Rapport de projet Gestion énergie

Joseph PLOT
Clément ROBERT
Kelian JAMBU

Sommaire

Introduction

Lors de ce projet, nous allons apprendre à concevoir et à développer une box de gestion énergétique qui concerne une zone de vie exploitant des capteurs autonomes très basse consommation. Ces capteurs permettent de construire des tableaux de bord pour l'aide à l'exploitation et l'intégration de stratégies de gestion énergétique rudimentaire. Il s'agit d'un projet global dont l'élaboration s'effectue en équipes collaboratives de plusieurs étudiants sur un mode mini-entreprise.

I. Introduction au deep learning et au traitement de données

Après avoir redécouvert le langage de programmation Python, nous avons dans un premier temps appris à construire un arbre de décisions qui se base sur un fichier csv (à l'aide du module panda) dont on doit extraire les données. Après la discrétisation des labels, nous avons défini les caractéristiques les plus importantes (Tin, Tout, etc). On a ensuite divisé les données caractéristiques en deux catégories: formation et test. Après l'implémentation du decision tree, il fallait calculer la précision ou la marge d'erreur. Après modification de la profondeur de l'arbre, on pouvait tracer la précision en fonction de la profondeur. On a ensuite tracé l'arbre de décisions et rendu le modèle flexible (relation précision coût).
Enfin, nous pouvions tracer l'occupation réelle par rapport à l'occupation prédite.

Ci-dessous, voici l'algorithme que nous avons réalisé lors des premières séances:

```python
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn import metrics

#read fiche .csv

import pandas as pd
data = pd.read_csv('data.csv', sep= ';')
Temperature=data['Tin']

labelcopy= []
for k in range (len(data['label'])):
    if ((type(data['label'])) != k):
        if (data['label'][k] <=0.5):
            labelcopy.append(0)
        else:
            labelcopy.append(1)

data['labelcopy']=labelcopy
print(data)

# for k in range(len(data['power'])):
#     if data['power'][k] ==0 and data['office_CO2_concentration'][k] < 400 and data['CO2_corridor'][k]<400:
#         print("There's no one")
#     elif data['power'][k] > 0:
#         if data['Door'][k]!=0:
#             print("There's at least one person")
#         if data['acoustic_pressure_dB'][k]!=0:
#             print("There's at least one person")


# Train_test_split
```

```python

xtrain, x_test, y_train, y_test = train_test_split (
    data[["Tin","Tout", "humidity", "detected_motions","power","office_CO2_concentration","Door","CO2_corrid
    data[["labelcopy"]], test_size = .3, random_state = 0)


# Implement the DT
def decision_tree(x_train, x_test, y_train, y_test,D):
    clf = tree.DecisionTreeClassifier(max_depth=D)
    clf = clf.fit(x_train, y_train)
    y_test_pred = clf.predict(x_test)
    accuracy = clf.score(x_test, y_test)
    print (accuracy)
    importance = clf.feature_importances_
    print ("Features importance", importance)
    indices = np.argsort(importance)
    #plt.title("Feature Importances")
    #plt.bar(range(len(indices)), importance[indices], color='r', align='center')
    #plt.yticks(range(len(indices)), [x_train.columns.values[i] for i in indices])
    #plt.xlabel("Relative Importance")
    with open("MyTree.dot", 'w') as file:
        tree.export_graphviz(clf, out_file=file)
    print(metrics.classification_report(y_test, y_test_pred,target_names=['Level 0', 'Level 1']))
    return y_test_pred

x_train, x_test, y_train, y_test = train_test_split(
    data[["Tin" , "Door", "detected_motions", "Tout", "humidity"]],
    data[['labelcopy']], test_size=.2, random_state=0)
y_pred_occupancy= decision_tree(x_train, x_test,y_train, y_test,5)

depth=[1,2,3,4, 5, 6, 7, 8, 9, 10, 11]
accuracy=[0.34,0.61,0.6,0.69,0.62,0.63, 0.65,0.64, 0.63, 0.69, 0.67]
#plt.plot(depth,accuracy)
plt.show()
```

