Machine Learning Applications to Financial Markets: A Deep Learning Framework to Predict Stock Returns

Güneykan Özkaya

Msc. in Finance and Banking

Barcelona School of Management

gozkaya@ku.edu.tr

Ji Qi

Msc. in Finance and Banking

Barcelona School of Management

ji.qi@alum.upf.edu

Advisors: Hasan Sinan Bank, José Suárez Lledó

Acknowledgements

We would like to thank Guney's dog Viktor for his endless support through the development of this thesis.

Abstract

"A blindfolded monkey throwing darts at a newspaper's financial pages could select a portfolio that would do just as well as one carefully selected by experts." (G. Malkiel (2007)). In this paper, we proved that active investing strategies through leveraging Deep Learning methodologies can outperform a passive investing strategy, thus the market itself. Our results demonstrate that, contrary to the general view in the industry, technical indicators capture the behavior of the market participants and through Deep Learning methodologies and technical indicators, we can predict investor behaviors and take advantage of the inefficiencies caused by the market participants.[1].

Keywords – Machine Learning, Deep Learning, Recurrent Neural Networks, Long-Short-Term Memory, Stock Returns Prediction, Bayesian-Based approach.

[1]For those who already have a background in Machine Learning, the code is self-explanatory and it is available online as Jupyter Notebook at https://github.com/guney1/LSTM_Stock_Prediction.

Contents

List of Figures

List of Tables

1 Introduction

Financial markets refers to a digital or physical platform where people can participate in trading financial securities with other participants of the market, incurring a transactional cost. The equity market refers to the financial market in which shares of a given set of companies are available for the participants to buy, sell or hold. The price of a stock will change over time, depending on the foreseeable profitability as the basis of supply and demand for that stock in any given moment. The objective of trading with company shares can be diverse, but in this paper, we are focusing on the case in which market participants invest money in the market, trade stocks and try to make an economic gain. Historically, there have been many strategies implemented to achieve this. The most conventional one is that the investors would try to gain an optimized profit by buying stocks when their price is estimated lowest (undervalued) and selling them when their price is predicted to be highest (overvalued). This approach relies on accurate predictions of the stocks' undervalued or overvalued states. An accurate prediction is extremely difficult due to the non-deterministic complexities and uncertainties. In this paper, we are going to offer a data-driven approach to predict stock market returns one day ahead. We expect our approach will be useful for the investors to make better trading decisions and improve returns on investment. Besides, this approach would be insightful to generate a semi-autonomous trading bot. Ever since Markowitz (1952) formulated the Modern Portfolio Theory, there has been a growing interest in applying quantitative methods to trading in the financial markets. Renowned portfolio selection methods include portfolio building using the variance-return framework, based on technical analysis or fundamental analysis, factor models, and risk parity methods. These methods often require extensive data and subject matter expertise (e.g., a professional asset manager) and both of them are expensive and/or difficult to gather. These approaches are useful up to a certain extent, but the resulting asset selection is significantly sensitive to the volatility due to unexpected changes in the economic cycle. Furthermore, traditional statistical and econometric models have had limited success at predicting stock prices. In fact, if markets were as efficient as suggested by the Efficient Market Hypothesis [EMH] (Malkiel and Fama (1970)) –in the sense that the current asset prices fully reflect all available information– the best prediction of future prices would be the current price. According to EMH, price

movements are random departures from the previous price levels, thus associating the stock prices with a random walk process. However, in the following years, evidence and research suggested that there is some predictability in the stock prices. Malkiel (2003) examines the main findings and critiques to the EMH. Primarily, market investors are found to be far from being rational, so their collective judgment about the stock market can sometimes be wrong — this results in pricing irregularities are the predictive patterns that investors can exploit to make profits. Therefore, finding ways to effectively predict stock prices become a crucial task for market participants to make informed and accurate decisions.

Traditional econometric methods have not had much success in predicting stock prices. The recent proliferation of Machine Learning methods has revolutionized many fields with a wide array of useful applications. In the financial domain, this has been emerging with the introduction of both ultra-fast algorithms and new effective ways to forecast the financial time series. An algorithm is an unambiguous mathematical formalism of how to solve a problem, a set of instructions describing the specific approach to proceed. When executed by a computational system, it follows the given guidelines and performs several tasks on a given set of initial data and produces an output. Machine learning in the context of this thesis belongs to the broader field of data science. It refers to the set of algorithms to solve a problem without being directly instructed to solve it, but instead, it is instructed to build a mathematical model on input data, to recognize patterns and infer the actions to solve the problem. There is a growing body of research applying machine learning and artificial neural network models to predict equity prices, stock movements and to measure equity risk premium (the difference between risk-free return and return obtained on a specific investment).

One significant difference between conventional statistical methods and machine learning methods is that the former is designed to infer the relationship between a set of variables and the variable of interest, whereas machine learning methods can be specifically designed to make the more accurate predictions. This will give us two main advantages for our purpose of forecasting stock returns. First, the set of input data can be expanded to include all the variables with possible relevance to the prediction. Machine learning methods are particularly indicated for discriminating between useful and useless variables given a very large set of input variables, without affecting the robustness of the

model. Second and most importantly, Machine Learning models can capture non-linear relationships between input and output variables and complex interactions among the input variables. Non-linear relationships occur when the ratio between the change in one variable and the change of a different dependent variable is not constant. Many studies have suggested that financial data present non-linear dependency effects. Research has concluded that conventional statistic methods have limited success in capturing complex non-linear relationships among variables. We expect to be able to improve these results through Deep Learning methodologies.

Our objective in this study is to use a type of machine learning algorithm, the Recurrent Neural Network (RNN) with Long-Short-Term Memory (LSTM) implementation to effectively predict the stock returns, using stock price and volume information of the company Apple Inc. and a set of technical indicators as inputs. We will then back-test the model by simulating a trading session in the stock market based on the predictions. We found that the returns generated by our methodology surpass by far those of a passive buy-hold strategy. This suggests that Deep Learning methodologies present an unprecedented potential for predicting stock returns and will help market participants to make better decisions for short term trading and portfolio allocation. The reason we choose a short term trading strategy over other long term approaches is mainly because a long time horizons will increase the difficulty of making accurate predictions. As the prediction time horizon increases, there will be more news and factors that can affect the stock prices, thus increasing its volatility over that period. This leads to a higher volatility of the model predictions, making them less accurate. This means that, everything else being equal, model predictions for the next day will be more accurate than those for e.g. the next week. Therefore, by making short term predictions, our model will have more reliable results.

2 Literature Review

2.1 Conventional Statistical Models

Around 1970, the hypothesis that the stock prices behave like a random walk process was generally accepted or at least, it was hard to refuse. However, in the coming years, a growing body of literature was proving that stock prices are at least partially predictable, mostly because it presented time series correlation. Several methods and approaches were used in an attempt to predict market behaviour. In the regime switching approach, Jaffe et al. (1989) found clear seasonality effects by showing important differences in the returns of January compared to other months. In the same paper, he also found the importance of size and earnings to predict the returns of the stock prices. Friedman et al. (1989) found that very extreme movements in the stock markets hindered simple relationships. By removing a few extreme points, they could find time dependence in the first lag using an AR(1) model, resulting in an R^2 value of 0.036. While attempting to search for more explanatory power using a broad selection of independent variables, Darrat (1990) found significant predictors of the stock prices are the volatility of the interest rates, industrial production, long term interest rates, and public budget deficit, resulting in an R^2 of 0.46.

Other approaches included cointegration relationships and chaos theory (see e.g. Hamao et al. (1990) , Sharma and Seth (2012), Liu et al. (1992)). The main problems with these methods are they are subject to high sensitivity to the changes in the scale of the data, heteroskedasticity, outliers and non-normal distributions. Many of these models have low explanatory power and can only make limited predictions of the stock market behaviour.

2.2 Early Machine Learning Implementations

There are many types of machine learning methods, typically classified by the task they perform. To give an idea, we will describe two common types. Firstly, supervised learning refers to the situation where we know the values or categories of the dependent variable/variables prior to the execution. They can solve classification or regression problems, using methods such as Lasso Regression, Decision Trees, Random Forest or

Support Vector Machines (SVM). On the other hand, unsupervised learning refers to the situation when the categories of the dependent variable are unknown or unclear beforehand. They typically solve clustering or dimensionality reduction problems, using methods such as K-Nearest Neighbourhood (KNN), Hidden Markov Model (HMM) or Principal Component Analysis (PCA).

These machine learning methods for predicting stock market returns have had relatively more success than conventional statistical methods. Because of its success in other fields, the SVM model is very commonly applied to stock prices to predict an upward or downward movement in the price level. Sheta et al. (2015) compared the prediction results of an SVM with those of a Multiple linear regression and showed that the SVM could yield more accurate predictions than the linear regression. Huang et al. (2005) used SVM to predict the direction of the NIKKEI 225 index and concluded that the SVM performed best compared to other models.

Furthermore, given the nature of the SVM, the solution is computationally less heavy to find and it is easier to optimize and achieve the global optimum loss value, unlike the result of some neural networks. Kim (2003) compared the predictive performance of SVM with those of backward propagation neural networks and found that SVM had superior performance. However, we expect that by fine-tuning the neural network hyperparameters, the latter would still perform better than an SVM model. Although the SVM model is better than others, machine learning models, in general, suffer from the problem of overfitting. This means that when the model is instructed to train on a set of specific data-set, the resulting model can give very good results on that set of data, but when it is used on a different set of data, the results drastically worsen. This is due to the reason, model being so well trained on one set of data that it will infer patterns that are specific to that set of data only, so when we try to generalize the model on other data-sets, the model is not capable to accurately make predictions. Tay and Cao (2002) used a C-ascending SVM model to predict financial time series accounting for non-stationarity and non-linearity. Their model suggests an improvement from standard SVM regarding generalization ability on other data-sets and less overfitting.

2.3 Neural Network Implementations

Our work aims to extend current empirical literature on stock return prediction with Deep Learning methodologies. Due to the reason, Deep Learning methodologies proved to be very successful in a wide range of industries; it was inevitable that these methodologies will be applied to financial markets to maximize profits. While we examined several Machine Learning methodologies, the focus of our paper is, particularly on Deep Learning thus, a large portion of the literature review focuses on Deep Learning implementations. There is extensive research going on in the finance industry regarding Deep Learning methodologies, while this extensive literature helps researchers to create state-of-the-art models, we should be careful to distinguish good practices from the wrong ones.

