# -\*- coding: utf-8 -\*-

"""

Created on Wed Apr 13 15:58:41 2022

@author: d84007

"""

from statsmodels.graphics.tsaplots import plot\_pacf

from statsmodels.graphics.tsaplots import plot\_acf

from statsmodels.tsa.arima\_process import ArmaProcess

from statsmodels.stats.diagnostic import acorr\_ljungbox

from statsmodels.tsa.statespace.sarimax import SARIMAX

from statsmodels.tsa.stattools import adfuller

from statsmodels.tsa.stattools import pacf

from statsmodels.tsa.stattools import acf

from tqdm import tqdm\_notebook

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

from statsmodels.tsa.ar\_model import AutoReg

from statsmodels.tsa.arima.model import ARIMA

from itertools import product

from sklearn.metrics import mean\_squared\_error, r2\_score, mean\_absolute\_error

from sklearn.metrics import median\_absolute\_error, mean\_squared\_log\_error

from sklearn.preprocessing import MinMaxScaler

# univariate multi-step lstm

from math import sqrt

from numpy import split

from numpy import array

from pandas import read\_csv

from sklearn.metrics import mean\_squared\_error

from numpy import array

from keras.models import Sequential

from keras.layers import LSTM

from keras.layers import Bidirectional

from keras.layers.convolutional import Conv1D, MaxPooling1D

from keras.layers import LSTM, GRU, SimpleRNN, Embedding

from keras.layers import Dense, Activation, Dropout, Flatten, LeakyReLU

from pandas import DataFrame

from pandas import concat

# split a univariate sequence into samples by creating 2 arrays, one for the input and the other for the output;

#we have then at the end 2 arrays of shape (seuence-n\_steps\_in,n\_steps\_in) (seuence-n\_steps\_out,n\_steps\_out)

def split\_sequence(sequence, n\_steps\_in, n\_steps\_out):

X, y = list(), list()

for i in range(len(sequence)):

# find the end of this pattern

end\_ix = i + n\_steps\_in

out\_end\_ix = end\_ix + n\_steps\_out

# check if we are beyond the sequence

if out\_end\_ix > len(sequence):

break

# gather input and output parts of the pattern

seq\_x, seq\_y = sequence[i:end\_ix], sequence[end\_ix:out\_end\_ix]

X.append(seq\_x)

y.append(seq\_y)

return array(X), array(y)

#same as above but with keeping only every separated sequences (no overlap)

def split\_sequence\_23(sequence, n\_steps\_in, n\_steps\_out):

X, y = list(), list()

i=0

while i<len(sequence):

# find the end of this pattern

end\_ix = i + n\_steps\_in

out\_end\_ix = end\_ix + n\_steps\_out

# check if we are beyond the sequence

if out\_end\_ix > len(sequence):

break

# gather input and output parts of the pattern

seq\_x, seq\_y = sequence[i:end\_ix], sequence[end\_ix:out\_end\_ix]

X.append(seq\_x)

y.append(seq\_y)

i=i+n\_steps\_in

return array(X), array(y)

#Conv1D with 30 in and 30 out ( represents a time series of daily entries, we then try to predict the next month)

X=array(data\_CCR['Number\_of\_calls'])

X=list(data\_CCR['Number\_of\_calls'].iloc[:245])

n\_steps\_in=30

n\_steps\_out=30

X\_input,Y\_input=split\_sequence(X, n\_steps\_in, n\_steps\_out)

X\_input.shape

Y\_input.shape

model = Sequential()

model.add(Conv1D(filters=124, kernel\_size=2, activation='relu', input\_shape=(30, 1)))

model.add(MaxPooling1D(pool\_size=2))

model.add(Flatten())

#to use the simpleRNN we would have to delete the previous maxpool and flatten

#model.add(SimpleRNN(24, activation='relu', input\_shape=(30, 1)))

model.add(Dense(60,activation='relu'))

model.add(Dense(30))

model.compile(optimizer='adam', loss='mse')

