Sustainable Computing

Candidate Number 1073022

Abstract

By 2026 it can be expected that Data Centres will consume roughly 1000 TWh. As coding is not expected to go anywhere in the foreseeable future; it is necessary to see what can be done to mitigate the effects on the environment. The aim of this project is to gain some insight into the effects of coding on the environment, it explores the effects of model choice, location and time of day on power usage and carbon emissions.

Contents

1 Methods

1.1 LightGBM

Libraries used:

`pandas, numpy, lightgbm, sklearn, matplotlib`

The LightGBM model is a machine learning-based model typically used for regression and classification, it is also commonly used to predict stock prices and sales. Hence, I thought it would be a suitable choice to predict carbon intensity based on previous data. To implement this model, I resampled the data for each time-frame to produce average hourly, daily, monthly and yearly data. I then performed a hyper-parameter grid search optimisation by training the data and comparing it to the test data. For each model, I used a training size of 0.95 and a test size of 0.05 of the historic generation data. I compared the forecasted data with the actual 2024 carbon intensity to find the RMSE (Root Mean Squared Error).

1.2 Prophet

Libraries used: `pandas, numpy, prophet, matplotlib`

Prophet is a time series forecasting statistical model developed by Facebook. It is typically used to capture trends and seasonality, to predict revenue, user traffic and user activity. I believed it to be a suitable choice as energy usage and carbon intensity typically exhibit time/seasonal based trends. For this model I resampled the data for each time-frame to produce average hourly, daily, monthly and yearly data. I then performed a hyper-parameter grid search optimisation to find the best fit. I compared the forecasted data with the actual 2024 carbon intensity to find the RMSE.

1.3 Power, Runtime and Carbon Emissions

Programs used - `sysfs`

API used - `https://api.carbonintensity.org.uk`

Arguments: `PYTHON SCRIPT, LOCATION ID`

I implemented a shell script that recorded the battery and timestamp before the python script was called, it then runs the python script and records the values after running the code. The difference between the values yielded the power consumed and runtime of the python script. The shell script then calls the carbonIntensity script and multiplies the power by the carbon intensity of the given location to calculate the carbon emissions due to the script being run. This implementation means that the power consumed not only refers to the script being run, but also accounts for the auxiliary processes that are necessary when executing code.

2 Forecast Comparisons

2.1 Hourly and Daily Forecasts

Methods used: `LightGBM, Prophet` Upon observation of Figure 1 and 2 one can see that the LightGBM model is better able to mimic the short-term variations in the data. The Prophet Model is better suited for understanding the long-term trends in the data, this is reflected in the lower RMSE. The Prophet model could be used for long-term strategic planning whereas the LightGBM model could be used for short-term decision making. RMSE for LightGBM, Prophet Model: 77.2, 53.7

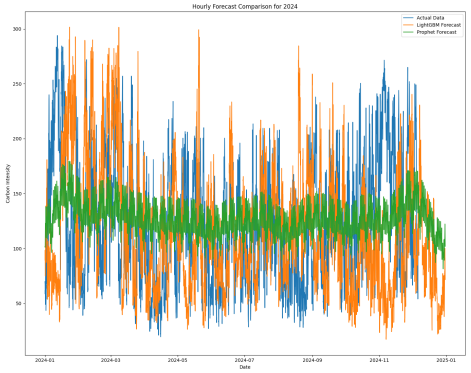

Figure 1: Hourly Forecast Plot of Actual Data, LightGBM and Prophet Hourly Models

