TP: Reconnaissance automatique des instruments de musique

Slim Essid, Olivier Gillet

7 février 2013

1 Introduction

La programmation pour ce TP sera effectuée sous Matlab. Un texte d'introduction à Matlab est disponible à l'adresse :
`http://www.tsi.enst.fr/~blanchet/MatlabF/docs/IntrodMatlab.pdf`

1.1 Principe

La majorité des systèmes de reconnaissance automatique des instruments de musique sont fondés sur des méthodes statistiques d'apprentissage. Tout d'abord, la classification est manuellement effectuée (par un opérateur humain) sur une collection de morceaux de référence dite *base d'apprentissage*. Un vecteur de paramètres (réels) est extrait de chaque morceau à classifier, ainsi que des morceaux de la base d'apprentissage. Ces paramètres décrivent différentes propriétés perceptuelles des signaux de musique. Un algorithme de classification est entraîné sur la base d'apprentissage, puis appliqué aux données à classifier.

1.2 Préparatifs

1. Télécharger dans votre répertoire de travail l'archive `tp-reco-inst.zip` à partir de `http://www.enst.fr/%7Eessid/resources.htm` ;
2. décompresser cette dernière ;
3. ajouter le répertoire cepstrum au *path* de Matlab.

2 Base de données

Nous utiliserons dans ce TP une base de données de 60 signaux musicaux, chacun d'entre eux étant un extrait de solo de l'un des 4 instruments suivants : clarinette, piano, trompette et violon (il y a 15 morceaux par instrument). De manière à tester les performances de l'algorithme de classification, le protocole suivant est utilisé :

- La base est divisée en 10 groupes de 6 signaux.
- L'algorithme de classification est entraîné sur les groupes 2 à 10 ; et la classification est effectuée sur le groupe 1. Il est donc possible de comparer, au sein du groupe 1, les instruments obtenus par classification automatique avec les instruments connus.
- L'étape précédente est répétée 9 fois en faisant tourner les ensembles d'apprentissage et de test.

La structure de données centrale utilisée pour ce TP est stockée dans la variable `database`. Ses différents membres sont :

- `features` : matrice 60×20 contenant, pour chaque morceau, un vecteur de 20 paramètres.
- `nfeatures` : matrice 60×20 contenant les paramètres sous forme normalisée.

– `instruments` : vecteur colonne 60×1 indiquant l'instrument joué dans chaque extrait musical, obtenu manuellement (1 : clarinette, 2 : piano, 3 : trompette, 4 : violon).
– `classifications` : vecteur colonne 60×1 indiquant l'instrument joué dans chaque extrait musical, obtenu par classification automatique.
– `filenames` : liste des noms des fichiers `.wav` contenant chacun des signaux.

3 Calcul des paramètres

Un vecteur de 20 paramètres décrivant le contenu du signal musical est extrait pour chaque morceau. Ces paramètres sont :

1 Le taux de passage par zéro (*zero crossing rate*), indiquant le nombre de fois par seconde où le signal, dans sa représentation temporelle, passe par la valeur centrale de l'amplitude (zéro pour un signal centré).

2-5 les 4 paramètres de *forme spectrale* obtenus à partir des 4 premiers moments du spectre d'amplitude, décrivant respectivement le centre de gravité spectral (*centroïde*), la largeur spectrale *(écart type*), l'asymétrie spectrale (*skewness*) et la platitude spectrale (*kurtosis*) ;

6-10 les 5 caractéristiques spectrales suivantes : l'oscillation du spectre, la pente du spectre, les 2 premiers coefficients du filtre Auto Régressif (excepté la constante 1) obtenu par analyse LPC (*Linear Prediction Coding*) du signal et la fréquence de coupure du spectre ;

11-20 10 coefficients MFCC.

La fonction `compute_features` calcule le vecteur de paramètres associé à chaque morceau de musique dans la base en moyennant, sur toute la longueur de chaque signal, les paramètres dits instantanés qui sont extraits sur des fenêtres d'analyse glissantes (de longueur 32 ms et avec un pas d'avancement de 16 ms, *i.e.* avec un taux de recouvrement de 50% des fenêtres successives). Cette fonction fait appel à différentes fonctions auxiliaires qu'il vous faudra compléter :

1. compléter la fonction `fc = features_freqCutoff(Spec,sr,N)` qui renvoie la fréquence de coupure moyenne d'un signal : cette fréquence est celle en dessous de laquelle 99% de l'énergie spectrale est prise en compte ;
2. compléter l'appel de la fonction `melcepst` dans `ss=features_mfcc(Frames,Spec,N,sr)` (voir répertoire `cepstrum`) pour l'extraction de 10 coefficients MFCC (excluant le coefficient 0), en réutilisant les spectres précédemment calculés et en choisissant pour les paramètres restants les paramètres suggérés par défaut dans l'aide de la fonction `melcepst.m`.

Les autres fonctions auxiliaires features_xxx calculent les paramètres restants. Une fois cette partie terminée et testée, lancer le script une dernière fois, puis modifier la ligne `partie_1_finie = 0` en `partie_1_finie = 1`. Cela évite le recalcul des paramètres dans les parties suivantes.

4 Normalisation

Les paramètres calculés à l'étape précédente sont d'ordres de grandeur variés, et occupent des plages de valeurs très diverses - ce qui peut perturber le fonctionnement des algorithmes de classification et d'apprentissage. L'étape de normalisation rémedie à ce problème en appliquant une transformation affine aux valeurs de chaque attribut.

