Stock Price Prediction App using Machine Learning Models Optimized by Evolution

[RO4] Final Year Project Report

By

CHAU Tsun Man,

SUEN Heung Ping,

TO Cheuk Lam,

WONG Cheuk Kin

Advised by

Prof. David ROSSITER

Submitted in partial fulfillment of the requirements for COMP 4981 in the Department of Computer Science, The Hong Kong University of Science and Technology, 2018-2019

Abstract

The project aims to provide retail investors with a third-party investment mobile application to navigate through the stock market. This is achieved through the use of machine learning and mobile web technologies. Several stock price prediction approaches and models are developed including dense, feedforward neural networks, recurrent neural networks, simple linear regressions, and linear interpolations. Model architectures and hyperparameters are optimized and automatically searched by evolution algorithm. Promising results are found for trend prediction. The project serves as a foundation for democratizing machine learning technologies to the general public in the context of discovering investment opportunities. It paves the way for extending and testing out new models, and developing AutoML in the financial context in the future.

Table of Contents

1. Introduction

1.1 Overview

There are over 2.2 million Hong Kong stock investors, who contributed about 15% of the cash market trading value in 2016. The total cash market trading turnover is around HK$1.6 trillion. In particular, retail investors have made buy or sell investment decisions worth a total turnover of $240 billion for the year of 2016 [1]. In Hong Kong, there are a lot of investment decisions that involve a large sum amount of money being made.

Retail investors spend a lot of time finding investment opportunities. Wealthier investors could seek professional financial advisory services, but for typical retail investors, the costs are prohibitive. Thus, retail investors have to figure out the market themselves and make informed decisions on their own. This makes investment very stressful in modern societies.

Unfortunately, humans are irrational in nature. Without quantitative, data-driven models, decisions get swayed by cognitive biases or personal emotions, resulting in unnecessary losses. Even if investors are cautious enough, most do not have sufficient skills to process a huge volume of data required to make good judgments. Institutional investors rely on sophisticated models supported by technologies to avoid traps, but retail investors do not have access to such technologies and often find themselves falling behind the market.

Without access to quantitative and data-driven models, one obvious approach retail investors could use to evaluate the market is through simple indicators, for example, linear regression and exponential moving average (EMA) (Figure 1.1). Two important indicators are 20-day EMA and 50-day EMA. When the 20-day EMA rises above the 50-day EMA, the stock is likely to trend upward, and vice versa. Another obvious approach retail investors might use to predict the stock market is to draw a linear regression line that connects the maximum or minimum of candlesticks.

Figure 1.1 Linear regression method to evaluate and predict the market trend

Inspired by the increasing popularity of deep learning algorithms for forecasting application, these algorithms might serve as potential tools to find hidden patterns in the trend of stock prices, this information could be useful to provide extra insights for retail investors when making investment decisions. Therefore, this final year project aims to investigate the

usefulness of deep learning algorithms in predicting stock prices and democratize such technologies through an easy to use interface for the general public.

1.2 Objectives

1.2.1 Introduction

The ultimate goal of our application is to serve retail investors as a third party investment tool that uses machine learning to help them navigate in the fast-changing stock market. The project aims to introduce and democratize the latest machine learning technologies for retail investors. No prediction is 100% accurate. Therefore, the upper bound and lower bound of the stock prices will be displayed to illustrate the trading range the investors should be looking at. This application serves as a supplementary quantitative tool for investors to see the market at a different perspective with the help of technology.

This project is divided into 2 parts, namely a research component and an application component, aiming to provide retail investors with stock price predictions using different machine learning models in a good user experience way for reference.

1.2.2 Research

This project will investigate how different machine learning techniques can be used and will affect the accuracy of stock price predictions. Different models, from linear regression to dense and recurrent neural networks are tested. Different hyperparameters are also tuned for better performance.

The search space for all neural network architectures and hyperparameter combinations is huge, and with limited time in conducting this project, apart from manually trying different reasonable combinations, the team optimizes the models with evolution algorithm, replicating AutoML techniques from other researches with promising results in the financial context.

1.2.3 Application

This project aims to provide stock price predictions based on the latest machine learning technologies to all retail investors. A mobile web application is developed to provide predictions in an intuitive way. Different models' performance and accuracy can also be compared. The application also serves as another user interface (UI) in visualizing results from the research apart from Jupyter notebooks with lots of tables and graphs.

1.3 Literature Survey

1.3.1 Stock Price Predictions

From the research paper “Machine Learning in Stock Price Trend Forecasting” written by Y. Dai and Y. Zhang in Stanford University, they used features like PE ratio, PX volume, PX EBITDA, 10-day volatility, 50-day moving average, etc. to predict the next-day stock price and a long-term stock price [2]. The machine learning algorithms used in the research are Logistic Regression, Gaussian Discriminant Analysis, Quadratic Discriminant Analysis, and SVM. The accuracy ratio is defined as the number of days that the model correctly classified the testing data over the total number of testing days. With the short term model predicting the next day stock price, it has very low accuracy, the Quadratic Discriminant Analysis is the best among all models, it scored a 58.2% accuracy. With the long term model predicting the next n days stock prices, the longer the time frame, the better in the accuracy for SVM. With a time window of 44 days, the SVM model’s accuracy reached 79.3%. Apart from that, it was found that by increasing the number of features, the accuracy increased. When all of the 16 features were used, the accuracy of the model reached 79%, while it fell to 64% when only 8 features were used, and 55% if only 1 feature was used. Our project will also investigate how the timeframe would affect the accuracy of price predictions of different models. As models have to reach a certain threshold to have significance for the users to work as a reference, it is essential for us to optimize our model to figure out what the optimal parameters and model structure are for our stock price prediction purpose.

The research paper “Predicting stock and stock price index movement using Trend Deterministic Data Preparation and machine learning techniques” written by J. Patel, S. Shah, P. Thakkar, and K. Kotecha for the “Expert Systems with Applications” international journal demonstrated a way to use trend deterministic data to predict stock price movement [3]. They conducted experiments in using 10 technical indicators’ signals as inputs, then they use prediction models to predict whether the stock will go up or down in the coming 10 days, Technical analysis indicators include SMA, EMA, Momentum, Stochastic SK, Stochastic SK, MACD, RSI, etc. The prediction models they have used include ANN, SVM, Random Forest, and Naive Bayesian models. The model outputs “up” or “down” movement signals. Experiments have shown random forest scored the highest performance with 83.56% accuracy with their inputs.

B. Wanjawa and L. Muchemi demonstrated the potential in predicting stock prices using ANN, as shown in the research paper “ANN Model to Predict Stock Prices at Stock Exchange Markets” [4]. They used 70% of the training data to predict the stock prices for the next 60 days. Through optimizations, they were able to predict the actual closing prices within 0.71% mean absolute percentage error (MAPE), with the highest variance -3.2% among all of the 62 days. This demonstrated a high potential for using machine learning to accurately predict stock prices. This is one of the key components in our application where algorithms have to be designed to have high accuracy, such that the platform could be useful for retail investors.

1.3.2 Neural Network

A neural network attempts to learn a function that maps the input features to the output predictions, serving as a universal function approximator [5]. It consists of a network of neurons, each of which represents a weighted sum of inputs. Outputs from neurons are fit into activation functions which introduce non-linearity to the system, and then passed to some other neurons. In a typical dense feedforward neural network, the network consists of layers of neurons stacked together, with neurons between individual layers fully connected.

Optimization of neural networks is usually done through backpropagation with gradient descent, which essentially propagates the error from the output layer back to the input layer, while computing the gradient of the error against each parameter in the process.

1.3.3 Recurrent Neural Network

Recurrent neural network [5] is a type of neural network where connections between neurons allow temporal, sequential information to be stored and processed in the network. One typical architecture is formed by feeding the output of the current unit back to the input with a time delay so that the network can use the information in processing the next input. Various techniques have been developed over the years to train such type of network. One of the popular approaches is backpropagation through time (BPTT) [6], whose central idea is to unroll the recurrent network into a feedforward network, where each layer represents a timestep. Backpropagation with gradient descent could then be performed to optimize the network, just like how we optimize a feedforward network. Unfortunately, it has been shown that

techniques like BPTT result in either vanishing or exploding gradients [7]. Vanishing gradients lead to unrealistically long training time, and sometimes training is infeasible while exploding gradients result in fluctuating weights, which leads to unstable training. Both are undesirable in neural network training. Thus, new training methods and architectures are needed to mitigate the problems.

1.3.4 Long Short-Term Memory (LSTM)

Long short-term memory [8] was first introduced by Hochreiter and Schmidhuber in 1997 to address the aforementioned problems. Long-short term memory tackles the problem of learning to remember information over a time interval, by introducing memory cells and gate units in the neural network architecture. A typical formulation involves the use of memory cells, each of which has a cell state that store previously encountered information. Every time an input is passed into the memory cell, and the output is determined by a combination of the cell state (which is a representation of the previous information), and the cell state is updated. When another input is passed into the memory cell, the updated cell state and the new input can be used to compute the new output.

1.3.5 Gated Recurrent Unit (GRU)

Gated recurrent unit [9] follows the same architecture as long short-term memory, except that it simplifies the design of the memory cell, by reducing the structure to contain only two gates, the reset gate, which controls how much information to forget when taking in the new information, and the update gate, which controls the proportion of cell state updated by the

contribution. Although it has been shown that LSTM is more powerful than GRU [10], GRU has the advantage of lower training time and may perform better on smaller datasets [11].

1.3.6 Evolution Algorithm

Researches have shown that large-scale evolution can auto-generate neural network model architectures and hyperparameters with performance comparable with state-of-the-art human-designed models. In a research in 2017 [12], a large-scale evolution for discovering image classification neural networks was run. It started with a huge population of randomized simple 1-layer models, then slowly evolved the population by removing a poor model and generating a new model by mutating some parameters of a good model in each iteration. After hundreds of hours of running the algorithm with huge computing power, most models in the population achieved state-of-the-art results on CIFAR datasets. In each iteration, only a simple mutation that changed 1 parameter was applied, which allowed searching in a large search space. The paper showed the possibility of finding good models by using lots of computational power to replace human-machine learning experts and has set the foundation of democratizing machine learning with AutoML.

2. Methodology - Design

2.1 System Architecture

The architecture of the system follows a client-server model, where the server and the client are loosely coupled.

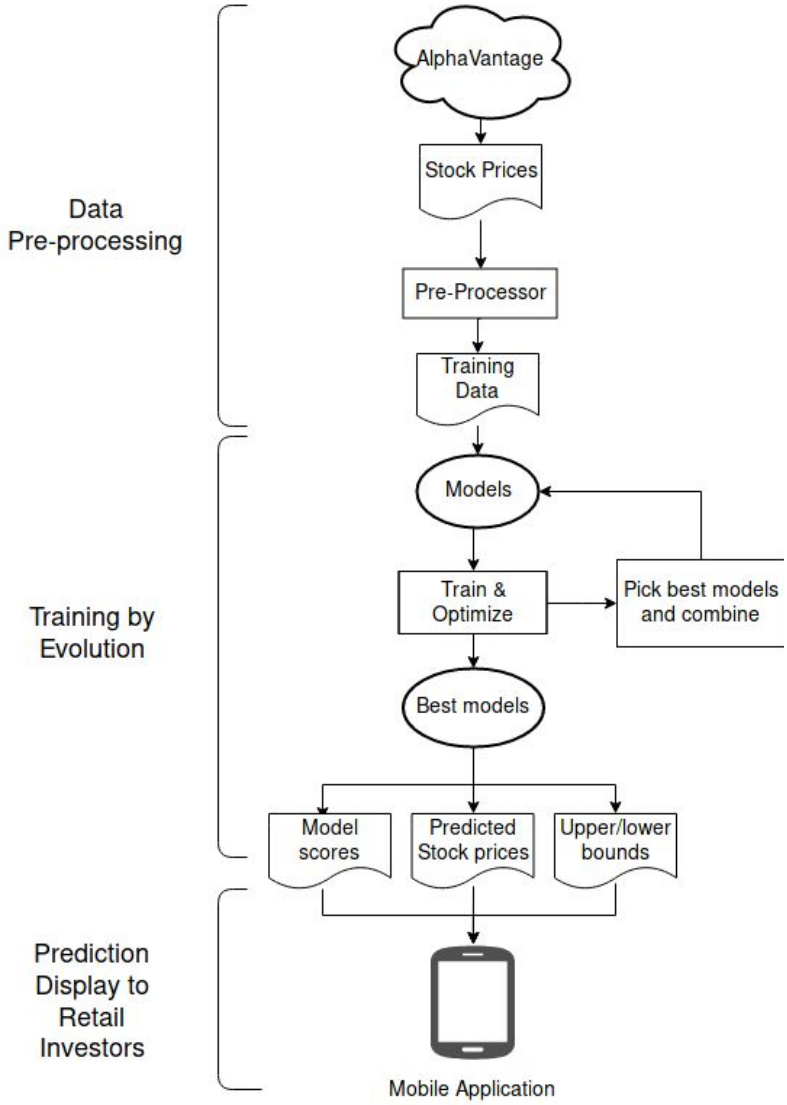


Figure 2.1 System Architecture Diagram

After relevant stock data are retrieved from the third-party data provider through the cloud, the backend pre-processes the data and builds the models. After that, predictions are made and the prediction results will be stored on another cloud, which can be retrieved from the mobile application.

The advantages of the loosely coupled architecture include improved scalability and ease of collaboration. The workload for the cloud which serves the models and the one which serves the mobile application will be very different. One cloud serves the model prediction results, which are simple text files; another cloud serves the mobile application with a lot of rich user content such as images and large UI libraries. Having two clouds to adapt to two different demand patterns is more efficient, especially since cloud providers these days usually serve content on demand.

Also, the separation allows different team members in the team to focus on different parts after agreeing on a common interface. It speeds up development as team members responsible for different parts of the system do not need to take care of the underlying implementation details. Also, it is easier to swap out different components, e.g. to replace the models the team could simply make changes to the backend, while the frontend remains unaffected.

2.2 Research Design

2.2.1 Problem Framing

The problem of the project is set to predict the stock price for the next 10 business days. “10 days” is chosen as the timeframe as short term price movements tend to depend more on trend momentum and price pattern, while long term price movements depend on the fundamentals of a stock (e.g. company management capabilities, revenue model, market demand, macroeconomic factors, etc.).

The loss function of the training algorithm is the mean squared error of the 10 predicted stock prices. The training algorithm or optimizer is set to minimize its value, and it serves as the basic performance metric for comparing different models.

Other scores are defined to provide more in-depth insights on a model predictability performance and finance-domain-based comparisons between models for investors.

Two different prediction approaches are mainly tested, predicting the stock prices for the next 10 days directly and predicting the stock price of the next day 1 at a time. It is suspected that the two different problem framing approaches will result in different abstractions learned hence performance for different use-cases.

As different stocks have very different characteristics and the stock prices exhibit different trends, individual models will be built for separate stocks.

For the project, S&P 500 stocks from different industries are selected. Multiple factors are considered when picking the stocks, including stock price volatility, the absolute magnitude of the price, the respective industries, company size, etc., and stocks exhibiting different characteristics are picked. The stocks are listed as below:

- Alphabet Inc., GOOGL (Technology)
- Amazon.com Inc., AMZN (Technology)
- Apple Inc., APPL (Technology)
- AT&T Inc., T (Telecom Services)
- Boeing Co., BA (Industrials)
- Caterpillar Inc., CAT (Industrials)
- Facebook Inc., FB (Technology)
- General Electric Co, GE (Industrials)
- Harley-Davidson, Inc., HOG (Consumer Cyclical)
- Microsoft Inc., MSFT (Technology)
- Procter & Gamble Co, PG (Consumer Defensive)
- Tesla Inc., TSLA (Consumer Durables)
- Walmart Inc., WMT (Consumer Defensive)

2.2.2 Robust Design

For the research side, the system is designed to be as robust as possible to facilitate model testing. Each model can be defined by a pair of model options and input options, specifying the

model configurations and the inputs it takes. This accelerates the process of testing out different model and/or input configuration combinations.

2.2.3 Data Pre-processing

Raw stock price data is pre-processed before inputting into machine learning models. Pre-processing includes transforming the raw data into a format that models can take from and operate on, most likely feature matrix. It also attempts to extract some features, financial-domain-specific especially, manually to improve results, allowing the model to learn more abstractions.

Two key features are selected as the input. First is a fixed-length list of some raw historical data like stock price and daily percentage change. The fixed length chosen specifies the length of the historical period to look back from today when predicting future stock prices. Referring to the principle of technical analysis, as the stock price reflects all relevant information, a technical analyst would focus on the trading pattern of the stock rather than the economic fundamentals and company fundamentals. Therefore, by getting a period of historical stock prices as the input for the training model, it could be a piece of useful information in finding the trading patterns and hence predicting the trend of future stock prices. Given a set lookback period, it is assumed that the price movement patterns that are predictive would occur in the specified historical period.

The second feature input is arithmetic moving averages. As mentioned in 1.1, one of the obvious approaches for retail investors to identify the trend of the market is through moving averages. With the robust system design, different period of moving averages could be used as the input into the model for stock price prediction, for example, a set of 22, 50, 100, 200 days moving averages, which are commonly used by investors [13].

2.2.4 Prediction Output

As mentioned in 2.2.1, 2 different prediction approaches are tested, which will have different outputs.

For 10-day predictions, there will be 10 output units, resulting in a one-dimensional vector with 10 stock prices, where the *i*-th element represents the *i*-th day stock price prediction.

For 1-day prediction, there will be 1 output unit which is the stock price in the following day. The predicted stock price of will then be the input of the next prediction, to predict the stock price in the second day, the process repeats until all 10 predictions are generated.

2.2.5 Model

Different common neural network models are tested, including dense neural network, simple recurrent neural networks (RNNs), long short-term memory networks (LSTMs) and gated recurrent unit networks (GRUs).

Different model architectures are tested by changing the number of hidden layers, the number of hidden units per hidden layer, and the activation function or recurrent activation function used in each hidden layer.

All recurrent neural networks, RNNs, LSTMs, and GRUs, are set to have the same high-level architecture (Figure 2.2), a stack of recurrent layers by passing the full output sequence to the next layer, followed by a stack of dense layers.

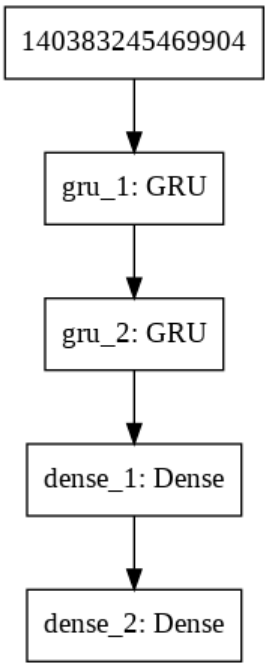


Figure 2.2 Example of the common high-level architecture of recurrent neural networks

Linear Regression on features, as well as trendlines which interpolate the stock prices next 10 days linearly, are also tested.

2.2.6 Model Architecture and Hyperparameter Search With Evolution Algorithm

Designing neural network architecture is challenging, even for computer scientists, researchers and machine learning experts. The team does not have the expertise in designing innovative and suitable architectures that will fit the requirement. Given the huge number of architecture types and hyper-parameters for each model, the search space is basically infinite, so a brute-force approach with grid search would not be practical.

Inspired by the paper [12], this project replicates the evolution algorithm in the context of stock price prediction. The algorithm serves as a heuristic for architecture search, using reasonable and affordable computing power to search for ideal architectures.

The same evolution algorithm was used in the paper [12] to train large-scale image classifiers. The following is the evolution algorithm used, and the corresponding algorithm parameters are defined in Appendix E.

1. Create a population of size POPULATION_SIZE of random simple neural networks.
2. Train all neural networks in the population.
3. Calculate the mean squared error on the test set for each trained neural network.
4. Randomly select 2 networks. Select the one with better performance (lower error) as the parent network, and remove the one with a worse performance from the population.
5. Mutate the parent network to generate a new network and add it to the population.
6. Train the new network.

7. Calculate the mean squared error of the new network on the test set.
8. Repeat steps 3 - 7 for ITERATIONS number of iterations.

Different mutations are used at each step to slowly evolve the population, for example adding a dense layer, changing the number of units in a certain layer or changing the learning rate. For a full mutation list, see Appendix D.

In theory, it is also possible to put the model inputs as a variable into the evolution algorithm, using the algorithm to find the optimal inputs. However, this would increase the search space significantly, and with limited resources, only a certain number of fixed inputs are tried.

2.2.7 Performance Evaluation

2.2.7.1 Motivation

As mentioned in 2.2.1, apart from the mean squared error that a model tries to minimize, different finance-specific scores are introduced to evaluate and compare performance of different models, namely model accuracy score, model trend score and stock buy/sell score. The scores are also designed to convey useful and meaningful messages to help investors understand a stock and make investment decisions.

2.2.7.2 Definitions

In this project, the test set is defined as the last 100 days stock price.

To clearly explain the performance evaluation rationale, the following symbols are defined.

$$P_i : \text{actual price} \quad \hat{P}_i : \text{predicted price} \quad \sigma = stdev(\frac{P_{t+1}}{P_t} - 1) \quad S : \text{set of } Snakes$$

"Snakes" is defined as 10-day disjoint prediction segments in the test set, which will be a set of 10 "snake". It includes the actual prices and the predicted prices for the last 100 days. Specifically, Snakes are defined below:

$$Snakes = \{Snake_i \mid i \in [0, 9] \}$$
$$Snake_i = \{(\hat{P}_{10i+j}, P_{10i+j}) \mid j \in [1, 10] \}$$

It is named as "Snakes" because intuitively the 10-day disjoint segments look like snakes when being plotted on a graph of historical prices.

