Arbres k-dimensionnels : évaluation de la qualité du vin

On dispose d'une base de données portant sur l'analyse de vins. Environ 4900 vins du Portugal ont été analysés sur la base de 11 critères objectifs (acidité, taux d'acide citrique, taux de sucre, pH, densité, ...). De plus, un panel de 3 experts a évalué chacun de ces vins et donné une note (subjective) entière entre 0 (mauvais vin) et 10 (bon vin). Les résultats sont donnés sous forme d'un fichier texte au format `csv` :

```
7;0.27;0.36;20.7;0.045;45;170;1.001;3;0.45;8.8;6
6.3;0.3;0.34;1.6;0.049;14;132;0.994;3.3;0.49;9.5;6
8.1;0.28;0.4;6.9;0.05;30;97;0.9951;3.26;0.44;10.1;6
7.2;0.23;0.32;8.5;0.058;47;186;0.9956;3.19;0.4;9.9;6
7.2;0.23;0.32;8.5;0.058;47;186;0.9956;3.19;0.4;9.9;6
8.1;0.28;0.4;6.9;0.05;30;97;0.9951;3.26;0.44;10.1;6
6.2;0.32;0.16;7;0.045;30;136;0.9949;3.18;0.47;9.6;6
7;0.27;0.36;20.7;0.045;45;170;1.001;3;0.45;8.8;6
...
```

Le but du TP est d'écrire un programme capable de prédire automatiquement la note d'un vin entre 0 et 10 à partir seulement des mesures objectives. Nous sommes donc dans le cadre de **l'apprentissage supervisé** et on utilisera l'algorithme des **K plus proches voisins**. Afin de gagner en efficacité, on utilisera la structure d'**arbre k-dimensionnel**.

1 Implémentation des arbres k-dimensionnels

On vous fournit deux fichiers `kdtree.h` et `kdtree.c` (incomplet) qui regroupent les définitions et fonctions concernant les arbres k-dimensionnels. Un arbre k-dimensionnel est représenté par le type :

```
#define KMAX 16

struct noeud_s {
    int vecteur[KMAX];
    int etiquette;
    struct noeud_s *gauche;
    struct noeud_s *droite;
};

typedef struct noeud_s * kdtree;
```

Il s'agit donc d'une structure d'arbre binaire dont les nœuds contiennent un vecteur codé comme un tableau d'entiers. Par anticipation, on a également ajouté un champ `etiquette` permettant de sauvegarder l'étiquette du vecteur. L'arbre vide est codé par `NULL`.

La spécification de chacune des fonctions demandées ci-dessous est précisée dans le fichier d'en-tête `kdtree.h`.

Question 1

Écrire les fonctions `kdtree_taille` et `kdtree_hauteur` permettant respectivement de calculer la taille et la hauteur d'un `kdtree`.

Question 2

Écrire la fonction `kdtree_noeud` permettant de construire un nouveau nœud, et donc un nouvel arbre, à partir d'un fils gauche, d'un fils droit, ainsi que des informations associées au nœud.

Question 3

Écrire la fonction `kdtree_find` permettant de trouver, s'il existe, un nœud dont la valeur est égale à un vecteur donné.

Question 4

Écrire la fonction `kdtree_insere` permettant d'insérer un nouveau vecteur dans l'arbre k-dimensionnel. Cette fonction renvoie le nouvel arbre obtenu (sans le duppliquer).

Question 5

Dans le fichier `tp.c` compléter la fonction `test_insere_find`. Cette fonction doit avoir le comportement suivant :

- Inserer dans l'arbre `a` chacun des vecteurs enregistrés dans le tableau `data` en lui associant comme étiquette son indice dans `data`.
- Vérifier la présence de chacun des vecteurs insérés à l'aide d'assertions et de `kdtree_find`.
- Afficher la taillet et la hauteur de l'arbre.
- Afficher l'arbre avec la fonction `kdtree_print` fournie.

Appeler cette fonction de test dans la fonction `main`. Vérifier le bon fonctionnement.

2 Recherche du plus proche voisin à l'aide du kdtree

On cherche maintenant à utiliser la structure de `kdtree` pour déterminer le vecteur le plus proche d'un vecteur donné.

Question 6

Écrire la fonction `distc` permettant de calculer le **carré** de la distance euclidienne entre deux vecteurs

Question 7

Écrire la fonction `best_node` prenant en entrée deux nœuds `x` et `y` et renvoyant celui qui est le plus proche d'un vecteur `u` donné. La valeur de `y` peut éventuellement être `NULL`, dans ce cas on renvoie simplement `x`.

Question 8

En déduire la fonction `kdtree_ppv` permetant de déterminer le nœud associé au plus proche voisin d'un vecteur `u` donné. On appliquera l'algorithme vu en cours qui consiste à élaguer la recherche dans le fils gauche ou droit sous certaines conditions.

Question 9

Dans `tp.c`, utiliser la fonction `test_ppv` pour vérifier le bon fonctionnement de `kdtree_ppv`.

3 Apprentissage supervisé

On souhaite maintenant appliquer nos résultats au problème concret de la notation des vins. On vous fournit une fonction `load_wine_file` permettant de charger le fichier de données. Les données sont alors enregistrées dans un tableau de `vecteurs` de dimension 11 ainsi qu'un tableau d'`étiquettes`. La fonction renvoie le nombre d'exemples lus. Afin de ne travailler qu'avec les entiers, cette fonction multiplie toutes les valeurs du fichier par 1000.

Question 10

Dans la fonction `main()` écrire le code permettant de charger le fichier de données et de stocker dans un kdtree les 1000 premiers exemples du fichiers. Afficher la taille et la hauteur du kdtree obtenu.

Question 11

À l'aide des fonctions codées précédemment, appliquer l'algorithme du plus proche voisin pour déterminer l'étiquette des 1000 exemples suivants. On calculera et affichera :

- l'écart moyen entre la note prédite et la note réelle
- le nombre de notes bien prédites : on considère qu'une note est bien prédite lorsqu'elle varie au plus de 1 par rapport à la note réelle.

Question 12

Écrire un programme permettant de construire et afficher la matrice de confusion sur ces 1000 exemples.

4 Pour aller plus loin...

Pour éviter le phénomène de *sur-apprentissage* il est préférable d'étendre notre algorithme en considérant plusieurs plus proches voisins et en prenant l'étiquette majoritaire.

Question 13

Écrire une fonction `kdtree_supprime` permettant d'enlever vecteur d'un kdtree. Pour nous simplifier la tâche on pourra procéder ainsi :

- Localiser puis supprimer le nœud.
- Remplacer ce noeud par un nouvel arbre obtenu en reconstruisant un `kdtree` à partir de tous les nœuds des fils gauche et droit.

Question 14

En déduire une fonction de prédiction de note utilisant l'algorithme des K plus proches voisins avec un K pouvant être choisi arbitrairement. Vérifier si l'on obtient de meilleurs résultats.