

THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

SCHOOL OF ENGINEERING

The Viability of Machine Learning in the Stock Market

NOLAN STEVENSON
SPRING 2023

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Computer Science
with honors in Computer Science

Reviewed and approved* by the following:

Dr. Jonathan Liaw
Professor of Computer Science
Thesis Supervisor

Dr. Wen-li Wang
Professor of Computer Science
Honors Adviser

* Electronic approvals are on file.

ABSTRACT

The purpose of this thesis is to explore the viability of machine learning in the stock market. The stock market is an inherently chaotic system, as complicated and intricate as it is vast. Human stockbrokers tend to perform sufficiently after years of training and education. Given the advancements made in machine learning in recent years, it seems reasonable to test a model in one of the most extreme environments possible. Proficiency in the stock market would demonstrate the endless possibilities of machine learning. While this thesis will not shake the field of economics to its core, it can be one that can be built and expanded upon in future studies.

For this project, a sequential neural network has been constructed. It is trained on a stock's price and numerous other technical analysis indicators like the Exponential Moving Average and the Stochastic Relative Strength Index. All of the data is divided based on its interval, leaving four intervals of data to use: one-minute, five-minute, fifteen-minute, and one-day intervals. There are four labels that this model can choose from when predicting. It will predict whether the price of a stock is a peak, uptrend, downtrend, or valley. Effectively, the model is predicting the best times to buy and sell a certain stock. The accuracy scores of this model's predictions vary based on the time interval. From an average accuracy of around forty percent with a one-day time interval to an overfit high of one hundred percent with a five-minute or one-minute interval. Additionally, the model's actual performance on the stock market was simulated using paper trading simulation on historical stock data. The model produced gains on sixty-two and a half percent of its paper trading simulations. Further studies can build upon this model's foundation and attempt to increase its accuracy and gain potential.

TABLE OF CONTENTS

LIST OF FIGURES	iii
LIST OF TABLES	iv
ACKNOWLEDGEMENTS	v
Chapter 1 Introduction	1
Chapter 2 Literature Review	2
Chapter 3 Stock Data Collection.....	4
3.1 Data Requirements.....	4
3.2 Data Sources	6
3.3 Data Labeling.....	7
Chapter 4 Peak and Valley Detection and Labeling	9
4.1 The NumPy Indicator	9
4.2 Zig-Zag Indicator.....	12
4.3 Relative Extrema Indicator	16
4.4 Indicator Selection and Rationale	18
Chapter 5 Building the Machine Learning Model	19
5.1 Additional Parameters and Normalization.....	19
5.2 Sequential Neural Network.....	21
5.3 Neural Network Layers.....	23
Chapter 6 Improving the Machine Learning Model	25
6.1 Initial Accuracy	25
6.2 Additional Validation Metrics	26
6.3 Current Accuracy	29
6.4 Simulating Paper Trading with the Current Model	32
Chapter 7 Conclusion.....	35
7.1 Takeaways	35
7.2 Future Research Potential	38
BIBLIOGRAPHY.....	39

LIST OF FIGURES

Figure 1. Midas GUI (Graphical User Interface).....	9
Figure 2. NumPy Detector Code.....	10
Figure 3. NumPy Detector Sample	12
Figure 4. ZigZag Detector Sample.....	13
Figure 5. Trend Detection Code.....	14
Figure 6. Full ZigZag Detector Sample	16
Figure 7. Relative Extrema Detector Code	17
Figure 8. Relative Extrema Detector Sample	18
Figure 9. Normalization Equations	21
Figure 10. Sequential Neural Network Code.....	23
Figure 11. Validation Metrics Formulas.....	29
Figure 12. Initial Model's 1-Minute Confusion Matrix.....	29
Figure 13. 15-Minute Validation Metrics	31
Figure 14. Current Model's 15-Minute Confusion Matrix.....	31
Figure 15. Current Model's 1-Day Validation Metrics	32
Figure 16. Current Model's 1-Day Confusion Matrix.....	32
Figure 17. Training and Testing Dataset Split.....	37

LIST OF TABLES

Table 1. Sequential Neural Network Layers.....	24
Table 2. Average Investment Portfolios After Paper Trading	35

ACKNOWLEDGEMENTS

This endeavor would not have been possible without the guidance and patience of my thesis supervisor Dr. Jonathan Liaw. Dr. Liaw's knowledge and expertise was invaluable for this project. Dr. Wen-li Wang has been a fantastic honors supervisor and academic advisor and helped me navigate through college. Without Dr. Liaw and Dr. Wang, this thesis would not have been possible.

Special thanks are in order for Dr. Michael Brown. Dr. Brown always offered support and advice whenever I needed it. There is also Mrs. Gummerson, who supports any scholar that is in need, and Dr. Carney. Words cannot express how thankful I am for the thesis writing sessions Dr. Carney hosted and the efforts of the entire honors administration at Behrend. Additionally, none of this would have happened without the involvement, structure, and guidance of the Schreyer Honors College.

Lastly, I would like to mention my wonderful family. Without their support, I would not be in college to begin with. Their kind words and belief in me encouraged me throughout the entire thesis process. Of course, my cat Bruno also provided much needed emotional support.

Chapter 1

Introduction

Machine learning is the process by which computers mimic the learning behaviors exhibited by humans. Namely, the ability to analyze the patterns within data and apply that pattern to infer attributes about other data. For this research, the domain of machine learning is crossed with the capital market domain, specifically in relation to stocks. There are commonly known terms and ideas, like the peaks and valleys of a stock price. These are the points where the price of a stock is at a high value and where it is at a low value, respectively. Additionally, this research utilizes the practice of technical analysis, which is the evaluation of stock patterns to determine the optimal time to buy or sell shares of a stock. Throughout this thesis, terms will be explained as they are introduced in the relevant chapters. The intersection of machine learning and capital market has certainly been studied before and the literature review in Chapter 2 will further explore other studies that have been conducted and relate to this one.

This research serves as the foundation of any future studies that would wish to reference it. For any field of research, there first needs to be foundational studies to show that continued and increasingly specific research in that field is warranted. For this project, the goal is to identify the viability of machine learning in the stock market. Specifically, whether a machine learning model can identify the peaks, valleys, and general trend of a stock's price. Analyzing a model's performance can help demonstrate the potential and useability of machine learning within the stock market. If the machine learning model is given sufficient high-quality data, by harvesting data from reliable sources and further processing it through technical analysis, it will

produce acceptable results. The aim is to achieve a sixty to seventy percent accuracy by the model when predicting what times are best to buy or sell a stock. The model will also be given various simulated stock portfolios to manage. If the model can produce gains on at least fifty-five percent of these simulated stock portfolios, it would also constitute a success. The remaining chapters of this thesis will outline the process of collecting the needed data, labeling, and expanding the data, building the machine learning model, improving the machine learning model, and the takeaways of this project.

Chapter 2

Literature Review

The intersection of machine learning and the stock market is by no means unique to this study. There are studies, as discussed below, that date back to the 1990s showing similar experimentation. However, the field of machine learning has changed quite a bit in just the past decade alone. It would be fruitful to continue to examine the viability of machine learning in the stock market as newer and more advanced models and practices are developed.

When it comes to most of the popular studies that tackle machine learning in the stock market, they tend to focus on stock price prediction. In the “A Machine Learning Model for Stock Market Prediction” paper published in the International Journal of Computer Science and Telecommunications in 2013. The model attempts to “determine the future value of a company stock or other financial instrument traded on a financial exchange” (Hegazy et al.). Our study deviates from this one and many others like it. Our goal is not to predict the price of a stock, but rather to predict when the best moments are to buy and sell a certain stock. While the desired

results are similar, the methodologies are rather different. A comparison between the results of two studies that use these differing methodologies could be quite interesting.

Another noteworthy paper was published in the 1990 IJCNN International Joint Conference on Neural Networks and authored by T. Kimoto, K. Asakawa, M. Yoda, and M. Takeoka. Like our own study, it was not focused on the price prediction of stocks, but rather the prediction of when to buy and sell stocks. Key differences arise from which stock market was used and the type of neural network used. This study utilized modular neural networks on the Tokyo Stock Exchange, while our study will focus on a sequential neural network being utilized on the New York Stock Exchange (Kimoto et al.). This project was also much bigger in scope than our own. Within modular neural network, “each independent neural network serves as a module and operates on separate inputs to accomplish some subtask of the task the network hopes to perform” (Azam). The results of our study could help to show the progression of machine learning since the 1990s. Perhaps the task that once required a network of interconnected neural networks can now be accomplished with a single neural network.

Of course, this study also needs to examine previous studies and articles that can illuminate certain aspects of the economics side of this project. Namely, sources were consulted to evaluate the potential of numerous technical analysis indicators. One paper examined the use of certain indicators like the Relative Strength Index (RSI) and Moving Average Convergence/Divergence (MACD) on the Spanish stock market (Rosillo et al.). In summary, it was found that the strength of an indicator greatly depends on the stock that it is being applied to.

