Rapport du projet

Time Series Classification: Appliance Detection Problem

RODRIGUES Mathieu, FANG Zicheng, NAJJAR Ahmad et ANDJELOVIC Lazar

Abstract
Ce rapport présente les résultats d'un projet en deux étapes qui vise à résoudre les problèmes liés à la détection d'appareils dans les séries temporelles de consommation d'électricité. La première étape de classification consiste à détecter la présence ou l'absence d'appareils dans la courbe de charge agrégée. La deuxième étape est de détecter les moments précis où les appareils sont en état "ON" en se basant sur la courbe de charge agrégée totale. Les deux problèmes sont résolus en utilisant des algorithmes d'apprentissage automatique qui analysent les données de consommation d'électricité. Les résultats de ce projet ont des implications importantes dans de nombreux domaines, notamment la gestion de l'énergie et la détection des comportements anormaux.

Contents

1 Introduction **1**

2 Problématique **1**

3 Step A **1**

4 Step B **3**

5 Algorithmes **6**

6 Conclusion **7**

1 Introduction

Le présent rapport expose les résultats du projet qui se déroule en deux étapes : le "STEP A" et le "STEP B". Le projet porte sur la classification de séries temporelles (Time Series Classification - TSC), un domaine de recherche important avec des applications dans de nombreux domaines.

Au cours des dernières années, de nombreux fournisseurs d'électricité ont installé des compteurs intelligents dans le monde entier. Ces compteurs enregistrent un grand nombre de séries de données de consommation d'électricité chronologique (également appelées courbes de charge). Des informations précieuses peuvent être extraites de ces séries chronologiques de consommation, telles que la détection de la présence d'un appareil dans une maison, ou la détection de la période dans les données où l'appareil est en état "ON". Ce projet vise à proposer une méthode pour résoudre ces deux problèmes.

2 Problématique

Les deux problèmes abordés dans ce projet concernent la détection d'appareils dans les séries de consommation d'électricité. Dans la première étape, le problème est formulé comme une tâche de classification dont le but est de détecter la présence/absence de différents appareils dans la courbe de charge agrégée. Les données d'entraînement se composent de séries temporelles de consommation et de vecteurs binaires de labels indiquant si chaque appareil a été allumé au moins une fois dans l'ensemble de la série.

La deuxième étape vise à détecter quand un appareil est en état "ON" en se basant uniquement sur la courbe de charge agrégée totale. Ce problème est également lié à la détection d'appareils dans les séries de consommation, mais avec le défi supplémentaire de détecter les moments précis où un appareil est en état "ON".

Dans les deux cas, on utilise des algorithmes d'apprentissage automatique pour analyser les données et classifier les séries de consommation. La première étape peut être abordée à l'aide de classificateurs binaires ou d'un classificateur multiclasse, tandis que la deuxième étape nécessite l'utilisation des séries binaires indiquant l'étiquette de statut pour chaque appareil à chaque instant.

En somme, les deux problèmes ont un objectif commun de détecter les appareils dans les séries de consommation d'électricité, mais diffèrent par le niveau de granularité de la détection.

3 Step A

L'ensemble de données utilisé dans ce projet donne lieu à plusieurs séries chronologiques de données sur la consommation d'électricité de 9 maisons différentes, échantillonnées à une fréquence de 10 secondes. Pour chaque maison, la consommation totale de puissance et la puissance au niveau de l'appareil ont été enregistrées. Dans une série chronologique de consommation, il est alors possible de savoir où un appareil spécifié a été mis en état activé.

Nous avons un fichier CSV pour l'entraînement avec les séries chronologiques avec leurs labels qui sont

0 en l'absence de consommation et 1 en présence. La première étape était de visualiser les données, nous avons donc chargé les données dans un DataFrame pandas (Figure 1) où chaque ligne représente une série chronologique avec son label pour chaque appareil .

