Frontiers in Business, Economics and Management
ISSN: 2766-824X | Vol. 17, No. 3, 2024

The Combination of Traditional Asset Pricing Models and Machine Learning

-- Starting from Multiple Linear Regression Models and Recurrent Neural Networks (RNNs)

Yanqing Ma

School of Finance, City University of Macau, Macau, 999078, China
F21090102549@cityu.edu.mo

Abstract: This article combines the CAPM theory with the concept of beta coefficient, proposes a multiple linear regression model and recurrent neural network (RNN), and predicts the stock of representative Apple company in SP500. The multiple linear models introduce the concept of CAPM (beta) and is based on multiple linear regression. It determines the values of these coefficients by minimizing the error between actual and predicted values. The LinearRegression class in the sklearn library is used to train and predict data related to Apple Inc. Recurrent neural network (RNN) is used to predict the stock price of Apple Inc. and combines the beta coefficient calculated from market index data to enhance the performance of the model. The results show that the predicted values are very close to the actual values. In addition, this article also compared and demonstrated the different prediction results of multiple linear regression models and recurrent neural networks (RNN) on whether to introduce CAPM related concepts (beta). The results showed that the citation of related CAPM concepts is very necessary, and it is particularly strong in recurrent neural network (RNN) models. Afterwards, this article diverged from this result and further demonstrated from both positive and negative perspectives whether more parameters would bring better predictive results to the model.

Keywords: CAPM, Stock price prediction, Multiple linear regression model, Structural Information of Recurrent Neural Network (RNN) Model.

1. Introduction

At present, traditional asset pricing models (CAPM, three factor, etc.) are widely used in finance and other fields, but they also have significant limitations, leading to biased prediction results. Traditional asset pricing theory assumes that investors are completely rational, while psychological research shows that people often have various "irrational" emotions when making decisions, especially when making decisions under complex and uncertain conditions [1]. The traditional asset pricing model assumes that all participating investors have equal information, which is almost impossible in practice. Investors with more and faster information clearly have a greater advantage. The traditional asset pricing model (e.g. CAPM, FF) only introduces a few factors and has a limited scope of consideration. The traditional asset pricing model is built on a static basis: that is, investors' preferences and risk attitudes remain unchanged in a short period of time. But as the market changes, the above may change over time. The fluctuation of stock prices affects the hearts of countless investors, and accurately predicting stock prices has always been a major challenge in the financial sector. In today's complex and ever-changing financial markets, traditional CAPM theory and emerging machine learning technologies have brought new hope for stock price prediction. In this article, we mainly selected two models: multiple linear regression model and recurrent neural network (RNN), and additionally supplemented the relevant concepts of CAPM (beta) to make the models more complete.

After establishing the model, divide the test set and training set, and finally obtain the final result and evaluate the two models. In addition, this article also compared the different results of CAPM related concepts (beta) in the predictions of the two models. Similarly, line charts, MAE, MSE, RMSE, and R2 reference indicators are used.

2. Theory

2.1. CAPM Basic Theoretical Model

According to the traditional CAPM model, the expected return on assets is directly proportional to the beta coefficient, it implies that there is a linear relationship between assets expected return and its quantity of market risk. The quantity of market risk is measured with the so called market beta (β) [2], which c2 measures how much of the returns are on a given asset move together (co movements) with the market. Fama (1968) provides the well-known beta form of the core CAPM: E (Ri)=Rf+β i (E (Rm) - Rf), where E (Ri) is the expected asset return rate. Rf is the risk-free rate, E (Rm) is the expected return rate of the market portfolio, and β i is the risk coefficient. In this article, although not entirely a standard CAPM implementation, concepts related to CAPM are introduced. The CAPM theory holds that the expected return on an asset is the risk-free rate plus the risk premium of the asset relative to the market portfolio. In this article, we attempt to capture the risk characteristics of Apple's stock relative to the market by calculating the beta coefficient of its return on SPY (usually considered a representative of the market portfolio). The beta coefficient measures the systematic risk of an asset, i.e. the sensitivity between asset returns and market returns.

If the beta coefficient is greater than 1, it indicates that the risk of the asset is higher than the market average risk; If the beta coefficient is less than 1, it indicates that the risk of the asset is lower than the market average risk.

2.2. Recurrent Neural Network (RNN)

Recurrent neural network is a type of directed recurrent neural network composed of units connected in a chain. Through the connections of each unit, recursive neural networks can store the relationship between the input of neurons at the current moment and the output of neurons at the previous moment, which is superior to other types of neural networks in processing time series data [3].