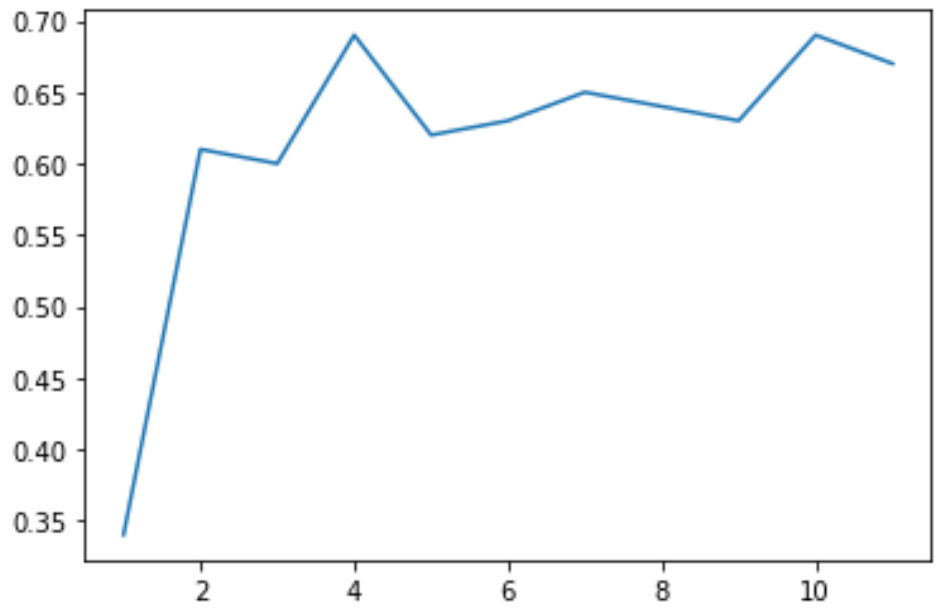
Le tableau ci-dessous nous renvoie l'importance des données dans l'arbre des décisions. On remarque que la troisième donnée du tableau, à savoir le power, a un poids plus important que les autres données puisqu'il influe à quasiment 92% sur notre arbre.

```
Features importance [0.02898854 0.02365949 0.91871176 0.01182669 0.01681353]
              precision    recall  f1-score   support

     Level 0       0.99      0.94      0.96       116
     Level 1       0.84      0.97      0.90        38

    accuracy                           0.95       154
   macro avg       0.92      0.96      0.93       154
weighted avg       0.95      0.95      0.95       154
```

Le graphe ci-dessous représente la précision en fonction de la profondeur de l'arbre. Globalement, on remarque à l'aide de ce graphe que plus la profondeur de l'arbre augmente, plus la précision est grande. Cependant, on remarque un petit pic local pour une profondeur de 4, ce qui signifie qu'il peut être inutile de choisir un arbre de décision à la complexité trop importante.