Gu et al. (2018) applied several Machine Learning methods to predict stock returns. They analyzed 30,000 individual stocks starting from 1957 to 2016, their indicators consisted of 900 baseline signals, and they found out that R^2 based on the predictions of Deep Neural Networks higher compared to other methods they experimented. Shah et al. (2018) implemented Deep Neural Networks (DNN) and RNN-LSTM, and their indicators consisted of previous 20 days closing prices of several stock indexes for DNN and previous 50 days closing prices for RNN-LSTM. While their results are promising in terms of directional accuracy and root mean squared error, they do not provide information about if they normalized the data before or after train, valid, test split thus their results are not reliable due to normalization done through mean and standard deviation of complete data set instead of only training data set. Malandri et al. (2018) conducted unique research, instead of predicting stock prices, they aim to predict best asset allocation, which will perform best in terms of returns. They analyze Multilayer Perceptron (MLP), LSTM networks, and Random Forests. They found out that the Stateful LSTM model outperforms other analyzed models and equal weight portfolio in terms of returns. Kim and Kang (2019) compared several Deep Learning methods (MLP, 1D Convolutional Neural Networks, LSTM and Attention LSTM). Their indicators only consisted of previous 20 days returns and their target variables are binary variables which represent an up or down trend for KOSPI index, they found out that LSTM when combined with attention mechanism, outperforms other networks. Moreover, Bao et al. (2017) experiment with an interesting approach; they used Wavelet transformation for denoising the data, then they

extracted deep features using autoencoders. Finally, they used LSTM to predict the next day closing price. Their indicators consisted of technical indicators and several macro-economic indicators, and they predicted the closing price one day in the future. While their results are awe-inspiring, they did not disclose if they did Wavelet transformation before or after train, valid, test split, if they conduct denoising before the split then they introduce bias to their model due to denoising done through standard deviation of whole data-set instead of using the standard deviation of only training data-set. Regardless, their methodology is unique and promising.

Finally, to discover the full potential of our model, we did a specific literature review on different hyperparameter optimization techniques, specifically, we analyzed traditional methods in the industry which are cross-validated GridSearch (GridSearchCV) and Random Search also we analyzed a more promising approach which is cross-validated Bayesian-based optimization with Tree of Parzen Estimator algorithm. Before Bergstra and Bengio (2012) proposed Random Search, GridSearchCV was the to go strategy in the industry after they proved, Random Search can achieve better results in terms of loss values and it is more efficient regarding performance and number of trials needs to be conducted, Random Search became the main hyperparameter optimization technique in the industry. Lastly, we analyzed an approach proposed by Bergstra et al. (2013) which is optimization through Tree of Parzen Estimator algorithm, they found out that the Bayesian optimization approach can achieve much lower error values with less number of trials compared to Random Search.

3 Data

The data we use in this project to train the model are the daily High, Low, Open, Close prices, and trading volume of the public company Apple Inc. (AAPL), covering the period from 1981-03-13 to 2018-11-12. From this period, we obtain 9500 data points. The data is extracted from the Yahoo Finance API. We chose to work with Apple because of the easy access and the long history of the company. Thus we can obtain a vast amount of price data, which is an essential requirement to train Neural Networks.

To calculate the daily returns, we computed the percentage changes from the mid-prices of one day to the next day. The mid-prices calculated as (Day High price + Day Low price)/2. These returns are then shifted one day into the future. This will be our dependent variable that our model tries to predict using the input variables.

For the input variables, we are using a set of 27 technical indicators calculated using the stock price and volume information, note that some of those indicators are composed of several features meaning those indicators gives us a total of 40 features together with the day high, low, open, close, mid, return and volume. We used the TA-lib library to calculate those indicators. Table 3.1 shows the list of indicators we used. Appendix B offers a brief explanation on each of these technical indicators.

Table 3.1: Input Variables

Day High	Chande Momentum Oscillator
Day Low	Money Flow Index
Day Open	Minus Directional Indicator
Day Close	Minus Directional Movement
Day Volume	Plus Directional Indicator
Day Mid	Plus Directional Movement
Day Return	Percentage Price Oscillator
Double Exponential Moving Average	Relative Strength Index
Kaufman Adaptive Moving Average	Stochastic Oscillator
Triple Exponential Moving Average	Moving Average Convergence/Divergence
Weighted Moving Average	Commodity Channel Index
Directional Movement Average Index	Momentum Indicator
Directional Movement Average Index Rating	Simple Moving Average
Directional Movement - Index	Rate of change
Absolute Price Oscillator	Ultimate Oscillator
Aroon	Williams' %R
Aroon Oscillator	Balance of Power

Because of the nature of financial data and the daily frequency, our data has a high degree of noise. Excessive noise creates false signals to the model, thus distorting model predictions. To denoise the data, we filter the input data using the double exponential moving average (DEMA) with a period of rolling three-day window, all the technical indicators calculated on filtered data.

To train and test our model, we split the dataset into three subsamples. The first 9000 observations (1981-03-13 to 2016-11-16) included in the training subset, the next 200 observations (2016-11-17 to 2017-09-05) included in the validation subset, and the last 300 observations (2017-09-06 to 2018-11-12) included in the test subset. First, we use 9000 data points to train the model. Then we validate the model on 200 data points to avoid overfitting. Finally, we test the model on the 300 data points which model never seen before. To further evaluate the performance of our model, we test it on the global data, we use the model trained on Apple's historical data to predict the returns of the S&P500 ETF (Exchange Traded Fund) and the Nvidia from 2017/09/06 to 2018/11/12.

4 Methodology

Our methodology composed of 5 main steps; First, the denoising of the data and calculation of Technical indicators, second, the windowed normalization of the data, third, the model construction, fourth, the hyper-parameter optimization and model execution and fifth, the trading strategy and back-testing. To provide a clear intuition to the reader, we divided this section into three sub-sections. The first sub-section will give an intuition of the data processing, the second section will focus on modelling and hyper-parameter optimization, and the last section will explain our back-testing and trading strategy.

4.1 Data Processing

Data processing is the most crucial part of this project, if we do not adequately process our data, even the most promising models will not yield reliable predictions thus we cannot use the model for trading purposes.

After we acquire High, Low, Open, Close, and Volume information from Yahoo Finance API, we calculate mid-day returns, then we denoise the independent variables using DEMA with a three-day window. It is important to note that DEMA calculated using a rolling window. For instance, to denoise the data of today, we only use data points from previous days, by doing this way, we avoid information leak from future data points to historical data points. After we denoise the data, we split it into three sub-samples; Training set, validation set, and test set. One can find a detailed explanation of the split in the Data section of the paper.

After the split, we normalize the training data using MinMaxScaler from Scikit Learn library. We choose to scale the data between 0 and 1. A critical aspect of the normalization is, instead of normalizing whole training data-set at once, we do it using 600 data point windows. By doing this way, each window normalized through its own mean and standard deviation instead of mean and standard deviation of whole training data-set. The reason for that is; If we do the normalization through mean and standard deviation of the entire training data-set, early data-points will be extremely small, and model will not be able to learn from these data-points also it will disturb the model's ability to generalize on other data sets. After we normalize training data-set, we normalize

the validation and test samples through mean and standard deviation of the last window of the training sample.

4.2 Modelling and Hyper-Parameter Optimization

While we used RNN with LSTM implementation in this particular project, one needs to have a clear understanding of Neural Networks to understand how RNN functions and why we choose to use RNN over MLP. In order to provide a clear understanding of our modelling process, the first part of this sub-section devoted to Feed Forward Neural Networks and intuition behind it, the second part will concentrate on RNN and LSTM, and the third part will explain hyper-parameter optimization method.

4.2.1 Feed Forward Neural Networks

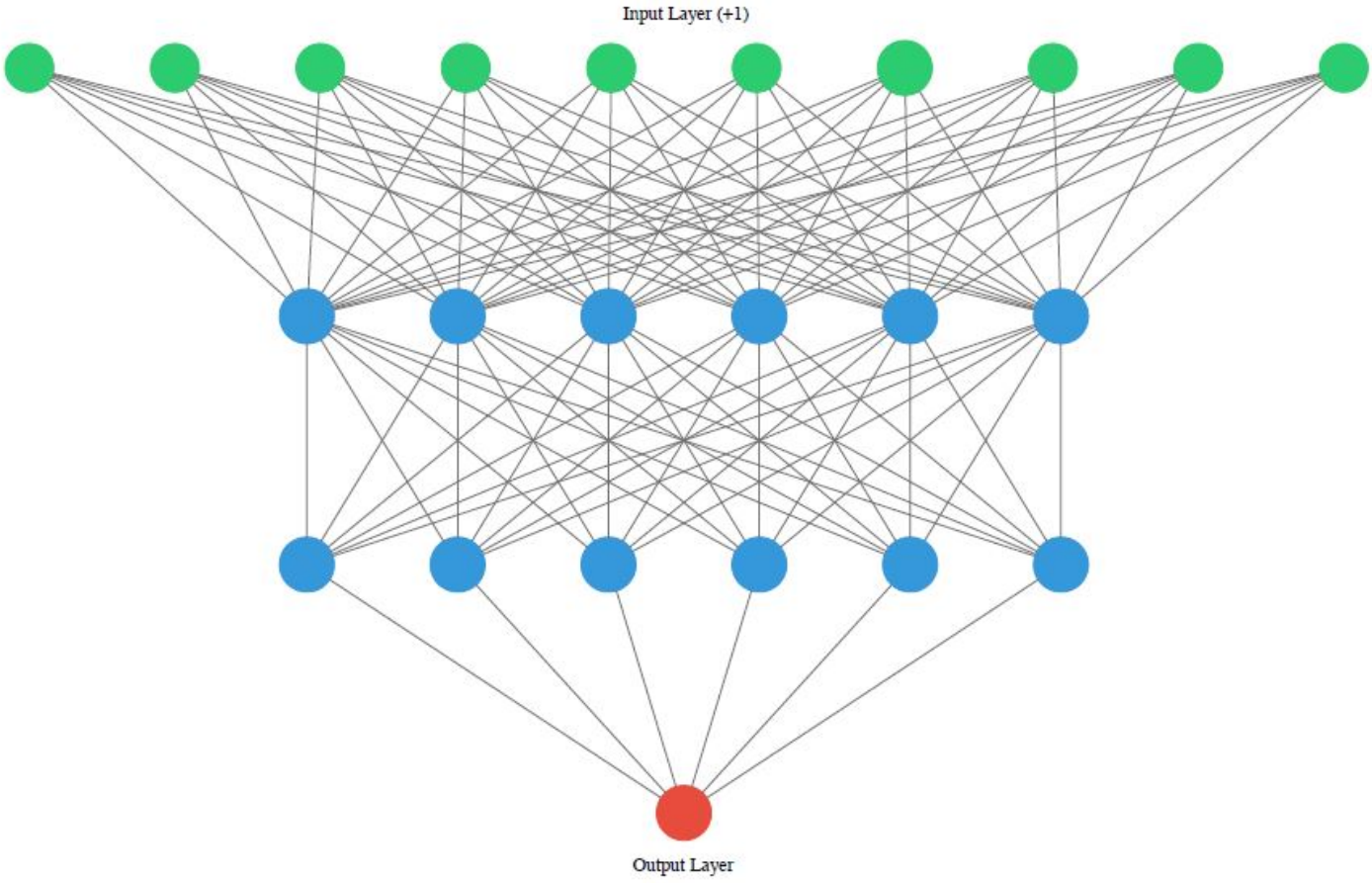
The Perceptron, one of the simplest type of Neural Networks, is based on a certain number of neurons and the activation function which turns on or off the neurons. One of the most commonly used activation functions is Rectified Linear Unit (ReLu). ReLu activates a specific neuron when the input of that neuron positive. When activated, a neuron outputs its input. If the input of the neuron is negative, ReLu deactivates that neuron, and that specific neuron outputs 0 instead of its input.

