Project 4 : Regression Analysis and Define your own Task

Sidarth Srinivasan (UID - 005629203), Shweta Katti (UID - 505604846)

March 18, 2022

Course : ECE 219 Large Scale Data Mining
Term : Winter 2022

In this project we explore regression analysis which is a powerful statistical method that allows us to examine the relationship between two or more variables of interest. We use regression analysis to assess the strength of the relationship between variables and for modeling the future relationship between them.
Regression analysis includes several variations, such as linear, multiple linear, and nonlinear. The most common models are simple linear and multiple linear. Nonlinear regression analysis is commonly used for more complicated data sets in which the dependent and independent variables show a nonlinear relationship.

In the first part of this project, we explore with two datasets provided to us:

- Diamond Characteristics Data Set
- Gas Turbine CO and NOx Emission Data Set

In the second part of the project we define our own task and work with twitter data set to build an end-to-end ML pipeline to achieve the goal of our chosen task.

1 Question 1

As suggested by the given task, we perform Standardization on our datasets to make them appear as a standard normally distributed data i.e; Gaussian with zero mean and unit variance. We need to perform this to avoid the dominance of a feature over the other features on our objective function. If a feature has a variance that is orders of magnitude larger than others, it might hold a greater weightage which might lead our estimator to learn incorrectly.

We standardize only the training data. We do this by calling a $StandardScaler()$ function on the dataset using fit and transform together on the training sets to mitigate skewness of the features. Mathematically, standard scaling subtracts the feature column from its mean divided by the standard deviation of the feature.
$Z = \frac{x - \mu}{\sigma}$

This also helps reduce sensitivity of the estimator to outliers, improves convergence speed, removes adverse effects of biased regularization term and leads to consistent, reliable results from the predictive model.

2 Question 2

Correlation coefficients are used to measure how strong a relationship is between two variables. There are several types of correlation coefficient, but the most popular is Pearson's. Pearson's correlation is a correlation coefficient commonly used in linear regression.

The value of this Pearson's correlation coefficient ranges from -1 to +1. A -1 means that there is a negative correlation between the variable, increase in one variable results in the reduction of the other variable value and a +1 means that the variables have a positive correlations, increase/decrease in a variable results in the simultaneous increase/decrease in the other correlated variable.

For diamond characteristics dataset, the heatmap is shown in Figure 1. Features x,y,z and cut have the highest absolute correlation with the target variables price. It demonstrates that the cut along with its dimensions is most informative in terms of predicting the price of the diamond.

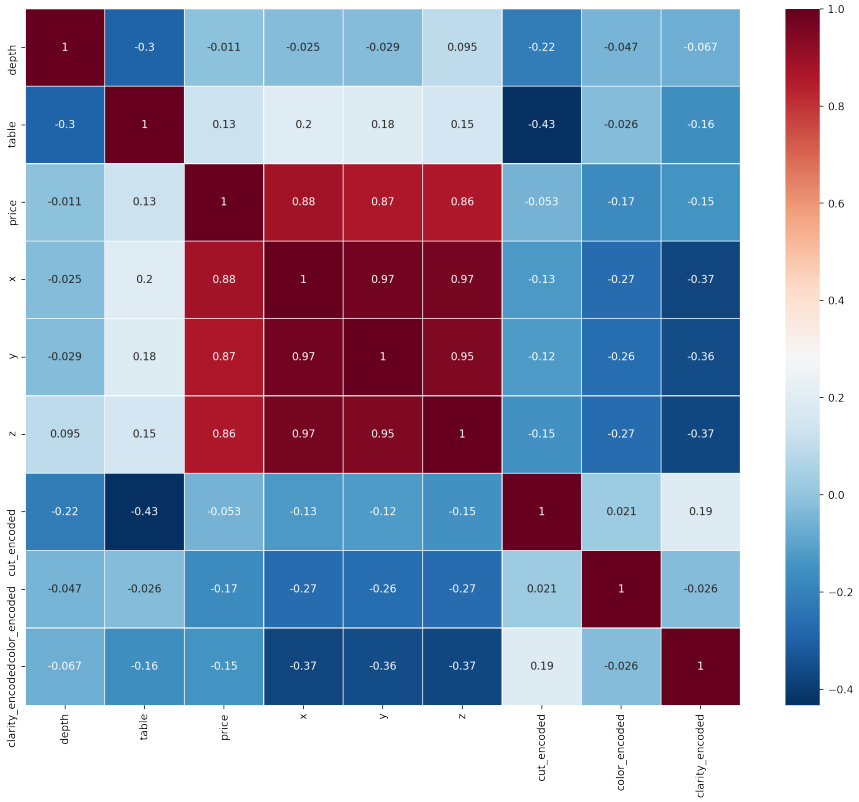


Figure 1: Heatmap for Diamonds Dataset

For gas emissions dataset, the heatmap is shown in Figure 2). Features AP and AH have the highest absolute correlation with the target variables CO and NOX. It demonstrates that the AP and AH values are most informative in terms of predicting the CO and NOX values. Our chosen target variable is NOX for the given gas emissions dataset.

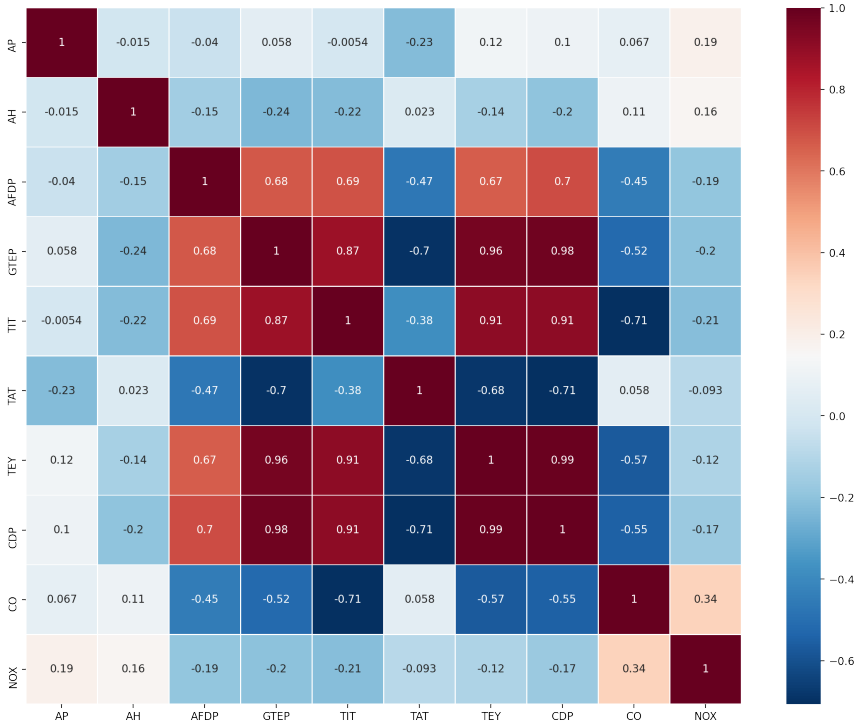
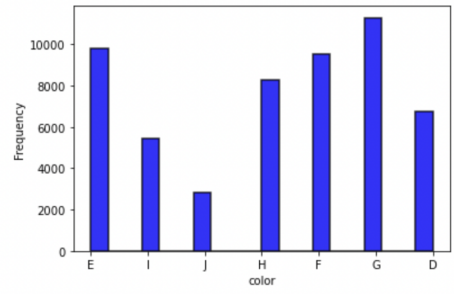


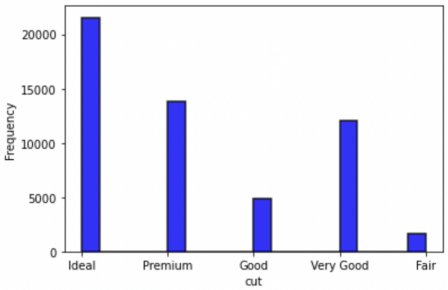
Figure 2: Heatmap for Gas Emissions Dataset

3 Question 3

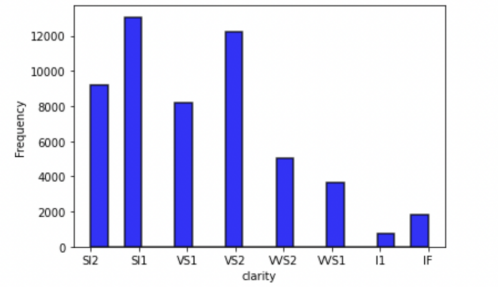
For diamond dataset, Figure 3 shows the histograms of all the numerical values against the frequencies. From the figures, The feature Price has a very high skewness. Preprocessing step like standardization, as discussed earlier has to be performed on those highly skewed features ahead of model fitting, otherwise extreme values will cause great harm to the robustness of the model.



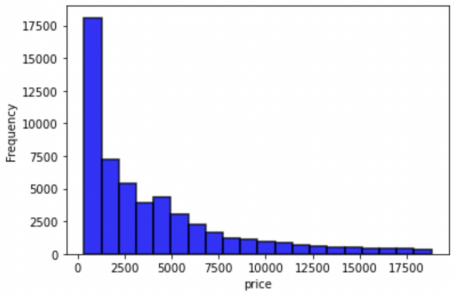
(a) Color



(b) Cut



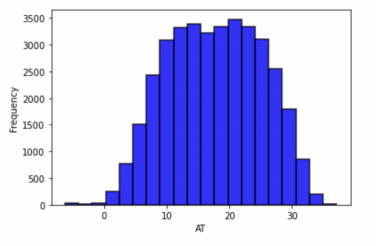
(c) Clarity



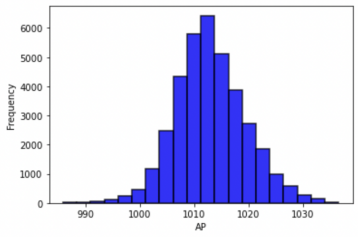
(d) Price

Figure 3: Histograms for Diamond Dataset

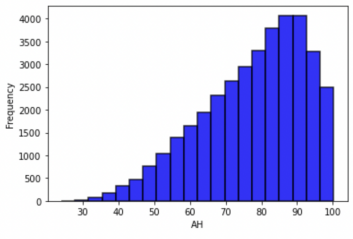
For gas emissions dataset, Figure 8 shows the histograms of all the numerical values against the frequencies. Obviously, there exists a bunch of numerical features with high skewness, including TAT, CO etc. Preprocessing step like standardization has to be performed on those highly skewed features ahead of model fitting, otherwise extreme values will cause great harm to model robustness.



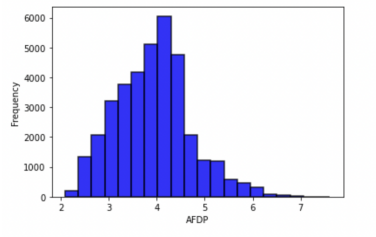
(a) AT



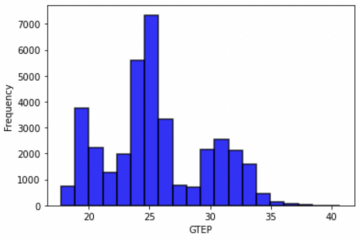
(b) AP



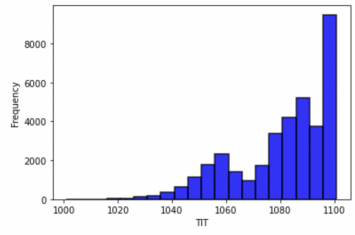
(c) AH



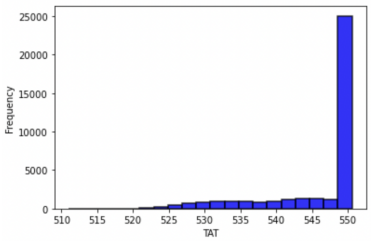
(d) AFDP



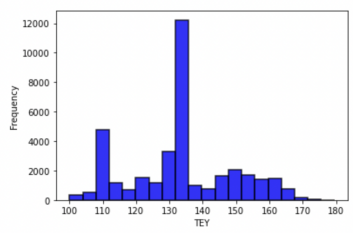
(e) GTEP



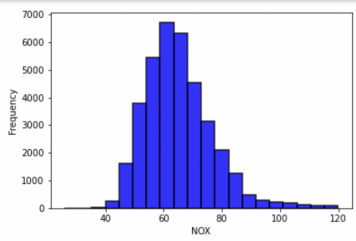
(f) TIT



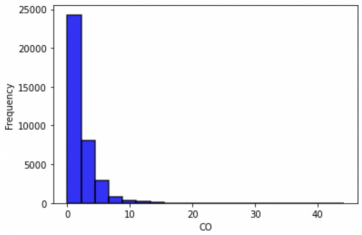
(g) TAT



(h) TEY



(i) NOX



(j) CO

Figure 4: Histograms for Gas Emissions Dataset

We should try and avoid features with high skewness as:

- The mean can't be used to represent the entire distribution accurately.
- They limit what kinds of valid statistical analysis one can perform on the said features.

