COMP4801

Individual Final Report

Group Topic:
Market News Analysis and Stock Price Prediction

Individual Topic:
Predicting Stock Price with Machine Learning and Optimization

Author
Chiang Cheng-Ru, 3035123538

Group Members
Huang Hsiang-Jui, 3035248065
Wang Ching-Yuan, 3035242499

Supervisor
Dr. S.M. Yiu
Abstract

With the rapid advancement of the data analytics field, there are a number of readily available software frameworks and tools that assist users to build applications for stock price prediction via analyzing data. In this project, the team has explored several data analytics techniques, including Natural Language Processing and Machine Learning, to build a standalone application with modularized components. With the outputs produced by this application, the ultimate goal is to achieve more accurate stock trend prediction. This report details the process that the team followed to conduct experiments and construction of the project, as well as the results that the team has achieved. The team has improved one of the word vectorization methods, context2vec, and Genetic Algorithm using the Simulated Orthogonal Array. Also, another focus of this project is to see whether the inclusion word vectorization data will increase the accuracy of the prediction, and the results have shown that in regards to short term return, using word vectorization data yielded more positive result than using simply historical price, even with sentiment score included.

Acknowledgement

I would like to express my gratitude towards Dr. S.M. Yiu for offering suggestions whenever the team encounters issues during the development. I would also like to thank my English instructor, Mr. Cezar Cazan, for giving clear guidelines on writing academic papers.
# Table of Contents

**Abstract** 1

**Acknowledgement** 1

**1. Introduction** 5

  1.1 background 5
  1.2 Objective 5
  1.3 Scope 6
  1.4 Outline 7

**2. Literature review** 7

  2.1 Approaches to Stock Price Prediction 7
  2.2 Word Embedding 8
  2.3 Stock Prediction and Model Optimization 9

**3. Methodology** 10

  3.1 Hardware Setup 10
  3.2 Software Setup 10
  3.3 Data collection 11
  3.4 Inputs formats 12
    3.4.1 Historical 13
    3.4.2 Sentiment 13
    3.4.3 c2v+_headline 14
  3.5 Stock Prediction 17
    3.5.1 Model Construction - OO approach 17
    3.5.2 Prediction Method and Evaluation Function 18
    3.5.3 Random Search for Hyperparameters 18

**4 Optimization** 19

  4.1 Multi-Trajectory Search 19
    4.1.1 Problem definition 19
    4.1.2 Simulated Orthogonal array(SOA) 20
    4.1.3 Procedure 21
  4.2 Genetic Algorithm 21
    4.2.1 Problem definition 22
    4.2.2 Procedure 22
  4.3 Integration of Simulated Orthogonal Array into Genetic Algorithm 24
5 Results and Analysis

5.1 Optimization results
5.2 Comparison of Optimization methods Performance
5.3 Integration of Simulated Orthogonal Array into Genetic Algorithm
5.4 Model performances
5.5 Data format performances
   Sentiment Performance Evaluation
   w2v_headline & w2v_content Performance Evaluation
5.6 Robustness Checking

6 Conclusion

Reference

Appendix
A: POS Tag List
B: SOA Creation Code
List of figures

Figure 1. Word Vectorization Visualization 9
Figure 2. IYE Index top 10 holdings 11
Figure 3. Tree representation of sentence structure 15
Figure 4 Software Design 17
Figure 5 Loss Calculation 18
Figure 6. MTS on RNN, short term return 25
Figure 7. GA on RNN, short term return 25
Figure 8. Model performance 29
Figure 9. Comparison on Historical and Sentiment on different terms of return 31
Figure 10. RNN on different data inputs, on short term return 32
Figure 11. MLP on different data inputs, on short term return 33
Figure 12. RNN on different data inputs, on mid term return 34
Figure 13. RNN on different data inputs, on long term return 34
Figure 14. c2v+_content benchmarking 35
Figure 15. IYE Index AdjClosing Price 36
Figure 16. SVM robustness test 37
Figure 17. RNN robustness test 38

List of Tables

Table 1. List of types data inputs 13
Table 2. POS tagging representation 16
Table 3. Hyperparameters for networks 19
Table 4. Comparison between Optimizations and Random Search 26
Table 5. Comparison between GA and MTS on RNN 27
Table 6. Comparison between GA and MTS on SVM 27
Table 7. generations required to find minimum loss 28
Table 8. minimum loss of the models 30
Table 9. Benchmark on short, mid, and long term return 30
1. Introduction

1.1 background

In the finance and the data analytics field, stock prediction has been a focus of research throughout the past few decades. Researchers are looking into different factors, such as numeric past stock price, governmental policies, macroeconomics data and etc., in an attempt to construct a better the prediction model. For example, one research used time series prediction using the historical stock data(Kaastra and Boyd, 1996[1]), and some other researches had similar focus(Skabar and Cloete, 2002[2]; Chan et al., 2000[3]). Due to the advancing development of the data analytics fields, including machine learning, natural language processing, more researches have started to include different data sources, apart from mere numerical data. For example, market news (Li, 2013[4]), financial reports (Lee, 2014[5]) and Twitter sentiment (Si, 2013[7]).

In this project, the team attempts to build finance-specific prediction models using machine learning and natural language processing technique. This report will mainly focus on the latter, where details on experimenting with different types of data input sources, and attempts to optimize the prediction models will be discussed.

1.2 Objective

The objective of this project is to build a standalone application that predicts the future stock movement, in the form of “up” and “down” based on both financial market news, and numerical price data from credible online sources, and the betterment of the prediction models by fine tuning their parameters using optimization algorithms.
The content of this report will detail the prediction model construction, and the optimization of the models. More specifically, this report will cover the following items.

