Comparison Studies on Predicting Time Series Using Traditional Statistical Methods and LSTM

Zimeng Li, Lehao Fu, and Shuyi Wang

Abstract

With the rapid development of artificial intelligence, recent years have seen various machine learning methods applied to time series modeling and forecast. LSTM is just one of those machine learning methods. We wonder whether LSTM performs better than out traditional statistical methods like ARIMA on different datasets. So, we apply the two methods to three datasets with different characteristics to make a comparison.
Keywords:Time series forecasting, LSTM, ARMA-GARCH model.

1. Introduction

Developing efficient models to predict time series is undoubtedly a significant part of time-series analysis. While there are many traditional approaches to build models for time series and do predictions based on the specific model, ARIMA, GARCH(Bollerslev (1986)), to name but a few, recent years have seen professionals turn to Main Learning and Deep Learning, trying to make even more accurate predictions. LSTM (Long Short-Term Memory)Hochreiter & Schmidhuber (1997) is one of the machine-learning methods that can be applied to fit a given time-series and make predictions.

LSTM has gained great popularity for this model considers both short-term and long-run implication. On top of that, artificial neural networks are good nonlinear function approximators, so they provide a natural approach to deal with time series that are suspected to have nonlinear dependence on inputs. Although LSTM-network enjoys the advantage of a more flexible estimating process, whether LSTM outperforms traditional methods when it comes to modeling time-series with different traits is a fascinating question.

When we use traditional statistical methods to analysis time-series, chances are that we first plot the series, figuring out some simple characteristics such as seasonality or evident rising trend, then we use specific methods to capture those traits, turning them into a mathematic form. But when applying machine-learning method, we don't have to pay too much attention to those features due to the model's flexibility.

In this article, we try to use both traditional models like ARMA-GARCH and LSTM to fit real-world data with different traits. To be specific, we use three datasets. The first one is the common studied "flights" contained in Python seaborn library, which serves as a representative of time-series with both rising trend in the long-term as well as seasonality. The second dataset shows the daily hit of the USTC-Secondary-Class platform and we will show in section 4 that this time series looks quite gentle without evident trend. The third dataset contains the GDP and population data of China in the past decades and we can see a steep rising trend without seasonality from the plot. The above three datasets cover the most common time-series in our daily life and thus studying them is of great importance.

The rest of the article is organized as follows. Section 2 introduces the LSTM method and the applied algorithm. We also present the GARCH model in this section. In Section 3 we employ both the traditional statistical model such as ARIMA and LSTM to three different datasets and analysis the results in detail. At last, we give a brief discussion in Section 4 to end this article.

2. Methodology

In this section, we give a detailed introduction to models or methods that are used in Section 3.

2.1. *Long Short-Term Memory Network (LSTM)*

Imagine that you are reading an essay, it's human nature that we understand each word based on our understanding of previous words in the sentence, which means our thoughts have persistence. However, traditional neural networks can't do this. To address this issue, RNN (Medsker & Jain (2001)) was proposed. This kind of networks has loops, allowing information to persist. In cases where the gap between the relevant information and the place that it's needed is small, (for example, when we try to predict the last word in a single sentence with complete meaning, it's likely that we don't need any further context), traditional RNN can give satisfied results. Unfortunately, when such gap grows and long-term information should be taken into consideration, RNNs become unable to connect the information. That's why LSTM was proposed.

LSTM is a special kind of RNN, capable of learning both long-term and short-term dependencies. It's such a valuable characteristic that make LSTM stand out and become a popular way to predict time-series. LSTM has the form of a chain of repeating modules and each single module is composed of four neural network layers interacting in a special way. Figure 1 below shows the basic structure of LSTM.

The key to LSTM is the **cell state**, which is the horizontal line running through the top of the diagram, representing long-term memory. It runs straight down the entire chain with some linear interactions. And the line at the bottom of the diagram is the **hidden state** representing short-term memories.

The first step in a LSTM is to decide what information we're going to throw away from the cell state. And this decision is made by a sigmoid layer called the "forget gate layer". It uses h_{t-1} (the short-term memory at time $t-1$) and x_t (the newly-input information at time t) to output a number between 0-1 after using the sigmoid function, determining what percentage of the Long-Term Memory is remembered. To be specific, we use $f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$, where W_f and b_f are given parameters which don not change among different modules. After this, the remaining long-term memory becomes $C_{t-1}f_t$.Figure 2(a) shows this procedure.

The next steps (shown in Figure 2(b) and 2(c)) aim to create a potential long-term memory based on the short-term memory at time $t-1$ and the input information. Concretely speaking, we have $i_t = \sigma(W_i[h_{t-1,x_t} + b_i])$,$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$. After that, we product the above two results to get potential long-term memory, which is later added to the long-term memory after step1 to get the final long-term memory. That is to say, we have $C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$.

The next stage (shown in Figure 2(d)) updates the short-term memory using the newly-updated long-term memory as input and is computed as $h_t = o_t * tanh(C_t)$.

The above steps show the whole procedure of each module. Use time series information as x_t, we can derive the fitted model.

