École nationale supérieure de
l'électronique et de ses applications

Rapport de projet : Causal Feature Selection in large Multivariate Time Series (MTS) for Forecasting Problems

Élèves :
EL MAAZOUZ AMINE
Barrie Mathys
Tadjouri Imad-eddine

Enseignant :
Vassilis CHRISTOPHIDES
Etienne VAREILLE

16 avril 2024

Table des matières

1 Description générale du projet

1.1 Mise en contexte

Les séries temporelles multivariées (MTS) sont omniprésentes dans de nombreux domaines scientifiques et d'ingénierie, notamment la santé, la finance, la météorologie. Une MTS est composée de plusieurs variables dépendantes du temps qui peuvent dépendre non seulement de leurs valeurs passées, mais aussi d'autres variables.

Lorsqu'on tente de prévoir (prédire) les pas de temps futurs d'une MTS en fonction des observations passées, le nombre de covariables (c'est-à-dire de variables dépendantes du temps mesurées simultanément) peut être trop élevé par rapport au nombre de points de référence (pas de temps), entraînant une modélisation imprécise en raison, paradoxalement, d'un manque de données.

1.2 Objectifs du projets

Notre objectif est d'amener des éléments de réponse à 3 questions :

- Quelle est la meilleure combinaison d'algorithmes de sélection de variables et de prédiction ?
- Quelles sont les meilleures variables à sélectionner ? Comment se comportent-elles vis-à-vis du graphe causal ?
- Quel est l'influence des hyperparamètres sur les performances de nos algorithmes ?

Pour cela, nous allons nous appuyer sur des résultats expérimentaux sur différents jeux de données.

1.3 Plan du projet

Notre projet va donc se répartir en 2 parties :

- Une première partie focalisée sur l'étude des algorithmes de sélection de variables, leur compréhension, ainsi que leur implémentation. Nous aurons ensuite à analyser nos résultats expérimentaux, les expliquer puis les mettre en relation avec les algorithmes de prédiction
- Une seconde partie qui consistera à étudier les algorithmes de prédiction, mettre en évidence le besoin de la sélection de variable, évaluer les résultats de la première partie.

1.4 Pipeline utilisé

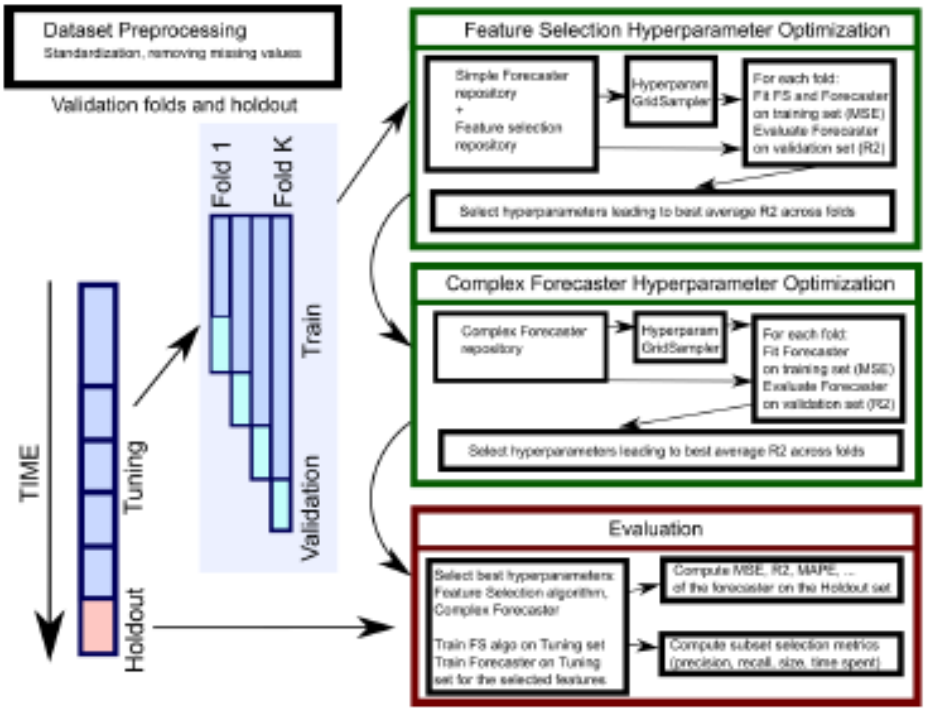


FIGURE 1 – Pipeline

3

2 Introduction des Datasets

Nous allons traiter deux jeux de données, VARsmall et 7ts2h.

2.1 Varsmall

Pour le premier jeux de données on a les caratéristiques suivantes :

- 10 caractéristiques dont 6 sont des caractéristiques pertinentes et le nombre de retard théorique est de 5
- Processus linéaire vectoriel autorégressif (VAR) généré synthétiquement
- 3500 horodatages

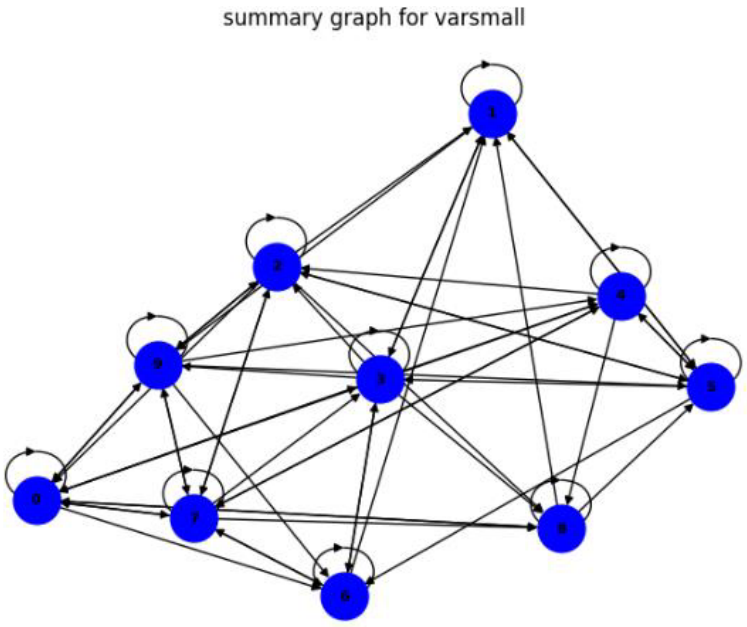


FIGURE 2 – Summary graph varsmall

2.2 7tns2H

Pour le deuxième jeux de données on a les caratéristiques suivantes :

- 7 caractéristiques dont 2 caractéristiques sont pertinentes
- le nombre de retard théorique est de 5
- Processus non linéaire généré synthétiquement

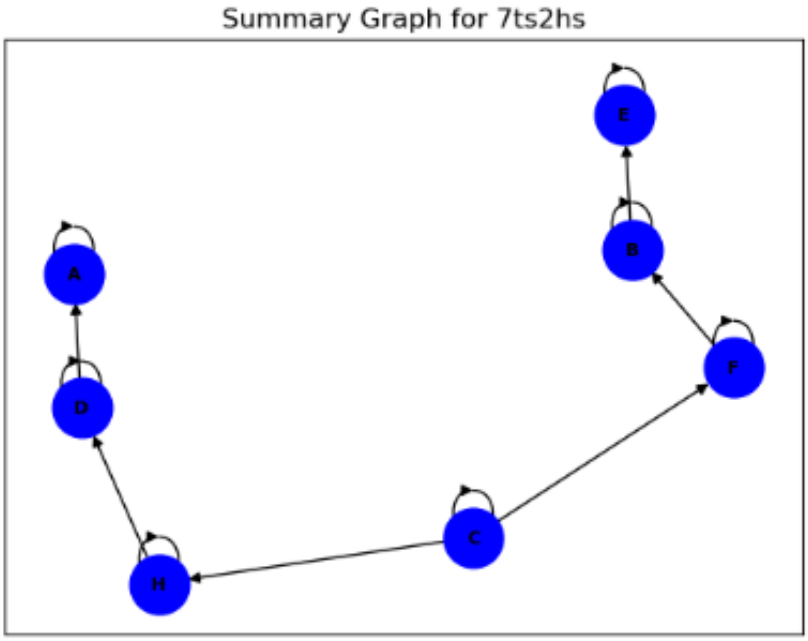


FIGURE 3 – summary graph 7ts2h

2.3 Vectorisation des données

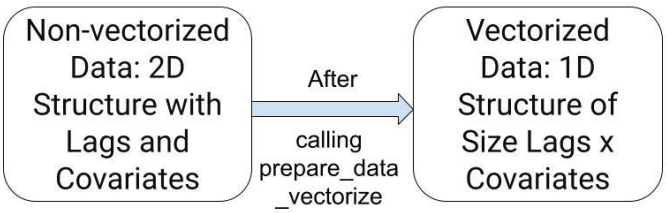


FIGURE 4 – Avant et après la vectorisation des données

Dans cette section, nous effectuons plusieurs étapes cruciales pour la préparation des données :

— Tout d'abord, nous extrayons la variable cible en excluant les premières lignes afin de garantir une représentation précise de nos données.

— Ensuite, à l'aide de la fonction 'prepare_data_vectorize' définie ci-dessous, nous créons des fenêtres glissantes sur les données brutes, facilitant ainsi une analyse temporelle approfondie.
— Enfin, nous transformons ces fenêtres en vecteurs unidimensionnels, prêts à être utilisés dans nos analyses ultérieures.

Prepare data vectorize ¶

```
import numpy as np
def prepare_data_vectorize(data, lags,target):
    y = data[target].iloc[lags:]
    indexes = y.index
    y = y.values
    window_X = [data.values[i:i+lags].reshape((-1,)) for i in range(len(data)-lags)]
    X = np.array(window_X)
    return X, y, indexes
```

FIGURE 5

Ce code définit une fonction Python appelée prepare_data_vectorize qui prépare les données pour une analyse ultérieure. La fonction prend trois arguments en entrée : data, lags, et target. Tout d'abord, elle extrait la variable cible du jeu de données en excluant les premières lignes. Ensuite, elle crée des fenêtres glissantes sur les données brutes, ce qui permet une analyse temporelle approfondie. Ces fenêtres sont ensuite transformées en vecteurs unidimensionnels pour faciliter leur utilisation dans les analyses ultérieures. Finalement, la fonction retourne les données préparées, comprenant les vecteurs caractéristiques, la variable cible et les index correspondants.

3 Etude des algorithmes de sélection de variables

3.1 Introduction

La première phase de notre projet se consacre à une analyse approfondie des algorithmes de sélection de variables appliqués aux séries temporelles multivariées (MTS). Cette étape fondamentale vise à établir une compréhension approfondie des méthodes de sélection de variables et à explorer spécifiquement deux classes d'algorithmes : RRelief et MRMR.

