W16_MVLR_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 0x7fb91f84e0d8>
```

Make all functions:

[12]:
```
GetGlobalData(28)
```

[12]:
```
                     hour_0  hour_1  hour_2  hour_3  hour_4  hour_5  hour_6  \
2018-12-31 23:00:00       0       0       0       0       0       0       0
2019-01-01 00:00:00       1       0       0       0       0       0       0
2019-01-01 01:00:00       0       1       0       0       0       0       0
2019-01-01 02:00:00       0       0       1       0       0       0       0
2019-01-01 03:00:00       0       0       0       1       0       0       0
…                      …     …     …     …     …     …     …
2019-12-31 18:00:00       0       0       0       0       0       0       0
2019-12-31 19:00:00       0       0       0       0       0       0       0
2019-12-31 20:00:00       0       0       0       0       0       0       0
2019-12-31 21:00:00       0       0       0       0       0       0       0
2019-12-31 22:00:00       0       0       0       0       0       0       0

                     hour_7  hour_8  hour_9  …  prod_T-162  prod_T-163  \
2018-12-31 23:00:00       0       0       0  …         0.0         0.0
2019-01-01 00:00:00       0       0       0  …         0.0         0.0
2019-01-01 01:00:00       0       0       0  …         0.0         0.0
2019-01-01 02:00:00       0       0       0  …         0.0         0.0
2019-01-01 03:00:00       0       0       0  …         0.0         0.0
…                      …     …     …  …          …          …
2019-12-31 18:00:00       0       0       0  …         0.0         0.0
2019-12-31 19:00:00       0       0       0  …         0.0         0.0
2019-12-31 20:00:00       0       0       0  …         0.0         0.0
2019-12-31 21:00:00       0       0       0  …         0.0         0.0
2019-12-31 22:00:00       0       0       0  …         0.0         0.0

                     prod_T-164  prod_T-165  prod_T-166  prod_T-167  \
2018-12-31 23:00:00         0.0         0.0         0.0         0.0
2019-01-01 00:00:00         0.0         0.0         0.0         0.0
2019-01-01 01:00:00         0.0         0.0         0.0         0.0
```

```
2019-01-01 02:00:00         0.0         0.0         0.0         0.0
2019-01-01 03:00:00         0.0         0.0         0.0         0.0
…                             …          …          …          …
2019-12-31 18:00:00         0.0         0.0         0.0         0.0
2019-12-31 19:00:00         0.0         0.0         0.0         0.0
2019-12-31 20:00:00         0.0         0.0         0.0         0.0
2019-12-31 21:00:00         0.0         0.0         0.0         0.0
2019-12-31 22:00:00         0.0         0.0         0.0         0.0

                     prod_T-168  day_mean  week_mean  production
2018-12-31 23:00:00         0.0       0.0        0.0         0.0
2019-01-01 00:00:00         0.0       0.0        0.0         0.0
2019-01-01 01:00:00         0.0       0.0        0.0         0.0
2019-01-01 02:00:00         0.0       0.0        0.0         0.0
2019-01-01 03:00:00         0.0       0.0        0.0         0.0
…                             …        …         …          …
2019-12-31 18:00:00         0.0       0.0        0.0         0.0
2019-12-31 19:00:00         0.0       0.0        0.0         0.0
2019-12-31 20:00:00         0.0       0.0        0.0         0.0
2019-12-31 21:00:00         0.0       0.0        0.0         0.0
2019-12-31 22:00:00         0.0       0.0        0.0         0.0

[8760 rows x 172 columns]
```

[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()
```

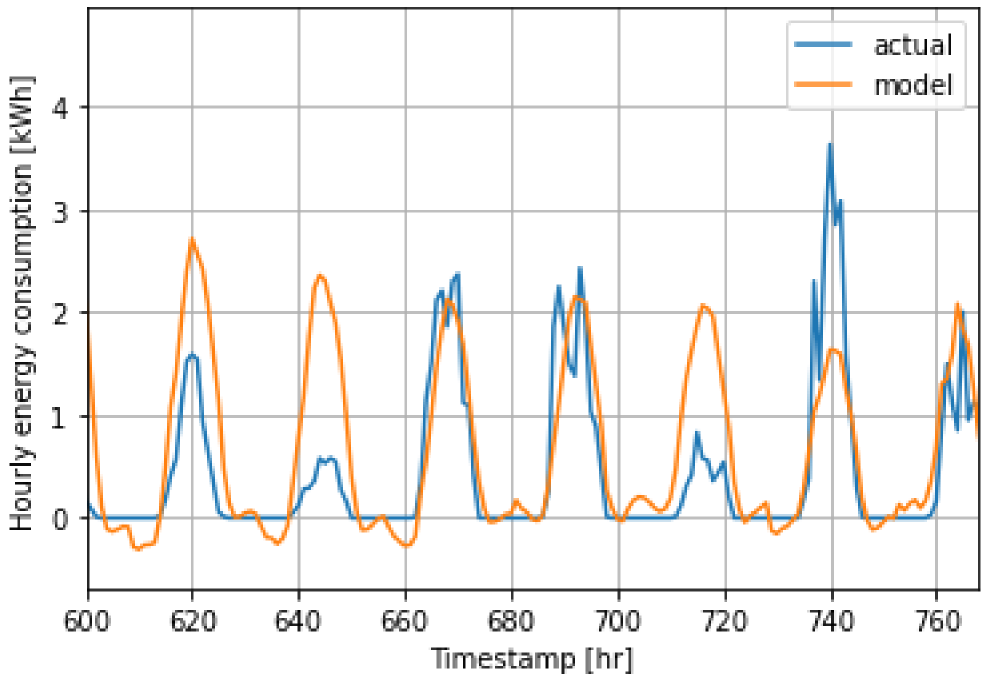
```python
    valid_X_t = torch.from_numpy(np.array(valid_X)).to(device).float()
    valid_y_t = torch.from_numpy(np.array(valid_y)).to(device).float()

    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_MVLR(train_X,train_y):
    """
    Function to train the MVLR.
    """
    regr = linear_model.LinearRegression()
    regr.fit(train_X,train_y[:,0])
    yhat = regr.predict(train_X)

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

def validate_MVLR(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)
```

