

Membres du groupe : RAVOUX Simon (C2b)
SACRÉ Alexandre (C3b)

ENSE3 - 1A - S6

Parcours à Choix : Gestion de l'énergie (G11)

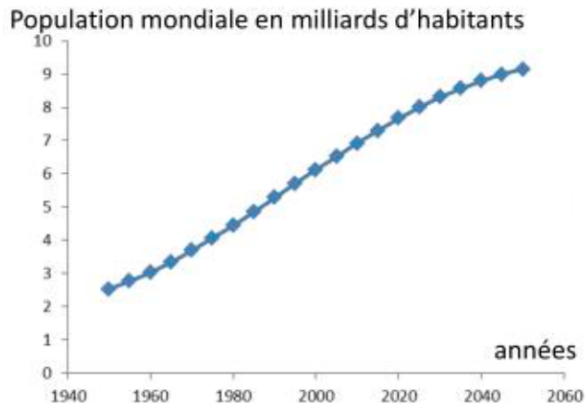


Plan du rapport

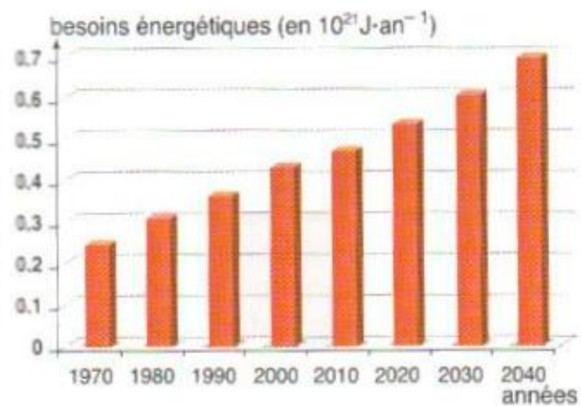
I.	Positionnement du problème / contexte	2
II.	Enjeux de la gestion de la consommation d'énergie	3
III.	Principe de mise en oeuvre d'une solution	4
IV.	Résolution du problème	4
V.	Première approche pour le problème	5
	A. Introduction	5
	B. Choix des paramètres importants	5
	C. Résultats	6
VI.	Seconde approche - réel problème	8
	A. Introduction	8
	B. Connaissance pratique	8
	C. Présentation des moyens à disposition	9
	D. Résultats	11
VII.	Conclusion	13
VIII.	Annexe/codes	14
	A. Codes personnels préliminaires Alexandre SACRÉ	14
	B. Codes personnels préliminaires Simon RAVOUX	17
	D. Codes projet	20

I. Contexte

Alors que la population mondiale ne cesse de croître et que la demande en énergie augmente également depuis des décennies, l'énergie et sa gestion sont devenues les principales préoccupations du XXI^{ème} siècle et des problèmes se posent quant aux perspectives d'avenir pour répondre à la demande. En effet, les graphiques ci-dessous montrent que sans une prise de conscience, la réponse à cette demande énergétique ne pourra pas être assurée.



Source : ONU



Source : Agence Internationale de l'énergie

Figure 1a :

Évolution de la population mondiale de 1950 à 2050

Figure 1b :

Évolution des besoins énergétiques de 1970 à 2040

On relève que les besoins énergétiques ont plus que triplé depuis les années 1970, et ce par habitant, d'où une demande mondiale qui semble croître exponentiellement.

II. Enjeux

L'enjeu majeur se révèle donc être une réduction de la consommation par habitant puisque si les courbes de la figure 1 ne cessent de croître dans les décennies à venir, une prise de conscience encore plus rude aura lieu d'être, avec notamment des solutions comme une décroissance sélective - solution potentielle aux nouveaux enjeux dans ce cas.

D'autres enjeux, inextricablement liés à cette surconsommation énergétique, peuvent avoir des conséquences dramatiques. Du point de vue **écologique et environnemental**, on note l'augmentation des émissions de gaz à effet de serre qui contribue au réchauffement climatique. La courbe de la figure 2 montre que la consommation en énergie et les émissions de CO₂ sont conjointement corrélées.

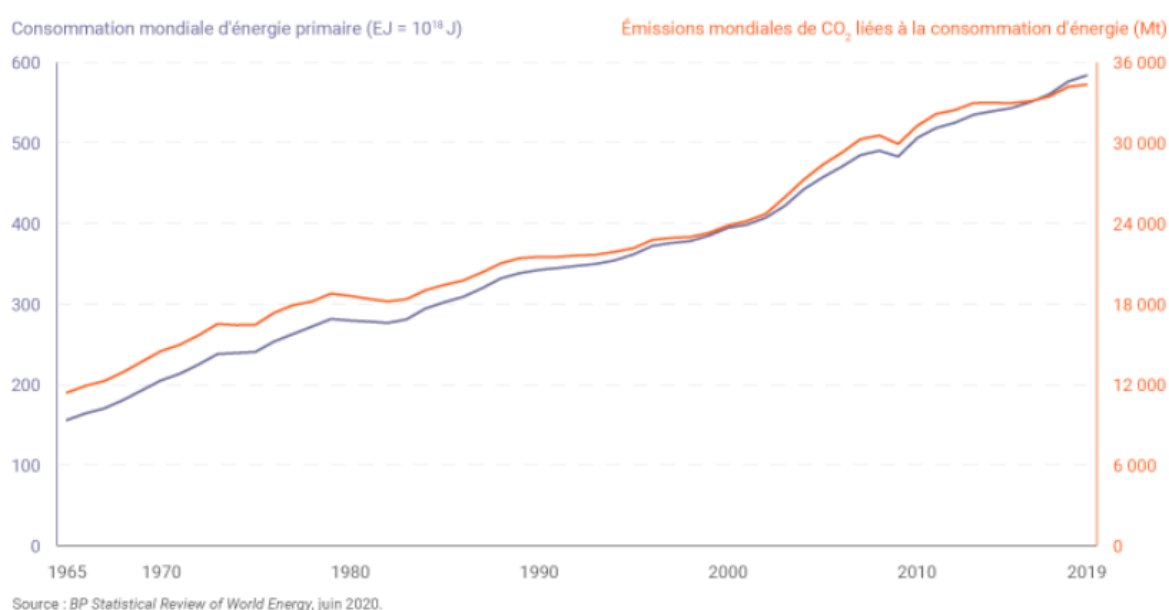


Figure 2 : Évolution de la consommation d'énergie et des émissions de CO₂ dans le monde

