Université Paris Cité

Rapport de Projet

Time Series Classification

- Appliance Detection Problem -

Projet réalisé par :

YHIA Ounas
THILAHENDRAN Thinujan
KOURBANHOUSSEN Idriss
Nom de l'équipe : IoT

Projet encadré par :

Themis Palpanas
Lucrezia Tosato

Promotion 2022/2023

I. Introduction

Ce projet vise à proposer une méthode pour la classification et la détection de périodes d'activation d'appareils électriques à partir de données de consommation d'électricité. Ce domaine de recherche appelé Classification de Séries Temporelles (TSC), est en constante évolution et a de nombreuses applications dans divers domaines.

Le projet est divisé en deux parties distinctes (step A / B) :

- Dans le step A, l'objectif est de fournir une solution pour résoudre le problème de classification des séries temporelles de consommation d'électricité. C'est-à-dire de déterminer la présence ou l'absence de différents appareils à partir de données de consommation électrique.
- Dans le step B, l'objectif est de fournir une solution pour détecter les périodes d'activation de ces appareils dans la série temporelle. C'est-à-dire de détecter les « Time steps » durant lesquels un appareil spécifié est en état « ON » (donc allumé).

Les résultats de ce projet peuvent être utiles pour les fournisseurs d'électricité, les entreprises spécialisées dans l'efficacité énergétique, ainsi que pour les utilisateurs domestiques souhaitant optimiser leur consommation d'énergie.

Pour mener à bien ce projet, nous avons à disposition un jeu de données constituée des séries chronologiques de données de consommation d'électricité de 9 maisons. Les mesures de consommation ont été échantillonnées à une fréquence de 10 secondes. Sur notre dataset, les données ont été prétraitées en plusieurs séries de consommation de 6 heures pour les 5 appareils suivants :

- Machine à laver
- Lave-vaisselle
- Sèche-linge
- Micro-ondes
- Bouilloire

II. Formulation du problème

Problème STEP A →

Le Step A du projet consiste à résoudre un problème de classification. Plus précisément, il s'agit de détecter avec le plus précision possible si l'un des appareils spécifiés précédemment était allumé ou non pour chaque série temporelle de consommation. L'objectif est donc d'implémenter un algorithme de Machine Learning permettant de réaliser ces classifications. Les données auxquelles nous avons eu accès sont réparties sur deux fichiers (*InputTrain.csv* et *StepOne_LabelTrain.csv*) :

- Le premier est composé d'un ensemble de données chronologiques représentant la consommation d'énergie globale (également appelée courbe de charge agrégée) répartie sur plusieurs séries.
- Le second est un fichier d'étiquettes associées à ces consommations d'énergie et signalant pour chaque appareil mentionné s'il a été mis en marche au moins une fois pour chacune des séries chronologiques.

Enfin, les données de test sur lesquelles nous avons réalisé la classification étaient présentent dans le fichier *InputTest.csv*. Elles se composaient de 2488 séries chronologiques de consommations d'une longueur de 2160, pour lesquelles aucune étiquette n'a été fournie.

Notre algorithme a été entraîné à fournir les libellés (0 : non détecté ou 1 : détecté) des 5 appareils sous la forme de 2 488 vecteurs de taille 5. Chacune des 5 positions du vecteur correspondait à l'un des 5 appareils, dans le même ordre que celui listé ci-dessus. En résumé, nous avons mis en place une méthode de classification minutieuse en entrainant notre modèle sur nos données d'apprentissage pour déterminer avec précision si chaque appareil était allumé ou non sur nos données de test.

Problème STEP B →

Le Step B du projet consiste également à résoudre un problème de classification. Dans cette étape, il s'agit de détecter pour chaque « Time steps » de notre série temporelle de consommation, si l'un des appareils électriques était allumé ou non. L'objectif dans cette partie est donc également d'implémenter un algorithme de classification pour résoudre ce problème. Les données misent à notre disposition étaient réparties sur 6 fichiers :

- Le premier était le même fichier *inputTrain.csv* présenté un peu plus haut
- Les 5 autres fichiers représentent les étiquettes associées à chaque « Time step » de chaque série chronologique de consommation électrique.