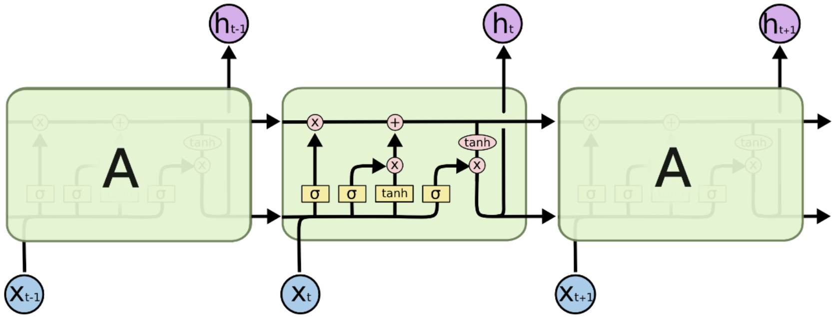
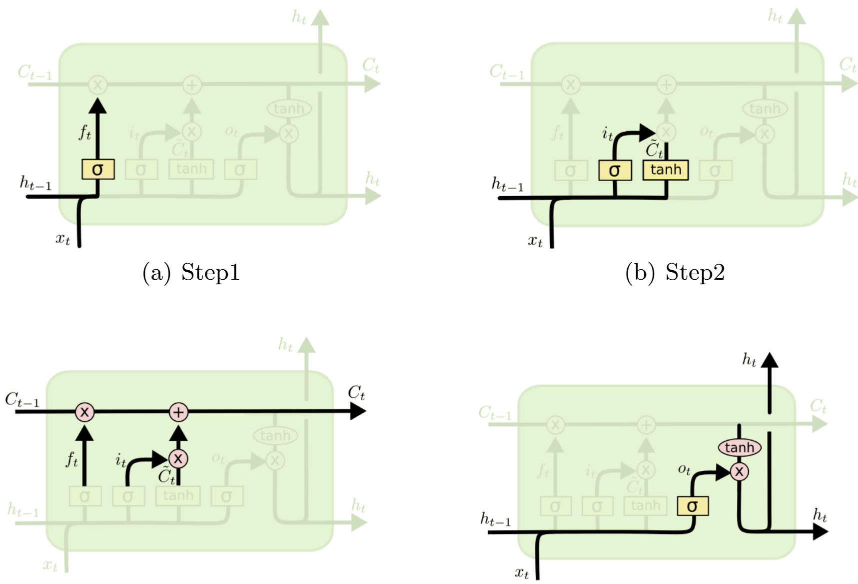


FIGURE 1. The basic structure of LSTM



(a) Step1

(b) Step2

(c) Step3

(d) Step3

FIGURE 2. Detailed steps

2.2. *Seasonal Auto Regressive Integrated Moving Average (SARIMA)*

We commonly see that financial data has evident cycle, especially those monthly or quarterly data. In a time series, if we find the data appears assimilation after S time-unit, for example, they are both at the peak or trough, we say that the specific time series has periodic characteristics with S as the cycle. And the model that depicts this kind of time series is called seasonal ARIMA model (SARIMA).

Assume that a time series with seasonality $\{X_t\}$ becomes a stationary series after $D - order$ seasonal difference, and the new series is $W_t = (1 - B^S)^D X_t$. Furthermore, we assume that $\{W_t\}$ follow $ARMA(P, Q)$. As a result, we get

$$U(B^S)(1 - B^S)^D X_t = V(B^S)\epsilon_t \tag{2.1}$$

$$U(B^S) = 1 - A_1 B^S - A_2 B^{2S} - \cdots - A_P B^{PS} \tag{2.2}$$

$$V(B^S) = 1 + H_1 B^s + H_2 B^{2s} + \cdots + H_Q B^{QS} \tag{2.3}$$

In the above-mentioned model, we assume that $u_t = \frac{U(B^S)(1-B^S)^D}{V(B^S)} X_t$ is a white-noise series, which is not necessarily the case. Since seasonal difference only deals with seasonality and the ARMA model based on W_t only considers the correlation between the same period points in different periods, short-term influence may be ignored. To solve this problem, we instead assume that u_t follows $ARIMA(p,d,q)$, and we will get

$$\Phi(B)U(B^S)\nabla^d\nabla_S^D X_t = \Theta(B)V(B^S)\epsilon_t \tag{2.4}$$

and we use $ARIMA(p,d,q)\times(P,D,Q)_S$ to denote this. For example, the $ARIMA(0,1,1)\times(0,1,1)_{12}$ represents the following model:

$$(1-B)(1-B^{12})X_t = (1+\theta_1 B)(1+\theta_{12}B^{12})\epsilon_t \tag{2.5}$$

In a nutshell, the SARIMA model is constructed by the following three steps:

- Use ARMA(p,q) to extract short-term correlation
- Use ARMA(P,Q) with cycle S to represent the seasonality
- Assume the multiplicative seasonal model with $\nabla^d\nabla_S^D X_t = \frac{\Theta(B)}{\Phi(B)}\frac{\Theta_s(B)}{\Phi_s(B)}\epsilon_t$

2.3. *Generalized Autoregressive Conditional Heteroscedastic Model (GARCH)*

In the ARIMA model, we assume that the innovation sequence ϵ_t is independently and identically distributed with the same variance. However, it has been found that stock prices and other financial variables have the tendency to move between high volatility and low volatility. And volatility is an important measure of risk. Then the GARCH model was proposed and has become an essential tool of modern asset pricing theory and practice.

Let r_t denotes a log-return sequence, and $a_t = r_t - \mu_t = r_t - E(r_t|F_{t-1})$ represents the innovation sequence. We say that $\{a_t\}$ follows the $GARCH(m,s)$ model, if $\{a_t\}$ satisfies

$$a_t = \sigma_t\epsilon_t \tag{2.6}$$

$$\sigma_t^2 = \alpha_0 + \Sigma_{i=1}^m \alpha_i a_{t-i}^2 + \Sigma_{j=1}^S \beta_j \sigma_{t-j}^2 \tag{2.7}$$

where $\{\epsilon_t\}$ is the independently and identically distributed white-noise sequence with zero-mean and unit variance, and $\alpha_0 > 0, \alpha_i \geqslant 0, \beta_j \geqslant 0, 0 < \Sigma_{i=1}^m \alpha_i + \Sigma_{j=1}^s \beta_j < 1$

3. Application to Real-World Data

In this section, we apply the LSTM and traditional statistical methods like ARIMA to three datasets with different traits, trying to make comparisons between the two methods.

