Lab 3 Report

Palaskar Adarsh Mahesh
B20EE087

1 Question 1

Pre-processing and Visualization:

Checked for null or missing values in the dataset and no such values were present. Then checked for outliers and the corresponding rows were dropped. Then all the features were label encoded.

Q1.1 Implementing a Decision Tree regressor

Implemented a decision tree regressor on the skeleton of the classifier made in Lab2. The cost function was variance reduction, where the variance of the splits was calculated for each value present in a feature. To optimize the process the variance of the splits was checked only on unique values of the feature. After the max depth of the tree is reached, the predicted value is calculated by averaging out all the values present at a leaf node during training.
The r2 score for our regressor was found to be 0.6172835.

Q1.2 Performing 5-fold cross validation to determine the best max_depth:

We perform 5-fold cross validation on the data for different maximum depths ranging from 2 to 14 and took the mean absolute error for each fold for each max_depth. For each varying max_depth, we averaged out the mean absolute

errors for each fold. The tree having the least average will be considered the best tree having the best max_depth.

Q1.3 Visualizing and summarizing results across validation sets: Plotting the graph for averaged out mean absolute errors v/s max_depth, we observe that the error is minimum when maximum depth is equal to 4. Thus the best max_depth for out regressor will be 4.

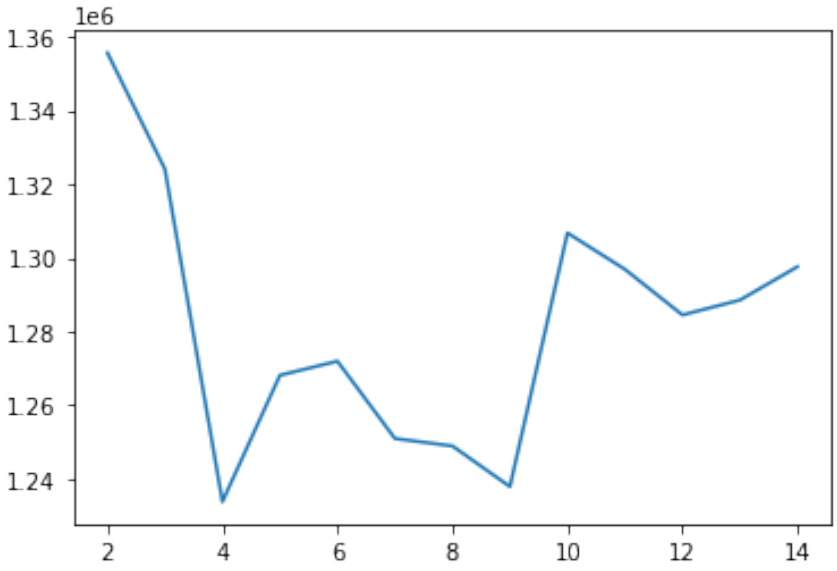

Figure 1: Mean Absolute error v/s max_depth

Q1.4 Implementing Bagging from scratch to create different training data sets:

To create different training data sets, we create a certain number of bags from the training data by using sampling with replacement. These bags will be our base learners. Since we use sampling with replacement, data overlapping is allowed. Also these base learners can over-fit the data and have low bias and high variance.

1. bag(dataframe,ratio):
   Creates different training set from dataframe, which contains ratio amount of data from dataframe.

2. Bagging(dataframe,ratio,no_of_bags, max_depth_of_each_bag):
   For each bag created, the function trains it on our model and returns the predictions for the same.

Q1.4 and Q1.5 :

Bags can be created and trained using the above functions. We observe that some trees have a r2 score of less than 0.5 and hence they are not weak learners and instead will decrease the accuracy of the final model. Such trees should be removed. The performance of each tree can be visualized from the graph below:

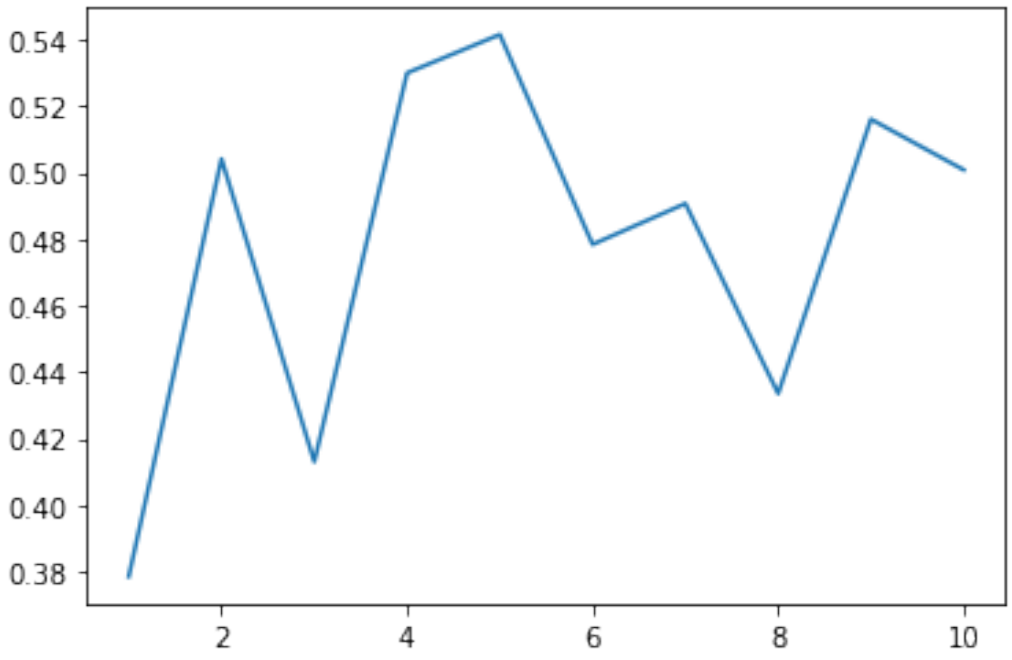

Figure 2: r2score for each tree

The average r2 score is equal to 0.478674.

Q1.7 Combining trees into a single prediction:

We can combine all the trees by taking the average of predictions computed by each tree for a particular test value. The r2_score we get for the complete bagging model is equal to 0.64186079, which is an improvement over the normal decision tree regressor. This score can be improved significantly if we remove the the unnecessary trees(r2_score < 0.50) and use only the weak learners.

Q1.8 Relation with max_depth:

We observe that when the maximum depth is increased in the above case, the r2_score increases and decreases when the maximum depth is decreases. But in general we observe that the model gives a peak r2_score at some depth 'd' and then r2_score decreases when it is increased or decreased, mainly due to over-fitting and under-fitting, respectively.

Q1.9 Random Forest Regressor:

Mean squared error = 727566432372.8971
Mean absolute error = 602583.5519607842

Q1.10 AdaBoost Regressor:

Mean squared error = 891134274740.0239
Mean absolute error = 720692.9895917708

2 Question 2

Pre-processing and Visualization:

Checked for null or missing values in the dataset and no such values were present. All the features were scaled using MinMaxScalar. Then checked for outliers and the corresponding rows were dropped. Then all the features were label encoded.

Q2.1 Decision Tree implementation form scratch:

Using object-oriented programming, we have created two classes:

1. Node: This class stores all the information for a decision node which includes:
   (a) predicted_class : (for leaf node) determines which class the tree predicts the input belongs to
   (b) feature_index : stores the index of feature on the basis of which the split is made(since it has the highest information gain)
   (c) threshold : the value at which the split is made.
   (d) left,right : left and right children of the decision node
   (e) max_info_gain : maximum information gain corresponding to the feature for which we store the feature index.
2. DecisionTreeClassifier : The actual training function which contain all the helper functions:
   It has a constructor, where we can provide values for maximum depth beyond which the tree cannot grow and the minimum information gain needed to make the next split.
   These are:
   (a) _best_split(self,X,y):
   It calculates the best possible split for the decision node by iterating through all the features and each value in each feature

and calculates the information gain corresponding to each value, and the sets the threshold for the value which provides the best information gain. It returns the index corresponding to the best feature and the threshold for the best split and the corresponding information gain.

(b) _grow_tree(self,X,y,depth = 0):
This function makes the split according the values returned by the _best_split() function and also repeats the process recursively on both left and right children for every decision node, unless it is a leaf node.

(c) Classify():
The classify function visits each node recursively for each value and finds out the predicted class for each input. It then returns and array containing the predicted values for each input.

Overall accuracy = 0.8333333333333334
Accuracy for class 0 = 0.5333333333333333
Accuracy for class 1 = 0.9583333333333334

Q2.2 Performing 5-fold cross validation to determine the best max_depth:

We perform 5-fold cross validation on the data for different maximum depths ranging from 2 to 14 and took the accuracy for each fold for each max_depth. For each varying max_depth, we averaged out the accuracy for each fold. The tree having the highest average accuracy will be considered the best tree having the best max_depth.