RMSE for LightGBM, Prophet Model: 74.0, 46.7

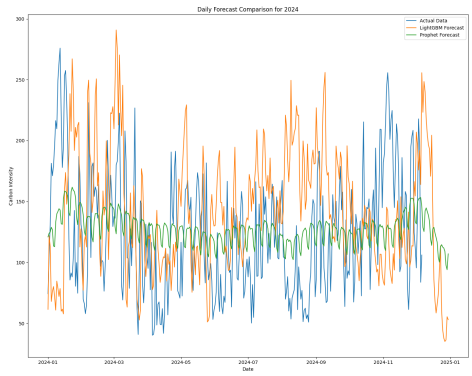

Figure 2: Daily Forecast Plot of Actual Data, LightGBM and Prophet Daily Models

2.2 Monthly Forecasts

Methods Used: `LightGBM, Prophet` As observed in Figure 3 both the forecasts don't accurately predict the trends in the data. Due to the small size of the training data, the forecasted data is static and doesn't mimic the volatility of the actual data. Whereas the Prophet model is better able to match the month to month variation which is reflected in the lower RMSE. However, it still lacks the sharp peak and dip characteristics of the actual data. RMSE for LightGBM, Prophet Model: 46.6, 25.1

Figure 3: Monthly Forecast Plot of Actual Data, LightGBM and Prophet Monthly Models

2.3 Annual Forecasts

Methods Used: `Holt-Winters, Prophet` In Figure 4, and in the RMSE, one can see that the Prophet model produces a close forecasted value whilst the GBM model overestimates the value significantly suggesting that the dataset is too small for a machine learning model to produce an accurate forecast. An alternative model with a closer forecasted value and can be seen in Section 6.2. RMSE for LightGBM, Prophet Model: 166.2, 7.5

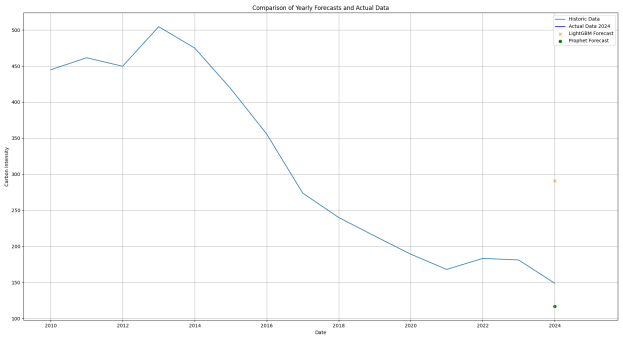

Figure 4: Yearly Forecast Plot of Actual Data, LightGBM and Prophet Daily Models

2.4 Runtime and Emission Comparisons

All the values plotted in Figure 5 are for training the model, then plotting the past data and the forecasted data on the relevant time scales. We can see that the runtime and power consumption of each method are comparable with the exception of the Hourly Prophet model. This is due to the additional time taken to build the model which has 24 times more data points than the Daily models. As seen in Figure 1, the increased time taken doesn't accurately predict the short term variations and only produces a slightly smaller RMSE when compared to the LightGBM model.

The daily and monthly models both have comparable runtimes and power consumption meaning the choice between these models should be based on accuracy alone.

As seen in Figures 1, 2 and 3 both methods have their advantages and disadvantages whilst Figure 4 shows that

the LightGBM model is not a suitable choice for a small dataset.

