



---

# Détection d'activité non intrusive à partir de la courbe de charge

---

UE INFMDI722  
Projet Fil Rouge  
Entreprise : Baalbek

Julien Lair  
Hugo Michel  
Paulin Kimpa  
Pierre Dal Bianco  
Pierrick Leroy

Under supervision of  
Stephan Clémenton  
Ekhine Irurozki Arrieta

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Contexte et enjeux . . . . .	2
1.1.1	Contexte . . . . .	2
1.1.2	Motivations . . . . .	2
1.1.3	Enjeux . . . . .	3
1.1.4	NILM . . . . .	4
1.1.5	Architecture d'un système NILM . . . . .	7
1.2	État de l'art . . . . .	8
1.3	Objectifs poursuivis . . . . .	8
1.4	Moyens nécessaires . . . . .	13
1.5	Planning . . . . .	14
1.5.1	Planning long terme . . . . .	14
1.5.2	Planning court terme . . . . .	14
<b>2</b>	<b>Projet</b>	<b>15</b>
2.1	Récupération et analyse des données . . . . .	15
2.2	Labellisation . . . . .	17
2.2.1	Méthodologie utilisée . . . . .	17
2.2.2	Discussion des résultats . . . . .	20
2.3	Modélisation . . . . .	21
2.3.1	Approche 1 : Algorithmes de classification supervisés . . . . .	21
2.3.2	Approche 2 : Time2Vec . . . . .	39
2.3.3	Approche 3 : Auto-Encodeur + Classifieur pour la détection d'anomalies . . . . .	49
2.3.4	Approche 4 : <i>Auto-Encodeur Convolutionnel</i> pour la détection d'anomalies . . . . .	56
2.4	Test des approches sur un autre jeu de données : UK-DALE . . . . .	69
2.4.1	Récupération et analyse des données . . . . .	69
2.4.2	Labellisation . . . . .	70
2.4.3	Résultats de l'approche 1 : Classifieur . . . . .	72
2.4.4	Résultats de l'approche 2 : Time2Vec . . . . .	73
2.4.5	Résultats de l'approche 4 : Auto Encodeur Convolutionnel . . . . .	74
2.5	Discussion des résultats . . . . .	74
2.6	Structure des modèles . . . . .	76

# Chapitre 1

## Introduction

### 1.1 Contexte et enjeux

#### 1.1.1 Contexte

Le développement du réseau intelligent, en particulier le déploiement à grande échelle des compteurs intelligents, offre la possibilité de développer des applications d'analyse de données et d'aider les clients à accroître leur efficacité énergétique. Pour les particuliers, la mise en œuvre de compteur intelligent fournit un accès sécurisé aux données de consommation d'énergie de leurs propres maisons ou appartements, qui peuvent également être révélées à un tiers de confiance avec l'autorisation de l'utilisateur. Un système de gestion de l'énergie appliqué aux charges dans les bâtiments permet aux clients d'ajuster leur consommation d'énergie en fonction d'un niveau de confort attendu, des variations des prix de l'énergie et parfois des impacts environnementaux (par exemple les émissions d'équivalent CO<sub>2</sub>). De telles stratégies de gestion de la demande nécessitent une évaluation précise de la quantité d'énergie qui peut être contrôlée. Par conséquent, l'identification de l'utilisation de chaque appareil est l'une des questions centrales de la gestion énergétique des bâtiments intelligents. Du point de vue des entreprises de services publics, la sensibilisation et la participation accrues des clients facilitent également le déploiement des compteurs intelligents et l'adoption de politiques telles que l'option de tarification en fonction de l'heure de consommation. Mais encore, l'identification de la charge peut également jouer un rôle important dans la prédiction future de l'utilisation d'appareils particuliers lorsque le processus de collecte des données historiques est rendu aussi peu intrusif que possible. Par conséquent, le développement d'outils d'analyse des données est la première étape vers une participation active des utilisateurs au développement futur des réseaux intelligents et des services associés. Sans aucun doute, l'analyse des données des compteurs intelligents offre de nouvelles possibilités aux consommateurs et aux entreprises de services publics pour la mise en place d'un réseau intelligent. Toutefois, à l'heure actuelle, les compteurs d'électricité intelligents ne fournissent que des données sur l'ensemble du logement. Cela signifie qu'il est nécessaire de séparer et d'identifier à partir de la charge totale la consommation électrique de chaque appareil. Cette méthode s'appelle la désagrégation de l'énergie et requiert le développement d'algorithmes pour identifier la consommation électrique de chaque appareil électronique. Afin d'éviter cela les appareils d'une maison pourraient être surveillés directement, mais au prix de la fabrication et de l'installation de nombreux nouveaux appareils dans les maisons, de la gêne pour l'utilisateur et du fait que de nouveaux capteurs doivent être installés pour tout nouvel appareil.

#### 1.1.2 Motivations

Le développement des compteurs intelligents tels que Linky, qui équipe désormais près de 35 millions de foyers en France (chiffres Enedis 2021), ouvre les champs des possibilités dans le domaine de la

détection d'activité. En effet, malgré le grand nombre de scénarios d'application potentiels, la détection de l'occupation des bâtiments reste un processus lourd, sujets aux erreurs et coûteux. L'occupation est généralement détectée à l'aide de dispositifs spécialisés tels que des capteurs infrarouges passifs, des interrupteurs magnétiques ou des caméras. Ces capteurs doivent être achetés, installés, calibrés, alimentés et entretenus. Cela pose un certain nombre de contraintes critiques, en particulier dans les environnements domestiques. Tout d'abord, le coût global de l'infrastructure de détection d'occupation doit rester faible. De plus, les capteurs fonctionnant sur batterie sont souvent utilisés pour éviter le déploiement des câbles d'alimentation. La disponibilité et la fiabilité des capteurs peuvent donc être affectées par des batteries épuisées en attendant d'être remplacées. En outre, dans un environnement domestique, l'un des résidents (souvent inexpérimenté sur le plan technique) joue le rôle d'administrateur du bâtiment qui installe et entretient le système. Les installations défectueuses et le manque de maintenance sont des conséquences fréquentes. L'ensemble de ces contraintes peut rendre des systèmes de détection d'occupation peu fiables et induire des comportements défectueux dans les systèmes domotiques qui en dépendent. Cela peut à son tour causer des désagréments aux résidents et entraver leur acceptation des systèmes.

Par ailleurs, au-delà des contraintes posées par ces types de dispositifs (lourde maintenance, coût élevé, installation défectueuse, ...) au sein d'un environnement domestique, il s'avère que ces derniers sont des solutions intrusives pas forcément acceptées par le public. Cela constitue alors la principale motivation de l'introduction de nouvelles technologies non intrusives à partir des courbes de charge de consommation électrique. Les méthodes non intrusives offrent une alternative intéressante, avec un coût d'installation très réduit. Les compteurs intelligents sont l'une des unités fondamentales des réseaux intelligents, car de nombreuses autres applications dépendent de la disponibilité d'informations à granularité fine sur la consommation et la production d'énergie. En effet, de nombreuses études ont montré que les mesures sur la consommation d'énergie à une granularité plus fine favorisent la sensibilisation au domaine de la consommation d'énergie et aident l'utilisateur à identifier les opportunités d'économie d'énergie. Pour ce faire, la communauté scientifique se mobilise pour proposer des algorithmes de désagrégation énergétique, qui génèrent automatiquement des informations détaillées sur la consommation d'énergie des appareils individuels. Ces informations peuvent être utilisées directement comme une mesure du retour d'expérience. Elles peuvent également aider les experts à analyser les modèles de consommation d'énergie des maisons résidentielles.

Ces dernières années une attention particulière a été accordée aux recherches visant à déduire l'activité des ménages par l'analyse de la consommation de leurs appareils. Ces systèmes sont connus sous le nom de la surveillance de la charge des appareils non intrusifs (NILM).

### 1.1.3 Enjeux

Le sujet de la détection d'activité est un sujet porté par de nombreux acteurs économiques et qui peut être source de nombreux services aux usagers. Notamment,

- Sécurisation des logements notamment les logements secondaires ou les logements mis en location temporaire ;
- Prolonger l'autonomie des senior en leur permettant un maintien à domicile, en alertant les proches ou le personnel médical en cas d'absence anormal d'activité ;
- Détection de l'heure du retour de l'école des enfants

Les solutions actuelles sont des solutions intrusives, pas forcément acceptées ni simples à installer : par exemple via l'installation de différents types de capteurs qui vont monitorer la présence des habitants etc...

La problématique NILM a longtemps été étudiée par des chercheurs. La plupart des approches sont basées sur le traitement du signal à un taux d'échantillonnage élevé (1 Hz typiquement) pour évaluer la signature de la charge de l'appareil et ensuite utiliser des techniques de reconnaissance des formes pour l'identification à partir de classificateurs préalablement formés. Cela nécessite l'installation d'un capteur pour chaque appareil dans la maison et est donc naturellement limité par cet important système de surveillance de la charge. D'un point de vue social, un obstacle majeur dans la recherche de NILM est le respect de la vie privée de l'utilisateur, car l'utilisation de l'appareil peut être liée au comportement de l'utilisateur. Par exemple, l'heure à laquelle les lumières sont éteintes peut être considérée comme l'heure de sommeil de l'habitant. Enfin, il reste à développer une méthode non intrusive qui fonctionne pour tous les appareils de la maison.

Tout l'enjeu de ce projet est donc travailler sur les technologies non intrusives de détection d'activité à partir des courbes de charge de consommation électrique. Plus concrètement, il s'agira de construire des algorithmes de Machine Learning qui vont permettre de faire une classification binaire prédisant l'activité du logement, avec la contrainte de pouvoir s'adapter à différents types de logements pour lesquels on n'a pas de données labellisées (unsupervised learning).



FIGURE 1.1 – Activité dans un logement

#### 1.1.4 NILM

Le champ de recherche dédiée à la désagrégation de l'énergie porte le nom de « Non Intrusive Load Monitoring » (NILM). Ces technologies convertissent une série chronologique de données de consommation d'énergie (courbe de charge) en une courbe de charge par appareil. En décomposant un ensemble de relevés de consommation d'énergie globale en ses parties constitutives consommées par des appareils individuels, les algorithmes de désagrégation de l'énergie permettent à un client résidentiel d'obtenir des informations détaillées sur la consommation d'énergie de tous les appareils de la propriété et donc de mieux comprendre la facture d'électricité. Un système de désagrégation de l'énergie est composé à la fois d'algorithmes de désagrégation et du matériel nécessaire à la collecte des données de consommations énergétiques comme un compteur intelligent. Bien que certains travaux existants proposent le

déploiement d'équipements spécialisés et parfois sophistiqués, la réduction des niveaux d'invasivité de ces systèmes est considérée comme l'étape cruciale vers des solutions pratiques de NILM.

Les méthodes de séparation des charges peuvent être classées en fonction du degré d'intrusion du processus de formation et de la nature de l'algorithme de classification (basé sur des événements ou non). L'algorithme basé sur les événements tente de détecter les transitions marche/arrêt, tandis que les méthodes non basées sur les événements tentent de détecter si un appareil est allumé pendant toute la durée de l'échantillonnage. Les figures ci-dessous présentent les deux cas de figure, l'un basé sur les événements (figure 1.2) et l'autre non basé sur les événements (figure 1.3).

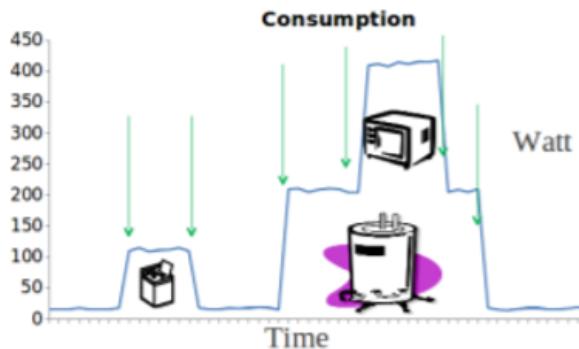


FIGURE 1.2 – Event based load monitoring method

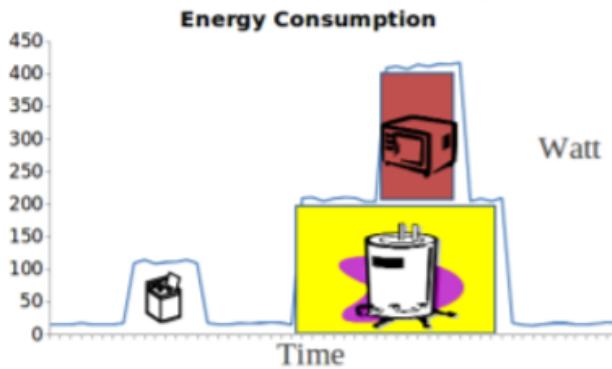


FIGURE 1.3 – Non event based load monitoring method

L'idée de la reconnaissance d'activité a été étudiée en premier lieu dans le domaine de la santé pour détecter les situations d'urgence en surveillant les activités et les mouvements quotidiens des patients. Mais ces techniques de reconnaissance reposent souvent sur des capteurs supplémentaires et sont donc intrusives et peuvent être coûteuses. Le NILM comprend un ensemble de techniques pour la surveillance non intrusive de la charge. Son but est de déterminer les changements de tension et de courant entrant dans une maison et de déduire quels appareils sont utilisés dans la maison.

Les méthodes NILM sont rangées dans deux catégories :

- **Activity Detection** : s'appuient sur la détection d'événements, par exemple, l'allumage et l'arrêt d'appareils et le changement de mode ou d'état de fonctionnement des appareils. Cela suppose alors que la variation de la charge est causée par un seul appareil or cela n'est pas toujours le cas en pratique.

— **Activity Modeling** : La charge électrique agrégés est un mélange de signaux de charge inconnus associés à plusieurs appareils électriques. Le but est de récupérer la consommation détaillée de chaque appareil à partir de la courbe de charge agrégée. Cependant, les performances du modèle dépendent fortement de la fréquence d'échantillonnage et les compteurs intelligents ne supportent souvent pas d'échantillonnage et de stockage des données à haute fréquence (relevé du compteur électrique toutes les 10s par exemple). Originellement, des méthodes de clustering (*Kwac et al - Lifestyle segmentation based on energy consumption data*) existent et sont souvent utilisées pour segmenter les styles de vie des consommateurs sur la base de leurs données de charges. Mais ces styles de vies ne permettent pas de modéliser les activités d'un consommateur au cours du temps. En fait ces styles de vies ne permettent pas de différencier les activités des clients. La théorie des pratiques sociales suggère que les activités des consommateurs devraient être traitées de manière holistique afin de refléter les patterns (i.e les routines quotidiennes). Dans ce cadre, des chaînes de Markov peuvent être utilisées pour détecter des séquences d'activité au travers de la vie quotidienne d'un consommateur. Ces séquences d'activité peuvent être construites comme des transitions d'états (i.e activité) de type chaîne de Markov.

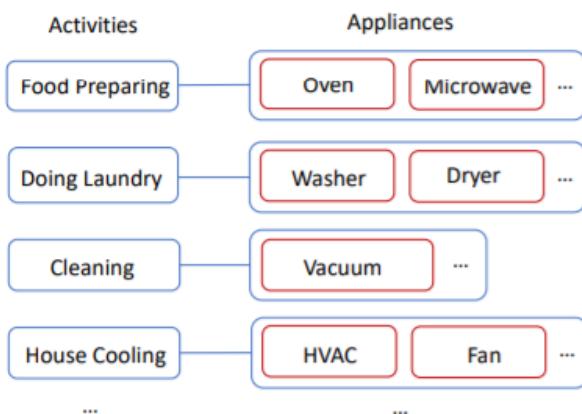


FIGURE 1.4 – L'activité liée aux appareils



FIGURE 1.5 – Modélisation des activités

Du point de vue des sciences du comportement, la compréhension du calendrier des activités du consommateur (activity modeling) facilite l'étude des pratiques sociales des consommateurs en termes d'ordonnancement et de chevauchement dans le temps. Du point de vue de l'ingénierie, la compréhension du calendrier des activités peut contribuer au déplacement de la charge et à la gestion de la demande puis concevoir des programmes de gestion de la demande appropriés. Dans cet exemple, nous considérons cinq activités (dormir, se doucher, préparer la nourriture, faire la lessive et laver la vaisselle). Chaque flèche indique une transition d'une activité à une autre et est associée à une probabilité montrant la

vraisemblance de la transition.

D'un point de vue de la recherche, les méthodologies employées pour résoudre des problématiques NILM englobent plusieurs domaines. Une majorité des premières recherches se sont concentrées sur ce problème du point de vue du traitement du signal. L'accent a été mis sur l'identification des différentes signatures d'appareils qui distinguent un appareil d'un autre en l'analysant à l'aide d'outils mathématiques.

Des recherches ultérieures ont également considéré le problème comme une tâche de séparation aveugle des sources et ont proposé des techniques pertinentes dans cette direction [Kolter 2010]. Ensuite, les problématiques NILM sont considérés comme un problème de classification temporelle pour l'identification des appareils électroménagers basée sur un processus d'apprentissage multi-classes utilisant un fenêtrage temporel où la seule entrée est le relevé d'énergie horodaté du compteur électrique.

Plus récemment, l'avènement du Deep Learning ouvre les champs des possibilités pour la résolution des problématiques NILM. Ces techniques et notamment les réseaux de neurones LSTM sont utilisés pour la détection d'activité à partir de la courbe de charge.

### 1.1.5 Architecture d'un système NILM

L'architecture du processus d'identification de la courbe de charge de chaque appareil électronique à partir de la courbe de charge totale du domicile se présente comme suit :

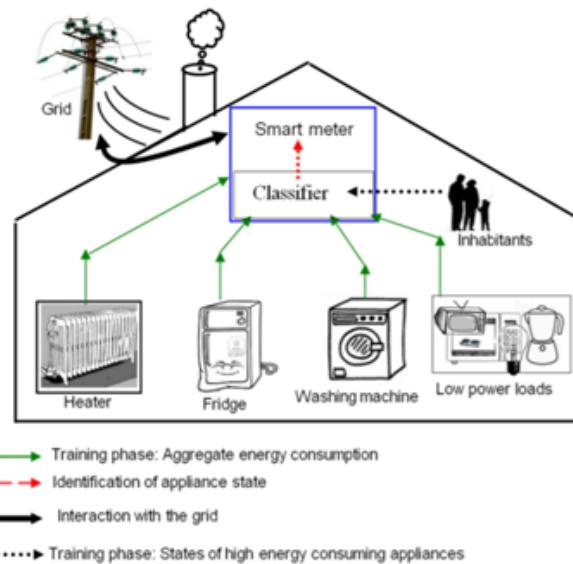


FIGURE 1.6 – Architecture du classifieur : Centraliser les informations d'un foyer intelligent

Dans une architecture NILM, les interactions entre les habitants, les consommations électriques de chaque appareil et le réseau électriques sont centralisées en un point de convergence. Ce point de convergence est la compteur intelligent. Un classifieur est directement intégré dans le compteur intelligent. Ce dernier est entraîné avec les données collectées pour chaque appareil par le compteur intelligent afin de prédire s'il y a activité ou non au sein d'un foyer à partir de la courbe de charge agrégée.

## 1.2 État de l'art

Durant la rédaction de l'état de l'art nous avons abordé différentes méthodologies d'approche sur le sujet de la problématique NILM. Nous avons vu deux grandes approches. Premièrement une approche que l'on peut nommer comme une approche traditionnelle qui se base sur l'apprentissage statique avec l'utilisation de modèles comme le kNN et le SVM, et une approche nouvelle qui se base sur l'utilisation de réseaux de neurones profonds (auto-encodeur, LSTM).

Concernant les modèles des approches traditionnelles, la littérature scientifique montre que ces derniers fournissent tous des performances assez similaires. Cependant, ces modèles étant des modèles supervisés, ils exigent que la courbe de charge totale soit labellisées. Avant d'appliquer ces modèles il conviendra alors de choisir la bonne stratégie à adopter pour labelliser nos données. Dans le cadre de notre projet fil rouge nous souhaitons tester le modèles tels que kNN et SVM sur notre jeu de données ainsi que des modèles d'ensemble learning notamment les Random Forest. Concernant les approches nouvelles, la littérature scientifique nous apprend que les approches à base de réseaux de neurones surclasse les approches traditionnelles en termes de performances. En effet, les modèles basés sur des auto-encodeurs variationnels combinés à un modèle de classification comme Random Forest, SVM ou KNN ainsi que les modèles utilisant des réseaux de neurones de type LSTM sont plus performants que les approches traditionnelles. En revanche, nous pensons ne pas avoir suffisamment de données à notre disposition afin de mettre en point un modèle à base de réseaux de neurones. En effet, ces modèles ont l'avantage de prendre en compte les effets de saisonnalité (période été/hiver, période de vacances induisant une consommation électrique plus importante que d'habitude) que l'on pourrait retrouver au sein d'une année entière. Or nous à ce jour, nous avons deux jeux de données chacun couvrant au maximum 72 jours de l'année.

Par ailleurs, les métriques d'évaluations mentionnées dans les papiers pour évaluer les modèles reposent pour la plupart sur le calcul du F1-score. Dans la suite de nos travaux nous allons étudier les différentes métriques pour voir lesquelles seront les plus adaptées à notre problématique spécifique. Enfin, concernant les méthodes de séparation de signal, nous exprimons une certaine réserve quant à leur applicabilité dans notre étude de cas. L'application d'outil ICA pour résoudre un problème NILM semble plus complexe, en raison du nombre sources indépendantes à décomposer. Cependant, des hypothèses plus fortes pourraient être faites, notamment sur la forme des signaux à retrouver (échelons on/off par exemple).

L'intégralité de l'état de l'art que nous avons réalisé sur le sujet de la détection d'activité est disponible [ici](#).

## 1.3 Objectifs poursuivis

Comme expliqué dans les parties précédentes la finalité du projet est de pouvoir détecter une activité à partir de la mesure instantanée de la puissance électrique d'un logement provenant d'un compteur intelligent Linky. Ceci peut être utile dans le cadre de nombreux services aux usagers.

Le but est donc de pouvoir faire une classification binaire (activité ou pas) à partir de la mesure de la consommation de puissance électrique d'un logement. Cette classification devra être la plus précise possible tout en s'adaptant à tout type de logement. À la vue des applications finales il sera important de ne pas avoir un taux de faux positifs trop important, c'est-à-dire éviter de prédire de l'activité alors qu'il n'y en n'a pas. Surtout dans le cadre de la détection d'activité pour un senior, il sera plus prudent

d'accepter un taux de faux négatifs élevé (ne pas prédire de l'activité alors qu'il y en a) ce qui aurait comme effet potentiel de déclencher une fausse alerte, plutôt que de ne pas déclencher d'alerte lorsque cela aurait été nécessaire. Cette philosophie sera importante pour le choix de la métrique de mesure de performance de nos algorithmes.

## Métriques d'évaluation

Ces contraintes nous permettent déjà d'orienter nos choix pour la métrique d'évaluation de nos classifiants. En plus de l'accuracy, il sera important de calculer et maîtriser le taux de rappels. Une métrique intéressante est le score  $F_\beta$  définie par :

$$F_\beta = (1 + \beta^2) \frac{\text{précision} \cdot \text{rappel}}{\beta^2 \cdot \text{précision} + \text{rappel}}$$

On peut également écrire ce score sous la forme suivante :

$$F_\beta = \frac{(1 + \beta^2) \cdot \text{vrai positif}}{(1 + \beta^2) \cdot \text{vrai positif} + \beta^2 \cdot \text{faux négatif} + \text{faux positifs}}$$

En prenant  $\beta = 0,5$  on peut ainsi donner moins de poids aux faux négatifs et donc plus de poids aux faux positifs (ceux-ci auront plus d'impact sur le score final).

Dans la littérature pour des classifications binaires de ce type à partir de courbes de charge, on observe souvent un score accuracy autour des 80%. Il sera donc intéressant de garder cet ordre de grandeur en tête lors de nos tests afin de voir si nous parvenons à reproduire ces résultats, voire à aller plus loin. Un challenge auquel on peut s'attendre sera de pouvoir obtenir des modèles suffisamment généralistes pour pouvoir être appliqués sur différents logements. Il sera notamment intéressant de pouvoir appliquer un modèle entraîné à partir des données d'un logement donné sur un autre logement tout en conservant un score acceptable.

## Autre stratégie d'évaluation

Les premières approches pour évaluer la qualité des prédictions d'un modèle reposaient sur la comparaison ligne à ligne de la classification de l'activité. Pour chaque instant, c'est à dire chaque seconde du dataframe, on se trouve alors dans l'un des 4 cas suivants :

1. activité réelle et prédiction d'activité : vrai positif
2. activité réelle et prédiction d'inactivité : faux négatif
3. inactivité réelle et prédiction d'inactivité : vrai négatif
4. inactivité réelle et prédiction d'activité : faux positif

Et à partir de ces quatre cas on peut dériver les métriques de performance usuelles : précision, recall ; et afficher la matrice de confusion. Cette approche est un bon point de départ mais l'on peut se poser des questions sur l'adéquation entre cette métrique et la tâche qui nous incombe :

- De changements erronés bref et à haute fréquence ne sont pas vraiment pénalisés par cette approche. Théoriquement il serait possible d'indiquer de l'activité au moins une seconde à chaque minute et tout de même avoir de très bonne performances.
- Un déphasage léger et constant est sanctionné, alors que ce n'est pas un problème en réalité.

- Enfin, la vision suggérée par cette première approche est que les échantillons peuvent être traités de manière indépendante, ce qui ne fait pas forcément sens si on considère le temps comme une succession d'instants.

Nous avons donc défini une nouvelle métrique de performance pour essayer de mieux refléter l'esprit du cas d'usage : peut importe qu'il y ait un peu de déphasage, l'important est que les périodes d'activité et d'inactivité s'alignent au maximum. Cette métrique est inspirée des mesures de performance en computer vision pour les tâches d'object detection et repose sur le calcul d'IoU (Intersection over Union)

Le premier changement est de passer d'une vision de Timestamps indépendants à une vision considérant les périodes, ce qui implique de retravailler les données pour obtenir des périodes d'activité réelle et prédictive.

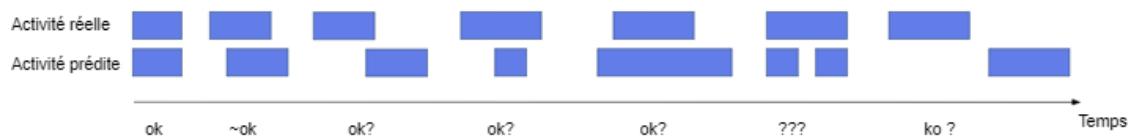


FIGURE 1.7 – Activité réelle vs activité prédictive (illustration)

Sur la figure 1.7, on observe différents cas possibles avec un plus ou moins fort chevauchement entre activité réelle et activité prédictive. L'enjeu est de déterminer de manière automatique les cas où l'on juge l'adéquation satisfaisante et ceux où elle est insatisfaisante.

On se donne alors deux métriques de performance :

1. mean Average Precision (mAP) : pour chaque période d'activité prédictive, quelle est le taux d'adéquation (y a-t-il une période d'activité réelle correspondante ou non)
2. mean Average Recall (mAR) : pour chaque période d'activité réelle, quelle est le taux de d'adéquation (y a-t-il une période d'activité prédictive correspondante ou non)

A noter que d'après les exigences de notre problème, nous devons en priorité satisfaire un niveau élevé sur le mAP. L'idée de la métrique de performance IoU est de calculer pour chaque période prédictive (cas du mAP) l'adéquation avec la/les périodes réelles les plus proches en terme d'IoU et de décider selon un seuil si la période prédictive est valide ou non. On fera ensuite varier le seuil, et on procèdera de même pour chaque période réelle ensuite (cas du mAR).

---

#### Algorithm 1 Principe d'évaluation : mAP

---

**Require:** dataframe des périodes réelles et prédictives,  $\tau$  intervalle entre chaque seuil de mesure

**return** scores du mAP pour différents seuils séparés par  $\tau$

$t \leftarrow 0$

**while**  $N \neq 1$  **do**

$t \leftarrow t + \tau$

**for** chaque période d'activité prédictive **do**

Calculer la meilleure IoU possible vis à vis des périodes d'activité réelle

Statuer sur la validité de la période prédictive au seuil  $t$

**end for**

**end while**

---

$$mAP(\tau) = \frac{valid_{pred}(\tau)}{total_{pred}} \quad mAR(\tau) = \frac{valid_{true}(\tau)}{total_{true}}$$

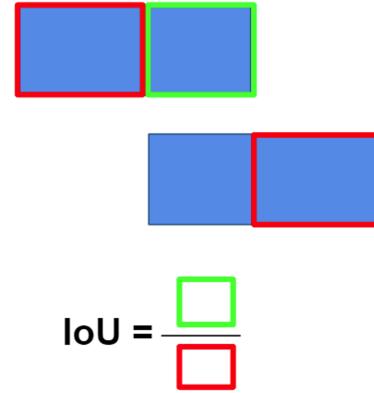


FIGURE 1.8 – Formule d'évaluation (gauche) et principe de l'IoU 1D (droite)

Finalement, on obtient le type de panel d'évaluation présenté sur la figure 1.9 avec :

1. Activité réelle (haut) vs activité prédite (bas)
2. Superposition activité réelle/prédite et association TP / FP / TN / FN = matrice de confusion
3. mAR et mAP pour différents seuils de la métrique de performance IoU

Cette méthode d'évaluation est utilisable de manière autonome, et ne dépend pas du modèle considéré.

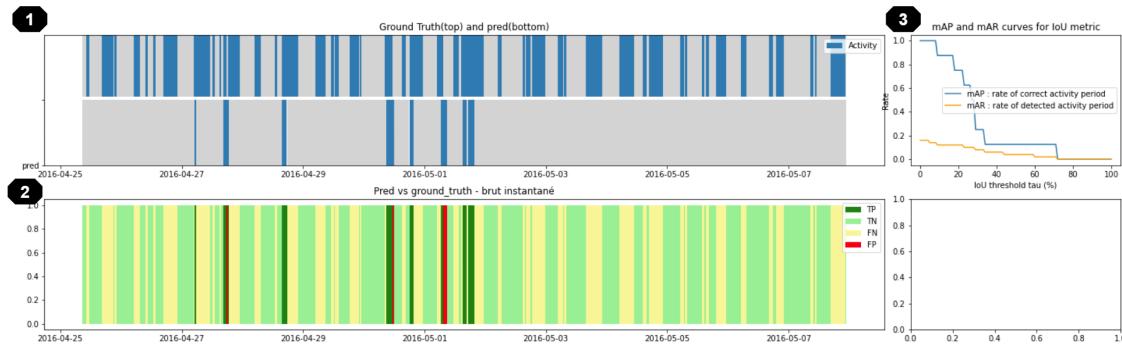


FIGURE 1.9 – Exemple d'évaluation

Une difficulté liée au jeu de données de départ est que ce dernier ne possède pas d'étiquettes pour l'activité. Pour la suite de ce projet nous allons devoir soit repérer sur des algorithmes non supervisés, ce qui serait limitant en termes de résultats et qui n'est pas forcément traité dans la littérature sur le sujet, soit appliquer un premier algorithme de d'étiquetage, afin de créer des étiquettes pour pouvoir ensuite appliquer des algorithmes d'apprentissage supervisé. Pour générer des étiquettes, nous pouvons utiliser des règles métier, si la puissance de certains équipements dépasse un certain seuil par exemple nous pouvons estimer qu'il y a de l'activité. Il conviendra d'appliquer une approche conservatrice à ce stade également pour éviter d'étiqueter de l'activité alors qu'il n'y en n'a pas. À la différence des données finales où nous ne disposerons que de la courbe de charge totale (agrégée), ce jeu de données présente l'avantage d'avoir plusieurs courbes de charges (une par disjoncteur). Nous reviendrons sur le jeu de données et ses particularités plus en détails dans le deuxième chapitre.

## La pipeline de traitement des données

Une fois que nous aurons générée les labels, nous pourrons générer un jeu de données sous la même forme que les données que nous souhaitons traiter lors de l'application finale, c'est-à-dire des relevés périodiques de puissance agrégée consommée dans un logement, avec une colonne labels supplémentaire. Ce jeu de données pourra alors servir de base pour générer de nouvelles colonnes features à partir des informations de date et heure (semaine, week-end, jour de la semaine en one hot, heure de la journée, etc.) et à partir de la puissance totale consommée dans le logement (dérivé discrète de la puissance, écart type, moyenne ou encore amplitude glissante sur x minutes / heures / jours, etc.). Nous pourrons alors ainsi générer plusieurs jeux de données à partir des mêmes données initiales, sous différents formats pour pouvoir appliquer différents algorithmes d'apprentissage automatiques. Nous pourrons notamment traiter le jeu de données comme des colonnes features et une colonne label et y appliquer des algorithmes de classification (kNN, SVM, etc.), ou de Deep Learning. Nous pourrons également traiter le jeu de données comme une série temporelle et y appliquer des algorithmes adaptés. L'objectif est de développer une pipeline afin d'automatiser le processus qui permet de passer des données labellisées aux données mises en forme avec toutes les colonnes feature nécessaires, puis gérer automatiquement l'entraînement et l'évaluation de différents modèles (cf fig 1.10).

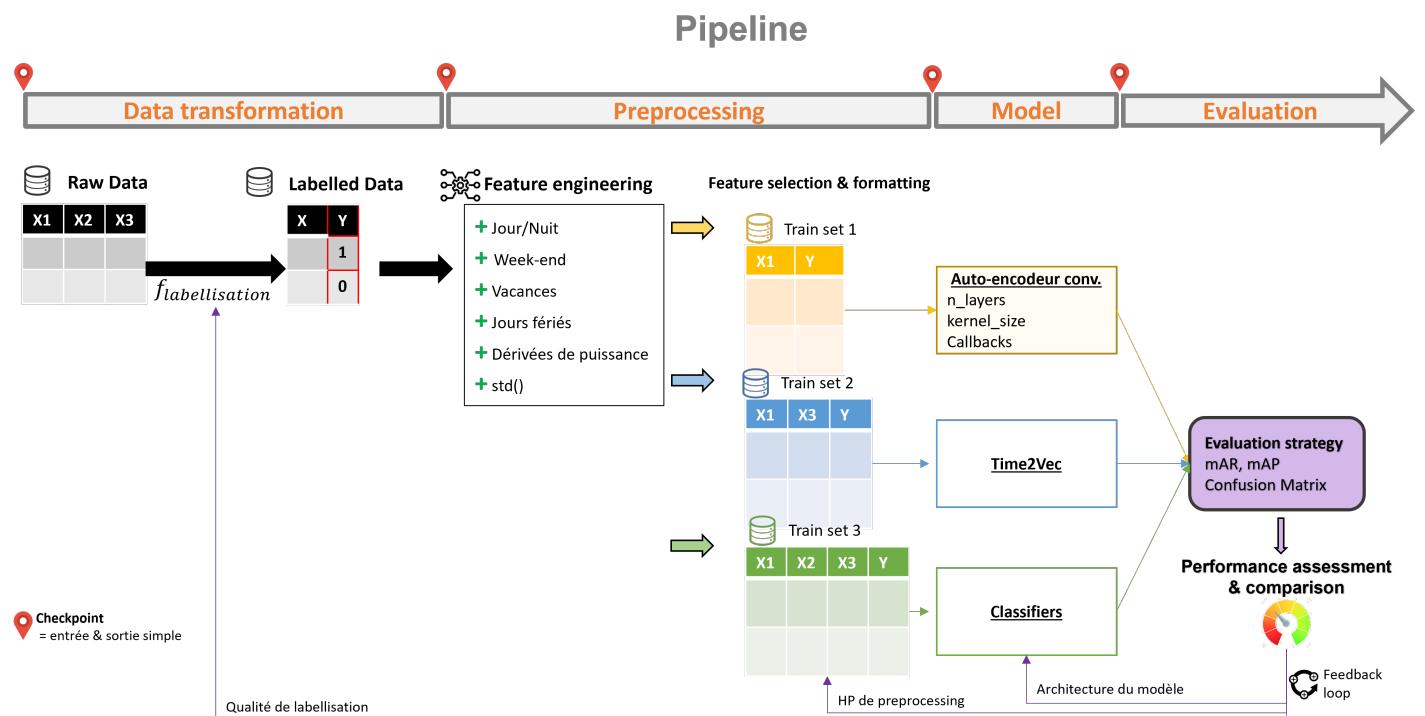


FIGURE 1.10 – Flux des données

## 1.4 Moyens nécessaires

La mise en œuvre de nos objectifs nécessite d'utiliser au mieux les ressources disponibles. L'entreprise partenaire de ce projet, Baalbek Management suggère d'utiliser le jeu donnée issu du papier *RAE : The Rainforest Automation Energy Dataset* (Makonin & Stephen, 2017). Ce jeu de données comprend des relevés de puissances dans deux logements séparés sur une période d'environ 60 jours. Les données sont échantillonnés à 1 Hz, ce qui représente un volume de données important. Il sera donc intéressant d'avoir accès à un calculateur de l'école pour accélérer l'entraînement et l'évaluation de nos modèles. Cela dit, l'ensemble du jeu de données ne pèse pas plus de 280 Mo et ne nécessite donc pas de solutions de stockage particulières. Les modèles peuvent être développés en Python à l'aide des librairies usuelles (Scikit-Learn, PyTorch, etc.) et ne nécessite pas d'autres logiciels spécialisés.

