DM8 MP2I : Files de priorités; algorithme KNN

Thèmes

1. Files de priorité ;
2. Algorithme KNN

Avant de commencer ce devoir, il faut se documenter sur les énumérations en C et les instructions **switch...case**.

Rendu

Devoir à faire seul ou en binôme. Dans l'archive **dm8.zip** on trouve :

— Tous les fichiers d'en-têtes du devoir : **bibli.h, iris.h, knn.h, tas.h** ;

— Un fichier **iris.c** contenant les données auxquelles appliquer l'algorithme KNN (il faudra compléter ce fichier en écrivant la fonction **void display_variety(enum variety v)**) ;

— Un fichier **test.c** contenant le **main** et les tests à effectuer ;

— Un fichier **out.txt** contenant les rendus des tests.

Dupont doit rendre sur **cahier de prépa** dans une archive **dm8_dupot.zip** :

— le fichier **tas.c** contenant les primitives des tas ainsi que certaines fonctions d'affichage ;

— Le fichier **iris.c** complété ;

— Éventuellement vos propres tests sous le nom **mytests.c** et dans ce cas un **Makefile** qui me permette d'appliquer alternativement mes tests et les vôtres ;

— un fichier **knn.c** contenant les fonctions permettant d'appliquer l'algorithme KNN.

Toutes les fonctions dont les prototypes sont donnés dans les fichiers d'en-tête sont à écrire. Mais il est possible d'y ajouter des fonctions auxiliaires.

Il n'est ni souhaitable ni autorisé d'inclure (par directive d'inclusion) des fichiers **.c** comme je l'ai trop souvent vu dans le DM6.

1 Files de priorité

Nous implantons la notion vue en cours de *files de priorités*. Il faut écrire toutes les fonctions de **tas.h**. Elles doivent passer les tests de la partie I.

2 Base de données des Iris

Pas grand chose à faire dans cette partie : juste implanter **void display_variety(enum variety v)** en utilisant une instruction **switch...case**.

3 Algorithme KNN

Il faut écrire 3 (ou peut-être 4) fonctions.

— On considère
  — un ensemble E de n points à p coordonnées représentant des données étiquetées.
  — un entier $k < n$ et $x \notin E$
— On cherche les k points de E les plus « proches » de x. Le plus souvent, on attribue ensuite à x la caractéristique la plus fréquemment partagée parmi ces k points.
— La notion de *proximité* amène à définir une *distance* entre deux points. Cette distance est calculée à partir des coordonnées des points. Exemple : distance euclidienne.

Il faut d'abord implanter la fonction **float dist(COORD_TYPE p1, COORD_TYPE p2)** qui calcule la distance euclidienne entre deux points à **COORD_TYPE** coordonnées.

Le listing 1 donne le pseudo-code de l'algorithme KNN.

Listing 1 – Algorithme KNN avec file de priorité

```
entree :
  un point x,
  un nuage de points E,
  une distance d entre points
sortie : une file de priorité contenant les k points de E les plus proches de x
debut:
  Créer T un tas-max vide;
  pour tout point p de E faire
    si |T| < k ajouter p à T avec d(p,x) comme priorité
    sinon
      si d(p,x) est plus petit que la priorité de la racine de T
         retirer la racine;
         ajouter p à T avec d(p,x) comme priorité
  fin_faire
  renvoyer T
fin
```

Le tas T contient les k plus proches voisins de x dans E à la fin de l'algirithme.

Écrire la fonction **knn** qui implante le KNN.

Ce KNN est un algorithme dit d'« apprentissage supervisé ». On utilise une partie des données d'apprentissages pour déterminer la classe d'un point et on se sert de l'autre partie pour faire des tests. Dans le code de **test_knn**, de construit un nuage d'apprentissage constitué d'un tiers des points de **coords**.

Une fois le tas plein, on le parcours et on détermine la classe majoritaire parmi les données stockées dans le tas. On renvoie cette classe. Dans l'exemple des iris, il s'agit donc de renvoyer une des trois valeurs **SETOSA, VERSICOLOUR, VIRGINICA**. C'est la fonction **attribution** qui réalise cela.

C'est tout pour le moment. Mais j'ajouterai peut-être une question sur la *matrice de confusion*.