear")),
    ("Linear SVM (gamma=15, C=25)", SVC(kernel="linear", gamma=15, C=25)),

    ("Poly SVM (DEFAULT gamma='scale', C=1)", SVC(kernel="poly")),
    ("Poly SVM (gamma=15, C=3)", SVC(kernel="poly", gamma=15, C=3)),

    ("Sigmoid SVM", SVC(kernel="sigmoid")),
    ("Gaussian Process", GaussianProcessClassifier(1.0 * RBF(1.0))),
    ("Random Forest", RandomForestClassifier(max_depth=5, n_estimators=10, max_features=570)),
    ("AdaBoost", AdaBoostClassifier()),
    ("QDA", QuadraticDiscriminantAnalysis())
]
```

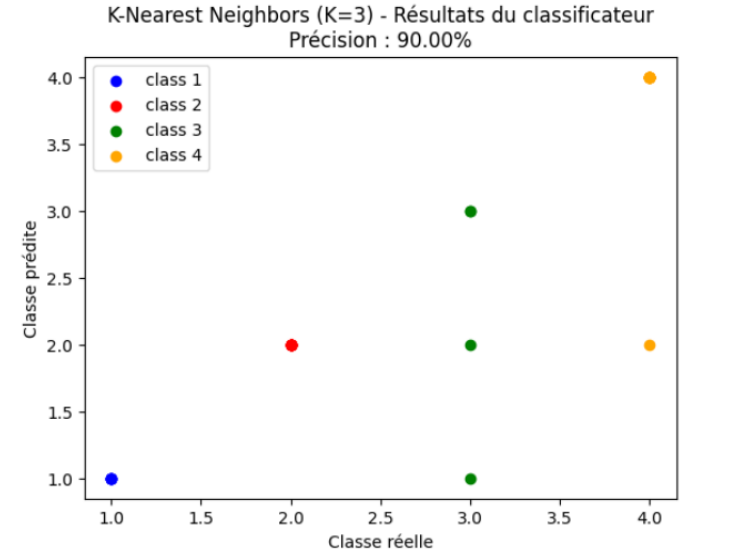
```
# Entrainement de chaque classificateur et affichage des résultats
max_accuracy = 0  # Initialisation de la variable contenant la précision maximale obtenue
for name, model in classifiers:
    model.fit(X, y)  # Entraînement du modèle avec les données d'entraînement

    # Chargement du fichier ARFF contenant les données de test pour l'évaluation des performances des classificateurs