L'étude débutera par une présentation exhaustive des différentes classes d'algorithmes de sélection de variables. Cette phase introductive jettera les bases conceptuelles en mettant en lumière les diverses approches utilisées pour identifier les variables les plus pertinentes dans un contexte de séries temporelles multivariées.

Le premier volet de notre exploration se concentrera sur l'algorithme RRelief. Ce dernier, initialement développé pour la classification, a été adapté pour la sélection de variables dans le contexte des séries temporelles. Nous examinerons en détail son mécanisme, ses avantages et ses limitations, ainsi que sa pertinence dans la sélection des caractéristiques temporelles cruciales. Le deuxième algorithme mis en avant est MRMR, qui se distingue par sa capacité à établir un équilibre entre la réduction de la redondance entre variables et la maximisation de leur pertinence pour la prédiction. Nous explorerons son cadre conceptuel, son application spécifique aux séries temporelles multivariées, et son impact sur la qualité des modèles prédictifs.

Cette première partie constitue un socle essentiel pour la suite du projet, fournissant une compréhension approfondie des outils de sélection de variables spécifiquement adaptés aux défis posés par les MTS. En se concentrant sur Relief et MRMR, nous jetterons les bases nécessaires pour évaluer leur efficacité dans le contexte complexe des séries temporelles multivariées.

7

3.2 Notre premier algorithme de sélection de variable : MRMR

3.2.1 Introduction

Dans le cadre de notre projet, nous explorons plusieurs méthodes de Feature Selection pour optimiser les pertinences des variables à utiliser dans l'algorithme de Forecasting. L'une des méthodes qui a été choisie est l'algorithme MRMR (Maximum Relevance Minimum Redundancy), une technique de Feature Selection qui cherche à identifier un ensemble de variables qui maximise la pertinence avec la variable cible tout en minimisant la redondance entre les variables sélectionnées. Nous allons établir les principes fondamentaux de cet algorithme, ses avantages ainsi que les résultats obtenus vis à vis des dataset définis.

3.2.2 Explication de l'algorithme

Pour appliquer l'algorithme de sélection de features, on commence tout d'abord par préparer notre dataset de base. Il est nécessaire de séparer les différentes features, entre les possibles target, les features du ground truth et celles que l'on souhaite utiliser pour la prédiction. Il faut également penser à éventuellement transformer les données catégorielles en données numérique pour utiliser correctement l'algorithme de sélection de features. Ensuite, il est important de vectoriser nos données avec une sliding windows.

A partir d'ici, nous pouvons utiliser MRMR en tant qu'algorithme itératif qui va ajouter les features une à une dans une liste que l'on pourra afficher à la fin du processus comme suit :

- Pour chaque caractéristique on calcule la mesure de pertinence avec la variable cible, la mesure de la pertinence va évaluer à quel point chaque caractéristique est individuellement liée à la variable cible. Ces mesures de pertinences se font selon des métriques comme la mesure de corrélation, l'information mutuelle...
- On classe les caractéristiques par rapport à leurs mesures de pertinences en ordre décroissant, les caractéristiques les plus pertinentes par rapport à la variable cible sont en tête de liste.
- On créer deux listes, une qui stocke les caractéristiques sélectionnée (S) et une autre pour les caractéristiques restantes (R).
- On choisit maintenant les caractéristiques de R qui ont la mesure de pertinence maximale avec la variable cible la plus élevée, on les place donc dans S.
- Pour chaque caractéristique dans S, on calcule la mesure de redondance entre la caractéristique sélectionnée et celle déjà présente dans S. La mesure de la redondance évalue à quel point une caractéristique apporte une information similaire à celle déjà sélectionnée. On peut mesurer la redondance avec des métriques comme le coefficient de détermination (R2), l'information mutuelle ou la corrélation.
- On choisit les caractéristiques de R qui minimise la redondance avec les caractéristiques déjà sélectionnées dans S, on les ajoute à S et les retire de R.
- On répète le processus des deux dernières étapes jusqu'à avoir une taille de S désirée ou bien l'épuisement de R.

3.2.3 Hyperparamètres

MRMR possède divers hyperparamètres, les voici :

- K : Il s'agit d'un entier qui définit le nombre maximal de features à sélectionner, la taille de la sortie doit être au moins égale à K
- Relevance : Il s'agit d'une fonction qui calcule la pertinence (relevance) pour chaque feature.
- Redundancy : Sur le même principe, cette fonction calcule la redondance(redundancy) de chaque features en fonction d'une target à indiquer.
- Denominator : Cette fonction de synthèse est appliquée au dénominateur du score MRMR calculé lorsque l'on effectue le calcul de relevance puis redundancy. Il permet de normaliser ce score. Il est par défaut utilisé avec un numpy.mean mais il est également possible d'utiliser numpy.max ou numpy.median, ce paramètre possède peu d'influence

3.2.4 Implémentation

Pour implémenter l'algorithme MRMR, il fallait importer la fonction mrmr_regression depuis la bibliothèque mrmr installée sur l'IDE :

```
import numpy as np
from mrmr import mrmr_regression
import pandas as pd
from sklearn.feature_selection import RFE
from sklearn.svm import SVR
from sklearn.linear_model import LassoLars
from sklearn.feature_selection import SelectFromModel
import matplotlib.pyplot as plt
import random
import time

from statsmodels.tsa.vector_ar.var_model import VAR
```

FIGURE 6 – Importaton des packages

On définit ensuite la fonction qui va appliquer à proprement parler la sélection de features selon MRMR, on l'appelle apply_mrmr.

Application de MRMR

```
[ ]  def apply_mrmr(data, i,target,lags):
            X, y, _ = prepare_data_vectorize(data, lags,target)
            X_df = pd.DataFrame(X, columns=[f'lag_{i}' for i in range(1, X.shape[1] + 1)])
            num_features_to_select = config['num_features_to_select']

            selected_features_columns = mrmr_regression(X_df, y, i,denominator=np.mean)
            vector_mask = [name in selected_features_columns for name in X_df.columns]
            selected_features_names=vector_mask_to_columns(vector_mask,data)


            return selected_features_names
```

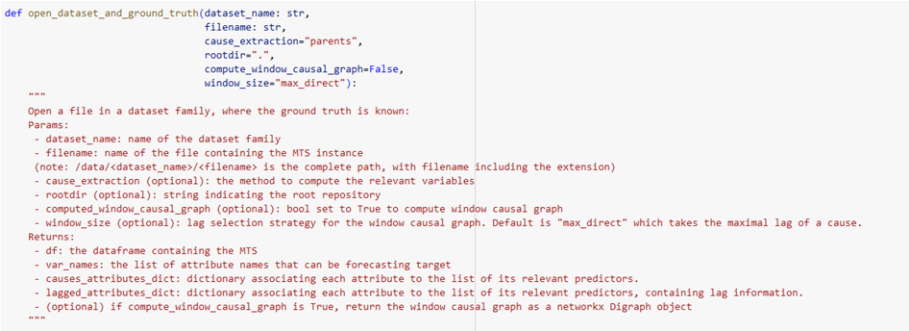
FIGURE 7

Cette fonction possède comme arguments les données que l'on utilise comme support, le nombre de caractéristiques que l'on souhaite sélectionner et la variable cible que l'on choisit parmi le jeu de données. La fonction ajuste le modèle aux données et vectorise les colonnes du jeu de données, sur ces colonnes on applique la fonction mrmr_regression qui sélectionne les variables selon la méthode définit précédemment. Finalement, les caractéristiques sont renvoyées selon le nom qu'elle possède dans le jeu de données.

3.2.5 Pipeline de l'expérimentation de feature selection

Nos expérimentations suivent un pipeline précis concernant les deux algorithmes de features sélection utilisés.

Il faut débuter par le chargement du dataset en utilisant la fonction open_data_set_and_ground_truth qui va nous retourner un dataframe contenant les données, les noms des variables, un dictionnaire des causes directes avec lags et sans lags ainsi que les targets possibles.



```
def open_dataset_and_ground_truth(dataset_name: str,
                                  filename: str,
                                  cause_extraction="parents",
                                  rootdir=".",
                                  compute_window_causal_graph=False,
                                  window_size="max_direct"):
    """
    Open a file in a dataset family, where the ground truth is known:
    Params:
     - dataset_name: name of the dataset family
     - filename: name of the file containing the MTS instance
     (note: /data/<dataset_name>/<filename> is the complete path, with filename including the extension)
     - cause_extraction (optional): the method to compute the relevant variables
     - rootdir (optional): string indicating the root repository
     - computed_window_causal_graph (optional): bool set to True to compute window causal graph
     - window_size (optional): lag selection strategy for the window causal graph. Default is "max_direct" which takes the maximal lag of a cause.
    Returns:
     - df: the dataframe containing the MTS
     - var_names: the list of attribute names that can be forecasting target
     - causes_attributes_dict: dictionary associating each attribute to the list of its relevant predictors.
     - lagged_attributes_dict: dictionary associating each attribute to the list of its relevant predictors, containing lag information.
     - (optional) if compute_window_causal_graph is True, return the window causal graph as a networkx Digraph object
    """
```

Nous préparons ensuite un dataframe qui va contenir nos différents résultats et informations nécessaires à ceux-ci comme les targets utilisées, les features sélectionnées, les métriques utilisées dans l'expérimentation (f1-score,precision,recall...), le nom du dataset et des filename ainsi que le temps d'éxecution.

```
def creation_data_frame(dataset_name,filename,df,lags=10,num_features_to_select=10):
    df, var_names, causes_attributes_dict, lagged_attributes_dict, ground_truth_graph = open_dataset_and_ground_truth(dataset_name, filename, rootdir="./", compute_window_causal_graph=True, window_size="max_direct")
    dict={}
    list_rows=[]
    List_oflags=[1,2,5,10]
    List_ofk=[10,20,30,50]
    total_columns=len(df.columns)
    for lags in List_oflags:
      for n in range(1,lags*total_columns+1,5):
        for target in var_names:
          for num_features_to_select in List_ofK:
            start_time=time.time()
            selected=apply_mrmr(df,n,target, lags,num_features_to_select)
            time_MRMR=time.time()
            ground_truth_causes = causes_attributes_dict[target]
            dict_metrics=compute_stats_selected(total_columns,selected,ground_truth_causes,None,selection_mode="variable")
            dict_metrics['target']=target
            dict_metrics['filename']=filename
            dict_metrics['dataset name']=dataset_name
            dict_metrics['selected features']= n
            dict_metrics['lags']=lags
            dict_metrics['selected_set']=selected
            dict_metrics['execution time']= time_MRMR-start_time
            list_rows.append(dict_metrics)
    list_rows.append(dict_metrics)
```

Il faut ensuite passer à l'expérimentation en elle-même, on utilise alors l'un des deux algorithmes, RRelief ou MRMR sur l'entièreté du dataset vectorisé, on calcule avec cela les métriques par rapport aux causes directes de la target. On privilégie l'utilisation des causes directes plutôt que la classification classique car nous sommes intéressés par les relations causales entre les features. On fixe également pour notre expérience un nombre de lags égal à 10.

```
def compute_stats_selected(totalcolumns, selected, causes, lagged_causes, selection_mode="variable"):
    """
    Compute metrics about the selected set versus the ground truth set.
    Params:
        totalcolumns: int, total number of columns in the dataset
        selected: list of str if selection_mode is "variable", the list of selected columns by the feature selection algorithm
                  list of (str, int) if selection mode is "variable, lag", the list of selected features by the FS algorithm
        causes: list of str, the list of ground truth relevant columns (without lag information)
        lagged_causes: list of str, the list of ground truth lagged causes. This argument is ignored if selection_mode is "variable".
        selection_mode (optional): str, "variable, lag" if the selected set contains lagged information, "variable" otherwise.
    Returns:
        row: dict, contains the different metrics indexed by name, and the associated value.
    """
```

```
def process_all_datasets(dataset_name,num_datasets=10,lags=10):
    all_dataframe=[]
    for i in range(num_datasets):
        filename=f'data_{i}.csv'
        data=creation_data_frame(dataset_name,filename,df,lags)
        all_dataframe.append(data)

    combined_dataframe=pd.concat(all_dataframe,ignore_index=True)
    pd.set_option('display.max_rows', None)
    return combined_dataframe
```

Pour l'obtention des résultats, on commence par afficher les courbes de precision et de recall sur une moyenne des targets en notant que l'aire des courbes corresponds à l'écart-type entre les targets.

