W16_SVR_MultipleHouses_production

January 6, 2021

1 settings

[1]:
```
#settings:
show_every = 10
houses = [28,37,40,42,105,115,56,51,58,70,99,100]
```

2 Initialization

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from tqdm import tqdm

from IPython.display import display, HTML
import time
```

[3]:
```
import random
#Neural Network imports
import torch
import torch.nn as nn
import torch.optim as optim

from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_squared_error as mese
from sklearn.metrics import mean_absolute_error

from sklearn import linear_model
from sklearn.svm import SVR


scalerx = StandardScaler()
scalery = StandardScaler()
```

[4]:
```
#cuda imports
ngpu = torch.cuda.device_count() # number of available gpus
device = torch.device("cuda:4") if (torch.cuda.is_available() and ngpu > 0)␣
↪else "cpu" #cuda:0 for gpu 0, cuda:4 for gpu 5
torch.backends.cudnn.benchmark=True # Uses cudnn auto-tuner to find the best␣
↪algorithm to use for your hardware

#Random Seed
random.seed(1337)
torch.manual_seed(1337)
```

[4]:
```
<torch._C.Generator at 0x7febbcc32108>
```

Make all functions:

[5]:
```
def det(tensor):
    """
    Zet de tensor om van een tensor naar numpy op de CPU.
    """
    return tensor.cpu().detach().numpy()

def calculate_metrics_for_model(output,target):
    """
    Calculates all the desired evaluation metrics for the model.
    """
    yhat = scalery.inverse_transform(det(output))
    y = scalery.inverse_transform(det(target))
    actual, pred = np.array(y), np.array(yhat)

    mae = mean_absolute_error(yhat, y)
    mse = mese(yhat, y)
    mape = np.mean(np.abs((actual - pred) / actual)) * 100
    r2 = r2_score(yhat, y)
    return [mae, mse, mape, r2]

def GetGlobalData(nr):
    """
    Get the data for the MVLR, SVR and NN.
    """
    house_nr = str(nr)
    if len(house_nr)==1:
        house_nr = "00"+str(house_nr)
    if len(house_nr)==2:
        house_nr = "0"+str(house_nr)
    df = pd.read_pickle('/home/18005152/notebooks/zero/Data:/testDataFrames/
↪TEST/MachineLearning_production_'+str(house_nr))
    return df
```

```
def NormalScaler(df):
    """
    Scale the data according to the normal method.
    """
    #scale the data
    #X:
    scalerx.fit(df.loc[:,~df.columns.isin(["production"])])
    scaled_dataX = scalerx.transform(df.loc[:,~df.columns.
 ↪isin(["production"])]).tolist()
    #Y:
    scalery.fit(df.loc[:,df.columns.isin(["production"])])
    datay = scalery.transform(df.loc[:,df.columns.isin(["production"])])
    return datay,scaled_dataX

def Split_Normal(dataX,dataY):
    """
    Split the data according to the method.
    """
    #split the data
    train_X = dataX[0:5800]
    train_y = dataY[0:5800].reshape(-1,1)

    valid_X = dataX[5800:7952]
    valid_y = dataY[5800:7952].reshape(-1,1)

    test_X = dataX[7952:8663]
    test_y = dataY[7952:8663].reshape(-1,1)
    return train_X, train_y, valid_X, valid_y, test_X, test_y

def MakeTrainLoader(train_X, train_y, valid_X, valid_y, test_X, test_y):
    """
    Function for making the dataloaders.
    This is mainly for the NN.
    """
    #Make tensors from the numpy arrays.
    train_X_t = torch.from_numpy(np.array(train_X)).to(device).float()
    train_y_t = torch.from_numpy(np.array(train_y)).to(device).float()

    valid_X_t = torch.from_numpy(np.array(valid_X)).to(device).float()
    valid_y_t = torch.from_numpy(np.array(valid_y)).to(device).float()

    test_X_t = torch.from_numpy(np.array(test_X)).to(device).float()
    test_y_t = torch.from_numpy(np.array(test_y)).to(device).float()

    #Tensor Datasets
    train_set = torch.utils.data.TensorDataset(train_X_t, train_y_t)
```

```
    valid_set = torch.utils.data.TensorDataset(valid_y_t, valid_X_t)
    test_set = torch.utils.data.TensorDataset(test_y_t, test_X_t)