    with open('OliveOil_TEST.arff') as f:
        test_data = arff.load(f)['data']
    X_test = np.array(test_data, dtype=np.float64)[:,:-1]
    y_test = np.array(test_data, dtype=np.float64)[:,-1]

    # Utilisation du modèle entraîné pour prédire les labels des données de test
    y_pred = model.predict(X_test)
```

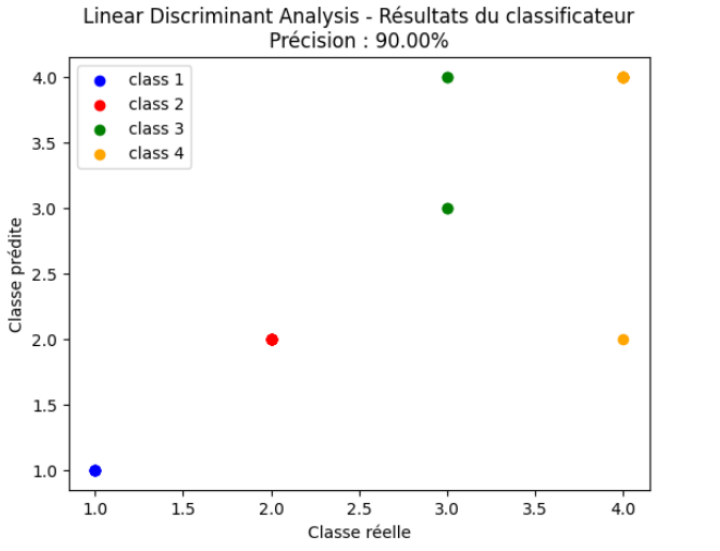
```
# Calcul de la précision du modèle
accuracy = accuracy_score(y_test, y_pred)

if accuracy > max_accuracy:
    max_accuracy = accuracy

# Affichage d'un graphe montrant les labels prédits par rapport aux labels réels pour chaque classe
colors = ['blue', 'red', 'green', 'orange']
labels = ['class 1', 'class 2', 'class 3', 'class 4']
for i in range(4):
    plt.scatter(y_test[y_test==i+1], y_pred[y_test==i+1], color=colors[i], label=labels[i])
plt.xlabel("Classe réelle")