3.2.6 Résultats

Pour le dataset VARSmall :

Il est intéressant de commencer par afficher la dataframe finale que l'on obtient recensant les résultats et observations que nous avions définit dans le pipeline d'expérimentation, pour VARsmall, on obtient une dataframe contenant 3609 lignes.

all_data_VARSmall

	FS_size	precision	recall	TP	FP	FN	TN	f1-score	target	filename	dataset name	selected features	lags	selected_set	Execution time
0	1	1.000000	0.166667	1	0	5	4	0.285714	0	data_0.csv	VARSmall/returns	1	1	[8]	9.050803
1	1	1.000000	0.166667	1	0	5	4	0.285714	1	data_0.csv	VARSmall/returns	1	1	[8]	0.310604
2	1	0.000000	0.000000	0	1	6	3	NaN	2	data_0.csv	VARSmall/returns	1	1	[8]	0.120060
3	1	1.000000	0.166667	1	0	5	4	0.285714	3	data_0.csv	VARSmall/returns	1	1	[3]	0.182199
4	1	1.000000	0.166667	1	0	5	4	0.285714	4	data_0.csv	VARSmall/returns	1	1	[4]	0.121695
5	1	1.000000	0.166667	1	0	5	4	0.285714	5	data_0.csv	VARSmall/returns	1	1	[8]	0.123627
6	1	1.000000	0.166667	1	0	5	4	0.285714	6	data_0.csv	VARSmall/returns	1	1	[6]	0.146012
7	1	0.000000	0.000000	0	1	6	3	NaN	7	data_0.csv	VARSmall/returns	1	1	[8]	0.136418
8	1	1.000000	0.166667	1	0	5	4	0.285714	8	data_0.csv	VARSmall/returns	1	1	[8]	0.130937
9	1	0.000000	0.000000	0	1	6	3	NaN	9	data_0.csv	VARSmall/returns	1	1	[8]	0.146885
10	6	0.500000	0.500000	3	3	3	1	0.500000	0	data_0.csv	VARSmall/returns	6	1	[1, 2, 5, 6, 8, 9]	0.411376
11	6	0.666667	0.666667	4	2	2	2	0.666667	1	data_0.csv	VARSmall/returns	6	1	[0, 1, 3, 5, 8, 9]	0.421591
12	6	0.500000	0.500000	3	3	3	1	0.500000	2	data_0.csv	VARSmall/returns	6	1	[0, 1, 2, 5, 8, 9]	0.414586
13	6	0.666667	0.666667	4	2	2	2	0.666667	3	data_0.csv	VARSmall/returns	6	1	[1, 3, 5, 7, 8, 9]	0.432321
14	6	0.333333	0.333333	2	4	4	0	0.333333	4	data_0.csv	VARSmall/returns	6	1	[1, 2, 4, 6, 8, 9]	0.221338
15	6	0.666667	0.666667	4	2	2	2	0.666667	5	data_0.csv	VARSmall/returns	6	1	[0, 1, 5, 7, 8, 9]	0.165328
16	6	0.666667	0.666667	4	2	2	2	0.666667	6	data_0.csv	VARSmall/returns	6	1	[1, 3, 5, 6, 8, 9]	0.170679

A l'aide de la bilbiothèque seaborn, nous affichons les différentes courbes que l'on analyse succinctement.

11

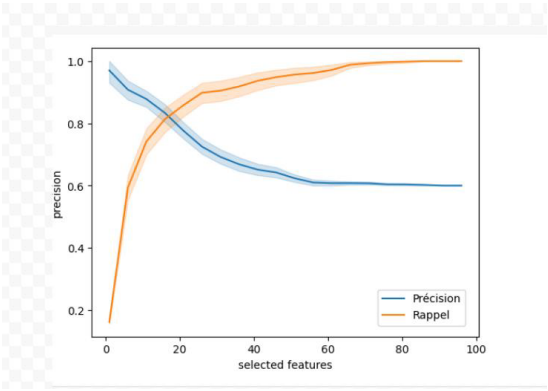


FIGURE 8

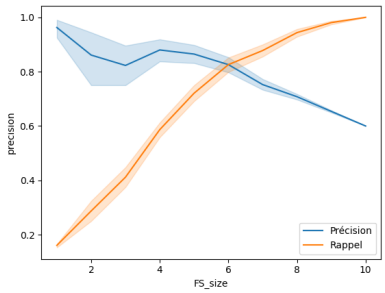


FIGURE 9

En observant la precision et le recall en fonction des features sélectionnées, nous constatons qu'on a tout d'abord une précision de presque 1 pour une variable sélectionnée, pour 6 variables sélectionnées (correspondant à la taille de la Markov Blanket théorique), nous obtenons un f1-score de 0.8, ce qui est une bonne chose, améliorable cependant.

Il est possible d'effectuer une analyse analogue en affichant precision et recall en fonction de FS_size. FS_size représente la taille des colonnes des noms des features sélectionnées après la transformation inverse des données (non-vectorisées). Cette analyse permet de se rapprocher d'une expérimentation avec un dataset plus proche de « l'original ». On constate que l'on a un tradeoff pour FS_size égal à 6, correspondant à la taille du ground_truth du dataset VARSmall.

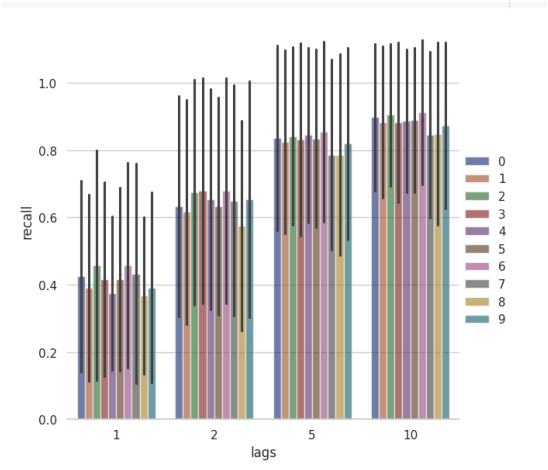


FIGURE 10

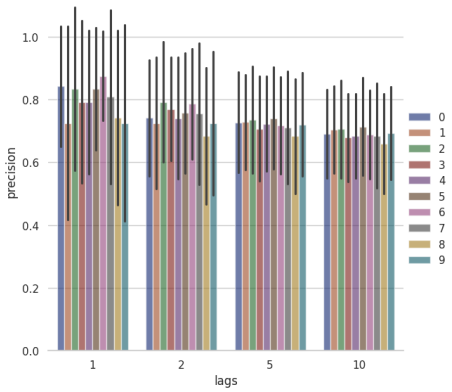


FIGURE 11

Finalement, on effectue un tracé de la precision et du recall en fonction des lags par rapport à chaque filename du dataset VARSmall, contenant 10 filename de data_0 à data_9.

Pour la precision, nous constatons une croissance forte au début, mais qui chute rapidement en fonction des lags après une certaine valeur, on interprète cela par du surapprentissage(overffiting). Pour le recall, on a une augmentation continue avec les lags, en effet, plus le nombre de lags est élevé plus on apporte d'information à l'algorithme à pro-

pos des observations passées, cela aide donc à l'identification des True Positive, améliorant donc la valeur de recall calculée

3.2.7 Conclusion varsmall

Pour MRMR, nous pouvons conclure plusieurs points après l'analyse des tracés obtenus. Tout d'abord, le temps d'éxecution total de l'algorithme comprenant le preprocessing du dataset est d'environ égal à 30610 secondes, soit environ dix heures. En relevant simplement le temps d'éxecution de l'algorithme sans ce preprocessing, nous comptons un temps d'éxecution égal à 3610 secondes soit environ une heure. Ensuite, on constate que l'augmentation de l'hyperparamètre K conduit à du sur-apprentissage ce qui baisse les performances de l'algorithme. Finalement, d'après les tracés obtenus, l'algorithme semble identifier correctement les features pertinentes (celles correspondant à celle du ground_truth).