- Software Design for the training components.
- Data source and input format
- Adopted prediction models
- Optimization methods and benchmarking
- Model performance explained

1.3 Scope

The scope of this project concerns 2 main modules, the Text Analysis Model exploiting the Natural Language Processing Technique, and the Price Prediction Model, using the Machine Learning Technique. The deliverable of the Text Analysis Model will be a proposed data set that integrates word vectorization, and sentiment analysis that yields good prediction results on the Price Prediction Model as input data. This module will also deliver an adaptation of the word embedding technique, known as context2vec.

In regards to the Price Prediction Model, the scope entails the analysis of different combinations of data input that takes different factors into account, such as the term of stock terms (short, mid, long), and the features, such as historical price, sentiment score, and word vectorization data.

This Fintech Application will not have a user interface, and users will have to communicate with it using a Command Line Prompt. Users can specify the kind of data format, optimization method, kind of machine learning models, and the return term they want to train the model with.

This application will only be able to analyze the financial news from the US energy market, and the reason will be further discussed in the Methodology section. The majority of the work will focus on the ways to conduct analysis with the data, and to make prediction accordingly.
1.4 Outline

The remainder of this report proceeds as follows. First, the project background section will introduce some theories and the relevant findings of this project, followed by the methodology which entails the implementation of the project. The experiment results of the usage of different data inputs and the model performance will then be discussed, and finally the report will end with the future work of this project.

2. Literature review

This section will detail some of the theoretical concepts that are related to this particular project. The section will start with the motivation of the project by introducing two of the main approaches to predicting stock price, and then it will move on to the introduction to the Natural Language Processing and Machine Learning Techniques, which are the major components of this project. The past research relevant to these topics will also be briefly mentioned, including how the word embedding is done, the kinds of data used for prediction, kinds of machine learning models used, and how the models can be further improved by the adoption of some optimization methods.

2.1 Approaches to Stock Price Prediction

Currently there exist two approaches to predicting the financial market movement - fundamental and technical. The technical approach conducts price prediction via consuming the historical quantitative market data. However, the dispute against such approach contends that it is impossible to conduct price prediction based on the historical market trend data, due to its context-specific nature. This argument is also supported by the Random Walk Theorem. Random Walk Theorem is widely adopted as a method for simulating the stock price in the market. To model a random walk in the market, many different mathematical forms are used based on the assumption one has. A common form would be Stochastic Process, which is denoted as,
\[ dS_t = \mu S_t dt + \sigma S_t dW_t \]

where \( dS_t \) represent the change of the stock price, \( W_t \) is the Brownian motion, \( \mu \) denotes the percentage drift term and and \( \sigma \) as the percentage volatility which is constant. However, the existence of such theorem only occurs in the efficient markets, and therefore there are still some research conducting time series analysis on historical price data. For example, Chen and Zhou leveraged the LSTM recurrent neural network to predict the stock price in China in 2015[12]. Also, Hsieh et al(2013) used the time series data, closing stock price for example, with linear regression, to find the underlying pattern[11].

On the contrary, the fundamental approach consumes other types of information, such as data related to the financial environment, geopolitics and etc. In the fundamental approach of predicting the financial market movement, aside from referencing numeric data obtained from the financial reports as benchmarks, analysts also look into textual data, which is available through the published financial news. The textual source varies from market news from Reuters and Bloomberg (Xie, 2013[6]), financial reports and SEC filing (Lee, 2014[5]) and Twitter sentiment (Si, 2013[7]). Thomas and Sycara used Genetic Algorithm to learn the significant rules in trading stock, along with text classification that processes textual posts from financial related websites, to predict “up”s and “down”s, and generated excess returns[9].

2.2 Word Embedding

Word embeddings is a technique that transforms the human comprehension of languages into machine representation. The aim of word embeddings is to include the semantic and syntactic meaning of words in a corpus. An ideal word vectorization should be able to correlate different words based on their semantic and syntactic meanings. An example is shown below.
Based on Figure 1, some context specific words can be inferred by giving other example words. Semantically, when given vectorization of three words, King, Man, and Woman, the model should be able to infer the fourth word, which is Queen in this case.

\[
\text{Vec}[^0\text{King}] - \text{Vec}[^0\text{Man}] + \text{Vec}[^0\text{Woman}] = \text{Vec}[^0\text{Queen}]
\]

Similarly, in the syntactic case, when given the words walking, walked and swam, the model should be able to get the fourth word, which is swimming.

\[
\text{Vec}[^0\text{walking}] - \text{Vec}[^0\text{walked}] + \text{Vec}[^0\text{swam}] = \text{Vec}[^0\text{Swimming}]
\]

On of the hot research topics in this field will be Word2Vec, which is a word embedding model that is developed by Google. (Mikolov et al., 2013).

2.3 Stock Prediction and Model Optimization

As with some research mentioned in section 2.1, numerous researches have been done in this area. In many of the researches, Support Vector Machine(SVM), Recurrent Neural Network(RNN), and Multi-Layer Perceptron(MLP) are used, as they generally produce more desirable results, meaning higher accuracy, or lower loss[1][2][3][11][12]. To further improve
the prediction results, some have resort to tuning the parameters of the models. For example, Srivastava et al adopted the dropout method to avoid overfitting. Tseng and Chun(2008) proposed Multi-Trajectory Search for large scale global optimization problems. Choudhry, Rohit, and Kumkum Garg implemented Support Vector Machine along with Genetic Algorithm to achieve Stock Market Forecasting[15].

In this project, the team has adopted the Multi-Trajectory Search and Genetic Algorithm in search for better hyperparameters combination in a given search space.