Recurrent neural network is a type of neural network specifically designed for processing sequential data. Its characteristic is that there is a hidden state at each time step, which receives the input of the current time step and the hidden state of the previous time step as inputs, thus capturing the temporal dependencies in the sequence.

2.2.1. Neuronal structure

The basic units of RNN are like neurons in traditional neural networks, but with additional self connections that allow it to retain memory of past information. This self connection allows information to be transmitted over time, enabling the network to handle temporal dependencies in sequential data.

Each neuron receives the input of the current time step and the hidden state of the previous time step as inputs and generates the output of the current time step and the updated hidden state through an activation function.

2.2.2. Time unfolding

To process sequential data, RNN can be unfolded over time, treating it as a sequence composed of multiple identical neurons, with each time step corresponding to a neuron. In this way, RNN can process each element in the sequence and pass information from one time step to the next.

By unfolding over time, RNNs can learn long-term dependencies in a sequence, namely the impact of past inputs on current and future outputs.

In this article, the SimpleRNN layer is used, which is a basic RNN layer. The structure of the model consists of two SimpleRNN layers, with two Dropout layers in between to prevent overfitting, and finally a fully connected Dense layer to output the predicted values. Stock price data is a typical time series data with temporal dependence. For example, the stock prices of the past few days may have an impact on future prices. RNN can utilize this temporal dependency to predict future prices by learning patterns from historical stock price data.

3. Empirical Research and Result Analysis

3.1. Data Preprocessing

The data selected in this article comes from the S&P 500 index and Apple Inc. data from December 2009 to January 2019. Firstly, the data was randomly sampled for authenticity testing, and it was found to be completely consistent with Yahoo Finance. After processing, it includes Date, Open, High, Low, Close, Adj Close, and Vo2ume as indicators. The data volume for both the S&P 500 index and Apple Inc. is 2282 rows * 7 columns

3.2. Multiple Linear Regression Model

Use pandas' read_csv function to read CSV data files from Apple Inc. (AAPL) and Standard Poor's 500 (SPY) separately and store the data in a DataFrame object. Extract independent variables from AAPL data, including opening price, highest price, lowest price, and trading volume, as well as the dependent variable closing price. And calculate the daily returns of AAPL and SPY by calculating the percentage change in closing price. Then, delete rows containing null values to ensure data integrity.

Use the statsmodels library to calculate the beta coefficient of AAPL return on SPY return. Firstly, a constant term is added to the return data of SPY, and then ordinary least squares (OLS) is used for regression fitting. The second value in the obtained model parameters is the beta coefficient.

Use StandardScaler to standardize the independent variable data of AAPL, so that it has a distribution with a mean of 0 and a standard deviation of 1. This helps to improve the performance of linear regression models and avoid certain features having a significant impact on the model due to large numerical ranges

Afterwards, divide the AAPL data into a training set and a testing set. The data from the previous (total number of data - 300) days is used as the training set for training the linear regression model; The data from the last 300 days will be used as the test set to evaluate the performance of the model.

Create a linear regression model object and train it using the training set data. By minimizing the error between actual and predicted values, adjusting the parameters of the model, the optimal linear relationship between the independent and dependent variables is established.

Use a trained linear regression model to predict the test set data and obtain the predicted closing price

Calculate the evaluation metrics of the linear regression model on the test set, including mean square error (MSE), root mean square error (RMSE), mean absolute error (MAE), and coefficient of determination (R^2). These indicators are used to measure the predictive accuracy and fitting degree of the model.

Use matplotlib to draw a line comparison chart between the actual closing price and the predicted closing price, visually demonstrating the predictive performance of the model.

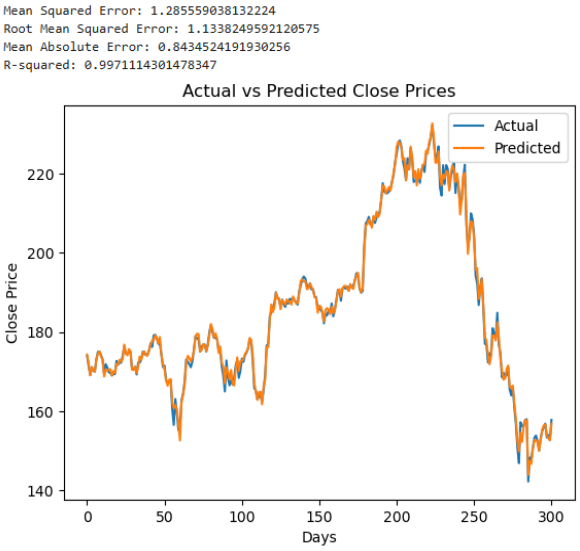

Figure 1. Test results of multiple linear regression model

The multiple linear regression equation is expressed as: y=99.8291+-27.4244 * Open+39.1100 * High+35.1811 * Low+0.0155 * Volume. The coefficients and intercepts in the equation are rounded to four decimal places, and the variable names are derived from the feature names of the original data.

