Analyse Discriminante Linéaire (LDA) - Multiclasse

Théorie

L'**Analyse Discriminante Linéaire (LDA)** est une technique de classification qui cherche à trouver une combinaison linéaire de caractéristiques maximisant la séparation entre plusieurs classes.

Contrairement à d'autres modèles initialement conçus pour des problèmes binaires et adaptés aux cas multiclasse via OVA ou OVO, **LDA 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 (`shrinkage`)** : contrôle la variance de la covariance estimée et est sélectionnée en fonction de la précision sur l'ensemble de validation.

Métriques d'évaluation

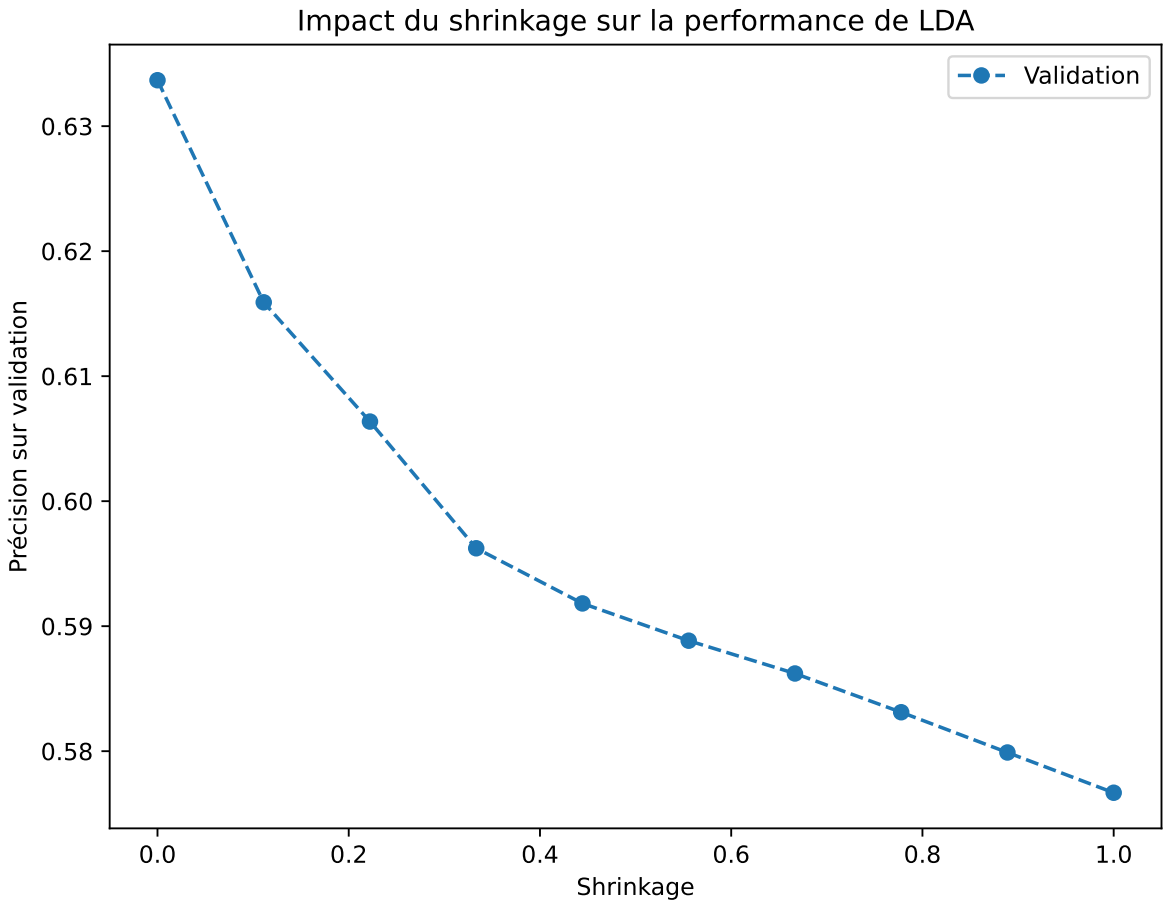
Nous afficherons :

- **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 `shrinkage` et évaluation

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
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('covertype_train.csv')
val_data = pd.read_csv('covertype_val.csv')
test_data = pd.read_csv('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 shrinkage
shrinkage_values = np.linspace(0, 1, 10)
val_accuracies = []

for shrinkage in shrinkage_values:
    lda = LinearDiscriminantAnalysis(solver='lsqr', shrinkage=shrinkage)
    lda.fit(X_train, y_train)
    acc = accuracy_score(y_val, lda.predict(X_val))
    val_accuracies.append((shrinkage, acc))

# Sélection du meilleur hyperparamètre
best_shrinkage, best_val_acc = max(val_accuracies, key=lambda x: x[1])
```

```
#  Affichage du graphique
plt.figure(figsize=(8, 6))
plt.plot(shrinkage_values, [acc for shrinkage, acc in val_accuracies],
 ↪  marker='o', linestyle='dashed', label="Validation")
plt.xlabel("Shrinkage")
plt.ylabel("Précision sur validation")
plt.title("Impact du shrinkage sur la performance de LDA")
plt.legend()
plt.show()

#  Modèle final avec le meilleur hyperparamètre
final_model = LinearDiscriminantAnalysis(solver='lsqr',
 ↪  shrinkage=best_shrinkage)
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 shrinkage sur l'échantillon de validation
 ↪  : {best_shrinkage:.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%}")
```




```
  Meilleur hyperparamètre shrinkage sur l'échantillon de validation : 0.00
Taux de bien classés sur l'échantillon de validation avec cet hyperparamètre : 63.37%

  Matrice de confusion sur l'échantillon de test, avec le meilleur hyperparamètre :
[[1344  520    1    0   27    2  225]
 [ 548 1865   48   17  218  124   13]
 [   0   29  902  130   32  337    0]
 [   0    0   44   53    0   13    0]
 [   2  160   27    0  179   12    0]
 [   0   53  210   25   42  364    0]
 [ 154    0    3    0    0    0  664]]

  Taux de bien classés par classe sur l'échantillon de test, avec le meilleur hyperparamètre
Classe 1 : 63.43%
Classe 2 : 65.83%
```

```
Classe 3 : 63.08%
Classe 4 : 48.18%
Classe 5 : 47.11%
Classe 6 : 52.45%
Classe 7 : 80.88%

 Taux de bien classés sur l'échantillon de test avec le meilleur hyperparamètre : 64.04%
```