Week8_DataCheck-Copy1

November 17, 2020

1 SVR toepassen

In dit programma wordt een SVR gemaakt op de solar data.

modules: Importeer de modules:

[1]:
```
#Import modules:
import numpy as np
import pandas as pd
from tqdm import tqdm
import matplotlib.pyplot as plt
import glob
from datetime import datetime
from datetime import timedelta

from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

variabelen: Zet alle variabelen op de juiste waardes:

[2]:
```
#define all variables:
loadpath_np = "/home/18005152/notebooks/zero/DATA/"
sdate = "2019-02-05" #start datum
edate = "2019-02-28" #eind datum
variables = 1 #The amount of variables chosen on the KNMI website.
path = 'KNMI_20201016_hourly_full.txt' #naam van weerdata file


#maak de dictionairies:
sheet_dict = {'solar': [3, 'Out'],
              'smartMeter': [7, 'Out'],
              'energyWtwReg': [2, 'In'],
              'energyHeatpump': [2, 'In']}
```

1.1 Functions

Hier worden alle functies gemaakt.

[3]:
```
#functions
def Jef_Func():
    houses = list()
    for i in range(1,2):
        df_delta = pd.DataFrame()
        for sheetname, j in sheet_dict.items():
            df = pd.DataFrame(np.load(loadpath_np + sheetname + '_' +␣
 ↪nParse3(i) + '.npy'))
            df = df.set_index(pd.DatetimeIndex(pd.to_datetime(df[0],unit='s').
 ↪values))
            df = df.resample('5min').sum()

            if sheetname == 'smartMeter':
                col = df[6].shift(-1) - df[6]
                col = col.shift(1)
                df_delta[str(sheetname)+'In'+'_delta'] = col

            col = df[j[0]].shift(-1) - df[j[0]]
            col = col.shift(1)
            df_delta[str(sheetname)+j[1]+'_delta'] = col

        houses.append(df_delta)
    return houses

#functions
def nParse3(n):
    """output a string that is 3 decimals long. Levy en Jefry kunnen uitleg␣
 ↪geven"""
    number = str(n)
    if len(number) == 1:
        number = "00" + number
    elif len(number) == 2:
        number = "0" + number
    elif len(number) == 3:
        number = number
    return str(number)

def scale(data):
    scaler = StandardScaler()
    scaler.fit(data)
    data = scaler.transform(data)
    return data
```

[4]:
```
def GetWeatherData(date0,date1):
    # source for the file: http://projects.knmi.nl/klimatologie/uurgegevens/
↪selectie.cgi
    df = pd.DataFrame()

    #reading file
    weer = pd.read_csv(path, header = variables+10)
    weer = weer.drop(0).drop(['# STN'], axis =1)

    #fix columns date and time
    weer['YYYYMMDD'] = weer['YYYYMMDD'].apply(lambda x: int(x))
    weer['   HH'] = weer['   HH'].apply(lambda x: int(x))

    ################################################################
    ###fix columns extra                                           #
    weer['    Q'] = weer['    Q'].apply(lambda x: int(x))      #

    ################################################################

    #format datetime string
    weer['datetime'] = pd.to_datetime(weer['YYYYMMDD'], format='%Y%m%d')
    df['date'] = weer.apply(lambda x: x['datetime']+timedelta(days=1) if x['  ␣
↪HH']==24 else x['datetime'], axis=1)
    df['date'] = df.apply(lambda x: x['date'].strftime('%Y%m%d'), axis=1)
    df['hour'] = weer.apply(lambda x: 0 if x['   HH']==24 else x['   HH'],␣
↪axis=1)
    df_04_to_05_is_05 = df

    #create a dataframe containing all columns
    data = df_04_to_05_is_05
    knmi = pd.DataFrame()
    knmi['Date'] = data['date']
    knmi['Hour'] = data['hour'].apply('{:0>2}'.format)

    ################################################################
    #Add the info columns                                          #
    knmi['straling'] = weer['    Q']                              #