3.1. *Flight (Time Series with Seasonality and Rising Trend)*

First, we employ LSTM and traditional methods to the *Flight* dataset, which is a commonly used benchmark datasets from *Python seaborn library*, containing monthly information about the number of passengers taking airplanes. The data consists 144 pieces of data, starting from January 1949. To compare the forecast results of the two methods, we chose the first 132 ones as training set, leaving the last 12 months as the test set, and we use the mean square error $mse = \|\boldsymbol{y}_{predict} - \boldsymbol{y}_{real}\|_2^2$ as the judging criterion, where $y_{predict}$ and y_{real} are both vectors representing corresponding information.

To give an intuitive impression of the data, we first plot the time series. It's distinctive in Figure 3 that this series contains seasonality and a rising trend as the same time. Taking the rising trend into consideration, in the traditional methods, we first differential the sequence and draw the auto-correlation function plot and partial auto-correlation

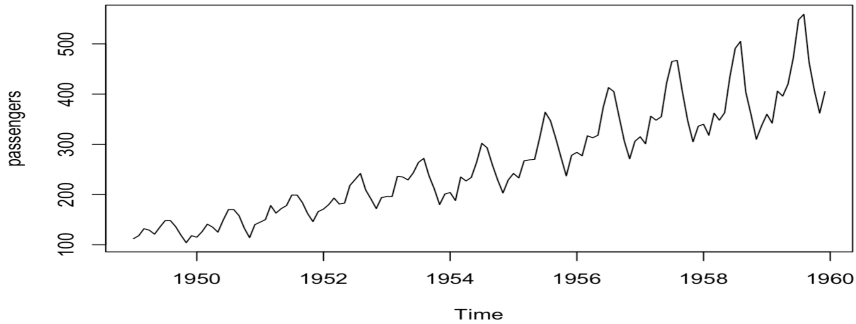


FIGURE 3. The time series plot of Flight dataset

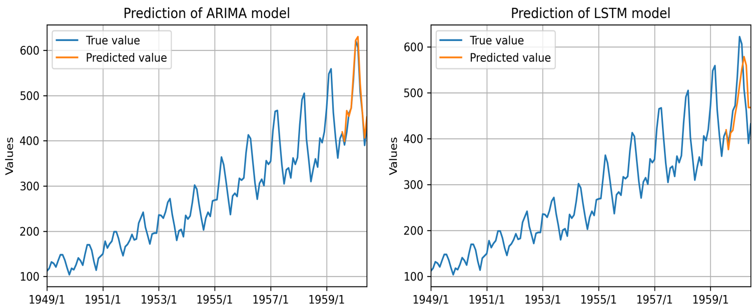


FIGURE 4. Forecasting results of the Flight dataset, with the left one showing the prediction of SARIMA model and the right one showing the prediction of LSTM model

plot. The two plots show that the series has evident seasonality, with distinct 12-order auto-correlation, which is consistent with our observation. After this, we find that the sequence after simple first-order difference and seasonal difference successfully passes the ADF test, showing that the newly-get sequence is stationary. Unfortunately, results from white-noise test indicate that we don't have enough evidence to reject the null hypothesis with the p-value around 0.05. After tentative modeling the sequence with ARIMA, we get a unsatisfying results. Reflecting on the whole process, we realize that the original sequence has a rapid rising trend with a high difference between the begin and end of the series. So, we instead do the logarithmic and differential treatment to the training series, and follows the above mentioned process. In the end, we find the residuals succeed to pass the white-noise test and the new model is much better with a much lower AIC and BIC. After getting a satisfying model, we predict the number of passengers 12 months ahead and gained the mse.

While using LSTM to predict the number of passengers of the following year, we first reconstruct the original dataset to make it suitable for our LSTM training. We build a new data frame with 13 columns, with the first 12 elements representing the short-term memory for the 13th one in each row. Then we feed the data frame into the LSTM model using L_2 loss as the minimization rule. After some parameter-tunning work, we gain the prediction of the last 12 months.

Figure 4 show the predicting results from the two methods respectively. Table 1 shows a clear comparison using mse. From simple observation, it seems that traditional methods outperforms the LSTM. To gain a more accurate conclusion, we compute the mse as mentioned before to make further comparison and the results is shown in Table1. From

Method	MSE
SARIMA	4.15e3
LSTM	4.18e4

Table 1. Mses of prediction results of Flight dataset show that SARIMA does a better job.

Table 1, it's intuitively that the SARIMA model performs much better than LSTM with a much smaller *mse*.

One reason that might account for the results is that the Flight sequence enjoys some good traits that can be easily expressed in mathematical language and we have corresponding models to deal with this issue. For example, we find SARIMA a great model to depict the seasonality. On the other hand, though LSTM enjoys popularity for its flexibility to a variety of time series, this nonparametric approach may fail to give accurate results when there does in fact exists an underlying math model.

3.2. *Second Class Access Data of USTC (Time Series Without Evident Fluctuation)*

The USTC Second Class program is an indispensable part of the students' campus life and every student get information about various activities through the second-class platform. From another perspective, for event organizers, it's of great importance to show their programs to as many as possible students so that more university man can take part in the program. In a nutshell, it's worthwhile to analysis the daily visits of the platform and give predictions about future visits for fear that organizers can publish their projects on days with potentially high hits. Hopefully, in this way, students are more likely to notice those fascinating activities and planners can draw more attention.

