Daily Climate Forecasting

Hamidreza Mafi
610399209
School of Computer Science,
College of Science,
University of Tehran

July 10, 2024

1 Introduction

The objective of this project is to forecast mean atmospheric pressure using historical climate data from Delhi. This involves applying and enhancing machine learning techniques, particularly focusing on neural network architectures tailored for time-series prediction, such as Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM) networks, and Gated Recurrent Unit (GRU) networks.

2 Methodology

2.0.1 Outlier Detection and Removal

The *meanpressure* feature contained outliers that could potentially skew the results. These outliers were removed to ensure the data's integrity. The plots below show the distribution of *meanpressure* before and after outlier removal.

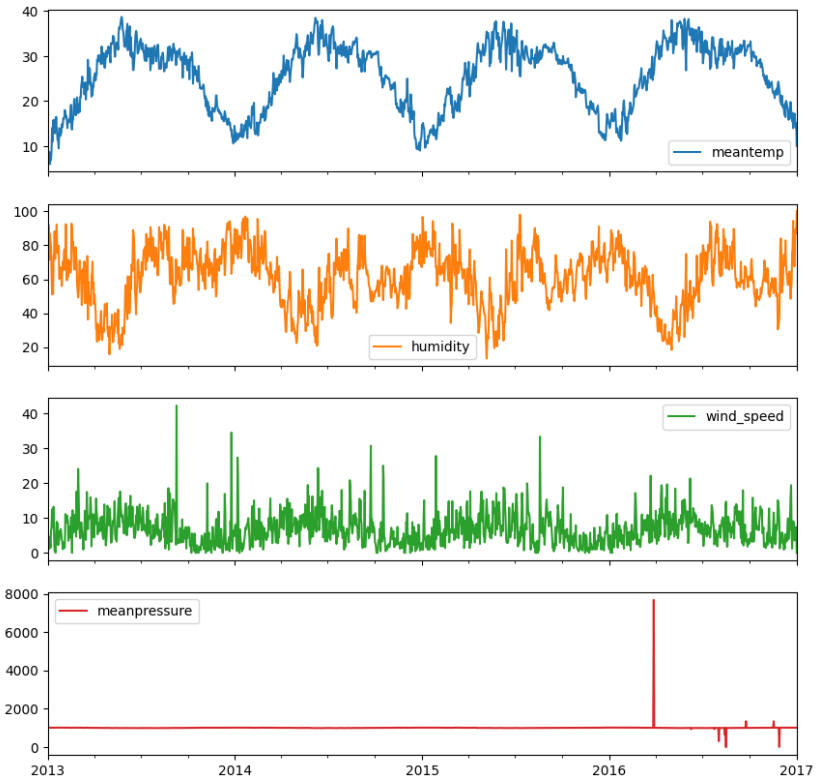

Figure 1: Mean Pressure before Outlier Removal

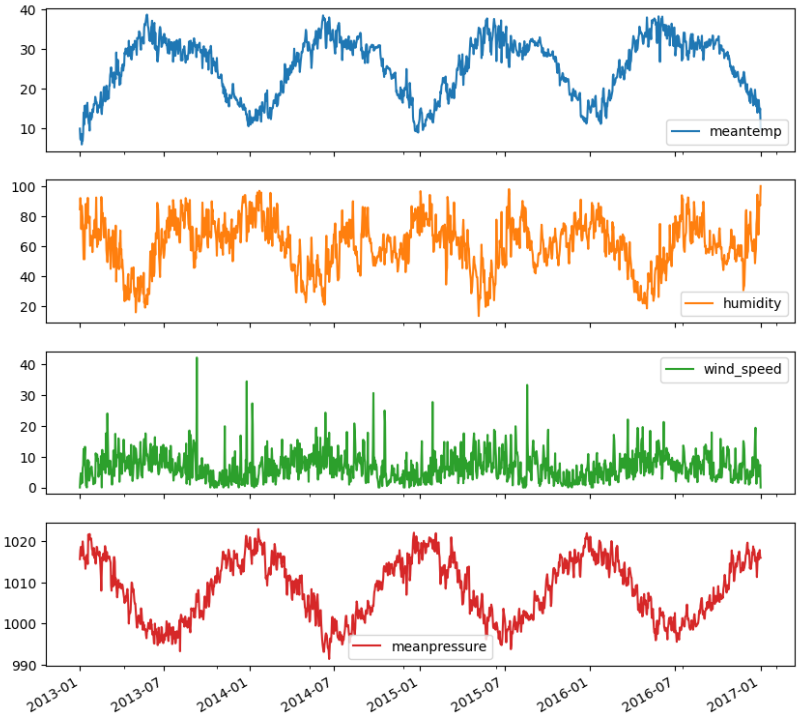

Figure 2: Mean Pressure after Outlier Removal

2.1 Data Scaling and Sequence Splitting

To prepare the data for training time-series forecasting models, we performed data scaling and sequence splitting.

