TP n° 12 – Projet : Instruments de musique

I Principe général

On a réalisé l'analyse de soixante sons musicaux obtenus par quatre instruments différents. Pour chacun des sons, vingt caractéristiques ont été obtenues. Ces vingt caractéristiques permettent en théorie de discriminer le timbre de l'instrument utilisé, supposé différent pour chaque instrument.
L'ensemble des données est réuni dans le fichier `instrument_features.csv` situé dans le sous-répertoire « PTSI/TP13 » du répertoire « Ressources ». Le fichier comprend une ligne pour chaque son et vingt colonnes recensant les caractéristiques et, en vingt-et-unième et vingt-deuxième colonnes, le nom du fichier musical source et le nom de l'instrument utilisé pour produire le son. Copier ce fichier dans votre répertoire personnel.
L'objectif de ce TP est de d'implanter en langage python un algorithme permettant de classer automatiquement un son dans une catégorie correspondant à un instrument. Les catégories étant connues à l'avance (piano, violon, clarinette, trompette) on parle de classification « surpervisée ».

II Méthode

Pour réaliser la classification supervisée automatique, on part du principe suivant :

1. pour un son donné joué par un instrument supposé inconnu (« son-candidat »), on calcule la racine carrée de la somme des carrés des écarts de ses caractéristiques avec les caractéristiques d'un son déjà catégorisé (« son-référence » dont on suppose connu l'instrument) ; on appelle ce résultat « distance candidat-référence » ;
2. on fait ce calcul pour chacun des sons-références (le plus simple est de supposer que pour un son-candidat donné, les cinquante-neuf autres sons sont des sons-références) ; on dispose donc d'un ensemble de distances candidat-références ;
3. on considère que le son-candidat doit être classé dans la catégorie du son-référence qui a la distance la plus petite.

III Principales étapes du projet

1. ouvrir le fichier de données ;
2. lire les données du fichier et les stocker dans une structure adaptée ;
3. séparer les données qui ont vocation à être traitées numériquement de celles qui sont purement textuelles et les stocker dans des structures distinctes ;
4. s'assurer que toutes les données ont un type correct ;
5. choisir un son-candidat (les autres sons seront les sons-références) ;
6. réaliser le calcul des distances pour le son-candidat tiré ;
7. déterminer le son-référence le plus proche et en déduire la catégorie à laquelle appartient le son-candidat
8. afficher ce résultat à l'écran ;
9. vérifier que la réponse trouvée est correcte.

IV Pour aller plus loin

1. À l'aide d'une boucle réaliser de façon automatisée le projet précédent pour tous les sons-candidats possibles avec un affichage automatique de la vérification de la catégorie trouvée.
2. Avec la méthode du projet tel que présenté au-dessus, la catégorie attribuée au son-candidat est dite du « premier voisin ». On peut aussi déterminer les catégories du second voisin, du troisième voisin etc. en identifiant les distances candidat-référence par ordre décroissant. Réaliser cette modification du script et écrire les réponses dans un fichier réponse qui indiquera sur une même ligne le numéro du son-candidat, la catégorie vraie telle qu'indiquée dans le fichier de données et les différentes catégories correspondant aux différents voisins successifs.

V Annexe – Méthodes, attributs et fonctions associées aux fichiers et chaîne de caractères

Se référer à l'énoncé du TP 6 « Manipulations de fichiers ».

