Rapport de Stage
Classification de séries temporelles avec des réseaux de neurones à convolution

REGUIG Ghiles
Tuteur : GALLINARI Patrick

October 30, 2017

Contents

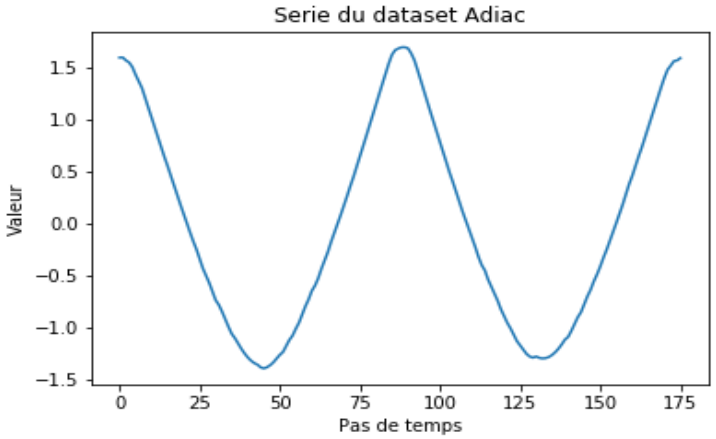

Figure 1: Série temporelle extraite du dataset *Adiac* (*UCR*)

Abstract

Dans le cadre de ce stage de recherche, on s'intéresse à la problèmatique de classification de série temporelle. Plus précisèment, on désire mesurer expérimentalement l'effet de l'utilisation de réseaux de neurones à convolution sur les performances en classification en les comparant à d'autres modèles, notamment les modèles se basant sur l'utilisation de shapelets.

1 Introduction

En raison de leur application dans de nombreux domaines tels que la médecine, l'astronomie, la météorologie ou encore l'économie, les séries temporelles représentent un domaine d'étude à part entière.
Durant ce stage, nous nous sommes intéressés à la classification de telles données en ayant recours à des réseaux de neurones à convolution. Nous commencerons, dans un premier temps, par introduire un ensemble de définitions relatives aux séries temporelles ainsi qu'aux réseaux de neurones à convolution, ensuite, dans un second temps, nous nous intéresserons à divers types représentation de série utilisés en classification. Enfin, nous exposerons et analyserons les résultats expérimentaux obtenus avec les modèles à convolution.

2 Définitions

2.1 Séries temporelles

On appelle série temporelle, ou encore série chronologique, une suite finie de valeurs représentant une évolution d'une ou plusieurs quantités en fonction du temps.
Soit x_t la valeur de la variable x au temps t. On appelle série temporelle de longueur n une suite de valeurs telle que : $(x_1, x_2, \ldots, x_n)$.
L'intervalle de discrétisation (différence temporelle entre le temps $n-1$ et n) peut être, selon les cas, la seconde, la minute, l'heure, le jour, ...
Un exemple de visualisation d'une série est représenté au sein de la figure 1.

2.2 Classification

Le problème de classification consiste à trouver une fonction f_W, paramétrée par un vecteur de poids W, prenant en entrée un exemple x (dans notre cas, il s'agit d'une série temporelle) et qui retourne la classe y^{pred} prédite par notre fonction de classification.
Le but d'une fonction de classification, que l'on appelle classifieur, est de minimiser le nombre d'erreurs. C'est-à-dire, faire en sorte que la classe prédite y_i^{pred} pour un exemple x_i soit la même que celle attendue y_i.
L'erreur est mesurée par une fonction dite de coût que l'on note C. Formellement, le problème de classification consiste donc à trouver les paramètres optimaux W^* tels que

$$W^* = \arg\min_W \sum_i C(f_W(x_i), y_i) + k\Omega(f_W)$$

avec

- C la fonction de coût permettant de mesurer l'erreur entre une prédiction et la véritable classe.
- f_w la fonction de prédiction du classifieur.
- x_i l'exemple d'indice i de la base d'apprentissage.
- y_i la classe de l'exemple d'indice i.
- k le coefficient de régularisation.
- Ω la fonction de régularisation permettant de contraindre la complexité de la fonction de prédiction.