One study that is extremely relevant to this work was published in the Proceedings of the International MultiConference of Engineers and Computer Scientists and authored by K. Kannan, P. Sekar, M. Sathik, and P. Arumugam. This paper focused on the use of data mining

and predictive technologies to analyze the hidden patterns within historical data and predict the future direction of the stock's price (Kannan et al.). Not only did the paper utilize the methods of “Typical Price (TP), Bollinger Bands, Relative Strength Index (RSI), [Candlestick Momentum Index] CMI and Moving Average (MA)” but it also “investigated various global events and their issues predicting on stock markets” (Kannan et al.).

Even though there have been similar studies before, there is always more room for experimentation and discovery. There is a paper by Gandhmal and Kumar, one of many others like it, that compares and reviews fifty research papers that cover a variety of techniques used for stock prediction. Papers like this help to find the problems and challenges within existing research and allow for their correction and rectification in future studies (Gandhmal and Kumar). Ideally, someone may someday read this paper and notice a problem that could be fixed to further improve it.

Our study is not the very first of its kind and will certainly not be the very last either. The main goal of this study is to build upon the ones that have come before it and provide an even greater foundation for future studies to build off.

Chapter 3

Stock Data Collection

3.1 Data Requirements

The overarching idea behind the field of machine learning is to use algorithms to mimic a sense of learning. The computer slowly and gradually improves upon itself and improves its

accuracy. To accomplish this goal, a machine learning model requires a vast amount of data. As a general guideline, it requires a number of data points equal to ten times the number of features you are using. That is, if your machine learning model examines 19 features on each data point, 190 data points are needed. Having a large amount of data allows the model to search for and analyze patterns within the parameters and use those patterns to predict information. There are numerous approaches to machine learning, and each requires slightly different data input. For this project, a supervised learning approach has been taken. Simply put, a supervised learning approach means that all data must be labeled. Since this model aims to predict when to buy and sell stocks on the stock market, every data point we gather must indicate whether a stock should be bought or sold.

Diving down into the deeper mechanics of machine learning is the use of training datasets and test datasets. When building a supervised learning machine learning model, the data that will be used must be broken into two sets. The first of these sets is the training set. Data within the training set is used for the construction and training of the model, as the name implies. Once the model has been fully constructed, the test set comes into play. The test set contains portions of the data sectioned off from the rest and not allowed for training. Instead, this data is used to evaluate the accuracy and performance of the machine learning model after it has been built and trained. To better illustrate this concept, imagine, if you will, a child that you are tutoring in basic mathematics. At first, you work with the child on the basic concepts, going through sample problems and giving them the answers as you go. That is the purpose of the training set, data from which the model can learn and improve. When you want to test their knowledge, you do not simply give them the same problems that you used to teach them the concepts. Instead, you create fresh problems that help to truly test their knowledge of the concepts. That is the purpose

of the test set, unfamiliar problems that can be used to test the model's accuracy and effectiveness.

3.2 Data Sources

Finding such a large quantity of data can be a daunting task, especially given the fact that high-quality data is necessary. Blank fields in a dataset or mislabeled entries can negatively affect a machine learning model. With these prevalent factors in mind, there are a plethora of online sources that can give high-quality dataset files for free. One such source is Kaggle.com, which boasts an extensive collection of datasets that range from data on TripAdvisor hotel reviews to information about rice production by each country from 1961 to 2021 (*Find Open Datasets and Machine Learning Projects*).

Once a titan of the internet during the 1990s, Yahoo has a repository of financial information available through Yahoo Finance. Yahoo Finance allows for the download of daily, weekly, or monthly historical stock data dating back years. In the case of Tesla, the historical data goes back to June 28th, 2010. For longer standing companies, the data can go back even further. Yahoo Finance claims to have IBM stock records dating back to June 1st, 1962. All this data can be easily downloaded as comma separated values, also known as a .csv file.

Of course, Yahoo Finance does not have any sort of monopoly on stock data and other sources have appeared throughout recent years. Another source of data is Alpaca, a website and API (Application Programming Interfaces) built for trading. One major strength of Alpaca is the interval of its data. While Yahoo Finance offered daily, weekly, and monthly data, Alpaca offers one-day, fifteen-minute, five-minute, and 1-minute intervals. Alpaca also has a certain

component that other sources lack: paper trading. Using the Alpaca API, you can create a program that participates in paper trading, which is a term used to describe a stock market simulator, where one can pretend to trade stocks using simulated currency. The Alpaca API provides the resources and functionality needed for a true test of the model's accuracy.

Integrating with the Alpaca API could allow our model to fully simulate the buying and selling of stocks using accurate, up-to-date information (*Developer-First API for Trading*). Success would evolve from being a percentage of correct predictions in the test set to tangible results, namely how much money the model made or lost while paper trading. Given the value of this functionality and the availability of high-quality data to boot, Alpaca was chosen as the data source for this project.

3.3 Data Labeling

The Alpaca datasets contain numerous helpful parameters for each data entry, but there is a key component that is missing. Since this project utilizes supervised learning, each data entry is required to have a label. Within this project's scope, each stock data entry would require a label denoting whether it should be bought or sold at that given time and price. There are no such high-quality datasets with this needed information that could be easily found. Instead, a method of automatic labeling was utilized. Using the price point of each data entry, an algorithm could be used to determine the appropriate label. Automatic labeling also comes with the added benefit of customization. Rather than having to use labels that came preloaded onto a dataset, specific labels could be generated that better suit this project. With this benefit in mind, four labels were created: peak, valley, upward trend, and downward trend. A peak represents the price point at

which a stock should be sold, a valley represents the price point at which a stock should be bought, and the last two labels denote an upward trend and a downward trend in the stock's price, respectively.

Once the labels were defined, the next step was to implement an algorithm that would assign a label to each data point in the dataset. There are an absurd number of peak detection algorithms available to choose from, too many in fact. Manually comparing each algorithm would be a long and laborious process on par with just manually labeling the data. Ideally, a tool or program could be used to visualize each algorithm's performance. If the label for each data point was plotted onto a graph, a human eye could easily scan them and search for any mistakes or flaws. Taking a visual approach like this would simplify the algorithm selection process. Thus, the Midas program was created to facilitate the selection of a peak detection algorithm. Written in Python, this program allows for the testing of peak detection algorithms on numerous datasets. Different algorithms could be added to the options available within the Midas program. When running the program, a stock dataset, time interval, and peak detection algorithm are selected, and the program generates a graph showing what points the algorithm labeled as peaks and valleys. Additionally, a specific date range within the stock dataset can be selected to narrow

the size of the displayed graph. The Midas program makes comparing peak detection algorithms easier and faster.

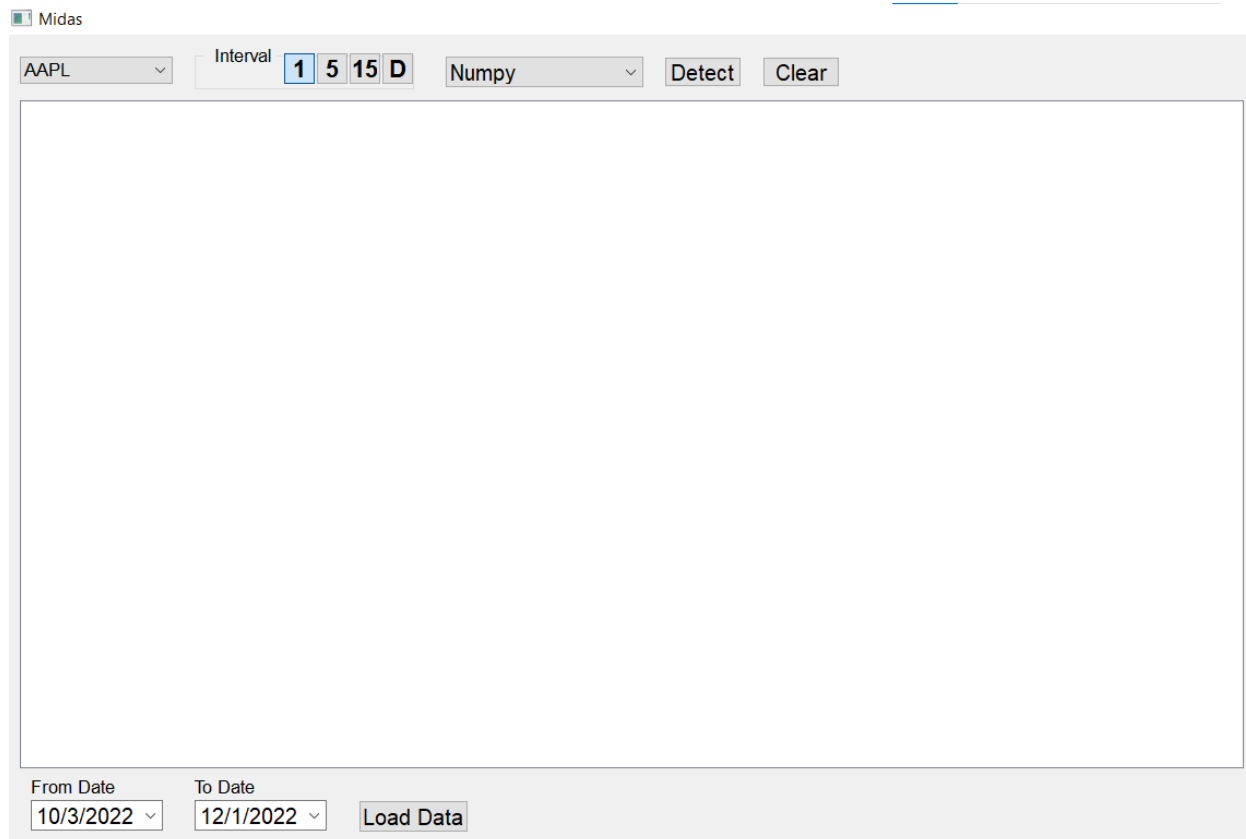


Figure 1. Midas GUI (Graphical User Interface)

