BABEŞ BOLYAI UNIVERSITY, CLUJ NAPOCA, ROMÂNIA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

Stock Market Prediction

– ITSG report –

Team members
Radu Galan, ICA, 256, radu.galan@stud.ubbcluj.ro
Ligia Novacean, ICA, 256, nicoleta.novacean@stud.ubbcluj.ro
Daniela Stircu, SE, 248, daniela.stircu@gmail.com

2021-2022

Abstract

In this report, we propose an application that can provide valuable information to anyone who wants to invest in stock market, but lacks the expertise. The information is provided through an intelligent assistant that was trained on historical stock price data to perform short- to mid-term stock price prediction using machine learning (ML) techniques. The stock market dataset was collected from Yahoo Finance and contains daily open, close, high and low prices for 1700 companies ordered by market value, starting from the moment of their IPO. We propose two different approaches for the stock data prediction: a recurrent neural network architecture named Gated Recurrent Unit (GRU) - our main method - and a trend prediction method based on template matching, followed by statistical prediction. Using Yahoo Finance, daily new stock information is collected and used to perform predictions for the upcoming days. Experimental results using the GRU approach show that good performance can be obtained for early days in the prediction, but the performance decreases for later days. Hence, we decided upon a 5-day prediction, obtained by an averaging the results in an ensemble of three GRUs: one predicts for 1 day ahead, one predicts for 3 days and one predicts for 5 days.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 What? Why? How?

One of the most important responsibilities of an adult is properly managing oneâs finances. Besides monthly or weekly budgeting this also refers to the ability to efficiently take advantage of all oneâs economies. A balanced investment portfolio will usually contain both long-term or short-term investments. More and more people are starting to overcome the knowledge barrier and begin investing themselves through private brokers instead of using a regular economy or investment account in a bank. However, the process of learning is still tedious and unavailable to most people but this does not mean that they should not be able to make good investments. All these people could take advantage of an assisting tool that would make predictions and recommend suggestions. Furthermore, such a tool, if complex enough, could also be of interest for investors.

Therefore, the goal of the current project is to provide investment recommendations on live data via an accessible web application. The web application integrates two intelligent assistants that predict the stock price for a list of selected financial instruments. The web app user can search a stock of interest, add it to the watch list and will afterwards be provided with daily updates about the price evolution for the stock.

To approach the stock price prediction problem, we have two different strategies:

- Gated Recurrent Unit (GRU): is a type of recurrent neural network that allows us to employ the time series nature of the stock prices for prediction generation. Using GRU, we can make single or multi-step predictions.

- Template Matching Based: for this method, trend attribute translation is performed via template matching. Afterwards, using a statistical model prediction, we generate the stock price prediction.

1.2 Paper structure and original contribution(s)

The main contribution of this report is to present two intelligent algorithm for solving the problem of stock price prediction.

The second contribution of this report consists of building an intuitive, easy-to-use and user friendly software application. Our aim is to build algorithms that will help people interested in investments to make better decisions regarding stock trading.

The present work contains bibliographical references and is structured in seven chapters as follows.

The first chapter is a short introduction in the stock price prediction problem, along with the two approaches we chose to tackle the problem.

The second chapter describes the previously introduced problem from a scientific and formal perspective.

The chapter 3 presents a short literature review, highlighting most frequently used techniques, along with their advantages and disadvantages.

Chapter 4 details the recurrent neural network and clustering approaches employed to perform predictions, while in chapter 6 we provide more details about the experimental setup and the obtained results.

Finally, in chapter 7 we discuss results and reach a set of conclusions, along with future work perspectives.

Chapter 2

Scientific Problem

In financial markets, machine learning (ML) algorithms gained popularity in past years in making investors improve their decisions. Despite the attractive return that can be obtained by trading financial instruments, investments are associated with a high risk due to the uncertainty and volatility of the market. Investors need to understand the behaviour of individual stocks and, at the same time, make effective and timely investment decisions. Therefore, machine learning algorithms have been used to help investors make the right decisions.

2.1 Problem definition

The stock market prediction problem can be approached using two different strategies: fundamental analysis and technical analysis. In fundamental analysis, one estimates the intrinsic value of a stock based on a set of internal and external factors that can affect the holding company (*e.g.,* financial performance, business environment, management changes, social and political impact). Fundamental analysis is well suited for long-term predictions. On the other hand, technical analysis requires the analysis of historical financial data in order predict future stock values. Generally, these predictions are based on trends in daily closing, opening, high and low prices, but other features may be considered in order to increase prediction accuracy (e.g. news). Based on these characteristics, technical analysis works well for short-term predictions.