plt.ylabel("Classe prédite")
plt.title(name + " - Résultats du classificateur\nPrécision : {:.2f}%".format(accuracy * 100))
plt.legend()
plt.show()
```

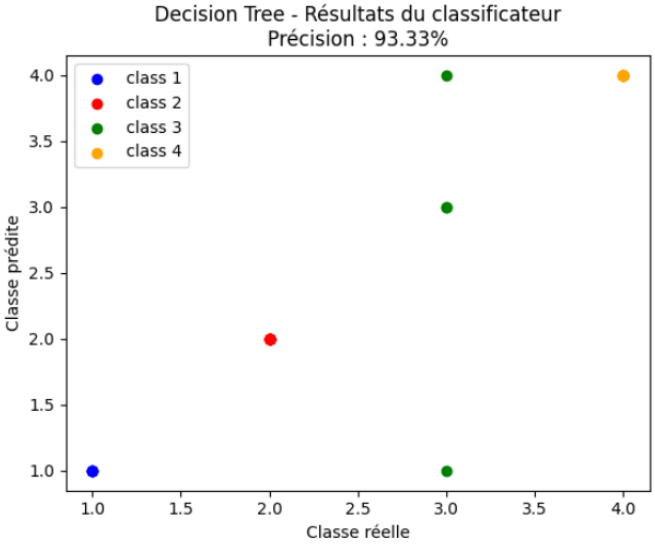
IV. Résultats :

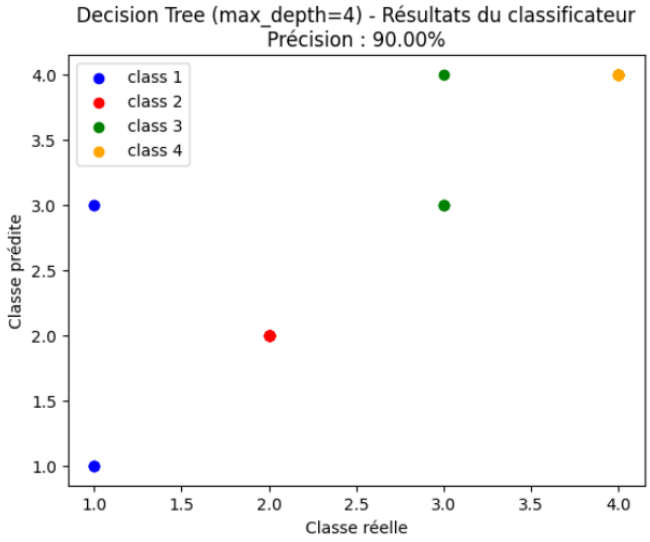


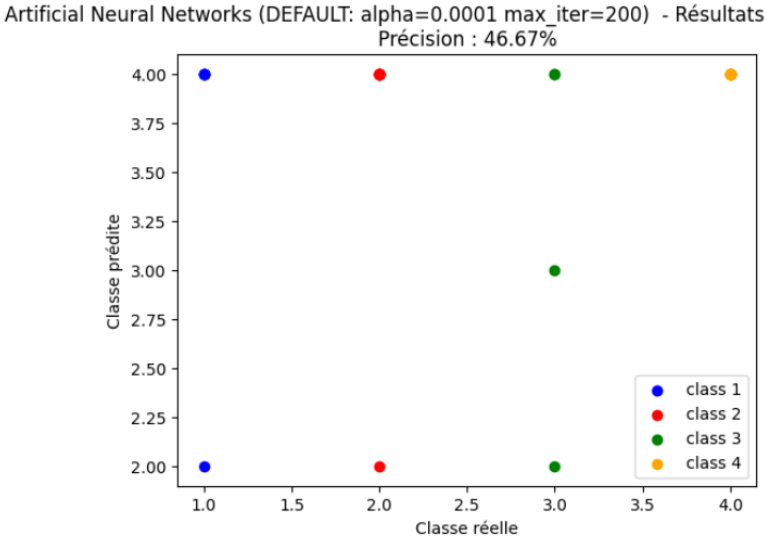

K-Nearest Neighbors (DEFAULT K=5) - Résultats du classificateur
Précision : 83.33%
class 1
class 2
class 3
class 4
Classe prédite
4.0
3.5
3.0
2.5
2.0
1.5
1.0

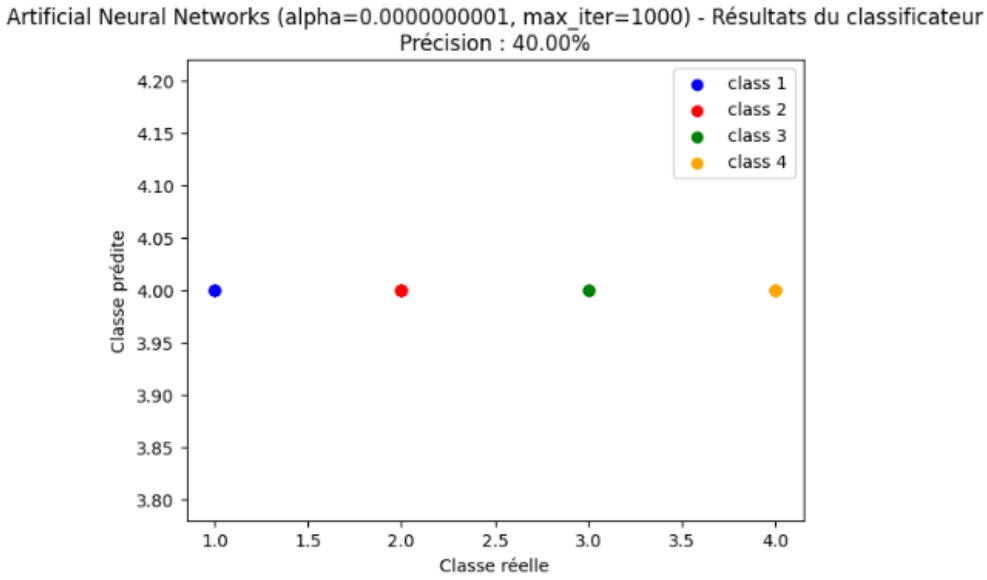

K-Nearest Neighbors (K=3) - Résultats du classificateur
Précision : 90.00%
class 1
class 2
class 3
class 4
Classe prédite
4.0
3.5
3.0
2.5
2.0
1.5
1.0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Classe réelle

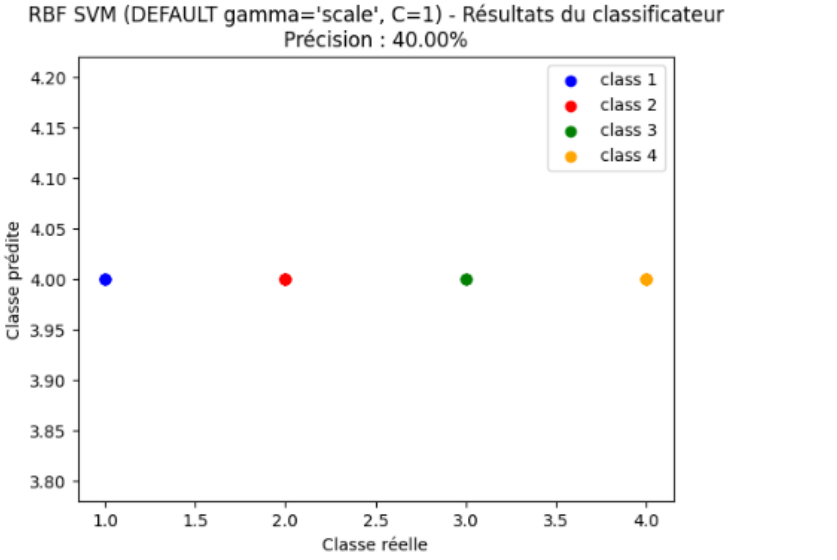

Linear Discriminant Analysis - Résultats du classificateur
Précision : 90.00%
class 1
class 2
class 3
class 4
Classe prédite
4.0
3.5
3.0
2.5
2.0
1.5