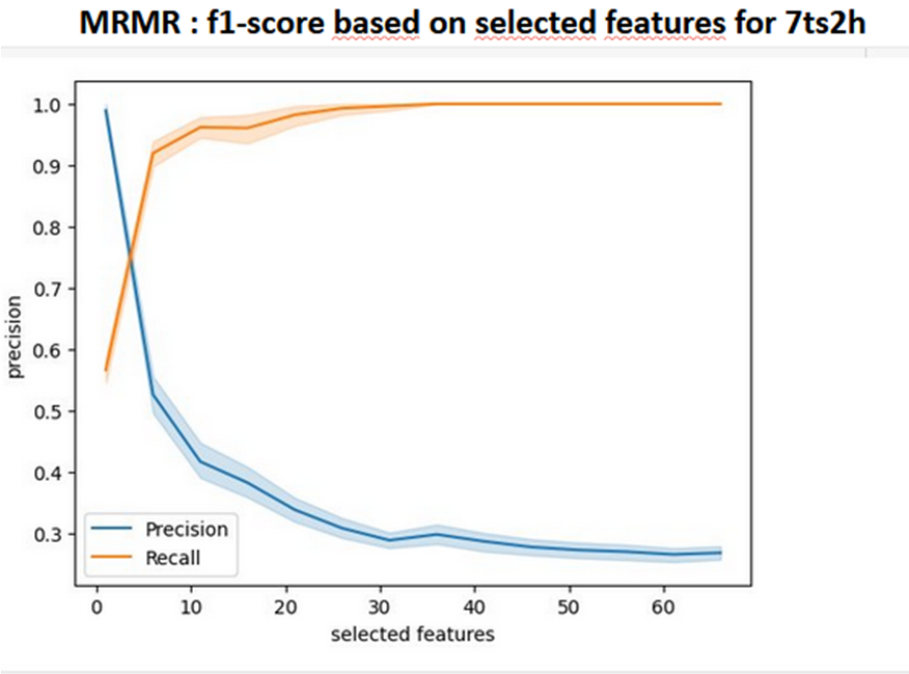
Pour la dataset 7ts2h :

Pour 7ts2h, en affichant la dataframe finale, nous constatons qu'elle possède 7289 lignes

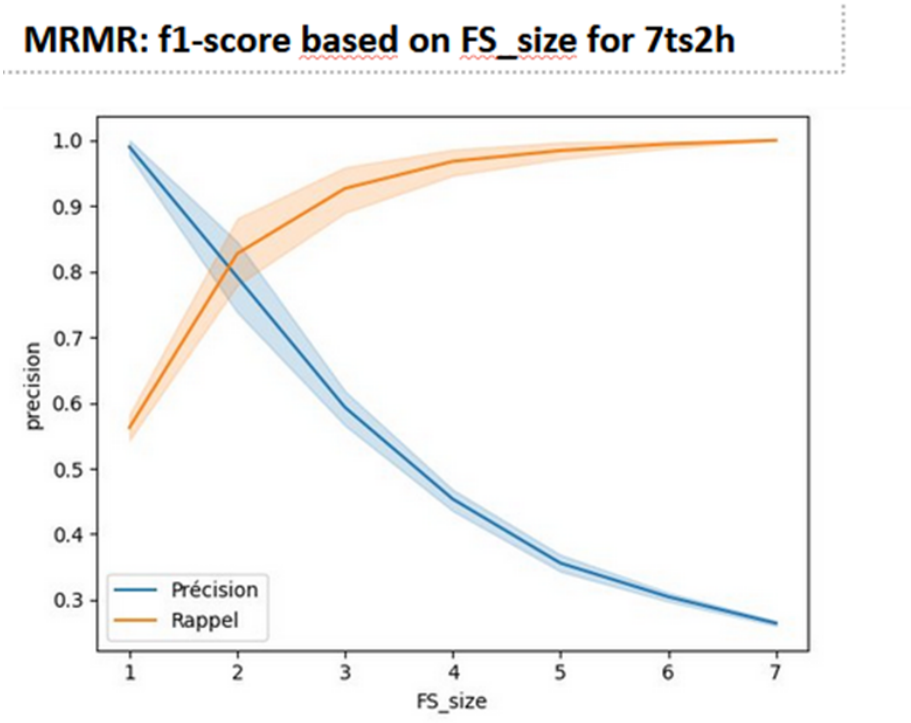
all_data_7ts2h

	FS_size	precision	recall	TP	FP	FN	TN	f1-score	target	filename	dataset name	selected features	lags	selected_set	Execution time
0	1	1.000000	0.5	1	0	1	5	0.666667	E	data_0.csv	SynthNonlin/7ts2h	1	1	[E]	1.504387
1	1	1.000000	0.5	1	0	1	5	0.666667	E	data_0.csv	SynthNonlin/7ts2h	1	1	[E]	0.049799
2	1	1.000000	0.5	1	0	1	5	0.666667	E	data_0.csv	SynthNonlin/7ts2h	1	1	[E]	0.043931
3	1	1.000000	0.5	1	0	1	5	0.666667	E	data_0.csv	SynthNonlin/7ts2h	1	1	[E]	0.042776
4	1	1.000000	0.5	1	0	1	5	0.666667	B	data_0.csv	SynthNonlin/7ts2h	1	1	[B]	0.045631
5	1	1.000000	0.5	1	0	1	5	0.666667	B	data_0.csv	SynthNonlin/7ts2h	1	1	[B]	0.045751
6	1	1.000000	0.5	1	0	1	5	0.666667	B	data_0.csv	SynthNonlin/7ts2h	1	1	[B]	0.043766
7	1	1.000000	0.5	1	0	1	5	0.666667	B	data_0.csv	SynthNonlin/7ts2h	1	1	[B]	0.042947
8	1	1.000000	0.5	1	0	1	5	0.666667	F	data_0.csv	SynthNonlin/7ts2h	1	1	[F]	0.045334
9	1	1.000000	0.5	1	0	1	5	0.666667	F	data_0.csv	SynthNonlin/7ts2h	1	1	[F]	0.043016
10	1	1.000000	0.5	1	0	1	5	0.666667	F	data_0.csv	SynthNonlin/7ts2h	1	1	[F]	0.044322
11	1	1.000000	0.5	1	0	1	5	0.666667	F	data_0.csv	SynthNonlin/7ts2h	1	1	[F]	0.041447
12	1	1.000000	1.0	1	0	0	6	1.000000	C	data_0.csv	SynthNonlin/7ts2h	1	1	[C]	0.045906
13	1	1.000000	1.0	1	0	0	6	1.000000	C	data_0.csv	SynthNonlin/7ts2h	1	1	[C]	0.043868
14	1	1.000000	1.0	1	0	0	6	1.000000	C	data_0.csv	SynthNonlin/7ts2h	1	1	[C]	0.047152

On trace ensuite également les courbes de precision et recall en fonction des features sélectionnées, puis de FS_size :



Nous constatons tout d'abord que comme VARSmall, pour une feature sélectionnée on est proche d'une precision de 1, ensuite, la precision décroît grandement avec l'augmentation des features sélectionnées à l'inverse du recall. Le tradeoff du f1-score est quant à lui à 0.75.



Pour compléter nos analyses, avec cette courbe on constate que ce tradeoff se passe pour FS_size égal à 2, correspondant à la taille de la Markov Blanket théorique, ce qui est un bon indicateur de la performance de l'algorithme.

Pour ce dataset, nous n'avons pas jugé nécessaire d'afficher les résultats de precision et recall en fonction des lags par filename en raison de sa grande similitude avec le dataset VARSmall sur ce point-ci, en effet, on constate à travers ces tracés que la precision baisse après un certains nombre de lags dût au sur-apprentissage et que le recall augmente avec l'apport d'informations sur les observations passées par les lags.

3.2.8 Conclusion partielle de MRMR pour le dataset 7ts2h :

Comme pour VARSmall, on établit une conclusion partielle à propos des résultats observés sur ce dataset. Le temps d'éxecution de l'algorithme en comprenant le preprocessing est environ égal à 21867 secondes soit approximativement 6 heures. Sans le preprocessing des données, nous avons un temps d'éxecution égal à 2867 secondes soit 47 minutes. Le reste des conclusions est semblable à ce que nous obtenions pour VARSmall avec MRMR.

3.2.9 Conclusion finale de MRMR :

Notre application de MRMR sélectionne pour les deux datasets utilisés les bonnes features correspondant à celles du ground_truth, ce qui authentifie la pertinence des features et donc des résultats obtenus. Nous observons également que le f1-score obtenu est assez élevé(proche de 1) ce qui donne un bon indicateur également de la performance générale de l'algorithme. Malgré tout, en comparaison à RRelief, MRMR possède un temps d'éxecution plus long.

3.2.10 Liaison avec les algorithmes de prédictions :

Dans l'optique d'évaluer les performances de MRMR, nous avons effectué une expérimentation sur 7ts2h et VARSmall en sélectionnant un maximum de 3 caractéristiques pour chaque target du jeu de donnée, avec un lags de 10 :

Pour Varsmall :

```
for target in var_names:
  selected=apply_mrmr(df,3,target,10)
  print(f'Pour la target{target} voilà les selected features {selected}')
```

```
['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
defaultdict(<class 'list'>, {'0': ['0', '2', '3', '7', '8', '9'], '1': ['1', '2', '3', '5', '6', '8'], '2': ['2', '3', '4', '5', '7', '9'], '3': ['0', '1', '3', '5', '6', '7'], '4': ['0', '3', '4', '5', '7', '9'], '5
100%|██████████| 3/3 [00:00<00:00,  9.48it/s]
Pour la target0 voilà les selected features ['0', '8', '9']
100%|██████████| 3/3 [00:00<00:00,  8.95it/s]
Pour la target1 voilà les selected features ['8']
100%|██████████| 3/3 [00:00<00:00,  9.04it/s]
Pour la target2 voilà les selected features ['2', '6', '7']
100%|██████████| 3/3 [00:00<00:00,  9.43it/s]
Pour la target3 voilà les selected features ['0', '3', '6']
100%|██████████| 3/3 [00:00<00:00,  9.04it/s]
Pour la target4 voilà les selected features ['0', '4']
100%|██████████| 3/3 [00:00<00:00,  9.10it/s]
Pour la target5 voilà les selected features ['5', '8']
100%|██████████| 3/3 [00:00<00:00,  9.12it/s]
Pour la target6 voilà les selected features ['0', '6']
100%|██████████| 3/3 [00:00<00:00,  9.23it/s]
Pour la target7 voilà les selected features ['0', '6', '9']
100%|██████████| 3/3 [00:00<00:00,  9.22it/s]
Pour la target8 voilà les selected features ['8']
100%|██████████| 3/3 [00:00<00:00,  9.26it/s]Pour la target9 voilà les selected features ['8']
```

Pour 7ts2h :

```
for target in var_names:
  selected=apply_mrmr(df,3,target,10)
  print(f'Pour la target{target} voilà les selected features {selected}')
```

```
['E', 'B', 'F', 'C', 'H', 'D', 'A']
defaultdict(<class 'list'>, {'E': ['E', 'B'], 'B': ['B', 'F'], 'F': ['F', 'C'], 'C': ['C'], 'H': ['C', 'H'], 'D': ['H', 'D'], 'A': ['D', 'A']})
100%|██████████| 3/3 [00:00<00:00,  9.80it/s]
Pour la targetE voilà les selected features ['E']
100%|██████████| 3/3 [00:00<00:00,  9.85it/s]
Pour la targetB voilà les selected features ['B']
100%|██████████| 3/3 [00:00<00:00,  9.67it/s]
Pour la targetF voilà les selected features ['F']
100%|██████████| 3/3 [00:00<00:00,  8.38it/s]
Pour la targetC voilà les selected features ['C', 'A']
100%|██████████| 3/3 [00:00<00:00,  8.43it/s]
Pour la targetH voilà les selected features ['C', 'H']
100%|██████████| 3/3 [00:00<00:00,  8.36it/s]
Pour la targetD voilà les selected features ['H', 'D']
100%|██████████| 3/3 [00:00<00:00,  9.70it/s]Pour la targetA voilà les selected features ['C', 'D', 'A']
```


15

3.3 Notre deuxième algorithme de sélection de variable : Rrelieff

3.3.1 Introduction

RReliefF est conçu pour évaluer les caractéristiques en fonction de leur pertinence vis-à-vis d'une variable cible. En se basant sur les différences entre les valeurs des caractéristiques, cet algorithme attribue des poids adaptés pour déterminer quelles caractéristiques sont les plus informatives pour la prédiction.

