Master 1 – BDIA
Analyse et visualisation des données

TP2 - Iris de Fischer

Exercice : Iris de Fisher

Le jeu de données `Iris de Fischer` contient des informations sur 150 fleurs d'iris appartenant à trois espèces : *setosa*, *versicolor*, et *virginica*. Les données portent sur des mesures de longueur et largeur des sépales et pétales. Chaque observation inclut quatre caractéristiques :

- Longueur des sépales (`Sepal.Length`),
- Largeur des sépales (`Sepal.Width`),
- Longueur des pétales (`Petal.Length`),
- Largeur des pétales (`Petal.Width`).

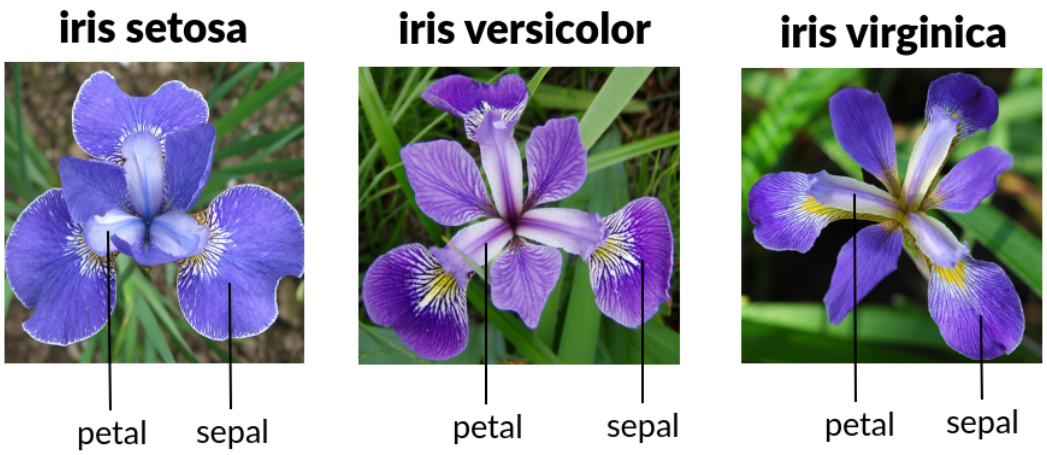

Figure 1: Iris de Fischer.

Ce jeu de données est disponible dans R. Vous l'utiliserez pour réaliser des statistiques descriptives et effectuer une classification à l'aide de l'algorithme des plus proches voisins (*k-Nearest Neighbors*, kNN) expliqué en annexe du document.

Objectifs de l'exercice

Cet exercice vous permettra de :

- Manipuler et visualiser des données en R,
- Effectuer des analyses statistiques simples,
- Comprendre et implémenter un algorithme de classification de base.

Questions :

1. Chargement et exploration des données

- Charger le jeu de données `Iris` inclus par défaut dans R.
- Afficher les 6 premières lignes du jeu de données.
- Afficher les dimensions du jeu de données (`dim()`).

2. Analyse statistique

- Calculer les statistiques descriptives (moyenne, médiane, variance, écart-type) pour chaque caractéristique numérique.
- Identifier les valeurs minimales et maximales de chaque caractéristique.

3. Visualisation

- Créer un graphique **boîte à moustaches** (*boxplot*) pour comparer les distributions de la longueur des sépales (`Sepal.Length`) pour chaque espèce. Voir annexe pour la signification du graphique.
- Réaliser un graphique **nuage de points** (*scatter plot*) montrant la relation entre la longueur des pétales (`Petal.Length`) et leur largeur (`Petal.Width`). Ajouter des couleurs pour différencier les espèces.

4. Préparation des données pour la classification

- Séparer le jeu de données en un ensemble d'apprentissage (*train set*, 80%) et un ensemble de test (*test set*, 20%).
- Normaliser les caractéristiques numériques (`Sepal.Length`, `Sepal.Width`, `Petal.Length`, `Petal.Width`) pour que leurs valeurs soient comprises entre 0 et 1.

5. Classification avec kNN : lire l'annexe

- Implémenter l'algorithme des plus proches voisins (`kNN`) pour classer les observations de l'ensemble de test en utilisant l'ensemble d'apprentissage.
- Utiliser le package `class` et la fonction `knn()` pour réaliser la classification avec $k = 3$.
- Évaluer la précision de votre modèle à l'aide d'une matrice de confusion (`table()`).
- Essayer différentes valeurs de k ($k = 1, 5, 7$) et comparez les performances.