    #Tensor DataLoaders
    train_loader = torch.utils.data.DataLoader(train_set, batch_size=64,␣
 ↪shuffle=False, num_workers = 0)#, pin_memory=True)
    valid_loader = torch.utils.data.DataLoader(valid_set, batch_size=64,␣
 ↪shuffle=False, num_workers = 0)#, pin_memory=True)
    test_loader = torch.utils.data.DataLoader(test_set, batch_size=64,␣
 ↪shuffle=False, num_workers = 0)
    return train_loader, valid_loader, test_loader

def train_SVR(train_X,train_y):
    """
    Function to train the MVLR.
    """
    regr = SVR()
    regr.fit(train_X,train_y[:,0])
    yhat = regr.predict(train_X)

    target = torch.from_numpy(np.array(train_y)).to(device).float()[:,0]
    yhat = torch.from_numpy(np.array(yhat)).to(device).float()
    return calculate_metrics_for_model(yhat,target), regr

def validate_SVR(d1,d2,regr):
    """
    Validate the MVLR with the input data.
    """
    yhat = regr.predict(d1)
    target = d2

    target = torch.from_numpy(np.array(target)).to(device).float()
    yhat = torch.from_numpy(np.array(yhat)).to(device).float()
    return calculate_metrics_for_model(yhat,target)
```

[6]:
```
list(GetGlobalData(28).columns)
```

[6]:
```
['hour_0',
 'hour_1',
 'hour_2',
 'hour_3',
 'hour_4',
 'hour_5',
 'hour_6',
 'hour_7',
 'hour_8',
 'hour_9',
```

```
'hour_10',
'hour_11',
'hour_12',
'hour_13',
'hour_14',
'hour_15',
'hour_16',
'hour_17',
'hour_18',
'hour_19',
'hour_20',
'hour_21',
'hour_22',
'hour_23',
'prod_T-24',
'prod_T-25',
'prod_T-26',
'prod_T-27',
'prod_T-28',
'prod_T-29',
'prod_T-30',
'prod_T-31',
'prod_T-32',
'prod_T-33',
'prod_T-34',
'prod_T-35',
'prod_T-36',
'prod_T-37',
'prod_T-38',
'prod_T-39',
'prod_T-40',
'prod_T-41',
'prod_T-42',
'prod_T-43',
'prod_T-44',
'prod_T-45',
'prod_T-46',
'prod_T-47',
'prod_T-48',
'prod_T-49',
'prod_T-50',
'prod_T-51',
'prod_T-52',
'prod_T-53',
'prod_T-54',
'prod_T-55',
'prod_T-56',
```

```
'prod_T-57',
'prod_T-58',
'prod_T-59',
'prod_T-60',
'prod_T-61',
'prod_T-62',
'prod_T-63',
'prod_T-64',
'prod_T-65',
'prod_T-66',
'prod_T-67',
'prod_T-68',
'prod_T-69',
'prod_T-70',
'prod_T-71',
'prod_T-72',
'prod_T-73',
'prod_T-74',
'prod_T-75',
'prod_T-76',
'prod_T-77',
'prod_T-78',
'prod_T-79',
'prod_T-80',
'prod_T-81',
'prod_T-82',
'prod_T-83',
'prod_T-84',
'prod_T-85',
'prod_T-86',
'prod_T-87',
'prod_T-88',
'prod_T-89',
'prod_T-90',
'prod_T-91',
'prod_T-92',
'prod_T-93',
'prod_T-94',
'prod_T-95',
'prod_T-96',
'prod_T-97',
'prod_T-98',
'prod_T-99',
'prod_T-100',
'prod_T-101',
'prod_T-102',
'prod_T-103',
```

```
'prod_T-104',
'prod_T-105',
'prod_T-106',
'prod_T-107',
'prod_T-108',
'prod_T-109',
'prod_T-110',
'prod_T-111',
'prod_T-112',
'prod_T-113',
'prod_T-114',
'prod_T-115',
'prod_T-116',
'prod_T-117',
'prod_T-118',
'prod_T-119',
'prod_T-120',
'prod_T-121',
'prod_T-122',
'prod_T-123',
'prod_T-124',
'prod_T-125',
'prod_T-126',
'prod_T-127',
'prod_T-128',
'prod_T-129',
'prod_T-130',
'prod_T-131',
'prod_T-132',
'prod_T-133',
'prod_T-134',
'prod_T-135',
'prod_T-136',
'prod_T-137',
'prod_T-138',
'prod_T-139',
'prod_T-140',
'prod_T-141',
'prod_T-142',
'prod_T-143',
'prod_T-144',
'prod_T-145',
'prod_T-146',
'prod_T-147',
'prod_T-148',
'prod_T-149',
'prod_T-150',
```

