STOCK PRICE PREDICTION USING MACHINE LEARNING

A THESIS SUBMITTED TO
THE FACULTY OF ACHITECTURE AND ENGINEERING
OF
EPOKA UNIVERSITY

BY

QEMAL ALLIU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR BACHELOR DEGREE
IN COMPUTER ENGINEERING

OCTOBER 2020

APPROVAL SHEET OF THESIS

This is to declare that I have read this thesis entitled **"STOCK PRICE PREDICTION USING MACHINE LEARNING",** and that in my professional opinion, it is fully adequate, in scope and quality, as a proper thesis for the Bachelor Degree of Computer Engineering.

_____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: ……..

Signature:

ABSTRACT

STOCK PRICE PREDICTION USING MACHINE LEARNING

Qemal Alliu
Department of Computer Engineering
Supervisor: Dr. Igli Hakrama

The main purpose of this thesis is to design and implement a program that gathers information about the stock prices over a period of time.
This data is to be analyzed and the program trains itself to predict a possible price for the future. This prediction comes in handy to people who are interested in stock trading, either investors, or people who want to learn more about stock trading, or Artificial Intelligence.
The program is implemented in Python, and it's open-source, with a user-friendly GUI, so it can be used by people who don't know much about coding, or them who already know how to code and see how the model is trained, even improve it when possible.

Keywords: Computer Engineering, Artificial Intelligence, Stock Market, Machine Learning, Prediction

ABSTRAKT

PARASHIKIMI I ÇMIMIT TË BURSAVE
NËPËRMJET MACHINE LEARNING

Qemal Alliu
Departamenti i Inxhinierisë Kompjuterike
Supervizor: Dr. Igli Hakrama

Qëllimi kryesor i kësaj teze është dizenjimi dhe implementimi i nje programi që grumbullon informacione rreth çmimit të bursave në një periudhë të caktuar kohe.
Të dhënat e grumbulluara, më pas analizohen për të trajnuar një model, dhe nëpërmjet Machine Learning, bëhet i mundur një parashikim i përafërt në lidhje me vijueshmërine e çmimit. Ky parashikim mund të përdoret nga persona të cilët janë të interesuar në tregun e bursave, investitorëve, ose nga persona të cilët duan të mësojnë rreth këtij tregu, ose inteligjencës artificiale,
Ky program është i implementuar në Python, dhe është open-source, bashkë me një ndërfaqe grafike, gjë që e bën të mundur të përdoret nga persona që nuk kanë shumë informacion rreth kodimit, gjithashtu dhe nga persona që dinë të kodojnë por duan të mësojnë rreth trajnimit të modelit, ose të përmirësojne algoritmat e përdorur.

Fjalë kyç: Inxhinieri Kompjuterike, Inteligjencë Artificiale, Bursa, Machine Learning, Parashikim

Dedicated to my family, for their endless love and support

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Dr. Igli Hakrama, for the support provided and motivating me to do my best for this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
API	Application Programming Interface
CSV	Comma Separated Values
GUI	Graphical User Interface
LSTM	Long Short-Term Memory
ML	Machine Learning
PC	Personal Computer
RAM	Random Access Memory

CHAPTER 1

INTRODUCTION

In the recent years, with the rapid development of technology, the buying and selling of stocks have become very easy. Investing can be done with a click in the online trading platforms, but this investing is usually made with an investment strategy in mind.

Every investor tries to predict the future price of the shares they own, or want to own. The prices of these shares are set based on the supply and demand, and every investor wants to buy for the lowest price possible, and sell for the highest one. This means that a successful prediction on the future of a stock price, can generate a good profit for the traders. When we look at the graph, we notice that there is a certain pattern on the increase and decrease of the price. After every price increase, the owners of the stocks start selling them, so the supply increases, and the price decreases. We notice the same thing with the low prices: more people will try to buy these stocks in a lower price, which means more demand, and a higher price. After doing some research about stock market and AI, I decided to design and implement a machine learning program, that trains itself with the daily prices of a stock, for a long period of time, and coming up with its own prediction on how the stock price will continue. To test if the projection is correct or not, I trained the algorithm with 95% of the previous data, and compare it to the last few days (other 5%). The design and implementation of this program is done in Python coding-language, which can also be used for software or web development. Even though it is not as good as a real programming language, it is quite reliable, since it has many libraries that we can conveniently use for this program, such as: Tensorflow and Keras for Machine Learning, Numpy and Pandas for data manipulation and processing, matplotlib for graph plotting, and Tkinter for GUI design.

CHAPTER *2*

LITERATURE REVIEW

2.1. Stock Market

2.1.1 What is a stock?

A stock is a share or equity in a company, that represents ownership in a company or in a corporation and it shows a proportionate claim on the assets and earnings.

Stock ownership means that the shareholder owns a part of the company equal to the number of shares held as a proportion of the company's total outstanding shares [1]. For instance, an individual or institution that owns 300.000 shares of a company that has one million leading shares would have around 30% ownership stake in it. Different companies have different volumes of shares.

2.1.2 How are stock prices set?

The price of one share in the stock market is set like the price of everything in the economy. It simply depends on the supply and demand of those shares.

For example, if too many shareholders want to sell their shares, and there are not many investors who want to buy those shares, they will lower the price, so it becomes more likely for them to start buying.

