## 1.1.2.2. Classification : algorithmes des séries temporelles

Nos jeux de données étant constitués d’observations pour plusieurs variables dans le temps, chacun d’entre eux peut être considéré comme une série temporelle multivariée.

Cependant, les algorithmes de scikit-learn supposent que les données soient structurées sous forme de tableau, et que chaque colonne représente des variables i.i.d. (indépendantes et identiquement distribuées).

Les séries temporelles ne remplissent pas ces critères car elles sont, par nature, caractérisées par la corrélation entre des observations proches dans le temps.

Afin de procéder aux modélisations de nos jeux de données en tant que séries temporelles, nous avons donc mobilisé sktime[[1]](#endnote-1), une bibliothèque qui étend l'API scikit-learn aux tâches d'apprentissage automatique avec des séries temporelles.[[2]](#endnote-2)

Parmi les algorithmes de classification implémentés par sktime, nous avons choisi 3 pour cette étude :

* K-Nearest Neighbors avec Dynamic Time Warping (KNN-DTW)
* Time Series Forest (TSF)
* RandOm Convolutional KErnel Transform (ROCKET)

Enfin, avant de construire les modélisations, quelques étapes complémentaires de prétraitement des données s’imposent afin d’assurer leur compatibilité avec sktime et garantir la cohérence méthodologique. Ces étapes sont détaillées dans la sous-section « Prétraitement complémentaire des données », présentée à la fin de cette section.

### K-Nearest Neighbors avec Dynamic Time Warping (KNN-DTW)

La méthode *k-nearest neighbors* (KNN) couplée à l’utilisation de la *dynamic time warping* (DTW, ou « déformation temporelle dynamique » en français) comme métrique de distance fait partie des approches de classification des séries temporelles basées sur les **distances** (*distance-based approaches*).[[3]](#endnote-3)

La DTW permet de mesurer la similarité entre deux séries temporelles pouvant évoluer à des vitesses différentes.[[4]](#endnote-4) Elle calcule la correspondance optimale entre les séries en les déformant de manière non linéaire dans le temps (Figure 1).

Une image contenant ligne, Tracé, diagramme, capture d’écran

Description générée automatiquement

Figure 1. Comparaison entre la distance euclidienne, utilisée dans l’implémentation classique de la KNN, et la DTW, utilisée dans l’adaptation de la KNN aux séries temporelles.[[5]](#endnote-5)

La déclinaison 1NN de la méthode KNN-DTW est notamment utilisée comme *benchmark* grâce à sa simplicité et sa robustesse : « *DTW in conjunction with 1-NN has been the gold standard for time series classification for the past decade and is almost always used as a comparative algorithm in benchmarking studies* ».3

Toutefois, la méthode peut être gourmande en calcul et fournit peu d'informations sur les critères d'affectation de classe.

Elle est implémentée dans sktime par le classificateur KNeighborsTimeSeriesClassifier du module classification.distance\_based, configuré avec les paramètres suivants :

* n\_neighbors = 1
* distance = "dtw"

### Time Series Forest (TSF)

La méthode *time series forest* (TSF) appartient à la famille des approches de classification des séries temporelles basées sur les **intervalles** (*interval-based approaches*).3

Il s'agit d'une adaptation aux séries temporelles de l'algorithme *random forest classifier* (RCF).[[6]](#endnote-6)

La TSF utilise un arbre de décision pour chaque intervalle, les arbres de décision agrégés constituant la forêt. Chaque arbre de décision est un modèle d'apprentissage automatique qui attribue ensuite une classe à son intervalle de données. Étant donné que les arbres de décision s'entraînent sur un intervalle différent de la série temporelle globale, ils peuvent ne pas produire la même classification, d'où la nécessité du vote d'ensemble à la fin du processus.3

Des études expérimentales ont montré que la TSF est capable de produire de meilleurs résultats que la KNN-DTW.[[7]](#endnote-7)

La méthode est implémentée dans sktime par le classificateur TimeSeriesForestClassifier du module classification.interval\_based.