En plus du jeu de données suggéré par l'entreprise il sera intéressant de voir si nous pourrons obtenir d'autres jeux de données, notamment sur des périodes de temps plus longues (pour voir si nous pouvons exploiter des phénomènes de saisonnalités annuelles) et des types de logements plus variés, le but final étant de pour générer un modèle suffisamment généralisable pour être appliqué sur n'importe quel logement.

Concernant l'organisation au sein du groupe, une première décision afin d'accélérer le développement d'une solution a été de découper le groupe en deux équipes, au moins dans un premier temps. Une première équipe composée de 3 personnes s'est occupée de l'état de l'art et l'autre équipe composée de 2 personnes s'est afféré à l'étiquetage des jeux de données (activité ou non), puisque le jeu données initial n'est pas labellisé. Une fois que ces deux tâches initiales ont été réalisée, nous nous sommes répartis les différentes approches pour répondre à la problématique initial et chaque membre du groupe tente de développer un ou plusieurs modèles différents. Des points réguliers sont organisés au sein de l'équipe afin de pouvoir partager nos retours d'expériences, travailler ensemble sur la résolution de certains problèmes et mettre un certain nombre d'éléments (fonctions ou classes Python, méthodes de calculs, etc.) en commun. Notre équipe est de plus encadrée par des Data Scientists qui peuvent aiguiller nos choix lors de nos travaux afin de s'assurer de prendre des choix qui conviendront à l'entreprise. Pour cela, des points bi-mensuels sont organisés afin de présenter les derniers avancements de notre équipe et de pouvoir s'assurer de rester en adéquation avec les besoins de l'entreprise.

Toujours pour faciliter les échanges et la coopération avec l'entreprise mais aussi au sein de l'équipe, un répertoire GitHub a été mis en place, ainsi qu'un Google Drive pour centraliser et garder une trace de toutes les présentations faites. L'entreprise a accès à ces ressources et peut donc facilement suivre l'évolution du projet. Afin de faciliter la coopération au sein de l'équipe, de centraliser les prises de décision et de répartir au mieux les différentes tâches au sein du groupe, nous utilisons l'outil de gestion de projets en ligne Trello. Cette application permet de lister les tâches à faire, en cours, faites, etc., de les répartir entre les membres de l'équipe et d'échanger des informations rapidement et de manière organisée. Enfin un planning pour le travail à réaliser a été mis en place dès le début du projet à l'aide du logiciel TeamGantt afin de pour utiliser au mieux des ressources disponibles en respectant les contraintes de temps du projet.

# 1.5 Planning

## 1.5.1 Planning long terme

Nous avons planifié notre projet en plusieurs phases :

- Découverte : EDA (Exploratory Data Analysis), Récupération des données et état de l'art
- Projet : mise en place, ateliers de travail et itérations
- Capitalisation : packaging du code, documentation et livrables

En termes d'efforts nous avions prévu d'attribuer environ 20% du temps à la phase de découverte, 20% à celle de capitalisation et 60% au projet en tant que tel qui constitue le coeur de notre travail.

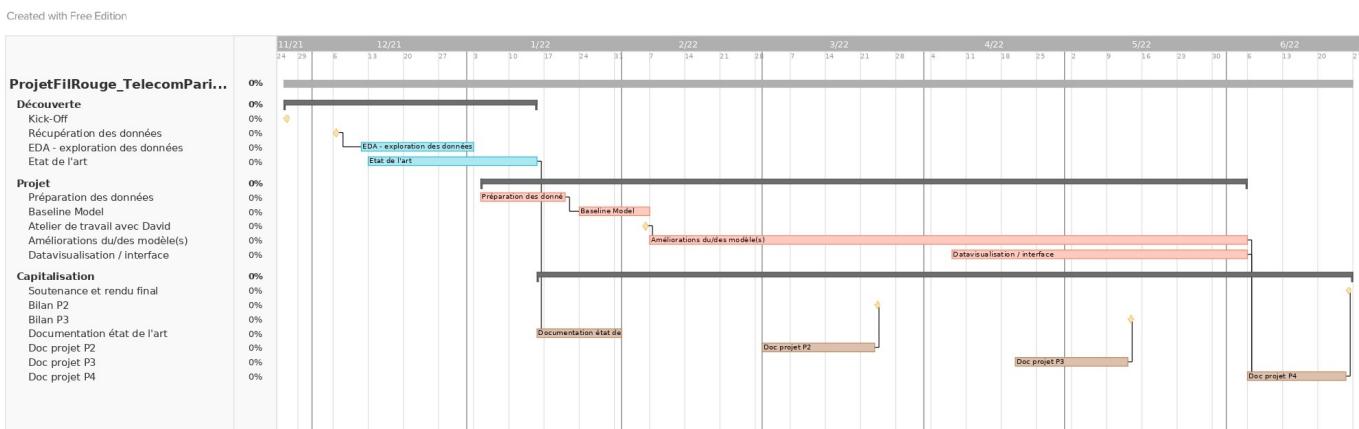


FIGURE 1.11 – Gantt Initial

## 1.5.2 Planning court terme

Pour la planification à court terme, nous nous sommes organisés via Trello en nous inspirant de la méthodologie agile. L'objectif étant de découper le travail en tâches suffisamment simples et de les attribuer aux membres du groupe. Nous avons également mis en place des processus d'idéation (brainstorming, mindmaps) pour faire le lien entre la vision long terme et la vision court terme. Ceci nous permet de convertir un objectif global en objectifs intermédiaire et prioriser les approches.

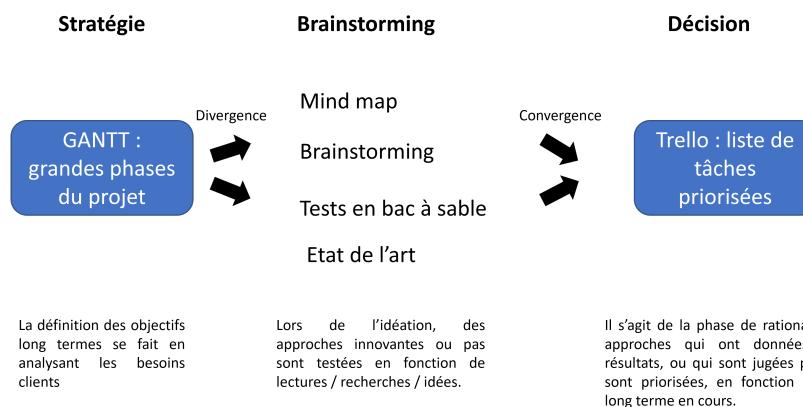


FIGURE 1.12 – Conversion de la stratégie long terme en liste de tâches

# Chapitre 2

## Projet

### 2.1 Récupération et analyse des données

Les données utilisées dans un premier temps sont celles provenant du papier *RAE: The Rainforest Automation Energy Dataset* (Makonin & Stephen, 2017). Le jeu de données comprend des relevés de puissances échantillonnés à 1 Hz sur deux maisons à Vancouver sur une durée de respectivement 72 jours (découpée en deux périodes de 9 puis 53 jours) et 59 jours. La première maison est équipée de 24 sous-compteurs (un par disjoncteur) et est habitée par 3 occupants au niveau supérieur et 1 occupant au niveau inférieur.

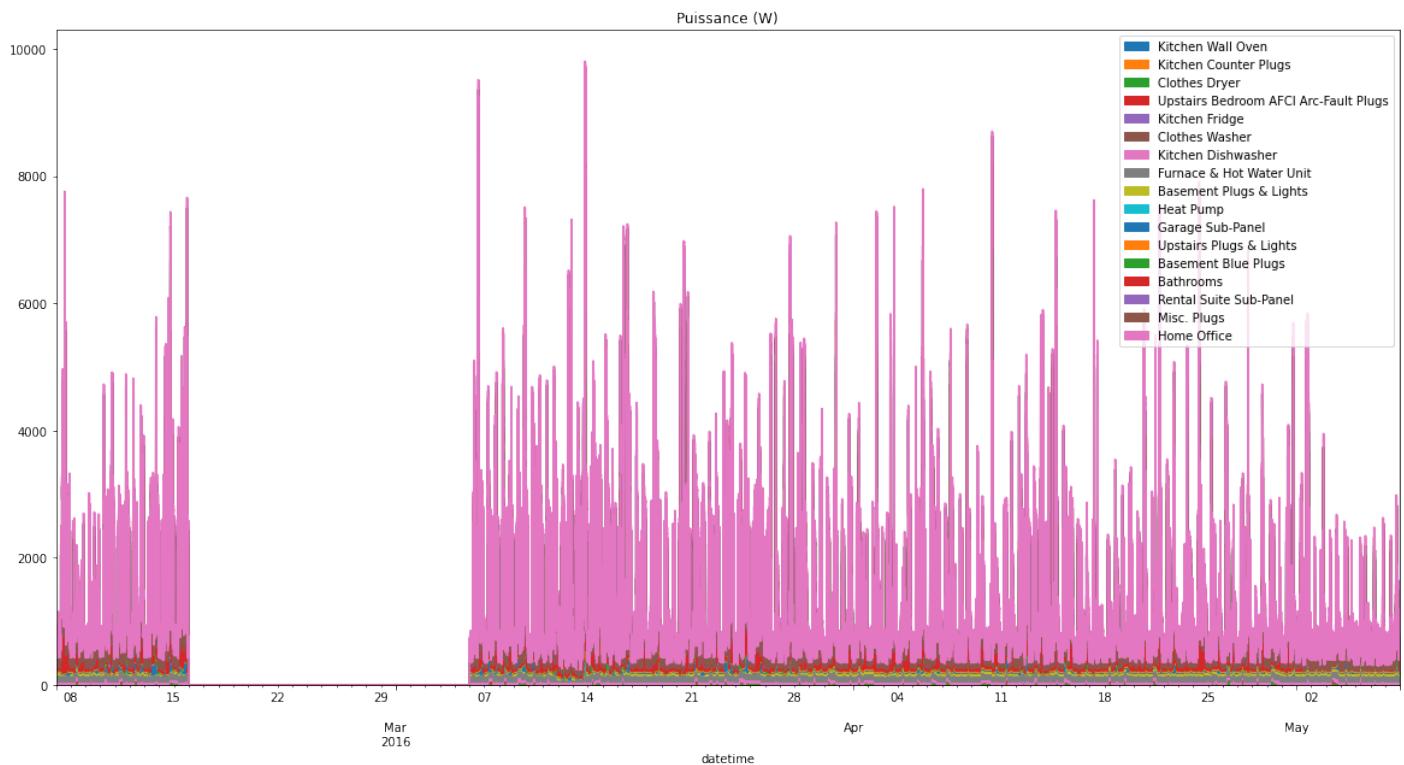


FIGURE 2.1 – Relevés de puissance dans la maison 1

La deuxième maison est équipée de 21 sous-compteurs et est occupée par 3 occupants. Chaque sous-compteur a été étiqueté avec le nom de l'appareil ou emplacement de la prise électrique pour pouvoir facilement identifier la source de consommation. Du fait que les deux maisons soient situées au Canada,

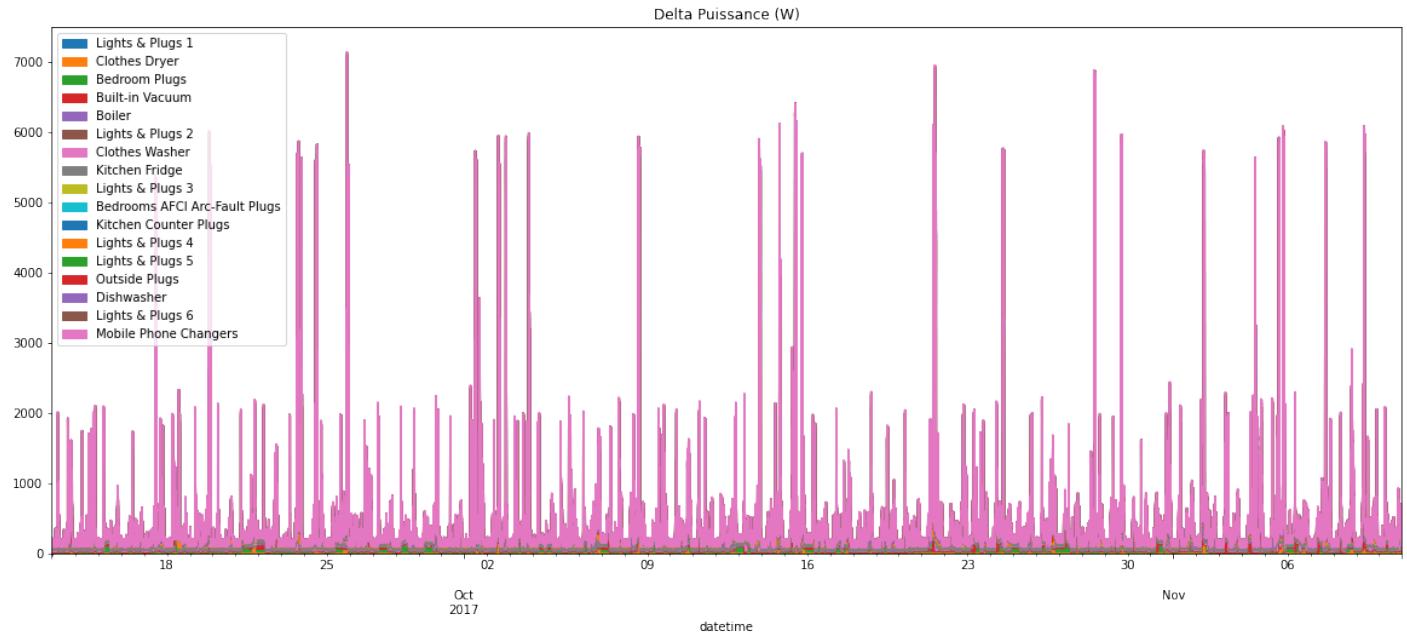


FIGURE 2.2 – Relevés de puissance dans la maison 2

certains gros équipements sont répartis sur deux sous-compteurs. Afin de n'avoir qu'une mesure de puissance par équipement les sous-compteurs ont été regroupés dans la suite de l'étude.

On peut observer le graphique circulaire de la répartition du pourcentage des puissances sur la puissance totale consommée dans les deux maison. Dans la maison 2 nous avons retiré le compteur

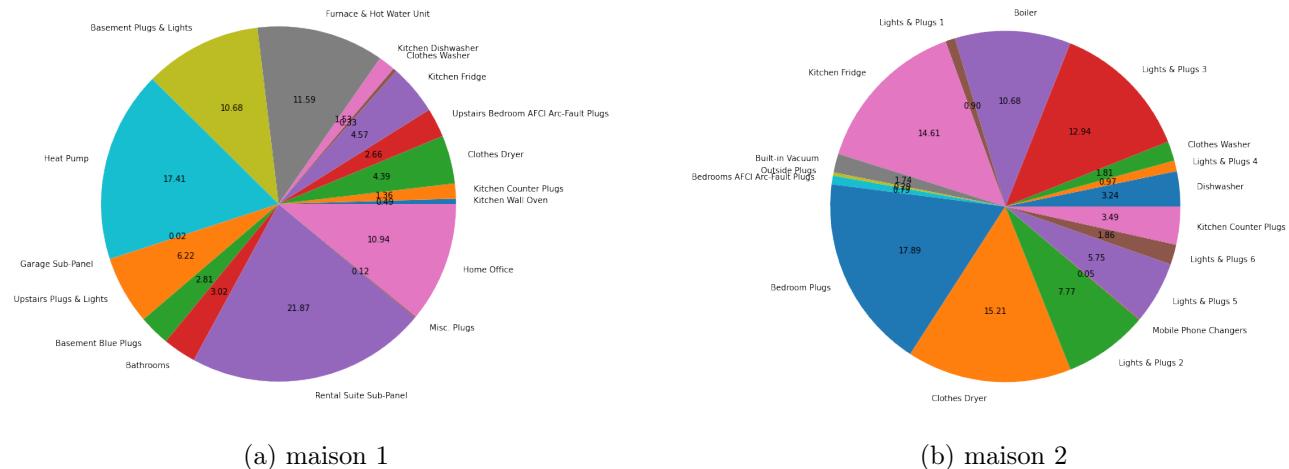


FIGURE 2.3 – Répartitions des puissances dans chaque maison

étiqueté 'House Sub-Panel' qui correspond à la courbe de puissance agrégée sur tous les sous-compteurs. On constate que dans la maison 1 la plus grosse consommation provient des sous-compteurs 'Rental Suite Sub-Panel'. Il s'agit d'un compteur pour le niveau inférieur complet (occupé par une personne), c'est-à-dire une courbe de puissance déjà agrégée, ce qui pourra poser problème pour la suite, il conviendra de rester vigilant avec ce sous-compteur dans la suite de notre étude.

## 2.2 Labellisation

Le Data Labelling ou étiquetage des données est une étape indispensable du Machine Learning. Pour entraîner une IA à partir de données, il est impératif d'étiqueter ces données au préalable. Le Machine Learning permet à nos ordinateurs d'apprendre de manière autonome, en s'entraînant à partir des données. Toutefois, pour que l'apprentissage de nos modèles puisse commencer, nous devons intervenir. Avant de nourrir le modèle avec des données, il est indispensable de les préparer.

À l'aide de divers outils, nous allons assigner des étiquettes aux données. C'est ce qui permettra ensuite à notre modèle d'apprendre à reconnaître s'il y a activité ou non. Dans notre cas afin d'entraîner convenablement nos modèles de Machine Learning, nous allons dans un premier temps labelliser manuellement chaque courbe de charge de chaque appareils. Pour identifier s'il y a activité nous ferons varier trois paramètres basés sur des règles métier : un seuil, un lissage puis un second seuil.

La prise de décision algorithmique est sujette à un biais axé sur la personne qui analyses les données ainsi qu'à un biais sur les données. Dans notre situation les étapes d'entraînement et d'évaluation des modèles reposent sur des données étiquetées manuellement et peuvent contenir un biais, ce qui entraînerait des omissions dans un modèle prédictif, bien que l'algorithme d'apprentissage automatique soit le plus optimal. Les données étiquetées utilisées pour former un algorithme d'apprentissage automatique spécifique doivent être un échantillon statistiquement représentatif et ou le moins biaisé possible afin d'obtenir les meilleurs résultats possible.

### 2.2.1 Méthodologie utilisée

Afin d'étiqueter nos jeux de données, chacune des courbes correspondantes à un sous-compteur ont été étudiées. Les sous-compteurs sont labellisés, certains sont liés à des appareils qui ne correspondent pas à de l'activité (pompe à chaleur, frigo, etc.). Ces courbes ont été ignorées. Parmi les autres courbes, celles présentant un signal trop faible (environ moins que 300W en charge maximum) ont également été ignorées car ces signaux seront noyés dans la courbe agrégée et risquent de ne pas pouvoir être détectés de manière fiable. Enfin les courbes présentant beaucoup de bruit ont également été ignorées lors de cette analyse car elles ne permettent pas de clairement identifier de phases d'activité vs non activité. Les courbes restantes ont été analysée et traitées en trois étapes décrites ci-dessous afin de rajouter un label activité vs non activité.

#### Etape 1 :

Dans cette première étapes nous créons un premier seuil en sommant la puissance moyenne à laquelle nous ajoutons l'écart-type multiplié par un coefficient que nous faisons varier pour chaque courbe de charge afin d'obtenir les meilleurs résultats possibles. Si la courbe de puissance dépasse ce seuil on considère qu'il y a activité.

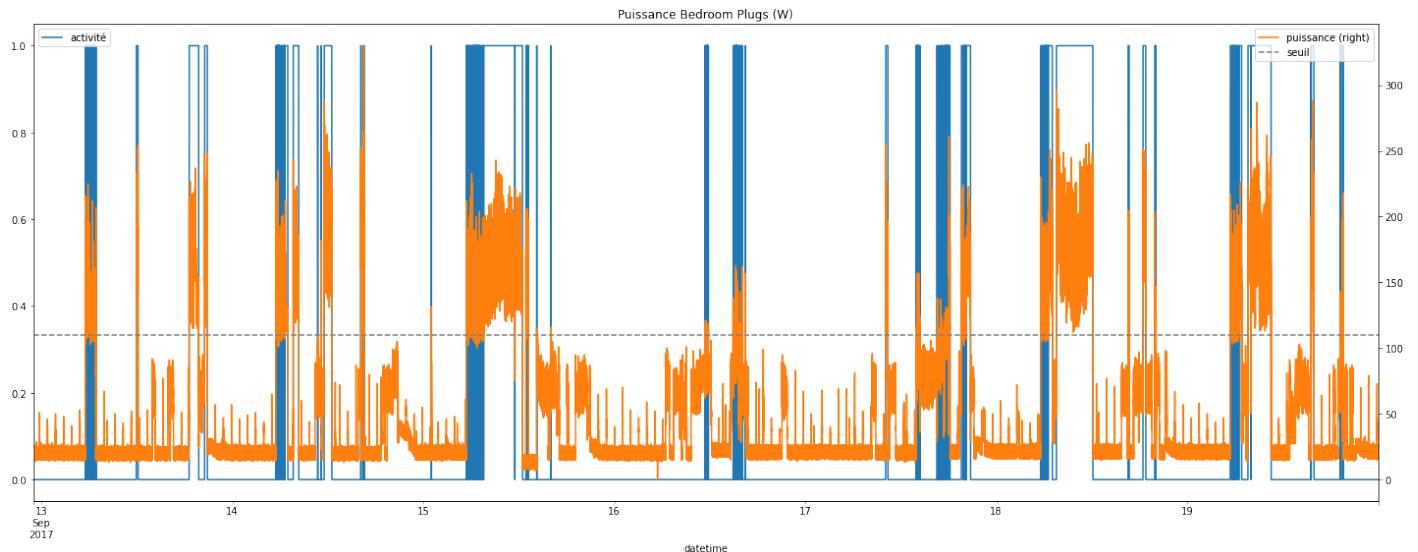


FIGURE 2.4 – Seuil à la moyenne + écart-type × coefficient

### Etape 2 :

Dans l'objectif de mieux visualiser la courbe de charge de l'appareil à analyser, et afin d'éviter les alternances activité / non activité à trop haute fréquence nous appliquons un lissage en faisant une moyenne glissante. La durée de la fenêtre sur laquelle la moyenne glissante est appliquée a été définie au cas par cas afin de coller au mieux aux périodes d'activité / non activité observées. Dans l'exemple ci-dessous le lissage fait d'avantage ressortir les variations de consommation de puissance de l'appareil.

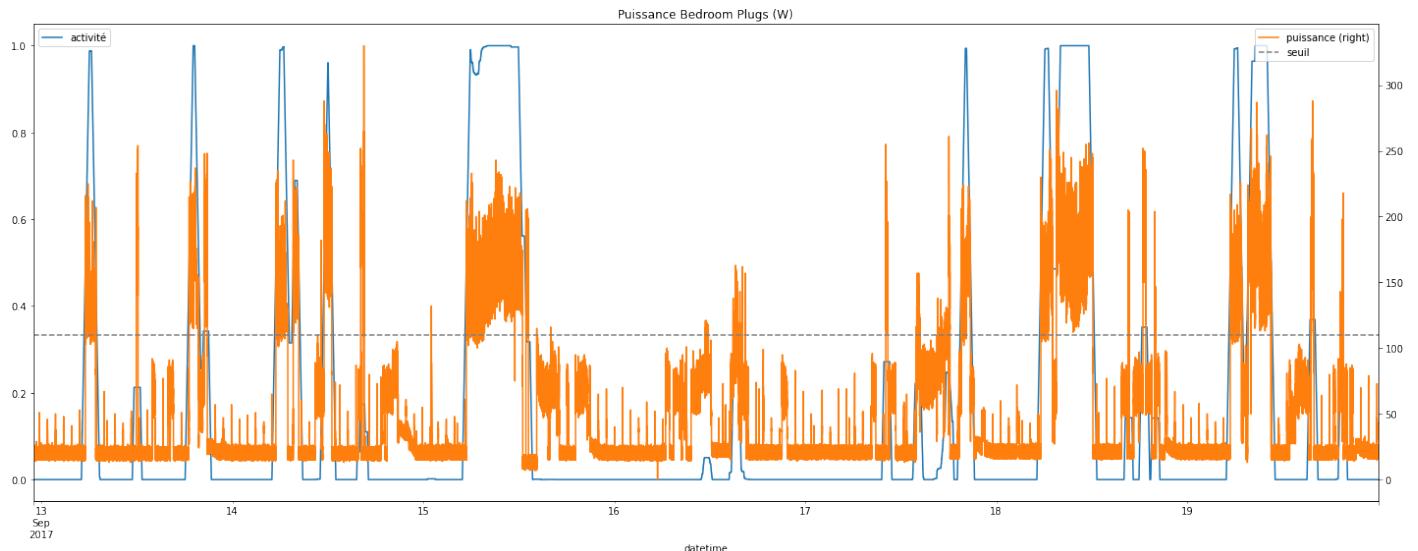


FIGURE 2.5 – Lissage de la courbe activité

### Etape 3 :

Une fois la courbe lissée nous appliquons un deuxième seuil. Nous considérons qu'il y a activité si la courbe d'activité lissée dépasse un certain seuil (le seuil égal à zéro dans cet exemple). Cela permet d'avoir un label 1 ou 0 qui correspond au label final pour ce sous-compteur : activité ou pas.

Une fois ce traitement fait sur chacune des courbes de charge retenues pour une maison données, les courbes d'activité par sous-compteur ont été agrégées afin d'obtenir la courbe finale de l'activité en fonction du temps pour les deux maisons de notre jeu de données.

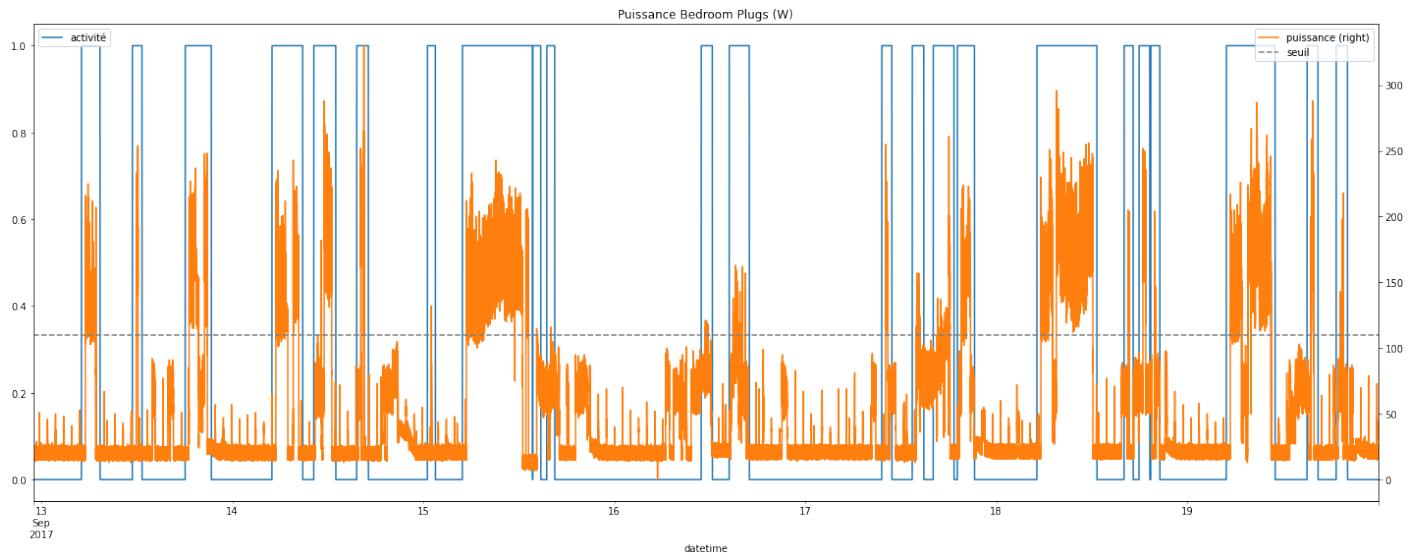


FIGURE 2.6 – Seuil de la courbe activité

On peut observer les histogrammes des labels activité par heure de la journée pour chaque courbe de charge désagrégée. Les appareil retenus pour le label activité final sont affichés en bleu, les appareils qui n'ont pas été pris en compte sont en gris.

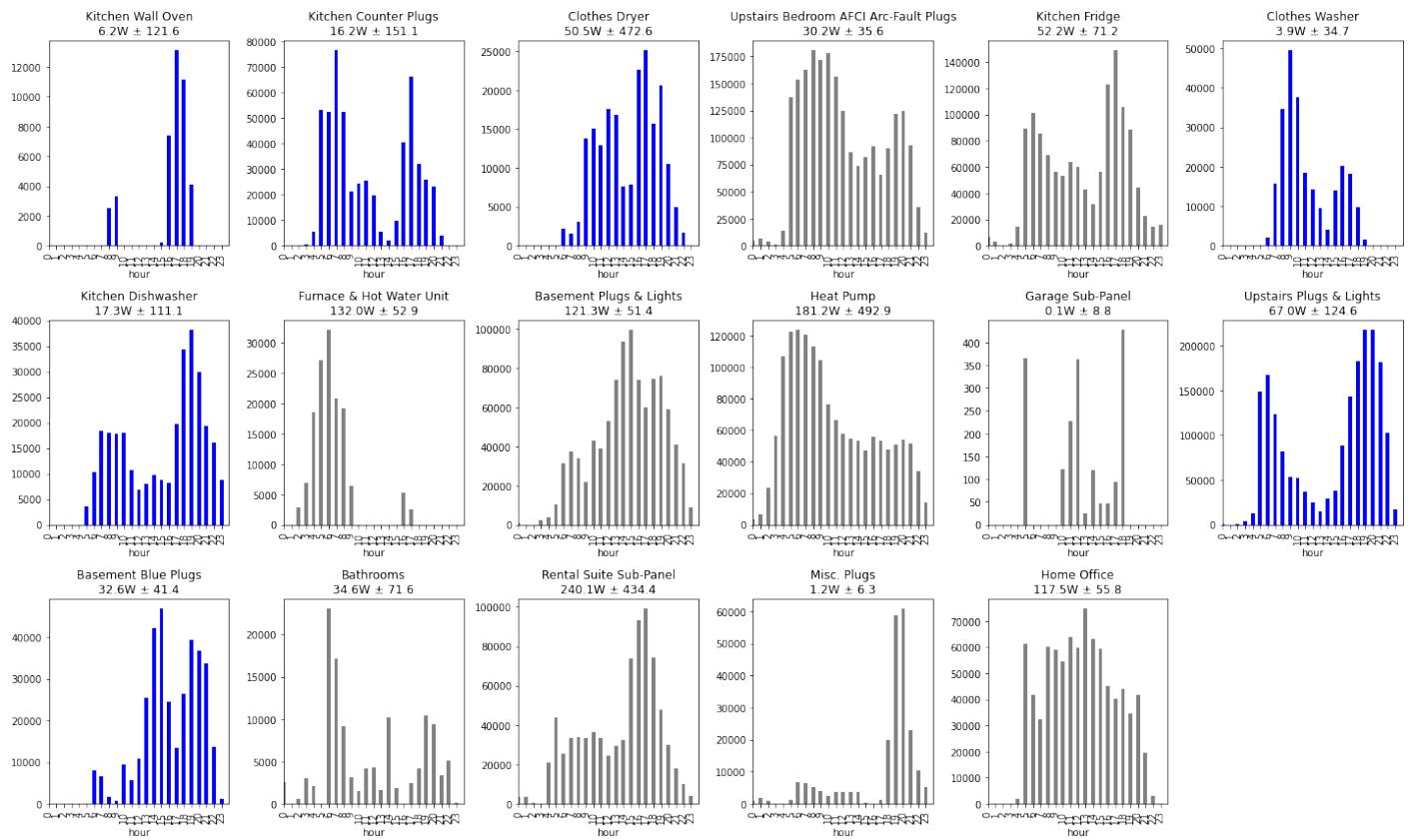


FIGURE 2.7 – Histogramme du label activité en fonction de l'heure de la journée pour chaque appareil de la maison 1 (appareil retenus en bleu, appareil ignorés en gris)

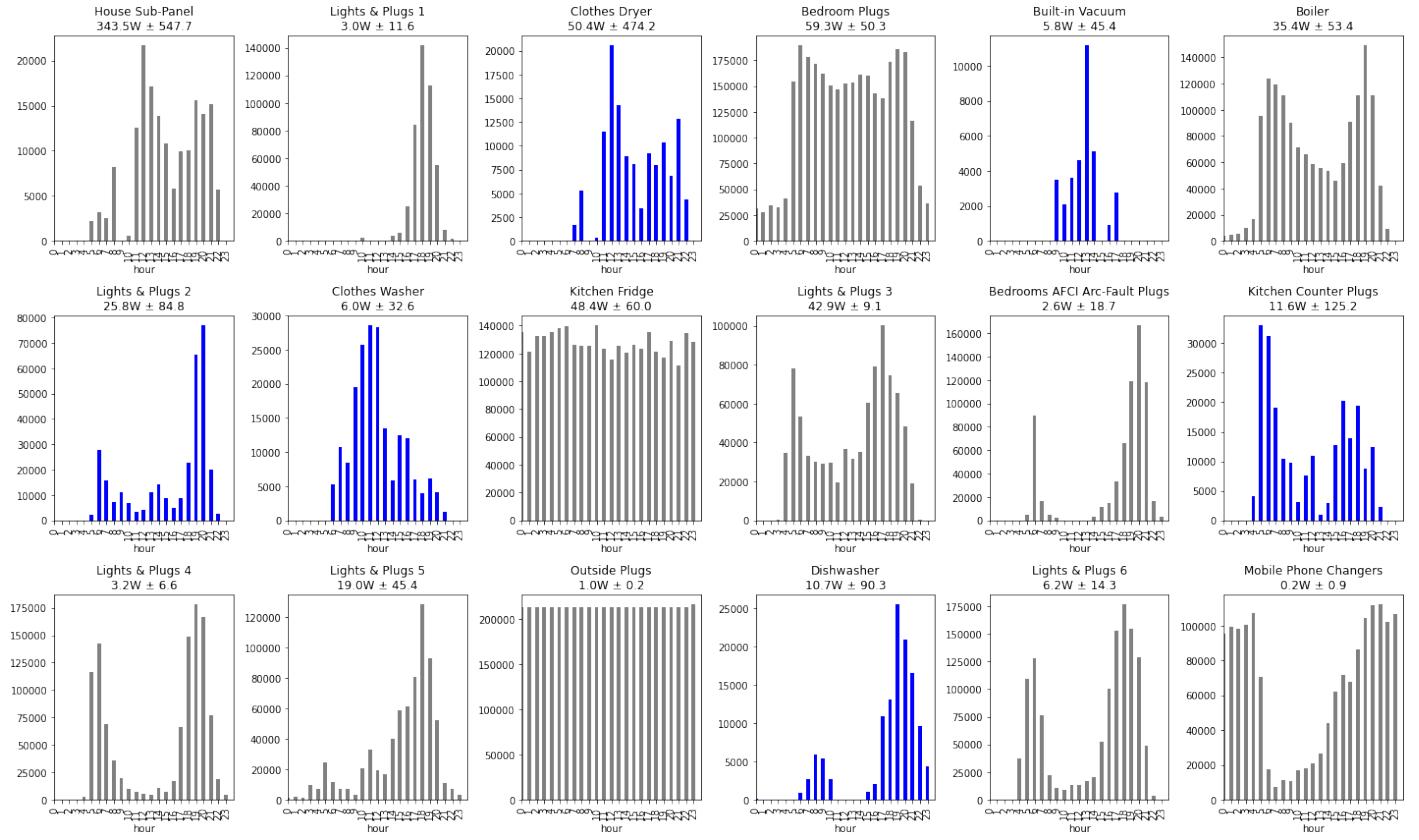


FIGURE 2.8 – Histogramme du label activité en fonction de l'heure de la journée pour chaque appareil de la maison 2 (appareil retenus en bleu, appareil ignorés en gris)

## 2.2.2 Discussion des résultats

Encore une fois, une telle méthode n'est pas idéale car on ne connaît pas la courbe d'activité réelle ("ground truth"), mais il s'agit de la meilleure approximation possible au vue des données disponibles. Les jeux de données correspondants à des courbes de charge désagrégées dans des logement sont rares, et il n'existe pas de jeux de données avec des labels correspondants à de l'activité dans un logement à notre connaissance. Nous allons utiliser ces labels pour nos tâches de classification, tout en restant prudents lors de l'interprétation des résultats.

