

Performances of the Moving Average Model and the Artificial Neural Network on the Forecast of Stock Market Indices

Peilin Jing, Jiachen Liu, Yaoyuan Zhang, Rashi Lodhi

Abstract

This project delves into the theoretical framework of two types of models. It aims to compare and contrast the performances of the moving average model and the artificial neural network on the prediction of stock market indices. We used the data collected from Yahoo Finance with daily frequency for the period from 1 January 2000 to 31 December 2021. By using a rolling window approach, we evaluated ARIMA-SGARCH to reflect the specific time-series characteristics and have better predictive power than the simple ARIMA model and Recurrent Neural Network models. In order to assess the precision and quality of these models in forecasting, we compared their equity lines, their forecasting error metrics and their performance metrics. The main contribution of this research is to show that the hybrid ARIMA SGARCH model outperforms the other models over the long term.

1. Introduction

The importance of this topic can be condensed to 4 points:

- **Interest in the area:** attracted attention of researchers, investors, speculators, and governments
- **ARIMA hybrid over ARIMA:** financial time series often do not follow ARIMA assumptions
- **newest ML techniques to improve models:** We use Recurrent Neural Network(RNN) model to determine if it can reflect the specific time series characteristics and predict better.

2. Literature Review

3. Methodology and Data

3.1 Data Analysis

The first step in the process was cleaning the data. Then, we transformed the adjusted price into a daily logarithmic return, which was calculated according to the following formula:

$$r_t = \ln \frac{P_t}{P_{t-1}}$$

Reasons to choose log returns:

- can be added across time periods to create cumulative returns
- easy to convert between log return and simple return
- log return follows normal distribution

Advantages to log return having normal distribution: - Distribution only dependent on mean and sd of sample - forecast with higher accuracy (log return) - Stock prices cannot be normal distribution

3.1.1 Descriptive Statistics - Stock prices

Figure 1 below presents the descriptive statistics of the adjusted closing prices

```
knitr::include_graphics("images/Descriptive_Statistics.png")
```

Statistics <chr>	Value <dbl>
Min Value	676.530029
1st Quantile	1173.805054
Median	1409.119995
3rd Quantile	2125.030029
Max Value	4793.060059
Mean	1772.378729
Skewness	1.400488
Kurtosis	1.457625

Figure 1: Descriptive Statistics for Prices

As seen in Figure 2 there are a few periods, such as 2008, 2011, 2015, and 2018, that show high volatility of returns. We can expect to build more accurate forecasting models if we are able to mitigate and “smooth” such periods.

```
knitr::include_graphics("images/PricesPlot.png")
```

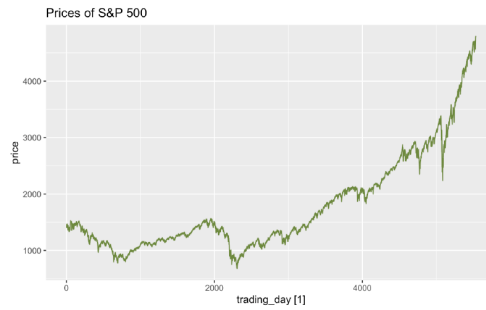


Figure 2: Prices Plot

Stock prices of SP 500 is that it is not normally distributed

3.1.2 Descriptive Statistics - Log Returns

Figure 3 presents the descriptive statistics of the adjusted closing prices

```
knitr::include_graphics("images/LogReturnDescriptive.png")
```

Statistics <chr>	Value <dbl>
Min Value	-0.1276521976
1st Quantile	-0.0047060675
Median	0.0006391235
3rd Quantile	0.0058129464
Max Value	0.1095719677
Mean	0.0002148568
Skewness	-0.4004359781
Kurtosis	11.0560760822

Figure 3: Descriptive Statistics for Log Returns

We use log returns to build models

3.2 Methodology

3.2.1 ARIMA (p,d,q)

The ARMA process is the combination of the autoregressive model and moving average [2] designed for a stationary time series. Autoregression (AR) describes a stochastic process, and AR(p) can be denoted as shown below:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \dots + \phi_p y_{t-p} + \varepsilon_t,$$

