

# Régression Logistique Binomiale - One-Versus-One (OVO)

## Théorie

La **régression logistique binomiale** est utilisée pour la classification binaire, mais elle peut être adaptée aux problèmes **multiclasse** via l'approche **One-Versus-One (OVO)**. Ici, un modèle est entraîné pour chaque paire de classes, ce qui peut améliorer la précision lorsque les classes sont bien séparées.

## Hyperparamètre utilisé

Nous allons optimiser :

- **Paramètre de régularisation (C)** : qui contrôle 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.multiclass import OneVsOneClassifier
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('covertime_train.csv')
val_data = pd.read_csv('covertime_val.csv')
test_data = pd.read_csv('covertime_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 = OneVsOneClassifier(LogisticRegression(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
        ↪ logistique (OVO)")
plt.legend()
plt.show()

# Modèle final avec le meilleur hyperparamètre
final_model = OneVsOneClassifier(LogisticRegression(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)

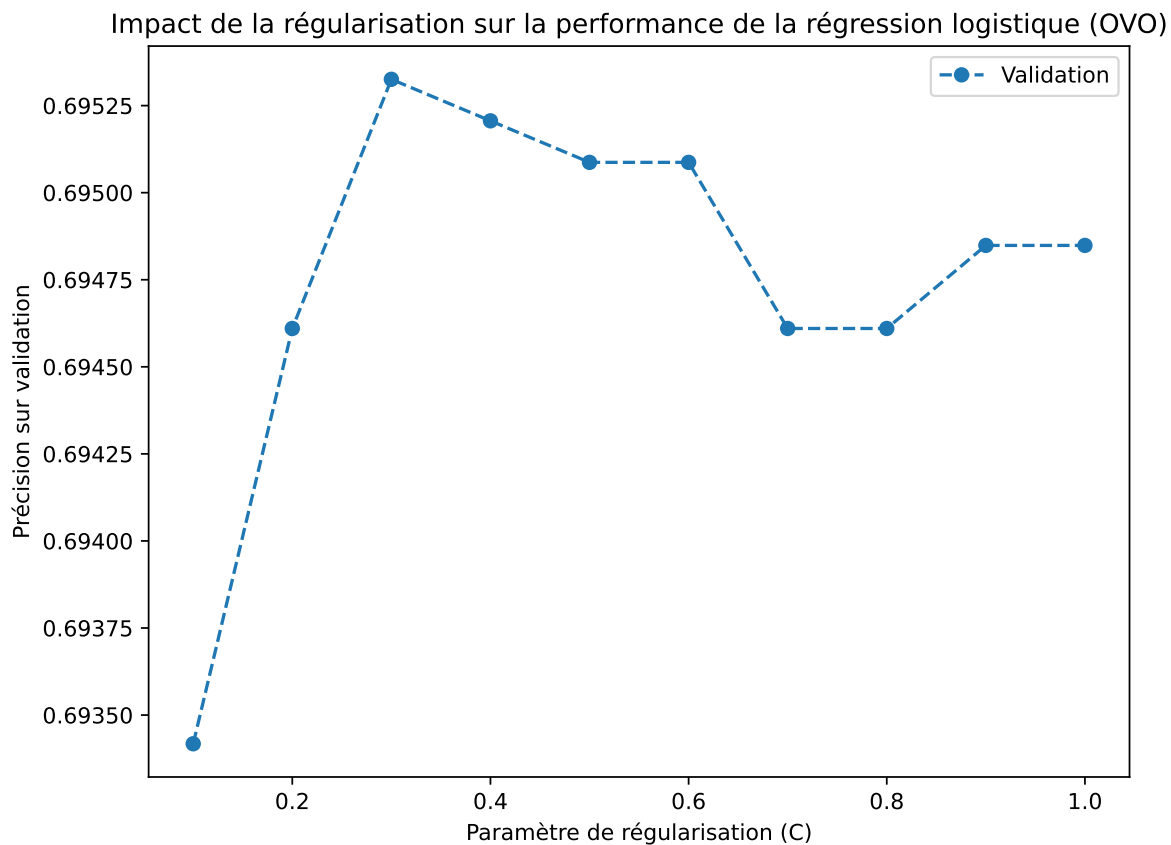
# Calcul des taux de bien classés par classe
class_accuracies = conf_matrix.diagonal() / conf_matrix.sum(axis=1)
overall_test_accuracy = accuracy_score(y_test, y_test_pred)

# 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%}")

```



Meilleur hyperparamètre C sur l'échantillon de validation : 0.30

Taux de bien classés sur l'échantillon de validation avec cet hyperparamètre : 69.53%

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

```
[[1419  584    1    0    6    1  108]
 [ 517 2136   85    1   36   50    8]
 [    0   44 1241   35    4  106    0]
 [    0    0   56   38    0   16    0]
 [    2  254   21    0   98    5    0]
 [    0   53  359    2    2  278    0]
 [ 158    1    2    0    0    0  660]]
```

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

Classe 1 : 66.97%

Classe 2 : 75.40%

Classe 3 : 86.78%

Classe 4 : 34.55%  
Classe 5 : 25.79%  
Classe 6 : 40.06%  
Classe 7 : 80.39%

Taux de bien classés sur l'échantillon de test avec le meilleur hyperparamètre : 69.99%