Temperature Forecasting Using Long Short Term Memory (LSTM)

AUTHOR
Gaurav Kesireddy, Fariha Moomtaheen, Nitul Singha

PUBLISHED
July 24, 2023

Introduction

Recurrent Neural Networks (RNNs) are one of the most popular data-driven approaches used for time-series prediction. RNNs benefit from feedback loops which makes the output of RNN in one time step to be seen as an input in the subsequent one. Having information in a sequential form, the input of the RNN in each time step includes the information of the corresponding time step in the sequence and the previous information provided by a feedback loop(Karevan and Suykens 2020). Recurrent networks have the capability to use feedback connections, enabling them to store recent input events as short-term memory, distinct from the long-term memory represented by slowly changing weights. This feature is highly valuable for various applications, such as speech processing, non-Markovian control, Time Series analysis, and music composition. However, the most widely used algorithms for learning short-term memory suffer from time-consuming processes and limited effectiveness, particularly when dealing with long time lags between inputs and corresponding signals. Despite the theoretical appeal, existing methods do not offer clear practical advantages over backpropagation in feed-forward nets with limited time windows. As a solution to these challenges, the “Long Short Term Memory” (LSTM) presents a novel recurrent network architecture in conjunction with an appropriate gradient-based learning algorithm (Hochreiter and Schmidhuber 1997).

Long Short-Term Memory(LSTM), a novel recurrent network architecture in conjunction with an appropriate gradient based learning algorithm comes up as a remedy to solve this problem. It can learn to bridge time intervals in excess of 1000 steps even in case of noisy, in-compressible input sequences, without loss of short term time lag capabilities. This is achieved by an efficient, gradient-based algorithm for an architecture enforcing constant error flow through internal states of special units (Hochreiter and Schmidhuber 1997). LSTM, standing for Long Short-Term Memory, is a gated Recurrent Neural Network (RNN) (Sherstinsky 2020) specifically designed to address issues related to long-term dependencies and the problems of gradient vanishing or exploding present in traditional RNNs. By incorporating forget gates, input gates, and output gates, LSTM can selectively retain crucial information from time series data based on its features while disregarding irrelevant information. This selective retention enhances LSTM’s capability in processing time series data effectively. Recurrent neural networks with Long Short-term Memory have emerged as an effective and scalable model for several learning problems related to sequential data. The central idea behind the LSTM architecture is a memory cell which can maintain its state over time, and non-linear gating units which regulate the information flow into and out of the cell. Most modern studies incorporate many improvements that have been made to the LSTM architecture since its original formulation.However, LSTM’s are now applied to many learning problems which differ significantly in scale and nature from the problems that these improvements were initially tested on(Greff et al. 2016).

The significance of LSTM is evident in its widespread application across various domains of time series data processing. For instance, LSTM has been utilized in tasks like text categorization, sentence generation, and machine translation (Zhu et al. 2019). Time series data, which refers to a collection of observations arranged chronologically, holds valuable decision support potential. Time series forecasting has attracted significant attention, and traditional methods for time series prediction can be categorized into linear and nonlinear models. Early on, linear models like Auto-Regressive (AR), Moving Average (MA), and Auto-Regressive Moving Average (ARMA) were proposed for time series forecasting [Wadhvani et al. (2017)](Mo and Tao 2016)(Ge and Kerrigan 2016). While computationally simple and effective for stationary time series data, their predictive accuracy is limited when dealing with non-stationary data. To address this limitation, the Auto-Regressive Integrated Moving Average (ARIMA) model was introduced (Ho and Xie 1998), which combines differencing operations to handle non-stationary data.

Due to information explosion in recent years, a large amount of data has become available online and manually analyzing this data is not feasible. Various supervised and unsupervised machine learning techniques have been developed to automatically gather and analyze data and make predictions(Yadav, Jha, and Sharan 2020). Given that many real-world phenomena exhibit nonlinear behaviors, the focus of research in time series data prediction has shifted towards nonlinear models. Common time series data prediction approaches involve Support Vector Machines (SVM) and neural networks [Vapnik (1999)](Hecht et al. 1989). SVM-based methods have been widely proposed and applied in various domains, including ground motion dynamic prediction and fault prediction [P. Li et al. (2011)](Z. Wang et al. 2017). However, neural networks, particularly Artificial Neural Networks (ANN) and Recurrent Neural Networks (RNN), have gained prominence due to their ability to handle large-scale and nonlinear data [Crone and Kourentzes (2010)](Boné, Assaad, and Crucianu 2003). While the Back-propagation (BP) neural network has demonstrated effectiveness in PM2.5 and traffic prediction [Y. Wang (2019)](Chen, Zhao, and Yan 2016), RNN has become popular due to its sequential structure capturing timing characteristics, and RNN-based time series prediction methods have shown superior performance (L. Li et al. 2019).

A critical area of machine learning is Time Series forecasting, as various forecasting problems contain a time component. It is one of the most applied data science technique in business, supply chain management, and production. A series of observations taken chronologically in time is known as a Time Series(TS). A time-series dataset is different from other datasets, as it adds a time element. This added element is both a limitation and a structure that offers an additional source of information(Yamak, Yujian, and Gadosey 2019). TS prediction is the method of forecasting upcoming trends/patterns of the given historical dataset with temporal features. A time series (TS) data can be break downed into trend, seasonality and error. A trend in TS can be observed when a certain pattern repeats on regular intervals of time due to external factors. Given the TS, it is broadly classified into 2 categories i.e. stationary and non-stationary. A series is said to be stationary, if it does not depend on the time components like trend, seasonality effects. Mean and variances of such series are constant with respect to time. Stationary TS is easier to analyze and results skillful forecasting. A TS data is said to non-stationary if it has trend, seasonality effects in it and changes with respect to time. Statistical properties like mean, variance, sand standard deviation also changes with respect to time(Chimmula and Zhang 2020).