2.2.7.3 Model Accuracy Score

The first indicator of the performance is the *Model Accuracy Score (MAS)*. It describes the accuracy of the price prediction regarding the actual price. It is a weighted sum of *Model Prediction Score (MPS)* and *Model Direction Score (MDS)*, ranging in [0,1]. A variable α is declared to adjust the weighting between MPS and MDS contributing to MAS. Its formula is defined below:

$$\text{Model Accuracy Score (MAS)} = (1 - \alpha) \cdot MPS \ + \ \alpha \cdot MDS$$

MPS is the average of Snake Prediction Scores (SPS). Each SPS is calculated by the prediction error in each of the 10-day disjoint segments, where the error is basically an average of the

absolute relative change between the predicted prices and the actual prices over the 10 days. It is defined that SPS is 0 If the error is larger than the standard deviation of the stock, as the prediction would have no reference value under this circumstance. If otherwise, a scoring concave upward function is applied to scale the error to a range of [0,1] based on the standard deviation. A concave upward function is applied because the marginal usefulness of the model decreases with a marginal increase in error.

$$\text{Model Prediction Score (MPS)} = \frac{1}{|S|}\sum_{i=1}^{|S|} SPS_i$$

$$\text{Snake Prediction Score (SPS)} = \begin{cases} (\frac{e-\sigma}{\sigma})^4 & \text{if } e \leq \sigma \\ 0 & \text{otherwise} \end{cases}, \text{ where } e = \frac{1}{10}\sum_{i=1}^{10}\left|\frac{\hat{P}_i}{P_i} - 1\right|$$

Meanwhile, MDS is the average of Snake Direction Scores (SDS). Each SDS is evaluated by the alignment of the prediction direction and the actual direction of the stock trend in each of the 10-day disjoint segments. If the prediction has a different direction with the actual direction, it means the prediction is giving a false trend signal to the users. Thus, SDS is 0. Otherwise, SDS would be evaluated based on the direction of the estimation error. In other words, if the prediction is overestimated, SDS is 0.8. Otherwise, it is 1. It is because it is assumed that an underestimated prediction means the model is more reserved and is better off than an overestimating model.

$$\text{Model Direction Score (MDS)} = \frac{1}{|S|}\sum_{i=1}^{|S|} SDS_i$$

$$\text{Snake Direction Score (SDS)} = \begin{cases} 1 & \text{if } sgn(\hat{P_{10}} - \hat{P_0}) = sgn(P_{10} - P_0) \quad \text{and} \quad |\hat{P_{10}} - \hat{P_0}| \leq |P_{10} - P_0| \\ 0.8 & \text{if } sgn(\hat{P_{10}} - \hat{P_0}) = sgn(P_{10} - P_0) \quad \text{and} \quad |\hat{P_{10}} - \hat{P_0}| > |P_{10} - P_0| \\ 0 & otherwise \end{cases}$$

2.2.7.4 Model Trend Score

Another indicator of the performance is the *Model Trend Score (MTS)*. It describes the correctness of the trend predicted by the models regarding the actual price, ranging in [0,1]. Since an accurate model in terms of the degree of price changes is difficult to obtain, sometimes the *Model Accuracy Score* might not be intuitive. As a result, instead of observing the exact changes in prices using MAS, we could look at the trend of the predictions which is easier to be accurate. With *Model Trend Score (MTS),* the users could gain accuracy insight on the future price change of the stock. It is defined as:

$$\text{Model Trend Score (MTS)} = \frac{1}{10}\sum_{i=1}^{10} TS_i$$

Where TS is the Trend Score for *i*-day Prediction. It is the percentage of having a correct trend prediction of price *i* days later.

$$\text{Trend Score for i-day predict } (TS_i) = \frac{\sum_{j=1}^{100-i} f(\hat{P_{i+j}} - \hat{P}_j \ , \ P_{i+j} - P_j)}{100 - i}, \text{ where}$$

$$f(\hat{D}, D) = \begin{cases} 1 & \text{if } sgn(\hat{D}) = sgn(D) \\ 0 & \text{otherwise} \end{cases}$$

2.2.7.5 Buy/Sell Score

An overall Buy/Sell Score is given to the users to indicate the likeness of the stock going up or down to assist the users in making decisions. It ranges in [-1, 1], with 1 means expecting an uptrend, -1 means expecting a downtrend, and 0 means the prediction is inconclusive. Some symbols are defined below:

$$M : \text{set of all models} \quad M' = \{m \mid m \in M,\ MTS_m \geq T\} \quad T : \text{score threshold}$$

A score threshold T is applied to filter out the inaccurate models. The remaining models M' would be used to calculate the Buy/Sell Score by multiplying the *Model Trend Score* of each valid model with its *Trend Direction (TD)* and then averaging them. TD is determined by voting from the directions of the predicted prices in the coming 10 days. It is 1 or -1 if the majority of the predicted prices are higher/lower than today's price. It is 0 when tie.

$$\text{Buy/Sell Score} = \frac{1}{|M'|}\sum_{i=1}^{|M'|}(MTS_i \cdot TD_i)$$

$$\text{Trend Direction (TD)} = \begin{cases} 1 & \text{if } \left|\{sgn(\hat{P}_i - P_0) = +ve \mid i \in [1,10]\ \}\right| > 5 \\ 0 & \text{if } \left|\{sgn(\hat{P}_i - P_0) = +ve \mid i \in [1,10]\ \}\right| = 5 \\ -1 & \text{otherwise} \end{cases}$$

2.2.7.6 Upper Bounds and Lower Bounds for Prediction

Stock prices are volatile in nature and predictions could almost impossibly be 100% accurate. To give investors more information about how the stock price may fluctuate, upper bounds and lower bounds of prediction error range are also calculated. The upper bounds and lower bounds are defined as:

$$upper_i = \hat{P}_i + stdev(\hat{P}_j - P_j)$$
$$lower_i = \hat{P}_i - stdev(\hat{P}_j - P_j)$$

Since the model predicts the stock prices of the following 10 days, i.e. [P_1, P_2, ... , P_{10}], each day of the prediction, P_i follows a different distribution, and the prediction error for each of the *k*-th day is different.

$\hat{P}_j$ is the collection of all *i*-th day prediction generated by the model in the test set, while P_j is the corresponding collection of actual stock prices. The standard deviation of the difference between the two shows how much the model's *i*-th day predictions vary. By adding and subtracting one standard deviation of the error from the predicted *i*-th day stock price to get the upper bounds and lower bounds of the stock price predictions, investors can know about how much the actual price might fluctuate around the predicted price.

2.3 Application Design

2.3.1 User Groups

Users are separated into two groups, normal users and advanced users. For users that would like to know about the historical (test set) performance of a model and more information behind the machine learning models like the architecture and inputs, they can enable advanced user mode in the settings page to view those details in each individual stock page.

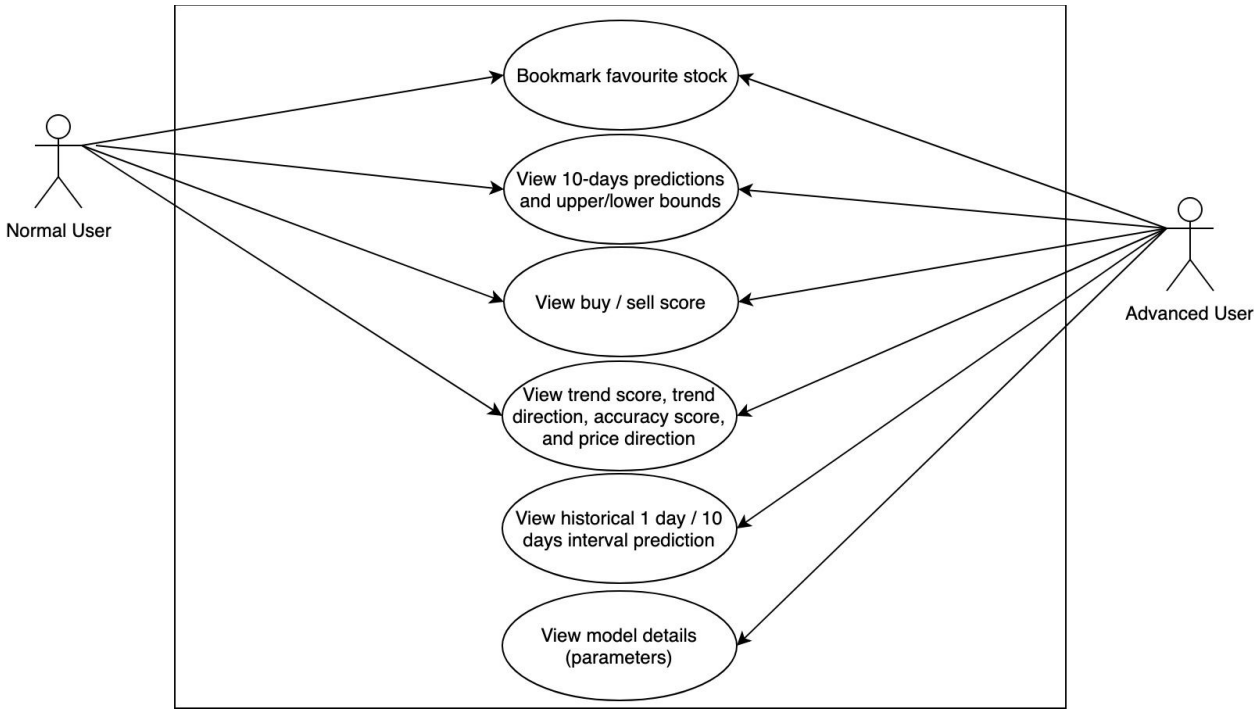


Figure 2.3 Functionality accessible by normal users and advanced users

2.3.2 User Journey

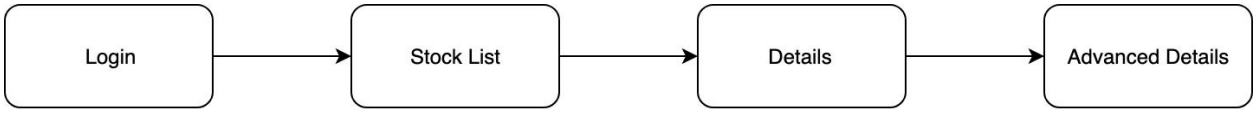


Figure 2.4 User Journey

First of all, users need to login to use the system, as there will be customization options for different users. Since users might not want to create a separate account just for our application, it will more convenient if users can log in with their existing social media accounts. In particular, Facebook login is a good option, since there are over 2 billion users worldwide. Thus, it might be possible to reach a larger market by adopting Facebook login. Only the very basic user information like the user name will be collected by the system.

For normal users (advanced user mode disabled), after logging into the system, they can view the stock list and search from it by stock name. After they find the stock that they are interested in, they can bookmark the stock as a favorite stock for easier access later. After selecting a stock, they can view 3 key information in the details page.

First, the next 10-day predictions and the corresponding upper/lower bounds of the pre-selected best model together with 3 months of historical stock prices.

Second, they can look at the buy/sell score to get an intuitive sense on whether the stock is currently on an uptrend or downtrend based on the predictions.

Third, the user can view the individual trend score, predicted trend direction, accuracy score, and predicted price movement for each individual prediction model.

For advanced users, apart from the 3 key information, they can view 2 additional pieces of information for more in-depth insights, understanding and analysis.

First, they can toggle to view the historical predictions, which helps to evaluate the trustworthiness of different models. There are 2 ways to view the historical prediction performance. For 1-day historical prediction, it could let the user understand how well the model could predict the next day stock price. For 10-days historical prediction (defined as *Snakes* in 2.2.7.2), it could let the user understand 10 trials of how well a model could predict the stock price in 10-day segments. Thus, this information would help the user to get more information to determine whether the model is relevant and accurate enough for their references.

The second additional information is the layers, hyperparameters and model inputs configured for each individual machine learning model. This information would be useful for them to understand the topology of the model.

All data, including stock prices, predictions, historical predictions and all scores will be updated everyday to reflect the latest information and predictions, which allows users to revisit their investment position.

2.3.3 UI/UX

2.3.3.1 Application Screenshots

The following is a set of screenshots of the implemented application.

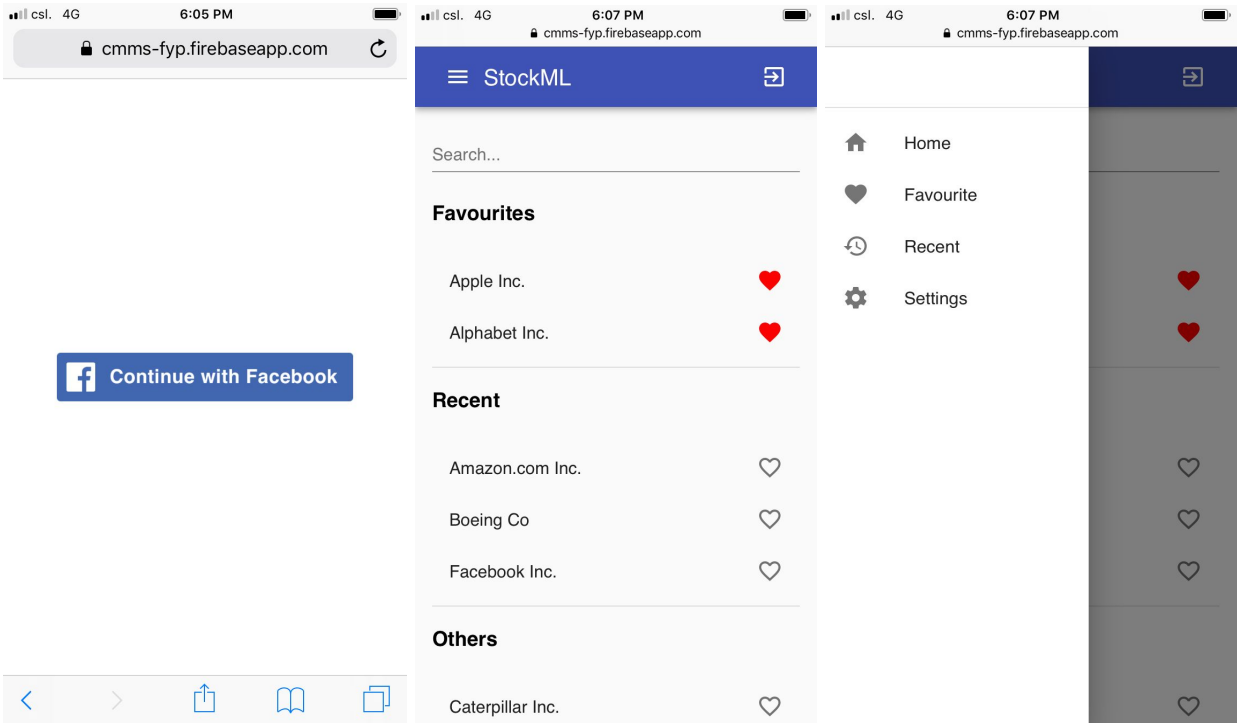


Figure 2.5 Login page (left), Home page (middle), Application drawer (right)

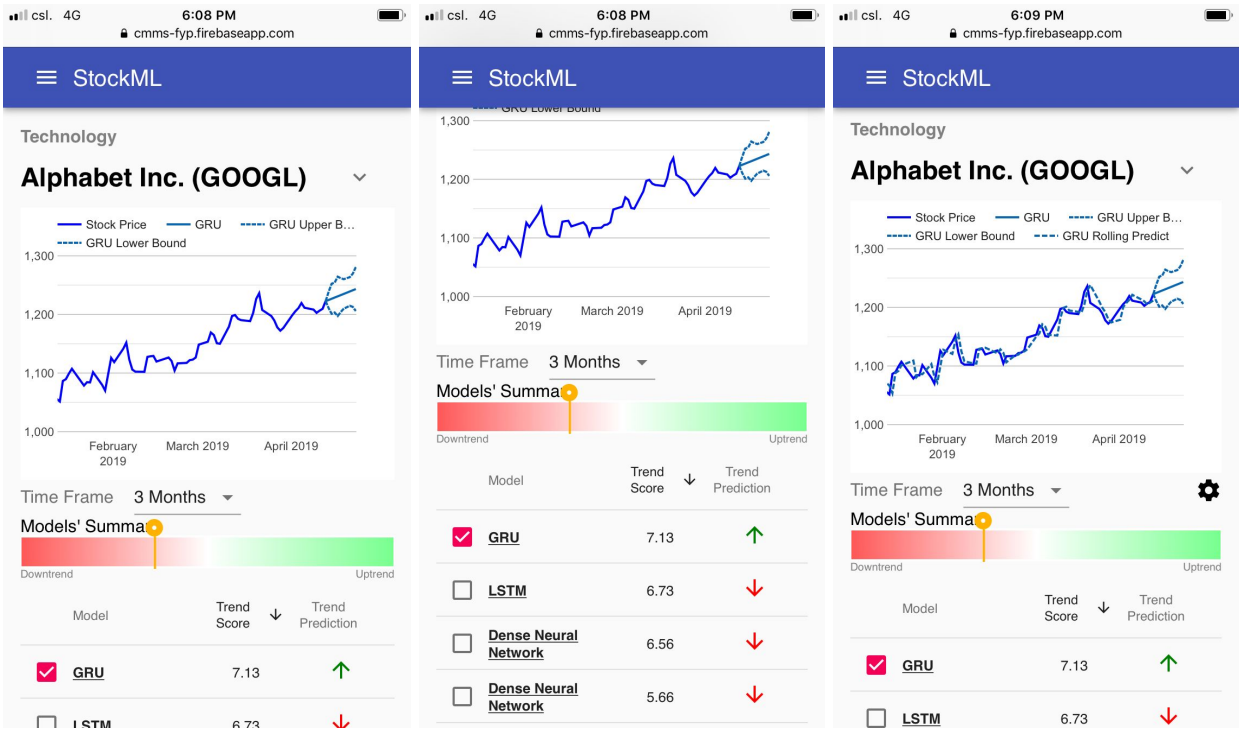


Figure 2.6 Details page - Normal user mode (left and middle), Advanced user mode (right)

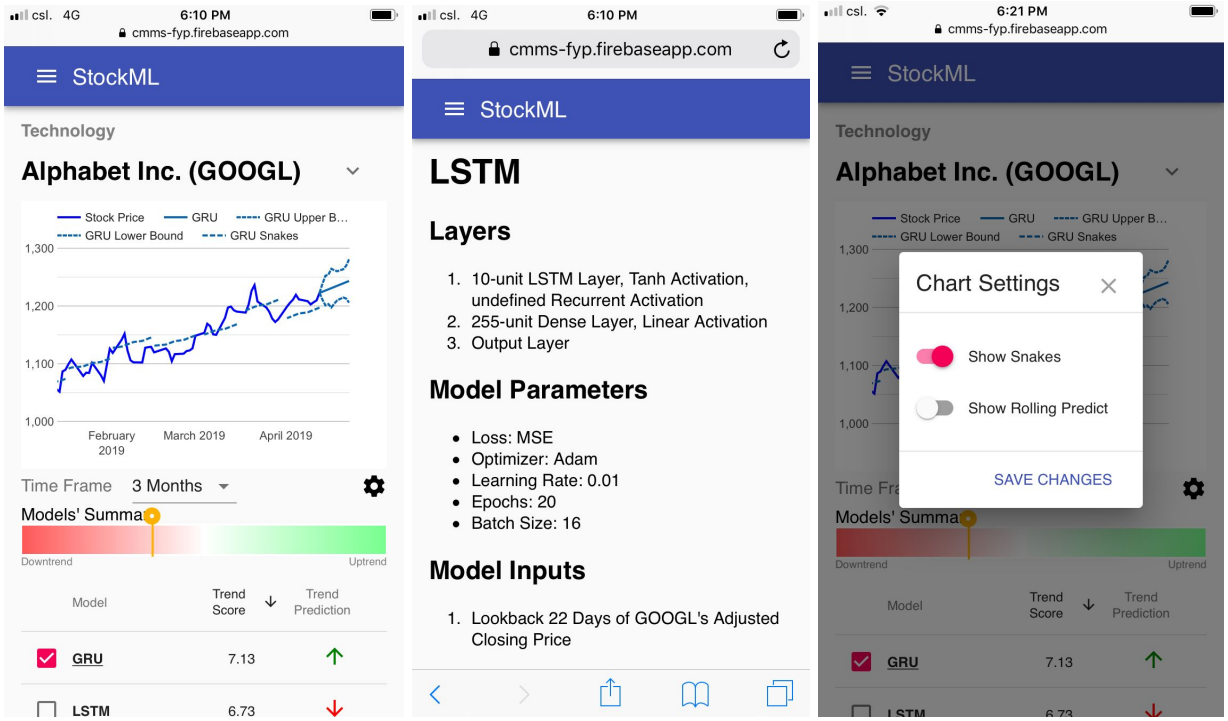


Figure 2.7 Advanced user mode: Snakes (left), Model information (middle), Chart settings (right)

2.3.3.2 Progressive Web Application Motivation

The application is written as a progressive web application (PWA) [14] instead of a native mobile application. The motivation behind this is that the application could be inherently adapt to desktop and mobile usage. It would be more costly to create native desktop and native mobile application separately. The web application can also allow the system to keep only one centralized instance, where information only has to be updated once without any duplicated effort.

A progressive web application is typically implemented as a single page application [15], where pages do not reload entirely like a web page refresh. Instead, the web application only uploads the components as needed as the users interact with the application. It enables smoother, more app-like user experience.

2.3.3.3 Responsive Design

The variety of devices and specifications is also a reason why a responsive-designed web application could be useful to solve such fragmentation problem. The experience between desktop and mobile should be seamless, and the only difference would be desktop has more information displayed than on a mobile screen. By having such configuration, the user's learning curve in adapting to the system on their laptops, tablets, and mobile could be

seamless, not only shorten their learning curve on our system across the platform, but also introduce familiarity with the platform.