The moving average process of order q is denoted as MA(q) and the created time series contains a mean of q lagged white noise variables shifting along the series.

$$y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \dots + \theta_q \varepsilon_{t-q},$$

d is the number of differencing done to the series to achieve stationarity with I (d) so the ARIMA model can be expressed as

$$\text{Expand: } y_t = c + y_{t-1} + \phi_1 y_{t-1} - \phi_1 y_{t-2} + \theta_1 \varepsilon_{t-1} + \varepsilon_t$$

p is the number of autoregressive terms (AR) q is the number of moving average terms (MA)

ARCH(p), GARCH(r,s) and Hybrid ARIMA-SGARCH

The ARCH(p) model is given:

$$\sigma_t^2 = \omega + \sum_{i=1}^p \alpha_i u_{t-i}^2$$

- Most volatility models derive from this - Returns have a conditional distribution (here assumed to be normal) ARCH is not a very good model and almost nobody uses it. - The reason is that it needs to use information from many days before t to calculate volatility on day t . That is, it needs a lot of lags. - The solution is to write it as an ARMA model. - That is, add one component to the equation, $\beta \sigma_{t-1}$.

The GARCH(p,q) model is

$$\sigma_t^2 = \omega + \sum_{i=1}^p \alpha_i u_{t-i}^2 + \sum_{j=1}^q \beta_j \sigma_{t-j}^2$$

Where: α is news. β is memory. The size of $(\alpha + \beta)$ determines how quickly the predictability of the process dies out.

This leads us to lastly, ARIMA-SGarch

ARIMA SGARCH - Overview

Stock prices can be tremendously volatile during economic growth as well as recessions. When homoskedasticity presumption is violated, it affects the validity or power of statistical tests when using ARIMA models. We consider the SGARCH effect. The error term of the ARIMA model in this process follows SGARCH(1,1) instead of being assumed constant like the ARIMA model.

ARIMA SCGARCH - Steps

- 1) We conduct a rolling forecast based on an ARIMA-SGARCH model with window size(s) equal to 1000.
- 2) The optimized combination of p and q which has the lowest AIC is used to predict return for the next point. At the end, the vector of forecasted values has the length of 3530 elements
- 3) We describe and review our implementation of dynamic ARIMA($p,1,q$)-SGARCH(1,1) models with GED distribution and window size(s) equal to 1000.
- 4) we evaluate the results based on error metrics, performance metrics, and equity curves.

Iteration of the forecasting model ARIMA($p,1,q$)-SGARCH(1,1)

```
knitr::include_graphics("images/ARIMA_iteration.png")
```

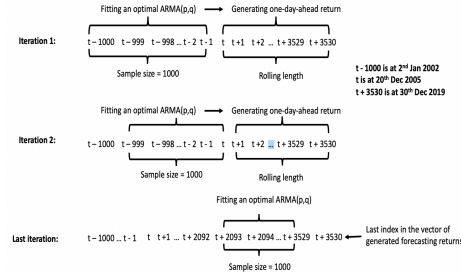


Figure 4: ARIMA iteration

Flowchart of the forecasting model ARIMA($p,1,q$)-SGARCH(1,1).

```
knitr::include_graphics("images/ARIMA_SGARCH_methodology.png")
```

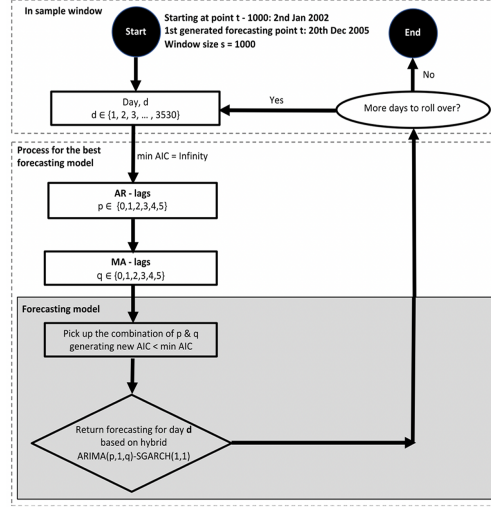


Figure 5: ARIMA_SGARCH methodology

Flowchart of the forecasting model $\text{ARIMA}(p,1,q)\text{-SGARCH}(1,1)$. This flowchart is for models with window size $s = 1000$.