2.3 Réseau de neurones à convolution

Un réseau de neurones à convolution est un réseau de neurones utilisant une ou plusieurs fonction(s) de convolution. Chaque couche utilise un ou plusieurs filtre(s) de convolution sur les données d'entrée afin de mettre en valeur certaines caractéristiques. On obtient en sortie autant de séries que la couche posséde de filtres.
Les différents hyper-paramétres à fixer dans un réseau à convolution sont :

- *la taille du noyau de convolution (kernel size)* : ce dernier permet de réguler le nombre de valeurs que prend en compte le filtre en même temps. Ainsi, plus la valeur est grande, plus la plage temporelle prise en compte est large. Les caractéristiques extraites seront plus globales/locales selon la taille du noyau.
- *le nombre de filtres de convolution* : ce dernier permet de définir le nombre de filtres différents à appliquer à l'entrée, chacun correspondant à une *feature*. Plus il y en a, plus il y a de *features* extraites, ce qui permet au classifieur une plus grande expressivité.
- *le pas du filtre (stride)* : ce dernier permet de définir le glissement de la fenêtre de convolution au sein de la série. Selon sa valeur, on pourra ignorer certains pas de temps ou les prendre en compte dans plusieurs fenêtres de convolution.

2.4 Shapelet

Une shapelet [Ye and Keogh, 2009] est une sous-séquence d'une série temporelle. Elle permet notamment de capturer des caractéristiques locales qui ne sont pas forcément prises en compte si l'on considére la série dans son entiéreté.

3 Classification de séries temporelles

Dans la littérature, il existe nombre de classifieurs de séries temporelles, ayant chacun ses spécificités. Dans [Bagnall et al., 2017], des benchmarks ont été établis pour 16 classifieurs sur chaque dataset UCR, tout en respectant un protocole expérimental non biaisé par les performances du modèle sur les données de test. En d'autres termes, l'apprentissage ainsi que la sélection de modèle se fait exclusivement à partir des données d'apprentissage fournies.
Dans ce rapport, nous nous intéressons à 3 types de classifications : celle qui se base sur la notion de distance entre les séries (Nearest Neighbor), celle qui se base sur des shapelets et enfin, celle qui se base sur des filtres de convolution.

3.1 Nearest Neighbor

Description du modèle

Un classifieur de type *nearest neighbor* (ou plus proche voisin), est un classifieur se basant sur une mesure de distance (ou de similarité) entre les exemples. La classification est faite par rapport aux exemples de la base d'apprentissage qui sont les plus proches, en terme de distance, de l'exemple à classer.

Spécificités

La plus grande spécificité de cet algorithme est qu'il est déterministe, c'est-à-dire qu'il ne dépend d'aucun facteur aléatoire. Ainsi, quelle que soit la machine qui exécute une telle classification, le résultat sera toujours le même, tant que la mesure de distance est calculée de manière similaire. Dans le cas des séries temporelles, ce classifieur s'avére particulièrement efficace, c'est pour cela qu'il fait office de référence notamment sur les données *UCR*.

Etant donné le caractére déterministe de cet algorithme, il n'a pas été jugé utile de le réimplémenter. Les résultats présentés ont été tirés de [Bagnall et al., 2017].

3.2 Shapelets

Description du modèle

L'extraction des shapelets les plus discriminantes est une tâche gourmande en temps de calcul, c'est pour cela qu'un modèle permettant de les apprendre a été proposé par [Grabocka et al., 2014]. Le modèle implémenté est inspiré du papier précédemment cité.
En premier lieu, les shapelets sont initialisées avec les valeurs des centroïdes d'un *k-means* effectué sur les données d'apprentissage (k représentant le nombre de shapelets).
Dans un second temps, on calcule la distance minimale approchée grâce à une fonction *softmin* entre notre dictionnaire de shapelets et la série à classifier. On fait le choix d'utiliser une fonction *softmin* au lieu du véritable minimum car la fonction *softmin* est différentiable. De ce fait, il est donc possible d'effectuer une backpropagation de l'erreur jusqu'aux *shapelets* et d'optimiser leurs valeurs.
Enfin, dans un troisième et dernier temps, la nouvelle représentation de la série est donnée en entrée à un classifieur correspondant à une couche linéaire suivie d'une fonction d'activation (*sigmoïde* si la classification est binaire, *softmax* sinon). On obtient alors une distribution de probabilité sur les différentes classes. Le classifieur prédit la classe avec la plus grande probabilité.

Spécificités

L'utilisation de *shapelets* présente l'avantage de prendre en compte des caractéristiques extrêmement localisées (comme montré par [Ye and Keogh, 2009]) ainsi que de pouvoir les visualiser.
Cependant, l'apprentissage des *shapelets* n'assure pas d'atteindre une valeur optimale car la fonction à optimiser n'est pas convexe et dépend fortement de l'initialisation faite via un *k-means*, qui est lui-même tributaire de sa propre initialisation. De plus, L'apprentissage de ce modèle est plutôt lent, en particulier si l'on prend un grand nombre de shapelets.