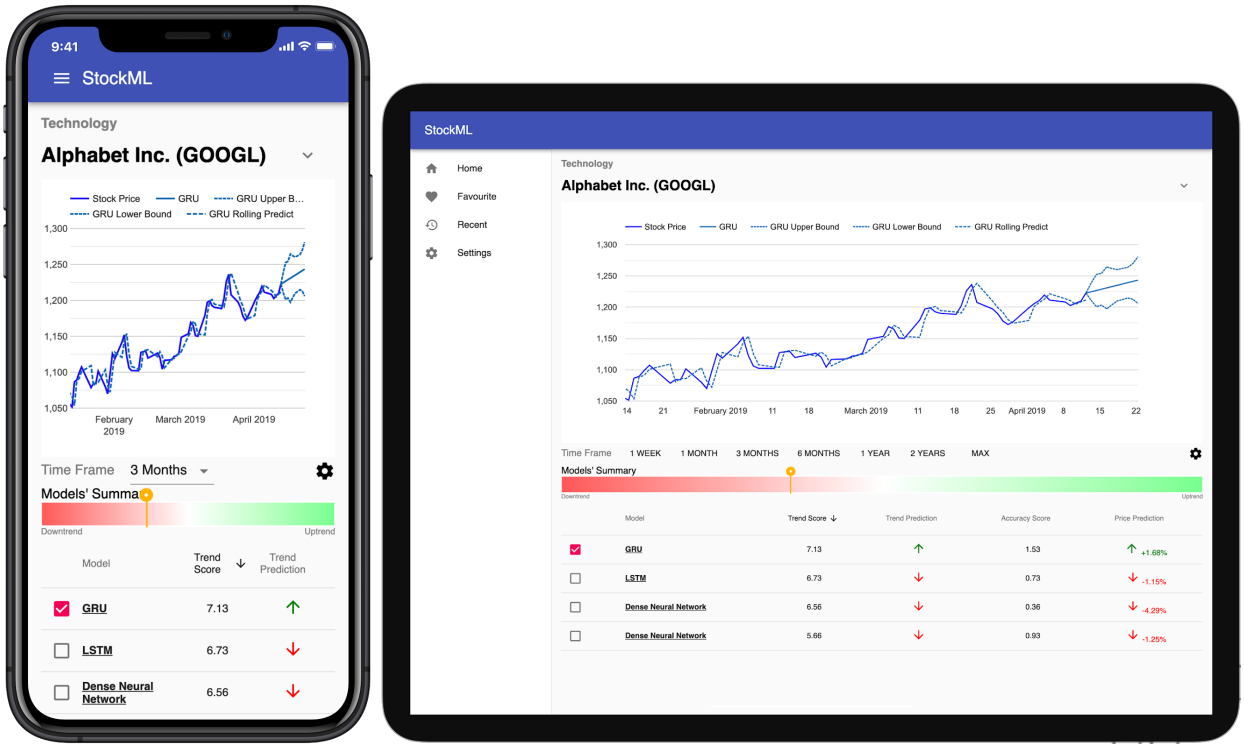


Figure 2.8 Smartphone view (left), Tablet / Desktop view (right)

2.3.3.4 Layout Motivation

The priority of information display should depend on the relevancy of such information from the user's point of view.

In the stock list page, a "favorite" section is placed above the "recent" section, and "others" are placed below the two (Figure 2.5 middle). This design considered the relevance of the individual stock in the user perspective. Favorite stocks should have higher priority as these are their

focused stock that they are interested in or have owned shares. The “recent” section is crucial as it provides quick access for the user to revisit their recent history and find the stocks that are closely relevant to what they have been checking on recently.

Inside the stock details page (Figure 2.6), the stock name with its stock code has a significant color and portion at the top of the display, and a gray colored industry tag is placed above the stock name. When the user clicked on the stock name, the section will expand to display a brief overview of the stock. This design can make the user quickly recognize which stock they are checking on.

As the core component of the application is the predictions with upper and lower bounds that provide insight for the retail investors to review the trend of the stock. The chart is placed just below the stock name and being centered on the screen whenever the page is loaded. This design let the user quickly review the prediction trend and recognize whether the predicted trend would interest them to continue checking out the stock or not.

The chart is also very important for the advanced users to cross-check whether the predictions have reference value according to its past performance. The 10-day interval historical prediction (Figure 2.7 left) and 1-day interval historical prediction (Figure 2.6 right) are plotted on the chart along with the historical stock price when the user enabled such options, this serves the purpose of letting the user know how well the model could predict the trend in the past. The legend labels are placed on the top of the graph to make it crystal clear what does

each color of the line represents. Regarding the interactiveness of the chart, the chart changes with the time frame that the user has selected. When the user hovers on the line of the chart, actual values and its corresponding legend label are displayed, the chart interface makes it easy for the user to validate actual values and demonstrate trust for the flexibility of user able to review on the historical performance of each model.

Following the chart, a buy/sell score is represented by a red-green gradient bar indicating the trend of the stock. It summarizes all the available predictions provided by different machine learning models. This provides a quick overview of the user to evaluate the trend. As it serves as a weighted average according to the trend prediction accuracy and the trend direction of each machine learning model, the red-green gradient bar simplifies all the findings and summarizes such trend predictions into an easy to interpret figure. The middle of the red-green gradient bar is colored as white because it means the stock does not have obvious direction according to the consensus of the predictions of the machine learning models. The color coding makes it obvious for the user to recognize the information at a glance.

The table with checkbox layout is designed for showing detailed results of each machine learning model. The user can correlate the model with its trend predictability and accuracy. This information would be useful for advanced users to evaluate which model topology or search algorithms are useful in providing insightful predictions. Although the trend scores and accuracy scores calculated are in range [0, 1], it is scaled to a 0 to 10 scoring scale, which allows easier understanding and perception. The checkbox interface allows further interactivity with the

chart display. The user could compare and contrast the recent and historical predictions of different machine learning models, and determine which of the algorithm could be as of most useful according to their definition. The table allows sorting according to trend score, trend prediction, accuracy score, and price prediction. This allows the user to prioritize the information according to the metric they would like to investigate. The table is sorted by the trend score in descending order and the best model with highest trend score is pre-selected initially, as that is presumed to be the first model that users care about.

For advanced users that have machine learning backgrounds, the application caters to their needs to look at the layers, hyperparameters, and inputs of the machine learning models. The model details page could provide insights for those users to further investigate the prediction method on their own and let them understand the underlying hypothesis of the machine learning model the application chose to include.

To let the application be smarter and more consistent in terms of user experience, user preferences, including whether a user is an advanced user and which historical predictions to view, are saved on a cloud database, updated and retrieved whenever the user logins and interacts with the preference settings.

A simple loading bar is also included for better user experience, as all data is get from the cloud, and the loading time may vary among users depending on the internet connection.

3. Methodology - Implementation

The implemented application can be accessed at https://cmms-fyp.firebaseapp.com/.

All implemented code can be found at https://github.com/chautsunman/FYP-AI, https://github.com/chautsunman/FYP-pwa, https://github.com/chautsunman/FYP-functions and https://github.com/chautsunman/FYP-server.

3.1 Research Implementation

All machine learning-related code are written in Python. Neural networks are implemented with Keras [16] while linear regression model is implemented with scikit-learn [17].

3.1.1 Stock Price Data Collection

Data is collected from Alpha Vantage Stock Price API [18]. It offers up to 20 years of daily stock price information on S&P500 stocks. A Python script is written to retrieve stock prices of different stocks automatically. The retrieved stock prices are stored as .csv files in a local folder during development and testing. In deployment, the downloaded stock price data will be transformed into a 2D JavaScript array and uploaded to Firebase Cloud Storage immediately. A cron job that launches the data-fetching and data-uploading script is scheduled to run every 8 p.m. (EDT) after NYSE and NASDAQ are closed.

3.1.2 Data Pre-processing

3 Python scripts are written to transform the raw stock prices (.csv files) into feature vectors, for training, predicting and testing respectively. The scripts take the input options and the raw stock prices as inputs and produce the correct features by building the lookback arrays and the moving averages. It concatenates the features into the final feature vectors, which will be passed to the model for training or testing. The 3 scripts share common operations in building a dataset except the output size and the range of dates to build from, so common functions are written to centralize the logic instead of repeating the same index-calculation-intensive work across functions.

NumPy and Pandas are used to build the datasets. Numpy [19] is a library that provides effective n-dimensional array data structures as well as functions for array manipulations. It is frequently used for machine learning tasks because it is much for performant than Python lists, as NumPy arrays are implemented as densely packed lists, instead of a dynamic array where the elements are not stored contiguously.

Pandas [20] is a popular framework for pre-processing time series data. It has various utilities for reading raw input files such as .csv and transforming time series data to the correct format. Pandas uses NumPy as the underlying data structure, so it is very convenient to interoperate between the two.

3.1.3 Model

A model base class is used as a common interface for all machine learning models. All models then have their own model class, specifying model-specific details like methods to build the model, train the model, use the model and save the model.

To decouple model configurations from software code to provide flexibility and robustness and save engineering effort as mentioned in 2.2.2, each model is defined by a JSON object, which specifies the model's architecture and hyperparameters with model options and the model inputs with input options. A corresponding model can then be created by passing the object to the model class constructor.

The model options specify which machine learning model to use, and the hyperparameters for the model like the number of hidden layers, the number of hidden units, activation functions used, as well as optimization algorithms and loss functions. Some example model options are in Appendix A.

Apart from model configurations, the input can also vary, as there are many possible features that could be added to or removed from the feature vectors. The input options specify the features input that a model should expect, like the number of previous stock prices as features and different moving averages. The input options are related to a model in terms of the input format. All neural networks built in Keras requires the input tensor shape for layer shape

inference during model building, a Python function is written to calculate the input shape for a given input option. Some example input options are in Appendix B.

3.1.4 Training

In training, a randomized initial model is first generated from the model options definition. A training set is generated by the build training dataset script, which generates the training set features from the input options and the raw stock price data. Then, the data is fed into the model for training.

3.1.5 Saving Trained Model

All trained models are saved for predicting stock prices in the future. Keras models are saved in h5 format, and scikit-learn models are saved with a Python library named pickle. A dedicated saving format is designed (Appendix C), such that same models (same hash for same model options and input options) for different stocks are saved in the same directory with no collision.

3.1.6 Predicting Stock Price

When predicting stock price, the saved model will first be loaded. Then, a feature vector specified by the input options is built with the build predict dataset script, which is the same as the build training dataset except it returns a flatten 1D feature vector. The feature vector is inputted into the model to predict stock price. For 10-day predict, the predictions are directly outputted. For 1-day predict, the predicted stock price is appended to the raw dataset as if it

happened before, then a new feature vector is generated for predicting the stock price for the day after, the process is repeated to predict the stock prices for all next 10 days.

3.1.7 Performance Evaluation

Each model is evaluated on the test set. A test set can be generated by the build test dataset script, which could generate either a full test set for predicting the last 100 days stock price in 1-day or 10-day disjoint intervals.

3.1.8 Model Score, Buy/Sell Score

Functions are written to calculate different scores for users, 1 for calculating model trend score, 1 for model accuracy score, and 1 for buy/sell score. For parts that share the same calculation just with different offsets, helper functions are written to separate the main calculation function from the repeating steps.

3.1.9 Save Predictions

For each stock, a prediction file can be generated from the save predictions script. It includes all the data and results that the application needs to display, including all 10-day predictions from all models, both 1-day predict test set and snakes test, and the model options and input options for each model. The saved predictions file is then saved to Firebase Cloud Storage and served to the application. During development, the saved predictions file is saved in a local directory.

3.2 Model Architecture and Hyperparameter Search With Evolution Algorithm

3.2.1 Running Evolution Algorithm

All training is done in the Jupyter notebook environment. With hardware limitations, apart from each team member's own computer, Google Colaboratory [21] which provides an easy-to-use Jupyter notebook environment and free GPU service is also used to train models and run the evolution algorithm.

Each team member is responsible for running the algorithm for a different stock. Since the evolution algorithm is a computing power intensive algorithm, training hundreds of neural networks, with the limitation in resources, it is impossible for the team to hold up their own personal computers entirely for the training job. Google Colaboratory also puts limitations on the free resources it provides, and could not be used to run the algorithm day and night unmonitored. Therefore, a checkpoint-like design is implemented, each person can run a certain number of iterations depending on their time and resource availability, then save that run result in a last_run.json JSON file, which could be loaded next time to pick up from where the algorithm left off and continue with further iterations. The team is also encouraged to run the algorithm in small batches, and frequently checkpointing the algorithm state, avoiding runtime errors and losing all results after running the algorithm for a long time.

3.2.2 Algorithm Data Collection for Analysis

Detailed algorithm and training data are gathered for analysis. All errors at each iteration are stored, resulting in a 2D iteration-by-population-size array, which is used to analyze the population evolution and algorithm convergence.

The error and the model options of all neural networks in the last population are also stored for getting the best model after running a lot of iterations and as a checkpoint for the small-batch running.

The Tensorboard log [22], which includes the neural network graph details and training details like loss over epochs, is also stored for every model for deep analysis using Tensorboard. A simpler network diagram of layers is also stored for every model.

The best model after running all iterations is saved, as re-training the model with the same model architecture and hyperparameters will also result in a different model with different predictions. All models are then manually copied and merged to the common saving format as introduced in 3.1.5.

3.3 Server

3.3.1 Flask

For local development and testing, the Flask micro web framework [23] is used to serve local saved data like raw stock price data and saved predictions file. It is written in Python and integrates well with the existing backend architecture. Moreover, it has a much more lightweight interface than other popular Python web frameworks such as Django and allows easier implementation of application programming interfaces. The Flask server simply serves the prediction result JSON files to the front-end application.

3.3.2 Firebase Cloud Storage

For actual deployment, Firebase is used instead. The prediction results are stored in Firebase Cloud Storage [24], which provides APIs for mobile clients to access the results as JSON objects directly.

3.4 Application Implementation

3.4.1 Stock Information Collection

Company information is collected from the IEX Stock API [25]. A Firebase Cloud Function [26] is written to get the data from the API and store it in Firebase Cloud Firestore [27], which the application will access through another Firebase Callable Cloud Function.

3.4.2 React

The application is developed in React. React is a library for building user interfaces declaratively [28]. A distinctive feature that gives rise to its popularity is the ability to build interfaces by combining different components. In this case, the UI was broken into various components, e.g. the application drawer and the stock list. It enables the separation of concerns and the division of work among team members effectively. Additionally, it also minimizes the effort to manage UI state changes, as there is no need to implement event listeners to watch for UI state changes. Last but not least, it does not directly manipulate the DOM tree. Instead, changes are written to a virtual DOM. React then finds what needs to be updated and optimizes how the DOM tree should be updated, which often results in better performance than doing manual DOM manipulations.

3.4.3 React Router

To navigate through the single page application, React Router [29] is used to define the routes of the application. The library provides utilities to manage browser history and switch between different sections of the application.

3.4.4 Redux

With the application's growing size, common states are often needed in multiple components. As different components make changes to the states, the states often become unmanageable as it gets hard to trace how the states are being manipulated from different components, and sometimes states across different components are not properly synchronized.

Redux [30] provides a framework and utilities for centralizing the management of UI states. In redux, a centralized container was used to store all the states needed across different components. In addition, a set of actions, which define changes to the states, and a set of reducers, which define how the new state should look like given a particular action specified. After that, individual components only need to dispatch an action to centralize changes to the UI states.

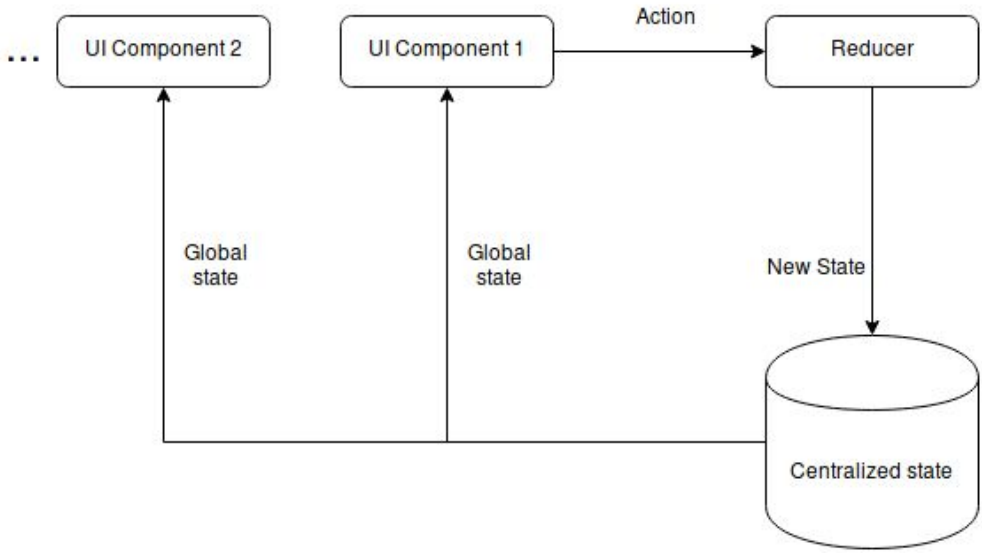


Figure 3.1 Redux Architecture

3.4.5 Immutable.js

When using Redux, developers need to make sure that a reducer does not mutate its arguments so that there will be no unintentional side effects. That means a new object needs to be created every time states are updated. To start with, it is difficult to ensure that objects are not mutated because JavaScript objects are mutable by default. Also, repeatedly creating new objects is inefficient. Immutable.js [31] is a library providing immutable, persistent collections that could be created and updated in an efficient manner. This was done through structural sharing, Using Immutable.js in combination with Redux guarantees that objects could not be modified unintentionally, without the performance penalty of repeatedly creating new JavaScript objects.

3.4.6 Material UI

Traditional web application development relies on setting styles on individual tags and pages with CSS. It is challenging to provide a consistent visual experience and theme to users.

To provide a consistent user experience to users, the popular Material Design [32] developed by Google was adopted, as it defines a set of layouts, colors, typography and behaviors for mobile websites. For ease of implementation, the Material UI library [33] is used, which contains a large collection of developed Material Design themed React components.

3.4.7 Google Charts

Google Charts [34] is used to plot the stock prices and the predictions. It provides a simple and separate set of APIs that does not depend on other libraries while maintaining customizability. Also, it integrates nicely with the user interface which follows the Material Design, which improves the overall user experience.

3.4.8 Facebook Login via Firebase Authentication

To facilitate user logins, the popular Facebook login service [35] powered by Firebase Authentication [36] is used, which provides a set of rich set of APIs to interact with a range of authentication providers, including Facebook Login, and integrates well with other Firebase services that the system relies on. It is also convenient to add other authentication service providers in the future.

3.4.9 Components

The whole application is broken down into the following hierarchy of components.

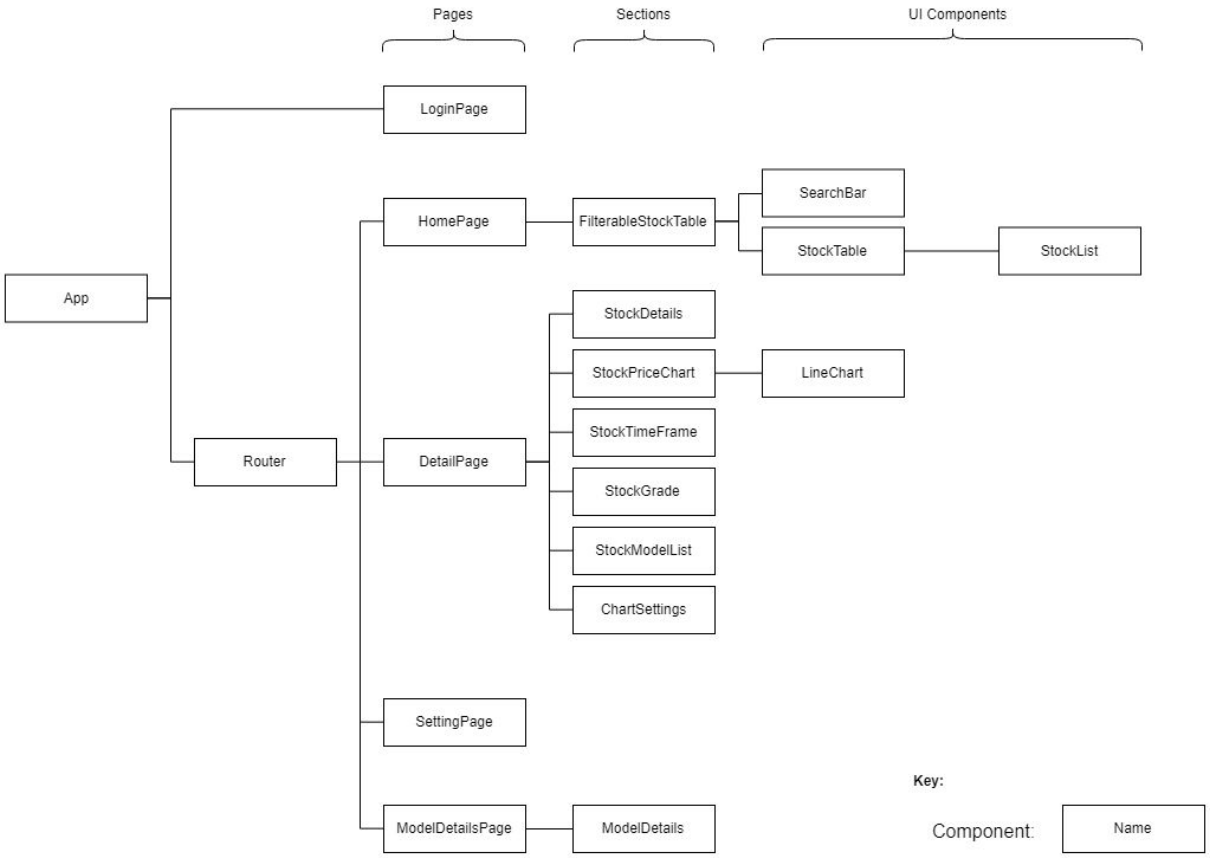


Figure 3.2 Component Diagram

At the top level, the App component brings everything together. If the user is not logged in he/she will be directed to the login page (LoginPage).

The router component (Router) controls where the user will end up at, including the home page (HomePage), which is the default starting point for users with a list of stocks and a search bar, all broken down into separate components.

Other components include the details page (DetailPage), which is where details about a stock price, including the stock price chart, a list of models with a model score attached to each model, along with a buy/sell score that indicates the overall predictions for whether the stock should be purchased or sold, are included in.

4. Methodology - Testing

4.1 Unit Test

The unittest module from Python [37] is used to implement all unit tests, as it is available by default in Python and integrates well with existing Python codes.

Unit tests are done for the build dataset script, which transforms the raw input data into feature vectors usable for training and testing, as well as model score calculations. Unit tests are conducted because the components are error-prone, calculation intensive. Also, they exhibit garbage-in-garbage-out properties, that the model will be completely wrong if it receives the wrong input, and if the model scores are wrong, the final buy-sell recommendation will be totally incorrect.