    ################################################################

    #Create datetime index
    knmi['DateTime'] = knmi['Date'].astype(str) + knmi['Hour'].astype(str)
    knmi['DateTime'] = pd.to_datetime(knmi['DateTime'], format='%Y%m%d%H')

    #drop unneccesary columns
    knmi = knmi.set_index(knmi['DateTime']).drop(['Date', 'Hour', 'DateTime'],␣
↪axis=1)
```

```
    return knmi[date0:date1]
```

1.2 Hoofd loop:

This is where the program will be ran.

1.2.1 train the SVR

[1]:
```
#get the data:
houses = Jef_Func()
df = houses[0]['solarOut_delta']
df = df.resample('60min').sum()

#prepare the data:
y = (df[sdate:edate]-(df[sdate:edate].values)[0]).values.reshape(-1,1).tolist()
y = np.array([i[0] for i in y])
x0 = (((df[sdate:edate].index).values).reshape(-1,1)).tolist()
x1 = (GetWeatherData(sdate,edate).values).reshape(-1,1).tolist()
x2 = df[sdate:edate].index.day
x3 = df[sdate:edate].index.hour
x4 = df[sdate:edate].index.month

#met scaler:
x = scaler(np.array([[x0[i][0],x1[i][0],x2[i],x3[i],x4[i]] for i in␣
 ↪range(0,len(x0))]))
#zonder scaler:
x = (np.array([[x0[i][0],x1[i][0],x2[i],x3[i],x4[i]] for i in␣
 ↪range(0,len(x0))]))

#split and scale the data:
X_train, X_test, y_train, y_test = train_test_split(x,y, test_size=5/24,␣
 ↪random_state=0,shuffle=False)


#train the model:
svr = SVR(C=1)
svr.fit(X_train,y_train)

#make data plot ready:
test = [i[0] for i in X_test]
train = [i[0] for i in X_train]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-d960a40d9664> in <module>
```

```
      1 #get the data:
----> 2 houses = Jef_Func()
      3 df = houses[0]['solarOut_delta']
      4 df = df.resample('60min').sum()
      5

NameError: name 'Jef_Func' is not defined
```

1.2.2 Plot results SVR

[10]:
```
%matplotlib notebook
plt.subplots(figsize=(16
                      ,5))
plt.plot(train,svr.predict(X_train),color='black',label="SVR prediction")
plt.plot(test,svr.predict(X_test),color='g',label="SVR prediction")
plt.scatter(train,y_train,color="r",label="Training Data")
plt.scatter(test,y_test,color="b",label="Testing Data")

#Nice layout:
plt.xlabel("Unix Time [s]")
plt.ylabel("Solar production hourly [kWh]")
plt.legend(loc="upper right")
plt.title("R^2= "+str(svr.score(X_test,y_test)))
plt.grid()
plt.show()
plt.savefig("SVG_500.png",dpi=1000)

#print de score:
print("R2-score test data:")
print(svr.score(X_test,y_test))
print("R2-score train data:")
print(svr.score(X_train,y_train))
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
R2-score test data:
-0.4265679001413081
R2-score train data:
-0.2418555437932941
```

1.2.3 Kijk wat de optimale parameter voor C is:

In de grafiek kan je zien wat de meest optimale waarde voor C is. Hierna kan je deze instellen, waardoor het model beter wordt.

[ ]:
```
start = 1
stop = 500
factor = 10
dick = {}

for qC in tqdm(range(start,stop*factor)):
    svr = SVR(C=qC/factor)
    svr.fit(X_train,y_train)
    r2 = svr.score(X_test,y_test)

    dick[str(qC)] = r2
```

```
 99%|      | 4954/4999 [09:12<00:10,  4.40it/s]
```

[79]:
```
x = [float(i)/1000 for i in dick.keys()]
y = dick.values()

%matplotlib notebook
plt.scatter(x,y,)
plt.xlabel("C-value")
plt.ylabel("r^2-score")
plt.grid()
plt.legend()
plt.show()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
No handles with labels found to put in legend.
```