University of Kent

Nattawat Apichitpitipong
MSc Business Analytics
Student ID: 24013576

BUSN9040: Machine Learning and Forecasting
Time Series Machine Learning Model for British Temperature Forecasting

Spring Term
Academic Year 2023/2024

1. Introduction

1.1 Dataset Exploration

According to the England temperature dataset, it is found that the dataset contains time series data which has 2 columns, including 'Year", the date of recorded temperature in string format, and "AverageTemperature", the temperature in float format.

	Year	AverageTemperature
0	1752-10-01	8.860
1	1752-11-01	7.241
2	1752-12-01	5.303
3	1753-01-01	1.564
4	1753-02-01	3.641
...	...	...
3126	01/04/2013	6.635
3127	01/05/2013	9.826
3128	01/06/2013	13.285
3129	01/07/2013	17.011
3130	01/08/2013	15.776

3131 rows × 2 columns

Figure 1: The Samples of England Temperature Dataset

Figure 1 shows that the dataset contains the record of temperature from 01/10/1752 to 01/08/2013, 3131 records. The temperature was recorded in a monthly pattern, every first day of the month. Moreover, the format of dates are also different, where the date before 01/01/1900 is recorded in YYYY-MM-DD format, but the data after 01/01/1900 is recorded in DD/MM/YYYY format. The reason for the different date format might be due to the limitation of software, which most of the software are able to document the date from 01/01/1900 onwards. Plus, since the data represents the monthly temperature which typically corresponds to annual temperature cycles due to seasons, the data is considered as seasonal data.

1.2 Machine Learning Models Considerations

Since the England temperature dataset is a seasonal time series data, the Seasonal Autoregressive Integrated Moving Average (SARIMA) and Long-Short Term Memory (LSTM) are considered to be used in this project. Due to SARIMA has the ability to model and forecast the time series data that exhibit seasonal patterns (Dubey, et al., 2021), SARIMA is then chosen to be utilised in the project. As the same as the LSTM model, which is a type of recurrent neural network (RNN). The LSTMs have been shown to be more effective in handling complex patterns and dependencies over time than the traditional models, such as ARIMA and SARIMA (Dubey, et al., 2021). The LSTM is applied to compare the performance to the SARIMA model for finding the best machine learning model for forecasting the England temperature.

1.3 Analysis Tool

Due to the application of SARIMA and LSTM, which is beyond the capability of SPSS software, Python programming language is selected to conduct the analysis in this study, where Python code is run on the Jupyter Notebook.

2. Data Preparation

According to the data exploration, it found that the dates recorded in "Year" column are in a string type. In order to effectively perform further analysis, the date's data type is supposed to be changed to the proper date format. Due to the different formats of dates mentioned in data exploration, the implementation of the regular expression (Regex) is used to extract the year, month, and day from the "Year" column before combining and converting the data to the date format, as the new "date" column, as the following figures.

```
def extract_year(date):
    date = str(date)
    regex = re.match(r"(\d{4})[-/]\d{2}[-/]\d{2}", date) or re.match(r"\d{2}/\d{2}/(\d{4})", date)
    if regex:
        return regex.group(1)

def extract_month(date):
    date = str(date)
    regex = re.match(r"\d{4}[-/](\d{2})[-/]\d{2}", date) or re.match(r"\d{2}/(\d{2})/\d{4}", date)
    if regex:
        return regex.group(1)

def extract_day(date):
    date = str(date)
    regex = re.match(r"\d{4}[-/]\d{2}[-/](\d{2})", date) or re.match(r"(\d{2})/\d{2}/\d{4}", date)
    if regex:
        return regex.group(1)
```

Figure 2: Python Code for Extracting Year, Month and Day

```
df['year'] = df['Year'].apply(extract_year)
df['month'] = df['Year'].apply(extract_month)
df['day'] = df['Year'].apply(extract_day)

df['year'] = df['year'].astype(int)
df['month'] = df['month'].astype(int)
df['day'] = df['day'].astype(int)


df['date'] = pd.to_datetime(df[['year', 'month', 'day']], errors='coerce')

df
```