Chapter 4

Peak and Valley Detection and Labeling

4.1 The NumPy Indicator

The first peak detection algorithm added to the Midas program for testing was from the NumPy package. NumPy is one of the fundamental packages needed when using Python for scientific computing or machine learning. Within the NumPy package, there are two key

functions needed for peak detection. The first function is named `diff()` and is used to “calculate the n-th discrete difference along the given axis” (*Numpy.diff*). The second function is the `sign()` function, which simply returns negative one if passed a negative number as a parameter, zero if passed zero, or positive one if passed a positive number (*Numpy.sign*). Figure 2 demonstrates how these functions can be weaved together to detect the local extrema, local minimums, and local maximums within a given dataset. In simple terms, the function is checking whether the difference between two data points is positive or negative. The addition of a one at the end of each line is done to counteract one of the effects of the `diff()` function. Whenever used, the `diff()` function reduces the original index number by one and simply adding one to the variables resolves this issue. Overall, the NumPy detector is simple and compact. While it certainly works in theory, the aforementioned Midas program allows users to look at its performance with their own eyes and judge accordingly.

```
local_extremes = np.diff(np.sign(np.diff(org_data))).nonzero()[0] + 1 # local min & max
lmin = (np.diff(np.sign(np.diff(org_data))) > 0).nonzero()[0] + 1 # local min
lmax = (np.diff(np.sign(np.diff(org_data))) < 0).nonzero()[0] + 1 # local max
```

Figure 2. NumPy Detector Code

The NumPy detector is already preloaded for use in the Midas program, allowing it to be selected and used on any valid stock dataset. Figure 3 shows the Midas program after the NumPy detector had been run. The dataset being used is that of Tesla stock closing prices from February 7th, 2022, to February 8th, 2022, on a one-minute interval. The graph shows the point of each peak and valley marked by the NumPy detector and it immediately stands out that there are a lot of peaks and valley detected, too many in fact. Technically, there is absolutely nothing wrong with this detector, but for our purposes it does not work. Each and every time the stock price changes its price trend, there is a peak or valley created and that is too sensitive. Stock prices

fluctuate quite a bit and if every single peak or valley, no matter how minute, is detected and marked, the dataset will be flooded with harmful data. Within Figure 3, take specific note of the peak and valley marked around February 7th, 2022, at 10:00 a.m. (both the date and specific peak and valley have been highlighted in yellow for convenience). These two points are not ones that should be labelled in our data as there are more extreme peaks and valleys located nearby. The second peak and fifth valley, when going from left to right, are proper peaks and valleys for our data (both have been highlighted in blue for convenience). These two points are the most extreme peak and valley before February 7th, 2022, at noon. None of the points between the true peak and valley should be marked at all as they are not as extreme of a peak or valley. Unfortunately, NumPy does not have any sort of sensitivity parameter that could be used to adjust this detector. Therefore, attention was shifted away from NumPy and towards other algorithms that were better suited for this research work.

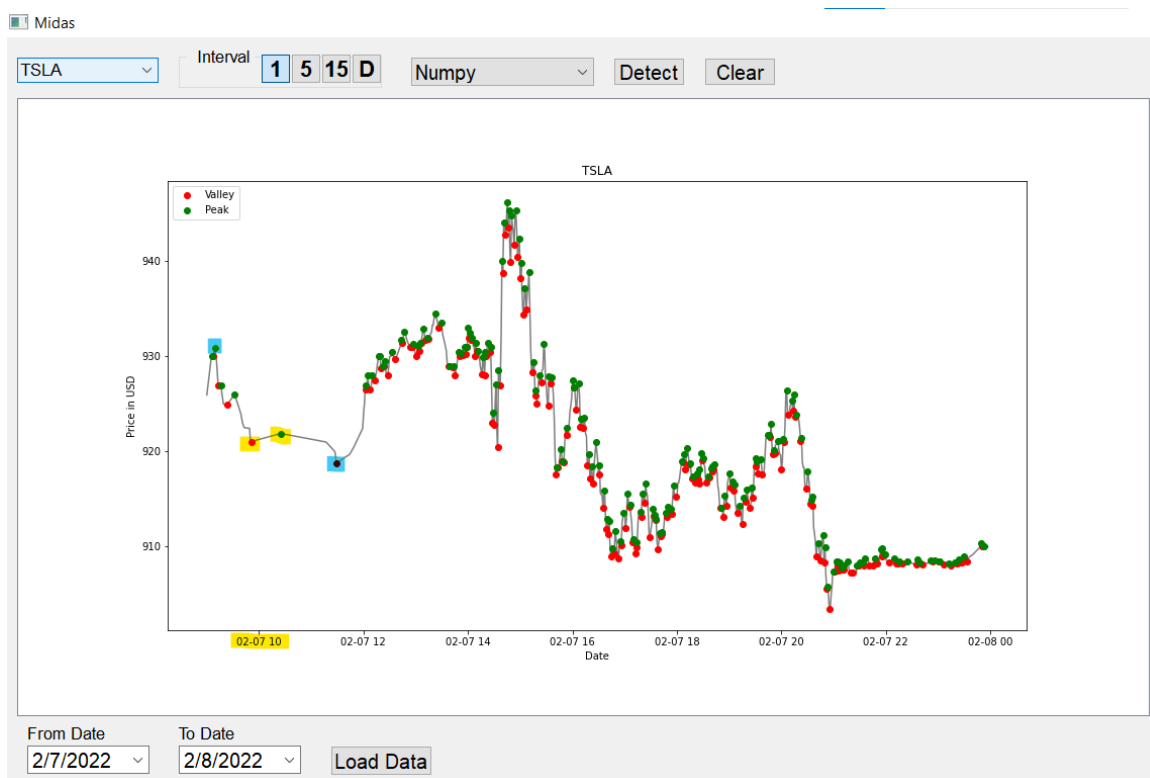


Figure 3. NumPy Detector Sample

4.2 Zig-Zag Indicator

For the next peak detection algorithm, we looked towards the *ZigZag* package. A small package and project with the sole goal of providing useful functions for Python projects using stock market data. The methodology used for this detector is rather fascinating, as the *ZigZag* detector simply finds the trendline of the stock in question. Basically, the detector creates a line of best fit for the price data of a stock. To determine where the peaks and valleys are, the detector simply looks for the pivots within the trendline and reports those as either a peak or a valley, depending on how the price is changing. Figure 4 shows the exact same scenario, Tesla stock closing prices from February 7th, 2022, to February 8th, 2022, on a one-minute interval. The only difference is that the *ZigZag* detector is being utilized instead of the NumPy one. Notice the

points around February 7th, 2022, at 10:00 a.m. again (highlighted in blue for convenience). The detection is ideal, as there are no points marked between the two of them. It may seem counterintuitive to prefer a detection algorithm that marks less peaks and valleys, but it all relates to data quality. There needs to be a significant disparity between the price of a valley and the price of a peak. Otherwise, the machine learning model could learn to sell as soon as the price begins to rise and sell as soon as it begins to dip. Such a trading strategy would be suboptimal and have the potential for losses. However, there are some points that the ZigZag indicator misses, namely around February 7th, 2022, around 9 p.m. where a valley should be marked. Further adjustments to the sensitivity of the detector could help to alleviate this problem.