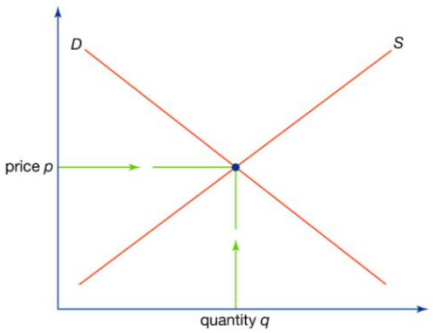


Figure 1 Supply-Demand Graph

Just like this, if there are too many investors who are trying to buy shares in a company, but not enough supply, they will increase the price, so they can earn as much as possible from this sale.

2.1.3 Investing in stocks and profit

With the development of technology, nowadays it is very easy for us to invest in a stock that we find promising. The number of shares that people buy in a company is very small, and totally insignificant on the company's value, so the only way of making profit off them is to sell them back [1]. A successful prediction in the future of a stock price can mean a significant profit for us.

2.2. Machine Learning

2.2.1 What is Machine Learning?

Using algorithms and mathematical techniques to perform a task and improve over time. If a program improves its performance over time with no change to the original algorithm, it is said to have learned [2].

Machine learning programs improve over time by changing values stored in nodes within a graph.

2.2.2 Types of Machine Learning

1. Supervised learning

The model is trained to get as close as possible to a set of values, by passing in inputs and the corresponding outputs [5].

2. Unsupervised learning

The model is given a set of values, without an output, and it tries to recognize patterns or clustering in the given data [5].

3. Reinforcement learning

The program is “rewarded” for achieving a positive result and “punished” for a negative result and the goal is to get a more positive result or a higher score [5].

2.2.3 Steps of building a Machine Learning program

1. Acknowledge what problem you are trying to solve and strategize about how to solve it [10].
2. Gather and format data - use online data banks or collect data yourself and make sure it is all the same format
3. Construct computational graph
4. Train and test the model - tweak and improve as necessary

2.2.4 Computational Graph

A Computational Graph in ML consists of a series of nodes, which are organized into connected layers. Each node has operations, values, and activation functions that dictate the strength of signal to pass onto the next node. The purpose of a Computational Graph is to take data as input, and transform it into output. During training, values change over time to alter the strength of output signal which ultimately affects the final output [7].

2.2.5 Training

Training a model is done by feeding it some input with the correct output, and having the model try to get as close as possible to that output. The difference between the model's output and the actual output is called loss, and we try to minimize it by having the model go through some iterative process, where it continuously alter's its values.

The closer the model's output is to the correct output, the more accurate it becomes.

2.3. Tensorflow

Tensorflow is an open-source library that provides tools for mathematical and statistical analysis of data [9]. It is particularly useful for building machine learning models. Tensorflow contains various types to help us build computational graphs. Provides training, testing, and evaluating functionality.

2.3.1 How does it work?

Tensorflow gathers and formats data, to build a computational graph, that consists of layers of connected nodes and operations to transform the data from input to output. Then, it trains the model based on the data set by using an optimizer and loss function. Finally, it evaluates and tests the model, by measuring accuracy and loss [9].

2.3.2 Components

- Nodes - carry values or operations
- Activation functions - attached to nodes, dictate strength of signal to pass on
- Loss functions - measure total current loss of the graph during training or testing
- Optimizer functions - change the values of the nodes to minimize loss
- Train and test steps - used to actually train and test the model

2.3.3 Pros of using Tensorflow

- It contains all the classes, variables, functions, etc. which are necessary to build machine learning models
- It is much easier to use Tensorflow than a program in the algorithms (loss functions, activation functions, etc.)
- Easy to build and connect layers of nodes which are highly customizable
- Fine control over graph structure allows for maximum efficiency and potentially more powerful models

2.3.4 Cons of using Tensorflow

- Can still be difficult to use if you have no prior machine learning experience
- Need to build everything from scratch so model construction can be a lengthy process
- Not quite as fast as some other machine learning libraries

2.3.5 Keras

Keras is a library, also included in Tensorflow, which is used to build neural networks for machine learning models. It is specially designed for deep neural networks.

It contains abstract classes used to build models, which makes it really useful for Tensorflow [4].

2.3.6 How does Keras work?

This library alone needs a back-end framework for the ML computations, like Tensorflow.

We add layers to a model, one at a time, using layer objects. Sequential models connect a layer to the next one, based on the order in which we added those layers. Functional models are more flexible and allow us more control over how to connect the layers [4].

2.3.7 Why use Keras?

- Model and layer objects make it much easier to build machine learning models
- No longer have to build up our layers by creating and connecting variable and operation nodes
- Specify loss and optimizer functions with the “compile” function
- Use the “fit” function to train the model and the “predict” function to use the trained model

2.3.8 Loss Functions

Keras has some built-in algorithms that seek to minimize a value, that we call loss, to generate a more correct prediction. Depending on the type of prediction, there are many loss functions available. The most efficient ones for regression are MeanSquaredError, and MeanSquaredLogarithmicError.

MeanSquaredError calculates the square of the difference between predicted output and real output.

MeanSquaredLogarithmicError calculates the square of the difference between the logarithmic error between predicted output and real output.

CHAPTER 3
SOFTWARE ANALYSIS AND DESIGN

3.1 Functional Requirements

3.1.1 Input Requirements

The program includes a very user-friendly GUI, which guides the users on how to use the application.