	Year	AverageTemperature	year	month	day	date
0	1752-10-01	8.860	1752	10	1	1752-10-01
1	1752-11-01	7.241	1752	11	1	1752-11-01
2	1752-12-01	5.303	1752	12	1	1752-12-01
3	1753-01-01	1.564	1753	1	1	1753-01-01
4	1753-02-01	3.641	1753	2	1	1753-02-01
...	...	...	...	...	...	...
3126	01/04/2013	6.635	2013	4	1	2013-04-01
3127	01/05/2013	9.826	2013	5	1	2013-05-01
3128	01/06/2013	13.285	2013	6	1	2013-06-01
3129	01/07/2013	17.011	2013	7	1	2013-07-01
3130	01/08/2013	15.776	2013	8	1	2013-08-01

3131 rows × 6 columns

Figure 3: Python Code for Converting String Data to Date Format

3. Machine Learning Analysis

The recorded temperatures are plotted in a line graph to see the overall pattern as the following.

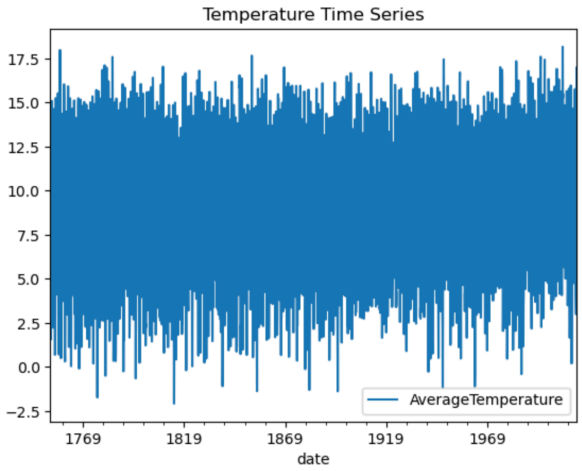

Figure 4: The Line Graph of Recorded England Temperature

Before analysing the time series data, verifying the stationarity of data is necessary since the non-stationary data could lead to unreliable results from the time series forecasting models. The Augmented Dickey-Fuller (ADF) is used to check the stationarity of the data. The ADF results are shown below.

```
rs = adfuller(df['AverageTemperature'])
print('ADF Statistic: %f' % rs[0])
print(f'p-value: {rs[1]}')
# A p-value > 0.05 suggests non-stationarity
```

```
ADF Statistic: -6.274711
p-value: 3.9262247744542974e-08
```

Figure 5: The ADF Test of Temperature Data

```
rs = adfuller(df['seasonal_difference'])
print('ADF Statistic: %f' % rs[0])
print(f'p-value: {rs[1]}')
# A p-value > 0.05 suggests non-stationarity
```

```
ADF Statistic: -16.550113
p-value: 1.9485411881739353e-29
```

Figure 6: The ADF Test after 1st Differencing

```
rs = adfuller(df['seasonal_difference_2'])
print('ADF Statistic: %f' % rs[0])
print(f'p-value: {rs[1]}')
# A p-value > 0.05 suggests non-stationarity
```

```
ADF Statistic: -20.968542
p-value: 0.0
```

Figure 7: The ADF Test after 2nd Differencing

The above figures present that the original temperature data is non-stationary, the original data achieved a 3.926 in p-value, therefore, the differencing is required to make the data stationary, achieving a p-value less than 0.05. The data has been differenced 2 times in order to achieve the target. After the first differencing, the data achieved a p-value of 1.949, which is still higher than 0.05. The data accomplished the target in the second differencing, in which the data achieved a 0.000 p-value, meaning the data is now stationary and ready for being applied to the time series forecasting model.

The Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) are plotted in the following.

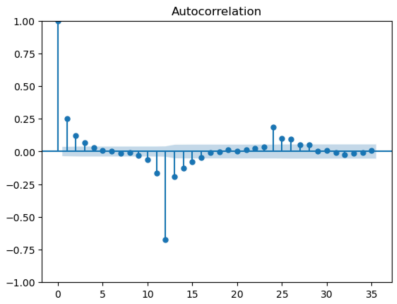

Figure 8: The Final ACF Plot

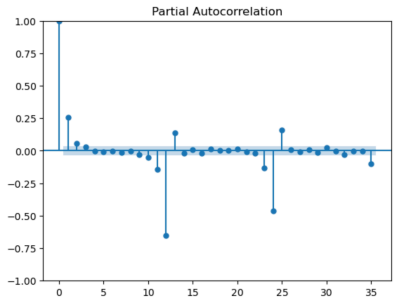

Figure 9: The Final PACF Plot

The differenced data is used for conducting the time series forecasting in both SARIMA and LSTM machine learning models.