3.3 Réseau de neurones à convolution

Description du modèle

On propose un modèle se basant sur une couche de convolution suivie par un *k-max*.
En premier lieu, les séries passent par une couche de convolution à un ou plusieurs filtres. On obtient ainsi en sortie de cette couche autant de séries que de filtres.
Puis en second lieu, on effectue un *k-max* sur les valeurs en sortie, afin de synthétiser au mieux les informations extraites lors de la première couche. Selon la valeur de k, qui est un hyper-paramètre à fixer, on retiendra plus ou moins d'informations.
Enfin, dans un troisième et dernier temps, on effectue la classification de manière classique, avec une couche linéaire suivie d'une activation (*sigmoïde* en cas de classification binaire, *softmax* sinon). Encore une fois la prédiction est faite selon la probabilité maximale en sortie.

Spécificités

Ce modèle vise à se rapprocher de la classification par dictionnaire de shapelets. En effet, les valeurs maximales obtenues après application du filtre de convolution correspondent à celles qui sont les plus représentatives de ce filtre. De même que pour les shapelets, ces derniers sont également visualisables, ce qui permet d'avoir une interprétabilité des résultats.

4 Expérimentations

Les expérimentations ont été effectuées sur les données UCR [Chen et al., 2015]. Le seul traitement effectué a été de reformater la numérotation des classes afin qu'elles aillent de 0 au nombre de classes. Dans ce rapport, on choisit 4 datasets parmi ceux proposés : *Adiac, CricketX, ECG200 et SwedishLeaf*. Leurs caractéristiques sont consignées dans la table 1.
Les deux modèles ont été implémentés en langage *Python* à l'aide de la librairie *Pytorch*. Chaque mention de régularisation fait référence à une régularisation *L2*.

Dataset	Train/Test	Longueur de la série	Nombre de classes
Adiac	390/391	176	37
CricketX	390/390	300	12
ECG200	100/100	96	2
SwedishLeaf	500/625	128	15

Table 1: Caractéristiques de certains datasets de la base *UCR*

4.1 Influence des hyper-paramétres du modèle à convolution

4.1.1 Taille du filtre

On s'intéresse en premier lieu à l'importance de la taille du filtre sur les performances du classifieur. Afin de pouvoir visualiser cela, on commence par tracer les filtres obtenus en faisant varier leur taille.
On utilise un modèle à convolution avec un unique filtre, un *k-max* de 1, un stride de 1. La taille du filtre varie entre 3 valeurs : 10, 20 et 30.
Pour faciliter la lecture graphique, un dataset avec une classification binaire est pris (*ECG200*). Les résultats sont ceux de la figure 2.
Sur le côté gauche de la figure, on a tracé les filtres extraits selon les différentes tailles. On constate qu'ils différent fortement d'une taille à l'autre, ce qui semble tout-à-fait logiqe étant donné que l'information prise en compte tend à être plus globale à mesure que la taille du filtre augmente.
Sur le côté droit, on représente 20 séries du dataset *ECG200* selon leur classe, en fonction de la valeur du 1-max. Cette représentation vise à montrer à quel point il sera facile ou non de classer ces données par le classifieur. Des trois graphiques, le plus simple à classifier est le second (avec une taille de filtre de 20) car les séries représentées sont parfaitement linéairement séparables, ce qui n'est pas le cas des deux autres.
En comparant les 3 graphiques, on constate également que, si l'on gagne à passer d'un filtre de taille 10 à un filtre de taille 20, ce n'est pas le cas lorsque l'on passe à un filtre de taille 30, où une série de la classe 0 vient se juxtaposer à un groupe de séries de classe 1.
On ne peut certes pas généraliser à partir de cet exemple, car la représentation ne concerne qu'une vingtaine de séries.

