KNN - K-Nearest Neighbors

Théorie

Le **K-Nearest Neighbors (KNN)** est un algorithme de classification basé sur la proximité des données dans un espace multidimensionnel. Il attribue une classe à un point en fonction des **K voisins les plus proches**.

Hyperparamètre utilisé

Nous allons optimiser :

- **Nombre de voisins (`k`)** : déterminé en fonction de la précision sur l'ensemble de validation.

Métriques d'évaluation

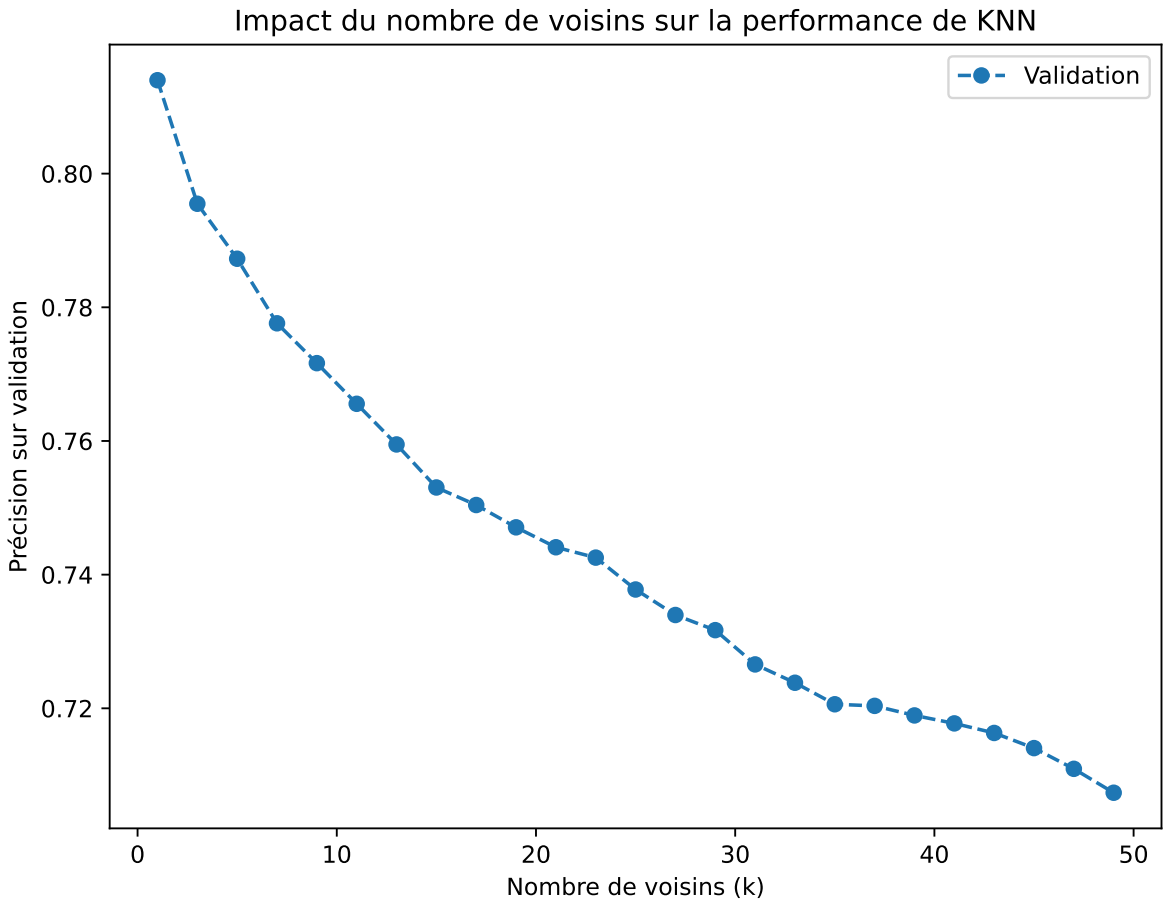
Nous afficherons :

- **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 `k` et évaluation

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
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 k
k_values = range(1, 51, 2)
val_accuracies = []

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    acc = accuracy_score(y_val, knn.predict(X_val))
    val_accuracies.append((k, acc))

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

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

#  Modèle final avec le meilleur hyperparamètre
final_model = KNeighborsClassifier(n_neighbors=best_k)
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 nombre de voisins (k) sur l'échantillon de validation :
 ↪  {best_k}")
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 nombre de voisins (k) sur l'échantillon de validation : 1
Taux de bien classés sur l'échantillon de validation avec cet hyperparamètre : 81.40%

  Matrice de confusion sur l'échantillon de test, avec le meilleur hyperparamètre :
[[1663  352    1    0   14    5   84]
 [ 370 2288   37    0   80   49    9]
 [   1   29 1235   20    4  141    0]
 [   0    3   29   70    0    8    0]
 [  13   53   10    0  296    8    0]
 [   3   31  142   11    0  507    0]
 [  40   10    0    0    1    0  770]]

  Taux de bien classés par classe sur l'échantillon de test, avec le meilleur hyperparamètre
Classe 1 : 78.48%
Classe 2 : 80.76%
```

```
Classe 3 : 86.36%
Classe 4 : 63.64%
Classe 5 : 77.89%
Classe 6 : 73.05%
Classe 7 : 93.79%

 Taux de bien classés sur l'échantillon de test avec le meilleur hyperparamètre : 81.42%
```