Une façon de pouvoir juger de la qualité des résultats obtenus est d'observer les histogrammes de la courbe activité par tranches d'une heure (de 0 à 23) pour chacune des maisons.

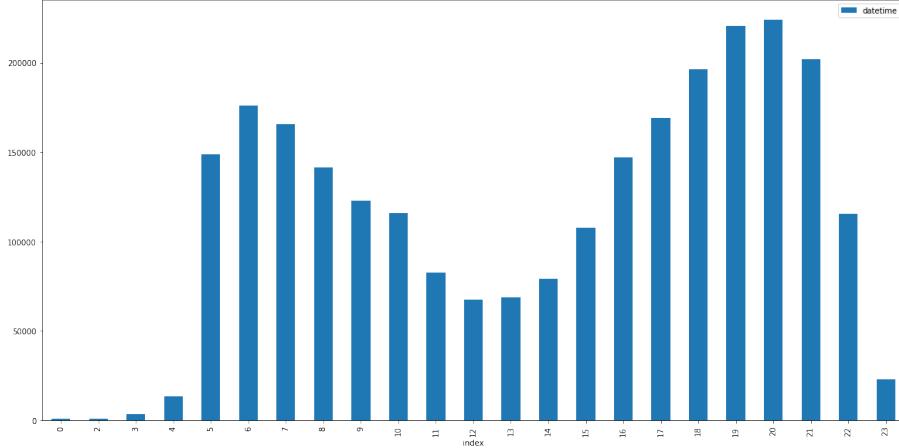


FIGURE 2.9 – Histogramme du label d’activité en fonction de l’heure de la journée pour la maison 1

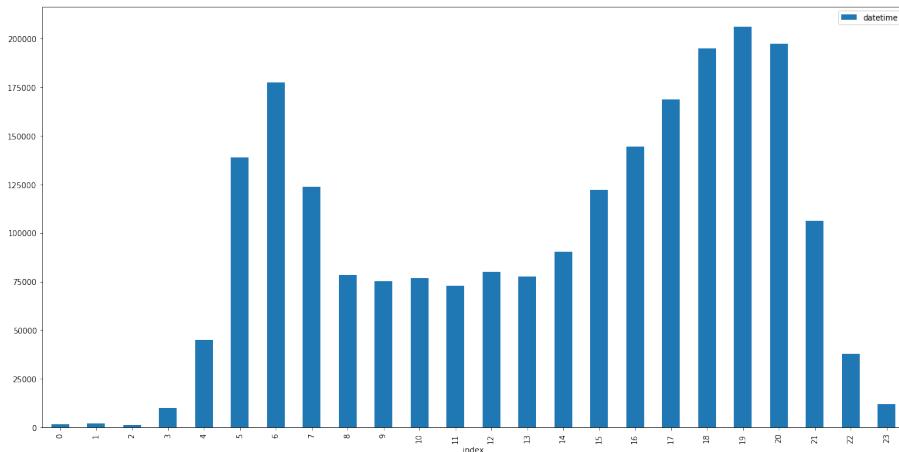


FIGURE 2.10 – Histogramme du label d’activité en fonction de l’heure de la journée pour la maison 2

Sur les deux histogrammes on observe deux pics d’activité : un vers 6h du matin et un autre vers 19h / 20h. Ces pics peuvent correspondre respectivement aux heures de levée et de retour du travail le soir. On observe qu’on ne labellise pas de périodes d’activité durant la nuit (de 23h à environ 4h du matin), ce qui semble cohérent avec ce à quoi on pourrait s’attendre pour l’activité d’un foyer.

## 2.3 Modélisation

Nous rappelons que l’objectif principal est de détecter de l’activité (classification binaire) à partir de la courbe de la puissance totale consommée dans le logement.

### 2.3.1 Approche 1 : Algorithmes de classification supervisés

Notre première approche consiste à rajouter des colonnes features (phase de features engineering) puis à utiliser des algorithmes de classification binaire classiques sur les colonnes créées. On peut noter que cette approche traite chaque point de mesure indépendamment, et ne tient pas compte de l’aspect série temporelle.

### 2.3.1.1 Pre-processing : Création de features

Notre approche consiste à générer des colonnes features à partir de la courbe de puissance agrégée et de la colonne date et heure. Les colonnes générées sont les suivantes :

- la moyenne glissante sur x minutes,
- l'écart type glissant sur x minutes,
- l'amplitude glissante (max - min) sur x minutes,
- le nombre de fois à la courbe de puissance agrégée coupe la moyenne glissante (pour avoir une mesure du niveau d'oscillations),
- l'heure de la journée (un nombre à virgule flottante de 0 à 23),
- le fait que le jour soit un jour de semaine ou de weekend (encodé en one hot)

Ces colonnes proviennent de règles métiers et sont des features couramment utilisées lors de la manipulation de données issues de relevés de mesures effectués au cours du temps. On peut calculer les coefficients de corrélation entre ces features et la colonne label 'activité' pour voir quelles colonnes semblent les plus pertinentes.

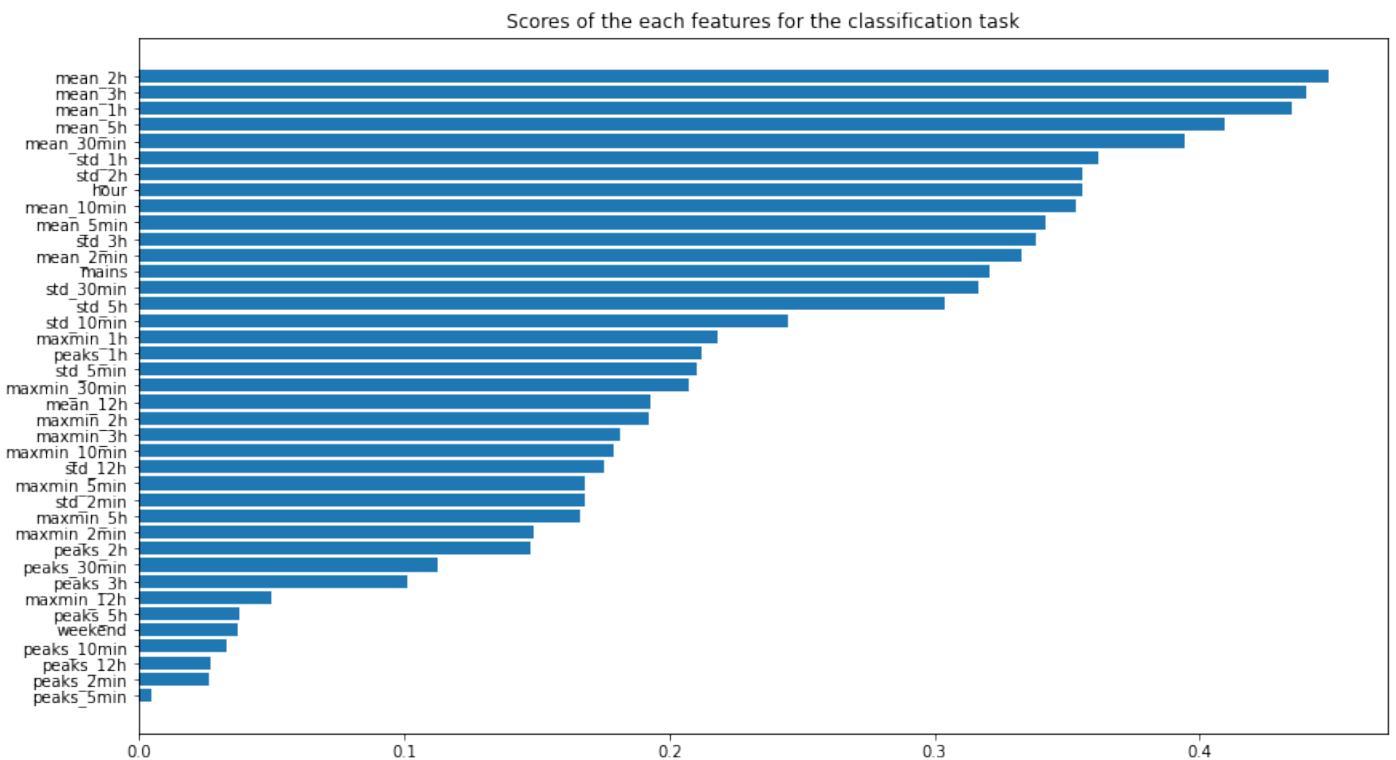


FIGURE 2.11 – Score de chaque feature pour la tache de classification pour la **maison 1**

Une pratique courante consiste à choisir une fenêtre de temps sur le long terme (de l'ordre de l'heure) et une sur le plus court terme (de l'ordre de la minute). En observant les coefficients de corrélation obtenus nous allons retenir des fenêtres de 1 heure et de 10 minutes pour toutes les colonnes générées. Le choix de fenêtres de temps homogènes pour toutes les colonnes générées a été fait afin de simplifier la démarche.

### 2.3.1.2 Entraînement des modèles et premiers résultats

Maintenant que les colonnes ont été sélectionnées nous pouvons faire une classification classique, c'est-à-dire prédire le label activité en fonction des colonnes features. Pour cela nous faisons d'abord un train/test split en sélectionnant des journées entières aléatoirement pour le jeu de test (30% du jeu

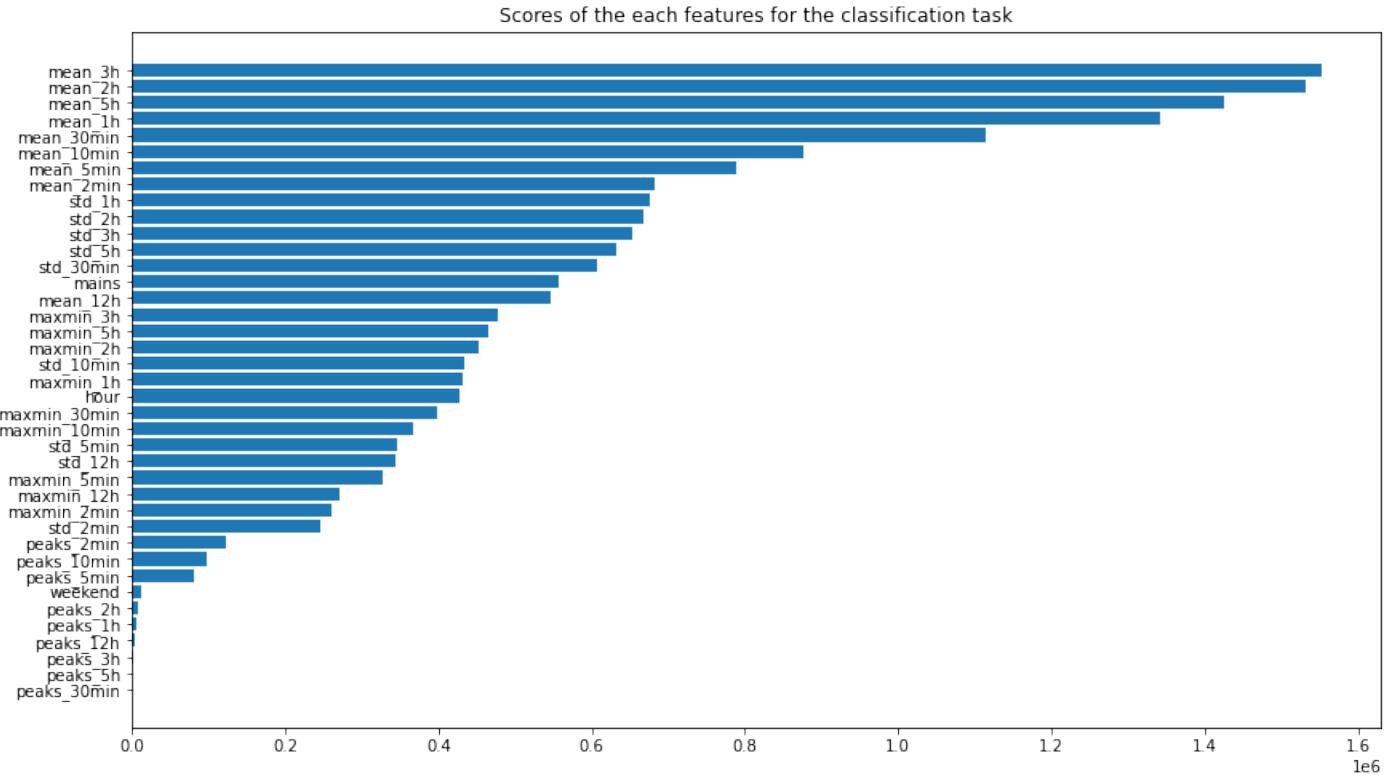


FIGURE 2.12 – Score de chaque feature pour la tache de classification pour la **maison 2**

de donnés, 70% réservés pour l’entraînement). Nous avons ensuite testé plusieurs algorithmes de classifications : *kNearestNeighbors*, *RandomForest*, *GradientBoosting* (dans sa version sur les histogrammes implémentée dans sklearn, qui permet de retourner des résultats assez précis pour de gros jeux de données tout en limitant les temps de calculs comparé à l’implémentation classique). Pour chaque algorithme, nous avons réalisé un grid-search afin d’estimer les meilleures hyper-paramètres. Les données ont été mises à l’échelle avec la méthode *StandardScaler* de sklearn, l’objet scaler ayant été ajusté (fitté) sur les données d’entraînement et appliqué sur les données de test (nous reviendrons sur ce points plus tard). Les meilleurs résultats obtenus par chaque famille d’algorithmes sont visibles sur la figure 2.13.

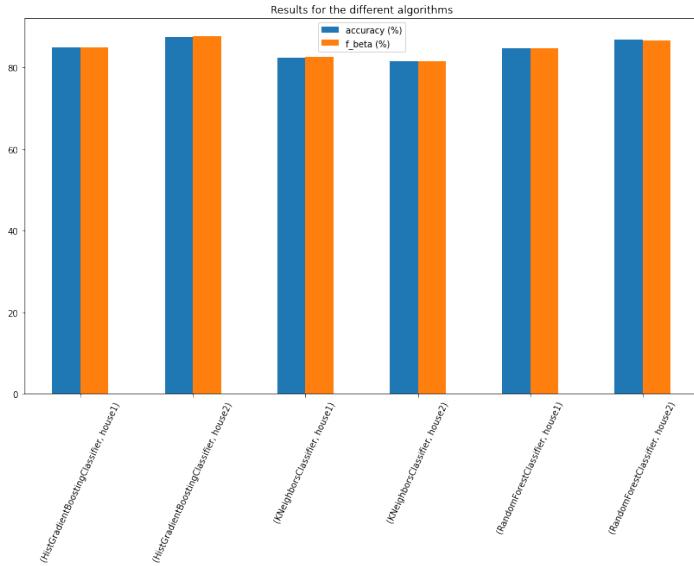


FIGURE 2.13 – Meilleur score obtenu pour chaque famille d’algorithme sur la tache de classification pour les **maison 1** et **maison 2**

Ces scores donnent une bonne idée générale de la performance de nos algorithmes de classification, mais nous devons rester prudents, ne connaissant pas le label activité / non activité réel. Une autre façon d’apprécier la performance de nos algorithmes est d’observer l’histogramme du label activité prédit par tranche horaire. On peut notamment essayer de voir si nos algorithmes sont capables d’obtenir une distribution du label à prédire qui s’approche de la distribution cible. Nous avons tracé ces histogrammes normalisés pour les labels prédits par le *Random Forest* avec les hyper-paramètres que nous avons déterminés précédemment, comme c’est un des algorithmes qui obtenait les meilleurs scores.

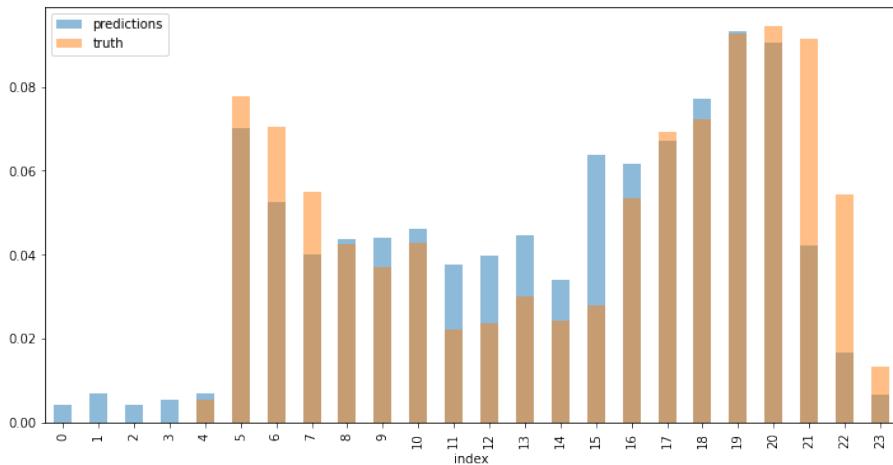


FIGURE 2.14 – Histogramme du label d’activité prédit par notre Random Forest en fonction de l’heure de la journée pour la maison 1

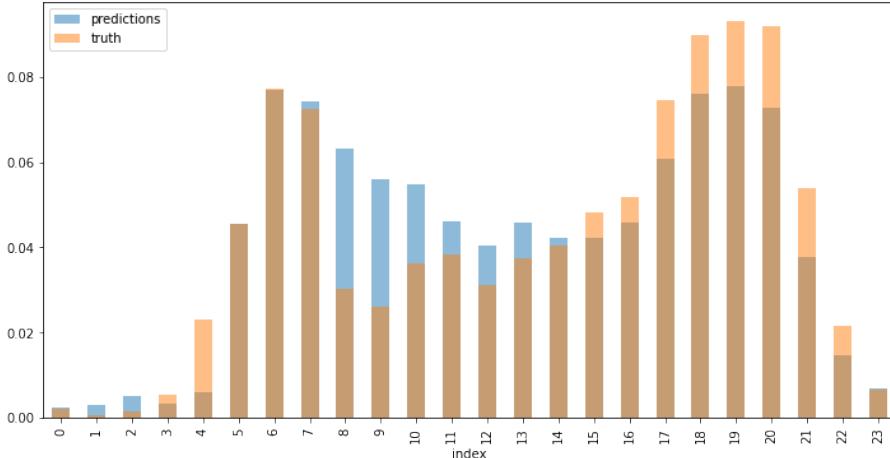


FIGURE 2.15 – Histogramme du label d’activité prédit par notre Random Forest en fonction de l’heure de la journée pour la maison 2

On constate que les distributions des labels prédits se rapprochent des distributions des labels cibles, avec des pics d’activité vers 6h et 19h et peu d’activité la nuit entre 23h et 4h du matin. On constate cependant un certain nombre de faux positifs, notamment durant les périodes de nuit.

### 2.3.1.3 Limitations de la méthode

Ces résultats peuvent paraître concluants, mais une grosse difficulté est de pouvoir obtenir des algorithmes généraux, capables d’obtenir de bons résultats d’une maison à l’autre sans entraînement supplémentaires. Des tests ont été effectués en entraînant un classifieur sur la maison 1 et en testant les résultats sur la maison 2 et vice-versa.

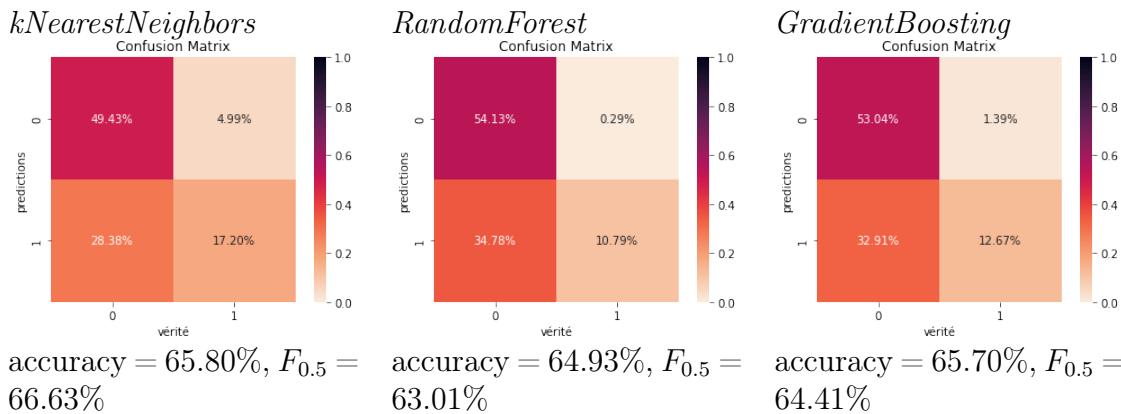


FIGURE 2.16 – Performance des classificateurs en entraînant sur la maison 1 et en testant sur la maison 2

On peut en faire de même en entraînant les classificateurs sur la maison 2 et en testant sur la maison 1.

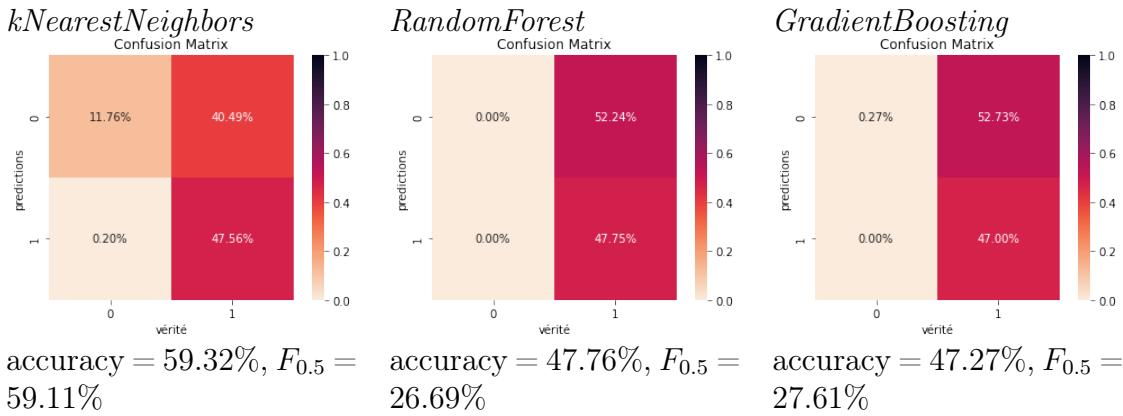


FIGURE 2.17 – Performance des classificateurs en entraînant sur la maison 2 et en testant sur la maison 1

On se rend alors compte que dans ces conditions les performances de nos classificateurs se retrouvent fortement dégradées et les résultats ne sont pas exploitables. Il semblerait que cette approche ne permette pas d'obtenir des classificateurs qui se généralisent d'un logement à un autre. En observant les données de plus près, il semblerait qu'il y ait un problème lors de la mise à l'échelle. L'objet scaler ayant été ajusté sur les données d'une maison, la mise à l'échelle des données de l'autre maison retourne des valeurs en dehors de la plage sur laquelle le classifieur a été entraîné.

### 2.3.1.4 Premier raffinement et résultats

Une idée afin de pallier à ce problème est d'utiliser deux objets scalers : un premier scaler fitté sur les données d'entraînement lors de l'entraînement du classifieur et un second fitté sur une partie des données de la maison sur laquelle on cherche à faire nos prédictions lors des tests. Cette approche pourrait être envisageable, car lorsqu'un nouveau client demande à accéder au service cela nécessiterait simplement de collecter des données de puissance agrégée sur une certaine période de temps afin de pouvoir ajuster un objet scaler avant de pouvoir faire des prédictions. Nous pouvons tester cette approche en utilisant la maison 2 comme jeu d'entraînement et la maison 1 comme jeu de test car c'est la situation qui retournait les résultats les plus défavorables. Nous avons de plus testé plusieurs méthodes de mise à l'échelle et sur plusieurs fenêtres de temps utilisées pour l'ajustement (fitting) de la méthode de mise à l'échelle. Le classifieur *Random Forest* a été utilisé avec les hyper-paramètres définis à l'aide du grid-search précédemment effectué. En effet ce classifieur retournait des résultats relativement bons comparés aux autres familles d'algorithmes tout en étant assez rapide à calculer. Les graphes ci-dessous montrent l'évolution du score (accuracy) et de score  $F_{0.5}$  en fonction de la fenêtre de temps utilisée pour fitter l'objet scaler (compté en nombre de jours).

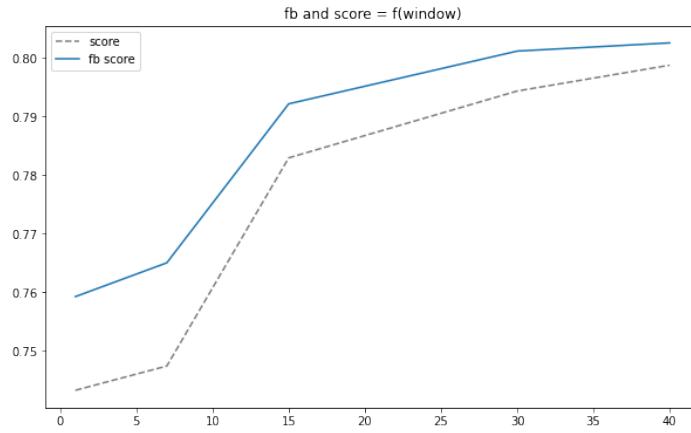


FIGURE 2.18 – Scores obtenus en faisant un fitting du scaler **MinMax** sur une partie du test

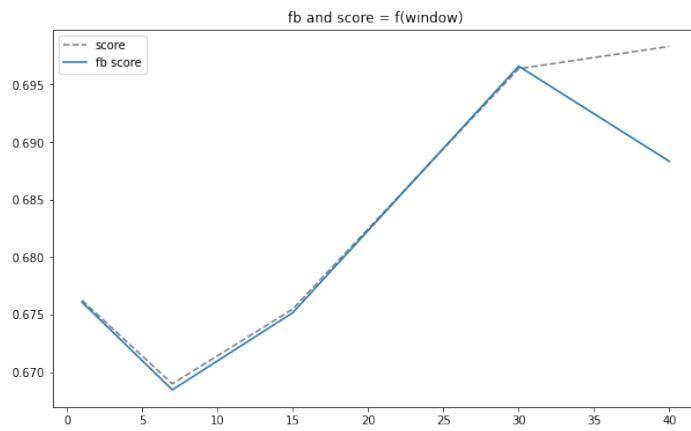


FIGURE 2.19 – Scores obtenus en faisant un fitting du scaler **Standard** (moyenne et écart-type) sur une partie du test

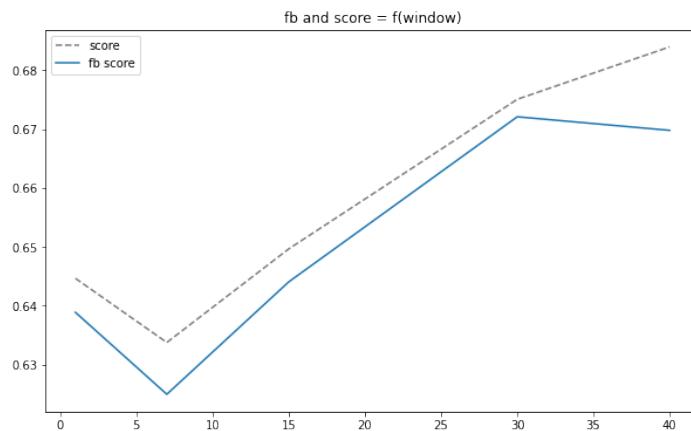


FIGURE 2.20 – Scores obtenus en faisant un fitting du scaler **Quantile** sur une partie du test

Le scaler MinMax semble retourner de bons scores et il présente l'avantage de n'avoir à enregistrer que deux valeurs : le minimum et le maximum de relevé de puissance. On pourrait dès lors envisager de faire des enregistrements sur 15 voire 30 jours pour chaque nouveau client qui souhaite bénéficier de la détection d'activité dans son domicile, en ne retenant que le minimum et le maximum consommé.

On serait alors en mesure de faire une mise à l'échelle des relevés de mesure dans ce logement et nous pourrons appliquer nos algorithmes de classification de façon plus fiable.

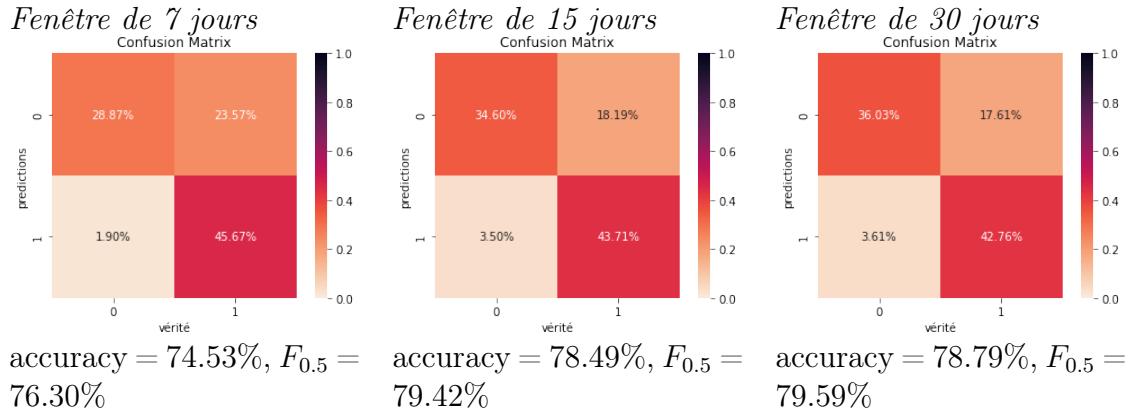


FIGURE 2.21 – Performance du classifieur entraîné sur le train (maison 2) en faisant un fitting du scaler MinMax sur différentes fenêtres du test (maison 1)

Vérifions les résultats obtenus en entraînant le classifieur sur la maison 1 et en testant sur la maison 2.

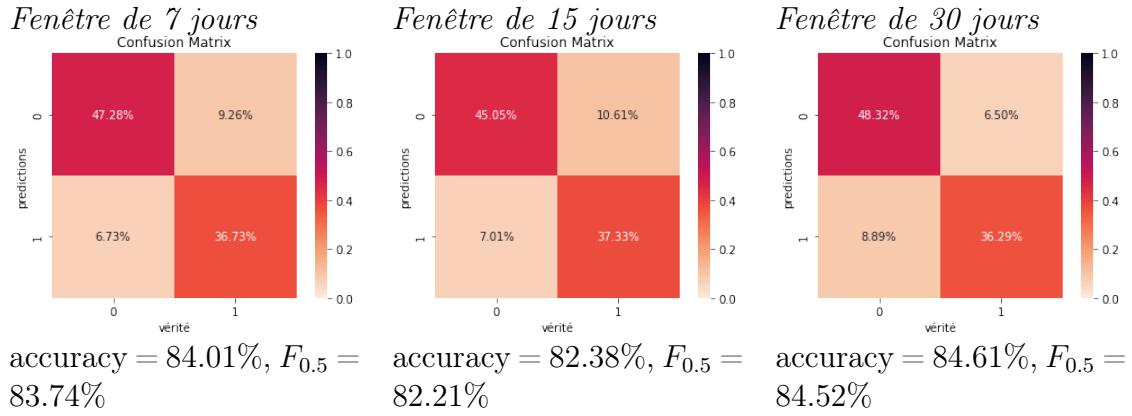


FIGURE 2.22 – Performance du classifieur entraîné sur le train (maison 2) en faisant un fitting du scaler MinMax sur différentes fenêtres du test (maison 1)

On peut encore une fois observer l'histogramme du label activité prédit par tranche horaire pour vérifier si nos algorithmes sont capables d'obtenir une distribution du label à prédire s'approchant de la distribution cible. Nous avons tracé ces histogrammes pour les labels prédits par notre *Random Forest* entraîné sur la maison d'entraînement (maison 2), après avoir fait un fitting du scaler MinMax sur une fenêtres de 15 jours du test (maison 1).

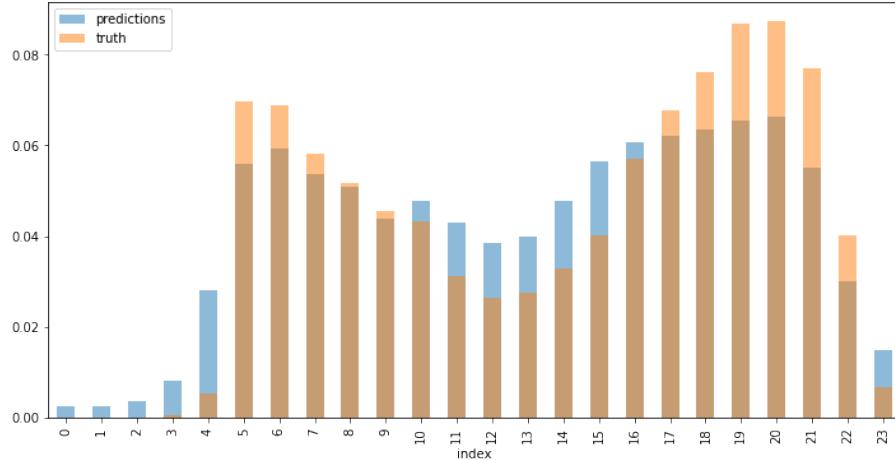


FIGURE 2.23 – Histogramme du label d’activité prédict en fonction de l’heure de la journée par notre Random Forest entraîné sur la maison 2 en faisant un fitting du scaler MinMax sur 15 jours du test (maison 1)

On peut également vérifier les résultats en entraînant le classifieur sur la maison 1 et en testant sur la maison 2.

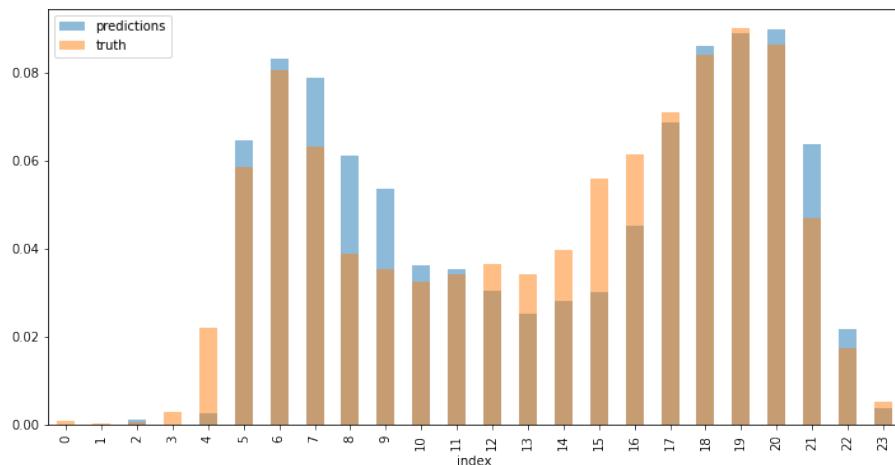


FIGURE 2.24 – Histogramme du label d’activité prédict en fonction de l’heure de la journée par notre Random Forest entraîné sur la maison 1 en faisant un fitting du scaler MinMax sur 15 jours du test (maison 2)

On observe une distribution qui se rapproche de la distribution cible, bien qu’on prédise plus d’activité la nuit, notamment à 23h et 4h du matin. On constate également plus d’activité en début/milieu d’après-midi comparé à notre distribution cible.