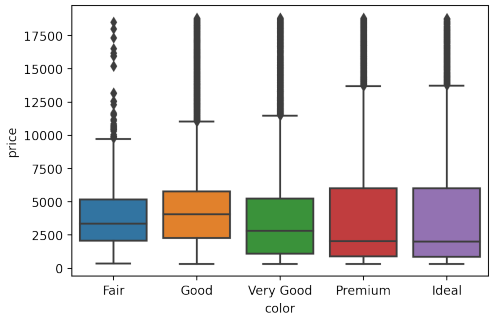
- It also leads to inaccurate feature influence estimate for the model predictions.

The following pre-processing techniques can be used for features with high-skewness:

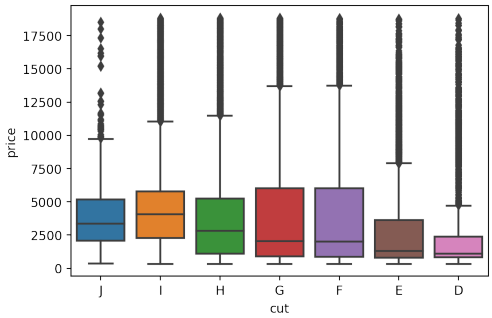
- Perform transformations such as square, cubic, logarithmic transforms for features that are positively skewed and power transforms for features that are negatively skewed.
- Remove/eliminate the influence of outliers by dropping them.
- Normalization with scaling or simply, **Standardization**.

4 Question 4

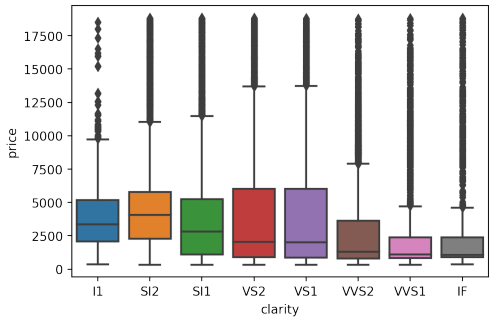
The following boxplots were observed for diamond dataset (Figure 5) and for the gas emissions dataset (Figure 6). The boxplots are observed for the categorical variables and the target variables for each dataset.



(a) Color against Price



(b) Cut against Price



(c) Clarity against Price

Figure 5: Boxplots for Diamond Dataset

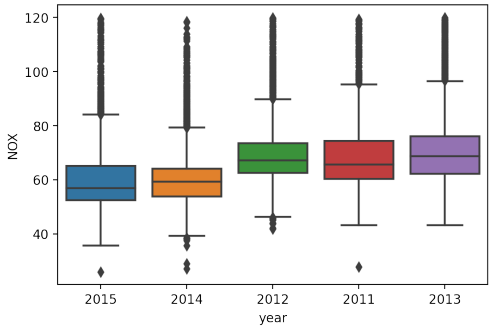
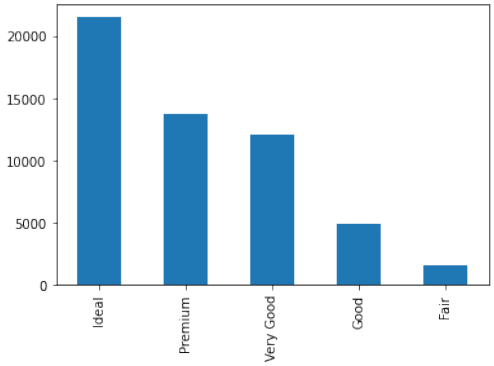
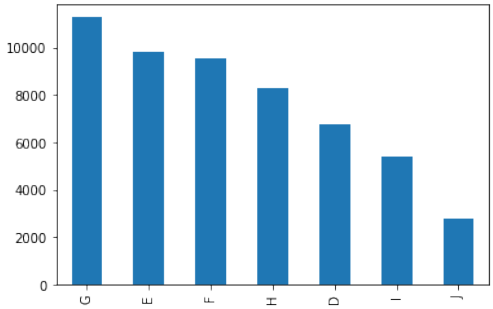


Figure 6: Boxplot for Gas Emission Dataset (NOX vs year)

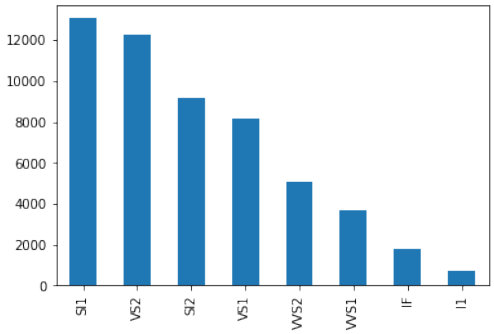
5 Question 5



(a) Histogram of counts for Color Feature



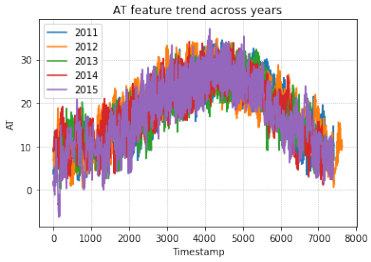
(b) Histogram of counts for Cut Feature



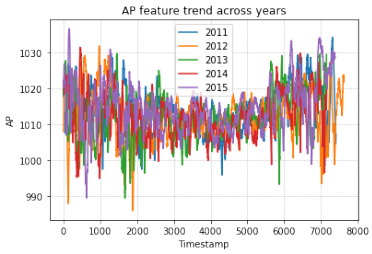
(c) Histogram of counts for Clarity Feature

Figure 7: Histogram of counts for different features

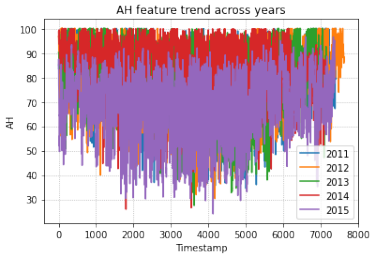
6 Question 6



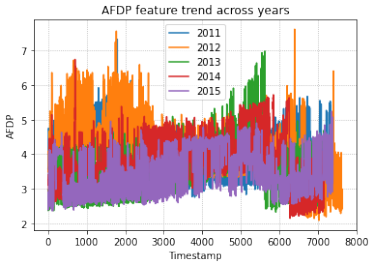
(a) AT



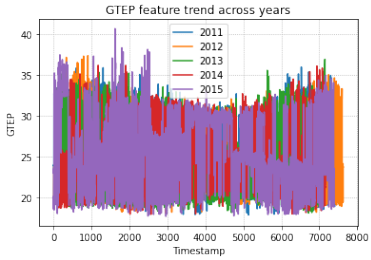
(b) AP



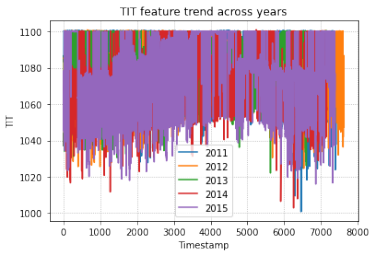
(c) AH



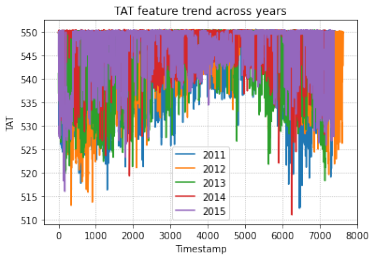
(d) AFDP



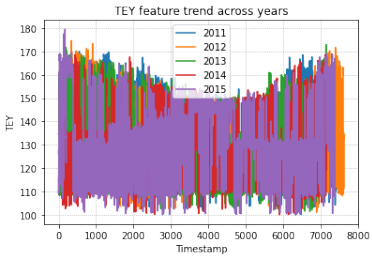
(e) GTEP



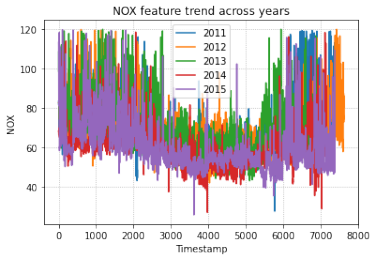
(f) TIT



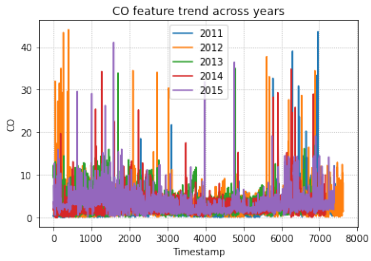
(g) TAT



(h) TEY



(i) NOX



(j) CO

Figure 8: Yearly Trends for Gas Emissions Dataset

7 Question 7

- Mutual information (MI) and F scores functions are used to select the most important features. This step is extremely useful as selecting just a part of input features helps in boosting the model performance for testing, especially for linear models.
- In simpler words, this helps us avoid the problem of overfitting by not considering the less relevant or redundant features.
- Mutual information is calculated between two variables and measures the reduction in uncertainty for one variable given a known value of the other variable. The mutual information between two random variables X and Y can be stated formally as follows:

$$I(X, Y) = H(X) – H(X|Y)$$

- The F1 score can be interpreted as a harmonic mean of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is:

$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

The effect of feature selection on Diamonds dataset can be seen in Figure 15

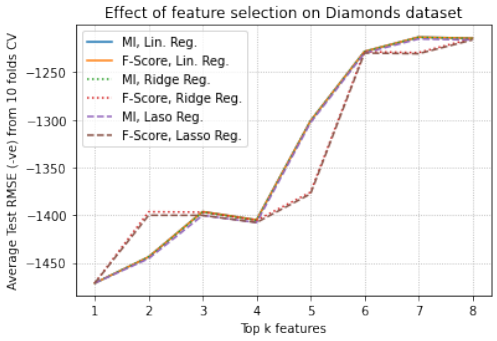


Figure 9: Effect of Feature Selection on Diamonds Dataset

The effect of feature selection on Gas Emission dataset can be seen in Figure 10 below:

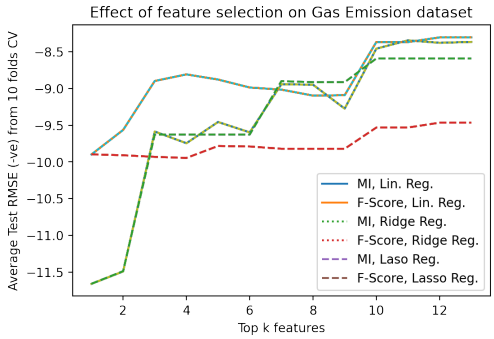


Figure 10: Effect of Feature Selection on Gas Emissions Dataset

From the above plots, we can infer that:

- For linear, ridge and lasso, we can say that the test score increases and converges at a given point. This is due to the fact that the top features have already conveyed the required information at this point of convergence. Feature selection reduces the complexity of the model while making it highly interpretable.
- MI is able to capture non-linear relationships between features and target variables but is slower than F1-score and requires more samples.