3. Methodology

In this section, the concrete implementation of the project will be detailed. Note that since this report only focuses on Machine Learning and Optimization, Data collection and Natural Language Processing will not be discussed in fine details. The reason for including such content is only to provide a higher level of understanding of this project. Please refer to Wang Ching-Yuan’s report for more details on Data Collection and Natural Language Processing.

3.1 Hardware Setup

Due to the large computational power needed for machine learning model training and optimization, the team has deployed the computation load on to the GPU server in the HKU GPU Farm. The server comes with an NVIDIA GeForce GTX 1080 GPU, which comes with 2560 Tensorcores and 8GB memory. The server itself also has Dual Xeon 8C/16T CPUs, and 128GB RAM.

3.2 Software Setup

The softwares that are used for the construction of the project are listed as below(only includes the one that Machine Learning models and Optimization used)

- python 3.6
- Nvidia CUDA 10.0.130
3.3 Data collection

Due to the context-specific nature of the textual data, the team has narrowed down the news data input only to only which concerns the US energy industry, given they contain less noises, such as fakes news. Plus, the relevant terms used in the news are less ambiguous than other sectors. News of such data sources is collected from the Financial Times website.

As to the target stock, the team chose the IYE index as the target stock to conduct prediction on. A distribution of top ten holdings is shown in the figure below. IYE is chosen in this project due to its high liquidity rating. With high liquidity, stock prices can better reflect the market information.

![IYE Top 10 Holdings](image)

Figure 2. IYE Index top 10 holdings

The news data collected are then inputted into the pipeline where sentiment score and word vectorization values are generated, and then passed to the Machine Learning Module along with the stock price data.
3.4 Inputs formats

Before the data gets plugged into the machine learning modules, some processings are to be done to ensure a uniform data format. In this project, the team has experimented with different kinds of inputs to see which combination will yield better predictions under the precision models. Table 1 lists the combinations the team has experimented with. The abbreviation column indicates the namings that are used for the plottings in this report.

<table>
<thead>
<tr>
<th>Input type</th>
<th>Abbreviation</th>
<th>X(feature columns)</th>
<th>y(prediction labels)</th>
</tr>
</thead>
<tbody>
<tr>
<td>plain price</td>
<td>historical</td>
<td>• High</td>
<td>return*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Low</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Closing</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• adjusted closing</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• open</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• volume</td>
<td></td>
</tr>
<tr>
<td>price with sentiment score</td>
<td>sentiment</td>
<td>• high</td>
<td>return*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• low</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• closing</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• adjusted closing</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• open</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• volume</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>headline sentiment score</strong></td>
<td></td>
</tr>
<tr>
<td>price with word vectorization</td>
<td>c2v+_headline</td>
<td>• high</td>
<td>return*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• low</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• closing</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• adjusted closing</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• open</td>
<td></td>
</tr>
<tr>
<td>price with word vectorization and sentiment score</td>
<td>c2v+_content</td>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>-------------</td>
<td>------</td>
<td>-----</td>
</tr>
</tbody>
</table>

Table 1. List of types data inputs

*return here is further divided into **short term**(price change compared to the previous working day), **mid term**(price change compared to the previous 5 working days), and **long term**(price change compared to the previous 20 working days).

### 3.4.1 Historical

The historical data consists of feature columns that are directly related to the price of the stock. The feature columns are, High, Low, Closing, Adjusted Closing, Open, and Volume.

### 3.4.2 Sentiment

The sentiment dataset, aside from the return dataset, has an additional sentiment score column appended to the whole matrix. The sentiment score represents the sentiment of the news headlines. The score is calculated by the NLTK Vadar Sentiment Analyzer, and it will represent the positivity/negativity of the news. For instance, the piece of news with the headline “Tesla and
Panasonic freeze Gigafactory expansion plans” might yield a negative sentiment score, and the implication on stock price will correlate to a drop.

However, the Vadar Sentiment Analyzer does not take the subjects of the sentences into account, and this could possibly lead to incorrect sentiment score generation, meaning the sentiment score is correct, but the sentiment does not come from the subject of interest. Such situations might happen in articles where the substitutions of the industry of interest, US Energy sector in this case, are present. For example, the sentence “When US imported oil, a strong dollar and lower oil prices were a good thing for families and businesses”, will possibly have a positive sentiment score outputted. However, this is not the sentiment score of the subject we are interested in. A proposed method would be using word vectorization to locate the keywords of the sentences to avoid words that are not of interest.

3.4.3 c2v+_headline

This dataset, also contains all the feature columns that sentiment dataset has, has additional 20 columns which contain the word vectorization of the 10 most significant words in the article, meaning each selected word is represented by 2 feature columns. Note that the 2 columns are the result of the dimensionality reduction using t-SNE. The original dimension was 300.

Keywords in a sentence are normally subjects of the sentence, and they possess significant meaning for the sentence. Without the keywords, the meaning of the sentences will then become unclear as there is no way to know what the sentences are referring to.

Before finding the keywords, the sentences to be processed will first be tokenized into separate words. For example, the sentence “Can you please buy me an Arizona Ice Tea? It’s $0.99.” will be tokenized into the list ['Can', 'you', 'please', 'buy', 'me', 'an', 'Arizona', 'Ice', 'Tea', '?', 'It', '“s', '$', '0.99', '.'] Note that the symbols ‘?’ , ‘,’ , ‘$’ are being separated and retained as well, as they do represent meaning semantically in certain parts of the sentence. With the list of tokenized words and symbols, pos taggings are performed to give each element in the tokenized list symbols to represent its relationship with other words that are in the same corpus. The graph below demonstrated how POS taggings are conducted.
Here we have a sentence “I shot an elephant in my pajamas”. The Words in capital letters are the POS tags. According to the NLTK book, to define the subject of a sentence $S$ in English, we can do it by identifying the noun phrase that is the child of $S$ and the sibling of $VP$. Applying this principle to the sentence shown above, “I” is a noun phrase, which is a child of $S$ and a sibling of $VP$, and therefore “I” would be the subject of the sentence. Refer to the appendix for more information on the pos taggings.