Figure 4. ZigZag Detector Sample

Since the ZigZag detector was able to pass the first stage of testing, it could progress to the next stage. The peaks and valleys are detected, but there are two other labels. Namely, when the price is up trending and when it is down trending. Achieving this was relatively simple, as shown in Figure 5. First, a note is made of the trend value at the current. If the trend value is 1, that means it was marked as a peak; likewise, if the trend value is -1, the point was marked as a valley. Whenever one of these marked points is detected, they are updated to 2 and -2 respectively. Additionally, a note is made of whether the price is down trending or up trending based on whether the last observed extreme was a peak or a valley. Finally, a check is made to see if the current index has a trend of 0; meaning that it was not marked as a peak or a valley. If this is the case, its trend is updated based on the stored trend variable. Once this is done, the trend of each data point shows a 2 for peaks, 1 for up trending, -1 for down trending, and -2 for valleys.

```
for index in range(df.shape[0]):
    if df.at[index, "trend"] == 1:
        df.at[index, "trend"] = 2
        trend = "Down"
    if df.at[index, "trend"] == -1:
        df.at[index, "trend"] = -2
        trend = "Up"
    if df.at[index, "trend"] == 0:
        if trend == "Down":
            df.at[index, "trend"] = -1
        elif trend == "Up":
            df.at[index, "trend"] = 1
```

Figure 5. Trend Detection Code

Finally, the range of the peaks and valleys needs to be expanded. The detector marks the highest and lowest relative prices as peaks and valleys; however, those are not the only prices that should be considered acceptable. Imagine that you bought a stock for \$5, and its price is

trending upwards. Eventually, the stock price reaches \$19, and you decided to sell it. The stock price continues to climb to \$20 before crashing back down to \$6. It would be inaccurate to say that selling the stock for \$19 was unwise or a mistake. Without a peak range, a machine learning model would label such a situation as a mistake since it did not sell the stock at the highest possible relative price. Therefore, we need to mark the points within a certain price range of each peak and valley as peaks and valleys themselves to give the machine learning model a reasonable range of prices that it could decide to buy or sell stocks at. To accomplish this, a function is run after all other detection has finished. First, there is a defined sensitivity for this function that determines the size of the reasonable price range; for this project, a sensitivity of half a percent was chosen. For each peak or valley, the function determines the reasonable price range by multiplying the point's price by the sensitivity value. That product is then subtracted from the original price and added to the original price to obtain the minimum and maximum, respectively, of the acceptable range. Once the range is calculated, nearby points are evaluated, and if they are within the reasonable range, they are marked appropriately. The function looks both forwards and backwards from the selected extreme to ensure that every appropriate data point is observed. Looking at Figure 6, we can see the final results of these detection steps. There are large and healthy ranges in which the machine learning model can decide to buy or sell the stock, as denoted by the color coded "X" markings. Additionally, the uptrend and downtrend detections are accurate, giving the machine learning model more information on the price's current trajectory. Overall, the ZigZag detector is a strong contender given its flexible parameters and suitability for this type of project.

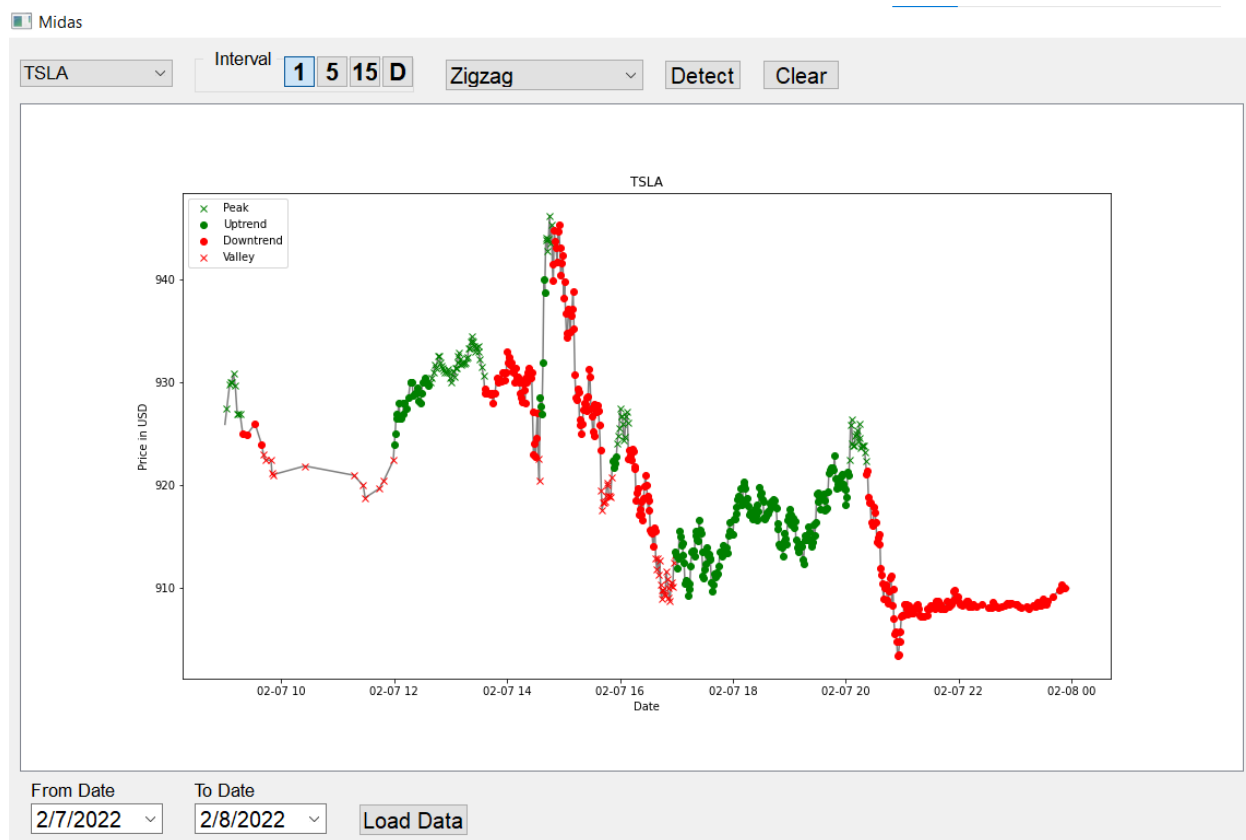


Figure 6. Full ZigZag Detector Sample

4.3 Relative Extrema Indicator

While the ZigZag detector is a clear frontrunner, there was one more peak detection algorithm that we wanted to investigate. The SciPy package, an open source, has a function dedicated to the calculation of relative extrema within data known as `argrextrema()` that will henceforth be referred to as Relative Extrema (*Scipy.signal.argrextrema*). The code for the Relative Extrema is exceedingly simple as it only requires four lines to operate as shown in Figure 7. First, we call the Relative Extrema function, passing in our original price data and the NumPy greater function. This tells the function that we want to find the peak values within our dataset. This is then repeated with the NumPy less function to find the valley values within our

dataset. The second line, where the array is reassigned to itself, is used to convert the NumPy array back to a regular one for ease of use.

```
# Find peaks(max).
peak_indexes = signal.argrextrema(org_data, np.greater)
peak_indexes = peak_indexes[0]

# Find valleys(min).
valley_indexes = signal.argrextrema(org_data, np.less)
valley_indexes = valley_indexes[0]
```

Figure 7. Relative Extrema Detector Code

By testing out the Relative Extrema detector within the Midas program, we can see that it suffers from the same problems as the NumPy detector. As Figure 8 shows, the troublesome points around February 7th, 2022, at 10:00 a.m. are once again present (highlighted in yellow for convenience). It is still an improvement over the NumPy detector, as there are less extraneous labels; however, it is nowhere near the standard that the ZigZag indicator has set.



Figure 8. Relative Extrema Detector Sample

4.4 Indicator Selection and Rationale

Our experimentation with peak and valley detectors only scratched the surface of what is available. Numerous other packages and implementations exist; however, this project's goal is not the evaluation of peak detection algorithms. The ZigZag indicator provides exemplary data in an easily workable form. As the clear frontrunner, the ZigZag indicator was selected as the peak detection algorithm that would be used for our research project. Not only did the other detectors fail to live up to the standard set by the ZigZag indicator, but the ZigZag indicator produced almost exactly what we sought. The odds of another indicator blowing ZigZag out of the water is slim; thus, it is not necessary to invest more time into algorithm analysis.

Chapter 5

Building the Machine Learning Model

5.1 Additional Parameters and Normalization

With the peak and valley detection sorted, there are still a few additional parameters that need to be calculated and added to our data points. The first of those is the Exponential Moving Average (EMA). The EMA helps with “measuring trend direction over a period of time” and the “EMA applies more weight to data that is more current” which helps make the EMA more accurate to the actual price (*What Is EMA? - Exponential Moving Average*). For our data, we measure three different EMAs with the difference between the EMAs being the number of data points they consider in their calculations. We utilize EMA20, EMA30, and EMA60 which calculate the EMA using the nearest twenty, thirty, and sixty data points, respectively.