3.1 Seasonal Autoregressive Integrated Moving Average (SARIMA)

3.1.1 Model Settings

To maximise the SARIMA performance, the settings of *p, d, q, P, D, Q, and m* are required. *p, d and q* are the non-seasonal components, which represent autoregression, differences, and moving average respectively. *P, D and Q* are the same components, but for seasonal components. *m* which represents the number of periods in each season is set to 12 based on the yearly temperature cycle. To find the best settings for SARIMA, the grid search, auto Arima model, has been used to find the right value of the mentioned parameters. The result of the grid search displays that the best parameters are $p = 1$, $d = 0$, $q = 1$, $P = 2$, $D = 0$, and $Q = 0$.

.

```
# Conduct grid search to find the right value of (p, d, q) (P, D, Q, m) in SARIMA m
model = auto_arima(df['seasonal_difference_2'], seasonal=True, m=12,
                   stepwise=True, suppress_warnings=True,
                   error_action="ignore", max_order=None, trace=True)

print(model.summary())
```

Figure 10: The Python for SARIMA Grid Search

```
Best model:  ARIMA(1,0,1)(2,0,0)[12]
Total fit time: 183.487 seconds
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                                 y   No. Observations:                 3107
Model:             SARIMAX(1, 0, 1)x(2, 0, [], 12)   Log Likelihood               -6344.545
Date:                             Mon, 06 May 2024   AIC                          12699.089
Time:                                     21:27:50   BIC                          12729.296
Sample:                                          0   HQIC                         12709.935
                                              - 3107
Covariance Type:                               opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.4220      0.064      6.634      0.000       0.297       0.547
ma.L1         -0.1863      0.068     -2.742      0.006      -0.320      -0.053
ar.S.L12      -1.0058      0.014    -72.210      0.000      -1.033      -0.978
ar.S.L24      -0.5009      0.014    -35.150      0.000      -0.529      -0.473
sigma2         3.4611      0.077     45.061      0.000       3.311       3.612
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                52.42
Prob(Q):                              0.97   Prob(JB):                         0.00
Heteroskedasticity (H):               0.82   Skew:                            -0.06
Prob(H) (two-sided):                  0.00   Kurtosis:                         3.63
===================================================================================
```

Figure 11: The Results of SARIMA Grid Search

3.1.2 Model Analysis and Results

The SARIMA model has been fit to the adjusted temperature data by the following Python code.

```
# Creating a SARIMAX Model
model = SARIMAX(df['seasonal_difference_2'],
                order=(1, 0, 1),
                seasonal_order=(2, 0, 0, 12))

# Fitting the model
fitted_model = model.fit()

# Forecasting future values
forecast_result = fitted_model.get_forecast(steps=12)

# Forecasted mean
forecast_mean = forecast_result.predicted_mean

# Confidence intervals
confidence_intervals = forecast_result.conf_int()
```

Figure 12: The Python Code for Fitting SARIMA Model

The predictions of the model have been plotted and shown in the following figures. Regarding the prediction plot, the plot presents the data from 01/01/2000 onwards to scope down to the similar patterns between the records and the predictions. The records are coloured in blue, while red represents the predictions. The pink area displayed in predictions represents the confidence interval of the forecasts provided by the SARIMA model.

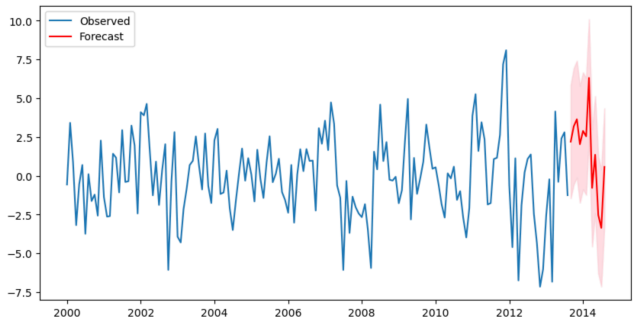

Figure 13: The Prediction Plot of the SARIMA Model

The predictions of the SARIMA are shown in Figure 13, which are in the differenced forms. To find the pure predicted temperature of the forecast, the predicted values have to be calculated inverted. After the inverted calculations, the predicted temperatures and the inverted confidence intervals are plotted and presented in Figure 14 and Figure 15 respectively.

