INFO0902-1 Structures des données et algorithmes

Projet 3 : Résolution de problèmes

Bachelier en sciences de l'ingénieur, Bloc 2

Samuel Degueldre et Corentin Van Putte

2016 - 2017

Première partie
Analyse Théorique

1 DTW

1.1 Expliquez brièvement le principe de cet algorithme

L'algorithme consiste à calculer un alignement de deux séries temporelles en tenant compte d'éventuelles différences de vitesse entre les échantillons. Une fois cet alignement réalisé, on peut extraire une distance entre les deux échantillons.[1]

1.2 Discutez de l'intérêt des contraintes de localité

Le principe de la contrainte de localité est d'éviter que l'algorithme n'associe deux points d'une série s'ils sont trop éloignés, ce qui donnerait une distance DTW effectivement plus faible mais correspondrait à un alignement de séquences qui n'a pas de sens dans la realité. Celà permet également d'accélérer l'algorithme puisqu'il ne doit alors plus calculer de score d'association entre des points trop éloignés temporellement.[1]

1. Sources : https ://en.wikipedia.org/wiki/Dynamic_time_warping

1.3 L'algorithme DTW est basé sur la programmation dynamique Donnez la formulation récursive correspondante

$$DTWcosts[i,j] = \begin{cases} 0, & \text{si } i = 0 \textit{ et } j = 0 \\ +\infty, & \text{si } i = 0 \textit{ ou } j = 0 \\ basecosts+ & \text{sinon} \\ min(DTWcosts[i-1,j-1], & \\ DTWcosts[i,j-1], & \\ DTWcosts[i-1,j-1]), & \end{cases}$$

1.4 Donnez la complexité de l'algorithme en fonction des longueurs des signaux comparés et de la contrainte de localité

Soit S1 la longueur du premier signal, S2 la longueur du second signal et L la contrainte de localité.

En général, c'est à dire lorsque L > S2, l'algorithme est Θ(S1 $\times$ S2).

En particulier, c'est à dire lorsque L $\leq$ S2, l'algorithme est Θ(S1 $\times$ L).

On notera tout de même que la matrice du DTW doit être dans tous les cas initialisée. Cette matrice ayant une taille S1 $\times$ S2, l'initialisation de la matrice est Θ(S1 $\times$ S2).

2 Découpage optimal

2.1 Formulez M(n) de manière récursive en précisant bien le cas de base

$$M(n) = \begin{cases} +\infty, & \text{si } size \geq lMax \textit{ ou } size < 2 \times lMin \\ & \textit{et } size < lMin \\ & \textit{et } \frac{size}{lMin} < \frac{size}{lMax} \\ predictDigit(n, database, locality), & \text{sinon} \end{cases}$$

où size est la taille du sous-signal.

2.2 Déduisez-en le pseudo-code d'un algorithme efficace pour calculer cette découpe

```
BESTSPLIT(signal, database, locality, lMin, lMax)
1.  if signal.length / lMin < signal.length / lMax
2.          return +∞
3.  Let splitScores[i...signal.length, j...signal.length] be a matrix
4.  for i = 0 to signal.length
5.          for j = 0 to signal.length
6.                  splitScore[i,j].digit = -1
7.  returnSequence = BESTSPLITWRAPPER(signal, database, locality,
8.                                                  lMin, lMax, splitScores, 0,
9.                                                  signal.size - 1)
10. return returnSequence


BESTSPLITWRAPPER(signal, database, locality, lMin, lMax,
                                splitScores, start, stop)
1.  size = stop - start
2.  if size < lMin or size / lMin < size / lMax
3.          return +∞
4.  bSplit = +∞
5.  if size ≥ 2×lMin
6.          for i = start + lMin to start + lMax + 1 and signal.length
7.                  rightSize = stop - i
8.                  if rightSize / lMin ≥ rightSize / lMax
9.                                  and rightSize ≥ lMin
10.                         if splitScores[start,i].digit == -1
11.                                 leftDigit = SPLITSCORE(signal, database,
12.                                                                 locality, start, i,
13.                                                                 splitScores)
14.                         else
15.                                 leftDigit = splitScores[start,i]
16.                         bestRightSplit = BESTSPLITWRAPPER(signal,
17.                                                 database, locality, lMin, lMax,
```

```
18.                                         splitScores, i, stop)
19.                     newSplit = UNITE(leftDigit, bestRightSplit, start)
20.                     if newSplit.score < bSplit.score
21.                         bSplit = newSplit
22. if size < lMax
23.     if splitScores[start,stop].digit == -1
24.         seqScore = SPLITSCORE(signal, database, locality, start,
25.                                             stop, splitScores)
26.     else
27.         seqScore = splitScore[start,stop]
28.     if bSplit.score > seqScore.score
29.         bSplit.nDigits = 1
30.         bSplit.score = seqScore.score
31.         bSplit.digits[0] = seqScore.digit
32.         bSplit.splits[0] = start
33. return bSplit
```

```
    SPLITSCORE(signal, database, locality, start, stop, splitScores)
 1. if splitScores[start,stop].digit == -1
 2.     Let subsignal be a signal of size [start,stop]
 3.     splitScores[start,stop] = PREDICTDIGIT(subsignal, database,
 4.                                                 locality)
 5. return splitScores[start,stop]
```

```
    UNITE(dscore, seq, start)
 1. retSeq.nDigits = seq.nDigits + 1
 2. retSeq.score = dscore.score + seq.score
 3. retSeq.digits[0] = dscore.digit
 4. retSeq.splits[0] = start
 5. for i = 1 to retSeq.nDigits
 6.     retSeq.digits[i] = seq.digits[i-1]
 7.     retSeq.splits[i] = seq.splits[i-1]
 8. return retSeq
```

