

Project of an application based on machine learning for stock market prediction

Code

Vladislav Sorokin

student record book number 295462

Dzianis Harbatsenka

student record book number 295452

Nikita Zakharov

student record book number 295464

thesis supervisor

prof. PW dr hab. inż., Jerzy Balicki

Warsaw 2021

Version 1.0

Warsaw,08.12.20

**DECLARATION**

We declare that this piece of work which is the basis for recognition of achieving learning outcomes in the Group Project course was completed on our own.

**Table of contents**

[1. SVR 4](#_Toc58338199)

[2. LSTM 5](#_Toc58338200)

[3. CNN 6](#_Toc58338201)

[4. Performance evaluation 8](#_Toc58338202)

[5. Glossary 14](#_Toc58338203)

# SVR

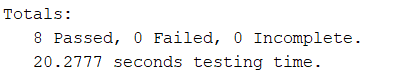
For SVR we have following functions:

* train\_svr – is used to train SVR model using data from .csv file and then save it to the models folder
* predict\_svr – is used to predict stock price using saved trained model
* RMSE – computes root-mean-square deviation
* MAPE – computes mean absolute percentage error

There are several unit tests provided for SVR implementation. There are 8-unit tests. The list of each test dedication is provided below:

1. Check if train works with gaussian kernel without auto hyper tunning
2. Check if train works with gaussian kernel with auto hyper tunning
3. Check if predict works for trained model with gaussian kernel
4. Check if train works with linear kernel without auto hyper tunning
5. Check if train works with linear kernel with auto hyper tunning
6. Check if predict works for trained model with linear kernel
7. Check if RMSE computed correct
8. Check if MAPE computed correct

The tests run in matlab. The output of test above is presented on Picture 1:



Picture 1 Unit test output

# LSTM

In order to test LSTM program and the way its functions are working there were created tests for each of the functions checking the fulfilment of the criteria that the function should perform.

The functions tested:

load\_data(ticker, n\_steps=50, scale=True, shuffle=True, lookup\_step=1,  
 test\_size=0.2, feature\_columns=['High', 'Low', 'Open', 'Close', 'Volume', "OpenMax", "OpenMin", "Day"]

)

create\_model\_test(input\_length, units=256, cell=LSTM, n\_layers=2, dropout=0.3,  
 loss="mean\_absolute\_error", optimizer="rmsprop"

)

train\_model()

test\_model()

numberOfEpochsChanged(self)

numberOfUnitsChanged(self)

numberOfLayersChanged(self)

start()

The TEST\_1 checks if the data is loaded and created correctly.

The TEST\_2 checks if the model is created

The TEST\_3 checks if the train procedure is passed and the model if saved after its finished

The TEST\_4 checks if the model is passing testing and prediction phase, that values of accuracy and loss are calculated

The TEST\_5 checks if during the GUI epochs field modification the global parameter EPOCHS is changed.

The TEST\_6 checks if during the GUI units field modification the global parameter UNITS is changed.

The TEST\_7 checks if during the GUI epochs field modification the global parameter N\_LAYERS is changed.

The TEST\_8 checks if during the GUI is initialized and pops up to the user.

parameters.py

Below the global parameters are defined for the program. It includes the settings of LSTM network, testing procedure and other useful parameters.

stock\_prediction.py

The below script servers as a help for the main function. The first function in it

def load\_data(ticker, n\_steps=50, scale=True, shuffle=True, lookup\_step=1,  
 test\_size=0.2, feature\_columns=['High', 'Low', 'Open', 'Close', 'Volume']):  
The function load\_data downloads the dataset from WIG\_20 and transforms the data into the form comprehensive for the network. The result of the function and the way the data will look like depends on the parameters provided. The specific feature columns can be set. The dataset can be shuffled or not.

def create\_model(input\_length, units=256, cell=LSTM, n\_layers=2, dropout=0.3,  
 loss="mean\_absolute\_error", optimizer="rmsprop"):

The function create\_model defines and initializes the network with the predefined settings.

start.py

The main script is provided below. The function InitUT initializes the GUI of the application. Then two functions for each button click event are provided (Train and Test).

def button1\_clicked(self):  
Triggers the training procedure to start. The model is initialized, the data is loaded and then the network starts to get fitted.

def button2\_clicked(self):  
Triggers the prediction of the model after training process is finished. At this stage the network evaluates the predicted values.

def plot\_graph(self, model, data)

The function plots the graph to see the comparison between real and predicted values.