1.0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Classe réelle

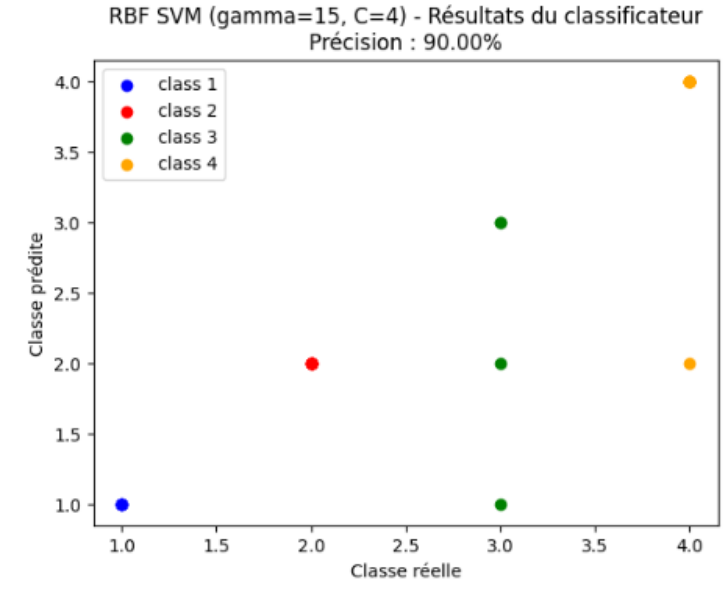

Decision Tree - Résultats du classificateur
Précision : 93.33%
class 1
class 2
class 3
class 4
Classe prédite
Classe réelle
4.0
3.5
3.0
2.5
2.0
1.5
1.0
1.0
1.5
2.0
2.5
3.0
3.5
4.0

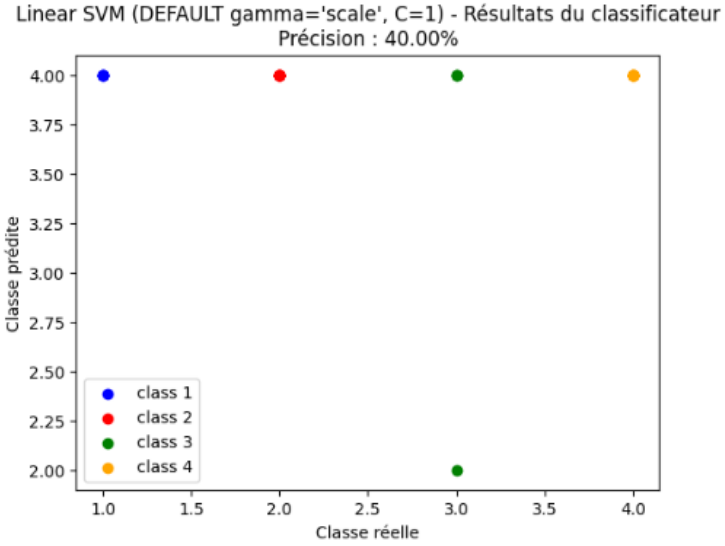

Decision Tree (max_depth=4) - Résultats du classificateur
Précision : 90.00%
class 1
class 2
class 3
class 4
Classe prédite
Classe réelle
4.0
3.5
3.0
2.5
2.0
1.5
1.0
1.0
1.5
2.0
2.5
3.0
3.5
4.0

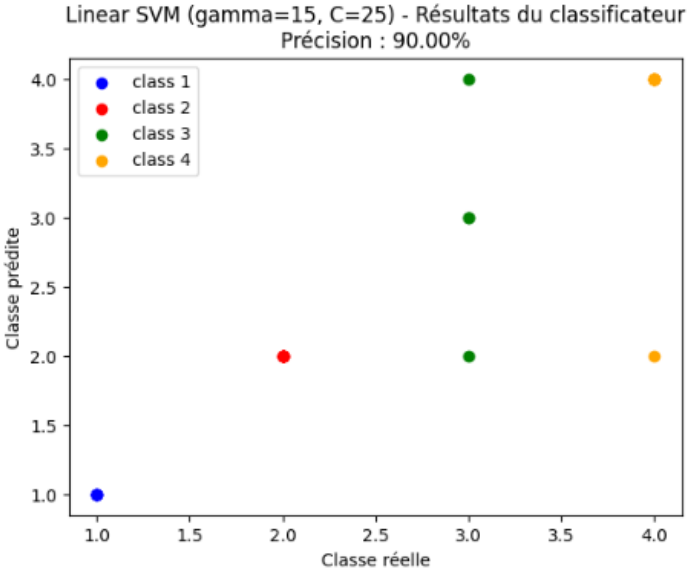

Artificial Neural Networks (DEFAULT: alpha=0.0001 max_iter=200) - Résultats du classificateur
Précision : 46.67%
class 1
class 2
class 3
class 4
Classe prédite
Classe réelle
4.00
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00
1.0
1.5
2.0
2.5
3.0
3.5
4.0

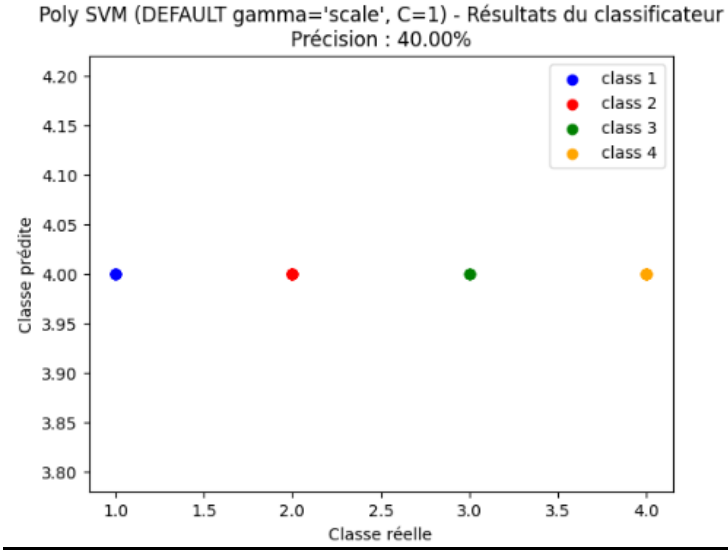

