# TD ITC$^2$ N° 9: ALGORITHMIQUE POUR L'I.A. 1
## ETIQUETAGE ET CLASSIFICATION

### EXERCICE N°1: Détermination de la provenance d'un vin

On dispose d'un set de données constitué des enregistrements de 13 caractéristiques chimiques quantitatives (codées sous forme de flottants de profondeur 64 bits pour 178 vins différents:

| | vin 1 | vin 2 | … |
|---|---|---|---|
| Alcohol | 11.0 | 14.8 | … |
| Malic Acid | 0.74 | 5.80 | … |
| Ash | 1.36 | 3.23 | … |
| Alcalinity of Ash | 10.6 | 30.0 | … |
| Magnesium | 70.0 | 162.0 | … |
| Total Phenols | 0.98 | 3.88 | … |
| Flavanoids | 0.34 | 5.08 | … |
| Nonflavanoid Phenols | 0.13 | 0.66 | … |
| Proanthocyanins | 0.41 | 3.58 | … |
| Colour Intensity | 1.3 | 13.0 | … |
| Hue | 0.48 | 1.71 | … |
| OD280/OD315 of diluted wines | 1.2 | 4.00 | … |
| Proline | 278 | 1680 | … |

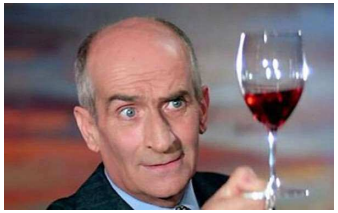

Figure 1: Chateau Léoville Las Cases 1953 ?

Ses vins proviennent de 3 régions de productions différentes en Italie qui seront simplement numérotées 0, 1, ou 2, et on souhaite implémenter un algorithme qui, à partir des caractéristiques chimiques d'un vin inconnu (mais issu d'une des 3 zones de productions), renverra le numéro de sa région de production.

1. Combien y-a-t-il de classes dans ce problème d'étiquetage? Les citer.
2. Importer le set de données *wine* du module sklearn à l'aide des commandes:

```
from sklearn import datasets
vin=datasets.load_wine()
```

   Ce set se présente sous la forme d'un *Bunch* de données que l'on interroge à l'aide de clés exactement comme un dictionnaire:

   - la clé `'data'` contenant un tableau `numpy` de dimensions (178,13)
   - la clé `'target'` contenant les 178 étiquettes des vins du dataset sous la forme d'un tableau `numpy` de dimension 1.
   - la clé `'frame'` vide
   - la clé `'target_name'` contenant le nom des étiquettes de classe (chaines de caractères) sous la forme un tableau `numpy` de dimension 1.
   - les clés `'DESCR'` et `'feature_names'` qui ne seront pas exploitées ici, et contenant respectivement l'ensemble des données du *dataset* sous forme d'une chaine de caractères, et les noms des caractéristiques des vins sous forme d'une liste de chaines de caractères.

## 1 Préparation des données

3. Extraire les données numériques sur les vins sous la forme d'un tableau `numpy` $X$ de dimensions $(178,13)$. Construire ensuite le dictionnaire `DictX` dont les clés sont les numéros des vins ($i \in [0,177]$), et les valeurs, les étiquettes correspondantes.

4. Construire de même le tableau `numpy` d'apprentissage $A$ constitué des 3/4 des valeurs du tableau $X$ prises au hasard; bâtir de même le dictionnaire `DictA` sur le même principe que `DictX`.

   On exploitera la fonction `sample(L:list,n:int) → list` du module `random` prenant en argument la liste $L$ et l'entier $n$, et renvoyant une liste de $n$ ($< len(L)$) éléments pris au hasard dans la liste $L$.

## 2 Algorithme des $k$ plus proches voisins

5. Compléter les fonctions Python:
   `kppv(A,DictA,k,x) → list` et `Etiqu_maj(L:list,epsilon:float) → int` afin que soit renvoyée l'étiquette inconnue d'un vin de caractéristique $x$. La fonction `tri` est supposée connue à ce stade de l'étude (son implémentation est proposée un peu plus loin), et renvoie la liste triée dans l'ordre croissant des distances aux voisins:

Listing 1: Fonction kppv

```
def kppv(A,DictA,k,x):
    L=[]
    N,n=A.shape
    for i in range(N):
        dist2=0.0
        for j in range(n):
            ............
        dist=np.sqrt(dist2)
        L.append((....,....))
    return Etiqu_maj(tri(L)[.....],1e-4)
```

Listing 2: Fonction Etiqu_maj

```
def Etiqu_maj(L,epsilon):
    if L[0][1]<=epsilon:
        return L[0][0]
    dicfreq={}
    freqmax=0
    for el in L:
        if el[0] in dicfreq:
            ............
        else:
            ............
        if dicfreq[el[0]]>=freqmax:
            clefreqmax=el[0]
    return clefreqmax
```

## 3 Fonction de tri

On souhaite implémenter un algorithme de tri performant.

6. Compléter les fonctions suivantes qui réalisent le tri des données de la liste L; quelle méthode de tri implémentent-elles?

