# TD : Analyse de séries temporelles

## 1 Introduction

Les séries temporelles sont un type particulier de données qui représentent des suites de valeurs ordonnées dans le temps. L'horodatage des données peut être explicite (une date est donnée pour chaque valeur) ou implicite (c'est l'ordre d'apparition des données qui est pris en compte).

Ces données peuvent par exemple représenter l'évolution des valeurs données par un capteur dans le temps (par exemple la température ou la pression), un électrocardiogramme, ou encore un son.

La classification de séries temporelles consiste à affecter une classe à une donnée en fonction de ses caractéristiques. Des électrocardiogrammes de patients peuvent par exemple être classés "normaux" ou "anormaux" en fonction de leurs caractéristiques.

Dans ce contexte, l'algorithme du plus proche voisin est souvent utilisé afin de classifier des séries temporelles. Étant donnée une série temporelle à classer, cet algorithme consiste à chercher la série temporelle "la plus proche" dans un ensemble de séries temporelles dont on connaît la classe. On possède ainsi deux ensembles de données, un premier, appelé "train" composé de séries temporelles dont la classe est connue, et une second, appelé "test" dont nous cherchons à prédire la classe.

## 2 Lecture des données

La première étape afin d'effectuer de la classification de séries temporelles consiste à lire les données. Commencez par récupérer 4 jeux de données à l'adresse suivante : http://germain-forestier.info/teaching/files/FD/data/timeseries.zip

Cette archive contient 4 jeux de données :

- 50words : représente le contour de mots : https://www.timeseriesclassification.com/description.php?Dataset=FiftyWords
- Coffee : représente le spectrographe de grain de café : http://timeseriesclassification.com/description.php?Dataset=Coffee
- ECG200 : représente des électrocardiogrammes : http://timeseriesclassification.com/description.php?Dataset=ECG200
- GunPoint : représente des mouvements : http://timeseriesclassification.com/description.php?Dataset=GunPoint

FIGURE 1 – Illustration du mouvement "Gun" du jeu de données GunPoint. © Eamonn Keogh

Pour chaque jeu de données, vous avez un fichier TRAIN et un fichier TEST. Ces fichiers contiennent respectivement les séries temporelles connues (TRAIN) et à classer (TEST).

Votre premier travail consiste à écrire un programme Python permettant de lire ces différents jeux de données.

# 3 Classification

## 3.1 Mesure de similarité

Afin d'implémenter l'algorithme du plus proche voisin, il est est nécessaire de choisir une mesure de similarité. Cette mesure a pour objectif d'évaluer de manière graduelle la proximité entre deux séries temporelles. La mesure de similarité la plus simple est la distance euclidienne. Elle consiste à calculer la racine de la somme des différences au carré des valeurs des deux séries : https://fr.wikipedia.org/wiki/Espace_euclidien.

Implémentez la distance euclidienne afin de pouvoir calculer cette distance entre deux séries temporelles.

## 3.2 Précision de classification

Afin d'évaluer si la distance choisie permet de classer correctement les séries temporelles, il convient de calculer la taux de bonnes classifications. Ce taux consiste à évaluer combien de fois l'algorithme a prédit correctement la classe d'une série, par rapport au nombre total de séries à classer.

Calculez le taux de bonnes classifications pour chacun des 4 jeux de données. Vos résultats doivent être identiques aux résultats présentés sur https://www.cs.ucr.edu/~eamonn/time_series_data/ à la colonne "1-NN Euclidean Distance".

# 4 Prise en compte des distorsions temporelles

Dynamic Time Warping (DTW) est un algorithme de comparaison de séries temporelles prenant en compte les possibles déformations temporelles entre les données (ce que n'est pas le cas de la distance euclidienne). Cet algorithm permet très souvent d'obtenir de meilleurs résultats que la distance euclidienne sur des données réelles, car celles-ci contiennent souvent des distorsions temporelles. De plus, DTW permet également de comparer des séries temporelles n'ayant pas la même longueur. DTW est calculé à l'aide d'un algorithme de programmation dynamique qui remplis récursivement une matrice d'alignement.

Voici le pseudo-code de DTW :

```
int DTWDistance(s: array [1..n], t: array [1..m]) {
   DTW := array [0..n, 0..m]

   for i := 1 to n
       DTW[i, 0] := infinity
   for i := 1 to m
       DTW[0, i] := infinity
   DTW[0, 0] := 0

   for i := 1 to n
       for j := 1 to m
           cost := d(s[i], t[j])
           DTW[i, j] := cost + minimum(DTW[i-1, j  ],    // insertion
```

```
                                        DTW[i   , j-1],     // deletion
                                        DTW[i-1, j-1])      // match

    return DTW[n, m]
}
```

Implémentez cet algorithme et étudiez le taux de bonnes classifications obtenu en utilisant DTW plutôt que la distance euclidienne pour la classification à l'aide du plus proche voisin. Vos résultats doivent être identiques aux résultats présentés sur http://www.cs.ucr.edu/~eamonn/time_series_data/ à la colonne "1-NN DTW, no Warping Window".

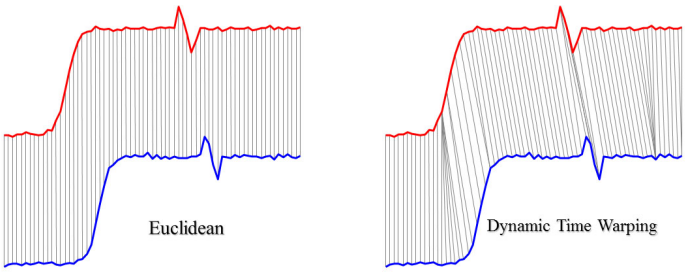


FIGURE 2 – Illustration de la différence entre la distance euclidienne et DTW. © Eamonn Keogh

## 5 Prise en compte de contraintes

Il peut arriver dans certains cas qu'il soit intéressant de limiter la quantité de distorsion allouée par l'algorithme DTW. Dans ce cas, on ajoute un paramètre supplémentaire, appelé $w$, qui limite le parcours du chemin d'alignement. Dans ce cas, la matrice d'alignement ne sera pas entièrement calculée.

Voici le pseudo-code de DTW avec la contrainte $w$ :

```
int DTWDistance(s: array [1..n], t: array [1..m], w: int) {
    DTW := array [0..n, 0..m]

    w := max(w, abs(n-m)) // adapt window size (*)

    for i := 0 to n
        for j:= 0 to m
            DTW[i, j] := infinity
    DTW[0, 0] := 0

    for i := 1 to n
        for j := max(1, i-w) to min(m, i+w)
            cost := d(s[i], t[j])
            DTW[i, j] := cost + minimum(DTW[i-1, j  ],     // insertion
                                        DTW[i   , j-1],     // deletion
                                        DTW[i-1, j-1])      // match

    return DTW[n, m]
}
```

Implémentez cet algorithme et écrivez un programme afin de trouver le meilleur $w$ pour chaque jeu de données. Vos résultats doivent être identiques aux résultats présentés sur http://www.cs.ucr.edu/~eamonn/time_series_data/ à la colonne "1-NN Best Warping Window DTW (r)".