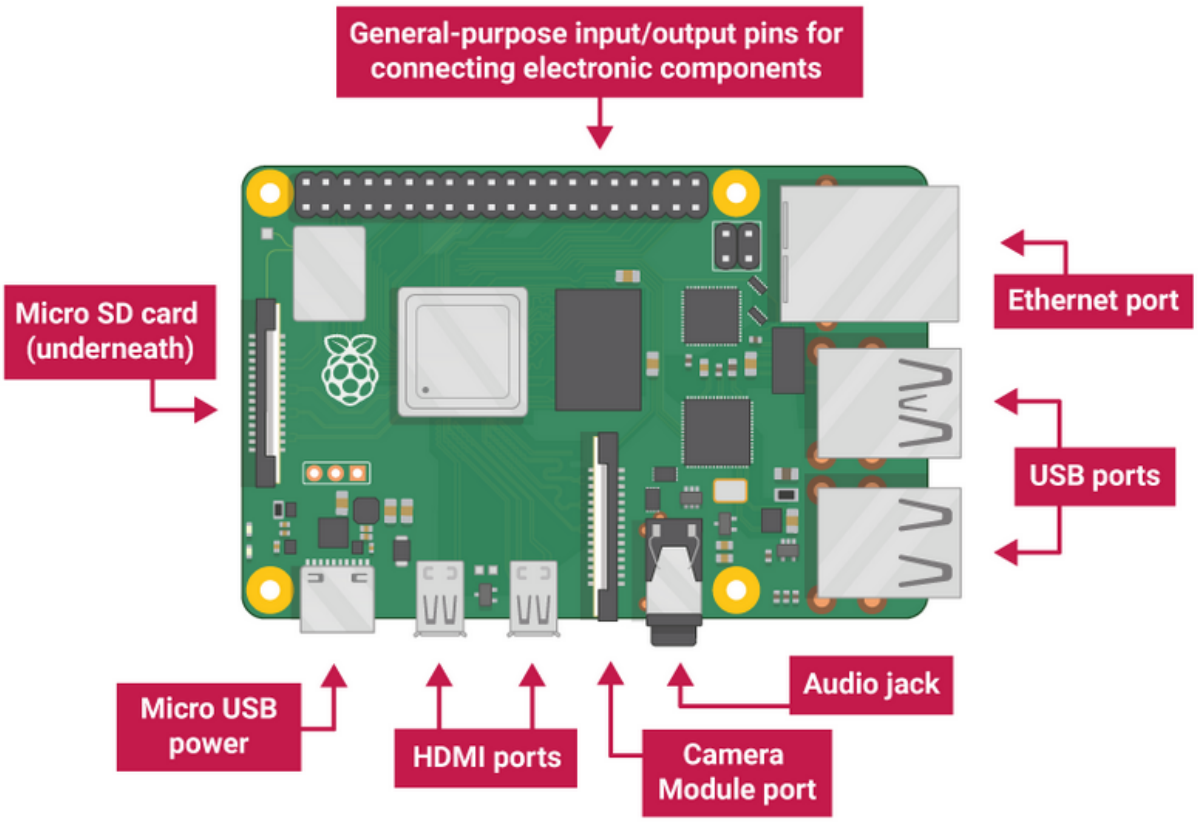


II. Initiation à la domotique sur Jeedom

Le logiciel Jeedom est Open Source, qui donne un accès total au logiciel qui gère la domotique d'un individu.
De son côté, Grafana est une application web open-source d'analyse et de visualisation interactive. Elle permet d'ingérer des données provenant d'un grand nombre de sources de

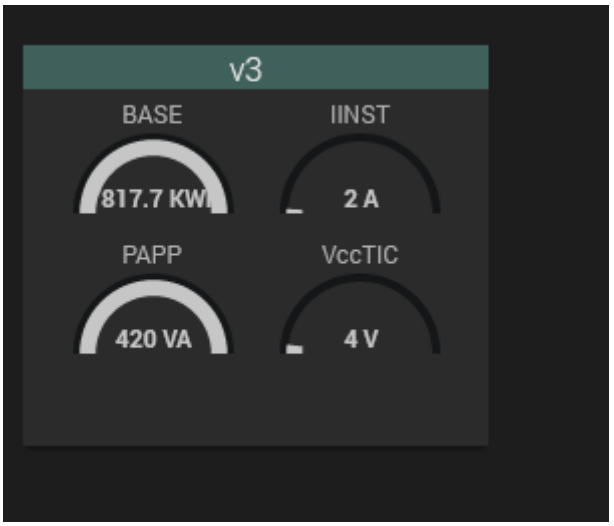
données, de les interroger et de les afficher sur de beaux graphiques personnalisables pour en faciliter l'analyse.

Grâce à Jeedom, nous avons le pouvoir de gérer de grands projets de domotique de manière très intuitive.
Une ébauche de cela a été directement réalisée en cours : armés d'une LED, nous avons tenté de provoquer son allumage à distance, à l'aide aussi d'un raspberry Pi. Voici là encore une notion un peu particulière : le Raspberry Pi est un ordinateur très bon marché qui fonctionne sous Linux, mais il fournit également un ensemble de broches GPIO (general purpose input/output), nous permettant de contrôler des composants électroniques pour l'informatique physique et d'explorer l'Internet des objets (IoT).
En voilà la disposition :



Comment gérer la domotique via Jeedom ?
Il faut pour cela créer notre projet (nous utilisions, pour notre part, la "Maison de Kélian"), à laquelle nous avons virtuellement ajouté plusieurs pièces.

Afin d'achever le "protocole domotique", nous avons ajouté un Broker qui nous permet de mettre les différentes caractéristiques des pièces en lien avec le raspberry grâce à l'obtention d'une "APIkey".
On obtient un écran de ce type :



Puis, nous pouvons, par l'intermédiaire d'un code, contrôler nous mêmes les données échangées dans les différentes pièces : par exemple, il nous serait possible de créer une alarme au cas où la consommation d'énergie serait superflue ou trop importante. On peut définir une valeur seuil à ne pas dépasser, puis un signal lumineux qui nous signalerait d'éventuels déplacements.