When we stack more than one fully connected layer, each composed of a certain number of neurons, the network will be able to capture non-linear relationships between features and the target variable or variables. Each neuron in a particular layer connected to all the neurons in the subsequent layer and each of these connections are composed of weights which model adjusts through the training process. To find the most appropriate weights which capture the relationship between inputs of the model and the target variable, we give model a loss function, and the model tries to minimize this loss function with every prediction it makes. There are different kinds of loss functions one can choose depending on the problem, he/she aims to solve, in its most simple form, a loss function is a difference between the real target variable and the predicted target variable.

With every prediction model makes (training step), an optimization algorithm defined by the constructor of the model calculates the partial derivative of the loss function with respect to each weight. Then multiplies these values with a given learning rate

and adjusts the weights with those values (Gradients), this process generally referred as Backpropagation. Learning rate can be stable or adaptive depending on the optimization algorithm we choose. Figure 4.1 provides a visual description of a simple MLP model.

Figure 4.1: Architecture of an MLP model.



4.2.2 Recurrent Neural Networks and Long-Short Term Memory

RNN is a particular type of Neural Networks family, contrary to Feed Forward Neural Networks which activation flows only in one direction (input layer to output layer), RNN has backward connections addition to its forward connections. One can think about RNN as a Feed Forward Neural Networks but unrolled through time. Instead of analyzing data-set point by point, RNN takes a sequence of data points and outputs a value by considering all the data points inside that sequence. One example which RNN suits well is natural language processing, to analyze a sentence or predict next word in a sentence,

we need to know all of the previous words in that particular sentence. Feed Forward Neural Networks do not consider previous data points when making predictions, or during backpropagation thus, it cannot make reliable predictions when data have sequence or time dependency. Equation 1 provides the mathematical intuition behind RNN.

$$\begin{aligned} h_t &= \phi(W_{x,t} \cdot x_t + W_h \cdot h_{t-1} + b_h) \\ y_t &= \phi(W_{y,t} \cdot h_t + b_y) \end{aligned} \tag{4.1}$$

Where:

$h_t \equiv Current\ hidden\ state$

$\mathrm{W}_{x,t} \equiv Input\ weights\ at\ time\ t$

$\mathrm{W}_h \equiv Hidden\ state\ weights$

$\mathrm{h}_{t-1} \equiv Previous\ hidden\ state$

$\mathrm{b}_h \equiv Hidden\ state\ bias$

$\mathrm{W}_{y,t} \equiv Output\ weights\ at\ time\ t$

$\mathrm{h}_t \equiv Current\ hidden\ state$

$\mathrm{b}_y \equiv Output\ layer\ bias$

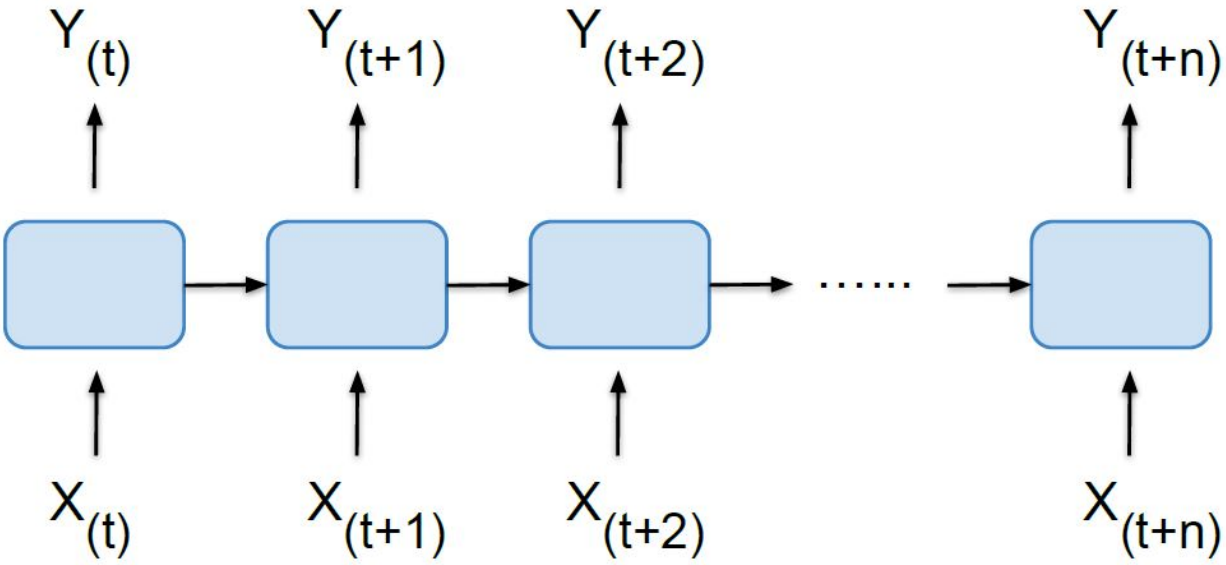
$\mathrm{x}_t \equiv Input\ at\ time\ t$

$\mathrm{y}_t \equiv Output\ at\ time\ t$

$\phi() \equiv Activation\ function$

Time dependency of RNN is the reason why we choose to work with RNN over MLP. Financial markets, in general, have time dependency, for instance, a decrease or increase in today's price of a particular stock may be affected by a sequence of events which happened in previous 10, 20, 30 or even 100 days, RNN allows us to capture information of the previous time steps thus makes more reliable predictions, figure 4.2 shows a hypothetical architecture of an RNN.

Figure 4.2: Architecture of an RNN model.



While RNN is a powerful tool for time series analysis, it has some problems too, widely known as Vanishing/Exploding Gradient. Vanishing gradient is a problem that occurs when we construct deep networks (consist of many layers). During backpropagation, when gradients reach to initial layers of the network, they become so small (Vanishing Gradient) they do not have any effect on the weights at all. When the weights are not adjusted any more, the model stops learning. This problem is more common in RNN due to the unrolling through time. When we construct an RNN model, data of previous time steps flows through memory cells, and if a data point flows through let's say 100 cells, information inside that data point fades away until it reaches to last time step and model cannot process information from that particular time step. On the other hand, the exploding gradient problem refers to exponential gradient accumulation during training. Thus weight updates become larger and larger with every training step causing the model to never converge to minimum loss value.

To avoid vanishing gradient problem and capture long-term dependencies, one can replace memory cells of RNN with LSTM cells. In its essence RNN stores both the current information and the previous information in the same cell. On the other hand, LSTM has two different states, referred to as long-term state and short-term state, and through training, LSTM can learn what to store in the long-term state (which parts of the information affect the prediction) and what to throw away (which parts do not). Inside of

an LSTM cell, there are four fully connected layers, and each serves a different purpose.

1. The first and foremost layer analyses current time step's input and previous hidden state (this is the only layer inside of regular RNN cell).

2. The second layer is called forget gate. It decides which part of the long-term state will be disregarded.

3. The third layer, called input gate, controls which part of the current input will be added to long-term state.

4. The fourth layer, called the output gate, controls which part of the long-term and short-term state affects the current time step output.

Equation 4.2 provides mathematical intuition behind the layers inside of an LSTM cell, and figure 4.3 presents the visual architecture of an LSTM cell.

$$
\begin{aligned}
i_{(t)} &= \sigma(W_{x,i}^T \cdot x_{(t)} + W_{h,i}^T \cdot h_{(t-1)} + b_i) \\
f_{(t)} &= \sigma(W_{x,f}^T \cdot x_{(t)} + W_{h,f}^T \cdot h_{(t-1)} + b_f) \\
o_{(t)} &= \sigma(W_{x,o}^T \cdot x_{(t)} + W_{h,o}^T \cdot h_{(t-1)} + b_o) \\
g_{(t)} &= tanh(W_{x,g}^T \cdot x_{(t)} + W_{h,g}^T \cdot h_{(t-1)} + b_g) \\
C_{(t)} &= f_{(t)} \otimes C_{(t-1)} + i_{(t)} \otimes g_{(t)} \\
y_{(t)} &= h_{(t)} = o_{(t)} \otimes tanh(C_{(t)})
\end{aligned}
\tag{4.2}
$$

Where:

$i_{(t)} \equiv Input\ gate$ $\quad$ $\mathrm{x}_{(t)} \equiv Input\ at\ time\ t$

$\mathrm{f}_{(t)} \equiv Forget\ gate$ $\quad$ $\mathrm{y}_{(t)} \equiv Output\ at\ time\ t$

$\mathrm{o}_{(t)} \equiv Output\ gate$ $\quad$ $\sigma \equiv Logistic\ activation$

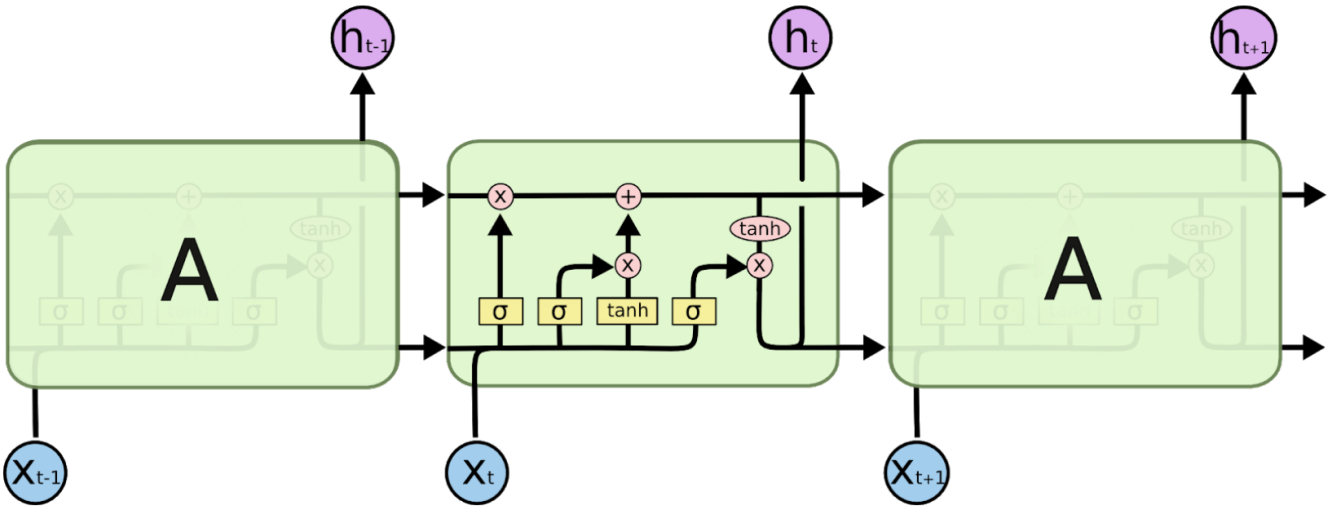
$\mathrm{g}_{(t)} \equiv Main\ layer$ $\quad$ $\tanh \equiv Hyperbolic\ tangent\ activation$

$\mathrm{C}_{(t)} \equiv Long-term\ state$ $\quad$ $\otimes \equiv Element\ wise\ multiplication$

$\mathrm{h}_{(t)} \equiv Short-term\ state$

$W_{x,i}^T, W_{x,f}^T, W_{x,o}^T, W_{x,g}^T \equiv Weights\ with\ respect\ to\ each\ layer$

Figure 4.3: Architecture of an LSTM cell.