3.3.2 Explication de l'algorithme

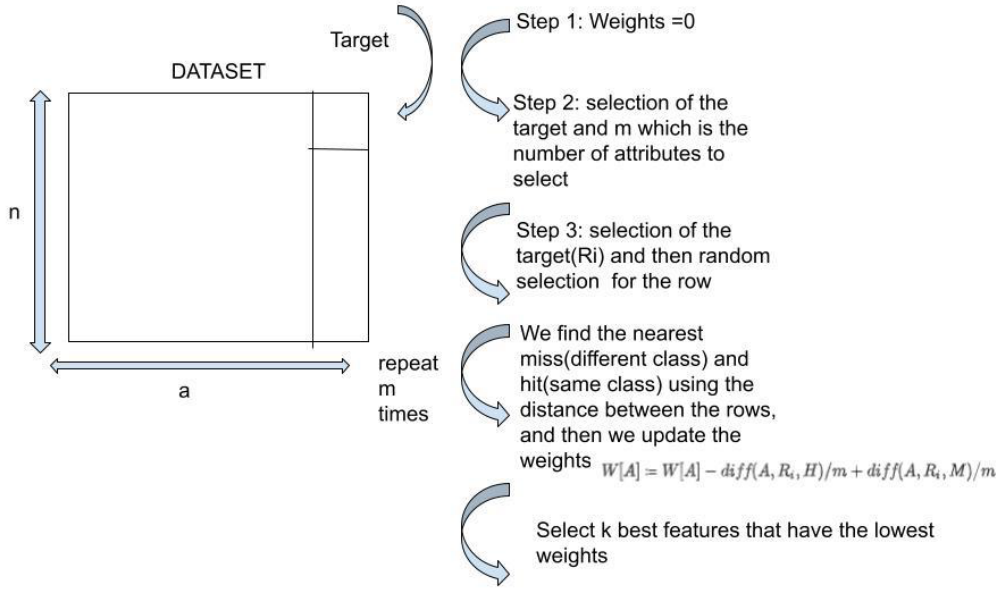


FIGURE 12 – RRelieff algorithme

Le filtre est fondé sur une approche itérative et suit les étapes suivantes :

- On commence déjà par définir : W_{dy} poids lié à la présence de valeurs différentes pour la variable réponse y , W_{dj} poids lié à la présence de valeurs différentes pour le prédicteur Fj et finalement $W_{dy \wedge dj}$ poids lié à la présence de valeurs différentes pour la variable réponse y et pour le prédicteur Fj en même temps.
- Les poids sont tous fixés à zéro
- On choisit aléatoirement une observation X_r
- Pour chaque voisin X_q parmi les k-voisins de X_r on met à jour les poids selon les formules suivantes :
- $W_{dy}^{i} = W_{dy}^{i-1} + \Delta_y(X_r, X_q).d_{rq}$ avec d_{rq} désigne la distance euclidienne entre r et q
- $W_{dj}^{i} = W_{dj}^{i-1} + \Delta_j(X_r, X_q).d_{rq}$
- $W_{dy \wedge dj} = W_{dydj}^{i-1} + \Delta_y(X_r, X_q).d_{rq}.\Delta_j(X_r, X_q)$

3.3.3 Hyperparamètres

Rrelieff possède plusieurs hyperparamètres dont ceci :

- K : Le nombre de voisins les plus proches à utiliser par classe (par défaut, 10).
- Approx_decimals : Le nombre de décimales à approximer pour les valeurs réelles lors du calcul des probabilités (par défaut, dépendant du contexte).
- Ramp : Vrai pour utiliser la fonction de distance en rampe (par défaut, Faux).
- Tdiff : Utile uniquement lorsque 'ramp=True', ces deux paramètres représentent respectivement le seuil de distance minimum à partir duquel deux valeurs sont traitées comme égales (la distance sera de 0), et le seuil de distance maximum à partir duquel deux valeurs sont traitées comme maximalemenet différentes (la distance sera de 1).
- Sigma : Paramètre d'hyper-scaling du facteur de distance.

3.3.4 Implémentation

Après avoir importé notre filtre RRelieff de sklearn_relief en tant que sr

```
def apply_Rrelieff(data, target, lags, i, cache={}):
    key = (id(data), target, lags)
    if key not in cache:
        X, y, _ = prepare_data_vectorize(data, lags, target)
        X_df = pd.DataFrame(X, columns=[f'lag_{i}' for i in range(1, X.shape[1] + 1)])
        rrelief_instance = sr.RReliefF(n_features=i)
        rrelief_instance.fit(X, y)
        selected_feature_indices = transform(rrelief_instance, X_df)
        vector_mask = [name in selected_feature_indices for name in range(len(X_df.columns))]
        selected_feature_names = vector_mask_to_columns(vector_mask, data)
        cache[key] = (rrelief_instance, X_df, selected_feature_names)
    else:
        rrelief_instance, X_df, selected_feature_names = cache[key]
        if rrelief_instance.n_features != i:
            rrelief_instance.n_features = i
            selected_feature_indices = transform(rrelief_instance, X_df)
            vector_mask = [name in selected_feature_indices for name in range(len(X_df.columns))]
            selected_feature_names = vector_mask_to_columns(vector_mask, data)
            cache[key] = (rrelief_instance, X_df, selected_feature_names)

    return selected_feature_names
```

FIGURE 13 – RRelieff implémentation

Dans ce code, une fonction nommée 'apply_Rrelieff' est définie pour appliquer l'algorithme ReliefF sur les données. Voici une explication de chaque étape :

- Initialisation du cache : Un cache est utilisé pour stocker les résultats calculés précédemment afin d'éviter de recalculer les mêmes valeurs. Le cache est un dictionnaire qui utilise une clé composée de l'identifiant unique des données, de la variable cible et de la longueur des retards (lags).
- Vérification du cache : Avant de procéder au calcul, la fonction vérifie si les résultats pour les données actuelles sont déjà présents dans le cache. Si les résultats sont trouvés, ils sont récupérés directement à partir du cache. Sinon, le calcul est effectué
- Préparation des données : Les données sont préparées en utilisant la fonction 'prepare_data_vectorize', qui extrait les caractéristiques des données en utilisant des fenêtres glissantes et les transforme en vecteurs unidimensionnels.

17

— Création d'un DataFrame des caractéristiques : Les caractéristiques extraites sont converties en un DataFrame pandas avec des noms de colonnes correspondant à chaque retard.
— Initialisation de l'algorithme ReliefF : Une instance de l'algorithme ReliefF est créée avec le nombre spécifié de caractéristiques à sélectionner (i).
— Apprentissage de l'algorithme ReliefF : L'algorithme ReliefF est ajusté aux données d'entraînement pour sélectionner les caractéristiques les plus importantes.
— Transformation des caractéristiques sélectionnées : Les indices des caractéristiques sélectionnées sont transformés en noms de colonnes à partir du DataFrame des caractéristiques.
— Mise à jour du cache : Les résultats sont stockés dans le cache pour une utilisation ultérieure.
— Renvoi des caractéristiques sélectionnées : Les noms des caractéristiques sélectionnées sont renvoyés en sortie de la fonction.

3.3.5 Utilisation du cache

Pour optimiser le processus et éviter de refaire le prétraitement du jeu de données à chaque fois, nous pouvons mémoriser l'instance rrelief_instance après son ajustement (fit), ainsi que la longueur de X_df.

3.3.6 Pipeline des experiences

Le filtre Relief utilise exactement le même pipeline que MRMR défini précédemment. Il fait également appel aux fonctions 'open_data_set_ground_truth' et 'process_all_datasets' pour obtenir un DataFrame contenant toutes les métriques dont nous aurons besoin.

3.3.7 Résultats

Pour le dataset VARSmall, nous obtenons ceci :

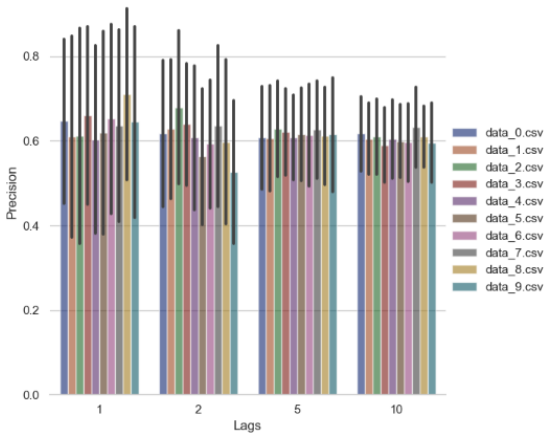


FIGURE 14

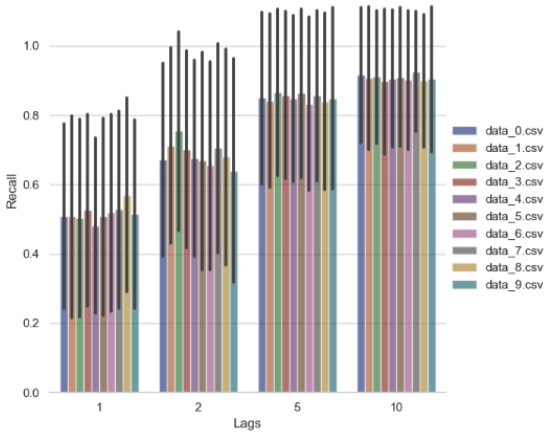


FIGURE 15

18

Remarques :

- L'intervalle de confiance diminue pour la précision, ce qui indique une incertitude.
- À mesure que davantage de retards sont ajoutés, le rappel augmente.
- L'intervalle de confiance à 95 % augmente à mesure que le nombre de retards augmente pour le rappel, ce qui indique une certitude.