In particular, unit tests are written for the functions to build the dataset for training and prediction and the function to build the snakes. Combinations of input options are tested, including n-day stock price lookback as well as n-day moving average. Correctness is ensured by asserting the feature vectors' shapes, as well as starting and ending elements.

For model score calculation unit tests, different scenarios are emulated, including the case when the model accurately predicts all the stock prices, the case when the model predicts all the stock prices wrongly by a very large magnitude, the case when the model predicts the trend

correctly but underestimates the trend, as well as the case when the model predicts the trend correctly but overestimates the trend.

4.2 Tools Used for Testing

Various tools have been used to assist in the development of the mobile application. In particular, Chrome Mobile Emulator is used to simulate the mobile view while developing the mobile application on desktop/laptop computers. After the application is deployed to the cloud, mobile phones with different operating systems and browsers, including Google Pixel running Android 9 (Google Chrome) and iPhone 7 running iOS 12.1 (Safari), are used to verify the user experience is consistent across different devices with different resolutions.

5. Methodology - Evaluation

The project's objective is to provide a third-party investment tool to investors with democratized machine learning technologies. The success of the project is primarily determined by two factors, namely, whether the investment tool provides useful, accurate stock price predictions to investors, and whether investors can use and understand the predictive information provided by the machine learning technologies. The first factor is evaluated by the model scores described in 2.2.7. However, the evaluation of the second factor is based on user experience. External users have to be involved in the evaluation. For this purpose, hallway testing is used.

Hallway testing involves allowing users who have not been involved in the development of the project to test the application and give constructive feedbacks about users feel about the application. Users participating in the tests are asked a set of questions about the usability and whether they understand what the information presented by the mobile application. This would give indications about whether the democratization of the machine learning technologies succeeds.

6. Findings

All results and findings graphs can be found in a Google Colaboratory notebook at https://colab.research.google.com/drive/1GYuxbYywhN8-_D3eycsiQ-iYLzv-YjXq.

6.1 General Findings

The following are some general findings from testing out different machine learning models.

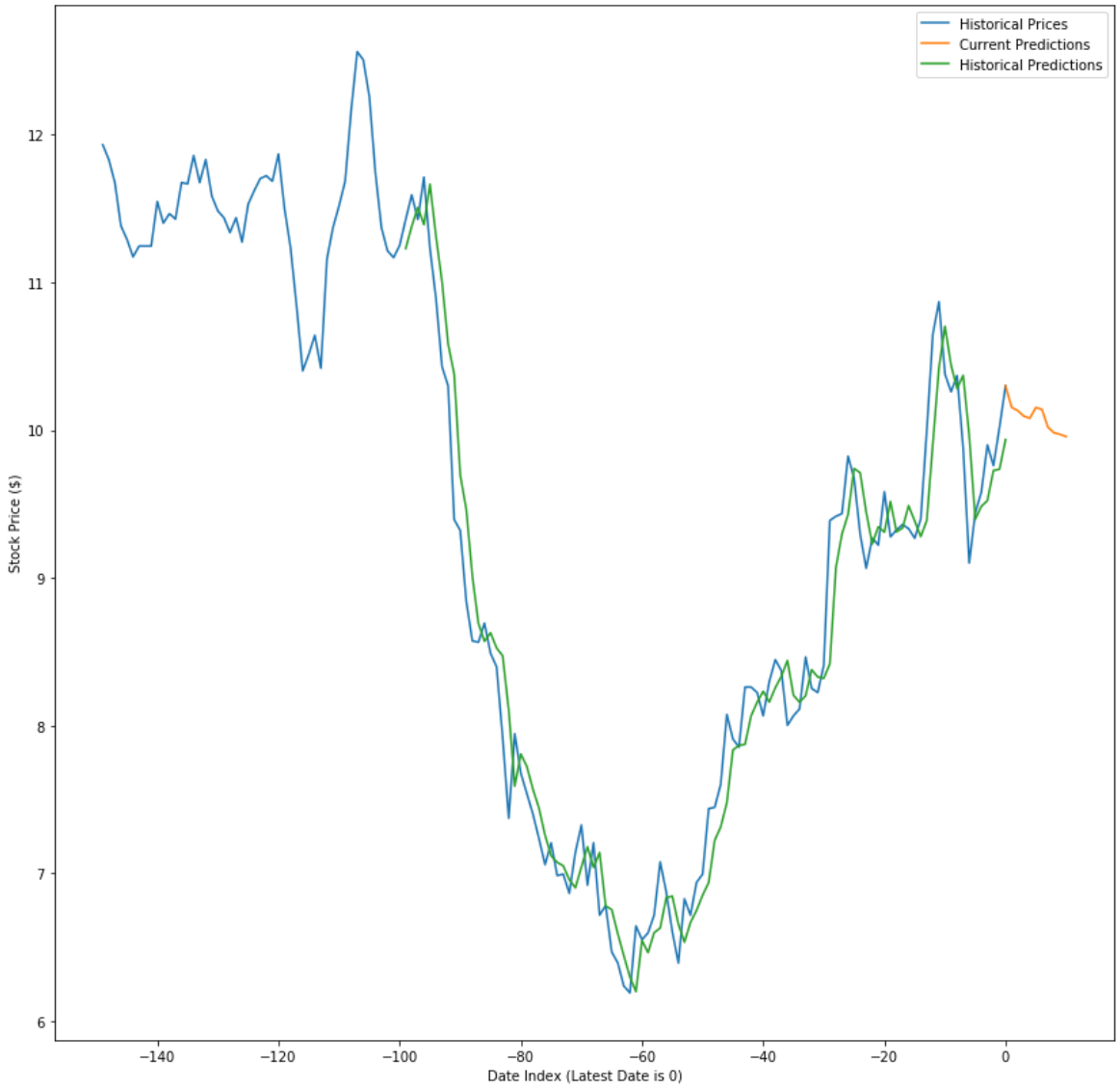


Figure 6.1a 1-day interval historical predictions (GE, Dense Neural Network)

From Figure 6.1a, it shows that the 1-day interval historical predictions line follows closely with the historical prices. The graph looks like the prediction line is just 1 day shifting from the historical prices, similar to a shifted and smoothed out historical prices line. Therefore, the shape of the historical predictions line is similar to the shape of the exponential moving averages (EMA), where the price changes from t to t+1 heavily depends on the direction and

magnitude of changes from t-1 to t, followed by decreasing importance from earlier historical prices. Other models in predicting stock prices of other stocks also show similar results.

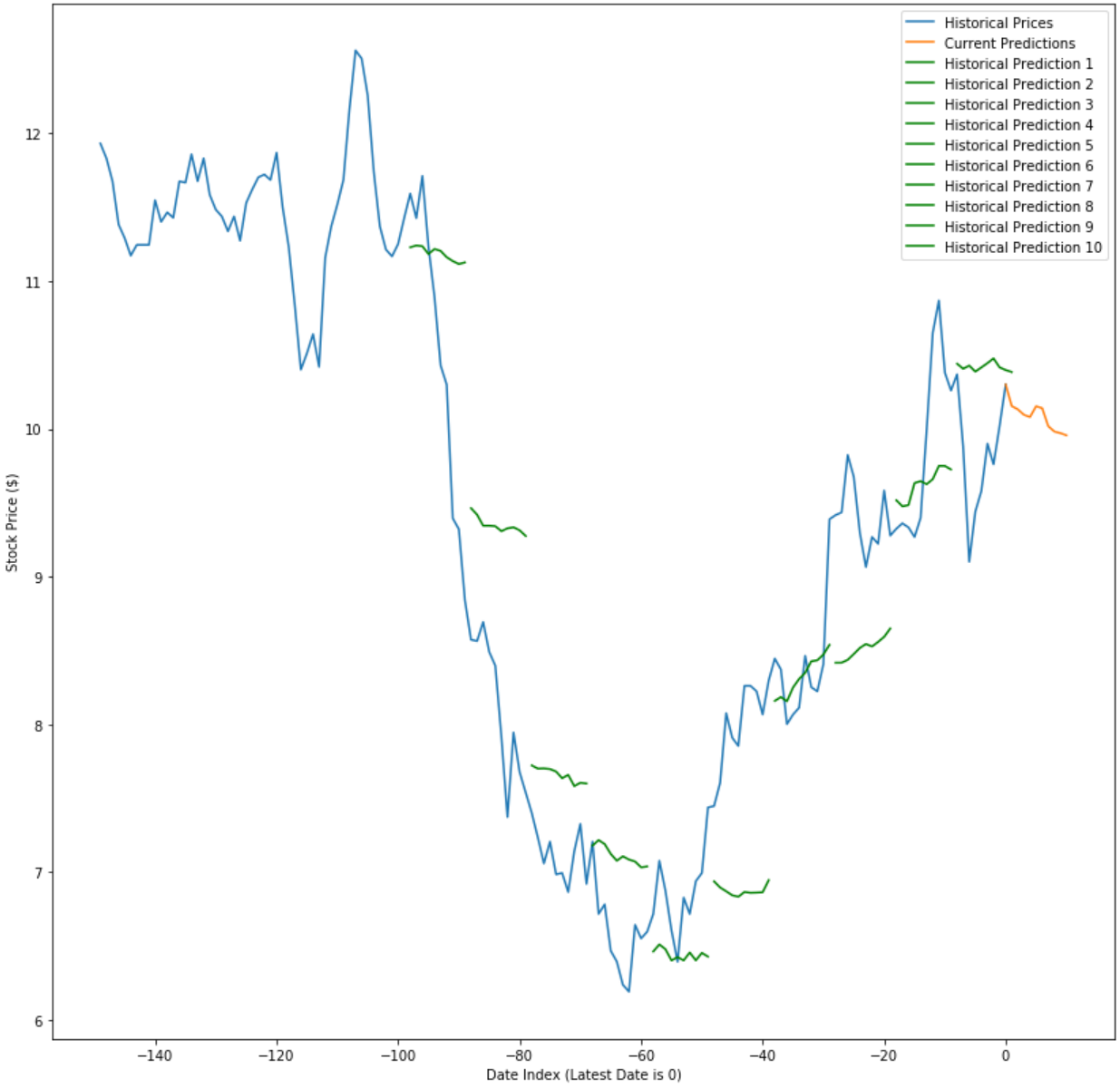


Figure 6.1b 10-day interval historical predictions (GE, Dense Neural Network)

From Figure 6.1b, it shows that the 10-day interval historical predictions line do not follow closely with the historical prices but could demonstrate the trend. For example, historical predictions 1, 2, 3, 4, 7, 8, 9, 10 provided insights on the correct market direction, yet the magnitude did not match the actual price movements. A possible reason for this error can be the 10-day interval prediction has to predict more values while having fewer data compared to the case of 1-day interval prediction, which for 1-day interval prediction, data of close prices until previous day are available. Therefore, a longer period of interval prediction could subject to greater changes in market fundamentals, including market news, macroeconomic factors, earning reports, etc. Other models in predicting stock prices of other stocks also show similar results.

Although price reflects all available information, the magnitude of price changes in the future might need other data for forecasting purpose, such as market sentiment, company announcement, retail and institutional investors' attention on the company, etc. This is one of the possible explanation of why the 10-day interval prediction might have a large difference to actual values as there are potential shifts in market momentum. Therefore, the price might be too compact and other information is required to make a more accurate prediction.

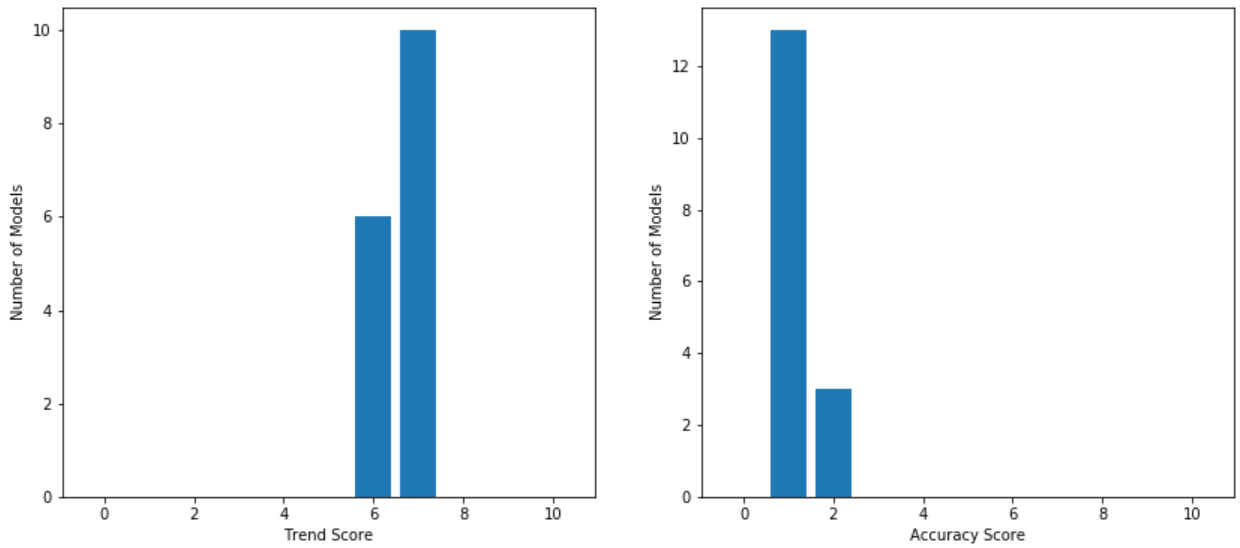


Figure 6.1c Trend score and accuracy score distribution (16 best models from evolution)

2 scores are used to measure the performance of historical predictions, trend score and accuracy score, introduced in 2.2.7. The higher the trend score means that the model is more accurate in trend prediction and could provide more meaningful price movement direction insights. The score representations used in this application could be useful for the user to interpret the errors of predictions in a quantifiable way. The higher the accuracy score means that the model could follow the actual stock prices more accurately. From Figure 6.1c, it shows that all best models generated from the evolution algorithm experiment have a trend score ranging from 6-7 but have an accuracy score ranging from 1-2 on the test set. This finding matches the earlier findings, that the trend could be predictable, especially for less volatile stocks, but exact price, especially further into the future, could hardly be predicted accurately.

Despite common research findings that recurrent neural networks in general perform better than dense feedforward neural networks at predicting time-series data such as stock prices, in this project feedforward neural network outperforms recurrent neural networks. One possible explanation is that training a recurrent neural network requires more data than the dense neural network in general, as recurrent neural networks have more parameters. As the models are trained using only daily stock prices dating back 20 years (or less if the stock is listed fewer than 20 years), there might not be enough data for training the recurrent networks to a good performance.

6.2 Prediction Approach Findings

As mentioned in 2.2.1, 2 approaches are tested in predicting the stock prices for the next 10 days, predicting all 10-day stock prices directly and predicting each stock price one at a time. The 2 different approaches frame the problem totally differently, which introduces a significant language bias.

According to the results (e.g. Figure 6.2a and 6.2b), for most stocks, most models that predict 10-day stock prices directly have a higher error than predicting individual stock price. However, the errors in predicting different days in the future are relatively constant for models that predict 10-day stock prices directly, while the error increases with the time from now for models that predict stock prices one day at a time.

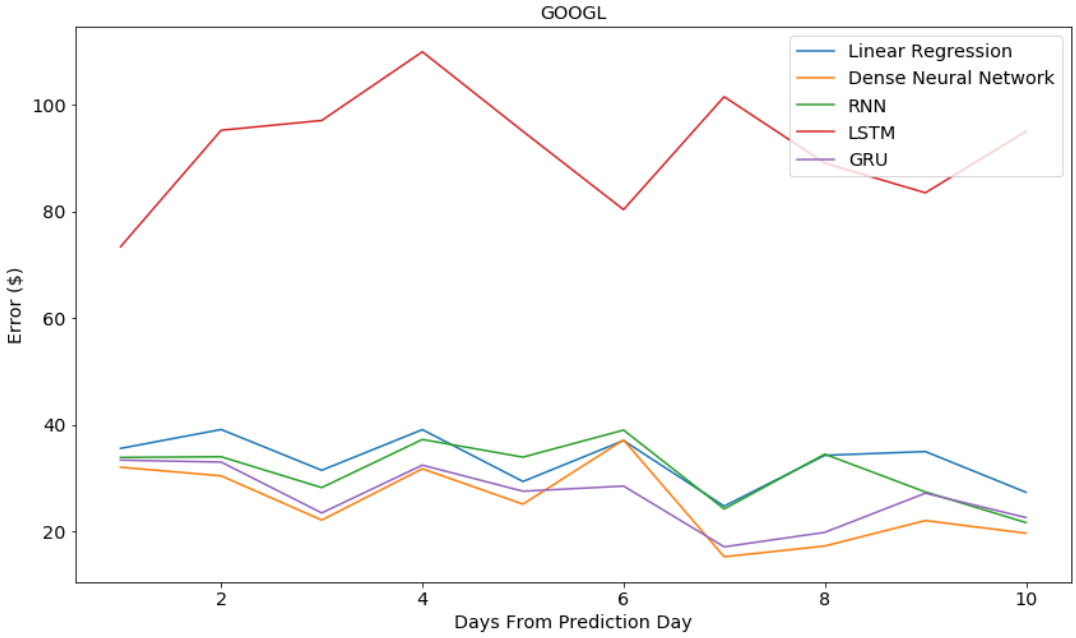


Figure 6.2a Prediction error in predicting stock price at different future dates (GOOGL, 10-day predict)

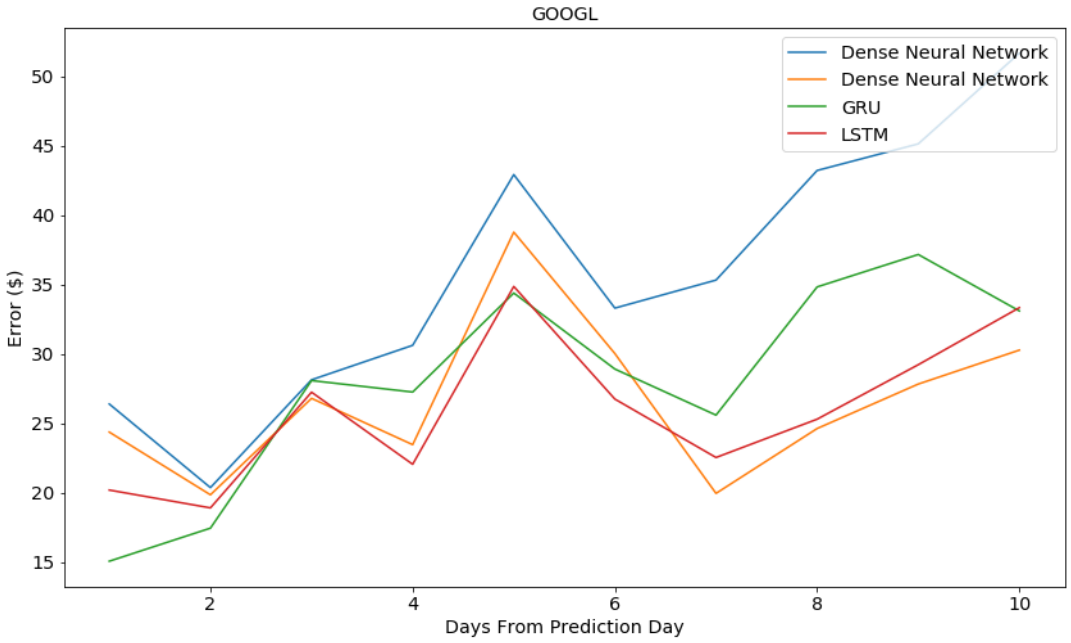


Figure 6.2b Prediction error in predicting stock price at different future dates (GOOGL, 1-day predict)

One possible explanation for such observation is that the 2 problem framing approaches drive the model to learn different abstractions. For models that predict 10-day stock prices directly, it will learn the abstraction over 10 days. It is assumed that the correlation between the predicted stock price and today's and earlier stock prices decreases when predicting further future. Since the learned abstractions need to be applicable throughout 10 days, the high error from further prediction because of low correlation is propagated to other closer predictions. It results in a constantly higher error for predicting all days.

On the other hand, predicting stock price one at a time allows the model to learn the relationships between more correlated data points. It can be observed from the results that the

first-day prediction is more accurate compared to the first prediction from models that predict 10 days directly.

However, not all 10-day predictions have a lower error, the error for further predictions are higher. As mentioned in 6.1, most models, especially those predicting stock price individually, behave like an EMA, which put more emphasis on more recent historical prices. Although this allows the model to accurately trace recent price movements, when predicting future stock prices iteratively the next predicted stock price is most strongly influenced by the previous prediction instead of real data. This results in reinforcement effect where predictions further ahead reinforce the unverified trend that the model predicts, and the errors amplifies and propagates to subsequent predictions.

6.3 Accuracy Findings

6.3.1 Definitions

$$i\text{-day Holding Period Return } (HPR_i) = \frac{P_i}{P_0} - 1$$

6.3.2 Baseline Investor

The baseline for model accuracy comparison is from a hypothesized investor who adopts a trading strategy of predicting the stock price will either go up or down by the holding period return calculated from historical data. There will also be a corresponding error for this strategy. Assume the hypothesized investor always correctly predicts the stock price movement direction. The baseline strategy error is defined as:

$$\text{Baseline Strategy Error} = min\{P_0 \cdot \left(1 + \overline{|HPR_i|}\right) - P_1,\ P_0 \cdot \left(1 - \overline{|HPR_i|}\right) - P_1)\}$$

If the hypothesized investor always predicts the wrong stock price movement, then the error is the maximum of the 2 terms.

6.3.3 Findings

The errors from 6.2 are compared with the baseline strategy introduced (Figure 6.3a and 6.3b). It is found that most models for most stocks achieve comparable performance in terms of error as the baseline strategy. Some models for some stocks have a slightly lower error than the baseline, and some have higher.

Since the baseline error is calculated based on the assumption that the hypothesized investor always makes a correct prediction on the price movement direction, despite only having marginal improvement on the accuracy or even lower in some cases, the machine learning models trained successfully predicted the trend of the price movement, which agrees with the findings in 6.1. See Appendix F for similar findings over other stocks.