Source: Google.

4.2.3 Hyper-Parameter Optimization

We decided to use Bayesian-Based hyper-parameter optimization instead of traditional approaches in the industry (GridSearchCV and Random Search). Bayesian-Based methods keep track of past evolution of the objective function then use a probabilistic model (Surrogate Function) which maps different hyper-parameter combinations to a probability score. This probability score called surrogate for the objective function, the selection function finds hyper-parameters that yields best probability scores on surrogate function and applies these hyper-parameters to the actual objective function, which is mean squared error in our case. We used HyperOpt library to apply Bayesian-based optimization, HyperOpt uses expected improvement as selection function and Tree of Parzen Estimator algorithm to optimize search space. We conducted 700 trials in Google Cloud Platform using double Nvidia Tesla P100 GPUs, and the process took around 200 hours. One can experiment with a broader search space and more trials if he/she has enough computing power. Table 4.1 provides our model setup which determined through Bayesian-based optimization.

Table 4.1: Model Setup

Time Steps	20
Number of Layers	2
Number of Neurons in each layer	200
Initializer	Xavier Uniform
Activation Function	Hyperbolic Tangent
Optimizer	Root Mean Square Propagation
Learning Rate	0.005
Loss Function	Mean Squared Error
Dropout rate	0.3
Gradient Clipping, Threshold	0.3
Batch size	180
Early stopping patience	40

4.3 Back-testing and Trading Methodology

For the back-testing period, we choose the period between October 2017 and June 2019, 400 trading days in total, we conduct back-testing in this specific period because all the previous data points included in the training and validation sub-samples. Testing model on the training and validation sub-sets will be bias and will not reflect the predictive ability of the model. For the trading strategy, we use a buy-hold-sell approach. Simply if the predicted mid-price of tomorrow is higher than today's closing price, we buy the stock at market close, if it is lower, then we sell the stock at market close. Otherwise, we hold our position. Note that we are predicting next day returns not the actual prices; Thus, we calculate the next day's mid-price through increasing today's mid-price by the amount of predicted return. In a real trading environment, one can implement more advanced strategies, for instance, using this methodology we are assuming we can only buy or sell a stock at the market close but in reality, one can buy or sell a stock at midday or market open. By designing a more sophisticated trading strategy, we can take advantage of the Machine Learning models even more. Simulation is conducted with Apple, Nvidia and SPDR ETF (SP-500), then we compared our strategy to a passive buy-hold strategy on the same data-set.

5 Results

The results section of the paper is divided into two parts: The first part will present the statistical results we obtained and the second part will focus on the results of the back-testing.

5.1 Statistical Results

Following our methodology, in each epoch, the model will try to minimize the Mean Square Error (MSE). In figure 5.1, we can see that the MSE values that our model obtains decrease with each iteration. Note that the training error is higher than the validation error. This is a rare case but it is not impossible, while we cannot know the reason for sure, most probably it is due to training set including periods hard to learn from such as the Savings and Loan crisis in 1986, the 1990 recession, the 2000 Dot-Com bubble, the 2007-2009 great recession and many more. These shocks have substantial long-term effects on financial markets, and unfortunately, we cannot predict those shocks by using technical indicators.

Figure 5.1: Mean Square Error Values in each Epoch.

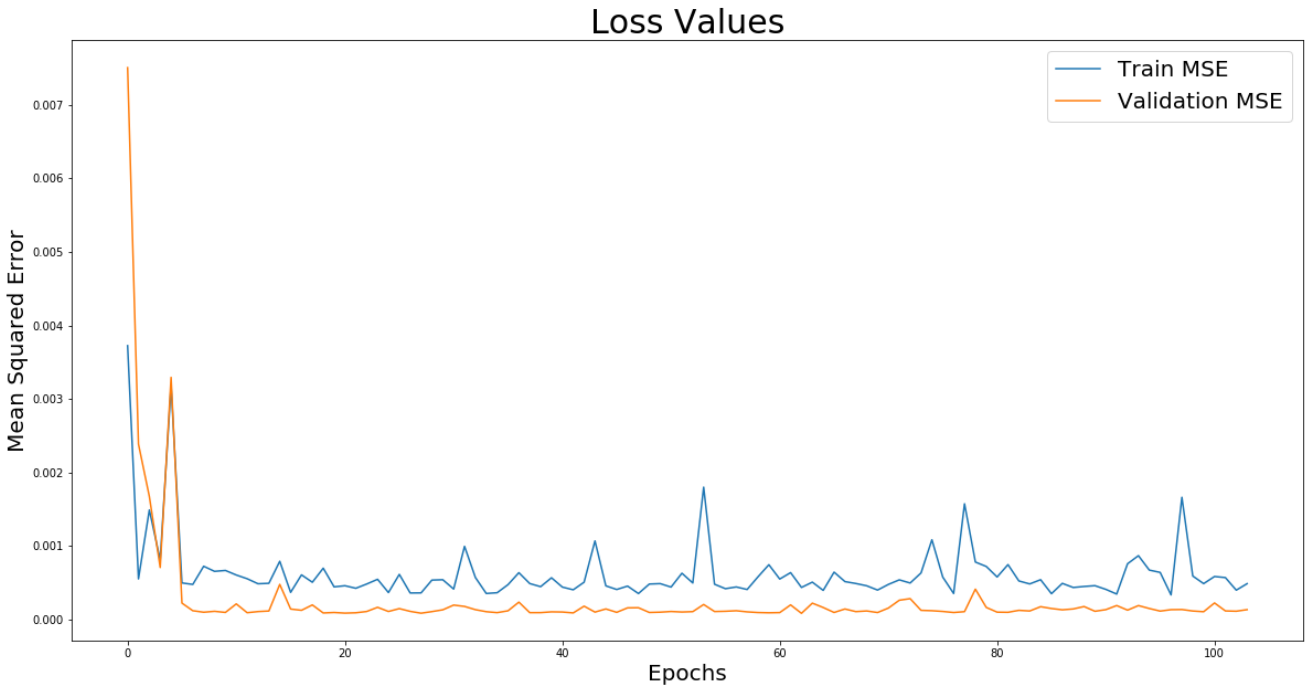


Table 5.1 presents the forecasting metrics for the test samples and table 5.2 provides directional scores for each data-set we used in this project. In order to provide clear comparison to the reader, we include the results of Support Vector Regression (SVR) as a benchmark model. Due to the reason, directional scores and statistical metrics for the SVR on Apple did not yield promising results, we did not test SVR on Nvidia and SPDR and we did not conduct back-testing according to predictions of SVR. Note that we did not optimize the hyper-parameters of SVR, we only include it for the sole purpose of comparison. Figures in A.1, A.2 and A.3 provide prediction plots of LSTM for each stock, figure A.4 provides the prediction plot of SVR on the test sample of Apple. Note that we trained our model on Apple and tested it on Nvidia and S&P-500 ETF. Our results prove that predictive ability of technical indicators is universal and models trained on a certain stocks can be applied to different stocks or indexes. One interesting approach would be transfer learning, we can pre-train a model on a certain stock, then we can freeze lower layers and train the pre-trained model on other stocks. By doing this way model can learn specific relationships from different stocks and take advantage of large amount of data, thus increase its generalization ability.

Table 5.1: Test Error Values.

Loss Values			
		MSE	MAE
LSTM	Apple	0.00013	0.00820
	Nvidia	0.00032	0.01310
	SPDR	0.000045	0.005068
SVR	Apple	0.00018	0.01
	Nvidia	-	-
	SPDR	-	-

Table 5.2: Directional Scores.

(a) LSTM - Apple

Output	Precision	Recall	F1-score
0	65%	62%	64%
1	71%	74%	72%
Macro Average	68%	68%	68%
Weighted Average	69%	69%	69%

(b) SVR - Apple

Output	Precision	Recall	F1-score
0	0%	0%	0%
1	56%	100%	72%
Macro Average	28%	50%	36%
Weighted Average	31%	56%	40%

(c) LSTM - Nvidia

Output	Precision	Recall	F1-score
0	66%	63%	64%
1	70%	72%	71%
Macro Average	68%	68%	68%
Weighted Average	68%	68%	68%

(d) LSTM - SPDR

Output	Precision	Recall	F1-score
0	56%	53%	54%
1	70%	72%	71%
Macro Average	63%	62%	63%
Weighted Average	64%	64%	64%

5.2 Back-testing Results

The trading strategy we constructed according to predictions of our model outperforms the buy-hold approach for each stock and SP-500 index. Table 5.3 provides a comparison of cumulative returns obtained by each strategy. One interesting point to mention here is 2018 is an eventful year in terms of political frictions and global economic

conditions. As mentioned before, these events have significant effects on financial markets. Our results prove that we can outperform the market with Deep Learning methodologies even in a time of economic uncertainty. Figures in A.5, A.6 and A.7 show the comparison of the cumulative returns between buy-hold-sell and buy-hold strategy. Buy and sell points determined by our model can also be found in the appendix section (Figures in A.8, A.9 and A.10).

Table 5.3: Comparison of Cumulative Returns.

	Cumulative Returns		
	Apple	Nvidia	SPDR
LSTM, Buy-Hold-Sell	34.45%	46.43%	28.24%
Buy-and-Hold	14.04%	-26.92%	10.36%

6 Conclusion

In this project, we experimented with LSTM networks to predict stock returns one day in the future and create a profitable trading strategy. Our results show that we can outperform passive buy-hold strategy in 400 trading days period. Note that, we did not take account of the transaction costs, and we did not test the model in a real-time trade environment. Thus we cannot use this model for live trading purposes without further tuning and a more reliable trading strategy. Our results showed that potential returns which we can obtain from Apple and Nvidia are higher than SPDR. The reason for that is diversification and volatility. Because of SP-500 composed of the largest 500 company in the US in terms of market capitalization, it is only exposed to market risk, and the market risk is more related to economic and sector fundamentals rather than investor behaviour. By using technical indicators, we can only capture inefficiencies caused by the market participants, and these inefficiencies are more common in individual stocks rather than diverse indexes like SP-500. Thus potential returns we can obtain from individual stocks are higher. As for every trading strategy, Machine Learning strategies are also subject to Risk-Return relationship, higher the volatility, higher the risk and higher the potential returns we can obtain.

7 Further improvements

With further time and computational power, we would potentially be able to further improve our results based on the methodologies specified in this paper. Further tuning of hyperparameters would potentially yield a model with lower prediction errors and more accurate predictions. While the model itself can be improved, we can also explore the following three main directions in which our methodology can be expanded.

Autoencoders and MLP

Instead of using specific technical indicators and LSTM networks, we can use Autoencoders to extract deep features from a very large number of technical indicators. Autoencoders is a specific type of Neural Networks model that can perform tasks such as dimension reduction or data denoising to a given set of data. One downside of Autoencoders is that data loses its time series characteristic, thus LSTM would not provide any advantage over MLP.

Fundamental and Sentiment Data

As mentioned in the paper, we cannot capture the fundamental information of the company by using technical indicators. Incorporating fundamental data such as accounting ratios can help the model to better infer the movement patterns in the stock index and make more accurate predictions. Implementing sentiment data would help us to capture investor behaviour to an even greater extent.