Enfin, les données de test sur lesquelles nous avons réalisé la classification étaient également présentent sur le fichier *InputTest.csv*

Notre algorithme a été entraîné à fournir les étiquettes (0 : OFF ; 1 : ON) à chaque « Time steps », pour chacune des 2488 séries de consommation de l'ensemble de test. Ainsi, notre algorithme fournit 2488 séries binaires de longueur 2160 pour chacun des 5 appareils ; soit un total de 2488 x 5 = 12 440 séries binaire de longueur 2160.

III. Solution

STEP A →

Pour le Step A, nous avons testé différentes approches pour prédire si les appareils électriques ont été utilisés pour chaque série de consommation. Nous ne sommes pas passés par l'étape de pré-traitement car il nous était précisé que les données que nous avons utilisées avaient déjà été prétraitées. Pour chaque approche, on distingue les variables **features** (X_train, X_test) qui correspondent aux données de consommation électrique de la variable **target** (y_train) qui désigne les étiquettes (0 ou 1) à prédire.

Dans notre première approche du problème, nous avons implémenté un algorithme simple qui entraîne un modèle de classification sur l'ensemble des variables features et target. L'idée était de pouvoir prédire ensuite, l'étiquette (0 ou 1) correspondante pour chaque appareil sur nos données de test ne contenant que les variables features.

Les prévisions réalisées par notre modèle ont été stockées dans un DataFrame appelé y_pred, qui est indexé avec les numéros de ligne de y_pred.

Nous avons testé cette approche avec différents types de classificateurs, tels que le **Decision Tree Classifier**, **le K-Nearest Neighbors Classifier** et **le Random Forest Classifier**. Cependant, le modèle **XGBoost** s'est avéré être le plus performant, avec un score de 0,329 sur Kaggle.

XGBoost est un algorithme de classification basé sur des arbres de décision, mais il se distingue des autres classificateurs par l'ajout d'une couche d'optimisation supplémentaire qui lui confère des avantages en termes de performance et de précision.

Le fonctionnement de XGBoost est similaire à celui des arbres de décision classiques. Il s'agit d'un algorithme d'apprentissage supervisé qui cherche à créer un modèle pour classer de nouvelles données. Le modèle est construit à partir d'un ensemble de données d'entraînement, où chaque observation est caractérisée par un ensemble de variables appelées "features" (caractéristiques) et une étiquette de classe appelées "target".

Comparé aux autres classificateurs, XGBoost présente plusieurs avantages. Tout d'abord, il est très rapide et efficace pour traiter des données de grande dimensionnalité. De plus, grâce à l'optimisation supplémentaire, il est moins sujet au surapprentissage des données d'entraînement, ce qui lui permet d'avoir une meilleure précision sur les données de test. Enfin, le processus de "boosting"

permet d'améliorer la précision de manière significative en combinant plusieurs modèles.

Ci-dessous notre pseudo code complet qui explique chaque étape de notre algorithme :

Importer les bibliothèques pandas, matplotlib, seaborn et xgboost

Lire les fichiers InputTrain.csv, StepOne_LabelTrain.csv et InputTest.csv en utilisant pandas et stocker les résultats dans df_x, df_y et df_z respectivement.

Supprimer les colonnes "Index" et "House_id" de df_z, df_y et df_x en utilisant la fonction "drop".

Joindre df_x et df_y sur la colonne "Index" et stocker le résultat dans dataset.

Supprimer la colonne "Index" de dataset en utilisant la fonction "drop".

Diviser dataset en X_train et y_train en supprimant les colonnes "Washing Machine", "Dishwasher", "Tumble Dryer", "Microwave" et "Kettle" de dataset et en stockant le reste dans X_train. Stocker les colonnes "Washing Machine", "Dishwasher", "Tumble Dryer", "Microwave" et "Kettle" dans y_train.

Stockez df_z dans X_test.

Entraîner un modèle de clasifier (exemple xgboost), avec l'utilisation des hyperparamètres, en utilisant les données X_train et y_train.