X\_input.shape

#X\_input.shape

#data\_CCR['Number\_of\_calls'].iloc[245:305]

X\_test=list(data\_CCR['Number\_of\_calls'].iloc[245:305])

n\_steps\_in=30

n\_steps\_out=30

X\_output,Y\_output=split\_sequence(X\_test, n\_steps\_in, n\_steps\_out)

model.fit(X\_input.reshape(186,30,1),Y\_input,epochs=100,validation\_data = (X\_output.reshape(1,30,1), Y\_output))

model.predict(X\_output.reshape(1,30,1))

Y\_output.astype(float)

test=Y\_output.astype(float)

predictions=model.predict(X\_output.reshape(1,30,1))

evaluation\_results = pd.DataFrame({'r2\_score': r2\_score(test, predictions)}, index=[0])

evaluation\_results['mean\_absolute\_error'] = mean\_absolute\_error(test, predictions)

evaluation\_results['mean\_squared\_error'] = mean\_squared\_error(test, predictions)

evaluation\_results['root\_mean\_squared\_error'] = np.sqrt(mean\_squared\_error(test, predictions))

evaluation\_results['mean\_absolute\_percentage\_error'] = np.mean(np.abs(predictions - test)

/np.abs(test))\*100

G=evaluation\_results

#print(i)

print(evaluation\_results['r2\_score'])

print(evaluation\_results['mean\_absolute\_error'])

print(evaluation\_results['mean\_squared\_error'])

print(evaluation\_results['root\_mean\_squared\_error'])

print(evaluation\_results['mean\_absolute\_percentage\_error'])

#here we split the entry sequence by separating week by week the entry (we are in workweek):

list\_year=[2021,2022]

list\_month=[1,2,3,4,5,6,7,8,9,10,11,12]

Transi=data\_CCR.where(data\_CCR.Days.dt.year==2021)

Transi=Transi.dropna(how='all')

Transi=Transi.where(Transi.Days.dt.month==1)

Transi=Transi.dropna(how='all')

data\_CCR\_split5=Transi.iloc[:5,]

data\_CCR\_split10=Transi.iloc[5:10,]

data\_CCR\_split15=Transi.iloc[10:15,]

data\_CCR\_split20=Transi.iloc[15:23,]

data\_CCR\_first20=Transi.iloc[:19,]

for y in list\_year:

print(y)

Transi=data\_CCR.where(data\_CCR.Days.dt.year==y)

Transi=Transi.dropna(how='all')

print(Transi.Days.dt.month.unique())

listmonth=list(Transi.Days.dt.month.unique())

for m in listmonth:

Transi\_m=Transi.where(Transi.Days.dt.month==m)

Transi\_m=Transi\_m.dropna(how='all')

Transi\_m=Transi\_m.where(Transi\_m['Number\_of\_calls']>1000)

Transi\_m=Transi\_m.dropna(how='all')

#Transi.reset\_index()

Transi\_5=Transi\_m.iloc[:5,]

Transi\_10=Transi\_m.iloc[5:10,]

Transi\_15=Transi\_m.iloc[10:15,]

Transi\_20=Transi\_m.iloc[15:23,]

Transi\_first20=Transi\_m.iloc[:19,]

data\_CCR\_split5=pd.concat([data\_CCR\_split5,Transi\_5])

data\_CCR\_split10=pd.concat([data\_CCR\_split10,Transi\_10])

data\_CCR\_split15=pd.concat([data\_CCR\_split15,Transi\_15])

data\_CCR\_split20=pd.concat([data\_CCR\_split20,Transi\_20])

data\_CCR\_first20=pd.concat([data\_CCR\_first20,Transi\_first20])

data\_CCR\_split5.drop\_duplicates(inplace=True)

data\_CCR\_split10.drop\_duplicates(inplace=True)

data\_CCR\_split15.drop\_duplicates(inplace=True)

data\_CCR\_split20.drop\_duplicates(inplace=True)

data\_CCR\_first20.drop\_duplicates(inplace=True)

#individual predictions week by week with the use of Simple RNN :