D'autres part, on peut noter que l'aspect financier n'est pas à négliger, voire pourrait paraître prépondérant pour certains acteurs de notre société. En effet, Jérôme Ferrari, un enseignant - chercheur "contrôle/commande" du laboratoire G2Elab, a mis en place un système de gestion d'énergie dans sa propre maison, qui lui a permis de réduire de 25% sa facture d'électricité, voire même 35% celle de gaz. Ces réductions montrent vraiment qu'il y a des solutions et qu'elles sont efficaces pour réduire ses consommations énergétiques. Au niveau mondial, de réels conséquences pourraient avoir lieu si de tels systèmes sont intégrés dans les foyers à travers le monde.

III. Principe de mise en oeuvre

Ces différentes statistiques et études anticipatives de la démographie, de la consommation énergétique mondiale, mais également de l'impact environnemental qui en est la conséquence évidente, ne laissent plus de place au doute quant à la possibilité d'une réduction de consommation d'énergie. Dès lors, il faut réfléchir à **optimiser** notre consommation, en essayant de la réduire au maximum, ou du moins l'utiliser à bon escient, en mettant à profit les connaissances mises en place grâce aux innovations technologiques.

Avant de vouloir gérer l'énergie dans une maison, il faut savoir créer un **modèle** en fonction des données à disposition, pour que celui-ci puisse prévoir les comportements à adopter en cas de pertes d'énergie, et dans quels cas elles ont lieu.

Les principes d'une domotique intelligente se base sur des **capteurs**, qui vont relever des températures, des taux de CO₂, mais aussi des données binaires, comme celles concernant la fenêtre (ouverte/fermée), le chauffage (éteint/allumé)... Ces valeurs, au cours du temps, seront stockées dans un fichier (CSV file), qui va, par la suite, être analysé par différents programmes informatiques. Enfin, on pourra conclure sur de potentielles **pertes d'énergie** en fonction d'une potentielle présence humaine dans les différentes salles de la maison et mettre en place des automatisations de systèmes pour fermer les fenêtres, éteindre le chauffage, etc... le cas échéant.

De prime abord, nous avons fait des simulations pour des **données fictives**, et ainsi avoir une première approche de résolution et d'appropriation du sujet. Puis, notre étude s'est portée sur une maison où vivent cinq personnes, et dotée de 360 points de mesures, répartis dans toutes les pièces pour relever différents types de données. Ces capteurs peuvent suivre plusieurs protocoles (avec ou sans fil), et fournissent des données en temps réel.

Les données transmises sont en "open source", qu'on peut consulter sur le logiciel de visualisation de données *Grafana*. On peut y lire ce type de données sous forme de graphiques au cours du temps, comme présenté sur la figure 10 plus loin dans le rapport.

IV. Résolution du problème

A. **Problème**

Sachant que la consommation est liée à l'activité humaine, le but est de reconnaître et prévoir le comportement humain pour les nouveaux services aux utilisateurs. Contrairement à la phase de compréhension, où on pourra construire un arbre de décision, dans le problème réel, aucun capteur n'est disponible pour mesurer directement l'occupation et les activités des usagers. Les capteurs utilisés sont communs, doivent préserver la confidentialité et peuvent s'adapter facilement à un nouveau contexte.

B. Méthodes de résolution

Lors de ce projet, il a été possible d'assimiler deux sortes de techniques d'apprentissage des algorithmes : la **classification** et le **clustering**.

La première méthode consiste en une division des données obtenues en deux catégories : "test" pour tester le modèle, et "train" accompagnées d'étiquettes indiquant la classe des observations (présence/absence). Puis les nouvelles données seront classées en fonction de l'ensemble de l'apprentissage.

La seconde méthode, le clustering, présente la caractéristique de ne pas avoir accès aux classes des données d'entraînement. Le but est donc d'établir l'existence des classes (clusters) à partir des données mesurées ou d'observations.

V. Première approche - données fictives

A. Introduction

Une première approche a été d'intégrer le principe de classification et de bien assimiler le problème en sachant interpréter les données issues d'un fichier les regroupant. La notion d'**arbre de décision** a été développée, et a permis d'en prévoir une conclusion quant à la présence humaine (dans un premier temps binaire : absence/présence, quel que soit le nombre de personnes présentes) ou non dans un local, après définition et classement de valeurs de l'importance des différents paramètres dans le local : températures (extérieure/extérieure), humidité, détection de mouvement, puissance, concentrations de CO₂ (local/couloir), pression acoustique et porte (ouverture/fermeture).

Pour implémenter l'arbre et construire un modèle, nous avons utilisé l'étiquette de sortie donnant directement la présence ou non dans le local, en ne prenant en compte qu'une certaine proportion des données (30% dans notre cas).

B. Choix des paramètres importants

De nos jours, la facture énergétique ne cesse de croître, mais qu'est ce qui la fait grimper significativement? Serait-ce par indifférence ou par scepticisme quant aux possibilités de baisse de la consommation énergétique?

En effet, comme le montre la figure 4, certains appareils domestiques sont étonnamment plus énergivores que d'autres.

