TP1 : KNN

Objectif du TP

- Comprendre le fonctionnement de l'algorithme KNN.
- Appliquer KNN à un jeu de données réel (classification).
- Comparer les performances avec différents paramètres (valeurs de *k*, normalisation, etc.).

Partie 1 – Préparation du projet

1.1. Importer les bibliothèques

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

1.2. Choisir un jeu de données

Vous pouvez utiliser par exemple :

- `Iris` (facile à comprendre)
- ou `Breast Cancer` (diagnostic cancer du sein)
  Disponible directement dans `sklearn.datasets`.

```
from sklearn.datasets import load_iris

data = load_iris()

X = data.data

y = data.target

df = pd.DataFrame(X, columns=data.feature_names)

df['target'] = y

df.head()
```

Partie 2 – Préparation des données

2.1. Séparer les données

```
X_train, X_test, y_train, y_test = train_test_split(
   X, y, test_size=0.3, random_state=42
)
```

2.2. Normaliser les variables

KNN est sensible aux échelles des données.

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Partie 3 – Entraînement du modèle KNN

3.1. Créer et entraîner le modèle

```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_scaled, y_train)
```

3.2. Prédire

```
y_pred = knn.predict(X_test_scaled)
```

Partie 4 – Évaluation du modèle

```
print("Accuracy :", accuracy_score(y_test, y_pred))
print("\nMatrice de confusion :\n", confusion_matrix(y_test, y_pred))
print("\nRapport de classification :\n", classification_report(y_test, y_pred))
```

Visualisation :

```
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cmap="Blues")
plt.xlabel("Prédictions")
plt.ylabel("Réel")
plt.show()
```

Partie 5 – Choix du meilleur k

Tester plusieurs valeurs de *k* pour trouver la meilleure précision :

```
scores = []
k_values = range(1, 21)
for k in k_values:
   knn = KNeighborsClassifier(n_neighbors=k)
   knn.fit(X_train_scaled, y_train)
   scores.append(accuracy_score(y_test, knn.predict(X_test_scaled)))
plt.plot(k_values, scores, marker='o')
plt.xlabel("Valeur de k")
plt.ylabel("Précision")
plt.title("Choix du meilleur k")
plt.show()
print("Meilleur k :", k_values[np.argmax(scores)])
```

Questions / Exercices à programmer

1. Implémentez vous-même l'algorithme **KNN sans utiliser scikit-learn**.
   (Calcul des distances, recherche des *k* plus proches voisins, vote majoritaire, prédiction et évaluation.)
2. Écrivez une fonction pour comparer différentes **mesures de distance** :
   - Euclidienne
   - Manhattan
   - Cosinus

   Affichez et commentez les résultats obtenus.
3. Écrivez un programme qui **affiche graphiquement les *k* plus proches voisins** d'un point choisi dans le jeu de test.
   (Utilisez `matplotlib` pour visualiser les points et leurs voisins.)
4. Appliquez votre code KNN sur **plusieurs jeux de données** (`iris`, `wine`, `breast_cancer`, ou un fichier CSV personnel) et comparez les précisions obtenues.
5. Écrivez un script qui teste **plusieurs tailles de jeux d'entraînement** (20 %, 40 %, 60 %, 80 %) et affiche l'évolution de la précision.

6. Créez un programme qui **cherche automatiquement la meilleure valeur de k** en testant plusieurs valeurs et en affichant la courbe “k vs précision”.
7. Modifiez votre implémentation pour **pondérer le vote des voisins** selon leur distance (ex. poids = 1 / distance²).
   Comparez les performances avec la version non pondérée.
8. Affichez les **frontières de décision du KNN** sur un jeu de données à deux variables. (Réduisez `iris` à deux features et visualisez avec `plt.contourf`.)
9. Comparez les performances du **KNN avec d’autres modèles** :
   - Arbre de décision
   - SVM
   - Régression logistique

   Présentez un tableau comparatif des précisions.
10. Écrivez un code qui **identifie et affiche les exemples mal classés**.
    Montrez leurs voisins les plus proches et discutez les raisons possibles des erreurs.
11. Implémentez une version **KNN pour la régression** : la prédiction doit être la moyenne (ou la moyenne pondérée) des *k* voisins les plus proches.
12. Mettez en place une **validation croisée manuelle** (ou avec `cross_val_score`) pour évaluer plus précisément la performance moyenne du KNN.
13. Utilisez les structures `KDTree` ou `BallTree` de `sklearn.neighbors` pour **accélérer la recherche des voisins** et mesurez le temps d’exécution avant/après.
14. Écrivez un programme qui ajoute du **bruit dans les données** (erreurs d’étiquettes ou variables perturbées) et étudiez son impact sur les performances du KNN.
15. Créez une **mini-application interactive** (par exemple avec `matplotlib` ou `streamlit`) où l’utilisateur peut sélectionner un point et visualiser ses *k* voisins.