The second parameter is the Stochastic Relative Strength Index (SRSI). The SRSI “ranges between 0 and 1” and “[SRSI] reading above .8 is considered overbought, while a reading below 0.2 is considered oversold” (Hayes). Knowing when a stock is being overbought or oversold helps to “alert traders that the RSI is near the extremes of its recent readings” (Hayes). Simply put, this parameter helps determine when a stock is reaching an unusually high or unusually low price given its recent activity. When a stock has crossed the overbought threshold, that is a good indication that the stock should be sold. Likewise, when the oversold threshold is crossed, it is a suitable time to buy that stock. For both parameters, we utilize the

pandas-ta python package. This package allows us to easily call function that calculates the values and adds them to our dataset.

For the final part of our data preprocessing, the data needs to be normalized.

Normalization is required whenever parts of a dataset have different ranges. Imagine that you have a dataset about houses containing the number of occupants (ranging from 1-10) and the price of the house (ranging from \$50,000-\$1,000,000). Due to the larger range and values of the price data, it will influence the machine learning model more than the number of occupants. However, depending on the problem you are trying to solve, the price may not be a more important indicator than the number of occupants. This example illustrates that in machine learning, there is an inherit bias that occurs when different number ranges are used. The model will give greater weight than it should to numbers on a larger numerical scale. Data is normalized to be on the same numerical scale to remove this bias. Normally, this range is from 0 to 1 or -1 to 1 but can vary depending on the data being normalized or the method of normalization.

There are two different ranges within our data, so normalization is necessary. Two types of normalization were employed, and either can be used for a run of our neural network. The first type of normalization is Min-Max normalization. The purpose of this normalization is to make sure that each data point is between the values of 0 and 1. The equation, as show in the left side of Figure 9, is the data value minus the minimum value throughout all the data. Then, that difference is divided by the difference of the maximum and minimum values in the data. The second type is Z-Score normalization which normalizes “every value in a dataset such that the mean of all of the values is 0 and the standard deviation is 1” (Bobbitt). The equation for Z-Score normalization is shown in the right side of Figure 9. Each value in the dataset first has the mean

of all the data subtracted from it; then it is divided by the standard deviation of all of the data.

Currently, our project utilizes Min-Max normalization, but it could be easily switch to Z-Score normalization if desired.

Min-Max	Z-Score
$X = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$	$X = (X - \text{Mean of Data}) / (\text{Standard Deviation of Data})$

Figure 9. Normalization Equations

5.2 Sequential Neural Network

Now that all of our data has finally been prepared, the actual machine learning model that will train and test off of this data needs to be constructed. Thankfully, there is a Python package that makes the construction and operation of machine learning models easy and efficient: Keras. It is an Application Programming Interface (API) based on TensorFlow, an “end-to-end open source platform for machine learning” that was developed by Google (*TensorFlow | Google Open Source Projects*). Keras was “developed with a focus on enabling fast experimentation. *Being able to go from idea to result as fast as possible is key to doing good research*” (*Keras Documentation: About Keras*).

Before getting into the specifics of the neural network, it is important that we establish a baseline understanding of the terms and functions of neural networks. First, we will tackle what exactly a neural network is. At its core, a neural network is a type of machine learning that is designed to mimic the human brain. There are layers of neurons that communicate with one

another. Importantly, neurons within the same layer can never communicate with each other. The first layer is the input layer, and its number of neurons is arbitrary. User experimentation is the best method for finding the number of neurons that works best; however, what does matter is the input dimension of the layer. For the first layer, the input dimension needs to be equal to the number of attributes your data has. In our case, we have seven attributes: price, trend, nEMA20, nEMA30, nEMA60, nSRSI_k, nSRSI_d. That means that our input dimension is equal to seven. After the input layer is the hidden layer(s). There is no limit to the number of hidden layers in a neural network, but there needs to be at least one. After the hidden layer(s) is the output layer. This layer basically determines the answer that the neural network gives for a problem. When processing data, each of the neurons that has a connection with another neuron uses that connection to affect the signal strength. Each layer also has an activation function, which is how a node outputs its signal. Finally, the output layer has a number of neurons equal to the number of labels possible. The neurons in this layer determine what answer the neural network will give. Imagine you have an output layer of two neurons. These neurons are used to determine whether the neural network is looking at a picture of a cat or a dog. The dog neuron has a signal of .45 and the cat neuron has a signal of .55. In that case, the neural network would say that it is currently looking at a picture of a cat. While quite simplified, this is the basics of how neural networks operate.

By utilizing the Keras API, a neural network can be designed and structured with a few lines of code as shown in Figure 9. First, the type of model that will be created is denoted. For our project, a sequential neural network is used. A sequential model simply means that our inputs move from layer to layer in order. After that, we can add the needed layers and denote their parameters. Further details of the model layers will be covered in the next chapter, Chapter 5.3.

```
model = keras.Sequential()
model.add(layers.Dense(16, input_dim=7, activation='relu', name='input'))
model.add(layers.Dense(12, activation='sigmoid', name='hidden'))
model.add(layers.Dense(4, activation='softmax', name='output'))
```

Figure 10. Sequential Neural Network Code

5.3 Neural Network Layers

The neural network that we created only has three layers with each of them being a dense layer. The structure of the model and its layers are shown in Figure 9 and a more detailed look at each individual layer is found in Table 1. There are 16 neurons within the first layer that is denoted by the first parameter. As previously established, we require an input dimension of seven due to the seven attributes our data has. Finally, the input layer has the “relu” activation parameter. This means that this layer will be utilizing a rectified linear unit activation function. This function returns 0 if the number is negative and returns the number itself if it is greater than or equal to 0. The next layer is the singular hidden layer within this model. It has 12 neurons within it and a “sigmoid” activation parameter. The sigmoid activation function takes the input value and returns a value between zero and one. The larger the input value, the closer to one the output value will be. As expected, that means that the smaller the input value is, the closer to zero it will be.

For the final layer, the output layer, there are a total of four neurons. This layer uses the SoftMax activation function. The SoftMax activation function assigns values to each neuron such that the total sum of all neurons in the layer is one. As mentioned before, this function distributes values between the four neurons such that the sum of all four neurons’ values will be one. Whichever of the neurons has the highest value determines which label the neural network

applies to this specific problem. For our specific project, this means that whichever of the neurons is the strongest will determine whether the data point is predicted to be a valley, downtrend, uptrend, or peak.

Finally, Table 2 gives additional information about the layers. Namely, the first column shows the layer's type. All of the layers of this model are dense, meaning that each neuron in the layer is connected to every neuron in the next layer. Additionally, this table shows the number of parameters that each layer has and the total amount of parameters in the model. Overall, this model structure is on the simpler side, but that is not necessarily a problem. If the model proves to be effective, then it does not matter how many layers it has. There is no set-in stone guideline for the number of layers required in a neural network.

Table 1. Sequential Neural Network Layers

Layer (type)	Output Shape	Param #
input (Dense)	(None, 16)	128
hidden (Dense)	(None, 12)	204
output (Dense)	(None, 4)	52
Total Params:		384
Trainable Params:		384
Non-trainable Params:		0

Chapter 6

Improving the Machine Learning Model

6.1 Initial Accuracy

With the model fully built, it was time to put it to the test. The first run of the model utilized Tesla stock data with a one-day time interval. Before the model was built and compiled, the Tesla data was split into two subsets. A training subset consisting of eighty percent of the original data and a testing subset containing the remaining twenty percent. The model is built and compiled using the training subset and then evaluated using the test subset. After roughly five initial runs, the average accuracy of the model was around thirty three percent. A deeper look into the model's results reveals its pattern of guessing. For its predictions, it would always choose one label, whichever label had been the most used in the training data. Since the model was producing the same result, no matter the parameters, it indicated that the model did not have enough data to formulate a pattern. Thus, it would simply resort to continuously guessing based on what the most commonly correct answer during its training had been. To rectify this, the model needed more data points. With a larger number of data points, it would be easier for the model to analyze and decipher any patterns within the data. Switching to the fifteen-minute interval greatly increased the number of data points available and showed a startling increase in accuracy. The model went from a mere thirty-three percent to an average of eighty-seven percent. Likewise, the model produced accuracies in the nineties and near one hundred percent when utilizing the five-minute interval and one-minute intervals. For each of these time intervals, the data was first split into training and testing sets to be used for compilation and evaluation, respectively. No longer was the model purely guessing based on the most popular label in the

training data. The model was quite accurately establishing a pattern within the data and most of the time correctly predicting the label. However, there were occasional runs that would reach an accuracy of one hundred percent. While a high accuracy can be a good sign, a consistent accuracy of one hundred percent could be evidence of overfitting. Further discussion of overfitting and its relevance to the model will be discussed in Chapter 6.2.