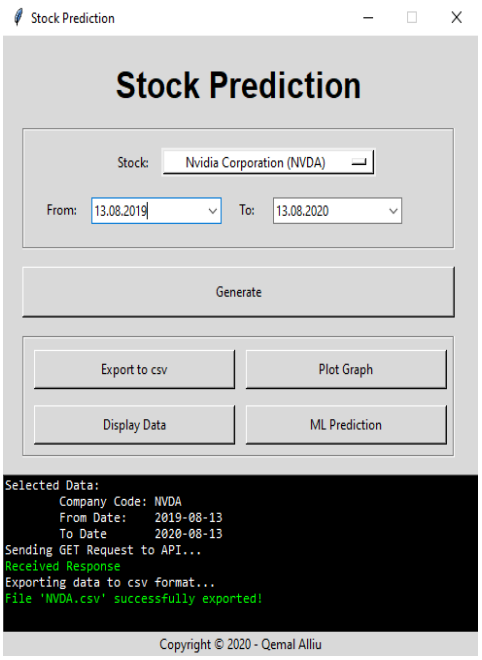


Figure 11 Application View

The user is required to insert 3 inputs:

1. Stock Symbol : : Chosen from a dropdown menu, in which some sample stock symbols are given for demonstration, or the user can choose “Other..” and give the input by themselves in a popup dialog box, after they press “Generate”

Figure 2 Stock Symbol Dropdown Menu

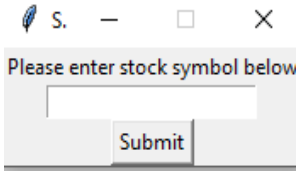


Figure 3. Stock Symbol Input Dialog Box

2. Start Date : Input of type date, taken from a date picker

3. End Date: Also of type date

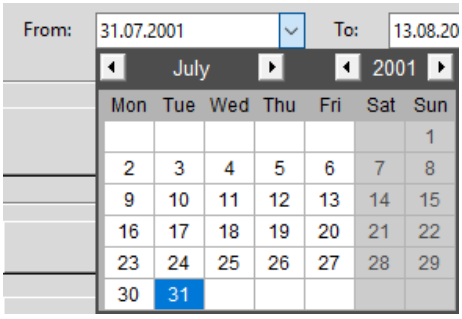


Figure 4 Date Picker

Other inputs that the application requires, is the daily stock data, which are collected from Tiingo's free API, by using the other inputs as request parameters.

3.1.2 Output requirements

The application should be able to process the data, and store it into a .csv file.

It should be able to generate a graph, use the data for other functionalities, such as creating datasets, training a model and creating predictions.

3.2 Non-functional requirements

3.2.1 Hardware Requirements

To run this application, we just need a computer, that can run Python 3, and process a significant amount of data. A modest PC, with 2 GB of RAM, and a dual-core 2.0 GHz processor, 2 GB disk space, should do just fine, but the data analysis process may take a while.

3.2.3 Software Requirements

To develop this application, we need :

- Tensorflow
- A version of Python that supports Tensorflow (Python 3.5 – 3.8) [9]
- pip 19.0 or later
- A supported operating system for Tensorflow (Ubuntu 16.04 or later, Windows 7 or later [9])
- A text editor (e.g: notepad++, vscode, vim) or an IDE (e.g PyCharm)