Reinforcement Learning

In order to use the model for live trading purposes, we can combine the LSTM network with Reinforcement learning and test the model in a live trading environment. A significant advantage of implementing reinforcement learning is that the agent can find the most optimal trading strategy without being instructed by the model builder. Another critical aspect of reinforcement learning is that model can learn on the fly (uninterruptedly) and can adapt to changing market conditions.

References

Bao, W., Yue, J., and Rao, Y. (2017). A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *PloS one*, 12(7):e0180944.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305.

Bergstra, J., Yamins, D., and Cox, D. D. (2013). Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in science conference*, pages 13–20. Citeseer.

Chande, T. S. and Kroll, S. (1994). *The new technical trader: boost your profit by plugging into the latest indicators*. John Wiley & Sons Inc.

Darrat, A. F. (1990). Stock returns, money, and fiscal deficits. *Journal of Financial and Quantitative Analysis*, 25(3):387–398.

Friedman, B. M., Laibson, D. I., and Minsky, H. P. (1989). Economic implications of extraordinary movements in stock prices. *Brookings Papers on Economic Activity*, 1989(2):137–189.

G. Malkiel, B. (2007). *A Random Walk Down Wall Street: The Time-Tested Strategy for Successful Investing*.

Gu, S., Kelly, B., and Xiu, D. (2018). Empirical asset pricing via machine learning. Technical report, National Bureau of Economic Research.

Hamao, Y., Masulis, R. W., and Ng, V. (1990). Correlations in price changes and volatility across international stock markets. *The review of financial studies*, 3(2):281–307.

Huang, W., Nakamori, Y., and Wang, S.-Y. (2005). Forecasting stock market movement direction with support vector machine. *Computers & Operations Research*, 32(10):2513–2522.

Jaffe, J., Keim, D. B., and Westerfield, R. (1989). Earnings yields, market values, and stock returns. *The Journal of Finance*, 44(1):135–148.

Kim, K.-J. (2003). Financial time series forecasting using support vector machines. *Neurocomputing*, 55(1-2):307–319.

Kim, S. and Kang, M. (2019). Financial series prediction using attention lstm. *arXiv preprint arXiv:1902.10877*.

Liu, T., Granger, C., and Heller, W. (1992). Using the correlation exponent to decide whether an economic series is chaotic. *Journal of Applied Econometrics*, 7(S1):S25–S39.

Malandri, L., Xing, F. Z., Orsenigo, C., Vercellis, C., and Cambria, E. (2018). Public mood–driven asset allocation: the importance of financial sentiment in portfolio management. *Cognitive Computation*, 10(6):1167–1176.

Malkiel, B. G. (2003). The efficient market hypothesis and its critics. *Journal of economic perspectives*, 17(1):59–82.

Malkiel, B. G. and Fama, E. F. (1970). Efficient capital markets: A review of theory and empirical work. *The journal of Finance*, 25(2):383–417.

Markowitz, H. (1952). Portfolio selection. *The journal of finance*, 7(1):77–91.

Shah, D., Campbell, W., and Zulkernine, F. H. (2018). A comparative study of lstm and dnn for stock market forecasting. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 4148–4155. IEEE.

Sharma, A. and Seth, N. (2012). Literature review of stock market integration: a global perspective. *Qualitative Research in Financial Markets*, 4(1):84–122.

Sheta, A. F., Ahmed, S. E. M., and Faris, H. (2015). A comparison between regression, artificial neural networks and support vector machines for predicting stock market index. *Soft Computing*, 7(8).

Tay, F. E. and Cao, L. (2002). Modified support vector machines in financial time series forecasting. *Neurocomputing*, 48(1-4):847–861.

Wilder, J. W. (1978). *New concepts in technical trading systems*. Trend Research.

Appendices

A Figures

Figure A.1: Actual returns and model predictions for Apple (2017-09 to 2018-11).

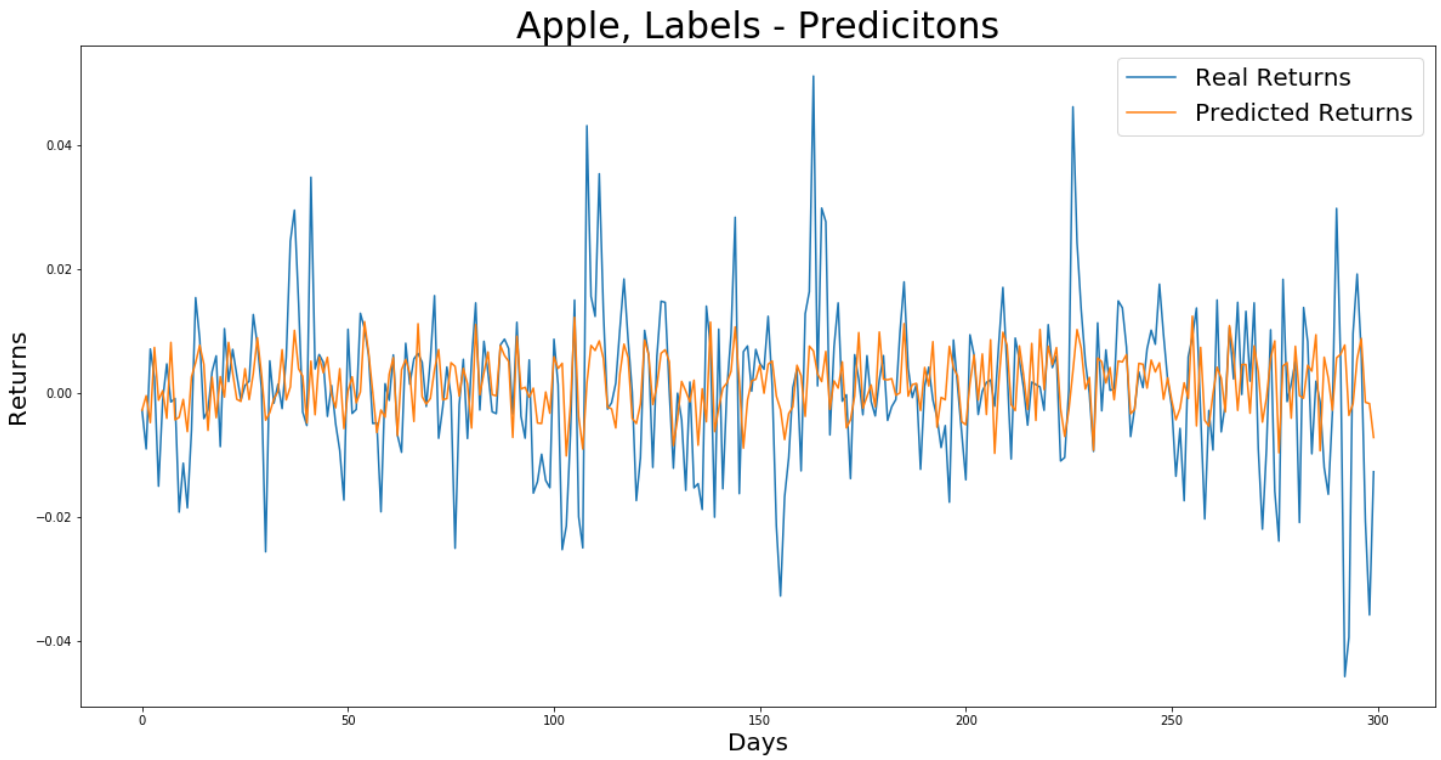


Figure A.2: Actual returns and model predictions for Nvidia (2017-09 to 2018-11).

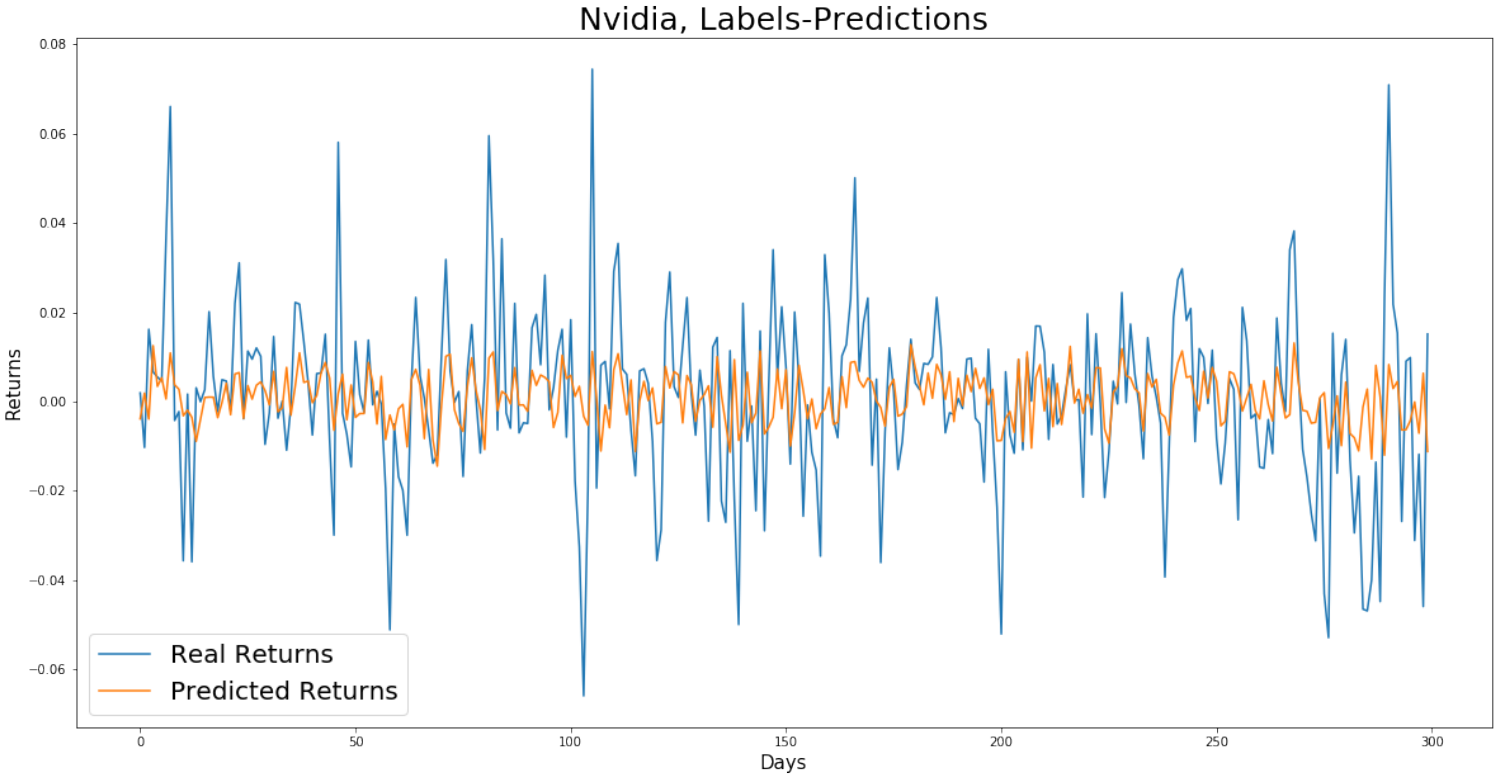


Figure A.3: Actual returns and model predictions for S&P500 (2017-09 to 2018-11).