	Index	House_id	TimeStep_0	TimeStep_1	TimeStep_2	TimeStep_3	TimeStep_4	TimeStep_5	TimeStep_6	TimeStep_7	...	TimeStep_2150	TimeStep_2151	TimeStep_2152	TimeStep_2153	Ti
0	0	1	180.000000	180.0	180.0	181.0	180.0	180.0	180.000000	181.0	...	238.0	238.0	238.0	236.0	
1	1	1	2437.000000	2426.0	2148.0	645.0	642.0	642.0	2256.666667	2436.0	...	235.0	233.0	233.0	236.0	
2	2	1	232.000000	232.0	232.5	233.0	233.0	234.0	233.000000	233.0	...	185.0	182.0	181.0	180.0	
3	3	1	180.333333	181.0	180.0	184.0	181.0	180.0	183.000000	184.0	...	357.0	351.0	349.0	351.0	
4	4	1	344.000000	341.0	341.0	327.0	327.0	318.0	318.000000	313.0	...	235.0	238.0	238.0	235.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
10416	10416	16	275.000000	272.0	270.0	275.0	269.0	267.0	269.000000	268.5	...	281.0	282.5	278.0	278.0	
10417	10417	16	279.000000	283.5	281.0	281.0	277.0	281.5	283.000000	284.5	...	231.0	230.0	231.0	230.5	
10418	10418	16	229.000000	229.0	231.0	231.0	230.0	230.0	229.500000	230.0	...	231.0	231.0	234.0	231.0	
10419	10419	16	250.000000	247.0	247.0	249.0	246.0	246.5	246.000000	249.5	...	250.0	249.0	247.0	247.0	
10420	10420	16	249.000000	250.0	246.5	249.0	249.0	247.0	247.000000	246.0	...	424.0	371.5	364.0	375.0	

10421 rows × 2162 columns

Figure 1: DataFrame

On peut observer que la proportion de label 0 et 1 pour chaque appareil est très déséquilibrée, il y a une grosse quantité de 0 par rapport au 1 (Figures 2,3,4,5 et 6). Lors de l'apprentissage, on peut se douter que l'apprenant va moins bien reconnaître les 1 car moins d'exemples durant l'apprentissage. C'est pourquoi il va falloir rééquilibrer le dataset pour un meilleur apprentissage. Pour cela, nous avons retiré les séries chronologiques pour lesquelles le vecteur de label avec 5 valeurs vaut 0. Puis nous avons suréchantillonné les données en multipliant par 2 la quantité de données.

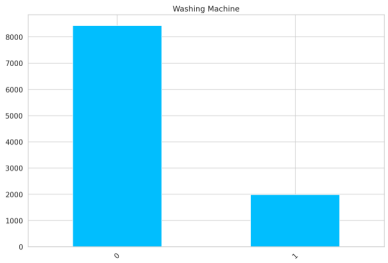

Figure 2: Washing Machine

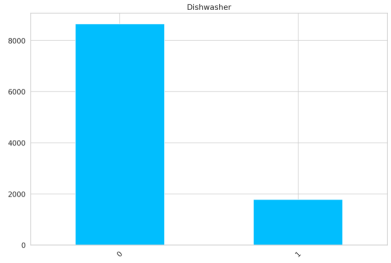

Figure 3: Dishwasher

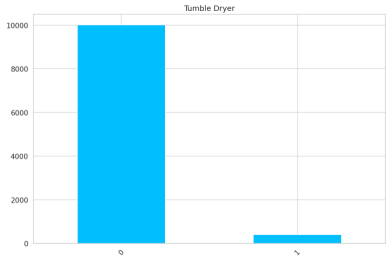

Figure 4: Tumble Dryer

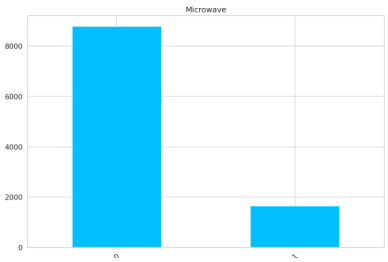

Figure 5: Microwave

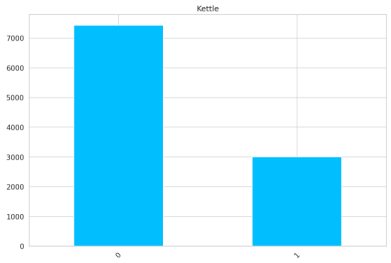

Figure 6: Kettle

Il existe différents algorithmes de classification sur Python. Dans notre cas, nous avons utilisé la librairie Python sktime/xgboost/sklearn pour l'analyse de séries chronologiques. Pour la phase d'apprentissage, nous avons décidé d'utiliser toutes les données d'entraînement pour faire l'apprentissage sans séparer en ensemble de validation. L'une des raisons était que nos données étaient très déséquilibrées et que si nous séparons nos données initiales, nous n'aurions pas assez d'exemples pour faire l'apprentissage du label 1. De plus, les résultats de validation sont généralement différents de ceux des tests et puisque nous avons la possibilité de tester sur des données de test réelles et avoir un feedback sur Kaggle, il était mieux de garder

les données pour l'apprentissage et faire des tests en répétitions.