2.3 Analysez la complexité au pire et au meilleur cas de votre algorithme en fonction des paramètres les plus appropriés

L'algorithme ne possède ni meilleur cas, ni pire cas.
Sa complexité est $\Theta(\text{DTW} \times (lMax - lMin)^2)$

Comme dit précédément (au point 1.4), la complexité du DTW est $\Theta(S1 \times S2)$ quand L > S2 et $\Theta(S1 \times L)$ quand $L \leq S2$.

On en déduit donc que la complexité du découpage optimal est :

Si L > S2 : $\Theta(S1 \times S2 \times (lMax - lMin)^2)$
Si $L \leq S2$: $\Theta(S1 \times L \times (lMax - lMin)^2)$

Deuxième partie

Analyse Empirique

1 Vérification de la pertinence du DTW pour l'identification de chiffres isolés

Contrainte de localité	Nombre d'erreurs
0	50
1	40
2	37
3	29
4	21
5	15
6	12
7	11
8	10
9	9
10	3
11	1
12	1
13	1
14 ou plus	0

Tableau 1 – Nombre d'erreurs de reconnaissance en fonction de la contrainte de localité[2]

2. Cette analyse est basée sur 50 échantillons de tests, 5 pour chaque nombre de 0 à 9

On remarque qu'à partir d'une contrainte de localité de 14, le DTW ne commet plus aucune erreur sur les échantillons de test, et ainsi, augmenter encore la localité est inutile.

On remarque également qu'avec une contrainte de localité égale à 0, l'algorithme commet 50 erreurs, c'est à dire qu'il ne reconnait aucun signal. C'est parce que quand la contrainte de localité est fixée à 0, l'algorithme est capable de détecter le bon nombre que si et seulement si les deux signaux (le signal de test et le signal de la database) sont les mêmes.

2 Vérification du bon fonctionnement de l'algorithme de découpage

Séquence n°	Séquence obtenue
1	0 4 7 3
2	1 2 3 9
3	1 9 7 5
4	2 0 3 6
5	3 4 8 2
6	4 8 1 5
7	5 4 3 2
8	9 9 8 4

Tableau 2 – Séquence de chiffres obtenue en fonction du numéro de la séquence

Afin d'augmenter la rapidité de l'algorithme ainsi que sa précision, nous avons du imposer les paramètres suivants :
-locality = 25
-lMin = 35
-lMax = 70

Nous avons pris une contrainte de localité de 25 car c'était une valeur qui permettait un bon rapport précision/rapidité. A noter qu'une contrainte de localité plus petite permettrait une plus grande rapidité de l'algorithme mais réduirait sa précision, et inversément.

Nous avons pris l'intervalle [35 ;70] ([lMin ;lMax]) car les échantillons de la base de donnée sont compris dans cet intervalle. A noter que tous les échantillons n'ont pas une valeur proche des bornes de cet intervalle mais celà permet d'avoir une marge de sécurité, pour éviter qu'un signal ayant une valeur proche des bornes ne vienne fausser les résultats.