3.3. Recurrent Neural Network (RNN)

Use pandas' read_csv function to read Apple Inc. stock data files and market index data files, in preparation for subsequent data processing and analysis.

Divide training and testing sets: Divide the training and testing sets from the stock data of Apple Inc. The training set contains data from earlier times, while the test set consists of data from 300 days later. This partitioning method helps evaluate the model's generalization ability on unseen data.

Calculate return and beta coefficient: First, calculate the return of Apple Inc. stock and market index, use the pct_change function to calculate the percentage change, and then use dropna to remove rows with missing values. Next, by performing linear regression on market returns and Apple stock returns, the beta coefficient is calculated. After reshaping the market return data into a shape suitable for linear regression, use LinearRegression for fitting, and finally obtain the beta coefficient from the fitting result. The beta coefficient reflects the risk characteristics of stocks relative to the market and is used as an additional feature in subsequent models.

Data normalization: Use MinMaxScaler to normalize the opening price data of the training and testing sets, mapping the data to a range of 0 to 1. Normalization can improve the training effectiveness and stability of the model, making data with different features have the same scale.

Building training and testing set data: Initialize a list used to store input features and labels for the training set and input features and labels for the testing set. Then, build the training and testing sets of data through a loop. For the training set, add the opening price and beta coefficient of 60 consecutive days as input features to the x_train list, and add the corresponding opening price of the 61st day as a label to the y_train list. Similarly, perform similar operations on the test set.

Random shuffling and array transformation: Set a random seed to ensure that the random shuffling of data is repeatable, and then randomly shuffle x_train and y_train separately. Finally, convert the list of training sets into a numpy array for subsequent processing and input into the model.

Data Reshaping: First, print the shape of the training set data before reshaping. Then, reshape the training set data into an intermediate shape and add a dimension for subsequent processing. Next, according to the input requirements of the model, further adjust the shape and transform the data into the form of [number of samples, time steps, number of features]. Perform a similar reshaping operation on the test set.

Build and Compile Model: Build a sequential model that includes two SimpleRNN layers, two Dropout layers, and a fully connected Dense layer. The first SimpleRNN layer has 80 hidden units and returns the output for each time step, while the second SimpleRNN layer has 100 hidden units. Dropout layer is used to prevent overfitting. Compile the model using Adam optimizer and mean square error loss function.

Model saving and callback functions: Set the path for saving the model. If there is already a saved model, load the model weights. Define a callback function to save the best model weights during training and monitor them based on the validation set loss.

Model training: Train the model using the training and testing sets, set the batch size to 64, train for 50 cycles, validate on the testing set at the end of each cycle, and save the model weights using callback functions.

Model summary and parameter saving: Print the structure and parameter information of the model. Save the trainable parameters of the model to a file for subsequent analysis or reloading.

Draw loss curves: Draw loss curves for the training and validation sets to observe the training process of the model. By comparing the training loss and validation loss, the overfitting and generalization ability of the model can be evaluated.

Prediction and evaluation: Use a trained model to predict the test set, and then perform inverse normalization between the predicted results and the actual results. Calculate and print the mean square error, root mean square error, mean absolute error, and coefficient of determination to evaluate the predictive performance of the model. Draw a comparison curve between real stock prices and predicted stock prices to visually demonstrate the predictive performance of the model. These evaluation indicators can help measure the accuracy and reliability of the model.