Dans l'ensemble, notre démarche était d'utiliser différents algorithmes comme les arbres de décisions, les réseaux de neurones, des classifieurs linéaires/non linéaires, etc., et de faire des apprentissages afin de trouver quels classifieurs donnaient le meilleur résultat. Ci-dessous, se trouve un tableau de nos trois modèles qui donnaient les meilleurs résultats sur 70% de l'ensemble de test et sur lesquels nous nous sommes concentrés davantage. Concernant les modèles qui ne sont pas présentés, car l'implémentation n'était pas bonne ou bien les résultats étaient très mauvais par rapport aux autres.

Modèle	Meilleur précision
XgbClassifier()	0,2738
RocketClassifier()	0,33685
TimeSeriesForestClassifier()	0,2357

L'apprentissage a été fait pour chaque appareil au lieu de le faire sur les cinq machines en même temps, parce que cela permettait de mettre en relation une série chronologique et un appareil durant l'apprentissage, pour éviter qu'un autre appareil n'influe sur les résultats des autres appareils. Comme vous pouvez le constater, le modèle qui a la meilleure performance est RocketClassifier() d'après les résultats. RocketClassifier dans sktime est un classificateur basé sur l'algorithme de forêt aléatoire. Plus précisément, il utilise une technique d'encodage de caractéristiques appelée Random Convolutional Kernel Transform (ROCKET) pour extraire des caractéristiques à partir de séries chronologiques. Ces caractéristiques sont ensuite utilisées comme entrée pour un classificateur de forêt aléatoire.

Afin que RocketClassifier puisse traiter nos données, il a fallu les convertir et faire en sorte que notre DataFrame soit transformée en une liste de séries numériques où les valeurs représentent la tension électrique. Sur l'image à gauche, les valeurs ont été z-normalisées avec la fonction zscore, car RocketClassifier est plus performant avec des valeurs normalisées. Pour l'ensemble des arbres, nous avons fait varier le nombre d'estimateurs entre 0 et 10 000. Nous constatons qu'entre 500 et 1000 estimateurs, la performance ne varie pas beaucoup, mais il y a une légère amélioration. Lorsque le nombre d'estimateurs est élevé jusqu'à 10 000, la performance diminue, car il y a sur-apprentissage.

En résumé, nous avons testé et obtenu différents résultats avec différents modèles. Ce que nous nous sommes dits, c'est que puisque nous avons le résultat de différents modèles, nous pouvons refaire un vote en nous basant sur les résultats obtenus pour chaque label prédit. Dans les résultats, nous allons décider selon la majorité : si plusieurs résultats de label sont à 1 pour une série chronologique et qu'il n'y a qu'un seul résultat à 0, alors la majorité décidera d'une nouvelle valeur résultante qui est 1 dans cet exemple. Ainsi, nous avons pu améliorer nos résultats qui passent à 0,39005 avec cette technique.

4 Step B

Notre projet repose sur un ensemble de données composé de 10 421 séries chronologiques de consommation électrique, chacune ayant une longueur de 2160. Pour chaque appareil ménager (machine à laver, lave-vaisselle, sèche-linge, micro-ondes et bouilloire), nous disposons d'une série binaire de même longueur indiquant l'état de fonctionnement de l'appareil (0 pour éteint et 1 pour allumé) pour chaque pas de temps.

Lors de l'exploration des données (Figure 7) nous avons remarqué que dans les valeurs des labels, les "1" correspondant à l'état allumé n'apparaissaient pas de manière "logique". Par exemple, en observant la courbe de tension, un appareil pouvait être allumé avec une courbe élevée (logique car une courbe de tension élevée signifie que des appareil consomme de l'électricité) mais aussi une courbe quasi nulle. Nous avons donc déduit que les états des labels étaient influencés par les tensions précédentes.