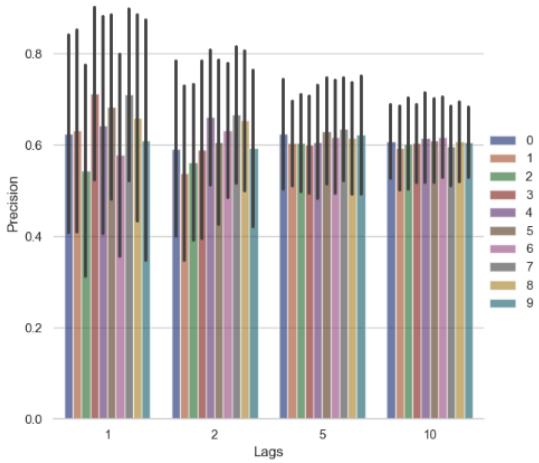


FIGURE 16

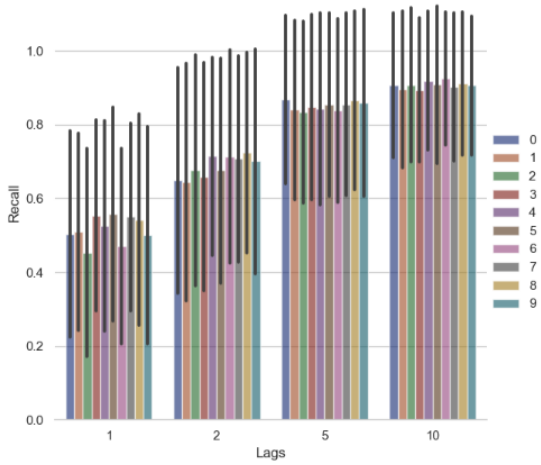


FIGURE 17

Remarques :

- Le rappel continue de s'améliorer même au-delà du nombre théorique de retards et atteint son maximum pour un retard de 10.
- On observe une stabilité indépendante des cibles pour des retards plus élevés.

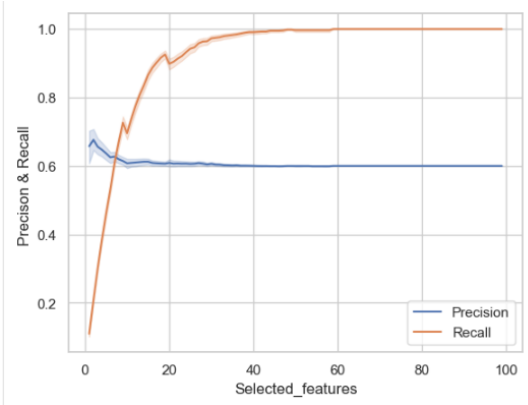


FIGURE 18

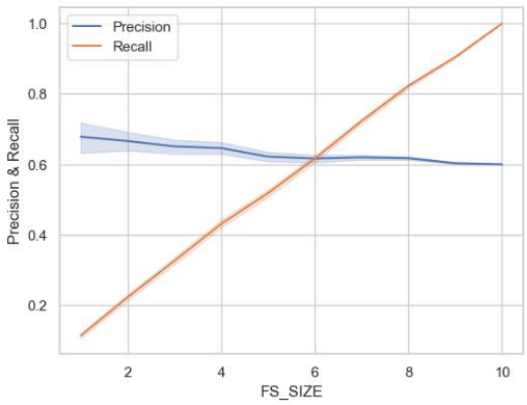


FIGURE 19

Remarques :

- La précision reste approximativement constante autour de 0,6
- Augmentation du rappel avec les caractéristiques sélectionnées
- Tradeoff pour FS_SIZE=6
- Le rappel augmente linéairement avec la taille de l'espace des caractéristiques (FS_SIZE).

19

3.3.8 Conclusion varsmall

Le temps d'exécution total est de 21264 secondes, une taille d'espace des caractéristiques (FS_SIZE) d'environ 6 pourrait être optimale pour ce jeu de données particulier, suggérant un bon compromis entre précision et rappel, tandis que le meilleur lag semble être de 10. Pour le dataset 7ts2h, nous obtenons ceci :

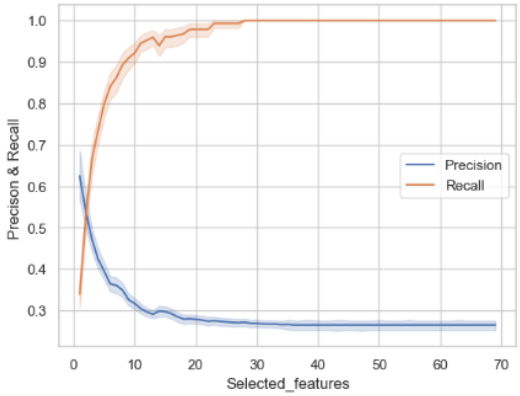


FIGURE 20

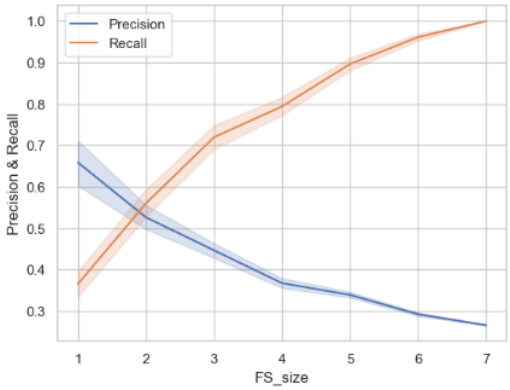


FIGURE 21

Remarques :

- Le rappel se stabilise à un niveau élevé, indiquant une récupération cohérente des instances pertinentes
- La précision atteint également un plateau, suggérant que le modèle maintient son exactitude dans la prédiction des instances pertinentes à mesure que davantage de caractéristiques sont incluses
- Un compromis est observé pour FS_SIZE=2
- Le rappel s'améliore, reflétant l'augmentation de la capacité du modèle à identifier toutes les instances pertinentes
- Il y a une diminution de la précision, ce qui signifie que le modèle capture plus d'instances non pertinentes dans ses prédictions

3.3.9 Conclusion finale de RRelieff

RReliefF est adapté aux scénarios nécessitant un temps d'exécution rapide plutôt que de maximiser le score F1, capable de sélectionner des caractéristiques avec un équilibre bénéfique entre précision et rappel à un retard optimal de 10.

3.3.10 Liaison avec les algorithmes de prédictions

Dans le but d'évaluer les performances de RRelief, nous avons mené une expérimentation sur les ensembles de données 7ts2h et VARSmall. Nous avons sélectionné un maximum de 3 caractéristiques pour chaque cible du jeu de données, avec un décalage de 10.

20

Pour varsmall :

```
for target in var_names:
    selected = apply_Rrelieff(df_varsmall, target, 10, 3)
    print(f'Pour la target {target } voilà les selected features {selected}')
```

```
Pour la target 0 voilà les selected features ['0', '4']
Pour la target 1 voilà les selected features ['0', '2', '6']
Pour la target 2 voilà les selected features ['0', '2']
Pour la target 3 voilà les selected features ['4', '6', '7']
Pour la target 4 voilà les selected features ['2', '3']
Pour la target 5 voilà les selected features ['2']
Pour la target 6 voilà les selected features ['2', '4', '7']
Pour la target 7 voilà les selected features ['2', '3', '7']
Pour la target 8 voilà les selected features ['2', '6']
Pour la target 9 voilà les selected features ['2', '4', '6']
```

FIGURE 22

Pour 7ts2h :

```
for target in var_names:
    selected = apply_Rrelieff(df4, target, 10, 3)
    print(f'Pour la target {target } voilà les selected features {selected}')
```

```
Pour la target E voilà les selected features ['E', 'B', 'C']
Pour la target B voilà les selected features ['B', 'F', 'C']
Pour la target F voilà les selected features ['F', 'D']
Pour la target C voilà les selected features ['C', 'H', 'D']
Pour la target H voilà les selected features ['C', 'H']
Pour la target D voilà les selected features ['D', 'A']
Pour la target A voilà les selected features ['D', 'A']
```

FIGURE 23

21

4 Etude des algorithmes de prédiction

Nous avons étudié principalement 3 algorithmes : le LSTM (Long-Short Term Memory, le TFT (Temporal Fusion Transformer) et dans une moindre mesure le DeepAR. Les réseaux LSTM (Long Short-Term Memory) sont une catégorie de réseaux de neurones récurrents (RNN) conçus pour capturer les schémas de répétition dans le temps.

Les LSTM intègrent des mécanismes de porte (portes d'oubli, portes d'entrée et portes de sortie) qui permettent de réguler le flux d'information à travers le réseau, facilitant ainsi la mémorisation de schémas complexes sur de longues séries temporelles.

Ils sont largement utilisés dans les applications de séries temporelles pour leur capacité à traiter des données avec des schémas de répétition complexes.

Le Temporal Fusion Transformer est un modèle de prédiction de séries temporelles qui s'appuie sur l'architecture Transformer.

Contrairement aux modèles basés sur les réseaux de neurones récurrents, le TFT utilise des mécanismes d'attention pour capturer les relations temporelles entre les différentes variables, ce qui en fait une option puissante pour les séries temporelles multivariées.

DeepAR est un modèle de séries temporelles probabiliste développé par Amazon.

Il utilise une architecture de RNN pour capturer les motifs temporels complexes. DeepAR est particulièrement adapté à la prédiction probabiliste, générant des distributions de probabilité pour les prédictions futures.

4.1 Nos algorithme de prédiction : Temporal Fusion Transformer (TFT), DeepAR

4.1.1 Explication des algorithmes

Le Temporal Fusion Transformer (TFT) utilise l'architecture Transformer pour prédire des séries temporelles multivariées. Le processus commence par encoder chaque série temporelle à l'aide d'embeddings temporels, préservant la structure temporelle. Les mécanismes d'attention sont employés pour mettre en évidence les relations importantes.

La fusion temporelle combine ces informations avec les poids d'attention, créant une représentation globale des séries temporelles.

Le TFT peut générer des prédictions sur plusieurs horizons temporels simultanément, anticipant les valeurs futures pour différentes variables.

Cependant, nous allons l'utiliser pour prédire une seule étape future. L'optimisation utilise la rétropropagation du gradient pour minimiser une fonction de perte (dans notre cas la RMSE), ajustant ainsi les paramètres du modèle.

En résumé, le TFT exploite les mécanismes d'attention de l'architecture Transformer pour capturer les relations temporelles dans les séries temporelles multivariées.

DeepAR utilise une architecture de RNN pour modéliser les séries temporelles multivariées.

Chaque série temporelle est encodée séquentiellement, permettant au modèle de capturer les relations temporelles et les dépendances entre les différentes variables. Les motifs temporels sont appris à travers les différentes couches du réseau de neurones, permettant à DeepAR de générer des prédictions précises et flexibles pour les séries temporelles.

Contrairement au TFT, DeepAR génère des distributions de probabilité pour chaque point de prédiction, offrant ainsi une estimation de l'incertitude associée à chaque prévision.

Finalement, DeepAR utilise une architecture de RNN pour modéliser les séries temporelles multivariées et génère des prédictions probabilistes pour les valeurs futures.

4.1.2 Implémentation

Cette sous-partie plonge dans les aspects pratiques de l'implémentation des algorithmes, décrivant les étapes cruciales de sa mise en oeuvre dans le contexte des séries temporelles multivariées.

Nous avons choisi pour l'implémentation du TFT et de DeepAR, la librairie PyTorch Forecasting qui est particulièrement adaptée aux problèmes de prédiction des séries temporelles multivariées.

L'implémentation a été assez difficile car la documentation disponible et les exemples d'utilisation ne sont pas très riches.

