Comparison of Random Forest and LSTM models on West Lafayette Climate Data

Karim El-Sharkawy

January 29, 2025

1 Introduction

The dataset I chose to use is the National Oceanic and Atmospheric Administration's (NOAA) Local Climatological Data (LCD) for West Lafayette, which is publicly accessible for free. Please refer to the .ipynb file for all the code and follow the instructions below to obtain the data yourself. I selected this dataset because: 1) it aligns with my future research interests, and 2) it provided opportunities to demonstrate the knowledge I've gained in this course. This dataset includes time-series data, and as a member of the climate data community, I understand that it requires more nuanced handling than simply removing missing values or calculating column averages.

My process involved acquiring weather data for West Lafayette from 2020 to 2024, followed by preprocessing and specialized cleaning. I then tested two models: a Random Forest and a Long Short-Term Memory (LSTM) network, which is a type of Recurrent Neural Network. Afterward, I plotted the models' losses and processing times on a 2D graph and interpreted the results. These analyses were conducted after multiple trials to fine-tune the parameters for both models, ensuring minimal underfitting or overfitting. For the feature to predict, I selected "Hourly Wet Bulb Temperature." The choice of West Lafayette and the specific time range wasn't particularly significant. Additionally, while I use "weather" and "climate" interchangeably in this project, I recognize that they are distinct concepts, though in this context, I believe it poses no issue. Finally, I am writing this document in LaTeX and I am unable to show code within a normal paragraph which may lead to underscores being removed from the code.

1.1 Data Access Instructions

To access the data you need, start by visiting NOAA's LCD Site. Begin by selecting your preferred location type and entering the relevant information; for instance, you can use the zip code 47906 to search for West Lafayette. Once you click on the West Lafayette Station, add the desired data to your cart—this service is free. To review your selections, click on the orange cart icon located on the right-hand side of the page.

Next, choose your preferred format for the data; in this case, CSV is the recommended option. Specify your date range, ensuring it covers less than ten years—my chosen range was from January 1st, 2020, to January 1st, 2024. After that, press continue and follow the steps provided on the side pages, including entering your email address. Once everything is completed, the data will be sent to your email, allowing you to download it via the link provided.

2 Metadata

The data, at least for West Lafayette, is updated daily. The Local Climatological Data (LCD) documentation provides detailed weather summaries for specific locations, including metadata on various climatic parameters. The station information includes location details like latitude, longitude, and elevation. Time data is reported in Local Standard Time (LST), using a 24-hour format without adjustments for Daylight Savings Time. Temperature data includes daily maximum, minimum, and average values in Fahrenheit, along with departures from the 1981-2010 normal temperatures. Humidity and dew point values are also provided, along with wet-bulb temperatures, heating and cooling degree days (base 65°F), and times for sunrise and sunset.

The dataset tracks various weather types using WT codes, which cover a wide range of events such as fog, thunderstorms, rain, and snow. Precipitation data includes total liquid content, snowfall, and snow depth. Wind information consists of daily averages, peak speeds, and directions. Atmospheric pressure is recorded both at the station level and adjusted to sea level, and extreme temperature and pressure events are highlighted in the monthly summaries. Hourly observations provide details on sky conditions, visibility, temperature (dry bulb and wet bulb), humidity, wind, and hourly precipitation amounts. The data is available in CSV or PDF formats, making it suitable for further analysis in spreadsheets or databases (which is how we're working with it in this project). Special indicators like "T" for trace amounts and "M" for missing values are used throughout the dataset to mark specific conditions. There is more information in the Dataset documentation and the code explains how these values are handled.