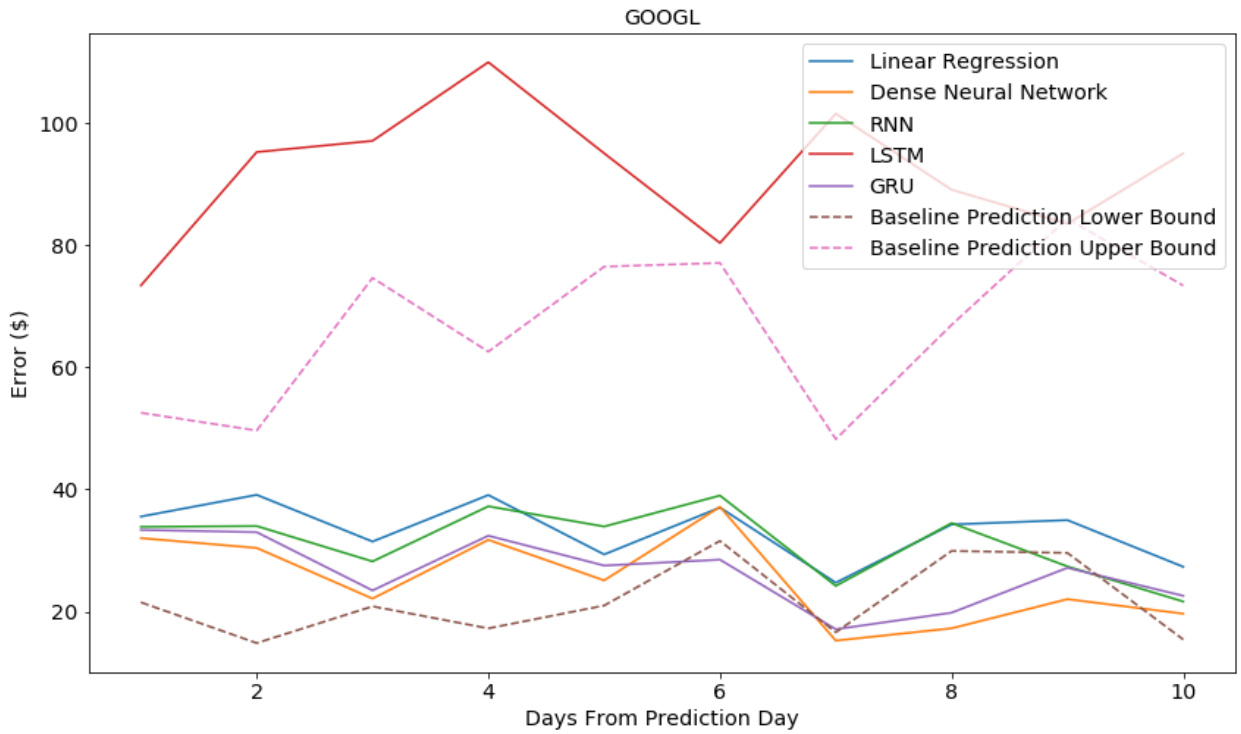


Figure 6.3a Comparison between different models (10-day predict) with baseline

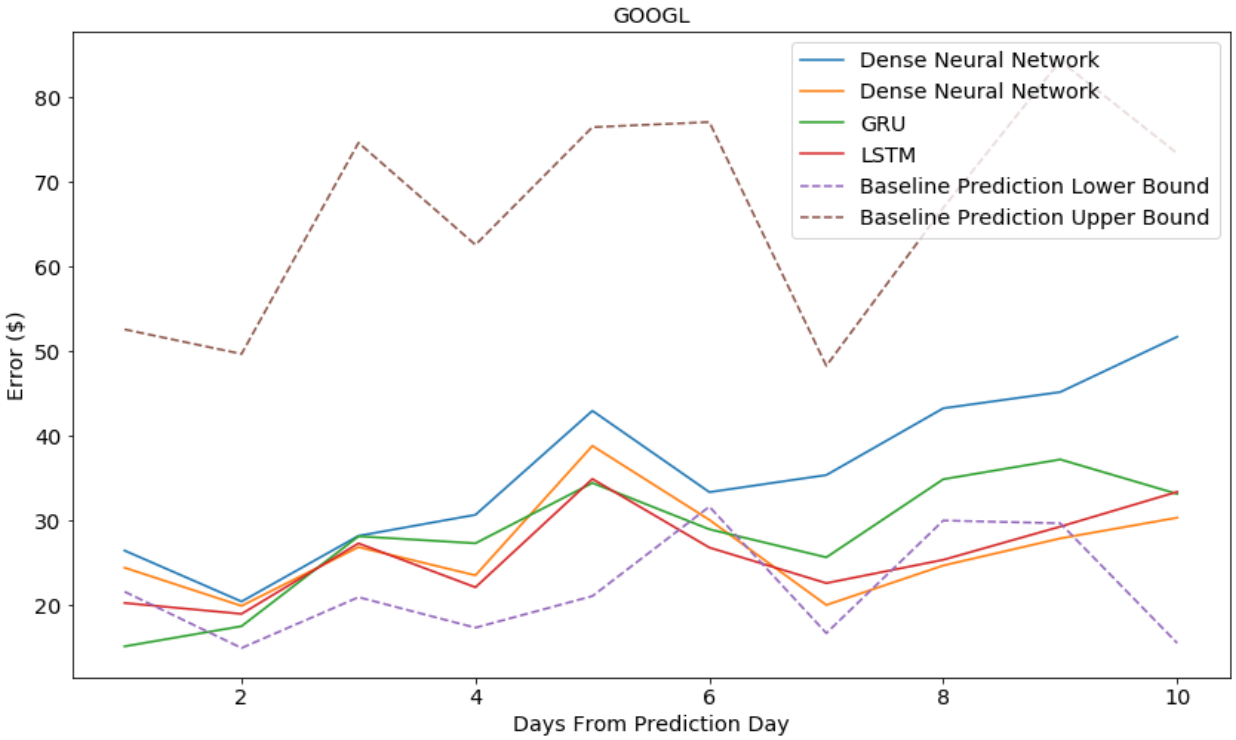


Figure 6.3b Comparison between different models (1-day predict) and baseline

6.4 Model Architecture and Hyperparameters Search with Evolution Algorithm Findings

The evolution algorithm experiment conducted has shown promising results in searching model architectures and hyperparameters. Multiple experiments are run, for different stocks, different neural network types and different inputs. From the prediction error recorded over each evolution iteration (e.g. Figure 6.4a and 6.4b), all experiments have shown that the evolution algorithm successfully finds better models over time. Hand-designed models based on the team's intuition and basic knowledge could not achieve an error rate lower than that achieved by the algorithm's explored models.

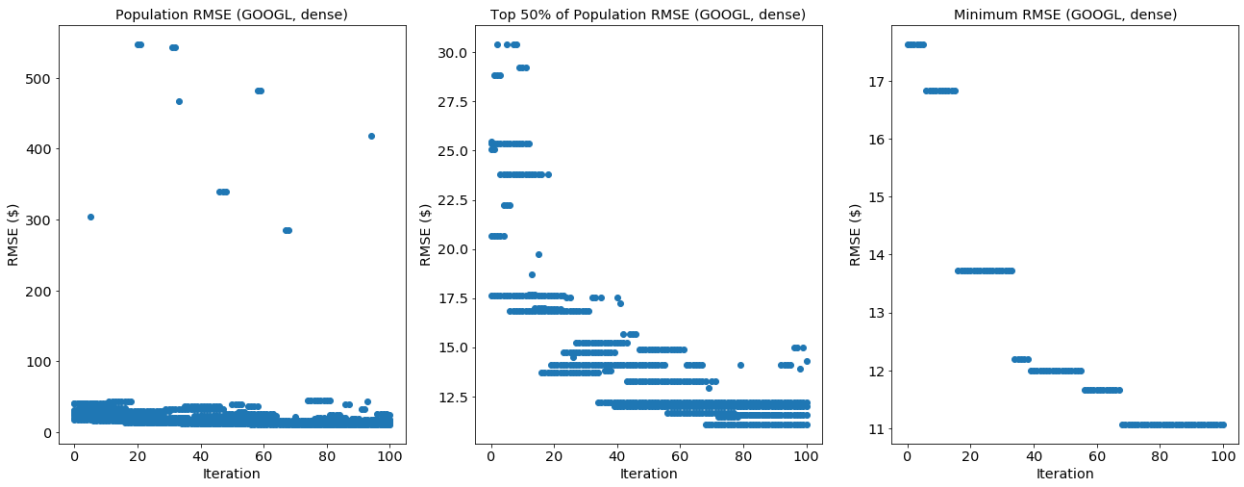


Figure 6.4a Evolution errors (GOOGL, dense neural network)

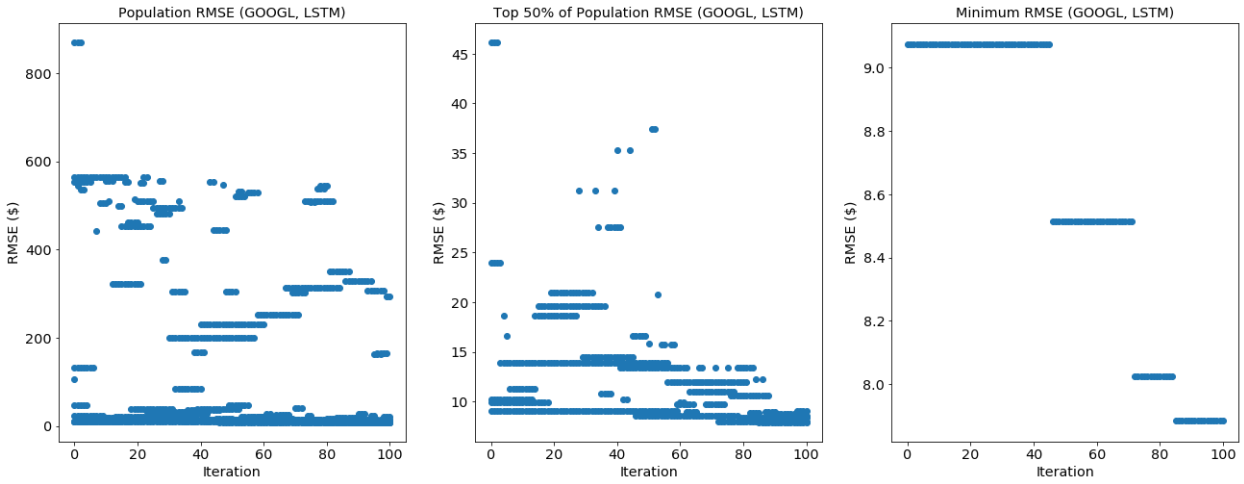


Figure 6.4b Evolution error (GOOGL, LSTM network)

One interesting and unexpected observation from the evolution algorithm results is that a number of best models found are fairly simple. The found models are 1 or 2 layers deep with a linear activation function. In the case of having stacks of linear layers, the model is mathematically equivalent to a linear regression over features. There are multiple possible explanations for this observation.

First, the evolution algorithm hyperparameters used limits the search space for possible model architectures and hyperparameters. Due to computational power constraints, only a small population with 10 models is used. This limits the variety of models explored by the algorithm as the variance within the population is small. Moreover, under constraints, each experiment is run for 100 iterations only, which also limits the exploration. On average, the whole population is only 10 steps or mutations away from the original random population, which may not be significant enough for deep exploration.

The evolution algorithm itself, together with its hyperparameters, introduces a search bias, the algorithm defines the possible explored models and the search path to achieve them.

Another possible explanation is that the dataset size is relatively small with just daily stock prices. Larger deep neural networks with more complicated architectures and thousands or even millions of weights require much more data to train and learn from.

A final possible explanation is that the stock market is at least weakly efficient, i.e., stock prices follow random walk given historical price data, and patterns with predictive power could not be found just from raw price data. If stock prices follow random walk, a good predicting method is to put strong weights at very recent prices, and hope the actual price will fluctuate closely around it, which is very similar to a linear model.

Only using stock price data and simple derivatives like moving averages introduces a language bias, as price movements are also highly dependent on news and sentiment. Although stock prices reflect information, it is a very compact representation of all information and news. Thus, it is difficult to reverse engineer features or information out from a single number, especially when only daily stock prices are available. Having other information like real-time news sentiment or summary may help to break stock prices down to more granular components for machine learning algorithms to learn from.

6.5 Other Findings

6.5.1 Trend lines

The accuracies of the predictions based on linear trendlines fluctuate very wildly, as they are simply linear interpolations, while real stock prices may fluctuate up and down. In particular, the accuracy of the predictions depends completely on the choice of the day based on which the linear interpolations are made. Since the choice is arbitrary, the predictions based on trend lines are not reliable at all.

6.5.2 Alternative Prediction Method - Skip Predict

To tackle the problem of input bias on the intermediate prediction result and the short term noise of the stock, an alternative prediction method “skip predict” is used.

This method has 2 key advantages. First, this method can decouple the dependency of the prediction result based on the previous day in the original model. With “skip predict”, the input data of n-day before is used for the prediction. For example, if the number of days skipped is 10, this represents that input data would not consider the recent 10 days, and the input data would use the shifted time frame.

Second, the prediction result can all depend on the historical prices and not the intermediate prediction result. This could be one of the methods to solve the reinforcement problem mentioned in 6.2. The error of the first predicted data point would not impact the following

predictions result. For example, the incorrect trend of the first predicted data point (t+1) would not serve as the input for the predictions later (t+2, t+3, ..., t+10). This creates an advantage that the error or bias would not accumulate and the result could solely depend on historical data. The hypothesis is that such method could generate a lower root mean square error compared to the original model the application is using.

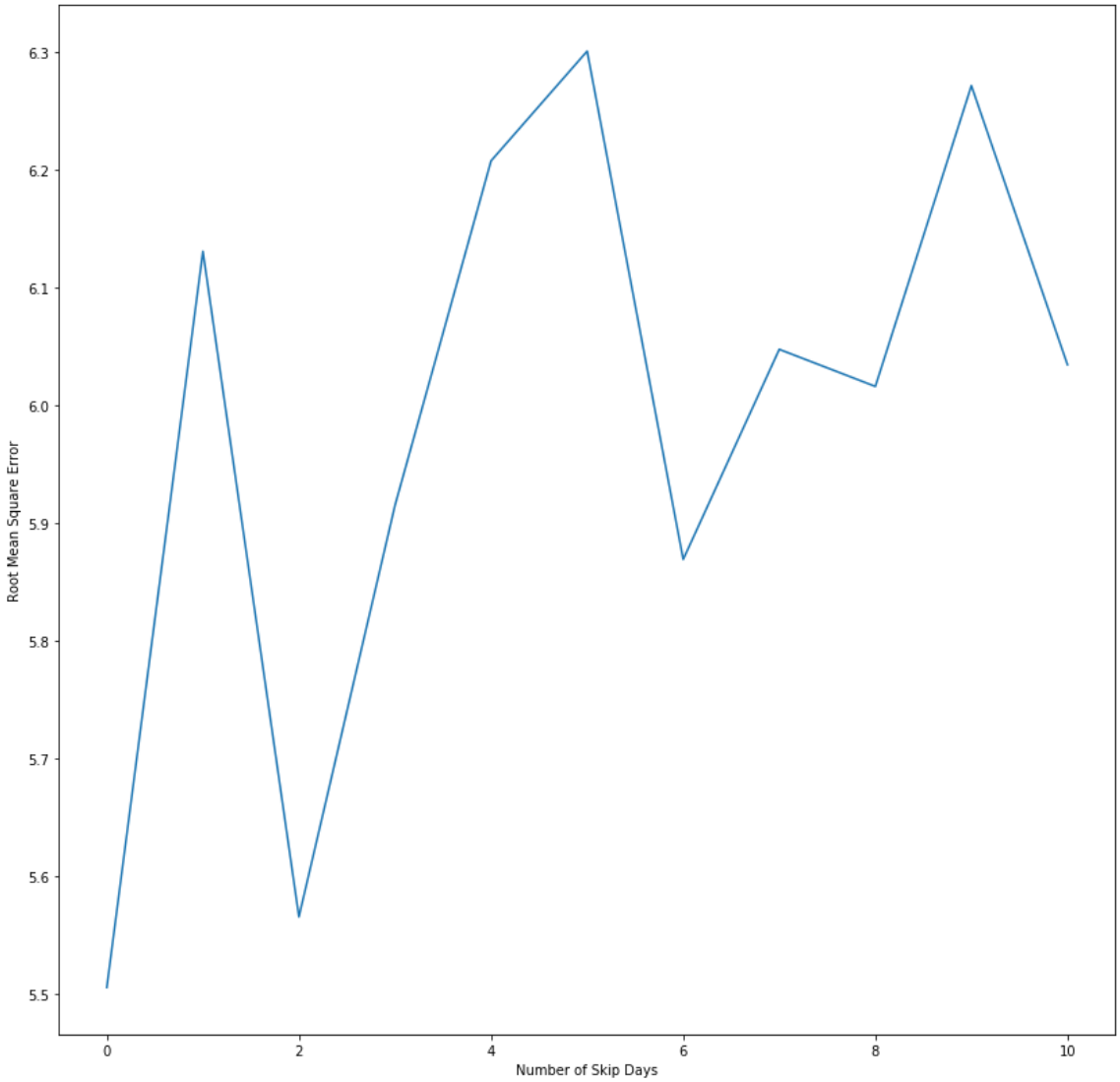


Figure 6.5 Skip predict RMSE (CAT, Dense Neural Network, 1-day predict)

From Figure 6.5, the increasing number of skip days can not effectively lower the root mean squared error. Similar observations are also seen in other stocks, including TSLA, PG, WMT, etc. This is an interesting finding that skipping more days could not improve the accuracy. One possible explanation is that the correlation between further apart stock prices is very small, so although the reinforcement effect could be solved by this prediction approach, the error is amplified by another low correlation factor, which results in a model that fails to capture instant and close-by news.

6.6 Mobile Application User Experience Testing

To evaluate the user experience of the mobile application, users who have not been involved in the development of the application have been invited to try out the mobile application and give constructive feedbacks. The major findings are summarized as follows:

6.6.1 Useful Insights for Finding General Trend

Despite the flaws found in the mobile application, users in the test agree that they were able to check out what are the possible movements of the stock prices predicted by the machine learning models and the general directions of the stocks are going. Users find it might be useful for finding stocks with upside potential for the coming few days.

6.6.2 Unclear Description of the Models

In the mobile application, different models are named after their architectures, such as LSTM, Dense Neural Network, and GRU. However, these technical names are not familiar to users who have no experience in machine learning and cause some confusions among users.

6.6.3 Unclear Presentations of the Prediction Results

Each stock is associated with a model trend score as described in 2.2.7. However, to laymen users, it might not always be clear what these scores represent, as the definitions are not clearly explained. The lack of clarity might confuse users or lower their confidence as they attempt to take actions following the predictions made by the models.

7. Discussion

As mentioned in 5, the success of the project is primarily determined by two factors, namely, whether the investment tool provides useful, accurate stock price predictions to investors, and whether investors can use and understand the predictive information provided by the machine learning technologies. The project's objectives are therefore partially fulfilled.

7.1 Accuracy of Stock Price Predictions

As shown in 6.1, while the 1-day stock price prediction follow closely with actual stock prices, the predictions for stock prices after 10 days deviate considerably from the actual stock prices. This shows that machine learning models fail to provide accurate stock price predictions to retail investors.

Nevertheless, some of the models have been shown to outperform predictions based on random walks as mentioned in 6.2, and therefore might still serve as a reference for more savvy investors, who might be able to compare the results with their own analysis findings to discover meaningful trends.

7.2 Democratization of Machine Learning Technology

Another factor when evaluating the project's success is whether investors can use and understand the predictive information provided by the machine learning technologies using our mobile application. In spite of the confusions found in some parts of the user interface, especially in the advanced user mode, users found useful insights provided by the machine learning models, such as identifying stocks with upside potential. The result is significant, in the sense that users with little background on machine learning technology and stock trading could find potential use cases for the application. The results imply that machine learning technologies could be democratized to serve the interest of the general public. Stock price prediction is a particularly exciting area, because the level of expertise required to succeed in making profitable short-term investments is considered to be prohibitive for small, retail investors, and trading with help of machine learning is a feat only institutional investors could perform. The application demonstrates one possible way retail investors could use machine learning technologies on their own.

8. Conclusion

The project lays the foundation for democratizing machine learning technologies for retail investors, connecting predictions made by machine learning models to retail investors through a mobile application. It helps investors navigate through the stock markets with additional analysis and help them make more informed decisions.

The findings demonstrated that the application provides significance in trend prediction. When compared to the baseline, the prediction shows useful trend tendency with the real stock trend. Through the application interface, the user can easily compare the predictions and model scores from different machine learning models, then choosing the one that fits their preference. The models used in the application will continue to improve itself by searching for a better model topology, structure and hyperparameters through evolution algorithm. The findings concluded the usefulness of evolution algorithm in lowering the mean squared error when predicting stock prices, which is helpful for improving the trend prediction for retail investors.

Therefore, with the application and research findings, to large extent the project team achieved the aim of creating an user-friendly system for retail investors whom does not have previous technical knowledge to navigate the machine model predictions result with useful benchmarks.

There are 4 possible further improvements building upon the findings of this project. First, multiple approaches to framing the problems could be explored in the future, such as

predicting whether the stock price goes up or down (binary classification) based on the previous stock prices. Other features could be incorporated, such as market news and sentiment. Combined with the development of more advanced machine learning techniques, the accuracy of the information provided to retail investors might be improved significantly.

Second, a larger scale of evolution with larger population size and more iterations could also be tested for achieving better results. Model inputs can also be included into the evolution algorithm as a variable to optimize. Regularized evolution [38] can be tested to eliminate old models regardless of their accuracy, which could allow the algorithm to search for more distant models in the search space.

Third, it is also possible to use more finance-specific scores, like those introduced, as the objective function instead of simple mean squared errors to achieve better results.