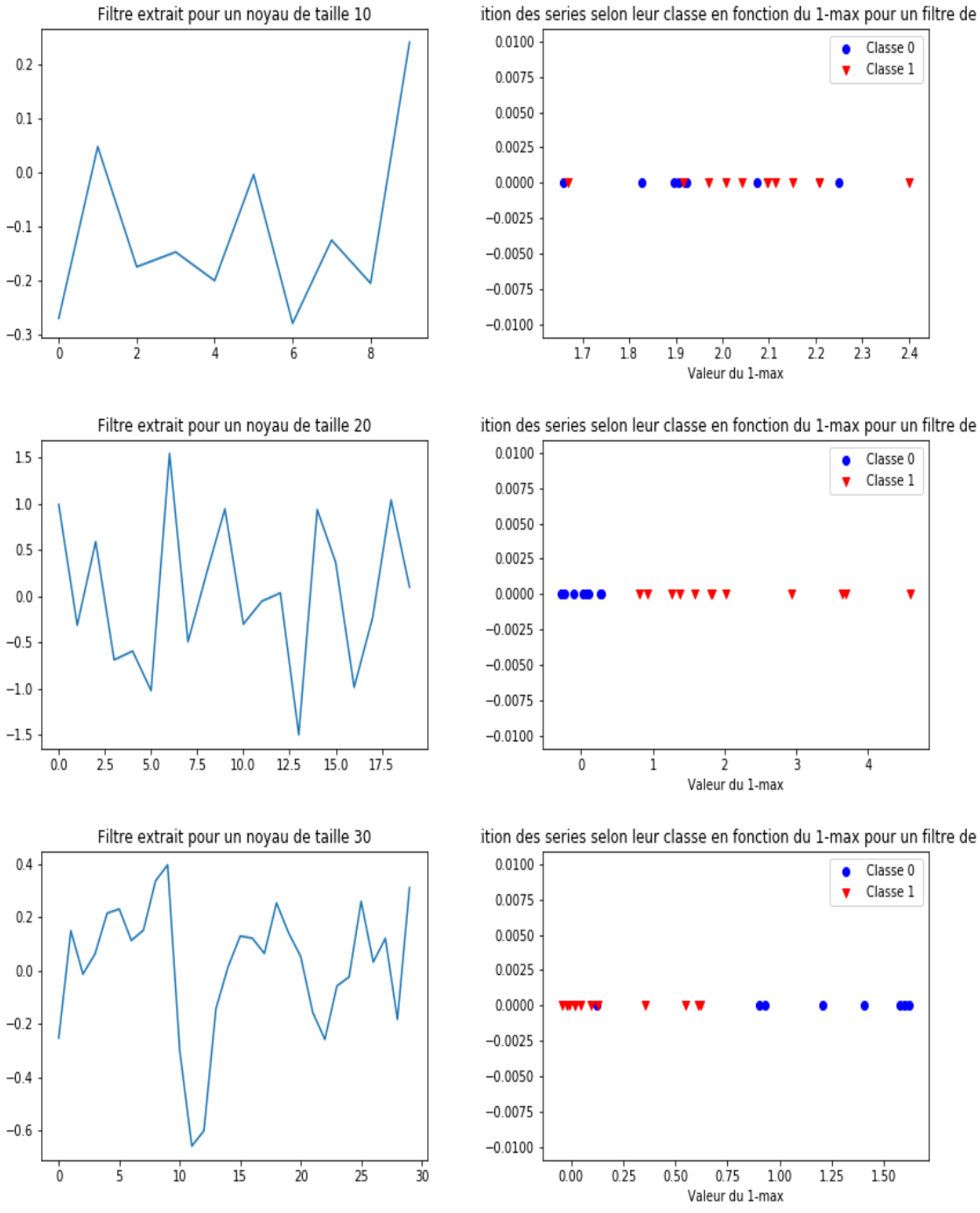

Figure 2: Influence de la taille du filtre sur des séries extraites d'*ECG200*

4.1.2 Le nombre de filtres

Un autre hyper-paramétre intéressant à étudier est le nombre de filtres de la couche de convolution. Celui-ci détermine le nombre de features extraites des séries temporelles. Comme elles sont ensuite passées en entrée au classifieur, cela permet notamment d'augmenter l'information utilisée pour la classification. Chaque nouveau filtre augmente le nombre de dimensions du classifieur de k, étant donné que l'on retient les k plus grandes valeurs.
En reprenant le dataset *ECG200* et les 3 modèles précédents, un second filtre est ajouté. De la même façon que pour l'expérience de la taille du filtre, on visualise les 2 filtres extraits et l'on représente une vingtaine de séries par classe en fonction des *1-max* des deux filtres.
Une première observation que l'on peut faire concerne les filtres. En effet, en en considérant 2 pour chaque taille, on obtient 2 filtres complétement différents de ceux obtenus avec des modèles à 1 filtre. L'ajout de cette seconde dimension change également la représentation graphique des séries: avec 1 filtre en plus, les données deviennent linéairement séparables avec un noyau de taille 10 et 30, ce qui n'était pas le cas précédemment.
Encore une fois, cela n'est pas représentatif de la performance de la classification du modèle sur ce dataset car seules 20 séries sont prises en compte