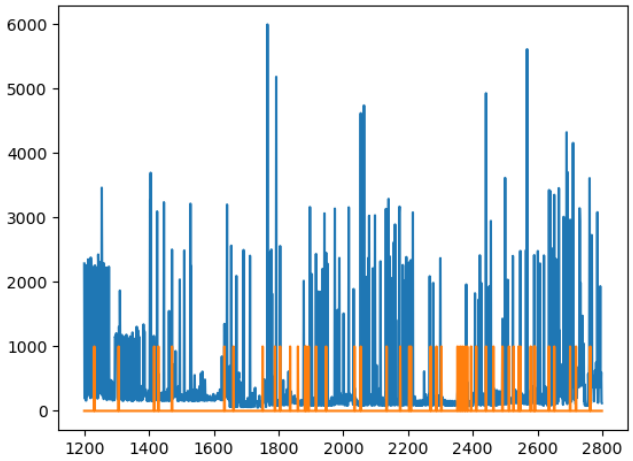

Figure 7: Représentation des données

Notre algorithme doit fournir les étiquettes pour chaque appareil, c'est-à-dire 0 ou 1 pour chaque pas de temps, pour chacune des 2488 séries chronologiques de consommation électrique du jeu de test. Ainsi, notre algorithme doit fournir 2488 séries binaires de longueur 2160 pour chacun des 5 appareils ménagers (soit un total de 12 440 séries binaires). Cet ensemble de données constitue un défi important en raison de sa taille et de la complexité des relations entre les différentes séries chronologiques.

Nous avons essayé plusieurs modèles pour résoudre notre problème de classification de séries chronologiques. Nous avons commencé par utiliser le modèle RandomForestClassifier, qui est un algorithme d'apprentissage supervisé basé sur des arbres de décision. Cependant, ce modèle n'a pas réussi à fournir les résultats souhaités, même après un traitement minutieux des données. Nous nous sommes alors tournés vers XGBClassifier, un modèle de la bibliothèque XGBoost qui utilise une méthode de gradient boosting pour améliorer les prédictions de classification. Nous avons obtenu des résultats encourageants dès les premiers essais avec ce modèle, nous avons donc décidé de l'utiliser comme base pour réaliser différents tests.

Nous avons notamment essayé d'augmenter le paramètre n estimators tout en diminuant le learning rate afin d'obtenir un résultat optimal. Cependant, certains entraînements prenait trop de temps, car nous avions utilisé Google Colab qui ne prêtait ses GPU que pendant une durée variable. Pour remédier à ce problème, nous avons mis en place une méthode appelée Early stopping avec une valeur de patience. Cela permet d'arrêter l'apprentissage si aucune amélioration n'est constatée pendant la patience observée. Ces techniques nous ont permis d'améliorer les performances de notre modèle et de parvenir à des résultats satisfaisants.Nous avons également testé des méthodes de scaling et de normalisation (min-max normalisation, coxbox) (Figure 8) sur l'ensemble des données pour voir si elles pouvaient améliorer les performances de notre modèle. Cependant, cette technique ne nous a pas permis d'augmenter le score de notre modèle.

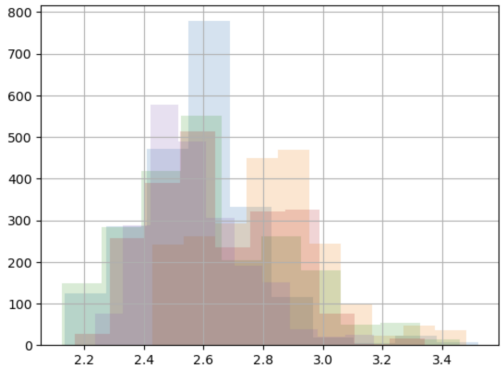

Figure 8: Données après normalisation coxbox - Histogramme des 5 premières lignes, 1 ligne = 1 couleurs

A ce moment, nous avions réalisé beaucoup de modifications sur les données, telles que la suppression de données "bruit" avec des techniques de corrélation (Figure 9), l'ajout de features additionnelles comme la moyenne, l'écart type, les valeurs min et max. Cependant, aucune de ces solutions n'a montré de résultats significatifs. Nous avons également testé des méthodes de scaling et de normalisation sur l'ensemble des données, mais cela n'a pas non plus amélioré notre score. En revanche, une méthode qui nous a permis d'améliorer notre score a été de supprimer les données qui ne contenaient que des 0 (Figure 10) (vecteurs de timestamp ne contenant que des labels 0). Cette solution nous a permis de passer d'un score de 0.06331 à 0.15537 (sur les 70% des données de test) après le testing sur Kaggle.