3.3 UML Diagrams

3.3.1 Class Diagrams

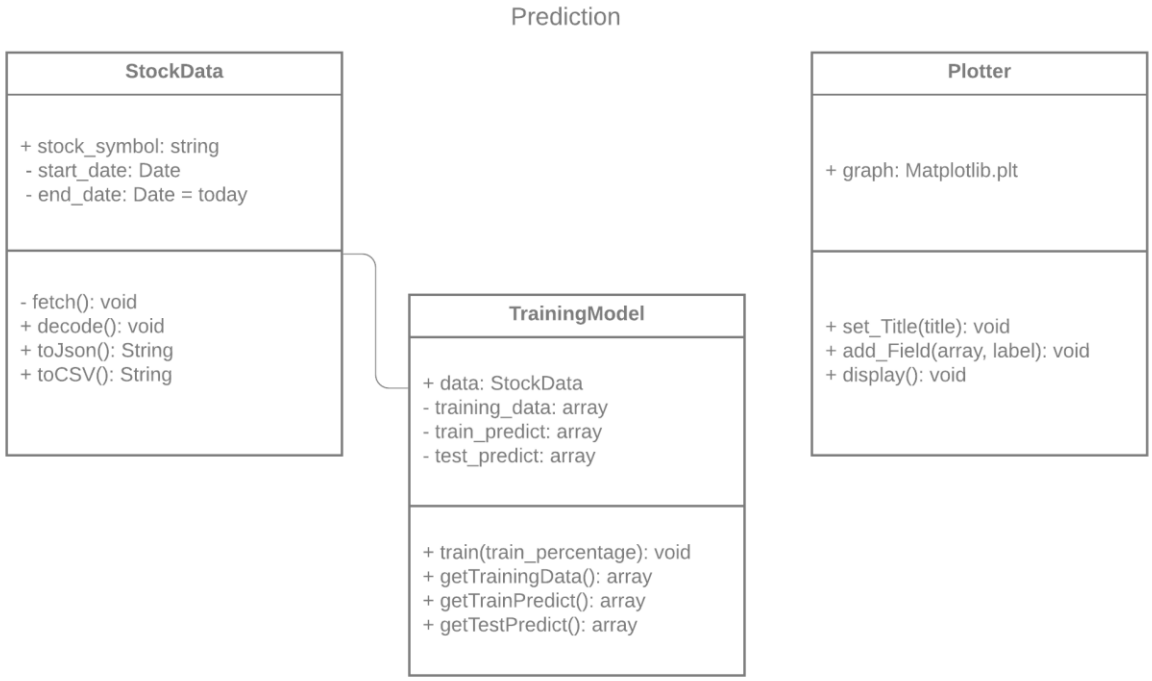


Figure 5. Class Diagram For Prediction Module

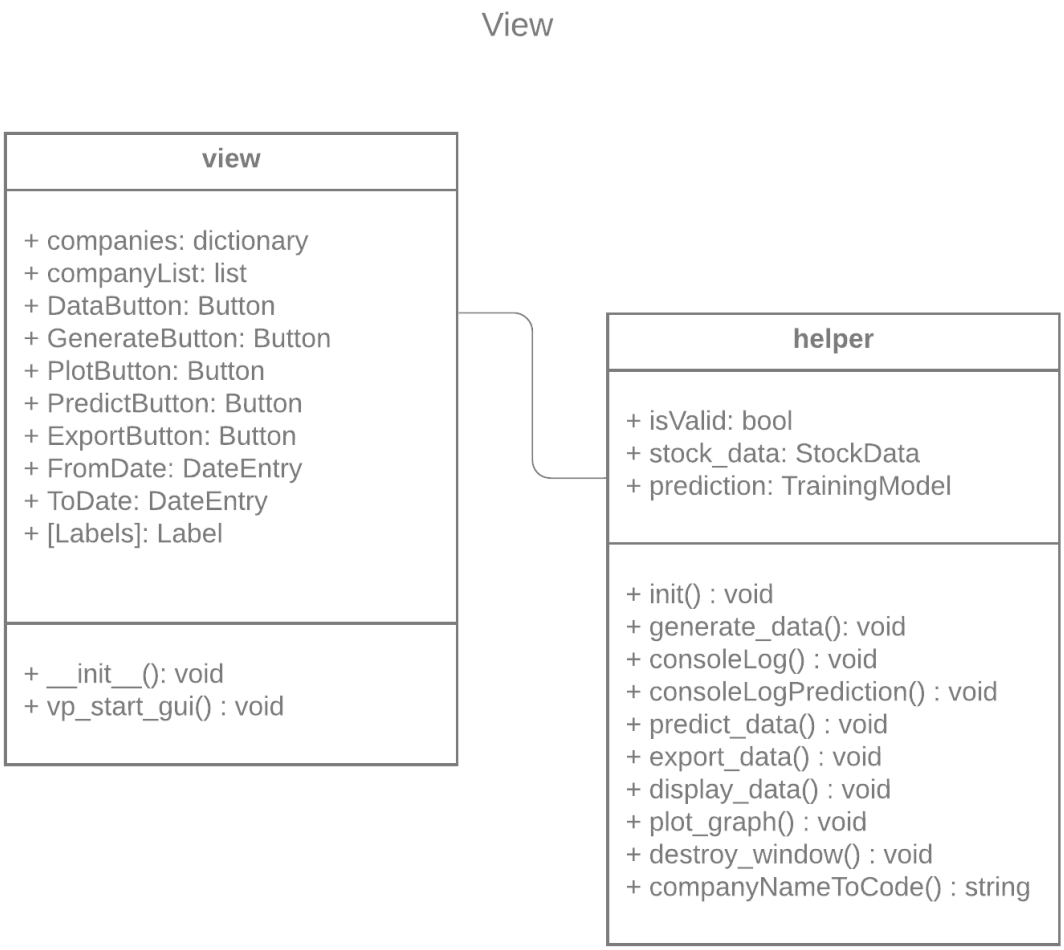


Figure 6. Class Diagram for View Module

3.3.2 Sequence diagram

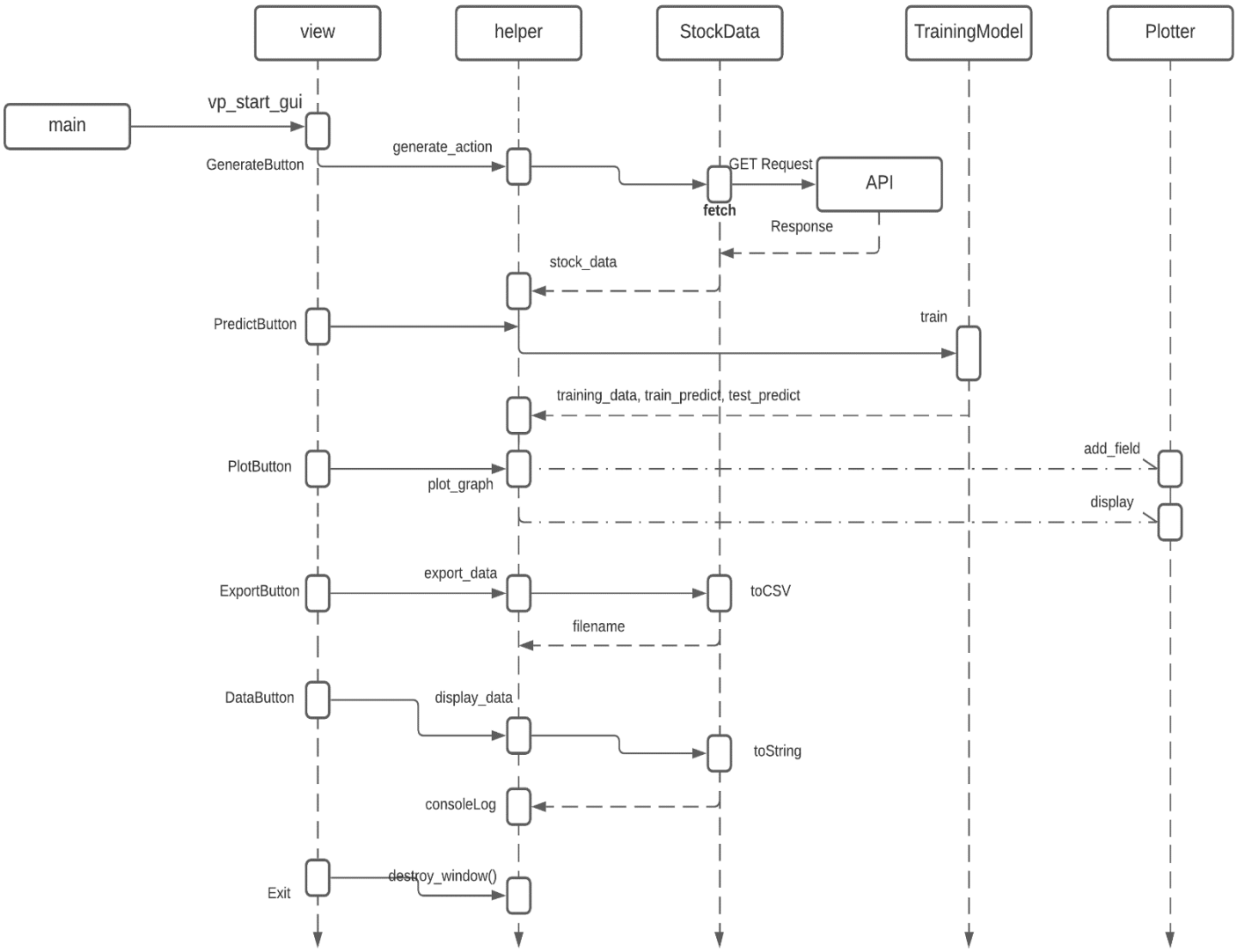


Figure 7 Sequence Diagram

CHAPTER 4
IMPLEMENTATION

4.1 Data Collection

4.1.1 Stocks API

To collect the data, I am going to use Tiingo's API. Tiingo [*www.tiingo.com*] is a highly popular, free API that allows us to query historical data [11]. It is easy to structure a query and read incoming data.

Tiingo requires us to be a user of the site, which is free, and we need an account.

4.1.2 Using an API

Each API uses a HTTP method. To retrieve data from the API we need to send a GET request to that API, and then process the response.

To send the GET request, we need to construct an URL, made of three parts:

- Base URL
- Query
- Parameters

We will pass a start date and end date as parameters, because we will search for daily historical data of a particular stock.

Tiingo also needs a token as a parameter, which can be easily found, once we have an account.

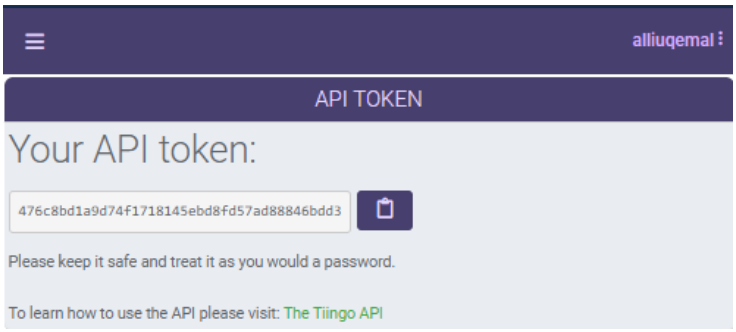


Figure 8 Tiingo API Setup

4.1.3 Retrieving Data

To retrieve the data, I am creating a function, which uses the requests package from Python.

```
# Fetch requested data from the API
def __fetch(self):
    if self.__data is None:
        base_url = 'https://api.tiingo.com/tiingo/daily/' +
self.stock_symbol + '/prices?'
        token = '476c8bd1a9d74f1718145ebd8fd57ad88846bdd3'
        payload = {
            'token': token,
            'startDate': self.__start_date,