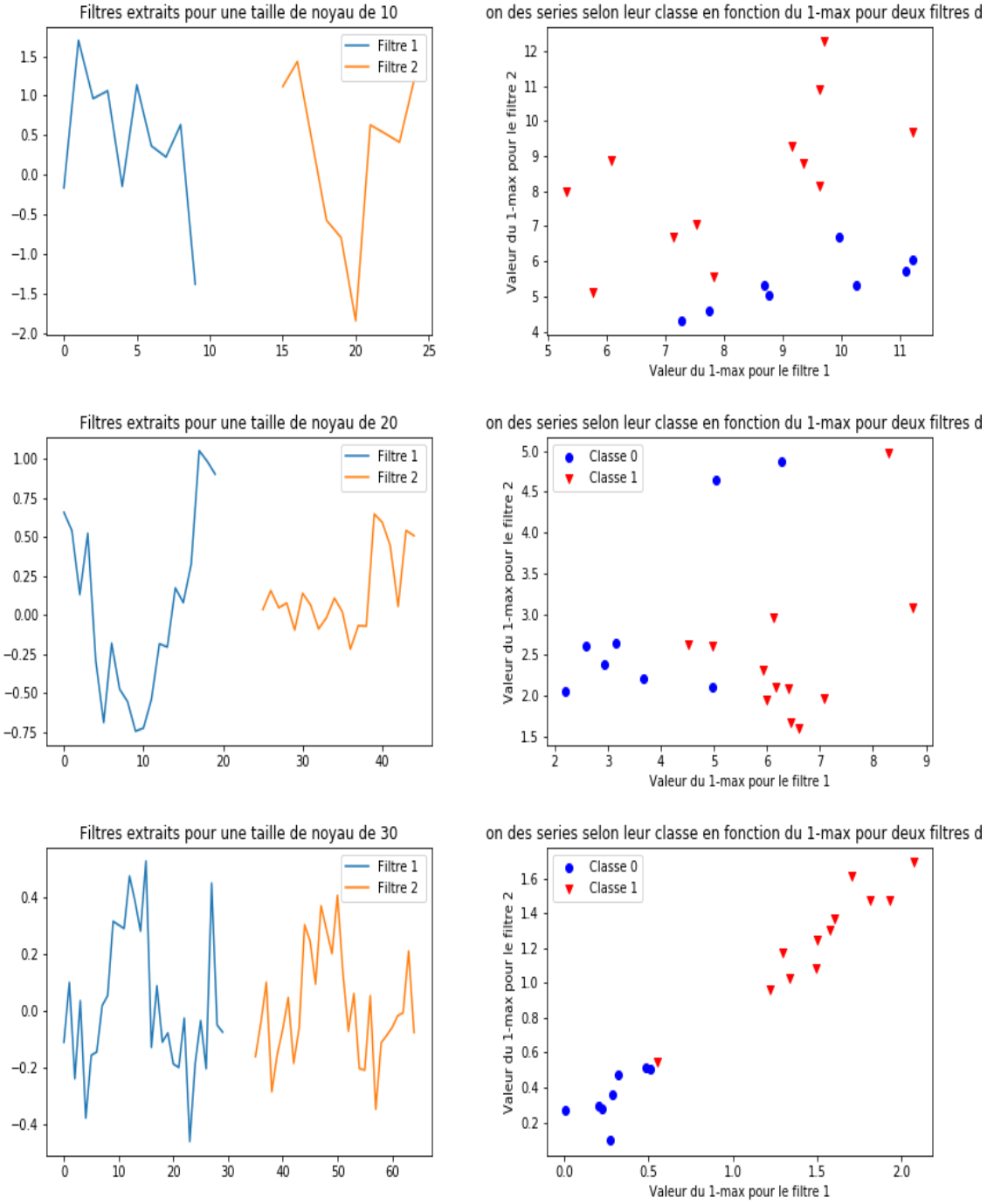

Figure 3: Influence du nombre de filtres pour des séries d'*ECG*

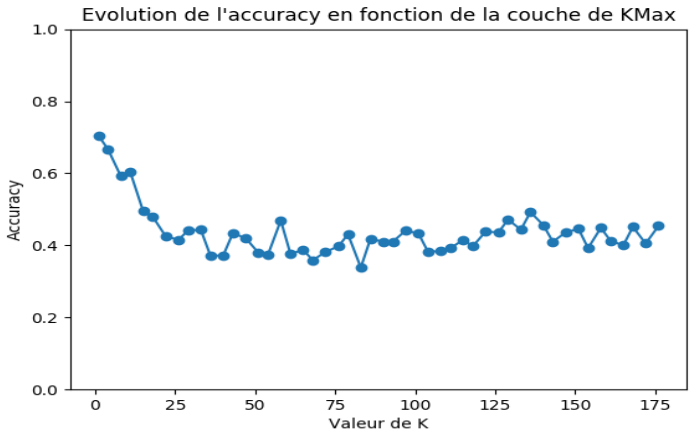

Figure 4: Influence du KMax sur la classification du dataset *Adiac*

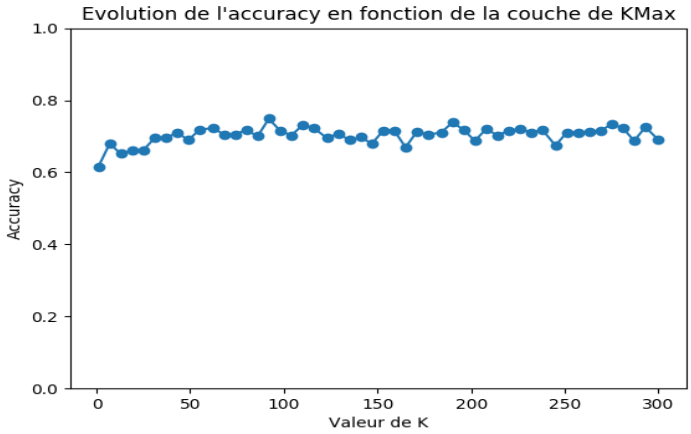

Figure 5: Influence du KMax sur la classification du dataset *Cricket X*

4.1.3 Le *k-max*

On désire ensuite vérifier l'impact de l'utilisation du *k-max* sur les performances de notre classifieur. Pour ce faire, on prend 50 valeurs différentes de K entre 1 et la taille de la série. On choisit des hyper-paramètres différents pour chaque dataset (table 2). Après apprentissage des modèles, ils sont évalués par crossvalidation, ce qui nous donne les figures 4 pour *Adiac*, 5 pour CricketX, 6 pour *ECG200* et 7 pour *SwedishLeaf*.

Dataset	Nombre de filtres	Taille du noyau	Stride	Régularisation
Adiac	20	30	1	0.0
CricketX	20	30	1	0.001