The goal of the developed application is to provide short-term predictions, hence the adequate strategy is technical analysis. The historical financial data used for technical analysis is, in fact, a time series. Hence, the stock prediction problem is a time series prediction problem. The chosen algorithm should be able to find patterns underlying the stock time series in order to make right forecasts. Given the large amount of data that needs to be analyzed to accomplish such a task, one can employ

intelligent algorithms to achieve the goal.

2.2 Problem formulation

In this section, we formulate the short-term stock prediction problem as a time series forecasting problem for mid-term horizon. The inputs of the problem are daily close, open, high and low stock prices. These features represent the following information:

- Open price: opening price of the index (PoI) at the start of the trading day
- Low price: minimum PoI for a trading day
- High price: maximum PoI for a trading day
- Close price: PoI at the moment when the market closes, for a given day

The dataset contains data for a number of 1700 companies, from the moment of their IPO. The companies were ordered and selected based on the market value.

The previously mentioned sequence of ordered stock price related observations, equally distributed along time intervals, form a time series. The data constructed by joining all daily information shows non-linear complex patterns. Using complex non-linear modelling techniques, the goal is to forecast one or multiple future values of the time series. While the input time series data has four features, the output has only one feature: the close price. Therefore, we can formally express the current problem using Eq. 2.1, where x_i represents the input vector at time i ($x_i = [close_i, open_i, high_i, low_i]$), y_i represents the output vector at time i ($y_i = [close_i]$) and f is the function representing the learnt machine learning model.

$$f([x_{n-p}, x_{n-p+1}, ..., x_n]) = [y_{n+1}, y_{n+2}, ..., y_{n+k}] \tag{2.1}$$

Chapter 3

State of the art/Related work

Stock market is one of the major domains that investors are interested in, thus stock market price prediction has been a hot research topic for a long time. According to literature, a large number of algorithms have been employed so far for stock price prediction.

Initially, statistical tools were applied for stock price forecasting. Methods in this category include the Fourier transform Autoregressive (AR), Moving Average or Linear Regression. Due to the high data volatility characteristic to financial markets, statistical approaches proved unstable. Gradually, the interest shifted towards machine learning algorithms such as support vector machines (SVMs), neural networks or recurrent neural networks (RNNs).

In [1], the authors experiment with autoregressive integrated moving average (ARIMA) and artificial neural networks (ANNs) for stock price prediction. The study uses published historical daily stock data from NYSE, more precisely the *Dell Inc. index* from August 17, 1988 to February 25, 2011. The resulting number of observations is 5680, with the following features: open price, low price, high price, close price and traded volume. Data is differentiated in order to make it non-stationary. To measure performance, the authors use the relative error. The resulting average error for a one month prediction is is 0.608 for ARIMA and 0.8614 for ANNs. Numerically, results indicate that ARIMA performs better, but in reality the ARIMA model fails to generate non-linear outputs for multiple days predictions.

A large number of papers in the literature use recurrent neural networks, which allow one to make use of the time series nature of stock prediction data. In [2], multivariate bidirectional and stacked LSTMs (Long Short-Term Memory) and GRUs (Gated Recurrent Units) are used for both short and long-term forecasting. The dataset consists of historical data from the SP500 index, downloaded from Yahoo finance for the January 1, 2010 - November 11, 2017. The S&P500 dataset is one of the most relevant benchmarks and a leading stock market index for 500 companies listed on the US

market. Trend and seasonality are removed from the data by differencing. The results of recurrent neural networks architectures are compared to the results of a Multi-Layer Perceptron Artificial Neural Network. Results indicate that stacked bidirectional LSTMs and GRUs perform similarly for both short and long-term forecasting, outperforming the MLP network.

In [4], short and long-term forecasting is achieved using LSTM and inputs data with gaps. The authors emphasize the difficulty brought by long-term predictions and attempt to overcome it by providing as input not continuous data, but rather closing prices selected at discrete time steps. Moreover, data provided as input is non-stationary. Experiments are performed on four time series datasets. Obtained results outperform the performance achieved by other papers, especially for long-term predictions (e.g. 20 days ahead). Furthermore, the skipping parameters that determines the gap between selected time stamps has a strong impact on the ability to predict for a long horizon.

Despite the popularity of the topic, literature review indicates that some problems that make it difficult to compare methods to each other. Firstly, the used datasets are different. The S&P500 is probably the most popular dataset, but the time frame used to train and evaluate is often different. Given the significant changes in stock behaviour in past years, using different time frames makes performance comparison invalid. Secondly, the metrics used to evaluate performance are different across papers. Most papers report MSE and MAE, but in the context of different datasets and the previously mentioned changing behaviour of stocks, these metrics lose their relevance. Besides MSE and MAE, some articles compute relative errors. However, the formulas are slightly different as some of the compute the relative error with respect to the absolute value, other with respect to the square value, other with respect to the mean. Reaching a common ground on appropriate metrics and establishing benchmark datasets is critical for enabling the contribution of researchers to the stock price prediction topic.