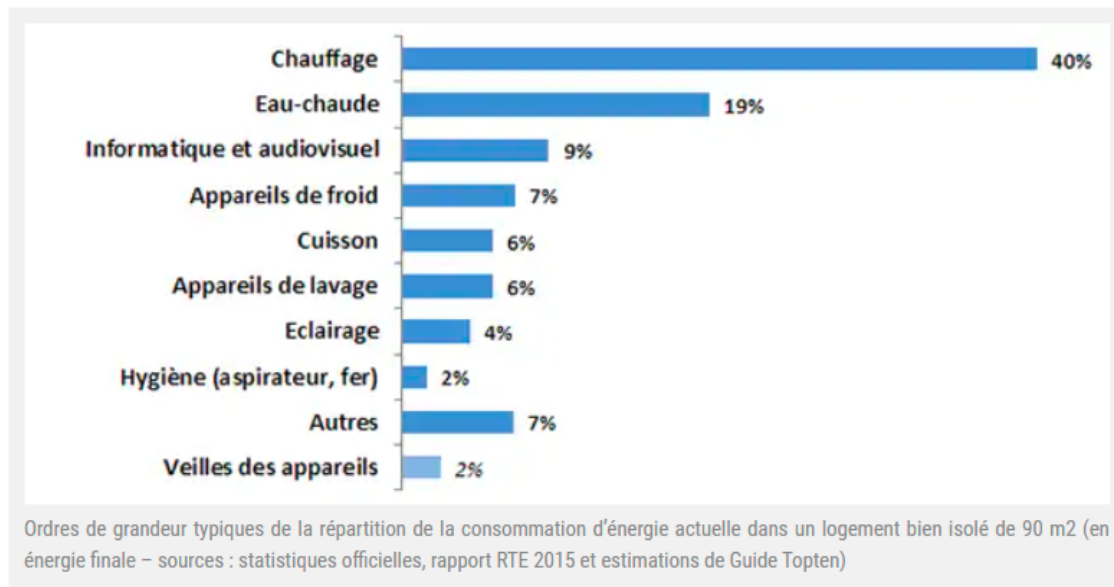


Figure 4 : Part de la consommation énergétique dans un foyer bien isolé

Dans un premier temps, on pourrait penser que c'est la consommation de veille, les répéteurs Wi-Fi ou un éclairage de la façade qui font monter la facture, mais les modèles de prédiction et d'optimisation de la consommation par détection de présence permettent d'en révéler bien plus que ces idées reçues, pourtant justes, mais non suffisantes. Concrètement, pour déterminer les paramètres importants qui vont permettre de construire le modèle et l'arbre de décision, l'entropie et le gain d'information sont des outils essentiels.

C. Résultats

On implémente une fonction sur le principe de l'arbre de décision (voir annexe) qui nous permet de mettre en évidence les paramètres les plus importants pour détecter ou non la présence dans la salle.

On obtient les résultats ci-dessous (figure 5). On voit très vite que la seule donnée qui compte "vraiment" ici est la puissance. Ainsi la puissance suffit à elle seule pour résoudre notre problème simplifié car plus le nombre décimal est élevé, plus la caractéristique associée sera importante.

```

Importance de Tin: 0.044326645028850496
Importance de Tout: 0.0095751432187156
Importance de Humidity: 0.022924745221885437
Importance de detected_motions: 0.0
Importance de Power: 0.8777352788080265
Importance de office_CO2_concentration: 0.011470442593565134
Importance de Door: 0.015910148024516974
Importance de CO2_corridor: 0.007197447915852917
Importance de acoustic_pressure_dB: 0.010860149188586958

```

Figure 5: Importance des différentes données

Après cela, on a pu implémenter l'arbre de décision pour obtenir une prédiction de l'occupation de l'office à partir des caractéristiques importantes. Les résultats de la figure 6 sont ceux obtenus en codant cette occupation binaire (présence : 1, absence : 0), sans prendre en compte le nombre de personnes dans le cas de la présence humaine.

```
the predicted occupancy is:
[0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 1. 1. 0. 1. 0.
 1. 0. 0. 1. 1. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1.
 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 1. 1. 0. 1. 0. 0. 0. 1. 1. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 1. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1.
 0. 1. 0. 0. 1. 1. 1. 0. 1. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 1. 0. 1. 1. 0. 0. 1. 0. 0. 1. 0. 0. 0. 1.
 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 1. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 1. 0. 0. 1. 1.]
Accuracy= 0.961038961038961
```

Figure 6 : Prédiction de l'occupation dans un office en fonction du temps (1 case = 1 prise de mesure à un instant donné)

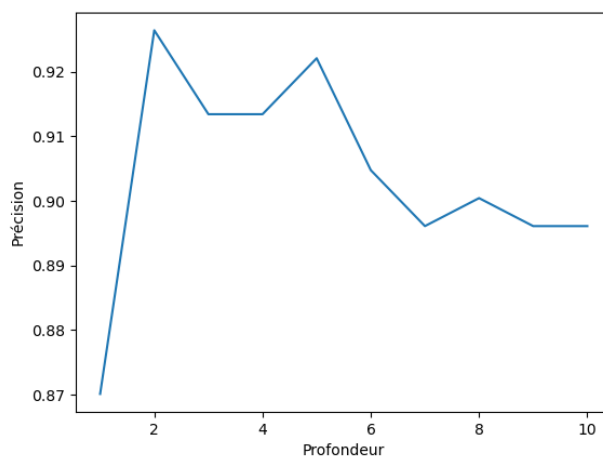


Figure 7.a: Évolution de la précision de l'arbre en fonction de sa profondeur

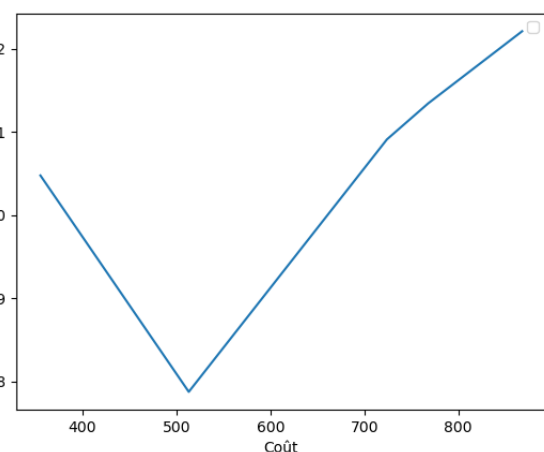


Figure 7.b: Évolution de la précision en fonction du coût des capteurs

Intuitivement et à titre de remarque concernant l'occupation, on pourrait imaginer des corrélations purement logiques, comme par exemple : si la fenêtre est ouverte, il y a vraisemblablement quelqu'un dans l'office, sauf si la personne est étourdie etc...