During these initial runs, the model had a slightly different layer composition than it does now. Namely, the second layer, known as the hidden layer, had a SoftMax activation function rather than a sigmoid activation function. This minor change in the model had quite an effect on the model's accuracy and will be further discussed, along with the rationale for the change, in Chapter 6.3.

6.2 Additional Validation Metrics

While the initial accuracies are quite good, there are no details to them. To better understand the true accuracy of the model, several additional validation metrics were added that provide greater context for the accuracy scores. The first of these metrics is k-cross validation. To utilize k-cross validation, a given training dataset is split into k-number of equally sized datasets. Take our project for example, where the k value is five. As mentioned previously, the original data set is split into two subsets. There is the training dataset with a roughly eighty percent share of the original data and the testing dataset that contains the other twenty percent of the original data. For k-cross validation, the training dataset is further split into five equally sized subsets. For the first run, also known as a fold, the first four subsets of the training dataset are used for training and the fifth subset is reserved for validating the model. Once the model has

been compiled using the training and validation sets, it is evaluated using the test set that was put aside at the very start. Since this testing set has data that the model has never seen before, it provides more authentic accuracy metrics. The model is then completely discarded, rebuilt, and trained again, this time utilizing all the training subsets except the fourth, which is now the subset that is reserved for validation. As before, once the model has been compiled, it is evaluated using the testing set which this newly trained model has never seen before. These folds are repeated for the value of k , such that each training subset is used for validation once. The accuracy metrics from each of the model's testing evaluations are then averaged to summarize the capability of the model. K-cross validation ensures that every data point is being used to test the model, without compromising the integrity of the accuracy score. Additionally, k-cross validation helps to avoid overfitting. In machine learning, overfitting is when a model has become too trained on a set of data, such that it will perform horribly on any new data and be unable to make accurate predictions.

Furthermore, accuracy is not the only metric by which a model's performance can be measured. Three of the most popular metrics are precision, recall, and F-1 score. Precision is calculated by dividing the number of true positives by the sum of the true and false positives and its formula can be seen in Figure 11b. Normally, precision is relied upon as a metric when the consequences of a false positive are rather detrimental. Imagine if a facial recognition model was used to find criminals. A false positive would have the potential to derail or even ruin an innocent person's life. Then there is recall, which is calculated by dividing the true positives by the sum of the true positives and false negatives as shown in Figure 11c. Recall is the other side of the coin when compared to precision and is used when the consequences of a false negative are severe. An example of this would be a model that predicts whether a patient has cancer or

not. If the model were to report that a patient did not have cancer when they did, a false negative, it would be disastrous and potentially cost someone their life. Finally, there is the F-1 score metric that is calculated by dividing the product of precision and recall by the sum of precision and recall and multiplying all of that by a factor of two as seen in Figure 11d. The F-1 score settles in a niche between precision and recall, normally finding use when there is not an even distribution between the labels of a model. Ideally, precision and recall should be .75 or greater and the F-1 score should be as close to one as possible. In this project, both the consequences of a false positive and a false negative could be severe. Within the context of this project, both false positives and false negative could equate to improper trade actions. A false positive means a trade is marked where there should not be one and a false negative means that the model missed a lucrative trade price. Trading at the wrong time on the stock market could lead to significant losses or a significantly reduced gain. Therefore, an F-1 score would be a valuable metric to have. As discussed, the calculations for the F-1 score require both precision and recall; therefore, those metrics have also been tracked and displayed for the model.

The final validation metric added to the model was a confusion matrix. A confusion matrix displays the predicted labels of the model in comparison to their true labels. Essentially, it shows where the model is going wrong. An example of a confusion matrix can be seen in Figure 12. In this case, the model is mislabeling all valleys as downtrends, but accurately predicting every other label. Breaking down the model's error in this way help to isolate potential problems with the model. In specific regards to the original model, the confusion matrix shows that the model mislabels valleys as downtrends, meaning that further tuning to the layers or the addition of parameters can be done to rectify this issue.

(a)		True Label	
		+	-
Predicted Label	+	True Positive	False Positive
	-	False Negative	True Negative
(b)			
Precision = $\frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$			
(c)			
Recall = $\frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$			
(d)			
F-1 Score = $2 \left(\frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \right)$			

Figure 11. Validation Metrics Formulas

```

Confusion Matrix:
                True Class:
                Valley  Downtrend  Uptrend  Peak
Valley Predicted As: [ 0      21142    0      0      ]
Downtrend Predicted As: [ 0      89807    0      0      ]
Uptrend Predicted As:  [ 0      0      96164  0      ]
Peak Predicted As:    [ 0      0      0      23418 ]
  
```

Figure 12. Initial Model's 1-Minute Confusion Matrix

6.3 Current Accuracy

In addition to the added validation metrics, there was one more tweak made to the model. As mentioned in Chapter 6.1, the hidden layer of the model previously employed a SoftMax activation function, but that has now been switched to a sigmoid activation function. Switching the activation function was done to better follow standard conventions. Normally, the SoftMax activation function is reserved for the output layer of a neural network. It was prudent to switch the hidden layer's activation function to one that was better suited for use in a hidden layer. The

average accuracy, precision, recall, and F-1 scores of the folds with the current model structure for the fifteen-minute data interval are shown in Figure 13. The peak validation metrics are consistently strong, averaging a precision of .61, a recall of .8, and an F-1 score of .68. Likewise, the valley validation metrics average a precision of .67, recall of .8, and F-1 score of .73. Additionally, Figure 14 showcases the confusion matrix of one of the folds. These metrics increase to near one hundred percent values within the one-minute and five-minute data intervals of the Tesla and Apple stock datasets. Once again, this could be evidence of overfitting. It is possible that these intervals have too many data points. When a model trains too long on a single sample set of data, it can end up become too in tune to the data's pattern and begin overfitting.

Despite the high validation metrics for the one-minute, five-minute, and fifteen-minute intervals, and the possible overfitting of the five-minute and fifteen-minute intervals, the one-day interval is not nearly as accurate. The scores in Figure 15 show the average validation metrics for the Tesla stock when utilizing the one-day interval. As seen in Figure 16, the model has trouble identifying peaks and valleys after having been trained on the one-day data interval. This is a major problem, as the model identifies no peaks and no valleys. Peaks and valleys make up a disproportionately small portion of the dataset when compared to uptrends and downtrends, but they are the most vital labels. The peaks and valleys determine when a stock is sold and bought, respectively. If there are no peaks or valleys, then the model will never buy or sell a stock and it will continue to hold onto it forever unless a human force intervenes. However, the mislabeling is not as bad as it could be. Valleys are being mislabeled as downtrends and peaks are being mislabeled as uptrends. It would be far worse if the peaks were to be mislabeled as downtrends or valleys, as that would indicate a deeper problem with the model or the data itself. For now, the

one-day interval is simply unviable for training the model, but further research could seek to rectify this. Further details about future research potential can be found in Chapter 7.2.

```

-----
Score per fold
-----
> Fold 1 - Loss: 0.323578417301178 - Accuracy: 85.81656813621521%
> Peak Precision: 0.0 - Peak Recall: 0 - Peak F1-Score: 0
> Valley Precision: 0.9237132352941176 - Valley Recall: 1.0 - Valley F1-Score: 0.9603440038222646
-----
> Fold 2 - Loss: 0.3798329532146454 - Accuracy: 80.78584671020508%
> Peak Precision: 0.45108947612424666 - Peak Recall: 1.0 - Peak F1-Score: 0.6217252396166134
> Valley Precision: 0.0 - Valley Recall: 0 - Valley F1-Score: 0
-----
> Fold 3 - Loss: 0.2616311013698578 - Accuracy: 96.69802784919739%
> Peak Precision: 0.9216330361332707 - Peak Recall: 1.0 - Peak F1-Score: 0.9592185592185593
> Valley Precision: 0.8130975143403442 - Valley Recall: 1.0 - Valley F1-Score: 0.8969153704191933
-----
> Fold 4 - Loss: 0.27969658374786377 - Accuracy: 95.61512470245361%
> Peak Precision: 0.9666512274201019 - Peak Recall: 1.0 - Peak F1-Score: 0.9830428638718793
> Valley Precision: 0.6835383159886471 - Valley Recall: 1.0 - Valley F1-Score: 0.8120258499578533
-----
> Fold 5 - Loss: 0.27388888597488403 - Accuracy: 95.70980668067932%
> Peak Precision: 0.7144181733889662 - Peak Recall: 1.0 - Peak F1-Score: 0.8334234721471067
> Valley Precision: 0.9480705097665555 - Valley Recall: 1.0 - Valley F1-Score: 0.9733431156762044
-----
Average scores for all folds:
> Accuracy: 90.92507481575012 (+- 6.436116853453666)
> Peak Precision: 0.610758382613317
> Peak Recall: 0.8
> Peak F-1 Score: 0.6794820269708317
> Valley Precision: 0.6736839150779329
> Valley Recall: 0.8
> Valley F-1 Score: 0.7285256679751031
-----