3 Procedure

3.1 Data Preparation

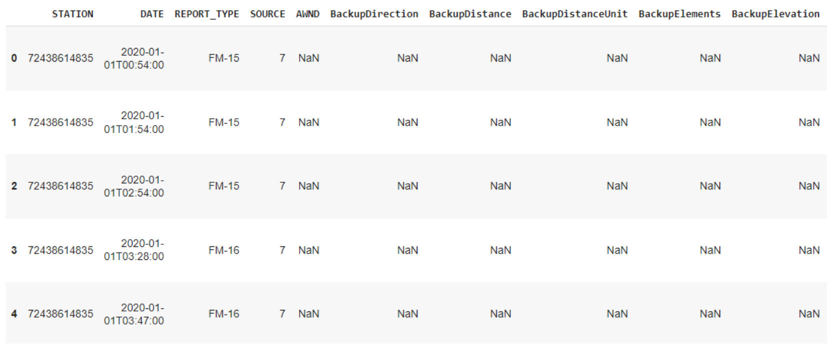
The code begins by setting up the environment necessary for working with climate data. Climate data needs to be handled carefully as it is time-series data and the documentation is even more important because weather data from two different places may be using different notations. I import the two core libraries, 'PANDAS', and 'numpy'. I only use the 'os' module to manage file paths at the beginning. However, pandas is central to the work at the beginning because It allows me to load, manipulate, and analyze the dataset efficiently and then clean/manipulate it properly. I use numpy for general numerical computations, and its array structures often complement the pandas DataFrame structures.

Before working with the data, reading the dataset documentation is important to understand what the values in the data mean and what can/can't be removed for our specific purposes. Leaping off of this point, I display all rows and columns of the data using

```
‘pd.set_option(’display.max_rows’, None)‘

‘pd.set_option(’display.max_columns’, None)‘
```

to thoroughly inspect the dataset without truncation, which helps me understand its shape and the kind of data it contains before making any modifications. There are many columns that aren't important for our purposes like "STATION", "BackupDistanceUnit", etc. It is good practice to create a copy of your dataset and work on that copy to ensure I have an unmodified version of the original data. This way, if something goes wrong during data cleaning or preprocessing, I can always refer back to the original.

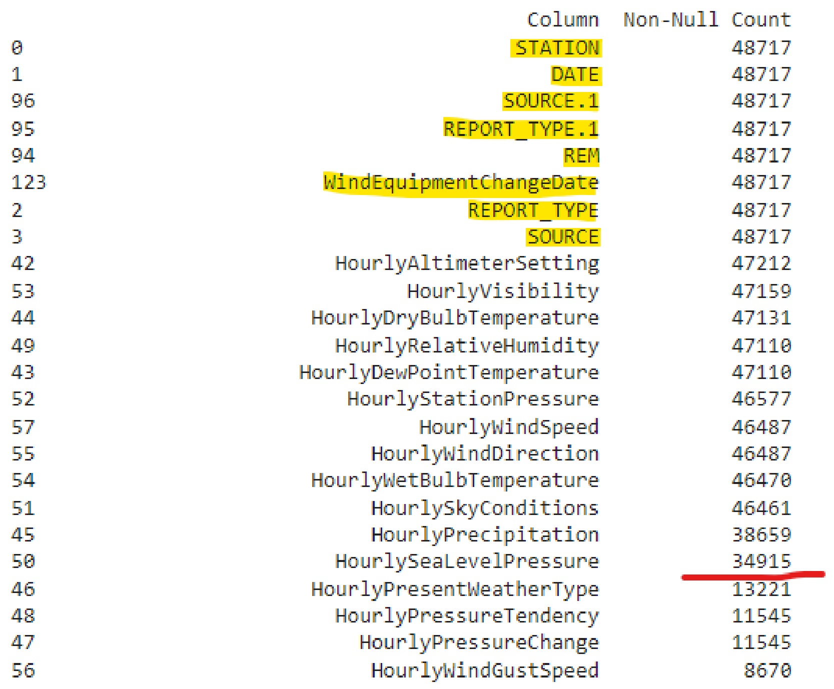


	STATION	DATE	REPORT_TYPE	SOURCE	AWND	BackupDirection	BackupDistance	BackupDistanceUnit	BackupElements	BackupElevation
0	72438614835	2020-01-01T00:54:00	FM-15	7	NaN	NaN	NaN	NaN	NaN	NaN
1	72438614835	2020-01-01T01:54:00	FM-15	7	NaN	NaN	NaN	NaN	NaN	NaN
2	72438614835	2020-01-01T02:54:00	FM-15	7	NaN	NaN	NaN	NaN	NaN	NaN
3	72438614835	2020-01-01T03:28:00	FM-16	7	NaN	NaN	NaN	NaN	NaN	NaN
4	72438614835	2020-01-01T03:47:00	FM-16	7	NaN	NaN	NaN	NaN	NaN	NaN

Figure 1: Data before any preprocessing, maniuplation, or cleaning

It's apparent that many of the columns have ‘NaN’ values, which depending on how many there are, could pose a problem to our models. I calculate and display the count of non-null (non-missing) values in each column using ‘climate data.notnull().sum()‘ (the spaces in code represent underscores). The result is stored in ‘non null counts‘, which includes the column names and the corresponding number of non-null values. I then sort this summary by the count of non-null values, which helps identify which columns contain a lot of missing data. I then decide on a good cut off point from there. It seems the number of null values drastically increases after ‘HourlySeaLevelPressure’ (underlined in figure 2), so I remove any columns that have less non-null values than that column.