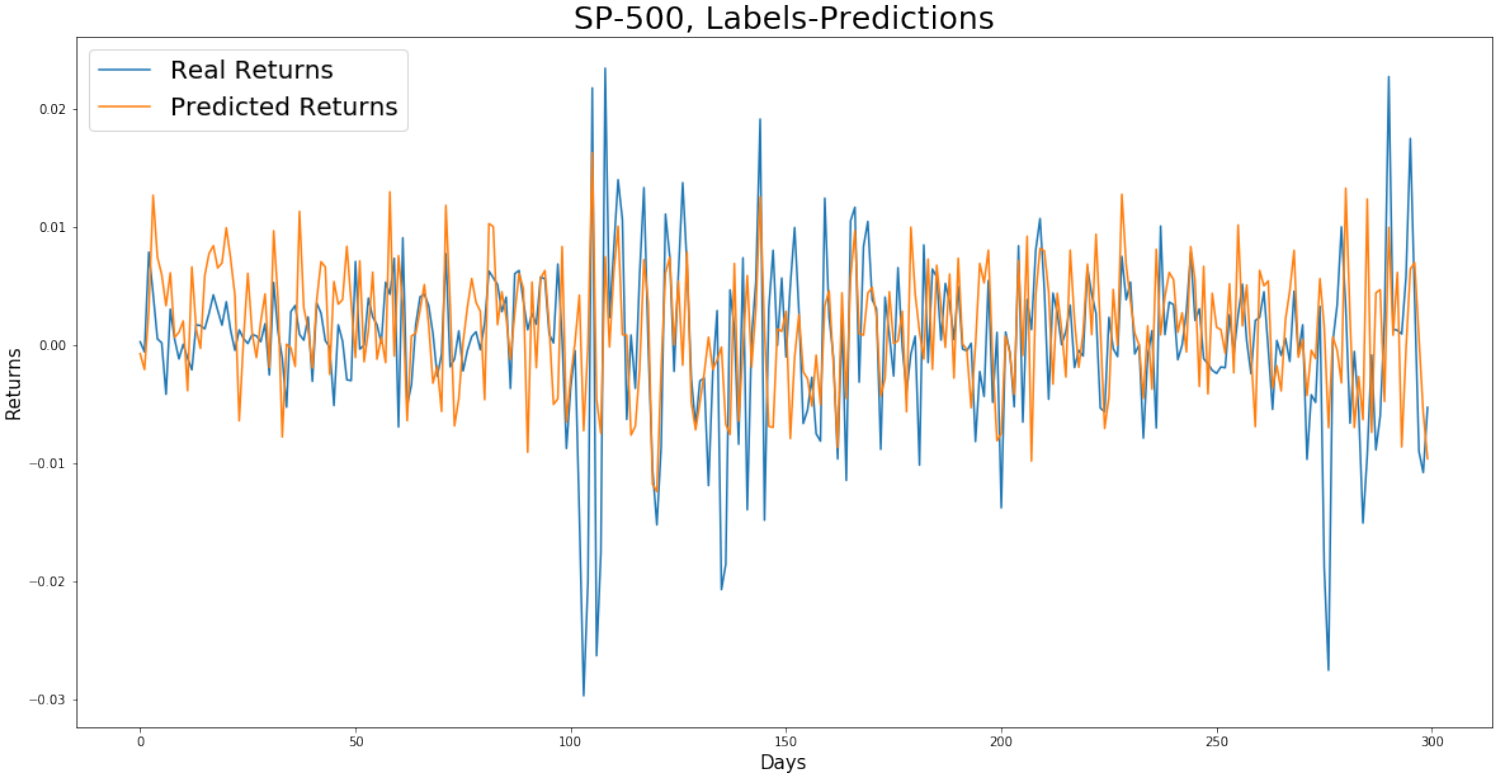


Figure A.4: Actual returns and model predictions for Apple-SVR (2017-09 to 2018-11).

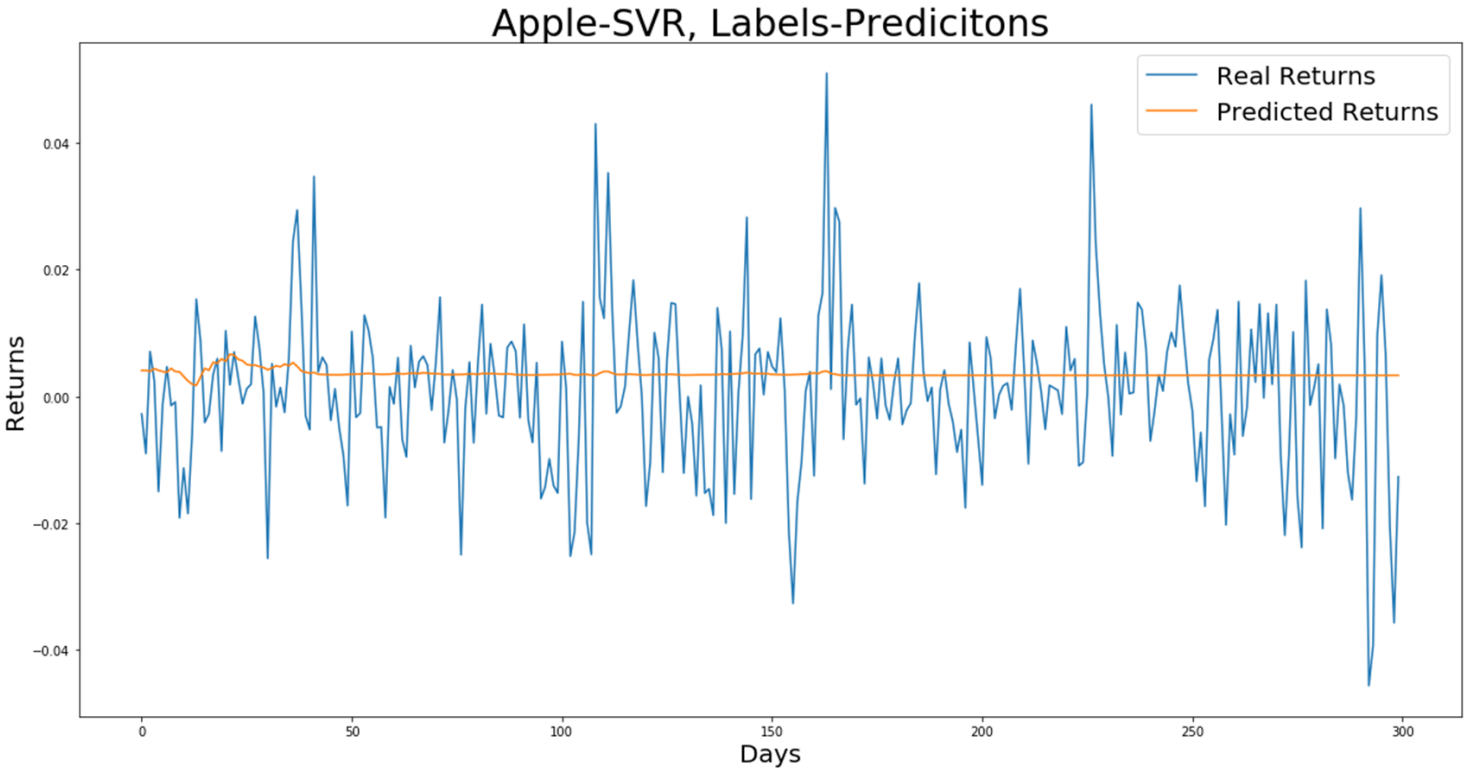


Figure A.5: Cumulative returns, Apple (2017-10 to 2019-06).

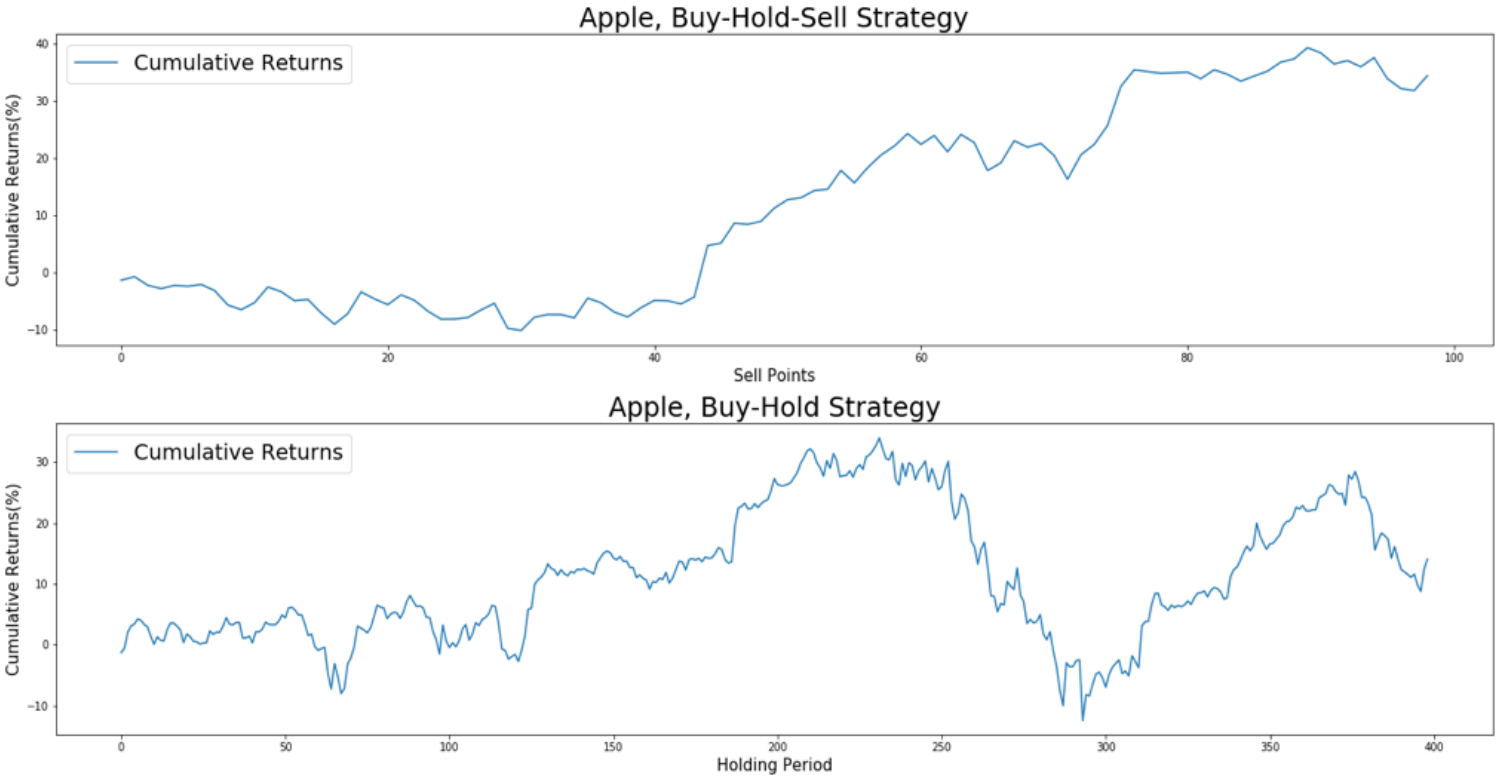


Figure A.6: Cumulative returns, Nvidia (2017-10 to 2019-06).