Pour aller plus loin, on peut également tester des méthodes d’apprentissage semi-supervisées. L’idée ici est d’utiliser les données labellisées sur une des maisons et de les combiner à des données non labellisées sur l’autre maison afin d’entraîner un classifieur. Pour cela on entraîne d’abord un classifieur sur des données labellisées. On fait ensuite une prédiction sur les données non labellisées. Nous considérons ensuite chaque prédiction qui dépasse une certaine indice de confiance (nous allons prendre 0.75, cet indice est obtenu avec une méthode du type `predict_proba`), comme un point de notre jeu labellisé avec comme label celui obtenu avec le classifieur. On re-itère ensuite jusqu’à avoir labellisé tout le jeu de données non labellisé. Dans le cas d’usage pratique ou pourrait imaginer entraîner un classifieur pour

chaque nouveau client à l'aide des données des deux maisons ainsi que des relevés de mesure issues du logement du nouveau client sur une certaine fenêtre de temps. On obtient alors un nouveau classifieur entraîné sur nos jeux de données labellisées mais également sur des données provenant du logement du client. On peut alors utiliser ce nouveau classifieur pour faire des prédictions sur chaque nouveau relevé de mesure (sans le re-entraîner à chaque fois, car cela prendrait beaucoup de temps). De même que précédemment un peut fitter l'objet scaler sur des données provenant d'une fenêtre de temps que nous allons faire varier. Nous allons en plus rajouter les données issues de cette fenêtre de temps à notre jeu de données comme données non labellisées. Pour nos tests nous avons utilisé un *RandomForest* avec les mêmes hyper-paramètres que ceux utilisés précédemment, ainsi qu'un scaler *MinMax*.

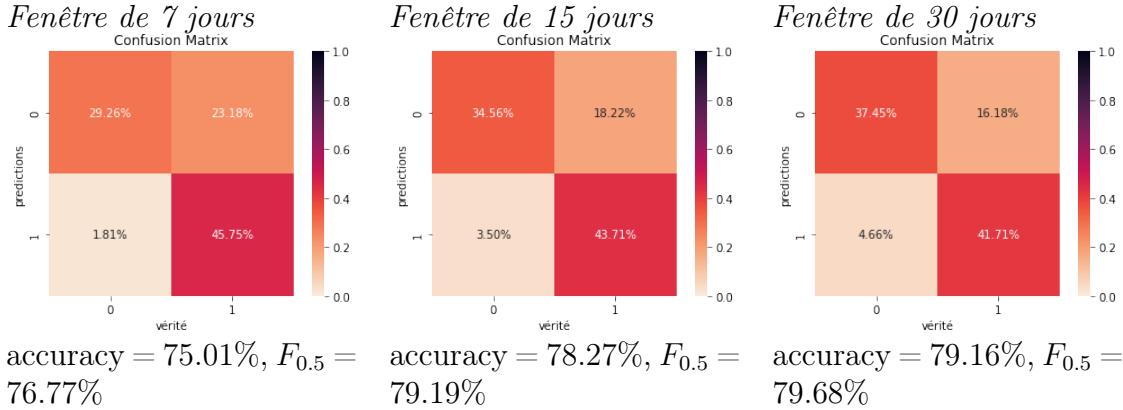


FIGURE 2.25 – Performance du classifieur entraîné sur le train (maison 2) en faisant un fitting du scaler *MinMax* et en appliquant un algorithme semi-supervisé avec différentes fenêtres du test (maison 1)

Comme on peut le voir cette méthode ne retourne pas des résultats beaucoup mieux que ceux obtenus en faisant simplement une mise à l'échelle à partir des données issues d'une même maison sur une certaine fenêtre de temps. Cette méthode n'est de plus pas bien fondée sur le plan théorique et les hypothèses faites ne sont pas claires. À l'issue de ces tests, on peut en conclure qu'il vaut mieux faire un entraînement classique sur un jeu de données et utiliser notre méthode de mise à l'échelle sur une fenêtre à définir.

### 2.3.1.5 Optimisation d'un modèle et résultats

Maintenant que nous avons une idée plus précise de la méthodologie à appliquer pour mettre nos données à l'échelle et entraîner nos modèles de classification nous allons tenter d'améliorer encore plus les performances en cherchant la meilleure combinaison de mise à l'échelle, algorithme et hyper-paramètres. Pour cela nous allons tester 5 algorithmes d'ensemble learning car c'est la famille d'algorithmes qui ont retournés les meilleurs résultats :

- Random Forest
- Histogram-Based Gradient Boosting
- Light Gradient Boosting Machine
- Categorical Boosting
- eXtreme Gradient Boosting

Le dernier algorithme d'ensemble learning que nous allons utiliser est un *VotingClassifier*, dans ce dernier cas pour notre algorithme de voting nous utiliserons les 4 algorithmes de boosting que nous avons cités précédemment. Nos données initiales seront transformées par la liste de scaler ci-dessous. Après transformation de nos données nous appliquerons un à un nos algorithmes d'ensemble learning et nous calculerons l'accuracy et le score  $F_\beta$  pour ces algorithmes. À noter que nous avons écarté l'algorithme

kNN de la suite de nos tests car celui-ci était long à calculer et retournait des résultats moins bons que les algorithmes de la famille des ensemble learning.

Nous allons de plus testé plusieurs méthodes de mise à l'échelle. Chacune des ces méthode est décrite ci-dessous.

**MinMaxScaler** : redimensionne l'ensemble de données de façon à ce que toutes les valeurs de la fonction soient dans la plage [0, 1].

$$\mathbf{x}_{\text{scaled}} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

**StandardScaler** : supprime la moyenne et met les données à l'échelle de la variance unitaire.

$$\mathbf{x}_{\text{scaled}} = \frac{x - \mu}{\sigma}$$

**QuantileTransformer** : applique une transformation non linéaire telle que la fonction de densité de probabilité de chaque caractéristique sera mappée à une distribution uniforme ou gaussienne. Dans ce cas, toutes les données, y compris les valeurs aberrantes, seront mappées à une distribution uniforme avec la plage [0, 1], rendant les valeurs aberrantes impossibles à distinguer des incrustations.

**RobustScaler** : transforme le vecteur caractéristique en soustrayant la médiane, puis en divisant par la plage interquartile (quartile 75 — quartile 25).

$$\mathbf{x}_{\text{scaled}} = \frac{x - x_{\text{med}}}{x_{75} - x_{25}}$$

**Normalized** : Chaque échantillon (c.-à-d. chaque ligne de la matrice de données) est rééchelonné indépendamment des autres échantillons de sorte que sa norme (l1, l2 ou inf) est égale à un.

**MaxAbsScaler** : Mise à l'échelle de chaque caractéristique par sa valeur absolue maximale.

$$\mathbf{x}_{\text{scaled}} = \frac{x}{\max(|x|)}$$

**PowerTransformer** : Appliquez une transformation de puissance en fonction des features pour rendre les données « plus gaussiennes ». Les transformations de puissance sont une famille de transformations paramétriques monotones qui sont appliquées pour rendre les données « plus gaussiennes ». Ceci est utile pour les problèmes de modélisation liés à l'hétéroscédasticité (variance non constante) ou d'autres situations où la normalité est souhaitée.

Pour la suite de nos tests, nous allons nous placer dans la situation optimale : nous avons un objet scaler ajusté sur l'ensemble du jeu d'entraînement et un objet scaler ajusté sur l'ensemble du jeu de

test. Bien que cette configuration ne soit pas réaliste, nous souhaitons nous placer dans les conditions optimales pour chaque méthode de mise à l'échelle et chaque algorithmes afin de juger des performance dans l'absolu et toutes choses étant égales par ailleurs.

### Entraînement sur la maison 1 et test sur la maison 2

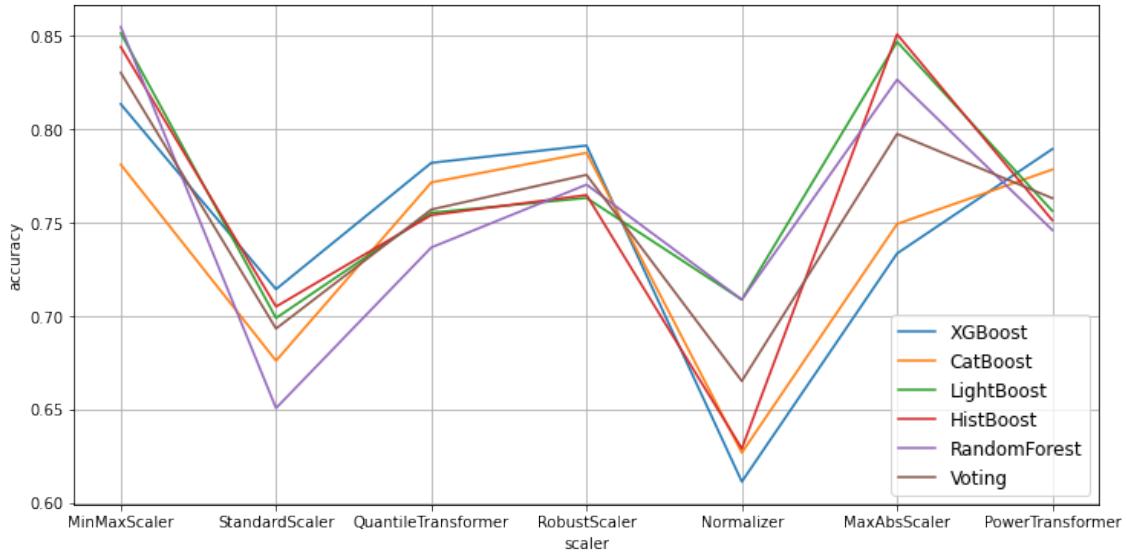


FIGURE 2.26 – Évolution de l'accuracy en fonction des scalers pour chaque algorithmes d'ensemble learning

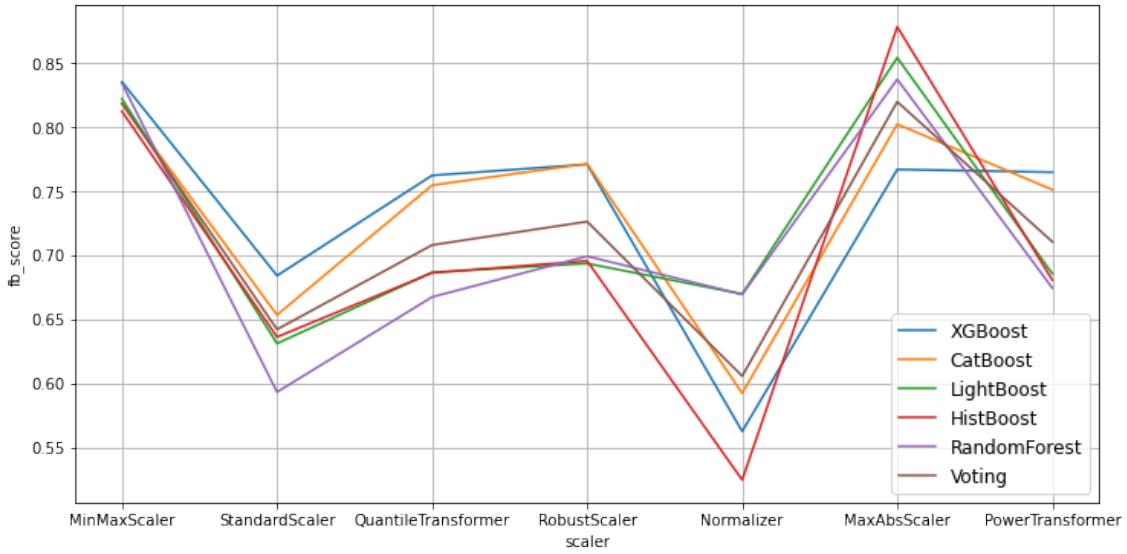


FIGURE 2.27 – Évolution du  $F_\beta$ -score en fonction des scalers pour chaque algorithmes d'ensemble learning

Le *Normalizer* est le scaler qui nous renvoie le plus mauvais score notamment pour le  $F_\beta$  score. Par ailleurs nous pouvions nous attendre à de meilleurs résultats avec le *VotingClassifier* mais il est à peine meilleur que les autres algorithmes et même dans certains cas il est moins bon. Cela est certainement dû au fait que les classifieurs prédisent chacun quasiment les mêmes résultats pour les labels.

Nous allons maintenant évaluer l'efficacité de nos classifieurs en fonction des scalers sur deux périodes différentes : le jour et la nuit. En effet ces deux périodes possèdent des caractéristiques et des distributions

distinctes et il sera intéressant de mesurer la performance des scalers et des classifieurs sur chacune des deux périodes. Nous allons aussi découvrir si cela permet d'améliorer un peu plus les scores. Dans le but d'accélérer nos calculs nous allons supprimer de nos listes le scaler *Normalizers*.

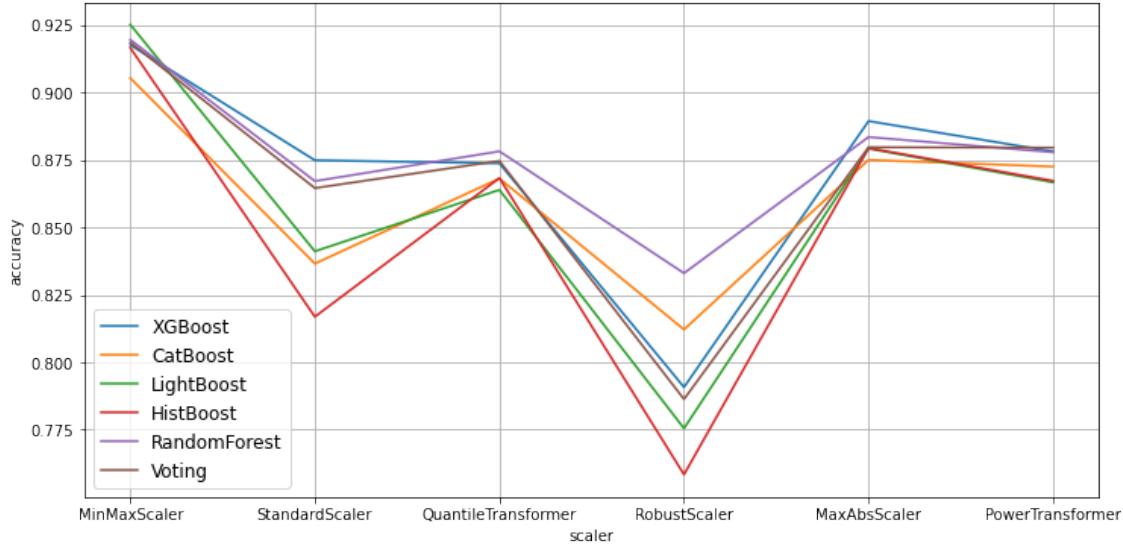


FIGURE 2.28 – Activité la nuit : Évolution de l'accuracy en fonction des scalers pour chaque algorithmes d'ensemble learning

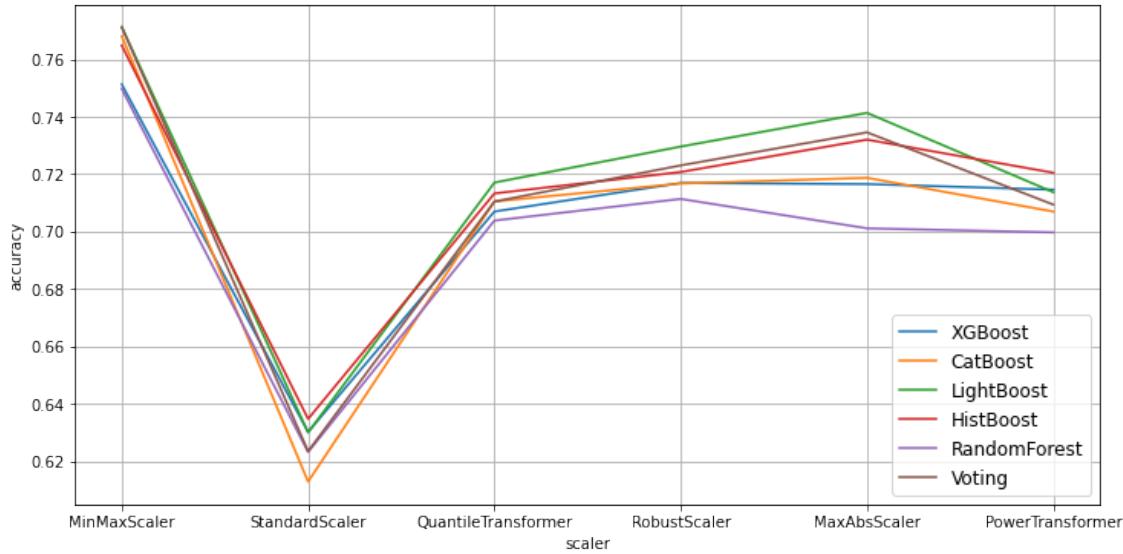


FIGURE 2.29 – Activité le jour : Évolution de l'accuracy en fonction des scalers pour chaque algorithmes d'ensemble learning

L'accuracy des classifieurs fluctuent davantage sur la période nocturne qu'en journée. Mais pour autant les modèles appliqués sur la période nocturne renvoient de meilleurs résultats que l'activité en journée. Notons que pour le *VotingClassifier* avec le scaler *MaxAbsScaler* nous renvoie le résultat le plus fluctuant avec 0.88 d'accuracy sur la période nocturne et 0.735 d'accuracy en journée. Les classifieurs *RandomForest* et *LGBMClassifier* avec *MinMaxScaler* nous donnent les meilleurs accuracy en période nocturne. En journée *LGBMClassifier* est le classifieur qui nous renvoie les meilleurs résultats notamment avec *MinMaxScaler*. Regardons ces mêmes résultats si nous nous

entraînons sur la maison 2 et que nous testons nos classifieurs avec scalers sur la maison 1.

### Entraînement sur la maison 2 et test sur la maison 1

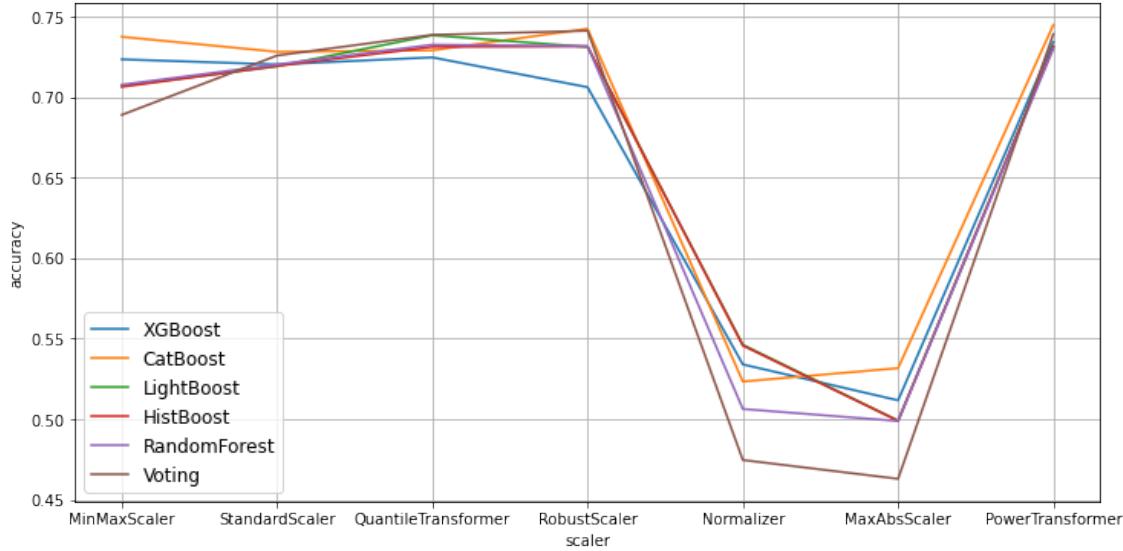


FIGURE 2.30 – Évolution de l’accuracy en fonction des scalers pour chaque algorithmes d’ensemble learning

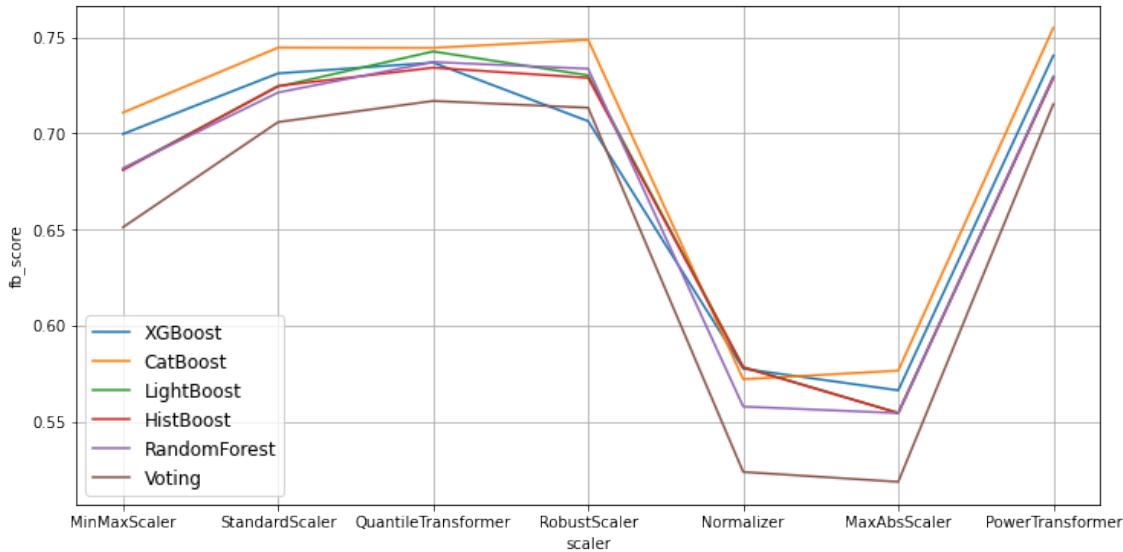


FIGURE 2.31 – Évolution du  $F_\beta$ -score en fonction des scalers pour chaque algorithmes d’ensemble learning

Les scalers *Normalizer* et *MaxAbsScale* dégradent fortement des résultats de nos classifieurs et le classifieur *RandomForest* nous retourne pour chaque scalers les plus mauvais résultats. Nous remarquons que les résultats de l’accuracy et du  $F_\beta$  score fluctuent de la même manières. Le classifieur *CatBoostClassifier* nous retourne de très bon résultat avec *QuantileTransformers* et *PowerTransformer*. Regardons maintenant l’évolution des résultats sur la période nocturne et en journée, nous retirons le scalers *Normalizer* de notre évaluation.

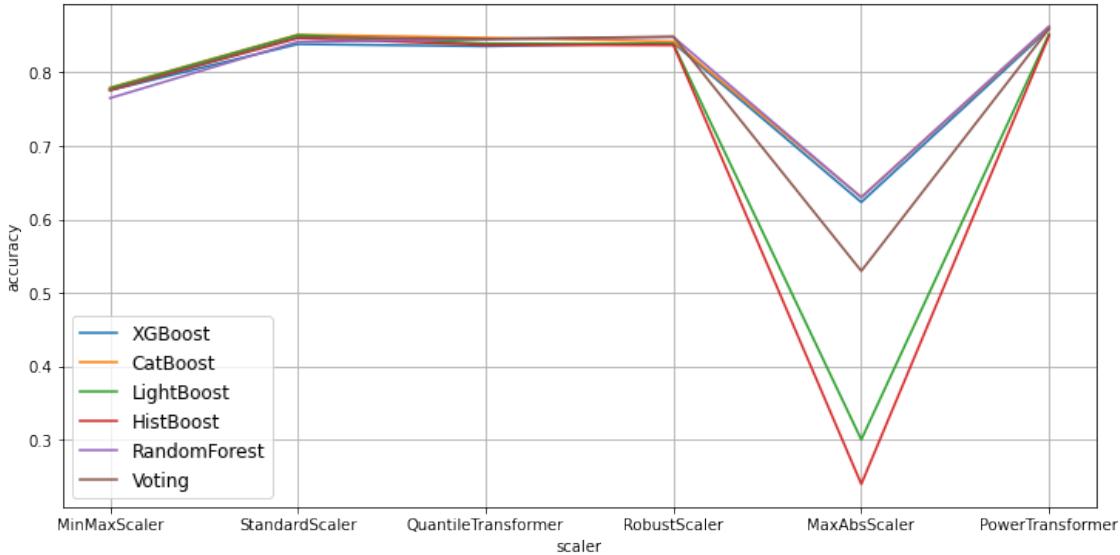


FIGURE 2.32 – Activité la nuit : Évolution de l’accuracy en fonction des scalers pour chaque algorithmes d’ensemble learning

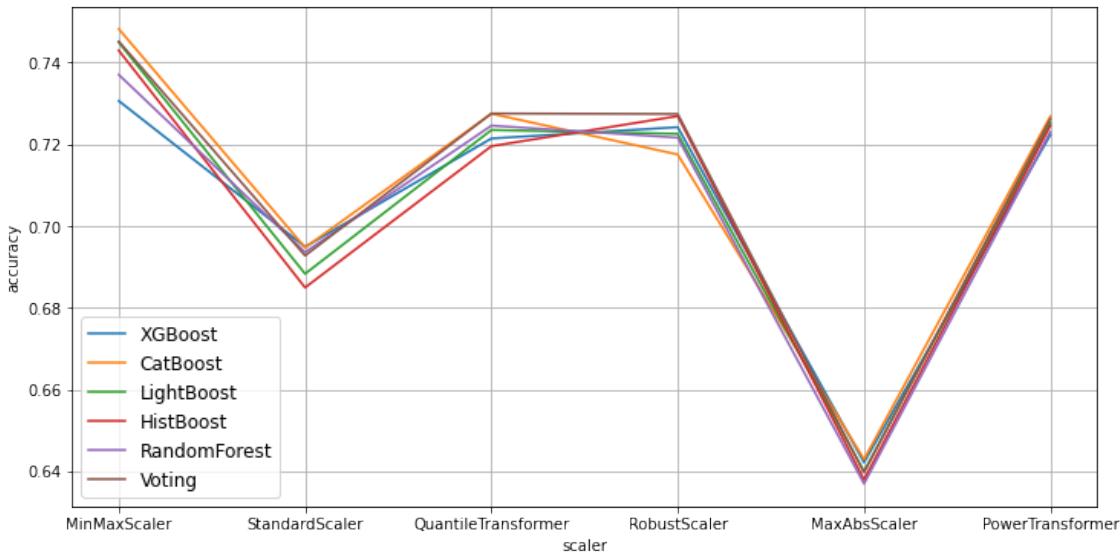


FIGURE 2.33 – Activité le jour : Évolution de l’accuracy en fonction des scalers pour chaque algorithmes d’ensemble learning

Les performances sont meilleurs sur la période nocturne qu’en journée avec des résultats fortement dégradés *MaxAbsScaler* en journée et la nuit.

### Choix du classifieur final

La combinaison *MinMaxScaler* et *CatBoostClassifier* ainsi que *LGBMClassifier* offrent de bons résultats et des temps de calculs maîtrisés. L’avantage du *MinMaxScaler* est qu’il est simple à calculer, le *LGBMClassifier* est rapide à calculer et présente de bons scores. Nous allons tenter d’optimiser cette combinaison de scaler et classifieur à l’aide de la librairie *Optuna* afin de sélectionner la meilleure combinaison d’hyper-paramètres.

## Entraînement et évaluation du classifieur final

Nous avons utilisé *Optuna* afin de trouver la meilleure combinaison possible pour les hyper-paramètres du *LGBMClassifier*. L'optimisation a été réalisée avec un mélange des données de la maison 1 et de la maison 2. Les jeux de données pour chacune des deux maisons ont été découpés en jeu d'entraînement (50% des jeux de données initiaux), de validation (25%) et de test (25%). Des objets scalers ont été ajustés sur les jeux d'entraînements correspondant à chacune des maisons. Nous avons ensuite mis à l'échelle chacun des jeux de données (entraînement maison 1/2, validation maison 1/2, test maison 1/2) avec l'objet scaler ajusté sur la maison correspondante. Les jeux d'entraînement et de validation préviennent d'une combinaison de points de données sélectionnés aléatoires dans chacune des maisons, puisque chaque point de données est traité indépendamment, sans tenir compte de la chronologie. Un classifieur avec une certaine combinaison d'hyper-paramètre est alors entraîné sur le jeu d'entraînement et les scores sont calculés sur les jeux de validation. Les valeurs des hyper-paramètres qui donnent le meilleur score sont alors retenues. On peut alors tester les performances de notre modèle optimisé sur les jeux de test de chacune des maisons.

## Performances du modèle

Les résultats obtenus avec notre classifieur optimisé sont les suivants :

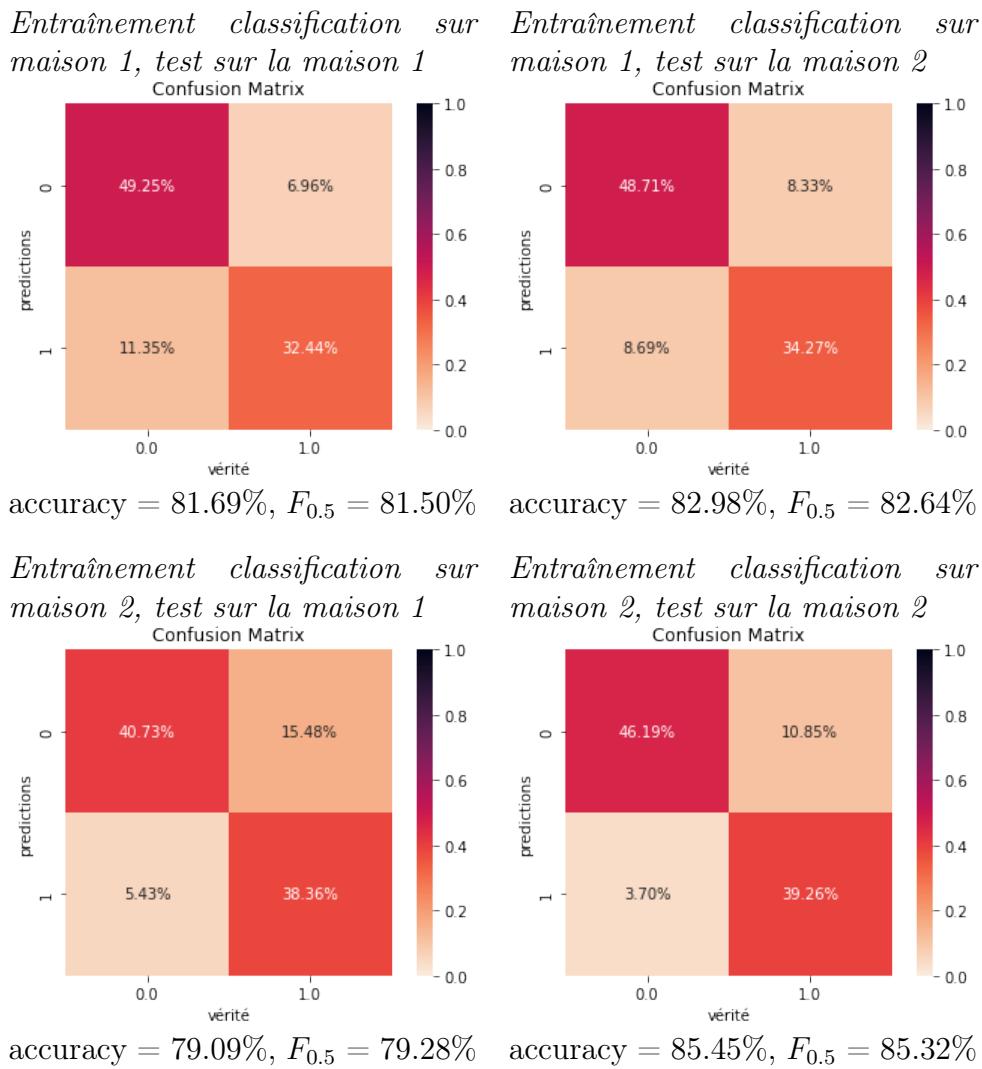


FIGURE 2.34 – Performance des classificateurs

Afin de mieux pouvoir apprécier nos résultats on peut observer l'histogramme du label activité prédict par tranche horaire. Nous avons tracé ces histogrammes pour les labels prédis par notre classifieur entraîné sur la maison 1.

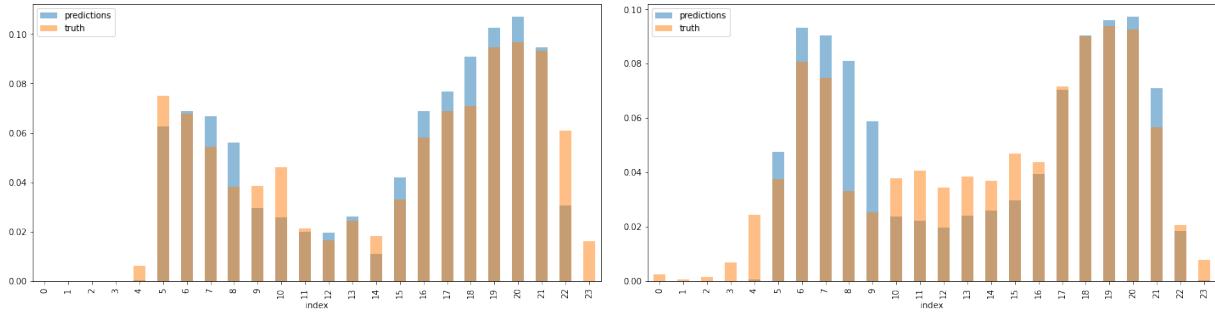


FIGURE 2.35 – Histogramme du label d’activité prédit en fonction de l’heure de la journée par notre classifieur entraîné sur la maison 1 en testant sur la maison 1 (gauche) et la maison 2 (droite)

Les distributions des labels prédis sur les maison 1 et maison 2 se rapprochent des distributions cibles.

Nous en avons fait de même pour les labels prédis par notre classifieur entraîné sur la 2.

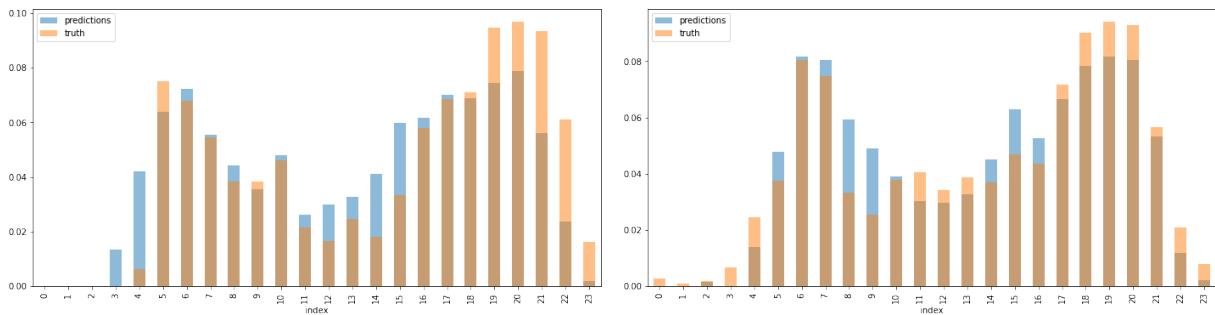


FIGURE 2.36 – Histogramme du label d’activité prédit en fonction de l’heure de la journée par notre classifieur entraîné sur la maison 2 en testant sur la maison 1 (gauche) et la maison 2 (droite)

Encore une fois on obtient des distributions pour les labels prédis qui sont assez proches des distributions cibles. On peut aussi observer les résultats de notre métrique d’évaluation avec notre modèle.