3.2.2 RNN-LSTM Model

Recurrent Neural Network(RNN)

A **recurrent neural network** is a class of artificial neural network that uses sequential or time series data [1]. Unlike Feedforward Neural Network, RNN allows the output from some nodes to affect subsequent input to the same nodes by using connections between nodes to create cycles. As a result, the hidden layers produce the outputs with the input information and prior “memory” received from previous learning.

Unroll RNN

Recurrent Neural Network(RNN)

Another distinguish characteristic of RNN is that they share parameters across each layer of the network. Unlike feedforward neural networks having individual weight and bias for each node in one layer, recurrent neural networks share the same weight parameter within each layer. However, these weights are still adjusted during the processes of backpropagation and gradient descent to facilitate reinforcement learning.

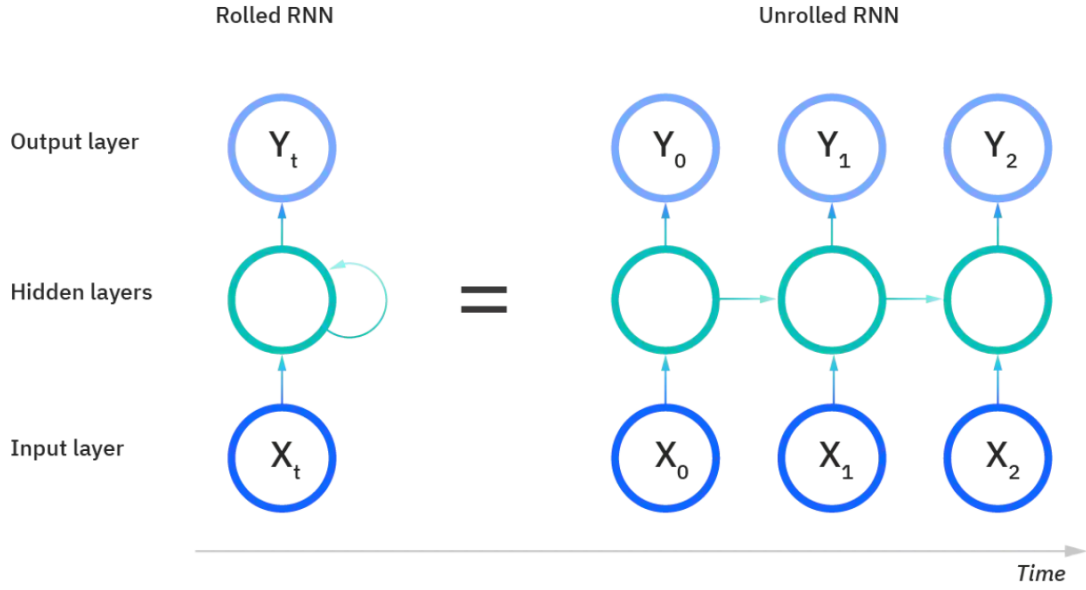


Figure 6: Rolled RNN and Unrolled RNN [1]

In feedforward neural network, backpropagation algorithm was used to calculate the gradient with respect to the weights. Recurrent neural network, on the other side, leverage backpropagation through time (BPTT) algorithm to determine the gradient as BPTT is specific to sequential data.

Activation Functions

In neural networks, an activation function determines whether a neuron should be activated and typically maps the input to $[0, 1]$ or $[-1, 1]$. The followings are two of the most commonly used activation functions and will be adopted later:

Sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Tanh (Hyperbolic tangent)

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

ReLU (Rectified Linear Unit) Activation Function

$$R(x) = \max(0, x)$$

Long Short-term Memory (LSTM)

Long short-term memory network, usually known as LSTM, is a specific RNN architecture first introduced by Sepp Hochreiter and Juergen Schmidhuber as a solution to vanishing gradient problem [2]. Recall with an RNN, similar with human reading a book and remembering what happened in the earlier chapter, it remembers the previous information and use it for processing the current input. The shortcoming of the NN is that it is not able to remember long term dependencies due to the vanishing gradient. The LSTM is designed to alleviate and avoid such issues.

