Introduction:

This problem revolves around predicting the number of rented bikes from a bike rental shop using data which includes climate conditions and time information. Since the target variable, the count of rented bikes, is a continuous variable, and we are interested in making a prediction in a continuous state space, we will be looking at regression machine learning algorithms to solve our problem.

Methods:

To start, we will require a baseline model to compare our results against. The algorithm that we will be using for this is a simple Linear Regressor. This is because Linear Regression is a robust algorithm, providing fewer worries about overfitting, and so we would expect it to perform at a good level without any configurations. This is preferable to sklearn’s Dummy Regressor since the Linear Regression algorithm is likely to produce a viable model to predict the count of rental bikes. Thus, any model that performs better than this Linear Regressor baseline is a testament.

The two algorithms that we will be using to solve our regression problem are a Random Forest Regressor and an XGBoost Regressor. This is because ensemble algorithms have a good record for predictive accuracy, we can control the fit of the model easily and they also have a reputation for stability. We also have some intricacies that differentiate the Random Forest and XGBoost Regressors with XGBoost a boosting algorithm that provides greater regularization capabilities as well as an improved speed.

Whilst training these models, we will be using several techniques so that we can effectively test the performance and robustness of each model. This will allow us to make a fair comparison between our two models and choose the most appropriate one. Firstly, we will be splitting our data into three samples: data to train the model with, data to test the model and tune our hyperparameters and data that will act as our unseen data to validate the performance results from the tuning. The training set will account for 60% of the data and the remaining 40% being split evenly between the test and validation datasets.

To evaluate our models, we will be using the as our performance metric. The of a model is defined as the proportion of variance in the dependent variable that we can predict from the independent variables. Mathematically, for observation , prediction and mean observed value, we have that

The benefit of using this metric is that, unlike Root Mean Squared Error (RMSE), it is not attached to any units, thus we can get a clear view of the model performance without needing to give context to a figure. However, a downside of the that the figure does not provide us with any more information about the residuals since it is detached from the predictions. Further, it does not highlight bias in the model predictions, so it is important to be careful of an overfitted model.

To choose our hyperparameters, we will be using the Optuna framework to assist us with optimisation. Optuna is a Bayesian approach to finding the optimal parameters which deals with the search problem probabilistically. It uses a Tree-structured Parzen Estimator which sequentially builds models to estimate hyperparameter performance based on past trials, and thus selects new hyperparameters to evaluate. What this does is provide a search algorithm that performs better than the more commonly used optimisers such as a Randomised Search or Grid Search in a clean structure.

Results:

Hyperparameters are parameters within the algorithm that can be used to adjust the learning process in a variety of ways. They can be split into two types of hyperparameters; ones to control the model complexity and ones to control the regularisation of the model – one might argue that an algorithm’s learning rate and optimiser fall slightly outside this grouping since they sit ‘on top’ of the algorithm.

Chart, bar chart

Description automatically generatedStarting with the results of our baseline Linear Regression Model, as displayed in **Figure 1**, we found that this baseline produced the following for the train and test datasets respectively: 0.567670 and 0.551524. As expected, this is a very good starting point. These scores are close, suggesting we have avoided mass-overfitting, and are at a suitable level to start our comparisons.

Figure : Bar Chart of for initial models for Train and Test data. Here we have evaluations for the following models: Linear Regression (Baseline) Model, Initial XGBoost Model and Initial Random Forest Model.

As previously mentioned, we will be utilising the XGBoost and Random Forest Regressors to model our problem. To start, we fitted each model to the training set without any hyperparameter adjustments. This is because the evaluation of these models will give us an indication to what hyperparameters we should be adjusting to get the best out the model. An overfit model would mean we would want to reduce the model complexity and increase the regularisation level with the opposite needing to happen for an underfit model.

As you can see in **Figure 1** both the XGB and RF models perform at a higher level than our baseline with in the range of 0.90 - 1.00. An exceptional start. However, this plot does show that we have overfitted our models since there is a great different in between the train and test sets in both cases. This tells us that we need to increase our regularisation and reduce our model complexity.

Due to this realisation, we chose to set the max\_depth = 3 for our XGB model as this would limit the complexity of the tree and so reduce the effect of overfitting in our evaluations. As you can see in **Figure 2**, this value was chosen since it was the point at which the for our train and test sets converged most rapidly – we felt that opting for max\_depth = 3 as opposed to max\_depth = 2 provided gains in performance without effecting the model’s fit.