```

Figure 13. 15-Minute Validation Metrics

	True Class:			
	Valley	Downtrend	Uptrend	Peak
Valley Predicted As:	[1990	109	0	0]
Downtrend Predicted As:	[0	5827	0	0]
Uptrend Predicted As:	[0	0	6816	0]
Peak Predicted As:	[0	0	616	1541]

Figure 14. Current Model's 15-Minute Confusion Matrix

```

-----
Score per fold
-----
> Fold 1 - Loss: 1.0281938314437866 - Accuracy: 84.91228222846985%
> Peak Precision: 0.0 - Peak Recall: 0 - Peak F1-Score: 0
> Valley Precision: 0.0 - Valley Recall: 0.0 - Valley F1-Score: 0
-----
> Fold 2 - Loss: 1.485758900642395 - Accuracy: 2.816901355981827%
> Peak Precision: 0.21052631578947367 - Peak Recall: 0.09302325581395349 - Peak F1-Score: 0.12903225806451613
> Valley Precision: 0.0 - Valley Recall: 0.0 - Valley F1-Score: 0
-----
> Fold 3 - Loss: 1.1492609977722168 - Accuracy: 84.50704216957092%
> Peak Precision: 0.0 - Peak Recall: 0 - Peak F1-Score: 0
> Valley Precision: 0.0 - Valley Recall: 0.0 - Valley F1-Score: 0
-----
> Fold 4 - Loss: 1.1519476175308228 - Accuracy: 36.61971688270569%
> Peak Precision: 1.0 - Peak Recall: 0.25925925925925924 - Peak F1-Score: 0.4117647058823529
> Valley Precision: 1.0 - Valley Recall: 1.0 - Valley F1-Score: 1.0
-----
> Fold 5 - Loss: 1.4135605096817017 - Accuracy: 7.394365966320038%
> Peak Precision: 0.0 - Peak Recall: 0.0 - Peak F1-Score: 0
> Valley Precision: 0.5454545454545454 - Valley Recall: 0.5454545454545454 - Valley F1-Score: 0.5454545454545454
-----
Average scores for all folds:
> Accuracy: 43.250061720609665 (+- 35.78354901676733)
> Peak Precision: 0.24210526315789474
> Peak Recall: 0.07045650301464254
> Peak F-1 Score: 0.10815939278937381
> Valley Precision: 0.3090909090909091
> Valley Recall: 0.3090909090909091
> Valley F-1 Score: 0.3090909090909091
-----

```

Figure 15. Current Model's 1-Day Validation Metrics

Confusion Matrix:

	True Class:				
	Valley	Downtrend	Uptrend	Peak	
Valley Predicted As:	[0	17	0	0]
Downtrend Predicted As:	[0	87	0	0]
Uptrend Predicted As:	[0	0	155	0]
Peak Predicted As:	[0	0	26	0]

Figure 16. Current Model's 1-Day Confusion Matrix

6.4 Simulating Paper Trading with the Current Model

With the model producing acceptable predictions, the next step was to truly test it through paper trading. Since the implementation of paper trading was done on a per fold basis, the final portfolio value was the average of all the fold's ending portfolio values. To simulate

paper trading, the model fold is given a starting portfolio value. For this model, the starting portfolio value was set at \$100,000. Once the model is trained within the fold, it moves on to making predictions on the unseen test set. As these predictions are made, the model can choose to buy stocks at valleys and sell them at peaks. The investment portfolio is updated for each trade to accurately reflect its value, with currently held stocks not being counted in its available capital. Additionally, there are standard safeguards put into place. Crucially, the model can only sell stocks if it has some in its possession. Currently, the model is set to buy and sell in groups of ten, but this variable can easily be tweaked. Due to the number of peaks and valleys in the data, a cooldown system is also in place. These are custom variables that dictate the amount of time between trades. For example, imagine that the price is approaching a peak. Without a cooldown, the model would instantly sell as soon as it predicts the first peak. However, this model does not predict a singular peak or valley, but rather a range of them. Therefore, it is more desirable for the model to wait for a few peaks to be detected before selling or for a few valleys to be detected before buying. That way, the trade happens closer to the highest peak or valley as they should be located closer to the middle of the predicted extrema range. By utilizing the cooldown variables, the model will ignore a specified number of peaks or valleys it detects before making a trade. When the model begins to predict downtrends or uptrends again, these cooldown variables are reset in anticipation of the next extrema cluster. Finally, once the model has reached the end of its test data, any stocks that are still held by the model are sold for the last price point available to completely liquidate the portfolio.

Four popular stocks were chosen for paper trading within this model: Tesla (TSLA), Apple (AAPL), Microsoft (MSFT), and Nvidia (NVDA). Each of these stocks were simulated in the folds of models built using the one-minute, five-minutes, fifteen-minute, and one-day data

intervals. Table 2 shows the average portfolio value for each stock's models after simulating paper trading. Across every data interval, the model lost money using the Tesla stock. The Apple portfolio performed successfully on every interval except for the one-day interval. This is to be expected given the poor predictions made by the model on one-day interval data. For both the Microsoft and Nvidia stock data, the model was rather successful across all data intervals. One of the most peculiar results is the fifteen-minute portfolio for Microsoft. A further look, specifically at the confusion matrices, shows that the model was unable to predict valleys or peaks and as such never bought or sold any stocks, leaving the portfolio unchanged.

Overall, the results are quite promising, with ten out of the sixteen simulations producing gains. Of course, there is obviously still a lot of room for improvement. As previously mentioned, the model was limited to selling or buying ten stocks at a time, which limits both the potential gains and potential losses. For most of the losses, they are not the result of poor trades being made by the model. Instead, they result from the model being forced to liquidate its assets at the end of the dataset. Normally, the model would wait for another peak to appear, but the stocks need to all be liquidated once the end of the test set has been reached. The main outlier is the one-day Tesla portfolio. Namely, there is a fold that predicts all points as valleys. This tanks the average of the portfolio, as the other folds tend to perform quite normally, only reporting losses due to reaching the end of the test set. Since every point is predicted as a valley, that particular fold buys stocks when it most certainly should not, including when the price is at a peak. This is to be expected, given the poor accuracy of the one-day interval's predictions. Without proper predictions, the paper trading is not going to perform well. The one-day interval is still considered unviable but was included in the paper trading simulations to give a complete picture of the model's performance.

Table 2. Average Investment Portfolios After Paper Trading

Stock Name	One Minute Portfolio	Five Minute Portfolio	Fifteen Minute Portfolio	One Day Portfolio
Tesla (TSLA)	\$98,276.62	\$99,604.04	\$99,900.74	\$96,990.96
Apple (AAPL)	\$102,853.36	\$100,688.86	\$100,072.30	\$99,574.18
Microsoft (MSFT)	\$101,968.88	\$100,723.94	\$100,000	\$100,196.5
Nvidia (NVDA)	\$101,377.56	\$101,344.72	\$100,448.18	\$100,546.58

Chapter 7

Conclusion

7.1 Takeaways

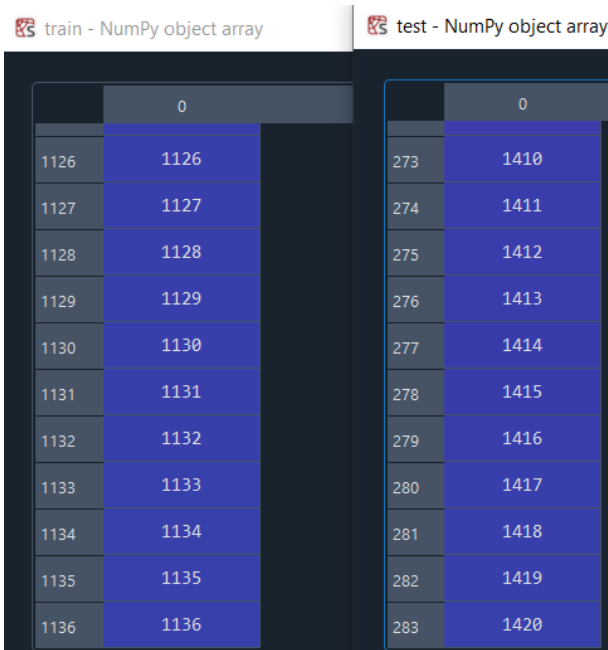
When examining the results of any project, the first thing to look at is the hypothesis. In Chapter 1, it was hypothesized that the machine learning model would be able to achieve an accuracy between sixty and seventy percent and make gains on fifty-five percent of its paper trading simulations. This model exceeded the expectations set by the hypothesis when using any data interval other than the one-day one. The fifteen-minute interval produced an average overall accuracy in the low nineties. The one-minute and five-minute intervals produced accuracies near one hundred percent; however, overfitting seems to be a problem. Numerous additional

validation metrics were employed to gain a better understanding of these scores and the fifteen-minute interval showed the most promise. The precision, recall, and F-1 scores are near the ideal thresholds. As previously mentioned, precision and recall should around .75 or greater and the F-1 score should be as close to one as possible. As for the paper trading simulations, the model was able to produce gains in sixty-two and a half percent of its test runs, exceeding the anticipated fifty-five percent. These results are a strong indicator that machine learning is quite potent within the stock market. Of course, there are nuances to its validity and numerous questions, ideas, and improvements that could be explored in future research.