Chapter 4

Investigated approach

To address the daily stock price prediction problem, we adopted two different approaches. The first one uses recurrent neural networks, namely the Gated Recurrent Unit network (GRU), to predict the stock price for a given number of days ahead. The second one uses template matching and statistical prediction strategies. However, our main method is the GRU approach, so further explanations will focus on this one.

4.1 Gated Recurrent Unit

The architecture used to solve the stated problem is based on Recurrent Neural Networks (RNNs), a type of neural networks that use the previous input to compute the current output. One of the biggest problems with RNNs is the vanishing gradient. Variations of RNN, such as LSTM and GRU, address this issue with the help of memory cells known as LSTM cells, respectively GRU cells. These network architectures are capable of learning long term dependencies in data, hence they are highly suitable for time series forecasting.

GRU [3] uses a gated process to control the flow of information between the cells of the network. At each moment of time t, the GRU cell has two inputs (see Fig. 5.3): the input x_t and the hidden state from the previous time stamp h_{t-1}. It only has one output, a new hidden state h_t, that will be passed to the next timestamp. The two types of gates (see Fig. 5.3 in a GRU cell are:

- Reset gate: responsible for short-term memory (*i.e.,* the hidden state); it helps the model decide how much of the earlier time steps information to discard
- Update gate: responsible for long-term memory; it helps the model renew the current memory of the network and select the relevant inputs it needs to remember

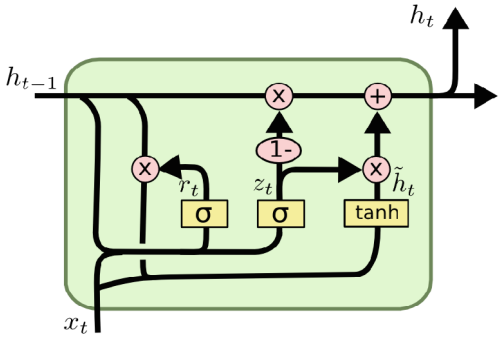

Figure 4.1: The structure of a Gated Recurrent Unit (GRU) cell.

Mathematically, for a given time stamp t, let x_t be a the input. The reset gate is defined by Eq. 4.1, where H'_t is the candidate hidden state at time t, W_{xh} and W_{hh} are weight parameters, b_h is the bias and $*$ denotes an element-wise multiplication. With this definition of the gate, the influence of previous state can be controlled by the element-wise product between the reset gate and the previous hidden state. When the values in the reset gate are close to 0, the candidate hidden state is the result of an MLP with x_t as input, so previous hidden states are reset to defaults.

$$\hat{H}_t = tanh(x_t W_{xh} + (r_t * h_{t-1} W_{hh} + b_h) \tag{4.1}$$

The effect of the update gate z_t is defined by Eq. 4.2. To decide the extent to which the new hidden state h_t is the old state h_t versus the extent to which it is the candidate state, we take the element-wise combinations of the two states. When the update gate is close to 0, the new hidden state becomes the candidate hidden state. Then the update gate is close to 1, the old state is retained. Thus, information from the current input is discarded.

$$H_t = z_t * h_{t-1} + (1 - z_t) * \hat{H}_t \tag{4.2}$$

Based on the previously described operation mode, GRU cells can be used to analyze and explore patterns in data through a self-learning process, while remembering information for a long time. GRU has fewer parameters than LSTM, so it is generally faster and easier to train.

To solve our problem, we used uni-directional stacked GRUs (see [2]). In stacked networks, multiple RNN layers are cascaded. In our case, we cascaded two GRU layers. The obtained architecture is depicted in Fig. 4.2. We train multiple models, each performing prediction for a different number of days. To model prediction for more than one day, we change the number of output neuron in the last fully connected layer of the architecture to the targeted number of days. The objective function used for

all the experiments is mean squared error. As optimizer we use Adam, along with a reduce-on-plateau learning rate scheduler.