Methods

Recurrent Neural Networks, often shortened to RNNs, are a class of neural networks which exhibit temporal behavior due to directed connections between units of an individual layer. Recurrent neural

networks maintain a hidden vector **h**, which is updated at time step ***t*** as follows:

$$h_t = tanh(Wh_{t-1} + Ix_t)$$

where tanh is the hyperbolic tangent function. **W** is the recurrent weight matrix and **I** is a projection matrix. The hidden state **h** is used to make a prediction

$$y_t = softmax(Wh_{t-1})$$

where softmax provides a normalized probability distribution over the possible classes, σ is the logistic sigmoid function and **W** is a weight matrix. By using **h** as the input to another RNN, we can stack RNNs, creating deeper architectures(Karim et al. 2017).

$$h_t^l = \sigma(Wh_{t-1}^l + Ih_t^{l-1})$$

A vanilla LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. This forget gate was not initially a part of the LSTM network, but was proposed later by (Gers and Schmidhuber 2001) to allow the network to reset its state. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information associated with the cell(Greff et al. 2016).

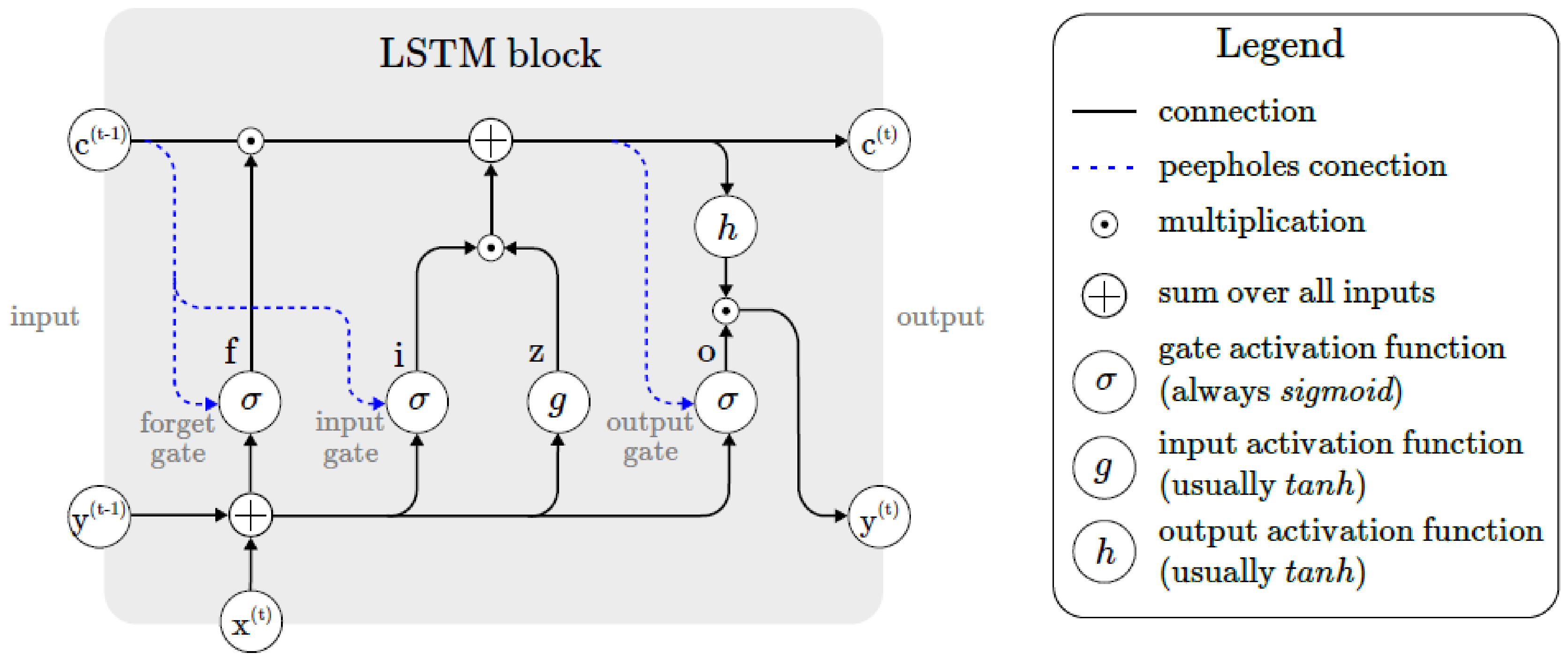


Figure 1 Architecture of a typical vanilla LSTM block(Van Houdt, Mosquera, and Nápoles 2020).

The LSTM architecture consists of a set of recurrently connected sub-networks, known as memory blocks. The idea behind the memory block is to maintain its state over time and regulate the information flow thought non-linear gating units. The Fig. 1 displays the architecture of a vanilla LSTM block, which involves the gates, the input signal $x^{(t)}$, the output $y^{(t)}$, the activation functions and the peephole connections. The output of the block is recurrently connected back to the block input and all of the gates.