8 Question 8

The objective functions for OLS, Lasso and Ridge are as defined below:

- Ordinary Least Square : $\min_{\beta} ||Y - X\beta||_2^2$
- Lasso : $\min_{\beta} ||Y - X\beta||_2^2 + \lambda||\beta||_1$
- Ridge : $\min_{\beta} ||Y - X\beta||_2^2 + \lambda||\beta||_2$
- Lasso regression is Linear regression with L1 regularization
  - Lasso regression is suitable for feature selection and construction of simple and sparse regression models, where features with zero weights are discarded. This is because the L1 regression intends all the weights to shrink to 0, regardless of the size of w. L1 regularization is more likely to shrink the weights to zero than L2 for similar test accuracies because it assume priors on the weights sampled from an isotropic Laplace distribution, which has a much lower Q factor than Gaussian distribution (exponential descent) and thus only a subeset of features are active in the learned hypothesis.
- Ridge regression is linear regression with L2 regularization
  - Ridge regression assumes priors on the weights sampled from a unit Gaussian distribution, is suitable for reducing the effects of collinear features, which can lead to increased variance of the model. The is because the subgradient of the norm of w depends not just the sign of w but also the magnitude of w, which can be thought of as scaling the variance of weights, reducing dependence of the model on new features and encouraging distributed contribution of all the features. This leads to a regression models with diffuse weights compared to sparse weights from L1 regularization. Hence ridge regression promotes participation of all the features in the learned hypothesis to prevent overfitting.
- Hence, L2 regularization term serves for shrinkage purpose while L1 can be used for feature selection or screening purposes.

9 Question 9

- For this question, we analyze the performance of train and test RMSE of linear regression, lasso regression and ridge regression on top 6 selected features from diamonds dataset and top 10 features from gas emission dataset.
- To compute the optimal penalty parameter for each regularization scheme as well as test the performance of the model, we construct a pipeline scheme by performing a grid search with 10-fold cross validation with λ in the range of $[10^-5, 10^5]$. We also used LassoCV and GridCV() to calculate the optimal penalty terms. We further explored the effects of feature scaling/standardization that is explained in the next question.

- For the diamonds dataset, the results can be observed from Table 1
- For the gas emissions dataset, the best test RMSE is obtained with lasso regression, giving a test score of -8.363 on features selected via F1 with an optimal penalty parameter of 0.01 and standardization/feature scaling enabled. The values have been summarized in Table 2

10 Question 10

From Table 1 and 2 above , we can see that feature scaling does make some difference in the RMSE when regularization is used, as our best RMSE is obtained with

Feature scaling does not cause any changes in the original dataset distribution. Without regularization, the change in values caused by feature scaling will be reflected by corresponding changes in the weights of the model to achieve the lowest RMSE. Since the data distribution is the same as before, the changes caused by feature scaling will reflect linearly on the changes in the coefficients, yielding no performance gains or losses. As a result, we see no change in test RMSE with or without feature scaling when regularization is absent.

As discussed in Question 3 , standardisation of data will help deal with skewness in the distribution. Consider two salient features, one which has a much smaller range and absolute value than the other one. Without normalization, the coefficients of these two features will be unbalanced, with the smaller feature having a larger coefficient than the larger feature in order to have a balanced outcome on the target variable. Since regularization aims at minimizing the weights of the regression model, it would remove or penalize the coefficients of the smaller feature, while having negligible effect on the weights of the larger feature. Feature scaling ensures that the weights assigned to each feature does not get adversely affected by a biased regularizer penalizing smaller feature when regularization is used and leads to more stable and well-conditioned models.

Diamonds Dataset					
Standardized	Feature Selection	Regression	Mean Train Score	Mean Test Score	Optimal Penalty Parameter
No	F1	Linear	-0.30621210838602225	-0.3078705218517502	-
No	MI	Linear	-0.30621210838602225	-0.3078705218517502	-
Yes	F1	Linear	-0.30621210838602225	-0.3078705218517502	-
Yes	MI	Linear	-0.30621210838602225	-0.3078705218517502	-
No	F1	Ridge	-0.3078705225656322	-0.3062121083860223	10^-5
Yes	F1	Ridge	-0.30787052199893694	-0.30621210838602225	10^-5
No	MI	Ridge	-0.3078705225656322	-0.3062121083860223	10^-5
Yes	MI	Ridge	-0.30787052199893694	-0.30621210838602225	10^-5
No	F1	Lasso	-0.3078847962874626	-0.30621215803483404	10^-5
Yes	F1	Lasso	-0.30787910228154536	-0.3062121397648972	10^-5
No	MI	Lasso	-0.3078847962874626	-0.30621215803483404	10^-5
Yes	MI	Lasso	-0.30787910228154536	-0.3062121397648972	10^-5

Table 1: Tabulated grid search results for diamonds dataset

Gas Emissions Dataset					
Standardized	Feature Selection	Regression	Mean Train Score	Mean Test Score	Optimal Penalty Parameter
No	F1	Linear	-7.891748070474053	-8.366168740813503	-
No	MI	Linear	-7.907643923208172	-8.456203306935215	-
Yes	F1	Linear	-7.891748070474053	-8.366168740813615	-
Yes	MI	Linear	-7.907643923208171	-8.45620330693624	-
No	F1	Ridge	-7.891748070474053	-8.364744402570944	100
Yes	F1	Ridge	-7.891748070474053	-8.366168741751999	10^-5
No	MI	Ridge	-7.907643923208172	-8.37780683768749	100
Yes	MI	Ridge	-7.907643923208175	-8.448143474001927	10
No	F1	Lasso	-7.891748071298531	-8.366196815773336	10^-5
Yes	F1	Lasso	-7.891748071400175	-8.363833147115464	0.01
No	MI	Lasso	-7.891748071298531	-8.364559939891265	10^-5
Yes	MI	Lasso	-7.907645379449599	-8.419347246809437	0.01

Table 2: Tabulated grid search results for gas emissions dataset

11 Question 11

The p-value for each feature tests the null hypothesis that the feature has no correlation with the target variable i.e, P-values in the linear regression model measures the probability of feature coefficients being equal to zero. Hence, if the p-value for some feature is very close to 0, we will have the confidence to say that particular feature is significant in the linear model.
For the diamond dataset, the p- values were as follows:

Diamonds Dataset	
Features	p-value
carat	0.000000e+00
color encoded	0.000000e+00
clarity encoded	0.000000e+00
cut encoded	1.824802e-169
depth	4.887995e-140
x	3.822643e-128
table	2.154390e-10
z	4.843589e-03
y	6.604406e-03

For the diamonds dataset, the p-values obtained were as follows:

Gas Emissions Dataset	
Features	p-value
AT	0.000000e+00
AH	0.000000e+00
TEY	1.062195e-149
AP	1.141390e-90
CDP	2.773263e-57
GTEP	4.397377e-10
TIT	3.900569e-08
year 2013	1.611442e-07
year 2012	2.026209e-07
year 2011	4.491527e-07
year 2014	4.418688e-06
year 2015	2.505455e-05
AFDP	9.734913e-05
TAT	2.104505e-01

12 Question 12

The most salient features are those with greatest absolute coefficients. We performed a gridsearch with 10-fold cross validation on each dataset to find these values

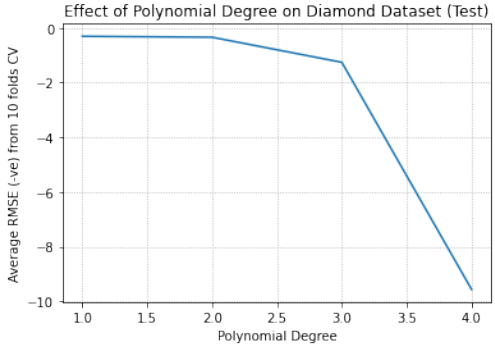
For the diamonds dataset, the top 5 salient features are: ['carat', 'x', 'clarity encoded', 'z', 'color encoded'] of which x and z also have the highest absolute correlation with the target as seen previously in the heatmap generated.

For the gas emissions dataset the top 5 Salient features were as follows: ['AT', 'TIT', 'CDP', 'AH', 'GTEP'] of which AT and AH also have the highest absolute correlation with the target as seen previously in the heatmap generated.

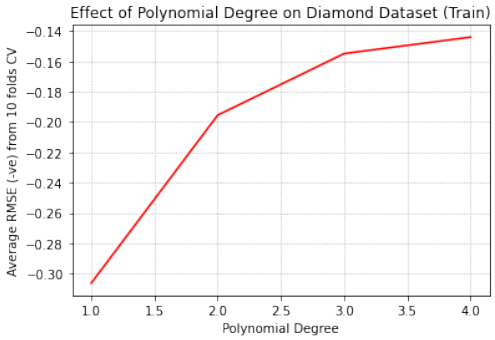
13 Question 13

- We tested the performance in terms of train and test RMSE of polynomial regression of various degrees on top 6 selected features in the diamonds and top 10 features for the gas emissions datasets.
- Since F1-Score performed better for feature selection than MI, we selected top 6 and 10 features respectively from F1.
- We then used ridge regression to incorporate L2 regularization term to prevent overfitting. We optimized the penalty parameter by using grid search with 10-fold cross validation.
- The optimal degree of polynomia we selected for the diamonds dataset is 2. From the test RMSE error, we can infer that the error is minimum for degree 2 and hence any degree above that might result in overfitting.
- The optimal degree of polynomial we select is 2 for the gas emissions dataset. From the test performance we can observe that starting from the polynomial degree of 2, test RMSE has an increasing trend, which results in overfitting. Thus we should stop increasing the polynomial degree beyond this point.
- Further we cannot increase the polynomial degree in an unconstrained manner because it can lead to exponential explosion of parameters to be stored in the memory and thereby can cause bottle neck to

the computational capacity as well. Although higher polynomial models have a larger model capacity, higher polynomial degrees lead to a greater possible combination of features from raw features which can lead to overfitted and complicated models. We can observe this trend for both datasets with the test RMSE error exploding as polynomial degree increasing.

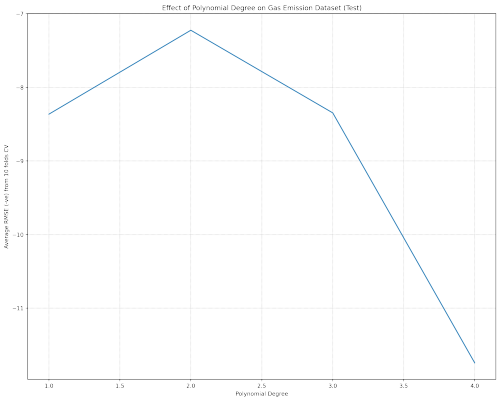


(a) Test Set

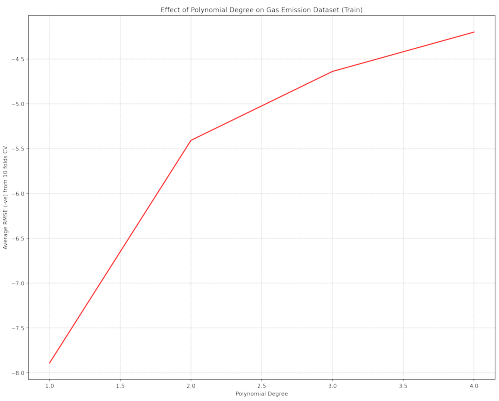


(b) Train Set

Figure 11: Diamonds Dataset



(a) Test Set



(b) Train Set

Figure 12: Gas Emissions Dataset

14 Question 14

Product of particular features might yield a new feature that is highly correlated with the target variable, which would otherwise be impossible to craft using regression. For the diamond dataset, an example of a features whose product should in theory, perfectly correlate with the price is the diamond area which is xy. In other words, product of dimensions will be highly correlated with the price of the diamond. Thus, the equation for our new feature is:

$new\_feature = xy$

We retrained the Ridge Regression obtained in previous question for the diamond dataset with the newly appended feature to the dataset. The obtained metrics are as follows:

Average Test RMSE (-ve) without new feature (degree = 2): -0.2087

Average Test RMSE (-ve) with new feature (degree = 2): -0.2034

We see that the addition of the new feature has caused slight improvements in the test RMSE and proves our hypothesis that the area of diamond is correlated with the price.

15 Question 15

The following results were obtained for the diamonds and gas emissions dataset with MLP, using grid search with 10-fold cross validation:

Dataset	Model	Test RMSE
Diamonds	Linear Regression	-0.302
Diamonds	MLP	-0.142
Gas Emissions	Linear Regression	-8.36616
Gas Emissions	MLP	-6.67871

From the above results, we can clearly note that MLP performs much better than our previous linear regression models.

a. The main reason MLP performs better than linear regression is that neural networks can capture non-linear relationship between features and target while linear regression models cannot.

b. Neural networks have more information capacity than linear regression models.Hence, Multi-layer perceptron model can fit very complicated relationship by introducing multiple hidden layers, thus we simply include all the features in this part.

c. Neural networks do not need explicit transformations or feature engineering proceeses to choose salient non-linear features from the input dataset, they do so automatically.