Layer (type)	Output Shape	Param #
simple_rnn (SimpleRNN)	(None, 60, 80)	6560
dropout (Dropout)	(None, 60, 80)	0
simple_rnn_1 (SimpleRNN)	(None, 100)	18100
dropout_1 (Dropout)	(None, 100)	0
dense (Dense)	(None, 1)	101

Total params: 24,761
Trainable params: 24,761
Non-trainable params: 0

Figure 2. Structural Information of Recurrent Neural Network (RNN) Model

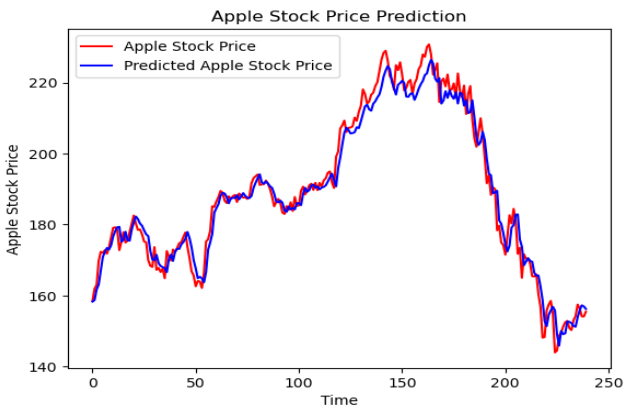

Figure 3. Line graph of Recurrent Neural Network (RNN) model

Mean Squared Error: 14.901471
Root Mean Squared Error: 3.860242
Mean Absolute Error: 3.003388
R-squared: 0.969658

Figure 4. Evaluation parameters of Recurrent Neural Network (RNN)

3.4. Evaluation Results and Summary

At this point, the two basic models have been constructed, and from the line chart, there is not a significant difference between the two models. The predicted and actual lines of both are relatively close, indicating good prediction results. The specific parameters are as shown above:

Multiple linear regression model:

Mean Squared Error: 1.285559038132224

Root Mean Squared Error: 1.1338249592120575
Mean Absolute Error: 0.8434524191930256
R-squared: 0.9971114301478347
Recurrent Neural Network (RNN):
Mean Squared Error: 14.901471
Root Mean Squared Error: 3.860242
Mean Absolute Error: 3.003388
R-squared: 0.969658

In terms of specific indicators, at least in the last 300 days of data from December 31, 2009 to January 25, 2019 for Apple Inc. in the SP500 prediction, the multiple linear regression model has the advantage, with smaller MSE, RMSE, and MAE, and R-squared leaning more towards 1. As for when the scenario is more suitable for these two models, investors can adjust it according to the actual situation or choose the best one after experimentation.

3.5. The necessity of Theoretical Participation in CAPM Model

Afterwards, we need to demonstrate the importance of the CAPM model, where the beta coefficient is involved in model construction, by comparing the current results with those without the participation of the beta coefficient and evaluating them using the same method. The specific steps, except for adding beta related code, are basically the same as those mentioned earlier. Therefore, we will not go into too much detail here and will directly provide the final results (the figures show the results of beta value participation and nonparticipation):

3.5.1. Multiple linear regression model

Beta participation

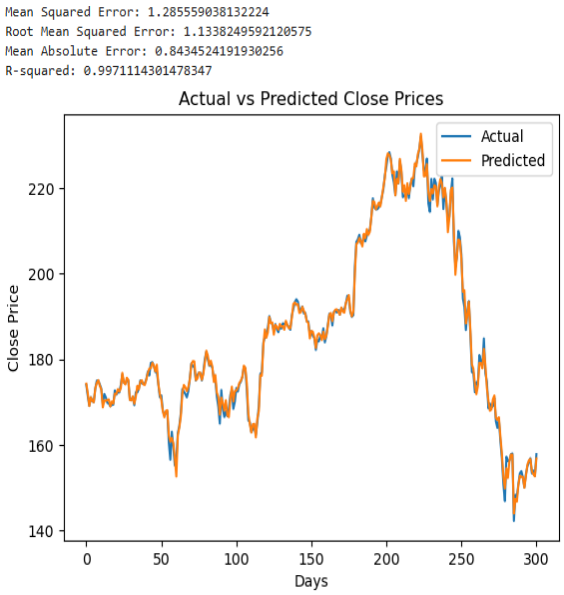

Figure 5. Line graph of multiple linear regression model (Beta participation)

