W16_SVR_MultipleHouses

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 0x7fe6881fa180>
```

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_consumption_'+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(["consumption"])])
    scaled_dataX = scalerx.transform(df.loc[:,~df.columns.
 ↪isin(["consumption"])]).tolist()
    #Y:
    scalery.fit(df.loc[:,df.columns.isin(["consumption"])])
    datay = scalery.transform(df.loc[:,df.columns.isin(["consumption"])])
    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)
```

[9]:
```
list(GetGlobalData(28).columns)
```

[9]:
```
['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',
'weekday_0',
'weekday_1',
'weekday_2',
'weekday_3',
'weekday_4',
'weekday_5',
'weekday_6',
'cons_T-24',
'cons_T-25',
'cons_T-26',
'cons_T-27',
'cons_T-28',
'cons_T-29',
'cons_T-30',
'cons_T-31',
'cons_T-32',
'cons_T-33',
'cons_T-34',
'cons_T-35',
'cons_T-36',
'cons_T-37',
'cons_T-38',
'cons_T-39',
'cons_T-40',
'cons_T-41',
'cons_T-42',
'cons_T-43',
'cons_T-44',
'cons_T-45',
'cons_T-46',
'cons_T-47',
'cons_T-48',
'cons_T-49',
```

```
'cons_T-50',
'cons_T-51',
'cons_T-52',
'cons_T-53',
'cons_T-54',
'cons_T-55',
'cons_T-56',
'cons_T-57',
'cons_T-58',
'cons_T-59',
'cons_T-60',
'cons_T-61',
'cons_T-62',
'cons_T-63',
'cons_T-64',
'cons_T-65',
'cons_T-66',
'cons_T-67',
'cons_T-68',
'cons_T-69',
'cons_T-70',
'cons_T-71',
'cons_T-72',
'cons_T-73',
'cons_T-74',
'cons_T-75',
'cons_T-76',
'cons_T-77',
'cons_T-78',
'cons_T-79',
'cons_T-80',
'cons_T-81',
'cons_T-82',
'cons_T-83',
'cons_T-84',
'cons_T-85',
'cons_T-86',
'cons_T-87',
'cons_T-88',
'cons_T-89',
'cons_T-90',
'cons_T-91',
'cons_T-92',
'cons_T-93',
'cons_T-94',
'cons_T-95',
'cons_T-96',
```

```
'cons_T-97',
'cons_T-98',
'cons_T-99',
'cons_T-100',
'cons_T-101',
'cons_T-102',
'cons_T-103',
'cons_T-104',
'cons_T-105',
'cons_T-106',
'cons_T-107',
'cons_T-108',
'cons_T-109',
'cons_T-110',
'cons_T-111',
'cons_T-112',
'cons_T-113',
'cons_T-114',
'cons_T-115',
'cons_T-116',
'cons_T-117',
'cons_T-118',
'cons_T-119',
'cons_T-120',
'cons_T-121',
'cons_T-122',
'cons_T-123',
'cons_T-124',
'cons_T-125',
'cons_T-126',
'cons_T-127',
'cons_T-128',
'cons_T-129',
'cons_T-130',
'cons_T-131',
'cons_T-132',
'cons_T-133',
'cons_T-134',
'cons_T-135',
'cons_T-136',
'cons_T-137',
'cons_T-138',
'cons_T-139',
'cons_T-140',
'cons_T-141',
'cons_T-142',
'cons_T-143',
```

```
 'cons_T-144',
 'cons_T-145',
 'cons_T-146',
 'cons_T-147',
 'cons_T-148',
 'cons_T-149',
 'cons_T-150',
 'cons_T-151',
 'cons_T-152',
 'cons_T-153',
 'cons_T-154',
 'cons_T-155',
 'cons_T-156',
 'cons_T-157',
 'cons_T-158',
 'cons_T-159',
 'cons_T-160',
 'cons_T-161',
 'cons_T-162',
 'cons_T-163',
 'cons_T-164',
 'cons_T-165',
 'cons_T-166',
 'cons_T-167',
 'cons_T-168',
 'day_mean',
 'week_mean',
 'consumption']
```