VI Annexe – Méthodes, attributs et fonctions associées aux tableaux numpy

VI.1 Forme et dimensions d'un tableau

Pour obtenir la forme d'un tableau numpy, on utilise l'attribut `.shape` ou la fonction `shape` qui prend comme argument un tableau numpy. Exemple pour une matrice 2 lignes 3 colonnes :

```
A=np.array([[1,2,3],[4,5,6]])
A
A.shape
np.shape(A)
```

renvoie

```
array([[1, 2, 3],
       [4, 5, 6]])
(2,3)
(2,3)
```

On peut récupérer le nombre de lignes ou de colonnes, en appelant l'élément correspondant de `shape`. Exemple pour une matrice 2 lignes 3 colonnes :

```
A.shape[0]
A.shape[1]
```

renvoie

```
2
3
```

VI.2 Préparation de tableaux numériques

Il est parfois utile de commencer à travailler avec un tableau numpy déjà rempli de façon automatisée. De nombreuses fonctions permettent de le faire. On en présente trois ci-dessous.

```
np.zeros((2,3)) # attention au parenthesage !
```

renvoie

```
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
```

```
A=np.ones((2,3),dtype=int) # fixe le type int (au lieu de float par defaut)
A
```

renvoie

```
array([[1, 1, 1],
       [1, 1, 1]])
```

```
np.diag([1,2,3])
```

renvoie

```
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])
```

VI.3 Maximum et minimum d'un tableau

On peut utiliser les méthodes `.min` et `.max` associées au tableau ou les fonctions `min` et `max` qui prennent comme argument un tableau numpy. Le code :

```
A=np.array([[1, 2, 3],[4, 5, 6],[7, 8, 9],[10,11,12]])
np.min(A)
np.max(A)
A.min()
A.max()
```

renvoie

```
1
12
1
12
```

Pour trouver la position d'un maximum ou d'un minimum on peut utiliser la fonction `where` qui prend comme argument une condition sur un tableau numpy. La fonction renvoie des tableaux numpy contenant les indices de position respectant la condition. Le code :

```
A=np.array([[1, 2, 3],[4,5,6],[7,8,9],[10,11,12]])
np.where(A==np.min(A))
np.where(A==np.max(A))
```

renvoie

```
(array([0]), array([0]))
(array([3]), array([2]))
```

Ce qui signifie que le minimum est en position (0,0) et le maximum en (3,2).
Si, par exemple, on veut accéder au numéro la ligne ou de la colonne sur laquelle se trouve le maximum, on fait :

```
int(np.where(A==np.max(A))[0])
int(np.where(A==np.max(A))[1])
```

ce qui renvoie, évidemment

```
3
2
```

VI.4 Concaténation de tableaux

Avec des tableaux numpy, l'opérateur + réalise une addition terme à terme et non pas la concaténation comme avec des listes.
Pour concaténer deux tableaux numpy à deux dimensions, on peut utiliser la fonction `np.concatenate` qui prend comme arguments les deux tableaux ou parties de tableaux et `axis` qui permet de préciser si les deux tableaux doivent être mis bout à bout en ajoutant le nombre de lignes sans toucher aux colonnes (`axis=0`) ou le contraire (`axis=1`). Le code :

```
A=np.array([[1,2,3],[4,5,6]])
B=np.array([[7,8,9],[10,11,12]])
C=np.concatenate((A,B),axis=0)
C
np.shape(C)
```

renvoie

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9],
       [10,11,12]])
(4,3)
```

Le code

```
C=np.concatenate((A,B),axis=1)
D
np.shape(D)
```

renvoie

```
array([[1, 2, 3, 7, 8, 9],
       [4, 5, 6, 10,11,12]])
(2,6)
```

VI.5 Suppression d'une ligne d'un tableau

Pour supprimer une ligne d'un tableau numpy à deux dimensions, on peut concaténer deux sous-tableaux excluant la ligne à supprimer. Par exemple, pour supprimer la ligne numéro `n` du tableau `tab`, on peut faire :

```
tab = np.concatenate((tab[0:n,:],tab[n+1:,:]),axis=0)
```

ou encore, de façon plus explicite :

```
tab = np.concatenate((tab[0:n,:],tab[n+1:np.shape(tab)[0],:]),axis=0)
```

On peut bien entendu supprimer la colonne `p`, grâce à :

```
tab = np.concatenate((tab[:,0:p],tab[:,p+1:]),axis=1)
```