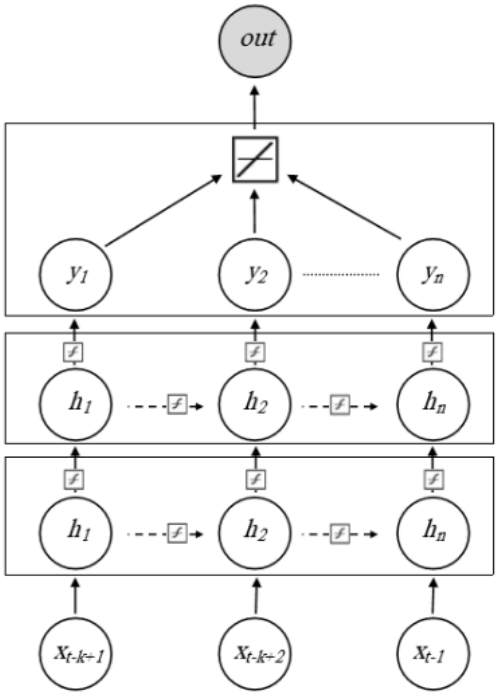

Figure 4.2: Structure of a stacked RNN architecture.

Chapter 5

Technological stack

5.1 Versioning tools

We used github to save and manage the different versions of a file (or set of files) so that any version is retrievable at will. Git also makes it easy to record and compare different file versions. This means that the details about what changed, who changed what, or who initiated an issue are reviewable anytime.

5.2 Software Design

In this project, we chose a distributed architecture: client-server (see Fig. 5.1). The Server is where all the processing, computing and data handling is happening, whereas the Client is where the user can access the services and resources given by the Server (Remote Server). The language and rules of communication are defined in a HTTP communications protocol. To formalize the data exchange even further, the server implements an API. To lower the impact on the server we separated the back-end service and front-end display. On the server side we opted for MVC pattern and the client was implemented as a single page application.

Currently, the system only supports off-line training. Using the initially collected dataset, we trained multiple models and then selected the ones we want to use in the final version of the application based on performance. As the number of data points that can be collected is limited given our context, there is currently no need to continuously update the model with new batches of data. For inference we use an online setup, where we predict on demand using the server.

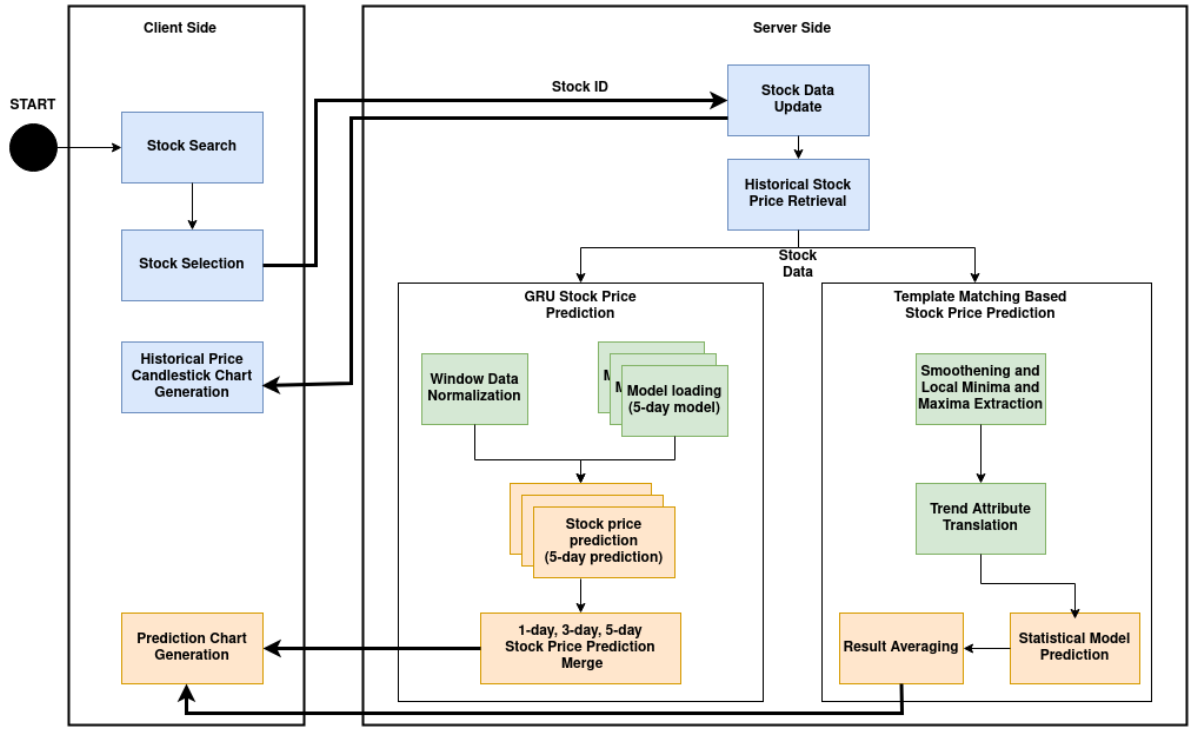

Figure 5.1: Deployment diagram of the developed application.