To achieve this, we emailed the person in charge of the second-class platform, aiming at getting daily visits to make further modeling and predictions. Many thanks for giving us such valuable data! At first, the whole data we got was too huge to make further analysis. Then after communicating many times with a student from the student union, two columns that are of our interest - date and visits were extracted, making the modeling and prediction possible. Unfortunately, after simple processing, we found that the data itself has abnormal faults due to recording and handling errors by technicians from the platform. As a result, we only chose undergraduate students' visiting data in October and November. The chosen data consists 61 pieces of information and we select the last 7 days as test data and the remaining days as training data. And we use the same criterion as that mentioned in Section 3.1 to do the comparison.

We first plot the data as shown in Figure 5. It seems that this series is relatively gentle without evident seasonality or trend. So, while using traditional methods, we first do ADF test to check whether the sequence is truly a stationary sequence without unit-root. It turned out that the p-value is 0.7, implying that the original series is not stationary. So, we differential the sequence, finding that the new series passes the stationary test but is not a white-noise sequence. Then we attempt to use ARMA model to fit this sequence and the residuals are truly white-noise sequence.

Then we use LSTM to fit the same sequence. This time we chose weekly information as the short-term memory, then follows the same steps as mentioned in detail before.

After the model-fitting process, we predict the last 7 days and compute corresponding *mse*, the predicting results and *mse* are shown in Figure 6 and Table2. We find that when predicting the second-class data, LSTM has better behavior than ARIMA model

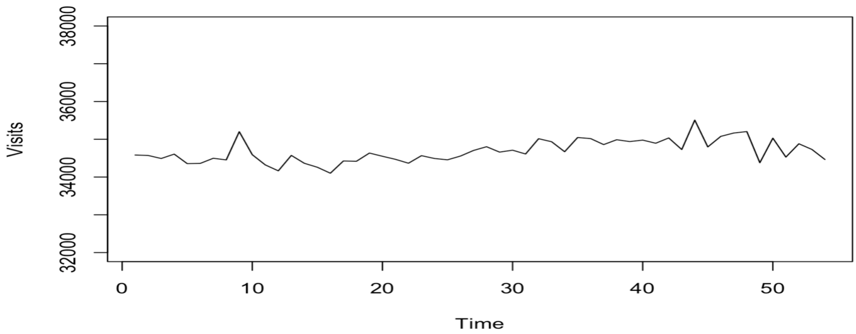


FIGURE 5. The time series plot of Second-Class dataset

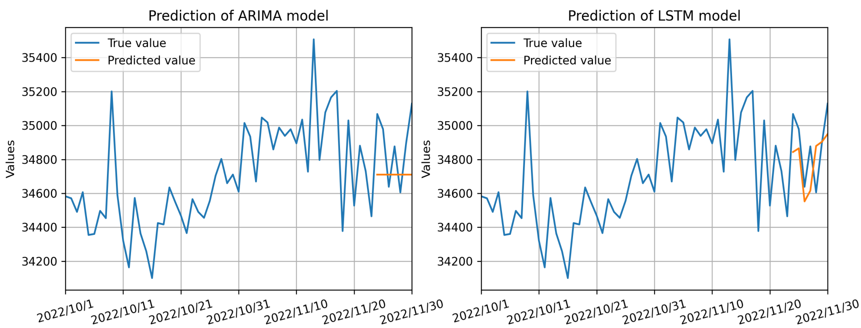


FIGURE 6. Forecasting results of the Second-Class dataset, with the left one showing the prediction of ARIMA model and the right one showing the prediction of LSTM model

Method	**MSE**
ARIMA	4.53e5
LSTM	2.46e5

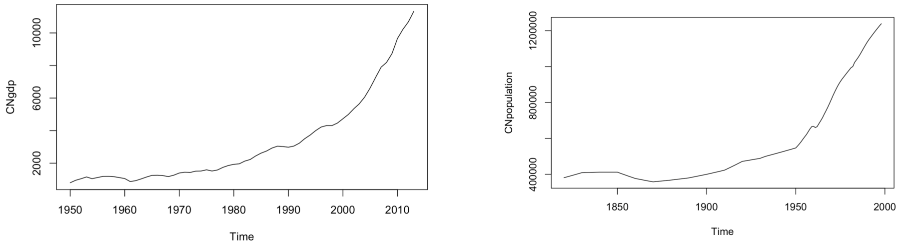
TABLE 2. Mses of prediction results of Second-Class dataset show that LSTM performs better.

according to the *mse*. And we notice that, since in the application process, we use MA model to deal with the sequence after first-order difference, and use the mean value of prediction model as our final result, the predictions keep time for the last 7 days, which may explain why the ARIMA model fail to give a satisfying result.

In this study, the results show that there may be a lack of appropriate mathematic model to fit the sequence, leading to the relatively poor performance. On the contrary, since LSTM is a nonparametric method without any model assumptions, it may work better under circumstances when it's hard to find suitable model.

3.3. *Chinese GDP and Population Data (Time Series with Distinct Rising Trend)*

The previous two real-world applications show the comparison results of time series with seasonality and series that seems stationary respectively. In this subsection, we apply the two methods to time series with rising trend but no seasonality, which can be observed immediately from the plots in Figure 7(a) and Figure 7 (b). The dataset used is from *Maddison Project Database 2018*, and we only use the Chinese data.