After filtering the columns based on the non-null threshold, I also specify a list of columns to be removed. These include metadata or irrelevant fields such as "STATION", "REPORT TYPE", and "WindEquipmentChangeDate". This



```
                              Column  Non-Null Count
0                            STATION           48717
1                               DATE           48717
96                          SOURCE.1           48717
95                     REPORT_TYPE.1           48717
94                               REM           48717
123            WindEquipmentChangeDate           48717
2                        REPORT_TYPE           48717
3                             SOURCE           48717
42             HourlyAltimeterSetting           47212
53                   HourlyVisibility           47159
44           HourlyDryBulbTemperature           47131
49             HourlyRelativeHumidity           47110
43           HourlyDewPointTemperature           47110
52              HourlyStationPressure           46577
57                    HourlyWindSpeed           46487
55                HourlyWindDirection           46487
54           HourlyWetBulbTemperature           46470
51                HourlySkyConditions           46461
45                HourlyPrecipitation           38659
50             HourlySeaLevelPressure           34915
46            HourlyPresentWeatherType           13221
48             HourlyPressureTendency           11545
47               HourlyPressureChange           11545
56                HourlyWindGustSpeed            8670
```

Figure 2: Finding the cutoff point

needs to be done because the "STATION" column, for example, is the same no matter what and so has 0 non-null values (highlighted in figure 2). These columns are redundant or unnecessary data that doesn't contribute meaningfully to the analysis as mentioned before, so I remove them to have a cleaner and smaller dataset. The only exception is the "DATE" column. Even though we're not utilizing it here, it is important to put the date as the index for time series data. However, this is just the bare minimum in terms of preprocessing.

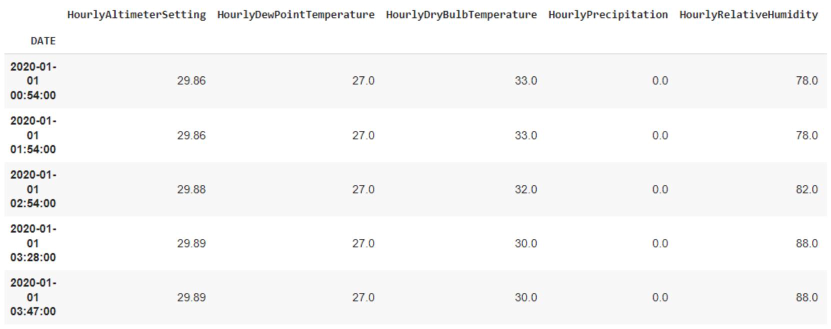
3.2 Preprocessing

The preprocessing section is perhaps even more important, because it goes more in-depth on the cleaning of the data. While we were doing a bit of it in the previous section, it doesn't touch on the important nuances of weather data. This is where reading the document is most helpful, since there are cases where character values appear instead of numeric data. Our models can only deal with numeric values, so these need to be taken care of.

Reading the documentation, we know that a "T" means "trace", and indicates that the value is < 0.005 inches water equivalent. Hence, I replace all "T" in the data with a value of 0.0025. Similarly, I replace all values of "s", "M", or "*" with NaN values to be safe.

When working with machine learning models, there are options to have the models replace all null values with the mean of that column. However, I think this is a bad idea in our case because clearly for most, if not all, measurements, they widely vary over a couple of months, let alone four years. Replacing the entire column with the mean of that column doesn't make much sense, and may even be harmful. Hence, I defined two functions to help fill in missing values in each column. The first function, 'compute window mean', calculates the mean of a specified column's null values within a defined window of 10 values around a given index. I specify the window size, which allows me to look back and forward from the index, ensuring I remain within the DataFrame's bounds. After extracting the relevant values, I drop any NaN entries and compute the mean of the remaining values. If there are no valid values left, the function returns 'np.nan'.