FIGURE 2.37 – Résultats de classification avec le classifieur entraîné sur la maison 1 et testé sur la maison 1

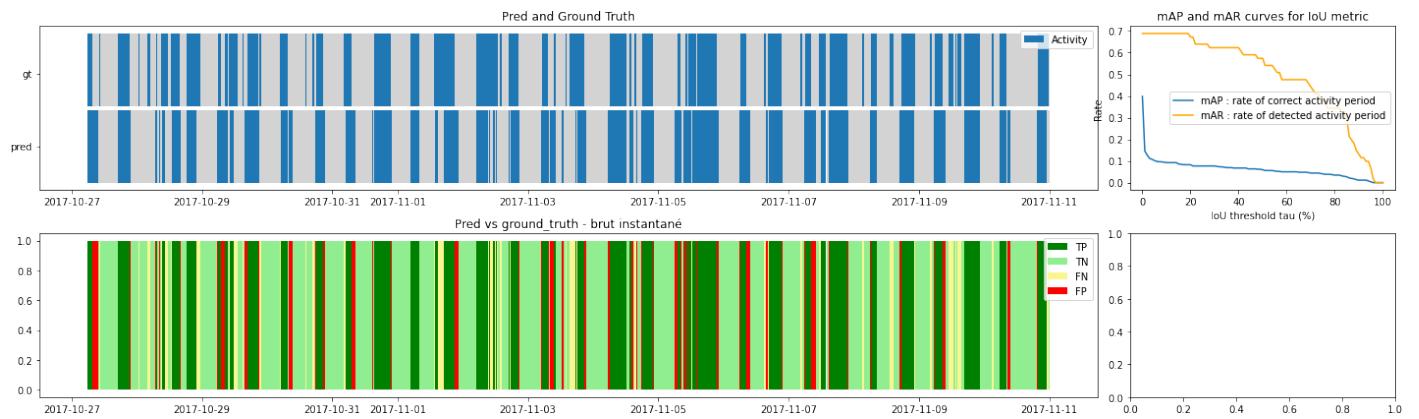


FIGURE 2.38 – Résultats de classification avec le classifieur entraîné sur la maison 1 et testé sur la maison 2

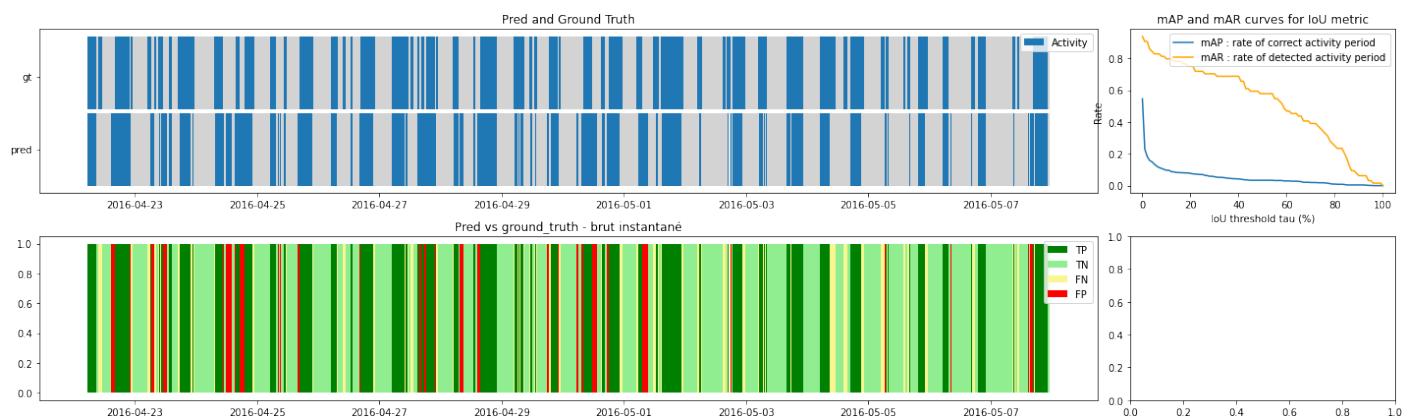


FIGURE 2.39 – Résultats de classification avec le classifieur entraîné sur la maison 2 et testé sur la maison 1

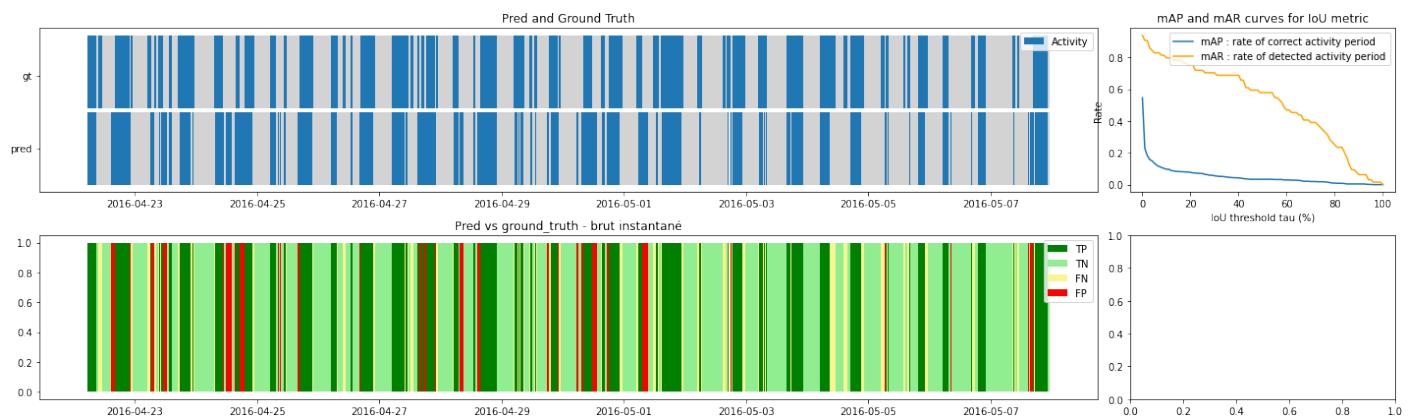


FIGURE 2.40 – Résultats de classification avec le classifieur entraîné sur la maison 2 et testé sur la maison 2

## 2.3.2 Approche 2 : Time2Vec

### 2.3.2.1 Principe du modèle

Une approche possible dans des situations comme la nôtre où nous disposons de beaucoup données non labellisées (il existe beaucoup de jeux de données sur des courbes de consommation) et de peu de données labellisées est de faire de l'apprentissage semi-supervisé. L'idée est d'apprendre une représentation utile des données sur une tâche prétexte non supervisée et d'utiliser cette représentation pour entraîner des algorithmes de façon supervisés de façon plus efficace. C'est par exemple le principe de Word2Vec en NLP. Un papier publié en 2019 par Seyed Mehran Kazemi et al. intitulé *Time2Vec: Learning a Vector Representation of Time* propose une approche similaire pour les séries temporelles. Le but est de pouvoir entraîner une représentation simple et universelle des séries temporelles sur une tâche prétexte : de la prédiction. L'expression mathématique de cette représentation est la suivante :

$$\mathbf{t2v}(\tau)[i] = \begin{cases} \omega_i\tau + \varphi_i, & \text{si } i = 0 \\ \mathcal{F}(\omega_i\tau + \varphi_i), & \text{si } 1 \leq i \leq k \end{cases}$$

Avec  $\tau$  un scalaire représentant le temps,  $\mathbf{t2v}(\tau)[i]$  représentant le  $i^{\text{ème}}$  élément du vecteur Time2Vec  $\mathbf{t2v}(\tau)$ ,  $\mathcal{F}$  est une fonction d'activation périodique (le papier utilise la fonction sinus, et c'est donc celle-ci que nous allons utiliser), et les  $\omega_i$  et  $\varphi_i$  sont des paramètres appris par un algorithme de descente de gradient. Cette représentation présente des propriétés telles que la capacité à capturer les motifs périodiques et non-périodiques des données ainsi que l'invariance aux changements d'échelle qui sont discutées et démontrées dans le papier original.

### 2.3.2.2 Architecture du modèle

Le but est de pouvoir entraîner des embeddings Time2Vec sur n'importe quel logement pour pouvoir avoir une représentation adaptée à chaque logement simplement à partir des relevés de puissance agrégées consommées dans celui-ci. Une fois ces embeddings obtenus on pourrait entraîner un réseau de neurones pour faire de la classification binaire à partir de notre jeu de données labellisé.

Pour l'entraînement des embeddings, nous avons dans un premier temps ré-échantillonné la courbe de puissance de chacun des logements avec un taux d'échantillonnage de 1 minute 30 afin de limiter le volume des données (les jeux de données originaux sont échantillonnés à la seconde). On réorganise le jeux de données en séquences de 60 minutes (40 observations, ce nombre a été choisi arbitrairement car cela donnait de bons résultats) avec un chevauchement de 32 observations sur chaque séquence. On fournit ensuite simplement une séquence à un réseau composé d'une couche d'embeddings Time2Vec suivi éventuellement d'une couche LSTM puis d'une couche fully connected, et on demande à notre réseau de reconstituer l'ensemble des points sur cette séquence en utilisant une loss MSE. Comme il y a un chevauchement sur les séquences prédites, on moyenne pour chaque point les prédictions faites par notre réseau.

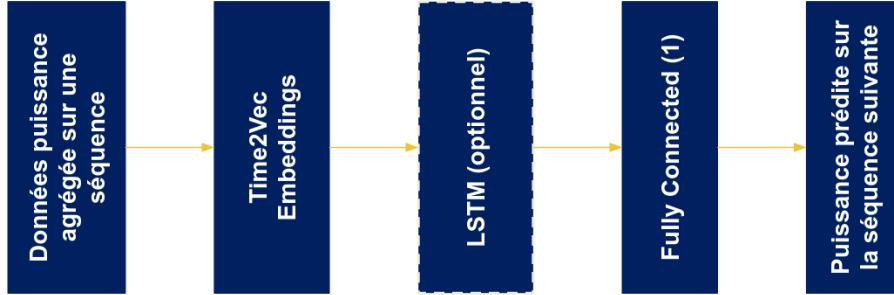


FIGURE 2.41 – Schéma de l'architecture pour l'entraînement des embeddings Time2Vec

Une fois les embeddings obtenus, on entraîne un nouveau réseau composé des embeddings pré-entraînés en fournissant une séquence de 60 minutes et en demandant au réseau de labelliser chacun des points de la séquence (1 pour activité, 0 sinon) avec une loss BCE. La dernière couche possède une fonction d'activation sigmoïde. Ici aussi il y a un chevauchement sur les séquences prédites, on moyenne donc pour chaque point les prédictions faites par notre réseau.

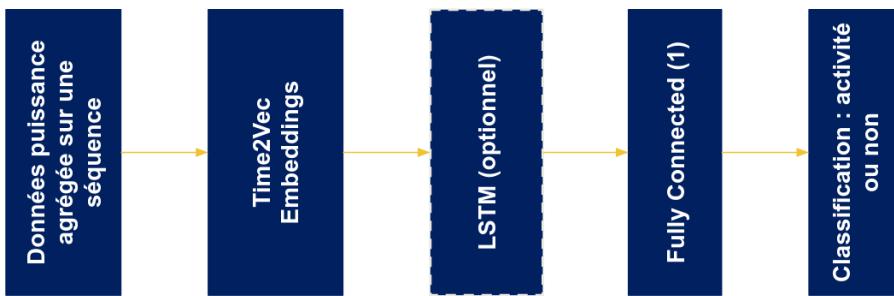


FIGURE 2.42 – Schéma de l'architecture pour l'utilisation de Time2Vec pour une tâche de classification

Voici le résumé généré par Tensorflow concernant l'architecture du modèle avec LSTM :

Model: "model"		
Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 40, 1)]	0
t2v (T2V)	(None, 40, 129)	5328
dropout (Dropout)	(None, 40, 129)	0
lstm (LSTM)	(None, 40, 32)	20736
dense (Dense)	(None, 40, 1)	33

Total params: 26,097
Trainable params: 26,097
Non-trainable params: 0

FIGURE 2.43 – Résumé de l'architecture du modèle Time2Vec avec LSTM

Le résumé pour le modèle sans LSTM est le même, sans la couche LSTM. Une couche d'activation de type *Leaky Relu* a été utilisée en sortie de LSTM pour l'entraînement des embeddings et pour l'entraînement du classifieur car cela donne de meilleurs résultats.

On peut de plus faire le choix de fine-tuner les embeddings Time2Vec durant la phase d'entraînement sur la classification ou non. Après plusieurs tests les résultats semblent peu varier avec ou sans fine-tuning, cependant les résultats semblent plus stables, plus répétables sans fine-tuning. Nous allons donc faire le choix de ne pas fine-tuner les embeddings pendant l'entraînement du réseau classifieur.

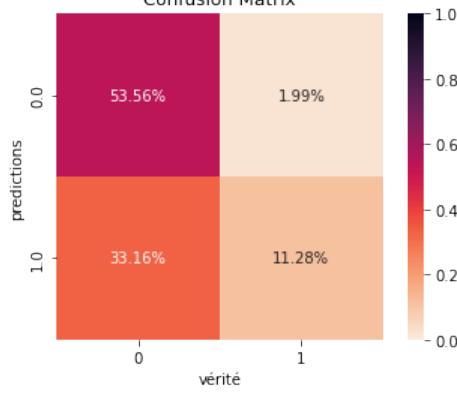
### 2.3.2.3 Entraînement et résultats du modèle

#### Résultats pour le modèle sans LSTM

Nous avons d'abord découpé les courbes de puissance agrégées des deux maisons en jeux d'entraînement et jeux de test (soit deux jeux par maison). Pour simplifier la démarche au moins dans un premier temps, les données des jeux de test d'une maison donnée ont été mis à l'échelle avec des objets scalers *MinMax* ajustés sur le jeu d'entraînement issu de la même maison (issu de la maison 1 pour la maison 1 et maison 2 pour la 2). Nous avons ensuite entraîné des embeddings Time2Vec sur chacun nos jeux d'entraînement et utilisé ces embeddings successivement pour entraîner un réseau pour une tâche de classification sur chacune des deux maisons et testé le modèle obtenu sur les deux jeux de test (8 combinaisons au total). Les résultats présentés dans cette section ont été obtenus sans fine-tuning des embeddings lors de la phase d'entraînement du classifieur, mais des résultats similaires (qui ne sont pas présentés ici) ont été obtenus avec fine-tuning.

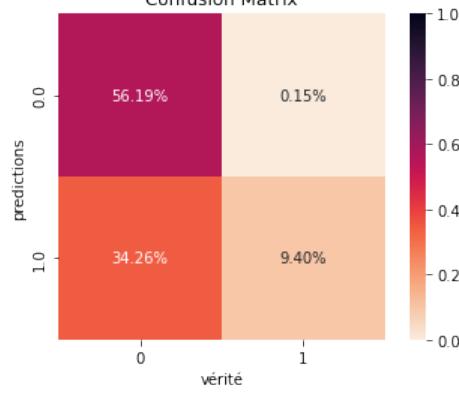
Les résultats obtenus avec les embeddings Time2Vec entraînés sur la maison 1 sont les suivants (sans LSTM) :

*Entraînement classification sur maison 1, test sur la maison 1*



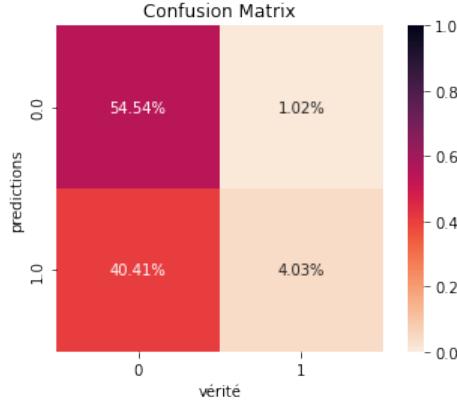
accuracy = 64.84%,  $F_{0.5} = 62.18\%$

*Entraînement classification sur maison 1, test sur la maison 2*



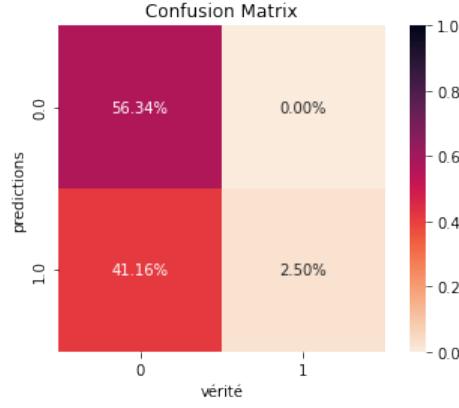
accuracy = 65.59%,  $F_{0.5} = 62.31\%$

*Entraînement classification sur maison 2, test sur la maison 1*



accuracy = 58.58%,  $F_{0.5} = 46.92\%$

*Entraînement classification sur maison 2, test sur la maison 2*

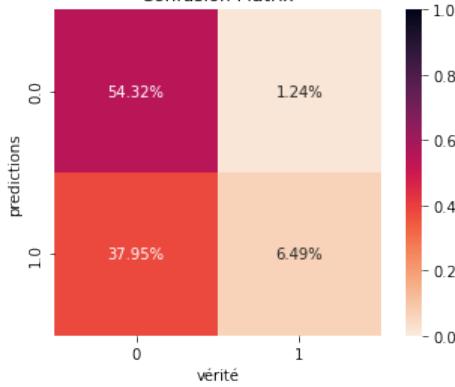


accuracy = 58.84%,  $F_{0.5} = 43.20\%$

FIGURE 2.44 – Performance des classificateurs entraînés avec les embeddings obtenus sur la maison 1

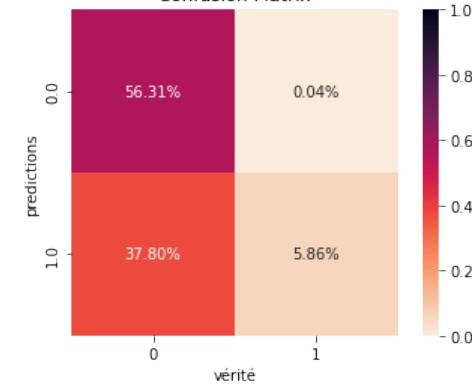
Les résultats obtenus avec les embeddings Time2Vec entraînés sur la maison 2 sont les suivants (sans LSTM) :

*Entraînement classification sur maison 1, test sur la maison 1*  
Confusion Matrix



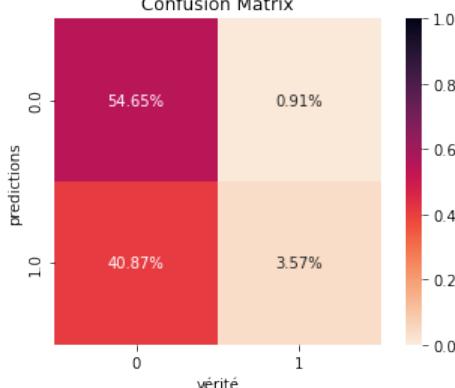
accuracy = 60.81%,  $F_{0.5} = 53.51\%$

*Entraînement classification sur maison 1, test sur la maison 2*  
Confusion Matrix



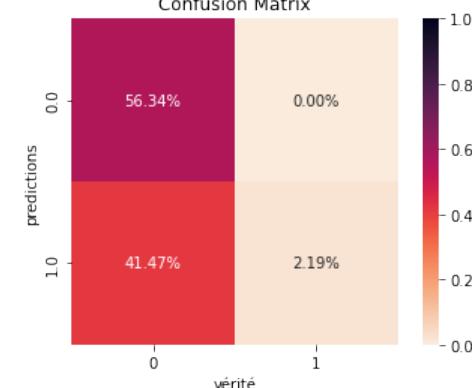
accuracy = 62.16%,  $F_{0.5} = 54.31\%$

*Entraînement classification sur maison 2, test sur la maison 1*  
Confusion Matrix



accuracy = 58.22%,  $F_{0.5} = 45.53\%$

*Entraînement classification sur maison 2, test sur la maison 2*  
Confusion Matrix



accuracy = 58.53%,  $F_{0.5} = 41.93\%$

FIGURE 2.45 – Performance des classificateurs entraînés avec les embeddings obtenus sur la maison 2

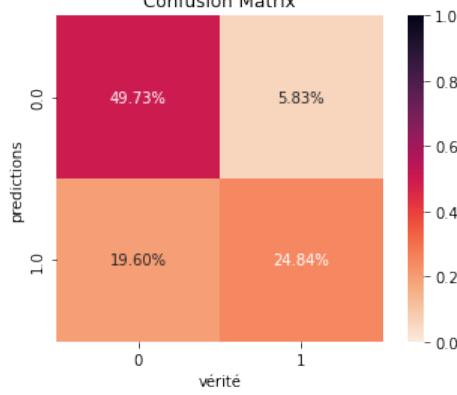
Il est intéressant de noter que les résultats sont les mêmes avec les embeddings obtenus par entraînement sur la maison 1 et ceux obtenus par entraînement sur la maison 2. Cela dit les résultats obtenus ne sont pas très bons pour les deux maisons.

### Résultats pour le modèle avec LSTM

Afin de tenter d'améliorer ces résultats on peut rajouter une couche LSTM en sortie des embeddings Time2Vec et avant l'entrée du fully connected. Cela permet de générer des features en entrée du fully connected qui ne tiennent compte qu'une partie de la séquence de façon automatique.

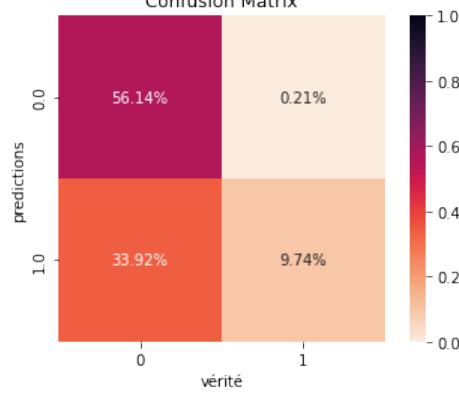
Les résultats obtenus avec les embeddings Time2Vec entraînés sur la maison 1 sont les suivants, avec l'ajout de la couche LSTM, toujours sans fine-tuning des embeddings :

*Entraînement classification sur maison 1, test sur la maison 1*



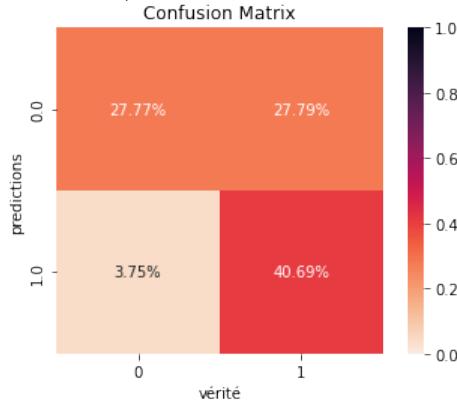
accuracy = 74.57%,  $F_{0.5} = 74.51\%$

*Entraînement classification sur maison 1, test sur la maison 2*



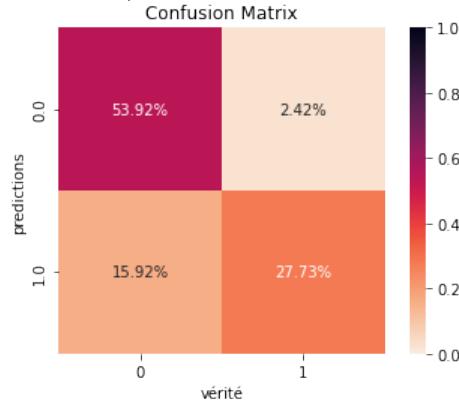
accuracy = 65.87%,  $F_{0.5} = 62.86\%$

*Entraînement classification sur maison 2, test sur la maison 1*



accuracy = 68.46%,  $F_{0.5} = 70.18\%$

*Entraînement classification sur maison 2, test sur la maison 2*

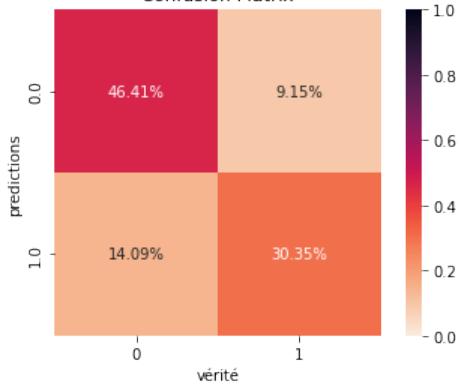


accuracy = 81.65%,  $F_{0.5} = 82.36\%$

FIGURE 2.46 – Performance des classificateurs entraînés avec les embeddings obtenus sur la maison 1

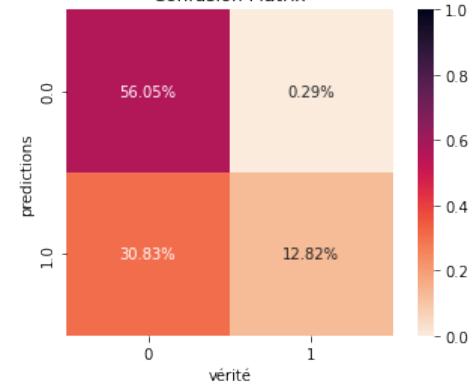
Les résultats obtenus avec les embeddings Time2Vec entraînés sur la maison 2 sont les suivants avec la couche LSTM :

*Entraînement classification sur maison 1, test sur la maison 1*  
Confusion Matrix



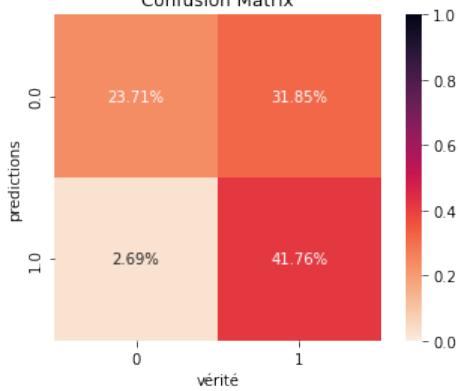
accuracy = 76.76%,  $F_{0.5} = 76.48\%$

*Entraînement classification sur maison 1, test sur la maison 2*  
Confusion Matrix



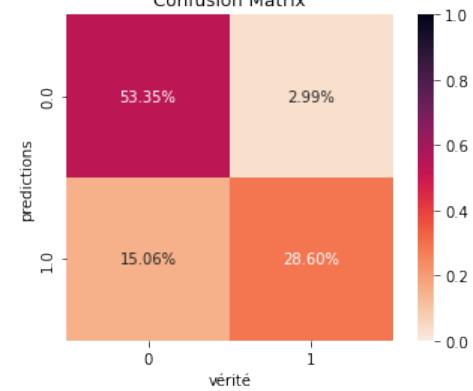
accuracy = 68.87%,  $F_{0.5} = 68.04\%$

*Entraînement classification sur maison 2, test sur la maison 1*  
Confusion Matrix



accuracy = 65.46%,  $F_{0.5} = 67.59\%$

*Entraînement classification sur maison 2, test sur la maison 2*  
Confusion Matrix



accuracy = 81.95%,  $F_{0.5} = 82.47\%$

FIGURE 2.47 – Performance des classificateurs entraînés avec les embeddings obtenus sur la maison 1

On peut observer que l'ajout du LSTM permet de fortement améliorer les résultats de classification. Cependant il est toujours difficile d'utiliser un classifieur entraîné avec une maison sur une autre, comme on peut le voir avec les résultats ci-dessus, notamment en entraînant le classifieur sur la maison 2 et en le testant sur la maison 1.

Encore une fois, afin de mieux pouvoir apprécier nos résultats on peut observer l'histogramme du label activité prédit par tranche horaire. Nous avons tracé ces histogrammes pour les labels prédits par notre réseau (avec LSTM) avec les embeddings Time2Vec entraînés sur la maison 1, et le classifieur entraîné sur la 1 puisque la maison sur laquelle nous entraînons les embeddings Time2Vec ne semble pas beaucoup influencer les résultats.

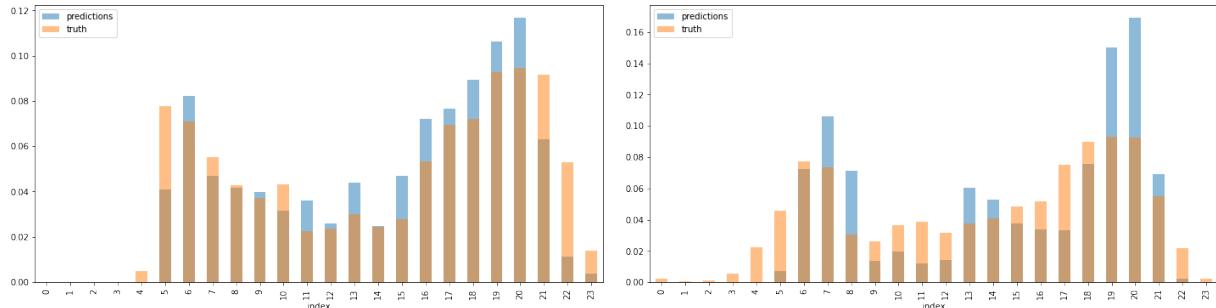


FIGURE 2.48 – Histogramme du label d’activité prédict en fonction de l’heure de la journée par notre réseau avec les embeddings Time2Vec avec LSTM entraînés sur la maison 1, et le classifieur entraîné sur la 1 en testant sur la maison 1 (gauche) et la maison 2 (droite)

Les distributions des labels prédits sur les maison 1 et maison 2 se rapprochent des distributions cibles. Nous en avons fait de même pour les labels prédits par notre réseau avec les embeddings Time2Vec entraînés sur la maison 1, et le classifieur entraîné sur la 2 (avec LSTM).

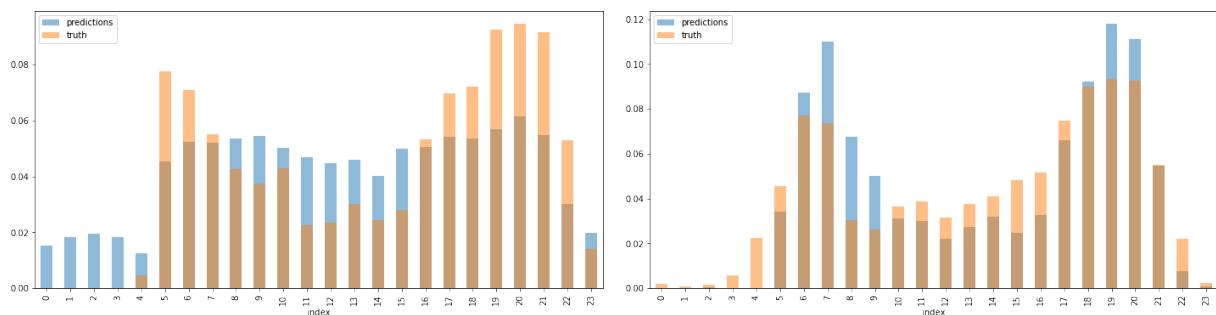


FIGURE 2.49 – Histogramme du label d’activité prédict en fonction de l’heure de la journée par notre réseau avec les embeddings Time2Vec avec LSTM entraînés sur la maison 1, et le classifieur entraîné sur la 2 en testant sur la maison 1 (gauche) et la maison 2 (droite)

On obtient également des distributions pour les labels prédits qui sont assez proches des distributions cibles hormis pour le test maison 2 vs 1. Le modèle semble prédire trop d’activité notamment le matin et l’après-midi, pendant les moments de pics d’activité.

Enfin, on peut observer les résultats de notre métrique d’évaluation avec notre modèle.

