Mini-Projet de traitement du signal
Reconnaissance d'extraits musicaux

B. Figliuzzi

2023-24

L'objectif de ce mini-projet est d'implémenter un algorithme de reconnaissance musicale à partir d'outils classiques de traitement du signal. Le principe général des algorithmes de reconnaissance musicale est d'associer à chaque morceau de musique une empreinte caractéristique ou *signature*, puis de créer une base de données contenant les empreintes calculées pour plusieurs millions de morceaux. Pour reconnaître un extrait musical, l'algorithme procède en calculant une empreinte pour cet extrait, puis en cherchant une correspondance entre l'empreinte de l'extrait et l'une des empreintes de la base de donnée.

Dans le cadre de ce mini-projet, nous travaillerons avec une base de morceaux très simplifiée contenant quelques morceaux libres de droits (extraits de la bibliothèque d'audios libres de droits de YouTube), disponible dans les fichiers attachés au projet. Vous pourrez vous appuyer sur les fichiers python, qui contiennent une pré-structure pour l'implémentation sous forme de classes Python, ou repartir de zéro pour l'implémentation selon votre préférence. Les livrables du mini-projet sont le code et un rapport concis contenant les réponses aux questions ci-dessous et les figures demandées.

1. L'algorithme que nous allons étudier, décrit dans l'article [1] référencé ci-après, s'appuie sur le spectrogramme afin d'extraire une signature caractéristique du morceau. Dans la classe Encoding du fichier algorithm.py, implémenter le calcul du spectrogramme pour un morceau de musique, ainsi qu'une fonction permettant d'afficher le spectrogramme. On pourra utiliser pour cela la libraire scipy.signal. L'énergie des morceaux de musique se répartit-elle de manière uniforme sur le spectrogramme? En partant de l'un des morceaux de musique de la base, déterminez le nombre minimal de coefficients du spectrogramme permettant de concentrer 90% de l'énergie du signal. Pour le calcul du spectrogramme, on pourra utiliser une taille de fenêtre de

128 échantillons, avec un recouvrement de 32 échantillons entre deux fenêtres successives.

2. Le nombre de coefficients du spectrogramme reste trop important pour permettre la comparaison directe d'un extrait musical à des millions de morceaux. Un morceau donné est caractérisé par la présence de fréquences bien spécifiques à des instants donnés. Au niveau du spectrogramme, la présence de ces fréquences se traduit par des maxima locaux dans le plan temps-fréquence. On peut donc simplifier la représentation du morceau en considérant les maxima du spectrogramme séparés d'une distance minimale en temps et en fréquence. L'ensemble de ces maxima constitue ce qu'on appelle une *constellation* et permet de condenser l'information du spectrogramme en un ensemble parcimonieux de coordonnées de points de l'espace temps-fréquence. Implémentez dans la méthode process de la classe Encoding le calcul de la constellation à partir du spectrogramme. Pour ne sélectionner que les maxima locaux dans une fenêtre temps/fréquences donnée, on pourra utiliser la fonction peak_local_max de la librairie scikit-image. On pourra fixer à 50 (pixels) la distance minimale entre maxima du spectrogramme, donnée en paramètre d'entrée à la fonction peak_local_max

3. La signature caractéristique du morceau considéré est obtenue en associant entre eux des points de la constellation. Supposons ainsi qu'on sélectionne un point de la constellation $a = (t_a, f_a)$, que nous appellerons une *ancre* dans la suite. On peut associer à l'ancre a l'ensemble des autres points $c_i = (t_i, f_i)$ de la constellation - appelés *cibles* - tels que $0 < t_i - t_a \leq \Delta t$ et $|f_a - f_i| < \Delta f$. Pour chaque ancre, on obtient via cette procédure un ensemble de couples ancre/cible $(c_a, c_i)_{i \in I}$. Pour chacun de ces couples, on peut finalement créer un *code de hachage* en gardant en mémoire:

   - l'intervalle de temps $t_i - t_a$ qui sépare l'ancre et la cible,
   - la fréquence f_i associée à l'ancre,
   - la fréquence f_i associée à la cible.

   Ces quantités sont enregistrées dans un vecteur $v_{i,a} = (t_i - t_a, f_a, f_i)$. On leur associe le temps t_a correspondant à la coordonnée en temps de l'ancre. L'ensemble des codes de hachage $(t_i - t_a, f_a, f_i)$ et leurs temps associés t_a vont constituer la signature d'un morceau et/ou d'un extrait de morceau. On pourra choisir les valeurs $\Delta t = 1$s et $\Delta f = 1500$ Hz.

   Implémentez, toujours dans la fonction process de la classe Encoding, une fonction permettant d'extraire la signature d'un morceau sous la forme d'une liste de codes de hachage. La représentation sous forme de code de hachage est-elle invariante par translation?

4. En exécutant le script "database.py", encoder l'ensemble des morceaux afin de créer un fichier de données contenant les signatures de chaque morceau.

5. Lorsqu'on cherche à comparer un extrait musical à un morceau, on commence par encoder l'extrait musical en utilisant la méthode décrite précédemment. On cherche ensuite à établir des correspondances entre les code de hachages. En particulier, on cherche l'ensemble des codes de hachage $(t_i - t_a, f_a, f_i)$ de l'extrait qui sont égaux aux codes de hachage $(\tilde{t}_i - \tilde{t}_a, \tilde{f}_a, \tilde{f}_i)$ du morceau, cela pour chaque morceau de la base de données. On peut ensuite créer un nuage de points en considérant les couples $(t_a, \tilde{t}_a)$ associés aux codes de hachages identifiés comme étant en correspondance. La classe Matching du fichier algorithm.py permet d'établir des correspondances entre les codes de hachage d'un extrait et ceux d'un morceau. Représentez le résultat des correspondances sous la forme d'un nuage de points prenant les temps associés aux codes de hachage en entrée, en utilisant la fonction display_scatterplot de la classe. Comment se structure le nuage de points lorsque l'extrait correspond au morceau? Lorsque l'extrait ne correspond pas au morceau?

6. Représentez l'histogramme des quantités $t_a - \tilde{t}_a$ pour l'ensemble des correspondances à l'aide de la fonction display_histogram de la classe Matching. Comment se structure l'histogramme de points lorsque l'extrait correspond au morceau? Lorsque l'extrait ne correspond pas au morceau?

7. A partir de l'histogramme des $t_a - \tilde{t}_a$, proposez un critère permettant de décider si un extrait appartient ou non à un morceau de la base de donnée. Compléter l'implémentation de la classe.

8. Evaluez la performance de votre algorithme sur la base de donnée. Pour ce faire, vous pourrez vous appuyer sur le code pré-implémenté dans le fichier demo.py

References

[1] Avery Wang et al. An industrial strength audio search algorithm. In *Ismir*, volume 2003, pages 7–13. Citeseer, 2003.