The University of Hong Kong

Faculty of Engineering

Department of Computer Science

COMP7704
Dissertation Title

Financial Market Forecast with Deep Learning and Triple-Barrier Labeling Method

Submitted in partial fulfilment of the requirements for the admission to the degree of Master of Science in Computer Science

By
Chan Kin Wah, Kendrick
1996202925

Supervisor's title and name: Dr J. R. Zhang
Date of submission: 15/11/2019

Abstract

Financial markets forecast has been known as a notoriously hard problem. Yet, previous studies have attempted to resolve the problem with classification models in machine learning, especially with deep learning that enables the learning from raw market data. However, their results were probably overfitted and little variation in the classification labels has been considered. Meanwhile, a new labeling approach for such forecast models, the triple-barrier labeling method, was proposed recently but it seems no empirical research has been done on that. This research examines to what extent a useful market forecast model can be built with deep learning and the alternative labeling approach. The experiments involved daily market data of U.S. stocks including MSFT, AAPL, AMZN, GOOGL, and FB. Forecast models of both the traditional and new labels were trained and validated with data from 2012 to 2015 and then benchmarked with a control trading strategy for the period from 2016 to 2018. Results show that the strategy with the triple-barrier forecast model outperformed all other strategies considered, reporting a 65% increase in Sharpe ratio and a 51% decrease in maximum drawdown over the baseline buy-and-hold strategy.

Keywords: *Deep learning, RNN, Machine learning, Market forecast, Raw market data, Label engineering, Triple-barrier label, US stocks, Trading strategy, Backtrader*

Declaration

I declare that this dissertation represents my own work, except where due acknowledgement is made, and that it has not been previously included in a thesis dissertation or report submitted to this University or to any other institution for a degree, diploma or other qualifications.

Chan Kin Wah, Kendrick

15 November 2019

Acknowledgement

I would like to express my gratitude to my supervisor, Dr J.R. Zhang for his guidance and support in this dissertation, as well as his lecture on Techniques in Computational Finance, which has opened the door of quantitative finance to me. With Dr Zhang's expertise in both machine learning and finance, useful advice was always available. In addition, I would also like to thank the second examiner, Dr Zhiyi Huang, for his insightful questions.

Table of Contents

1. Introduction

Financial markets forecast has been known as notoriously hard because of the immensely low signal-to-noise ratio and non-stationary natures in the time series data involved. Moreover, the famous Efficient Markets Hypothesis (EMH) claims that current market prices reflect all past trading information [1], which implies ones cannot take advantages from historical market data. Yet, previous studies [2]-[4] on this topic have gradually provided evidence that the hypothesis might not always hold.

With the prevalence of machine learning, a technology that allows machines to learn a task without explicit programming [5], such research direction seemed to shift from traditional rule-based methods to this approach. A paradigm of trading systems enabled by such market forecasts contains two components: a forecast model and a trading strategy. The forecast model is a supervised learning model, most likely a classification model, in machine learning, with features (X) that capture the window of past N observations in the market, and predicts a label (Y) that indicates what would happen in the next K periods. The predictions are fed to a manually designed trading strategy as the entry or exit signals.

Traditionally, a popular approach to build such forecast models was to represent the features with technical indicators [6], which are arithmetic calculations based on historical prices and volumes, and train it with shallow learning algorithms such as, tree-based methods and Support Vector Machines. However, a challenge then arises regarding the need for pre-defined parameters, which could be hard to tune. Moreover, such indicators were developed by domain experts as early as the 60s based on simple arithmetic rules. While this might be a good idea at the beginning, financial markets are non-stationary and subject to change upon participants behaviour. When they are exploited increasingly, they may overcrowd the markets.

Recently, increasing focus has been drawn on a machine learning algorithm, called deep learning, especially with its profound successes in various applications, such as computer vision [7], speech recognition [8], and gaming [9], which enables the learning from raw data. Characterised by stacked layers of affine and non-linear transformations, the algorithm can lead to increasingly meaningful data representations being learned in successive layers [10]. It, therefore, is regarded as a means to extract useful features automatically for problems without hand-crafted features [11]. This is especially noteworthy as [12] suggests that reliance on that human expert knowledge could be

expensive, prone to human biases and limiting. Take the gaming sector as an example, robots trained with deep learning from raw board configurations has outperformed those requiring hand-crafted features and attained the superhuman performance by training of just a few weeks [9], [13].

A review of related literature shows that deep learning has been adopted in the market forecast models that learn solely from features of raw market data with minimal pre-processing [14]-[17]. However, their models seemed to overfit to the immensely small testing dataset and little variation in labels design was considered. Meanwhile, a new labeling technique, the Triple-Barrier method, was introduced by [18] recently, where the author suggested this method "makes more sense" but without further elaboration. To the best of my knowledge, there is no empirical research has been done this. We are, therefore, interested in how this new labeling method is compared to the traditional one. The purpose of this research is to examine to what extent a useful financial market forecast model can be built with the use of deep learning and the triple-barrier labeling method. To benchmark forecast models of different labels, a controlled trading strategy is adopted.

Throughout this dissertation, the terms model, forecast model and market forecast model, are used interchangeably to mean the classification model, that predicts the future outcomes of a financial market. Features, also known as independent variables or inputs, refers to the inputs of such forecast models; and the term label is used instead of target, output or independent variable. Moreover, the term strategy means the trading strategy and the trading system as a whole that makes trading decisions based on the forecast model it connects to or self-defined rules.

The rest of the dissertation is organized as follows. Chapter 2 reviews the literature related to applications of deep learning with raw market data in finance. Chapter 3 describes the general methodology common to all experiments in this research. Chapter 4 describes and discusses the experiment of deep learning forecast models with the traditional label. The best deep learning architecture and features options were thereafter chosen. Chapter 5 proposes the forecast model with the triple-barrier label and discusses the experiment results. A control trading strategy is designed and simulated to benchmark both forecast models. Finally, Chapter 6 concludes the dissertation with major implications and recommendations.

2. Literature Review

We hypothesize that financial markets can be forecasted with raw data with minimal pre-processing using deep learning technology. In connection to this, various previous related studies that learned solely from raw market data in the finance sector were selected. Their applications include price pattern recognition [19], [20], market forecast for pair trading [21], [22], market forecast for delta trading [14]-[17], automatic trading robots [23], and stocks clustering [24], as summarised in Table 2.1.

Regarding the methodologies, the selected literature relied only on raw market data, including Open, High, Low, Close, and Volume (OHLCV) of various timeframe, as given in Table 2.2. For the pre-processing procedures, [21], [22] normalized the price data cross-sectionally, because their objective was to compare the relative strength among a portfolio of stocks; [16], [17], [19], [20], and [23] normalized the OHLCV variables among the windows. Window size of 20 bars [14], [15], [21], [24] and 30 bars [16], [17], [20] seemed to be popular choices. [14], [15], and [24] created features by rendering the candlestick price chat into images and perform image classification. These indicate learning from raw market data might be a researchable area in the future, but our research focuses on raw market data with normalizations only.

Table 2.1: Applications of deep learning based on raw market data

Ref	Title	Author	Year
Task: Price Pattern Recognition			
[19]	Stock price pattern recognition: A recurrent neural network approach	Kamijo & Tanigawa	1990
[20]	Stock Chart Pattern recognition with Deep Learning	Velay & Daniel	2018
Task: Forecast for Pair Trading			
[21]	Applying deep learning to enhance momentum trading strategies in stocks	Takeuchi & Lee	2013
[22]	Deep learning with long short-term memory networks for financial market predictions	Fischer & Krauss	2018
Task: Forecast for Delta Trading			
[14]	Financial time-series data analysis using deep convolutional neural networks	Chen, Chen & Huang	2016
[15]	Deep Candlestick Predictor: A Framework Toward Forecasting the Price Movement from Candlestick Charts	Guo, Hsu & Hung	2018
[16]	Recurrent Neural Networks Approach to the Financial Forecast of Google Assets	Di Persio & Honchar	2017
[17]	Neural networks for algorithmic trading. Multivariate time series	Honchar	2017
Task: Automatic Trading Robots			
[23]	Financial Trading as a Game: A Deep Reinforcement Learning Approach	Huang	2018
Task: Stocks Clustering			
[24]	Deep Stock Representation Learning: From Candlestick Charts to Investment Decisions	Hu et al.	2018

Table 2.2: Dataset and features involved in learning from raw market data

Ref	Dataset	Features
[19]	1,152 stocks from Tokyo, 1W bar HLC	Window size 13 to 35, returns from exponentially smoothed C, normalized HL.
[20]	Alphabet C stock, 1-min bar OHLCV	Window size 30, normalized.
[21]	3,282 stocks from NYSE, AMEX, and NASDAQ	Monthly cumulative returns of past 12 months and past 20 days normalized into cross-sectional z-scores.
[22]	S&P 500 stocks	daily returns of the past 240 days normalized into cross-sectional z-scores.
[14]	Taiwan index futures, 1-min OHLC	Window size 20, transformed into 2D images [20x20].
[15]	6 TW index futures, daily OHLC	Window size 20, series of 576D latent features learned from 3 candlesticks rendered into 48x48 greyscale images.
[16]	GOOGL stock, 1D bar OHLCV	Window size 30, rescaled to [-1, 1] within the window.
[17]	Apple stock, daily OHLCV	Window size 30, standard normalized within windows.
[23]	12 currency pairs, 15-min bars	8 most recent log-returns and volumes normalized into z-scores among 96 bars.
[24]	FTSE 100 stocks, daily OHLC	Window size 20, rendered into 224x224 candlestick images.

Various types of machines learning tasks and deep learning architectures have been considered, as given in Table 2.3. While [23] was a reinforcement learning task and [24] was clustering, classification [19]-[22], [14]-[17] remained as the most popular task among the selected papers. Among them, unsupervised learning, such as Restricted Boltzmann Machine (RBM) [21] and Autoencoder (AE) [15], [24] were applied to extract latent features from the raw data; the others learned directly from the pre-processed raw data. Recurrent Neural Network (RNN) is thought as the most suitable network architecture for time-series data such as the stock prices, basic RNN [19], [16] and Long Short-Term Memory (LSTM) [20], [22], [23], [16] were both applied to these problems. Meanwhile, Convolution Neural Network (CNN) of 2-dimension form [20], [14], [15] and 1-dimension form [20], [15], [17] were also applied. This architecture has excelled in learning localized spatial structures from raw data, such as images, and stock price data is believed to share the characteristic of spatial structures too. Imagine if a price chart is shuffled along the time axis, it would look weird and its implications are lost. With this argument, our research is motivated to build a classification model for market forecasts with the 1-D CNN architecture as well.

Table 2.3: Techniques involved in learning from raw market data

Ref	ML Task	Deep Neural Network Architecture
[19]	Binary Classification	RNN [64, 24]
[20]	Binary Classification	1D-CNN, 2D-CNN (AlexNet), LSTM [10]
[21]	Binary Classification	Stacked RBM [40, 4] + DNN [50, 2] fine tuning
[22]	Binary Classification	LSTM [4]
[14]	Multiclass Classification	2D CNN
[16]	Binary Classification	2-layer RNN, LSTM & GRU
[17]	Binary Classification	CNN [16-8-64-2]
[15]	Binary Classification	2D CNN-AE [576D] + 1D CNN […64-1]
[23]	Reinforcement Learning	Dense [256, 256] + LSTM [256]
[24]	Clustering	CNN-AE [VGG16 + 512D + 7-layer CNN Decoder]

To the problem of market forecasts for delta trading strategies, which we tackle in this research, the related literature [14]-[17] is summarized in Table 2.4. They seemed to have achieved impressive results in term of classification, with "testing" accuracies of 72% [16], 69% [15] and 65% [17] reported. However, the problem remains unsolved, because no coherent train-valid-test data separation procedure was considered in these studies. Therefore, these models were skeptically overfitted to the "testing" dataset, and not generalizable to other unseen data. Moreover, most of them were tested only a single asset and tested for just a few months [16], [17], which results were likely to be too sensitive. For examples, a single sample being classified correctly or not could have

increased or decreased the accuracy by 2% already. In connection to this, this research is motivated to evaluate the forecast models in a fairer manner, by conducting a formal train-valid-test data separation, including five stocks to check their generalizability across multiple assets, as well as for an extended period of 3 years.

Table 2.4: Forecast models for delta trading strategies

Ref	Label	Train / Test Periods	Accuracy
[14]	Whether rise, fair or fall at the next 7th bar	Jan 2001 - Apr 2015; Train 4 yrs; Test 1 yr	53.76%
[16]	Whether the next 5th day's closing price goes up, or down	Jan 2012 - Dec 2016; 10% testing data	72%
[17]	Whether the next day grows up or down	Jan 2012 - Dec 2015; 10% test set	65%
[15]	Whether the next day grows up or down	Jul 1998 - Dec 2016; 10% testing data	69.11%

On the other hand, the classification labels in [14]-[17] are all very similar, as summarized in Table 2.4, they were formulated as follow: whether the next K day price goes up or down, with some predicting fair as well. Although [14] empirically finds the optimal value of K is about 1 to 5, these studies show innovation in labeling is lacking

for such forecast models. [18] proposes the triple-barrier labeling method and claims it "makes more sense", but without further elaboration. With this regard, this research conducts an empirical study to compare how models this new labeling method perform relative to the traditional one.

In conclusion, supported by the selected studies, learning from raw market data with deep learning seems possible. To the problem of market forecasts for delta trading strategies, train-valid-test data separation was often ignored, leading to their forecast performance probably overestimated. This research is, thus, motivated to evaluate the market forecasts models in a fair manner, by testing their generalizability across multiple assets and over a longer test period. It is also expected to examine, to what extent, the more innovative triple-barrier labeling method could benefit to the market forecast problem.

3. General Methodology

This research aims to build market forecast models that solely learn from raw market data with deep learning technology. In order to benchmark the models, there could be two approaches. One is to evaluate the model in terms of classification metrics, such as accuracy and precision, and compare with a model of random guesses, as the problem was formulated into a classification problem. The other is to implement a trading strategy that takes signals from such forecast models and then compare its trading performances to some naive strategy, such as buy-and-hold, through simulation.

This research proceeded with two stages. The first stage replicated some prior studies related to deep learning forecasts with raw market data features and explored if there are potential improvements. Such prior studies were typically classification models with the traditional label of bi-direction, and the validation process omitted. The second stage was motivated to improve those traditional forecasting models with an alternative labeling strategy, the triple-barrier labeling method. The models were benchmarked with a control trading strategy that takes signals from different forecast models, as well as other baseline strategies.

Details of the two studies are presented in the next two chapters, while this chapter focuses on describing the common methodologies applied, including data, tools, and evaluation methods.

3.1 Data Preparations

We considered the top five stocks listed in the U.S. market, including Microsoft (MSFT), Apple (AAPL), Amazon (AMZN), Alphabet (GOOGL) and Facebook (FB). They were the most liquid stocks in the world such that we may ignore the transaction costs issues. Moreover, the five stocks shared the same market and the same sector (technology), which justify their inclusion in a common model due to their correlations.