The second function, 'fill missing values', utilizes the previous function to fill in the missing values it couldn't before by lowering the window size. If all else fails, I fall back on the global mean of the column. This two-pass approach helps ensure that I fill in as many missing values as possible.



	HourlyAltimeterSetting	HourlyDewPointTemperature	HourlyDryBulbTemperature	HourlyPrecipitation	HourlyRelativeHumidity
DATE					
2020-01-01 00:54:00	29.86	27.0	33.0	0.0	78.0
2020-01-01 01:54:00	29.86	27.0	33.0	0.0	78.0
2020-01-01 02:54:00	29.88	27.0	32.0	0.0	82.0
2020-01-01 03:28:00	29.89	27.0	30.0	0.0	88.0
2020-01-01 03:47:00	29.89	27.0	30.0	0.0	88.0

Figure 3: Data after all the cleaning

3.3 Machine learning models

3.3.1 Random Forests and their applications

Random Forest is an ensemble learning method primarily used for classification and regression tasks. It operates by constructing a multitude of decision trees during training and outputting the mean prediction (for regression) or mode of predictions (for classification) from these trees. Each tree in a random forest is built from a random subset of features and a random sample of the training data (with replacement), which introduces randomness into the model. This randomness reduces overfitting and improves the model's generalizability.

Weather data is complex and highly variable, with many features such as temperature, humidity, pressure, and wind speed. Random Forests work well because they provide a natural ranking of feature importance. In weather prediction, this is useful because it helps identify the most important factors that influence weather patterns. For instance, in predicting rainfall, pressure and humidity might be more critical than temperature. Part of the complexity of weather phenomena is the non-linearity. Random Forests can capture complex interactions between variables without the need for explicit feature engineering. Moreover, as we've seen in the previous section, weather data can be noisy or incomplete, especially when dealing with historical records or sensor errors. Random Forest is inherently robust to noise due to the averaging of multiple decision trees. No single tree dominates the decision-making process, which helps reduce the impact of noisy data. Since we're using tabular data, a Random Forest model is particularly useful due to its efficiency. Part of their usefulness, although we won't be using it, is their ability to make accurate predictions. This is important in weather (not so much climate) as forecasts sometimes need to be made in a short amount of time.

In this context, Random Forest could be used for predicting numerical weather outcomes like temperature, humidity, or wind speed at a future point in time based on historical weather data (regression). They can also be used to predict categorical outcomes like "Rain" vs. "No Rain" or storm classification based on features like cloud cover, wind speed, etc. (classification).

In my Random Forest model, the goal is to predict future wet bulb temperature based on a variety of meteorological features like temperature, humidity, pressure, and wind speed as inputs to help the model understand patterns in the data. To improve model performance and reduce overfitting, I tuned several hyperparameters. I set the number of trees (n estimators) to 100, ensuring that the model is robust by averaging the predictions of multiple decision trees. To prevent overfitting, I controlled the complexity of the trees by limiting their maximum depth (max depth) to 10, which helps the model generalize better to new data. Any more than that and it would have increased risk of overfitting. I also used a min samples split of 2 and min samples leaf of 1 to control the minimum number of samples required for splitting nodes and forming leaves, respectively. This fine-tuning ensures that the trees do not grow overly complex while still capturing essential patterns in the weather data. For evaluating the model, I calculated the Mean Squared Error (MSE) to understand the average squared differences between the predicted and actual values. The lower the MSE, the better the model's accuracy. These metrics, combined with the tuned hyperparameters, help me assess the performance of the Random Forest model and guide further improvements if necessary.

3.3.2 LSTMs and their applications

LSTM is a type of Recurrent Neural Network (RNN) specifically designed to model sequential data, which is data that has an inherent order, such as time-series data. Unlike standard RNNs, which struggle with long-term dependencies (i.e., remembering information over longer sequences), LSTMs have memory cells that can store and recall information over long periods. This is crucial for tasks like weather prediction, where past weather conditions strongly influence future states.

LSTM uses gates to control the flow of information which help them learn which aspects of the data to keep and which to ignore, making them particularly effective for weather forecasting, where some historical weather events (like seasonal trends or recurring storms) need to be remembered while other less important information (like short-term fluctuations) should be forgotten. While this isn't particularly useful in our case, it is still important to note.

