!pip install yfinance

!pip install statsmodels

!pip install arch

!pip install keras

import pandas as pd

import yfinance as yf

import numpy as np

import matplotlib

import matplotlib.pyplot as plt

from tqdm import tqdm\_notebook as tqdm

from data\_preparation import \*

from scipy.stats import pearsonr

from sklearn.metrics import r2\_score

import itertools

from sklearn.metrics import mean\_squared\_error as mse

def get\_stocks(ticker, start = "1900-01-01", end = None):

tickerData = yf.Ticker(tickerSymbol)

df = tickerData.history(period='1d', start=start, end=end)

if "Stock Splits" in df.columns:

df.drop(["Stock Splits"], axis = 1, inplace = True)

if "Dividends" in df.columns:

df.drop(["Dividends"], axis = 1, inplace = True)

#if "Volume" in df.columns:

#df.drop(["Volume"], axis = 1, inplace = True)

#if "High" in df.columns:

#df.drop(["High"], axis = 1, inplace = True)

#if "Low" in df.columns:

#df.drop(["Low"], axis = 1, inplace = True)

return df

tickerSymbol = 'MSFT'

df = get\_stocks(tickerSymbol, end = "2022-05-10")

df

df.apply(lambda x: x == 0).sum()

df.isnull().sum()

def fill\_missing(df):

new = df.copy()

idx = pd.date\_range(new.index[0], new.index[-1])

new = new.reindex(idx, fill\_value=0)

n = new.shape[0]

for i in range(1, n):

if new.iloc[i, :].sum() == 0:

new.iloc[i, :] = new.iloc[i-1, :]

return new

df = fill\_missing(df)

def plot\_open(df, tickerSymbol, start = None, end = None):

if not start:

start = df.index[0]

if not end:

end = df.index[-1]

plt.figure(figsize=(24, 10))

plt.title("Цена открытия акций " + tickerSymbol, fontsize = 20)

plt.plot(df["Open"][start:end])

plt.grid()

plt.xlabel('Дата', fontsize=15)

plt.ylabel("Цена", fontsize=15)

plt.show()

years = 5

data\_prep = get\_dataset(df,start=-years\*365, lag\_max = lag\_max)

data\_prep

from keras.models import Sequential

from keras.layers import Dense, Dropout, LSTM

from sklearn.preprocessing import MinMaxScaler

import math

np.random.seed(7)

daf = pd.DataFrame(data\_prep["Open"].copy())

dataset = daf.values

dataset

dataset = dataset.astype('float32')

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

dataset = scaler.fit\_transform(dataset)

train\_size = int(len(dataset) \* 0.959)

test\_size = len(dataset) - train\_size

train, test = dataset[0:train\_size,:], dataset[train\_size:len(dataset),:]

print(len(train), len(test))

def create\_dataset(dataset, look\_back=1):

dataX, dataY = [], []

for i in range(len(dataset)-look\_back-1):

a = dataset[i:(i+look\_back), 0]

dataX.append(a)

dataY.append(dataset[i + look\_back, 0])

return np.array(dataX), np.array(dataY)

look\_back = 1

trainX, trainY = create\_dataset(train, look\_back)

testX, testY = create\_dataset(test, look\_back)

trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))

testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))

model = Sequential()

model.add(LSTM(4, input\_shape=(1, look\_back)))

model.add(Dense(1))

model.compile(loss='mean\_squared\_error', optimizer='adam')

model.fit(trainX, trainY, epochs=100, batch\_size=1, verbose=2)

trainPredict = model.predict(trainX)

testPredict = model.predict(testX)

# invert predictions

trainPredict = scaler.inverse\_transform(trainPredict)

trainY = scaler.inverse\_transform([trainY])

testPredict = scaler.inverse\_transform(testPredict)

testY = scaler.inverse\_transform([testY])

trainScore = math.sqrt(mse(trainY[0], trainPredict[:,0]))

print('Train Score: %.2f RMSE' % (trainScore))

testScore = math.sqrt(mse(testY[0], testPredict[:,0]))

print('Test Score: %.2f RMSE' % (testScore))

r2=r2\_score(testY[0],testPredict[:,0])

print(r2)

trainPredictPlot = np.empty\_like(dataset)

trainPredictPlot[:, :] = np.nan

trainPredictPlot[look\_back:len(trainPredict)+look\_back, :] = trainPredict

# shift test predictions for plotting

testPredictPlot = np.empty\_like(dataset)

testPredictPlot[:, :] = np.nan

testPredictPlot[len(trainPredict)+(look\_back\*2)+1:len(dataset)-1, :] = testPredict

# plot baseline and predictions

plt.plot(scaler.inverse\_transform(dataset))

plt.plot(trainPredictPlot)

plt.plot(testPredictPlot)

plt.show()