(a) Chinese GDP data (b) Chinese Population Data

FIGURE 7. Chinese GDP and Population data

	GDP	Population
ARIMA-GARCH	7.83e8	3.52e9
LSTM	8.90e4	5.25e9

TABLE 3. Mses of prediction results of Chinese GDP and Population dataset show that LSTM does better on the GDP dataset while ARIMA-GARCH does a better job on the Population dataset.

For the GDP prediction study, we totally have 69 pieces of data staring from 1950 and we choose the last 5 years as the test data. Considering the significant growth trend, while using traditional method, we differential the logarithmic sequence at the first step, checking the stationarity of the sequence after transformation. Then we use the ARIMA model to fit the sequence and find that the residuals come from white-noise series. On top of that, since GDP data is one type of finance data, where volatility analysis is of great significance, we also check whether the white-noise residuals has the *ARCH* effect and it turns out that the series does has *ARCH* effect. After deciding the orders by taking a closer look at the acf and pacf graphs, we use *ARMA-GARCH(1,1)* model to refit the logarithmic difference sequence. In the end, we forecast the last 5 years' GDP based on the model we have built. Since LSTM have fixed input format and operation framework, we just repeat our work again, choosing the past 5 years as short memory and doing parameter-tunning.

Then it comes the population forecasting work. This time we obtain 199 terms of data recording Chinese population from year 1820 to year 2018. We select the last 20 years to test the modeling results. When following the traditional methods, we first differential the logarithmic sequence, trying to get a stationary series. And then we try to fit the series into ARIMA model and it turns out that we better carry out the first-order difference again. After finishing the mean-value construction work, we focus our attention to deal with the innovation series for it is proven to have arch effect after testing. Then after order-determination, we use ARMA-GARCH(2,0) model to fit the series. It worth mentioning that, since we can never know the actual underlying model, we adopt a tentative modeling method, during which we find that different GARCH models will contribute to quite different fitting results, which is similar to the parameter-tunning process in LSTM. This time in LSTM, we use the past 10 years as the short-term memory.

The forecasting results from the two approaches are represented in Figure 8, Figure 9 and Table3. In the GDP application, we find that LSTM holds up better than the

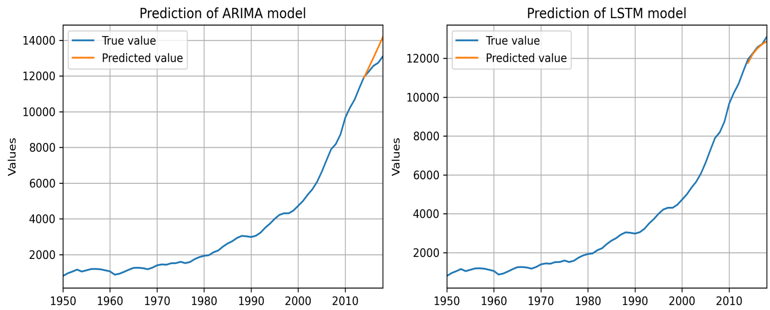


FIGURE 8. Forecasting results of the Chinese GDP dataset, with the left one showing the prediction of ARIMA-GARCH model and the right one showing the prediction of LSTM model

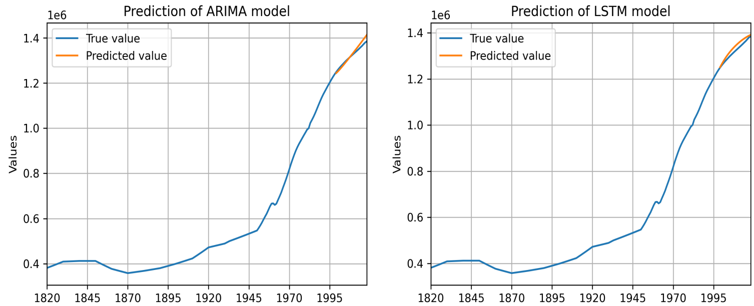


FIGURE 9. Forecasting results of the Chinese population dataset, with the left one showing the prediction of traditional methods and the right one showing the prediction of LSTM model

ARIMA-GARCH model concluding from both the graphs and their mse. Nevertheless, it's clear that traditional methods do better than LSTM when it comes to forecasting the population data. In a nutshell, even though the GDP and population series own similar features, their comparison outcomes are opposite. We think this might be a result of improper modeling using the traditional methods to fit the GDP data. Taking a closer look at the GDP data, we find that every ten years or so, there is a period of slowing growth, which we fail to reflect in the modeling process. And even though it seems better to use traditional methods when predicting the population data, we have to point out that we have tried many models and some of them have quite poor performance. On top of that, it's surprising that LSTM does a quite good job in the GDP case, showing the great power of machine learning.

4. Conclusion and Remarks

To conclude this article, we first summarize our experimental results and analysis conclusions. To make it clear, a comparison of **mses** using the two methods on the three datasets are shown in Table 4.

On the whole, we conclude that when the time series satisfies the model assumptions of a given mathematic model, the traditional methods perform much better than LSTM for the simple reason that LSTM only learn from data itself without considering the underlying model. Unfortunately, chances are that real-world data fail to meet the model assumptions. Or in another words, it's hard for us to find the mathematic model perfectly

Dataset	LSTM	Traditional Methods	Winning Method
Flight	4.18e4	4.15e3	Traditional Methods
Second-Class	2.46e5	4.53e5	LSTM
Chinese GDP	8.90e4	7.83e8	LSTM
Chinese Population	5.25e9	3.52e9	Traditional Methods

Table 4. mses of prediction results of all datasets

fits the series. Under such circumstances, LSTM may stand out, giving great outputs. Additionally, LSTM is expert in capturing even smaller features, such like that shown in the GDP dataset where there is a slowing growth every ten years or so.