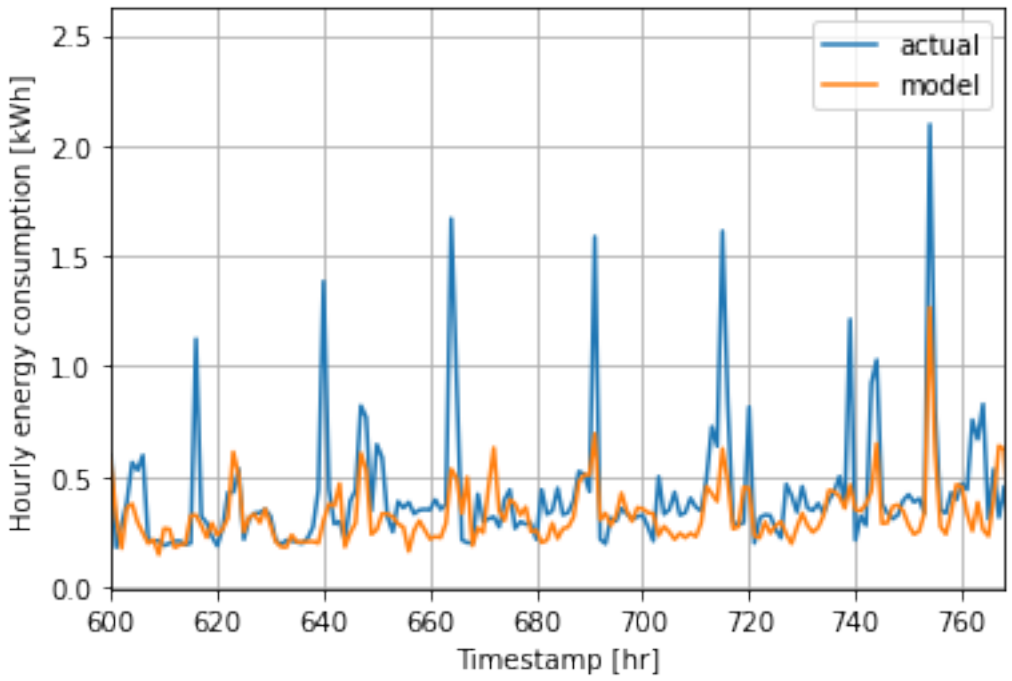
3 Main loop:

[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 [03:11<00:00, 15.96s/it]
```

[8]:
```
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_consumption_house100.png")
```



[9]:
```
SVR_stats
```

[9]:

	28	37	40	42	105	\
MAE_train	0.108973	0.153376	0.209247	0.257211	0.158199	
MSE_train	0.058915	0.095632	0.146547	0.247174	0.094154	
MAPE_train	208418.579102	32.083011	30.470884	33.140785	28.017536	
R2_train	-0.167171	0.290441	-0.000258	-0.785021	0.427932	
MAE_valid	0.196403	0.254474	0.441853	0.491927	0.398854	
MSE_valid	0.102652	0.169090	0.397407	0.515350	0.310017	
MAPE_valid	66.310585	88.754028	110.392249	127.456450	146.655595	
R2_valid	-1.396538	-1.850042	-4.479527	-6.768188	-1.978974	
MAE_test	0.232786	0.385785	0.554809	0.674689	0.519288	
MSE_test	0.120281	0.296163	0.712023	0.789924	0.446236	
MAPE_test	44.427726	127.023423	85.945469	140.976596	143.093348	
R2_test	-2.399453	-1.464884	-10.905163	-10.753199	-3.246903	

	115	56	51	58	70	99	\
MAE_train	0.216196	0.181610	0.202907	0.126193	0.188902	0.154818	
MSE_train	0.170272	0.116834	0.150680	0.072427	0.155518	0.099804	
MAPE_train	22.308004	29.630730	19.674735	20.129289	46.649137	23.064919	
R2_train	-0.123810	0.186443	0.133624	-0.003712	-0.524865	-0.333568	
MAE_valid	0.406555	0.355374	0.341987	0.216449	0.292953	0.266481	
MSE_valid	0.382607	0.281609	0.276530	0.127668	0.248966	0.170920	
MAPE_valid	53.937298	99.483383	80.849880	65.166992	134.108102	92.441976	
R2_valid	-2.964482	-3.848333	-1.954739	-1.989386	-6.728658	-2.485834	

```
MAE_test     0.443855   0.422309   0.422336   0.274021    0.428011   0.345318
MSE_test     0.385477   0.307679   0.347784   0.170160    0.318563   0.283007
MAPE_test   36.660984  67.093855  48.540682  43.519148  202.327037  45.256296
R2_test     -4.256996  -5.803983  -4.249617  -2.725327   -3.382225 -13.856383
```

```
                  100
MAE_train    0.112779
MSE_train    0.054393
MAPE_train  20.273179
R2_train     0.163580
MAE_valid    0.169210
MSE_valid    0.077249
MAPE_valid  58.604467
R2_valid    -0.878069
MAE_test     0.172123
MSE_test     0.073003
MAPE_test   25.993633
R2_test     -2.414455
```

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

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

4 summarize stats with mean

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