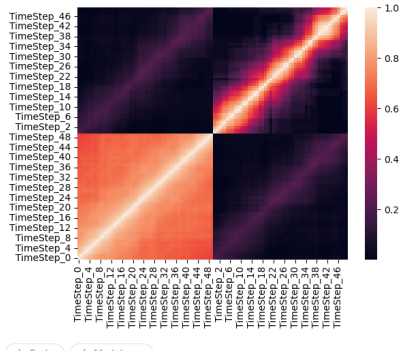

Figure 9: Corrélation entre les features sélectionnées

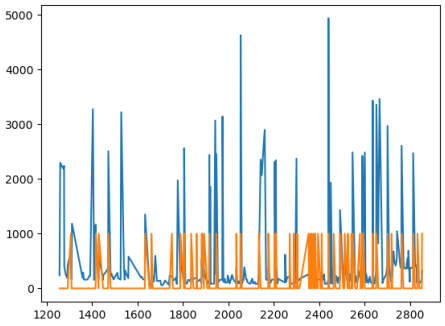

Figure 10: Nouvelle représentation des données

L'amélioration précédente nous a permis de revenir sur les autres mais nous voulions faire partir des premiers du classement on a donc continué les tests. Nous avons donc poursuivi nos tests et avons trouvé une autre amélioration, inspirée de nos tests durant le STEP A : le suréchantillonnage de données. Nous avons augmenté notre ensemble de données de deux fois et sommes passés de 0.15537 à 0.17232 de score (sur les 70% des données de test).

Lors de notre exploration de données, nous avons remarqué que les états des labels semblaient influencés par les tensions précédentes. Cette observation nous a poussés à chercher de nouvelles méthodes de classification qui pourraient mieux répondre à nos besoins. Suite à des remarques lors de nos travaux pratiques de data science, nous nous sommes intéressés aux modèles LSTM. Les LSTM (Long Short-Term Memory) sont des réseaux de neurones récurrents bien adaptés pour traiter des données séquentielles, comme des séries temporelles. Nous avons donc décidé de les tester pour notre problème de classification.

Pour notre modèle LSTM, nous avons utilisé la bibliothèque TensorFlow, une bibliothèque de programmation en Python pour créer un modèle de réseau de neurones. Dans un premier temps, nous avons créé un modèle simple composé d'une couche d'entrée pour spécifier le format de nos données en entrée, une couche LSTM avec 64 neurones et une couche de sortie dense avec une taille de sortie de 2160, correspondant à un vecteur de 2160 timestamps. Nous avons utilisé l'optimiseur Adam et sparse categorical cross-entropy comme fonction de perte. Ces choix ont été faits après des tests et ajustements afin d'optimiser les performances du modèle.

Nous avons tenté différentes solutions pour faire converger notre modèle, telles que changer de fonction de loss, ajouter ou retirer des couches de neurones cachées, ajouter des neurones dans chaque couche et modifier la taille des vecteurs d'entrées en passant de 2160 à 1, 3, 5 ou 40 (on a également modifier la taille des vecteurs de sortie pour que lorsque nécessaire pour avoir des résultats cohérent). Finalement, nous avons obtenu des résultats satisfaisants avec une précision de 0,97 pour l'ensemble d'entraînement et de 0,80 pour l'ensemble de validation. Cependant, notre modèle a obtenu un score très faible de 0,00841 lors de l'évaluation sur Kaggle. Nous avons alors examiné les résultats de prédiction et avons constaté que le classifieur avait mis toutes les prédictions à 0. Pour résoudre ce problème, nous avons mis en place un seuil variable des prédictions en fonction de la moyenne. Par exemple, une moyenne de prédiction de 0,2 signifierait 0 pour les valeurs inférieures ou égales à 0,2 et 1 pour les valeurs supérieures à 0,2. Nous avons donc obtenu un score de 0.04304, un score qui ne nous satisfaisait pas et loin du 0.17232 obtenus avec XGBClassifier.