Besides, the application process give us a deeper understanding of the modeling procedure and just like what we have learned in class, there's no omnipotent model or method and we still have a long way to go to get more accurate forecasting results.

Here we also discuss several possible topics for future study. First, we only consider univariate time series. But there are many great machine-learning methods like LST-Net(Lai *et al.* (2018)) and MTGNN(Wu *et al.* (2020)) that considers temporal and spatial correlations among multivariate time series. We may compare those methods with traditional methods like VAR and DCC-GARCH(Engle (2002)) in the future. Besides, we find that both traditional methods and LSTM are far from perfect. We may explore ways to combine the two methods to gain better results in the future.

Contributors

In our group, Zimeng Li reviewed literature, provided ideas and was responsible for writing thesis. Lehao Fu offered precious ideas as well. He also collected all the data and wrote codes for LSTM methods, during which tunning parameter was a harsh job. Shuyi Wang processed the data and mainly wrote codes using traditional methods, she also helped to check the thesis. At last, we have to point out that we helped each other a lot in order to complete the project and all the three group members **contributed equally to this project** considering the harsh process attaining data, writing codes to realize those applications and writing thesis as well as revising the thesis.

REFERENCES

Bollerslev, Tim 1986 Generalized autoregressive conditional heteroskedasticity. *Journal of econometrics* **31** (3), 307–327.

Engle, Robert 2002 Dynamic conditional correlation: A simple class of multivariate generalized autoregressive conditional heteroskedasticity models. *Journal of Business & Economic Statistics* **20** (3), 339–350.

Hochreiter, Sepp & Schmidhuber, Jürgen 1997 Long short-term memory. *Neural computation* **9** (8), 1735–1780.

Lai, Guokun, Chang, Wei-Cheng, Yang, Yiming & Liu, Hanxiao 2018 Modeling long- and short-term temporal patterns with deep neural networks. In *The 41st international ACM SIGIR conference on research & development in information retrieval*, pp. 95–104.

Medsker, Larry R & Jain, LC 2001 Recurrent neural networks. *Design and Applications* **5**, 64–67.

Wu, Zonghan, Pan, Shirui, Long, Guodong, Jiang, Jing, Chang, Xiaojun & Zhang, Chengqi 2020 Connecting the dots: Multivariate time series forecasting with graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 753–763.

Appendix

In the appendix, we show the main parts of our codes.

```
#The processing procedure of the second-class dataset,written in R
library(dplyr)
#data-reading
sys_log7 <- read.csv("ek\\sys_log7.csv", header = TRUE, na.strings =
    "")
sys_log8 <- read.csv("ek\\sys_log8.csv", header = TRUE, na.strings =
    "")
sys_log9 <- read.csv("ek\\sys_log9.csv", header = TRUE, na.strings =
    "")
sys_log10 <- read.csv("ek\\sys_log10.csv", header = TRUE, na.strings
    = "")
sys_log11 <- read.csv("ek\\sys_log11.csv", header = TRUE, na.strings
    = "")
sys_log12 <- read.csv("ek\\sys_log12.csv", header = TRUE, na.strings
    = "")

#data-integration
data <- rbind(sys_log7, sys_log8, sys_log9, sys_log10, sys_log11, sys
    _log12)

#remove duplicate lines
data <- data %>% distinct(userid, createTime, .keep_all = TRUE)

#data-processing
data_process <- function(data){
  data$createTime <- as.Date(data$createTime)
  data <- as.data.frame(table(data$createTime))
  colnames(data) <- c("Time", "Visits")
  return(data)
}

dataPB <- data[substr(data$userid, 1, 2) == "PB" & !is.na(data$userid
    ) & !is.na(data$createTime), ]
dataPBB <- data_process(dataPB)
write.csv(dataPBB, file = "ek\\dataPBB.csv")

#Tradition-Methods Flight dataset, written in R
data1_1 <- read.csv("data\\flights.csv", header = TRUE)$passengers
data1 <- read.csv("data\\flights.csv", header = TRUE)$passengers[1 :
    132]
plot(ts(data1, start = c(1949,1), frequency = 12), ylab = "passengers
    ")