### RandOm Convolutional KErnel Transform (ROCKET)

La méthode « *RandOm Convolutional KErnel Transform* » (ROCKET) s'inspire des approches de classification des séries temporelles basées sur les **dictionnaires** (*dictionary-based approaches*)3, ainsi que des **réseaux de neurones convolutifs** (CNN).[[8]](#endnote-8)

La méthode consiste en deux étapes principales8 :

1. **Transformation** des séries temporelles à l'aide de noyaux de convolution aléatoires (*random convolutional kernels*)
2. **Entraînement** d’un classificateur linéaire sur les séries temporelles transformées

Contrairement aux noyaux des CNN, ceux de ROCKET sont extrêmement variés car leurs paramètres (p. ex., longueur, dilatation, *padding*) font l’objet d’un échantillonnage aléatoire (qui reste toutefois contrôlé).7 Cette diversité, couplée au très grand nombre de noyaux (10 000 par défaut), permet à la méthode de capturer des motifs à des fréquences et des échelles contrastées.8

La transformation des séries temporelles se fait par leur convolution avec les noyaux, générant une *feature map* qui est ensuite agrégée en 2 *features* par noyau (soit 20 000 *features* au total)8 :

* la **valeur maximale** (*maximum value* ; similaire au *global max pooling* des CNN), et
* la **proportion de valeurs positives** (*ppv*).

Une fois transformées, les séries temporelles sont passées à un classificateur linéaire dont le choix dépend de la taille du jeu d’entraînement7 :

* S’il y a moins de 20 000 lignes, il est conseillé d’utiliser un classificateur de régression *ridge*. En effet, plus l’écart entre le nombre de *features* (ici, 20 000) et la taille du jeu d’entraînement est grand, plus la régularisation est importante, et la capacité d’un classificateur de régression *ridge* à exploiter la validation croisée généralisée permet de trouver un paramètre de régularisation adapté assez rapidement.
* S’il y a plus de 20 000 lignes, il est conseillé de se tourner plutôt vers un classificateur de régression logistique entraîné à l'aide de la descente de gradient stochastique (*stochastic gradient descent* / SGD). En effet, l’entraînement de ce classificateur est relativement rapide pour un coût de mémoire fixe.

Des études ayant comparé ROCKET aux algorithmes concurrents, notamment les CNN, ont démontré que ROCKET peut atteindre le même niveau de *accuracy* en une fraction du temps.7

La méthode est implémentée dans sktime par le transformeur Rocket du module transformations.panel.rocket.

Pour les classificateurs linéaires, nous avons utilisé les suivants, issus du module linear\_model de sklearn :

* Jeux d’entraînement avec moins de 20 000 lignes :   
  RidgeClassifierCV(alphas = np.logspace(-3, 3, 10))
* Jeux d’entraînement avec plus de 20 000 lignes :   
  SGDClassifier(loss = "log\_loss")

### Prétraitement complémentaire des données

#### Tri

Pour le bon fonctionnement de sktime, et comme dans toute analyse de séries temporelles, il est impératif que les données soient classées par ordre chronologique.

Or, lors de leur chargement, les données sont d’abord classées selon la variable Location avant de l’être selon la variable Date. Cela n’a aucune incidence sur les jeux qui ne sont composés que d’une seule station météorologique (p. ex., les notebooks 1.X.X), mais pour les autres, cela pourrait provoquer des dysfonctionnements.

Afin d’éliminer ce risque, à chaque chargement des données dans un DataFrame pandas, nous les avons indexés par la variable Date (index\_col = 1) et avons ajouté un tri selon cet index (.sort\_index()).

Cette opération est systématiquement appliquée à partir des notebooks 2.X.X.

#### Découpage

Afin de préserver l’ordre chronologique des données lors de leur découpage en jeux d'entraînement et de test, nous avons utilisé le validateur croisé TimeSeriesSplit du module model\_selection de sklearn.

TimeSeriesSplit est une variation adaptée aux séries temporelles de l'itérateur de validation croisée KFold de la même bibliothèque.

Bien que ces objets soient principalement destinés au découpage des données lors d'un processus de validation croisée, ils peuvent également être appliqués au jeu d'origine pour créer les jeux d'entraînement et de test. D'ailleurs, l'itérateur train\_test\_split, qui est habituellement utilisé à cette étape, n'est autre qu'un *wrapper* autour de l'itérateur ShuffleSplit.

Contrairement aux autres itérateurs, TimeSeriesSplit génère des jeux d'entraînement successifs qui sont des surensembles (*supersets*) de ceux qui les précèdent. Par ailleurs, les jeux d'entraînement sont systématiquement plus anciens que les jeux de test. Ce comportement permet de prendre en compte la particularité structurelle des séries temporelles (Figure 2).