16 Question 16

To find the optimal hyperparameters, we first create a hyperparameter space to construct a 10-fold cross validation grid search pipeline. The hyperparameter space was as follows:

- Model activation functions : 'logistic', 'tanh', 'relu'

- Weight Decay : $[10^-4, 10^2]$
- Hidden Layer Sizes : Chosen from a combination of 10,20,30,50. The complete list of combinations were as follows:
  $[(10,), (20,), (30,), (50,), (10,10), (10,20), (10,30), (10,50), (20,20), (20,30), (20,50), (30,30), (30,50), (50,50),$
  $(10,10,10), (10,10,20), (10,10,30), (10,10,50), (10,20,20), (10,20,30), (10,20,50), (10,30,30), (10,30,50), (10,50,50),$
  $(20,20,20), (20,20,30), (20,20,50), (20,30,30), (20,30,50), (20,50,50), (30,30,30), (30,30,50), (30,50,50), (50,50,50),$
  $(10,10,10,10), (10,10,10,20), (10,10,10,30), (10,10,10,50), (10,10,20,20), (10,10,20,30), (10,10,20,50),$
  $(10,10,30,30), (10,10,30,50), (10,10,50,50), (10,20,20,20), (10,20,20,30), (10,20,20,50), (10,20,30,30),$
  $(10,20,30,50), (10,20,50,50), (10,30,30,30), (10,30,30,50), (10,30,50,50), (10,50,50,50), (20,20,20,20),$
  $(20,20,20,30), (20,20,20,50), (20,20,30,30), (20,20,30,50), (20,20,50,50), (20,30,30,30), (20,30,30,50),$
  $(20,30,50,50), (20,50,50,50), (30,30,30,30), (30,30,30,50), (30,30,50,50), (30,50,50,50), (50,50,50,50)]$

For the diamonds dataset, the best parameters observed were as follows:

- Activation : tanh
- alpha : 0.05
- Hidden Layer Sizes : (50, 50)
- Test RMSE : -0.1261
- Train RMSE : -0.1420

For the gas emissions dataset, the best parameters observed were as follows:

- Activation : ReLU
- alpha : 10.0
- Hidden Layer Sizes : (30, 30, 30)
- Test RMSE : -6.678714994846084
- Train RMSE : -5.31714805654669

17 Question 17

Since the task given to us is a regression analysis problem, the neural network is solving a regression problem where the outputs are continuous between the range of $(-\infty, +\infty)$.
Thus, the activation function should either be None or linear. Hence, our choice of ReLU, tanh and logistic are justified as:

- ReLU is an activation which is bounded to only values that are $>= 0$.
- tanh is an activation which is bounded between -1 and +1.
- Logistic or sigmoid activation ranges from 0 to 1.

18 Question 18

There are several reasons as to why we cannot (or should not) increase the depth of the neural network in an unconstrained manner:

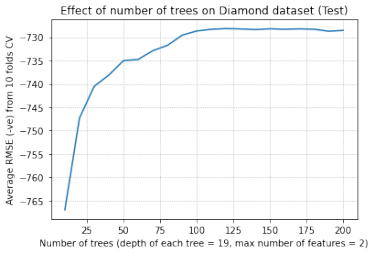
- Vanishing gradients: If a network is too deep, then the gradients will become exponentially small as the information back-propagates from the final layers towards the initial layers. This will lead to slow or premature convergence to a sub-optimal point during the training phase, with the weights in the initial layers being extremely small compared to the weights in the final layers.
- Overfitting: A deeper neural network has more trainable parameters than a shallow one, leading to a complex model with enough capacity to memorize the entire training set, consequently overfitting on the training data and generalizing poorly on the test set.
- Interpretability: Large networks with millions of parameters can lead to complex relationships between features and target variables beyond human understanding, leading to black-box models.
- Compute constraints: A complex network with many layers takes more time, memory and compute power than a simpler model for training, neural architecture search and real-time deployment.
- Data requirements: If the network has millions of trainable weights, the neural network requires a large training set to avoid overfitting and provide robust and usable outputs during deployment

19 Question 19

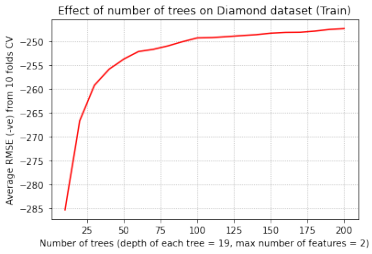
- We tested the performance in terms of train and test RMSE of polynomial regression of various degrees on top 6 and 10 selected features in the diamonds and gas emissions datasets.
- Since F1-Score performed better for feature selection than MI, we selected top 6 and 10 features from F1. We then used ridge regression to incorporate L2 regularization term to prevent overfitting. We optimized the penalty parameter by using grid search with 10-fold cross validation.
- To find the optimal hyperparameters, we first create a hyperparameter space to construct a 10-fold cross validation grid search piperline. The hyperparameter space was as follows:
  - max features = 1-5
  - number of estimators/trees = 10-200
  - depth of each tree = 1-20

The following optimal parameters were obtained for both the datasets:

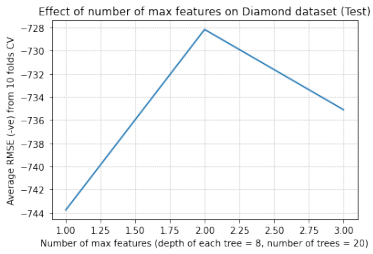
RANDOM FORESTS					
Dataset	Max Depth	Max Features	Max Estimators	Train RMSE	Test RMSE
Diamonds	19	2	120	-0.1195	-0.1948
Gas Emissions	18	3	80	-1.7583	-7.1004



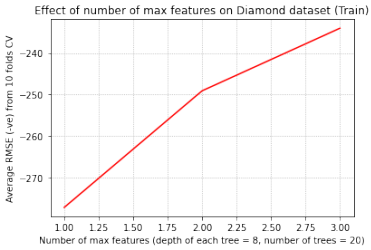
(a) a



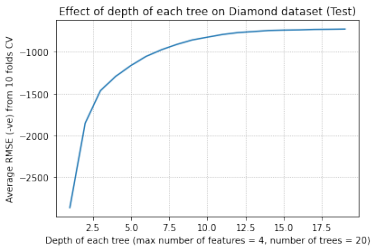
(b) b



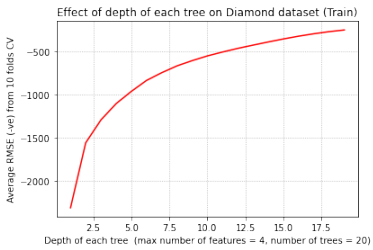
(c) c



(d) d



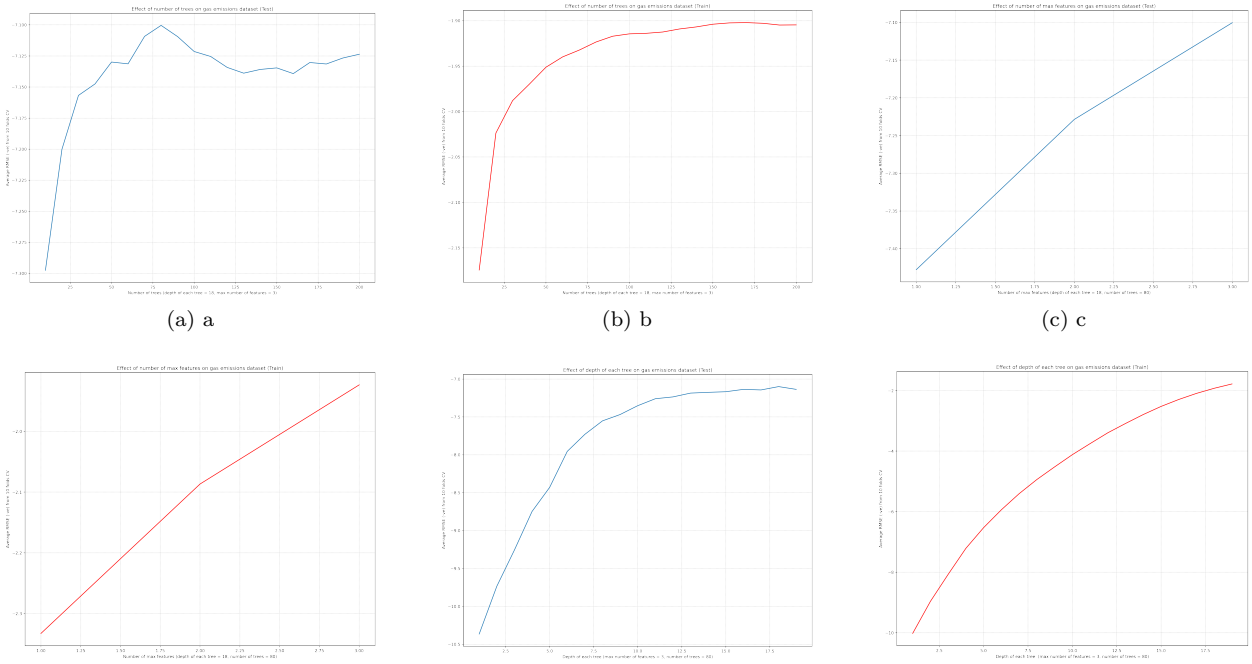
(e) e



(f) f

Figure 13: Random forests for Diamonds Dataset

From Figures above, overall, we see that the overall model performance is non-monotonous with respect to the number of trees, i.e. increasing the number of trees seem to improve or stabilize the performance but the performance improvement is non-monotonic. With all other hyperparameters fixed, the only effect the number of trees have on the model's loss is to decrease it stochastically. One should select a sufficiently large value for the number of trees (e.g. 64 to 128) within compute constraints, as more number of trees does not cause overfitting.



(a) a (b) b (c) c

(d) d (e) e (f) f

Figure 14: Random forests for Gas emission dataset

- In a random forest, each tree along with its output target variable are both independent and identically distributed random variables (weak law of large numbers - WLLN) as the trees are grown using a randomization technique on their individual bootstrap subsamples uncorrelated with growth of other trees. Thus, according to WLLN, the target variable and tree-decision has finite variance, leading to the overall decision (and any other statistic) of the random forest to converge towards a mean value with diminishing returns when the number of trees approach infinity (Jensen's inequality).
- The expected error rate for a random forest ensemble is a non-monotonous function of the number of trees. In fact, error metrics such as RMSE are noisy once a sufficiently large number of required estimators have been used. The convergence rate of the error rate curve does not depend on the number of trees and is only dependent on the distribution of the expected value of the decisions of the trees.
- While increasing the number of trees does not cause overfitting according to WLLN, other hyper-parameters might result in unnecessary variance in the model and cause over-correlation within the ensemble, breaking the notion of independent trees within the random forest.

The following figures show the effects of the depth of each tree on the overall model performance for diamond and CO emission datasets respectively.

From the above figures, for both datasets, increasing the depth of each tree dramatically improves the error rate for both the training and test sets, converging to a constant value. For the test set, the error rate might start to degrade if the depth of each tree exceeds a certain threshold. This indicates that the depth of each tree acts as a regularizer, moderately increasing the depth of each tree improves both fitting and generalizability of the random forest, while very large values of depth of each tree will lead to overfitting by treating each data point including noise in the dataset as a leaf in the tree. A tree with a larger depth will have more fine-grained splits than a tree with shallower depth, having more model capacity and ability to fit

complicated mathematical relationships, but also prone to overfitting. In other words, increasing tree depth decreases bias but increases variance.
The figures below show the effects of the maximum number of features on the overall model performance for diamond and co emission datasets respectively.

From the above figures we can observe that increasing the maximum number of features improves the training RMSE monotonically, converging to a mean value, while the test RMSE starts to degrade after showing improvement until a critical point. This indicates that the number of features also acts as a regularizer like depth of tree, with very large values leading to overfitting on the training set and poor generalization on the test set. This is because increasing the number of features for each tree improves the model capacity of individual trees, but also increases the correlation between each tree, breaking the notion of independent trees within the random forest. Selecting a fraction of the features aims to decorrelate the individual trees and improve generalization error rates, increasing the strength of the random forest.