Pre-processing

Daily stocks price and volume data of the five stocks ranging from Jan 2012 to Dec 2018 were downloaded from Yahoo! Finance. Stocks data was stored in five CSV files individually, each possessed the following fields: Date, Symbol, Open, High, Low, Close, Adj Close, and Volume.

The Adj Close field stored the closing price adjusted for splits and dividends, but the

other price fields such as Open, High, and Low were not adjusted. Therefore, we updated them accordingly. Take the Open price as an example, the update rule (3.1) was applied. And the Close Price was directly replaced with Adj Close.

$$\text{Open} := \text{Open} \times \frac{\text{Adj Close}}{\text{Close}} \tag{3.1}$$

In each trading day, the Open, High, Low, Close, and Volume together formed a Bar record - the basic unit of sequential data in technical trading terminology. The Bars data was also tested against missing and anomaly, such as Low being greater than High. Fortunately, because they were the most well-known in the world, the data quality was good and there were no such issues.

Data Separation

In machine learning, model performance depends on how well the model generalizes to unseen data. In connection to this, the seven-year stock dataset was separated into three portions, namely training, validation, and testing dataset, according to the corresponding process of machine learning associated with. Training is the process that fits parameters to models from observed data; Validation is that of selecting the optimal model structure and hyper-parameters based the results of trained models; While testing

is to evaluate the generalizability of models for data outside training and validation. Data separation guarantees datasets remain unseen for the latter processes. Table 3.1 summarises how the three datasets were separated.

Table 3.1: Data Separation

Dataset	Period	Sample Size
Training	Jan 2012 – Jun 2015	4,265
Validation	Jul 2015 – Dec 2015	640
Testing	Jan 2016 – Dec 2018	3,770

Figure (3.1) to (3.5) highlights the three segments of stock price data separated for each for the stocks we study. As can be seen from the figure, most of the periods were bullish, the forecasting model should be good at learning the upward trend. However, the turmoil after the second half of 2018 might create a challenge for the models.

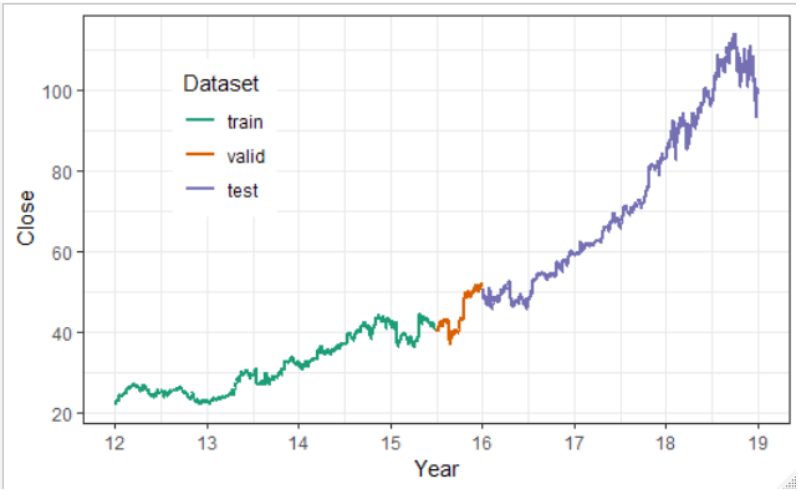

Figure 3.1: Stock price data from MSFT

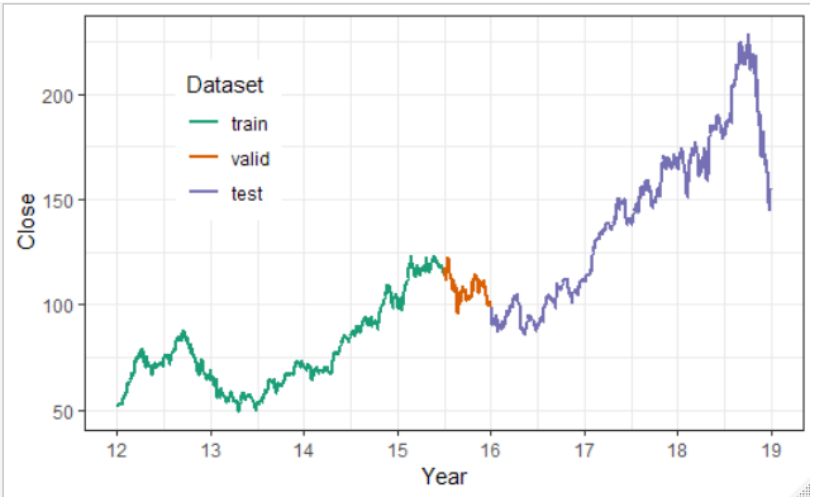

Figure 3.2: Stock price data from AAPL

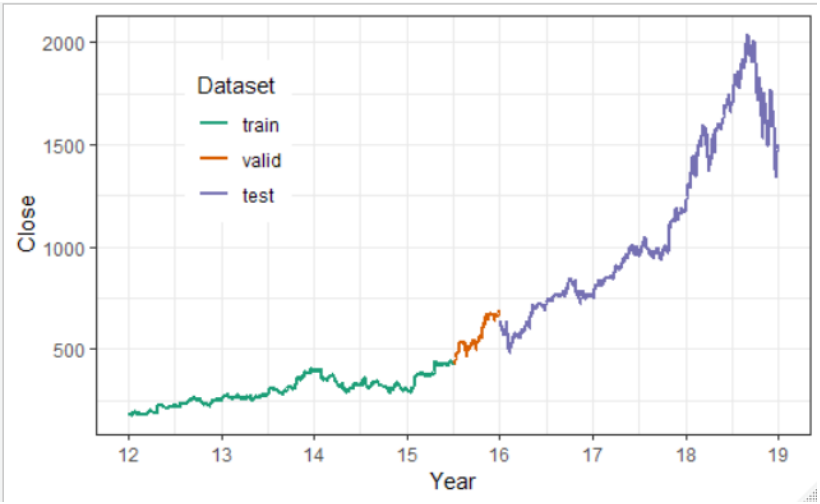

Figure 3.3: Stock price data from AMZN

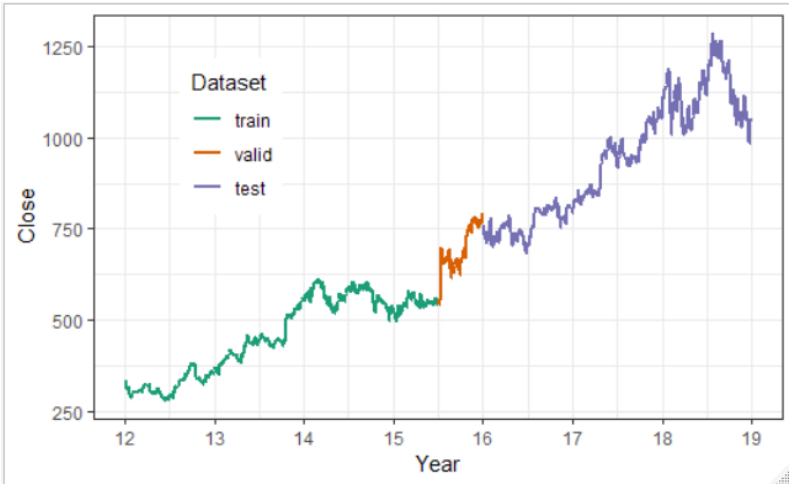

Figure 3.4: Stock price data from GOOGL

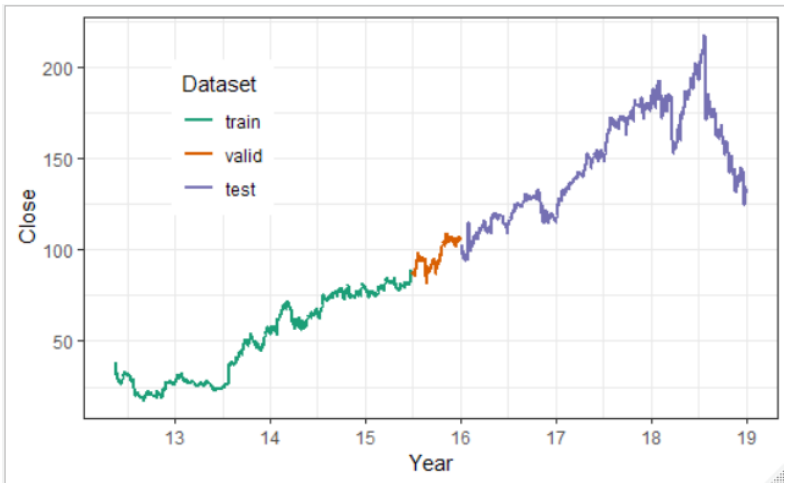

Figure 3.5: Stock price data from FB

Leakage Prevention

Data leakage is a well-known phenomenon in data science competitions, such as those hosted by Kaggle. This could be a technique to exploit information that is not supposed to be available in predictions. Those models trained with data leakage are rated high in competitions but perform poorly in real predictions, that it needs to be prevented. In our case, our models predict a label of future outcomes based on the features from the past.

Such future data used in labels generation is unknown in real predictions, therefore it should be removed in preparing features for training.

3.2 Performance Evaluation

The market forecast models studied was classification models. Therefore, they were evaluated with classification metrics. On the other hand, they were also designed to work in trading systems with two phases. In the first phase, predictions were generated from such classification models. In the second phase, the predications were treated as signals to issue buying or selling orders in trading strategies. In connection with this, the forecast models were alternatively evaluated according to their performance in trading strategies as well.

Classification Models

Regarding evaluation metrics for classification models, a simpler case of binary classification is first discussed, the variation for multi classes follows by. For both kinds of classification, the first evaluation task is to construct the confusion matrix that summarises the number of labels correctly predicted with regard to the actual labels. A sample confusion matrix for binary classification is given in Table 3.2 for an example.

Table 3.2: A sample confusion matrix for binary classification

		Predict	
		False (0)	**True (1)**
Actual	**False (0)**	TN	FP
	True (1)	FN	TP

These are the meanings for the abbreviated symbols:

- **True Negative (TN)** – the number of samples which actual label is False (0) and predicted correctly

- **False Negative (FN)** – the number of samples which actual label is True (1) but predicted as False (0)

- **True Positive (TP)** – the number of samples which actual label is True (1) and predicted correctly

- **False Positive (FP)** – the number of samples which actual label is False (0) and predicted as True (1)

After the confusion matrix is constructed, we may compute classification metrics including accuracy, precision and recall from the table. The metrics are given by the following formulas.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{3.2}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3.3}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.4}$$

If not specified otherwise, the basis class for precision and recall is assumed to be the positive (1) class. As the above formulas suggest, accuracy is the number of correct predictions among all samples; precision is the number of correctly prediction positive samples among all positive samples; recall calculates the number positive samples captured by the prediction. Accuracy is usually maximized when the loss function of models is minimized. The metric is used when we have no bias in both labels and expect the model to perform equally well for predictions of both labels.

However, if one label is more important than the other one in the problem, then precision and recall have to be considered. It depends on the consequence of the label is wrong predicted or it was not captured by the prediction to determine which metric to use. For examples, if our forecast model is to predict the time when the market is abnormal, and consequently without the orders in a market-making strategy. Its

consequence is more severe if the positive label is not captured. In this case, the recall metric should be utilized. While in our case, we design the forecast model for active trading. In general, we have no consequence if there is a profitable chance but we did not capture. But it is definitely unfavourable if the model predicted the chance but it turns out to be losing. Therefore, we choose the precision metric.

The confusion matrices for multi-class classification are constructed similarly but expands to NxN dimension, where N is the number of classes. Table 3.3 illustrates the one for 3-class classification. Its values are represented by symbols of two characters. Take AB as an example, it means the number of samples which actual class is A but predicted as B.

Table 3.3: A sample confusion matrix for 3-class classification

		Predict		
		A	**B**	**C**
Actual	**A**	AA	AB	AC
	B	BA	BB	BC
	C	CA	CB	CC

The accuracy for multi-class classification is calculated similarly, that is the total

number of corrected predicted samples over the total number of samples. However, for precision and recall, the base class must be specified. Suppose class C is the base class, the precision and recall for the example in Table 3.3 are given as follows.

$$\text{Precision(C)} = \frac{\text{CC}}{\text{AC+ BC + CC}} \tag{3.4}$$

$$\text{Recall(C)} = \frac{\text{CC}}{\text{CA + CB + CC}} \tag{3.5}$$

Trading Strategies

Professional trading strategies are usually tied up with multiple objectives. Besides maximizing the profits, which is obvious, they are also expected to incur low risks, grow steadily in capitals, and scale with increased capitals (although scalability is not studied in the research). In respect to this, there are many performance metrics to measure their achievements in these aspects.

Among these performance dimensions, the risk is regarded as the most prominent because it relates to the maximum loss that could incur in the worst scenario. Value at Risk (VaR) and maximum drawdown (MDD) are two common approaches in measuring the risk and MDD was chosen. A drawdown means the decrease in portfolio

value (PV) after a peak is made and it is expressed in percentage as given in Formula (3.6). As its name suggests, MDD measures the maximum drawdown that occurred in a historical investment period being studied and represents how much loss in a roll that investment could incur in the worst case if history repeats itself. The risk could be the most important performance aspect because it controls the extern of leverage that could apply to a strategy. Moreover, if a drawdown is too high, it might force the portfolio to stop-loss, even though the price may surge afterwards.

$$\text{Drawdown} = \frac{PV_{peak} - PV_{low}}{PV_{peak}} \tag{3.5}$$

Stability is the next important dimension in evaluating a trading strategy. It can be expressed as the Sharpe ratio as given in Formula (3.6). It can be thought of as the annualized returns (μ_A) normalized by the annualized volatility (σ_A). The risk-free rate term (r_f) there is insignificant because it is just considered as a constant over the investigation period. Generally, the higher the returns or the lower the volatility gives a higher value in this metric. Opposite to risk, stability goes down usually when the risk goes up. However, instead of measuring the worst moment one-off, stability is measured continuously. A strategy with stable growth guarantees consistent returns, no matter when the portfolio starts and profits are taken.

$$\text{Sharpe Ratio} = \frac{\mu_A - r_f}{\sigma_A} \tag{3.6}$$

Finally, profitability is measured in annual return, which is the rate of change of portfolio value (PV) during an investment period adjusted to the yearly representation, such that it is comparable to investments of different lengths. This metric is given in Formula (3.7), where N is the length of the investment expressed in years.

$$\text{Annual Return} = \sqrt[N]{\frac{PV_{end}}{PV_{start}}} - 1 \tag{3.7}$$

Alternatively, profitability can also be measured in risk-adjusted returns, as given in Formula (3.8). This captures how profitable a strategy is when maximum leverage allowed by MDD is applied. Although the maximum leverage may not be usually applied, this metric gives a fair judgement in profitability regarding its risk associated.

$$\text{Risk Adj Reutrn} = \frac{\text{Annual Return}}{\text{MDD}} \tag{3.8}$$

Besides the performance metrics discussed above, we may also investigate the quality of trades that induces such performance. It can be reflected in trades statistics including winning rate and profit-to-loss ratio, which are utilized in this research.