            'endDate': self.__end_date
        }
        response = requests.get(base_url, params = payload)
        self.__data = response
```

4.1.4 Parsing the data

The data we are going to parse is in JSON format. We simply treat the JSON as a dictionary where keys are strings and values are JSON objects, arrays, or values.

For this, I also created a function that stores the highest and lowest values of each day.

```
def decode(self):
    self.highs = []
    self.lows = []
    for object in self.toJson():
        self.highs.append(int(object['high']))
        self.lows.append(int(object['low']))
```

4.2 Plotting the Graph

To plot the graph, I am creating a model that uses an instance of matplotlib.pyplot. Matplotlib is a python library, that produces 2D graph images. These images are pretty high quality, and can be found in a variety of hard-copy formats, and in an interactive environment.

Using pyplot's functions, we can use matplotlib just like MATLAB. There is a specific function to make any change we want, create a figure, a plotting area in a figure, decorating the figure with legends, and labels.

```
import matplotlib.pyplot

class Plotter:
    def __init__(self):
        self.graph = matplotlib.pyplot

    def set_title(self, title):
        self.graph.title(title)

    def add_field(self, data, label=None):
        self.graph.plot(data, label=label)

    def display(self):
        self.graph.show()
```

4.3 Data Exporting

To export the data, I am using pandas, which is a library used for data manipulation and analysis. It provides data structures and operations for manipulating numerical tables, and time series.