Listing 3:

```
def tri(L):
    if len(L)<=1:
        .....
    else:
        m=.....
        return fct(tri(L[:m]),tri(L[m:]))

def fct(L1,L2):
    if L1==[]:
        .....
    if L2==[]:
        .....
    if L1[0][1]<L2[0][1]:
        return [L1[0]]+fct(L1[1:],L2)
    else:
        return .....
```

## 4 Performance de l'algorithme

7. Compléter la fonction `MatConf(X,DictX,SplA,A,DictA,k)` → `M:array` qui renvoie la matrice de confusion:

Listing 4:

```
def MatConf(X,DictX,SplA,A,DictA,k):
    N,n=X.shape
    nb_etiqu=len(vin['target_names'])
    M=np.zeros((nb_etiqu,nb_etiqu),dtype=int)
    compt=0
    liste_i=[]
    while compt<N//4: # on realise les tests sur 25% du nb total de données
        i=rd.randint(0,N-1)
        if i ..............:
            x=.......
            etiqu_vraie=.......
            etiqu_estim=.......
            M[......,.....].....
            ...........
            compt+=1
    return M
```

8. Ecrire une fonction `perf(X,DictX,SplA,A,DictA,k)` → `float` qui renvoie le pourcentage de bonnes prédictions de l'algorithme sur l'ensemble des classes. Prendre $k = 15$ et le tester.
9. Ecrire enfin une fonction python `optim_k(X,DictX,SplA,A,DictA)`→ `int` renvoyant la valeur optimale $k_{opt}$ de $k$, chaque valeur étant testée 100 fois. Lancer cette fonction pour déterminer $k_{opt}$.

---

## EXERCICE N°2: Construction d'une classification (*clustering*)

On va chercher dans cet exercice à retrouver la classification du *dataset* des vins utilisé en exercice n°1 en partant des données du *dataset* initialement mélangées.

On rappelle ici la structure du *dataset* `wine`:

- la clé `'data'` contient un tableau `numpy` de dimensions (178, 13)
- la clé `'target'` contient les 178 étiquettes des vins du *dataset* sous la forme d'un tableau `numpy` de dimension 1.
- la clé `'frame'` vide
- la clé `'target_name'` contient le nom des étiquettes de classe (chaines de caractères) sous la forme un tableau `numpy` de dimension 1.
- les clés `'DESCR'` et `'feature_names'` qui ne seront pas exploitées ici, et contiennent respectivement l'ensemble des données du *dataset* sous forme d'une chaine de caractères, et les noms des caractéristiques des vins sous forme d'une liste de chaines de caractères.

# 1 Préparation des données

En plus des modules `random` et `numpy`, charger le *dataset* sur les vins avec les commandes:

```
from sklearn import datasets
vin=datasets.load_wine()
```

1. Les données sur les 178 vins du set `wine` sont disposées dans un tableau de dimensions $(178,13)$, chaque ligne contenant les 13 caractéristiques d'un vin. Dans l'algorithme des k-moyennes, les données seront au contraire disposées en colonne et leur caractéristiques en ligne.

   Ecrire une fonction python `transpose(X:array) → array` recevant un tableau numpy $X$ et renvoyant son tableau transposé. On l'appliquera par la suite au tableau de données des vins pour définir le nouveau tableau de données $X1$ qui sera exploité pour réaliser la classification.
2. Ecrire une fonction python `aleat(X:array) → (Xf:array,d:dict)` recevant le tableau des données originel $X$ de dimensions $(178,13)$ et renvoyant un tableau $Xf$ de dimension $(13,178)$ contenant les caractéristiques des vins pris au hasard dans $X1$ et **sans répétition**.

   On fera usage d'un dictionnaire $d$ **qui sera également renvoyé**, et qui permettra à la fois de vérifier l'absence de répétitions dans la sélection, et **de pouvoir retrouver la position originelle de l'élément dans le tableau** $X1$ **en vue d'une vérification de sa classe d'origine**. Sa structure sera la suivante:

$$d = \left\{\text{indice de l'élément dans Xf : indice de l'élément dans X1}\right\}$$

# 2 Construction et vérification de la classification

La classification sera réalisée par les fonctions `k_moyennes(Xf,k)` et `barycentre(X)` fournies (ce sont celles du cours!). On va chercher à vérifier qu'en posant $k = 3$ on retrouve la classification originelle du set de données.

**NB:** On rappelle que la fonction `k_moyennes(Xf,k)` renvoie un tuple contenant la classification sous la forme d'une liste $P$ de $k$ listes, chacune correspondant à une classe reconstituée, et la liste $C$ des barycentres de chacune des $k$ classes.

3. Proposer une fonction python:

   `Etiqu_maj(L:list,d:dict,classes:array) → (maj:int,freq:list)`

   recevant une liste $L$ d'entiers, le dictionnaire $d$ décrit plus haut, ainsi que le tableau `classes` des classes originelles, et qui renvoie un tuple contenant le numéro de la classe majoritaire dans la liste $L$ (les 3 classes originelles sont numérotées 0, 1 et 2) ainsi qu'une liste $freq$ d'occurrence des 3 classes dans $L$.
4. Lancer enfin à plusieurs reprises le script suivant pour vérifier la qualité de la classification:

   Listing 5:

```
X=vin['data'] #dataset originel
Xf,d=aleat(X) #constitution d'un dataset mélangé, et du dictionnaire mémoire

classif=k_moyennes(Xf,3) #classification en 3 classes

print(Etiqu_maj(classif[0][0],d,vin['target']))
print(Etiqu_maj(classif[0][1],d,vin['target']))
print(Etiqu_maj(classif[0][2],d,vin['target']))
```

   Commenter les résultats successifs.