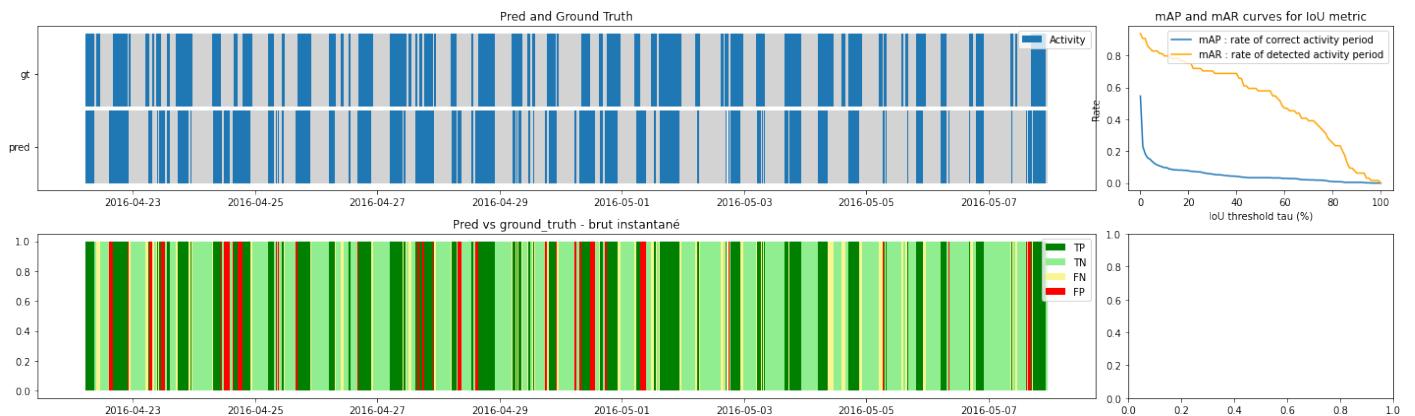


FIGURE 2.50 – Résultats de classification avec le classifieur t2v + LSTM entraîné sur la maison 1 et testé sur la maison 1

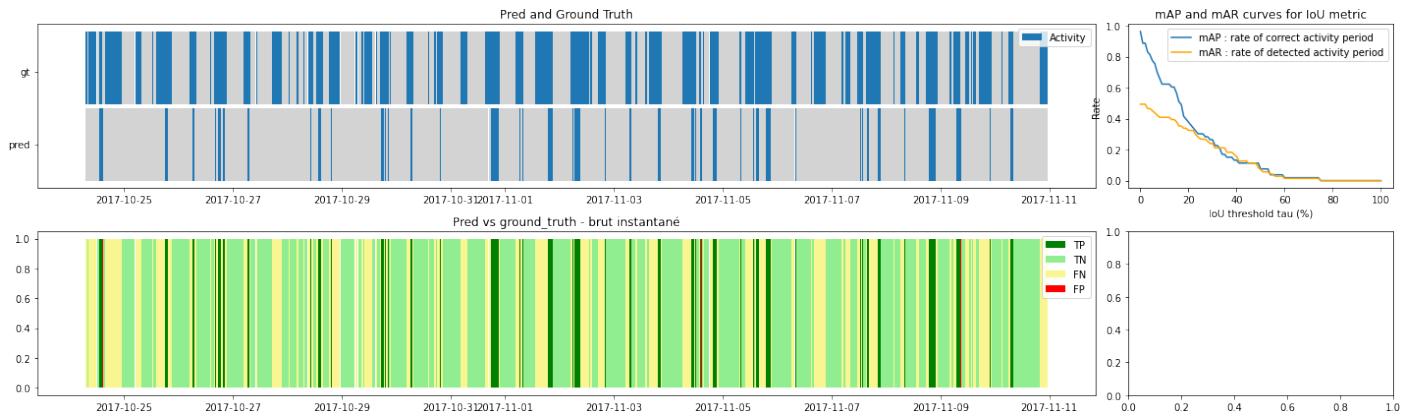


FIGURE 2.51 – Résultats de classification avec le classifieur t2v + LSTM entraîné sur la maison 1 et testé sur la maison 2

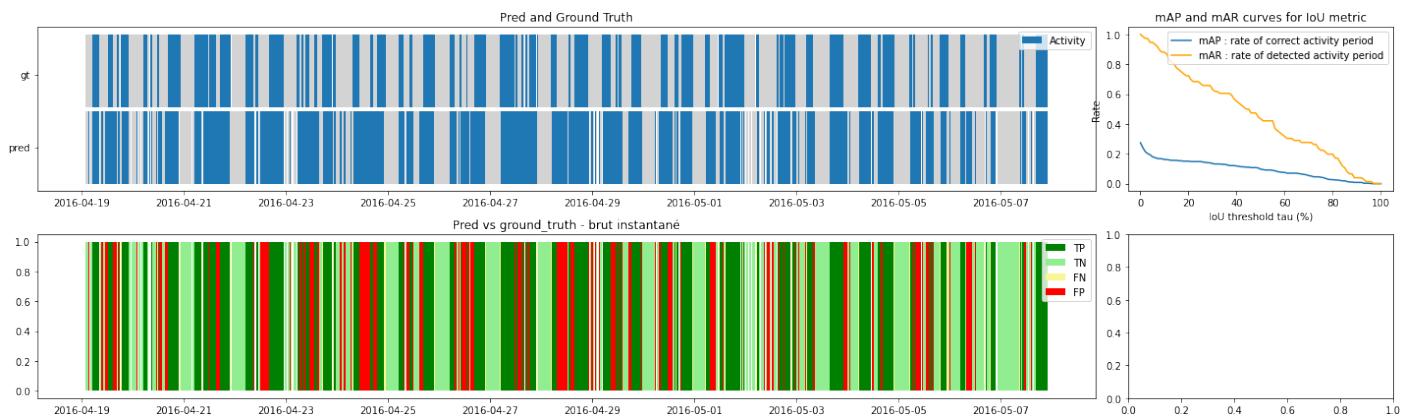


FIGURE 2.52 – Résultats de classification avec le classifieur entraîné t2v + LSTM sur la maison 2 et testé sur la maison 1

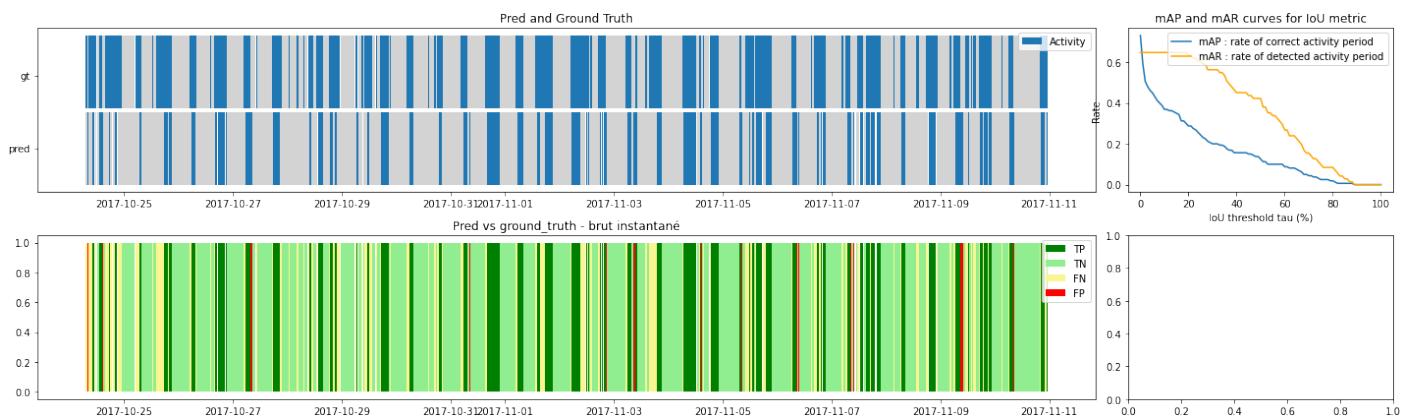


FIGURE 2.53 – Résultats de classification avec le classifieur entraîné t2v + LSTM sur la maison 2 et testé sur la maison 2

Il semblerait que les résultats soient sensibles à la façon dont les données sont mises à l'échelle, puisque nous avons pu constater que nous obtenons de meilleurs résultats en utilisant une fenêtre plus petite pour filtre l'objet scaler, résultat que nous peinons à expliquer. Il semblerait de plus en relançant plusieurs fois

l'entraînement que le modèle classifieur soit sensible à l'over-fitting. Lorsqu'il y a over-fitting du modèle on obtient de très bonnes performances sur la maison d'entraînement et des résultats très mauvais sur la maison de test. Pour bien comprendre ce qui peut se passer lors de l'entraînement on peut tracer les courbes de la fonction de perte (loss) en fonction des epochs. Nous avons ici enregistré la valeur de la fonction de perte sur le jeu d'entraînement mais aussi sur le jeu de test où le jeu d'entraînement correspond aux données d'entraînement de la maison 1 et le jeu de test, les données de test de la maison 2. Le but est de voir comment obtenir un modèle qui aura les meilleures performances possibles sur un autre logement que celui sur lequel il a été entraîné.

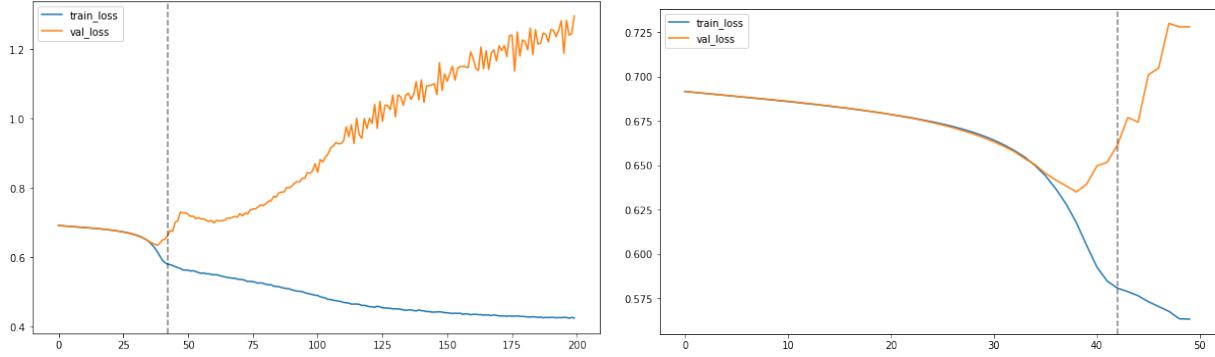


FIGURE 2.54 – Graphes de la valeur de la fonction de perte (loss) en fonction de l'epoch avec le jeu d'entraînement issu de la maison 1 et le jeu de test issu de la maison 2

Nous avons entraîné le classifieur sur 200 epochs pour observer le comportement de la fonction de perte, la courbe de gauche montre l'ensemble des 200 epochs et la courbe de droite correspond à la même courbe mais croupée à 50 epoch pour avoir une vision plus locale. On observe qu'entre 0 et 40 epoch la loss diminue sur le jeu d'entraînement et sur le jeu de test. Au delà de 40 epoch, la loss continue à diminuer sur le jeu d'entraînement mais augmente sur le jeu de test. On peut interpréter cela comme le fait que le modèle devienne de mieux en mieux entraîné mais de plus en plus spécifique à un jeu de données. Il faut donc trouver un compromis entre entraîner suffisamment le classifieur, mais pas trop pour qu'il reste généralisable et retourne de bonnes prédictions sur d'autres logements. Après avoir effectué plusieurs tests, il semblerait que le meilleur compromis soit autour de 40 à 50 epochs, ce qui a été représenté sur les graphes par les droites verticales. On observe que cela correspond à un coude de la courbe de la fonction perte sur le jeu d'entraînement. Une grosse difficulté est le fait qu'en conditions réelles nous n'avons pas accès aux labels de la maison de test, et il est donc impossible d'observer la loss sur ce logement. Il faut donc arrêter l'entraînement autour de 40 à 50 epochs et espérer ne pas avoir sur-entraîné le modèle. Cette règle empirique n'est valable que pour les hyper-paramètres utilisés lors de ces tests (notamment le learning rate et la taille de batch) et il faudrait la re-tester à chaque fois qu'une valeur est modifiée, ce qui impacte la répétabilité de cette méthode.

#### 2.3.2.4 Discussions de l'approche du modèle

Un des avantages des embeddings Time2Vec d'après leurs créateurs est de générer des features pertinentes automatiquement, ce qui est bien le cas puisque nous avons pu entraîner des classifieurs binaires avec de bonnes performances (surtout sur la maison 2) sans features engineering. Les embeddings obtenus sont de plus facilement transférables d'un logement à un autre sans dégrader les performances puisque nous avons vu que l'on peut utiliser les embeddings obtenus par entraînement sur la maison 1 ou 2 pour entraîner nos classifieurs sans que cela n'impacte trop les performances. On pourrait de plus envisager d'entraîner des embeddings spécialement pour chaque logement puisque cet entraînement est non supervisé, ou encore de faire du fine-tuning des embeddings sur le logement sur lequel on

souhaite déployer le modèle. C'est cette technique que nous allons tester dans la partie 2.4.5. Cela dit comme on pouvait s'y attendre on obtient de bien meilleurs résultats en entraînant un modèle de classifieur spécifique pour chaque logement, ce qui n'est pas envisageable si nous ne possédons pas de données labellisées. On se retrouve donc confronté à des problématiques similaires à celles liées à l'entraînement d'algorithme de classification supervisés avec la phase de features engineering en moins. Après plusieurs tests il semblerait de plus que cette méthode soit sensible à la façon dont les données sont mises à l'échelle, notamment lors de l'entraînement du modèle de classification sur une maison et en testant sur l'autre. Le scaler *MinMaxScaler* semble retourner les meilleurs résultats. Le choix des hyper-paramètres et notamment la taille de batch influence aussi les résultats obtenus. Enfin le modèle est sensible à l'over-fitting et il faut veiller à choisir un pas et un nombre d'epochs adaptés, ce qui peut se révéler délicat.

### 2.3.3 Approche 3 : Auto-Encodeur + Classifieur pour la détection d'anomalies

Cette seconde approche a pour but de travailler avec des données non labellisées de façon à détecter des anomalies concernant la consommation électrique d'un foyer.

De manière plus concrète, l'objectif est d'analyser la courbe de charge afin de modéliser le comportement normal dans les activités quotidiennes d'une maison.

Au moyen de la consommation des appareils, l'idée est de détecter les comportements anormaux qui induisent l'apparition d'éventuels problèmes dus à l'évolution du modèle de consommation.

Pour ce faire, nous avons fait le choix d'utiliser une architecture auto-encodeur pour la détection d'anomalies combiné à un classifieur pour mener à bien la tâche de classification des données en anomalie ou fonctionnement normal.

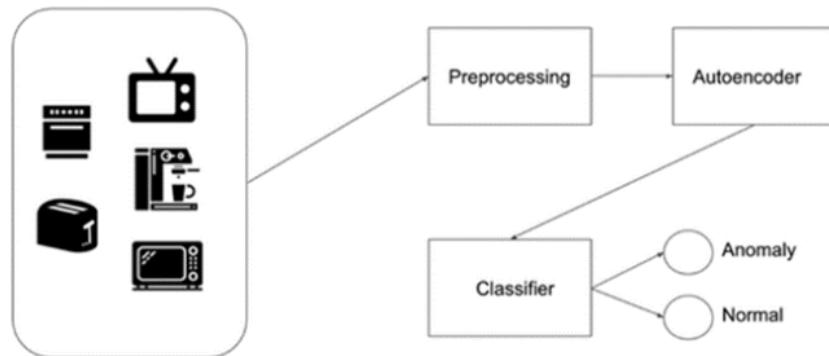


FIGURE 2.55 – Workflow de l'approche

#### 2.3.3.1 Pre-processing

##### Echantillonage

Nous échantillonnons la consommation d'électricité chaque heure, et analysons la façon dont elle varie au cours de la journée, ce qui constituent un apport de 24 entrée pour notre réseau.

##### Génération d'anomalies

L'idée à travers la génération artificielle d'anomalies, est de simuler le comportement des personnes âgées qui souffrent d'Alzheimer ou de démence et qui commencent à ressentir une activité nocturne et

un désordre dans leur routine quotidienne. Pour ce faire, nous avons générée des anomalies en décalant les données de 8 heures et en multipliant chaque mesure par une valeur aléatoire entre 0 et 2.

### 2.3.3.2 Architecture de l'auto-encodeur

L'architecture d'un auto-encodeur prend la forme d'un diabolo car elle possède un centre resserré avec des entrées et des sorties élargies. Grâce à cette architecture, l'encodeur est capable de représenter par exemple une image tout entière grâce à un minimum de points sur un plan 2D. Cette architecture constitue une étape préliminaire de traitement de données ce que l'on appelle l'extraction de caractéristiques. Généralement, le réseau de neurones attaque directement les données brutes pour réaliser par lui-même l'extraction de caractéristique. Plus précisément, les auto-encodeurs sont des modèles d'apprentissage non supervisé qui utilisent des variables latentes pour décrire et contenir l'information de chaque pixel d'une image.

Pour apprendre, l'auto-encodeur se base sur un modèle d'apprentissage non supervisé. Pour entraîner un auto-encodeur, on doit fournir à ce dernier une série d'exemples en entrée en lui demandant d'apprendre à reproduire ces exemples de manière la plus fidèle sur la couche de sortie (décodeur) tout en minimisant une erreur de reconstruction. Cette erreur est calculée par la fonction de coût d'erreur de reconstruction qui évalue la différence entre l'image originale et l'image reconstruite. L'encodeur se charge alors de compresser l'information et de la stocker dans le moins de variables latentes possibles. Le décodeur lui se chargera de reconstruire l'image du mieux qu'il peut. On peut remarquer tout de même un goulot d'étranglement au niveau de la couche dédiée aux variables latentes. En effet, le nombre de variables latentes constraint l'auto-encodeur à trouver une représentation parcimonieuse des données qui l'on été fournies en entrée.

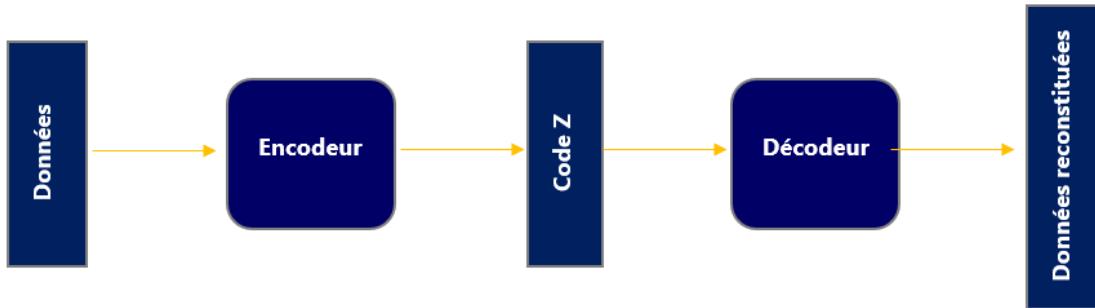


FIGURE 2.56 – Schéma d'un auto-encodeur

L'intérêt de l'auto-encodeur n'est pas dans la génération de la donnée de sortie mais dans l'effet induit d'apprendre une représentation condensée et représentative d'un ensemble de données (exemples) d'un espace à un grand nombre de dimensions vers un espace à peu de dimensions puis de reconstruire l'entrée à partir de l'information compressée.

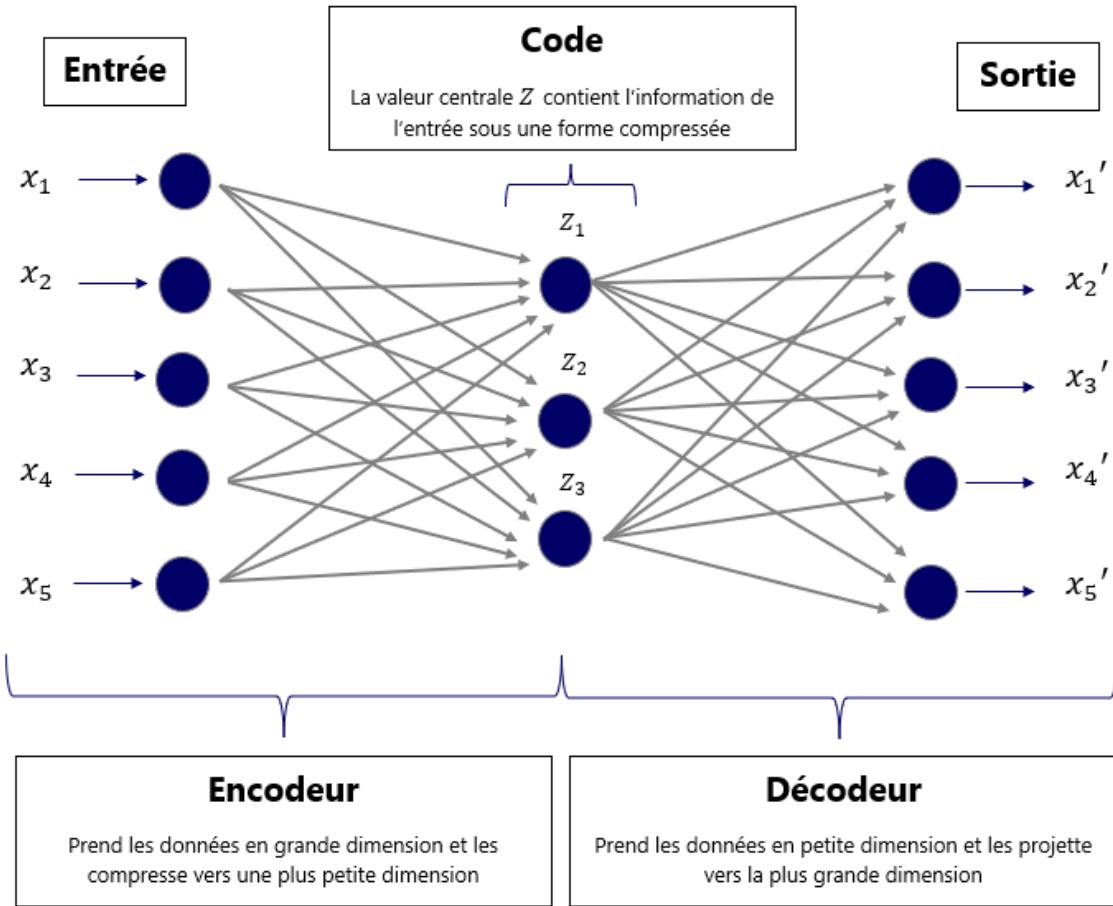


FIGURE 2.57 – Architecture diabolo simple avec une couche pour l'encodeur et le décodeur

- L'encodeur, a pour rôle de construire un espace de variables latentes. Cependant les différentes variables latentes que l'algorithme découvre ne peuvent pas être interprétables directement.
- Le décodeur, a pour rôle de reconstruire les données d'entrée à partir d'un point dans l'espace latent

Le choix de architecture de l'auto-codeur est crucial pour aboutir à de bonnes performances et permettre ainsi au décodeur d'aboutir à une présentation la plus fidèle possible des données fournies en entré du modèle.

Dans notre cas, nous travaillons sur la consommation quotidienne d'appareils échantillonnée 24 fois par jour. Nous utilisons alors un auto-codeur de 24 neurones d'entrée/sortie, et 2 neurones dans la couche intermédiaire pour réduire la complexité des données. En sortie de l'auto-codeur, un classifieur *Random-Forest* est utilisé pour classer les données reconstruites par le décodeur. L'architecture que nous avons retenu est la suivante :

- 24 neurones d'entrée/sortie car comme évoqué précédemment, les appareils sont échantillonnes 24 fois par jour
- 2 neurones dans la couche intermédiaire pour réduire la complexité des données.

Nous avons fait le choix d'utiliser un réseau avec une seule couche intermédiaire afin de prévenir le surajustement du modèle. Par ailleurs 2 neurones dans la couche cachée semblent être suffisants pour

cette tâche. Cette configuration permettra de diminuer le temps nécessaire pour la tâche de classification et aussi éviter les problèmes qui apparaissent dans les hautes dimensions. De plus, la configuration retenue facilite la visualisation et l'interprétation de l'espace latent.

Les hyperparamètres du réseau de l'auto-encodeur qui ont été choisi sont les suivants :

- $latentSize = 2$
- $windowLength = 24$
- $midActivation = 'sigmoid'$
- $outActivation = 'sigmoid'$

### 2.3.3.3 Entrainement et résultats du modèle $\{AE + Classifieur\}$

Les paramètres de l'entraînement sont les suivants :

- $epoch = 200$
- $batchSize = 32$
- $optimizer = Nadam$
- $aeLoss = categoricalCrossentropy$

Formule de la fonction de perte *categoricalCrossentropy*

$$\text{Loss} = - \sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i \quad (2.1)$$

La fonction de perte *categoricalCrossentropy* est une très bonne mesure de la façon dont deux distributions de probabilité discrètes se distinguent l'une de l'autre. Dans ce contexte,  $y_i$  est la probabilité que l'événement  $i$  se produise et la somme de tous les  $y_i$  est égale à 1, ce qui signifie qu'un seul événement peut se produire. Le signe moins garantit que la perte diminue lorsque les distributions se rapprochent l'une de l'autre.

### Visualisation de l'espace latent généré par le modèle $\{AE + RandomForest\}$

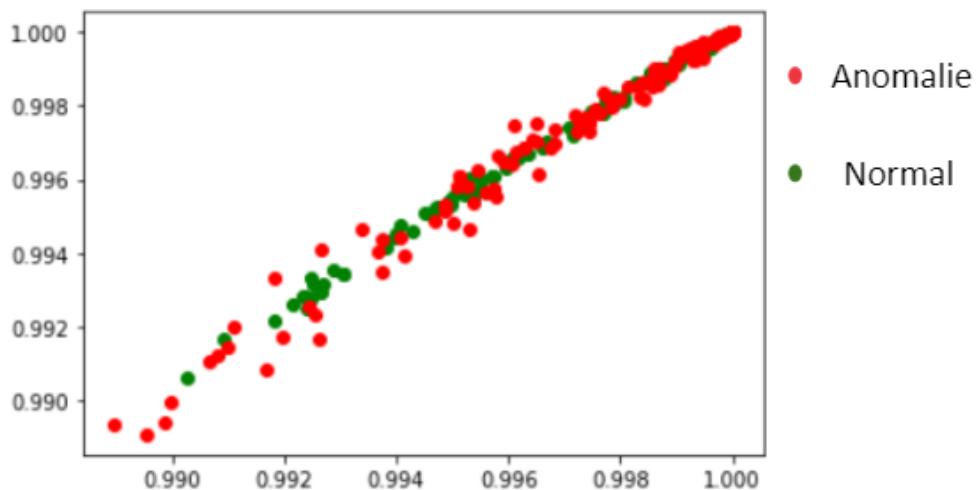


FIGURE 2.58 – Espace latent généré par le modèle  $\{AE + RandomForest\}$

## Les performances du modèle $\{AE + RandomForest\}$

```
----- Confusion matrix -----
[[88 63]
 [63 88]]
```

```
----- Classification Report -----
      precision    recall   f1-score   support
0         0.58     0.58     0.58     151
1         0.58     0.58     0.58     151

accuracy                           0.58     302
macro avg                         0.58     0.58     302
weighted avg                       0.58     0.58     302
```

FIGURE 2.59 – Rapport de classification pour le modèle  $\{AE + Classifieur\}$

### Interprétation des résultats

On remarque que les performances du modèle  $\{AE + RandomForest\}$  avec une précision de 0.58 ne sont pas très bonnes puisque le modèle prédit un petit mieux que l'aléatoire. Toutefois, les résultats peuvent être améliorés en essayant d'affiner les hyperparamètres associés au classifieurs grâce la fonction `GridSearchCV()` de python. Toutefois, nous nous attendons à obtenir de mauvaise performance avec l'utilisation modèle auto-encodeur simple.

#### 2.3.3.4 Entrainement et résultats du modèle $\{VAE + RandomForest\}$

Nous avons souhaité poursuivre le travail qui a été effectué auparavant avec le modèle auto-encodeur en réalisant un auto-encodeur variationnel car nous pensons qu'avec ce modèle nous pouvons obtenir des performances bien meilleures.

A noter que l'architecture de l'auto-encodeur variationnel suit la même architecture que celle utilisé pour l'auto-encodeur simple. A savoir, les couches d'activations, le nombre de couche qui compose l'encodeur et le décodeur ainsi que le nombre de neurones pour la couche de l'encodeur, décodeur et la couche intermédiaire sont identiques.

Il s'avère en pratique que les auto-encodeurs tels qu'ils ont été présentés précédemment fonctionnent mal. L'auto-encodeur variationnel est une amélioration de l'auto-encodeur et intègre 2 ajouts.

**1er Ajout :** Au lieu d'encoder chaque image par un point  $\lambda$  dans l'espace latent comme le fait l'auto-encodeur simple, un auto-encodeur variationnel choisi au hasard un point proche du point  $\lambda$ . Pour ce faire, l'auto-encodeur variationnel calcule une distribution de probabilité centrée sur  $\lambda$  et d'écart type  $\sigma$ , les deux étant appris par l'encodeur. Ainsi l'auto-encodeur variationnel comporte 2 modules distincts qui calculent la moyenne et la variance d'une distribution normale. Ce processus conduit à flouter l'image et constraint l'auto-encodeur variationnel à découvrir des encodages plus efficaces pouvant aboutir à une représentation plus fidèle que la représentation parcimonieuse apprise par un auto-encodeur simple.

**2ème Ajout :** L'ensemble des points associés à toutes les images de l'ensemble d'entraînement forment un nuage de points dans l'espace latent. On peut penser que la forme de ce nuage est arbitraire. Or pour forcer le nuage de point à avoir une forme simple (loi normale) l'auto-encodeur variationnel intègre en plus de la fonction de coût d'erreur de reconstruction, une fonction de coût qui corrige la forme du nuage si la distribution des points s'éloigne trop d'une loi normale standard.

Puisque nous modélisons la génération probabiliste de données, les réseaux de codeurs et de décodeurs sont probabilistes.

Par rapport à un auto-encodeur, l'auto-encodeur variationnel a l'avantage d'assurer une bonne correspondance entre une petite variation au niveau de l'espace latent et une variation au niveau de l'espace des données. Une analogie est l'approximation de surfaces terrestres à 3 dimensions dans des cartes à 2 dimensions.

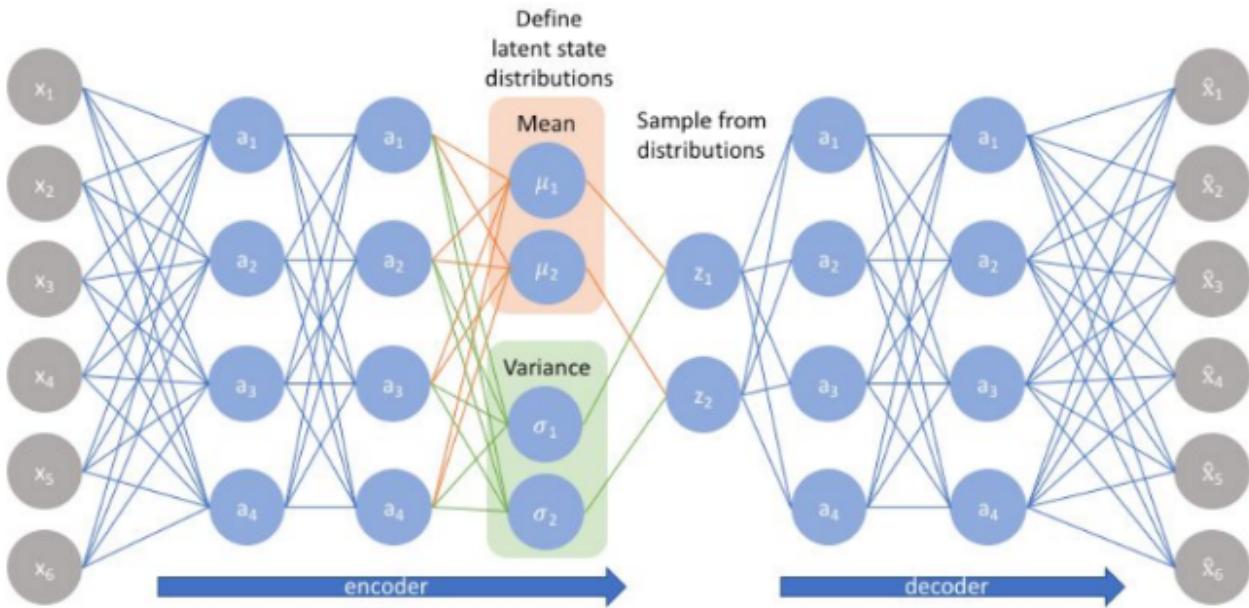


FIGURE 2.60 – Architecture d'un VAE

Les paramètres de l'entraînement sont les suivants :

- $epoch = 200$
- $batchSize = 32$
- $optimizer = Nadam$
- $vaeLoss = \text{reconstruction loss} + \beta \cdot KL \text{ loss}$  avec  $\beta \in \{1, 0.1, 0.01, 0.001, 0.0001, 0\}$

## Visualisation de l'espace latent généré par le modèle $\{VAE + RandomForest\}$

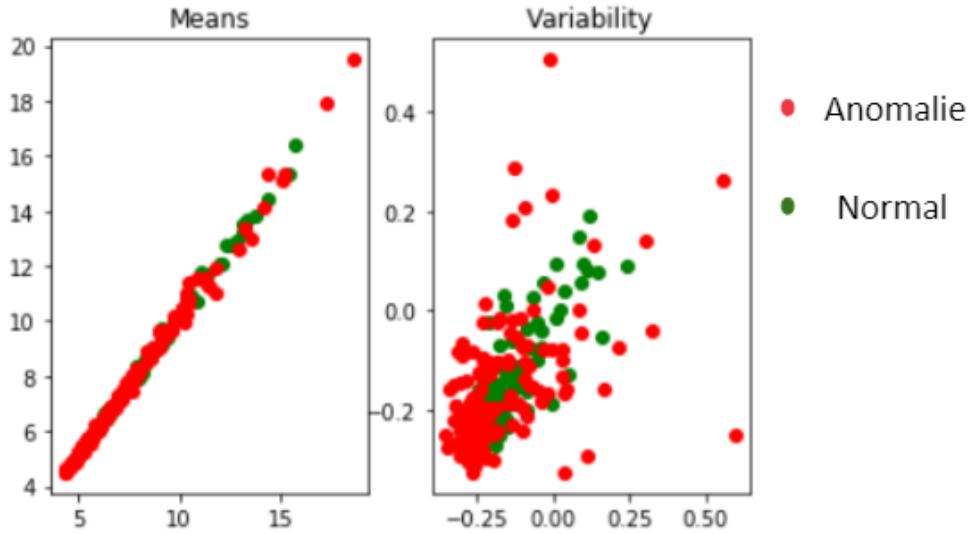


FIGURE 2.61 – Espace latent généré par le modèle  $\{VAE + RandomForest\}$

## Les performances du modèle $\{VAE + RandomForest\}$

		precision	recall	f1-score	support
0	120	0.73	0.79	0.76	151
1	31	0.78	0.71	0.74	151
		accuracy		0.75	302
		macro avg		0.75	302
		weighted avg		0.75	302

FIGURE 2.62 – Rapport de classification pour le modèle  $\{VAE + RandomForest\}$

## Interprétation des résultats

On remarque que les résultats obtenus par le modèle  $\{VAE + RandomForest\}$  sont bien meilleures que ceux obtenus par un auto-encodeur simple. Cela peut s'expliquer par le fait que VAE utilise un encodeur probabiliste pour modéliser la distribution des variables latentes plutôt que la variable latente elle-même. Avec un VAE la variance de l'espace latent peut être prise en compte dès la procédure d'échantillonnage. Cela élargit le pouvoir d'expression du VAE par rapport à l'AE simple dans la mesure où, même si les données normales et les données d'anomalie peuvent partager une valeur moyenne identique, la variance peut différer. Les données anormales auront probablement une plus grande variance et présenteront une plus faible probabilité de reconstruction. Puisque les mappings déterministes des auto-encodeurs simples peuvent être considérés comme une mappage à la valeur moyenne, l'AE simple n'a pas la capacité de gérer la variance des données. Enfin, comme pour l'auto-encodeur simple que nous avons implémenté précédemment, les performances peuvent être améliorées en affinant les hyperparamètres du classifieur à l'aide de la fonction python `GridSearchCV()`

### 2.3.3.5 Discussion de l'approche du modèle $\{Auto-Encodeur + RandomForest\}$

A travers la première approche nous avons évalué les performances des approches traditionnelles basé sur l'apprentissage statistique. En effet, les modèles de classifications basés sur l'apprentissage statistique tel que *Random Forest*, *k-NN* et méthodes ensemblistes tel que *Gradient Boosting* ont été testé sur la tâche de classification de périodes d'activités à partir de la courbe de charge électrique. Cependant, ces modèles étant des modèles supervisés, ils exigent que la courbe de charge totale soit labelisées. Pour mener à bien la phase de labelisation, des hypothèses fortes ont du être émises. Il en résulte, un degré d'incertitude quant à la véracité des périodes d'activité et d'inactivité labelisées. Par ailleurs cette même incertitude ce répercute automatiquement sur les prédictions du modèle. Finalement, nous avons notamment pu remarquer que cette approche n'est pas du tout généralisable dans le sens où les données de consommation peuvent être très différent selon la composition d'un foyer.

Avec l'approche auto-encodeur nous sommes capable d'apprendre le comportement normal de consommation électrique d'un foyer sans recourir au label d'activité. L'avantage de cette seconde approche est qu'elle aboutit à la construction d'un modèle disposant d'un pouvoir de généralisation bien plus grand que les modèles d'apprentissage statistique de l'approche n°1. Toutefois, cette approche présente des inconvénients. Premièrement, l'approche n'interprète pas la courbe de charge comme une série temporelle en tant que telle mais prend en entrée des valeurs de puissance échantillonnés chaque heure, donc l'information contenu au sein de la série temporelle ne peut pas être totalement capturé et pris en compte par le modèle. Deuxièmement, il nous est impossible de vérifier la pertinence des prédictions générées par le modèle dans des situations réelles car nous ne disposons pas d'un jeu de test labellisé pour l'activité. Enfin, le modèle auto-encodeur semble être difficilement transposable à notre cas d'usage à savoir la détection d'activité dans la mesure où les anomalies sont générées artificiellement. S'ajoute à cela le fait qu'il nous est impossible de vérifier la pertinence des prédictions générées par le modèle dans des situations réelles car nous ne disposons pas d'un jeu de test labellisé pour l'activité.

### 2.3.4 Approche 4 : *Auto-Encodeur Convolutionnel* pour la détection d'anomalies

A travers l'approche n°2 dans laquelle nous avons essayé les modèles auto-encodeur + Random Forest et auto-encodeur variationnel + Random Forest pour la détection d'anomalie nous n'utilisions pas la courbe de charge électrique comme un série temporelle en tant que telle. En effet, nous ne fournissons pas l'entièreté de la courbe de charge en entré du réseau de neurones de notre auto-encodeur mais seulement 24 valeurs correspondant à la consommation journalière échantillonnée toutes les heures.

L'objectif de cette approche n°4 n'est plus de fournir des données de consommation échantillonné mais plutôt d'analyser la courbe de charge agrégée d'un domicile comme une réelle série temporelle et d'y détecter automatiquement les anomalies (pics de surconsommation électrique) et ce sans recourir à un classifieur pour la tâche de classification des données (anomalie ou fonctionnement normal). Autrement dit, il s'agit ici d'une approche non supervisée.

Pour ce faire, nous utilisons un modèle *Auto-Encodeur Convolutionnel* de reconstruction pour déceler des anomalies dans des données de séries temporelles.

L'objectif recherché ici est de modéliser le comportement normale des activités quotidiennes d'un foyer. Autrement dit, l'idée principale qui se cache derrière la détection d'activité est d'utiliser les

caractéristiques de la consommation d'électricité qui se produisent uniquement dans les ménages d'une personne. Pour détecter une activité anormale, la consommation électrique de base (i.e courbe de charge de base) doit être identifiée préalablement. La charge de base fait référence à une charge générée par des sources de consommations d'électricité constantes comme un réfrigérateur, un congélateur, un appareil électronique branché en veille,...). En extrayant la charge de base de la consommation totale d'électricité, nous pouvons identifier l'activation de l'appareil par une seule personne.