LSTMs are particularly effective for weather forecasting due to their ability to capture temporal dependencies in time-dependent weather data. They excel in recognizing long-term patterns, which is essential for predicting phenomena like rainfall, heatwaves, or storm activity. Additionally, LSTMs are adept at identifying seasonal components inherent in weather patterns, enabling them to analyze historical data for future predictions. Their memory capabilities also allow them to handle non-stationary data, adapting to changes in statistical properties over time, which is crucial in the dynamic nature of weather systems.

Training an LSTM model for weather forecasting involves several key steps. First, the model processes sequential data, such as hourly or daily weather measurements (we're using hourly), to learn temporal relationships and generate future predictions. Input features like temperature, pressure, and humidity, are used to help the model understand how these metrics change over time. I found a batch size of 16 was best for the small size of the dataset. Finally, I use MSE to evaluate the errors.

In a pipeline that combines Random Forest and LSTM for weather prediction, the two models can enhance each other's capabilities. Random Forest may be utilized as a feature extractor or for initial predictions based on static weather features, such as daily averages or summary statistics. Following this, LSTM can process time-series data, such as hourly or daily measurements, to model the temporal evolution of weather patterns for more nuanced forecasts. Additionally, an ensemble approach can be employed, where the final prediction results from a weighted combination of outputs from both models, effectively leveraging Random Forest's strength in capturing complex feature interactions alongside LSTM's ability to model temporal dependencies. Overall, both Random Forest and LSTM have strong applicability to weather data, and their use in tandem could provide highly accurate and robust weather forecasts, taking advantage of the different strengths of each model type. Within the climate science community, using multiple ensembles is a popular technique because as mentioned before, a problem in one model can be ignored if we replace that part with another model's predictions. However, we are using both models separately in this project.

3.4 Comparison & Visualization

Let's break down the code that deals with the graphs, explain its purpose, and discuss the relationships being visualized. Firstly, why is visualizing data essential in weather analysis? Well, it aids in understanding the relationships among various meteorological variables, identifying patterns, and spotting outliers or trends that may impact predictions. Graphs provide insights into seasonal and temporal changes in weather, reveal correlations between variables such as temperature, humidity, and pressure, and illustrate the distribution of weather events like rain and heatwaves over time. Additionally, visualizations can highlight feature importance in machine learning models, such as Random Forests, through importance plots. However, we're only using them here to show the trends of loss over batch sizes/folds.

I used the matplotlib library to visualize the performance of the two models by plotting their Mean Squared Error (MSE) loss and training times across various folds or batches. I began by defining a list representing the folds (from 1 to 10) and set up a figure for plotting. The first subplot illustrates the MSE loss of the RF model over the 10 folds, providing insights into how the model's prediction accuracy varies. I did the same thing for the LSTM model's performance.

Next, I initialized another figure to display the MSE loss of the LSTM model, following a similar format. I included a comparative subplot to overlay the MSE losses of both RF and LSTM models, enabling a direct comparison of their performance across the defined folds or batches (both were 10). Ultimately, this code allows me to effectively compare and illustrate to others the relative strengths of these models in weather prediction tasks.

4 Results & Discussion

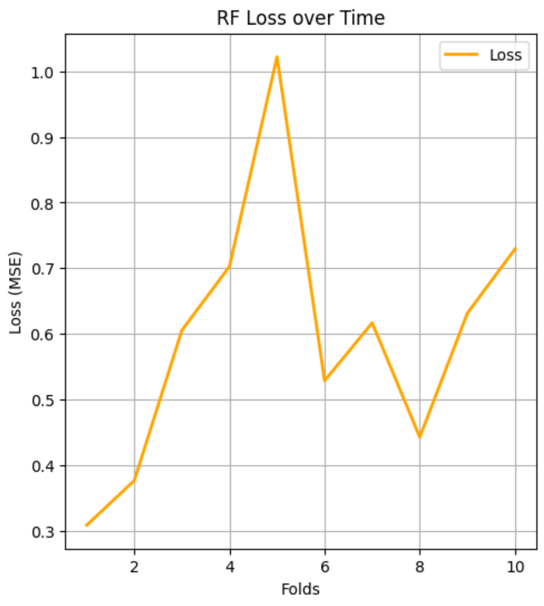


Figure 4: RF loss over 10 folds.