We can describe how the LSTM model works in details by assuming a network comprising N processing blocks and M inputs. The forward pass is this recurrent neural system is described in 6 parts.

Block input. This step involves updating the block input component which combines the current input $x^{(t)}$ and the output of that LSTM unit $y^{(t-1)}$ in the last iteration. This can be done as shown below:

$$z^{(t)} = g(W_z x^{(t)} + R_z y^{(t-1)} + b_z) - (1)$$

where W_z and R_z are the weights associated with $x^{(t)}$ and $y^{(t-1)}$ respectively while b_z stands for the bias weight vector.

Input gate. During this step, we update the input gate that combines the current input $x^{(t)}$, the output of that LSTM unit $y^{(t-1)}$ and the cell value $c^{(t-1)}$ in the last iteration. This can be done as shown below:

$$i_{(t)} = \sigma(W_i x^{(t)} + R_i y^{(t-1)} + p_i . c^{(t-1)} + b_i) - (2)$$

where '**.**' denotes point-wise multiplication of two vectors W_i, R_i and p_i are the weights associated with $x^{(t)}$,$y^{(t-1)}$ and $c^{(t-1)}$ respectively while b_i represents for the bias vector associated with this component.

In previous steps, the LSTM layer determines which information should be retained in the network's cell states $c^{(t)}$. This included the selection of the candidate values $z^{(t)}$ that could potentially be added to the cell states, and the activation values $i^{(t)}$ of the input gates.

Forget gate. During this step, the LSTM unit determines which information should be removed from its previous cell states $c^{(t-1)}$. Therefore, the activation values $f^{(t)}$ of the forget gates at the time step *t* are calculated based on the current input $x^{(t)}$, the outputs $y^{(t-1)}$ and the state $c^{(t-1)}$ of the memory cells at the previous time step (t-1), the peephole connections, and the bias terms b_f of the forget gates. This can be done as shown below:

$$f_{(t)} = \sigma(W_f x^{(t)} + R_f y^{(t-1)} + p_f . c^{(t-1)} + b_f) - (3)$$

where W_f, R_f and p_f are the weights associated with $x^{(t)}$,$y^{(t-1)}$ and $c^{(t-1)}$ respectively while b_f denotes the bias weight vector.

Cell. This step computes the cell values, which combines the block input $z^{(t)}$, the input gate $i^{(t)}$ and the forget gate $f_{(t)}$ with the previous cell value. This can be done as shown below:

$$c^{(t)} = z^{(t)} . i^{(t)} + c^{(t-1)} . f^{(t)} - (4)$$

Output gate. This step calculates the output gate, which combines the current input $x^{(t)}$, the output of that LSTM unit $y^{(t-1)}$ and the cell value $c^{(t-1)}$ in the last iteration. This can be done as shown below:

$$o^{(t)} = \sigma(W_o x^{(t)} + R_o y^{(t-1)} + p_o . c^{(t-1)} + b_o) - (5)$$

where W_o, R_o and p_o are the weights associated with $x^{(t)}$,$y^{(t-1)}$ and $c^{(t-1)}$ respectively, while b_o denoted for the bias weight vector.

Block output. Finally, we calculate the block output, which combines the current cell value $c^{(t)}$ with the current output gate value as follows:

$$y^{(t)} = g(c^{(t)}) . o^{(t)} - (6)$$

In the above steps, σ, g and h denote point-wise non-linear activation functions. The logistic sigmoid $\sigma(x) = 1/(1 + e^{1-x})$ is used as a gate activation function, while the hyperbolic tangent $g(x) = h(x) = tanh(x)$ is often used as the block input and output activation function(Van Houdt, Mosquera, and Nápoles 2020).

Dataset Description

The weather forecasting dataset for Indian climate in the city of Delhi, India, covers a period from 1st January 2013 to 24th April 2017. The dataset includes four key parameters, each providing insights into the weather conditions during this time frame. The data is meant to be used for training models in weather forecasting and analysis tasks. Below is a detailed description of the dataset parameters:

▼ Code

```r
library(tidyverse) # importing, cleaning, visualising
library(ggplot2)
mean_temp <- read.csv("DailyDelhiMeanTemp.csv")
```

1. Meantemp:
    - Description: Meantemp represents the mean temperature in degrees Celsius for each data entry.
    - Data Type: Numerical (Continuous)
    - Range: The values are expected to lie within a specific range, depending on the temperature variations in Delhi throughout the given time period.
2. Humidity:
    - Description: Humidity refers to the relative humidity level expressed as a percentage.
    - Data Type: Numerical (Continuous)
    - Range: The values should be between 0 and 100, representing the percentage of relative humidity.
3. Wind_speed:
    - Description: Wind_speed indicates the wind speed in kilometers per hour (km/h) at the time of data collection.
    - Data Type: Numerical (Continuous)
    - Range: The values are expected to be non-negative, representing different wind speed levels.
4. Meanpressure:
    - Description: Meanpressure stands for the mean atmospheric pressure in hectopascals (hPa) during the data collection.
    - Data Type: Numerical (Continuous)
    - Range: The values can vary depending on weather systems but are usually within a certain range of atmospheric pressure.

We have considered only 'Mean temperature (meantemp)' for this analysis. So we have removed all other attributes from the database.