Bien évidemment, il faut disposer de beaucoup de données diverses afin de déterminer si de l'énergie est consommée à tort, ou si les dépenses sont justifiées.
On peut notamment mettre en rapport des caractéristiques comme la concentration de CO2 dans une pièce, la luminosité et le bruit au cours d'une journée dans une pièce afin d'en déterminer le nombre de personnes qui se situent en son sein.

Il nous sera alors plus facile de gérer l'énergie dans notre bâtiment, et c'est ce que nous allons faire dans la partie suivante.

III. Gestion d'énergie d'un logement habité

Dans cette partie, on s'intéresse directement à la gestion d'énergie d'un logement habité. Le code Python réalisé pour cette partie est présent ci-dessous, et l'on peut commenter ce code:

```python
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn import metrics
import pandas as pd

import time
import datetime

def stringdate_to_datetime(stringdatetime: str):
    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')))


TV_power= pd.read_csv('/Users/Utilisateur/Documents/Ense3/Projet G11/Données/TV_power.csv', sep= ',')
TV_power["datetime"]=TV_power["time"].apply(stringdate_to_datetime)

Box_TV= pd.read_csv('/Users/Utilisateur/Documents/Ense3/Projet G11/Données/Box_TV.csv', sep= ',')
Box_TV["datetime"]=Box_TV["time"].apply(stringdate_to_datetime)

Bruit= pd.read_csv('/Users/Utilisateur/Documents/Ense3/Projet G11/Données/Bruit.csv', sep= ',')
Bruit["datetime"]=Bruit["time"].apply(stringdate_to_datetime)

Lumière= pd.read_csv('/Users/Utilisateur/Documents/Ense3/Projet G11/Données/Lumière.csv', sep= ',')
Lumière["datetime"]=Lumière["time"].apply(stringdate_to_datetime)

Luminosité= pd.read_csv('/Users/Utilisateur/Documents/Ense3/Projet G11/Données/Luminosité.csv', sep= ',')
Luminosité["datetime"]=Luminosité["time"].apply(stringdate_to_datetime)
```

Dans la partie du code ci-dessus, on importe tout d'abord les différents modules nécessaires au fonctionnement du programme, entre autres les modules pandas et time. Dans un premier temps, il s'agit de lire les fichiers csv qui nous avaient été fournis, à l'aide du module pandas, puis on ajoute la colonne temps (datetime) sous le format défini par la fonction 'stringdate_to_datetime' définie un peu plus haut, qui enlève les chaînes de caractères.

```python
Fenêtre= pd.read_csv('/Users/Utilisateur/Documents/Ense3/Projet G11/Données/Fenêtre.csv', sep= ',')
Fenêtre["datetime"]=Fenêtre["time"].apply(stringdate_to_datetime)

Humidité= pd.read_csv('/Users/Utilisateur/Documents/Ense3/Projet G11/Données/Humidité.csv', sep= ',')
Humidité["datetime"]=Humidité["time"].apply(stringdate_to_datetime)

Température= pd.read_csv('/Users/Utilisateur/Documents/Ense3/Projet G11/Données/Température.csv', sep= ',')
Température["datetime"]=Température["time"].apply(stringdate_to_datetime)

CO2= pd.read_csv('/Users/Utilisateur/Documents/Ense3/Projet G11/Données/CO2.csv', sep= ',')
CO2["datetime"]=CO2["time"].apply(stringdate_to_datetime)


mergeData=[TV_power,Box_TV,Bruit,Lumière,Luminosité,Fenêtre,Humidité,Température,CO2]
result=pd.concat(mergeData, axis=1, join='inner')

print(result)

TV_power.set_index('datetime',inplace=True)
print("isnull",TV_power.isnull().any())
TV_power.ffill()
newTV_power=TV_power.resample('30T').sum()
print(newTV_power)


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

Ici, on continue de récupérer et de lire les données qui relèvent la présence ou non d'individus dans la maison et qui font état de l'utilisation de l'énergie dans la maison. Ensuite, on modifie ces données au niveau de la période (resample de 30T).

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


Lumière.set_index('datetime',inplace=True)
print("isnull",Lumière.isnull().any())
Lumière.ffill()
newLumière=Lumière.resample('30T').sum()
print(newLumière)


Luminosité.set_index('datetime',inplace=True)
print("isnull",Luminosité.isnull().any())
Luminosité.ffill()
newLuminosité=Luminosité.resample('30T').sum()
print(newLuminosité)


Fenêtre.set_index('datetime',inplace=True)
print("isnull",Fenêtre.isnull().any())
Fenêtre.ffill()
newFenêtre=Fenêtre.resample('30T').sum()
print(newFenêtre)


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

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

CO2.set_index('datetime',inplace=True)
print("isnull",CO2.isnull().any())
CO2.ffill()
newCO2=CO2.resample('30T').sum()
print(newCO2)

mergeData2=[newTV_power,newBox_TV,newBruit,newLumière,newLuminosité,newFenêtre,newHumidité,newTempérature,newCO2]
result2=pd.concat(mergeData2, axis=1, join='inner')

print(result2)

#result2.set_index('datetime',inplace=True)

hours=[]
result2_datetimes=pd.to_datetime(Luminosité.iloc[:,0], format='%Y-%m-%dT%H:%M:%S')

for dt in result2_datetimes:
    hours.append(dt.hour)

hours2=[]
for k in range (146):
    hours2.append(hours[k])

journée=[00.50 , 1,1.5,2,2.5,3,3.5,4,4.5,5,5.5,6,6.5,7,7.5,8,8.5,9,9.5,10,10.5,11,11.5,12,12.5,13,13.5,14,14.5,15,15.5,16,16.5,17,17.5,18,

hours3=[]
journée2=journée*10
for i in range(146):
    hours3.append(journée2[i+35])

result2['Heure']=hours3
```