ECG200	20	30	1	0.0001
SwedishLeaf	20	10	1	0.0

Table 2: Hyper-paramétres utilisés pour les différents datasets

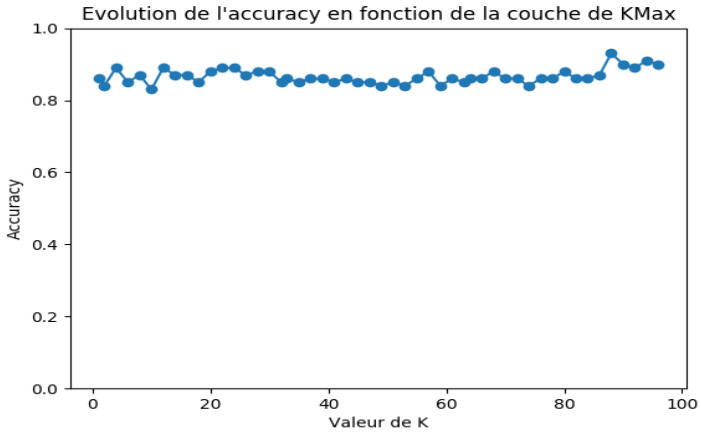

Figure 6: Influence du KMax sur la classification du dataset *ECG200*

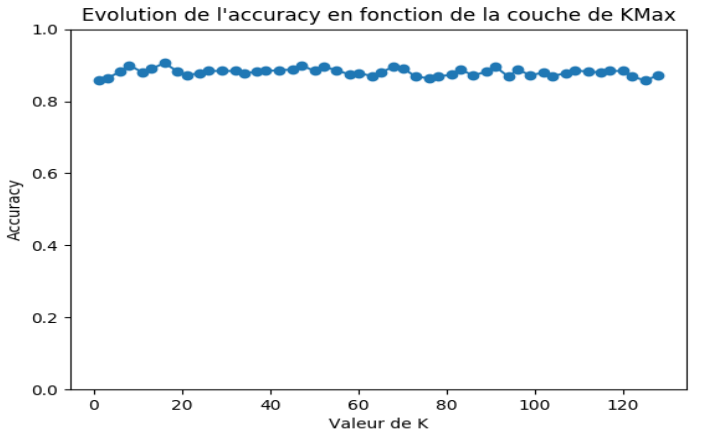

Figure 7: Influence du KMax sur la classification du dataset *SweidshLeaf*

4.2 Comparaison des deux modèles

4.2.1 Visualisation des features

On s'intéresse en premier lieu à l'interprétation des résultats de nos modèles et à leur comparaison. On commence alors par apprendre nos deux modèles selon les mêmes conditions : on les contraint à l'utilisation d'un unique filtre/shapelet, de taille fixée à 5. En ce qui concerne le modèle à convolutions, on fixe le *k-max* à 1, afin d'obtenir le maximum. On extrait ensuite le filtre/shapelet que l'on applique sur une série (*Softmin/Convolution*). On obtient les graphiques de la figure 8.

