Documentation

1. Introduction

The dataset provided is the number of rental bikes out for rent each hour over a year. There are 8760 records and 14 features, out of which four are categorical and ten are numerical. The problem statement is to build a model to predict the number of rental bikes out at each hour, given the values of other features of the dataset. There are a couple of reasons why this is a regression problem. Firstly, this is a supervised machine learning problem. The required algorithm models the dependencies and relationships between the target output and input features to predict the value for new data given the labelled data, ruling out the possibility of an unsupervised clustering problem. Secondly, the target feature (Rented Bike Count) is a **continuous feature** which rules out the possibility of a classification problem. The output space or the target value (y) corresponds to a continuous feature (R) in regressionbased problems. The model predicts the target values based on input features from the data fed into the system. Due to these reasons, regression algorithms are used.

All further analysis, preprocessing steps and detailed model evaluation can be referred from the jupyter notebook attached with this report

1.1 Training – Validation – Test

Splitting the data into training and test sets helps ensure that our model can generalise. I further split the training set into training and validation sets to test the model's performance on unseen data and utilize the validation set to assess with different parameter values. The test set is not included until a model is selected to evaluate the model performance with unseen data. I considered a train-valid-test split of 80:10:10.

2. Model (Evaluation & Selection)

I have used the following metrics to evaluate the model performance:

1) **R Square:** R Square measures how the model can explain much variability in the target variable. R Square value is between 0 to 1, and a bigger value indicates a better fit between prediction and actual value. I have considered this metric because R square is a good measure to determine how well the model fits the data, which is helpful for our regression problem. However, the limitation of the R square is that **it does not consider the overfitting problem**.

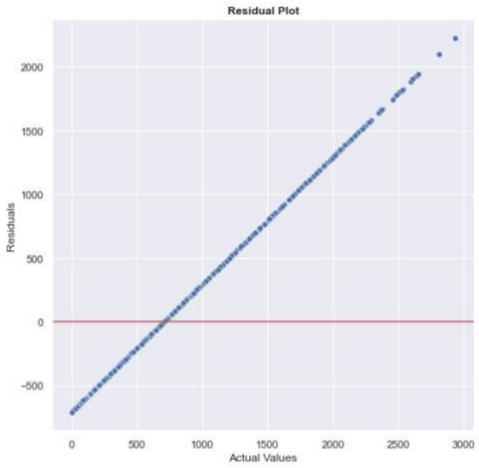
2) **Root Mean Squared Error (RMSE):** While R Square is a relative measure, Mean Square Error is an absolute measure of the goodness for the fit. It gives us an absolute number of how much the predicted results deviate from the actual number. I considered RMSE because sometimes the MSE value can be too big to compare, and the square root of RMSE brings it back to the same level of prediction error and makes it easier for interpretation. I also used it to compare other model results and select the best regression model.

I have used the following regression models to carry out the regression problem:

- Dummy Regressor (Baseline Model)
- K-Nearest Neighbors (Regressor)
- Decision Tree (Regressor)

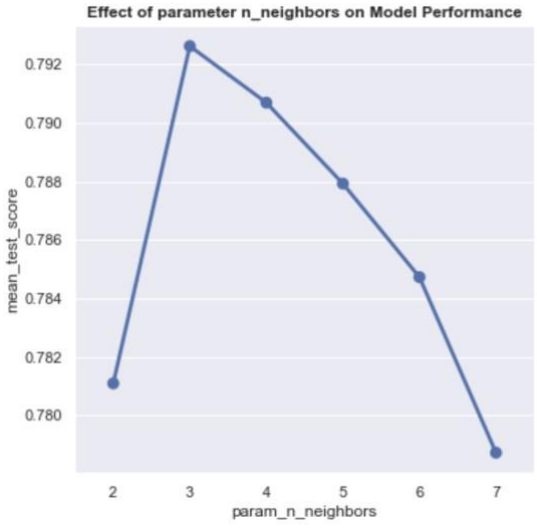
1) **Dummy Regressor (Baseline Model)**: Regressor that makes predictions using simple rules. I used the Dummy regressor as my baseline model as it computes a simple metric like mean and predicts the

same to every data point. I used this baseline model to compare the performance with the other regression models.



I used residual plots to evaluate the models and show how much the predicted values vary from the actual values. As we can see from the above residual plot, the model is poorly performing in predicting the values. I used this formula to obtain the residuals: **residuals = actual values – predicted values**. So, if the residual is equal to 0, then our prediction and actual values are equal. If the residual is negative, then our model **overestimated** the actual value. If the residual is positive, then our model **underestimated** the actual value. So, any data point on the red line represents the correct prediction of the model.