Dans les deux parties ci-dessus, on continue la création des nouvelles températures, humidités etc., que l'on place ensuite dans la liste 'mergeData2', que l'on obtient sous forme de tableau avec 'result2'. On créé le tableau des heures de la journée que l'on associe avec le tableau result2, ce qui nous sera utile pour associer les pertes d'énergies à ces heures de la journée (précision: la demi-heure est de 0.5 dans notre tableau).

```
146
147   # Occupancy
148
149   occupancy= []
150   for k in range (len(hours3)):
151       if 17<=hours3[k]<=22:
152           occupancy.append(1)
153       elif 7<=hours3[k]<=9:
154           occupancy.append(1)
155       else:
156           occupancy.append(0)
157
158
159   #Waste energy
160
161   window=result2['Fenêtre']
162   CO2_2=result2['CO2']
163   light=result2['Lumière']
164   TV=result2['TV_power']
165
166   for k in range (len(result2)):
167       if window[k] ==1 and CO2_2[k]<=2000:
168           print('Waste of energy at', hours3[k], "h, because the window is open")
169       # else:
170       #     print('OK. Heure:',hours3[k])
171
172   for k in range (len(result2)):
173       if (occupancy[k] ==0) and (light[k]==1 or TV[k]!=0):
174           print('Waste of energy at', hours3[k], "h, because of the light or the TV")
175
176
```

On s'attèle alors à l'occupancy (occupation de la maison par un ou des potentiels individus), qui est ici sous forme de liste. On sait que l'occupation compte de 17h à 22h et de 7h à 9h: l'occupation est alors de 1, elle est de 0 le reste du temps.
Pour le gaspillage d'énergie (waste of energy), on prend en compte les données que l'on dispose. Suivant ce que l'on souhaite prendre en compte, on peut afficher directement l'heure et la cause de ce gaspillage d'énergie
E.n définissant nous-même le vecteur "Occupancy", nous obtenons en lançant le programme:

