Analyse Discriminante Quadratique (QDA) -Multiclasse

Théorie

L'Analyse Discriminante Quadratique (QDA) est une technique de classification qui, contrairement à LDA, permet aux classes d'avoir des matrices de covariance différentes. Cela le rend plus flexible mais peut aussi augmenter le risque de sur-apprentissage.

Contrairement à d'autres modèles initialement conçus pour des problèmes binaires et adaptés aux cas multiclasse via OVA ou OVO, **QDA est intrinsèquement multiclasse**. Il attribue directement une observation à l'une des classes disponibles en estimant des distributions normales multivariées et en utilisant la règle de Bayes.

Hyperparamètre utilisé

Nous allons optimiser :

• Régularisation (reg_param) : contrôle la variance de la covariance estimée et est sélectionnée en fonction de la précision sur l'ensemble de validation.

Métriques d'évaluation

Nous afficherons:

- Matrice de confusion : montrant les erreurs de classification sur l'échantillon de test.
- Taux de bien classés sur l'échantillon de validation avec le meilleur hyperparamètre.
- Taux de bien classés sur l'échantillon de test avec ce même hyperparamètre.
- Taux de bien classés par classe sur l'échantillon de test pour observer la précision sur chaque classe.

Recherche du meilleur reg_param et évaluation

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.discriminant_analysis import

→ QuadraticDiscriminantAnalysis

from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.preprocessing import StandardScaler
import warnings
# Suppression des avertissements inutiles
warnings.filterwarnings("ignore", category=UserWarning)
# Chargement des ensembles de données
train_data = pd.read_csv('../data/covertype_train.csv')
val_data = pd.read_csv('../data/covertype_val.csv')
test_data = pd.read_csv('../data/covertype_test.csv')
# Préparation des données
X_train, y_train = train_data.drop('Cover_Type', axis=1),

    train_data['Cover_Type']

X_val, y_val = val_data.drop('Cover_Type', axis=1),

    val_data['Cover_Type']

X_test, y_test = test_data.drop('Cover_Type', axis=1),

    test_data['Cover_Type']

# Normalisation des données
scaler = StandardScaler()
X_train, X_val, X_test = scaler.fit_transform(X_train),

    scaler.transform(X_val), scaler.transform(X_test)

# Recherche du meilleur hyperparamètre reg_param
reg_params = np.linspace(0, 1, 10)
val_accuracies = []
for reg_param in reg_params:
    qda = QuadraticDiscriminantAnalysis(reg_param=reg_param)
    qda.fit(X_train, y_train)
    acc = accuracy_score(y_val, qda.predict(X_val))
    val_accuracies.append((reg_param, acc))
# Sélection du meilleur hyperparamètre
best_reg_param, best_val_acc = max(val_accuracies, key=lambda x:
 \rightarrow x[1])
# Affichage du graphique
```

```
plt.figure(figsize=(8, 6))
plt.plot(reg_params, [acc for reg_param, acc in val_accuracies],

    marker='o', linestyle='dashed', label="Validation")

plt.xlabel("Régularisation (reg_param)")
plt.ylabel("Précision sur validation")
plt.title("Impact de la régularisation sur la performance de

    QDA")

plt.legend()
plt.show()
# Modèle final avec le meilleur hyperparamètre
final model =
QuadraticDiscriminantAnalysis(reg_param=best_reg_param)
final_model.fit(X_train, y_train)
y_test_pred = final_model.predict(X_test)
# Matrice de confusion
conf_matrix = confusion_matrix(y_test, y_test_pred)
# Calcul des taux de bien classés par classe
class_accuracies = conf_matrix.diagonal() /

    conf_matrix.sum(axis=1)

overall_test_accuracy = accuracy_score(y_test, y_test_pred)
# Affichage des résultats
print(f"\n Meilleur hyperparamètre reg_param sur l'échantillon de

    validation : {best_reg_param:.2f}")

print(f"Taux de bien classés sur l'échantillon de validation avec

    cet hyperparamètre : {best_val_acc:.2%}")

print("\n Matrice de confusion sur l'échantillon de test, avec le

→ meilleur hyperparamètre :")
print(conf_matrix)
print("\n Taux de bien classés par classe sur l'échantillon de

    test, avec le meilleur hyperparamètre :")

for i, acc in enumerate(class_accuracies, start=1):
    print(f"Classe {i} : {acc:.2%}")
print(f"\n Taux de bien classés sur l'échantillon de test avec le

→ meilleur hyperparamètre : {overall_test_accuracy:.2%}")

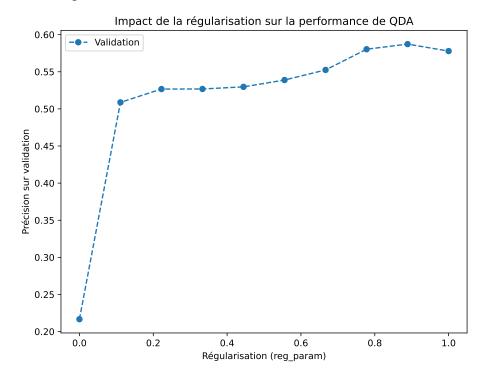
/home/ensai/.local/share/virtualenvs/postagram_ensai-i0XV51HB/lib/python3.10/site-packages/
  warnings.warn(
/home/ensai/.local/share/virtualenvs/postagram_ensai-i0XV51HB/lib/python3.10/site-packages/
  warnings.warn(
/home/ensai/.local/share/virtualenvs/postagram_ensai-i0XV51HB/lib/python3.10/site-packages/
  warnings.warn(
```

/home/ensai/.local/share/virtualenvs/postagram_ensai-i0XV51HB/lib/python3.10/site-packages/swarnings.warn(

/home/ensai/.local/share/virtualenvs/postagram_ensai-i0XV51HB/lib/python3.10/site-packages/swarnings.warn(

/home/ensai/.local/share/virtualenvs/postagram_ensai-i0XV51HB/lib/python3.10/site-packages/s
warnings.warn(

/home/ensai/.local/share/virtualenvs/postagram_ensai-i0XV5lHB/lib/python3.10/site-packages/swarnings.warn(



Meilleur hyperparamètre reg_param sur l'échantillon de validation : 0.89 Taux de bien classés sur l'échantillon de validation avec cet hyperparamètre : 58.73%

Matrice de confusion sur l'échantillon de test, avec le meilleur hyperparamètre :

[[1163		626	1	1	89	14	225]
	650	1561	51	27	402	125	17]
	0	7	948	136	50	289	0]
	0	0	45	53	0	12	0]
	28	103	27	19	191	12	0]
	11	34	230	53	31	335	0]
[103	35	3	0	16	0	664]]

Taux de bien classés par classe sur l'échantillon de test, avec le meilleur hyperparamètre

Classe 1 : 54.88% Classe 2 : 55.10% Classe 3 : 66.29% Classe 4 : 48.18% Classe 5 : 50.26% Classe 6 : 48.27% Classe 7 : 80.88%

Taux de bien classés sur l'échantillon de test avec le meilleur hyperparamètre : 58.60%