K-plus proches voisins

1 Introduction

1.1 Objectifs poursuivis

Le but de cette séance de travaux pratiques est essentiellement de reprendre les idées vu en cours. Le stockage et la représentation des données différeront très légèrement, mais le principe du classifieur est le même. On cherche cette fois à identifier, à partir d'un morceau de musique, à quelle genre musical il appartient. On distingue six genres musicaux (la classification provient de la plate-forme musicale d'où les extraits ont été tirés) :

⓪ « *Electronic music* » ;
① « *Latin music* » ;
② « *Pop music* » ;
③ « *Rythm'n Blues* » ;
④ « *Rap* » ;
⑤ « *Rock* ».

On dispose d'une base de données étiquetée construite à partir de 32833 morceaux musicaux provenant de la plate-forme musicale, environ 5000 morceaux pour chacun des genre. À chaque morceau, on a associé une liste de dix paramètres numériques, qui peuvent être mesurés automatiquement :

- *danceability* : estimation du caractère « dansant » de l'extrait musical (entre 0, peu, et 1, très) ;
- *energy* : estimation de l'« énergie » de l'extrait (entre 0 et 1)
- *loudness* : volume sonore de l'extrait, en décibels ;
- *mode* : caractère mineur ou majeur de l'extrait (0 représente un morceau en mineur, 1 en majeur) ;
- *speechiness* : présence de texte dans l'extrait (entre 0, sans, et 1, en quantité, une valeur de 0.7 correspond à un extrait contenant quasiment tout le temps du texte) ;
- *acousticness* probabilité que l'extrait soit acoustique (entre 0 et 1) ;
- *instrumentalness* probabilité que l'extrait soit instrumental (entre 0 et 1) ;
- *liveness* probabilité que l'extrait soit enregistré avec un public (entre 0 et 1) ;
- *valence* caractère positif (joyeux...) ou négatif (triste) de l'extrait (entre 0 et 1) ;
- *tempo* rythme en battements par minute.

1.2 Chargement des données

Tout d'abord, on téléchargera le fichier contenant les informations utiles depuis cette adresse :

```
http://cdn.sci-phy.org/psi/songs.csv
```

On placera le fichier téléchargé dans la racine du disque « Données » de l'ordinateur (généralement `D:`)

On charge ensuite la base de données en Python avec :

```python
songs = []

for line in open("D:/songs.csv", encoding="utf8"):
    name, genre, *data = line.split(",")
    songs.append( (name, int(genre), [float(x) for x in data]) )
```

Dans l'objet nommé `songs` ainsi obtenu, on trouve 32833 triplets (un par extrait musical) constitués d'un titre, d'un entier entre 0 et 5 inclus (le genre de l'extrait) et d'une n-uplet de dix flottants, correspondant aux valeurs des dix paramètres mentionnés tantôt.

2 Classification

2.1 Prétraitements

On souhaite tout d'abord, dans un premier temps, réorganiser les données pour faciliter les traitements par la suite.

1. Construire un dictionnaire `data` dont les clés sont les entiers de 0 à 5 (inclus), une pour chaque genre, et les valeurs associées la listes des couples (titre, n-uplet de paramètres) pour les extraits du genre considéré

On doit avoir les résultats suivants :

```
>>> len(data)
6

>>> len(data[0])
6043

>>> len(data[0][0])
2

>>> len(data[0][0][1])
10
```

```
>> data[0][0][1]
('Disease - KATFYR Tokyo Bound Remix',
 [0.677, 0.926, 0.0, -3.307, 1.0, 0.0456,
     0.00298, 0.507, 0.337, 0.108, 128.014])

>> data[0][177]
('Step Back (VIP Mix)',
 [0.822, 0.762, 1.0, -6.198, 1.0, 0.0574,
     0.00103, 0.039, 0.333, 0.767, 125.0])
```

On pourra vérifier que l'on dispose bien d'entre 4951 et 6043 exemples dans chaque « classe ». L'étape suivante consiste à construire un groupe de données qui serviront de représentants pour l'algorithme des k plus proches voisins, et un groupe de données qui serviront à tester les résultats.

On souhaite conserver 500 morceaux de musique pour chaque genre comme tests, le reste servant de référence pour la reconnaissance. En théorie, il conviendrait de les sélectionner au hasard. Pour faciliter les échanges entre vous (et avoir les mêmes résultats), nous allons exceptionnellement omettre cette étape [1] (les exemples de la base ne sont de toute façon pas rangés dans un ordre particulier). Ainsi, pour chaque classe avec N morceaux de musique :

- les N – 500 premiers morceaux serviront de « références » ;
- les 500 derniers morceaux serviront pour nos tests.

2. Construire un dictionnaire `ref` dont les clés sont les entiers de 0 à 5 et tel que `ref[i]` contienne des couples (titre, n-uplet de taille 10) de `data[i]` correspondant aux morceaux de musique de référence.

3. Construire de même un dictionnaire `test` dont les clés sont les entiers de 0 à 5 et tel que `test[i]` contienne les 500 derniers couples (titre, n-uplet de taille 10) de chaque `data[i]`.

2.2 Définition de la distance

Pour mesurer (le carré de) la distance entre deux objets 1 et 2, décrits par les n-uplets, on utilisera la formule suivante :

$$d^2 = \sum_{i=0}^{9} (a_i(v_2 - v_1))^2$$

où les a_i sont les valeurs suivantes :

```
Ai = [7, 5.5, 0.3, 0.3, 2, 10, 4.5, 4.5, 6.5, 4.3, 0.04]
```

1. Cela resterait en principe possible même avec un mélange avec les fonctions pseudo-aléatoires dont on dispose en forçant la graîne aléatoire, mais c'est plus délicat à garantir.