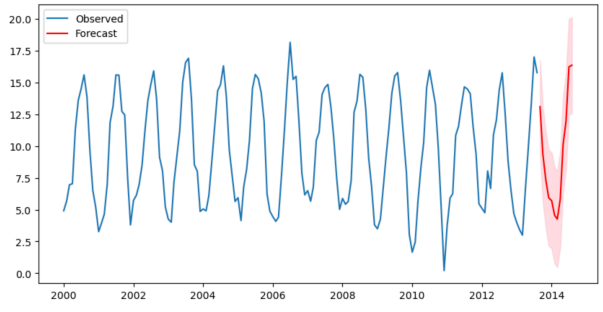

Figure 14: The Predicted Temperature of the SARIMA Model

	PredictedTemperature	LowerPredictedTemperature	UpperPredictedTemperature
2013-09-01	13.098734	9.452388	16.745080
2013-10-01	9.334574	5.588345	13.080802
2013-11-01	7.327655	3.563918	11.091391
2013-12-01	5.933832	2.166986	9.700678
2014-01-01	5.707912	1.940513	9.475311
2014-02-01	4.551422	0.783925	8.318920
2014-03-01	4.253193	0.485678	8.020708
2014-04-01	5.819235	2.051717	9.586753
2014-05-01	10.120767	6.353248	13.888286
2014-06-01	11.986035	8.218516	15.753554
2014-07-01	16.227401	12.459882	19.994920
2014-08-01	16.363047	12.595529	20.130566

Figure 15: The Predicted Temperature, Lower and Upper Confidence Intervals

The Root Mean Squared Error (RMSE) of the model is 1.868 according to Figure 16.

```
# Calculate the RMSE
rmse = np.sqrt(np.mean(fitted_model.resid**2))
print(f'RMSE: {rmse}')
```

```
RMSE: 1.8679498186709216
```

Figure 16: The RMSE of the SARIMA Model

3.2 Long-Short Term Memory (LSTM)

3.2.1 Model Settings

In the LSTM model, several parameters need to be adjusted, including the unit of neurons, dense layer, batch sizes, epochs, dropout rate, etc. In this study, the unit of neurons, dense layers, batch sizes, and epochs are mentioned and adjusted to achieve the best LSTM model. The number of neuron units could affect to the capacity of the model to memorise and learn from the data. The larger number of the units the more overfitting the LSTM model is. In the same way, the smaller number of the more underfitting of the LSTM model is. As the same as the number of epochs, which represents how many times the learning algorithm will work through the entire training dataset. Both units of neurons and epochs have no criteria of number that explicitly determine the overfitting and underfitting model, it also depends on the size of data and other factors. In this study, the units of neurons and epochs are set to 50 as the beginning value as the below figure. Both parameters are adjusted according to the performance after running each scenario. Unlike the dense layer, which is set to 1, since the model is applied for predicting a future single value of time series data. As well as the batch size which is set to 1 due to the avoidance of struggling of the model during a training session. The Python code of model's setting is presented in the below figure.

```
# Build the LSTM model
model = Sequential()
model.add(LSTM(50, input_shape=(look_back, 1)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')

# Fit the model
model.fit(X_train, y_train, epochs=50, batch_size=1, verbose=2)
```

Figure 17: The Python Code for Creating the LSTM Model

3.2.2 Model Analysis and Results

The RMSE is used to verify the performance of the model. The calculation of the RMSE is conducted by the following Python code.

```
from sklearn.metrics import mean_squared_error
import math

# Calculate RMSE of 50 epochs and 50 units
testScore = math.sqrt(mean_squared_error(y_test_rescaled[0], test_predict[:,0]))
print('Test RMSE: %f' % (testScore))
```

```
Test RMSE: 2.151407
```

Figure 18: The Python Code for RMSE Calculations

The model is set to have 5 scenarios, which each scenario has a different setting of the units of neurons and epochs according to the performance, the RMSE of the model. The summary table of each scenario's settings is displayed in the following.