Figure 2

Initial max\_depth change

Once this was done, we created an OptunaSearchCV instance for each model to help tune our hyperparameters. OptunaSearchCV includes a cross-validation component in which the training data was randomly split into ten folds to improve the robustness of our model. For each search, we performed 200 trials. Each trial would select a value for each hyperparameter from the given options, train the model using these hyperparameters – with our cross-validation element – and produce the mean value of our cross-validation scores. OptunaSearchCV uses these scores, and the Tree-structured Parzen Estimator sampler to final the maximal score.

With regards to hyperparameters, we shall start with our XGB model. Two intuitive ways to control model complexity – literally reducing the size of the model – were min\_child\_weight, to control how finely the data is split for each leaf, and n\_estimators, the number of decision trees generated in the boosting process. We decided to tune n\_estimators as opposed to using the early\_stopping\_rounds hyperparameter in XGBoost’s *fit* method so we could maintain consistency across our cross-validation folds as n\_estimators can be reproduced, unlike early stopping.

We also chose to include subsample and colsample\_bytree into our tuning pool to introduce a stronger element of randomness into our training. Subsample denotes the proportion of observations to be randomly sampled for each tree and colsample\_bytree is similar, but it indicates the fraction of columns to be randomly sampled for each tree. This was done so that our training data was more representative of the entire population such that we did not draw conclusions specific to our training data. We also used three regularisation hyperparameters: gamma, reg\_lambda and reg\_alpha. Each of these hyperparameters, regularise the model and a different stage. Gamma controls the complexity of the gradient boosting; reg\_alpha smooth the prediction line by penalising the features that cause the loss function to increase (L1 regularisation); and reg\_lambda, L2 regularisation, regularises the weights of the features. Finally, we also looked at tuning the learning\_rate of the model since a high rate will cause the model the take larger steps in the training’s optimisation phases and thus avoid overfitting.

Since XGBs and RFs are tree-based algorithms, there was an overlap in hyperparameters used such as max\_depth and n\_estimators – both denoting the same aspect of the model as with XGB. However, we also introduced min\_samples\_split and min\_samples\_leaf. Min\_samples\_split controls the minimum number of samples needed for a node to be split and min\_samples\_leaf works in a similar way to XGB’s min\_child\_weight whereby it states the minimum number of samples on a leaf node. RF-specific hyperparameters we also chose to tune were max\_features, the number of features to use when looking for an optimal split, and the Boolean hyperparameter bootstrap. This is used to determine if we want to use bootstrap samples to build decision trees which is useful for the model only being representative of the training data.

Figure 3

Improvement in r2.. lineplot

After 200 trials, the optimised hyperparameters produced an XGB model which was evaluated at the following for train and test respectively: ##### and #####. These are better because…

Further, 200 trials on the RF model gave tuned for train and test respectively: ##### and #####. Can be seen in **figure 3**

**Figure 4** shows the search’s slice plot which shows the objective value produced by each trial hyperparameter colour-coordinated by the trial number. We can see that… as trials increased, more focussed at specific point….. comment on notable hyperparameters.

Figure 5

Importance plots (both)

Figure 4

Slice plots (both)

* Optuna param importance plots to show what to tweak.
* Talk about tweaking to improve
* Show improvement
* Effects of different hp choices: optimisation history
* And so we decided on the following parameters…..
* Validation data… histograms

Figure 7

Optimisation history plots (both)

Figure 6

Hp changes.. improvements

Figure 8

Final, overall improvements

* Hyperparameters
  + 3 pts: Show that you understand what hyperparameters are and how they can be selected.
  + 5 pts: Look at the effects of different hyperparameter choices on the performance of your models.
  + 5 pts: Present the effects of the different hyperparameter choices on the performance of your models using tables, plots, or other presentation.
    - Optimisation history
    - Max\_depth – how this makes the model more overfit. (plot)
    - Lambda – effect. (plot)
  + 2 pts: State what hyperparameter choices you make and why.
* Training/testing
  + 4 pts: Train models and select hyperparameters in a way that gives robust performance
    - Train/test/val split
    - Optuna opt history plots
    - Train/test score values – comment on robustness
    - Cross validation – do we need test set?
  + 3 pts: Test the performance of your models and compare their performance
    - Show results on train/test data – overfit, values etc.
  + 3 pts: Make sure your models are tested in a way that shows whether they are able to generalise to unseen data
    - Comment on val data – unseen
    - histograms