Utiliser le modèle entraîné pour prédire les valeurs pour X_test et stocker les résultats dans y_pred.

Créer un DataFrame à partir de y_pred et l'indexer avec les numéros de ligne de y_pred.

Ajouter une colonne "Index" au début du DataFrame et l'initialiser avec les numéros de ligne de y_pred.

Renommer les colonnes de ce DataFrame en "Index", "Washing Machine", "Dishwasher", "Tumble Dryer", "Microwave" et "Kettle".

Exporter ce DataFrame sous forme de fichier CSV appelé "Sample_LabelTest.csv".

Dans notre deuxième approche du problème, nous avons entraîné notre modèle sur chaque appareil séparément, puis fusionné les résultats dans un seul DataFrame. Autrement dit, nous avons entrainé notre modèle pour qu'il prédise l'étiquette d'un seul appareil, puis avons répété l'expérience sur chaque appareil. Ci-dessous le pseudo code en rapport avec cette partie :

X_train = l'ensemble des variable features

Pour chaque appareils electrique A :

y_train = etiquettes associées à l'appareil A

Entrainement du modèle sur X_train et y_train

y_pred+"A" = Prédiction sur les données de test X_test pour l'appareil A

Fusion des données y_pred prédite pour chaque appareil

Notre score Kaggle n'a été que de 0,23 en utilisant cette approche.

Enfin, pour la dernière approche du problème, nous avons réalisé les prédictions avec différent classifieurs (tels que le **Decision Tree Classifier** et le **Random Forest Classifier**). Ensuite, nous avons comparé les résultats obtenus (0 ou 1) pour chaque prédiction et avons retenu celle qui était majoritaire. L'algorithme que nous avons utilisé est repris de celui de la première approche mais en le dupliquant pour chaque modèle utilisé. Notre score Kaggle n'a été que de 0,22 en utilisant cette approche.

STEP B →

De la même façon que présenté un peu plus haut, nous avons testé différentes approches pour résoudre le problème posé par le Step B.

Dans notre première approche du problème, nous avons implémenter un algorithme qui entraine puis prédit séparément l'étiquette (0 ou 1) de chaque appareil pour chaque « Time step » donné. Plus simplement, on entrainait notre modèle de classification sur un « Time step » précis et son label correspondant pour un appareil donné et ensuite on réalisait la prédiction sur les données de test sur ce même « Time step ». On reproduisait la démarche pour le « Time step » suivant jusqu'à avoir prédit les 2160 « Time step » de l'appareil et on recommençait avec l'appareil suivant. Ci-dessous le pseudo code correspondant à cette partie (df_x correspond au dataset du fichier *InputTrain* et les variables $df\_y_{1...5}$ correspondent aux labels des appareils électrique pour chaque « Time steps ») :

```
For y in (df_y1, df_y2, df_y3, df_y4, df_y5) :
    cptCol = 0
    Pour i allant de 0 à 2159 :
        X_train = df_x[['TimeStep_'+str(cptCol)]]
        y_train = y[['TimeStep_'+str(cptCol)]]
        Entrainement du modèle sur X_train et y_train
        Prediction sur les données de test et stockage du résultat dans un dataframe correspondant
Fusion des différents dataframe
```

Le défaut de cette méthode est qu'elle est très lente à l'exécution et ne fournit pas un score satisfaisant sur Kaggle. En effet, nous l'avons testé avec différents classifieurs (**Random Forest, Decision Tree**, **XGBoost**, etc.) et le meilleur score que nous avons obtenu était de 0.027 avec Random Forest Classifier.

Dans notre deuxième approche du problème, nous avons décider d'implémenter un réseau de neurones récurrents (RNN).

Les réseaux de neurones récurrents (RNN) sont un type de modèle de classification supervisée. Ils sont particulièrement adaptés aux jeux de données composés de séries chronologiques car ils repèrent les relations de dépendance complexes et non-linéaires entre les différents éléments du dataset. Une illustration simple serait par exemple que l'état d'un appareil (allumé ou éteint) sur un « Time step » influence directement la consommation électrique correspondante.