20 Question 20

In a random forest, each tree is grown using a randomization technique. Hence, the tree along with its output target variable are both independent and identically distributed random variables. Thus, the target variable and tree-decision has finite variance which leads to the overall decision of the random forest to converge towards a mean value. In other words, a random forest contains a large number of uncorrelated models, with the final target variable being a majority decision fusion from all of the trees. This outperforms other individual models because of the following reasons:

- It reduces overfitting of models, assuming that most of the trees are making correct decisions.
- Each decision tree is trained on a different and independent random subspace, forcing there to be a variation among trees. This ensures that all the features within the superspace are properly utilized by the random forest to form a decision.
- Random forest is a method of ensembling. By ensembling, we are essentially getting the "mean of means", which provides us a robust model.
- It does not require scaling or standardization of features.

21 Question 21

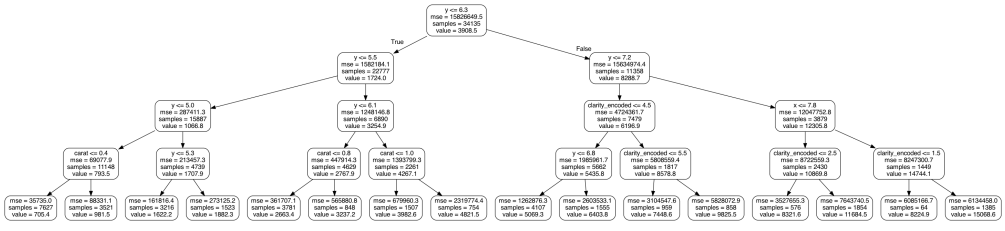


Figure 15: Random Tree in the Random Forest Model for the Diamonds Dataset

For diamond dataset, from branching at the root node, 'price' is selected. This means that feature is the most important feature used to start the splitting process. In a decision tree, the features closer to the root node are more salient and significant than the features near the leaf nodes. The top features (in order) for this sample tree are:

- Level 1: 'y'
- Level 2: 'x'
- Level 3: 'z', 'clarity encoded'

'carat', x, y, z were the most important features found in the previous questions. We can say that the most important features for both datasets are similar to the ones whose p-values were significant for linear regression in the previous sections. Thus, it is safe to say that the most important features found from p-values of linear regression have significant overlap with the most important features found in a random decision tree within the random forest for the diamonds dataset.

22 Question 22

For lightGBM, we select the following hyperparameters along with associated proper search space:

- Boosting type ('boosting_type'): gradient boosting decision tree (GBDT), Dropouts meet Multiple Additive Regression Trees (DART), Random Forest (RF). 'boosting_type' selects one of four gradient boosting algorithms (we omitted Gradient-based One-Side Sampling (GOSS) as it was throwing out errors during training):
  - GBDT (XGBoost): Default and most widely known boosted decision tree algorithm due to its accuracy, reliability and efficiency without requiring considerable effort for hyperparameter tuning. GBDT treats individual decision trees within an ensemble as weak learners, with the first tree aiming to fit the feature set to target variables and the succeeding trees aiming to reduce the residual error between the predicted target variable and ground truth target variable of preceding trees, with the entire ensemble trained via backpropagation of error gradients. The drawback of GBDT is the associated memory and compute requirements for finding the optimal split points for each individual tree, thereby not scaling well when the number of trees is very large. GBDT also suffers from over-specialization, with the latter trees making meaningful predictions for only a few samples in the dataset (sensitive to a few instances) and not generalizing well for the majority of the other samples.
  - DART: DART solves the overspecialization problem in GBDT by introducing dropout, which is used in neural networks to drop weights for regularization and improving generalization for unseen test data.
  - RF: Random forest, which was introduced in previous questions. In a random forest, each tree along with its output target variable are both independent and identically distributed random variables (weak law of large numbers - WLLN) as the trees are grown using a randomization technique on their individual bootstrap subsamples uncorrelated with growth of other trees. Thus, according to WLLN, the target variable and tree-decision has finite variance, leading to the overall decision (and any other statistic) of the random forest to converge towards a mean value with diminishing returns when the number of trees approach infinity (Jensen's inequality). In other words, a random forest contains a large number of uncorrelated models, with the final target variable being a majority decision fusion from all of the trees.
- Depth of each tree ('max_depth'): 1 to 90 in steps of 10. Increasing the depth of each tree improves the error rate for both the training and test sets, converging to a constant value. For the test set, the error rate might start to degrade if the depth of each tree exceeds a certain threshold. This particular hyperparameter acts as a regularizer and performance contributor, moderately increasing the depth of each tree improves both fitting and generalizability, while very large values of depth of each tree will lead to overfitting by treating each data point including noise in the dataset as a leaf in the tree. A tree with a larger depth will have more fine-grained splits than a tree with shallower depth, having more model capacity and ability to fit complicated mathematical relationships, but also prone to overfitting and large training time. In other words, increasing tree depth decreases bias but increases variance.

- Number of leaves ('num_leaves'): 20 to 990 in steps of 10. Similar to the depth of each tree, the number of leaf nodes acts as a regularizer and performance contributor, moderate values improve both fitting and generalizability, while very large values lead to overfitting by treating each data point including noise in the dataset as a leaf in the tree. LightGBM adds leaf nodes to trees based on performance gains from adding those leaves, regardless of depth of each tree. As a result, it is necessary to tune both the maximum number of leaves as well as the depth of each tree to control model complexity.

- Number of trees ('n_estimators'): 10 to 3900 in steps of 100. Specifies the number of boosted trees to fit. Increasing the number of trees improves and stabilizes model performance non-monotonically. The expected error rate for a boosted tree ensemble is a non-monotonous function of the number of trees, being noisy once a sufficiently large number of required estimators have been used. With all other hyperparameters fixed, the only effect the number of trees have on the model's loss is to decrease it stochastically. One should select a sufficiently large value for the number of trees within compute constraints, as more number of trees does not cause overfitting similar to random forests. The overall decision of the ensemble converges towards a mean value with diminishing returns when the number of trees approach infinity (Jensen's inequality). Too few trees, however, will hurt model performance.

- L1 regularization (alpha) penalty parameter ('reg_alpha'): $[10^{-4}, 10^{4}]$. Lasso regularization penalty term on the weights. Small values indicate no regularization (which may cause overfitting when other hyperparameters are large), large values improve generalization but may hurt model performance.

- L2 regularization (lambda) penalty parameter ('reg_lambda'): $[10^{-4}, 10^{4}]$. Tikhonov regularization penalty term on the weights. Small values indicate no regularization (which may cause overfitting when other hyperparameters are large), large values improve generalization but may hurt model performance.

- Subsampling ratio or bagging fraction ('subsample'): [0.1,1]. Specifies the fraction of rows (samples) to be randomly sampled for fitting each tree. The process of training over multiple random samples without replacement is called "bagging", and is similar to how individual trees in a random forest are fitted. Decreasing subsampling ratio helps with overfitting and generalization issues and acts as a regularizer; very small values of bagging fraction may hurt model performance while very large values (no bagging) improves model performance on the training set but generalizes poorly on the test set. Bagging aims at reducing the variance of the ensemble. A similar hyperparameter for selecting features is feature fraction ('colsample_by tree') or random subspace method, which we do not use as we already selected the 10 most salient features for the ensemble to work on. We do however, test this on CatBoost for sake of completeness.

- Subsampling frequency or bagging frequency ('subsample_freq'): 0 to 45 in steps of 5. Controls how often bagging is performed, in terms of epochs.

- Minimal gain to perform split ('min _split _gain'): . Minimum reduction in training loss that results from adding further split points required to partition a leaf node of the tree. LightGBM selects the split point that has the highest gain when adding a new tree node. Increasing this ratio acts as a regularizer, causing the ensemble to ignore very small training performance improvements that do not have meaningful impacts on generalizability of the model.

For CatBoost, we select the following hyperparameters along with associated proper search space:

- Feature fraction ('colsample_by level' or 'rsm'): $[0.1, 1]$. This is a random subspace method that selects the fraction of features to use at each split. Decreasing feature fraction helps with overfitting and generalization issues by selecting only the most salient and important features that have an impact on most of the samples, acting as a regularizer; very small values of feature fraction may hurt model performance while very large values (no random subspace) improves model performance on the training set but generalizes poorly on the test set. Random subspace aims at reducing the variance of the ensemble.

- Number of trees ('num _trees' or 'num _boost _round' or 'n _estimators'): 10 to 1000 in steps of 100. Specifies the number of boosted trees to fit. Increasing the number of trees improves and stabilizes model performance non-monotonically. The expected error rate for a boosted tree ensemble is a non-monotonous function of the number of trees, being noisy once a sufficiently large number of required estimators have been used. With all other hyperparameters fixed, the only effect the number of trees have on the model's loss is to decrease it stochastically. One should select a sufficiently large value for the number of trees within compute constraints, as more number of trees does not cause overfitting similar to random forests. The overall decision of the ensemble converges towards a mean value with diminishing returns when the number of trees approach infinity (Jensen's inequality). Too few trees, however, will hurt model performance.
- Number of leaves ('num _leaves' or 'max _leaves'): 20 to 990 in steps of 10. This is used only for lossguide growing policy. Similar to the depth of each tree, moderate values improve both fitting and generalizability, while very large values lead to overfitting by treating each data point including noise in the dataset as a leaf in the tree.
- Depth of each tree ('max _depth' or 'depth'): 1 to 16 (16 is the maximum CatBoost supports on CPU) in steps of 2. Increasing the depth of each tree improves the error rate for both the training and test sets, converging to a constant value. For the test set, the error rate might start to degrade if the depth of each tree exceeds a certain threshold. This particular hyperparameter acts as a regularizer and performance contributor, moderately increasing the depth of each tree improves both fitting and generalizability, while very large values of depth of each tree will lead to overfitting by treating each data point including noise in the dataset as a leaf in the tree. A tree with a larger depth will have more fine-grained splits than a tree with shallower depth, having more model capacity and ability to fit complicated mathematical relationships, but also prone to overfitting and large training time. In other words, increasing tree depth decreases bias but increases variance.
- L2 regularization penalty parameter ('l2 _leaf _reg' or 'reg _lambda'): $[10^{-4}, 10^{4}]$. Tikhonov regularization penalty term on the weights. Small values indicate no regularization (which may cause overfitting when other hyperparameters are large), large values improve generalization but may hurt model performance
- Bagging temperature ('bagging _temperature'): $[0.1, 10]$. CatBoost uses Bayesian bootstrap aggregation for regression problems. If bagging temperature is 0, the weights assigned to the sampled subspaces are equal to 1. If bagging temperature increases, the intensity of bootstrap increases and the weights are sampled from exponential distribution. CatBoost uses minimum variance sampling or weighted sampling at the tree level and not the split level.
- Score function ('score _function'): Cosine, L2. This is used only for symmetric or depthwise growing policies. Score function is used to choose between candidate trees when adding a new tree to the ensemble. L2 and Cosine are first-order score functions.
- Grow policy ('grow _policy'): Symmetric Tree, Depthwise Tree and LossGuide. Specifies how the trees will be generated from the leaves. Unlike LightGBM which uses leaf-wise tree growth (best-first) to allow for an imbalance tree (may cause overfitting for small datasets) regardless of the depth of each tree, Catboost grows a balanced tree such that at each level, the feature-split pair that minimizes the loss function is selected and utilized for all level nodes.
  - Symmetric Tree (ST): Level by level growth strategy; on each iteration, all leaves from the last tree level are split with the same condition.
  - Depthwise Tree (DT): Level by level growth strategy; on each iteration, all non-terminal leaves from the last tree level are split depending on the feature-split pair that minimizes the loss function.
  - LossGuide (similar to LightGBM) (LG): Leaf by leaf growth strategy; on each iteration, non-terminal leaf with the best loss improvement is split.