We applied this method to locate all the subjects words in a news article, and then selected the top 10 words with the highest financial related score $Z$. The financial related score $Z$ is defined as

$$Z = \frac{\text{# of occurrence in } FWB}{\text{# of occurrence in } GWB}$$
FWB is defined as the Financial Word Base and GWB is defined as the General Word Base. With the defined score $Z$, the words which have more appearances in the Financial word base will obtain higher score, and these words will have higher significance compared to other words with lower $Z$ score. Thus, the data format would be a vector $X$: 

$$X = [Vec, Vec, ..., Vec, Open, High, Low, Close, adjClose, Volume, Sentiment]$$

### 3.4.4 c2v+_content

With the aim to further improve the quality of the information contained in the dataset, instead of using the sentiment score of the whole news article, as done with the c2v+_headline dataset, we selectively calculate the sentiment of some certain words in the same sentence of a keyword. Below shows an example of this operation.

Here we have a sentence “The costs of renewable energy and storage are continuing to fall”, and the result after the POS tagging process is shown in Table 2.

<table>
<thead>
<tr>
<th>The costs of renewable energy and storage are continuing to fall</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT NNS CC JJ NN CC NN VB VBG TO VB</td>
</tr>
</tbody>
</table>

Table 2. POS tagging representation

In this example, the word “cost” will be the subject, and the words “renewable”, “continuing”, and “fall”, will contribute to the sentiment score calculation, which will be -0.2732 in this case. To make word selection even more specific, if the subject word is also contained in the target stock names, company names, or energy terms in IYE, it will be prioritized for selection. For the $Z$ score formula, it is revised as the follows in order to account for the sentiment in the same sentence as the subject word.
\[ Z = |\text{sentiment}| \times \frac{\# \text{ of occurrence in } FWB}{\# \text{ of occurrence in } GWB} \]

Thus, the data format would be a vector \( X^- \):

\[ X^- = [\text{Vec}, \text{Vec}, \text{Senti,...}, \text{Vec}, \text{Vec}, \text{Senti}, \text{Open}, \text{High}, \text{Low}, \text{Close}, \text{adjClose}, \text{Vol}] \]

### 3.5 Stock Prediction

#### 3.5.1 Model Construction - OO approach

This model construction section concerns the building process for the stock prediction module. To avoid writing up similar scripts for trainings of different models, the team has constructed the software following the Object-Oriented Design. Since the design is carefully generated and has good separation of responsibility among different components, users can easily conduct trainings by specifying the model type, optimization method, the data format(with sentiment, word vectorization or without), and terms for return(short term, mid term or long term). A pictorial illustration is shown in Figure 4.

![Diagram](image)

**Figure 4. Software Design**
After user’s specifying the model type, data input, and terms for returns, the optimizer.py will compile a list of combinations of the hyperparameters, using the specified optimization method, suitable for the specified model type. These combinations will then form objects of network classes in network.py, and finally each combination will then be used to train models in the train_and_score function written in train.py, where the buildings of the models lie.

3.5.2 Prediction Method and Evaluation Function

In regards to the prediction method selection, the team has decided to emphasize more the direction of the stock movement instead of the extent of the movement. Therefore, an evaluation function with a combination of classification and regression result is adopted. However, a heavier weighting (0.75) is given to the classification output and the regression output, in the form of RMSE (root mean square error), takes the other (0.25). Figure 5 shows how it is calculated.

```python
def loss_direction(tar, pred):
    scl = MinMaxScaler()
    se = np.power((tar - pred), 2)
    arr = scl.fit_transform(se.reshape(-1, 1))
    normalized_rmse = np.sqrt(np.sum(arr) / tar.shape[-1])
    return (0.75 * np.sum((pred * tar) < 0) / tar.shape[-1] + 0.25 * normalized_rmse)
```

Figure 5. Loss calculation

Note that unlike the typical classification method where y labels are preprocessed as labeled classes, the team still uses the normalized numerical values for the learning process; that is, the y labels where not preprocessed into the form of 1, meaning “ups”, and 0, meaning “downs”. To integrate such outputs into the cost function, the value of the predicted outputs and the actual targets are multiplied. If the product is positive, it means the model predicts the correct direction and vice versa. The proportion of the incorrect prediction will indicate the loss function of the model.

3.5.3 Random Search for Hyperparameters

To better the training results of the models, Random Search is conducted to retrieve some better hyperparameter combinations. 80 combinations of hyperparameters are tested for every model,
and the sets are listed in Table 3. Note that the Random Search ran on the historical dataset with short term return.