```

```
#differentiate the series
diff_data1 <- diff(log(data1))
plot(ts(log(data1), start = c(1949,1), frequency = 12), ylab = "
    passengers")
plot(ts(diff_data1, start = c(1949,1), frequency = 12), ylab = "
    passengers")
acf(diff_data1)
pacf(diff_data1)

#seasonal differentiate
dd_data1 <- diff(diff_data1, lag = 12)
plot(ts(dd_data1, start = c(1949,1), frequency = 12), ylab = "
    passengers")
adf.test(dd_data1) #unit-root test

#white-noise test
Box.test(dd_data1, lag = 30, type = "Ljung")
t.test(dd_data1)
acf(dd_data1)
pacf(dd_data1)
auto.arima(ts(diff_data1, start = c(1949, 1), frequency = 12),
    seasonal = TRUE)

#refit the data
mod1 <- arima(log(data1), order = c(0, 1, 1), seasonal = list(order =
     c(0, 1, 1), period = 12))
res1 <- residuals(mod1)
Box.test(res1, lag = 30, type = "Ljung")

#predict
pre1 <- predict(mod1, n.ahead = 12)
pre1
round(exp(pre1$pred), 2)
plot(data1_1, ylab = "passengers", xlab = "Time", type = "l")
points(133 : 144, round(exp(pre1$pred), 2), type = "l", col = "red")

#computing mse
er1 <- sum((data1_1[133 : 144] - exp(pre1$pred)) ^ 2)
round(er1, 2)

#Traditional Methods for Second-Class Data
data2_1 <- read.csv("data\\dataPB(10-11).csv", header = TRUE)$Visits
data2 <- read.csv("data\\dataPB(10-11).csv", header = TRUE)$Visits[1
    : 54]
plot(ts(data2), ylab = "Visits")

#differentiate the series
diff_data2 <- diff(data2)
adf.test(diff_data2) #p-value<0.05
plot(ts(diff_data2), ylab = "Visits")
```

```
Box.test(diff_data2, lag = 30, type = "Ljung") #p-value<0.05
t.test(diff_data2) #p-value>0.05

#modeling
auto.arima(diff_data2)
mod2 <- arima(data2, order = c(0, 1, 1))
tsdiag(mod2)
Box.test(residuals(mod2), type = "Ljung-Box")

#prediction
pre2 <- predict(mod2, n.ahead = 7)
pre2
round(pre2$pred, 2)
plot(ts(read.csv("data\\dataPB(10-11).csv", header = TRUE)$Visits), ylab = "Visits")
points(55 : 61, pre2$pred, type = "l", col = "red")
er2 <- sum((data2_1[55 : 61] - pre2$pred) ^ 2)
round(er2, 2)

#Chinese GDP
data3_1 <- read.csv("./data/CNgdp2018.csv", header = TRUE)$gdp
data3_1 <- as.numeric(gsub(',', '', data3_1))
data3 <- read.csv("./data/CNgdp2018.csv", header = TRUE)$gdp[1 : 64]
data3 <- as.numeric(gsub(',', '', data3))
plot(ts(data3, start = 1950, frequency = 1), ylab = 'CNgdp')

#modeling
logdiff_data3 <- diff(log(data3), 1)
plot(ts(log(data3), start = 1950, frequency = 1), ylab = 'log_CNgdp')
plot(ts(logdiff_data3, start = 1950, frequency = 1), ylab = 'logdiff_CNgdp')
acf(logdiff_data3)
pacf(logdiff_data3)
adf.test(logdiff_data3)
Box.test(logdiff_data3, lag = 20, type = "Ljung") #p-value<0.05
t.test(logdiff_data3) #p-value<0.05
mean3 <- mean(logdiff_data3)
logdiff_data3 <- logdiff_data3 - mean3
Box.test(logdiff_data3 ^ 2, lag = 20, type = "Ljung")

#arch-effect
archTest(logdiff_data3, lag = 20) #p-value<0.05
auto.arima(logdiff_data3)
mod3 <- arima(logdiff_data3, order = c(1, 0, 0))
res3 <- residuals(mod3)
Box.test(res3, lag = 20, type = "Ljung") #p>0.05
archTest(res3, lag = 20)
acf(res3 ^ 2)
pacf(res3 ^ 2)
spec.sp1 <- ugarchspec(
```

```
  mean.model = list(armaOrder = c(1, 0), include.mean = FALSE),
  variance.model = list(model = "sGARCH", # standard GARCH model
                        garchOrder = c(1, 1))
)
mod33 <- ugarchfit(spec = spec.sp1, data = logdiff_data3)
show(mod33)

#predict
pre3 <- ugarchforecast(mod33, n.ahead = 5)
fitted(pre3)
pre33 <- c(0.0078836, 0.0036962, 0.0017330, 0.0008125, 0.0003809)
pre33 <- pre33 + mean3
pre333 = c()
for(i in 1 : 5){
  temp = log(data3)[64]
  for(j in 1 : i){
    temp = temp + pre33[j]
  }
  pre333 = c(pre333, temp)
}
pre333
exp(pre333)
data3_1[65:69]
plot(1 : 69, data3_1, ylab = "CNgdp", xlab = "Time", type = "l")
points(65 : 69, exp(pre333), type = "l", col = "red")
er3 <- sum((data3_1[65 : 69] - pre333) ^ 2)

#Chinese Population Data
data4_1 <- read.csv("./data/CNpop2018.csv", header = TRUE)$pop
data4_1 <- as.numeric(gsub(',', '', data4_1))
data4_train <- data4_1[1 : 179]
plot(ts(data4_train, start = 1820, frequency = 1), ylab = 'CNpopulation')

#modeling
logdiff_4<-diff(log(data4_train),1)
plot(log(data4_train),type="l")
plot(logdiff_4,type="l",ylim=c(-0.05,0.05))
#ADF-test
adf.test(logdiff_4)#p-value=0.05 stationary
Box.test(logdiff_4)#p-value<2.2e-16
logdiff_4<-logdiff_4-mean(logdiff_4)
auto.arima(logdiff_4,trace = T)
mod4 <- arima(logdiff_4, order = c(1, 1, 2))
logdiff_diff<-diff(logdiff_4)
mod5<-arima(logdiff_diff,order = c(1,0,2))
res4 <- residuals(mod4)
res5<-residuals(mod5)
Box.test(res5, lag = 20, type = "Ljung")
#p-value=0.8432, confirmed that this is actually a white-noise series
```

```
#arch-effect
archTest(res5)
acf(res5^2)
pacf(res5^2)
#has archeffect
spec.test1 <- ugarchspec(
  mean.model = list(armaOrder = c(1, 2), include.mean = FALSE),
  variance.model = list(model = "sGARCH", # standard GARCH model
                        garchOrder = c(2, 0))
)
mod4 <- ugarchfit(spec = spec.test1, data = logdiff_diff)

#predict
pre4 <- fitted(ugarchforecast(mod4, n.ahead = 20))
origin<-logdiff_4[178]
#get the log-difference prediction sequence
pre44 = c()
for(i in 1 : 20){
  temp = origin
  for(j in 1 : i){
    temp = temp + pre4[j]
  }
  pre44 = c(pre44, temp)
}
pre444 = c()
pre44 <- pre44 + mean(logdiff_4)
for(i in 1 : 20){
  temp = log(data4_train[179])
  for(j in 1 : i){
    temp = temp + pre44[j]
  }
  pre444 = c(pre444, temp)
}
prediction_4<-exp(pre444)
real_4<-data4_1[180:199]
plot(1 : 199, data4_1, ylab = "CNpop", xlab = "Time", type = "l")
points(180 : 199, prediction_4, type = "l", col = "red")
er4 <- sum((real_4 -prediction_4 ) ^ 2)
```

```
#LSTM codes for the Flight dataset,written in Python
#Codes for other applications is quite similar
import torch
import torch.nn as nn

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
```

```python

random_seed = 123
torch.manual_seed(random_seed)


# define our model: LSTM
class LSTM(nn.Module):
    def __init__(self, input_size=1, hidden_layer_size=100, output_
        size=1):
        super().__init__()
        self.hidden_layer_size = hidden_layer_size

        self.lstm = nn.LSTM(input_size, hidden_layer_size)

        self.linear = nn.Linear(hidden_layer_size, output_size)

        self.hidden_cell = (torch.zeros(1, 1, self.hidden_layer_size),
                            torch.zeros(1, 1, self.hidden_layer_size))

    def forward(self, input_seq):
        lstm_out, self.hidden_cell = self.lstm(input_seq.view(len(
            input_seq), 1, -1), self.hidden_cell)
        predictions = self.linear(lstm_out.view(len(input_seq), -1))
        return predictions[-1]


def load_seg(data_path, variable_name, test_data_size):
    # load the file flights.csv
    df = pd.read_csv(data_path)
    all_data = df[variable_name].values.astype(float)

    # segment the train and test
    train_data = all_data[:-test_data_size]
    test_data = all_data[-test_data_size:]

    # do the MinMaxScaler
    scaler = MinMaxScaler(feature_range=(-1, 1))
    train_data_normalized = scaler.fit_transform(train_data.reshape
        (-1, 1))
    train_data_normalized = torch.FloatTensor(train_data_normalized).
        view(-1)
    return train_data_normalized, scaler


# define the function in order to train and predict
def create_inout_sequences(input_data, batch_size=12):
    inout_seq = []
    L = len(input_data)
    for i in range(L - batch_size):
        train_seq = input_data[i:i + batch_size]
```

```
        train_label = input_data[i + batch_size:i + batch_size + 1]
        inout_seq.append((train_seq, train_label))
    return inout_seq


def train(train_data_normalized, batch_size=12):
    train_inout_seq = create_inout_sequences(train_data_normalized, batch_size)
    # set the loss, optim and epoch
    loss_function = nn.MSELoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
    epochs = 1000
    # train
    for i in range(epochs):
        for seq, labels in train_inout_seq:
            optimizer.zero_grad()
            model.hidden_cell = (torch.zeros(1, 1, model.hidden_layer_size),
                                 torch.zeros(1, 1, model.hidden_layer_size))

            y_pred = model(seq)

            single_loss = loss_function(y_pred, labels)
            single_loss.backward()
            optimizer.step()

        if i % 25 == 1:
            print(f'epoch: {i:3} loss: {single_loss.item():10.8f}')

    print(f'epoch: {i:3} loss: {single_loss.item():10.10f}')
    return model


def predict(model, train_data_normalized, scaler, batch_size=12, test_data_size=12):
    test_inputs = train_data_normalized[-batch_size:].tolist()
    model.eval()
    for i in range(test_data_size):
        seq = torch.FloatTensor(test_inputs[-batch_size:])
        with torch.no_grad():
            model.hidden = (torch.zeros(1, 1, model.hidden_layer_size),
                            torch.zeros(1, 1, model.hidden_layer_size))
            test_inputs.append(model(seq).item())

    actual_predictions = scaler.inverse_transform(np.array(test_inputs[batch_size:]).reshape(-1, 1))
    return actual_predictions