Après avoir constaté que le modèle ne prédisait que des zéros, nous avons commencé à nous demander si cela était dû au manque de données positives dans l'ensemble d'entraînement. Pour résoudre ce problème, nous avons tenté de créer de nouvelles

données en utilisant les données existantes. Pour ce faire, nous avons aplati notre dataframe, passant de 2160 colonnes à 1 colonne par observation. Ensuite, nous avons concaténé chaque observation avec celle précédente et celle suivante, ce qui a produit un dataframe de 3 colonnes par observation, avec un nombre de lignes multiplié par 2160.

Nous avons finalement décidé de revenir à notre classifieur précédant, le XGBClassifier, face aux nombreuses difficultés rencontrées avec le modèle LSTM. Nous avons ainsi choisi de tenter d'augmenter notre score en sachant que la marge de progression serait limitée compte tenu des efforts déjà déployés.

5 Algorithmes

Algorithm 1: Pseudo-code pour l'apprentissage.

Initialisation des données dans du DataFrames:
    Train = charger le CSV train
    Test = charger le CSV test
    Label = charger le CSV Label
Rééquilibration de la proportion de label 0 et 1: //en enlevant les vecteurs de label à 0
    Train = Train[garde les lignes où le vecteur label correspondant est à 0]
Sur-échantillonage:
    Train = Train **multiplier par 2**
Normalisation de Train et test:
    Train = **Normalisation**(Train)
    Test = **Normalisation**(Test)
Convertir les données: //pour correspondre au format approprié des modèles d'apprentissage
    X_Train = **convertir**(Train) en un input pour le modèle choisi
    X_Test = **convertir** (Test) en un input pour le modèle choisi
prediction = **emptyList()**
Faire l'entrainement pour chaque appareil:
        Y_train = le label correspondant à X_Train
        Model = On choisi un modèle d'apprentissage avec les paramètres (rocket/xgboots...)
        Model.fit(X_Train,Y_train)
        Model.prediction(X_Test) ajouter à prediction

Algorithm 2: décision du label par vote majoritaire.

On récupère les résultats de nos 5 meilleurs apprenants:
    data1 ← données du CSV de l'apprenant 'A'
    data2 ← données du CSV de l'apprenant 'B'
    data3 ← données du CSV de l'apprenant 'C'
    data4 ← données du CSV de l'apprenant 'D'
    data5 ← données du CSV de l'apprenant 'E'

cols_label ← ['Washing Machine', 'Dishwasher', 'Tumble Dryer', 'Microwave', 'Kettle']
mean_cols ← [ ] seuil ← longueur de cols_label / 2 //division entière
Pour chaque col dans cols_label faire:
    res ← somme des valeurs de data1[col], data2[col], data3[col], data4[col], et data5[col]
    res ← [1 si x est supérieur ou égal au seuil, sinon 0 pour x dans res]
    ajouter(res à mean_cols)
Créer(DataFrame df à partir du dictionnaire des paires clé-valeur formé par zip(cols_label, mean_cols))
Ajouter(colonne "Index" à df avec les valeurs de data1['Index'])
Écrire(df dans le fichier CSV 'condition7.csv', sans inclure l'index automatique.)

6 Conclusion

En conclusion, notre étude a montré que la prédiction des états de tension dans un système électrique est un défi complexe, qui nécessite une compréhension approfondie des données et des techniques de modélisation avancées. Nous avons exploré différentes approches, notamment l'utilisation de modèles LSTM et XGBoost, ainsi que des techniques de sur-échantillonnage et de génération de données synthétiques. Bien que nous ayons obtenu des résultats encourageants, nous avons également constaté des limites dans notre approche, avec un score final sur Kaggle de seulement 0.17305 (lors du test Private). Des recherches futures pourraient inclure l'exploration de nouvelles techniques de modélisation, l'ajout de nouvelles variables ou encore l'utilisation d'algorithmes d'apprentissage automatique plus avancés. En somme, ce projet nous a permis d'en apprendre davantage sur les techniques d'apprentissage machines en nous offrant la possibilité de tester nos connaissances sur un problème potentiellement réel.