Rapport Projet Alignement - ATIAM

Guilhem Marion

2 novembre 2018

1 Q-1.1 Implement the Needleman-Wunsch (NW) algorithm

L'algorithme a été implémenté dans la fichier *functions.py*, il prend comme paramètres un fichier contenant la matrice de distances (une fonction d'aide a été faite pour extraire la matrice des fichiers fournis) et les valeurs des gaps. Un autre fichier *testFunctions.py* a été fait afin d'implémenter des jeux de tests : des mots sont tirés aléatoirement et leur alignement est calculé à l'aide de la fonction fournie par nwalign et la nôtre, les résultats sont comparés. On obtient 100% de bon retours de scores et un peu moins d'exacts mêmes alignements, cela est tout à fait normal étant donné que la construction de l'alignement dépend du chemin à trouver dans la matrice et qu'il peut en exister plusieurs.

Une option a été ajoutée permettant d'utiliser une fonction afin de calculer les distances entre caractères, par exemple l'argument *matrix="Linear"* permet de calculer une distance binaire, et ce, pour n'importe quel caractère. Des arguments optionels *bonus* et *malus* permettent alors de choisir la valeur de la distance en cas d'adéquation ou non.

2 Q-2.1 Sort the collection of composers by decreasing number of tracks

On construit le dictionnaire *dico[compositeur] = nb de pièces* et on le trie par valeurs. Le code est dans *atiam-fpa.py*.

3 Q-2.2 Apply the NW algorithm between all tracks of each composer

Cet algorithme, permettant d'évaluer la ressemblance des chaines de caractères, n'est pas très efficace, en effet *prélude no.1* et *prélude no.2* vont forcément correspondre alors que *premier prelude* et *prélude no.1* en auront beaucoup moins de chances.

On décide donc de choisir un seuil qui permettra de laisser passer des exemples comme *prlude n.1* et *prélude no.1* mais aussi *prélude no.1* et *prélude no.2*. Pour cela on choisit un seuil qui dépend de la taille de la plus petite chaine de caractères, on prend la formule suivante :

$$0.8 \cdot min(|chaine1|, |chaine2|) \cdot 5 - 0.2 \cdot min(|chaine1|, |chaine2|) \cdot 4$$

Cela correspond intuitivement au fait que 80% de l'alignement est juste, ce qui justifie une faute de frappe ou une legère variation. Une liste des correspondances est présente dans le fichier *Data/matches1.p*.

4 Q-2.3 Extend your previous code so that it can compare

Afin de passer le calcul à l'echelle d'une base de données conséquente (37 426 entrées), il faut trouver une astuce. En effet, notre algorithme est en $\mathcal{O}(m \cdot n)$ avec m la taille de la première chaine et n la taile de la seconde. Les chaines étant de tailles sensiblement égales, on peut généraliser à $\mathcal{O}(n^2)$ avec n la taille moyenne des entrées, quadratique donc. Pour opérer le test d'adéquation de chacun des titres avec tous les autres on obtient $n + n - 1 + n - 2... + 1 = \frac{n \cdot (n+1)}{2}$ opérations. Soit quadratique en n le nombre d'entrées dans la base. On est donc en $\mathcal{O}(n^2 \cdot m^2)$, avec n le nombre d'entrées et m la taille moyenne des mots, ce qui n'est pas raisonnable.

Pour résoudre ce problème, prenons notre valeur seuil, et considérons des règles simples qui nous permettent de nous assurer qu'elle ne sera pas atteinte. Nous choisissons deux règles :

1. Si les mots ont des tailles trop differentes, on considère qu'ils ne pourront pas correspondre.
2. On calcule l'occurence des chacune des lettres de chacun des mots, on en calcule l'erreur quadratique moyenne, si elle est trop grande on estime que les mots ne pourronts pas correspondre (implémenté dans *checkFirstSimilarities()*).

Cela permet de reduire considérablement le temps de calcul, et surout de pouvoir effectivement procéder au calcul. Les fonctions d'aides sont présentes dans *functions.py* et l'application à la base de données dans *atiam-fpa.py*. Le pickle contenant les mots correspondant se trouve dans *Data/matches2.p*.