La figure 7.b présente l'évolution de la précision en fonction du coût des capteurs. Ce résultat est assez exclusif à notre cas, dans le sens où mettre seulement des capteurs de puissance permet d'avoir une précision plus que correcte (point le plus à gauche du graphique). On peut ici facilement déterminer le meilleur compromis au vu des résultats. Mais l'intérêt est ici de comprendre la démarche qui permet, dans des cas plus complexes, d'utiliser uniquement des capteurs utiles à notre objectif.

VI. Seconde approche - étude réelle

A. Introduction

La première différence avec la première approche est que nous n'avons plus la prédiction donnée sur la présence (noté y_label dans les algorithmes) i.e. **pas de données d'entraînement**. La méthode de "decision tree" n'est plus applicable de la même manière que lors de la première partie. Une autre méthode est possible : celle du **clustering**, permettant de regrouper des valeurs statistiques et les regrouper par catégories, nécessaires à la construction du modèle. Il faut récupérer les données brutes. Or, ces dernières ne sont pas synchronisées entre les capteurs, et il peut y avoir des incohérences (valeurs absurdes, voire pas de valeurs dans le cas de capteurs défaillants). Une première étape est donc de synchroniser les données car les capteurs ne relèvent pas forcément des données avec un même pas de temps.

B. Connaissance pratique

Pour comprendre l'utilité et le fonctionnement des réseaux de capteurs présents dans la maison, nous avons pu mettre en place une connectique numérique avec une LED pour la contrôler via les GPIO avec la Raspberry pi, très polyvalente.

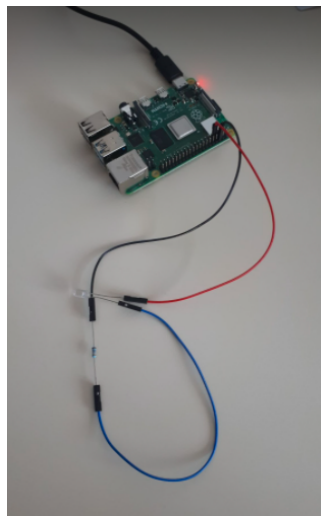


Figure 8.a :
La LED s'allume lors de la commande



Figure 8.b :
Connection et obtention des données

De même, on a pu manipuler et comprendre la synchronisation pour un capteur de température et d'humidité, après avoir rentré des codes dans la fenêtre Putty.

Après la phase de compréhension du fonctionnement des capteurs, le logiciel "open source" *Jeedom* nous a permis de simuler une ébauche de protocole domotique en créant des bâtiments, des salles, d'intégrer des capteurs associés... Puis on s'est également imprégné des notions de "broker" qui peuvent recevoir les messages.

C. Présentation des moyens à disposition

Mis à part les différents logiciels que nous avons pu découvrir pour comprendre le monde de la domotique, nous avons également pu avoir accès à une base de données qui gère des données réelles dans une maison. En effet, InfluxDB est cette base de données open source. Le stockage des valeurs est optimisé pour une bonne disponibilité et une récupération rapide.

De nombreux capteurs relèvent les valeurs des différentes caractéristiques dans différentes pièces de la maison avec un certain pas de temps entre deux relevés. Par exemple, le bureau ci-dessous présente une vingtaine de capteurs à différents endroits.

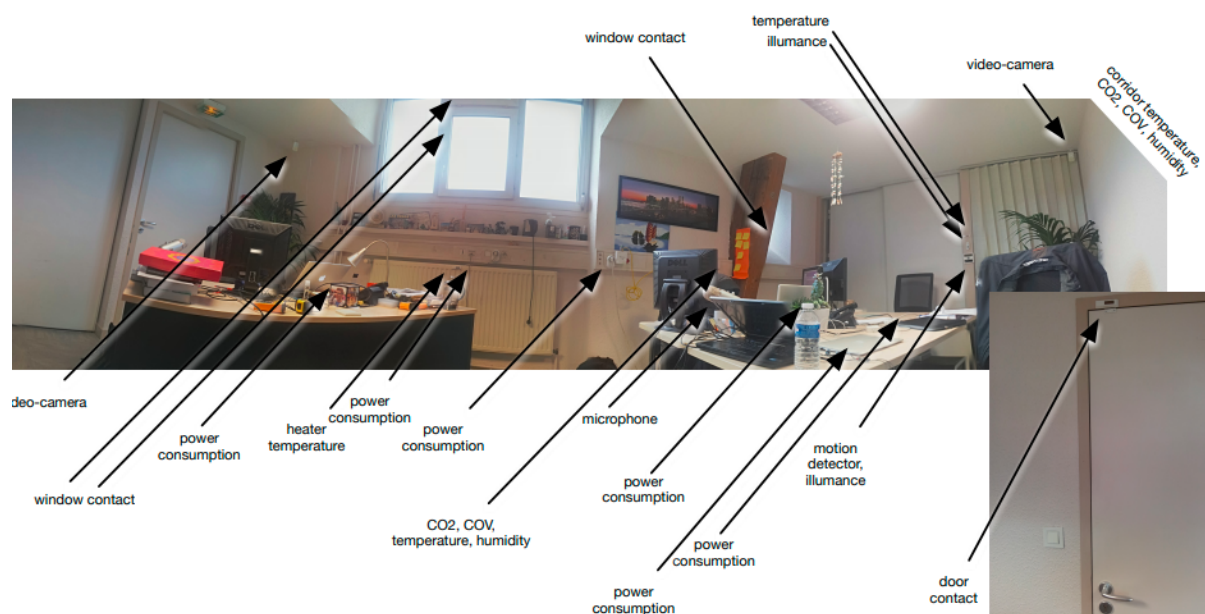


Figure 9 : Position des capteurs dans un bureau de la maison

Le logiciel "open source" *Grafana* permet d'avoir accès aux données des 360 points de mesure de la maison sur plusieurs années. En fonction de la date d'installation de ces derniers, on peut visualiser les différentes caractéristiques de consommation énergétique sur au plus 5 ans, comme figurant sur la figure 10.



Figure 10: Évolution de différents paramètres sur une période de 30 jours

On pourrait penser à titre d'exemple, d'après les évolutions de la concentration en CO₂, de la puissance de la Freebox TV, mais aussi de l'utilisation de la lumière, que les personnes n'étaient pas présentes le week-end du samedi 23 et dimanche 24 mai... Donc allumer le chauffage, ou bien ne pas allumer les appareils électroménagers peut s'avérer être intéressant.