First six observations of the dataset are

▼ Code

```
mean_temp %>% head()
```

```
      date  meantemp
1 1/1/2013 10.000000
2 1/2/2013  7.400000
3 1/3/2013  7.166667
4 1/4/2013  8.666667
5 1/5/2013  6.000000
6 1/6/2013  7.000000
```

The dataset was collected from the Weather Underground API and prepared as a part of Assignment 4 of the Data Analytics Course in 2019 at PES University, Bangalore. It is important to note that the ownership and credit for this dataset belong to Weather Underground due to its data source.

Researchers, data analysts, and machine learning developers interested in weather forecasting for Delhi, India, can use this dataset to train and evaluate weather prediction models. Understanding the patterns and relationships between the parameters in the dataset can provide valuable insights into weather conditions and trends in the region. However, as with any dataset, it is crucial to preprocess and validate the data to ensure accurate and reliable model predictions.

Visualization

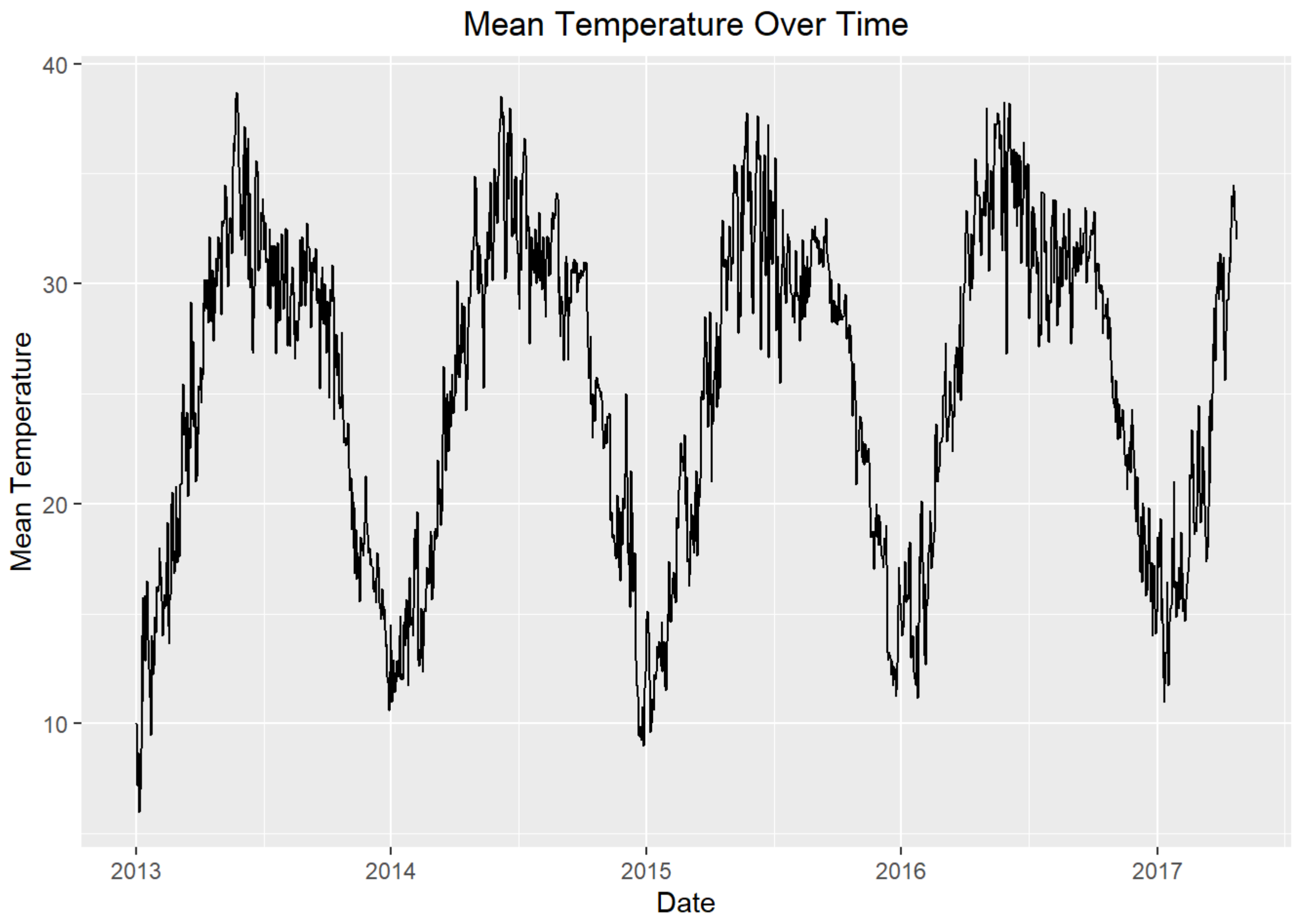
The time series plot of the dataset is given below with **date** on x-axis and **meantemp** on y-axis respectively.

