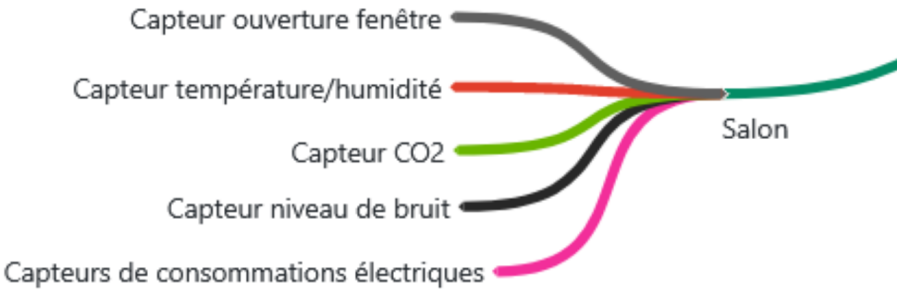
OLIVEIRA Victor
PHANTHAVONG Hugo
03/02/2021

Projet Gestion d’énergie



Objectifs

L’objectif de ce projet est de déterminer s’il y a gaspillage d’énergie dans une pièce grâce aux informations récoltées par des capteurs. On apprendra ainsi à extraire les données des capteurs d’un serveur, puis à les traiter afin de pouvoir travailler avec, et enfin à les utiliser pour construire un arbre de décisions qui permet de déterminer l’occupation ou non de la pièce. On définit le gaspillage d’énergie s’il y a occupation et les données sur le chauffage, la fenêtre et le CO2.

Source de nos données

Dans le cadre de notre projet, on travaillera sur les données récoltées du SmartHouse du G2Elab et on choisit de travailler plus précisément dans le salon où cinq capteurs sont présents.

Les données des capteurs sont récoltées et stockées par le réseau InfluxDB. On utilise le serveur Jeedom. Le tout se fait sur Python.

```
7 ############################
8 #    SCRIPT SETTINGS
9 ############################
10 # Set the port where you want the bridge service to
   run
11 PORT_NUMBER = 1234
12 # InfluxDB Server parameters
13 INLUXDB_SERVER_IP = '82.65.155.71'
14 INLUXDB_SERVER_PORT = 8086
15 INFLUXDB_USERNAME = 'eleves'
16 INFLUXDB_PASSWORD = 'SmarthouseG2Elab'
17 INFLUXDB_DB_NAME = 'jeedom'
18 ############################
19
20 client = InfluxDBClient(INLUXDB_SERVER_IP,
   INLUXDB_SERVER_PORT, INFLUXDB_USERNAME,
   INFLUXDB_PASSWORD, INFLUXDB_DB_NAME,ssl=True,
   verify_ssl=False)
21
22 print(client.get_list_database())
23
24 client.switch_database('jeedom')
25
```

Lecture des données et mise en forme des données

On choisit de travailler sur les données sur un an. On convertit les données issues de InfluxDB en fichier CSV.

```
26 datasheet = client.query('SELECT "value" FROM "jeedom
   "."autogen"."885" WHERE time > now() - 365d')
27 print(datasheet)
28
29 # Data
30 #Fenêtre Gauche = 4286 Unité Binaire 0 fermé 1 ouvert
31 #Fenêtre Droite = 4281 Unité Binaire 0 fermé 1 ouvert
32 #Fenêtre = 885
33 #TV Power = 1372  Unité W  ok
34 #Box TV = 1392 Unité W
35 #CO2= 3887 ppm
```

```
#Température = 3889 °C
#Humidité = 3892 %
#Lumière = 516 Unité Binaire 0 éteint 1 Allumé
#Bruit = 3888 Unité Décibel
#Luminosité = 7029 Lux
#Chauffage = 2022 Unité Binaire 0 éteint 1 allumé

exported_data = list(datasheet.get_points())
header_list = list(exported_data[0].keys())

with open("outputFenetre.csv", "w", newline='') as fp
:
    writer = csv.writer(fp, dialect='excel')
    print(header_list[1:])
    value_header = header_list[1]
    offset = sum(c.isalpha() for c in value_header)
    print(offset)
    #header_list[1:] = sorted(header_list[1:], key=
lambda x: int(x[offset:]))
    header_list[1:] = ['value']
    # print(header_list)
    writer.writerow(header_list)
    for line in exported_data:
        # print(line)
        writer.writerow([line[kn] for kn in
header_list])

```

On convertit les données en tableau grâce au module Pandas.

```
# Read CSV file
data = pd.read_csv('outputChauffage.csv', sep=',')
data["datetime"]=data["time"].apply(
stringdate_to_datetime)

data1 = pd.read_csv('outputCO2.csv', sep=',')
data1["datetime"]=data1["time"].apply(
stringdate_to_datetime)

data2 = pd.read_csv('outputFenetre.csv', sep=',')
data2["datetime"]=data2["time"].apply(
stringdate_to_datetime)
```