The LSTM consists of three parts [5]:

- **Forget Gate:** Choose whether the information coming from the previous time stamp should be remembered or can be forgotten
- **Input Gate:** Learn new information from the input to this cell
- **Output Gate:** Passes the updated information tot the next time stamp

Forget Gate

In an LSTM cell, the cell first need to decide if the information from previous time stamp should be kept or forgotten. The equation of the forget gate is:

$$f_t = \sigma(W_f \cdot [x_t, h_{t-1}] + b_f)$$

Where

- x_t = input to the current time stamp
- h_{t-1} = hidden state of the previous time stamp
- W_f = weight matrix associated with hidden state
- b_f = constant

After that, a **sigmoid** function is applied over f_t and make it a number between 0 and 1. Then f_t is multiplied with the previous cell state. If $f_t = 0$, the network will forget everything from the previous time stamp while $f_t = 1$ represents that the network will remember everything.

Input Gate and new information

Next we decide what new information we will store in the cell state. First, the input gate decides which values we'll update with **sigmoid** activation function:

$$i_t = \sigma(W_i \cdot [x_t, h_{t-1}] + b_i)$$

Where

- W_i = weight matrix of the input associated with hidden state

Next, the new information is sent through a **tanh** layer to create the new candidate values:

$$\tilde{C}_t = \tanh(W_C \cdot [x_t, h_{t-1}] + b_C)$$

New Cell State C_t

With previous work, the LSTM cell now updates the new cell state for the current time stamp as:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

The current cell state C_t combines how much we decide to remember from the previous cell state C_{t-1} scaled by the forget gate and how much we wish to take in from the new current input \tilde{C}_t scaled by the input gate.

Output Gate

Finally, the cell needs to decide what it is going to output and by how much. The filter of the output is the output gate, with the following equation:

$$o_t = \sigma(W_o \cdot [x_t, h_{t-1}] + b_o)$$

The equation of the output gate is very similar with the forget gate and the input gate. Then, we push the cell state C_t through the **tanh** activation function to maintain the value staying in between -1 and 1 , and multiply it by the output gate:

$$h_t = o_t * \tanh(C_t)$$

Overall Module

The previous steps conclude the architecture of the LSTM. The whole process can be summarized and displayed as the following:

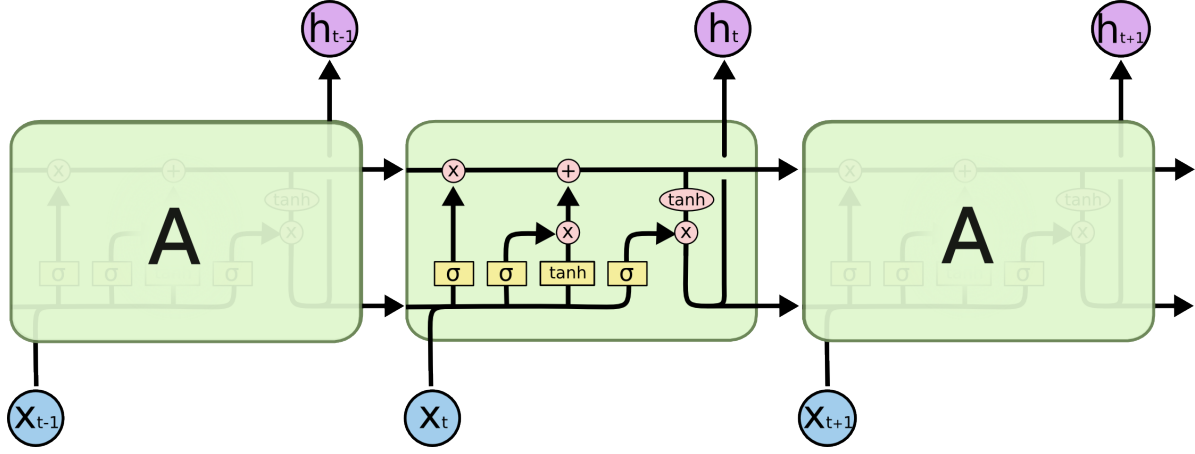


Figure 7: LSTM Chain [4]