5.3 Software Pipeline

App flow diagram is shown below.

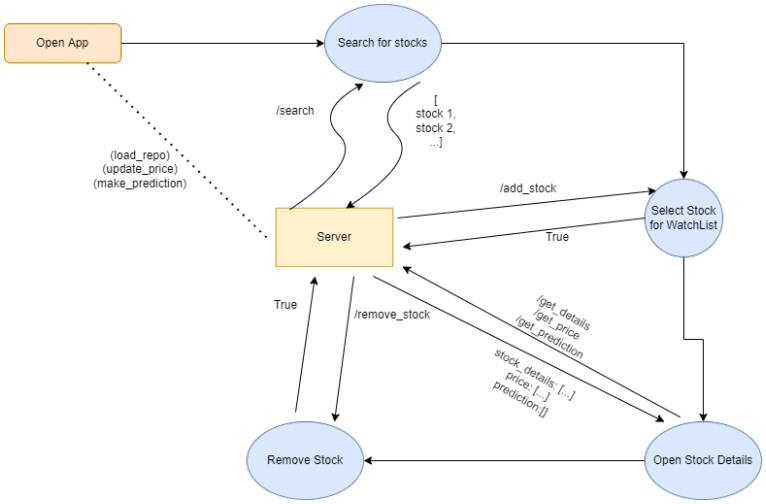

Figure 5.2: App-flow

5.4 Software Code

For the development of the intelligent algorithms, we used Python as programming language and the PyTorch library. PyTorch has an implementation of the GRU model, hence the creation of the network architecture was an easy step in the pipeline. The development was done in Visual Studio Code and PyCharm and debugging was used to check data integrity in the entire pipeline.

The server was built in Flask, a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries.

For the user interface we created a web application. The main technology stack used were HTML to create the web pages, CSS to style the look and formatting of the web pages and JavaScript to add behavior to web pages. Our goal was to faster transitions and make the website feel more like a native app so we made a single-page application using react, a JavaScript library for building user interfaces. For the communication with the server we used web APIs. The API server was built in Python.

To identify bugs, issues, and defects in the software application we used manual testing.

5.5 Model Deployment

For the client app we chose to make the deploy via netlify, a platform which provides hosting and deployment tools for websites whose source files are stored in Git. We attached to a netlify job our GIT repository, master branch and the build configuration. Each time we make a push to the master branch, the netify job will automatically make a new deploy for us.

For the client to remotely access the web-server over the internet we used socketxp which creates a secure HTTP tunnel to the localhost web service.

The application ca be found at this url: https://friendly-hodgkin-d3a53e.netlify.app/

5.6 Continuous Delivery

To ensure the continuous delivery for our application we use agile process. The method doesnât require short release iterations and simply allows the commitment of new pieces of code when they are ready.

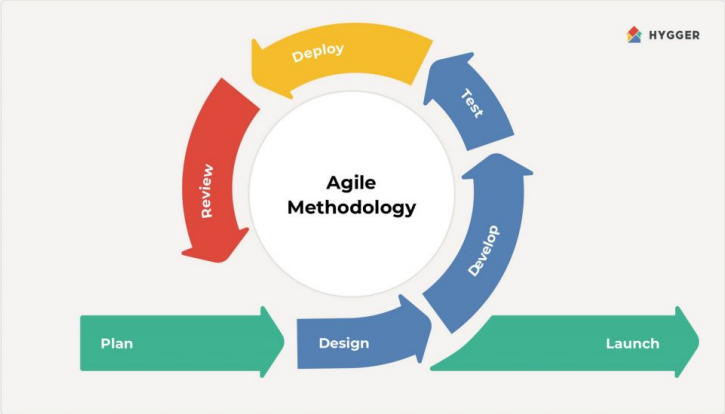

Figure 5.3: Agile Process

Chapter 6

Application (numerical validation)

6.1 Data

One of the most important steps for an ML project is data acquisition. In our case, we collected historical stock price data using Yahoo finance. We selected data for 1700 companies based on market value. Therefore, our dataset contains information about companies like Apple, Microsoft, Tesla or Amazon. For every company, we use all the price related data available in Yahoo Finance. Consequently, the number of days in the time series for a company depends on the date of the Initial Public Offering (IPO). However, historical prices in Yahoo Finance do not go back earlier than 1970.

For storing data, we chose an approach inspired by data lakes. More precisely, we store the collected data as unstructured, without any previous pre-proccessing. The reasoning behind this choice is that different algorithms may require different approaches for data cleaning and unstructured data is more flexible and scalable. When new historical stock price data is fetched for a company, it can be stored immediately in the data lake. If a specific algorithm needs data, the relevant portion of stock price information is extracted from the data lake, cleaned according to the requirements and further processed.