Приложение(код) часть 2

!pip install yfinance

!pip install statsmodels

!pip install arch

import pandas as pd

import yfinance as yf

import numpy as np

import matplotlib

import matplotlib.pyplot as plt

import csv

from tqdm import tqdm\_notebook as tqdm

from numpy import genfromtxt

from data\_preparation\_lag import \*

from scipy.stats import pearsonr

from sklearn.metrics import r2\_score

def get\_stocks(ticker, start = "1900-01-01", end = None):

tickerData = yf.Ticker(tickerSymbol)

df = tickerData.history(period='1d', start=start, end=end)

if "Stock Splits" in df.columns:

df.drop(["Stock Splits"], axis = 1, inplace = True)

if "Dividends" in df.columns:

df.drop(["Dividends"], axis = 1, inplace = True)

if "Volume" in df.columns:

df.drop(["Volume"], axis = 1, inplace = True)

if "High" in df.columns:

df.drop(["High"], axis = 1, inplace = True)

if "Low" in df.columns:

df.drop(["Low"], axis = 1, inplace = True)

if "Close" in df.columns:

df.drop(["Close"], axis = 1, inplace = True)

return df

tickerSymbol = 'MSFT'

df = get\_stocks(tickerSymbol, end = "2022-05-10")

df

def fill\_missing(df):

new = df.copy()

idx = pd.date\_range(new.index[0], new.index[-1])

new = new.reindex(idx, fill\_value=0)

n = new.shape[0]

for i in range(1, n):

if new.iloc[i, :].sum() == 0:

new.iloc[i, :] = new.iloc[i-1, :]

return new

df=fill\_missing(df)

df

forward = 73

lag\_max = 3

history\_r2 = []

data\_prep = get\_dataset(df,start=-5\*365,lag\_max=lag\_max)

#data\_prep=df.copy()

data\_prep

def smooth(df, ws = 7):

df\_upd = np.copy(df)

for i in range(ws//2, len(df\_upd) - ws//2):

df\_upd[i] = np.median(df\_upd[i-ws//2:i+ws//2])

return df\_upd

def visualization\_result(X\_train,y\_train, X\_test,y\_test, predict, name, forward, ws = None):

if ws:

y\_train = smooth(y\_train, ws)

y\_test = smooth(y\_test, ws)

predict = smooth(predict, ws)

plt.rc('figure', figsize=(20, 8))

plt.plot(X\_train[-forward\*5:], y\_train[-forward\*5:], color = "g", label = "Train")

plt.plot(X\_test, y\_test, label = "Test")

plt.plot(X\_test, predict, lw=5, label = "Prediction")

plt.grid()

plt.title(f"{name}", fontsize = 20)

plt.xlabel('Дата', fontsize=15)

plt.ylabel("Цена", fontsize=15)

plt.legend(prop={'size': 20})

plt.show()

import itertools

from sklearn.metrics import mean\_squared\_error as mse

def all\_comb(params):

keys = params.keys()

values = (params[key] for key in keys)

combinations = [dict(zip(keys, combination)) for combination in itertools.product(\*values)]

return combinations

def Greed\_Search(model, X\_train, y\_train, X\_test, y\_test, lag\_max, params):

best\_mape = float("inf")

best\_combo = {}

combs = all\_comb(params)

for comb in tqdm(combs):

CB = TimeModel(model = model , look\_back= lag\_max, \*\*comb)

CB.fit(X\_train, y\_train)

y\_pred = CB.forecast(X\_test)

result = mse(y\_test, y\_pred)

if result < best\_mape:

best\_mape = result

best\_combo = comb

return best\_mape, best\_combo

X\_train, y\_train, X\_test, y\_test = get\_train\_test(data\_prep, forward)

train\_ind, test\_ind = data\_prep.index[:-forward], data\_prep.index[-forward:]

from sklearn.svm import SVR

from sklearn.pipeline import make\_pipeline

from sklearn.preprocessing import StandardScaler

sv = SVR(C=100,gamma=0.001,kernel='rbf')

sv.fit(X\_train, y\_train)

y\_pred = sv.predict(X\_test)

r2 = r2\_score(y\_test, y\_pred)

print(f"R^2 score(coefficient of determination): {r2} ")

history\_r2.append(r2)

visualization\_result(train\_ind, y\_train, test\_ind, y\_test, y\_pred, "SVR", forward, 4)

from sklearn.neighbors import KNeighborsRegressor

KN = KNeighborsRegressor(n\_neighbors=6,p=3,weights='uniform')

KN.fit(X\_train, y\_train)

y\_pred = KN.predict(X\_test)

r2 = r2\_score(y\_test, y\_pred)

print(f"R^2 score(coefficient of determination): {r2} ")

history\_r2.append(r2)

visualization\_result(train\_ind, y\_train, test\_ind, y\_test, y\_pred, "KNeighborsRegressor", forward, 4)