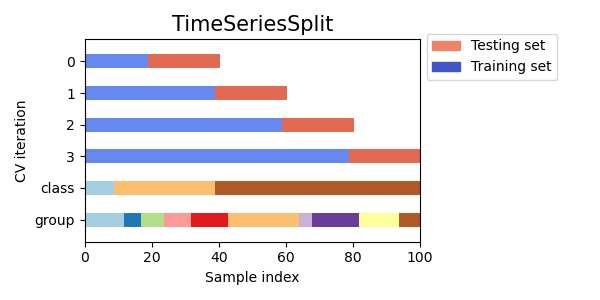


Figure 2. Découpage d’un jeu de données en jeux d'entraînement et de test par TimeSeriesSplit sur plusieurs itérations de validation croisée.[[9]](#endnote-9)

#### Conversion

Bien qu’il soit globalement facile à utiliser, sktime comporte une contrainte importante : les données doivent d’abord être converties en un format compatible.[[10]](#endnote-10)

Pour KNN-DTW et TSF, nous avons effectué cette conversion avec la fonction artisanale numpyfy. Elle prend en argument les jeux d’entraînement et de test (X\_train, X\_test, y\_train, y\_test) et les transforme en arrays NumPy de 3 dimensions, définies comme suit :

* Dimension 1 : nombre d'**instances** de séries temporelles. Pour cette étude, dont l'unité temporelle est le jour, nous considérons 1 jour comme 1 instance.
* Dimension 2 : nombre de **variables explicatives** par instance de série temporelle.
* Dimension 3 : nombre de **points temporels** observés par instance de série temporelle. Pour cette étude, comme l'unité temporelle est le jour, chaque instance de série temporelle correspond à 1 seul point temporel.

Pour TSF, il est nécessaire en plus de transformer les jeux de données multivariées en univariées. Afin de réaliser cette opération, nous avons utilisé le transformateur ColumnConcatenator du module transformations.panel.compose de sktime.

Enfin, pour ROCKET, nous ne sommes pas parvenus à faire fonctionner la méthode avec numpyfy. Nous avons donc repensé la structure des données afin de les convertir en format "Panel".15 Cela implique le changement de paradigme suivant :

* Chaque date (ligne) est toujours considérée comme une **instance** (un échantillon).
* En revanche, pour chaque instance, toutes les **variables explicatives** sont concaténées en une seule.
* Chaque concaténation est considérée comme une série de **points temporels**, disposés *horizontalement*.

Cette formulation n'est pas intuitive : en effet, il est plus naturel de définir une série temporelle *verticalement*, en suivant l'index chronologique.

Elle est néanmoins utile car elle permet d'adapter le jeu de données à la méthode ROCKET.

Elle s'inspire de l'analyse des images en tant que séries temporelles par le déroulement de leur contour sur un hypothétique axe temporel, permettant par la suite d'y appliquer des méthodes réservées aux séries temporelles (Figure 3).

Pour implémenter cette transformation, nous avons utilisé tslearn, une autre bibliothèque Python qui fournit des outils d'apprentissage automatique pour l'analyse de séries temporelles.[[11]](#endnote-11)

Il propose, dans son module utils, la fonction to\_sktime\_dataset, qui permet de convertir un jeu de données vers un format de scitype "Panel" et de mtype "nested\_univ" tels que définis par sktime.10 Ce format correspond également à celui du jeu de données dans la démonstration de ROCKET par sktime.[[12]](#endnote-12)

Une image contenant diagramme

Description générée automatiquement

Figure 3. Conversion du contour d’une image en une série temporelle pour analyse.[[13]](#endnote-13)

1. <https://www.sktime.net/en/stable/> [↑](#endnote-ref-1)
2. <https://towardsdatascience.com/sktime-a-unified-python-library-for-time-series-machine-learning-3c103c139a55> [↑](#endnote-ref-2)
3. <https://developer.ibm.com/learningpaths/get-started-time-series-classification-api/what-is-time-series-classification> [↑](#endnote-ref-3)
4. <https://en.wikipedia.org/wiki/Dynamic_time_warping> [↑](#endnote-ref-4)
5. <https://rtavenar.github.io/blog/dtw.html> [↑](#endnote-ref-5)
6. <https://towardsdatascience.com/a-brief-introduction-to-time-series-classification-algorithms-7b4284d31b97> [↑](#endnote-ref-6)
7. <https://link.springer.com/article/10.1007/s10618-020-00701-z> [↑](#endnote-ref-7)
8. <https://pub.towardsai.net/rocket-fast-and-accurate-time-series-classification-f54923ad0ac9> [↑](#endnote-ref-8)
9. <https://scikit-learn.org/stable/modules/cross_validation.html#time-series-split> [↑](#endnote-ref-9)
10. <https://www.sktime.net/en/stable/examples/AA_datatypes_and_datasets.html> [↑](#endnote-ref-10)
11. <https://tslearn.readthedocs.io/en/stable/index.html> [↑](#endnote-ref-11)
12. <https://www.sktime.net/en/latest/examples/transformation/rocket.html> [↑](#endnote-ref-12)
13. <https://izbicki.me/blog/converting-images-into-time-series-for-data-mining.html> [↑](#endnote-ref-13)