4.2.2 Performances sur les datasets

En prenant comme données les 4 datasets précédemment vus (cf table 1), on désire mesurer les performances de nos modèles.
Le protocole expérimental est le suivant : un *GridSearch* est effectué sur les données d'apprentissage fournies par la base *UCR* afin de déterminer les hyper-paramétres les plus efficaces. Pour évaluer les classifieurs, on a recours à une *crossvalidation* sur les données d'apprentissage : ces dernières sont divisées en 3 sous-groupes, puis pour chaque configuration possible, le modèle apprend sur 2 sous-groupes et est évalué sur le dernier. Les performances sont ensuite moyennées.
Comme il est possible d'avoir plusieurs combinaisons optimales, ces dernières sont ré-évaluées 10 fois de suite en prenant la totalité du set *train* pour l'apprentissage et en testant sur les données de *test*. Encore une fois, les résultats sont moyennés sur les 10 expérimentations.
En ce qui concerne l'apprentissage, on utilise un coût des moindres carrés, l'optimiseur *Adam* avec un learning rate de 0.001 pour le modèle à convolution et de 0.1 pour le modèle à *shapelets*. On fixe un nombre d'epochs à 10000 en sachant qu'on implémente également un *EarlyStopping* sur la valeur du coût. Si cette dernière ne diminue pas d'au moins 0.0001 pendant 1000 epochs, l'apprentissage est arrêté et le modèle reprend les meilleurs paramétres.
La totalité des résultats sont disponibles à cette adresse : https://github.com/GReguig/StageShapelets/tree/master/ResultatsUCR.
Le répertoire contient 3 dossiers contenant les performances du modèle :

1. ResultatsHTML : ce dossier contient la totalité des données brutes du GridSearch effectué via la bibliothéque *sklearn* : score en train/test en moyenne et sur chaque split, temps d'apprentissage moyen par modèle,... pour chaque configuration possible.

2. ResultatsExcel : ce dossier contient des fichiers au format .xlsx synthétisant les données disponibles dans ResultatHTML. Ces derniers contiennent, pour chaque configuration, les performances moyenne en train/test ainsi que leur écart-type.

3. ResultatsExcelBestParams : ce dossier contient des fichiers identiques à ceux de ResultatsExcel concernant les meilleures configurations trouvées par le GridSearch après un moyennage sur une dizaine d'apprentissage; Les performances en test présentées dans ces fichiers sont les seules à concerner les données de test

Paramétres	Nombre de filtres	Taille du noyau	Stride	k-max	Régularisation
Valeurs	[5; 10; 15; 20]	[10; 20; 30]	[1; 15; 20]	[5; 7; 10]	$[0.0; 10^{-3}; 10^{-4}; 10^{-5}]$

Table 3: Hyper-paramétres utilisés en *GridSearch* pour le modèle à convolution

Paramétres	Nombre de shapelets	Taille des shapelets	Scale	Régularisation
Valeurs	$\log(totalNumSegments) \times (numClasses - 1)$	[0.1; 0.2]	[2; 3]	$[0.0; 10^{-1}; 10^{-2}]$

Table 4: Hyper-paramétres utilisés en *GridSearch* pour le modèle avec *shapelets*[1]

Dataset	Train	Test	Nbre Filtres	Taille Noyau	Stride	k-max	Régularisation
Adiac	0.867(±0.018)	0.689(±0.029)	20	30	1	5	0.0
ArrowHead[2]	1(±0)	0.720(±0.029)	10	20	15	5	0.0
Beef[2]	0.987(±0.016)	0.650(±0.021)	5	30	15	10	0.0001
Beetlefly	0.995(±0.015)	0.66(±0.109)	20	30	20	5	0.001
BirdChicken	1(±0.0)	0.72(±0.078)	5	30	15	10	0.0
Car	0.983(±0.007)	0.73(±0.028)	20	30	15	10	0.0001
CBF	1(±0.0)	0.945(±0.051)	5	30	1	5	0.001
Coffee[2]	1(±0.0)	0.957(±0.021)	5	10	1	7	10^{-5}
CricketX	0.930(±0.010)	0.637(±0.017)	20	30	1	7	0.001
ECG200[2]	1(±0)	0.829(±0.022)	20	20	15	5	0.001
FaceFour[2]	1(±0.0)	0.727(±0.060)	15	30	1	10	0.0001
MedicalImages[2]	0.975(±0.009)	0.681(±0.017)	20	20	1	7	0.0001
SwedishLeaf	0.991(±0.020)	0.916(±0.015)	20	10	1	10	0.0
SyntheticControl[2]	1(±0.0)	0.992(±0.005)	5	30	1	5	10^{-5}

Table 5: Performances du modèle à convolution

[1]GridSearch tiré de http://fs.ismll.de/publicspace/LearningShapelets/

[2]Datasets avec plusieurs configurations optimales trouvées. Seule celle avec la meilleure performance en test a été consignée