L'utilisation de RNN nécessite souvent une préparation des données en amont, comme la normalisation par exemple, et exige de réaliser plusieurs tests jusqu'à trouver les bons hyperparamètres permettant d'optimiser le score.

Dans notre algorithme, nous avons configurer un réseau de neurones composé d'une couche RNN *SimpleRNN*, suivis d'une couche *Dense* pour effectuer la classification.

La première couche est composée de 64 neurones et prend en entré l'ensemble des données d'entrainement (données de consommation du fichier *InputTrain.csv* et labels correspondant pour chaque appareil, à chaque « Time step »). Cette première couche réalise un ensemble de calcul complexe et renvoie ses résultats à la deuxième couche. La deuxième couche est une couche *Dense* composé de 5 neurones (un pour chaque appareil à classifier). Elle prend en entrée les résultats renvoyés pas la couche *SimpleRNN* et renvoie la probabilité (entre 0 et 1) pour chaque « Time step » qu'un appareil soit allumé ou éteint.

Cette deuxième approche nous a permis d'obtenir le score le plus élevé sur Kaggle : 0.10712

Ci-dessous notre pseudo code complet qui explique chaque étape de notre algorithme :

Importer les bibliothèques nécessaires: pandas, matplotlib, seaborn, numpy, tensorflow, et MinMaxScaler de sklearn.preprocessing.

Charger les données d'entraînement et de test depuis les fichiers CSV.

Normaliser les données d'entraînement à l'aide de MinMaxScaler.

Supprimer les colonnes 'Index' et 'House_id' des données d'entraînement et des cinq états des appareils électroménagers.

Fusionner les cinq états des appareils électroménagers en un seul DataFrame.

Normaliser les états des appareils électroménagers à l'aide de MinMaxScaler.

Préparer les données d'entrée pour le modèle RNN.

Définir le modèle RNN en utilisant Sequential de TensorFlow et les couches SimpleRNN et Dense.

Compiler le modèle avec l'optimiseur 'adam' et la fonction de perte 'binary_crossentropy'.

Entraîner le modèle avec les données d'entraînement pour un certain nombre d'epochs et de batch_size.

Évaluer le modèle sur les données d'entraînement pour obtenir la précision.

Prédire les états des appareils électroménagers pour les données de test à l'aide du modèle entraîné.

Copier les données prédites dans un nouveau dataset avec la forme (2488, 2160, 5).

Créer un DataFrame pandas à partir du nouveau dataset.

Enregistrer le DataFrame dans un fichier CSV nommé 'TestSample.csv'.

IV. Expériences

Données du STEP A →

Comme expliqué précédemment, les données que nous avons utilisées afin d'entrainer nos modèles, étaient réparties sur deux fichiers (*InputTrain.csv* et *StepOne_LabelTrain.csv*). Le premier est composé d'un ensemble de données chronologiques représentant la consommation d'énergie globale répartie sur plusieurs séries. Le second est un fichier d'étiquettes associées à ces consommations d'énergie.

Ainsi, les données d'apprentissage que nous avions à notre disposition se composaient :

- De 10421 séries temporelles de consommations d'énergie, chacune ayant une longueur de 2160 (fichier *InputTrain.csv*)
- Ainsi que 10421 vecteurs binaires de 5 éléments chacun (fichier *StepOne_LabelTrain.csv*). Ces vecteurs binaires correspondent aux étiquettes des 5 appareils électriques listés un peu plus tôt et indiquaient si chacun d'entre eux avait été allumé au moins une fois au cours de chaque série.

Enfin, les données de test sur lesquelles nous avons réalisé la classification étaient présentent dans le fichier *InputTest.csv*. Elles correspondaient en 2488 séries chronologiques de consommations d'une longueur de 2160, pour lesquelles aucune étiquette n'a été fournie.