Dans notre étude, Il y aura une perte d'énergie lorsque, par exemple, on aura un chauffage allumé, une fenêtre ouverte et un taux de CO₂ bas, d'où la nécessité d'une amélioration de comportement pour réduire les pertes et diminuer la consommation. Pour ce faire, on doit définir s'il y a une présence dans la pièce, et calculer les indicateurs pour déterminer s'il y a pertes.

Ces données ont été prises sur *Grafana* dans la rubrique "Maison Expé", qui donne seulement les capteurs qui nous intéressent pour simplifier le projet, alors que la rubrique "Maison" rassemble tous les capteurs de la maison, certains moins pertinents que d'autres.

D. Résultats

0. Acquisition des données

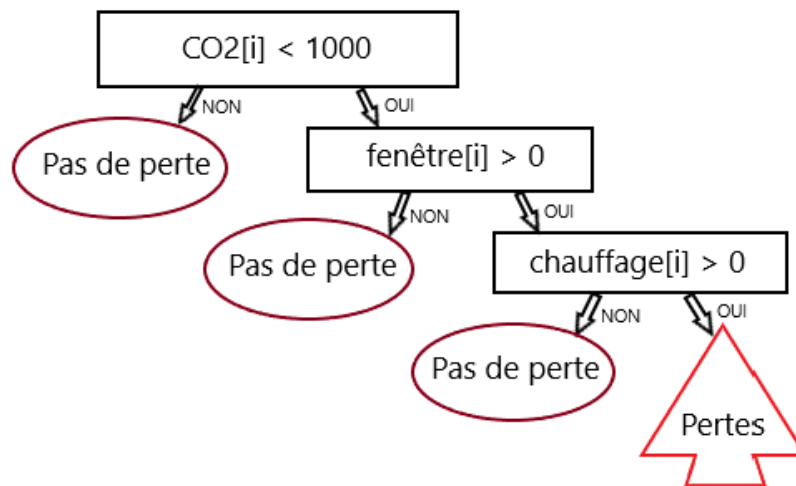
Avant de commencer l'exploitation des données, nous avons d'abord dû les extraire de la base de données. Pour cela, nous avons utilisé un script python qui, à partir d'ID associé à chaque capteur, réalise l'extraction des données dans une base de données gérée par influxDB. Après extraction, les données sont transformées en un fichier CSV (voir les lignes de codes caractéristiques en annexe). Nous avons créé un fichier CSV pour chacune des caractéristiques, avant de synchroniser les mesures de chacun.

1. Sans utilisation du DataTree

Dans un premier temps nous n'avons pas cherché à utiliser le "DataTree". On conjecture qu'il y a une présence entre 17h et 23h. On analyse donc les données uniquement sur ces périodes de la journée.

Ici, le principe était assez simple : on regarde par exemple si la fenêtre est ouverte, si le chauffage est allumé et si la concentration en CO₂ est supérieure à 1000 ppm. Si ces trois conditions sont réunies, alors le programme détecte une perte d'énergie. Ces paramètres n'ont pas été choisis arbitrairement, mais nous avons considéré que le contexte du problème était le même que dans la première approche, avec une approche physique et logique du problème pouvant refléter une présence facilement et **intuitivement**. Par ailleurs, cela a été plus aisé à coder car pour le chauffage et la fenêtre, on pouvait traiter des valeurs **binaires**, alors que pour le taux de CO₂, le critère <1000 ppm paraissait correspondre au modèle de l'arbre construit dans la première partie.

Une visualisation du principe d' "arbre" de décision est proposée ci-dessous. A chaque itération *i*, c'est à dire à chaque prise de mesures à un instant *t*, on aura :



D'autres arbres auraient pu être construits de même, comme dans la première approche, où on souhaitait déterminer s'il y avait une occupation dans l'office. Dans le cas où des pertes sont détectées, il faudra agir pour ne plus qu'il y en ait, soit en envoyant une notification sur les smartphones des utilisateurs, soit en automatisant l'ensemble de la maison pour la rendre "auto-correctrice".

Enfin, nous avons gardé ces paramètres en question pour des raisons de **sécurité** (fenêtre ouverte = risque d'intrusion si pas d'occupation par exemple), ou pour des raisons plus évidentes de **baisse de consommation**, où on a vu que le chauffage représentait 40% des dépenses liées à l'énergie dans un foyer.

Nous avons créé notre propre étiquetage, car il n'était pas à disposition dans ce cas, par la commande "hours" (cf code en annexe), qu'il est possible d'utiliser dans l'arbre de décision. Nous avons récupéré des données sur quelques jours; et les résultats obtenus sont les suivants.

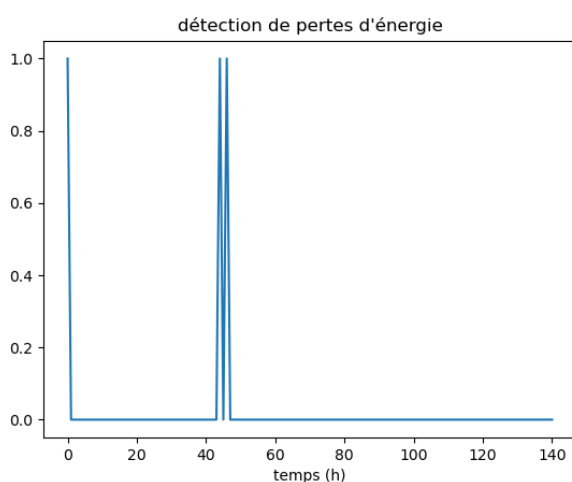


Figure 11.a : Détection de pertes d'énergie

	fenetre	hour	CO2	chauffage
datetime				
2021-05-06 08:00:00	2.0	32	540.5	1.0
	fenetre	hour	CO2	chauffage
datetime				
2021-05-07 06:00:00	1.0	12	850.75	3.0
	fenetre	hour	CO2	chauffage
datetime				
2021-05-07 07:00:00	1.0	14	707.25	1.0

Figure 11.b : Instants des pertes d'énergie

Ainsi on remarque sur la figure 11.a que l'on a détecté trois fois une perte d'énergie. La figure 11.b nous donne le détail des instants où ont lieu ces pertes.