Fourth, mobile applications with better presentation of stock price predictions could be developed to help investors understand the implications of the stock price predictions, e.g. when to buy or sell. This would allow investors to make more informed decisions based on the machine learning models and truly democratize machine learning technologies, which were believed to be only in the hands of very few people.

9. References

[1] "Survey Finds Hong Kong Securities Market Attracts Wide Range of Investors," *HKEX*, 13 Jul 2017*;* http://www.hkex.com.hk/news/news-release/2017/170713news?sc_lang=en.

[2] Y. Dai and Y. Zhang, "Machine Learning in Stock Price Trend Forecasting," *Stanford University*; http://cs229.stanford.edu/proj2013/DaiZhang-MachineLearningInStockPriceTrendForecasting.pdf.

[3] J. Patel, S. Shah, P. Thakkar, and K. Kotecha, "Predicting stock and stock price index movement using Trend Deterministic Data Preparation and machine learning techniques," *Expert Systems with Applications: An International Journal,* Vol. 42, Jan. 2015, pp. 259-268

[4] B. Wanjawa and L. Muchemi, "ANN Model to Predict Stock Prices at Stock Exchange Markets," arXiv:1502.06434 [q-fin.ST], 2014

[5] D. Mandic and J. Chambers, *Recurrent Neural Networks for Prediction*, Wiley, 2001

[6] R. Williams and D. Zipser, "Gradient-based learning algorithms for recurrent networks and their computational complexity", in *Back-propagation: Theory, Architectures and Appli-Cations*, Hillsdale, NJ: Erlbaum, 1992, pp. 433 - 486

[7] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies", in *A Field Guide to Dynamical Recurrent Neural Networks*, S. C. Kremer and J. F. Kolen, eds., IEEE press, 2001

[8] S. Hochreiter and J. Schmidhuber, "Long Short-term Memory", *Neural Computation*, vol. 9, no. 8, pp. 1735 - 1780, 1997

[9] K. Cho et al., "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation", arXiv:1406.1078 [cs.CL], 2014

[10] W. Gail, G. Yoav, and Y. Eran, "On the Practical Computational Power of Finite Precision RNNs for Language Recognition", arXiv:1805.04908 [cs.NE], 2018

[11] J. Chung, C. Gulcehre, K. Cho and Y. Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". arXiv:1412.3555 [cs.NE]. 2014

[12] E. Real, et al., “Large-Scale Evolution of Image Classifiers,” arXiv:1703.01041 [cs.NE]. Jun 2017.

[13] D. Alajbeg, Z. Bubas and D. Vasic, “Price Distance To Moving Averages And Subsequent Returns”, *International Journal of Economics, Commerce and Management*, Vol. V, Dec 2017, pp. 33 - 47

[14] Progressive Web Apps, Google. Available: https://developers.google.com/web/progressive-web-apps/

[15] Neoteric, “Single-page application vs. multiple-page application”, Medium. 2016. Available: https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58

[16] Keras: The Python Deep Learning library, Keras. Available: https://keras.io/.

[17] Documentation of scikit-learn 0.19.2. Available: http://scikit-learn.org/stable/documentation.html.

[18] Alpha Vantage API Documentation, Alpha Vantage. Available: https://www.alphavantage.co/documentation/.

[19] NumPy v1.14 Manual, The SciPy community. Available: https://docs.scipy.org/doc/numpy-1.14.5/.

[20] pandas: powerful Python data analysis toolkit, Pandas. Available: http://pandas.pydata.org/pandas-docs/version/0.23/.

[21] Hello, Colaboratory - Colaboratory - Google, Google. Available: https://colab.research.google.com.

[22] TensorBoard: Visualizing Learning, Google. Available: https://www.tensorflow.org/guide/summaries_and_tensorboard.

[23] Welcome to Flask — Flask 1.0.2 documentation, Pallets Team. Available: http://flask.pocoo.org/docs/1.0/.

[24] Cloud Storage, Google. Available: https://firebase.google.com/docs/storage/.

[25] Getting Started, The Investors Exchange. Available: https://iextrading.com/developer/docs/.

[26] Cloud Functions for Firebase, Google. Available: https://firebase.google.com/docs/functions/.

[27] Cloud Firestore, Google. Available: https://firebase.google.com/docs/firestore/.

[28] React – A JavaScript library for building user interfaces, Facebook Inc; https://reactjs.org/.

[29] React Router: Declarative Routing for React.js, React Training. Available: https://reacttraining.com/react-router/web/guides/philosophy.

[30] Read Me - Redux, Redux. Available: https://redux.js.org/.

[31] Immutable collections for JavaScript, Facebook Inc. Available: https://github.com/facebook/immutable-js/.

[32] Introduction - Material Design, Google. Available: https://material.io/design/introduction/.

[33] Material UI, Material UI Team. Available: https://material-ui.com/

[34] Using Google Charts, Google. Available:

https://developers.google.com/chart/interactive/docs/.

[35] Facebook Login, Facebook. Available:

https://developers.facebook.com/docs/facebook-login/

[36] Firebase Authentication, Google. Available: https://firebase.google.com/docs/auth/

[37] unittest — Unit testing framework, Python Software Foundation. Available:

https://docs.python.org/3.6/library/unittest.html

[38] E. Real, A. Aggarwal, Y. Huang, Q. Le, "Regularized Evolution for Image Classifier Architecture Search," arXiv:1802.01548 [cs.NE]. Feb 2018.

[39] GitHub features: the right tools for the job, Github Inc; https://github.com/features.

10. Appendices

A - Model Options Example

Dense neural network model options example:

```
"modelOptions": {
   "network_type": "dense",
   "net": {
      "layers": [
         {"units": 32, "activation": "relu", "is_input": true, "inputUnits": 10},
         {"units": 64, "activation": "relu"},
         {"is_output": true, "activation": null}
      ],
      "loss": "mse",
      "optimizer": "adam",
      "learning_rate": 0.001,
      "epochs": 20,
      "batch_size": 32,
      "metrics": ["accuracy"],
      "evaluation_criteria": {
         "minimize": true,
         "threshold": 10
      }
   },
   "predict_n": 10
}
```

Multi-layer LSTM network model options example:

```
"modelOptions": {
   "network_type": "LSTM",
   "net": {
      "layers": [
         {
            "layer_type": "LSTM",
            "units": 32,
```

```
            "activation": "relu",
            "recurrent_activation": "sigmoid",
            "stateful": false,
            "is_input": true,
            "inputUnits": [10, 1],
            "return_sequences": true
          },
          {
            "layer_type": "LSTM",
            "units": 32,
            "activation": "relu",
            "recurrent_activation": "sigmoid",
            "stateful": false,
            "is_input": true,
            "inputUnits": [10, 1],
            "return_sequences": false
          },
          {"units": 64, "activation": "relu"},
          {"is_output": true, "activation": null}
        ],
        "loss": "mse",
        "optimizer": "adam",
        "learning_rate": 0.001,
        "epochs": 20,
        "batch_size": 32,
        "metrics": ["accuracy"],
        "evaluation_criteria": {
          "minimize": true,
          "threshold": 10
        }
    },
    "predict_n": 10
}
```

B - Input Options Example


```
"inputOptions": {
   "config": [
      {"type": "lookback", "n": 10, "stock_code": "GOOGL", "column": "adjusted_close"},
      {"type": "moving_avg", "n": 10, "stock_code": "GOOGL", "column": "adjusted_close"}
   ],
   "stock_codes": ["GOOGL"],
   "stock_code": "GOOGL",
   "column": "adjusted_close"
}
```

C - Trained Model Saving Format

The following is the directory and saving format for trained models.

```
/saved_models
  /dnn
    /<model_hash_1>
      /<stock_code_1>
      /<stock_code_2>
    /<model_hash_2>
      /<stock_code_1>
      /<stock_code_2>
    models_data.json
  /linear_regression
    /<model_hash_1>
      /<stock_code_1>
      /<stock_code_2>
    models_data.json
```

The following is the data structure of models_data.json, which saves the saved path, model options and input options of all trained models of the same type. Each model hash is a SHA256 hash calculated from a JSON string combining a model's model type, model options and input options, which prevents collisions from saving all trained models with systematic timestamp-based names.

```
{
  "models": [
    "<model_hash_1>": {
      "<stock_code_1>": [
        {
          "model_name": "<model_hash_1>_<timestamp>.h5",
          "model_path": "<relative_model_path>",
          "model": "dnn"
        }
      ],
      "<stock_code_2>": [
```

```
            {
              “model_name”: “<model_hash_2>_<timestamp>.h5”,
              “model_path”: “<relative_model_path>”,
              “model”: “dnn”
            }
          ]
        },
        “<model_hash_2>”: {
          “<stock_code_1>”: [
            {
              “model_name”: “<model_hash_1>_<timestamp>.h5”,
              “model_path”: “<relative_model_path”,
              “model”: “LSTM”
            }
          ],
        }
    ],
    “modelTypes”: {
        “<model_hash_1>”: {
          “model”: “dnn”,
          “modelOptions”: <model_options_dict>,
          “model”: <input_options_dict>,
        },
        “<model_hash_2>”: {
          “model”: “dnn”,
          “modelOptions”: <model_options_dict>,
          “model”: <input_options_dict>,
        }
    }
}
```

D - Evolution Algorithm Mutations

Different mutations are available at each step to slowly evolve the population.

Mutation	Description	Options
add_dense_layer	Add a dense layer with a random number of units and random activation function	
remove_dense_layer	Remove a dense layer	
change_units	Change the number of units in a randomly chosen layer	8, 16, 32, 64, 128
change_activation	Change the activation function in a randomly chosen layer	ReLU, sigmoid, tanh, linear
learning_rate	Change the learning rate	0.01, 0.001, 0.0001
batch_size	Change the batch size	16, 32, 64

All recurrent neural networks (simple RNN, LSTM network, GRU network) have other additional mutations.

RNN type	Mutation	Description	Options
Simple RNN	add_rnn_layer	Add an RNN layer with a random number of units and a random activation function to the RNN layer stack	
Simple RNN	remove_rnn_layer	Remove an RNN layer from the RNN layer stack	

LSTM	add_lstm_layer	Add an LSTM layer with a random number of units, random activation function and a random recurrent activation function to the LSTM layer stack	
LSTM	remove_lstm_layer	Remove an LSTM layer from the LSTM layer stack	
LSTM	change_recurrent_activation	Change the recurrent activation function in a randomly chosen LSTM layer	sigmoid, hard sigmoid
GRU	add_gru_layer	Add a GRU layer with a random number of units, random activation function and random recurrent activation function to the GRU layer stack	
GRU	remove_gru_layer	Remove a GRU layer from the GRU layer stack	
GRU	change_recurrent_activation	Change the recurrent activation function in a randomly chosen GRU layer	sigmoid, hard sigmoid

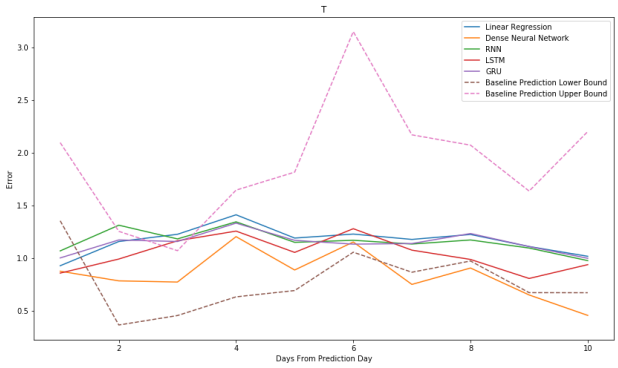
E - Evolution Algorithm Hyperparameters

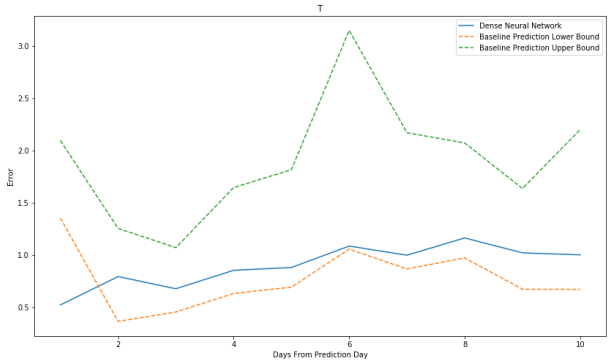
There are a number of hyperparameters that can be tuned for the evolution algorithm, which are set to a certain value for the experiment.

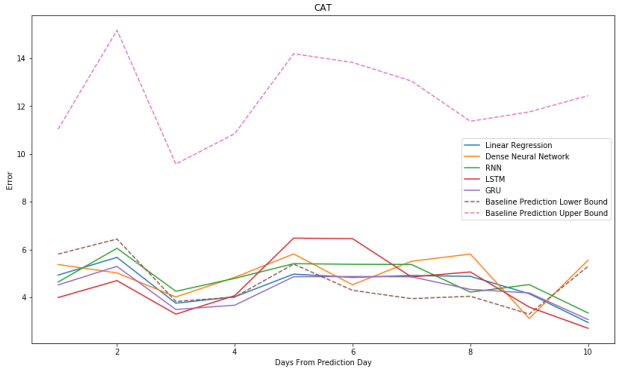
Hyperparameter	Value
POPULATION_SIZE	10
ITERATIONS	at least 100
Optimizer used in all training	Adam optimizer
Epochs	20

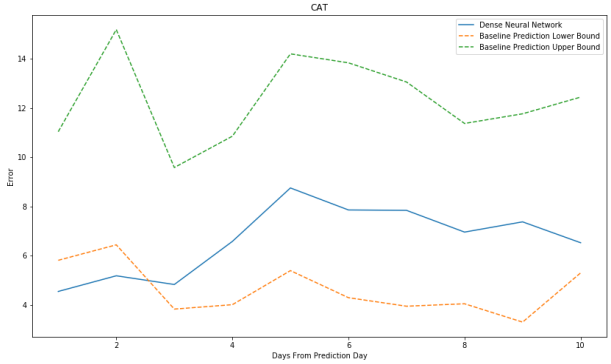
F - Prediction Results

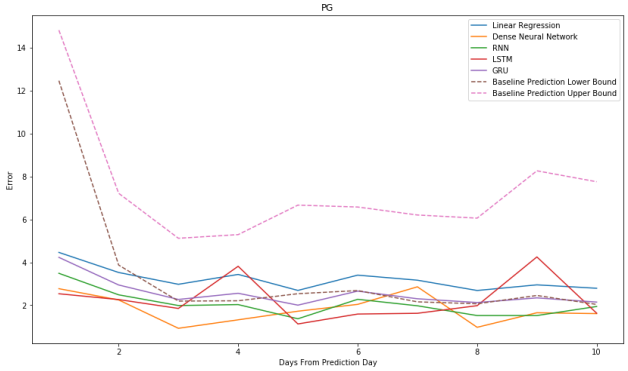
The following are the performance of different models on different stocks with the 2 different prediction approaches. Each row is the result of a certain stock. The left column is from models that predict 10-day stock prices directly, the right column is from models that individually predict 1-day stock price.

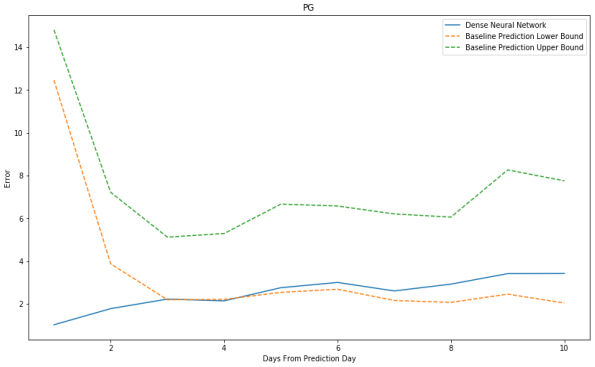


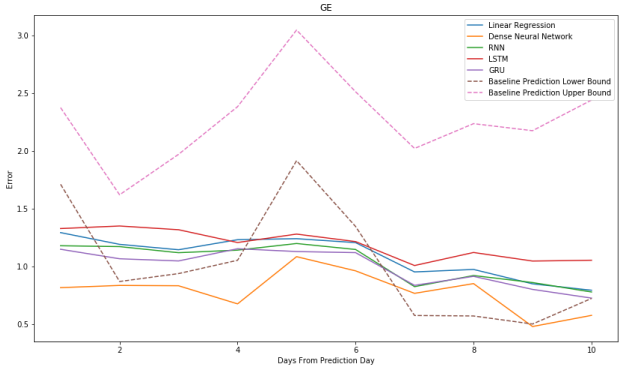


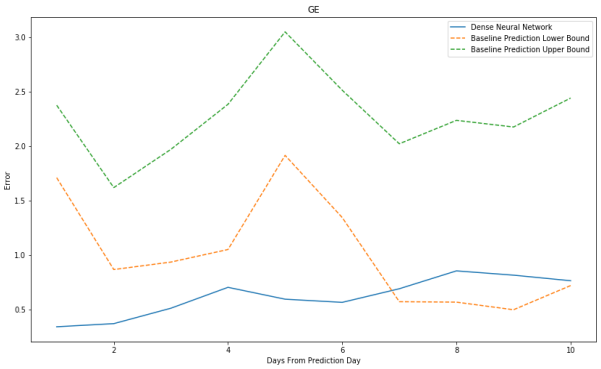


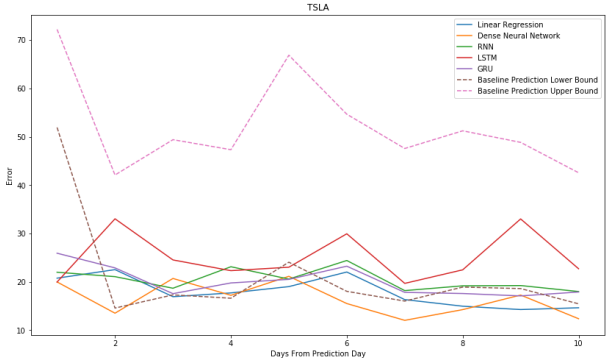


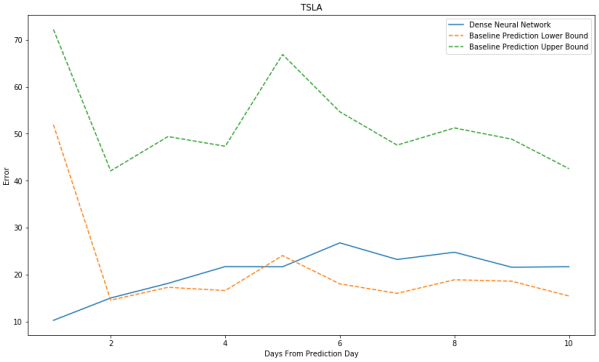


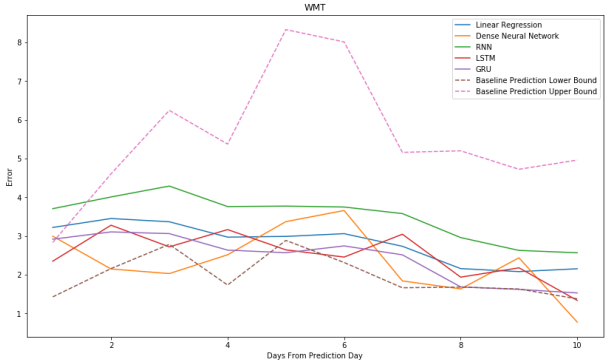


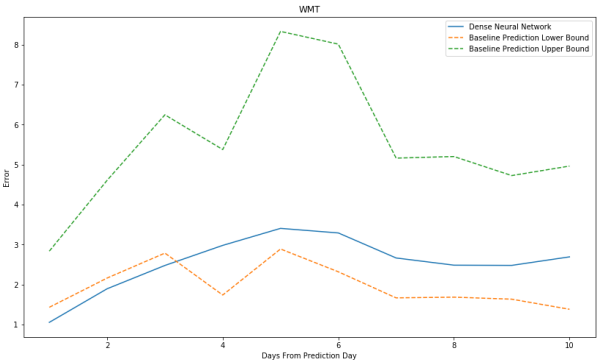

GE
Linear Regression
Dense Neural Network
RNN
LSTM
GRU
Baseline Prediction Lower Bound
Baseline Prediction Upper Bound
Error
Days From Prediction Day

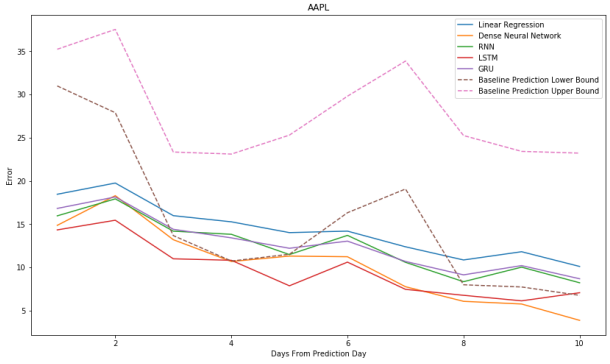

GE
Dense Neural Network
Baseline Prediction Lower Bound
Baseline Prediction Upper Bound
Error
Days From Prediction Day

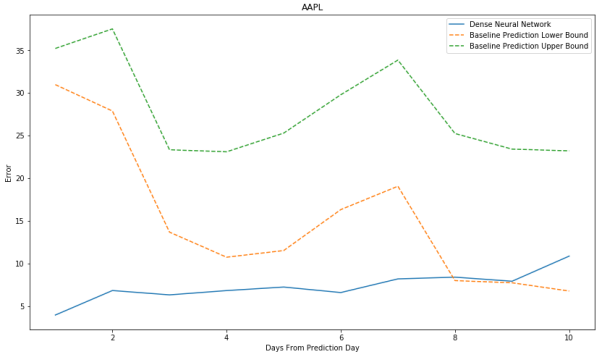

TSLA
Linear Regression
Dense Neural Network
RNN
LSTM
GRU
Baseline Prediction Lower Bound
Baseline Prediction Upper Bound
Error
Days From Prediction Day