<table>
<thead>
<tr>
<th>network</th>
<th>hyperparameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>RNN</td>
<td>{'batch_size': 128, 'lstm_units_1': 256, 'lstm_units_2': 0, 'droup_out_1': 0.2, 'activation': 'relu', 'optimizer': 'sgd'}</td>
</tr>
<tr>
<td>MLP</td>
<td>{'batch_size': 32, 'mlp_unit_1': 384, 'mlp_unit_2': 256, 'activation': 'tanh', 'optimizer': 'adagrad'}</td>
</tr>
<tr>
<td>SVM</td>
<td>{'gamma': 0.05, 'C': 50.0, 'kernel': 'rbf', 'coef0': 0}</td>
</tr>
<tr>
<td>RF</td>
<td>{'n_estimators': 1800, 'criterion': 'gini', 'max_depth': 50, 'min_samples_split': 5, 'min_samples_leaf': 2, 'max_features': 'sqrt', 'max_leaf_nodes': None, 'min_impurity_decrease': 0, 'bootstrap': False}</td>
</tr>
<tr>
<td>GB</td>
<td>{'learning_rate': 0.05, 'n_estimators': 1000, 'criterion': 'gini', 'max_depth': 10, 'min_samples_split': 20, 'min_samples_leaf': 4, 'max_features': 'auto', 'max_leaf_nodes': 4, 'min_impurity_decrease': 0.01}</td>
</tr>
</tbody>
</table>

Table 3. Hyperparameters for networks

4 Optimization

To achieve better prediction accuracy, the team has also implemented 2 optimization methods, Multi-Trajectory Search and Genetic Algorithm, to find the best combination of hyperparameters. Both algorithms concerns numerical optimization for large search spaces, and are discussed in details in this section.

4.1 Multi-Trajectory Search

4.1.1 Problem definition

The problem definition for Multi-Trajectory Search is written as the follows.
The input consists of a n-dimensional R space bounded by \( l = (l_1, l_2, \ldots l_n) \), and \( u = (u_1, u_2, \ldots u_n) \). The goal is to minimize \( F(X) \), which is known as the objective function, and the feasible solution is \( X = (x_1, x_2, \ldots X_n) \), where \( l_i \leq x_i \leq u_i \) and \( u_i - l_i = q \) for every \( i \). Note that \( q \) is an integer.

If one would like to find a feasible solution in the space defined above, a brute force way would be computing for every permutation, and therefore with an k-dimensional space, the total number of computation needed would be \( q^k \). However, it is not very efficient to do all the \( q^k \) computation, if the computation of a single permutation takes a long time.

Multi-Trajectory Search attempts to find the \( X \) efficiently by first computing some representative samples in the search space. The method for choosing such samples is by constructing the simulated orthogonal array.

4.1.2 Simulated Orthogonal array(SOA)

To form a set of representative samples in a space, we shall make sure the samples chosen are as uniformly distributed over the search space as possible. To denote a simulated orthogonal array constructed with n dimensions and k values within each dimension, we use \( \text{SOA}(n, k) \), where \( n \) indicates the number of the feasible solutions, and \( k \) is the number of the dimensions of the search space. To construct a Simulated Orthogonal Array, we need to satisfy the requirement that for every \( k \), every level occurs the same number of times. Note that due to this requirement, the number of rows \( n \) has to be a multiple of \( q \), and in this project, the factor is set to 1. The procedure for building a simulated orthogonal array is shown as the follows.

```plaintext
For i = 1 to k:
    q_indices_for_factor_k = random.shuffle(all_levels_in_q)
    q_indices[i] = q_indices_for_factor_k
return q_indices
```
Still, the above procedure only returns the SOA using the index of the level stored in the initial input. The figure in appendix belows shows the conversion from index to actual level for every dimension.

4.1.3 Procedure

To implement Multi-Trajectory Search in this project, as the search space is not of a large size (1500 combinations of hyperparameters in Multi-Layer Perceptron for example), the team used a simplified version, and the procedure is shown as the follows.

```plaintext
4.1.3 Procedure

global_min = infinity
For i = 1 to #predefined_iterations:
    networks = Build simulated orthogonal array SOA_{n,k}
    For j = 1 to #predefined_local_search_limit:
        //Train all the solutions in SOA
        soa_min_loss = train_network(networks)
        If j < #predefined_local_search_limit:
            If soa_min_loss < global_best:
                //local search the best network in SOA
                //networks is sorted, so index 0 is the best network
                local_search_min_loss = local_search(networks[0])
                If local_search_min_loss < global_min:
                    global_min = local_search_min_loss
```

4.2 Genetic Algorithm

Genetic Algorithm is based on Charles Darwin’s theory of natural selection (Davis, 1991[16]). This algorithm simulates the process of natural selection, where the selected fit individuals are chosen to reproduce new instances for the next generation. Interestingly, this algorithm can be used for optimizing the solution in a search space, with the combination of hyperparameters encoded as feasible solutions.
4.2.1 Problem definition

The input consists of a n-dimensional R space bounded by \( l = (l_1, l_2, \ldots, l_n) \), and \( u = (u_1, u_2, \ldots, u_n) \). The goal is to maximize the fitness function \( F(X) \), which in the context of this project would be the negation of the loss of the trained network. The feasible solution is \( X = (x_1, x_2, \ldots, x_n) \), where \( l_i \leq x_i \leq u_i \).

Similar to Multi-Trajectory Search, Genetic Algorithm attempts to find the optimal solution in a defined search space more efficiently by evolving the feasible solutions in every generation with the fitter individual (feasible solutions).

4.2.2 Procedure

Genetic Algorithm consists of 5 phases, initiation of the first population, fitness evaluation, selection, crossover, and mutation. The procedure of each phase is explained as the follows, aided by some pseudo code.

The initiation of the first population is the phase where a set of hyperparameter combinations is created, with the set size specified by the population size. Every combination is generated by random sampling..

```python
population = []
For i = 1 to #in_population:
    combination = []
    For j = 1 to k:
        combination.append(random.choice(possible_values_for_factor_j))
    population.append(combination)
```
After conducting trainings with each combination, the fitness evaluation is conducted, where we compared the fitness score of the combinations. In our implementation, the fitness score is the negation of the loss value, which is run by the evaluation function shown in section 3.5.2.

In the selection phase, the combinations with higher fitness score are retained in the population, and the rest are removed. The number of combinations to retain is specified by the retain ratio, defined by the user. A generation is started when the crossover phase is run, where new combinations of hyperparameters are generated based on 2 other combinations passed down from the previous generation. For every newly generated combination, known as the child, the value of every factor will be inherited from either 2 of the combinations, known as the parents, that it bases on.