In Figure 4, the RF loss over time graph illustrates how the Random Forest model's performance fluctuates across different cross-validation folds. The x-axis represents the number of folds used during cross-validation, and the y-axis shows the Mean Squared Error (MSE), which measures the model's error in predicting the target weather values. The loss begins at a relatively low level of about 0.3 MSE for Fold 1, indicating good initial performance. However, as the model moves through the folds, the loss increases sharply, peaking at Fold 4 with an MSE greater than 1.0. This spike indicates a significant error in predicting weather patterns during that specific fold. After the peak, the loss drops drastically at Fold 6 to around 0.6 MSE, showing an improvement in performance. However, the loss remains inconsistent, with another rise toward the end at Fold 10 (0.8 MSE). The large fluctuations in this graph suggest that the Random Forest model is highly sensitive to the different subsets of data used in cross-validation, possibly due to the inherent variability in the weather data. This could indicate that the model struggles to generalize well across different folds, leading to erratic performance.

Looking carefully, it's challenging to determine whether the random forest model is overfitting. On one hand, the test MSE is lower than the average MSE across all folds, suggesting that the model is generalizing well to new data. However, the MSE values for each fold fluctuate significantly, as seen in Figure 1. Ideally, these values should decrease over time, but in most cases, they tend to increase with each fold. Given that the average MSE is lower than the test MSE, I believe the model is not overfitting and is performing as expected. The variation in each fold's loss may be due to the dataset itself. After cleaning, we are left with four years of data, and many of the features are interrelated, making this small dataset relatively complex. For this analysis, I used 10 folds with a random state of 42. A fifth of the data was set aside for testing, and the remainder was used for training. Each forest consisted of 100 trees. I believe these parameters are appropriate for the size of the dataset, though 10 folds might be slightly excessive.

In contrast, Figure 5, the LSTM loss over time graph, represents the learning curve of the LSTM model over ten training batches. The x-axis displays the training batches, while the y-axis shows the corresponding MSE. At the start (Batch 1), the loss is quite high, with an MSE around 310, signifying that the LSTM model initially has difficulty in predicting weather values accurately. However, there is a sharp drop in loss by Batch 2 (294 MSE), indicating that the model rapidly improves as it learns from the data. After this significant decrease, the loss fluctuates but stays within a narrower range between 294 and 298 MSE. These minor oscillations suggest that the model is refining its predictions, though further improvements are less dramatic. The LSTM model shows relatively stable performance, with no extreme spikes in error, implying that it may be more consistent in learning time-dependent weather patterns compared to the Random Forest model. Despite some small increases and decreases in loss over the batches, the overall trend suggests that the model is steadily converging towards a lower error rate.

Figure 6 presents a detailed comparison of the time taken per fold/batch

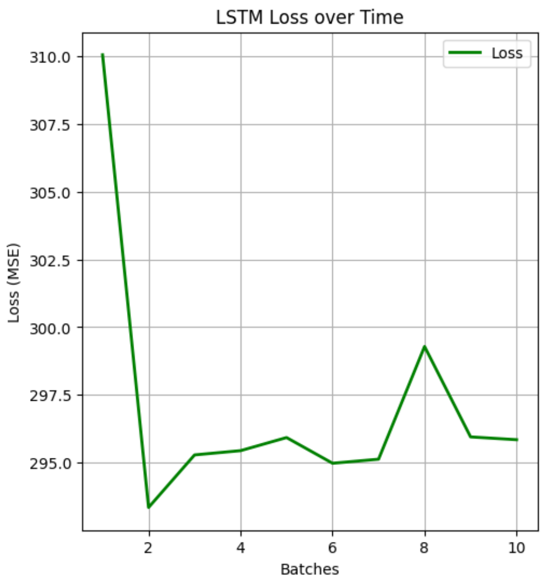
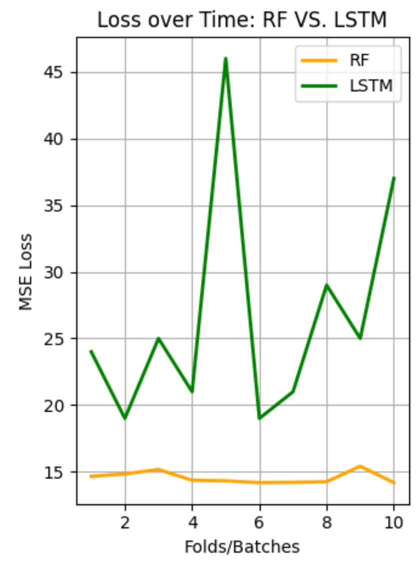


Figure 5: LSTM loss over 10 batches.