2. Implémentation de notre propre "DataTree"

Pour cette partie nous avons d'abord créé notre propre "label" pour pouvoir entraîner notre arbre de décision. Ainsi nous avons ajouté la donnée "bruit" à notre tableau de données (si le capteur détecte du bruit, il y a une présence, sinon non) . A partir de là, nous avons fait face à un problème: le manque de données. Après arrangement des 4 paramètres dans notre tableau, il ne nous reste plus que quelques dizaines d'instants à exploiter, ce qui a entraîné, malgré l'implémentation d'un DataTree (voir annexe) un dysfonctionnement (importance de l'ensemble des paramètres nulle...). C'est pourquoi, un modèle valide d'un arbre pour cette application n'a pas pu être abouti. Mais, nous comptons mobiliser nos équipes d'ingénieurs pour y remédier.

VII. Conclusion

Après détection d'une potentielle perte d'énergie, on pourra agir en connaissance de cause pour réduire la consommation énergétique. Par exemple, si on relève une perte d'énergie, on pourra communiquer et suggérer aux utilisateurs, via smartphone, différentes méthodes pour y remédier : éteindre le chauffage dès que cette perte d'énergie est détectée entre les mois de mars à octobre si la température est supérieure à 20°C...

De surcroît, des conseils pourront être donnés aux usagers : utiliser la machine à laver pendant les heures dites creuses, après études des prix de l'électricité en fonction des heures de la journée, et pouvoir ainsi mieux réguler la gestion de l'énergie sur une plus grande échelle (nationale...), ainsi que leur budget de la même manière.

Par ailleurs, on peut également imaginer un système de LED qui suit le mouvement de la personne dans une salle par un système de capteurs pour n'éclairer que l'emplacement de l'individu lui-même.

Réduire ses consommations énergétiques n'est pas nécessairement synonyme de restrictions au quotidien, mais rime plutôt avec optimisation des moyens développés dans un passé proche pour le mettre à profit dans un futur, que l'on espère, riche de perspectives et d'innovations, tout en se souciant de ses enjeux énergétiques. Le stockage de l'énergie et sa production sont pour l'instant limités; l'énergie suscite aujourd'hui aussi bien attention, convoitise que angoisse quant aux solutions viables compte tenu des évolutions prévisionnelles de la consommation.

VIII. Annexe/codes

A. Codes Alexandre - première approche

```
##### Objectives #####
# Estimate the number of people, so the occupancy in an office, having values of features
and occupancy for different given moments.
# Implementation to get results : decision tree classification

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
#import graphviz

from math import *
# from sklearn.cluster import KMeans
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn import metrics

#power, detected_motions, office_CO2_concentration, door = [], [], [], [] # even more if we
want other features

##### Read the CSV file with panda #####
data = pd.read_csv('C:\Users\REELA\Desktop\ENSE3\S6\Cours d'approfondissements
filière\Parcours à choix\G11 Énergie\Cours 3 17 Mai\data.csv', sep=";")

##### Read the CSV file #####
# file = open('/Users/amayrim/Downloads/g11/data.csv', 'r')
# Title = file.readline()
# for line in file:
#     line_token = line.split("; ")
#     power.append(str(line_token[1])), detected_motions.append(str(line_token[2])),
office_CO2_concentration.append(str(line_token[3])), door.append(str(line_token[4])) #str
pour stocker les valeurs dans différentes colonnes
# file.close()
#
# Index
=["Tin","Tout","humidity","detected_motions","power","office_CO2_concentration","Door","CO
2_corridor","acoustic_pressure_dB"]
##### Definition of the most important features #####
def features_importance(x_train,y_train):

classifier=tree.DecisionTreeClassifier(random_state=0,criterion='entropy',max_depth=None)
classifier.fit(x_train,y_train)
return classifier.feature_importances_
```

```

##### Discretize labels to (0,1) #####
print (data.label[100])

occupancy=data.label

for n in range(len(occupancy)):
    if ((type(occupancy[n])!= int):
        if (occupancy[n] < 0.5):
            occupancy[n] = 0
        else:
            occupancy[n] = 1

data['occupancy']=occupancy

print(data.occupancy[100]) #pour visualiser la différence avant et après discrétisation


##### Divide the data into training and testing #####
# x_train, x_test, y_train, y_test = train_test_split(data[["Tin", "Tout", "humidity",
"detected_motions", "power", "office_CO2_concentration", "Door",
"CO2_corridor","acoustic_pressure_dB"]],
#
data[['occupancy']],test_size=.3,random_state=0)
x_train, x_test, y_train, y_test = train_test_split(data[["Tin", "Tout", "humidity",
"detected_motions", "power", "office_CO2_concentration", "Door",
"CO2_corridor","acoustic_pressure_dB"]],
data[['occupancy']],test_size=.3,random_state=0)
imp = features_importance(x_train,y_train)
print("feat importance = ", imp)


##### Implement the Decision Tree #####
def decision_tree(x_test,y_test,x_train,y_train):
    classifier=tree.DecisionTreeClassifier(random_state=0, criterion='entropy',
max_depth=None)
    classifier=classifier.fit(x_train, y_train) #fit the tree
    y_test_predictor = classifier.predict(x_test)
    classifier.feature_importances_
    with open("tree.dot", "w") as file:
        f = tree.export_graphviz(classifier, out_file=file) #the function
sklearn.tree.export_graphviz generates a GraphViz representation of the decision tree,
which is then written in out_file
    s = graphviz.Source.from_file("tree.dot")
    s.view()

    return classifier.predict(x_test) #mettre le return en dehors de la fonction pour avoir le
file