Finally, in the mutation phase, some hyperparameters in the newly created combinations will be randomly changed to other values. The probability of the mutation occurrence is also specified by the user, defined as the mutation ratio.

The whole process will repeat for the specified number of generations.

```python
retain_length = int(#in_population * retain ratio)
parents = population[:retain_length]
children_length = #in_population - retain_length
children = []

While len(children) < children_length:
    combination_1 = random_pick(parents)
    combination_2 = random_pick(parents)

If combination_1 != combination_2:
    // crossover: create new combination
    new_params = []
    For i = 1 to k:
        new_params.append(random.choice(combination_1[i], combination_2[i]))
    If mutate_chance > random():
        d = random(0, k)
        new_params[index_of_param_to_mutate] = random.choice(params_in_dimen_d)
```
4.3 Integration of Simulated Orthogonal Array into Genetic Algorithm

As discussed in section 4.2, the implemented Genetic Algorithm initialized the first population by random sampling. Using this method could possibly result in all the feasible solutions clustering at some points in the search space, and therefore more generations of evolution will be required before the combination with the minimum loss is found. To tackle this problem, the team has integrated the Simulated Orthogonal Array into the Genetic Algorithm, replacing the original random generation method. The results are shown in section 5.

5 Results and Analysis

This section details the results of the experiments that the team has run, as well as the analysis of the findings. First, the experiments on the optimization methods will be discussed, to verify the optimization methods are feasible for searching hyperparameter combinations that yield lower losses in the training process. Subsequently, the result of the integration of simulated Orthogonal Array into Genetic Algorithm will be discussed, followed by the adoption of different data formats for training will be discussed. Lastly, the robustness of the prediction models will be examined by looking at their prediction accuracy during some certain time intervals, including bad time, average time, and good time of the stock.
5.1 Optimization results

Experiments on the optimization methods were run on all the combination of the machine learning models, data input format and different terms for returns, and the Multi-trajectory Search and Genetic Algorithm results on historical price return in short term are shown in Figure 6 and Figure 7.

Figure 6. MTS on RNN, historical price, short term return

Figure 7. GA on RNN, historical price, short term return
Although Multi-Trajectory Search and Genetic Algorithm has different way for defining the set of sample before its evolution, generation in Genetic Algorithm and iteration in Multi-Trajectory Search have both shown downward trends. In order to validate the effectiveness of adopting such optimization methods, benchmarkings are then to be performed.

To verify the performance of the Multi-Trajectory Search and Genetic Algorithm, benchmarkings marking of them against Random Search were performed. 80 combinations were randomly searched from the Recurrent Neural Network search space and trainings on these combinations were performed. This benchmarking was run on historical dataset on short term return. The process was repeated for 10 times, and the average of the minimum losses are listed as the follows.

<table>
<thead>
<tr>
<th></th>
<th>Random Search</th>
<th>Multi-Trajectory Search</th>
<th>Genetic Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loss</td>
<td>0.336066</td>
<td>0.319672131147541</td>
<td>0.319672</td>
</tr>
</tbody>
</table>

Table 4. Comparison of Optimizations and Random Search

As shown in the figure, both optimization methods did outperform the Random Search. With this result, both optimization models will be used to optimize the machine learning models, for searching the best combination of parameters.

### 5.2 Comparison of Optimization methods Performance

In terms of the optimization methods performance, no significant differences in the losses are observed, meaning that in many cases, both optimizations simply found the same set of the hyperparameters at the end of the operation. Table 5 and Table 6 show the comparison between Multi-Trajectory Search and Genetic Algorithm on different terms of returns, on the RNN, and the SVM model.
<table>
<thead>
<tr>
<th></th>
<th>Genetic Algorithm</th>
<th>Multi-Trajectory Search</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RNN</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>short term return</td>
<td>0.319672</td>
<td>0.319672131147541</td>
</tr>
<tr>
<td>mid term return</td>
<td>0.163934</td>
<td>0.16393442622950818</td>
</tr>
<tr>
<td>long term return</td>
<td>0.073770</td>
<td>0.08196721311475409</td>
</tr>
</tbody>
</table>

Table 5. Comparison between GA and MTS on RNN

<table>
<thead>
<tr>
<th></th>
<th>Genetic Algorithm</th>
<th>Multi-Trajectory Search</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SVM</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>short term return</td>
<td>0.267206</td>
<td>0.2914979</td>
</tr>
<tr>
<td>mid term return</td>
<td>0.388664</td>
<td>0.380566</td>
</tr>
<tr>
<td>long term return</td>
<td>0.336032</td>
<td>0.3360323</td>
</tr>
</tbody>
</table>

Table 6. Comparison between GA and MTS on SVM

With the above results shown, it can be concluded that both optimization methods can be used to find hyperparameter sets that could construct models with higher accuracy given a fixed data set.

5.3 Integration of Simulated Orthogonal Array into Genetic Algorithm

During the course of running the optimizations, the team has also noticed that Multi-Trajectory Search usually finds the the minimum loss in earlier phases compared to Genetic Algorithm. This result can be attributed to the usage of the Simulated Orthogonal Array, as the selected experimental samples are more distributed in the search space, and therefore, more representative. With this knowledge, the team has tried to replace the random population creation process with the simulated orthogonal creation method, and the results are discussed in the following.
To determine whether the adoption of using Simulated Orthogonal Array will result in the discovery of minimum loss in earlier generation, the originally implemented Genetic Algorithm and the implementation with Simulated Orthogonal Array, were run 12 times on the Recurrent Neural Network with the population size equals to 10 and 10 generations, on the same dataset. The result is shown in Table 7.