# CNN

Function to initialize CNN

model\_initialization <- function(timeseires\_length)

Takes integer as input and uses in initialization of a input shape of a model. Initialized model is returned.

Function to train CNN

model\_training <- function(model, X, Y)

Takes initialized model, train data set and validation array as inputs and performs a model training on predefined hyperparameters. X and Y dimensions should fit the input shape of a model.

Function to predict with CNN

model\_prediction <- function(model, X)

Takes trained model and test data set as inputs and performs a test prediction. The array of predicted values is returned.

Function to save a model

model\_save <- function(model, name)

Takes model and string name as inputs and saves a model with specified name in local storage.

Function to load existing model

model\_load <- function(name)

Takes string name as an input and return a model form a local storage of a chosen name if exists.

Function to prepare data for train, test and prediction

prepare\_data <- function(data\_frame, timeseires\_length, t\_num, p\_num = 5)

Takes stock data frame, length of a single timeseries, test length integer and prediction length integer as inputs and returns a list of data sets, needed for program execution.

Function to make prediction based on backpropagation

real\_prediction <- function(model,last\_ts,pnum = 5)

Takes a trained model, timeseries array and prediction length integer as inputs and performs a prediction based on a backpropagation.

Function to divide data into train, test and prediction sets

divide\_data <- function(data, tnum, ts\_length, pnum = 5)

Takes data set, test length integer, single timeseris length integer and prediction length integer as inputs and returns a list of actual, train, test and prediction data sets.

Function to load data from a chosen csv

choose\_data <- function(name)

Takes a name as input and returns a data frame of a data if exists.

There are several unit test provided for CNN implementation. There are 8 unit tests which in total check 15 testing conditions. The list of each test dedication is provided below:

1. Check if upload data function works correctly
2. Check if divide\_data function works properly
3. Check if prepare\_data function works properly
4. Check if model initialization is working and produces an initialized model
5. Check if model training is functioning
6. Check if model prediction produces the output
7. Check if append\_timeseries works properly
8. Check if real prediction of a model function properly and produces output.

The tests where run in R with usage of “testthat” package. It provides with useful tools testing R code. The output of test above is presented on Picture 3:



Picture 3 Unit test output

# Performance evaluation

There is performance evaluation provided on 3 different periods different in sizes. For specifically, the aim was to determine, how models behave regarding the train size. For testing purposes 1 month, 3 month and a year time intervals were taken. Results of such tests are presented on Figure 1,3,5. Next scope is to test how models could make predictions based on backpropagation. The main idea here is to feed a model with a single value, a starting point, make a prediction based on that values and use predicted values as next input for prediction. That step is performed several steps, 5 times in our case. That may show, how models could look for dependencies in such dynamic data. Results can be observed on Figures 2,4,6.

Figure 1 shows the results of training of each model on a 1-month period. As it may be seen, SVR model was most accurate in terms of training, where CNN and LSTM performed a little bit worse, but not significantly.

Figure 1 A month train comparison

Figure 2 shows the results of prediction of each model on a 1-month period. SVR was able to make the most accurate result on a short period, thus such training length is not perfectly fit CNN and LStM models.

Figure 2 A month prediction comparison

Figure 3 shows the results of training of each model on a 3-month period. CNN and LSTM where able to extract more features these time, thus there training curvature was much close to actual and SVR lines.

Figure 3 3-months train comparison

Figure 4 shows the results of prediction of each model on a 3-month period. LSTM and SVR where able to extract dependencies form such train data and perform good prediction.

Figure 4 3-month prediction comparison

Figure 5 shows the results of training of each model on a year period. It’s hard to notice, how exactly these graphs behave, however, that indicates that all 3 models where really close to the actual value, thus blue line rarely can be seen

Figure 5 A year train comparison

Figure 6 shows the results of prediction of each model on a year period. Each of a model was able to dependencies and perform better prediction overall.

Figure 6 A year prediction comparison

In conclusion, models may be modified to make backpropagation prediction even better. In can be achieved by change of the model inputs, adding more dependencies, and probably come up with a bit more complex models, with more layers included, however, we have to be aware of overfitting which implies to even worse prediction overall.

# Glossary

CNN – Convolution neural network

SVR - Support Vector Regression

LSTM - Long short-term memory

MAPE - Mean absolute percentage error

RMSE - Root-mean-square deviation