23 Question 23

For this question, we find the optimal hyperparameters in terms of train and test RMSE (average negative test RMSE, higher is better) of boosted trees (LightGBM and CatBoost) using Bayesian optimization on the chosen dataset (diamond dataset). In Bayesian optimization, a Gaussian process is used to approximate the objective function (called surrogate model), which allows priors on the distribution of moments (mean and uncertainty) to propagate forward as the search progresses. The acquisition function decides the next set of hyperparameters to sample from the design space using Monte Carlo sampling with Bayesian Upper-Confidence Bounds (UCB), also known Thompson sampling, which balances exploration and exploitation. Thompson sampling ensures that the acquisition function does not get stuck in a local optimum early in the search, with the exploration parameter decreased as the confidence in the Pareto-frontier grows. We used 10 iterations for the CatBoost regressor, and 20 iterations for the LightGBM regressor, coupled with 10-fold cross-validation. Table below outlines the best set of hyperparameters obtained for both the regressors. From the table, we see that lightGBM performs slightly better than CatBoost in terms of average train and test RMSE on the diamond dataset. However, there is no significant performance difference among the two classifiers. LightGBM is more robust to overfitting than catboost, possibly due to the presence of more regularization hyperparameters than CatBoost.

Regressor	Light GBM	CatBoost
Boosting type	GBDT	-
Depth of each tree	21	9
number of trees	710	210
number of leaves	40	N/A
Min. split gain	0.0001	-
Bagging Fraction	0.5	-
Bagging Frequency	35	-
L1 reg	0.0001	-
L2 Reg	0.0001	0.01
Fracture Fraction	default	0.8
Grow Policy,Score func	-	ST, Cosine
Bagging Temp	-	2.1
Train RMSE(-ve)	-0.05959	-0.08968
Test RMSE (-ve)	-0.2896	-0.2701

24 Question 24

Figure below shows the effects of the 9 hyperparameters of LightGBM on the average negative train and test RMSE for diamond dataset obtained from Bayesian hyperparameter tuning. Since Bayesian optimization is stochastic, we also plot trend lines to showcase the overall tendency of each hyperparameter.

a. Boosting Type: From Figure, we see that GBDT and RF perform similar to each other, while both jointly outperform DART. Theoretically, DART should improve generalizability of GBDT by introducing dropout. However, DART also has implicit hyperparameters that require tuning, such as model seed, the probability of skipping dropout during an iteration, dropout rate and whether to use XGBoost dropout or uniform dropout, which we did not tune due to compute constraints. Too many floating parameters can result in unexpected performance of DART.

b. Depth of each tree: From Figure, we see that increasing the depth of each tree improves the error rate for both the training and test sets slightly. This particular hyperparameter acts both as a regularizer

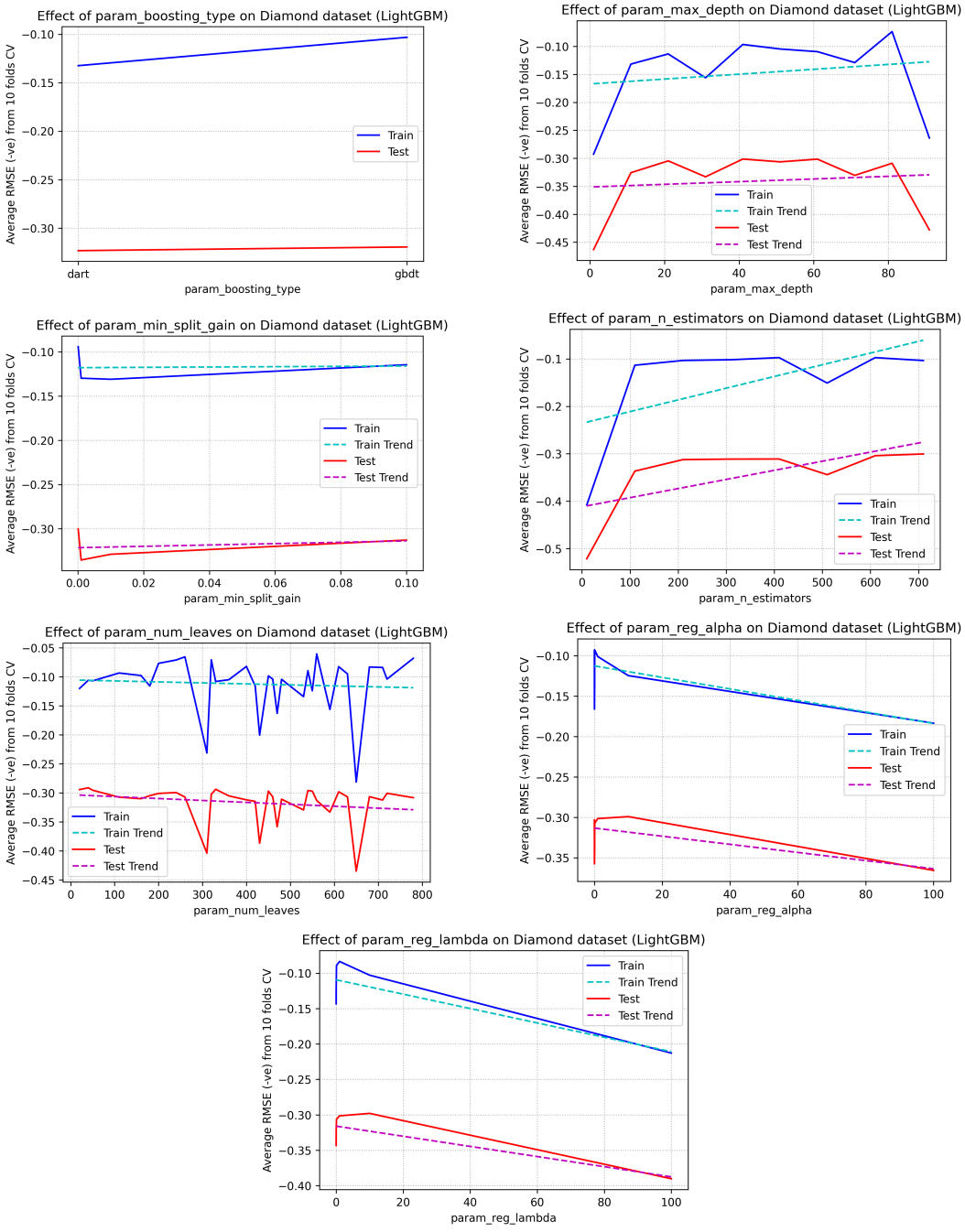


Figure 16: Effect of the hyperparameters of LightGBM individually on diamond dataset

and a performance contributor, moderately increasing the depth of each tree improves both fitting and generalizability, while very large values of depth of each tree will lead to overfitting by treating each data point including noise in the dataset as a leaf in the tree. A tree with a larger depth will have more fine-grained splits than a tree with shallower depth, having more model capacity and ability to fit complicated mathematical relationships, but also prone to overfitting and large training time. In other words, increasing tree depth decreases bias but increases variance.

c. Minimal gain to perform split: From Figure, we see that the both training and test RMSE improves slightly with an increase in the gain ratio. Increasing this ratio acts as a regularizer, causing the ensemble to ignore very small training performance improvements that do not have meaningful impacts on generalizability of the model. However, the graph does not show the full picture as the Bayesian agent did not cover the full range of the ratio. More aggressive bagging is likely to cause drop on model performance due to regularization effects.

d. Number of trees: From Figure, we do not see any performance gains or losses as the number of trees increases. Increasing the number of trees may improve and stabilize model performance non-monotonically. The expected error rate for a boosted tree ensemble is a non-monotonous function of the number of trees, being noisy once a sufficiently large number of required estimators have been used. With all other hyperparameters fixed, the only effect the number of trees have on the model's loss is to decrease it stochastically. One should select a sufficiently large value for the number of trees within compute constraints, as more number of trees does not cause overfitting similar to random forests. The overall decision of the ensemble converges towards a mean value with diminishing returns when the number of trees approach infinity (Jensen's inequality). Too few trees, however, will hurt model performance (weak law of large numbers). Thus, unless the number of trees is very small, increasing the number of trees has no effect on the model performance.

e. Number of leaves ('num_leaves' or 'max _leaves'): Similar to the depth of each tree, the number of leaf nodes acts as a regularizer and performance contributor, moderate values improve both fitting and generalizability, while very large values lead to overfitting by treating each data point including noise in the dataset as a leaf in the tree. Figure shows slight improvement in performance as the number of leaves increases.

f. L1 regularization term: From Figure, we observe that both train and test RMSE initially improves with finite values of Lasso regularization, but performance drops when aggressive regularization is performed. Small values indicate no regularization (which may cause overfitting when other hyperparameters are large), large values improve generalization but may hurt model performance.

g. L2 regularization term: From Figure, we observe that both train and test RMSE initially improves with finite values of Tikhonov regularization, but performance drops when aggressive regularization is performed. Small values indicate no regularization (which may cause overfitting when other hyperparameters are large), large values improve generalization but may hurt model performance. Note that L1 regularization is a stronger regularization and causes more aggressive regularization than L2, as L1 regularization zeros out weights more often than L2 regularization. From more details on L1 and L2 regularization, we urge the grader to look at Question 11.

h. Bagging frequency: From Figure, we see that as bagging is conducted more often, the performance drops. This is expected because bagging acts as a regularizer. More frequent bagging (dropping samples) drops a subset of training data more frequently, which can cause important training samples to never be considered during training.

i. Bagging fraction: From Figure, we see that as subsampling ratio increases, the performance also increases monotonically. Decreasing subsampling ratio helps with overfitting and generalization issues and acts as a regularizer; very small values of bagging fraction may hurt model performance while very large values (no bagging) improves model performance on the training set but generalizes poorly on the test set. Bagging aims at reducing the variance of the ensemble

The following figure shows the effects of the 8 hyperparameters of CatBoost on the average negative train and test RMSE for diamond dataset obtained from Bayesian hyperparameter tuning. Since Bayesian optimization is stochastic, we also plot trend lines to showcase the overall tendency of each hyperparameter.

a. Bagging temperature: From the figure(1), we see that as bagging temperature increases, the tran and test RMSE degrades. If bagging temperature is 0, the weights assigned to the sampled subspaces are equal to 1. If bagging temperature increases, the intensity of bootstrap increases and the weights are sampled from exponential distribution. Since, we are already supplying top 10 salient features to CatBoost, assigning extreme weights to these features can hurt model performance.

b. Feature fraction: From Figure, we see that as feature fraction increases, the performance improves as well. Decreasing feature fraction helps with overfitting and generalization issues by selecting only the most salient and important features that have an impact on most of the samples, acting as a regularizer; very small values of feature fraction may hurt model performance while very large values (no random subspace) improves model performance on the training set but generalizes poorly on the test set. Random subspace aims at reducing the variance of the ensemble.

c. Grow policy: From Figure, we observe that symmetric growth policy performs better than both depthwise and lossguide growth policies. Lossguide uses leaf-wise tree growth (best-first) to allow for an imbalance tree, which usually causes overfitting when the feature space is small. In addition, symmetric tree considers all leaves from the last iteration instead of just the non-terminal leaves for making splits, thus having a more generalized overview of the earlier feature subspaces than depthwise tree.

d. L2 regularization: From Figure, we observe that overall, more aggressive regularization leads to performance drops. Small values indicate no regularization (which may cause overfitting when other hyperparameters are large), large values improve generalization but may hurt model performance.

e. Depth of each tree: From Figure, we see that increasing the depth of each tree improves the error rate for both the training and test sets slightly. This particular hyperparameter acts both as a regularizer and a performance contributor, moderately increasing the depth of each tree improves both fitting and generalizability, while very large values of depth of each tree will lead to overfitting by treating each data point including noise in the dataset as a leaf in the tree. A tree with a larger depth will have more fine-grained splits than a tree with shallower depth, having more model capacity and ability to fit complicated mathematical relationships, but also prone to overfitting and large training time. In other words, increasing tree depth decreases bias but increases variance.

f. Number of leaves (only for lossguide): Similar to the depth of each tree, the number of leaf nodes acts as a regularizer and performance contributor, moderate values improve both fitting and generalizability, while very large values lead to overfitting by treating each data point including noise in the dataset as a leaf in the tree. Figure, however, shows no significant performance changes with changes in number of leaves, which is probably because other hyperparameters rendered the leaf hyperparameter change contribute insignificantly towards the final performance.

g. Number of trees: From Figure , we see slight performance gains as the number of trees increases. Increasing the number of trees may improve and stabilize model performance non-monotonically. The expected error rate for a boosted tree ensemble is a non-monotonous function of the number of trees, being noisy once a sufficiently large number of required estimators have been used. With all other hyperparameters fixed, the only effect the number of trees have on the model's loss is to decrease it stochastically. One should select a sufficiently large value for the number of trees within compute constraints, as more number of trees does not cause overfitting similar to random forests. The overall decision of the ensemble converges towards a mean value with diminishing returns when the number of trees approach infinity (Jensen's inequality). Too few trees, however, will hurt model performance (weak law of large numbers). Thus, unless the number of trees is very small, increasing the number of trees has no effect on the model performance.