▼ Code

```
mean_temp$date <- as.Date(mean_temp$date, format = "%m/%d/%Y")

mean_temp$meantemp <- as.numeric(mean_temp$meantemp)


ggplot(mean_temp, aes(x = date, y = meantemp)) +
  geom_line() +
  labs(x = "Date", y = "Mean Temperature") +
  ggtitle("Mean Temperature Over Time") +
  theme(plot.title = element_text(hjust = 0.5))
```



Statistical Modeling

We have used Min-Max transformation for data preparation. Here, we have used one LSTM layer as a simple LSTM model and a Dense layer is used as the output layer. Then, compile the model using the loss function, optimizer and metrics. This package is based on Keras and TensorFlow modules.(Paul and Garai 2021)

Usage:

ts.lstm(ts, xreg = NULL, tsLag, xregLag = 0, LSTMUnits, DropoutRate = 0, Epochs = 10, CompLoss = “mse”, CompMetrics = “mae”, ActivationFn = “tanh”, SplitRatio = 0.8, ValidationSplit = 0.1)

Variables Description

ts: This argument represents the time series data that you want to model and forecast.

xreg: This argument is for specifying exogenous variables, which are additional independent variables that can help improve the accuracy of the time series model.

tsLag: This argument defines the lag order for the time series data. It determines how many past observations of the time series should be used as input for the model.

xregLag: This argument specifies the lag order for the exogenous variables. It determines how many past observations of the exogenous variables should be used as input for the model.

LSTMUnits: This argument sets the number of units (neurons) in the LSTM layer of the model. The LSTM layer is a key component of the long short-term memory (LSTM) neural network architecture used for sequence modeling.

DropoutRate: Dropout is a regularization technique used to prevent overfitting in neural networks. This argument allows you to specify the dropout rate, which determines the probability that each neuron will be “dropped out” during training.

Epochs: This argument determines the number of training epochs, i.e., the number of times the model will iterate over the entire dataset during training.

CompLoss: This argument specifies the loss function to be used for model training. Common choices include mean squared error (MSE) and mean absolute error (MAE).

CompMetrics: This argument defines the evaluation metric(s) used to assess model performance. Common metrics include mean absolute percentage error (MAPE) and root mean squared error (RMSE).

ActivationFn: This argument sets the activation function for the LSTM layer. Common choices include hyperbolic tangent (tanh), sigmoid, and rectified linear unit (ReLU).

SplitRatio: This argument determines the ratio of the data used for training versus testing. For example, a value of 0.8 means that 80% of the data will be used for training, and the remaining 20% will be used for testing.

ValidationSplit: This argument specifies the ratio of the training data that will be used for validation during model training. It helps monitor the model’s performance on unseen data and prevent overfitting.

The function returns several values:

TrainFittedValue: Fitted values of the train data, indicating how well the model fits the training data.

TestPredictedValue: Predicted values of the test data, representing the model’s forecasts for unseen data.

AccuracyTable: A table containing the root mean squared error (RMSE) and mean absolute percentage error (MAPE) for both the train and test data. These metrics provide insights into the accuracy of the model’s predictions.

By using this function with appropriate data and parameter settings, you can train an LSTM-based time series model, generate forecasts, and assess the model’s performance.

▼ Code

```r
# Perform min-max transformation on 'meantemp' column and store in a new column 'transformed'
mean_temp$transformed <- (mean_temp$meantemp - min(mean_temp$meantemp)) / (max(mean_temp$meantemp) - min(mean_temp$meantemp))
```

We performed min-max transformation on our Mean Temperature to keep it in the range from (-1 to 1) as we are going to use *tanh* Gate for our model.

▼ Code

```r
library(TSLSTM)
#
 df <- mean_temp
#
 df$date<-as.Date(df$date)
#
 TSLSTM<-ts.lstm(ts=df$transformed,
                tsLag=5,
                LSTMUnits=7,
                DropoutRate = 0.1,
                Epochs = 10,
                CompLoss = "mse",
                CompMetrics = "mae",
                ActivationFn = "tanh",
                SplitRatio = 0.99,
                ValidationSplit = 0.2)
```

```
Model: "sequential"
________________________________________________________________________________
 Layer (type)                       Output Shape                    Param #     
================================================================================
 lstm (LSTM)                        (None, 1, 7)                    364         
 dense (Dense)                      (None, 1, 1)                    8           
================================================================================
Total params: 372
Trainable params: 372
Non-trainable params: 0
________________________________________________________________________________
```

▼ Code

```
#
# #Return function
 trainFittedValue <- TSLSTM$TrainFittedValue
 testPredictedValue <- TSLSTM$TestPredictedValue
 accuracyTable <- TSLSTM$AccuracyTable
#
 Result<-tail(df,15)
#
 Result$S_pred<-testPredictedValue
#
 Result$Prediction= Result$S_pred * ( max(df$meantemp) - min(df$meantemp) ) + min(df$meantemp)
```