2) **K-Nearest Neighbors (Algorithm - 1)**: In this algorithm, the target is predicted by local interpolation of the targets associated with the nearest neighbors in the training set. I used KNN without scaling and later scaled with a min-max scaler. To improve any model, we use hyperparameter tuning. Hyperparameter tuning relies more on experimental results than theory. Thus, the best method to determine the optimal settings is to try many different combinations and evaluate the performance of each model. To try different parameter combinations, I used RandomSearchCV and GridSearchCV. I used two cross-validation methods because RandomSearchCV calculates the parameters on random splits, which is fast but not very accurate and GridSearchCV calculates all the combinations of the parameters, which is accurate but slow. So, I first used RandomSearchCV to obtain the range of the values and GridSearchCV to give the optimal parameter value. I considered the **n_neighbors** hyperparameter, the number of neighbors required for each sample for hyperparameter tuning.

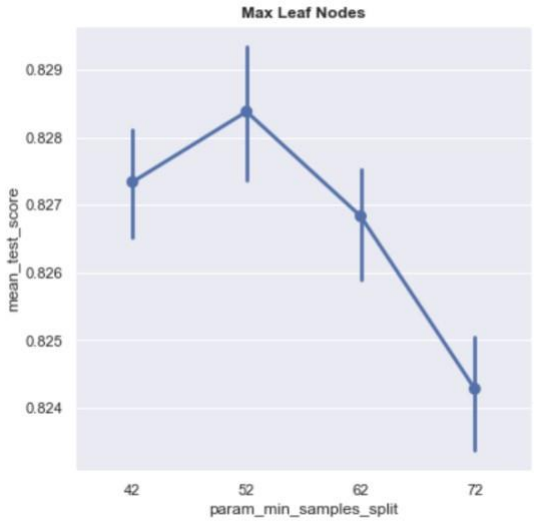


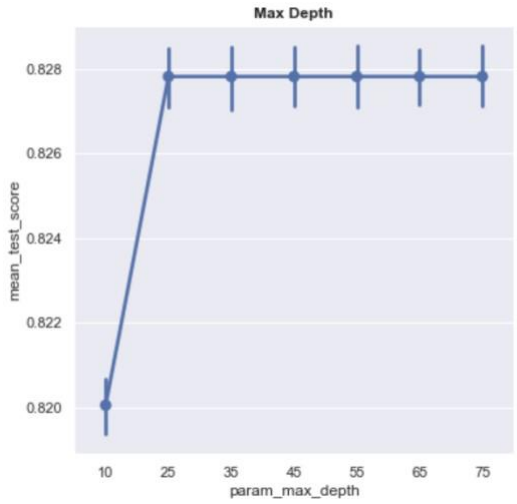
I generated a point plot to see the effect the parameter n_neighbors had on the model performance. As we can see from the plot, model performance is the highest when n_neighbors is three, and that is what GridSearchCV also recommended using, so I used this hyperparameter value in the model.

3) **Decision Tree (Algorithm -2)**: Model that predicts the value of a target variable by learning simple decision rules inferred from the data features. Initially, the model was **overfitting** the data, so to make the model generalise well and improve the performance, I used hyperparameter tuning. I considered the following hyperparameters:

- max_depth: The maximum depth of the tree.
- min_samples_split = The minimum number of samples required to split an internal node - min_sample_leaf = The minimum number of samples required to be at a leaf node.

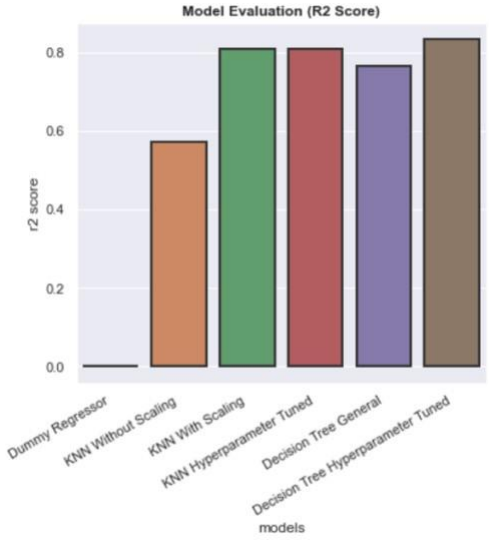
I followed the same steps for hyperparameter tuning as KNN, i.e. RandomSearchCV followed by GridSearchCV.

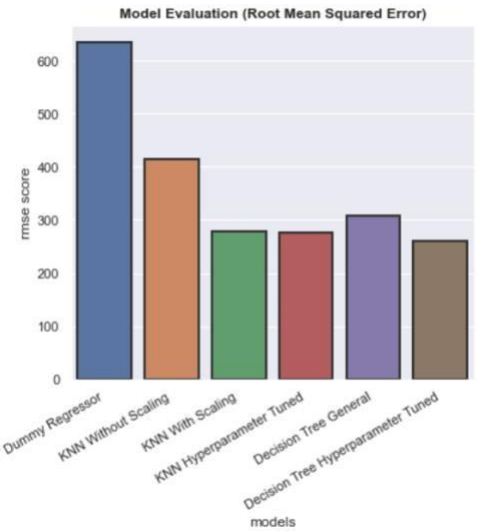




These are a few of the hyperparameter results obtained. I plotted the GridSearchCV results (range of values) to visualise the scores varying with different hyperparameter values. I used the recommended hyperparameter values in the model.