1. Soit x_i l'ensemble des valeurs prises par l'attribut d'indice i. On peut restreindre les valeurs de cet attribut à l'intervalle $[-1, 1]$ par la transformation $x'_i = 2\frac{x_i - \min x_i}{\max x_i - \min x_i} - 1$.
   Quels peuvent être les inconvénients d'une telle méthode ?
2. La standardisation consiste à appliquer pour chaque attribut la transformation $x'_i = \frac{x_i - \mu_i}{\sigma_i}$, où μ_i est la moyenne des x_i (fonction `mean`) ; et σ_i leur écart type (fonction `stdev`).

Justifier l'intérêt de cette transformation et compléter la fonction `function Y = standardize(X)` renvoyant une matrice de paramètres standardisés à partir de la matrice de paramètres X.

5 Classification par l'algorithme du plus proche voisin

Pour rappel, l'algorithme du plus proche voisin (nearest neighbour, ou 1-NN) est le suivant :
- Calculer la distance euclidienne entre le vecteur de paramètres du morceau à classifier ; et les vecteurs de paramètres de chacun des morceaux dans la base d'apprentissage.
- Sélectionner le morceau de la base d'apprentissage pour lequel cette distance est minimale.
- Renvoyer l'instrument de ce morceau.

1. Implémenter cet algorithme en complétant la fonction :

```
function class = knn_classify(vector, model)
  X = model.X; // vecteurs de paramètres des morceaux de la base
                  d'apprentissage (54 x 20)
  Y = model.Y; // instruments des morceaux de la base d'apprentissage (54x1)
  // ...
endfunction
```

6 Evaluation

En plus de l'algorithme de classification par la méthode des plus proches voisins qui vient d'être implémenté, les fonctions `gaussian_classify` et `gaussian_train` utilisant un modèle Gaussien sont données. Ces deux méthodes de classification peuvent donc être évaluées et comparées. La fonction `cross_validation` effectue la validation croisée (les 10 rotations d'apprentissage / test) pour un couple d'algorithmes de classification et d'apprentissage donnés. La fonction `display_results` analyse et affiche les résultats. Les résultats affichés sont :
- Le pourcentage de morceaux correctement classés.
- La matrice de confusion. Les éléments a_{ij} de cette matrice correspondent au nombre de fois où un morceau faisant intervenir l'instrument i a été classé dans la classe de l'instrument j. Par exemple, la matrice :

$$\begin{bmatrix} 10 & 2 & 3 & 0 \\ 0 & 14 & 1 & 0 \\ 0 & 0 & 15 & 0 \\ 0 & 1 & 0 & 14 \end{bmatrix}$$

Se lit : "Parmi les morceaux faisant intervenir l'instrument 1 (clarinette), 10 ont été correctement classés, 2 ont été classés dans la classe de l'instrument 2 (piano), 3 dans l'instrument 3 (trompette). Parmi les morceaux faisant intervenir l'instrument 2 (piano), 14 ont été correctement classés, et 1 seul a été classé dans l'instrument 3 (trompette)..."

1. Comparer les performances des deux algorithmes (1-NN et modèle Gaussien).
2. Identifier les morceaux mal classifiés. Ecouter les signaux correspondant, et tenter de justifier les erreurs faites par le classifieur.
3. Il est possible de n'utiliser pour la classification et l'apprentissage qu'un sous-ensemble des 35 paramètres calculés. Cela se fait en modifiant la ligne commençant par `database.nfeatures =` en `database.nfeatures = standardize(database.features(:,[1 5:8]));` (Ici, on n'utilise que les paramètres 1, 5, 6, 7, 8). Quelle est la famille de paramètres parmi ceux cités en 3 qui, utilisée seule, donne les meilleurs résultats en classification ? Illustrer cette discussion à l'aide de la fonction

`plotmatrix(database.nfeatures, database.instruments)` qui trace un tableau (i, j) de graphes contenant :
- En (i, i), la distribution du paramètre i pour chacune des classes.
- En (i, j) avec $j > i$, les nuages de points pour le couple de paramètres (i, j) et chacune des classes.

7 Améliorations possibles

1. **Modèle de rejet, détection de nouveauté.** Exécuter la commande `test_piccolo(database)`. Cette fonction entraîne les deux algorithmes de classification sur l'ensemble de la base puis classifie un morceau de piccolo : les classes renvoyées par chacun des deux algorithmes sont affichées. Expliquer les résultats. Dans l'idéal, on aimerait que les morceaux d'un instrument inconnu soient signalés comme tels. Effectuer une modification simple de l'algorithme du plus proche voisin pour qu'il effectue une telle discrimination (La classe *instrument inconnu* sera numérotée 5). Faire de même pour le classificateur par modèle Gaussien.
2. k **plus proches voisins.** L'algorithme du plus proche voisin n'est pas robuste au bruit. On lui préfère souvent l'algorithme des k plus proches voisins. Cet algorithme consiste à attribuer au vecteur à classifier la classe la plus représentée parmi les k points les plus proches. Modifier la fonction `knn_classify` pour implémenter l'algorithme des 4 plus proches voisins.
3. **Autres paramètres (BONUS)** Proposer et tester d'autres paramètres extraits facilement du signal qui pourraient être utilisés pour la reconnaissance des instruments de musique (pensez aux caractéristiques de votre instrument préféré...).
4. **Réduction de dimensionnalité (BONUS)** Augmenter le nombre de paramètres permettrait de raffiner la description des signaux, et donc d'améliorer les résultats en classification. Cependant, cela augmenterait également la complexité du système, sa sensibilité au bruit, et introduirait des redondances entre paramètres. On aimerait pouvoir obtenir, par combinaison linéaire des paramètres originaux, un ensemble de paramètres réduit qui préserverait l'information des paramètres originaux tout en éliminant la redondance. Connaissez-vous des méthodes statistiques pour effectuer cette réduction de dimensionnalité ?