14/04/2021

IA – Classification Challenge

Datascience et IA

OBADIA Elie
ESILV A3

Tableau de bord

Le programme KNN se compose de plusieurs fonctions permettant soit de réaliser une tache simple soit de combiner les fonctions entre elles. Voici le nom des différentes fonctions composant le programme :

- get_type : retourne le type de l'élément en argument
- Split : permet de séparer les éléments d'une liste par type
- Répartition : permet de répartir aléatoirement les éléments d'une liste en deux autres listes
- ListeFinale : permet de créer deux liste similaire (dépendant de la parité de la base de données d'entrée) avec une répartition similaire de type
- Ordination : permet de trier les éléments de la base de données en fonction de leurs distances avec l'élément testé
- Dist : retourne la valeur de la p-ème distance entre deux caractéristiques de deux éléments (p est un paramètre)
- Distance : retourne la distance moyenne entre deux éléments
- Frequence : retourne le type de l'élément le plus présent de la liste d'entrée
- Apprentissage : permet de créer une base de données globale au programme « apprise »
- Confirmation : permet de créer la matrice de confusion et calculer le pourcentage d'erreur du programme selon la liste d'apprentissage et de validation
- Ligne : retourne la ligne associée au type de l'élément dans la matrice de confusion
- Colonne : retourne la colonne associée au type de l'élément dans la matrice de confusion
- MatriceConfusion : créer la matrice de confusion de l'algorithme
- Initialisation : permet de regénérer des listes d'apprentissages et de validation pour minimiser le taux d'erreur
- KNN : retourne le type de l'élément trouvé par l'algorithme

Le programme possède plusieurs paramètres réglables par l'utilisateurs selon ces besoins :

- K : le nombre de voisins a considéré
- P : valeur de la puissance dans la formule pour le calcul de la distance (2 = euclidienne)
- reussite : pourcentage de bonne déduction voulue

Il m'a semblé primordial de pouvoir mettre en place ces paramètres qui permettent une grande flexibilité de l'algorithme.

J'ai choisi 88% de réussite avec 3 voisins et la distance euclidienne pour résoudre le problème. Cela permet d'avoir un grand taux de réussite et un temps d'exécution convenable de quelques minutes tout au plus.

J’ai tout d’abord commencé par visualiser le projet dans son ensemble et je l’ai divisé en petite tâches pour permettre de pouvoir intervenir plus rapidement et aisément sur un quelconque problème.

J’ai choisi de créer une grande et unique base de données à partir de *data* et *preTest* que je sépare aléatoirement en deux listes une pour l’apprentissage et une pour la validation. J’ai tenté grâce à cela de permettre à mon algorithme d’être le plus pertinent possible tout en évitant le sur apprentissage.

J’ai eu l’idée de réaliser x-exécutions de l’algorithme avec une même liste d’apprentissage et de validation. Cela aurait donné x types pour l’élément à trouver et j’aurais pris le type dont l’occurrence est la plus fréquentes pour le même élément ce qui permettrait de minimiser le taux d’erreur. Cependant je n’ai pas programmé cela car avec mon ordinateur le temps de calcul est trop long.