Beta does not participate

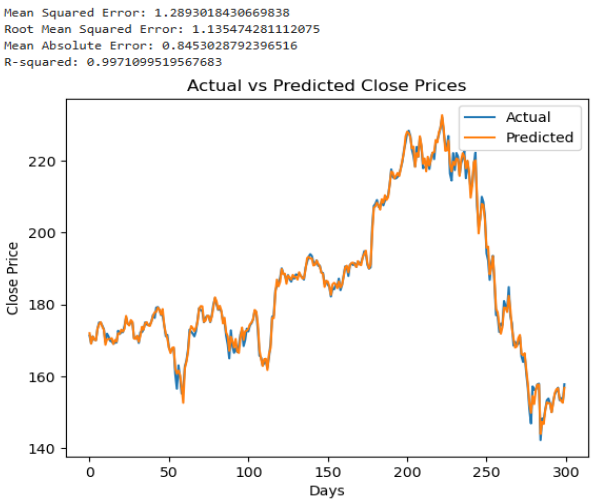

Figure 6. Line graph of multiple linear regression model (Beta not included)

3.5.2. Recurrent Neural Network (RNN)

Beta participation

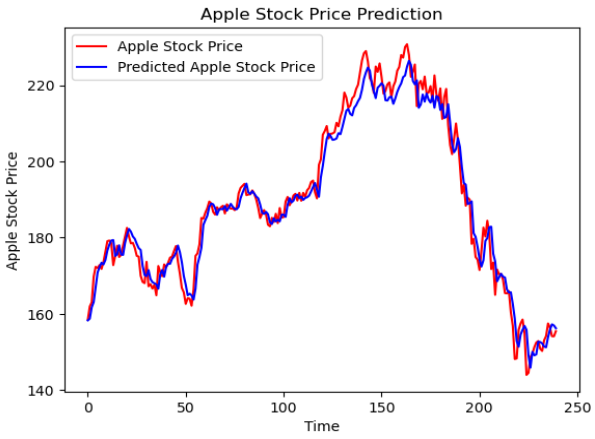

Figure 7. Line graph of Recurrent Neural Network (RNN) model (Beta participation)

```
Mean Squared Error: 14.901471
Root Mean Squared Error: 3.860242
Mean Absolute Error: 3.003388
R-squared: 0.969658
```

Figure 8. Evaluation parameters of Recurrent Neural Network (RNN) (Beta participation)

Beta does not participate

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn (SimpleRNN)       (None, 60, 80)            6560

dropout (Dropout)            (None, 60, 80)            0

simple_rnn_1 (SimpleRNN)     (None, 100)               18100

dropout_1 (Dropout)          (None, 100)               0

dense (Dense)                (None, 1)                 101

=================================================================
Total params: 24,761
Trainable params: 24,761
Non-trainable params: 0
_________________________________________________________________
```

Figure 9. Structural information of recurrent neural network (RNN) model (Beta does not participate)

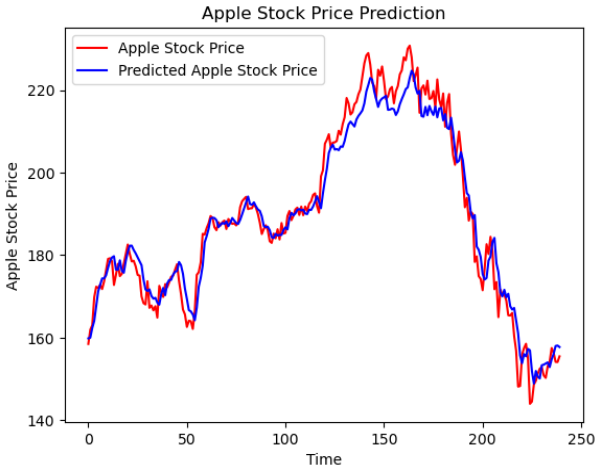

Figure 10. Line graph of Recurrent Neural Network (RNN) model (Beta not involved)