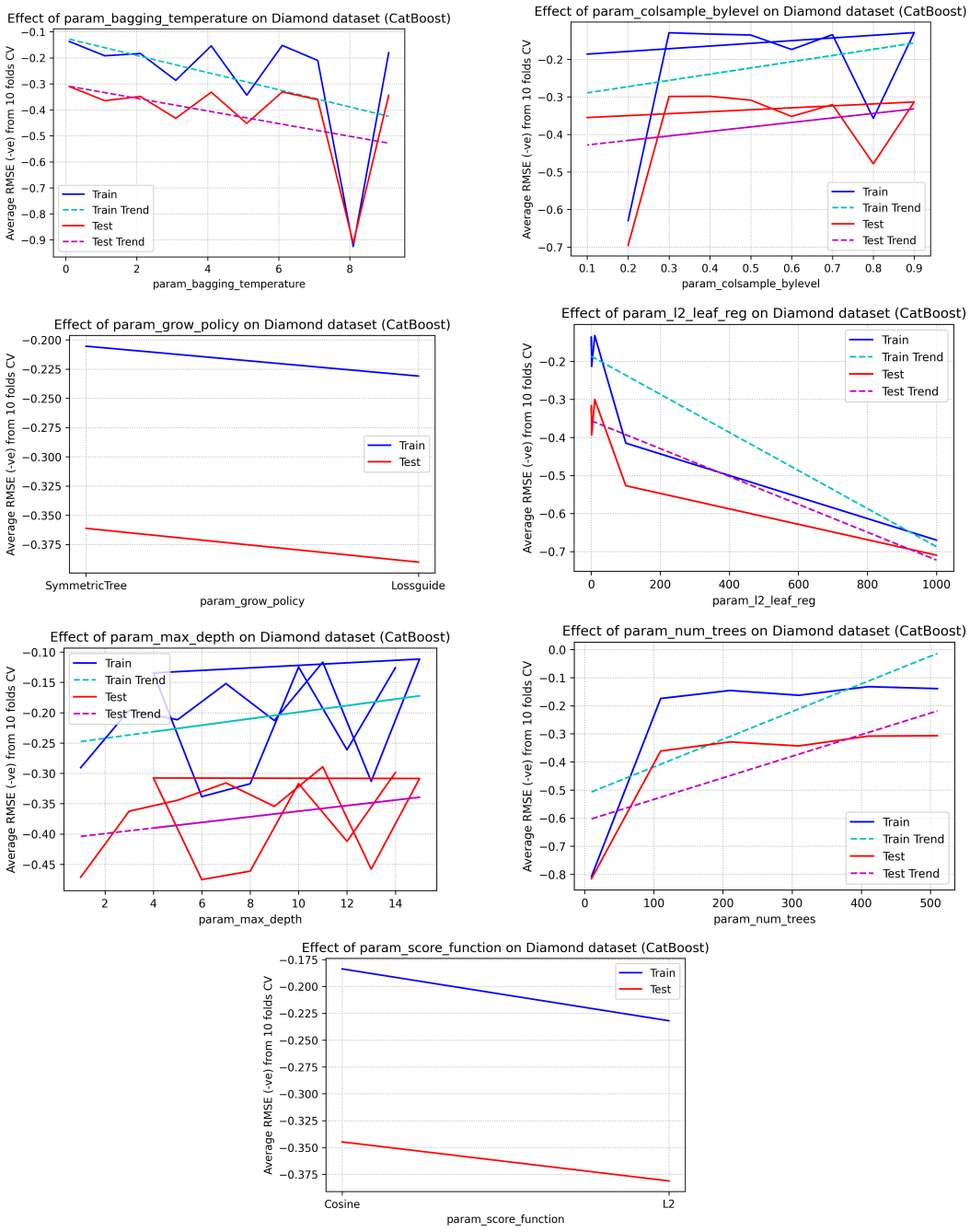


Figure 17: Effect of the hyperparameters of Catboost individually on diamond dataset

h. Score function: From Figure, we see that cosine distance performs better as a score function than L2 distance. This is expected because cosine similarity is not affected by the magnitude of the features, meaning that the range and scale of the features does not affect the distance metric, but rather it associates features based on angle between sample points. This corrects the effects of non-standardized or unscaled features. In addition, in high dimensions, the rapid increase in volume causes the data to become sparse in each dimension, causing the Euclidean distances to converge to a constant value between all sample points. Since all feature points become equidistant, models working with L2 distances cannot find distinguishable features within the data. Thus, for textual clustering, cosine similarity is the ideal metric over L2 distances

25 Question 25

Model	Best average train RMSE		Best average test RMSE	
	Diamond	Gas Emission	Diamond	Gas Emission
Linear	-0.302	8.3247	-0.302	-8.3661
Lasso	-0.302	-7.8917	-0.303	-8.3660
Ridge	-0.3		-7.8917	-8.3661
Polynomial	-0.13	-5.8752	-0.15	-0.6164
MLP	-0.1261	-5.3171	-0.1420	-6.6787
Random Forest	-0.1195	-1.7583	-0.1948	-7.1004
LightGBM	-0.05959	-	-0.2896	-
Catboost	-0.08968	-	-0.2701	-

We see that usually, the negative training RMSE is higher (absolute training RMSE is lower) than the test (validation) RMSE. This happens because complex models tend to overfit on the characteristics of the training set, which may not be present in the validation set and lead to models not generalizing well on the validation set.

For all the grid searches, we evaluate both training and validation RMSE as reported in the sections above. These two metrics can be significantly different, since with an increase in the model complexity, the training RMSE will decrease monotonically. However, at last we will approach a point where the problem of overfitting occurs.

In general, as training RMSE decreases monotonically, validation RMSE tends to decrease first and then increase. We perform various fine-tuning procedures in order to find the optimal model structure with optimal penalty parameters to give us the minimal validation RMSE.

26 Question 26

In this question, we report the OOB error for the random forest model in previous question. The OOB error for the best random forest models on the two datasets are as follows:

- OOB, diamond: 0.8980
- OOB, CO emission: 0.7973.

In a random forest, each tree along with its output target variable are both independent and identically distributed random variables (weak law of large numbers - WLLN) as the trees are grown using a randomization technique on their individual bootstrap subsamples uncorrelated with growth of other trees. Each

decision tree is trained on a different and independent random subspace, forcing variation, decorrelation and diversification among individual trees. OOB score is a technique for validating random forest models, defined as the number of correctly predicted samples (rows) from the out of bag sample rows via majority decision fusion. An out-of-bag sample is defined rows in the original dataset that was not present in the bootstrap samples of a particular decision tree. Since OOB is being calculated on unseen test data, it serves as a form of validation score for the ensemble.

R^2 (coefficient of determination), on the other hand, is defined as the ratio of the variance in the target variable that can be predicted by the features in the dataset (also known as goodness of fit). It includes all the data in the training (and validation) set regardless of whether a tree within an ensemble saw some of the samples or not, along with decision from all the decision trees. For R^2 score calculation, separate steps are required for training and validation score calculation.

27 Question 27

Report the following statistics for each hashtag, i.e. each file: Average number of tweets per hour, Average number of followers of users posting the tweets per tweet (to make it simple, we average over the number of tweets; if a users posted twice, we count the user and the user's followers twice as well), Average number of retweets per tweet.

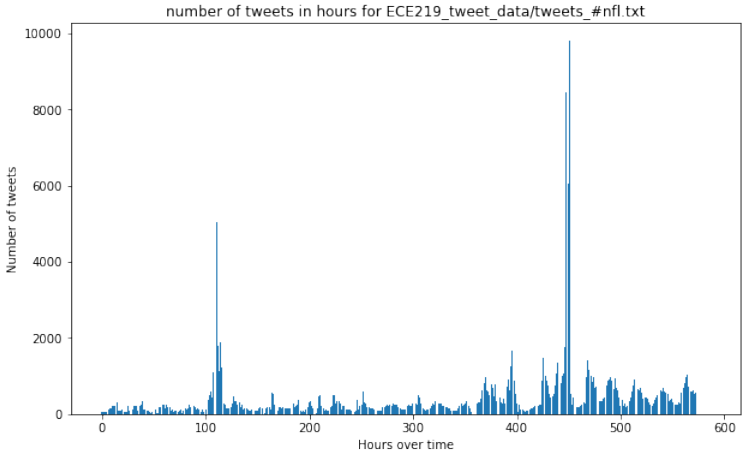
Hashtags	Avg. no. of tweets	Avg. no. of followers	Avg. no. of retweets
#gohawks	292.48785062173687	2217.9237355281984	2.0132093991319877
#gopatriots	40.954698006061946	1427.2526051635405	1.4081919101697078
#nfl	397.0213901819841	4662.37544523693	1.5344602655543254
#patriots	750.8942646068899	3280.4635616550277	1.7852871288476946
#sb49	1276.8570598680474	10374.160292019487	2.52713444111402
#superbowl	2072.11840170408	8814.96799424623	2.3911895819207736

From the above results, it is obvious that #sb49 and superbowl have higher values for all the three statistics among the six hashtags. The reason might be that these two hashtags are more general and all the fans would like to tag them. But for hashtags like #gohawks, #gopatriots and #patriots, they would only be tagged by their supporters which included a smaller group of people.
In addition, since Seattle Seahawks was the defending champion, the #gohawks was more popular than #gopatriots which was reflected in the three statistics. But New England Patriots finally sealed the win, so the hashtag #patriots burst and #hawks did not.

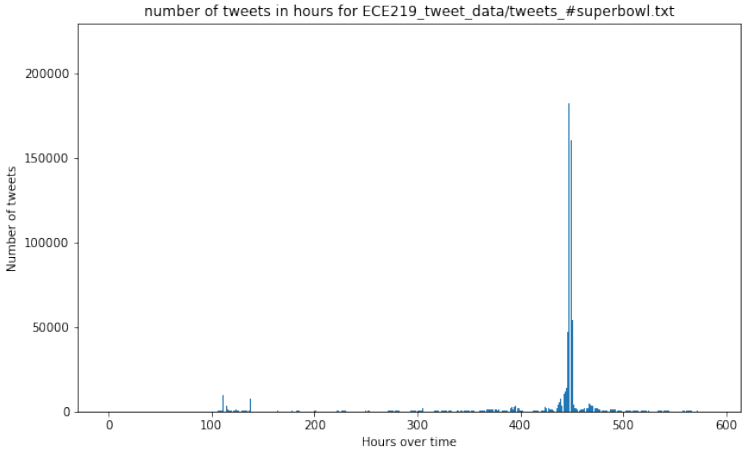
28 Question 28

The histograms above show the lifecycle of two hashtags #superbowl and #NFL. The two have different lifecycle patterns but it is clear that they both peak at around the 450th hours. Since the Twitter data is collected starting from 2 weeks before the game to a week after the game, the peak time is exactly corresponding to the actual time that Superbowl took place.

In addition, there are also some discussions about the #NFL during the other times and a small spike at around the 100th hour, which might corresponding to the NFL conference championships game time. But the discussion for #superbowl concentrated tightly around the actual time it have taken place.



(a) Number of tweets in hour for Superbowl



(b) Number of tweets in hour for NFL

29 Question 29

We have chosen our task - Developing a Library of Prediction Tasks given a tweet:

- Predict how likely it is that a tweet belongs to a specific team fan
- Predict the number of retweets from a tweet
- Predict positive/negative tweets in a specified time frame, with some analysis using Wordcloud.

Predict the tweet belongs to a specific team fan

29..1 Task Description

We are using the data from tweets_#superbowl.txt dataset of 5.2GB for this purpose. We read the file as a json object, and it has multiple keys like firstpost_date, title, url, tweet, hashtags, retweet_count, followers_count, citations.
For the purpose of this prediction task we are interested in finding which fan team does the tweet belong to. We have formulated the task as a binary classification problem for prediction between the teams - Hawks and Patriots (teams that have entered the superbowl).