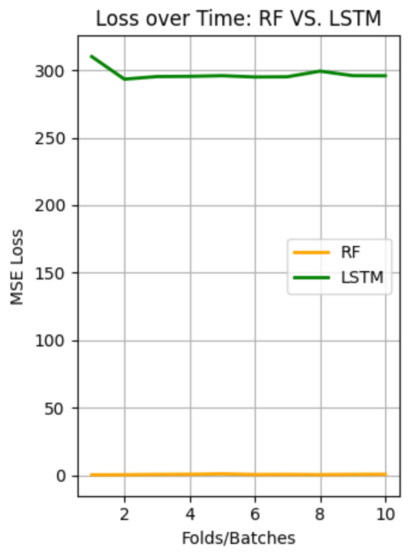
and the mean squared error (MSE) loss between the Random Forest (RF) and Long Short-Term Memory (LSTM) models. The two subplots offer insights into the computational efficiency and performance of these models. However, it's important to note that the left plot is incorrectly labeled; instead of showing loss over time, it shows the time (in seconds) per fold for the Random Forest and per batch for the LSTM model.

In the left plot, the comparison of computation time shows a significant difference between the two models. The Random Forest (orange line) is much more computationally efficient, with times consistently ranging between 14 and 15 seconds per fold. There is minimal fluctuation, indicating that the Random Forest is relatively stable in terms of training time. In contrast, the LSTM (green line) requires considerably more time per batch, with values ranging from 15 to 46 seconds. The LSTM's time fluctuates significantly, with peaks at batch 4 and dips at batches 2 and 6 (over 20 second difference). This variation highlights the more computationally intensive nature of LSTMs due to their sequential architecture and the need to maintain information across time steps, making them slower compared to the more parallelizable Random Forest.

The right plot illustrates the performance of each model in terms of MSE loss. The LSTM model (green line) maintains a relatively constant MSE of around 300 throughout the training batches, with only minor fluctuations. This suggests that while the model is learning, it struggles to significantly reduce the error, indicating that the LSTM might be having difficulty capturing the underlying patterns in the weather data. On the other hand, the Random Forest model (orange line) exhibits a much lower MSE, close to 0 throughout the cross-



(a) Comparison of times per batch between both models



(b) Comparison of loss per batch between both models

Figure 6

validation folds, suggesting that it achieves better predictive accuracy on this dataset. The large discrepancy in loss between the two models implies that while LSTM might be more suitable for time-series data, the Random Forest is outperforming it in this particular application.

Although LSTM is more time-consuming and less efficient in terms of reducing loss, its sequential nature might still be essential for capturing complex patterns in time-series data. Meanwhile, the Random Forest, while less specialized for time-sequential relationships, shows a strong ability to predict weather data with minimal error and much faster computation time.

I initially expected the LSTM model to perform better than the random forest, particularly since LSTMs are well-suited for handling time series data. However, this turned out to not be so straightforward, since "better" is a very broad term and depends on the context and what the data handler wants.

5 Conclusion

In this analysis of climate data from West Lafayette using both Random Forest and Long Short-Term Memory (LSTM) models, several key insights emerged regarding the performance and efficiency of these approaches in predicting weather patterns.

The Random Forest model demonstrated strong predictive capabilities, evidenced by its lower Mean Squared Error (MSE) across various folds of cross-validation, suggesting that it effectively generalizes to unseen data. Despite showing some fluctuation in performance, the model maintained overall stability and efficiency, completing training in a fraction of the time required by the LSTM model. This indicates that for smaller datasets, like the one utilized in this study, Random Forest is a compelling choice, balancing speed and accuracy.

Conversely, the LSTM model, while theoretically better suited for time-series data, struggled to outperform the Random Forest in this context. Its higher computational demands and less consistent performance highlight the challenges of applying complex sequential models to smaller datasets, where the intricate relationships captured by LSTM may not provide a substantial advantage. The results suggest that the memory mechanisms of LSTM may not be fully leveraged in datasets with limited variability, such as the one analyzed here.

Overall, this project reinforces the importance of selecting appropriate modeling techniques based on dataset size and complexity. Moreover, the use of ensemble models, which is common in the climate data science community, is an attractive approach to handling any kind of weather/climate data.