To visualize stock price data, two approaches are very popular. Firstly, if one wants to use all the available prices for a day (open, close, low, high), the adequate representation is a candlestick chart (see Fig. 6.1. The open and close prices are represented by the thick body of the candlestick, while the high and low prices are represented by the candle wick. Densely packed with information, the candlestick chart is probably the most frequently used type of financial chart. It allows one to clearly represent trading patterns over short periods of time. Secondly, if one is only interested in plotting one of the price features (*e.g.,* close price), line charts can be used to display the time series data.

After data has been collected and visualized, every method processes it according to its own re-

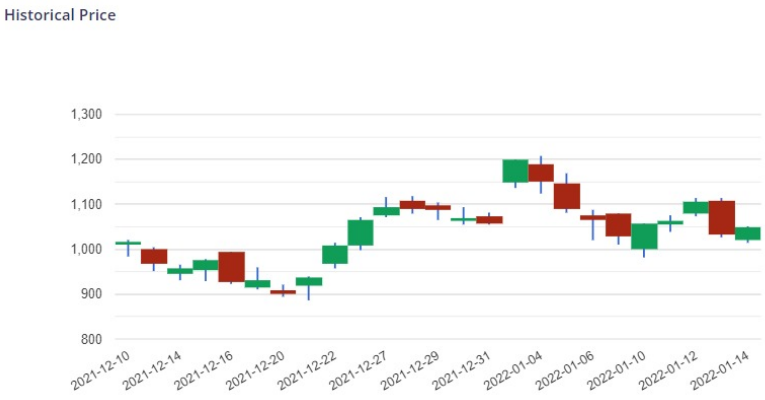

Figure 6.1: Tesla Daily Stock Price Candlestick Chart.

quirements.

6.1.1 GRU data pipeline

Using GRU, we perform experiments where we train the network individually for every stock and experiments where we train a global model using the first 100 companies, sorted by market value. The assumption of the second approach is that patterns in data are not tightly coupled to the stock they represent and a universal model suffices to identify patterns in all data.

For the approach based on GRU, the following steps are used to process data:

- Data cleaning: remove the days where any of the four input features (close, open, high, low) have a NaN value
- Data pre-processing: two main steps are part of the pre-processing workflow, namely making data stationary and normalization. Experiments are performed with multiple combinations for these two steps. It is not mandatory to make data stationary, but in case this option is selected, two options are further available: differencing as absolute values or differencing as relative values. For normalization, one can use either min-max normalization or standardization.
- Sliding window generation: in order to provide a stock as input to the RNN architecture, we split it into overlapping sliding windows, with an overlap step of 1. The size of the sliding window is one of the parameters we experiment with.
- Randomization: the windows are shuffled such that they are not taken in consecutive order for the train/test split. The reasoning behind this choice, which was not observed in other papers in literature, is that patterns in stock prices change over time. For example, in recent years,

the stock market has a much higher volatility and more sudden price increases can be observed. Therefore, we aim to have these patterns in both training and test data.

- Train/Test split: We split the large windows array in 70% train data and 30% test data.

Finally based on these steps, the following are the inputs and output for the GRU network:

- Input: mini-batch containing the close, open, high and low prices for windows of k consecutive days, considered past time stamps
- Output: mini-batch containing the predicted close price for n consecutive days ahead, for every window in the input

6.2 Methodology

6.2.1 GRU experiments

The proposed GRU method was implemented in Python, using PyTorch. For the performed experiments, we vary the value of parameters like then number of hidden units, the data pre-processing steps, the size of the past information sliding window, the size of the prediction or the data used for training. Throughout all of our experiments, the objective function is mean squared error (MSE), the optimizer is Adam and the learning rate scheduler is of type reduce on plateau. Based on these information, we can formulate the following questions that are tested by out experiments:

- Do we need individual models for every stock or can we train a global model that learns valid patterns for all stocks?
- How does the network perform for multi-step prediction? Does the performance decrease as we move several days ahead?
- Does a larger past window size result in better performance?

The experimental study was conducted for two types of datasets: one dataset that contains aggregated information from all the stocks and one dataset where stocks are treated individually and one model is trained for every one of them. For each dataset, the shortest term is 1 step ahead and the longer horizon prediction is 3, 5 or 7 steps ahead. After that, the results on different metrics are calculated and reported for the test sets.

To evaluate the performance of a network, we compute three different metrics: root mean square error (RMSE) (Eq. 6.1), mean absolute error (MAE) (Eq. 6.2) and the relative error (%) (Eq. 6.3).