```
Mean Squared Error: 19.439925
Root Mean Squared Error: 4.409073
Mean Absolute Error: 3.402995
```

Figure 11. Evaluation parameters of Recurrent Neural Network (RNN) (Beta does not participate)

Through the longitudinal comparison of the two models themselves, observing the indicators and charts, both models have improved to varying degrees after adding the beta indicator. This is sufficient to demonstrate the necessity and importance of incorporating the CAPM theoretical foundation into the model.

3.6. Sensitivity Analysis of Recurrent Neural Networks (Rnns) and Multiple Linear Regression Models

Besides, there is another place worth paying attention to. That is to say, the improvement of recurrent neural networks (RNNs) is much greater than that of multiple linear regression models. This can be seen from the MSE: the MSE of recurrent neural networks (RNNs) decreased from about 19.44 to 14.90, a decrease of about 4.54 points, while the MSE of multiple linear regression models decreased from about 1.2893 to about 1.2855, which is far less significant than that of recurrent neural networks (RNNs). Other parameters are also the same, and readers can test them themselves.

As for the reasons, this article believes that there may be the following factors:

3.6.1. Model expression ability

The complexity of neural networks:

Neural networks, especially recurrent neural networks (RNNs), have powerful expressive power. They can learn complex nonlinear relationships in data, automatically extract features, and perform advanced pattern recognition.

In contrast, multiple linear regression is a relatively simple model that can only capture linear relationships. For complex time series data such as stock prices, there may be a large number of nonlinear relationships, and neural networks are more capable of capturing these complex patterns, thereby improving prediction accuracy.

Adaptive feature learning:

Neural networks automatically adjust weights and biases through backpropagation algorithms to minimize the loss function. During the learning process, it can adaptively learn which features are important for prediction, including the complex interactions between beta coefficients and other input features.

However, multiple linear regression models have relatively fixed processing of features, usually assuming a linear relationship between features and target variables, and limited modeling ability for the interaction between features.

3.6.2. Ability to process time series data

Memory and sequence dependent learning:

RNN is specifically designed for processing time series data and has a memory mechanism that can remember past information and use it for current predictions. This is important for stock price forecasting because stock price movements are usually influenced by factors and trends over a longer time horizon[4]. For example, the price trends of the past few days may have a significant impact on future prices. Neural networks can learn long-term dependencies in time series, while multiple linear regression models are relatively weaker in handling this characteristic of time series data.

Dynamic modeling:

Neural networks can continuously update their internal states and weights with the input of new data, thus being able to adapt to the dynamic changes in data. In the stock market, prices are influenced by various factors, and market conditions are constantly changing. Neural networks can better adapt to this dynamism.

Multiple linear regression models typically assume that data relationships are static and not easily adaptable to dynamic changes in data.

3.6.3. The utilization of beta coefficient

Nonlinear Fusion:

In neural networks, beta coefficients can be fused with other input features in a non-linear manner. Neural networks can learn the complex nonlinear relationship between beta coefficients and stock prices, thereby better utilizing this additional information to improve prediction accuracy.

In multiple linear regression, the beta coefficient is usually only added as a linear term to the model, and its contribution to prediction is relatively limited.

Feature extraction and abstraction:

Neural networks can gradually extract and abstract input features through multi-layer structures, transforming raw data into higher-level representations. The beta coefficient can be integrated with other features into more meaningful feature representations during this process, providing stronger support for prediction.

The multiple linear regression model lacks the ability to extract and abstract features and can only directly use the original features for linear combination to make predictions.

4. Summary

So, in summary, we can reasonably infer that if more coefficient parameters are given to the recurrent neural network (RNN), the model can have more parameters to adapt to the complexity of the data. If these coefficients can effectively capture key features and patterns in the data, the model may better fit the training data, thereby improving prediction accuracy to a certain extent, and even surpassing the prediction accuracy of multiple linear regression in this article. In addition, more coefficients mean that the model has higher flexibility and can better adapt to different data distributions and changes. This is particularly important for handling complex time series data, as time series data such as stock prices often have high nonlinearity and dynamism.

Of course, apart from favorable factors, there are also some unknown factors. Adding too many coefficients can make the model more complex, thereby increasing the risk of overfitting. At the same time, it will also reduce the interpretability of the model, increase computational costs, and other unknown risks. Complex models may be difficult to explain how their predicted results are generated, making it difficult for people to trust and apply these models. Being too complex is also not conducive to widespread application and promotion.

In summary, providing more coefficients to the recurrent neural network does not necessarily lead to better prediction results. In practical applications, it is necessary to carefully select and adjust the number of coefficients in the model, and determine the most suitable model structure and parameter settings through experiments and evaluations to balance the accuracy, generalization ability, and computational cost of the model.

References

[1] Chen Z Y, & Li C Q. (2017). Media reporting and asset pricing: A review. Foreign Economics and Management, 39(3), 16.

[2] Melissa Vergara-Fernández, Conrad Heilmann, Marta Szymanowska, Describing model relations: The case of the capital asset pricing model (CAPM) family in financial economics, Studies in History and Philosophy of Science, Volume 97, 2023,

[3] Chen Yiying, Zhang Zexing, &Li Wenbin (2014). A neural network-based stock price prediction model Computer Applications and Software, 31 (5), 4

[4] Cao Aiqing, & Wu Miao. (2022). Daily Stock price prediction based on long short-term memory neural network. Information and Computers (034-001).