One nuance of the machine learning model's performance is the time interval of the dataset. Currently, the sequential neural network is able to achieve a near one hundred percent accuracy on the one-minute and five-minute intervals. However, that accuracy dips to an average of forty percent when the dataset has a one-day interval. As discussed in the previous chapter, Chapter 6.3, the confusion matrix shows that the model struggles to identify the peaks and valleys when using the one-day data interval. However, there is a small upside to the one-day interval accuracy. The model is misidentifying peaks as uptrends and valleys as downtrends. Obviously, a misidentification is never the ideal result; however, it is promising that the model still properly identifies the direction of the stock price perfectly. It would be much more alarming if the model misidentified a peak as a downtrend or especially if it were misidentified as a valley. Within the scope of this project, the one-day interval has shown to be suboptimal; however, it is not unsalvageable. Overall, even when utilizing the one-day interval, there were certain folds that showed impressive accuracy metrics.

The main nuance to the model's performance is due to the outstandingly high accuracy it is reporting on certain time intervals. Overfitting is a critical concern for this model. Despite the

implementation of k-cross validation, the model seems to be performing too well. Overfitting can occur when the training set and test set are not properly separated. As mentioned in Chapter 3.1, separating the two sets is vital. Otherwise, the model is being tested on the problems that it already practiced. However, this does not appear to be the case for this model. The training and test datasets appear to be separating properly, as shown in Figure 17. Within Figure 17 are two arrays, there is the training data array on the left and the test data array on the right. Inside of the arrays are index numbers correlating to entries within the stock dataset. As can be seen, indexes are not appearing in both arrays and the training set is about four times as large as the testing set. This is indicative of a properly split data set. Additionally, the split preserves the order of the data, with the testing set containing the newest data from the original dataset. This is crucial for stock data as the order of the values needs to be preserved. If the model were to be continuously fed new data, it could help to definitively identify whether overfitting is occurring. This could be accomplished through the Alpaca API and is further detailed in the next chapter, Chapter 7.2.



train - NumPy object array		test - NumPy object array	
	0		0
1126	1126	273	1410
1127	1127	274	1411
1128	1128	275	1412
1129	1129	276	1413
1130	1130	277	1414
1131	1131	278	1415
1132	1132	279	1416
1133	1133	280	1417
1134	1134	281	1418
1135	1135	282	1419
1136	1136	283	1420

Figure 17. Training and Testing Dataset Split

7.2 Future Research Potential

The results of this thesis project are certainly promising. Perhaps the most exciting aspect of this study is the potential questions that can be researched in the future. First, there is the improvements that could be made to the paper trading simulations. As previously mentioned in Chapter 3.2, paper trading is when someone or something participates in the stock market without using actual capital. It allows for a testing of mettle within the stock market without any of the real gain or real risk. The data collection for this project was done using the Alpaca API, which also has paper trading functionality. Within this study, the paper trading was simulated using the test subsets of the historical stock data gathered. Future studies could further augment the model with the Alpaca API and allow it to harvest real-time stock data and simulate paper trading with that data. Real-time harvesting covers a wider variety of market conditions and limitless test data. Such studies could also provide more insight into the possibility of an overfitting phenomenon occurring.

Other future studies could further explore the one-day interval and determine what other layers need to be added or what other data processing needs to be done to bring it in line with the other time intervals. Alternatively, it could be a matter of data points. It could be that the one-day interval simply suffers because there are not enough data points for the machine learning model to properly establish a pattern within the data. Of course, those are simply ideas for future studies that build directly from this one.

BIBLIOGRAPHY

- Azam, Farooq. "Biologically Inspired Modular Neural Networks." *Doctoral Dissertations*, 19 May 2000, <http://hdl.handle.net/10919/27998>. Accessed 27 Mar. 2023.
- Bobbitt, Zach. "Z-Score Normalization: Definition & Examples." *Statology*, 12 Aug. 2021, <https://www.statology.org/z-score-normalization/#:~:text=Z%2Dscore%20normalization%20refers%20to,the%20standard%20deviation%20is%201>.
- "Developer-First API for Trading." *Alpaca*, AlpacaDB, Inc., <https://alpaca.markets/>.
- "Find Open Datasets and Machine Learning Projects." *Kaggle*, <https://www.kaggle.com/datasets>.
- Gandhmal, Dattatray P., and K. Kumar. "Systematic analysis and review of stock market prediction techniques." *Computer Science Review* 34 (2019): 100190.
- Hegazy, Osman, Omar S. Soliman, and Mustafa Abdul Salam. "A machine learning model for stock market prediction." arXiv preprint arXiv:1402.7351 (2014).
- Hayes, Adam. "Stochastic RSI - STOCHRSI Definition." *Investopedia*, Investopedia, 30 Jan. 2023, <https://www.investopedia.com/terms/s/stochrsi.asp>.
- Kannan, K. Senthamarai, et al. "Financial stock market forecast using data mining techniques." *Proceedings of the International Multiconference of Engineers and computer scientists*. Vol. 1. 2010.

“Keras Documentation: About Keras.” *Keras*, Keras Team, <https://keras.io/about/>.

Kimoto, Takashi, et al. "Stock market prediction system with modular neural networks." 1990
IJCNN international joint conference on neural networks. IEEE, 1990.

“Numpy.diff.” *Numpy.diff - NumPy v1.24 Manual*, NumPy,
<https://numpy.org/doc/stable/reference/generated/numpy.diff.html>.

“Numpy.sign.” *Numpy.sign - NumPy v1.24 Manual*, NumPy,
<https://numpy.org/doc/stable/reference/generated/numpy.sign.html>.

Rosillo, R., et al. “Technical Analysis and the Spanish Stock Exchange: Testing the RSI, MACD, Momentum and Stochastic Rules Using Spanish Market Companies.” *Applied Economics*, vol. 45, no. 12, 2 Feb. 2012, pp. 1541–1550.,
<https://doi.org/10.1080/00036846.2011.631894>

“Scipy.signal.argrextrema.” *Scipy.signal.argrextrema - SciPy v1.10.1 Manual*, SciPy,
<https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.argrextrema.html>.

“TensorFlow | Google Open Source Projects.” *Google Open Source*, Google,
<https://opensource.google/projects/tensorflow>.

“What Is EMA? - Exponential Moving Average.” *Fidelity*, <https://www.fidelity.com/learning-center/trading-investing/technical-analysis/technical-indicator-guide/ema>.

ACADEMIC VITA

NOLAN STEVENSON

EDUCATION

The Pennsylvania State University

Bachelor of Engineering

Expected Graduation: May 2023

Major: Computer Science

Minor: Game Development

Academic Awards & Honors: Dean's List 2019-2023, National Honor Society, Behrend

Honors Program, Schreyer Honors College

ACADEMIC PROJECTS

The Pennsylvania State University | Erie, PA

Game Development Capstone - Chainbound

Spring 2023

- Designed and implemented key game systems in Unity.
- Led team meetings and delegated project duties.
- Mentored three team members through pair programming.
- Released as an open-source title.

The Pennsylvania State University | Erie, PA

Schreyer Honors Thesis

Spring 2023

- Harvested enormous quantities of data for an SQLite database.
- Labeled the peaks and valleys of stock prices through three different indicators.
- Predicted when to buy and sell stocks through a sequential neural network.

The Pennsylvania State University | Erie, PA

Penn State Behrend Wrestling Team Scraper

Spring 2023

- Obtained massive quantities of data from numerous webpages.
- Filtered data based on desired parameters.
- Cataloged data into Excel spreadsheets for ease of use and organization

The Pennsylvania State University | Erie, PA

Advanced Game Design Final Project – Night

Spring 2022

- Developed enemy behavior using state machines in Godot.
- Created clean and informative User Interfaces.
- Directed the opening cutscene.
- Contributed to player animations and art assets.

EXPERIENCE

Tops Markets, LLC | Jamestown, NY

Cashier

June 2017-Present

- Received payment by cash, check, credit cards, vouchers, or automatic debits.
- Answered customers' questions and provided information on procedures or policies and item locations.
- Updated price tags throughout the store on a weekly basis.

RELATED COMPETENCIES & SKILLS

- C | C++ | Java | Python
- HTML | CSS | JavaScript | SpringBoot | Qt
- MIPS | MARS | LOGISIM
- Unity | Godot | C# | GDScript
- GitHub | Gradle