In Eq. 6.2-6.3, s_t denotes the actual closing price and $\hat{s}_t$ denotes the predicted closing price. When prediction is performed for multiple days ahead, we compute the previously mentioned metrics for every individual day. Moreover, to monitor the training process we plot the train and test losses cross epochs. Finally, we create prediction line charts, where we evaluate the performance of the network on individual stocks.

$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(s_t - \hat{s}_t)^2} \tag{6.1}$$

$$MAE = \frac{1}{n}\sum_{t=1}^{n}|s_t - \hat{s}_t| \tag{6.2}$$

$$RelativeError(\%) = 100 * \frac{1}{n}\sum_{t=1}^{n}|\frac{s_t - \hat{s}_t}{s_t}| \tag{6.3}$$

6.3 Results

6.4 GRU Experimental Results

In order to answer the questions in Section 6.2.1, we performed a series of experiments that we monitored and logged using three different techniques. Firstly, we monitor the training procedure by plotting train and test losses. Secondly, once the model is trained, we compute RMSE, MAE and relative error (%) for both the training and test set. Finally, we plot the actual versus predicted closing price for the entire time series as a line chart (see Fig. 6.2).

We analyzed the impact of multi-step prediction on the prediction performance, more precisely we monitor whether the performance decreases once we try to predict for more days or whether it decreases the further away the day is. Tbl. 6.1 shows the relative error obtained on the test set, for the model trained using 100 aggregated stocks, across multiple days, for models with different prediction sizes. For 1, 3 and 5 days ahead experiments, the number of hidden units is 32, while for 7 days ahead there are 64 hidden units. The past window size, for all experiments, is 30 days. Results indicate that increasing the prediction size does not have an impact on the model performance on early days. However, further away time steps have a higher relative error. Consequently, increasing the size of the prediction comes with higher risk or wrong prediction in later days.

Furthermore, for the 1-step prediction (i.e. one day ahead), we experiment with sliding window size. On one hand, we register a 3.7% increase in relative error when we reduce the sliding window

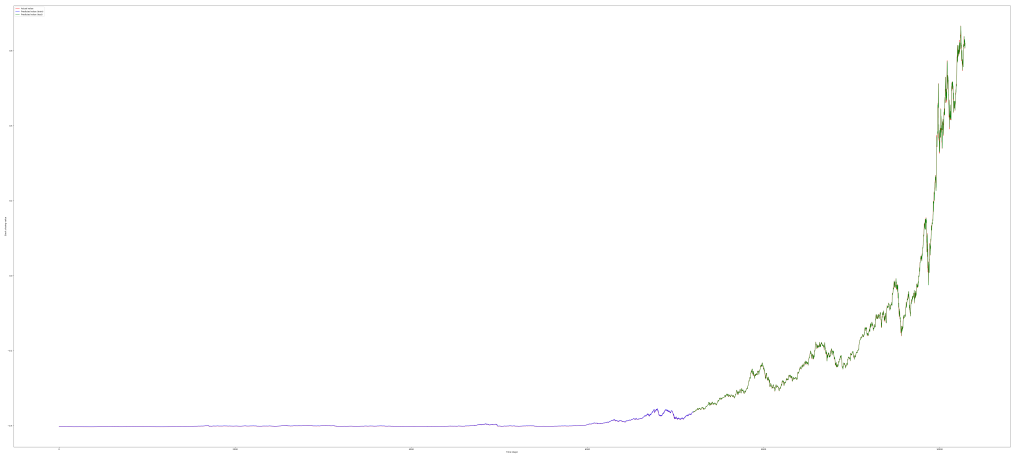

Figure 6.2: Actual vs. predicted closing price for the MSFT stock.

Table 6.1: Global model performance (relative error (%)) for every prediction step, based on the number of steps in the model prediction.

Day	**1-day model**	**3-day model**	**5-day model**	**7-day model**
Day 1	7.75	7.63	7.75	7.57
Day 2	-	11.01	11.28	11.35
Day 3	-	14.12	14.09	13.78
Day 4	-	-	15.09	15.91
Day 5	-	-	16.03	16.15
Day 6	-	-	-	19.58
Day 7	-	-	-	18.91

size from 30 days to 10 days. On the other hand, when we increase the sliding window size from 30 to 50 days, we also register an increase in relative error, even though much smaller (1.2%). Then, 30 days is the preferred time frame for further experiments.