6. Analyse finale

- Quels sont les avantages et inconvénients de l'algorithme kNN sur ce jeu de données ?

- Quelle valeur de k semble la plus adaptée, et pourquoi ?

Annexe A : Graphique boite à moustaches

Le graphique boîte à moustaches (ou boxplot) est un graphique statistique qui résume la distribution d'un jeu de données de manière visuelle. Il est particulièrement utile pour visualiser la dispersion, la symétrie et la présence de valeurs aberrantes. Il représente :

- Médiane (Q2 - 50%) : La ligne à l'intérieur de la boîte correspond à la médiane, qui sépare la moitié inférieure et la moitié supérieure des données.
- Premier quartile (Q1 - 25%) : La borne inférieure de la boîte représente le premier quartile (Q1), soit la valeur en dessous de laquelle se situent 25% des données.
- Troisième quartile (Q3 - 75%) : La borne supérieure de la boîte représente le troisième quartile (Q3), soit la valeur en dessous de laquelle se situent 75% des données.
- Intervalle interquartile (IQR) : C'est l'écart entre Q3 et Q1 (IQR = Q3 - Q1). Il représente l'intervalle contenant les 50% centraux des valeurs.
- Moustaches : Elles s'étendent jusqu'à 1,5 fois l'IQR au-dessus de Q3 et en dessous de Q1.
- Valeurs aberrantes (outliers) : Les points en dehors des moustaches sont considérés comme des valeurs atypiques.

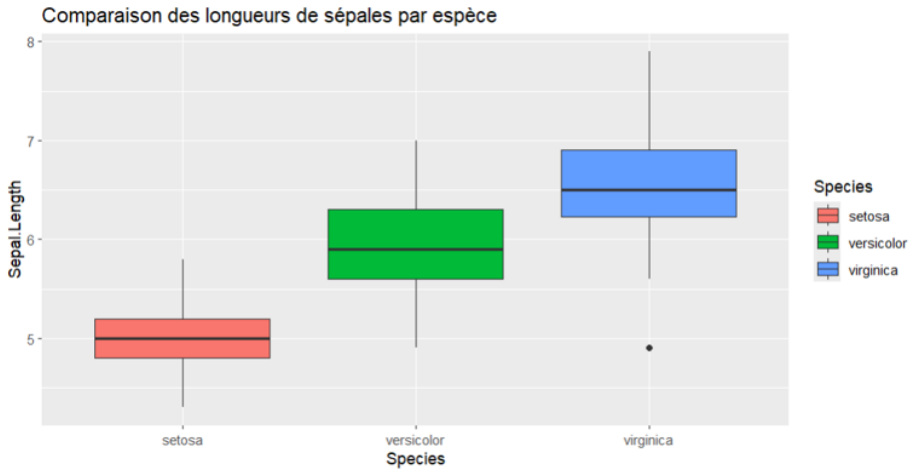

Figure 2: Graphique Boite à moustaches.

Annexe B : Algorithme des Plus Proches Voisins

L'algorithme des plus proches voisins (k-Nearest Neighbors, kNN) est une méthode simple et efficace utilisée en **classification** et parfois en **régression**. Il repose sur l'idée que des observations proches dans l'espace des caractéristiques sont susceptibles d'appartenir à la même classe.

Principe de fonctionnement

1. **Représentation des données** : Chaque observation est représentée comme un point dans un espace à n-dimensions, où n correspond au nombre de caractéristiques.
2. **Étape d'apprentissage** : kNN est un algorithme **non paramétrique** et **paresseux**, ce qui signifie qu'il ne construit pas explicitement un modèle pendant la phase d'entraînement. Les données d'apprentissage sont simplement stockées.
3. **Étape de classification** : Pour prédire la classe d'une observation inconnue, l'algorithme :
   (a) Calcule la distance entre cette observation et chaque observation dans les données d'entraînement.
   (b) Identifie les k observations les plus proches (*k-nearest neighbors*).
   (c) Assigne à l'observation inconnue la classe majoritaire parmi ces k voisins.

Avantages

- **Simplicité** : Facile à comprendre et à implémenter.
- **Pas d'hypothèses** : Aucune hypothèse sur la distribution des données.
- **Adapté aux petites bases de données** : Efficace pour des jeux de données de taille modeste.