Before exporting, we need to transform the data to pandas' DataFrame, that has the exporting options, one of which is the to_csv() method I'm using.

```
# Generate a CSV with all the data
def toCSV(self, filename=None):
    if filename is None:
        filename = self.stock_symbol + "-" + str(date.today()) +
".csv"
    table = pandas.json_normalize(self.toJson())[['date', 'high',
'low', 'adjClose', 'volume']]
    table.to_csv("files/" + filename, index=False)
    return filename
```

4.4 Prediction

4.4.1 LSTM Cells

LSTM (long short-term memory) cells are used in RNNs, they are very useful for series and sequences of data, as they can "remember" previous values.

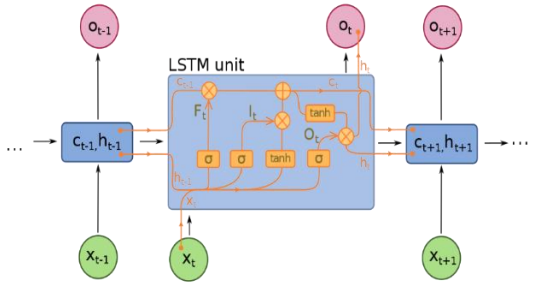


Figure 9 LSTM Graph

4.4.2 RNNs

RNNs (Recurrent Neural Networks) are looped networks that allow information or state to persist between runs. They do not operate on a fixed number of layers. It cycles the input multiple times, combining the state with a new input every cycle [3].

Its most known use is speech recognition, text generation, and image captioning.

4.4.3 Why are we using LSTM cells?

LSTM cells are very good with sequence data, like time series. It fixes the problem of vanishing gradient, where small values do not change, and AI training fails.

Since stock prices change over time, we are considering this as time series, and LSTM is our best option.

4.4.4 Creating the Dataset

```
def createDataset(data, timestep=1):
    data_x, data_y = [], []
    for i in range(len(data) - timestep - 1):
        data_x.append(data[i:(i + timestep), 0])
        data_y.append(data[i + timestep, 0])
    return numpy.array(data_x), numpy.array(data_y)
```

4.4.5 Building the Sequential Model

For this we are going to create a Keras sequential model, and add Dense and LSTM layers.

```
model = Sequential()
model.add(LSTM(256, input_shape=(1, 1)))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='mean_squared_error', optimizer='adam', metrics=[])
```

We also need to compile loss and optimizer functions. There are many loss functions in Sequential, but we need a regression loss function [6], and I found Mean Squared Error to be more effective.

Mean Squared Error formula:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

Same thing for the optimizer. I am using *adam* algorithm, since it is more effective for this kind of training [8]. We can also specify metrics for the output, to help us determine the accuracy of the prediction.

4.4.6 Training and Testing the Model

To train the model, we can simply use the fit method of Sequential class, and put all the datasets as arguments [4]. I'm also adding the number of epochs and batch_size, which is 1, because for our RNN, we only need to pass through the data once, and also make the fit verbose, so we can see the progress of the training.

```
model.fit(train_x,
          train_y,
          epochs=1,
      batch_size=1,
          verbose=1)
```

4.4.7 Model Outputs

We use the predict() function from the model, to generate the output.

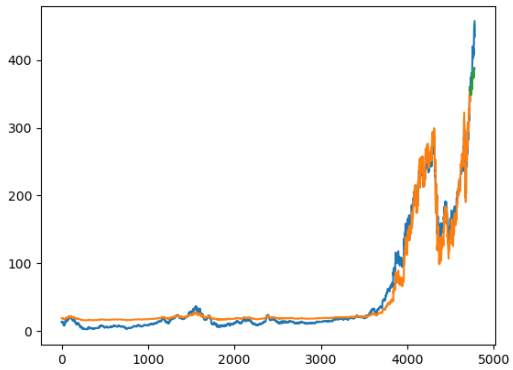
We are using this function for both train and test arrays.

```
train_predictions = model.predict(train_x)
test_predictions = model.predict(test_x)
```

Then, we plot them all into a graph to see the result.

4.4.8 Result

The result of this function, will be a pyplot graph with 3 lines in it:



- Blue : the original stock price
- Orange: the trained model, which is trained using 95% of the data.
- Green : the test prediction, to see if it matches the actual price, on the last 5% of the graph.

Figure 10 Result Graph

4.4.9 Analyzing the Result

In machine learning, it can be quite hard to analyze the results of a prediction. Keras has some built-in metrics algorithms to find how correct the prediction is.

- *Accuracy*

This algorithm is the most used metrics algorithm on Keras and Tensorflow. It simply calculates how often the prediction is equal to the real output. We are not calculating exact values, so, it's not useful in our case, because the price changes so unexpectedly, and our prediction rarely meets the real output at one point.

Since we are studying the pattern, and not calculating exact values. We need to look for probabilistic or regression metrics.

- *LogCoshError*

This algorithm is designed for regressions. It computes the logarithm of the hyperbolic cosine (cosh) of the prediction error [4].
The formula for this algorithm is:

$$L(y, y^p) = \sum_{i=1}^{n} \log(\cosh(y_i^p - y_i))$$

- *BinaryCrossentropy*

This algorithm is used for probabilistic predictions, and it calculates the crossentropy metric between the prediction and the real output [4]. Theoretically, Crossentropy is a value, calculated by measuring the difference between the probability distributions of two events.
The formula for this algorithm is:

$$H_p(q) = -\frac{1}{N} \sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Practically, it seems to have a better result than other metrics algorithms.