#compurte the results month by month

list\_dataset=[data\_CCR\_split5,data\_CCR\_split10,data\_CCR\_split15,data\_CCR\_split20]

list\_mean\_absolute\_percentage\_error3=[]

list\_root\_mean\_squared\_error3=[]

list\_input\_shape=[7,8,9,10,11]

k=0

for i in [40,45,50,55,60]:

j=1

for data in list\_dataset:

X=array(data['Number\_of\_calls'])

X=list(data['Number\_of\_calls'].iloc[:i])

n\_steps\_in=5

n\_steps\_out=5

#X\_input,Y\_input=split\_sequence(X, n\_steps\_in, n\_steps\_out)

X\_input,Y\_input=split\_sequence\_23(X, n\_steps\_in, n\_steps\_out)

#array(X\_input[1:2:,],X\_input[22:23,])

#X\_input[1:2,]

X\_input.shape

Y\_input.shape

model = Sequential()

model.add(SimpleRNN(24, activation='relu', input\_shape=(5, 1)))

#model.add(MaxPooling1D(pool\_size=2))

#model.add(Flatten())

model.add(Dense(60,activation='relu'))

#model.add(Dense(30,activation='relu'))

model.add(Dense(5))

model.compile(optimizer='adam', loss='mse')

X\_input.shape

X\_test=list(data['Number\_of\_calls'].iloc[i:i+10])

X\_output,Y\_output=split\_sequence(X\_test, n\_steps\_in, n\_steps\_out)

model.fit(X\_input.reshape(list\_input\_shape[k],5,1),Y\_input.reshape(list\_input\_shape[k],5),epochs=100) #,validation\_data = (X\_output.reshape(1,19,1), Y\_output))

#model.fit(X\_input.reshape(11,19,1),Y\_input,epochs=100)

#X\_input.shape

#data\_CCR['Number\_of\_calls'].iloc[245:305]

n\_steps\_in=5

n\_steps\_out=5

X\_output,Y\_output=split\_sequence(X\_test, n\_steps\_in, n\_steps\_out)

model.predict(X\_output.reshape(1,5,1))

Y\_output.astype(float)

test=Y\_output.astype(float)

predictions=model.predict(X\_output.reshape(1,5,1))

if j==1:

predictions\_concat=pd.DataFrame(np.transpose(predictions))

test\_concat=pd.DataFrame(np.transpose(test))

else :

predictions\_concat=pd.concat([predictions\_concat,pd.DataFrame(np.transpose(predictions))])

test\_concat=pd.concat([test\_concat,pd.DataFrame(np.transpose(test))])

j=j+1

evaluation\_results = pd.DataFrame({'r2\_score': r2\_score(test\_concat, predictions\_concat)}, index=[0])

evaluation\_results['mean\_absolute\_error'] = mean\_absolute\_error(test\_concat, predictions\_concat)

evaluation\_results['mean\_squared\_error'] = mean\_squared\_error(test\_concat, predictions\_concat)

evaluation\_results['root\_mean\_squared\_error'] = np.sqrt(mean\_squared\_error(test\_concat, predictions\_concat))

evaluation\_results['mean\_absolute\_percentage\_error'] = np.mean(np.abs(predictions\_concat - test\_concat)

/np.abs(test\_concat))\*100

G=evaluation\_results

#print(i)

print(evaluation\_results['r2\_score'])

print(evaluation\_results['mean\_absolute\_error'])

print(evaluation\_results['mean\_squared\_error'])

print(evaluation\_results['root\_mean\_squared\_error'])

print(evaluation\_results['mean\_absolute\_percentage\_error'])

list\_mean\_absolute\_percentage\_error3.append([j,i,evaluation\_results['mean\_absolute\_percentage\_error']])

list\_root\_mean\_squared\_error3.append([j,i,evaluation\_results['root\_mean\_squared\_error']])

k=k+1

list\_mean\_absolute\_percentage\_error3

list\_root\_mean\_squared\_error3

#predictions by seuqences of 19 working days (trying to get the best prediction for the next month)

