**Methodology**

**Lightbgm:** This is a gradient boosting framework, using tree-based learning algorithms. By using gradient based one side sampling and exclusive feature bundling, the lightbgm a very lightweight and fast algorithm. It is mainly implemented on large datasets due to it being prone to overfitting small datasets. Whilst a decision tree splits the tree tree-wise, the lightbgm algorithm splits the tree leaf-wise.

**RNN:** A recurring neural network is a form of machine learning that returns on itself to increase accuracy. For a typical neural network, firstly, an example from a dataset is loaded (in this case, the training data supplied). The network takes that example, and applies mathematical formulae to it using random variables, yielding a predicted result. Using the validation data, this prediction can be compared, and the difference between them will give an error. Returning the error back through the same path will adjust the variables, and this is all repeated until the variables are defined with minimal error. The recurrent neural network takes this a step further, by instead of taking in one example at a time and producing one result, it takes multiple neural networks which feed information to each other. This allows for a low time complexity and makes it suitable for dealing with large datasets.

**Decision Tree:** The decision tree uses rules learnt during its training phase to make decisions. You begin with the root node, which splits off to two different regions. These regions also split into a further two different regions and this process continues. The decision tree uses the rules supplied to decide which region to go to at a node, and this process is continued until all the rules are applied or until there are no data points left. The decision tree time complexity is of form O (n log(n) \* d), where d is dimensionality in the data. A random forest uses multiple decision trees with an element of randomness for more accuracy, however due to the multiple decision trees the computational memory cost is significantly higher than that of a single decision tree.

**KNN**: KNN works using ‘feature similarity’ to predict the values of new data points. Effectively, new points are assigned based on its resemblance in comparison to the training dataset. The value of k can be adjusted, such that more of the data is included in a single prediction. This algorithm can be used for either classification or regression, however for large datasets it can be extremely costly in terms of memory in comparison to other models with a time complexity of O (n) for testing.

**Experimental Setting:**

As part of the pre-processing, one of the main factors was memory usage due to the large nature of the data. By converting the data to feather files and applying a function to reduce the memory usage (mainly by converting int64s to int8s, or float64s to float32s) this allowed us to run the programme more efficiently with very little loss of data. The data was also merged with the weather data and the building data to allow for more features to be selected, and the timestamp was broken down into hours, days, day of the week, day of the year, the month, and the year, which allows us to get the exact date in integer values. A log transformation was also applied to the meter reading and square feet, which was done to account for the large range of values and make the training of the model more effective. Furthermore, months were added along with an ‘isDayTime’ Boolean in form 1 or 0, 1 being it is between the hours of 06:00 and 18:00. The feature selection allowed us to remove various features from the data to acquire a better result from the data.

**Decision Trees**

For the decision trees model, the log transformation was not used as converting the raw train meter readings to integer values instead produced higher accuracy rates. However, to achieve accurate results it also required a large enough maximum depth which, in turn, increases the complexity, and therefore memory usage of the model exponentially.

Chart, line chart

Description automatically generatedWith these adjustments the hyper parameters that were found to have the greatest impact were the random state and max depth of the trees. To tune these hyperparameters, a small sample from the train data was looped and the highest accuracy values for these were found.

*Figure N: A plot of the accuracy score for each value of random state tested.*

As expected, the random state showed a random distribution of accuracy values, and therefore the highest point the most accurate value that was tested was used. This hyperparameter is especially valuable when considering that the random state of the value has very, negative effects on memory or performance of the model.

Chart, line chart

Description automatically generated*Figure N: A plot of the accuracy score for each value of max\_depth tested.*

Figure N shows an inverse logarithmic shape, with it plateauing at roughly 40. Therefore, any value above this would cause the model to be unnecessarily complex and cause performance issues. However, the memory requirements of the max depth value of 40 was far too great for our computational resources. The value of max depth 14 was the maximum possible with the computational power available and was therefore the compromise used.

**RNN**

(talk about validation and training ratio and number of epochs etc here)

**Results (Kind of)**

**Decision Trees**

The decision trees classifier had a runtime of ~3 hours to fit and predict the values. These large values are understandable given the large training set to test ratio, and the magnitude of data sets in general. The Kaggle score was INSERT (*test runs show score of ~2.3 so will base round that value for now*). Considering the max depth value used and how that effected the accuracy, shown in Figure N, this could be expected to be much higher if the computational resources were available for a more complex model. Therefore, despite showing promising signs that this model could produce strong results for the dataset, its memory usage for more complex and accurate models, does have weakness that could make it deficient to other methods.

**RNN**

A recurring neural network model was used to predict the meter readings data, the reasoning being that an RNN has a low time complexity. Given the extremely large nature of the data set (~41 million rows of data), this is an extremely useful property. With epochs = 10, the RMSLE was 1.41, and the prediction returned a Kaggle score of 2.33. With epochs = 23, the RMSLE was 1.02, however the prediction still returned a Kaggle score of 2.33. With epochs = 50, the RMSLE was 1.05, and the prediction returned a Kaggle score of 2.54.

Epochs: 10

RMSLE Train = 1.41

RMSLE Test = 1.41

Kaggle score = 2.327

Epochs: 23

RMSLE Train = 1.02

RMSLE Test = 1.02

Kaggle score = 2.327??

(*This could be more appropriate for analysis section)* Whilst this model is not as accurate as other models, such as lightbgm or random forest, the low memory usage and quick running time makes it suitable for usage on extremely large datasets, where the accuracy can be within a tolerable range.