```

```
y_prediction_occupancy=decision_tree(x_test,y_test,x_train,y_train)
print("The predicted occupancy is: \n", y_prediction_occupancy)
```

```
##### Accuracy #####
```

```
Accuracy= metrics.accuracy_score(y_test, y_prediction_occupancy, normalize=True)
print("The accuracy is: ", Accuracy)
```

```
##### Average error #####
```

```
# er=[]
# for n in range (len(y_test)):
#     distance=abs(y_test[n] - y_prediction_occupancy[n])
#     er.append(distance)
#
# print ("The error is : ", er)
```

```
##### Implement the Decision Tree, View the most important features, and accuracy
#####
```

```
def decision_tree(x_test,y_test,x_train,y_train):
    classifier=tree.DecisionTreeClassifier(random_state=0, criterion='entropy',
max_depth=None)
    classifier=classifier.fit(x_train, y_train) #fit the tree
    y_test_predictor = classifier.predict(x_test)
    Accuracy=classifier.score(x_test,y_test)
    features_importance=classifier.feature_importances_
    indices=np.argsort(features_importance) #that module from numpy will return the
indices that would sort an array
    plt.yticks(range(len(indices)), [x_train.columns.values[i] for i in indices]) #current y-axis
graduation locations and labels, soit plt.ylabel
    plt.xlabel('Importance features') #rename the axis x (as for yticks...)
    plt.title('Most important features view')
    plt.barh(range(len(indices)), features_importance[indices], color='r', align='center') #to
plot a horizontal bar graph, thanks to the module barh(y, feature_importance, left=None, *,
color='red', align='center', **kwargs)
```

```
with open("tree.dot", "w") as file:
```

```
    f = tree.export_graphviz(classifier, out_file=file) #the function
sklearn.tree.export_graphviz generates a GraphViz representation of the decision tree,
which is then written in out_file
    print("features_importance", features_importance)
    print(metrics.classification_report(y_test,y_test_predictor)) #on pourrait donner un nom
aux différentes cibles car cette commande est du type : sklearn.metrics.classification_report(
y_test , y_test_predictor , *, labels = None, target_names = None, sample_weight = None,
digits = 2, output_dict = False, zero_division = 'warn')
    return (y_test_predictor)
```

```
##### Time (en anticipation des notions pour la seconde approche) #####
# data_datetimes = pd.to_datetime(data.iloc[:0], format='%Y-%d-%m %H:%M:%S')
# strftime format
#
# hours = []
# for dt in data_datetimes:
#     hours.append(dt.hour)      #on prend comme base de temps l'heure pour ne prendre
#                               #que certaines valeurs parmi toutes les données
# data['hour'] = hours
```

B. Codes Simon - première approche

```
import pandas as pd
from sklearn import tree
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

data = pd.read_csv('data.csv', sep=';')
label=data['label']

for i in range(len(label)):
    if label[i]<=0.5:
        label[i]=int(0)
    elif 0.5<label[i]<1.5:
        label[i]=int(1)
    elif 1.5<label[i]<2.5:      #Cette partie permet de complexifier l'arbre
        #pour plus de précision
        label[i]=int(2)
    else:
        label[i]=int(3)

data['label']=label

print(data)

def decision_tree(x_train, x_test, y_train, y_test,D):
    clf=tree.DecisionTreeClassifier(max_depth=D,random_state=0)
    clf=clf.fit(x_train,y_train)
    pred=clf.predict(x_test)
    accuracy=clf.score(x_test,y_test)
    importance=clf.feature_importances_
    return importance,accuracy
```

```
x_train, x_test, y_train, y_test = train_test_split(data[["Tin","Tout", "humidity",
"detected_motions","power","office_CO2_concentration","Door","CO2_corridor","acoustic_pressure_dB"]],data[["label"]], test_size=.3, random_state=0)
```

```
#Boucle en faisant varier la profondeur de l'arbre
```

```
L1=[1,2,3,4,5,6,7,8,9,10]
```

```
L2=[]
```

```
for i in range(1,11):
```

```
    L2.append(decision_tree(x_train, x_test, y_train, y_test,i)[1])
```

```
plt.xlabel("Profondeur")
```

```
plt.ylabel('Précision')
```

```
plt.plot(L1,L2)
```

```
#Importance des datas
```

```
print(decision_tree(x_train, x_test, y_train, y_test,5)[0])
```

```
Z=decision_tree(x_train, x_test, y_train, y_test,5)[0]
```

```
print("\nImportance de Tin:",Z[0])
```

```
print("\nImportance de Tout:",Z[1])
```

```
print("\nImportance de Humidity:",Z[2])
```

```
print("\nImportance de detected_motions:",Z[3])
```

```
print("\nImportance de Power:",Z[4])
```

```
print("\nImportance de office_CO2_concentration:",Z[5])
```

```
print("\nImportance de Door:",Z[6])
```

```
print("\nImportance de CO2_corridor:",Z[7])
```

```
print("\nImportance de acoustic_pressure_dB:",Z[8])
```

```
# Calcul de la précision de la prévision avec plus ou moins de capteurs pris en compte"
```

```
L=[]
```

```
Cost=[] #liste du coût dans chaque cas
```

```
#avec seulement la data "power"
```

```
x_train, x_test, y_train, y_test = train_test_split(data[["power"]],data[["label"]],
```

```
test_size=.3, random_state=0)
```

```
print("\nSeulement power:",decision_tree(x_train, x_test, y_train, y_test,5)[1])
```

```
L.append(decision_tree(x_train, x_test, y_train, y_test,5)[1])
```

```
Cost.append(355)
```

```
#avec tout sauf power
```

```
x_train, x_test, y_train, y_test = train_test_split(data[["Tin","Tout", "humidity",
```

```
"detected_motions","office_CO2_concentration","Door","CO2_corridor","acoustic_pre  
ssure_dB"]],data[["label"]], test_size=.3, random_state=0)
```

```
print("\nTout sauf power:",decision_tree(x_train, x_test, y_train, y_test,5)[1])
```

```
L.append(decision_tree(x_train, x_test, y_train, y_test,5)[1])
```

```
Cost.append(513)
```

```
#sans humidity et Température
x_train, x_test, y_train, y_test =
train_test_split(data[["detected_motions","power","office_CO2_concentration","Door",
"CO2_corridor","acoustic_pressure_dB"]],data[["label"]], test_size=.3,
random_state=0)
print("\nSans les capteurs de température et d'humidité:",decision_tree(x_train,
x_test, y_train, y_test,5)[1])
L.append(decision_tree(x_train, x_test, y_train, y_test,5)[1])
Cost.append(724)
```

```
#sans les capteurs de température
x_train, x_test, y_train, y_test = train_test_split(data[["humidity",
"detected_motions","power","office_CO2_concentration","Door","CO2_corridor","aco
ustic_pressure_dB"]],data[["label"]], test_size=.3, random_state=0)
print("\nSans les capteurs de température:",decision_tree(x_train, x_test, y_train,
y_test,5)[1])
L.append(decision_tree(x_train, x_test, y_train, y_test,5)[1])
Cost.append(768)
```

```
#calcul de précision avec toutes les data
x_train, x_test, y_train, y_test = train_test_split(data[["Tin","Tout", "humidity",
"detected_motions","power","office_CO2_concentration","Door","CO2_corridor","aco
ustic_pressure_dB"]],data[["label"]], test_size=.3, random_state=0)
print("\nTout les capteurs:",decision_tree(x_train, x_test, y_train, y_test,5)[1])
L.append(decision_tree(x_train, x_test, y_train, y_test,5)[1])
Cost.append(868)
```