<table>
<thead>
<tr>
<th>run</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>GA with random population creation</td>
<td>7</td>
<td>4</td>
<td>10</td>
<td>4</td>
<td>2</td>
<td>6</td>
<td>10</td>
<td>4</td>
<td>9</td>
<td>1</td>
<td>6</td>
<td>7</td>
<td>5.83</td>
</tr>
<tr>
<td>GA with SOA</td>
<td>7</td>
<td>6</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>9</td>
<td>7</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>4.08</td>
</tr>
</tbody>
</table>

Table 7: generations required to find minimum loss

With the averaged result, it showed that the original Genetic Algorithm generally found the minimum cost in the 5th generation; whereas the Genetic Algorithm using the Simulated Orthogonal Array found the minimum loss in the 4th generation. Thus, it can be concluded that the inclusion of Simulated Orthogonal into Genetic Algorithm can reduce the number of generations run before the hyperparameter set with the minimum loss is found.
5.4 Model performances

With the optimized prediction model, we then select some models to for running data input format. The performance of the models is shown in Figure 8.

As with the results shown from the literature, Recurrent Neural Network, Multi-Layer Perceptron, and Support Vector Machine are indeed the ones that have better performance.

Note that although only 3 models were selected for further inspection, in fact, all the models did perform above the benchmark. Our benchmark is defined as the larger of the accuracy under the situation where the model would only guess “up” for all the prediction, or “down”. The benchmark is shown in Table 9.
<table>
<thead>
<tr>
<th>Term</th>
<th>Positive Sample Portion</th>
<th>Negative Sample Portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>48.91%</td>
<td>51.09%</td>
</tr>
<tr>
<td>Mid</td>
<td>51.34%</td>
<td>48.66%</td>
</tr>
<tr>
<td>Long</td>
<td>49.96%</td>
<td>50.04%</td>
</tr>
</tbody>
</table>

Table 8. Benchmark on short, mid, and long term return

As the Figure 9 shows, when looking at the short term return, if the model simply guesses “up” for all prediction, it will achieve 48.91% accuracy, meaning the loss will be 0.5109. Similarly, if it guesses “down”, the accuracy will be 51%, meaning the loss will be 0.4891. The loss of the models is shown in Table 8, and it can be verified that indeed all the models outperformed the benchmark.

<table>
<thead>
<tr>
<th>Model</th>
<th>Minimum Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>RNN</td>
<td>0.319672</td>
</tr>
<tr>
<td>MLP</td>
<td>0.32377</td>
</tr>
<tr>
<td>SVM</td>
<td>0.267206</td>
</tr>
<tr>
<td>Random Forest</td>
<td>0.425101</td>
</tr>
<tr>
<td>Gradient Boost</td>
<td>0.352227</td>
</tr>
</tbody>
</table>

Table 9. minimum loss of the models, with GA optimization

5.5 Data format performances

As stated in section 3.3, in this project, the team trained models with different sources of the inputs, including historical, sentiment, w2v_headline and w2v_content, and we are interested in seeing which input source will reflect the stock movement better, meaning yielding prediction with lower loss.
Sentiment Performance Evaluation

To evaluate the performance of the sentiment dataset, benchmarkings were run on all the models against the historical data on all the terms for return. All models yielded consistent result on short term return, and the results on Support Vector Machine, Recurrent Neural Network, and Multi-Layer Perceptron are shown below.

As seen in the figures above, when running on short term term, the Sentiment dataset has consistently produced lower loss than the Historical dataset, and therefore it is clear that on short term returns, the inclusion of the headline sentiment will streamline the prediction result. Still, when running on mid and long term, no significant patterns are observed.
w2v_headline & w2v_content Performance Evaluation

To perform benchmarking for w2v_headline and w2v_content, the dimensions of the dataset, including historical, sentiment, and w2v_headline, were expanded into 51 feature columns after the splitting X and the target y. The expanded columns were filled with random values from -1 to 1, before being standardized. Figure 9 shows the results of the 4 input sources using the Recurrent Neural Network running with the Genetic Algorithm, on short term return.

![Figure 9](image)

Figure 10. RNN on different data inputs, on short term return

As per this figure, we can again verify the incorporation of sentiment score of the news will yield lower loss during evaluation. Furthermore, w2v_headline and w2v_content outperformed historical by a 12% and 15% margin, respectively. Compared to sentiment, both have also outperformed it by a 7% and 13% margin. The result suggested that the adoption of the significant subject words did outperform the simple usage of sentiment of the whole article. From the result, it is also observed that w2v_content performed slightly better than w2v_headline, suggesting that the use of the sentiment of the subject words did give more accurate sentiment.
To further verify this result, the same data was run on the Multi-Layer Perceptron network, and the results is shown below.

![Graph showing different data inputs for short term return](image)

Figure 11. MLP on different data inputs, on short term return

As can been seen from the figure, similar result was drawn on MLP. However, when running with mid term return as and long term return label as target, sentiment, w2v_headline and w2v_content, yielded worse results than simply running with price, as can be verified by the Recurrent Neural Network. The results are shown below.
Figure 12. RNN on different data inputs, on mid term return

Figure 13. RNN on different data inputs, on long term return
The figure below indicates the difference between c2v+_content and historical (the benchmark), on short, mid and long terms is shown below.

![Figure 14. c2v+_content benchmarking](image)

As we can see, only short term c2v+_content is lower than its benchmark. Therefore, we can conclude that the inclusion of textual related data, including sentiment, w2v_headline, and w2v_content, only works for prediction with short term return. The result also aligned with the intuition that the stock price shall reflect the immediate release of the news data.