Artificial Neural Networks (alpha=0.0000000001, max_iter=1000) - Résultats du classificateur
Précision : 40.00%
class 1
class 2
class 3
class 4
Classe prédite
Classe réelle
4.20
4.15
4.10
4.05
4.00
3.95
3.90
3.85
3.80
1.0
1.5
2.0
2.5
3.0
3.5
4.0

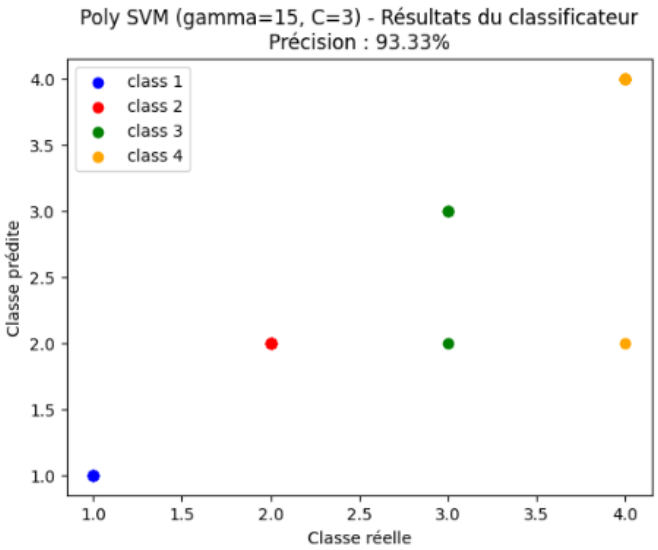

RBF SVM (DEFAULT gamma='scale', C=1) - Résultats du classificateur
Précision : 40.00%
class 1
class 2
class 3
class 4
Classe prédite
Classe réelle
4.20
4.15
4.10
4.05
4.00
3.95
3.90
3.85
3.80
1.0
1.5
2.0
2.5
3.0
3.5
4.0

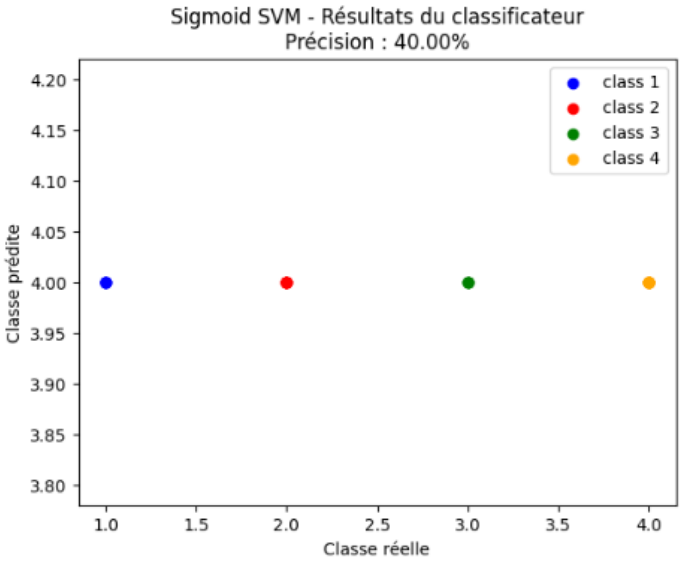

RBF SVM (gamma=15, C=4) - Résultats du classificateur
Précision : 90.00%
class 1
class 2
class 3
class 4
Classe prédite
Classe réelle
4.0
3.5
3.0
2.5
2.0
1.5
1.0
1.0
1.5
2.0
2.5
3.0
3.5
4.0

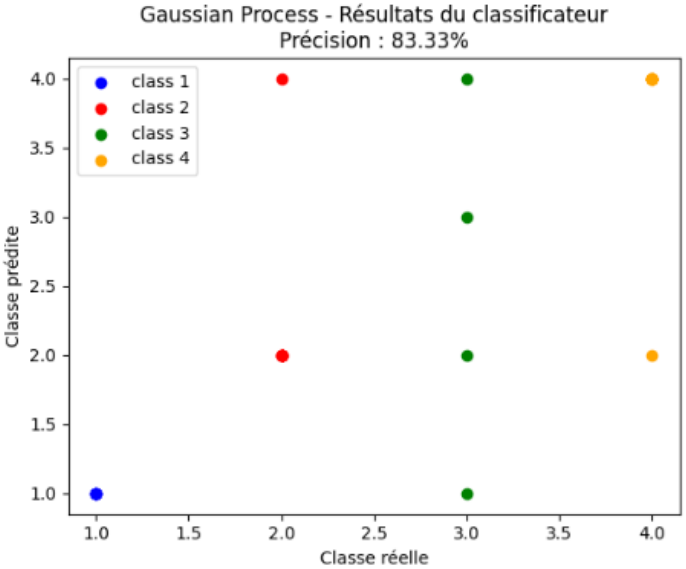

Linear SVM (DEFAULT gamma='scale', C=1) - Résultats du classificateur
Précision : 40.00%
Classe prédite
Classe réelle
class 1
class 2