Another important component of the research was to analyze whether individual models perform better than models trained on aggregated datasets. Firstly, it is important to mention that models trained on individual stocks are prone to overfitting. Early stopping and dropout were used to prevent overfitting. The results depicted in Tbl. 6.2 indicate that the model trained on a set of stocks performs, in average, better than individual stocks. However, we notice the statistical difference is not significant enough, given the large standard deviations. There are stocks for which global patterns perform well, but there are also stocks where it results in high relative errors. However, considering the very high cost of training individual models, along with the difficulty introduced by overfitting, the global approach is preferred over the individual models approach on the long run.

Table 6.2: Individual stock training vs. global stock training performance. Reported as the average over 10 stocks for 1-step ahead prediction.

Dataset type	**RMSE**	**MAE**	**Relative Error (%)**	**Relative Errors Std**
Individual stocks	0.035	0.074	15.56	3.46
Aggregated stocks	0.019	0.041	7.22	3.13

6.5 Discussion

Stock price prediction is a difficult problem, given the noisy data, the increasing market volatility and the large number of external factors that affect price. We tried to develop two approach and experiment with them in order to identify the setup that can provide the users of our app with a performance as good as possible.

For GRU related experiments, we focused on understanding whether we need individual or global models and on the impact of the prediction size on the network performance. We found out that individual models are difficult to train as they are prone to over-fitting. Also, the global model results in better average performance on a subset of 10 evaluated stocks. Considering the fact that it is easier to train a single model than it is to train one model for every stock, it is preferable to use the so-called 'global' approach for the stock prediction problem. The downside of this problem is that

stocks with peculiar trends might be predicted very poorly. We hypothesize that in a setup where only stock prices are used as input features, patterns repeat themselves amongst stocks and peculiar patterns characterizing a single stock are isolated cases. However, if one was to consider not only prices as inputs, but also news or information about the behaviour and reaction of investors to the news, developing individual models might be a better solution.

We also performed GRU experiments to understand the behaviour of prediction across multiple dats. Results show that further away days have a higher relative error than closer days. However, the first day prediction of a 1-day ahead model has almost the same relative error as the first day prediction of a 7-day ahead model. Thus, the network is properly trained, but the historical price information is not enough for it to be make adequate long-term predictions. Based on these experimental results, we decided to make prediction for 5 days. Moreover, we employ 3 models: one performing prediction for 1 day, one for 3 days and another one for 5 days. To obtain the final prediction for a 5-day time frame, we average them together. Using multiple models to decide on the prediction for a single day, like an ensemble technique, improves the confidence of the system.

Chapter 7

Conclusion and future work

We developed an application that performs stock price prediction and helps users, ranging from investors to people who lack expertise, to improve their stock trading decisions. The resulting web application is has a client-server architecture, developed in Python, where we allow users to add stocks of interest to a watch list and provide them with daily updates for price predictions, based on live data acquired from Yahoo Finance.

Two methods were developed. The first method uses an ensemble of recurrent neural network architectures, namely Gated Recurrent Unit (GRU), to perform stock price prediction for 1, 3 and 5 days ahead. The second one performs trend extraction from historical price information, then uses template matching to translate trend attributes and statistical modelling to generate the final prediction. Throughout this report, the focus is on the GRU-based method. GRU experimental results indicate that for a stock market dataset containing only price information, a single model can be used to learn and predict patterns in all the stocks. Therefore, there is no need to train individual models for each available stock. Furthermore, we find that the relative error increases as we try to predict more days. Then, our GRU predict spans only 5 days and is provided using a set of 3 averaged predictions.

All these aspects lead to further work ideas. More precisely, an interesting and most likely helpful next step would be to integrate news as input features. GRU allows us to easily integrate a larger set of features. Then, we can improve our models and further experiment with strategies that can be used for longer-term price prediction. Moreover, we could switch from an off-line training method to online training and perform periodic updates to our model.

Finally, we can conclude that we achieved our goal of providing users with a tool that can guide them through the process of stock trading. We provide two methods, one based on recurrent neural networks and one based on template matching and statistical modelling, that result in stable mid-term

closing stock price predictions.

Bibliography

[1] Ayodele Adebiyi, Aderemi Adewumi, and Charles Ayo. Comparison of arima and artificial neural networks models for stock price prediction. *Journal of Applied Mathematics*, 2014:1–7, 03 2014.

[2] Khaled Al-Thelaya, El-Sayed El-Alfy, and Salahadin Mohammed. Stock market forecast using multivariate analysis with bidirectional and stacked (lstm, gru). pages 1–7, 04 2018.

[3] Kyunghyun Cho, Bart Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. 06 2014.

[4] Preeti Patarwal, Rajni Bala, and Ram Singh. Financial and non-stationary time series forecasting using lstm recurrent neural network for short and long horizon. pages 1–7, 07 2019.