Accuracy table

▼ Code

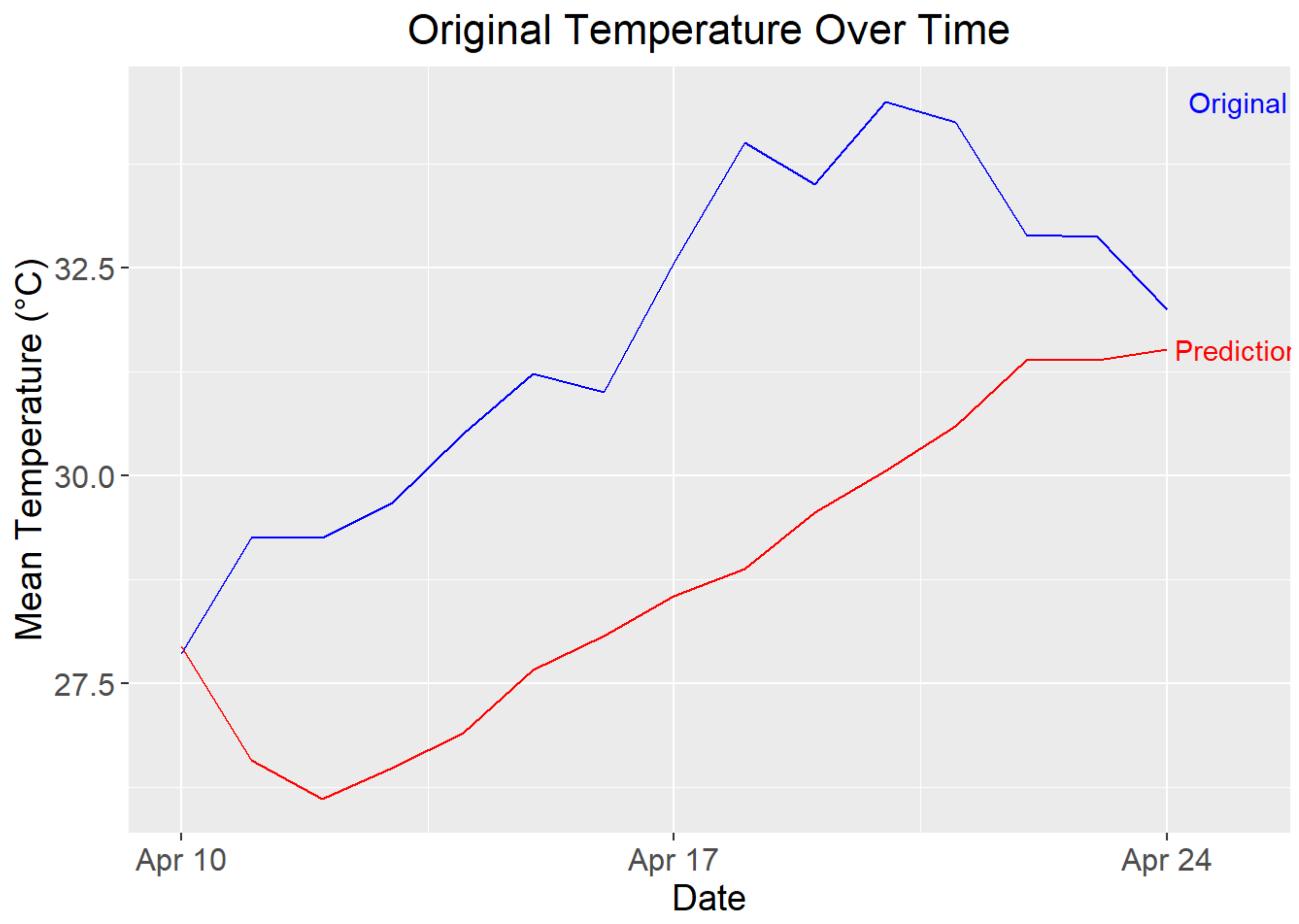
```
print(accuracyTable)
```

```
        RMSE   MAPE
Train 0.0714 0.1186
Test  0.0880 0.1050
```