Ainsi, l'encodeur se charge de trouver une représentation fidèle de la consommation normale des activités quotidienne d'une maison en y apprenant la courbe de consommation électrique de base c'est à dire la courbe de charge pour laquelle aucune activité est détectée (courbe de charge nocturne par exemple). Quant au décodeur, son objectif sera de reconstruire la courbe de consommation électrique normale apprise par l'encodeur.

Plus précisément, la courbe de charge de consommation normale (ou courbe de charge de base) correspond tout simplement, à la consommation totale des appareils électroniques en veilles à savoir les appareils qui sont continuellement en fonctionnement ou bien qui s'activent automatiquement à certaines périodes de la journée sans aucune intervention humaine, comme c'est le cas pour des appareils tel que le frigo, le réfrigérateur et les appareils de chauffage tel que la pompe à chaleur et la climatisation qui s'allument automatiquement par exemple pour maintenir une température souhaitée à l'intérieur d'une maison.

Pour obtenir cette courbe de charge de consommation normale synonyme d'inactivité dans un foyer, deux possibilités s'offrent à nous :

- Stratégie n°1 : Se référer à la courbe de charge nocturne de manière à capturer la consommation électrique totale de tous les appareils en veille ou en fonctionnement constant.
- Stratégie n°2 : Sélectionner individuellement la consommation électrique de chacun des appareils qui lorsqu'ils sont en fonctionnement nous permettent d'en déduire une activité au sein de la maison. Ces appareils en question pourraient être par exemple, les réfrigérateur, le frigo, la pompe à chaleur autrement dit tous les appareils qui sont constamment sous tension et ce même si le logement est inocupé.

Nous avons fait le choix de la stratégie n°1 car l'approche n°2 est moins généralisable car on ne connaît jamais à l'avance le nombre et le type d'appareils électriques utilisés au sein d'une maison.

Les grandes étapes que nous avons suivies pour la détection des anomalies dans les données en utilisant un *Auto-Encodeur Convolutionnel* sont les suivantes :

1. **Pre-processing** : Découper la courbe de charge de base qui se présente sous la forme d'une série temporelle en plusieurs séquences.
2. **Entraînement d'un Auto-Encodeur Convolutionnel** sur la courbe de charge de base. Ainsi, nous supposons qu'il n'y a pas d'activité et donc que les données sont considérées comme étant "normales".
3. **Reconstruction de la courbe de charge de base** : Utilisation l'*Auto-Encodeur Convolutionnel* pour reconstruire la courbe de charge de base.
4. **Calcul de l'erreur de reconstruction** : L'erreur sur les données correspond à la distance maximale entre la courbe de charge fournit à entrée du modèle et la courbe de charge reconstruite par le modèle (i.e perte maximale MSE).

5. **Prédiction des activités** : Si l'erreur de reconstruction pour les données de test est supérieure au seuil (*threshold*), nous étiquetons les points de données de la séquence en question comme une anomalie.
6. **Post-Processing** : Pour affiner nos prédictions, étant donné que plusieurs séquences utilisées pour l'entraînement du modèle peuvent se chevaucher, un point de données peut se trouver à la fois dans une séquence prédite comme étant une anomalie et inversement. Pour résoudre ce problème nous appliquons un vote majoritaire. Cela signifie que si le point de données en question se retrouve par exemple 3 fois dans une séquence prédite comme une anomalie et 2 fois dans une séquence prédite comme normale alors ce point de données sera annoté comme une anomalie.

#### 2.3.4.1 Pre-processing

Le Pre-processing est une étape crucial dans notre approche sans quoi l'entraînement de l'*Auto-Encodeur Convolutionnel* ne sera pas possible. Comme évoqué précédemment, le rôle de la phase de pre-processing est de découper la courbe de charge agrégé en plusieurs séquences. Toutefois avant de se lancer dans la construction des séquences, des étapes transformations préalables doivent être opérées sur les données.

Plus précisément, les étapes de pré-processing sont les suivantes :

1. Lecture et re-échantillonage de la courbe de charge
2. Séparation des données de consommation électrique en un jeu de données d'entraînement et jeux de données de test. Les dimensions du jeu de test est de choisi par l'utilisateur
3. Normalisation des données d'entraînement et de test
4. A partir du jeu de données d'entraînement, construction d'un jeu de données qui correspond à la courbe de charge de consommation de base. Pour cela, l'utilisateur définit la plage horaire des heures creuses. Bien souvent cette plage horaire correspond à la nuit.
5. Création de séquences d'entraînement et de test. La longueur des séquences et le taux de chevauchement entre les séquences sont définis par l'utilisateur.

En amont de la phase de pré-processing, nous avons décidé de laisser à l'utilisateur le choix de déterminer certains paramètres qui régissent la création des séquences d'entraînement à partir de la courbe de charge de base. Cela permet alors au modèle de gagner en flexibilité mais aussi de pouvoir s'adapter au comportement de la consommation électrique quotidienne d'un foyer. De cette façon, le modèle bénéficie d'un pouvoir de généralisation plus élevé puisque ce dernier adapte la construction de la courbe de charge de base à chacun des foyer dont il doit détecter l'activité. Or il s'avère bien souvent qu'en fonction de la composition du foyer, les routines et habitude quotidienne peuvent varier très fortement. Par exemple, une personne qui pratique les postes de nuit n'a pas la même routine qu'une personne exerçant une fonction en tant que cadre. Grâce à la flexibilité offerte par le modèle durant la phase de pre-processing, ces spécificités peuvent être pris en compte.

#### Choix des paramètres d'entrée

Ainsi, l'utilisateur doit choisir les valeurs des paramètres suivants :

- *timeStep* : durée d'une étape dans l'ensemble de données de re-échantillonnage (initialement 1 seconde)
- *durationTime* : durée d'une séquence
- *overlapPeriodPercent* : taux de chevauchement des données entre chaque séquence

- *timeframes* : plage horaire des heures creuses
  - *randomdays* : pour sélectionner des jours aléatoires (19h->18h59 du lendemain)
- Construction du jeu d'entraînement et de test**

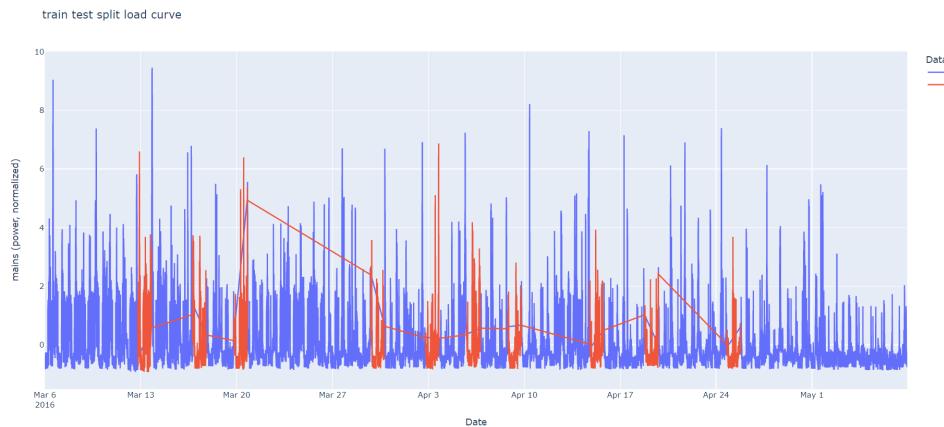


FIGURE 2.63 – Découpage de la courbe de charge en train, test

### Courbe de charge d'entraînement

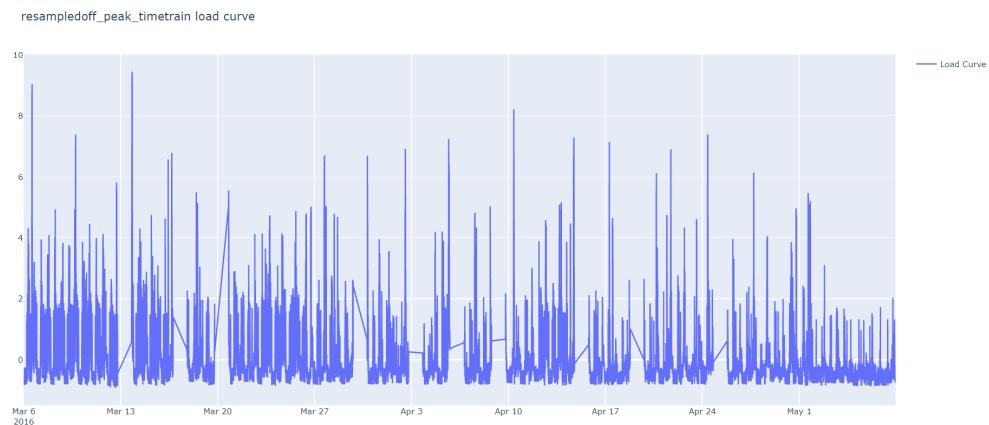


FIGURE 2.64 – Découpage de charge d'entraînement

### Courbe de charge de test



FIGURE 2.65 – Courbe de charge de test

## Construction des séquences

Sur la base des valeurs entrées par l'utilisateur, nous avons défini une fonction `convertToSequenceParameters()` qui prend en argument l'ensemble des paramètres expliqués précédemment afin de déterminer la longueur de la séquence optimale ainsi que la période de chevauchement des données entre chacune des séquences.

On comprend bien que la longueur de la séquence ainsi que la période de chevauchement des données sont cruciaux afin de pouvoir capturer l'ensemble des fluctuations rapides ou lentes de la courbe de charge de base. En effet, une activité au sein d'une maison peut aussi bien être de très courte durée (quelques minutes) que d'une durée plus longue s'étalant sur plusieurs heures.

La phase de pré-processing retourne alors un tableau en **3 dimensions** [*samples*, *sequenceLength*, *features*]. Il s'agit de la structure de données requis pour pouvoir être ingéré par le réseau convolutionnel de l'auto-encodeur.

- *samples* : Contient l'ensemble des échantillons de la courbe de charge qui ont été découpé en séquence temporelle
- *sequenceLength* : Longueur de la séquence temporelle
- *features* : Contient la valeur de puissance de consommation électrique à un instant  $T$

A noter que par exemple, si nous souhaitons que notre réseau ait une mémoire de **10 jours** alors, nous définissons *sequenceLength* = 10.

En conclusion, tout l'intérêt du preprocessing est de s'adapter au mieux à la routine quotidienne d'une famille et ce quelque soit sa composition. C'est pour cela que nous avons souhaité rendre la phase de preprocessing la plus flexible et paramétrable possible en laissant le choix à l'utilisateur de définir un certains nombre de paramètres qui sont déterminantes pour la construction des séquences de la courbe de charge de base et qui doit refléter le plus fidèlement possible à la fois la configuration du foyer et la routine quotidienne de la famille.

## Visualisation d'une séquence de la courbe de charge de base (Courbe de charge nocturne)

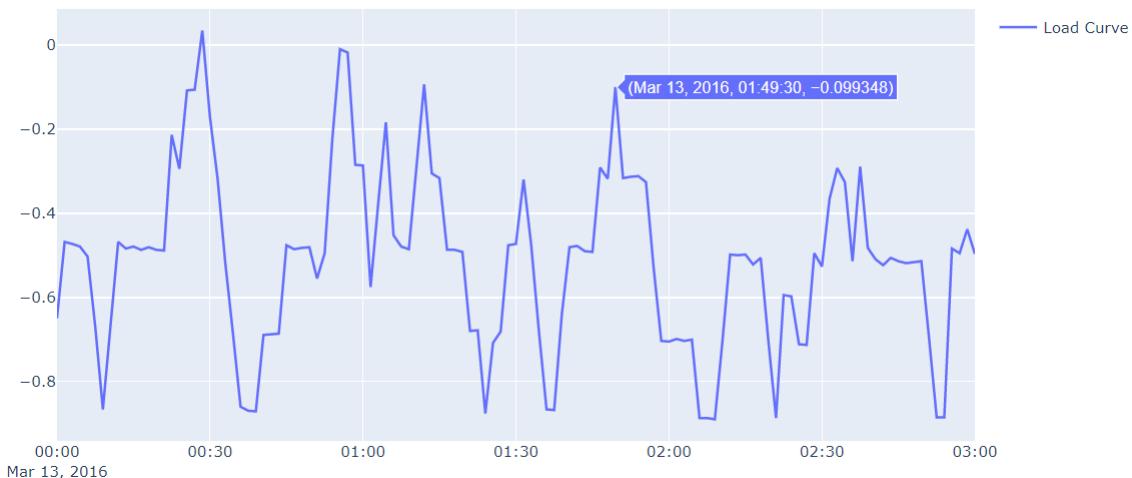


FIGURE 2.66 – Séquence de la courbe de charge de base

#### 2.3.4.2 Architecture du modèle

Pour détecter les activités au sein d'une maison, nous avons décidé d'utiliser un auto-codeur convolutionnel. Le modèle prend en entrée une séquence de la dimension suivante ( $batchSize$ ,  $sequenceLength$ ,  $numFeatures$ ) et retourne une sortie de la même forme. Dans notre cas,  $numFeatures$  sera toujours égale à 1 car nous avons besoin d'une seule features à savoir la puissance consommée par le foyer à l'instant  $T$ . Enfin,  $sequenceLength$  prend la valeur qui a été définie lors de la phase de pré-processing.

En ce concerne l'architecture de l'*Auto-Encodeur Convolutionnel*, cette dernière est composée de 2 couches de convolution ainsi que 3 couches de couches de dé-convolution. Entre chaque couche un dropout est appliqué pour prévenir le risque de sur-ajustement du modèle.

Voici le résumé généré par Tensorflow concernant l'architecture du modèle *Auto-Encodeur Convolutionnel* :

Model: "sequential"		
Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 20, 32)	256
dropout (Dropout)	(None, 20, 32)	0
conv1d_1 (Conv1D)	(None, 10, 16)	3600
conv1d_transpose (Conv1DTra nspose)	(None, 20, 16)	1808
dropout_1 (Dropout)	(None, 20, 16)	0
conv1d_transpose_1 (Conv1DT ranspose)	(None, 40, 32)	3616
conv1d_transpose_2 (Conv1DT ranspose)	(None, 40, 1)	129

---

Total params: 9,409
Trainable params: 9,409
Non-trainable params: 0

---

FIGURE 2.67 – Architecture de l'AE convolutionnel par Tensorflow

### 2.3.4.3 Entraînement du modèle

Concernant l'entraînement du modèle *Auto-Encodeur Convolutionnel*, il est important de noter que puisque l'objectif de l'auto-encodeur est de reconstruire la courbe de charge de base fournie en entrée, nous avons utilisé le jeu de données d'entraînement  $X_{train}$  à la fois comme entrée et comme variable cible puisqu'il s'agit dans notre cas d'un mode de reconstruction.

Les hyperparamètres de l'entraînement sont les suivants :

- $epoch = 50$
- $batchSize = 128$
- $optimizer = Adam$
- $aeLoss = mse$

Techniquement, l'objectif recherché est de minimiser l'erreur de reconstruction en fonction d'une fonction de perte. La fonction de perte que nous avons choisi est l'erreur quadratique moyenne (Mean Squared Error). En effet nous voulons davantage pénaliser les valeurs aberrantes étant donné que nous souhaitons apprendre la courbe de consommation de base.

#### MSE (Mean Squared Error)

$$\sum_{i=1}^D (x_i - y_i)^2$$

#### Évolution de la fonction de perte

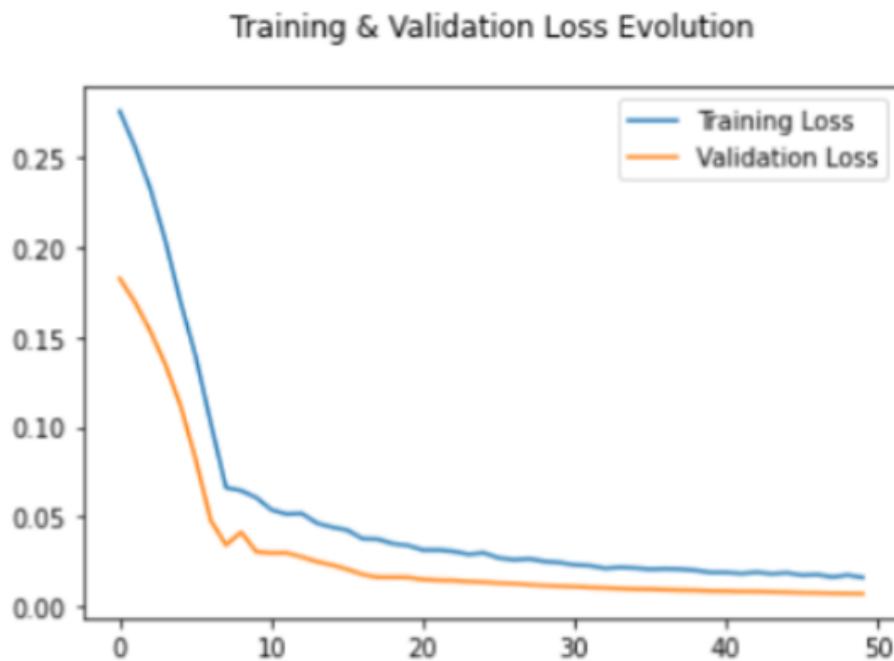


FIGURE 2.68 – Evolution de la fonction de perte

#### Visualisation de la reconstruction de la courbe de charge base

Voyons comment notre modèle a reconstruit le premier échantillon. Il s'agit des **40** pas de temps du **jour 1** de notre jeu de données d'entraînement.

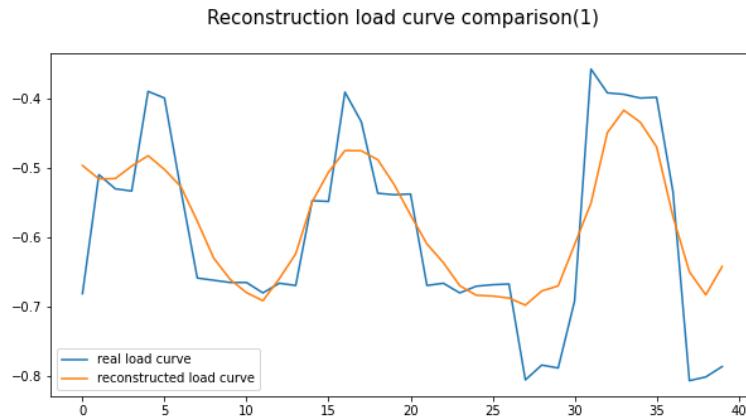


FIGURE 2.69 – Reconstruction de la courbe de charge de base par l’AE

#### 2.3.4.4 Détection des anomalies

A ce stade, le modèle à appris à reconstruire la courbe de charge de base. Cela signifie que le modèle a appris à modéliser le comportement normal des activités quotidiennes d'un foyer. Sur cette base, la détection des anomalies peut être réalisée en évaluant tout simplement les écarts entre la courbe de charge de base apprise par le modèle et la courbe de charge quotidienne d'un foyer qui comprend des pics d'activité (i.e surconsommation électrique). Cet écart est calculé avec la fonction de perte *Mean Absolute Error*. Nous avons choisi cette fonction de perte car elle est plus robuste aux données aberrantes ce qui correspond dans notre cas aux anomalies. Sur les différents histogrammes, on voit l'évolution de cet écart pour le jeu d'entraînement et le jeu de test.

Ainsi, pour détecter les anomalies, nous procédons de la manière suivante :

1. Calcul de la perte *MAE* sur les échantillons d'entraînement.
2. Identification de la valeur maximale de perte *MAE*. C'est la plus mauvaise performance de notre modèle en essayant de reconstruire un échantillon. Nous en ferons le **seuil** (i.e *threshold*) pour la détection des anomalies.
3. Si la perte de reconstruction d'un échantillon est supérieure à cette valeur **seuil**, alors nous pouvons en déduire que le modèle fait face à un comportement qui ne lui est pas familier. Nous allons étiqueter cet échantillon comme une **anomalie**

## MAE (Mean Absolute Error)

$$\sum_{i=1}^D |x_i - y_i|$$

Évolution de la perte  $MAE$  sur l'ensemble de jeu d'entraînement

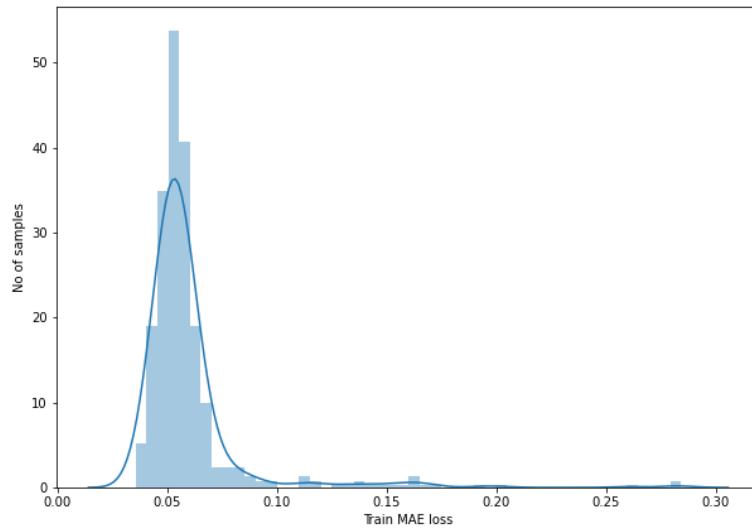


FIGURE 2.70 – Évolution de la fonction de perte  $MAE$  sur  $X_{train}$

Évolution de la perte  $MAE$  sur l'ensemble de jeu de test

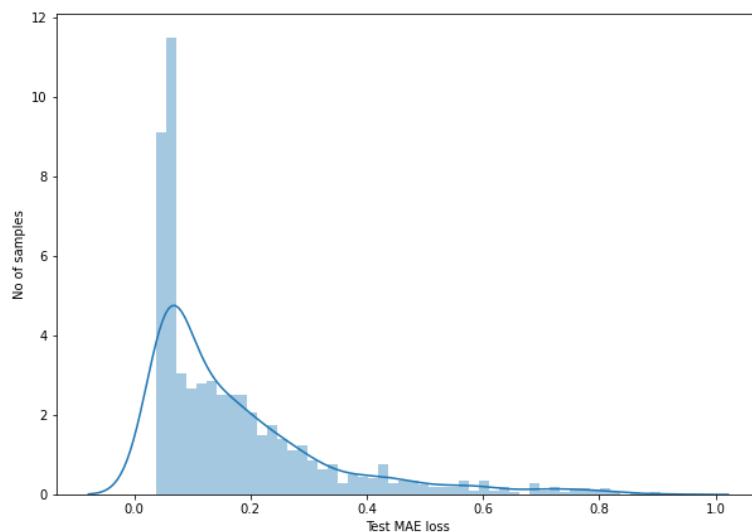


FIGURE 2.71 – Évolution de la fonction de perte  $MAE$  sur  $X_{test}$

#### 2.3.4.6 Post-Processing

Comme évoqué précédemment, la phase de post-processing a pour objectif d'affiner les prédictions. En effet, étant donné que les séquences se chevauchent, il arrive parfois qu'un point de donnée se trouve à la fois dans une séquence prédicté par le modèle comme une anomalie et une séquence prédicté comme un fonctionnement normal. Par conséquent on se heurte ici à un dilemme. Le point de données en question, est-il finalement réellement une anomalie ou non. Pour pallier ce problème le post-processing réalise un vote majoritaire pour chaque point de données se trouvant dans cette situation. Par conséquent si ce dernier apparaît davantage dans des séquences prédicté comme anomalie alors, le point de donnée en question sera étiqueté comme une anomalie et inversement. Bien évidemment pour éviter les cas des égalités parfaites, nous avons un nombre de chevauchement de séquences impair fixé préalablement lors de la phase de pré-processing.

	Timestamp	list_idx_sequence_no_activity	list_idx_sequence_activity	nb_no_activity	nb_activity	total	method_prediction_1
868	2016-04-26 06:30:00	[]	[104, 105, 106, 107, 108]	0	5	5	1
869	2016-04-26 06:31:30	[]	[104, 105, 106, 107, 108]	0	5	5	1
870	2016-04-26 06:33:00	[]	[104, 105, 106, 107, 108]	0	5	5	1
871	2016-04-26 06:34:30	[]	[104, 105, 106, 107, 108]	0	5	5	1
872	2016-04-26 06:36:00	[109]	[105, 106, 107, 108]	1	4	5	1
873	2016-04-26 06:37:30	[109]	[105, 106, 107, 108]	1	4	5	1
874	2016-04-26 06:39:00	[109]	[105, 106, 107, 108]	1	4	5	1
875	2016-04-26 06:40:30	[109]	[105, 106, 107, 108]	1	4	5	1

FIGURE 2.72 – Anomalies détectées après le post processing

Le but du pré-processing est d'affiner les prédictions du modèle car il arrive parfois que un point de donnée se trouve à la fois dans une séquence prédicté comme étant une anomalie et inversement.

Par exemple, sur l'image ci-dessus le point de données encadré en rouge correspond à une donnée de consommation à la date du 26 avril 2016 à 06 :36 :00. Ce point de données appartient à plusieurs séquences qui se chevauchent à savoir la séquence n° 109 qui est prédicté en non activité et les séquences n° 105, 106, 107 et 108 qui sont prédictes comme étant des activités. Ainsi par vote majoritaire, le post-processing prédit une activité pour cette date. Cela doit correspondre semble correspond à l'heure où la personne se lève le matin avant de se rendre au travail et prend une douche ou bien encore son petit déjeuner.

Ainsi le résultat du post-processing est le suivant

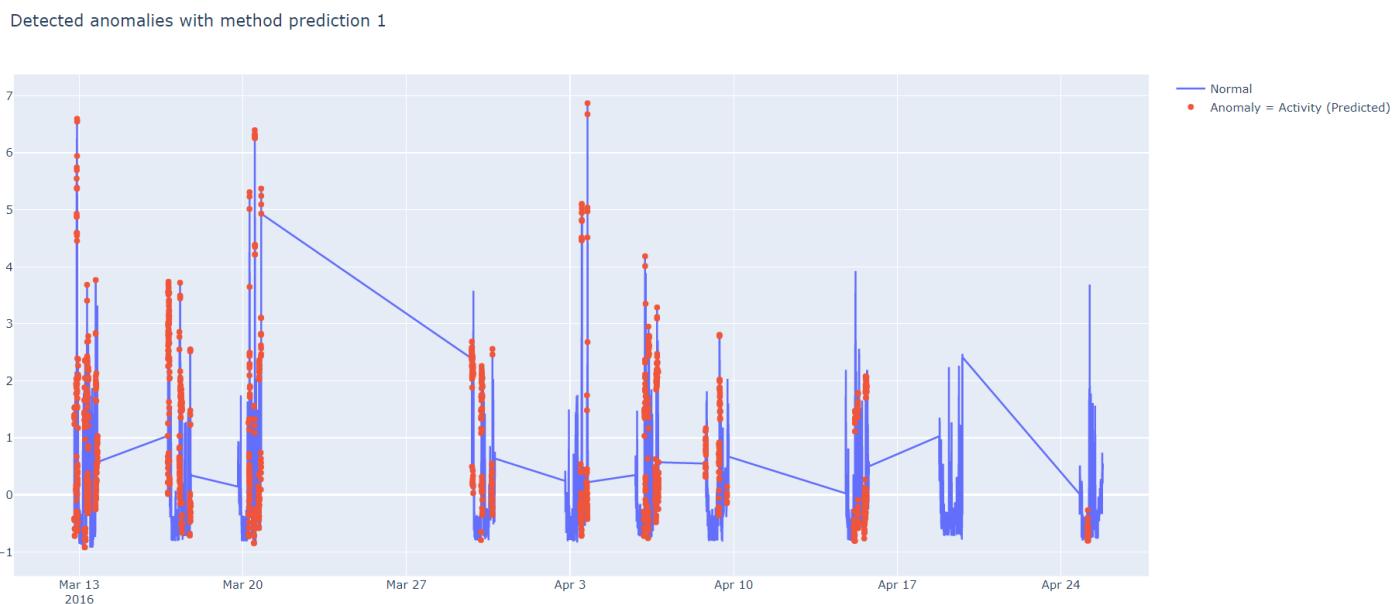


FIGURE 2.73 – Anomalies détectées sur la courbe de charge (Après post-processing)

#### 2.3.4.7 Pipeline du modèle

1. Load dataset
2. Convert user input
3. Pre-processing (build sequences)
4. Build model
5. Train model
6. Plot Reconstructed curve
7. Compute Threshold
8. Make predictions
9. Detecting activities
10. Post-processing (majority vote)
11. Plot detected activities
12. Evaluate model
13. Plot Evaluation model (IoU threshold)

#### 2.3.4.8 Evaluation du modèle

##### Matrice de confusion

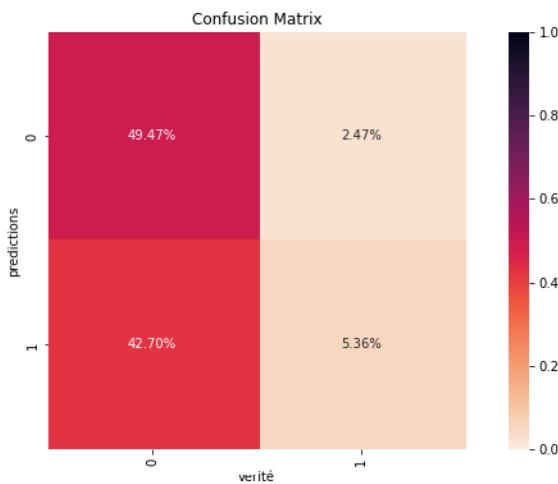


FIGURE 2.74 – Matrice de confusion

##### Activité prédictive VS Activité réelle

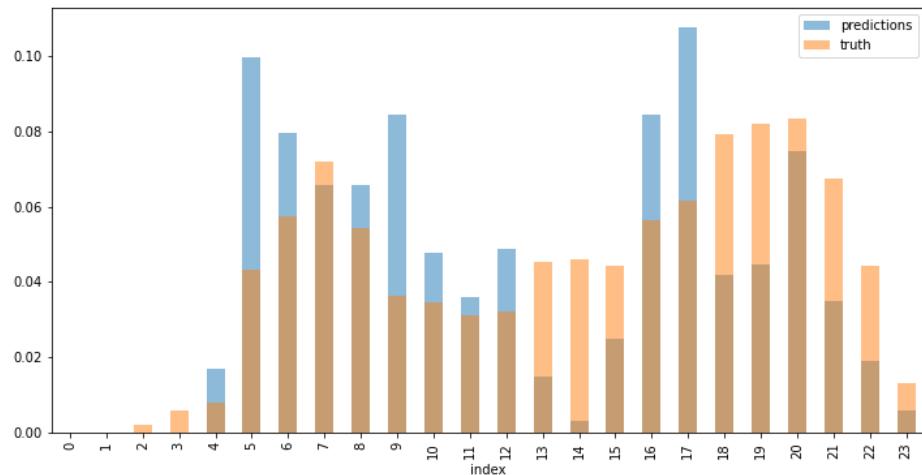


FIGURE 2.75 – Histogramme des activités Réelles VS Prédites

## IoU Threshold

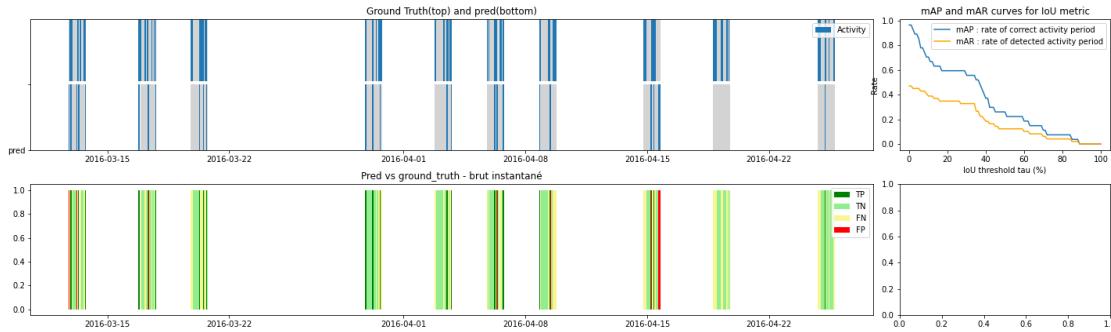


FIGURE 2.76 – Evaluation avec la métrique IoU Threshold

### 2.3.4.9 Discussion de l'approche du modèle

Comme pour l'approche n°2 où nous avons utilisé un modèle  $\{Auto-Encodeur + RandomForest\}$  avec l'*Auto-Encodeur Convolutionnel*, nous sommes capable d'apprendre une représentation fidèle de la consommation électrique de base d'un foyer sans avoir recours au label d'activité. Ainsi cette approche est généralisable dans la mesure où le modèle est capable d'apprendre la consommation électrique de base et ce quelque soit la composition de la famille et le nombre des appareils électroniques utilisés dans la maison. De plus cette méthode est capable d'analyser la consommation électrique d'une maison en se référant à la consommation électrique comme une série temporelle ce qui n'était pas le cas de l'approche n°2. Par conséquent, cette approche semble être bien plus adaptée à notre cas d'usage à savoir la détection des anomalies en quasi-temps réel puisque nous analysons la courbe de charge toute les secondes contre toutes les heures avec l'approche n°2. Enfin, comme pour l'approche n°2, nous éprouvons des difficultés pour vérifier la pertinence des prédictions générées par le modèle dans des situations réelles car nous ne disposons pas d'un jeu de test labellisé pour l'activité. Finalement, à travers cette approche, l'idée est de se mettre dans le cas où un client souhaite "monitored" son activité. Ainsi il s'agirait simplement d'enregistrer la courbe de charge électrique agrégée du foyer pour une durée de 1 mois et ensuite entraîner le modèle sur cette dernière avant de pouvoir déployer le service de détection d'activité chez un utilisateur.

### 2.3.4.10 Futures améliorations possibles

Le modèle Auto-Encodeur Convolutionnel est encore perfectible. En effet, des améliorations sont possibles pour rendre le modèle encore plus performant et surtout plus généralisable. Par conséquent, l'une des améliorations possibles est d'affiner le threshold. Cela consiste à utiliser différentes stratégies pour le calcul du threshold de détection d'activité. Initialement, le threshold correspond à la perte MSE maximale. L'une des stratégies envisageable pour affiner la frontière de décision du modèle est de calculer le quantile plutôt que le "hard maximum" sur le jeu d'entraînement. Cela permet d'attribuer plus de flexibilité au modèle dans le choix threshold qui risque de mal de se généraliser.

Une seconde amélioration consiste à affiner l'architecture de l'auto-encodeur. Une architecture optimale pourrait améliorer les performances de détection d'activité du modèle. Pour trouver l'architecture optimale, il convient de réaliser ce que l'on appelle en machine learning un Neural architecture Search (NAS) qui est une technique d'automatisation de la conception de réseaux neuronaux artificiels (ANN). Il vise à découvrir la meilleure architecture pour un réseau de neurones pour un besoin spécifique. Le NAS reprend essentiellement le comportement du processus d'un humain peaufinant manuellement un réseau de neurones et apprenant ce qui fonctionne bien, et automatise cette tâche pour découvrir des

architectures plus complexes. Ce domaine représente un ensemble d'outils et de méthodes qui testeront et évalueront un grand nombre d'architectures dans un espace de recherche en utilisant une stratégie de recherche et sélectionneront celle qui répond le mieux aux objectifs d'un problème donné en maximisant (ou minimisant selon le cas d'usage) une métrique d'évaluation personnalisée. Toutefois, les méthodes NAS explorent de nombreuses solutions potentielles avec des complexités variables et sont donc très coûteuses en calculs. Plus leurs espaces de recherche sont grands, plus il y a d'architectures à tester, entraîner, évaluer. Par ailleurs, ce domaine souffre de plusieurs autres limitations. En effet, il est difficile de savoir comment un modèle potentiel se comportera sur des données réelles. Comme les architectures sont évaluées avec des données d'apprentissage, ces dernières doivent être de bonne qualité si l'on s'attend à un modèle performant sur des données réelles. Il reste nécessaire de définir comment l'algorithme trouvera et évaluera ces architectures. Cette tâche se fait encore à la main et doit être peaufinée. Cependant, le manque de connaissance du domaine ne va pas pénaliser l'efficacité de l'architecture. Cette connaissance est utile pour accélérer le processus de recherche, elle guidera la recherche et ainsi l'algorithme convergera plus rapidement vers une solution optimale.