```
 'prod_T-151',
 'prod_T-152',
 'prod_T-153',
 'prod_T-154',
 'prod_T-155',
 'prod_T-156',
 'prod_T-157',
 'prod_T-158',
 'prod_T-159',
 'prod_T-160',
 'prod_T-161',
 'prod_T-162',
 'prod_T-163',
 'prod_T-164',
 'prod_T-165',
 'prod_T-166',
 'prod_T-167',
 'prod_T-168',
 'day_mean',
 'week_mean',
 'production']
```

3 Main loop:

[6]:
```
#stats savelist:
SVR_stats = pd.DataFrame()#SVR

#Learning loop:
for i in tqdm(range(len(houses))):
    house_number = houses[i]
    """
    Data loading...
    """
    #laad de data:
    df = GetGlobalData(house_number)
    #scale de data:
    Y_data, X_data  = NormalScaler(df)
    #splits de data
    train_X, train_y, valid_X, valid_y, test_X, test_y =␣
 ↪Split_Normal(X_data,Y_data)

    """
    Training
    """
    SVR_train_stats, svr = train_SVR(train_X,train_y)
```

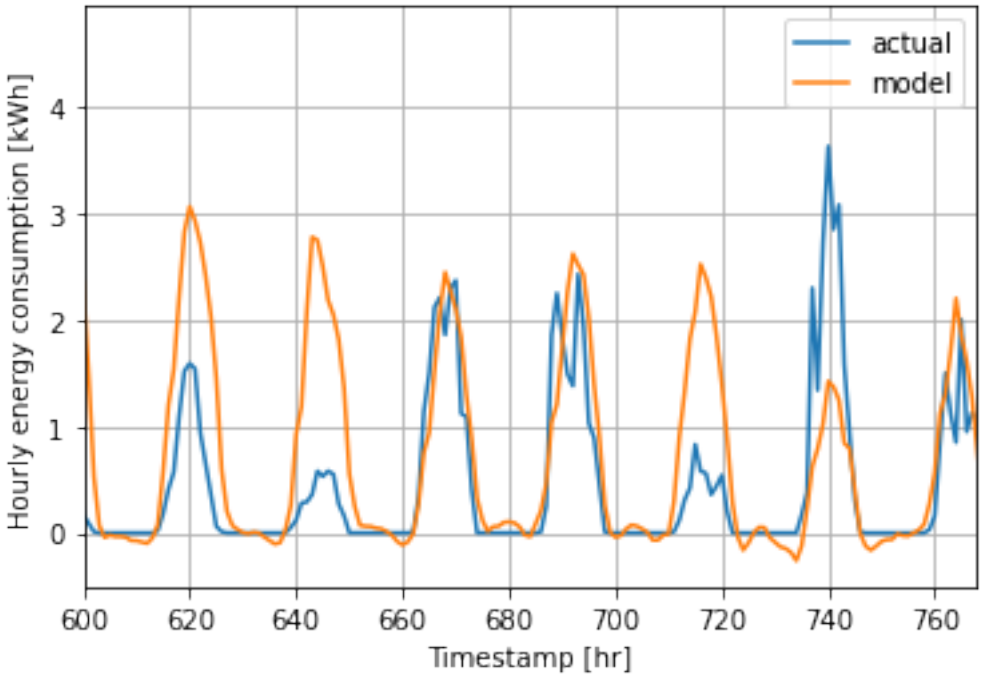
```
    """
    Evaluation
    """
    SVR_valid_stats = validate_SVR(valid_X,valid_y,svr)

    """
    Testing
    """
    SVR_test_stats = validate_SVR(test_X,test_y,svr)

    """
    Save metrics
    """
    index =␣
↪["MAE_train","MSE_train","MAPE_train","R2_train","MAE_valid","MSE_valid","MAPE_valid","R2_v
    New_Stats = pd.DataFrame(SVR_train_stats+SVR_valid_stats+SVR_test_stats,␣
↪index=index, columns=[str(house_number)])
    SVR_stats = pd.concat([SVR_stats,New_Stats],axis=1)
```