5 Q-3.1 Extending to a true musical name matching

Afin de s'adapter aux noms de pieces de musique classique et de pouvoir renvoyer des résultats cohérants (deux chaines correspondent si et seulement si elles réfèrent à la même pièce) il nous faut trouver une autre façon de procéder, en voici les idées clefs :

— Certains compositeurs comme Mozart ou Bach possédent un catalogue (respectivement Kochel et Bach-Werke-Verzeichnis) qui permettent d'identifier de manière unique chacune de leurs oeuvres.
— Les nombre présents dans les titres sont d'une extrême importance (prelude no.1 et prelude no.2 se réfèrent à des choses très differentes).
— Un système de règles peut permettre de désambiguïser cetains symboles très importants (La mineur est la même chose que a minor)

Notre algorithme utilise donc la procèdure suivante :

1. Si on trouve un indice de catalogue (KV ou BWV) on effectue la comparaison sur celui-ci
2. Sinon, on remplace les caractères confondant comme les tirets ou les parenthèses, puis on passe tout en minuscules
3. On découpe le titre par mots, et on procède à la comparaison mot à mot (dans tous le sens possibles mais un mot ne peut correspondre qu'à au plus un mot).

4. La comparaison se fait à partir du système de règles rapprochant par exemple (1, un, one, premier) ainsi que les tonalités, les dièses et les bémols.
5. Si le mot est un nombre la comparaison est très stricte (1 s'ils sont égaux, 0 sinon)
6. Si on a pas réussi à comparer avec les règles précédentes on utilise Needleman avec le même seuil que précédemment afin de ne pas tenir compte des fautes de frappe.
7. Si tous les mots de la plus petite chaine de caractères correspondent, on accepte.

L'algorithme est présent dans *functions.py*, quand on l'applique aux exemples de tout à l'heure on s'aperçoit qu'il y a beaucoup moins de correspondances, mais qu'elles sont beaucoup plus pertinentes qui étaient précédemment noyées dans la masse. Quelques examples :

— O God, Who by the Leading of a Star
o_god_who_by_the_leading

— Would my conceit that first enforcd my woe (First Book of Songes and Ayres XVI),
Would my conceit that first enforst my woe

— Carmen - Prelude,
'Carmen Prelude

6 Q-4.1 Import and plot some MIDI files

On utilise la librairie *Music21*, le code est dans *atiam-fpa.py*

7 Q-4.2 Exploring MIDI properties

On utilise quelques propriétés que propose *Music21*, on s'intéresse à des representations temporelles des pièces qui nous permettent de voir s'il y a du bruit quand au placement rythmique.

8 Q-5.1 Automatic evaluation of a MIDI file quality

On définit la qualité d'un fichier midi comme le fait qu'il n'y ait pas ou peu de bruit rythmique : que les notes soient jouées sur le temps et non un peu avant, un peu après. Il faut pour cela s'interesser à la quantification du rythme. Nous optons pour la solution suivante :

1. On quantifie le même morceau avec deux niveaux de quantifications : un très élévé et un autre correspondant à la durée la plus courte du morceau.
2. On trie les début de notes en fonction du temps
3. On calcule l'erreur quadratique moyenne de la position normalisée de ces notes.

Si cette valeur est très grande cela veut dire que les notes sont jouées à des endroits imprévus (le fichier est donc enregistré par un musicien et non écrit à la main) et qu'après quantification on peut faire des erreurs. Il faut bien entendu prendre cette valeur avec précaution car, certes, en musique classique les partitions sont écrites généralement sur des parties simples et sans grandes ambiguités du temps, mais en Jazz des rythmes complexes (after the beat, swing, croches arrondies ...) peuvent apparaîtres qui ne sont pas synonymes d'imprécisions rythmiques, bien au contraire.

9 Q-6.1 Extending your alignment algorithm to MIDI scores

Dans le cas de comparaisons de tranches de piano roll, le calcul de distances est très aisé car nous n'avons à faire qu'à des 1 et des 0, une distance simple comme l'argument *matrix="Linear"* de notre fonction propose sera donc parfait.

L'algorithme est très simple, on applique Needleman sur chaque tranche de midi, on extrait l'alignement et on reconstruit la pièce. Cette méthode fonctionne à merveille mais a l'inconvénient d'être coûteuse en calcul (Needleman sur de grandes chaines coûte cher et il doit être lancé pour chaque note de chaque voix de la partition ...).

L'algorithme est implémenté dans *functions.py* et est mis en oeuvre dans *atiam-fpa.py* avec deux exemples très parlant et dont l'execution est très rapide.