4. Results

4.1 Model: Arima-SGARCH

In the paper, it had several different models, including SGARCH, EGARCH and Arima, etc. In the best model Arima-SGARCH, it tested for different window size and ,distribution model, and made the conclusion that hybrid Arima(p,1,q)- SGARCH(1,1) with $w=1000$ is the best model with best window size and distribution model according to the error metrics. In this section, we used ‘rugarch’ package to fit the hybrid Arima-GARCH model, compare the hybrid model, and do rolling forecast. so first, we will introduce a little bit about this package.

‘rugarch’ package exploration

- **ugarchspec()**: Method for creating a univariate GARCH specification object prior to fitting.
- **ugarchfit()**: Method for fitting a variety of univariate GARCH models.
- **ugarchroll()**: Method for creating rolling density forecast from ARMA-GARCH models with option for refitting every n periods with parallel functionality.
- **ugarchboot()**: Method for forecasting the GARCH density based on a bootstrap procedures (see details and references).
- **ugarchforecast()**: Method for forecasting from a variety of univariate GARCH models.

- **ugarchfilter()**: Method for filtering a variety of univariate GARCH models.
- **ugarchpath()**: Method for simulating the path of a GARCH model from a variety of univariate GARCH models.

We used the `ugarchspec()` to define our model, and `ugarchfit()` to fit our GARCH model. Then, we used forecasting functions in our project to do volatility forecast, which are `ugarchroll()` and `ugarchforecast()`.

Specify model

Specify sGarch model

```
# Specify sGARCH model
spec <- ugarchspec(
  variance.model =
    list(model = "sGARCH",
          garchOrder = c(1,1)),
  mean.model =
    list(armaOrder = c(0,0),
          include.mean = TRUE),
  distribution.model = "ged"
)
```

We choose the best model from the paper and reproduce it first. The best model is hybrid model ARIMA(p,1,q)-SGARCH(1,1) with GED distribution (SGARCH.GED 1000), so we define the model = “sGARCH” and define the distribution model as ged.

Information Criteria for sGARCH

Akaike	-6.518885
Bayes	-6.512893
Shibata	-6.518886
Hannan-Quinn	-6.516795

Specify eGarch model

```
# Specify eGARCH model
spec <- ugarchspec(
  variance.model =
    list(model = "eGARCH",
```

```

        garchOrder = c(1,1)),
    mean.model =
    list(armaOrder = c(0,0),
        include.mean = TRUE),
    distribution.model = "ged"
)

```

Information Criteria for eGARCH

Akaike	-6.553511
Bayes	-6.546321
Shibata	-6.553514
Hannan-Quinn	-6.551004

SGARCH and EGARCH comparison

These two models have similar information criteria, so we can say that it's somehow close to the comparison result in the paper since it has similar error metrics and performance metrics.

Forecast for fitted model

Forecast using ugarchforecast()

```

*-----*
*      GARCH Model Forecast      *
*-----*
Model: sGARCH
Horizon: 5
Roll Steps: 0
Out of Sample: 0

0-roll forecast [T0=2021-12-10]:
      Series  Sigma
T+1 0.0007399 0.01151
T+2 0.0007399 0.01153
T+3 0.0007399 0.01155
T+4 0.0007399 0.01158
T+5 0.0007399 0.01160

```

It's convenient to use `ugarchforecast()` for forecast future returns, but it will have look-ahead bias, which it use the information that is not yet available or known. So we use...

Rolling forecast using `ugarchroll()`

We defined our window size in the rolling forecast, and according to the paper, there are three different window size, which are 1000, 500, 1500. We changed the window size in `ugarchroll()` function and hold all other things the same to do comparison.