D'autres paramètres peuvent également être ajustés en plus de ceux directement liés à l'architecture du modèle. Par exemple la méthode d'agrégation des prédictions du modèle repose sur un vote majoritaire, mais on pourrait également jouer sur cet aspect en exigeant par exemple un consensus. De même nous avons posé les bases d'un système d'optimisation des hyperparamètres tels que la longueur des séquences considérées (observation pendant 1h de la courbe de charge vs observation pendant 10 min pour classifier) ou le pas de temps utilisé pour échantillonner le dataframe (1 mesure par seconde ou 1 mesure par minute ?).

Enfin, les changements d'échelle ou d'habitude de consommation peuvent mettre à mal les performances du modèle. Pour s'assurer d'avoir un modèle durable. Une parade possible à ce phénomène est d'effectuer des mises à jours fréquentes du modèle, par exemple en attribuant plus de poids aux données récentes. Ainsi des changements d'équipements électriques pourraient être pris en compte dynamiquement par la solution, de même que des changements de mode de vie comme le nombre d'habitants, la saisonnalité...

## 2.4 Test des approches sur un autre jeu de données : UK-DALE

### 2.4.1 Récupération et analyse des données

Notre jeu de données présente deux maisons, ce qui nous a permis de tester nos approches et d'essayer de voir si nos modèles se généralisent bien d'une maison à une autre. Pour aller plus loin il serait cependant intéressant d'avoir un jeu de données indépendant pour tester nos modèles sur d'autres scénarios.

Pour cela nous avons utilisé le jeu de données provenant du papier *The UK-DALE dataset, domestic appliance-level electricity demand and whole-house demand from five UK homes* (Jack Kelly, 2015). Ce jeu de données est très riche puisqu'il comprend des relevés de puissances échantillonnés à 6 Hz sur cinq maisons au Royaume-Unis sur des durées allant de 39 à 234 jours. Chaque maison est découpée en plusieurs sous-compteurs qui correspondent tous à un appareil électrique. Les appareils électriques ne sont pas tous relevés mais chaque maison dispose du relevé total agrégé sur tout le logement.

Nous avons fait le choix de labelliser la maison 2 de ce jeu de données car c'est la maison qui présente des relevés de mesure sur la plus grande période de temps (234 jours au total, 142 jours après réconciliation de toutes les mesures et nettoyage) et avec un bon niveau de détail au niveau des appareils électriques

dont la puissance est mesurée.

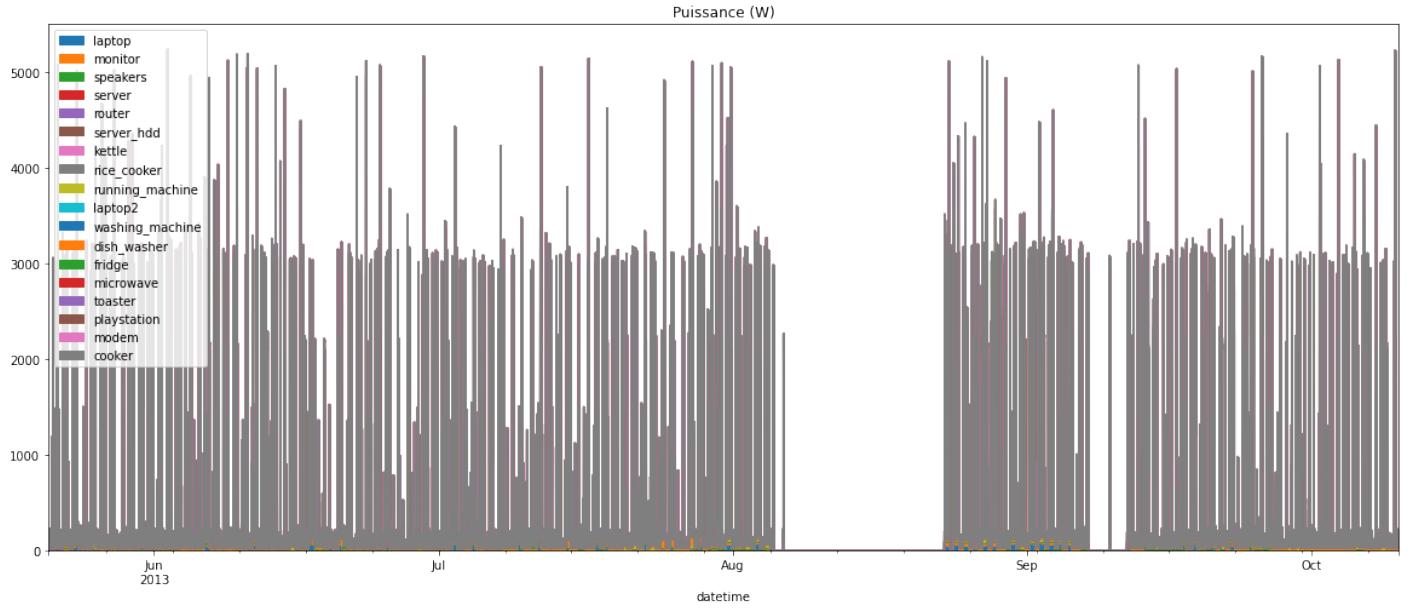


FIGURE 2.77 – Relevés de puissance dans la maison 2 UK-DALE

On peut également observer le graphique circulaire de la répartition du pourcentage des puissances sur la puissance totale consommées ce logement.

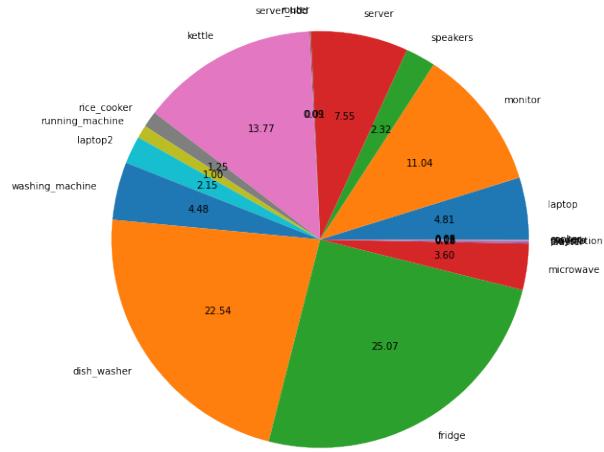


FIGURE 2.78 – Répartitions des puissances dans la maison 2 UK-DALE

#### 2.4.2 Labellisation

La même méthode de labellisation que celle utilisée pour le jeu de données RAE a été appliquée. Chacune des courbes de charge correspondantes à un appareil ont été étudiés. Seuls les courbes des appareils correspondants à de l'activité et ne présentant pas trop de bruit ou de trop faibles amplitudes ont été retenues. Les trois étapes (seuil, lissage puis second seuil) d'attribution du label activité ont été appliquées à chacune des courbes de charges retenues et les labels activités de chaque courbe de charge ont été agrégées.

On peut observer les histogrammes des labels activités par heure de la journée pour chaque courbe de charge désagrégée. Les appareil retenus pour le label activité final sont affichés en bleu, les appareils qui n'ont pas été pris en compte sont en gris.

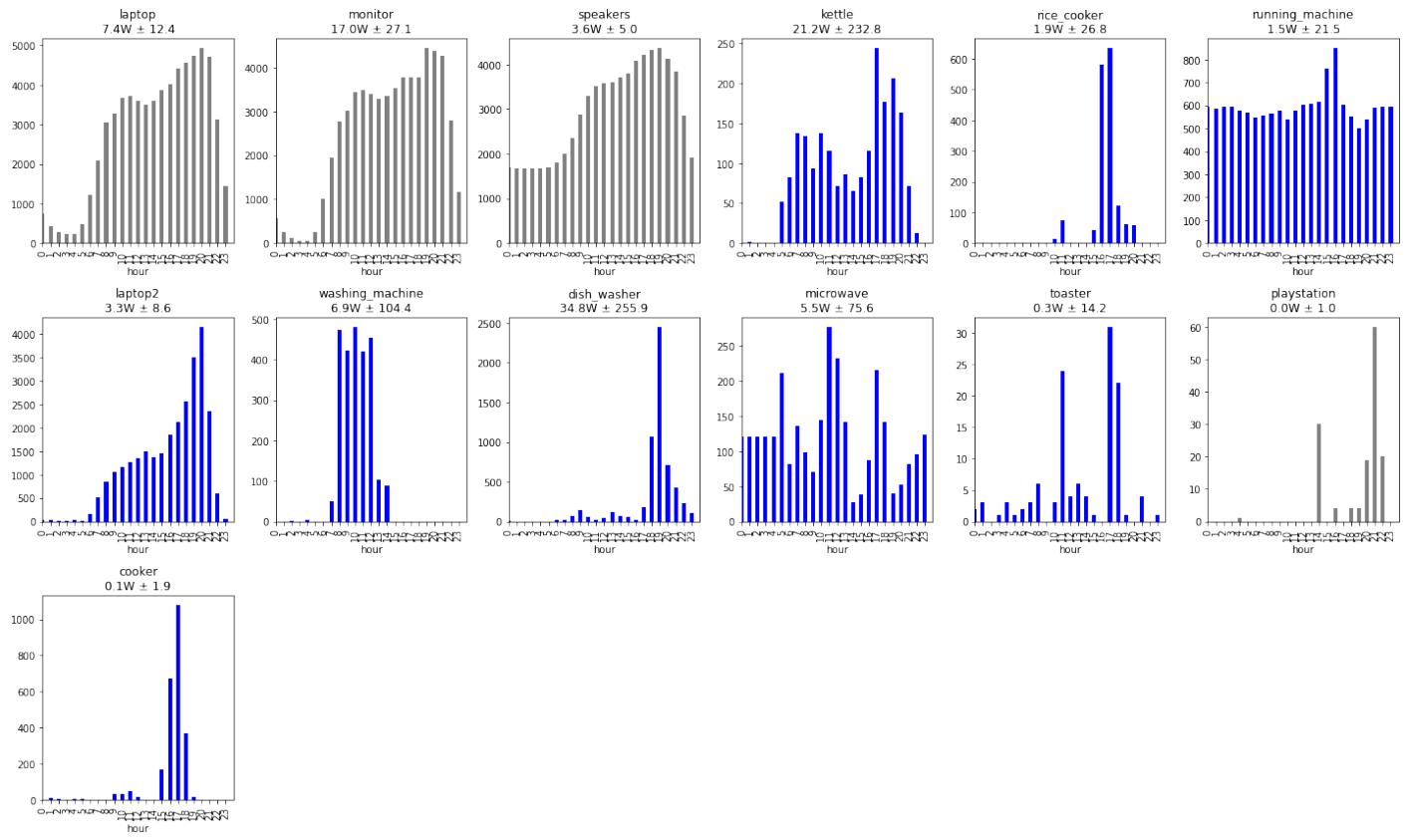


FIGURE 2.79 – Histogramme du label activité en fonction de l'heure de la journée pour chaque appareil de la maison 2 UK-DALE (appareil retenus en bleu, appareil ignorés en gris)

L'histogramme de la courbe activité agrégée par tranches d'une heure (de 0 à 23) est le suivant :

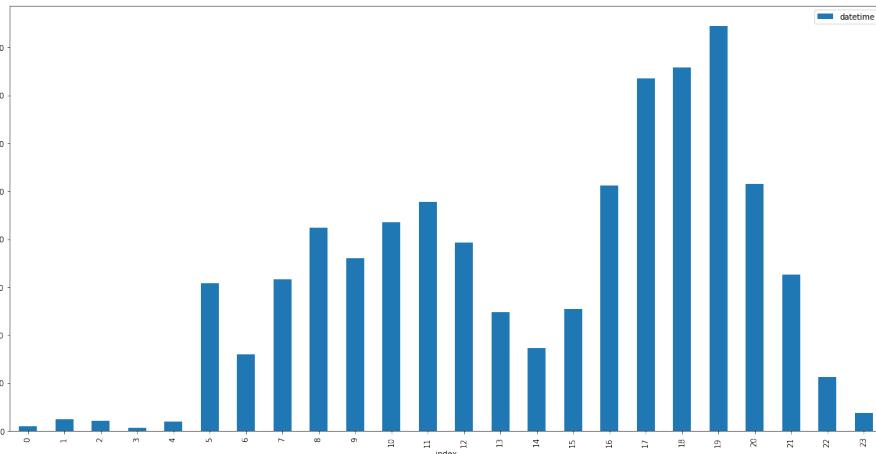


FIGURE 2.80 – Histogramme du label d'activité en fonction de l'heure de la journée pour la maison 2 UK-DALE

Il est intéressant de noter que cette distribution est un peu différente des distributions obtenues avec les logements du jeu RAE. On observe des pics d'activité moins marqués, débutant à 5h du matin,

un léger creux vers 14h-15h puis un pic le soir aux alentours de 18h-19h. Il sera intéressant d'observer comment nos algorithmes se comportent sur ce jeu de données.

### 2.4.3 Résultats de l'approche 1 : Classifieur

Nous avons allons tester le classifieur *LGBMClassifier* avec les hyper-paramètres déterminés précédemment. L'entraînement du classifieur est réalisée avec un mélange des données de la maison 1 et de la maison 2. Tous les jeux de données ont été ré-échantillonnés à la minute (puisque UK-DALE est échantillonné à la minute). Chaque point de données est traité indépendemment, sans tenir compte de la chronologie. Nous avons ensuite générés les features nécessaires sur chacun des jeux de données puis nous avons mis à l'échelle les jeux de données correspondants aux maisons 1 et 2 du jeu RAE avec des objets scalers (*MinMaxScaler*) ajustés sur la maison correspondante. L'objet scaler pour la mise à l'échelle de la maison UK-DALE a été fitté sur les 15 jours premiers jours du jeu UK-DALE pour se mettre dans des conditions où l'on dispose de 15 jours de relevé de puissance totale d'un logement sur lequel on souhaite détecter de l'activité. Le reste du jeu de données UK-DALE (du 16ème jour au dernier) a servi de jeu de test.

Les résultats obtenus avec notre classifieur optimisé sont les suivants :

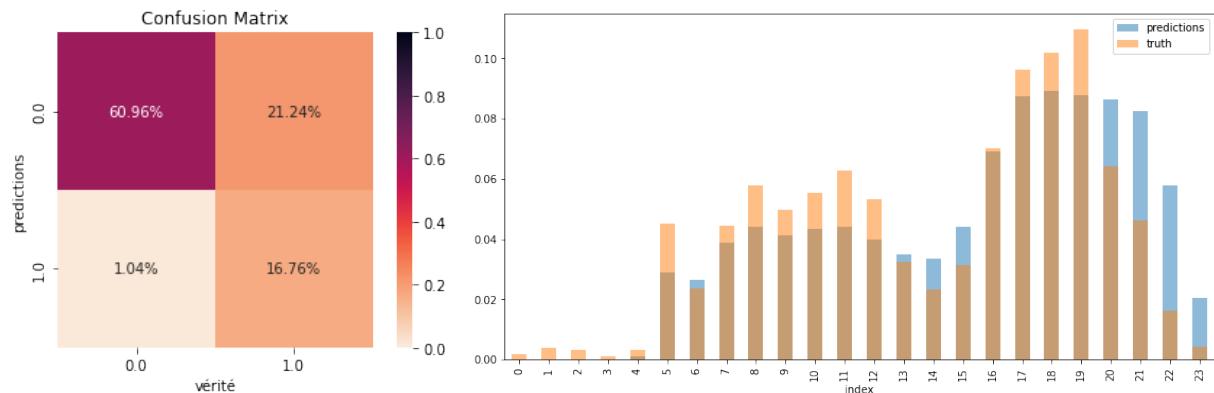


FIGURE 2.81 – Matrice de confusion (accuracy = 77.72%,  $F_{0.5} = 70.83\%$ ) et histogramme d'activité prédictive

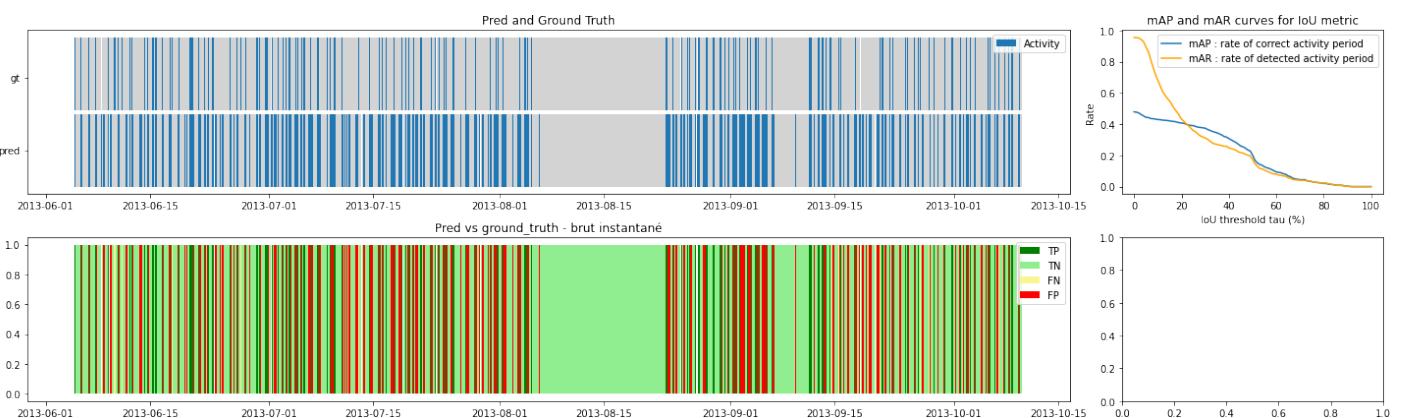


FIGURE 2.82 – Résultats de classification

On constate que notre modèle entraîné sur le jeu de données RAE parvient à faire des prédictions pertinentes sur la maison 2 du jeu UK-DALE. Cela dit les résultats sont moins bons que lors des tests qui se limitent uniquement au jeu de données RAE.

#### 2.4.4 Résultats de l'approche 2 : Time2Vec

Pour tester ce modèle nous entraînons d'abord les embeddings time2vec sur la maison 1 sur 10 epochs (learning rate de 0.001, optimiseur Adam) puis la maison 2 sur 10 epochs, puis nous faisons un fine-tuning des embeddings sur la maison 2 UK-DALE avec un learning plus faible (learning rate de 0.0001). Pour cela nous utilisons 10% du jeu UK-DALE (soit 14 jours) comme jeu d'entraînement et les 139 jours restants comme jeu de test. Nous entraînons ensuite un classifieur constitué de ces embeddings, d'une LSTM et d'une couche fully connected sur la maison 1 sur 16 epochs puis la maison 2 sur 16 epochs (learning rate de 0.0001). On teste alors les performances du classifieur obtenu sur la maison 2 UK-DALE.

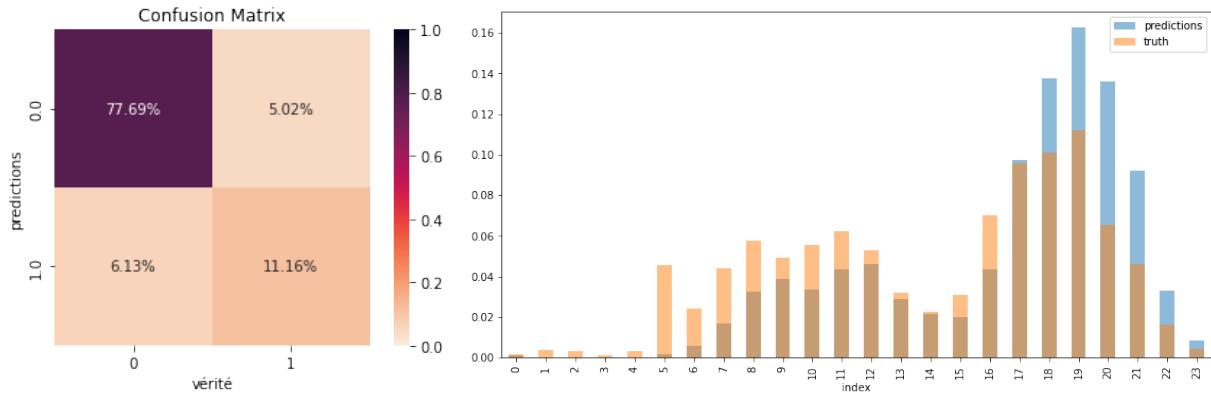


FIGURE 2.83 – Matrice de confusion (accuracy = 88.84%,  $F_{0.5} = 80.47\%$ ) et histogramme d'activité prédite

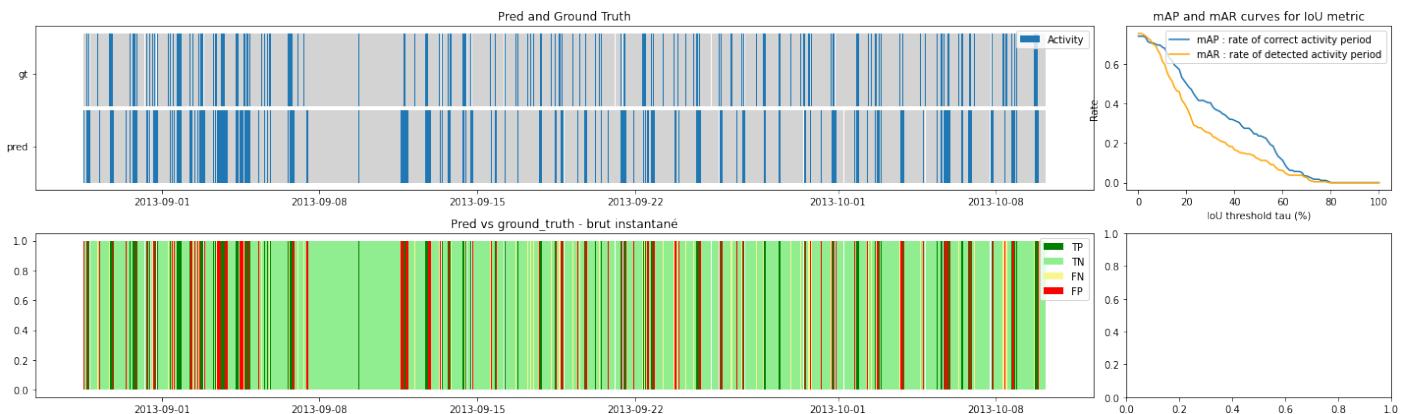


FIGURE 2.84 – Résultats de classification

Il est intéressant de noter que l'on obtient de meilleurs résultats avec cette approche qu'avec l'approche 1, alors que l'approche 1 semblait obtenir de meilleurs scores en entraînant le modèle et en le testant uniquement sur le jeu RAE. Il semblerait donc que l'approche time2vec permette d'obtenir des modèles plus facilement généralisables. Il se peut que les fenêtres de temps choisies lors de la phase de features engineering de l'approche 1 soient adaptées au jeu RAE et moins adaptées pour d'autres jeux de données. Le fait de générer des embeddings de façon non supervisée permet de contourner ce problème. Il est aussi intéressant de noter qu'on obtient ici des résultats très stables en ré-exécutant le code plusieurs fois.

## 2.4.5 Résultats de l'approche 4 : Auto Encodeur Convolutionnel

Pour tester ce modèle nous entraînons l'auto encodeur sur la maison 2 UK-DALE. Afin d'obtenir les résultats détaillés ci-dessous nous utilisons 20% du jeu UK-DALE (soit 28 jours) comme jeu d'entraînement et les 114 jours restants comme jeu de test. En dessous de 28 jours comme jeu d'entraînement les performances sont fortement dégradées.

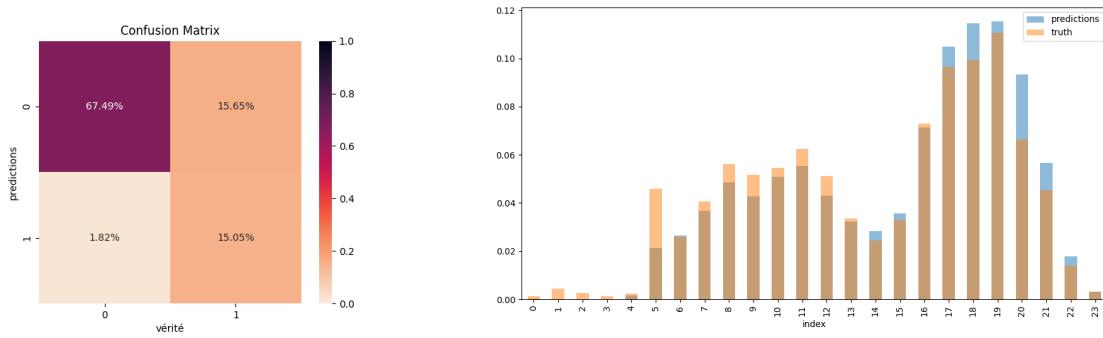


FIGURE 2.85 – Matrice de confusion (accuracy = 82.54%,  $F_{0.5} = 73.75\%$ ) et histogramme d'activité prédite

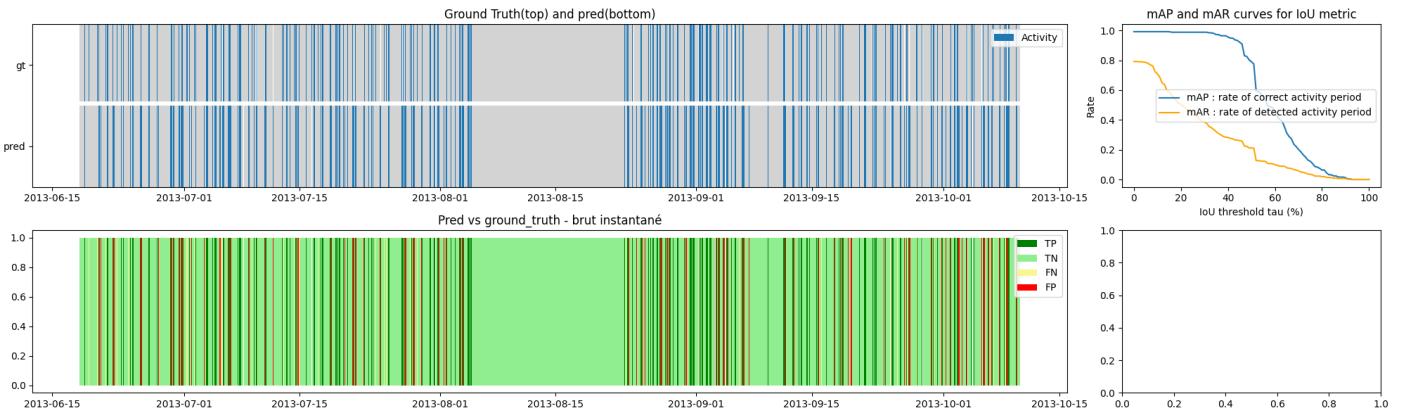


FIGURE 2.86 – Résultats de classification

Nous obtenons ici de bons résultats, le gros avantage de cette méthode est qu'elle ne nécessite pas de données labellisées. Elle requiert cependant près d'un mois de relevés de puissance agrégée du logement dans lequel on souhaite détecter l'activité.

## 2.5 Discussion des résultats

Pour la détection d'activité nous avons donc testé 2 différentes approches :

Une approche supervisée avec le modèle Time2Vec et les Classificateurs (ensemble learning) L'avantage de ces modèles est qu'ils sont rapides à entraîner. Le modèle **Time2Vec** est le modèle qui fournit les meilleures performances sur le jeu UK-DALE. Toutefois, le modèle est peu généralisable car il nécessite les labels des activités pour être entraîné. Cela signifie que le modèle a appris à détecter les activités quotidiennes pour un foyer en particulier. Ainsi, si la composition du foyer vient à changer le modèle va perdre en performance car les activités quotidiennes ne seront plus les mêmes que ceux qu'il a appris à détecter.

Ensuite la seconde approche est une approche non supervisé qui s'appuie sur un **Auto-Encodeur Convolutionnel** : Au delà du fait que cette méthode est rapide à entraîner, le modèle est plus flexible et généralisable que le modèle **Time2Vec** car il n'a pas besoin des labels d'activité pour s'entraîner et le threshold de détection d'activité peut être ajusté très facilement en fonction des habitudes quotidienne d'un foyer. Le modèle bénéficie également d'une explicabilité plus forte que le modèle **Time2Vec** car la frontière de décision du modèle est connue à l'avance étant donné que le threshold de détection d'activité peut être connu pour chaque foyer. A l'inverse, comme pour le modèle **Time2Vec**, le modèle est sensible aux choix des hyperparamètres ce qui rend son déploiement complexe.

## Avantages et inconvénients des approches

Approches supervisées	Approche non supervisée
{ BoostingClassifier , Time2Vec }	
→ Besoin de 15 jours de données	→ Besoin de 28 jours de données
+ Rapide à entraîner	+ Rapide à entraîner
+ Bonnes performances	+ Modèle ajustable (S'adapte à la composition du foyer)
+ Simple à automatiser (time2vec)	+ Généralisable
- Peu généralisable (nécessite des labels)	+ Meilleure explicabilité (Connaissance du threshold de détection d'activité)
- Explicabilité mauvaise (superposition de réseau de neurones) (t2v)	- Pas facile à déployer (beaucoup de paramètres à choisir)
- Sensible aux hyperparamètres (t2v)	- Résultats variables : dépend de l'initialisation des couches de l'AE
- Tendance à "sur-apprendre" (over-fitting) (t2v)	- Sensible aux hyperparamètres

FIGURE 2.87 – Discussion des approches

Enfin on peut noter que l'on retrouve des scores d'accuracy autour des 80% avec toutes ces méthodes, ce qui est cohérent avec les scores observés dans la littérature pour ce genre de problématiques. Les approches supervisées semblent retourner de bons résultats et ne nécessitent que 15 jours de données de relevés de puissances agrégées pour pouvoir les adapter à un nouveau foyer. Il cependant est difficile de présumer de la qualité des résultats sur de nouveaux foyers et ces résultats sont difficilement explicables. À l'inverse la méthode non supervisée permet d'avoir un modèle adapté à chaque logement mais nécessite près de 1 mois de relevés de puissances agrégées pour pouvoir entraîner le modèle. Cette méthode présente de plus l'avantage d'obtenir des résultats qui sont plus facilement explicables puisqu'on détecte de l'activité lorsque la puissance dépasse un certain seuil.

## 2.6 Structure des modèles

L'ensemble des modèles que nous avons implémenté suivent arborescence de fichiers suivante :

### Structure du projet

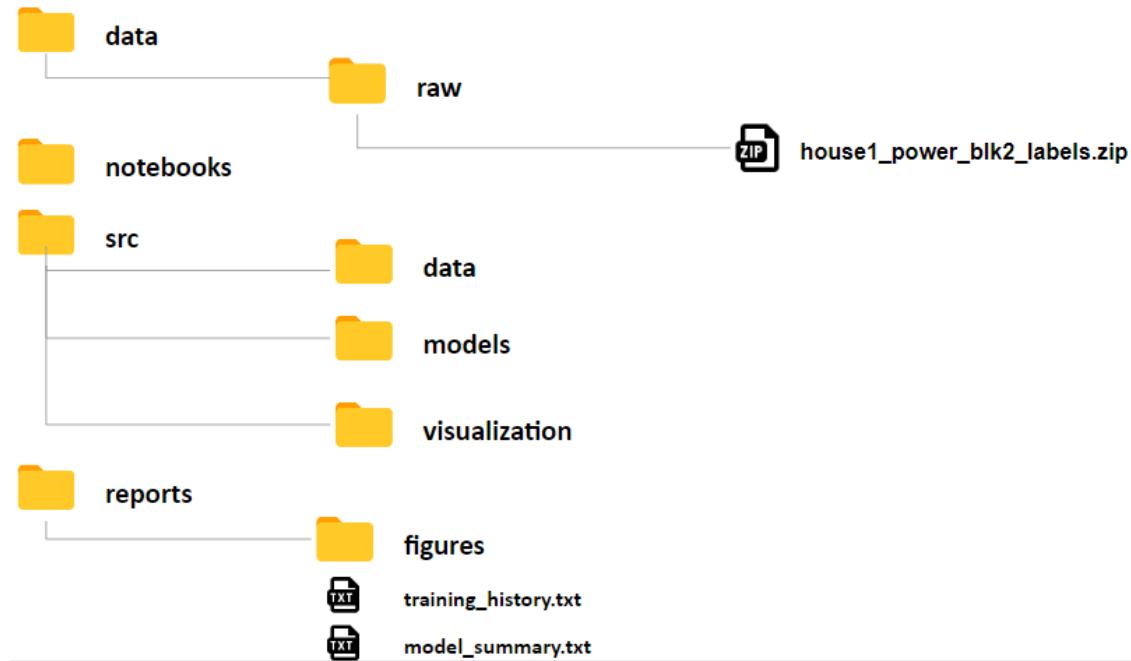


FIGURE 2.88 – Arborescence des fichiers

- *README.md* : top-level *README* for developers using this project
- *data/raw* : the original, immutable data dump
- *notebook* : Jupyter notebook
- *reports* : generated analysis as HTML, PDF, TXT
- *reports/figures* : generated graphics and figures to be used in reporting

## Structure du dossier source "src"

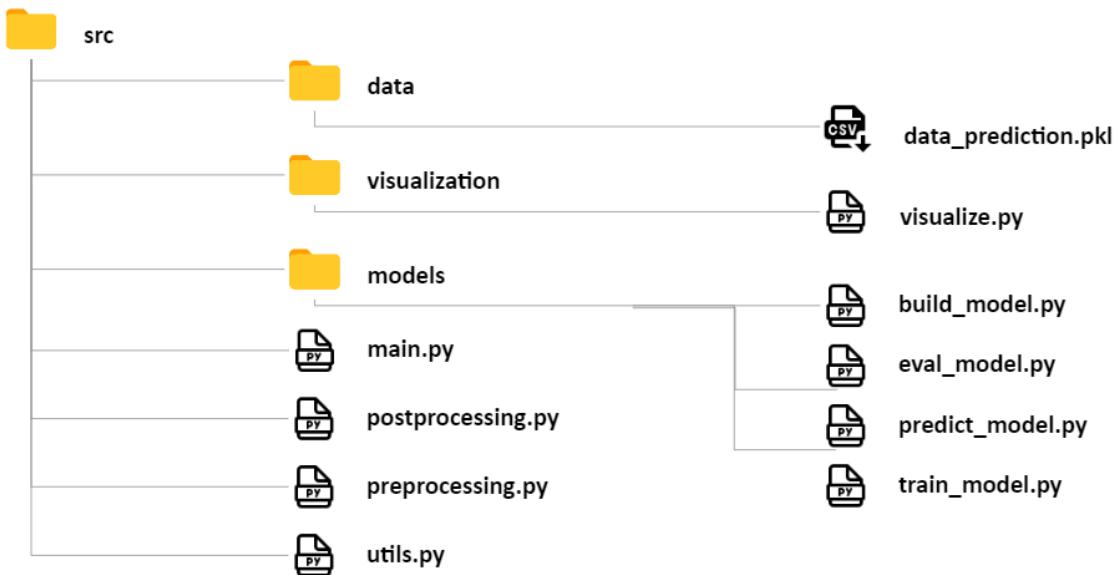


FIGURE 2.89 – Arborescence des fichiers du dossier "src"

- *src : source code for use in this project*
- *src/main.py : run pipeline of the model*
- *src/postprocessing.py : run post-processing*
- *src/preprocessing.py : run pre-processing*
- *src/utils.py : intermediate function for the execution of the pipeline*
- *src/data : generated dataset from the preprocessing*
- *src/visualization/visualize.py : scripts to create exploratory and results oriented visualizations*
  - src/model/buildModel.py : script to build the neural network architecture of the model*
  - src/model/evalModel.py : script to make the evaluation of the model*
  - src/model/predictModel.py : script to generate the predictions*
  - src/model/trainModel.py : script to train the model*