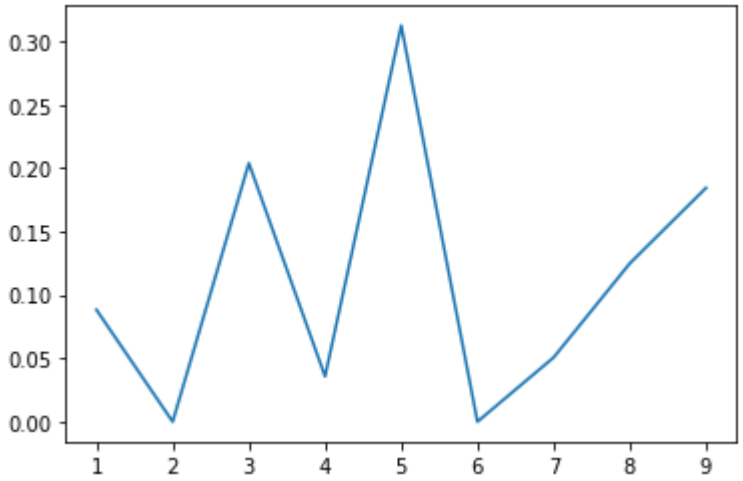
```
Waste of energy at 7 o'clock, because the window is open
Waste of energy at 14.5 o'clock, because of the light or the TV
Waste of energy at 15.5 o'clock, because of the light or the TV
Waste of energy at 16.5 o'clock, because of the light or the TV
Waste of energy at 12.5 o'clock, because of the light or the TV
Waste of energy at 13.5 o'clock, because of the light or the TV
Waste of energy at 14.5 o'clock, because of the light or the TV
Waste of energy at 15.5 o'clock, because of the light or the TV
Waste of energy at 4 o'clock, because of the light or the TV
Waste of energy at 4.5 o'clock, because of the light or the TV
Waste of energy at 5 o'clock, because of the light or the TV
Waste of energy at 6 o'clock, because of the light or the TV
Waste of energy at 10 o'clock, because of the light or the TV
Waste of energy at 12 o'clock, because of the light or the TV
Waste of energy at 14 o'clock, because of the light or the TV
Waste of energy at 15 o'clock, because of the light or the TV
Waste of energy at 15.5 o'clock, because of the light or the TV
Waste of energy at 16 o'clock, because of the light or the TV
```

On a comme repère temporel l'heure uniquement, nos données étant sur 4 jours, nous aurions pu par un algorithme plus complexe déterminer le jour exact. Ce n'était pas chose aisée en raison de la particularité de notre tableau 'return2' qui prend la forme d'un tableau mais n'en a pas toutes les propriétés. Ainsi nous n'avons pu utiliser la

colonne contenant la date et l'heure exacte de la prise de données. (Mme Amayri n'a pu nous aider sur ce point).

```
171  def decision_tree(x_train, x_test, y_train, y_test):
172      clf = tree.DecisionTreeClassifier(max_depth=5)
173      clf = clf.fit(x_train, y_train)
174      y_test_pred = clf.predict(x_test)
175      accuracy = clf.score(x_test, y_test)
176      importance=clf.feature_importances_
177      return importance,accuracy
178
179
180
181  x_train, x_test, y_train, y_test = train_test_split (
182      result2[["TV_power","Box_TV", "Bruit", "Lumière","Luminosité","Fenêtre","Humidité","Température","CO2"]
183      occupancy, test_size = .3, random_state = 0)
184
185  y_pred_occupancy= decision_tree(x_train, x_test,y_train, y_test)
186
187  depth=[1,2,3,4, 5, 6, 7, 8,9]
188
189
190  plt.plot(depth,y_pred_occupancy[0])
191  plt.show()
```

En ce qui concerne l'arbre de décision (decision tree) on peut définir une fonction qui retourne l'importance des caractéristiques (features) et la précision de l'arbre. On peut alors afficher le graphique de l'importance en fonction de la profondeur de l'arbre (depth).



Nous ne connaissons pas réellement les attentes concernant ce graphe, et ne pouvons donc le commenter.

Conclusion

Ce projet nous a permis de découvrir de nouveaux domaines comme le traitement de données ou les systèmes intelligents en informatique. Après nous avoir permis de revoir et d'approfondir le langage Python, on a pu découvrir le logiciel Jeedom et coder différentes fonctions comme l'arbre de décisions qui est capable de nous présenter l'importance et la précision de certaines caractéristiques de la maison en

fonction de la profondeur de l'arbre, en ce qui concerne le gaspillage de l'énergie. On se rend compte qu'il est alors possible de définir quand exactement il y a gaspillage énergétique suivant les données que l'on dispose. A long terme, il est vraiment possible d'adapter et de réduire sa consommation énergétique.