4.4.10 Alternative Solutions

Time Series Analysis is definitely one of the most challenging studies in Data Science. It consists of recording the data at a regular interval of time and it can be used to investigate seasonal behaviours, trends, changes etc. Some other solution for time series prediction are:

- *ARIMA* (Autoregressive integrated moving average)

ARIMA is a combination of three models [13].

Autoregression – A model that uses the relationship between an observation and a number of lagged observations.

Integrated – A model that replaces the data with the difference between their value and the previous values.

Moving Average – A model that uses the linear combination of error terms values as a regression error.

- *Vector Autoregressive Model or VAR Model*

Vector Autoregression is a model that is used to study the relationship between multiple quantities as they change over time. VAR is a type of stochastic process model [12]. VAR models generalize the single-variable (univariate) AR model by allowing for multivariate time series.

CHAPTER 5

CONCLUSION

In this thesis was proposed a design and implementation of a simple system that comes in handy to people who are interested in Machine Learning, or stock trading.

It explains how to create a ML model using certain techniques such as LSTM and Regression.

They showed quite accurate predictions and positive results, but we cannot say that these machine learning predictions are to be trusted in real stock trading.

One improvement that I think of implementing in the future is by training the model with the datasets from many companies, which will have more software and hardware requirements, like parallelizing the data analyzing algorithms, and using GPU instead of CPU devices, for better multithreading.

It can still be improved, if we study the other factors that affect the stock price of the company, from the news, and maybe train the artificial intelligence to read all the news, and study the impact on the stock price, and determine a possible future price based on the current news.

However, the market has always been unpredictable. Even if such algorithms are implemented, I believe the market will most likely find a way to change the way it works, because such accurate predictions will have a big impact on the supply and demand, which will mess more with the price.

REFERENCES

[1] - Investopedia's article on companies' stock price

https://www.investopedia.com/ask/answers/how-companys-stock-price-and-market-cap-determined/

[2] - Expertsystem's post on definitions of Machine Learning

https://expertsystem.com/machine-learning-definition/

[3] - A. Graves, S. Fernandez, F. Gomez, J. Schmidhuber. Connectionist Temporal

Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks. Proceedings of the International Conference on Machine Learning (ICML-06, Pittsburgh), 2006.

[4] - Keras API Reference, from the official documentation

https://keras.io/api/

[5] - Types of Machine Learning, Article by Hunter Heidenreich: towardsdatascience.com

https://towardsdatascience.com/what-are-the-types-of-machine-learning-e2b9e5d1756f

[6] - Machine learning: an introduction to mean squared error and regression lines; Post by freecodecamp: https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-c7dde9a26b93/

[7] - Deep Neural Networks As Computational Graphs: Post by Tyler Elliot Bettilyon

https://medium.com/tebs-lab/deep-neural-networks-as-computational-graphs-867fcaa56c9

[8] - Gentle Introduction to the Adam Optimization Algorithm for Deep Learning
https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

[9] - Tensorflow official Documentation
www.tensorflow.org

[10] - General Approaches – Machine Learning Process. Article by kdnuggets.com https://www.kdnuggets.com/2018/05/general-approaches-machine-learning-process.html

[11] - Tiingo API Documentation

https://api.tiingo.com/documentation/general/connecting

[12] - An article about VAR Model by Science Direct

[13] - https://www.sciencedirect.com/topics/economics-econometrics-and-finance/var-model

[14] - Wikipedia : Autoregressive integrated moving average

https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average

(All websites and links are lastly visited on October 4, 2020)

APPENDIX A

APPLICATION USE

1. Processing and Exporting the Data

Step 1: We insert the inputs, and press Generate

Step 2: We press “Export to csv”

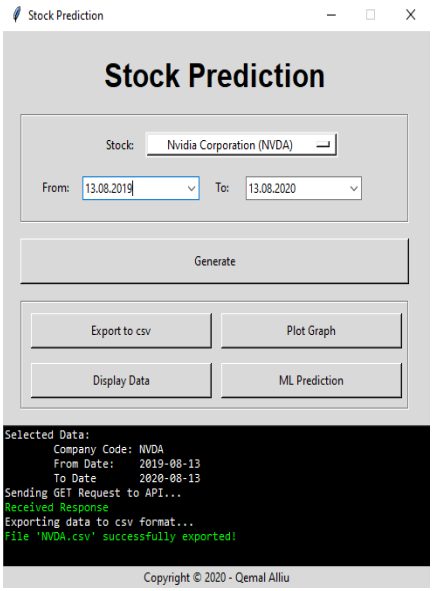


Figure 11 Application View

After following the steps, we can see from the log that the data is exported, and we go look for the file, with the name ‘NVDA-(*date*).csv’.

The csv file contains the following data:

date	high	low	adjClose	volume
2019-08-13...	157.9	149.8201	155.6114614248	9012811
2019-08-14...	152.95	148.3111	149.6482666838	10497033
2019-08-15...	151.63	147.394	148.351920001	13861551
2019-08-16...	161.54	156.88	159.1115974683	25384628
2019-08-19...	171.42	163.47	170.3000665307	20408061
2019-08-20...	170.58	167.02	167.3982443408	11588040
2019-08-21...	173.45	169.66	170.7488019209	10697773
2019-08-22...	173.33	169.9045	170.9980993599	7593918
2019-08-23...	170.59	161.64	161.9835039656	14237433
2019-08-26...	166.58	163.91	164.9850451312	7964854
2019-08-27...	167.1	160.62	161.3453025218	7288739
2019-08-28...	163.34	159.0	160.9264828243	6376719
2019-08-29...	168.3	164.76	166.6959597547	8959326
2019-08-30...	170.86	166.78	167.2050312486	7272712
2019-09-03...	165.91	163.252	163.8711120535	7412770
2019-09-04...	169.06	166.61	168.4527554982	5703009
2019-09-05...	179.99	172.8	179.4127653072	17412125
2019-09-06...	181.09	177.12	178.3247497615	9424419
2019-09-09...	183.98	179.84	180.171381651	10438133
2019-09-10...	184.25	178.79	182.8465024423	8820305
2019-09-11...	186.27	182.6006	183.994408752	9037782