The winning rate captures the chances of winning per trade and is given in Formula (3.9), where the won trades are regarded as those resulted in positive returns and the loss trades are those otherwise. This figure is expected to be increased by raising the precision of forecast models associated with the strategies.

$$\text{Winning Rate} = \frac{\text{\# of Trades Won}}{\text{Total \# of Trades}} \tag{3.9}$$

The profit-to-loss ratio, on the other hand, captures the average profits of winning trades with regards to the average loss from losing trades, as given in Formulas (3.10) to (3.12). It is believed to be indirectly related to the reward-to-risk ratio. What difference between the reward-to-risk ratio and the profit-to-loss ratio is that the former refers to the target set up by the strategy, while the latter is the actual trading results.

$$\text{Avg Profit} = \frac{\text{Sum of Profits}}{\text{\# of Trades Won}} \tag{3.10}$$

$$\text{Avg Loss} = \frac{\text{Sum of Losses}}{\text{\# of Trades Lost}} \tag{3.11}$$

$$\text{Profit/Loss Ratio} = \frac{\text{Avg Profit}}{\text{Avg Loss}} \tag{3.12}$$

To understand how these two figures affect the performance in a trading strategy, the expected return equation as given in (3.13) is investigated, where W is the probability of winning, which can be approximated by the winning rate.

$$\mathrm{E[Return]} = W \times E[Profit] - (1 - W) \times E[Loss] \quad (3.13)$$

Dividing both sides by the expected loss, we obtain the following equation.

$$\frac{\mathrm{E[Return]}}{E[Loss]} = W \times \left(1 + \frac{\mathrm{E[Profit]}}{E[Loss]}\right) - 1 \quad (3.14)$$

The left-hand side in (3.14) is just another measurement for the risk-adjusted returns. On the right-hand side, the term of expected profit over expected loss is well approximated by the profit-to-loss ratio. Therefore, if a combination of winning rate and profit-to-loss ratio gives a positive value on the right-hand side, then a positive return is expected (because the dominator on the left-hand side must be positive). Moreover, the higher this value is, the higher risk-adjusted return can be expected from the trading strategy.

3.3 Tools

Hardware

All data pre-processing, model training and trade simulations in this research were conducted on a personal computer with Intel i5-3320M dual-core CPU @ 2.60GHz and 16 GB RAM.

Software

R and Python were the two main software package used in this research. R is a software environment with programming language interface, which is popular for statistical computing and data visualization. In this research, it was used for data pre-processing, data analysis and data visualization. While Python is a high-level programming language, which is popular for scientific computation. Regarding functionality, it was used in data downloading, machine learning, and trading strategy backtests. The table below lists the detail libraries involved.

Table 3.4: Software utilizations

Name	Version	Description
R	3.6.1	Core R
Tidyverse	1.2.1	Collection of new R packages that follows the "tidy" design philosophy. Among them, Dplyr and Tidyr are used for data wrangling, and Ggplot2 is used for data visualization.
Tsibble	0.8.3	Provides data structures and wrangling functions for time series data.
TTR	0.23-4	Provides functions of technical indicators.
Lubridate	1.7.4	Provides functions of date and time manipulations.
Glue	1.3.1	Provides functions of text formatting.
Python	3.6.8	Core Python
Numpy	1.16.4	Provides numeric functions for scientific computation.
Pandas	0.24.2	Provides data structures and wrangling functions for generic data
Pandas-DataReader	0.7.0	Provides the downloading function for Yahoo! Finance stocks data
Matplotlib	3.1.0	Provides data plotting functions
Scikit-Learn	0.21.2	Provides auxiliary functions for machine learning
Karas	2.2.4	Provides the high-level wrapper for Tensorflow
Tensorflow	1.13.1	A deep learning framework
Backtrader	1.9.74	Provides functions in trading system backtests and performance evaluations.

4. Deep Learning Market Forecast

This is the first of two studies conducted in our research. In this study, we replicated some previous research where deep learning together with raw market data features was used in market forecasting models. Typically, they were classification models with a binary directional label. However, the research problem remains unanswered, because 1) the sample size was too small, and 2) no appropriate data separation procedure was considered in these studies, which led to their generalizability being doubted.

Our research addressed this issue by two means. The first was to include multiple stocks to increase the sample size. The second was to conduct the data separation as discussed in the previous chapter. However, only the training and validation processes were considered up to this study because the testing dataset was intended for later usage. For the validation in this study, the optimal combination of features (in raw market data sense), as well as the optimal deep learning network architecture was figured out for the market forecasting problem. The remainder of this chapter discusses the methodology, results, and implication of this study.

4.1 Problem Formulation

The market forecast problem was formulated as a classification problem, which is under the supervised learning approach in machine learning taxonomy. Supervised learning is, in a nutshell, a task of function approximation: suppose there are multi-dimensional features set (X) and label (Y), the objective is to find the best fitting function (M) that maps X to Y. When Y is one or a set of multinomial variables, the problem reduces to classification. The followings describe the preparation of features X and label Y for our problem, as well as the options for X being investigated.

Features

Windows of 30 most recent bars were considered as the features X. The size 30 was chosen, which was according to general observations from the literature review. The windows were prepared as follows. As discussed in the previous chapter, Bars data was there as a result of the data pre-processing step. Three options of Bars setup were investigated, including OHLCV, OHLC and Close Only. Different options were treated similarly to produce three features set options to be validated. Take the OHLCV option as an example, for a sample of current day T being considered, the window representing trading days {T-29, T-28, ..., T-1, T} was consolidated as the features, having a dimension 30x5.

Followed by the consolidation of windows, normalization was performed. This is because only price patterns but not the absolute price levels within a window should be relevant in predictions. The Open, High, Low, Close, and Volume attributes in a window formed five channels, and each channel was standardised as the update rule (4.1). Take the Close channel as an example, its mean and standard derivation within the channel was first calculated. Then, each of the 30 values in the channel was centred and normalized by its standard derivation.

$$\text{Close} := \frac{\text{Close} - \mu_{close}}{\sigma_{close}} \tag{4.1}$$

Label

The label Y was considered as the binary variable indicating whether the next 5th day (T+5) asset price goes up or down (the rare zero cases were counted as down). Alternatively, the label was regarded as the sign of 5-day return, which was calculated as Formula (4.2), where the term Close_T means the closing price on day T.

$$\text{Return}_\text{T} = \frac{\text{Close}_{\text{T}+5} - \text{Close}_\text{T}}{\text{Close}_\text{T}} \tag{4.2}$$

The number of future days, 5, was chosen according to prior study [14], where the range

of optimal numbers was found to be around 5 to 10. To match the output format of our deep learning model, the binary indicator was further one-hot encoded into tuples, that is [1, 0] and [0, 1]. Figure 4.1 shows the distribution of the two label classes. As can be seen, the class distribution was approximately balanced.

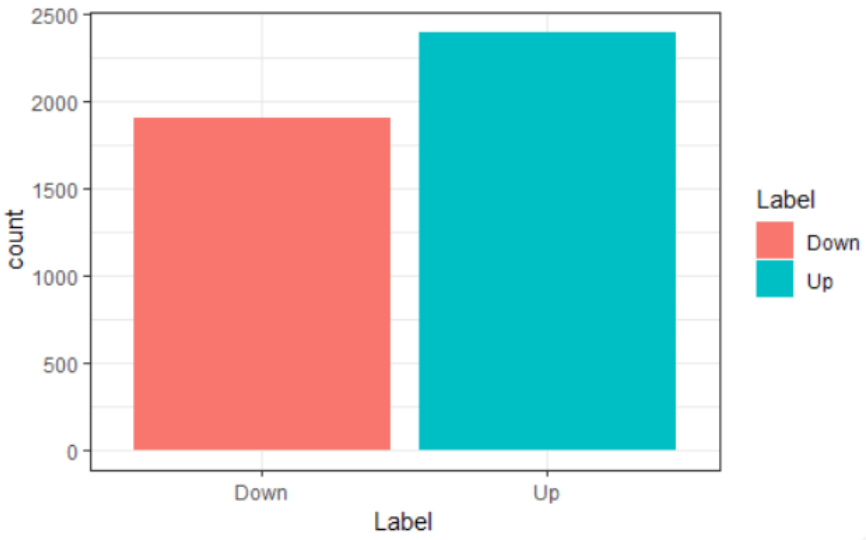

Figure 4.1: Distribution of the binary direction label

4.2 Deep Learning Models

Deep learning was adopted for the classification models in this research. The deep learning technology is basically an extension from Artificial Neural Network (ANN), with techniques in various areas including weights sharing, weights initialization, activation functions, optimization algorithms, and regularizations being applied to train

deep neural networks with many layers, which was not possible for ANN before. Among these techniques, there are two popular approaches in weights sharing. Such approaches result in two unique classes of network architectures, known as Convolution Neural Network (CNN) and Recurrent Neural Network (RNN).

Previously, both architectures have been applied for financial market forecasts as mentioned in the literature review. In this study, we built models of both architectures and investigated which one best performed for our problem. The remaining contents in this section describe the designs of the two neural networks, as well as the validation process. However, a basic understanding of deep learning techniques is assumed, ones may refer to the classic material [25] if necessary. Moreover, if a configuration is not mentioned, the default setting in the Keras package is assumed.

Convolution Neural Network

1D-CNN was considered for the sequential bar data with 5, 4 and 1 channels respective to different features options. The CNN filters convolved over the time dimension. To understand this, one might draw an analogy to the popular 2D-CNN scenario for images, where a 2D image with three colour channels is convolved along the height and width dimensions. 2D-CNN excels in detecting edges or simple shapes in local regions from

images. Similarly, 1D-CNN is expected to detect whether a segment of price charts is the peak, valley, flat or slope. The combination of such segment features is expected to be valuable information in predicting future price outcomes of financial assets.

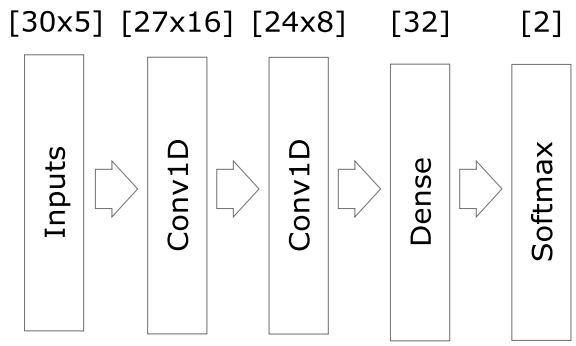

Figure 4.2: The architecture of the CNN model for market forecast.

Our 1D-CNN model was designed based on [17], which was a two-layer 1D-CNN model. Figure 4.2 illustrates the architecture after our adoption. The bracketed values on the top specify the dimension of data as it was transformed layers by layers. The network takes input features with dimension 30x5, as the OHLCV features option is assumed and outputs prediction with 2 values, each representing probabilities of the corresponding market direction, and they should sum up to one. If the OHLC or the Close only options are chosen, the input dimensions are then 30x4 and 30x1 respectively.

There are two layers of Conv1D block in the network, each encapsulates multiple inner layers. The first inner layer is a 1D convolution layer with affine transformation only. Its convolution filter size is 4. The filter outputs 16 units in the first network layer and 8 units in the second network layer. There is no padding for the convolution, instead of padding zeros from the original design. This is because such an arrangement induced virtual fluctuations around the window edges, which was especially bad at the right side. The second inner layer is batch normalization. Followed by that, there is a leaky ReLU activation function for the non-linear transformation, which is essentially a standard ReLU function but with a mild slope at the left side, as shown in figure 4.3. Finally, the Conv1D block ends with a dropout with a keep rate of 0.5 for regularization.

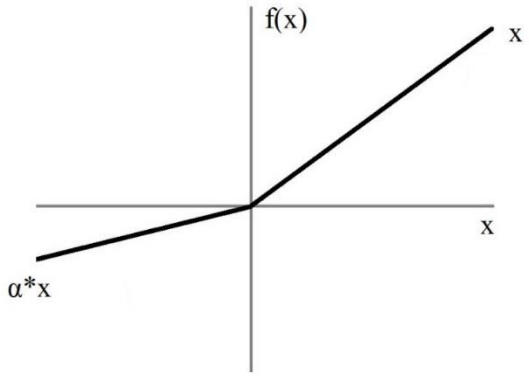

Figure 4.3: Leaky ReLU activation function with slope alpha=0.3 at the left side.

Followed by the Conv1D block, a Dense block is placed at the second last network layer. It includes an affine transformation, and then a batch normalization, finally a

ReLU activation layer. The number of output units is reduced from 64 of the original design to 32, as we want to control the variance error. The total number of parameters in the network is 7,258.

To optimize these parameters, the advanced mini-batch stochastic gradient descent algorithm called Nesterov Adam (Nadam) [26] was adopted. The batch size was from 64 increased to 128, with considering the increase in sample size over the original research.

Here, we briefly explain the Nadam algorithm. In general, there are two approaches to improve the classic stochastic gradient descent algorithm - the moment approach and the adaptive learning rate approach. The classical moment algorithm helps to accelerate the gradient descent and escape from local minima but might overshoot. The Nesterov momentum algorithm, due to its overshoot control, is proved to perform better than the classical one in most cases [27]. On the other hand, adaptive learning rate algorithms, such as RMSprop [28], enable the balanced weight update along each dimension. Adam [29] was the first algorithm that takes benefits of both approaches by combining the classical momentum with RMSprop. Given the fact that Nesterov is superior to classical momentum, Nadam [26] is motivated to combine with this momentum algorithm

instead. Although the final algorithm is more complicated, thanks to Keras, we are allowed to apply it with just a few code changes.

Concerning a noisy problem such as the financial forecast problem, the validation metrics seldom converge. We followed the original research to adopt early stopping to capture the model with maximum validation results. But a deterministic approach was considered, instead of that from the original study, where the final model was selected from one of the 100 epochs by inspecting validation results. In our approach, models were trained with different learning rates with 100 epochs fixed. And then the final number of epochs and learning rate were determined by the maximal validation result. Once determined, they were fixed for testing.

We took capital preserving as the first principle in trading. That is, we could forgive the opportunities that were not grabbed, but when an upward prediction was made, its correct percentage was maximized. In connection with this, there were two procedures involved in the validation. Firstly, the number of epochs and learning rate were selected that maximize the precision of upward direction (with 0.5 thresholds). Once they were selected, the classification threshold was adjusted to further maximize the final precision of the model.

Recurrent Neural Network

RNN architecture has been the natural choice for sequential data, such as the case in speech recognition. Therefore, it was considered for our sequential financial data. The design from [30], as shown in Figure 4.4, was adopted, It was a 2-layer RNN applied to forecast daily Bitcoin prices. The network inputs and outputs were the same as our CNN, but the weights were shared differently.