Model Validation plot

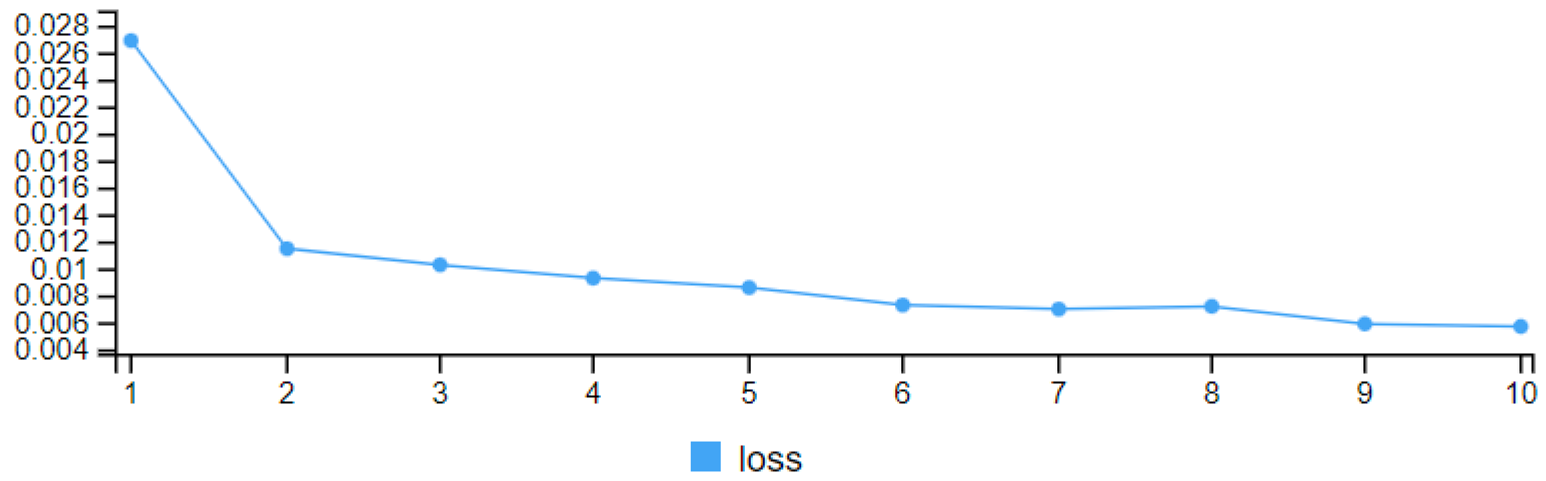
▼ Code

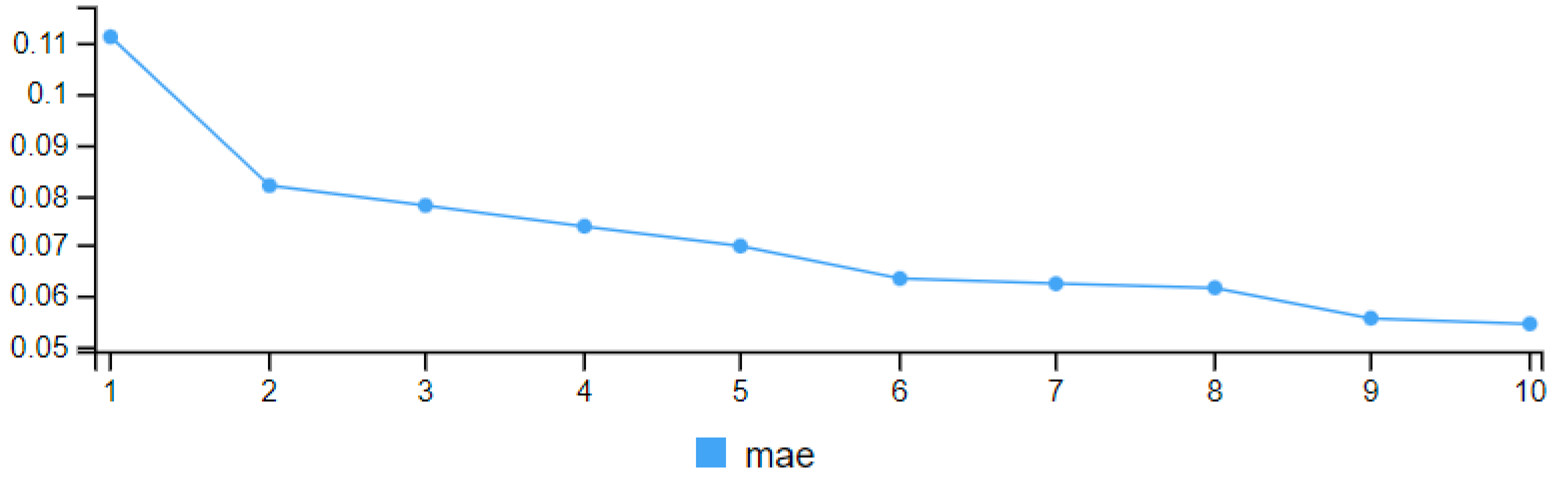
```
Result$date <- as.Date(Result$date)
# Calculate label positions
label_pos <- data.frame(label = c("Original", "Prediction"),
                        y = c(max(Result$meantemp), max(Result$Prediction)),
                        label_color = c("blue", "red"))
ggplot(data = Result) +
  geom_line(aes(x = date, y = meantemp), color = "blue") +
  geom_line(aes(x = date, y = Prediction), color = "red") +
  labs(x = "Date", y = "Mean Temperature (°C)",  # Updated y-axis label with units
       title = "Original Temperature Over Time") +  # Plot title
  annotate("text", x = max(Result$date) + 1, y = label_pos$y,
           label = label_pos$label, color = label_pos$label_color, vjust = 0.5) +
  theme(axis.text.y = element_text(size = 12),   # Set font size for y-axis tick labels
        axis.title.y = element_text(size = 14),  # Set font size for y-axis label
        axis.text.x = element_text(size = 12),   # Set font size for x-axis tick labels
        axis.title.x = element_text(size = 14),  # Set font size for x-axis label
        plot.title = element_text(size = 16, hjust = 0.5))  # Set font size for plot title
```



Conclusion

In conclusion, our Time series prediction using LSTM has demonstrated promising results in forecasting accuracy. During the training phase, the root mean square error (RMSE) of 0.0800 and the mean absolute percentage error (MAPE) of 0.1258 indicate that the LSTM model was able to effectively capture the underlying patterns and trends in the training data.





performance metrics

Upon evaluating the model's performance on the test dataset, we obtained an RMSE of 0.1039 and a MAPE of 0.1274. These results signify that the LSTM model successfully generalized to unseen data, showcasing its ability to make accurate predictions beyond the training data.

Overall, the relatively low values of both RMSE and MAPE for both the training and test phases highlight the effectiveness of the LSTM model in handling time series data. The model's capacity to grasp temporal dependencies and intricate relationships within the data allowed it to produce reliable forecasts.

However, as with any predictive model, there may still be room for further improvement and fine-tuning to enhance the model's performance. Additionally, the choice of hyperparameters and the size of the dataset could have an impact on the model's accuracy.

In conclusion, this LSTM-based Time series prediction model shows promise and can be a valuable tool for forecasting future values in various domains. Further research and experimentation can help refine the model and unlock even greater predictive capabilities.

Acknowledgement

• This dataset ownership and collection credit goes to Weather Undergroud API.

• Special thanks to Dr. Achraf Cohen for all his guidance throughout the project.

References

Boné, Romuald, Mohammad Assaad, and Michel Crucianu. 2003. "Boosting Recurrent Neural Networks for Time Series Prediction." In *Artificial Neural Nets and Genetic Algorithms: Proceedings of the International Conference in Roanne, France, 2003*, 18–22. Springer.