29..2 Data Pre-processing

Before performing our actual feature selection and prediction task we have performed some pre-processing on the tweets dataset as explained below :

- Tweet is extracted from the tweets dataset using the referencing json_obj['tweet']['text'].
- We then proceed to build the labels for each tweet, as this is not inherently present in the dataset. Labels over here refer to which team's fan is the tweet inclining to.
- We make use of regex and find all the hashtags in the tweet, and then scan them to see if they match the pattern of hawks or patriots.
- One-hot label encoding is employed where [0,1] would mean the tweet is from a Patriots fan, and [1,0] would mean it's from a Hawks fan.
- We also remove the # from the tweet text before feature selection because we want the words associated with the hashtags to also be considered by our feature selection models.
- The tweets which have labels belonging to both patriots and hawks or neither are removed from the dataset. So, the size of the dataset now would be 158153 tweets.
- Train_test_split function is used on this processed dataset, with a test_size of 0.2.

29..3 Feature Selection

- We employ the feature selection methods learnt in Project 1 through 3 for this task.
- Each tweet in the dataset is first cleaned. Cleaning involves removing the HTML artefacts. This cleaned data is then passed through a lemmatization function which uses pos_tagging.
- A TfidfVectorizer(stop_words='english', min_df=3) is then fit to the lemmatized data features. This will assign weights to each word in our dataset and build a model out of it.
- To reduce the number of features we select the top 50 features using SVD decomposition, Truncated SVD() function.
- The above feature selection is projected on both x_train and x_test. So, the final shape of x_train is (126522, 50), and x_test is (31631, 50). This will reduce the processing time, and chances of model overfitting as we are only retaining the important features for the task.

29..4 Team Classification

- The first model used for Team Classification is LogisticRegression().
- A parameter grid search is performed using GridSearchCV() for a 4 fold cross validation. 'L2' and no penalty are the penalty parameters, along with C ranging from 0.01 to 100.
- The best model from this grid search is - LogisticRegression(C=0.01, penalty='none').
- The second model employed for comparison purpose is a RanfomForestClassifier().
- A parameter grid search is performed here as well with max_depth varying from 10 to 200.
- The best model found from grid search is - Random Forest Classifier with max_depth = 50.

Table 3: Evaluation Results for Team Prediction

Model	Accuracy	Recall	Precision	F1-Score
Logistic Regression	0.980	0.987	0.978	0.982
Random Forest	0.980	0.980	0.984	0.982

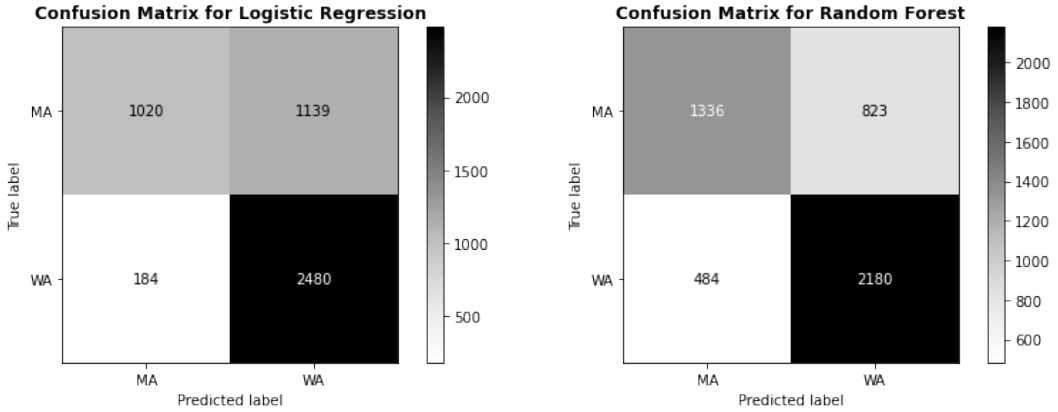


Figure 18: Confusion Matrix

29.a Model Evaluation

The results of the above described models are summarised in the table below.
The confusion matrix for the prediction tasks are -
Our inference from these results is that the model does pretty well for the task of classifying between the fan teams of Patriots and Hawks from the superbowl dataset. This inference is backed by the accuracy metrics and the confusion matrix where a clear binary classification between 2 classes is observed.

29.b Sentiment Analysis over time

Analysis in particular time frames has been done on the #gohawks and #gopatriots dataset. We notice an interesting trend which explains why our classifiers perform so well on these datasets.
In the figure above we plot the average sentiment polarity over time for the seahawks and patriots. If we look at the peaks in both datasets we notice that when Seahawks has a high polarity, Patriots has a very low polarity roughly around the 10th hour. And at the 11th hour when patriots peaks to a high polarity, seahawks is at its lowest polarity. Also, to start off with Seahawks in general has a very high average polarity compared to Patriots. This says that the Seahawks fans are more happy with their team's performance throughout till the 14th hour, when Patriots also catches up and both teams have a similar polarity.
From the figures above we can conclude that the positive and negative tweets are well separable with respect to time in both datasets. And on an average, the number of positive tweets is higher than the number of negative tweets.

29.c Analysis using Word Cloud

We further analyzed this by using a word cloud, (also called tag cloud or weighted list) is a visual representation of text data. Words are usually single words, and the importance of each is shown with font size or color. Python fortunately has a wordcloud library allowing to build them.
Obviously the words taking up more space in the cloud are the ones common to both kinds of tweets - GoHawks, SeaHawk, Superbowl. However if we look closely there are a lot of positive words associated with

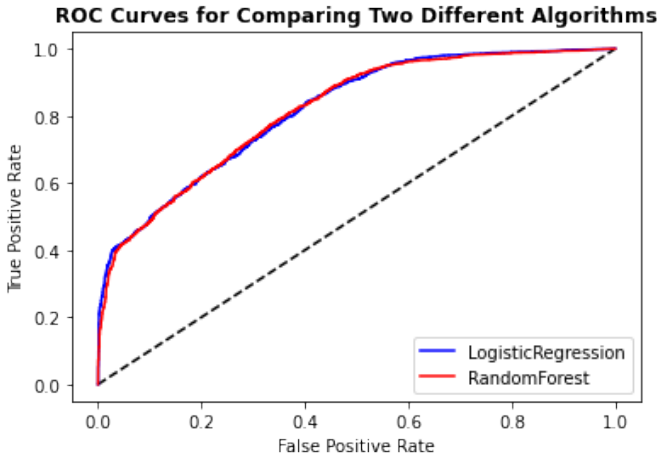


Figure 19:
Comparision of ROC both two different algorithms

higher polarity tweets like - good, great , excited, good luck, happy, love, first. Whereas in the cloud for negative tweets we don't find such words present and it is shadowed with words like hate, little, mean, sorry. This is a great visualisation tool that aids us in understanding our dataset better.

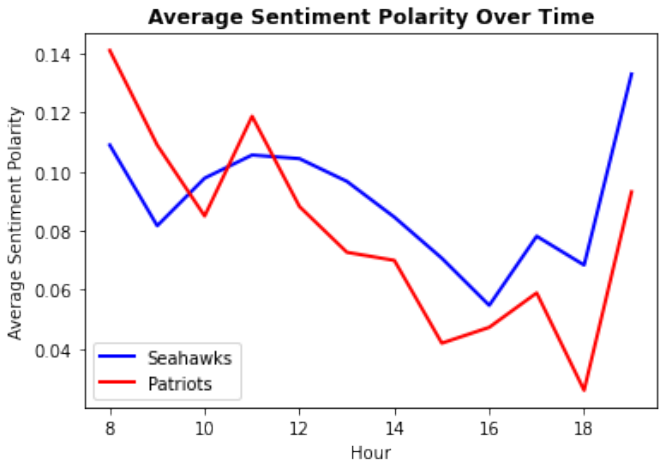


Figure 20:
Average sentiment polarity over time

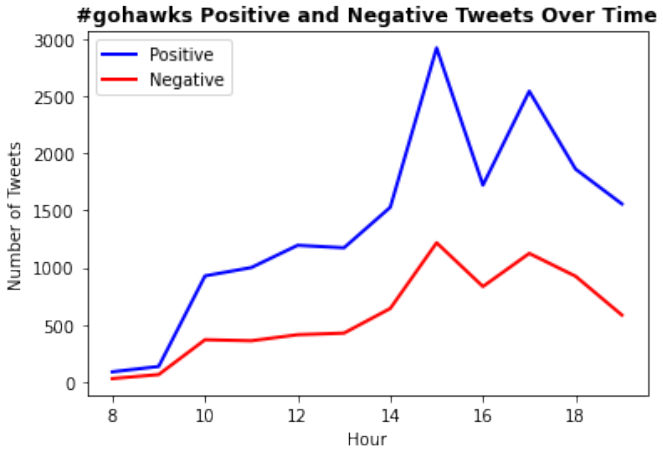


Figure 21:
#gohawks positive and negative tweets over time

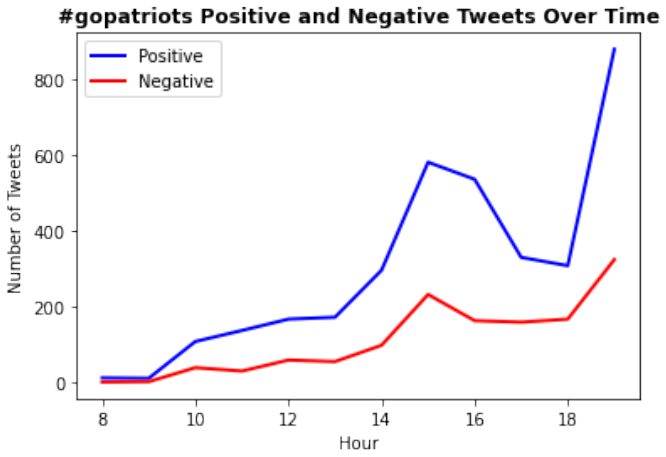


Figure 22:
#gopatriots positive and negative tweets over time

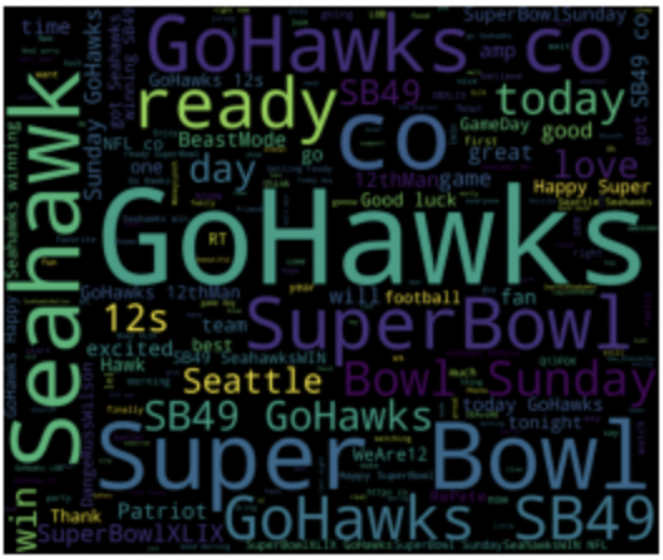


Figure 23: Wordcloud analysis for positive tweets in #gohawks

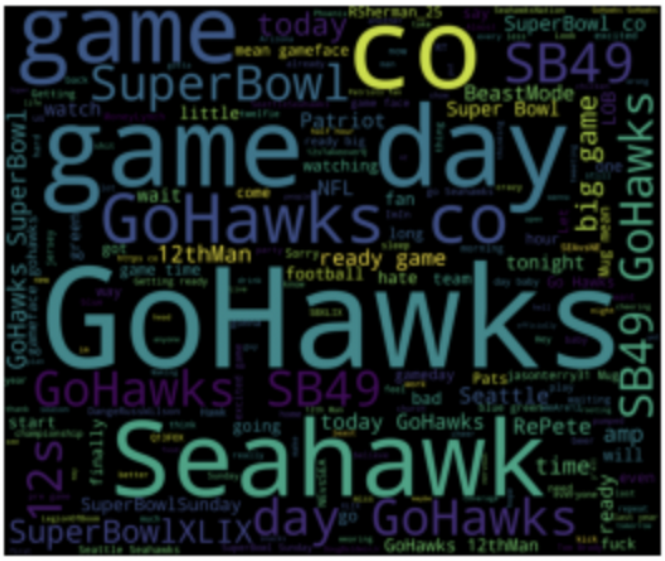


Figure 24: Wordcloud analysis for positive tweets in #gopatriots