17-634: Dataset Model Training & Performance Evaluation

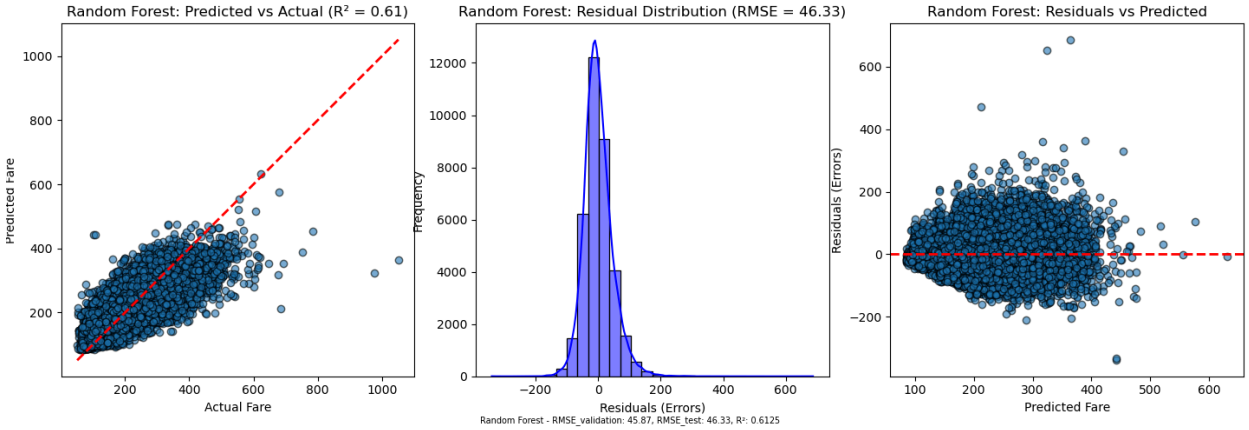
Xuchao Zhou

Partition of Dataset

In order to meet the requirements for the training and testing of the machine learning models, and evaluating their performance, I partitioned the original dataset into three subset: data_train (70%), data_test (15%), and data_validation (15%). I used simple random sampling because this is a straightforward way of splitting data, it ensures that no biases are introduced, and there are no subgroups within the dataset so stratified random sampling is not needed. As the name of each set shows, I will use data_train for training the models, use data_validation for the validation process during training, and use data_test to test the model's performance on unseen data after it is fully trained. The 70:15:15 ratio provides a good balance between adequately training the model and having enough data for the validation and testing processes.

Model Comparison

The model should be predicting the fare from the 8 given features: Year, quarter_1, quarter_2, quarter_3, quarter_4, nsmiles, passengers_log, and route. All features have already gone through transformation or encoding processes suggested in the previous homework (Feature Engineering). I begin by a random forest model - I used RandomForestRegressor from the sklearn package in Python and I set 100 trees and a max depth of 10. After training the model, this is the result I got from testing:

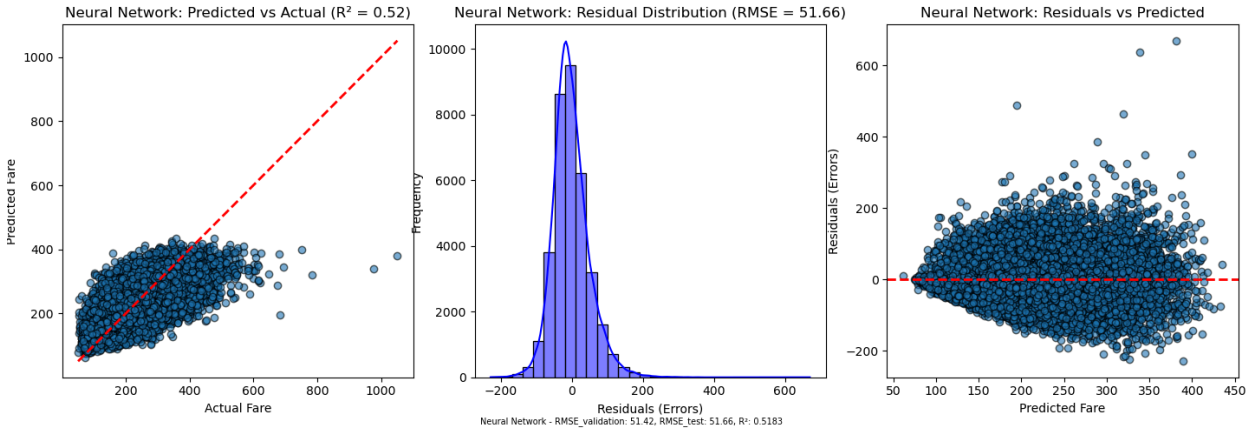


From these figures we can see that the performance of this model is moderately strong, as in the first graph the points are around the 45 degrees line, the second graph shows the residuals are centered around 0, and the third graph does not show any specific pattern (no overfitting). The RMSE for validation and testing are similar, indicating that the model's performances on seen and unseen data are similar, and around 46 dollars off from the predicted fare is acceptable when predicting air ticket price. The 0.6125 R-squared value also shows that the majority of variation in fare can be explained by the model.

17-634: Dataset Model Training & Performance Evaluation

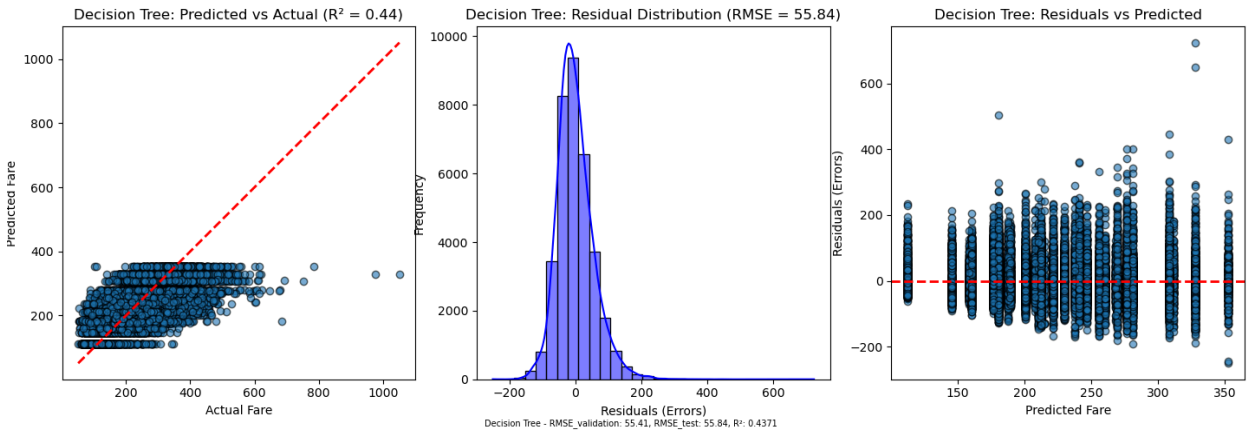
Xuchao Zhou

We then move on to a neural network model by defining an MLPRegressor with two layers - first layer with 64 neurons and second layer with 32 neurons. I trained this model for up to 500 iterations. The features are the same as the random forest model. This is the result I got from this model:



The graphs look quite comparable to the random forest model, but we notice that the residuals have a larger range (more scattered). This is also seen in the RMSE and R-squared statistics - larger RMSE and lower R-squared values indicate that this model's performance is slightly worse than that of the random forest model.

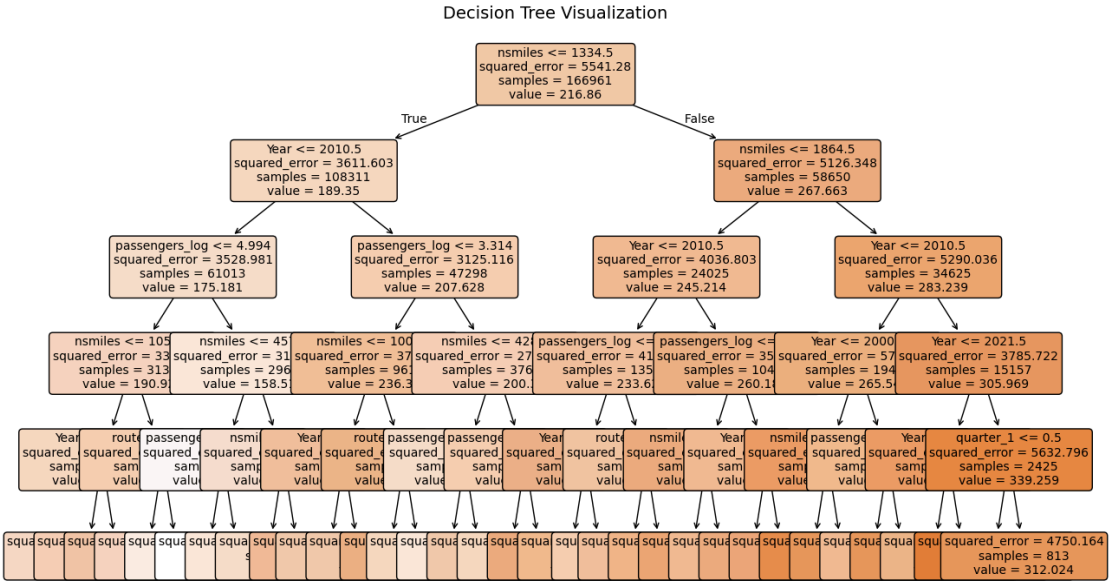
After this, we try some simpler models. The attached figures below shows the performance of a DecisionTreeRegressor with maximum depth of 5:



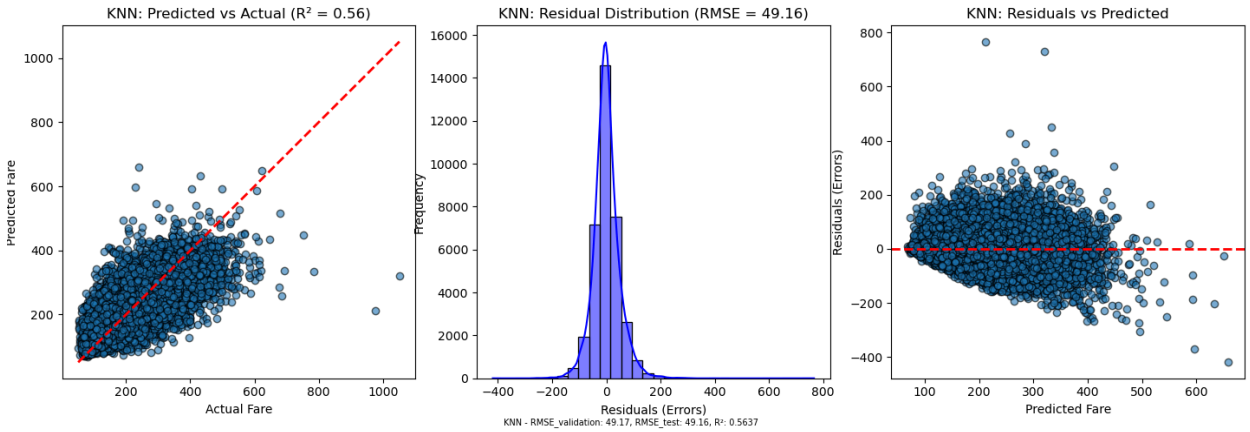
These graphs are quite comparable to the ones for the neural network model. One thing to note is that the residuals seem to form vertical lines, which is because of the approach of a decision tree - the results are actually several categories instead of directly calculating a value. The R-squared statistics is 0.4371, which is relatively low as it indicates only 43.71% of the variability in fare is explained by the decision tree, which means that its performance is relatively poor. However, there is one advantage of this model - its process is transparent and can be easily show by a graph:

17-634: Dataset Model Training & Performance Evaluation

Xuchao Zhou



The last model we trained is a k-nearest neighbor model. We use KNeighborsRegressor and set the number of neighbors to 5, as well as using ‘weights=distance’ to give more importance to closer neighbors. The performance of this model is shown below:



These graphs show that this KNN model performs quite well. The residuals are relatively small and are equally distributed on the right-most graph (no overfitting). The RMSE and R-squared statistics are also very close to those of the strongest model (random forest).

Model Selection

17-634: Dataset Model Training & Performance Evaluation

Xuchao Zhou

Among the four models I tried, two of them have outstanding performance: the random forest and the k-nearest neighbors. Because the performance statistics of these two models are roughly the same and both models have no obvious flaws (e.g. overfitting), we tend to choose the model with less complexity. Therefore, I recommend using the k-nearest neighbor model for deployment.

Before deploying this model, we can do some improvements to both the model and the data to improve the model performance. For the model, we can consider optimizing the number of neighbors by using cross-validation to find the best k-value; we can also try different distance metrics - KNN uses Euclidean distance by default, but Manhattan or Minkowski distance may perform better. For the data, we can consider removing or transforming extreme values in nsmiles and passengers_log as KNN is sensitive to outliers; if possible, using larger training dataset would also be an option because KNN benefits from more data to make better neighborhood comparisons.