4.2 Résultats

4.2.1 Expériences

Expérience 1 Dans cette première expérience, nous avons poursuivi notre évaluation des performances des algorithmes de prévision TFT et DeepAR dans trois configurations distinctes.

Nous avons mesuré ces performances dans trois cas différents : d'abord, en utilisant les variables sélectionnées par l'algorithme MRMR, ensuite en utilisant les variables sélectionnées par l'algorithme Rrelief, et enfin sans aucune sélection de variable.

Cette approche nous a permis de continuer à explorer l'impact de la sélection des variables sur les performances prédictives des modèles, tout en étudiant comment ces algorithmes réagissent dans différentes conditions de sélection de caractéristiques.

Ces résultats fourniront des informations précieuses pour orienter nos choix dans le processus de sélection et d'optimisation des variables pour améliorer les performances des modèles de prévision.

Voici les résultats :
Les résultats obtenus sur le Dataset VARSmall révèlent une amélioration significative des performances moyennes pour les deux méthodes de sélection de variables.

En utilisant l'algorithme MRMR, nous avons observé une amélioration moyenne de 14%, tandis que l'utilisation de l'algorithme Rrelief a conduit à une amélioration moyenne de 8%.

Sur le dataset 7TS2H, une tendance similaire a été observée, avec une amélioration moyenne de 11% pour l'algorithme MRMR et une amélioration moyenne de 16% pour l'algorithme Rrelieff.

Ces résultats démontrent l'efficacité des deux approches de sélection de variables pour améliorer les performances des modèles de prévision, avec des gains significatifs tant sur le Dataset VARSmall que sur le dataset 7TS2H.

Cette expérience met clairement en évidence l'importance cruciale de la sélection de variables dans le processus de modélisation de prévision de séries temporelles multivariées.

Les résultats démontrant des améliorations significatives des performances moyennes avec l'utilisation des méthodes MRMR et Rrelief soulignent l'efficacité de ces approches pour identifier les caractéristiques les plus pertinentes.

Ces constatations renforcent l'idée selon laquelle une sélection minutieuse des variables, en utilisant des techniques telles que MRMR et Rrelieff, peut conduire à des modèles de prévision plus performants et plus précis.

Experience 2 Pendant le processus d'optimisation sur les ensembles de données 7TS2H et VARSmall, nous avons entrepris une analyse approfondie de l'influence des hyperparamètres sur les performances des modèles de prévision de séries temporelles multivariées.

Pour ce faire, nous avons calculé le coefficient de variation moyen de la RMSE pour chaque cible. Cette approche nous a permis d'évaluer la stabilité des performances des modèles à travers différentes configurations d'hyperparamètres. En observant la variation des performances pour chaque cible, nous avons pu identifier les combinaisons d'hyperparamètres qui offraient une meilleure stabilité et des prévisions plus fiables.

Cette analyse approfondie nous a fourni des informations précieuses sur la sensibilité des modèles aux variations des hyperparamètres, ce qui a éclairé nos décisions en matière de sélection et d'ajustement des hyperparamètres pour obtenir des modèles de prévision optimaux.

Voici les résultats :

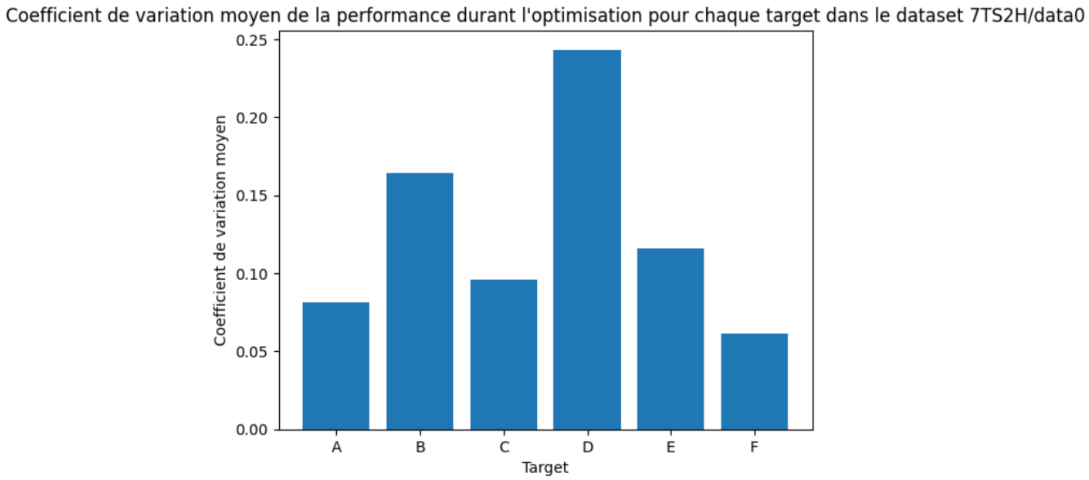


FIGURE 24

Au cours du processus d'optimisation sur les ensembles de données 7TS2H et VARSmall, une analyse approfondie de l'impact des hyperparamètres sur les performances des modèles de prévision de séries temporelles multivariées a été entreprise.

Le coefficient de variation moyen de la RMSE pour chaque cible a été calculé afin d'évaluer la stabilité des performances des modèles à travers différentes configurations d'hyperparamètres.
Cette approche nous a permis de comprendre la sensibilité des modèles aux variations des hyperparamètres et d'identifier les combinaisons offrant une meilleure stabilité et des prévisions plus fiables.

Les résultats ont révélé une variation intéressante allant jusqu'à 25% de la moyenne pour la cible D.

Cette variabilité souligne l'importance de l'hyperparamétrage pour chaque cible, avec des ajustements différents nécessaires pour chacune.

En considérant que le coefficient de variation n'est qu'une mesure de l'écart type, il est plausible que les points extrêmes dépassent cette valeur.

Ces conclusions soulignent l'impératif d'une sélection et d'un ajustement minutieux des hyperparamètres pour chaque cible afin d'obtenir des modèles de prévision optimaux et robustes.

Experience 3 Dans cette expérience, nous avons entrepris d'évaluer la stabilité des prévisions en analysant le coefficient de variation.

Pour ce faire, nous avons divisé le jeu de données en deux groupes en fonction de chaque cible par rapport à sa médiane.

En calculant le coefficient de variation pour chaque groupe, nous avons pu quantifier la

variabilité relative des prévisions par rapport à la médiane pour chaque cible.

En comparant les valeurs du coefficient de variation entre les deux groupes, nous avons pu déterminer si les prévisions étaient relativement plus stables ou plus volatiles pour certaines cibles par rapport à d'autres.

Cette approche nous a permis d'identifier les cibles pour lesquelles les prévisions étaient plus cohérentes et celles pour lesquelles elles étaient plus variables, fournissant ainsi des insights précieux sur la robustesse des modèles de prévision en fonction des différentes cibles.

Voici les résultats pour le dataset 7TS2H :