Rolling Forecast for window size 1000

```
# Example code with window size 1000
roll <- ugarchroll(spec = spec,
  data = data,
  n.ahead = 1,
  n.start = 3000,
  refit.every = 50,
  refit.window = "moving",
  solver = "hybrid",
  window.size = 1000,
  keep.coef = TRUE)

show(roll)
```

```
*-----*
*                GARCH Roll                *
*-----*

No.Refits      : 51
Refit Horizon  : 50
No.Forecasts   : 2521
GARCH Model    : eGARCH(1,1)
Distribution    : ged

Forecast Density:
      Mu  Sigma Skew Shape Shape(GIG) Realized
2011-12-06 6e-04 0.0147  0 1.372          0  0.0011
2011-12-07 6e-04 0.0139  0 1.372          0  0.0020
2011-12-08 6e-04 0.0132  0 1.372          0 -0.0211
2011-12-09 6e-04 0.0159  0 1.372          0  0.0169
2011-12-12 6e-04 0.0148  0 1.372          0 -0.0149
```

2011-12-13 6e-04 0.0163 0 1.372 0 -0.0087

.....
Mu Sigma Skew Shape Shape(GIG) Realized
2021-12-03 9e-04 0.0162 0 1.3263 0 -0.0084
2021-12-06 9e-04 0.0159 0 1.3263 0 0.0117
2021-12-07 9e-04 0.0142 0 1.3263 0 0.0207
2021-12-08 9e-04 0.0137 0 1.3263 0 0.0031
2021-12-09 9e-04 0.0118 0 1.3263 0 -0.0072
2021-12-10 9e-04 0.0121 0 1.3263 0 0.0095

Elapsed: 36.4266 secs

Rolling forecast error metrics

Error Metrics for 1000 window size

GARCH Roll Mean Forecast Performance Measures

Model : eGARCH
No.Refits : 51
No.Forecasts: 2521

Stats
MSE 0.0001064
MAE 0.0065300
DAC 0.5518000

Error Metrics for 500 window size

GARCH Roll Mean Forecast Performance Measures

Model : eGARCH
No.Refits : 51
No.Forecasts: 2521

Stats
MSE 0.0001065

MAE 0.0065300
DAC 0.5411000

Error Metrics for window size 1500

GARCH Roll Mean Forecast Performance Measures

Model : eGARCH
No.Refits : 51
No.Forecasts: 2521

Stats
MSE 0.0001064
MAE 0.0065340
DAC 0.5518000

Compared these the error metrics for these three models with different window size, $w = 1000$ and $w = 1500$ have the similar MSE, but $w = 1000$ has lower MAE, and $w = 1000$ and $w = 500$ has similar MAE but $w = 1000$ has lower MSE. It's obviously that hybrid Arima-SGARCH model with $w = 1000$ is the best model among these three models. Also, hybrid Arima(p,1,q)-SGARCH(1,1) with $w=1000$ will be the best model for predicting S&P 500 log return volatility since it has best performance.

5. Conclusions

References

- [1] IBM Cloud Education, "Recurrent Neural Network." IBM. September 14, 2020. <https://www.ibm.com/cloud/learn/recurrent-neural-networks#toc-types-of-r-q1VkG6gmhttps://www.ibm.com/cloud/learn/recurrent-neural-networks#toc-types-of-r-q1VkG6gm>
- [2] Hochreiter, Sepp and Schmidhuber, Jürgen, "Long Short-Term Memory." Neural Computation. 1997. <http://dx.doi.org/10.1162/neco.1997.9.8.1735>
- [3] Jahan, Israt, "Stock Price Prediction Using Recurrent Neural Networks." North Dakota State University. June, 2018. <https://hdl.handle.net/10365/28797>
- [4] Saxena, Shipra, "Introduction to Long Short Term Memory(LSTM)." Analytics Vidhya. March 16, 2022. <https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/>

[5] Olah, Christopher, “Understanding LSTM Networks.” colah’s blog. August 27, 2015.
<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>