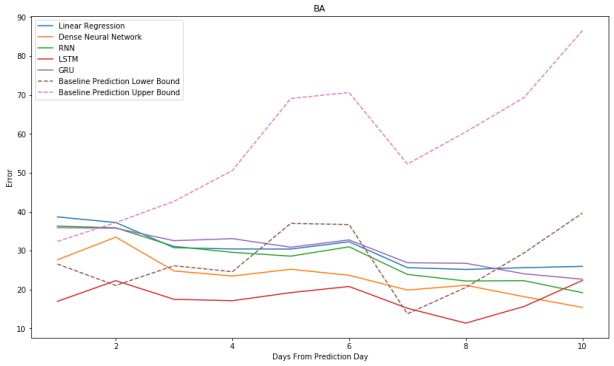

TSLA
Dense Neural Network
Baseline Prediction Lower Bound
Baseline Prediction Upper Bound
Error
Days From Prediction Day

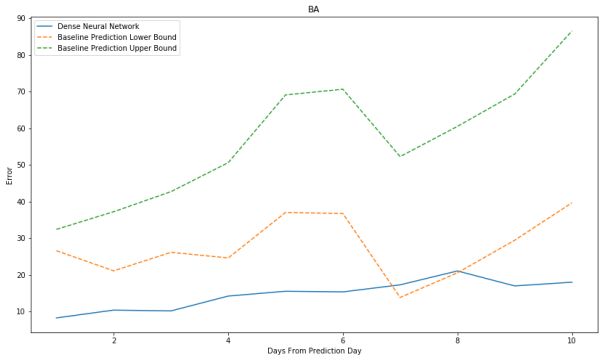

WMT
Linear Regression
Dense Neural Network
RNN
LSTM
GRU
Baseline Prediction Lower Bound
Baseline Prediction Upper Bound
Error
Days From Prediction Day

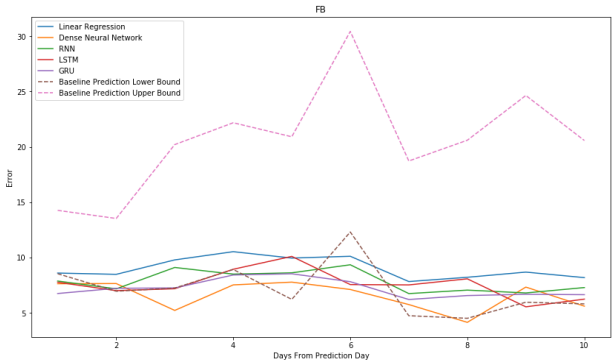

WMT
Dense Neural Network
Baseline Prediction Lower Bound
Baseline Prediction Upper Bound
Error
Days From Prediction Day

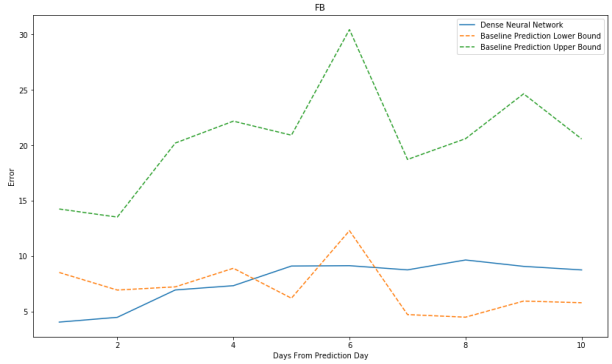

AAPL
Linear Regression
Dense Neural Network
RNN
LSTM
GRU
Baseline Prediction Lower Bound
Baseline Prediction Upper Bound
Error
Days From Prediction Day

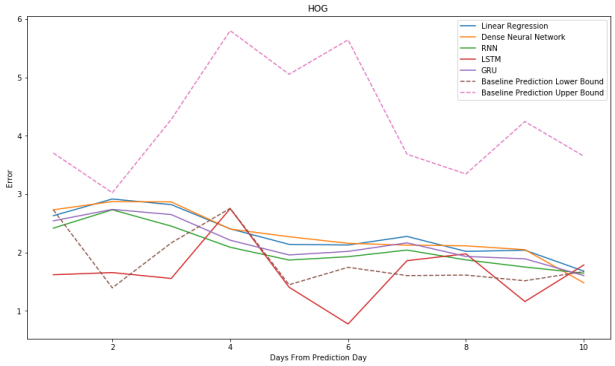

AAPL
Dense Neural Network
Baseline Prediction Lower Bound
Baseline Prediction Upper Bound
Error
Days From Prediction Day

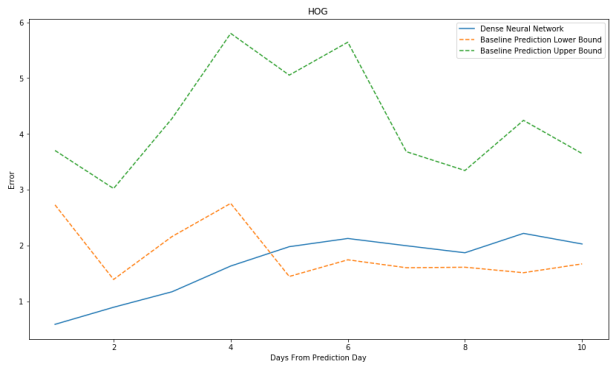

BA
Linear Regression
Dense Neural Network
RNN
LSTM
GRU
Baseline Prediction Lower Bound
Baseline Prediction Upper Bound
Error
Days From Prediction Day

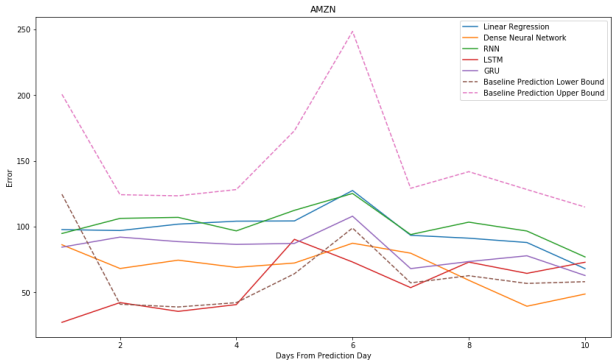

BA
Dense Neural Network
Baseline Prediction Lower Bound
Baseline Prediction Upper Bound
Error
Days From Prediction Day

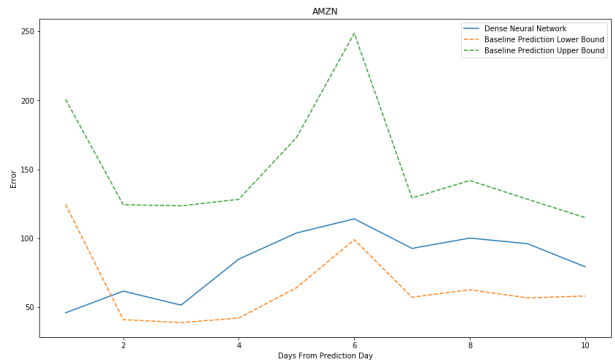

FB
Linear Regression
Dense Neural Network
RNN
LSTM
GRU
Baseline Prediction Lower Bound
Baseline Prediction Upper Bound
Error
Days From Prediction Day


FB
Dense Neural Network
Baseline Prediction Lower Bound
Baseline Prediction Upper Bound
Error
Days From Prediction Day


HOG
Linear Regression
Dense Neural Network
RNN
LSTM
GRU
Baseline Prediction Lower Bound
Baseline Prediction Upper Bound
Error
Days From Prediction Day


HOG
Dense Neural Network
Baseline Prediction Lower Bound
Baseline Prediction Upper Bound
Error
Days From Prediction Day


AMZN
Linear Regression
Dense Neural Network
RNN
LSTM
GRU
Baseline Prediction Lower Bound
Baseline Prediction Upper Bound
Error
Days From Prediction Day


AMZN
Dense Neural Network
Baseline Prediction Lower Bound
Baseline Prediction Upper Bound
Error
Days From Prediction Day

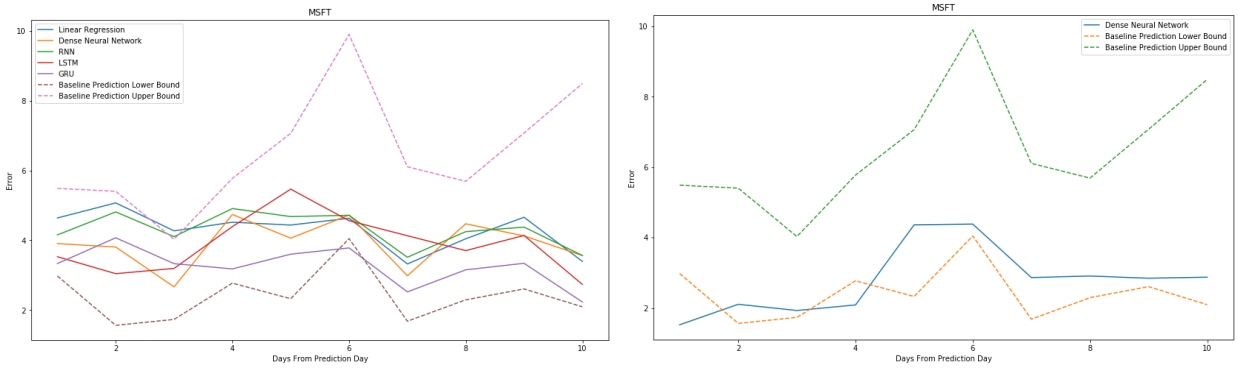
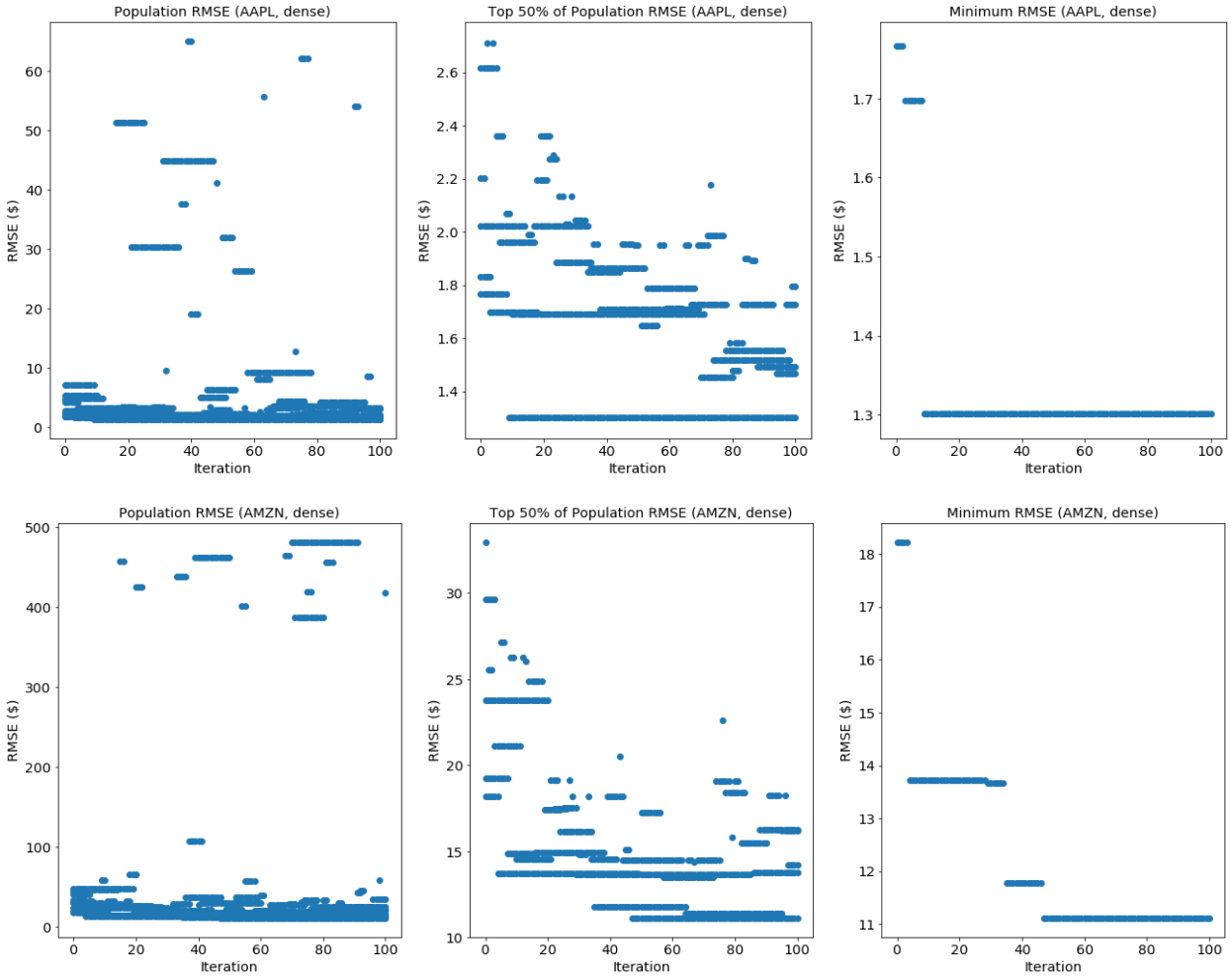
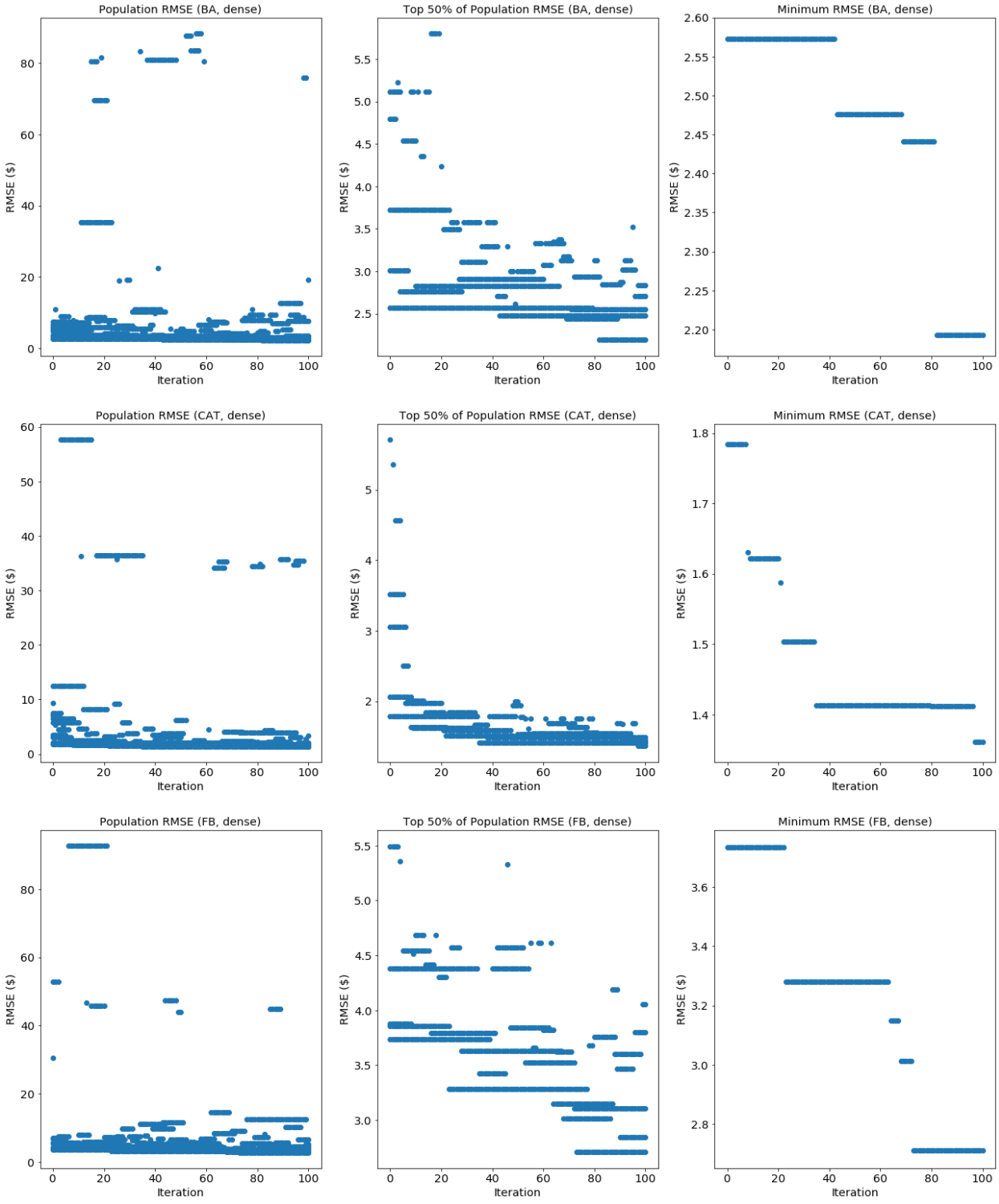


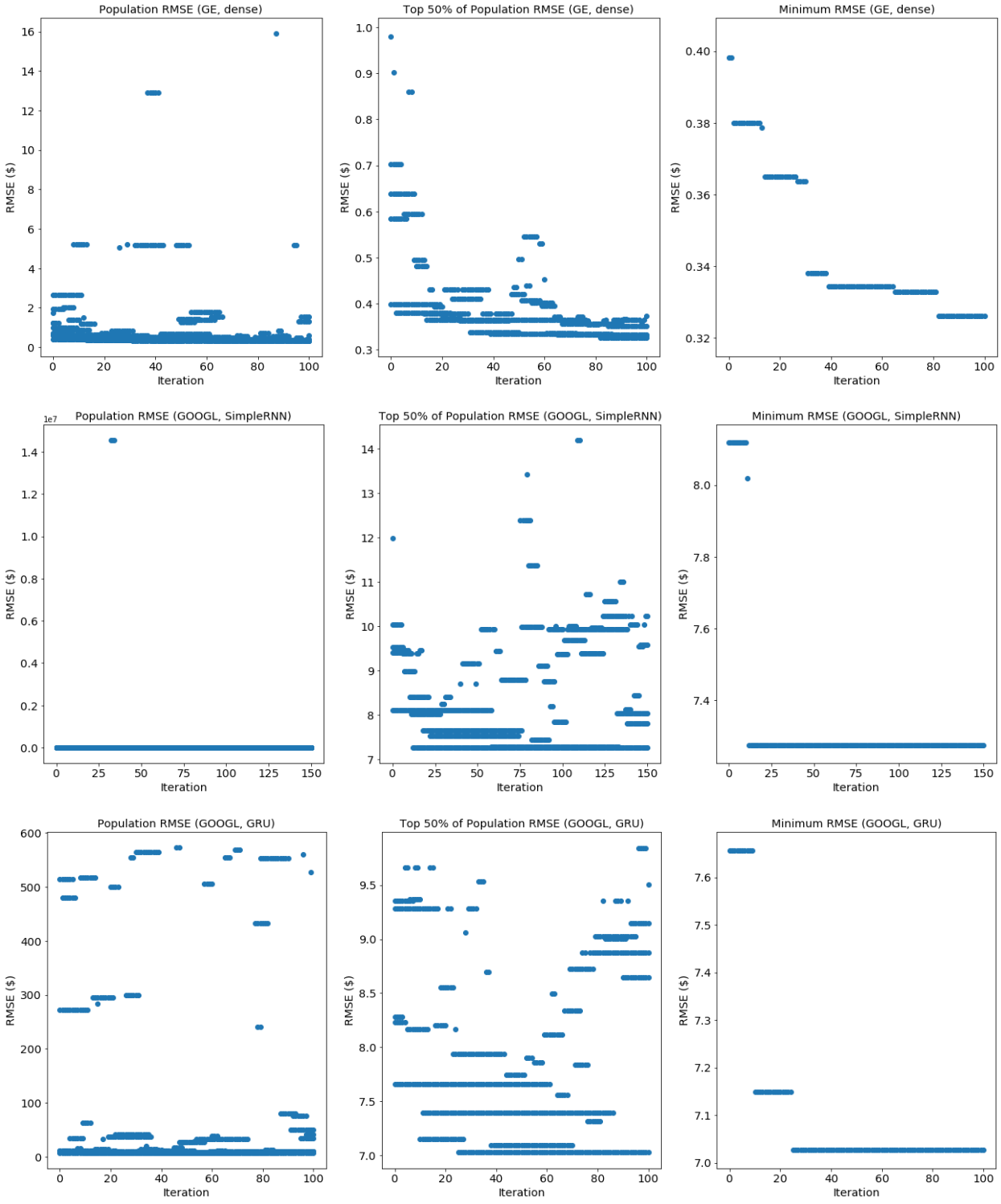
Figure 9.1 Prediction results from 12 different stocks in S&P 500

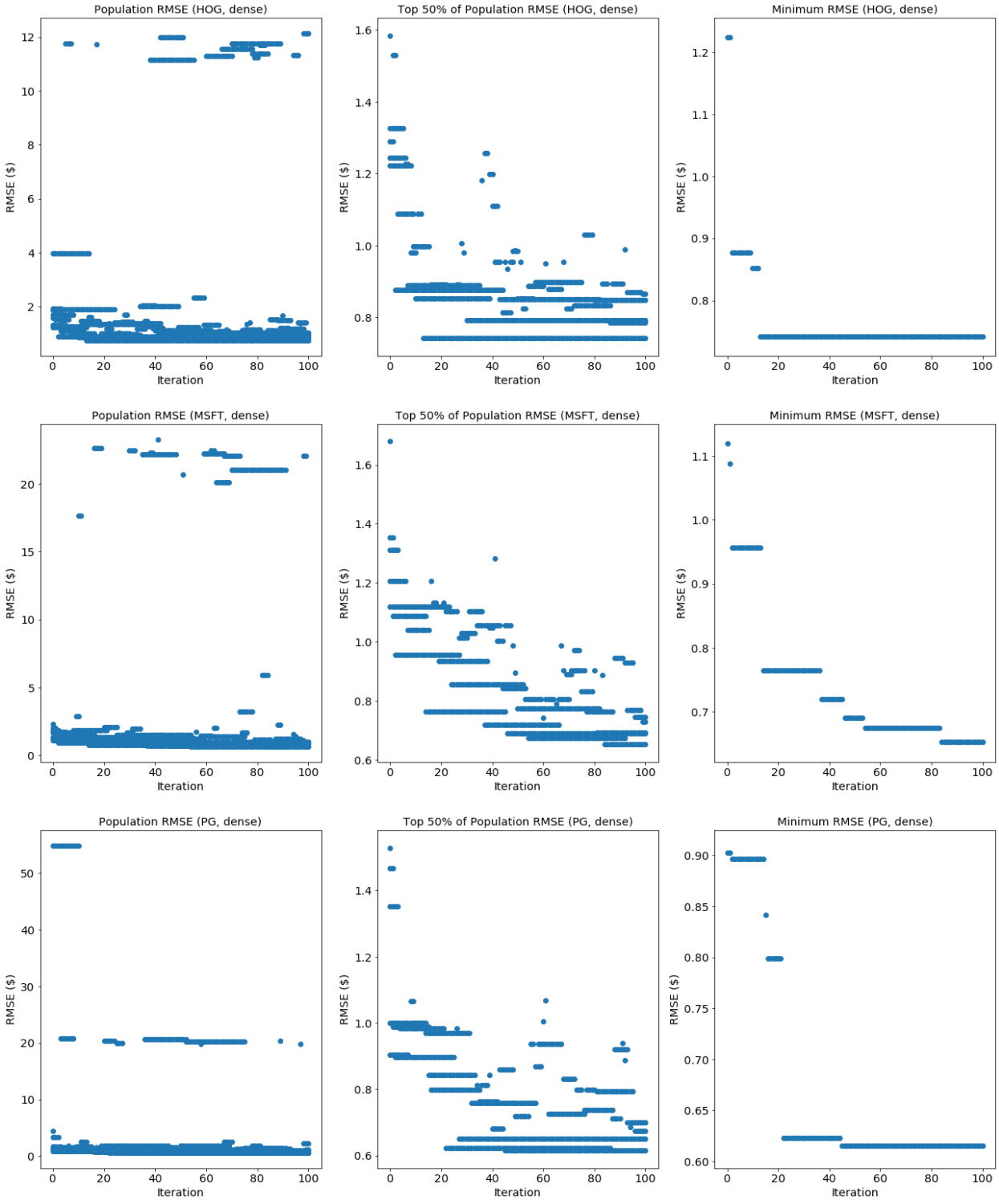
G - Evolution Algorithm Experiment Results

The following are the results of running the evolution algorithm for all 13 stocks. All results have shown that evolution is exploring better and better model architectures and hyperparameters over time.