Inconvénients

- **Complexité computationnelle** : Le temps de prédiction augmente avec la taille des données d'entraînement.
- **Sensible au bruit** : Des voisins mal classés peuvent fausser le résultat.
- **Nécessité de normalisation** : Les caractéristiques doivent être mises à l'échelle pour éviter que des dimensions avec des valeurs plus grandes dominent les calculs de distance.
- **Choix de k** :
  - Si k est trop petit, l'algorithme est sensible au bruit.
  - Si k est trop grand, il peut inclure des voisins d'autres classes, réduisant la précision.

Application au Jeu de Données Iris

L'algorithme kNN peut être appliqué au célèbre jeu de données Iris de Fisher. Voici les étapes principales :

1. Charger le jeu de données avec `data(iris)`.
2. Normaliser les caractéristiques (`Sepal.Length`, `Sepal.Width`, etc.) pour que leurs valeurs soient comprises entre 0 et 1.

3. Diviser les données en deux ensembles : *train set* (80%) et *test set* (20%).
4. Utiliser la fonction `knn()` du package `class` pour appliquer l'algorithme kNN. Par exemple, pour $k = 3$:

```
library(class)
predictions <- knn(train = train[, 1:4], test = test[, 1:4], cl
= train$Species, k = 3)
table(Predicted = predictions, Actual = test$Species)

```

5. Tester différentes valeurs de k ($k = 1, 3, 5, 7$) pour comparer les performances.

Évaluer la précision à l'aide d'une matrice de confusion

Lorsqu'on applique un modèle de classification, comme l'algorithme des plus proches voisins (*k-Nearest Neighbors*, kNN), il est essentiel de mesurer ses performances. La **matrice de confusion** est un outil fondamental pour cette évaluation.

Qu'est-ce qu'une matrice de confusion ?

Une matrice de confusion est un tableau qui résume les performances du modèle en comparant :

- Les observations correctement classées.
- Les erreurs de classification (observations mal classées).

Pour un problème de classification avec n classes, la matrice de confusion est une matrice $n \times n$, où :

- Les **lignes** représentent les **classes réelles** (vraies étiquettes).
- Les **colonnes** représentent les **classes prédites** (par le modèle).

Exemple

Supposons un jeu de données `Iris` où $k = 3$. Après application de kNN, nous obtenons une matrice de confusion :

Actual\Predicted	setosa	versicolor	virginica
setosa	10	0	0
versicolor	0	8	2
virginica	0	1	9

Interprétation :

- La valeur 10 (ligne 1, colonne 1) indique que 10 observations de la classe réelle `setosa` ont été correctement prédites comme `setosa`.
- La valeur 2 (ligne 2, colonne 3) indique que 2 observations de la classe réelle `versicolor` ont été incorrectement prédites comme `virginica`.
- Les valeurs diagonales représentent les classifications correctes.

- Les valeurs hors diagonale représentent les erreurs de classification.

Précision du modèle

La **précision** (*accuracy*) mesure le pourcentage de classifications correctes parmi toutes les observations. Elle est calculée comme suit :

$$\text{Précision} = \frac{\text{Nombre de classifications correctes}}{\text{Nombre total d'observations}}$$

Pour l'exemple ci-dessus :

- Nombre de classifications correctes (diagonale) : $10 + 8 + 9 = 27$.
- Nombre total d'observations : $10 + 0 + 0 + 0 + 8 + 2 + 0 + 1 + 9 = 30$.
- Précision :
$$\text{Précision} = \frac{27}{30} = 0.9\,(90\%)$$

Ces métriques sont particulièrement utiles pour des classes déséquilibrées.

Utilisation de `table()` en R

En R, la fonction `table()` est utilisée pour construire une matrice de confusion. Exemple :

```r
# Prédictions du modèle kNN
predictions <- knn(train = train[, 1:4], test = test[, 1:4], cl =
    train$Species, k = 3)

# Matrice de confusion
confusion <- table(Predicted = predictions, Actual = test$Species)
print(confusion)

# Calcul manuel de la précision
accuracy <- sum(diag(confusion)) / sum(confusion)
cat("Précision : ", accuracy, "\n")
```

Conclusion

L'algorithme kNN est une méthode efficace pour résoudre des problèmes de classification sur des jeux de données comme le jeu Iris de Fischer. Sa simplicité en fait un excellent choix pour des tâches où les frontières entre classes sont complexes, mais il reste limité par sa sensibilité au bruit et sa complexité computationnelle pour de grands ensembles de données.