Scenario	Units of Neurons	Epochs	RMSE
Scenario 1	50	50	2.151
Scenario 2	50	100	2.076
Scenario 3	30	100	1.996
Scenario 4	20	100	2.004
Scenario 5	70	100	1.986

Table 1: The Parameters Setting in 5 Scenarios

Referring to the above table, it shows that the increasing of epochs in scenario 2 decreases RMSE from 2.151 to 2.076, approximately 3.487%. The adjustment strongly positively impacts to the RMSE. In the scenario 3, the number of neurons is decreased to 30. The result also shows the less error, achieving 1.996 RSME. Nevertheless, after continuous decreasing the neurons to 20, the RMSE slightly increase to 2.004 in the scenario 4. In the last scenario, scenario 5, achieve the best performance compared to other scenarios, which achieve 1.986 in RMSE. The unit of the neuron is set to 70 with 100 epochs in scenario 5. The increase of neurons is stopped at scenario 5 to avoid the overfitting of the LSTM model. Subsequently, the last scenario 5 is chosen to be the final model of the study, which the graph of the predicted temperature is plotted below.

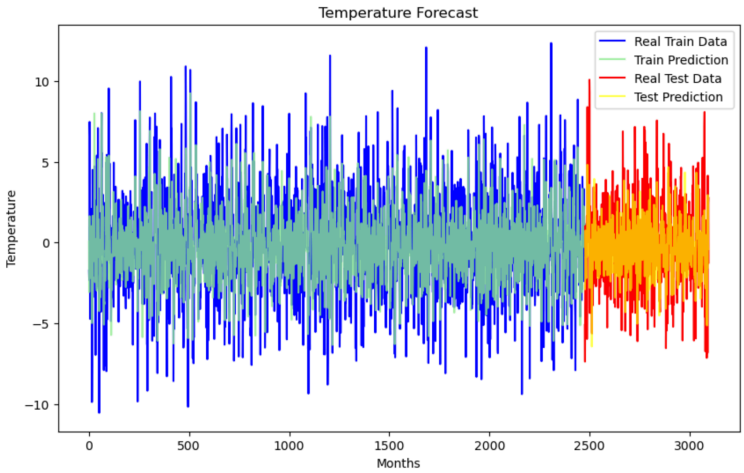

Figure 19: The Predicted Temperature Plot of the LSTM Model

Regarding the predicted temperature plot, the data is plotted and painted with 4 different colours according to the 4 categories of data, including training data which painted in blue colour, predictions of training data that is green colours, testing data coloured in red, and predictions of test data, which is painted in yellow.

4. Conclusion and Discussion

4.1 Conclusion

After training, testing and experimenting to find the best performance of both the SARIMA and LSTM models, the findings of the study discovered that the SARIMA model has a better performance than the LSTM model in forecasting the England temperature in monthly cadence, with the RMSE of 1.868, less than the RMSE of LSTM model which is 1.986, 0.118 units or 5.942% better performance.

The SARIMA model also provides an outcome which is easier to understand than the LSTM. The SARIMA's output explicitly presents the predicted values in a continuous pattern as shown in Figure 13. Furthermore, the model also displays the confidence intervals of the forecasts, which helps the reader to understand the upper and lower boundaries of the forecast.

4.2 Discussion

Even though the result of this study concludes that the SARIMA model achieves better performance than the LSTM model, it does not convey that the SARIMA is completely more effective than LSTM. Since the adjustment of LSTM's parameter has a wider range than the SARIMA, there might be other settings of the LSTM which could achieve better performance of the forecast of monthly England temperature. However, unlike the SARIMA where the best settings could be found by applying the auto Arima model, the LSTM has to randomly indicate the number or unit of the parameter manually. This could be a challenge for a future study of the LSTM model.

The result also does not draw the assumption that the SARIMA model is the best time series forecasting model to predict the temperature. The study of performance comparison between the SARIMA model and other time series forecasting models, such as Random Forest, Support Vector Regression, Vector Autoregression, etc., are needed to test and verify in future studies.

5. Reference

Dubey, A. K. et al., 2021. Study and analysis of SARIMA and LSTM in forecasting time series data. *Sustainable Energy Technologies and Assessments,* Volume 47.