class 3
class 4

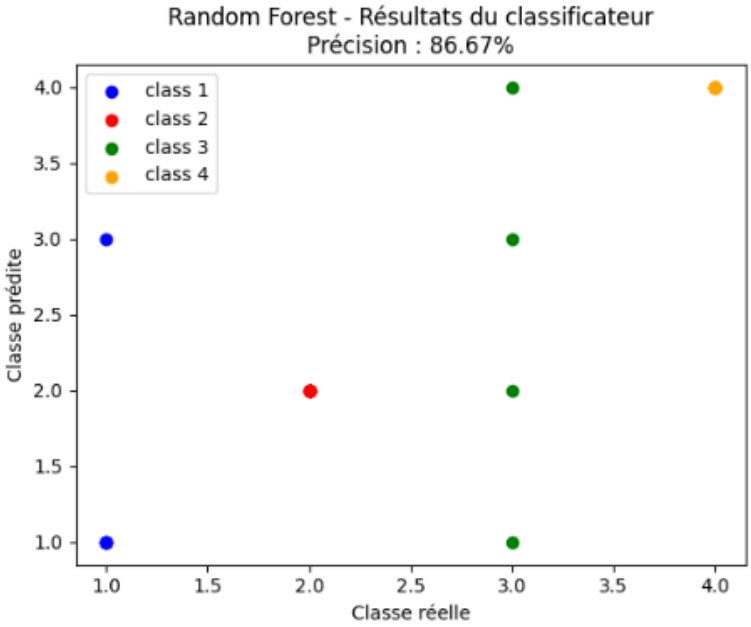

Linear SVM (gamma=15, C=25) - Résultats du classificateur
Précision : 90.00%
Classe prédite
Classe réelle
class 1
class 2
class 3
class 4

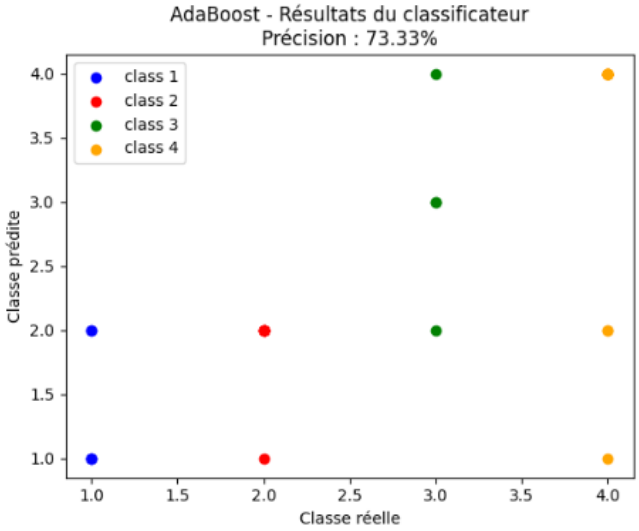

Poly SVM (DEFAULT gamma='scale', C=1) - Résultats du classificateur
Précision : 40.00%
Classe prédite
Classe réelle
class 1
class 2
class 3
class 4

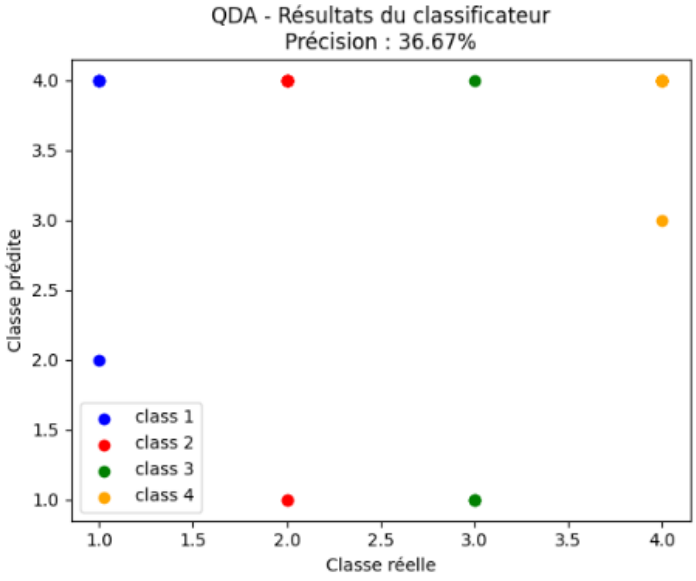

Poly SVM (gamma=15, C=3) - Résultats du classificateur
Précision : 93.33%
class 1
class 2
class 3
class 4
Classe prédite
Classe réelle
4.0
3.5
3.0
2.5
2.0
1.5
1.0
1.0
1.5
2.0
2.5
3.0
3.5
4.0


Sigmoid SVM - Résultats du classificateur
Précision : 40.00%
class 1
class 2
class 3
class 4
Classe prédite
Classe réelle
4.20
4.15
4.10
4.05
4.00
3.95
3.90
3.85
3.80
1.0
1.5
2.0
2.5
3.0
3.5
4.0


Gaussian Process - Résultats du classificateur
Précision : 83.33%
class 1
class 2
class 3
class 4
Classe prédite
Classe réelle
4.0
3.5
3.0
2.5
2.0
1.5
1.0
1.0
1.5
2.0
2.5
3.0
3.5
4.0


Random Forest - Résultats du classificateur
Précision : 86.67%
class 1
class 2
class 3
class 4
Classe prédite
Classe réelle
1.0
1.5
2.0
2.5
3.0
3.5
4.0


AdaBoost - Résultats du classificateur
Précision : 73.33%
class 1
class 2
class 3
class 4
Classe prédite
Classe réelle
1.0
1.5
2.0
2.5
3.0
3.5
4.0


QDA - Résultats du classificateur
Précision : 36.67%
class 1
class 2
class 3
class 4
Classe prédite
Classe réelle
1.0
1.5
2.0
2.5
3.0
3.5
4.0

V. Interpretation et Comparaison des différents algorithmes :

Dans le cadre de notre étude visant à réduire les coûts et le temps nécessaires à la détection d'adulération des huiles d'olive, nous avons utilisé différents classificateurs d'apprentissage automatique pour effectuer une analyse de classification sur le jeu de données OliveOil.