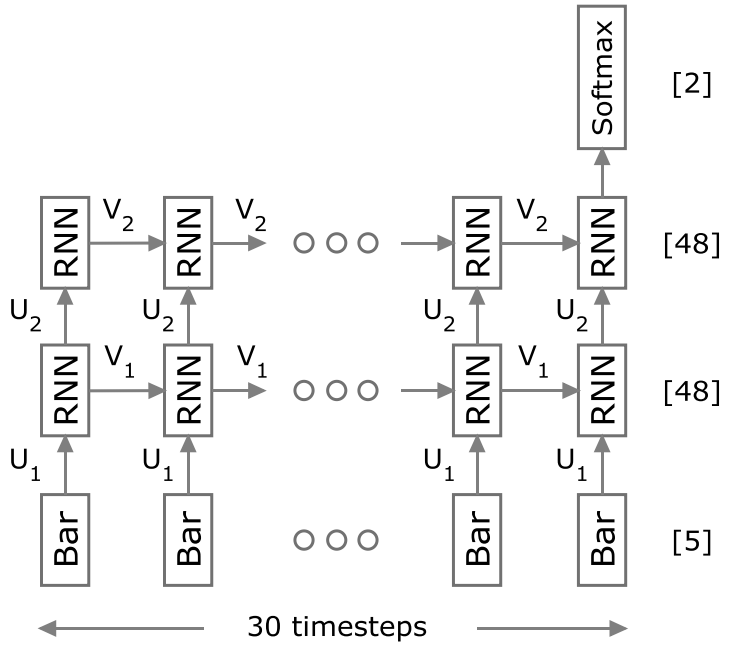

Figure 4.4: The architecture of our RNN model for market forecast.

A typical RNN architecture is as follow. It characterises with the RNN cells, which have two sets of weights U and V. The cells are applied to each time step recursively, outputting a multi-dimensional status up to the time step. The U weights matrices

connect the cells to a time step of the input data, or the outputs of the same time step from the underneath RNN layer; while the V weights matrices connect to the status of the previous time step. In case it is the first timestep, the feed-in status becomes zeros. Finally, the RNN cell of the last time step at the last layer connects to the Softmax layer for classification.

We followed the conventions to use Tanh for all activation functions in RNN cells. Our RNN cells output 48 units each, the same in both layers. This was considered to attain a similar total number of parameters as its 1D-CNN counterpart. In terms of regularization, dropouts with keep rate of 0.5, are applied in both layers, but only for vertical signal flows. There is a total of 7,346 parameters in the network.

Regarding optimization algorithm, we followed the original research to adopt RMSprop [28]. What else remained in the validation process followed the CNN design, however. In the end, the number of epochs, learning rate and classification threshold were determined.

In addition, a more advanced RNN architecture called Long-short Term Memory (LSTM) was also studied. LSTM is essentially the same as basic RNN when viewed

externally; it interacts with input data and neighbouring cells in the same way as basic RNN. However, its cells are designed with a complicated circuit inside, which enables a more efficient gradient backpropagation along the time axis. This is especially important when the input data sequence is long or even indefinite.

In spite of this advantage, one drawback in LSTM is its requirement of more weights - the 4 times as required by RNN cells. The increase in network parameters complicates the approximating function, and hence extra variance error is introduced. To control this, we adjusted the LSTM cell output units to 24 accordingly. Finally, the total number of parameters for our LSTM model was 7,634. We kept the default activation functions setting for all gates inside LSTM cells. All other regularization measures, training and validation procedures followed the RNN design.

4.2 Results

Market data from five stocks, including MSFT, AAPL, AMZN, GOOGL, and FB, were divided into two datasets; the training set from Jan 2012 to Jun 2015, and the validation set from Jul 2015 to Dec 2015. Three sets of features options, namely OHLCV, OLHC, and Close Only, were derived from the market data. The training set was repeatedly

inputted to three deep learning models, the 1D-CNN, the RNN, and the LSTM. Based on the performance in the validation set, the best model and the best features option were selected at the end. The details are as follows.

Models Selection

Most hyper-parameters for the three models, the 1D-CNN, the RNN, and the LSTM, were pre-determined with references to prior research and amended as discussed in the previous section. Learning rate and number of training epochs were the last ones to be determined. First, the features option of OHLCV was fixed. The optimal learning rates for each model were found from grid-search with 100 epochs trained for each setting. Figure 4.5 to 4.7 shows the training results of each model for the learning rates selected.

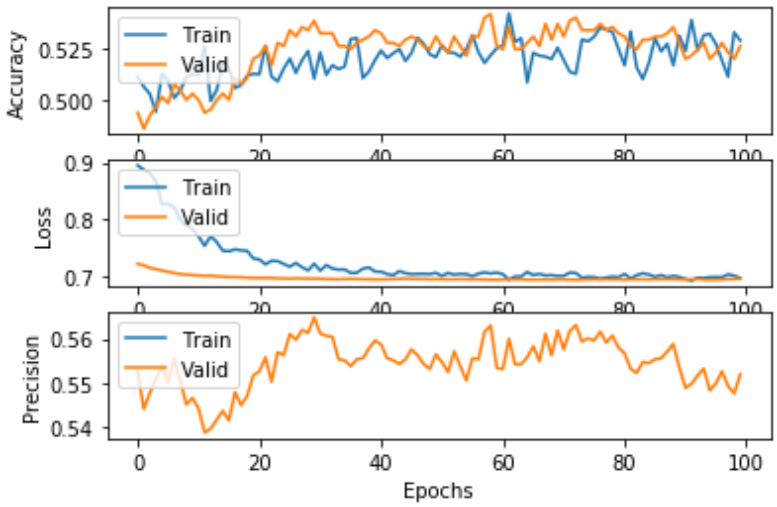

Figure 4.5: Training result of the 1D-CNN model with learning rate 1e-4

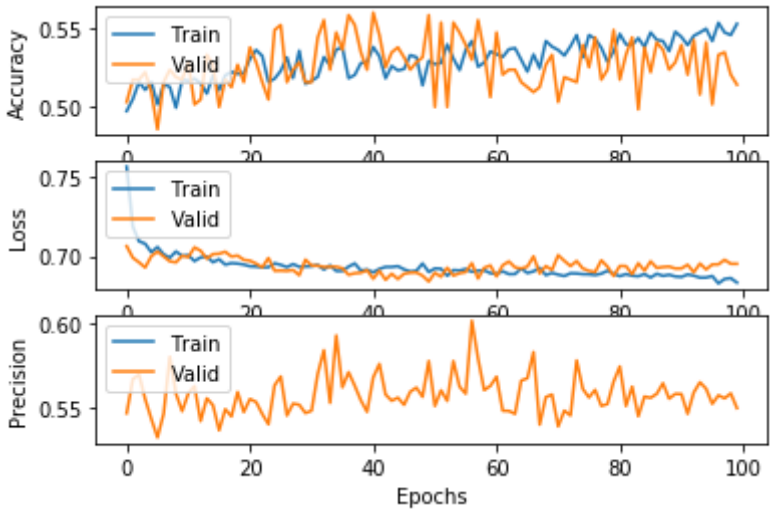

Figure 4.6: Training result of the RNN model with learning rate 1e-3

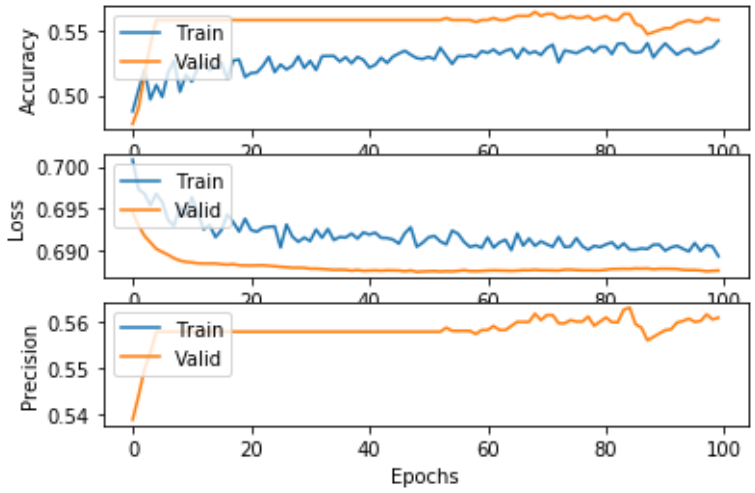

Figure 4.7: Training result of the LSTM model with learning rate 1e-4

From the grid search, the optimal learning rates for 1D-CNN and RNN were found to be 1e-4 and 1e-3 respectively. As can be seen from figure 4.5 and 4.6, validation accuracies increased with training accuracies; and validation losses decreased with training losses, up to a few dozens of epochs. This means both models were improving their generalization ability to the validation set during the initial training processes. On

the contrary, even though LSTM achieved higher scores as shown in figure 4.7, the model was abandoned because it seemed to learn to predict all samples as upward definitely without generalization.

Comparing figure 4.5 and 4.6, the RNN model generally had higher validation accuracy and precision and a lower validation loss than the 1D-CNN model. The peak validation accuracy in RNN was 60%, better than the 57% in the 1D-CNN counterpart. Therefore, the RNN model was selected.

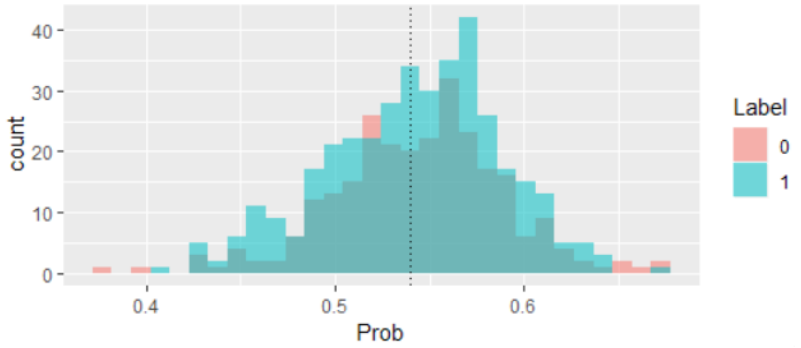

Figure 4.8: The predicted probabilities (upward) distribution of the RNN classifier.

To further improve the performance of the RNN model, the model was retrained with 40 epochs, around which its performance was maximized. Figure 4.8 presents the distribution of predicted probabilities (upward) in the validation samples among the two labels. A threshold of 0.54, as indicated by the dotted vertical line, was chosen to

balance the numbers of positive and negative predictions and maximize the precision. The confusion matrix is hence shown in table 4.1. As induced from the confusion matrix, an accuracy of 50.94% and a precision (upward) of 58.52% were calculated. However, considering the actual positive rate was already 58.75%, which suggests the result was not impressive.

Table 4.1: The confusion matrix of RNN model classified threshold 0.54

		Predict	
		Down	**Up**
Actual	**Down**	113	151
	Up	163	213

Features Selection

In the previous experiment, the RNN model was chosen and selected with a learning rate of 1e-3 for the RMSprop optimization algorithm. Continued with the OHLCV features option being tested, the other two options, the OHLC and the Close Only, were tested accordingly with the same procedure.

As the results from figure 4.9 and 4.10, the numbers of early stopping epochs were

determined to be 30 and 35 respectively for OHLC and Close Only. The classification thresholds were set at 0.53 and 0.52 according to the distributions in figure 4.11 and 4.12.

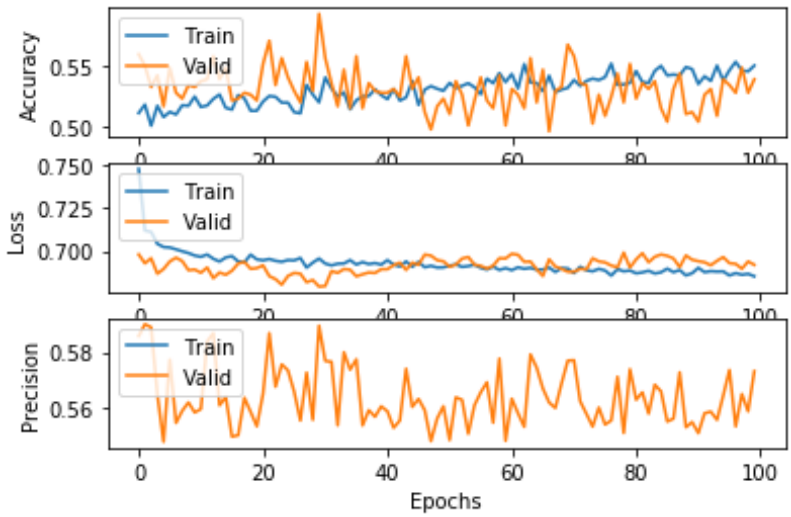

Figure 4.9: Training results of the RNN model with OHLC features option.

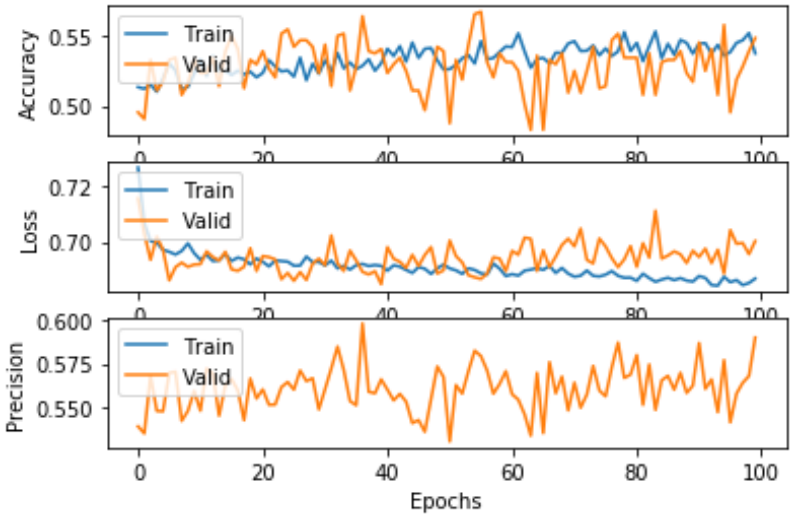

Figure 4.10: Training results of the RNN model with Close Only features option.

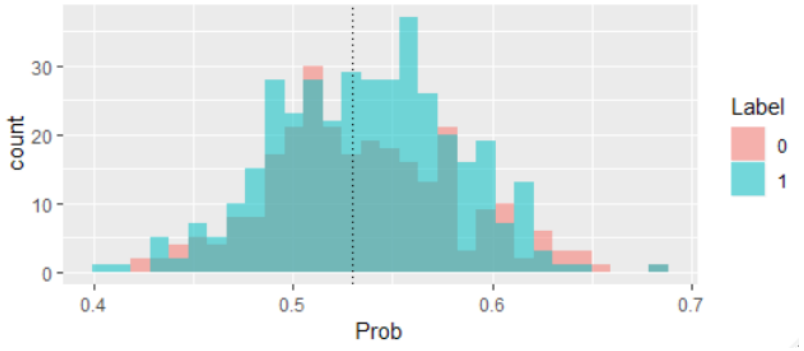

Figure 4.11: Distribution of the predicted probabilities (upward) of the RNN model with OHLC features option trained with 30 epochs.

