# **Climate Change Prediction using LSTM**

# **Summary:**

# This Document is going to describe to you about time series prediction model using a Long Short-Term Memory (LSTM) neural network. The dataset used for training and testing is South Dakota climate data, including features like average temperature, minimum temperature, maximum temperature, precipitation, and heating degree days.

# Here's a detailed breakdown how I implemented the whole code and the procedure:

# **1. Data Loading and Preprocessing:**

# The South Dakota climate data is loaded from a CSV file.

# The selected columns are normalized using MinMaxScaler to transform values between 0 and 1, as LSTM networks are sensitive to magnitude.

# The dataset is split into training and testing sets (90% training, 10% testing).

# Sequences are created using the `to\_sequences` function, which takes a sliding window approach to create input-output pairs for training the LSTM model.

**Parameters:**

* **seq\_size:** The number of time steps to look back. In this case, it is set to 6.
* **step\_size:** The number of steps to advance the window for the next sequence. It is set to 1, meaning each sequence is created by shifting the window by one time step.
* **target\_size:** The number of values to predict in the output sequence. Here, it is also set to 6.

**2. LSTM Model Architecture:**

* The model is built using the Sequential API from Keras.
* It consists of three LSTM layers with dropout to prevent overfitting.
* The final layer is a Dense layer with 5 units, matching the number of features in the dataset.
* The model is compiled using the mean squared error loss function and the Adam optimizer.

# **3. Model Training:**

# The model is trained using the training data with a validation split of 10%.

# Early stopping is implemented to monitor the validation loss and stop training if it does not improve for a certain number of epochs.

* Here the first 90% of the data (i.e., 1390 out of 1545) is used for training “train variable” and the remaining 10% (i.e., 155 out of 1545) is used for testing “test variable.”
* The to\_sequences function creates sequences of input-output pairs for the LSTM model. It uses a sliding window approach to extract sequences from the original time series data. For example, if seq\_size is 6, it looks at six consecutive time steps as input (trainX) and predicts the next 6-time steps (trainY).
* The resulting shapes for trainX, trainY, testX, and testY are as follows:
  + trainX. Shape: (1379, 6, 5) - 1379 sequences, each with 6-time steps and 5 features.
  + trainY. Shape: (1379, 6, 5) - Corresponding target values for trainX.
  + testX. Shape: (144, 6, 5) - 144 sequences for testing.
  + testy. Shape: (144, 6, 5) - Corresponding target values for testX.
* These sequences are then used to train the LSTM model, allowing it to learn patterns in the data and make predictions for future time steps.

# **4. Model Evaluation:**

# The training and validation losses are plotted over epochs to visualize the model's performance during training.

# The trained model is used to make predictions on both the training and testing sets.

# Predictions are inverse transformed to the original scale using the MinMaxScaler.

# Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE) are calculated to evaluate the model's performance.

# **5. Accuracy Calculation (Binary Classification):**

# The predictions are binarized using a threshold of 0.5 for both training and testing sets.

# Accuracy is calculated using scikit-learn accuracy score function.

* For training the Model has got an accuracy score of 94.25% and for testing it has achieved an accuracy score of 95.13%

# **6. Results**

# The RMSE for both training and testing sets is around 0.08, indicating relatively good performance.

# MAE is approximately 0.057 for the testing set.

# MAPE is calculated, and the accuracy for binary classification on both training and testing sets is reported.

# The above procedure demonstrates the whole process of loading, preprocessing, training, and evaluating an LSTM model for time series prediction using South Dakota climate data. The model performs well in terms of accuracy and error metrics there will be increase in the accuracy if the data columns are more in future, I try to implement this with more data so that it can be more efficient.

**### Example Setup:**

* Sequence Length (`seq\_size`): 6
* Target Size (`target\_size`): 6
* Number of Features (`num\_features`): 5
* Step Size (`step\_size`): 6

Here is the sample dataset which comprises of 12 fields with 5 features each.

[

[1.1, 1.2, 1.3, 1.4, 1.5],

[2.1, 2.2, 2.3, 2.4, 2.5],

[3.1, 3.2, 3.3, 3.4, 3.5],

[4.1, 4.2, 4.3, 4.4, 4.5],

[5.1, 5.2, 5.3, 5.4, 5.5],

[6.1, 6.2, 6.3, 6.4, 6.5],

[7.1, 7.2, 7.3, 7.4, 7.5],

[8.1, 8.2, 8.3, 8.4, 8.5],

[9.1, 9.2, 9.3, 9.4, 9.5],

[10.1, 10.2, 10.3, 10.4, 10.5],

[11.1, 11.2, 11.3, 11.4, 11.5],

[12.1, 12.2, 12.3, 12.4, 12.5]

]

**Iterations:**

**Iteration 1:**

**Input (trainX):**

[

[1.1, 1.2, 1.3, 1.4, 1.5],

[2.1, 2.2, 2.3, 2.4, 2.5],

[3.1, 3.2, 3.3, 3.4, 3.5],

[4.1, 4.2, 4.3, 4.4, 4.5],

[5.1, 5.2, 5.3, 5.4, 5.5],

[6.1, 6.2, 6.3, 6.4, 6.5]

]

**Output (trainY):**

[

[7.1, 7.2, 7.3, 7.4, 7.5],

[8.1, 8.2, 8.3, 8.4, 8.5],

[9.1, 9.2, 9.3, 9.4, 9.5],

[10.1, 10.2, 10.3, 10.4, 10.5],

[11.1, 11.2, 11.3, 11.4, 11.5],

[12.1, 12.2, 12.3, 12.4, 12.5]

]

In this iteration, the model takes the first 6 elements as input and is trained to predict the next 6 elements.

**#### Iteration 2:**

**Input (trainX):**

[

[7.1, 7.2, 7.3, 7.4, 7.5],

[8.1, 8.2, 8.3, 8.4, 8.5],

[9.1, 9.2, 9.3, 9.4, 9.5],

[10.1, 10.2, 10.3, 10.4, 10.5],

[11.1, 11.2, 11.3, 11.4, 11.5],

[12.1, 12.2, 12.3, 12.4, 12.5]

]

**Output (trainY):**

# Next 6 elements after the previous input

[

[13.1, 13.2, 13.3, 13.4, 13.5],

[14.1, 14.2, 14.3, 14.4, 14.5],

[15.1, 15.2, 15.3, 15.4, 15.5],

[16.1, 16.2, 16.3, 16.4, 16.5],

[17.1, 17.2, 17.3, 17.4, 17.5],

[18.1, 18.2, 18.3, 18.4, 18.5]

]

Evaluation Metrics for LSTM & Transformer: (Reference paper has an accuracy of 98.67%)

|  |  |  |
| --- | --- | --- |
| **Evaluation Metric** | **LSTM Values** | **Transformer Values** |
| Mean Squared Error (MSE) | 0.006887497 | 0.0075875786 |
| Root Mean Squared Error (RMSE) | 0.08 | 0.09 |
| Mean Absolute Error (MAE) | 0.05782311 | 0.060120333 |
| Accuracy | 0.9488425925925926 | 0.9460648148148149 |
| Precision | 0.9897901304594441 | 0.9734953064605191 |
| Recall | 0.8957905544147844 | 0.9050308008213552 |
| F1 Score | 0.9404473187819994 | 0.9380154296355413 |
| R-squared (R2) | 0.9045726220852164 | 0.894872877615476 |