Population RMSE (BA, dense)
Top 50% of Population RMSE (BA, dense)
Minimum RMSE (BA, dense)
Population RMSE (CAT, dense)
Top 50% of Population RMSE (CAT, dense)
Minimum RMSE (CAT, dense)
Population RMSE (FB, dense)
Top 50% of Population RMSE (FB, dense)
Minimum RMSE (FB, dense)
RMSE ($)
Iteration


Population RMSE (GE, dense)
Top 50% of Population RMSE (GE, dense)
Minimum RMSE (GE, dense)
Population RMSE (GOOGL, SimpleRNN)
1e7
Top 50% of Population RMSE (GOOGL, SimpleRNN)
Minimum RMSE (GOOGL, SimpleRNN)
Population RMSE (GOOGL, GRU)
Top 50% of Population RMSE (GOOGL, GRU)
Minimum RMSE (GOOGL, GRU)
RMSE ($)
Iteration


Population RMSE (HOG, dense)
Top 50% of Population RMSE (HOG, dense)
Minimum RMSE (HOG, dense)
Population RMSE (MSFT, dense)
Top 50% of Population RMSE (MSFT, dense)
Minimum RMSE (MSFT, dense)
Population RMSE (PG, dense)
Top 50% of Population RMSE (PG, dense)
Minimum RMSE (PG, dense)
RMSE ($)
Iteration

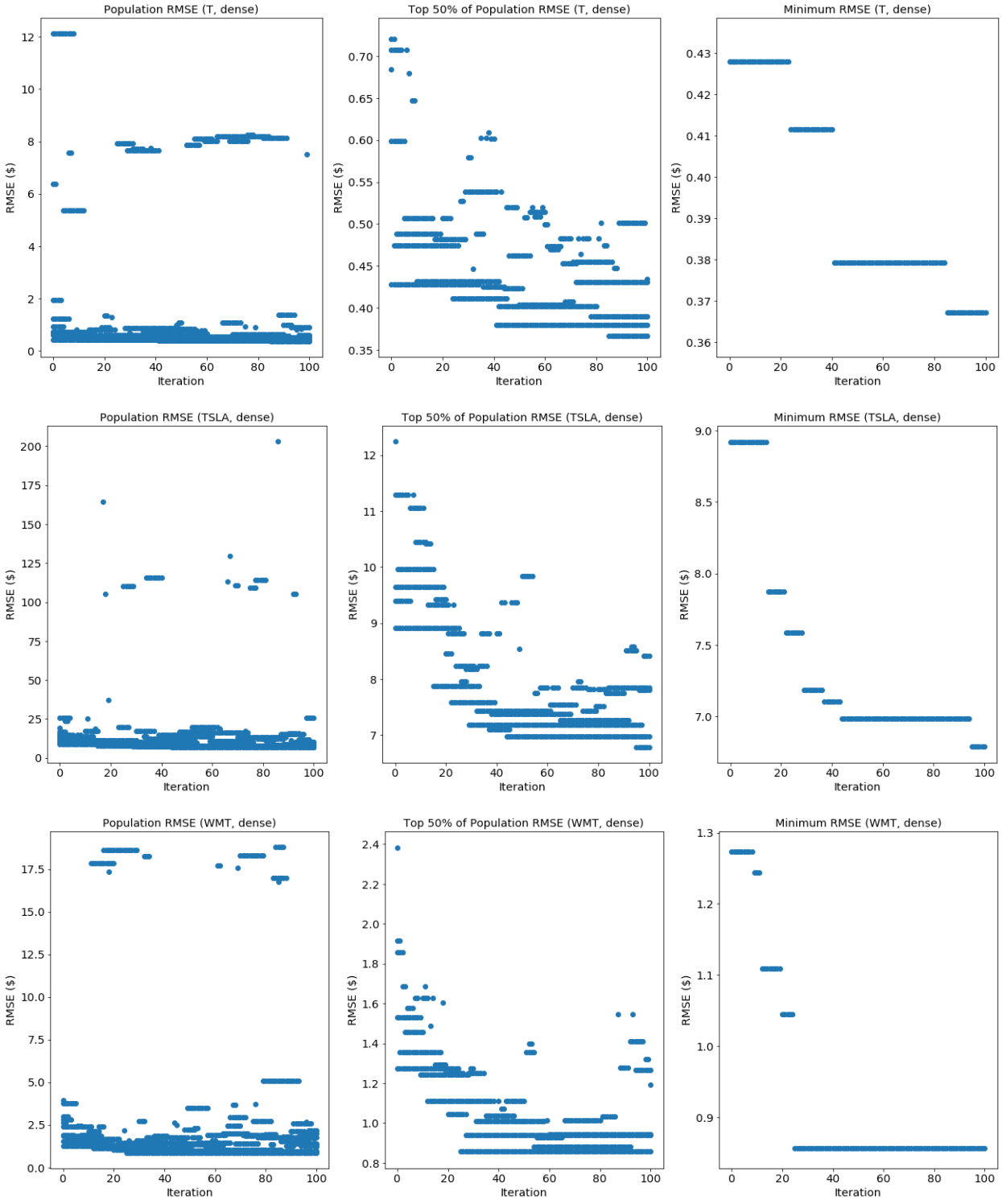


Figure 9.2 Evolution algorithm experiment results

H - (Project Planning) Division of Work

Task	Man	Cameron	Steven	Michael
Base Code Setup	✓✓	✓	✓	
App UI Design				✓✓
Model Exploration		(Leader)		
Trend line	✓	✓	✓	✓
Linear regression	✓	✓		
Neural network	✓	✓		
Recurrent neural network	✓	✓		
Long-short term memory network	✓	✓		
Pre-process and Build Training Dataset		✓		
Separate model and data		✓		
Options				✓
Evolution Algorithm	✓✓			
Architecture Search with Evolution Algorithm	(Leader)			
Dense Neural Network	✓	✓	✓	✓
Recurrent Neural Network	✓	✓	✓	✓
Long-Short Term Memory Network	✓	✓	✓	✓
GRU Network	✓	✓	✓	✓
Analyze algorithm results	✓	✓	✓	✓
Investors App				(Leader)
Basic layout	✓		✓	
Plot old stock price data	✓			
Change chart time frame			✓	
Toggle predictions			✓	
Search stock				✓
Company information				✓
Plot historical predictions				✓
Calculate upper and lower bound			✓	
Model details	✓			

Refine and polish UI	✓	✓	✓	✓
User Acceptance Test	✓	✓	✓	✓
Scoring for Investors			(Leader)	
Trend line			✓	
Model score formula	✓	✓	✓	✓
Buy/sell score formula	✓	✓	✓	✓
Calculate model score			✓	
Calculate buy/sell score			✓	
Investors Application Data	(Leader)			
Get old stock prices	✓			✓
Cron job to collect stock price daily	✓			
Cron job to predict stock prices daily	✓			
Report	(Leader)			
Proposal report	✓	✓	✓	✓
Draw system architecture		✓	✓	
Progress report	✓	✓	✓	✓
Final report	✓	✓	✓	✓

I - (Project Planning) Gantt Chart

Task	Start	Target End	Aug	Sep	Oct	Nov	Dec	Jan	Feb	Mar	Apr
Base Code Setup	Aug 10	Oct 15									
App UI Design	Aug 10	Oct 20									
Model Exploration											
Trend line	Oct 1	Oct 30									
Linear regression	Oct 16	Oct 30									
Neural network	Nov 2	Nov 6									
Recurrent neural network	Dec 21	Jan 11									
Long-short term memory network	Dec 21	Jan 10									
Pre-process and Build Training Dataset	Nov 7	Nov 13									
Separate model and data	Nov 7	Nov 16									
Options	Nov 7	Nov 16									
Evolution Algorithm	Dec 18	Jan 29									
Architecture Search with Evolution Algorithm											
Dense Neural Network	Feb 1	Feb 22									
Recurrent Neural Network	Feb 15	Mar 1									
Long-Short Term Memory Network	Feb 15	Mar 1									
GRU Network	Feb 15	Mar 1									
Analyze algorithm results	Mar 15	Mar 29									

Investors App											
Basic layout	Aug 21	Sep 30									
Plot old stock price data	Aug 22	Sep 25									
Change chart time frame	Oct 1	Oct 30									
Toggle predictions	Oct 1	Oct 30									
Search stock	Oct 1	Oct 30									
Company information	Dec 21	Jan 24									
Plot historical predictions	Jan 24	Feb 8									
Calculate upper and lower bound	Jan 24	Feb 8									
Model details	Feb 4	Feb 8									
Refine and polish UI	Feb 15	Mar 8									
User Acceptance Test	Mar 15	Mar 29									
Scoring for Investors											
Calculate trend line model score	Nov 7	Nov 14									
Model score formula	Feb 1	Feb 8									
Buy/sell score formula	Feb 1	Feb 8									
Calculate model score	Feb 8	Mar 1									
Calculate buy/sell score	Feb 8	Mar 1									
Investors Application Data											
Get old stock prices	Aug 20	Sep 30									
Cron job to collect stock price daily	Mar 1	Mar 15									
Cron job to predict stock prices daily	Mar 15	Mar 22									
Report											
Proposal report	Aug 30	Sep 20									
Draw system architecture	Dec 21	Feb 13									
Progress report	Feb 1	Feb 14									
Final report	Mar 15	Apr 17									

J - Collaboration

Git and GitHub

Git is used for version control. 4 GitHub repositories [39] are created, one for each sub-system in the project, app, AI server, Node.js data server, and Firebase Cloud Functions. Each new feature is implemented on a separate branch, maintained by 1 developer, and merged into master when ready.

Task Management

The whole project is divided into small tasks in the research or the application side. A Google Sheet is used to manage the progress of each task, which describes the task details, the person responsible for working on it, the start date, the expected finish date, and the actual finish date. The Git branch which the work is written on is also recorded.

42	Copy technologies used from proposal to report	Report	Man	2018-12-21		2019-02-01	
43	StockScore render	PWA	Cameron	2018-12-21			
44	Fix prediction plotting	PWA	All	2018-12-21			
45	Company information	PWA	Michael	2018-12-21	2019-01-24	2019-01-24	company_info
46	GRU	AI	Cameron	2019-01-11	2019-01-11	2019-01-11	lstm
47	plot snakes (historial prediction)	AI	Cameron	2019-01-24	2019-01-30	2019-01-30	
48	calculate upper/lower bound	AI	Cameron	2019-01-24	2019-01-30	2019-01-30	
49	calculate error + rating + 紅綠燈	AI	Steven	2019-01-29	2019-02-08	2019-02-08	
50	UI display (plot snakes) - with hardcode data	PWA	Michael	2019-01-29	2019-02-08	2019-02-08	plot-snakes
51	UI display (plot uppper/lower bound) - reference to "predictions" array of design	PWA	Michael	2019-01-29	2019-02-08	2019-02-08	plot_upperLowerBound
52	UI display (error and rating) - up/down arrow + scoring display	PWA	Steven	2019-01-29	2019-02-08	2019-02-08	
53	UI display (Traffic light)	PWA	Steven	2019-01-29	2019-02-08	2019-02-08	
54	Basic report sections	Report	Man	2019-02-01		2019-02-01	
55	Evolution	Report	Man	2019-02-01			
56	Model details	PWA	Man	2019-02-04		2019-02-08	model-details
57	Model details	AI	Man	2019-02-04		2019-02-08	model_options
58	Predict next day	AI	Man	2019-02-15	2019-02-22		predict-output-size

Sheet1 | Gantt Chart | Division of Work

Figure 9.3 Task Management List on Google Sheet

Meeting

The team started with general meetings from August to mid-October, discussing the idea and the system design. Later on, the project is divided into small tasks, and each member worked individually and report to the group the features added or changes made once finished. Meetings were held weekly to discuss the next stage of features to work on, while most discussions were made on WhatsApp. There were a small to-do or to-discuss list before each meeting. Monthly meetings were held with the professor to report the progress and changes in idea if any, and seek advice for things to do.

Starting from February, the project stepped into the testing phase, which every member tested out numerous models with the evolution algorithm. Weekly meetings are scheduled on every Thursday afternoon to discuss and analyze different models and findings.

K - Meeting Minutes

Date: 2018-11-07

Time: 09:00 - 18:00
Team Members: Man, Cameron, Steven, Michael

- Walked through the whole system design and architecture
- Discussed the project plan
- Discussed separating model and data and input configs
- Discussed input configs format
- Discussed how to calculate the error and score of trend lines
- Discussed how to do evolution with options
- Discussed options format

To-dos:

Man	● General linear regression and support vector regression ● Separate model and data (Re-write train_models.py and save_predictions.py)
Cameron	● Build dataset based on input configs
Steven	● Calculate trend line models error and score
Michael	● Generate random options and mutate options based on option configs

Date: 2018-11-09

Time: 15:00 - 17:00
Team Members: Man, Cameron, Steven, Michael

- Worked on linear regression and support vector regression models
- Worked on trend line models
- Worked random option generation
- Discussed and defined input config formats

To-dos:

Man	● General linear regression and support vector regression ● Separate model and data (Re-write train_models.py and save_predictions.py)
Cameron	● Build dataset based on input configs
Steven	● Calculate trend line error and score
Michael	● Generate random options and mutate options based on option configs

Date: 2018-11-13

Time: 12:00 - 13:30
Team Members: Man, Cameron, Steven, Michael

- Worked on trend line models
- Worked on random option generation
- Worked on the module to build the dataset based on input config
- Discussed documentation issues

To-dos:

Man	● Document and comment code
Cameron	● Build dataset based on input configs
Steven	● Calculate trend line models error and score
Michael	● Generate random options and mutate options based on option configs

Date: 2018-11-14

Time: 15:00 - 16:00
Team Members: Man, Cameron, Steven, Michael

- Worked on documentation and comments
- Worked on trend lines

To-dos:

Man	● Document and comment code
Cameron	● Build dataset based on input configs
Steven	● Calculate trend line models error and score
Michael	● Generate random options and mutate options based on option configs

Date: 2018-11-16

Time: 15:00 - 16:00
Team Members: Man, Cameron, Steven, Michael

- Worked on documentation and comments
- Worked on options generation and mutation based on option config

To-dos:

Man	● Document and comment code
Cameron	● Build dataset based on input configs
Steven	● Calculate trend line models error and score
Michael	● Generate random options and mutate options based on option configs

Date: 2018-11-26

Time: 11:30 - 12:30

Team Members: Man, Cameron, Steven, Michael

- Discussed implementation of recurrent neural networks, e.g. LSTM
- Discussed evolution algorithms
- Discussed the possible additional features, e.g. company information on mobile application

To-dos:

Man	● Evolution algorithms
Cameron	● Recurrent Neural Network, LSTM
Steven	● Calculate trend line models error and score
Michael	● Displayed Company information on Mobile Application

Date: 2019-01-24
Time: 14:00 - 17:00
Team Members: Man, Cameron, Steven, Michael

- Discussed how to calculate accuracy, upper bounds and lower bounds of different models
- Discussed new interfaces between server and mobile clients to allow upper/lower bounds and previous predictions to be passed
- Discussed how to present the upper/lower bounds and accuracy on mobile application

To-dos:

Man	● Evolution algorithms
Cameron	● Calculate upper/lower bounds for stock price predictions ● Get previous predictions to show stock prices' accuracies
Steven	● Plot score indicators for different models
Michael	● Plot upper bounds/lower bounds for stock price predictions on mobile application ● Plot previous predictions on mobile application

Date: 2019-02-08

Time: 13:00 - 15:00

Team Members: Man, Cameron, Steven, Michael

- Worked on calculating accuracy, upper bounds and lower bounds of different models
- Worked on plotting previous predictions, upper bounds and lower bounds of different models in the mobile application
- Discussed progress report

To-dos:

Man	● Evolution algorithms ● Progress Report
Cameron	● Calculate upper/lower bounds for stock price predictions ● Get previous predictions to show stock prices' accuracies ● Progress Report
Steven	● Plot score indicators for different models ● Progress Report
Michael	● Plot upper bounds/lower bounds for stock price predictions on mobile application ● Plot previous predictions on mobile application ● Progress Report

Date: 2019-02-15

Time: 13:00 - 15:00
Team Members: Man, Cameron, Steven, Michael

- Discussed one-day prediction option
- Discussed formulation of model scores
- Discussed formulation of buy/sell scores

To-dos:

Man	● Implement rolling one-day predictions of stock prices
Cameron	● Implement build_dataset for one-day predictions of stock prices
Steven	● Calculate model score ● Calculate buy/sell score
Michael	● Plot model score ● Plot buy/sell score

Date: 2019-02-21
Time: 16:30 - 18:00
Team Members: Man, Cameron, Steven, Michael

- Separated build dataset script to build training dataset and build predict dataset
- Unit-tested build dataset script
- Discussed the uses of Facebook authentication
- Discussed the design of user profile
- Discussed the design of user page

To-dos:

Man	● Separate build dataset to build training dataset and predict dataset ● Write unit tests for build_dataset
Cameron	● Write unit tests for build_dataset
Steven	● Calculate model score ● Calculate buy/sell score
Michael	● Plot model score ● Plot buy/sell score

Date: 2019-02-28

Time: 16:30 - 18:00

Team Members: Man, Cameron, Steven, Michael

- Discussed what config options to include in the evolution model
- Discussed normalization of stock prices
- Discussed user interface changes
- Demonstrated progress for stock price predictions (in successfully comparing different models and plotting it in graphs)

To-dos:

Man	● Normalization of stock prices ● Evolution config options
Cameron	● Favourite list ● Timeframe select
Steven	● Calculation model score and plot score
Michael	● Toggle for the user to choose whether they want to plot “rollingPredict” or “snakes” historical predictions (or both)

Date: 2019-03-04

Time: 16:30 - 18:00

Team Members: Man, Cameron, Steven, Michael

- Discussed the overall user interface design
- Discussed possible improvements to the UI
- Discussed how to start evolution

To-dos:

Man	● Host server ● Evolution config options
Cameron	● Start doing evolution
Steven	● Make improvements on the UI ● Start doing evolution
Michael	● Start doing evolution

Date: 2019-03-14

Time: 16:30 - 18:00

Team Members: Man, Cameron, Steven, Michael

- Discussed model score formula
- Discussed UI improvements
- Discussed prediction results
- Discussed evolution scope

To-dos:

Man	● Revise model score formula ● Work on evolution
Cameron	● Work on evolution
Steven	● Revise model score formula ● Work on evolution
Michael	● Work on evolution

Date: 2019-03-28

Time: 16:30 - 18:00
Team Members: Man, Cameron, Steven, Michael

- Reviewed page & component diagram for our web application
- Discussed what we need for our final report
    - New findings regarding evolution algorithm
    - Results of hallway testing
    - Unit tests
    - Implementation result (with component diagram)
- Use Google slides for final presentation

To-dos:

Man	● Work on evolution ● Host the application to the server
Cameron	● Work on evolution
Steven	● System diagram ● Work on evolution
Michael	● Work on evolution ● Add chart settings user profile to cloud ● Add maximum time frame

Date: 2019-04-08
Time: 12:00 - 18:00
Team Members: Michael, Man, Cameron

- Discussed the flow of the final report
- Use case diagram / Functions
- Questions to the professor
    - Should we put mockup or real
    - Should we put application/research part first in our design (2.2 - Application, 2.3 Research)

To-dos:

Man	● Plot evolution results as graphs ● Start working on the final report
Cameron	● Start working on the final report
Steven	● Start working on the final report
Michael	● Start working on the final report

L - Required Hardware and Software

Hardware

- 4 x Windows / Linux / MAC laptops for development

Software

- Python 3.6 with machine learning libraries (e.g. scikit-learn, Keras, Tensorflow)
- Visual Studio Code / Sublime Text for programming
- Google Chrome for debugging web applications

Platforms

- Google Colaboratory for running evolution
- Firebase Hosting for app
- Firebase (Cloud Datastore, Cloud Storage, Authentication, Cloud Functions)