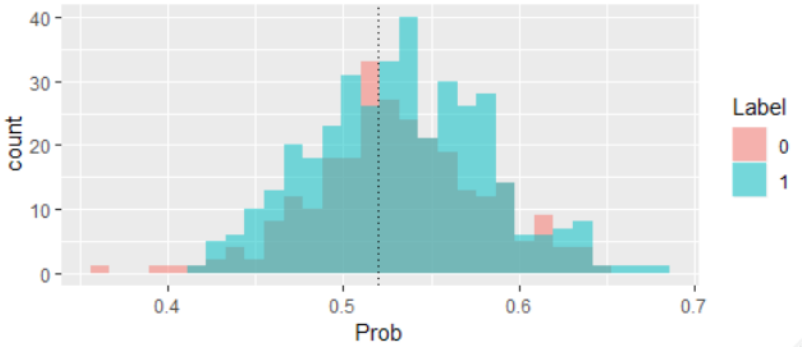

Figure 4.12: Distribution of the predicted probabilities (upward) of the RNN model with Close Only features option trained with 35 epochs.

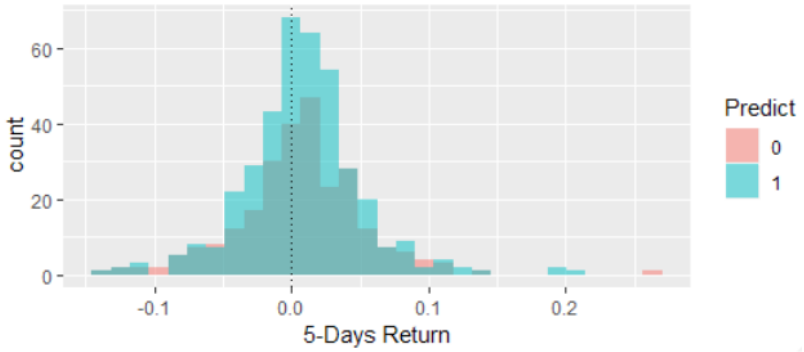

Figure 4.13: Distribution of 5-days return among the validation samples being predicted.

Another view of how the validation dataset being classified is presented in figures 4.13.

The two areas of both colours almost overlapped perfectly, which suggests the classifier

performed poorly in separating the two classes, and it was not ideal in forecasting both directions. However, the blue area in the right sides overwhelms the red area, suggesting there may be an edge for the long side forecast.

The confusion matrices are summarized in table 4.2 and 4.3. As induced from the figures, the accuracy of OHLC was 54.22% and that of Close Only was 51.56%; the precisions of OHLC was 61.76% and that of Close Only was 58.63%. From the findings, both features options performed much better than the previous OHLCV option. The OHLC option especially achieved the best results, with precision 61.76%, significantly higher than the actual positive rate, 58.75%, in the validation set. This indicates an advantage may exist in predicting the upward trend with the combination of RNN and OHLC features option.

Table 4.2: Confusion matrix for RNN model with OHLC features option classified with threshold 0.53

		Predict Down	Predict Up
Actual	**Down**	129	135
	Up	158	218

Table 4.3: Confusion matrix for RNN model with Close Only features option classified with threshold 0.52

		Predict Down	Predict Up
Actual	**Down**	129	135
	Up	180	196

4.3 Discussion

Models Selection

The blueprints from [17] and [30] were referenced to build market forecast models of three deep learning architectures, namely the 1D-CNN, the RNN, and the LSTM. The three models were trained and validated, and the RNN was found to be the best model.

The 1D-CNN model was found to be inferior to the RNN model in the validation, as shown in figure 4.5 and 4.6, which was not surprised because RNN has long been regarded as the deep learning architecture for sequential data. However, the potential for 1D-CNN model for this kind of data cannot be ignored. It could be just the hyper-parameters or model configurations not being optimized that hinders the performance. While 1D-CNN is good at learning localized patterns and RNN excels in learning from data in sequence. Ones could try to combine both architectures by placing 1D-CNN at the bottom and RNN on the top.

The RNN model was chosen over the LSTM, the reasons are more complicated because LSTM is usually considered to be better than basic RNN, due to its enhancement in gradient backpropagation along the time axis. Here is an attempt for the explanations.

Firstly, our window size was fixed at 30, which was not too to be handled by basic RNN. The improvements from LSTM could be more significant if the period length is longer or even indefinite. Secondly, the financial market data was so noisy that the RNN model spent 60 epochs to learn a validation accuracy of about 55%, but the metric could already be 58.78% when all samples were naively predicted as positive. This could be a source of issues. Thirdly, our actual objective was to optimize the precision instead of accuracy or loss. It was expected that the precision could be maximized as well when the loss was minimizing. But there was no hope for the LSTM model because the accuracy rose too quickly in the first few epochs, leaving no time for the precision to improve.

Comparing to previous studies [16], [17] where "testing" accuracies of 65% and 72% were reported, our validation accuracies were significantly lower. However, it is noticed that no validation procedure was mentioned in these studies and their testing sample sizes were very small (about 100). Therefore, it seemed that their models might have been overfitted to those specific samples, where the claimed high performance could only exist.

Features Selection

Three features options in term of raw market sense, namely the OHLCV, the OHLC, and the Close Only, were validation with the RNN model, and the OHLC was found to be the best features option. The result is not surprised that features composed of the Open, High, Low and Close price surpass those of Close price only, it reveals that excess information could be found from the Open, High, and Low price as well, although the Close is considered as the main source of information. It is noteworthy that most of the popular technical indicators, e.g. moving average, MACD, RSI, and Bollinger Bands, are derived from the Close price only. Therefore, it could be one of their limitations.

The features option with volume presented, on the contrary, being inferior to that without volume is out of our expectation because the volume is generally thought as a useful source of signals. One of the possible reasons could be a quality problem from the free Yahoo! Finance data, as the field is usually less attended. Another reason could be our models only capturing the short-term volume changes within the 30-day windows but what important is the long-term effect in volume.

Limitations and Recommendations

It was well noticed that the financial market forecast problem was so different from other machine learning problem, such as image classification, in a way that the noise-to-signal ratio was especially low, the highest accuracy we could achieve was just about the fifties. In connection with this, there are several implications. Firstly, we have seen validation accuracies higher training accuracies, which is actually normal. Secondly, the validation metrics would not converge if the models were trained extensively. Thirdly, even though the label classes were slightly imbalanced, the model learned to predict the majority class definitely.

To cater to these problems, one could adopt early stopping in the training, which we have done correctly. Moreover, one could consider balancing the classes distribution by introducing a class weight. However, we should not give up due to such low accuracies or precisions. As hinted from Formula (3.13), we could yield a positive expected return if the expected profit over the expected loss of a trading strategy is high enough.

Another issue due to non-convergence in the training was the randomness of the models generated. Initial randomness has been introduced in deep learning techniques including weight initialization, dropouts and stochastic gradient descent. It is not an

issue for normal problems, such as image classification, because the optimal model is converged at the end, but ours do not. This could be a two-bladed-sword, on the one hand, we are not sure if the optimal model has resulted and its predictions are reliable. On the other hand, ensemble learning could be implemented to include as many random models as we want and average their result. This could effectively lower the variance error without increasing the bias. However, for the simplicity of experiments design, in this research, all models were trained three times and one with middle performance was selected.

Even though the RNN model and OHLC features options were validated to the best combination, the classification performance was still bad in the overall, as can be visualized in figure 4.13, and the validation accuracy was 54.22% only, even lower than the actual positive rate. We hypothesize that this is because the inappropriate label design could have led to the inefficient classification. Take a look at figure 4.14 where the distribution of 5-day returns of the training samples was plotted, most of the samples were centralized in the middle region of the gaussian-like histogram. The label cut at the middle to separate the positive and negative samples into two parts but they were tightly coupled. An originally positive sample might have fallen to the negative side due to noises. The classifier confusion in separating these so similar samples could have

led to the inefficiency.

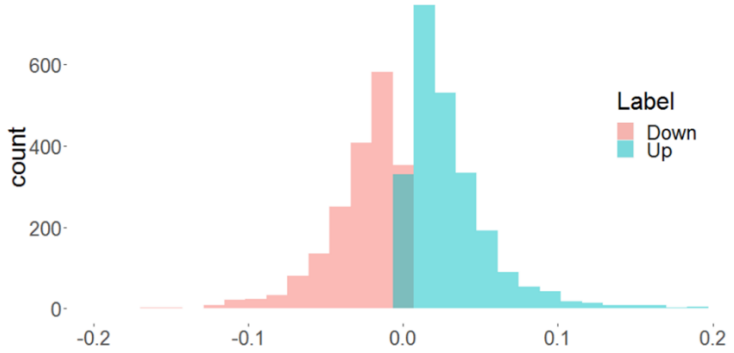

Figure 4.14: Distribution of 5-day returns in the training set

In addition, another issue associated with the label is that it could have no linkage to the ultimate objective of problems, such as making a profitable trade. Imagine a sample of 0.001 return is classified as positive but we might not profit from that in real situations because of noise and transaction costs.

To deal with such label issues in supervised machine learning, there could be two approaches. One is to formulate the problem in reinforcement learning (RL), another form of machine learning, such as that in [23], where supervised learning models might still be used in the function approximation or deep learning version of RL but the labels would then be directly related to the long-term objectives of problems, such as accumulated profits. The other approach is then to investigate a more reasonable label for the supervised learning problem, which is motivated in the next chapter.

5. Triple-Barrier Labeling Method

In the previous chapter, the RNN architecture and OHLC features option were selected as the best combination for the forecast model of binary direction label. The label is popular for market forecasting problems but its potential issues have been discussed. The model resulted in a validation precision of 61.7%, but it is yet to be tested for trading simulation with testing data. Meanwhile, an alternative labeling strategy, called triple-barrier labeling method, has been introduced by [18] recently. This chapter was motivated to have it compared with the traditional labeling method.

To proceed with the experiment, the corresponding labels for all samples were first prepared. Then, the previous RNN model for binary classification was modified to adapt the new three-class outputs and re-validated. Next, the binary direction model and the triple-barrier one were compared using the unseen testing dataset. Buying signals generated by the two models were fitted to a trading strategy template leading to two strategies, which were then simulated for the testing period. Finally, the performance of both trading strategies was analysed. The followings discuss the steps in details.

5.1 Problem Formulation

To adapt the triple-barrier label, the problem formulation was changed from binary to three-class classification. Although the labels are different, the features involved are the same. The OHLC features option remained selected as it was validated to be the optimal features set. The remainder of this section explains this special label and its generation process.

Triple-Barrier Label

For market forecasting models, we input features of a past window to predict outcomes of a future window. The direction of up and down is only one of the ways to conclude the future window. As opposed to that, the triple-barrier method set up three barriers for the window, and count which one being touched first as the label. Figure 5.1 illustrates the idea. In other words, during the next K periods, an upper and a lower boundary are set, and the label determine whether 0) the lower boundary, 2) the upper boundary, or 1) neither boundaries is reached first.

Figure 5.1: The **t**riple-barrier labeling explained.

Moreover, the levels of boundaries can be asymmetric as demonstrated in figure 5.1, an upper-to-lower ratio of 2:1 is set. Remember our principle of trading as discussed in Chapter 4 is capital preserving, and it is expected this approach enables that by increasing the precision for the up-side signals. Put it in this way, for a certain percentage the prediction of rising 2 units to reach the upper boundary to be correct, the percentage of rising only one unit must be even higher.

Coming to the concrete levels of boundaries, we favoured them to be adjusted automatically according to volatilities. This is because each stock has its own volatility levels and subject to change in different periods too. For examples, a 10% change in 10 days for a stock may be regarded as high but normal for another. In measuring volatility, we considered the Average True Range (ATR). To explain it, we first need to know that True Range (TR) for current time T is calculated as follows:

$$\mathrm{TR_T} = \max\left\{ \begin{array}{c} \mathrm{High_T} - \mathrm{Low_T} \\ |\,\mathrm{High_T} - \mathrm{Close_{T-1}}\,| \\ |\,\mathrm{Low_T} - \mathrm{Close_{T-1}}\,| \end{array} \right\} \tag{5.1}$$

The ATR is just the moving average of TR values over the past N days, and N was chosen to be 30. Finally, the upper bound level was chosen as 3.0 ATR above the current close price; and the lower bound level 1.5 ATR below.

Vectorization in Label Computing

Vectorization is a technique of writing functional programs that replaces iterative operations with linear algebraic or elementwise ones. In a nutshell, that is to avoid the for-loop statements as much as possible. By doing so, the programming codes become more concise, easier to read and maintain. More importantly, it speeds up the executions by letting the underneath APIs to handle the parallelization or pass the complex operations to a higher performance programming language, such as C++.

In computing the triple-barrier, vectorization was relied upon. Listing 5.1 shows the relevant codes in the R programming language. As can be seen, we could complete the logic within 20 lines of codes and only one for-loop inside.

Listing 5.1: Triple-Barrier Labeling Algorithm in R

```
# calculate atr
df$ATR <- TTR::ATR(df[, c('High', 'Low', 'Close')], 30)[, 'atr']
df$Return.10D <- (lead(df$Close, 10) - df$Close) / df$ATR

# calculate boundaries
df$Lower.Bound <- df$Close - (1.5 * df$ATR)
df$Upper.Bound <- df$Close + (3.0 * df$ATR)

# calculate break days
df$Lower.Days <- 999
df$Upper.Days <- 999
for (t in 10:1) {
    cond <- (lead(df$Low, t) < df$Lower.Bound) %>% replace(is.na(.), FALSE)
    df[cond, 'Lower.Days'] <- t
    cond <- (lead(df$High, t) > df$Upper.Bound) %>% replace(is.na(.), FALSE)
    df[cond, 'Upper.Days'] <- t
}

# prepare label neutral
df$Label <- 1

# upper bound first
cond <- df$Upper.Days < df$Lower.Days
df[cond, 'Label'] <- 2

# lower bound first
cond <- df$Lower.Days < df$Upper.Days
df[cond, 'Label'] <- 0
```

Label Analysis

After computing the labels, its distribution is shown in figure 5.2, where the classes distribution was so imbalanced. In the later section, we discuss its implication and how to solve the problem.