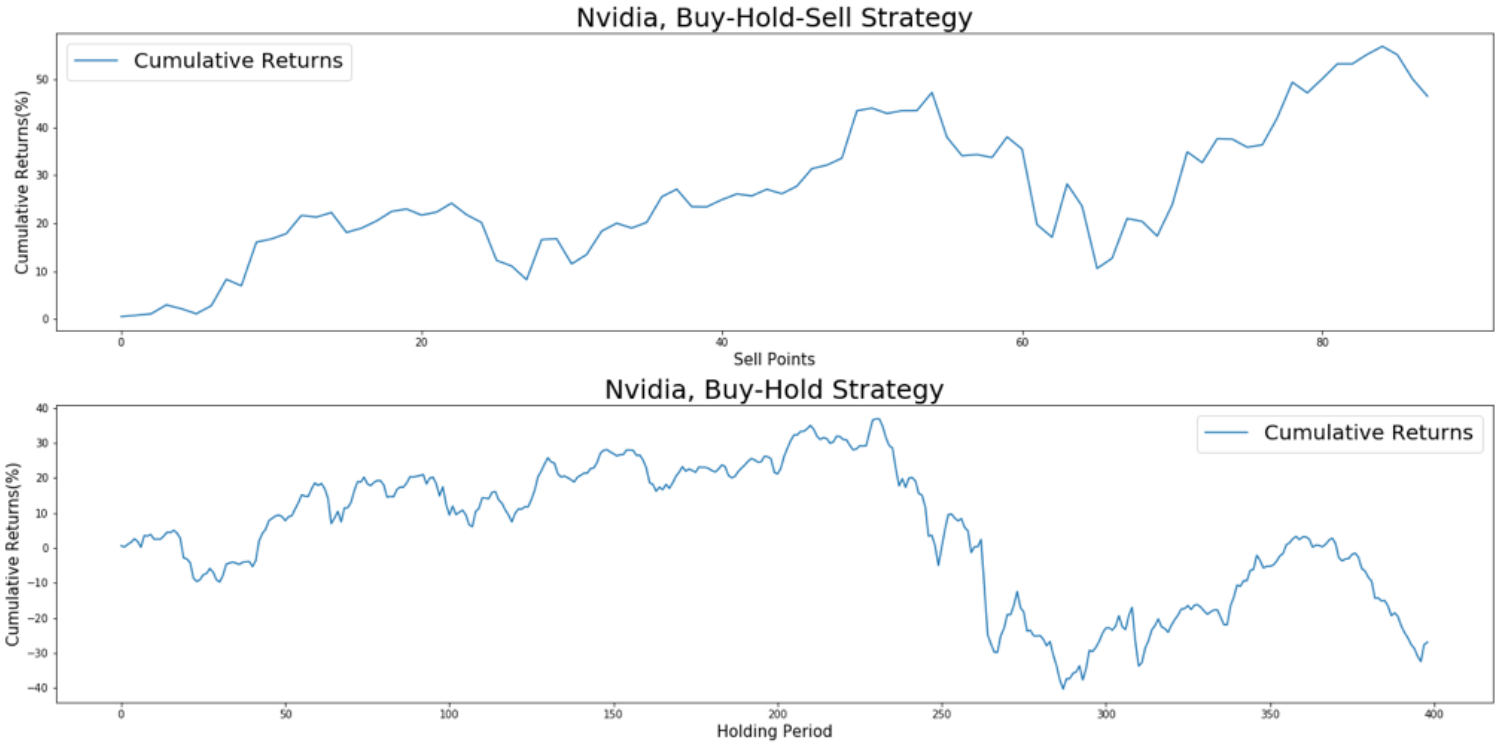


Figure A.7: Cumulative returns, S&P500 Index (2017-10 to 2019-06).

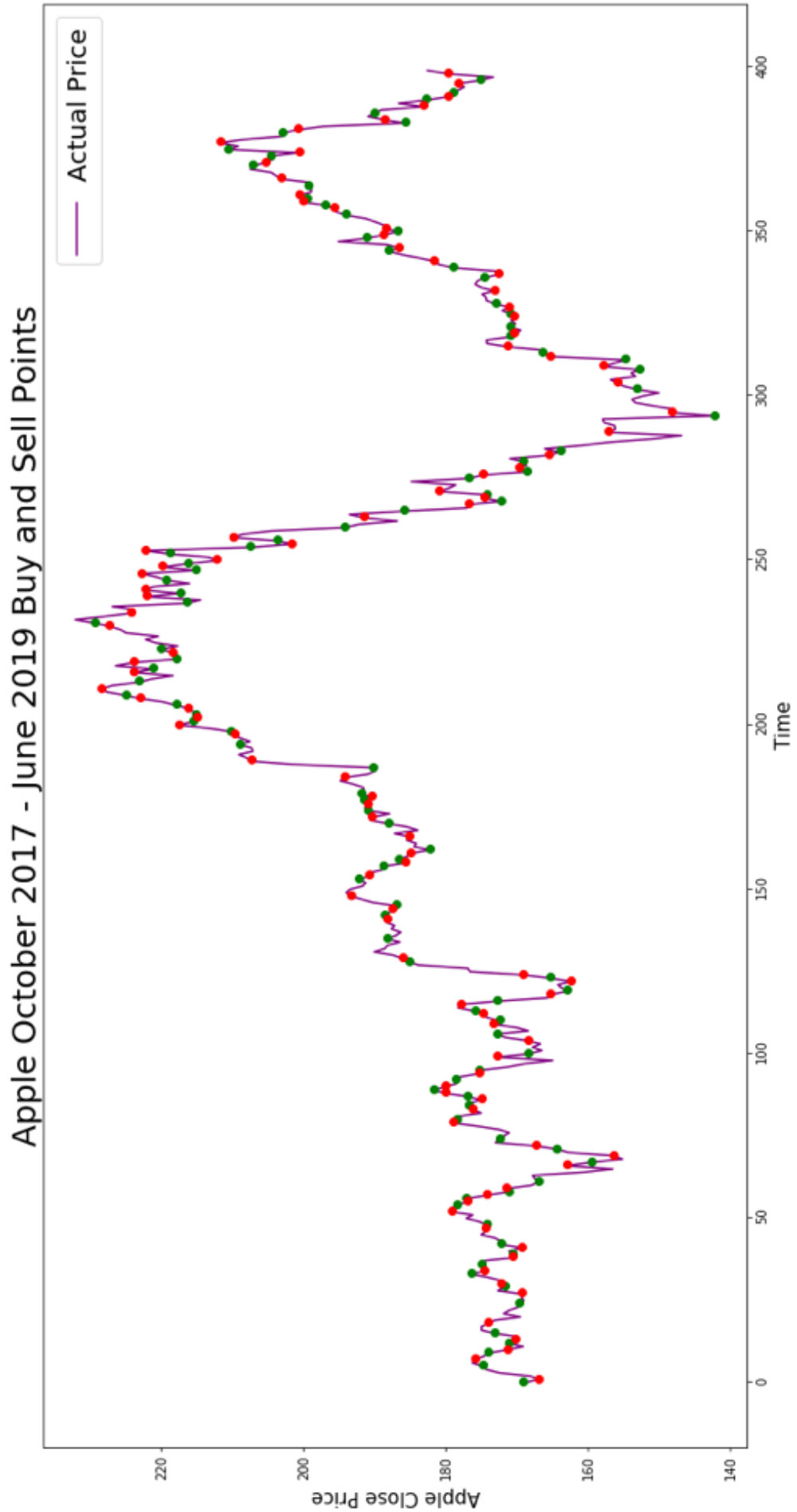


Figure A.8: Buy and sell points, Apple (2017-10 to 2019-06).

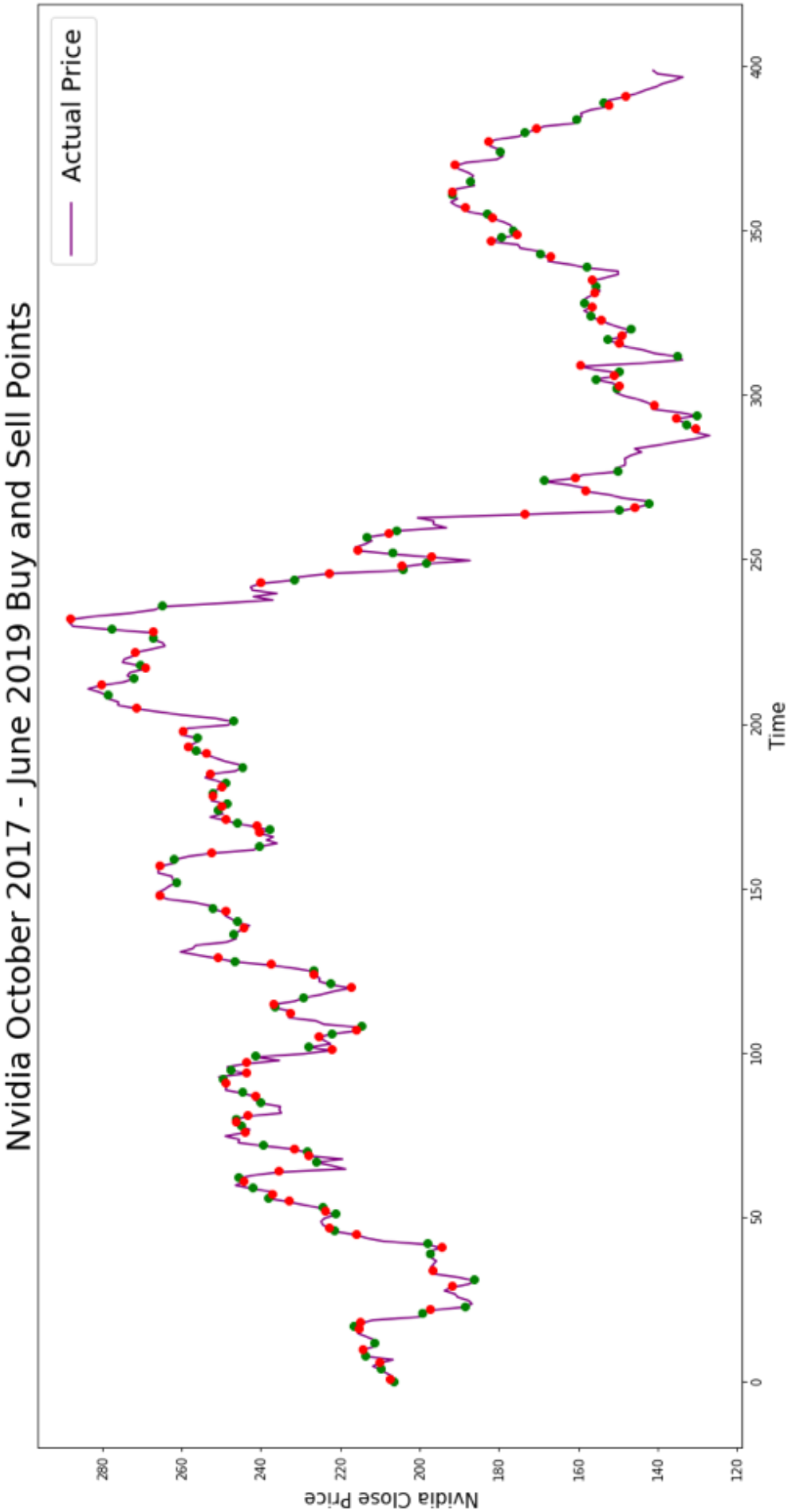


Figure A.9: Buy and sell points, Nvidia (2017-10 to 2019-06).