Nous avons testé les performances de Naive Bayes, K-Nearest Neighbors, Linear Discriminant Analysis, Decision Tree, Artificial Neural Networks, SVM, Gaussian Process, Random Forest, AdaBoost et Quadratic Discriminant Analysis. Les résultats de précision varient considérablement entre les différents classificateurs, avec des taux allant de 36,67% à 93,33%.

Les meilleurs résultats ont été obtenus avec les classificateurs Decision Tree (93,33%), K-Nearest Neighbors avec K=3 (90%), Linear Discriminant Analysis (90%) et RBF SVM avec gamma=15 et C=4 (90%). Ces classificateurs ont produit des résultats de classification très précis sur le jeu de données OliveOil.

Cependant, les moins bons résultats ont été obtenus avec QDA (36,67%), suivi de près par les classificateurs Artifical Neural Network (40%) et RBF SVM avec la configuration par défaut (40%). Ces classificateurs ont produit des résultats de classification beaucoup moins précis.

Il est important de noter que les performances des différents classificateurs peuvent varier en fonction du jeu de données et des paramètres utilisés pour l'entraînement. Par conséquent, il est important d'effectuer des tests de classification sur plusieurs jeux de données et de choisir le classificateur le plus approprié pour chaque tâche spécifique.

Ces résultats soulignent l'importance de notre projet dans la réduction des coûts et du temps associés à la détection d'adulération des huiles d'olive. En utilisant des classificateurs d'apprentissage automatique efficaces, il devient possible d'automatiser et d'accélérer le processus de détection, contribuant ainsi à une gestion plus économique et plus efficace de la qualité des huiles d'olive.

VI. Conclusion

En conclusion, notre projet a réussi à atteindre son objectif de développer un système de détection d'adulération des huiles d'olive en utilisant des classificateurs d'apprentissage automatique. Les résultats obtenus démontrent l'efficacité et la diversité des classificateurs utilisés dans la prédiction de la région correspondante aux séries temporelles de mesures d'absorbance d'huile d'olive. Parmi les classificateurs évalués, le Decision Tree, le K-Nearest Neighbors, le Linear Discriminant Analysis et le RBF SVM ont obtenu des performances remarquables, avec des taux de précision allant jusqu'à 93,33%.

Cependant, il est important de noter que le choix du classificateur approprié dépend des caractéristiques spécifiques du jeu de données et des paramètres utilisés. Les performances peuvent varier considérablement en fonction de ces facteurs. Ainsi, il est essentiel de poursuivre les recherches en élargissant le jeu de données, en testant davantage de classificateurs et en ajustant les paramètres pour améliorer encore les performances de prédiction.

Dans l'ensemble, ce projet a ouvert des perspectives prometteuses pour l'utilisation de classificateurs d'apprentissage automatique dans la détection d'adulération des huiles d'olive, en

réduisant à la fois les coûts et le temps associés à ce processus. Les résultats obtenus fournissent des bases solides pour des applications similaires dans le domaine de l'analyse des séries temporelles d'absorbance d'huile d'olive. Les futures analyses et développements dans ce domaine contribueront à améliorer continuellement les résultats de classification et à favoriser une utilisation plus efficace de l'apprentissage automatique pour garantir la qualité des huiles d'olive.