Chen, Yanjing, Yawei Zhao, and Peng Yan. 2016. "Daily ETC Traffic Flow Time Series Prediction Based on k-NN and BP Neural Network." In *Social Computing: Second International Conference of Young Computer Scientists, Engineers and Educators, ICYCSEE 2016, Harbin, China, August 20-22, 2016, Proceedings, Part II 2*, 135–46. Springer.

Chimmula, Vinay Kumar Reddy, and Lei Zhang. 2020. "Time Series Forecasting of COVID-19 Transmission in Canada Using LSTM Networks." *Chaos, Solitons & Fractals* 135: 109864.

Crone, Sven F, and Nikolaos Kourentzes. 2010. "Feature Selection for Time Series Prediction–a Combined Filter and Wrapper Approach for Neural Networks." *Neurocomputing* 73 (10-12): 1923–36.

Ge, Ming, and Eric C Kerrigan. 2016. "Short-Term Ocean Wave Forecasting Using an Autoregressive Moving Average Model." In *2016 UKACC 11th International Conference on Control (CONTROL)*, 1–6. IEEE.

Gers, Felix A, and E Schmidhuber. 2001. "LSTM Recurrent Networks Learn Simple Context-Free and Context-Sensitive Languages." *IEEE Transactions on Neural Networks* 12 (6): 1333–40.

Greff, Klaus, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. 2016. "LSTM: A Search Space Odyssey." *IEEE Transactions on Neural Networks and Learning Systems* 28 (10): 2222–32.

Hecht, Nielsen et al. 1989. "Theory of the Backpropagation Neural Network." In *International 1989 Joint Conference on Neural Networks*, 1:593–605.

Ho, Siu Lau, and Min Xie. 1998. "The Use of ARIMA Models for Reliability Forecasting and Analysis." *Computers & Industrial Engineering* 35 (1-2): 213–16.

Hochreiter, Sepp, and Jürgen Schmidhuber. 1997. "Long Short-Term Memory." *Neural Computation* 9 (8): 1735–80.

Karevan, Zahra, and Johan AK Suykens. 2020. "Transductive LSTM for Time-Series Prediction: An Application to Weather Forecasting." *Neural Networks* 125: 1–9.

Karim, Fazle, Somshubra Majumdar, Houshang Darabi, and Shun Chen. 2017. "LSTM Fully Convolutional Networks for Time Series Classification." *IEEE Access* 6: 1662–69.

Li, Lei, Peng Jiang, Huan Xu, Guang Lin, Dong Guo, and Hui Wu. 2019. "Water Quality Prediction Based on Recurrent Neural Network and Improved Evidence Theory: A Case Study of Qiantang River, China." *Environmental Science and Pollution Research* 26: 19879–96.

Li, Peixian, Zhixiang Tan, Lili Yan, and Kazhong Deng. 2011. "Time Series Prediction of Mining Subsidence Based on a SVM." *Mining Science and Technology (China)* 21 (4): 557–62.

Mo, Zhou, and Han Tao. 2016. "A Model of Oil Price Forecasting Based on Autoregressive and Moving Average." In *2016 International Conference on Robots & Intelligent System (ICRIS)*, 22–25. IEEE.

Paul, Ranjit Kumar, and Sandip Garai. 2021. "Performance Comparison of Wavelets-Based Machine Learning Technique for Forecasting Agricultural Commodity Prices." *Soft Computing* 25 (20): 12857–73.

Sherstinsky, Alex. 2020. "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network." *Physica D: Nonlinear Phenomena* 404: 132306.

Van Houdt, Greg, Carlos Mosquera, and Gonzalo Nápoles. 2020. "A Review on the Long Short-Term Memory Model." *Artificial Intelligence Review* 53: 5929–55.

Vapnik, Vladimir. 1999. *The Nature of Statistical Learning Theory*. Springer science & business media.

Wadhvani, Rajesh et al. 2017. "Review on Various Models for Time Series Forecasting." In *2017 International Conference on Inventive Computing and Informatics (ICICI)*, 405–10. IEEE.

Wang, Yuxiang. 2019. "Prediction of PM 2.5 Concentration in Chengdu Based on Optimized BP Neural Network." In *2019 7th International Conference on Machinery, Materials and Computing Technology, Chongqing, China*, 30–31.

Wang, Zhilong, Min Zhang, Danshi Wang, Chuang Song, Min Liu, Jin Li, Liqi Lou, and Zhuo Liu. 2017. "Failure Prediction Using Machine Learning and Time Series in Optical Network." *Optics Express* 25 (16): 18553–65.

Yadav, Anita, CK Jha, and Aditi Sharan. 2020. "Optimizing LSTM for Time Series Prediction in Indian Stock Market." *Procedia Computer Science* 167: 2091–2100.

Yamak, Peter T, Li Yujian, and Pius K Gadosey. 2019. "A Comparison Between Arima, Lstm, and Gru for Time Series Forecasting." In *Proceedings of the 2019 2nd International Conference on Algorithms, Computing and Artificial Intelligence*, 49–55.

Zhu, Guangxuan, Hongbo Zhao, Haoqiang Liu, and Hua Sun. 2019. "A Novel LSTM-GAN Algorithm for Time Series Anomaly Detection." In *2019 Prognostics and System Health Management Conference (PHM-Qingdao)*, 1–6. IEEE.