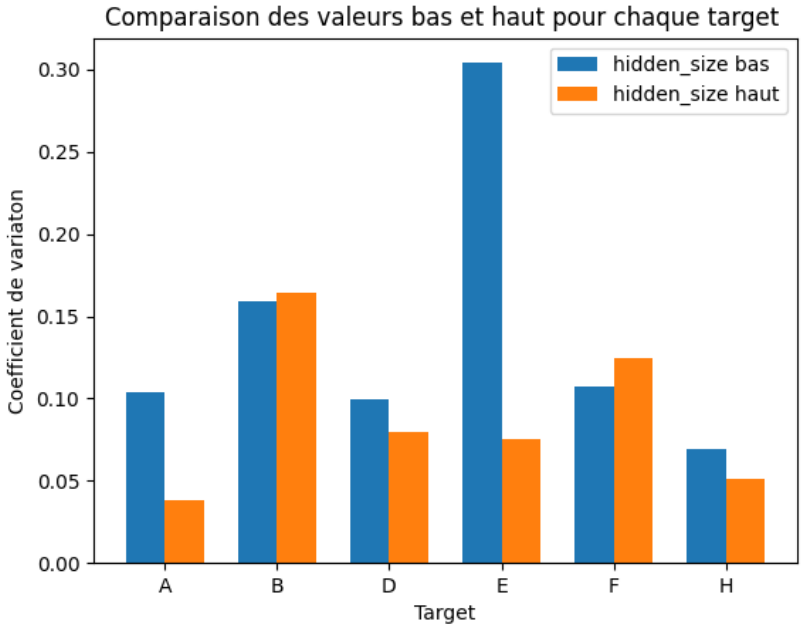


FIGURE 25

26

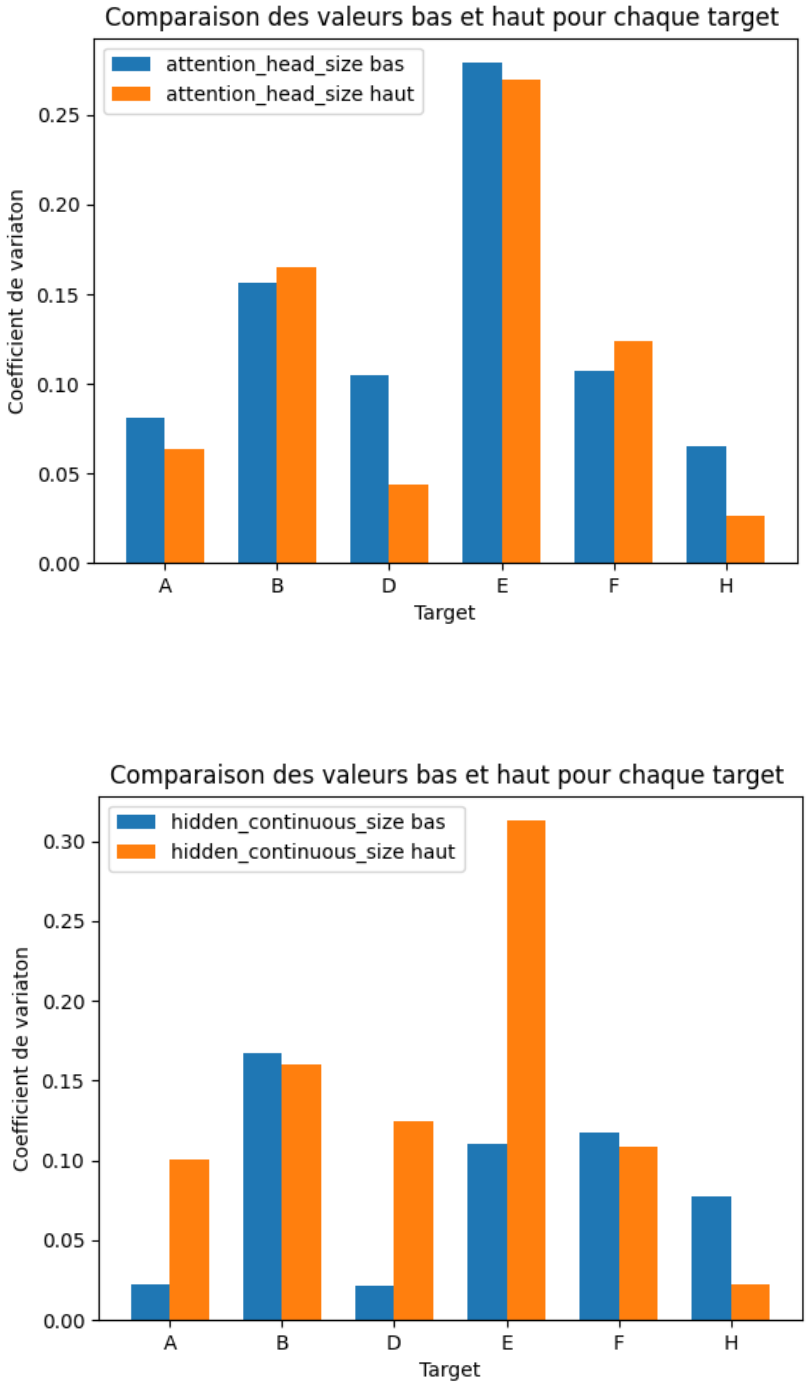
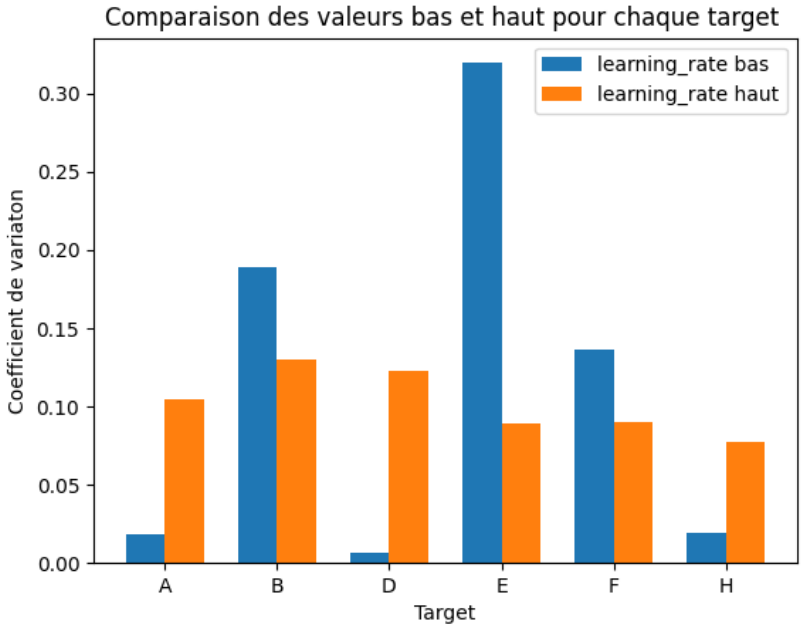
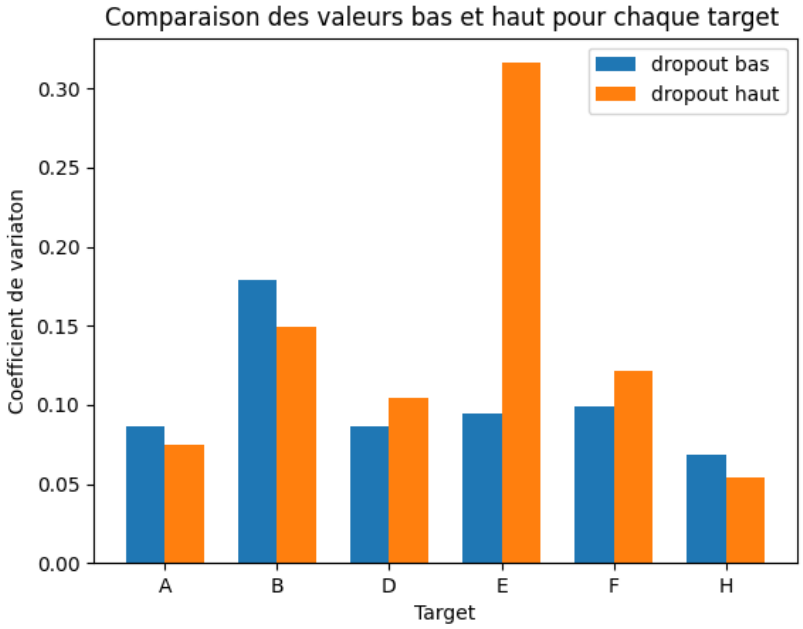

Comparaison des valeurs bas et haut pour chaque target
attention_head_size bas
attention_head_size haut
Coefficient de variaton
0.00
0.05
0.10
0.15
0.20
0.25
A
B
D
E
F
H
Target
Comparaison des valeurs bas et haut pour chaque target
hidden_continuous_size bas
hidden_continuous_size haut
Coefficient de variaton
0.00
0.05
0.10
0.15
0.20
0.25
0.30
A
B
D
E
F
H
Target

FIGURE 26

27





Les résultats de cette expérience révèlent une diversité significative dans la stabilité des prévisions selon les différentes cibles.

Bien qu'aucun schéma clair n'ait émergé, nous avons observé que les variables A, D et H présentaient des profils similaires, tandis que B et F exhibaient des similitudes entre elles.

En revanche, la variable E se distinguait nettement des autres. De plus, nous avons remarqué que les zones de fortes variations des hyperparamètres variaient selon les cibles, en plus de leur amplitude.

28

Cette observation souligne l'importance d'une optimisation individuelle pour chaque cible, afin de tenir compte de ses spécificités et de garantir des prévisions plus stables et précises.

En conclusion, cette analyse met en lumière la nécessité d'adopter une approche personnalisée dans l'optimisation des modèles de prévision, en tenant compte des particularités de chaque cible pour obtenir des résultats optimaux.

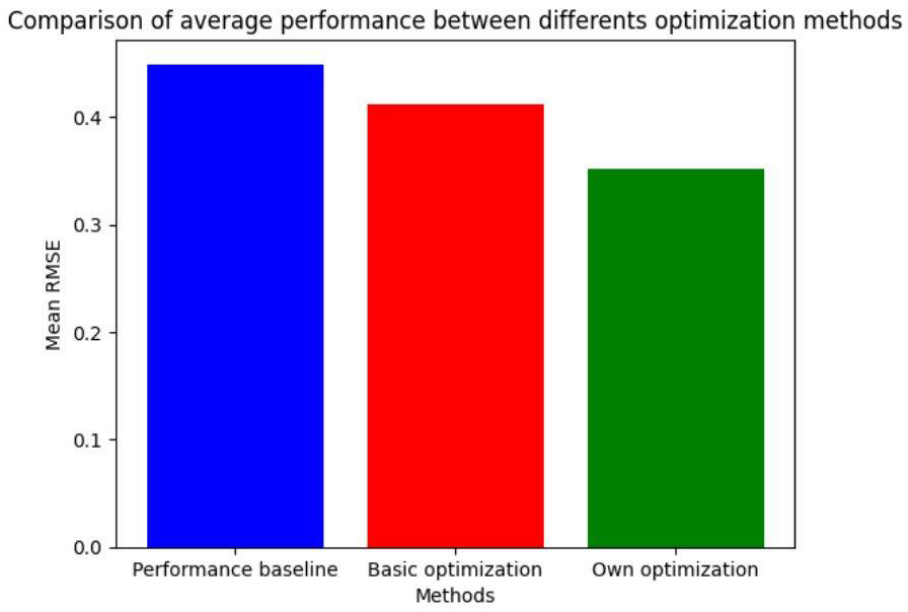
Experience 4 Dans cette dernière expérience, nous avons procédé à une comparaison directe entre deux méthodes d'optimisation des hyperparamètres.

Nous avons calculé la performance minimale, mesurée par la RMSE, pour deux approches différentes : d'une part, nous avons optimisé les hyperparamètres sur l'ensemble du dataset, et d'autre part, nous avons optimisé les hyperparamètres cible par cible.

Cette méthodologie nous a permis d'évaluer l'efficacité de chaque approche dans l'amélioration des performances des modèles de prévision de séries temporelles multivariées.

En comparant les résultats obtenus avec ces deux méthodes, nous avons pu déterminer laquelle offre les gains les plus significatifs en termes de précision prédictive, mettant ainsi en lumière la pertinence de l'optimisation individualisée des hyperparamètres pour chaque cible spécifique.

Voici les résultats :



Les résultats de cette expérience sont révélateurs des bénéfices significatifs qu'offre une approche d'optimisation individualisée des hyperparamètres pour chaque cible spécifique.

En effet, nous avons observé une amélioration de 6% lorsque les hyperparamètres ont été optimisés sur l'ensemble du dataset.

En revanche, lorsque nous avons adopté une stratégie d'optimisation cible par cible, nous avons constaté une amélioration bien plus importante de 17%.

29

Ces résultats mettent en évidence l'impact crucial d'une optimisation fine et personnalisée des hyperparamètres pour chaque série temporelle, soulignant ainsi la nécessité d'adapter cette approche à chaque cible individuelle.

Cette conclusion renforce l'idée selon laquelle une stratégie d'optimisation unique pour l'ensemble du dataset peut ne pas suffire à garantir des performances optimales pour chaque cible, et suggère que chaque série temporelle nécessite une attention et une optimisation spécifiques pour obtenir des prévisions précises et fiables.

4.3 Conclusion

En conclusion, ces expériences ont apporté des réponses essentielles à plusieurs questions cruciales dans le domaine de la prévision de séries temporelles multivariées.

Tout d'abord, nous avons pu confirmer l'influence significative de l'optimisation des hyperparamètres sur les performances des modèles de prévision. L'expérience démontrant une amélioration de 6% lorsque les hyperparamètres ont été optimisés sur l'ensemble du dataset, comparée à une amélioration beaucoup plus importante de 17% lorsqu'ils ont été optimisés cible par cible, souligne l'importance capitale d'une optimisation individualisée.

De même, la nécessité d'une sélection minutieuse des variables a été mise en évidence, avec des améliorations significatives des performances moyennes grâce aux méthodes MRMR et Rrelieff, confirmant ainsi leur efficacité pour identifier les caractéristiques les plus pertinentes.

Ces résultats renforcent l'idée que chaque série temporelle nécessite une approche d'optimisation spécifique, tant pour les hyperparamètres que pour la sélection des variables, afin d'obtenir des modèles de prévision précis et fiables.

Enfin, ces expériences ont fourni des insights précieux sur la sensibilité des modèles aux variations des hyperparamètres et sur l'importance de choisir des algorithmes de sélection de variables performants comme MRMR et Rrelieff.

En combinant ces analyses, nous avons pu affiner notre compréhension des meilleures pratiques en matière de modélisation de prévision de séries temporelles multivariées, jetant ainsi les bases pour des développements futurs dans ce domaine crucial.