from numpy import concatenate

from pandas import concat

from pandas import merge

from pandas import to\_datetime

from pandas import DataFrame

from datetime import timedelta

from keras.layers import LSTM

from keras.models import Sequential

from keras.layers import concatenate

from keras.layers import Dense

from sklearn.preprocessing import MinMaxScaler

import config

import tensorflow as tf

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

def fit(train):

scaler: ndarray of shape (n\_features,)

training\_y = train[[config.IMPUTED\_QTY]]

values = training\_y.values

# normalize features

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled = scaler.fit\_transform(values)

# frame as supervised learning

# for lag = 1

# reframed = series\_to\_supervised(scaled, 1, 1)

# for lag = 5

reframed = series\_to\_supervised(scaled, 5, 1)

# split into train and test sets

values = reframed.values

n\_train = len(values) - 5

train = values[:n\_train, :]

test = values[n\_train:, :]

# split into input and outputs

train\_X, train\_y = train[:, :-1], train[:, -1]

test\_X, test\_y = test[:, :-1], test[:, -1]

# reshape input to be 3D [samples, timesteps, features]

train\_X = train\_X.reshape((train\_X.shape[0], 1, train\_X.shape[1]))

test\_X = test\_X.reshape((test\_X.shape[0], 1, test\_X.shape[1]))

# design network

model = Sequential()

model.add(LSTM(60, input\_shape=(train\_X.shape[1], train\_X.shape[2])))

model.add(Dense(1))

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

# fit network

history = model.fit(train\_X, train\_y, epochs=75, batch\_size=15, validation\_data=(test\_X, test\_y), verbose=0, shuffle=False)

def predict(model, scaler, medicine\_item, df\_network, validation=True):

"""Predicts for a fitted LSTM Model

Parameters

----------

model: model object

Fitted LSTM model.

scaler: ndarray of shape (n\_features,)

Per feature relative scaling of the data.

medicine\_item: string

Medicine\_item (Medicine\_item Aggregate) for which forecast is being generated.

df\_network: Pandas DataFrame

validation: Boolean

# forecast

if validation:

TRAIN\_END = config.TRAIN\_END

else:

TRAIN\_END = config.TEST\_END

# Forecast df formation - with Lag = 4

FCST\_X\_START = TRAIN\_END - timedelta(weeks=config.FORECAST\_HORIZON + 3)

fcst = df\_network[

(df\_network[config.DATE\_COLUMN] >= FCST\_X\_START) & (df\_network[config.DATE\_COLUMN] <= TRAIN\_END)]

fcst = fcst[[config.IMPUTED\_QTY]]

values = fcst.values

# normalize features

scaled = scaler.fit\_transform(values)

# frame as supervised learning

# for lag = 1

# reframed = series\_to\_supervised(scaled, 1, 1)

# for lag = 4

reframed = series\_to\_supervised(scaled, 4, 1)

# split into input and outputs

values = reframed.values

pred = values

pred\_X = pred[:, :-1]

# reshape input to be 3D [samples, timesteps, features]

pred\_X = pred\_X.reshape((pred\_X.shape[0], 1, pred\_X.shape[1]))

# make a prediction

yhat\_pred = model.predict(pred\_X)

pred\_X = pred\_X.reshape((pred\_X.shape[0], pred\_X.shape[2]))

# invert scaling for forecast

inv\_yhat\_pred = concatenate((yhat\_pred, pred\_X[:, :]), axis=1)

inv\_yhat\_pred = scaler.inverse\_transform(inv\_yhat\_pred)

inv\_yhat\_pred = inv\_yhat\_pred[:, 0]

df\_predict = DataFrame(inv\_yhat\_pred)

df\_predict.columns = ["yhat"]

df\_predict["id"] = range(0, df\_predict.shape[0])

# for lag = 4

# df\_predict["id"] = df\_predict["id"] + 4

# Create Future DS

for j in range(0, config.FORECAST\_HORIZON):

date = TRAIN\_END + timedelta(days=7 \* (j + 1))

if j == 0:

future = DataFrame([date])

else:

future = future.append([date])

future = DataFrame(future)

future.columns = [config.DATE\_COLUMN]

future[config.DATE\_COLUMN] = to\_datetime(future[config.DATE\_COLUMN])

future["id"] = range(0, future.shape[0])

future = merge(future, df\_predict,on=["id"], how='left')

future = future.drop(["id"], axis=1)

fcst\_future = future.copy()

fcst\_future[config.MEDICINE\_ITEM\_ID] = medicine\_item

if validation:

# create dataframe prediction set

# fcst\_future = forecast[forecast[config.DATE\_COLUMN] > config.TRAIN\_END][[config.DATE\_COLUMN, 'yhat']]

# fcst\_future[config.MEDICINE\_ITEM\_ID] = medicine\_item

# add actual network values

temp\_df = df\_network[df\_network[config.DATE\_COLUMN] > TRAIN\_END][[config.MEDICINE\_ITEM\_ID, config.DATE\_COLUMN, config.QTY]]

fcst\_future = fcst\_future.merge(temp\_df,on=[config.DATE\_COLUMN, config.MEDICINE\_ITEM\_ID], how='left')

return fcst\_future