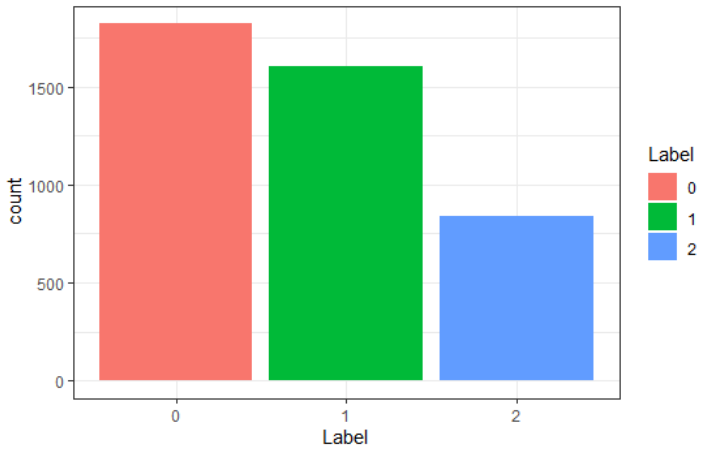

Figure 5.2: Distribution of triple-barrier labels in the training set.

The separation of data by the label is illustrated in figure 5.3. The 10-day returns measured in ATR was used to summarise the natures of the samples. Samples with similar trend tendency should have similar values. As can be seen from the figure, the losing samples (red) were more separated with the winning samples (blue) under this label configuration. We hypothesize that this has created an easier task for the classifier.

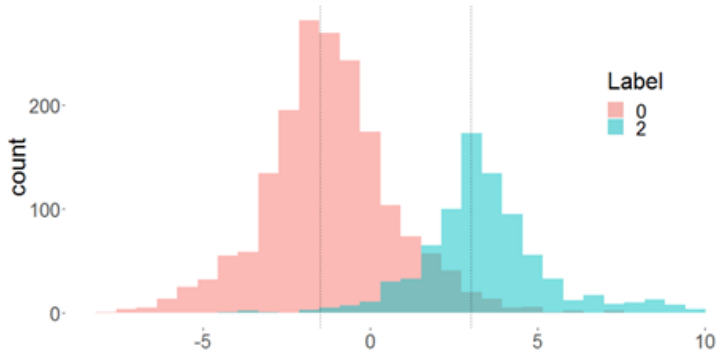

Figure 5.3: Distribution of 10-Day Returns (ATR) among the triple-barrier labels for the samples in the training set.

5.2 Deep Learning Forecast Model

Model Design

The design of models with the triple-barrier label followed essentially the same as that selected for binary direction label, that is the RNN model. This is because it has been validated to be the best model among all alternatives in the previous chapter. However, its classification head was amended for these three classes outputs, as shown in figure 5.4. It has the 2-layer RNN architecture, with each RNN cells output 48 units, the final Softmax layer outputs 3 units. We followed the binary direction case to optimize it with the RMSprop algorithm.

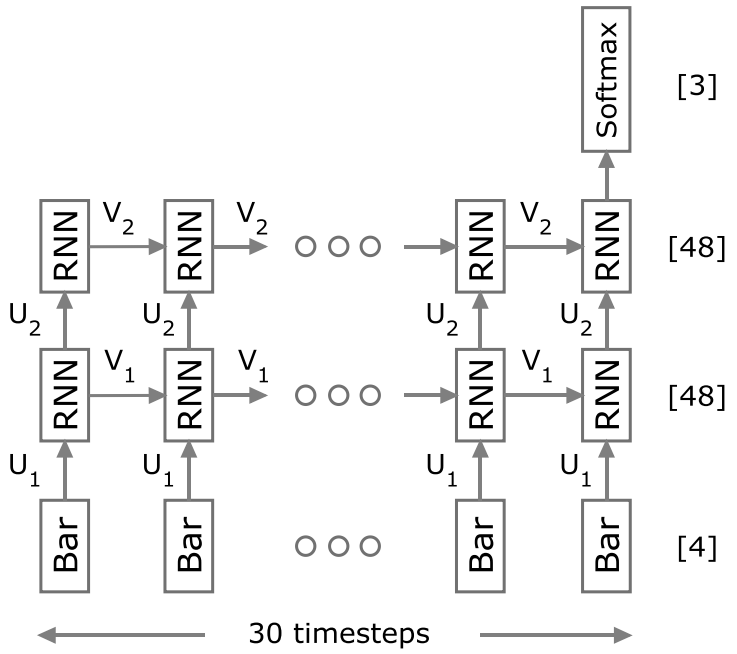

Figure 5.4: Architecture of the RNN model for the triple-barrier label

Hyperparameters Tuning

With hyperparameters tuning, the learning rate and the number of training epoch for the triple-barrier label were tuned to the optimal values, similar to the approach for binary direction label model. However, we considered maximizing a specialized metric, namely the won-to-lost ratio, measured in the validation set. The metric is just similar to precision for upper bound condition, but with a slightly different denominator, and it also incorporates the capital preserving idea in mind. The won-to-lost ratio was calculated by UU divided by LU, where UU and LU are defined in the confusion matrix template as shown in table 5.1.

Table 5.1: Template of the confusion matrix for triple-barrier label

		Predict		
		Lower	**Neutral**	**Upper**
Actual	**Lower**	LL	LN	LU
	Neutral	NL	NN	NU
	Upper	UL	UN	UU

On the other hand, the distribution of the triple-barrier label classes was highly imbalanced, as indicted form Figure 5.2. This could lead to the model bias in predicting the majority class (the lower bound condition), and that could be even worse when the model was trained extensively. To overcome this problem, we added a class weight for the minority class (the upper bound condition). Whenever the minority class was wrongly predicted, the loss was multiplied with this value. This seemingly increased the number of samples for the minority class to balance the overall classes distribution. The minority class weight was calculated by the number of the majority samples divided by the number of minority samples in the training set. In our case, that was 2.17 (1819 / 840).

Walk-forward Testing

Up to this stage, the testing dataset remained unused in models training and validation. Meanwhile, we are interested to know how our models generalize to unseen data. Therefore, a walk-forward test using the testing dataset was conducted. Figure 5.5 illustrates the idea of walk-forward testing, where a model was trained from 6 months of testing data and used to predict the labels of the next month. This model was then discarded, and another model was trained for the consecutive 6 months and used to predict the forward one month's labels. This process was repeated until predictions of 36 months in the testing period for both models were all generated.

Considering training samples in the walk-forward testing were much smaller than that in the validation, some hyper-parameters were adjusted accordingly. The mini-batch sizes were decreased to 64, and the learning rates were increased by a one-third logarithmic step size, that is multiplying by 3 roughly.

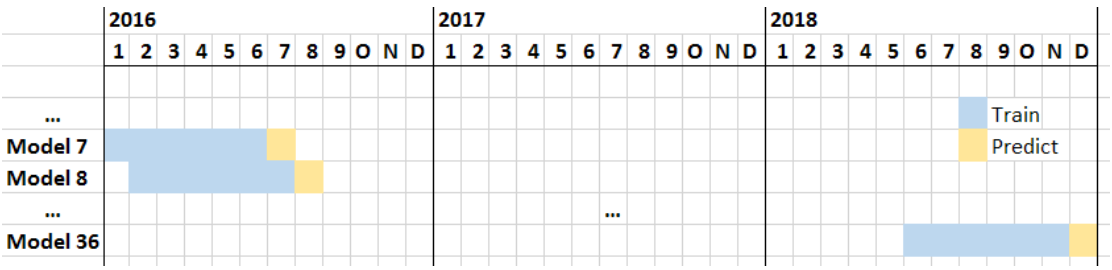

Figure 5.5: Walk-forward testing illustration.

5.3 Trading Strategies

There were four trading strategies being considered in total, the first two were strategies designed to incorporate signals from forecast models, serving to evaluate the models indirectly. The last two were baseline strategies for controlling and comparing the usefulness of the previous two strategies.

Backtrader, an event-driven backtesting package for Python, was used to backtest the four trading strategies. It simulates real-world market environments by modelling market orders and order fulfilments into events. Our strategies were first implemented in Backtrader and then simulated for the testing period. Performance metrics and trade statistics were thereafter computed for all strategies, reflecting several dimensions including profitability, stability, and risks, as well as the quality of trades.

Long with Trailing Stop-Loss

A trading strategy is usually associated with an entry rule and an exit rule, which correspond to opening and closing a position. For the former two trading strategies mentioned earlier, a strategy template was applied, which is described as follows. The template characterises an entry rule that is a placeholder for forecasting models. Whenever the forecast model issues a buying signal and there is no opening position, it

creates a market buy order allocated with all outstanding capitals. The order is assumed to execute with the day's closing price.

The exits occur in stop-loss conditions. Stop-loss is a common tactic for controlling the risks in trading strategies. In our case, an initial stop-loss order is attached to the buy order with a stop-loss level of 1.5 ATR below the current closing price. If that level is hit, the opened position is closed, and the stop-loss order is assumed to be executed with the opening price of the next day after the hit. However, if the price goes upward, and makes a new high, the stop-loss loss order is adjusted to 1.5 ATR below the highest price. The trailing stop-loss condition is illustrated in figure 5.5.

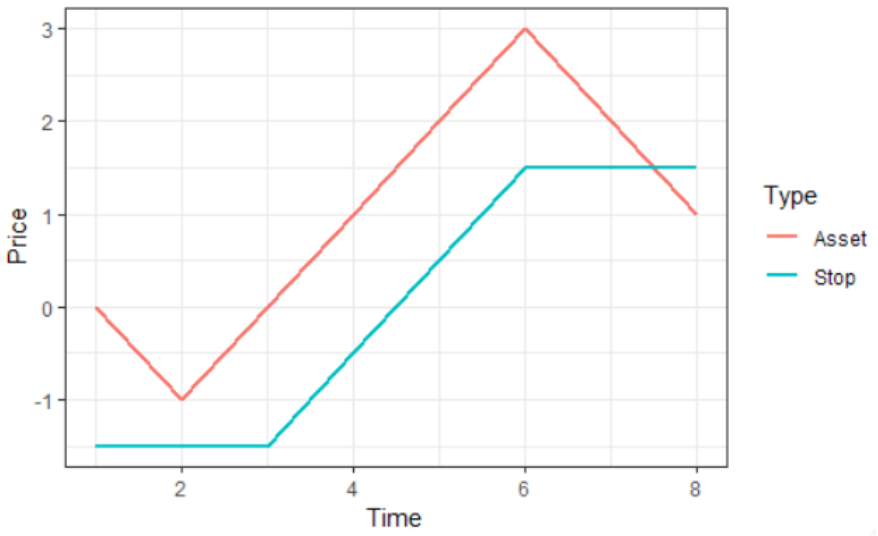

Figure 5.5: Illustration of the trailing stop-loss mechanism

Substituting the two forecasting models to the placeholder, two trading strategies were resulted, namely:

1) trailing stop-loss with triple-barrier forecast signals; and

2) trailing stop-loss with binary directions forecast signals.

The buying signals were taken as the upper bound conditions for triple-barrier model, and the upward conditions for the binary directions model.

Baseline Strategies

The following two baseline strategies were considered:

3) tailing stop-loss with random signals

4) the buy-and-hold

The first baseline strategy employed the template as described earlier, but the placeholder was substituted with a naïve model that issues buying signals randomly, always with 0.5 probability. It served as the control to verify whether the performance of the previous two strategies was explained by the market, the trading strategy, and the forecast model itself.

The second baseline strategy was as simple as its name suggests, it bought the relevant stocks at the beginning and held till the end. It involved no closed trade, so no trade statistics were generated. It portfolio value fluctuated just as how the stock price went. Because it involved no trading decisions, other complicated strategies such as the former two must defeat this to rationalise their existence. However, due to the testing period being studied was an exceptional bull market in recent years, it was still not trivial to surpass.

5.4 Results

Continued from the experiments in the previous chapter, the RNN model was selected for this classification task and modified to adapt the new label with three classes outputs. This experiment proceeded in two stages; the first stage fine-tuned some of the important hyper-parameters in the new RNN model with validation; the second stage generated the buying signals for the testing period through the walk-forward testing for both binary direction forecast and triple-barrier forecast model.

A trailing stop-loss strategy was designed for this study. The performances of both models in the testing period were evaluated by feeding their buying signals to their

trading strategy correspondingly. Their performances were also compared with two baseline strategies, the 1) buy-and-hold strategy and 2) the trailing stop-loss strategy taking random buy signals, with an aim to verify that the result was not due to the strategy or the market but the forecast models.

Model Validation

Different learning rates were validated; the optimal value was found to be 3e-4. Figure 5.6 shows the training results of 100 epochs with this value. As shown from the figure, the validation accuracy and won-to-lost ratio for the upper bound signal peaked at around epoch 60. Therefore, 60 was selected as the early stopping epoch.

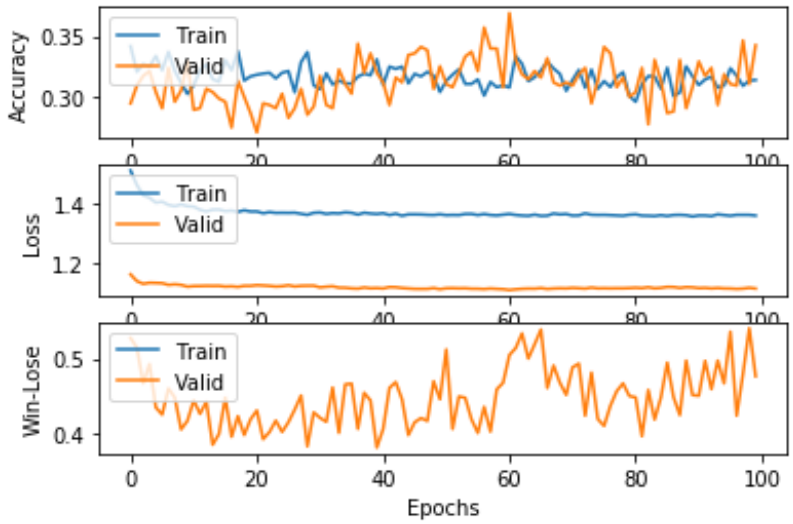

Figure 5.6: Training results of the triple-barrier model with learning rate 3e-4

The model was then retrained with this number of epochs. The confusion matrix that summarises the prediction results in the validation set in given in table 5.2. We quantitatively evaluate the model as follows. Suppose we adopt a simple trading strategy which stops loss and stops profit definitely when the lower and upper bounds are hit, then a reward-to-risk ratio of 2 is yielded. From the confusion matrix, the winning rate of 36.36% can be calculated (108 / (108+189)). When we substitute these two numbers into Equation (3.14) in Chapter 3, then the expected return over expected loss yields 0.0909, which is a positive number, meaning positive return is expected for this simple trading strategy, at least for the five stocks in the validation period.

Table 5.2: Confusion matrix of the triple-barrier model trained with 60 epochs

		Predict		
		Lower	**Neutral**	**Upper**
Actual	**Lower**	74	72	189
	Neutral	55	63	142
	Upper	24	23	108

Trading Strategies Backtests

The optimal model for the triple-barrier label was selected in the previous section. We moved on to compare it with the model for a binary direction label. In connection with

this, we took the testing dataset reserved for walk-forward testing and generated predictions for the testing period with the models. The positive predictions, which are the upward condition for binary direction and the upper bound condition for triple-barrier, were fed to the designed tailing stop-loss strategy as entry signals. To evaluate the strategies with different forecasting models, as well as the baseline strategies, all four strategies were implemented in Backtrader, the backtest package. That includes:

1) trailing stop-loss strategy with binary direction forecast signals;
2) trailing stop-loss strategy with triple-barrier model forecast signals;
3) trailing stop-loss strategy with random signals; and
4) buy-and-hold strategy;

Simulations were run with Backtrader using the market data from the testing period between 2016 and 2018, and trading results were collected. Figures 5.8 to 5.12 are the equity curves that illustrate the changes in portfolio values for each stock with different strategies. Tables 5.3 to 5.7 summaries the performance metrics and trade statistics for the strategies.