Ces valeurs, correspondant aux inverses des écarts-type de chacune des donnée dans les n-uplets, ont pour vocation de donner un poids équivalent à chacun des paramètres.

4. Proposer une fonction `dist2(d1, d2)` retournant un flottant représentant le carré de la distance euclidienne entre deux couples issus de `ref[i]` ou `test[i]`.

On pourra tester la fonction précédente de la sorte (on remarquera au passage que deux morceaux d'un même genre sont plus proches que deux morceaux de genres distincts) :

```
>>> dist2(ref[0][0], ref[0][1])
11.95825359194641

>>> dist2(ref[0][0], ref[1][0])
34.83255596063225

>>> dist2(ref[0][0], test[0][0])
10.428927649699999
```

2.3 Calcul des distances

Pour un morceau dont on souhaite déterminer le genre à partir des données mesurées (n-uplet de taille 10), on doit déterminer sa « distance » à chacun des extraits « référence ».

5. Proposer une fonction `toutes_distances(p, ref)` qui prend en argument un couple (titre, n-uplet de longueur 10) `p` et un dictionnaire `ref` contenant les données des morceaux de référence, et retournant une liste de couples `(d2, k)` où `d2` représente le carré de la distance euclidienne à un des chiffres de référence, et `k` est l'entier (entre 0 et 9 correspondant à ce chiffre.

6. Modifier la fonction précédente pour que la liste retournée soit triée par ordre croissant des distances (on pourra utiliser `List.sort`).

On pourra tester la fonction de la façon suivante pour obtenir les 10 plus proches voisins pour l'extrait de test `test[0][0]` (donc de genre 0) :

```
>>> toutes_distances(test[0][0], ref)[:10]
```

7. Faire quelques essais à la main pour voir si le classement est relativement satisfaisant.

2.4 Classifieur et analyse des résultats

8. Écrire une fonction `gagnant(lst)` prenant en argument une liste d'entiers entre 0 et 5 (inclus), et retournant l'entier apparaissant le plus grand nombre de fois dans cette liste (en cas d'égalité, on retournera, parmi les ex-aequos, celui qui apparaît en premier dans la liste). On s'efforcera d'obtenir une complexité linéaire en la taille de la liste.

9. En s'appuyant sur les fonctions `toutes_distances` et `gagnant`, proposer une fonction `kppv(p, ref, k)` prenant en argument un couple (titre, n-uplet de longueur 10) `p`, le dictionnaire des références et un entier k, et retournant l'entier entre 0 et 5 correspondant à une classification avec l'algorithme des k plus proches voisins sur les données de référence `ref`.

10. Sur les 3000 extraits de notre ensemble de tests, combien sont bien classés pour $k = 1$? $k = 17$? Attention, le calcul est long, on pourra afficher le nombre d'extraits bien classés tous les 100 extraits traités pour vérifier le bon déroulement de la fonction.

11. Construire la matrice de confusion (de taille 6×6) de notre classifieur pour $k = 17$. Quels sont les genres les mieux identifiés? Les plus souvent confondus?

En remplaçant certains des coefficients a_i par 0 dans la fonction `dist2`, on peut désactiver l'influence de tout ou partie des paramètres, ce qui peut diminuer l'efficacité de la classification ou l'augmenter

12. Faire quelques essais pour essayer de déterminer les paramètres les plus pertinents parmi les dix paramètres fournis.

2.5 Choix de k

On souhaite déterminer l'évolution du nombre de chiffres bien classés en fonction de k. Il est possible d'utiliser les fonctions précédentes, mais cela prendrait un peu de temps. On va procéder un peu différemment.

13. Proposer une fonction `majoritaire(lst)` prenant en argument une liste d'entiers `lst` et retournant une liste `res` de même longueur, telle que `res[i]` contienne l'entier le plus représenté dans `lst` aux positions d'indice `0` à `i` inclus (en cas d'égalité, on choisira l'entier qui apparaît en premier).

14. En déduire une fonction `classification(d, ref)` qui prend en argument un couple (titre, n-uplet de longueur 10) et un dictionnaire contenant les références, et retournant une liste `res` de 30 entiers telle que `res[i]` contienne le résultat de la classification par k plus proches voisins pour $k = i + 2$ (k varie donc de 2 à 31 inclus).

15. Construire une liste `resultats` de 3000 listes de 30 éléments, contenant les listes construites par la fonction précédente pour les 3000 morceaux servant aux tests. Les 500 premiers éléments correspondront au premier genre, les 500 suivants au deuxième, etc.

La construction de cette liste prend du temps, on s'efforcera de préserver la liste ainsi construite pour la suite!

16. Proposer une fonction `taux_succes(resultats, k)` qui retourne le pourcentage de classement correct en fonction de k (k étant un entier entre 2 et 31).

17. Tracer la courbe du taux de succès en fonction de `k`. Quel `k` peut-on choisir?

18. Proposer une fonction `matrice_confusion(resultats, k)` retournant une liste `conf` de 6 listes contenant 6 entiers, telle que `conf[i][j]` contienne le nombre d'imagettes correspondant au chiffre `i` reconnues comme le chiffre `j` pour la méthode des k plus proches voisins (k étant un entier entre 2 et 49)

19. Quels sont les genres les plus souvent confondus pour le k choisi?