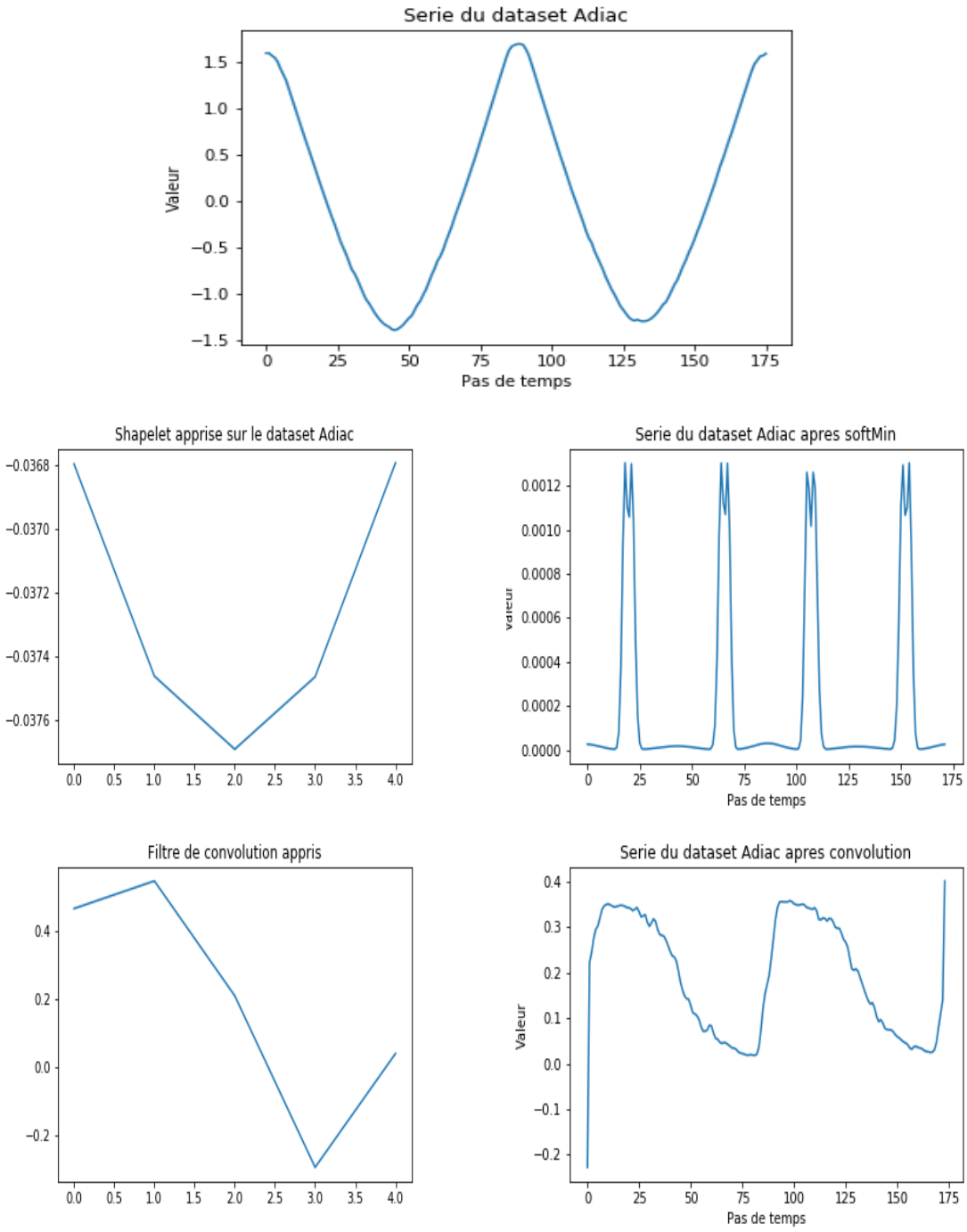

Figure 8: Résultats de l'application d'un softmin et d'une convolution sur une série du dataset *Adiac*

References

[Bagnall et al., 2017] Bagnall, A., Lines, J., Bostrom, A., Large, J., and Keogh, E. (2017). The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 31(3):606–660.

[Chen et al., 2015] Chen, Y., Keogh, E., Hu, B., Begum, N., Bagnall, A., Mueen, A., and Batista, G. (2015). The ucr time series classification archive. www.cs.ucr.edu/~eamonn/time_series_data/.

[Grabocka et al., 2014] Grabocka, J., Schilling, N., Wistuba, M., and Schmidt-Thieme, L. (2014). Learning time-series shapelets. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 392–401. ACM.

[Ye and Keogh, 2009] Ye, L. and Keogh, E. (2009). Time series shapelets: a new primitive for data mining. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 947–956. ACM.