DIAMOND REGRESSION DATA SET DISCUSSION

Exploratory Data Analysis:

The EDA that I undertook for this part of the assignment mostly involved the traditional checks of missing data, outliers, and anything that appeared to be incomplete. I pre-processed by turning all of the string values into numeric ones. For this part of the assignment I did not split the string values into the binary format (e.g. gender-female / gender-mal as was changed in the second part of the assignment), but because of the inherent ranking of quality of diamonds in terms of price I replaced the string values with a rank where 1 was the best and it counted down to the worst value, rankings which were taken from the csv details on kaggle about the dataset.
I looked at the data distribution as well and noted that there appeared to be a large amount of skew in some of the data features (y and z in particular). Taking a log transformation of the data would have helped this skew and given it a more normalised or Gaussian distribution but due to the low values in general (<60 for y and <30 for z) I decided not to take a log transformation of this data set. I also ran the various regression algorithms on the dataset and the results were strong enough I decided not to take the log transformation. The dataset itself was very neat, without any missing values or outliers, so there was not a huge amount of data cleaning that needed to take place.
With the data exploration I noted that x, y, z and carat had the highest correlation with each other and the price of the diamond. Depth and table both had very weak correlations with the other data and also with the price. Due to the small number of features I decided not to employ PCA on this dataset, instead keeping the dimensionality as it was to give the most general view of the dataset and also to ensure that all factors would have some use in the final regression models.
The final step of data pre-processing that I undertook (aside from dividing the data up into test and training using the 309 seed) was I standardised the data across the board. This was something that I did in the beginning for some algorithms but not others, but once all of the data was standardised the results of the algorithms improved significantly. I did not change the data any more than this for any of the algorithms, finding that some methods (such as normalisation for the KNN regressor) decreased all of the performance metrics that the model was evaluated on some significantly, so standardised data was the best to use overall.

Parameter Tuning:

For Linear Regression, SGD and Ridge Regression I did not tune any parameters, as the basic algorithm settings worked well enough that I was content with the output and it did not seem like it needed tuning.
For KNN I used the neighbours set to 21, as this was a number that gave strong results for the R2 while still running quickly enough that the algorithm did not take too long. It was an improvement on the other neighbours I tried in the same range, and is also a commonly used value that has decent performance across the board on other algorithms. I set the weights to use the distance metric as this gave a lower MSE, RMSE and MAE value while taking roughly the same amount of time and giving a higher R2 value as well (uniform weights = 0.9585, distance = 0.9609)
For the Decision Tree, Random Forest and Gradient Booster I kept the impurity decrease at 1.0. This was to avoid overfitting the tree on the training data, to prune the tree as it was built to ensure that it did not become too large and too confusing, and also because it gave a good, balanced result for the performance metrics. It also decreased the time taken to build the decision tree. For random forest I used 20 estimators to decrease the amount of time taken to run the algorithm, which reduced the random pool of the trees but it also gave enough difference that there was a strong

performance across the performance metric. Gradient Boosting also had the parameter change of the learning rate included, which I set to 0.2 as per most suggestions in Comp 307. It also performed well.
For the two SVR algorithms I left most of the parameters as default but I adjusted the c value up to 1500. When running the SVR without a C set the algorithm performed poorly, achieving an R2 score of ~0.5. By setting the C value up to 1500 I achieved a far better performance, even more than the C=1000 setting, by increasing the variance within the model and lowering the bias so that it would perform better and be better able to generalise to the unseen test data. I left the gamma setting as it was already set though, as it was small enough that it worked with the increase in C instead of against it. The C value was also small enough that I balanced the threat off too high C causing overfitting.
For the perceptron I used general settings, increasing the number of iterations as the default of 200 was not enough for the perceptron to converge. I set momentum down from 0.9 to 0.2 to make smaller changes overall instead of the larger changes in gradients which can come from a large momentum or learning rate, and set the solver to lbfgs as according to the documentation on sci kit learn it was better for smaller numbers of features and in the grand scheme of ML 10 features is incredibly small.

Results:

Note: All values were taken while using a 5 fold cross validation, hence the high execution times

Regression Algorithm	R^2	MSE	RMSE	MAE	Execution Time (seconds)
Linear Regression	0.91	1529981.84	1240.92	820.12	0.238
K-Neighbours Regression	0.96	672882.01	820.29	420.73	1.408
Ridge Regression	0.91	1543886.19	1242.53	820.59	0.111
Decision Tree Regression	0.96	646264.74	803.91	399.33	0.662
Random Forest Regression	0.97	370080.48	608.34	399.33	7.919
Gradient Boosting Regression	0.98	349351.21	591.06	298.18	9.917
SGD Regression	0.90	1575047.71	1255.01	851.88	0.175
Support Vector Regression (SVR)	0.97	436738.85	660.86	321.52	241.937
Linear SVR	0.88	1917067.18	1384.58	713.47	6.525
Multi-layer Perceptron Regression	0.98	377779.12	614.64	306.49	585.552

Comparison of algorithms:

Gradient Boosting was the algorithm which performed the best overall on the diamonds dataset. This is because of the way in which it builds the regressor, creating regression trees in each stage of the algorithm to fit on the negative gradient. By default it uses the least squares regression as the cost function which provides a strong overall model due to minimising the value which is prevalent through each of the formulae to calculate the various performance metrics. It was not the quickest due to creating the regression trees with each pass, but it had the highest performance or equal highest for all four metrics.

Decision Tree and Random Forest regressions also performed well across the board, getting the same MAE as each other while the Random forest performance was generally better than the DT. This is due to the random forest creating a number of DTs and picking the best one, allowing for the best performance overall against the single DT that will be created as the regressor runs through the algorithm on the dataset. The impurity split being kept to above 1.0 also ensured that the trees would not overfit to the training set, allowing it to work well on the test set as well. DT and Random Forest also use the MSE criterion to optimise when building the trees measuring split quality, leading to an overall high performance.

Ridge had roughly the same performance as linear regression in this dataset due to how Ridge works by penalising the features which do not have a large impact on the final output. It works with a sum of squares error term and a penalty term to give the regression line, and an advantage it would have on a large dataset with many more features is that it would punish the less important features and give more weight to those features which had more of an impact on the data. Due to the small number of features in this dataset Ridge's regression performs the same as linear, with the penalties becoming less impactful than they would be in a bigger data set.
The linear SVR model had the worst results across the board apart from the execution time. It had a 0.03 percent worse R2 than the nearest low algorithm (Ridge's) and a much lower MSE, MAE and R2 than the non-linear SVR, which used the default setting of RBF. This is due to the correlation between features within the dataset throwing off the way in which linear SVR worked to calculate errors, grouping together the closer features and ignoring the lesser features in a similar way to how the Ridge's and Linear regression models worked. Linear SVR performed even worse than these two due to the fact that it needed to project the features to a higher dimension in order to linearly separate them, as with all SVR algorithms, which was more difficult due to the high dependence between the features and thus gave a higher MSE which resulted in lower R2 values, along with the higher MSE and MAE.