Interestingly, we also note that in the experiment results, the evaluation losses were significantly lower when using historical and sentiment on mid and long term return than on short term return, and the reason is yet to be investigated.

### 5.6 Robustness Checking

In this section, the 3 models which yielded the minimum loss, Recurrent Neural Network, Multi-Layer Perceptron and Support Vector Machine, are selected for the robustness check throughout different time intervals. The selection of the time intervals includes the dates ranging
from Oct.6, 2014 to Dec.4, 2014, regarded as bad times; Nov.28, 2017 to Nov.20, 2018 as average times, and Jan.20, 2016 to Dec.4, 2017 as the good times. The movement of IYE index is also shown in the graph below.

Figure 15. IYE Index Adj Closing Price

The original test set (dates ranging from Apr.6, 2018 to Apr.1, 2019) was used as benchmark to evaluate performance. The results are shown below.
As seen in the these figures, when running on short term return, c2v+ content and Sentiment datasets have consistently yielded lower loss compared to Historical, suggesting the trained SVM model is robust enough for extreme time period prediction. However, when looking at results on mid and long term, no obvious trend can be observed. Similar results were drawn from the experiments on the Recurrent Neural Network.
6 Conclusion

In this project, the team has built an application that predicts stock movement using different combinations of data inputs, including historical stock price, sentiment score, and word vectorization. The results on the running on Sentiment dataset produced lower losses than results on running on Historical price data. Results have also shown that by incorporating the c2v+_content dataset, the accuracy of the prediction increased 15% compared to the simple usage of historical stock price on short term return. Work on optimizing the hyperparameters of the prediction models was also implemented by using Multi-Trajectory Search and Genetic Algorithm. With the optimized model, the robustness was verified by looking at different time intervals, including intervals in bull and bear markets. One suggested future work would be to investigate the reasons resulting difference in losses when running stock returns on short term, mid term and long term.
Reference


Appendix

A: POS Tag List

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC</td>
<td>Coordinating conjunction</td>
</tr>
<tr>
<td>CD</td>
<td>Cardinal number</td>
</tr>
<tr>
<td>DT</td>
<td>Determiner</td>
</tr>
<tr>
<td>EX</td>
<td>Existential there</td>
</tr>
<tr>
<td>FW</td>
<td>Foreign word</td>
</tr>
<tr>
<td>IN</td>
<td>Preposition or subordinating conjunction</td>
</tr>
<tr>
<td>JJ</td>
<td>Adjective</td>
</tr>
<tr>
<td>JJR</td>
<td>Adjective, comparative</td>
</tr>
<tr>
<td>JJS</td>
<td>Adjective, superlative</td>
</tr>
<tr>
<td>LS</td>
<td>List item marker</td>
</tr>
<tr>
<td>MD</td>
<td>Modal</td>
</tr>
<tr>
<td>NN</td>
<td>Noun, singular or mass</td>
</tr>
<tr>
<td>NNS</td>
<td>Noun, plural</td>
</tr>
<tr>
<td>NNP</td>
<td>Proper noun, singular</td>
</tr>
<tr>
<td>NNPNS</td>
<td>Proper noun, plural</td>
</tr>
<tr>
<td>PDT</td>
<td>Predeterminer</td>
</tr>
<tr>
<td>POS</td>
<td>Possessive ending</td>
</tr>
<tr>
<td>PRP</td>
<td>Personal pronoun</td>
</tr>
<tr>
<td>PRPS</td>
<td>Possessive pronoun</td>
</tr>
<tr>
<td>RB</td>
<td>Adverb</td>
</tr>
<tr>
<td>RBR</td>
<td>Adverb, comparative</td>
</tr>
<tr>
<td>RBS</td>
<td>Adverb, superlative</td>
</tr>
<tr>
<td>RP</td>
<td>Particle</td>
</tr>
<tr>
<td>SYM</td>
<td>Symbol</td>
</tr>
<tr>
<td>TO</td>
<td>to</td>
</tr>
<tr>
<td>UH</td>
<td>Interjection</td>
</tr>
<tr>
<td>VB</td>
<td>Verb, base form</td>
</tr>
<tr>
<td>VBD</td>
<td>Verb, past tense</td>
</tr>
<tr>
<td>VBG</td>
<td>Verb, gerund or present</td>
</tr>
<tr>
<td>VBZ</td>
<td>Verb, 3rd person singular</td>
</tr>
<tr>
<td>VBN</td>
<td>Verb, past participle</td>
</tr>
<tr>
<td>VBR</td>
<td>Verb, non-3rd person singular</td>
</tr>
<tr>
<td>WDT</td>
<td>Wh-determiner</td>
</tr>
<tr>
<td>WP</td>
<td>Wh-pronoun</td>
</tr>
<tr>
<td>WP$</td>
<td>Possessive wh-pronoun</td>
</tr>
<tr>
<td>WRB</td>
<td>Wh-adverb</td>
</tr>
</tbody>
</table>

B: SOA Creation Code

```python
def create_SOA(self):
    # Create orthogonal array
    popu = []
    permu = {}  
    for k in self.nn_param_choices.keys():
        l = list(range(self.arg_length))
        random.shuffle(l)
        permu[k] = l
        init_arr = []
        for i in range(self.arg_length):
            indi_dcit = {}
            for k, v in self.nn_param_choices.items():
                inx = permu[k][i]
                indi_dcit[k] = v[inx]
            indi_dcit.update(self.fixed_param)
            hashed = self.get_dict_hash(indi_dcit)
            if hashed not in self.visited:
                self.visited.add(hashed)
                network = Network(self.nn_param_choices)
                network.create_set(indi_dcit)
                popu.append(network)
    return popu
```