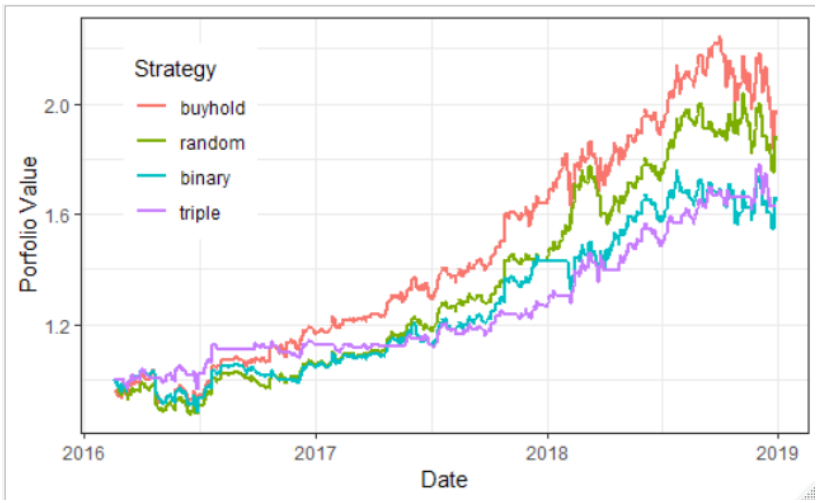

Figure 5.8: Equity curves for MSFT backtested from 2016 to 2018

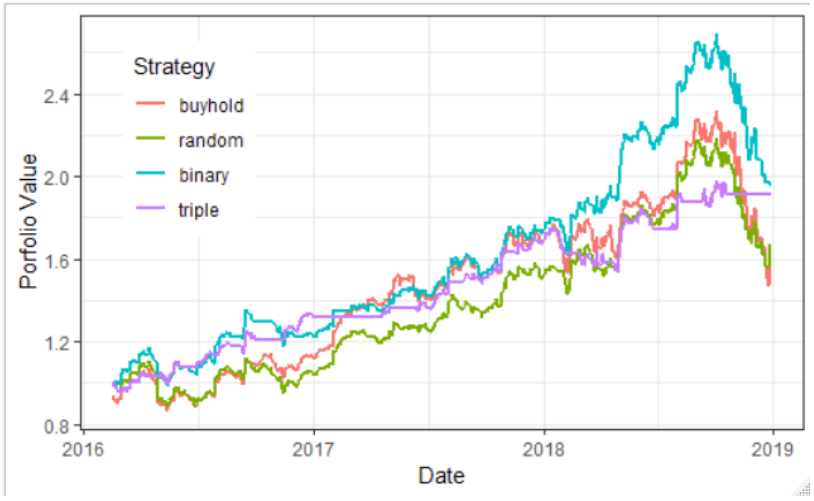

Figure 5.9: Equity curves for AAPL backtested from 2016 to 2018

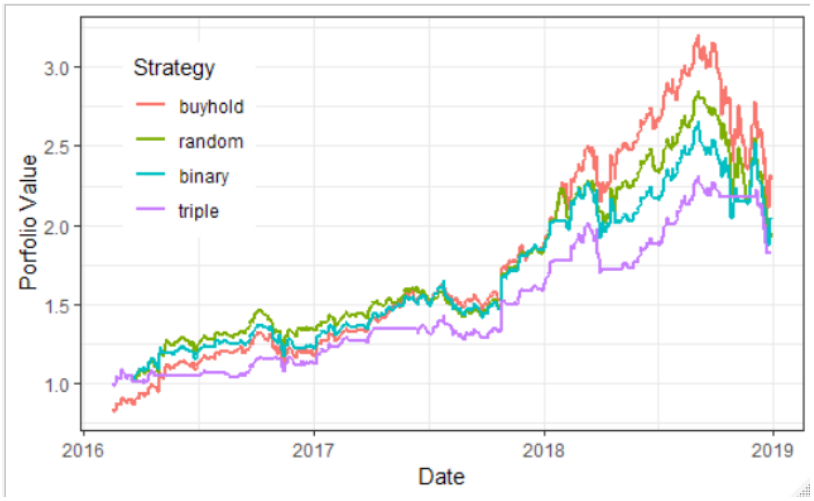

Figure 5.10: Equity curves for AMZN backtested from 2016 to 2018

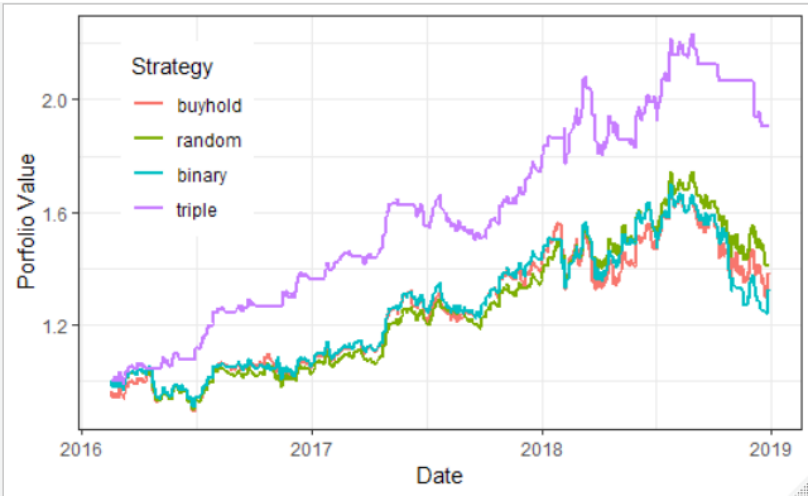

Figure 5.11: Equity curves for GOOGL backtested from 2016 to 2018

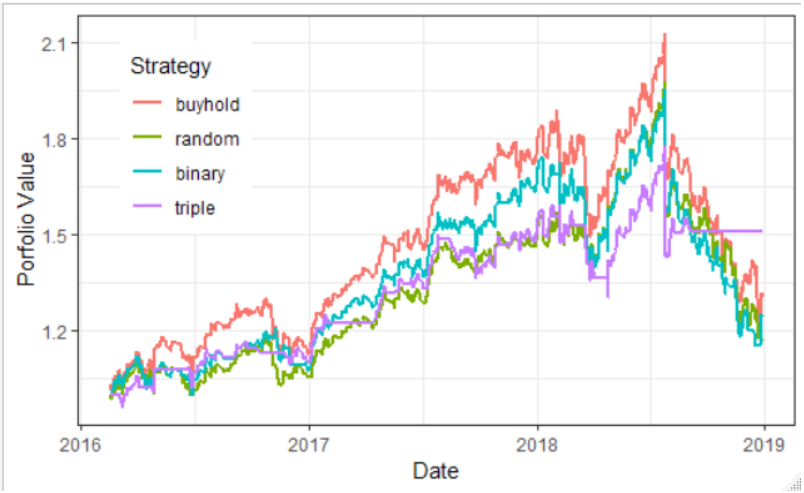

Figure 5.12: Equity curves for FB backtested from 2016 to 2018

Table 5.3: Strategies Average Performance Metrics Summary

Symbol	Annual Return	Sharpe Ratio	Max Drawdown	Risk Adj. Return	Winning Rate	Profit / Loss
Triple-Barrier	20.51%	2.04	15.27%	148.70%	54.71%	1.73
Bi-Direction	17.51%	1.23	27.94%	71.59%	44.22%	1.63
Random	16.80%	1.19	27.04%	76.46%	45.01%	1.66
Buy & Hold	18.81%	1.23	31.04%	69.26%	-	-

Table 5.4: Performance Metrics for the Buy & Hold Strategy

Symbol	Annual Return	Sharpe Ratio	Max Drawdown	Risk Adj. Return	Winning Rate	Profit / Loss
MSFT	25.09%	2.27	18.23%	137.65%	-	-
AAPL	16.06%	0.76	36.51%	43.99%	-	-
AMZN	32.37%	1.98	34.10%	94.94%	-	-
GOOGL	11.28%	0.76	23.39%	48.22%	-	-
FB	9.22%	0.4	42.96%	21.47%	-	-
Average	**18.81%**	**1.23**	**31.04%**	**69.26%**	**-**	**-**

Regarding the natural stocks price developments from the buy-and-hold strategy, all stocks presented positive annualized returns over the three-year testing period, as given in table 5.4. In general, they had a clear upward trend until mid-2018, then the downward trend, as shown in figure 5.8 to 5.12. An average annual return of 19% was scored, which indicates an excellent profit especially for a strategy that involves no trading decision. This resulted in an average Sharpe ratio of 1.2, which is a normal level in terms of an average managed fund. However, the average maximum drawdown incurred, 31%, demonstrates a high risk.

The stocks can be mainly divided into two groups, the first group contained MSFT and AMZN, which were performing very well during the period, with annualized returns higher than 25%, and Sharpe ratios around 2.0 and beyond (Table 5.4), which means

they had strong and stable growths. Especially, the stock price of AMZN was tripled before the turmoil after mid-2018. The second group was the other average stocks in the period including AAPL, GOOGL, and FB.

Table 5.5: Performance Metrics for the Strategy with Random Signal

Symbol	Annual Return	Sharpe Ratio	Max Drawdown	Risk Adj. Return	Winning Rate	Profit / Loss
MSFT	23.15%	1.75	14.07%	164.57%	49.43%	1.73
AAPL	18.46%	1.08	28.54%	64.67%	39.78%	1.97
AMZN	25.03%	1.92	32.17%	77.82%	47.19%	1.53
GOOGL	12.09%	0.91	19.36%	62.43%	44.58%	1.68
FB	5.26%	0.29	41.07%	12.80%	44.05%	1.39
Average	**16.80%**	**1.19**	**27.04%**	**76.46%**	**45.01%**	**1.66**

Table 5.6: Performance Metrics for the Strategy with Binary Direction Forecast

Symbol	Annual Return	Sharpe Ratio	Max Drawdown	Risk Adj. Return	Winning Rate	Profit / Loss
MSFT	18.09%	1.33	15.19%	119.11%	42.68%	1.99
AAPL	25.15%	2.23	27.14%	92.67%	47.06%	1.77
AMZN	27.20%	1.69	29.53%	92.13%	45.56%	1.64
GOOGL	9.67%	0.55	26.99%	35.83%	44.94%	1.41
FB	7.44%	0.35	40.86%	18.21%	44.32%	1.35
Average	**17.51%**	**1.23**	**27.94%**	**71.59%**	**44.91%**	**1.63**

Next, the strategy with the binary direction forecast model was compared to the baseline of random signals. The comparison showed mixed results among the stocks in terms of all metrics (table 5.5 and 5.6). For the overall statistics, the strategy with binary direction forecast even demonstrated the poorer maximum drawdown (27.9% vs. 27.0%), risk-adjusted return (71.6% vs. 76.5%), and winning rate (44.2% vs. 45.0%). Therefore, we cannot conclude that the binary direction forecast model provides more useful signals to the trailing stop-loss strategy than tossing a coin, even though it is computationally more complex.

Table 5.7: Performance Metrics for the Strategy with Triple-Barrier Forecast

Symbol	Annual Return	Sharpe Ratio	Max Drawdown	Risk Adj. Return	Winning Rate	Profit / Loss
MSFT	17.61%	2.21	8.71%	202.16%	56.00%	1.79
AAPL	24.16%	3.14	12.14%	199.04%	55.32%	2.09
AMZN	22.12%	1.69	21.24%	104.14%	53.85%	1.52
GOOGL	23.94%	2.09	14.69%	162.97%	51.85%	1.97
FB	14.73%	1.05	19.59%	75.21%	56.52%	1.27
Average	**20.51%**	**2.04**	**15.27%**	**148.70%**	**54.71%**	**1.73**

Then, the strategy with the triple-barrier forecast model was investigated. As inferred from the statistics in table 5.5 and 5.7, this strategy significantly outperformed the

baseline of random signals. Especially, there was a 22.8% (20.5% vs. 16.8%) increase in annual return, 71.4% (2.04 vs. 1.19) increase in Sharpe ratio, and 43.5% (15.3% vs.27.0%) reduction in maximum drawdown, as well as 25.6% increase in winning rate. This indicates the strategy is more profitable and stable, lower in risk, and more probable for winning trades.

However, when looked microscopically, the well-performing stocks group, in contrast, underperformed in annual returns and Shape ratios under this strategy. The annual returns of MSFT and AMZN were 23.9% (17.6% vs. 23.2%) and 11.6% (22.1% vs. 25.0%) lower than the baseline strategy, respectively; while the Sharpe ratios of AMZN was 31.9% lower (1.69 vs. 1.92). If the maximum leverage applicable was considered, the strategy with triple-barrier forecast for these two stocks was still better than the baseline strategy because their risk-adjusted returns were significantly higher (202.2% vs 164.6% for MSFT; and 104.14% vs. 77.82% for AMZN).

Finally, the triple-barrier strategy was compared to the buy-and-hold baseline strategy. Similar to the previous comparison, the triple-barrier strategy significantly outperformed the baseline in general, with an average 50.8% reduction (15.3% vs. 31.0%) in maximum drawdown, 65.9% increase (2.04 vs. 1.23) in Sharpe ratio, and

114.7% increase (148.7% vs. 69.3%) in risk-adjusted return, as induced from table 5.4 and 5.7. Although if leverage was considered, all stocks under this strategy profited better than the baseline, microscopically MSFT and AMZN under this strategy performed poorer than the baseline in both annual return and Sharpe ratio. (17.61% vs. 25.09% returns for MSFT; 22.12% vs. 32.37% returns for AMZN; 2.21 vs. 2.27 Sharpe ratio for MSFT; and 1.69 vs. 1.98 Sharpe ratio for AMZN).

To conclude the findings, although computationally more complex, the strategy with binary direction forecast was just on par with the baseline of random signals. However, the strategy with triple-barrier forecast significantly defeats the baseline strategy, this also implies it surpasses its counterpart with a binary direction forecast model as well. Moreover, the triple-barrier strategy significantly defeats the buy-and-hold baseline strategy in terms of risk, stability, risk-adjusted returns, and winning rate. Meanwhile, its under-performance in profitability for the two well-performing stocks, MSFT and AMZN, deserves an investigation.

5.4 Discussion

To compare the performance of the traditional binary direction forecast model with that of the triple-barrier forecast model, trading strategies were designed to take buying signals from both models and their trading performance were evaluated and compared with baseline strategies.