Both the models are hyperparameter tuned and fitted. I created a dataset of all R square scores and RMSE each model has produced and plotted it to select the best model for test set evaluation.

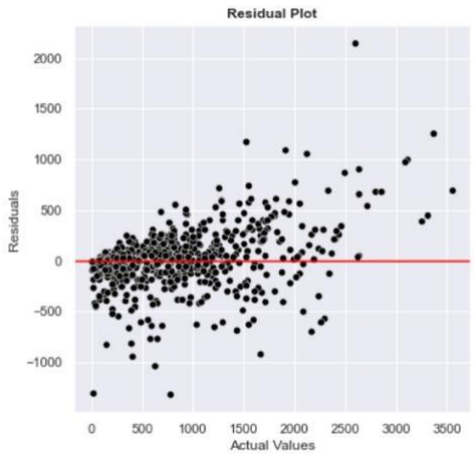


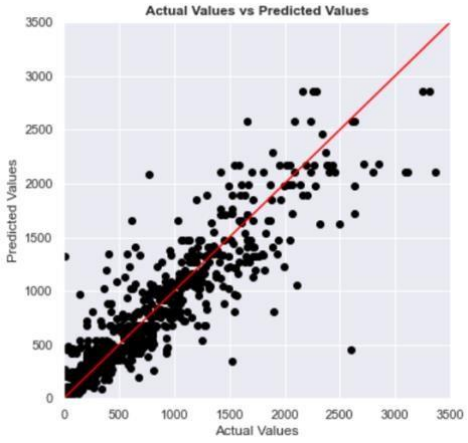


From the above plots, both the hyperparameter tuned models, KNN and Decision Tree, perform better, followed by KNN with scaling and other models. All these models are evaluated based on the R square score (higher is better) and RMSE (Lower is better). **The hyperparameter tuned models reduced the error by 60% and increased the R square score by 82% than the baseline model**. As it is clear from the plots, I selected **Decision Tree Hyperparameter tuned model to be evaluated in the test set**.

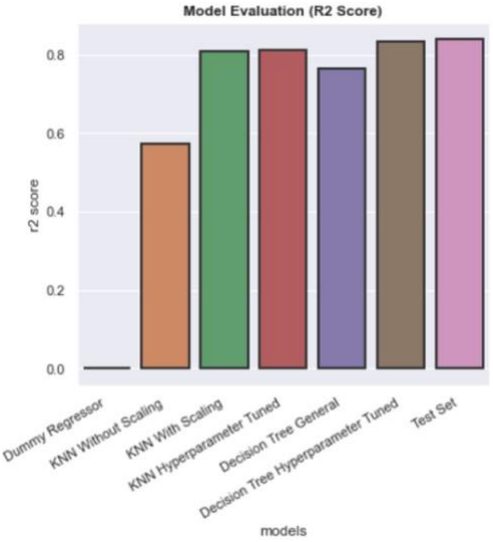
3. Test Set Evaluation

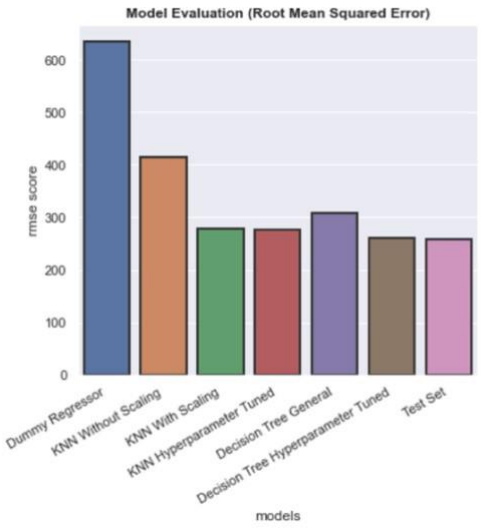
The same hyperparameter values are taken and are used to predict the test set **(unseen set)**.





For the test set, we can see in the residual plot that significantly fewer actual values are overestimated or underestimated by the model. We can see from the regression plot that data points are concentrated closer to the regression line, meaning our model fits the predicted values closer to the actual values.





The above plots are the same ones created for model selection with the test set added. The test set outperforms every other model, even the selected model, which means **the selected model is generalising well with unseen data**. The test set performed the best compared to the other models.

Model Evaluation table

Model	Train R square score	Validation R square score	RMSE	Performance
Dummy Regressor	0.0	-0.000008	634	Very Poor
KNN Without Scaling	0.69	0.57	415	Poor
KNN With Scaling	0.87	0.80	277	Good
KNN Hyperparameter Tuned	0.90	0.81	276	Good
Decision Tree	1.0	0.76	307	Poor
Decision Tree Hyperparameter Tuned	0.88	0.83	259	Very good
Test Set	0.88	0.83	258	Very good

This is the model evaluation table comprising all the models and their respective results.

Overall, the results show that hyperparameter tuning generalises the selected model and improves its performance.

Thank You