Les prévisions réalisées par notre modèle ont été stockées dans un DataFrame appelé y_pred, qui est indexé avec les numéros de ligne de y_pred. Nous avons rajouté une colonne supplémentaire "Index" au début de ce DataFrame, puis l'avons remplie avec les numéros de ligne de y_pred. Nous avons ensuite renommé les colonnes pour les faire correspondre aux noms des appareils électroménagers ("Washing Machine", "Dishwasher", "Tumble Dryer", "Microwave" et "Kettle"). Tout cela, afin de

faire coïncider la mise en forme de notre dataframe avec celle demandé sur kaggle. Enfin, ce DataFrame a été exporté sous forme de fichier CSV appelé "Sample_LabelTest.csv".

Données du STEP B →

Les données utilisées pour entrainer nos modèles pour le problème du Step B, contrairement à ceux du Step A, sont réparties sur 6 fichiers. Le premier était le même fichier *inputTrain.csv* présenté un peu plus haut. Les 5 autres fichiers représentent les étiquettes associées à chaque « Time step » de chaque série chronologique de consommation électrique.

Ainsi, les données d'apprentissage se composent de :

- 10421 séries temporelles de consommations d'énergie, chacune ayant une longueur de 2160 (fichier *InputTrain.csv*)
- 10421 vecteurs binaires de 5 éléments chacun pour la **machine à laver**
- 10421 vecteurs binaires de 5 éléments chacun pour le **lave-vaisselle**
- 10421 vecteurs binaires de 5 éléments chacun pour le **sèche-linge**
- 10421 vecteurs binaires de 5 éléments chacun pour le **Micro-ondes**
- 10421 vecteurs binaires de 5 éléments chacun pour la **Bouilloire**

Enfin, les données de test sur lesquelles nous avons réalisé la classification étaient également présentent sur le fichier *InputTest.csv*

Les prévisions réalisées par notre modèle ont été stockées de la même manière que les prédictions du Step A, dans un DataFrame appelé y_pred, qui est indexé avec les numéros de ligne de test_pred. Nous avons rajouté une colonne supplémentaire "Index" au début de ce DataFrame, puis l'avons remplie avec les numéros de ligne de test_pred. Mais contrairement au Step A, notre fichier csv contient 5 374 080 lignes. En effet, nous avons aplatit chaque série chronologique pour chaque appareil en un seul vecteur, cela signifie mettre les séries prévues l'une après l'autre dans un long vecteur (2160 x 2488 = 5374080) pour chaque appareil.

Matériels utilisés pour la réalisation du projet →

La machine utilisée pour réaliser l'ensemble de ces expériences possède les caractéristiques suivantes :

- Processeur Intel Core i7-9750H (6 cœurs)
- 16 Go RAM

De plus, nous avons utilisé Jupyter Notebook pour réaliser notre projet. Il permet l'intégration de code et de texte et facilite la visualisation de données. En outre, il permet l'exécution de code "pas à pas".

V. Conclusion

En conclusion, afin de résoudre le problème du Step A qui consistait à prédire l'utilisation des appareils électriques, différentes approches ont été évaluées. Le modèle XGBoost a été choisi pour sa performance supérieure, avec un score de 0,329 sur Kaggle.

Pour résoudre le problème du Step B, qui était une classification supervisée de séries chronologiques pour déterminer si chaque appareil électrique était allumé ou éteint à chaque instant, deux approches ont été testées. La première approche, qui consistait à entraîner et prédire séparément l'étiquette de chaque appareil pour chaque instant, était lente et peu performante. La deuxième approche a utilisé un réseau de neurones récurrents (RNN), qui s'est avéré être adapté aux séries chronologiques. L'algorithme a été configuré avec une couche SimpleRNN et une couche Dense, permettant d'obtenir le score le plus élevé sur Kaggle : 0.10712

Dans l'ensemble, ce projet a été bénéfique pour nous, car il nous a permis de mettre en pratique les différents outils et approches que nous avons étudiés en cours et pendant les TP. Nous avons maintenant une vision plus détaillée de la science des données et de son utilité pour les entreprises dans la réalisation de prévisions et de l'analyse d'un grand nombre de données. Cependant, le seul inconvénient que nous avons rencontré était lié à la taille des données utilisées. L'exécution des scripts prenait beaucoup de temps, nous empêchant de tester un grand nombre de classificateurs et ainsi d'améliorer notre score sur Kaggle.