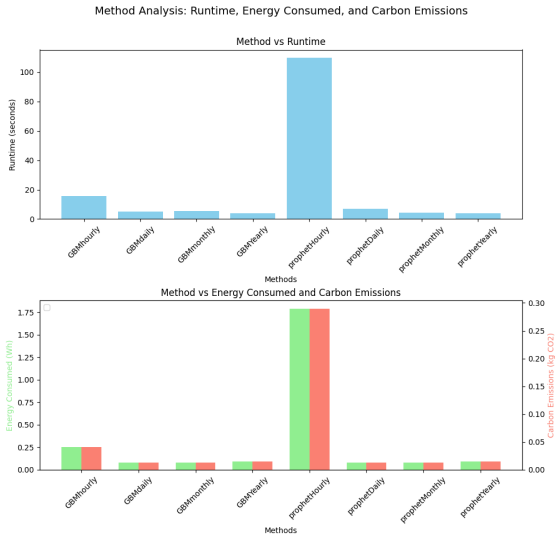

Figure 5: Runtime and Emission Comparisons of Different Methods

3 Hourly Forecast Optimisation

The changes made to the optimised script: It uses fewer estimators/trees; this is a trade-off between the training time and model performance. It limits how many leaves (how many outputs there are for a given path) are in each tree, how may bins (how many intervals the continuous values are divided into) are used, how many data points are in each leaf, how many sums of hessians are done in each leaf. It also uses Gradient Based One-Side Sampling as opposed to Gradient Boosting Decision Tree, which reduces runtime by focusing primarily on the most important samples. These optimisations reduce training time and memory usage at the cost of accuracy, the effects can be seen in the reduced runtime and power consumption seen in Figure 6. In Figure 7, we can see that the changes made to reduce the runtime and power consumption of the model have had a notable effect on the forecasted data. The optimised model appears to be significantly more volatile leading to a slightly increased RMSE. Hence, this optimisation was at a small cost of accuracy. RMSE for LightGBM, Optimised LightGBM Model: 77.220, 78.211

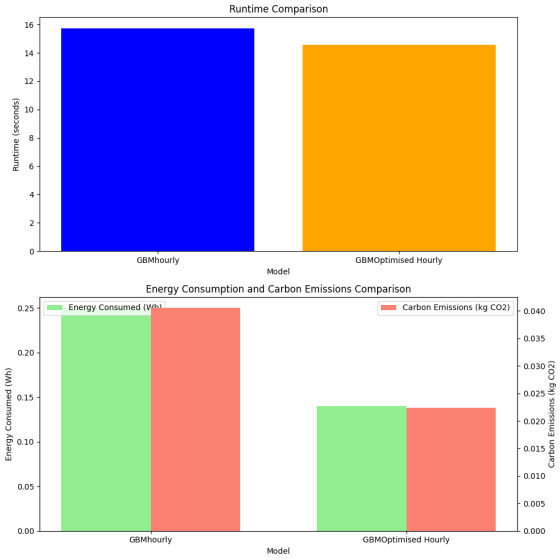

Figure 6: Emissions and Runtime of Optimised vs Un-Optimised Model

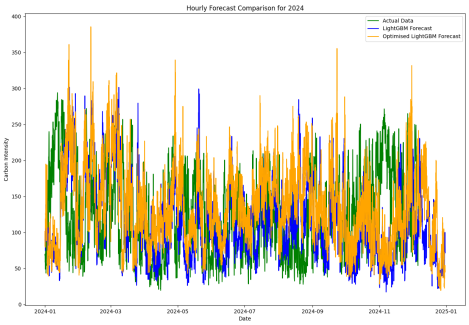

Figure 7: Plot of Actual Data, LightGBM and LightGBMOptimised

4 Emission Comparisons

4.1 Different Machines

I chose to compare a workstation Thinkpad running Linux to a lightweight touchscreen Lenovo laptop running Windows, the specifications can be seen in Table 1.

To ensure the comparison was valid, I measured power usage for the entire device during the execution of the code.

In Figure 8, it can be seen that the workstation laptop has a significantly shorter runtime. This shorter runtime

likely played a large role in the reduced power consumption and hence the reduced carbon emissions.

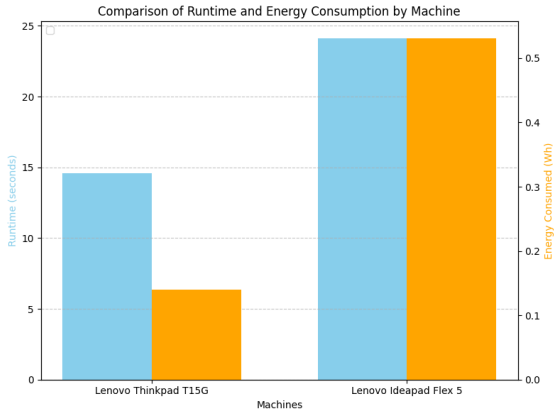

Figure 8: Emissions and Runtime Comparison of Different Machines

4.2 Different Times and Locations

I chose to run the code during three different times of the day: morning, early evening and late at night. As expected all regions had reduced carbon intensity late at night likely due to the reduced energy demand. As seen in Figure 9, Scotland consistently had much lower carbon intensity than the other regions being considered suggesting the energy mix in Scotland is much more green than the rest of the United Kingdom.