```
100%|      | 12/12 [02:05<00:00, 10.47s/it]
```

[7]:
```
yhat = scalery.inverse_transform(svr.predict(valid_X))
y = scalery.inverse_transform(valid_y)
plt.plot(y, label='actual')
plt.plot(yhat, label="model")
plt.xlabel("Timestamp [hr]")
plt.ylabel("Hourly energy consumption [kWh]")
plt.grid()
plt.legend()
plt.xlim([600, 768])
plt.savefig("SVR_production_house100.png")
```



[8]:
```
SVR_stats
```

[8]:

	28	37	40	42	\
MAE_train	3.146637e-01	3.372785e-01	3.096375e-01	3.137025e-01	
MSE_train	3.275852e-01	3.731878e-01	3.090853e-01	3.139773e-01	
MAPE_train	8.787165e+07	9.103637e+10	7.634822e+07	5.644288e+08	
R2_train	8.419702e-01	8.402451e-01	8.414111e-01	8.371447e-01	
MAE_valid	2.940759e-01	2.968012e-01	2.773961e-01	2.879628e-01	
MSE_valid	2.834624e-01	2.974526e-01	2.549775e-01	2.706500e-01	
MAPE_valid	1.131793e+09	1.130880e+12	9.366085e+08	6.779086e+09	
R2_valid	7.012283e-01	7.120988e-01	7.097221e-01	6.912135e-01	
MAE_test	1.252476e-01	1.239290e-01	1.109520e-01	1.222837e-01	
MSE_test	4.564782e-02	4.841750e-02	3.580950e-02	4.483492e-02	
MAPE_test	4.370756e+08	4.505390e+11	3.541129e+08	2.567658e+09	
R2_test	5.467441e-01	5.871921e-01	5.937426e-01	4.884072e-01	

	105	115	56	51	\
MAE_train	2.831755e-01	3.151765e-01	3.015018e-01	3.179873e-01	
MSE_train	2.809149e-01	3.151356e-01	2.962717e-01	3.250829e-01	
MAPE_train	1.447462e+08	2.930210e+08	1.754823e+08	1.432479e+08	
R2_train	8.442882e-01	8.405128e-01	8.486119e-01	8.420105e-01	
MAE_valid	2.429238e-01	2.923188e-01	2.541344e-01	2.978489e-01	
MSE_valid	2.068460e-01	2.772360e-01	2.231059e-01	2.863996e-01	
MAPE_valid	1.799770e+09	3.751342e+09	2.079552e+09	1.834467e+09	
R2_valid	7.252879e-01	6.974527e-01	7.283444e-01	6.984884e-01	

```
MAE_test    1.053893e-01  1.171851e-01  1.066230e-01  1.233492e-01
MSE_test    3.364427e-02  4.359355e-02  3.706085e-02  4.617957e-02
MAPE_test   7.324272e+08  1.402893e+09  8.038954e+08  6.973038e+08
R2_test     5.946899e-01  5.335910e-01  5.610948e-01  5.307270e-01

                      58            70            99           100
MAE_train   2.909167e-01  3.178481e-01  3.180643e-01  3.171931e-01
MSE_train   2.882661e-01  3.235432e-01  3.202993e-01  3.199436e-01
MAPE_train  1.092624e+08  9.603404e+07  1.252224e+09  2.958518e+08
R2_train    8.445024e-01  8.372018e-01  8.403412e-01  8.426551e-01
MAE_valid   2.462275e-01  2.941454e-01  2.961173e-01  2.957847e-01
MSE_valid   2.079606e-01  2.838298e-01  2.835235e-01  2.840449e-01
MAPE_valid  1.250298e+09  1.203912e+09  1.562293e+10  3.708620e+09
R2_valid    7.234813e-01  6.886734e-01  6.933030e-01  6.994200e-01
MAE_test    1.086189e-01  1.216256e-01  1.170923e-01  1.125220e-01
MSE_test    3.929573e-02  4.619518e-02  4.070373e-02  3.753047e-02
MAPE_test   4.957497e+08  4.463458e+08  5.675748e+09  1.331753e+09
R2_test     5.481120e-01  4.694924e-01  5.309358e-01  5.610221e-01
```

[9]:
```
SVR_stats.to_pickle("SVR_statistics")
```

[10]:
```
pd.read_pickle("SVR_statistics").to_excel("SVR.xlsx")
```

4 summarize stats with mean

[11]:
```
(SVR_stats).mean(axis=1)
```

[11]:
```
MAE_train     3.114288e-01
MSE_train     3.161077e-01
MAPE_train    7.856241e+09
R2_train      8.417412e-01
MAE_valid     2.813114e-01
MSE_valid     2.632907e-01
MAPE_valid    9.758155e+10
R2_valid      7.057261e-01
MAE_test      1.162348e-01
MSE_test      4.157609e-02
MAPE_test     3.879033e+10
R2_test       5.454792e-01
dtype: float64
```