2.1.1 Data Scaling

We used the `MinMaxScaler` from `sklearn.preprocessing` to scale the combined data (both training and testing data). Scaling ensures that all features are within a specific range, typically between 0 and 1, which is crucial for improving the performance and convergence speed of many machine learning algorithms. After scaling, the data was split back into training and test sets:

2.1.2 Sequence Splitting

To convert the time-series data into a supervised learning problem, we implemented a sequence splitting function. This function transforms the univariate or multivariate time series into input-output pairs suitable for model training.

The function *split sequence* works as follows:

- It iterates over the sequence and creates input-output pairs.
- For each position `i`, it creates an input sequence `seq_x` consisting of `n_steps` consecutive data points and an output `seq_y` which is the next value in the sequence.
- The function returns arrays `X` (inputs) and `y` (outputs) which can be used to train time-series forecasting models.

2.2 Model Selection and Training

Different models were trained and evaluated using appropriate evaluation metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared. Advanced models like RNN, LSTM, and GRU were utilized, and hyperparameters were tuned to optimize performance.

2.3 Model Development and Fine-Tuning

The first model we developed is a Long Short-Term Memory (LSTM) network implemented using Keras. Below is the architecture and the fine-tuning process for the model.

2.3.1 LSTM Model Architecture

The model consists of the following layers:

- **First LSTM Layer:** This layer has 128 units and processes the input sequences. The `input_shape` parameter specifies the shape of the input data. The `return_sequences` parameter is set to `True` to return the full sequence of outputs.

- **Dropout Layer:** A dropout layer with a rate of 0.2 is added to prevent overfitting by randomly setting 20% of the input units to 0 at each update during training.
- **Second LSTM Layer:** Another LSTM layer with 128 units processes the output of the previous layer. This layer does not return sequences, allowing it to output a single vector.
- **Dropout Layer:** Another dropout layer with a rate of 0.2 is added to further prevent overfitting.
- **Dense Layer:** A dense (fully connected) layer with a single unit is added at the end to produce the final output.

The model is compiled using the Adam optimizer with a learning rate of 0.01, and the loss function used is Mean Squared Error (MSE). The Adam optimizer is chosen for its efficient handling of sparse gradients and adaptive learning rate.

Model Training :

The model is trained using the training data with the following parameters:

- **Epochs:** The number of epochs is set to 50. Each epoch represents one complete pass through the training dataset.
- **Batch Size:** The batch size is set to 25, meaning that the model updates its weights after every 25 samples.

During training, the model's performance is monitored, and adjustments are made to the hyperparameters such as the learning rate, number of epochs, and batch size to improve the model's accuracy and prevent overfitting. The `history` object stores the training metrics, which can be used to analyze the model's performance over time.

Model: "sequential_4"

Layer (type)	Output Shape	Param #
lstm_8 (LSTM)	(None, 30, 128)	68,096
dropout_8 (Dropout)	(None, 30, 128)	0
lstm_9 (LSTM)	(None, 128)	131,584
dropout_9 (Dropout)	(None, 128)	0
dense_4 (Dense)	(None, 1)	129

Total params: 599,429 (2.29 MB)
Trainable params: 199,809 (780.50 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 399,620 (1.52 MB)

Figure 3: LSTM summary

2.4 GRU Model Architecture

GRU Layers:
The GRU layers are recurrent layers that are used to process sequential data. Each GRU layer has the following parameters:

- **units**: The number of neurons in the GRU layer. In this model, each GRU layer has 50 units.
- **return_sequences**: When set to `True`, each GRU layer returns the full sequence of outputs for each input sequence. This is necessary for stacking multiple GRU layers.
- **input_shape**: The shape of the input data. This is only specified for the first GRU layer.
- **activation**: The activation function used in the GRU cells. Here, the `tanh` activation function is used.