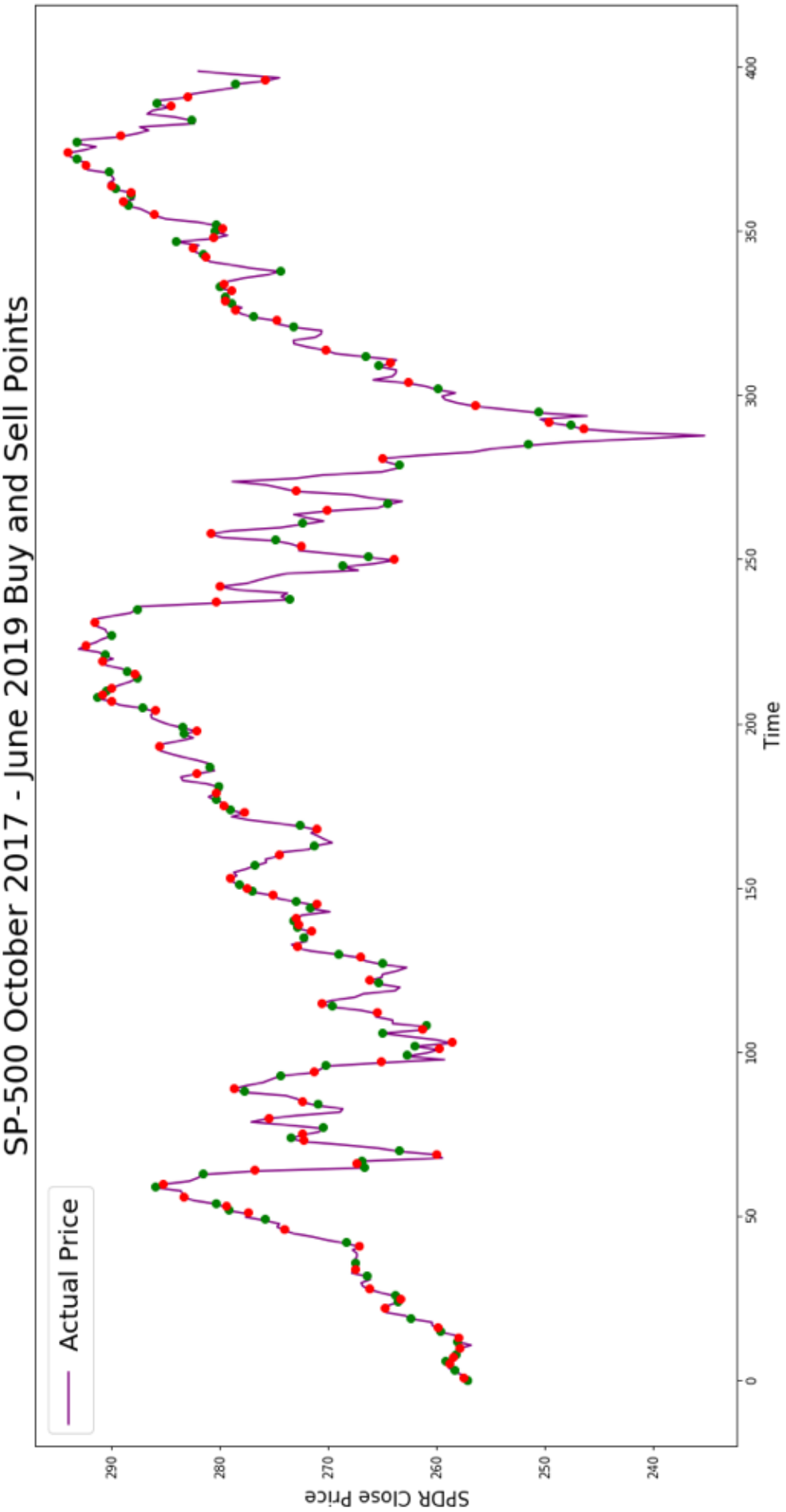


Figure A.10: Buy and sell points, S&P500 index (2017-10 to 2019-06).

B Technical indicators

1. Day High: Maximum price level reached by the security in a given trading day.

2. Day Low: Minimum price level reached by the security in a given trading day

3. Day Open: The price at which the security first trades upon the opening of an exchange on a trading day.

4. Day Close: The price at which the security last trades upon the closing of an exchange on a trading day.

5. Day Volume: The total number of times that a security was traded (bought or sold) during a given trading day.

6. Day Mid: Computed as the (Day Open + Day Close)/2 in any given trading day.

7. Day Return: Computed as the percentage change of the Day Mid prices from one day to the next.

8. Double Exponential Moving Average: The exponential moving average (EMA) makes a price transformation that intends to give more importance to the most recent lags of the moving average. This indicator smooths noise of the data but remains very reactive to changes in the stock index. The double exponential moving average (DEMA) gives even more importance to the most recent lags.

$$EMA_{N,t} = P_t \times \frac{2}{1+N} + EMA_{N,t-1} \times (1 - \frac{2}{1+N})$$
$$DEMA = 2 \times EMA - EMA(EMA)$$

   Where N is the number of periods in the smoothing period. In this case we use 30 days. P_t is the closing price at a given trading day t.

9. Kaufman Adaptive Moving Average: Attributed to Perry Kaufman, this indicator is based on a moving average that follows the price closely when the volatility of the stock price is low, but smooths the movements when the volatility is high. We are using a window of 30 days.

10. Triple Exponential Moving Average (TEMA): This indicator gives more importance to recent lags in the moving average, therefore it is more reactive to the stock index changes. We are using a window of 30 days.

$$TEMA = 3 \times EMA - 3 \times EMA(EMA) + EMA(EMA(EMA))$$

11. Weighted Moving Average (WMA): This moving average indicator uses another approach to give more importance to the most recent lags.

$$WMA = \frac{P_t \times N + P_{t-1} \times (N-1) + ... + P_{t-N-1} \times 1}{\sum_{k=1}^{N} k}$$

Where N is the number of days in the smoothing period. Here we are using 30 days.

12. Directional Movement Average Index (ADX): Part of the Directional movement system developed by Wilder (1978), this indicator measures the strength of positive or negative trend. We are using a time period of 14 days.

13. Directional Movement Average Index Rating (ADXR): This is a more smoothed version of the ADX. Similar to the ADX, it quantifies the level of momentum in a given trend. We are using a time period of 14 days.

14. Directional Movement Index (DMI): Widely used for trend trading strategies, this indicator measures the direction and strength of a trend. We use a time period of 14 days.

15. Absolute Price Oscillator (APO): This indicator is derived from the absolute value of the difference between two EMA. If the APO is positive, the market is considered to be bullish, if negative, it is considered to be bearish. We use 14 days for the short term EMA and 26 days for the long term EMA.

16. Aroon: This indicator measures the time passed between highs and the time passed between lows. It is used to identify changes in the trend and its strength. In general, a strong uptrend will regularly present new highs, and a strong downtrend will regularly present new lows. We use a time period of 14 days.

17. Aroon Oscillator: This indicator uses the Aroon indicator to measure the strength of the current trend. We use a time period of 14 days.

18. Chande Momentum Oscillator: Introduced by Chande and Kroll (1994), it measures the momentum of uptrend and downtrend days, to indicate if the stock is overbought or oversold. It follows the stock price more closely than other momentum indicators. We are using a time period of 14 days.

19. Money Flow Index: This indicator incorporates both price and volume to identify changes in trend, overbought or oversold conditions in an asset. We use a time period of 14 days.

20. Minus Directional Indicator: Part of the directional movement system developed by Wilder (1978), this indicator measures when a downward trend is present. We use a time period of 14 days.

21. Minus Directional Movement: This indicator measures the strength in absolute terms of the downward trend. We use a time period of 14 days.

22. Plus Directional Indicator: Part of the directional movement system developed by Wilder (1978), this indicator measures when a upward trend is present. We use a time period of 14 days.

23. Plus Directional Movement: This indicator measures the strength in absolute terms of the downward trend. We use a time period of 14 days.

24. Percentage Price Oscillator: This indicator compares the percentage difference between two moving averages, one long term and one short term. The differences measured in percentages allows for comparison over long periods of time. We use a period of 20 days for the long term moving average and a period of 10 days for the short term moving average.

25. Relative Strength Index (RSI): Introduced by Wilder (1978), this oscillator is widely used to measure the magnitude of recent price changes, from which we can infer whether an asset is overbought or oversold. We use a time period of 14 days.

26. Stochastic Oscillator: This indicator compares the closing price of an asset in a given day to its prices in a period of time in the past. It gives signals about an overbought or oversold situation. We use a time period of 14 days.

27. Moving Average Convergence/Divergence (MACD): This is a momentum indicator calculated from the difference between a long term and short term EMA. It is widely used to infer a overbought or oversold situation. We use a period of 26 days for the long term EMA and a period of 12 days for the short term EMA.

28. Commodity Channel Index: This indicator measures the difference between the current price and the historical average price. It shows the direction and strength of the trend, as well as whether the stock is overbought or oversold. We use a time period of 14 days.

29. Momentum Indicator: This indicator compares the current market price of an asset to a price of that asset from a given number of periods in the past. It is used to identify upward or downward trends. We include one momentum indicator using a time period of 10 days and another using a time period of 20 days.

30. Simple Moving Average (MA): This indicator is computed from the arithmetic average of the closing prices of each day in the past N days, where N is a fixed number of days. It can be used to smooth noise or identify trends. We use one MA of the past 10 days and another one of the past 20 days.

31. Rate of Change: This indicator is the percentage change of the closing price of a give day relative to the closing price of a day in the past: $(\frac{P_t}{P_{t-N}} - 1) \times 100$. It can be used to identify upward or downward trends. We use a time period of 10 days.

32. Ultimate Oscillator: This momentum indicator is computed from the weighted average of the price momentum of an asset in three time frames (7, 14 and 28 days). The shortest time frame is given most weight, whereas the longest is given least weight. It is used to identify upward or downward trends.

33. Williams' %R: This momentum indicator compares the closing price at a given day with the high-low price range over a period in the past. It is used to identify oversold or overbought situations. We use a time period of 14 days.

34. Balance of Power (BOP): This indicator measures the market strength of buyers against sellers by assessing the ability of each side to drive prices to an extreme level. It is computed as follows:

$$BOP = \frac{(Close\ price - Open\ price)}{(High\ price - Low\ price)}$$