#evaluation month by month

list\_mean\_absolute\_percentage\_error4=[]

list\_root\_mean\_squared\_error4=[]

list\_input\_shape=[6,7,8,9,10,11,12]

k=0

data\_CCR\_first20.iloc[247:]

for i in [133,152,171,190,209,228,247]:

test\_filter=data\_CCR\_first20['Number\_of\_calls'].where(data\_CCR\_first20['Number\_of\_calls']>1000)

test\_filter=test\_filter.dropna(how='all')

#same but with other loop (19days sequences with an input entry month by month):

X=array(data\_CCR\_first20['Number\_of\_calls'])

#X=list(data\_CCR\_first20['Number\_of\_calls'].iloc[:190])

X=list(test\_filter.iloc[:i])

n\_steps\_in=19

n\_steps\_out=19

#X\_input,Y\_input=split\_sequence(X, n\_steps\_in, n\_steps\_out)

X\_input,Y\_input=split\_sequence\_23(X, n\_steps\_in, n\_steps\_out)

#array(X\_input[1:2:,],X\_input[22:23,])

#X\_input[1:2,]

X\_input.shape

Y\_input.shape

model = Sequential()

#model.add(Embedding(input\_dim=11,output\_dim=24))

model.add(GRU(96, activation='relu',return\_sequences=True)) # input\_shape=(19, 1),

model.add(SimpleRNN(48, activation='relu'))

#model.add(MaxPooling1D(pool\_size=2))

#model.add(Flatten())

model.add(Dense(60,activation='relu'))

model.add(Dense(30))

model.add(Dense(19))

model.compile(optimizer='adam', loss='mse')

X\_input.shape

#X\_test=list(data\_CCR\_first20['Number\_of\_calls'].iloc[190:228])

X\_test=list(test\_filter.iloc[i:i+38])

X\_output,Y\_output=split\_sequence(X\_test, n\_steps\_in, n\_steps\_out)

#model.fit(X\_input.reshape(11,19,1),Y\_input,epochs=100) #,validation\_data = (X\_output.reshape(1,19,1), Y\_output))

#model.fit(X\_input.reshape(11,19),Y\_input,epochs=100)

model.fit(X\_input.reshape(list\_input\_shape[k],19,1),Y\_input,epochs=100)

#X\_input.shape

#data\_CCR['Number\_of\_calls'].iloc[245:305]

n\_steps\_in=19

n\_steps\_out=19

X\_output,Y\_output=split\_sequence(X\_test, n\_steps\_in, n\_steps\_out)

model.predict(X\_output.reshape(1,19,1))

#model.predict(X\_output.reshape(1,19))

Y\_output.astype(float)

test=Y\_output.astype(float)

predictions=model.predict(X\_output.reshape(1,19,1))

#predictions=model.predict(X\_output.reshape(1,19))

evaluation\_results = pd.DataFrame({'r2\_score': r2\_score(test, predictions)}, index=[0])

evaluation\_results['mean\_absolute\_error'] = mean\_absolute\_error(test, predictions)

evaluation\_results['mean\_squared\_error'] = mean\_squared\_error(test, predictions)

evaluation\_results['root\_mean\_squared\_error'] = np.sqrt(mean\_squared\_error(test, predictions))

evaluation\_results['mean\_absolute\_percentage\_error'] = np.mean(np.abs(predictions - test)

/np.abs(test))\*100

G=evaluation\_results

#print(i)

print(evaluation\_results['r2\_score'])

print(evaluation\_results['mean\_absolute\_error'])

print(evaluation\_results['mean\_squared\_error'])

print(evaluation\_results['root\_mean\_squared\_error'])

print(evaluation\_results['mean\_absolute\_percentage\_error'])

list\_mean\_absolute\_percentage\_error4.append([i,evaluation\_results['mean\_absolute\_percentage\_error']])

list\_root\_mean\_squared\_error4.append([i,evaluation\_results['root\_mean\_squared\_error']])

k=k+1