3 Main loop:

```
[6]: #stats savelist:
MVLR_stats = pd.DataFrame()#MVLR

#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
    """
    MVLR_train_stats, mvlr = train_MVLR(train_X,train_y)

    """
    Evaluation
    """
    MVLR_valid_stats = validate_MVLR(valid_X,valid_y,mvlr)

    """
    Testing
    """
    MVLR_test_stats = validate_MVLR(test_X,test_y,mvlr)

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

```
100%|      | 12/12 [00:11<00:00,  1.05it/s]
```

[7]:
```
yhat = scalery.inverse_transform(mvlr.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("MVLR_production_house100.png")
```



[8]:
```
MVLR_stats
```

[8]:

	28	37	40	42 \
MAE_train	4.116472e-01	4.497618e-01	4.130452e-01	4.177692e-01
MSE_train	4.546895e-01	5.311521e-01	4.398874e-01	4.428327e-01
MAPE_train	1.537968e+09	1.594922e+12	1.310917e+09	9.492492e+09
R2_train	7.456332e-01	7.389269e-01	7.403009e-01	7.355712e-01
MAE_valid	3.022222e-01	3.048383e-01	2.873257e-01	3.012000e-01
MSE_valid	2.555815e-01	2.652604e-01	2.283122e-01	2.424586e-01
MAPE_valid	1.164522e+09	1.169081e+12	9.698930e+08	7.135434e+09
R2_valid	6.817123e-01	7.044150e-01	6.963208e-01	6.794138e-01
MAE_test	1.800483e-01	1.778228e-01	1.610584e-01	1.743069e-01
MSE_test	8.064768e-02	8.237299e-02	6.679278e-02	7.577623e-02
MAPE_test	6.584574e+08	6.529128e+11	5.317714e+08	3.999598e+09

```
R2_test     5.450462e-01  5.855638e-01  5.859166e-01  5.533538e-01

                     105           115            56            51  \
MAE_train   3.610134e-01  4.210402e-01  4.032020e-01  4.232243e-01
MSE_train   3.757905e-01  4.489148e-01  4.275295e-01  4.589821e-01
MAPE_train  2.549397e+09  5.197990e+09  3.036382e+09  2.504225e+09
R2_train    7.614202e-01  7.388221e-01  7.488109e-01  7.423288e-01
MAE_valid   2.519729e-01  3.064490e-01  2.738141e-01  3.092119e-01
MSE_valid   1.935320e-01  2.496955e-01  2.090931e-01  2.567988e-01
MAPE_valid  1.873435e+09  3.895246e+09  2.180885e+09  1.899535e+09
R2_valid    7.123137e-01  6.772059e-01  7.055347e-01  6.806785e-01
MAE_test    1.535093e-01  1.774780e-01  1.612308e-01  1.803227e-01
MSE_test    6.186283e-02  7.702770e-02  6.584720e-02  8.020745e-02
MAPE_test   1.066927e+09  2.163371e+09  1.222927e+09  1.063923e+09
R2_test     5.873401e-01  5.448262e-01  5.826516e-01  5.459436e-01

                      58            70            99           100
MAE_train   3.782464e-01  4.233046e-01  4.251457e-01  4.239199e-01
MSE_train   3.957430e-01  4.568989e-01  4.578442e-01  4.587210e-01
MAPE_train  1.839224e+09  1.665540e+09  2.173646e+10  5.094916e+09
R2_train    7.536165e-01  7.345917e-01  7.372899e-01  7.400545e-01
MAE_valid   2.597381e-01  3.070714e-01  3.095774e-01  3.075366e-01
MSE_valid   1.951032e-01  2.533525e-01  2.557056e-01  2.546894e-01
MAPE_valid  1.320027e+09  1.266450e+09  1.633479e+10  3.838791e+09
R2_valid    7.053114e-01  6.778355e-01  6.750884e-01  6.798122e-01
MAE_test    1.598525e-01  1.789805e-01  1.773236e-01  1.722523e-01
MSE_test    6.817910e-02  7.942472e-02  7.704069e-02  7.215891e-02
MAPE_test   7.678155e+08  7.097355e+08  9.064457e+09  2.072672e+09
R2_test     5.816744e-01  5.445435e-01  5.453613e-01  5.524788e-01
```

[9]:
```
MVLR_stats.to_pickle("MVLR_statistics")
```

[10]:
```
pd.read_pickle("MVLR_statistics").to_excel("MVLR.xlsx")
```

4 summarize stats with mean

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

[11]:
```
MAE_train     4.126100e-01
MSE_train     4.457488e-01
MAPE_train    1.375740e+11
R2_train      7.431139e-01
MAE_valid     2.934131e-01
MSE_valid     2.382986e-01
MAPE_valid    1.009133e+11
```

```
R2_valid      6.896369e-01
MAE_test      1.711822e-01
MSE_test      7.394486e-02
MAPE_test     5.635287e+10
R2_test       5.628916e-01
dtype: float64
```