```

```python

def plot(test_data_size, df, variable_name, actual_predictions, save_path, title):
    x = np.arange(df.shape[0]-test_data_size, df.shape[0], 1)
    plt.title(title)
    plt.ylabel('Values')
    plt.grid(True)
    plt.autoscale(axis='x', tight=True)
    plt.plot(df[variable_name], label="True value")
    plt.plot(x, actual_predictions, label="Predicted value")
    plt.legend()
    plt.savefig(save_path)


def loss(actual_predictions, df, variable_name, test_data_size):
    actual_predictions = actual_predictions.squeeze()
    actual_predictions = [round(x, 2) for x in actual_predictions]
    all_data = df[variable_name].values.astype(float)
    true_data = all_data[-test_data_size:]
    print(f"The true value of last {test_data_size}: {true_data}\n")
    print(f"The predicted value of last {test_data_size}: {actual_predictions}\n")
    print(f"mse = {np.sum(np.square(true_data - actual_predictions))}")


data_path = "data/flights.csv"
variable_name = "passengers"
test_data_size = 12
batch_size = 12
df = pd.read_csv(data_path)

train_data_normalized, scaler = load_seg(data_path, variable_name, test_data_size)
model = LSTM()
model = train(train_data_normalized, batch_size)
actual_predictions = predict(model, train_data_normalized, scaler, batch_size, test_data_size)
plot(test_data_size, df, variable_name, actual_predictions,
     save_path="output/flights.png", title="Time vs Passengers")
loss(actual_predictions, df, variable_name, test_data_size)
```