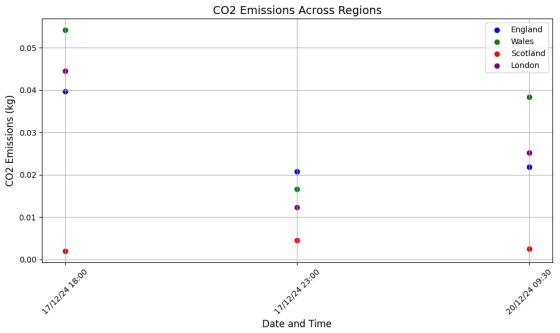

Figure 9: Emission Comparison in Different Locations and Times

5 Discussion and Insights

The aim of this project was to gain some insight into how code can be written and executed to reduce the energy usage and carbon intensity. The main takeaways from this project are:

- it is vital to chose an appropriate model/method for the code, this should be done by weighing up the accuracy against the runtime and power consumption.
- if possible coding (and almost any activity that requires power e.g. laundry) should be done during the night as generally the carbon intensity is lower. In the event this is not possible simulations and code with longer runtimes should be left to run overnight.

Another possible solution is to use a lower level language e.g. C++ as they offer more control over the memory and cache. Implementing the same code in a low level language almost always results in a decreased runtime which typically leads to a decreased power consumption. However, the obvious cost is that the code becomes much harder to write and often takes longer as there are not as many in built libraries.

Though this project provided some insight into when carbon intensity is at its' lowest, only taking 3 different times is not significant enough to fully understand the variation in carbon intensity. Currently, non-renewable sources are typically used to bridge the gap between the renewable production and the demand. To be able to better understand the carbon intensity, further multivariate analysis would have to be undertaken. Understanding factors such as weather e.g. solar intensity, wind speed and the day of the week e.g. bank holidays, weekends are crucial to be able to accurately forecast the carbon intensity.

In certain cases when the code is only required to be run once or a handful of times, optimising the code is a frivolous endeavour as the power saved through optimisation is outweighed by the power consumed by the device during the code editing and testing.

6 Appendix

6.1 Machine Specifications

	Lenovo Thinkpad T15G Gen 2	**Lenovo Ideapad Flex 5 14**
CPU	Intel Core i7-11800H (8 cores)	Intel i5-1035G1 (4 cores)
GPU	NVIDIA RTX 3070 Laptop GPU	iGPU
RAM	32GB DDR4	8GB DDR4
STORAGE	2 x 1 TB NVMe SSD	256GB NVMe SSD
NETWORK INTERFACE	Intel Wi-Fi® 6E AX210, 802.11ax 2x2 Wi-Fi	Wi-Fi® 5, 802.11ac 2x2 Wi-Fi
ACTUAL BATTERY CAPACITY	84.8Wh	52.5Wh
OPERATING SYSTEM	Ubuntu 24.04 LTS	Windows 11

Table 1: Comparison of Specifications for Lenovo Thinkpad T15G Gen 2 and Lenovo Ideapad Flex 5 14

6.2 Holt-Winters

Libraries used: `pandas, numpy, statsmodels, matplotlib` Another method I explored was the Holt-Winters Model, it is a time-series forecasting statistical model that uses exponential smoothing to forecast data such as energy consumption and demand. I chose it to implement the yearly model as it works well on small datasets and handles seasonality very well. I used the additive model to predict the average 2024 carbon intensity and then added this data point to a plot of the average annual carbon intensity from 2009 to 2023. I found the RMSE by comparing the forecasted data with the average 2024 carbon intensity. As seen in Figure 10, it produces the a better estimate than any of the other methods.

Figure 10: Annual Forecast Plot of Actual Data, Prophet and Holt-Winters Yearly Models

RMSE for Holt-Winters, Prophet Model: 1.354, 7.481