Binary Direction Forecast

As the results indicate, the difference in performance between the strategy with the binary direction forecast model and the random signals baseline strategy was not significant. This suggests that the forecast model does not predict better than blind guessing when working with the trailing stop-loss strategy. As can be seen from table 5.3 and 5.6, the main problem was the remarkably low winning rate, 44.2%, on the average. This well supports the hypothesis that the inadequate label design has resulted in an inefficient classification model with low precision.

However, back in Chapter 4, a high validation precision 61.7% was reported for this model. One of the reasons for the difference is that validation metrics are deemed to bias to high values because every hyper-parameters and model configurations have been tried to maximize these values for the validation data. A deterioration in the testing

phase should be expected when the model predicts on the unseen dataset. Another reason is that the conversion of wining in the trailing stop-loss strategy is likely to be more demanding than predicting the upward condition of binary direction label. This can be explained in figure 5.13, to declare winning, the stock price must reach the safety zone, a region above 1.5 ATR, quickly enough before touching the ever-rising stop-loss level. It becomes apparent that another drawback for the binary direction forecast model, that is it has no direct connection to the trading strategies.

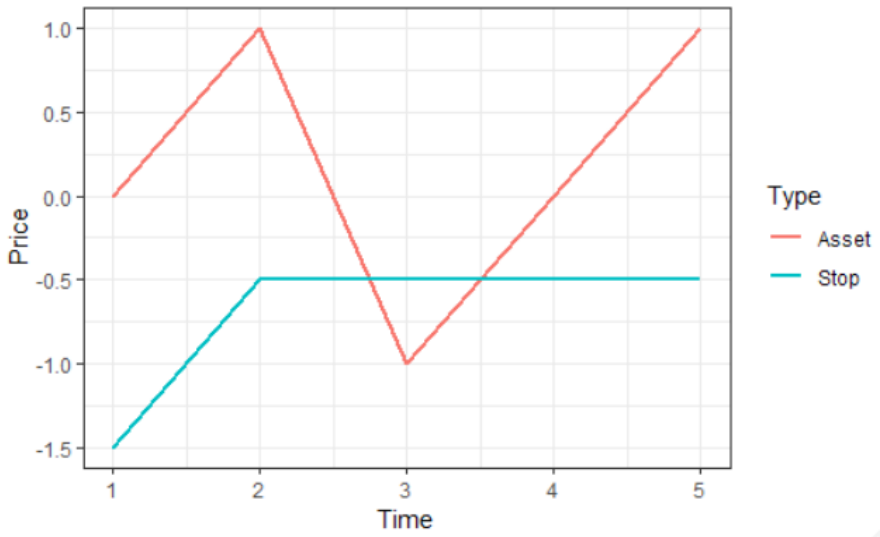

Figure 5.13: Illustration of a correctly predicted upward samples but lost in a trade.

Triple-Barrier Forecast

On the other hand, the strategy with the triple-barrier forecast model significantly outperformed its binary direction counterpart, as well as the other two baseline

strategies. Here the concern is whether this exceptional performance is due to the bull market, the carefully designed trading strategy, or the forecast model itself. But the strategy with random signals has acted as the good control, such that we could exclude the other factors and attributes the success to the forecast model.

Especially, the winning rates of all stocks are well above those from the binary direction strategy and the control strategy, scoring an average 54.7%, significantly higher than 44.2% for its binary direction counterpart. This might be explained as follows. Firstly, the lower bound condition in triple-barrier label induced stop-losses in the strategy definitely, the precision favouring approach in model validation could have possibly prevented a significant number of stop-losses from happening. Secondly, for the upper bound predictions to be correct in a certain percentage, it must reach the 3.0 ATR level without touching the -1.5 ATR level within future 10 days. The samples tend to rise straight to the safety region of 1.5 ATR without touching the rising stop-loss. Thirdly, as our hypothesis suggests, the clever triple-barrier label design might have well enabled the more efficient classifier for the task.

However, we noticed that there were decreases in profitability under the strategy with this forecast model, especially for the two best-performing stocks, MSFT and AMZN.

This can be hinted from the significantly lower number of trades under this strategy, as shown in table 5.8. The strategy behaved more conservatively in buying, and this could have been a disadvantage for the bullish stocks, such as MSFT and AMZN. But this suggests that the precision we have emphasised on the forecast model design might have well transformed into the caution and precise trading strategy. Indeed, the lower risk associated could be the even more important success factor.

Table 5.8: Number of Trades Won and Lost per Strategies

Symbol	Trades Won	Trades Lost	Total Trades	Winning Rate
Triple-Barrier	136	113	249	54.71%
Bi-Direction	195	239	434	44.91%
Random	196	240	436	45.01%

Our results could be a piece of evidence that the Efficient Market Hypothesis (EMH), even in its weak form, does not hold for these stocks during the period from 2016 to 2018. It is probably that there is still excess information from the historical market data, that could be exploited to forecast the future market. However, innovative approaches, such as deep learning and label engineering, have to be considered in order to take the edge form that data.

Limitations and Recommendations

Despite the initial success, we believe the trading performance has not yet been maximized. Especially, the lower bound level and the upper-to-lower bound ratio in the triple-barrier label, as well as the stop-loss level in the trading strategy, were chosen arbitrarily. With different combinations of these attributes, different winning rates and profit-to-loss ratios should be obtained. According to the equation in (3.14), this could pose a direct impact on the expected risk-adjusted return. Ones could study more extensively the combinations of these settings and find out the optimal values.

Moreover, we observed that the portfolio value among all stocks dropped with the stock prices during the turmoil period after mid-2018 too, which suggests the triple-barrier strategy could not hedge the downward trends. This could be resolved to a two-state Hidden Markov Model (HMM) [31]. HMM is a Bayesian network model that takes a sequence observation data as input and infers the most probable states in each time point. By inputting the sequence of daily returns of stocks, it is expected to infer whether a period is bullish or bearish for the stocks. By treating this as a regime filter, our strategies could have avoided most of the losing trades during the bearish periods. In addition, besides taking only raw market data, the HMM shares an advantage that it does not suffer from overfitting because it takes no hyper-parameters besides the

number of states, which is simply 2.

Other possible improvements in the trading strategies might include sizing and portfolio optimization. Sizing takes the variation of position sizes into account. More specifically, the strategies could buy more stocks whenever the forecast model is more confident about a signal. While portfolio optimization is about including multiple stocks in a portfolio and cancelling off the ups and downs among stock prices, in order to produce a performance of higher stability and lower risk in the end. Moreover, ones could consider other types of strategies such as the long-short strategies [21][22], which buy the best-performing stocks and short-sell the worst.

In regard to the forecast model, one of the requirements for applying deep learning is the enormousness in data. However, due to the non-stationary nature in financial markets, we cannot include data dated back too long along, otherwise, it is just irrelevant to the samples we predict. This problem could be resolved in two approaches. The first one is to study the stock data with higher frequency timeframe [20][14][23]. For example, if the minute data is studied instead of the daily one, then there would be 480 times more samples, assuming daily trading hour is 8. The second approach relies on unsupervised learning, which involves two stages in building the forecast model.

The first stage uses unsupervised deep learning techniques, such as Sparse Coding (SC) [32], Autoencoder (AE) [15][24], and Restricted Boltzmann Machine (RBM) [21], to build an unsupervised learning model that extract useful features from unlabelled data. The unsupervised learning model is allowed to be trained with enormous historical samples irrelevant to the testing data. Then the second stage builds a shallow supervised learner, such as Random Forest or Support Vector Machine models, to predict the unknown testing labels based on the unsupervised features extracted from the recent and relevant data.

Finally, a reminder has to be made that historical simulation results might not represent future trading performance. It is because financial markets are dynamic and always subject to change by participants' behaviour. When a successful strategy is published, its edge could be diminished as being exploited increasingly. In addition, our strategies are obviously long bias, their performance under long-term bear markets is untested. Moreover, some assumptions have been made in our simulations that could be thought as unrealistic, although the effect was estimated to be minor because our trading frequency is low. These assumptions include transaction costs and market impact being ignored, stocks being bought with closing prices, and the allowance to trade with any discretionary units.

6. Conclusion

We have presented an empirical study of financial market forecast model with the combination of deep learning, raw market data and the triple-barrier label [18]. The model, when worked with a trailing stop-loss trading strategy, outperformed the market in terms of reduced risk, stability, and risk-adjusted returns for the stocks and period considered. This provides a new piece of evidence that the Efficient Markets Hypothesis (EMH) [1] does not always hold. The research suggests that excess information might still exist in the market, even in the form of past trading histories, but it is probably not exploitable unless a new approach is attempted. Indeed, financial markets are non-stop evolving, profits or predictability of an idea is soon diminished. Ones have to innovate to keep up the fast pace, and we hope this dissertation provides an inspiration.

In retrospect, financial markets forecast remain as a notoriously hard problem. Its noisy and non-stationary data making the deep learning model training a very different experience than in other problems, which data usually has more signal and irrelevant to the time changes. Take the raw pixels of a cat as an example, it is still predicted as a cat no matter 10 years later or before, but this is not the case for financial markets.

Therefore, testing the models' generalizability to an unseen period is especially important. Unfortunately, it was often ignored in prior research.

Moreover, enormousness in data is especially important for deep learning, rather than other algorithms in machine learning. By knowing this, the research should have conducted with market data of higher frequencies. Besides this, lots of ideas were brainstormed during the experiment development, which should immediately improve the trading performance, but they were excluded to keep the theme focus. The easiest implementing ones include ensemble learning, Markov regime filter [31], long-short trading strategy [21], [22], and parameters optimization, which could be left for future works.

In a broader view, there are many artificial intelligence systems that involve decisions making. This research demonstrates the two-step approach of supervised learning, that is “predicts and acts”. Some others attempt the one-step reinforcement learning (RL) methods, which instead directly infer the optimal sequence of actions [11], in order to avoid the inefficient conversion of predications to actions. However, training of such RL models could be more challenging because the significantly larger state-action spaces are needed to explore, and most of the initial actions learned could be irrelevant

to the optimal action. Perhaps ones might consider falling back to the two-step approach and paying more attention to label engineering and execution systems as demonstrated in this research.

7. References

[1] B. G. Malkiel, and E. F. Fama. "Efficient capital markets: A review of theory and empirical work." *The journal of Finance* 25, no. 2, pp. 383-417, 1970.

[2] R. Sullivan, A. Timmermann, and H. White. “Data-Snooping, Technical Trading Rule Performance, and the Bootstrap.” The Journal of Finance, 54, pp. 1647–1691, 1999.

[3] S. Schulmeister. “Profitability of technical stock trading: Has it moved from daily to intraday data?” Review of Financial Economics 18, pp. 190–201, 2009.

[4] Y. Dai, T. Zhang. “Machine Learning in Stock Price Trend Forecasting.” Stanford University, 2013.

[5] A. Ng. Coursera, [Online]. Available: https://www.coursera.org/learn/machine-learning [Accessed 15 Sep 2019].

[6] B. Krollner, B. J. Vanstone, and G. R. Finnie. "Financial time series forecasting with machine learning techniques: a survey." In ESANN. 2010.

[7] H. Lee et al., “Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations,” in *Proc. 26th Annu. Int. Conf. Mach. Learn*, pp. 609–616, 2009.

[8] A. Graves, A.R. Mohamed, and G. Hinton, “Speech recognition with deep recurrent neural networks.”*, Speech Signal Process*, pp. 6645–6649. IEEE, 2013.

[9] D. Silver et al. "Mastering the game of Go with deep neural networks and tree search." *nature* 529, no. 7587, pp.484, 2016.

[10] F. Chollet. "Deep learning with Python." (2017).

[11] R. S. Sutton, and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[12] J. Heinrich, and D. Silver. "Deep reinforcement learning from self-play in imperfect-information games." *arXiv preprint arXiv:1603.01121*, 2016.

[13] G. Tesauro. "Temporal difference learning and TD-Gammon." *Communications of the ACM* 38, no. 3, pp. 58-68, 1995.

[14] J.F. Chen, et al. "Financial time-series data analysis using deep convolutional neural networks." In *2016 7th International Conference on Cloud Computing and Big Data (CCBD)*, pp. 87-92. IEEE, 2016.

[15] S.J. Guo, F.C. Hsu, and C.C. Hung. "Deep Candlestick Predictor: A Framework toward Forecasting the Price Movement from Candlestick Charts." In *2018 9th International Symposium on Parallel Architectures, Algorithms and Programming (PAAP)*, pp. 219-226. IEEE, 2018.

[16] L. Di Persio, and O. Honchar. "Recurrent neural networks approach to the financial forecast of Google assets." *International journal of Mathematics and Computers in simulation* 11, 2017.

[17] A. Honchar. Medium, [Online]. Available: https://medium.com/@alexrachnog/neural-networks-for-algorithmic-trading-2-1-multivariate-time-series-ab016ce70f57 [Accessed 15 Sep 2019]

[18] M. L. De Prado. "*Advances in financial machine learning*." John Wiley & Sons, 2018.

[19] K. Kamijo, and T. Tanigawa. "Stock price pattern recognition-a recurrent neural network approach." In *1990 IJCNN International Joint Conference on Neural Networks*, pp. 215-221. IEEE, 1990.

[20] M. Velay, and F. Daniel. "Stock Chart Pattern recognition with Deep Learning." *arXiv preprint arXiv:1808.00418*, 2018.

[21] L. Takeuchi, and Y.Y.A. Lee. "Applying deep learning to enhance momentum trading strategies in stocks." In *Technical Report*. Stanford University, 2013.

[22] T. Fischer, and C. Krauss. "Deep learning with long short-term memory networks for financial market predictions." *European Journal of Operational Research* 270, no. 2, pp. 654-669, 2018.

[23] C. Y. Huang. "Financial Trading as a Game: A Deep Reinforcement Learning Approach." *arXiv preprint arXiv:1807.02787*, 2018.

[24] Hu et al. "Deep stock representation learning: From candlestick charts to investment decisions." In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2706-2710. IEEE, 2018.

[25] I. Goodfellow, Y. Bengio, and A. Courville. "*Deep learning*." MIT press, 2016.

[26] T. Dozat. "Incorporating nesterov momentum into adam.", 2016

[27] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. "On the importance of initialization and momentum in deep learning." In Proceedings of the 30th International Conference on Machine Learning (ICML-13), pp. 1139–1147, 2013.

[28] T. Tieleman and G. Hinton. "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude." Cousera: Neural Networks for Machine Learning, 4, 2012.

[29] D. Kingma and J. B. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[30] S. McNally, J. Roche, and S. Caton. "Predicting the price of Bitcoin using Machine Learning." In *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, pp. 339-343. IEEE, 2018.

[31] Y. Yuan and G. Mitra. "Market Regime Identification Using Hidden Markov Models." *Available at SSRN 3406068,* 2016.

[32] R. Rosas-Romero, A. Díaz-Torres, and G. Etcheverry. "Forecasting of stock return prices with sparse representation of financial time series over redundant dictionaries." *Expert Systems with Applications* 57, pp. 37-48, 2016.