Figure 12 CSV View

2 Plotting the Graph

When we press “Plot Graph” button, the program will show us the graph of the stock highest and lowest price for each day, using PyPlot from the matplotlib library.

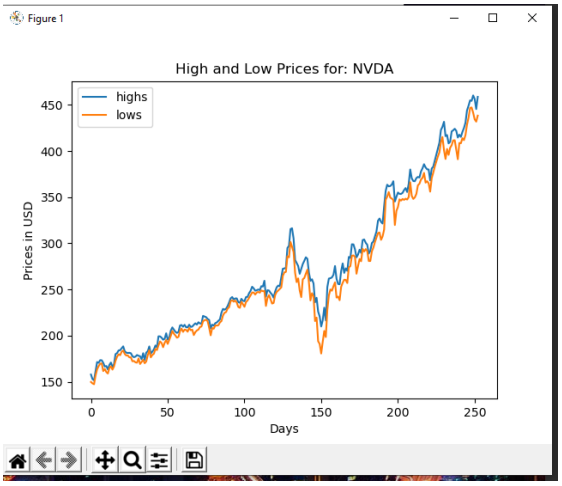


Figure 13 Result Graph

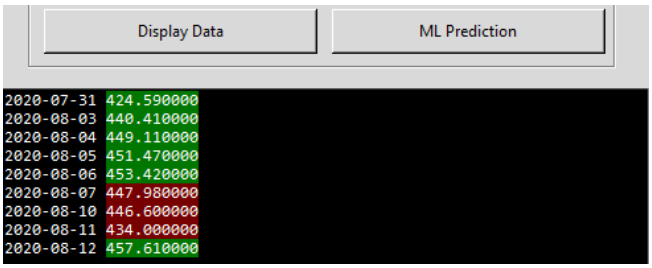
It also allows the user to zoom in and out in the graph, export the graph, etc.

To make the program run faster, I avoided adding the date in the x-axis, and only show the number of the day, where day 0 is the start date and the last date is the end date.

3 Data Display

The user may need to see the collected data, and not export them to a separate file.

For this I added the “Display Data” button, that prints all the data into the log area.



When the stock price is decreasing, it is highlighted in red, and when it’s increasing, in green.

Figure 14 Data Display

4 Prediction

This button displays a new graph with the progress of training of the Artificial Intelligence.

It only works best when we take a period as long as possible.

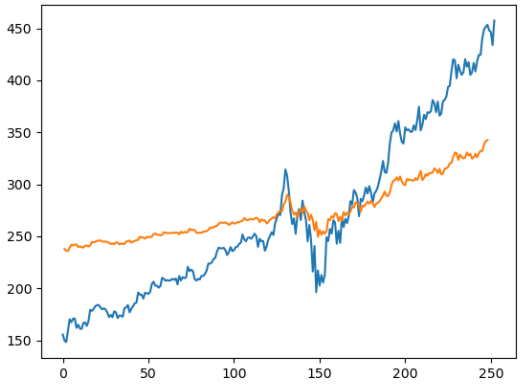


Figure 15 Training Graph

The orange line shows the training model, and the blue line shows the actual stock close price.

As we can see, in a short amount of time, it tries to imitate the actual stock price, but it is still very far from the actual price.

In a second test, I am inserting all the available data of this company, as an input.

This takes a couple of seconds to run, since I didn’t implement any parallelizing methods.

Now we can see that it comes quite close to the actual price, and if we zoom it in the last 5% of the days, we can also see the green line, which is the test model, based on the one trained from the previous 95% of the data.

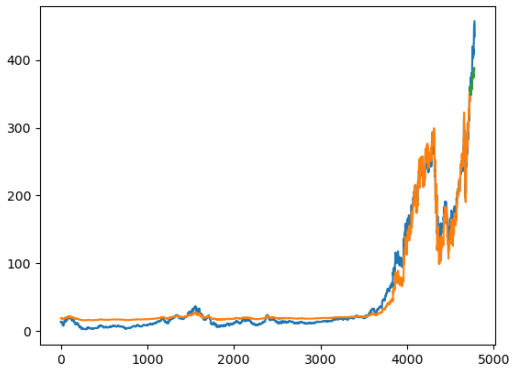


Figure 16 - Prediction Graph

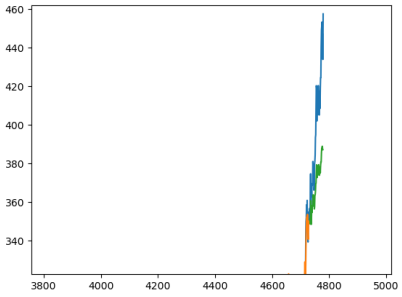


Figure 17 - Train Prediction Graph

What we notice, is that the test model predicts the increase of the stock price, but it is still wrong by $80, which is a lot, but as we know there are many factors that the model does not consider.