On convertit au préalable la colonne date en une donnée travaillable. Les données ne sont pas relevées au même temps. Les données des différents capteurs sont relevées à différents temps, il faut alors les mettre à une même période. Ainsi, on remplit par des nan les temps qui n’ont pas de valeurs, et on réunit et somme les valeurs qui sont situées entre.

```
# Read time
def stringdate_to_datetime(stringdatetime: str):
 # transform a date string representation into an internal datetime representation
 # :param stringdatetime: date string representation '%d%m%Y %H:%M:%S'
 # :return: internal datetime representation
 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')))
# data
data.set_index('datetime', inplace=True)
print("isnull", data.isnull().any()) #find gaps
data.ffill()
Chauffage=data.resample('20T').sum()

# data1
data1.set_index('datetime', inplace=True)
print("isnull", data1.isnull().any()) #find gaps
data1.ffill()
CO2=data1.resample('20T').sum()

# data2
data2.set_index('datetime', inplace=True)
print("isnull", data2.isnull().any()) #find gaps
data2.ffill()
Fenetre=data2.resample('20T').sum()

# Merge data
mergeData=[Chauffage,CO2,Fenetre]
result=pd.concat(mergeData, axis=1, join='inner')
result.columns = ['chauffage', 'CO2', 'fenetre']
```

Mise en place d’un arbre de décisions pour détecter la présence de quelqu’un

Etude préalable pour créer un arbre de décisions

Objectif : On souhaite à partir des données des capteurs créer un arbre de décisions.

Un arbre de décisions met à différentes priorités les capteurs. Il teste rapidement si une valeur est “impossible” si elle vérifie un seuil sinon il continue le test avec les autres capteurs. Pour déterminer quel est le seuil, l’arbre se construit/s’entraîne à partir de données qu’on lui fournit.

Objectif secondaire : On souhaite déterminer quels capteurs sont les plus pertinents pour la détection de présence.

Les données utilisées pour la création/l’entraînement de l’arbre de décisions

On sépare en deux les données à 30% en données pour construire le modèle (x_train, y_train) et à 70% en données pour tester la précision de notre modèle (x_test, y_test).

```
21 # Divide the data into training and testing (70/30
   different days)
22 x_train, x_test, y_train, y_test = train_test_split(
23  data[["Tin", "Tout", "humidity", "detected_motions"
   , "power", "office_CO2_concentration", "Door", "
   CO2_corridor", "acoustic_pressure_dB"]],
24  data[['newlabel']], test_size=.3, random_state=0)
```

Recherche des capteurs les plus performants pour déterminer l'occupation

Grâce à cette fonction, on trouve quels capteurs sont les plus performants dans l'arbre de décisions.

```
26 # Define the most important features :
   feature_importances_
27 def importance(x_train, y_train):
28  clf = tree.DecisionTreeClassifier(random_state=0,
   criterion='entropy', max_depth=None)
29  clf = clf.fit(x_train, y_train)
30  features_importance = clf.feature_importances_
31  return features_importance
```

Création de l'arbre de décisions

On implémente notre arbre de décision à partir du module DecisionTree.

```
33 #Implementation of decision tree
34 def decision_tree(x_train, y_train, x_test, y_test):
35  clf= tree.DecisionTreeClassifier(random_state=0,
35 criterion='entropy', max_depth=None)
36  clf= clf.fit(x_train, y_train)
37  y_test_pred= clf.predict(x_test)
```

Précision de notre arbre de décisions

On veut vérifier la précision de notre arbre de décisions avec cette fonction.

```
42  #Accuracy
43  accuracy = clf.score(x_test,y_test)
44  print("accuracy is", accuracy)
```

Application à notre cas

Cette arbre de décisions se fait par classification car on a accès pour la construction du modèle de la donnée occupation. Dans notre cas, on n'a pas cette donnée occupation. Il faudrait alors utiliser une méthode clustering pour créer notre modèle, mais on se facilite ici drastiquement la tâche en restant à une méthode classification en créant la donnée occupancy = 1 entre 8h et 21h.

Malheureusement, on n'a pas réussi à implémenter l'arbre de décisions.

Critères de gaspillage d’énergie

On définit un gaspillage d’énergie par le salon a la fenêtre ouverte, l’aération est suffisante (le CO2 de la pièce est faible), la chaudière est allumée et personne est dans la pièce.

```
69 # Waste of energy
70 waste_of_energy = []
71 chauffage = result['chauffage']
72 CO2 = result['CO2']
73 fenetre = result['fenetre']
74 for k in range(len(chauffage)):
75  if (chauffage[k]>0) and (CO2[k]<1000) and (fenetre[
   k]>0.2) :
76   waste_of_energy.append(1)
77  else :
78   waste_of_energy.append(0)
79 result['waste_of_energy']= waste_of_energy
```

On range les données dans un CSV et on peut ainsi observer à quelle période le salon est en gaspillage d’énergie.

Conclusion

Notre cas d’étude a permis de découvrir comment on traitait les informations récoltées par les capteurs, ensuite on a pu travailler avec les données sur Python. On a pu voir comment fonctionnait et se construisait un arbre de décisions. Sur Python, on se sent libre d’entreprendre car des fonctions sont développées en opensource.
Une fois le processus acquis, la méthode est la même pour d’autres projets tout aussi variés.

Le but de cet article est ainsi de pouvoir démocratiser d’autres projets de la sorte qui sont finalement assez facile d’accès. Il est question simplement de s’initier.