Régression Multinomiale (Softmax Regression)

Théorie

La **régression multinomiale**, aussi appelée **régression logistique multinomiale**, est une extension de la régression logistique qui permet de gérer plusieurs classes. Elle utilise une fonction **Softmax** en sortie pour assigner une probabilité à chaque classe.

Hyperparamètre utilisé

Nous allons optimiser :

- **Paramètre de régularisation (`C`)** : contrôle la pénalisation de la complexité du modèle et est sélectionné 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 `C` et évaluation

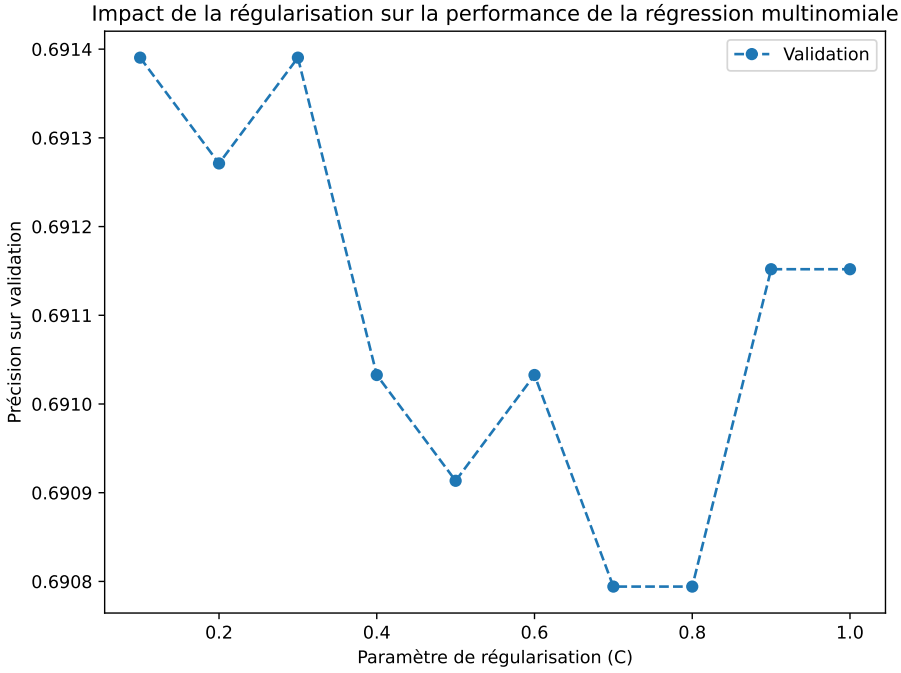
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.preprocessing import StandardScaler
```

```
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 C
C_values = np.arange(0.1, 1.1, 0.1)
val_accuracies = []

for C in C_values:
    model = LogisticRegression(multi_class='multinomial',
 ↪  solver='saga', C=C, penalty='l2', max_iter=500)
    model.fit(X_train, y_train)
    acc = accuracy_score(y_val, model.predict(X_val))
    val_accuracies.append((C, acc))

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


#  Affichage du graphique
plt.figure(figsize=(8, 6))
plt.plot(C_values, [acc for C, acc in val_accuracies],
 ↪  marker='o', linestyle='dashed', label="Validation")
plt.xlabel("Paramètre de régularisation (C)")
plt.ylabel("Précision sur validation")
plt.title("Impact de la régularisation sur la performance de la
 ↪  régression multinomiale")
```

```
plt.legend()
plt.show()

#  Modèle final avec le meilleur hyperparamètre
final_model = LogisticRegression(multi_class='multinomial',
 ↪  solver='saga', C=best_C, penalty='l2', max_iter=500)
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 C sur l'échantillon de
 ↪  validation : {best_C:.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-i0XV5lHB/lib/python3.10/site-packages/s
  warnings.warn(
/home/ensai/.local/share/virtualenvs/postagram_ensai-i0XV5lHB/lib/python3.10/site-packages/s
  warnings.warn(
/home/ensai/.local/share/virtualenvs/postagram_ensai-i0XV5lHB/lib/python3.10/site-packages/s
  warnings.warn(
/home/ensai/.local/share/virtualenvs/postagram_ensai-i0XV5lHB/lib/python3.10/site-packages/s
  warnings.warn(
/home/ensai/.local/share/virtualenvs/postagram_ensai-i0XV5lHB/lib/python3.10/site-packages/s
  warnings.warn(
/home/ensai/.local/share/virtualenvs/postagram_ensai-i0XV5lHB/lib/python3.10/site-packages/s
  warnings.warn(
/home/ensai/.local/share/virtualenvs/postagram_ensai-i0XV5lHB/lib/python3.10/site-packages/s
```

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




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


  Meilleur hyperparamètre C sur l'échantillon de validation : 0.10
Taux de bien classés sur l'échantillon de validation avec cet hyperparamètre : 69.14%

  Matrice de confusion sur l'échantillon de test, avec le meilleur hyperparamètre :
[[1416  583    1    0    6    1  112]
 [ 511 2142   87    1   40   44    8]
 [   0   48 1249   38    8   87    0]
 [   0    0   61   34    0   15    0]
 [   2  257   24    0   92    5    0]
 [   0   64  374    3    8  245    0]
 [ 163    1    3    0    0    0  654]]
```

```
 Taux de bien classés par classe sur l'échantillon de test, avec le meilleur hyperparamètre
Classe 1 : 66.82%
Classe 2 : 75.61%
Classe 3 : 87.34%
Classe 4 : 30.91%
Classe 5 : 24.21%
Classe 6 : 35.30%
Classe 7 : 79.66%

 Taux de bien classés sur l'échantillon de test avec le meilleur hyperparamètre : 69.54%
```