TT=["Uniquement le capteur de puissance","Tout sauf la Puissance","Tout sauf humidité et température","Tout sauf température","Tous les capteurs"]

```
plt.figure("Coût/Precision")
plt.xlabel('Coût')
plt.ylabel('Précision')
plt.plot(Cost,L)
plt.legend()
plt.show()
```

C. Codes projet - seconde approche

```
import pandas as pd
from sklearn import tree
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

import datetime
import time
import numpy as np
from sklearn.cluster import KMeans

CO22= pd.read_csv('CO22.csv', sep=',')
chauffage2= pd.read_csv('chauffage2.csv', sep=',')
fenetre2=pd.read_csv('fenetre2.csv', sep=',')

def stringdate_to_datetime(stringdatetime: str): #fonction qui permet de modifier l'écriture de
chaque instant (retirer les caractères inutiles)
    if "." in stringdatetime:
        stringdatetime=stringdatetime.split(".")[0]
    else:
        stringdatetime=stringdatetime.split("Z")[0]

    return datetime.datetime.fromtimestamp(time.mktime(time.strptime(stringdatetime,
'%Y-%m-%dT%H:%M:%S'))))

#On applique la fonction ci-dessus à toutes nos données.

CO22= pd.read_csv('CO22.csv', sep=',')
CO22["datetime"]=CO22["time"].apply(stringdate_to_datetime)

chauffage2= pd.read_csv('chauffage2.csv', sep=',')
chauffage2["datetime"]=chauffage2["time"].apply(stringdate_to_datetime)

fenetre2=pd.read_csv('fenetre2.csv', sep=',')
fenetre2["datetime"]=fenetre2["time"].apply(stringdate_to_datetime)

bruit=pd.read_csv('Bruit.csv', sep=',')
bruit["datetime"]=bruit["time"].apply(stringdate_to_datetime)

mergeData=[chauffage2, fenetre2,CO22,bruit]
result=pd.concat(mergeData, axis=1, join='inner')

print(result)
```

```
fenetre2.datetimes=pd.to_datetime(fenetre2.iloc[:,0],format='%Y-%m-%dT%H:%M:%S')
hours=[]
for dt in fenetre2.datetime:
    hours.append(dt.hour)
fenetre2['hour']=hours
print("hours=",hours)
```

#On arrange nos données pour qu'à chaque instant soit associé une série de données

```
fenetre2.set_index('datetime', inplace=True)
print("isnull",fenetre2.isnull().any())
fenetre2.ffill()
newfenetre2=fenetre2.resample('30T').sum()
print(newfenetre2)
```

```
CO22.set_index('datetime',inplace=True)
print("isnull",CO22.isnull().any())
CO22.ffill()
newCO22=CO22.resample('30T').mean()
print(newCO22)
```

```
chauffage2.set_index('datetime',inplace=True)
print("isnull",chauffage2.isnull().any())
chauffage2.ffill()
newchauffage2=chauffage2.resample('30T').sum()
print(newchauffage2)
```

```
bruit.set_index('datetime',inplace=True)
print("isnull",bruit.isnull().any())
bruit.ffill()
newbruit=bruit.resample('30T').sum()
print(bruit)
```

```
"""hours=[]
for dt in fenetre2.datetime:
    hours.append(dt.hour)
fenetre2['hour']=hours"""
```

```
#On regroupe le tout dans un tableau
mergeData=[newfenetre2,newCO22,newchauffage2,newbruit]
result=pd.concat(mergeData,axis=1, join='inner')

print (result)
```

```
#Études des data SANS DecisionTree
```

```
Lis,T=[],[]
for i in range(len(result.chauffage)):
    if (result.CO2[i]<1000) and (result.fenetre[i]>0) and (result.chauffage[i]>0):
        #print("fermez fenetre")
        Lis.append(1)
        T.append(i)
        print(result.iloc[i:i+1,:])
    else:
        Lis.append(0)
        T.append(i)

plt.plot(T,Lis)
plt.title("détection de pertes d'énergie")
plt.xlabel("temps (h)")
plt.show()
```

```
#Data Tree
```

```
Label=[]
for i in result.value:
    if i > 0:
        Label.append(1)
def decision_tree(x_train, x_test, y_train, y_test,D):
    clf=tree.DecisionTreeClassifier(max_depth=D,random_state=0)
    clf=clf.fit(x_train,y_train)
    pred=clf.predict(x_test)
    accuracy=clf.score(x_test,y_test)
    importance=clf.feature_importances_
    return importance,accuracy

x_train, x_test, y_train, y_test = train_test_split(result[["fenetre","CO2","chauffage"]],Label,
test_size=.3, random_state=0)
```

```
"""
```

```
def KMeans_clustering(jalousie):
    myarray_jalousie=np.array(jalousie).reshape(len(jalousie),1)
    km=KMeans(n_clusters=2, random_state=0)
    km.fit(myarray_jalousie)
    centers=km.cluster_centers_
    centers[centers<0]=0 #the minimization function may find very small negative
numbers, we threshold them to 0
    prediction=km.fit_predict(myarray_jalousie)
    return (prediction)"""
```