Dropout Layers:
The dropout layers are used to prevent overfitting by randomly setting a fraction of input units to 0 at each update during training time.

- **rate**: The fraction of the input units to drop. In this model, a dropout rate of 0.2 is used.

Dense Layer:
The dense layer is the output layer of the model.

- **units**: The number of neurons in the dense layer. For regression tasks, this is typically set to 1.

Optimizer:
The optimizer used for training the model is the Stochastic Gradient Descent (SGD) optimizer. The parameters for the optimizer are:

- **learning_rate**: The step size at each iteration while moving toward a minimum of the loss function. Here, it is set to 0.01.
- **decay**: The learning rate decay over each update. This helps in reducing the learning rate over time.
- **momentum**: The momentum factor used to accelerate SGD in the relevant direction and dampen oscillations.
- **nesterov**: Whether to apply Nesterov momentum.

Loss Function:
The loss function used is the Mean Squared Error (MSE), which is commonly used for regression tasks.

Model Training:
The model is trained on the training dataset (`X_train` and `y_train`) for 50 epochs with a batch size of 25. The number of epochs determines how many times the learning algorithm will work through the entire training dataset. The batch size is the number of samples that will be propagated through the network at a time.

Layer (type)	Output Shape	Param #
gru_36 (GRU)	(None, 30, 50)	8,400
dropout_50 (Dropout)	(None, 30, 50)	0
gru_37 (GRU)	(None, 30, 50)	15,300
dropout_51 (Dropout)	(None, 30, 50)	0
gru_38 (GRU)	(None, 30, 50)	15,300
dropout_52 (Dropout)	(None, 30, 50)	0
gru_39 (GRU)	(None, 50)	15,300
dropout_53 (Dropout)	(None, 50)	0
dense_15 (Dense)	(None, 1)	51

Total params: 108,704 (424.63 KB)
Trainable params: 54,351 (212.31 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 54,353 (212.32 KB)

Figure 4: GRU summary

3 Results

Model	MAE	MSE	RMSE	R-squared
LSTM (50 epochs, batch size 25)	3.1681	15.3635	3.9196	0.4457
GRU (50 epochs, batch size 25)	2.3153	8.8274	2.9711	0.6815
LSTM (100 epochs, batch size 20)	2.2114	8.1232	2.8501	0.7069
GRU (100 epochs, batch size 20)	2.3746	9.0490	3.0082	0.6735

Table 1: Performance metrics of LSTM and GRU models for different training configurations.

3.1 Discussion

The performance of the LSTM and GRU models was compared under different training configurations, and the strengths and weaknesses of each approach were analyzed. Key observations include:

- **LSTM Model (50 epochs, batch size 25):** The LSTM model achieved a MAE of 3.1681, MSE of 15.3635, RMSE of 3.9196, and an R-squared value of 0.4457. While the LSTM model captured the overall trend in atmospheric pressure, it was less accurate than the GRU model.

- **GRU Model (50 epochs, batch size 25):** The GRU model outperformed the LSTM model with a MAE of 2.3153, MSE of 8.8274, RMSE of 2.9711, and an R-squared value of 0.6815. The GRU's performance indicates better handling of the sequential nature of the data, leading to more accurate predictions.

- **LSTM Model (100 epochs, batch size 20):** With an increased number of epochs and a smaller batch size, the LSTM model showed significant improvement, achieving a MAE of 2.2114, MSE of 8.1232, RMSE of 2.8501, and an R-squared value of 0.7069. This configuration resulted in better performance compared to the previous LSTM configuration.

- **GRU Model (100 epochs, batch size 20):** Similarly, the GRU model with 100 epochs and a batch size of 20 yielded a MAE of 2.3746, MSE of 9.0490, RMSE of 3.0082, and an R-squared value of 0.6735. This performance, while slightly improved in some metrics, did not surpass the LSTM model under the same configuration.

4 Conclusion

In this project, we developed and evaluated predictive models for forecasting atmospheric pressure using historical climate data. The GRU model initially demonstrated superior performance over the LSTM model; however, with increased training epochs and a smaller batch size, the LSTM model surpassed the GRU model in accuracy and predictive power. This highlights the importance of model selection and hyperparameter tuning in time-series prediction tasks.