Q2.3 Visualizing and summarizing results across validation sets:

Plotting the graph for averaged out accuracy v/s max_depth, we observe that the accuracy is maximum when maximum depth is equal to 4. Thus the best

max_depth for out classifier will be 4.

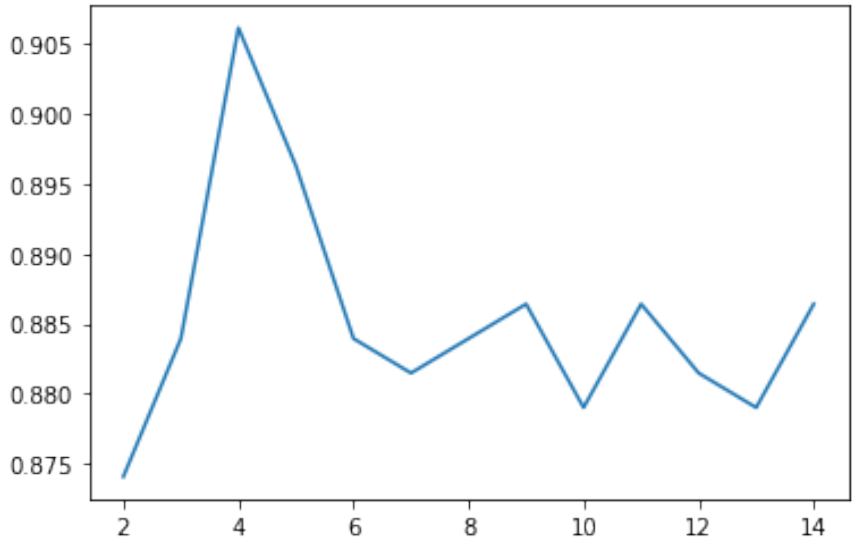

Figure 3: Average accuracy v/s max_depth

Q2.4 and Q2.5 XGBoost implementation:

Using the inbuilt model in sklearn, we set the hyper-parameters(subsample = 0.7, max_depth = 4) as given in the question and train the model. Over multiple iterations, the minimum validation_0-error was found out to be = 0.088235.
The overall accuracy obtained on test data = 0.911764705
The corresponding confusion matrix is plotted below:

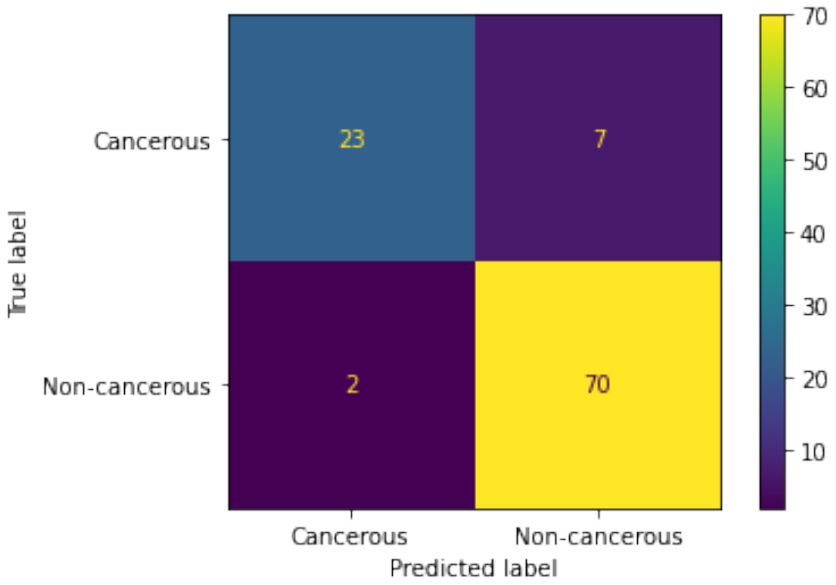

Figure 4: Confusion Matrix

Q2.6 and Q2.7 LightGBM implementation:

Using the inbuilt model in sklearn, we trained the model using the corresponding hyper-parameters(max_depth = 3) in the code. The highest valid_0's auc obtained during training = 0.99.
The graph for accuracy vs num_leaves when max_depth is 3 and 5 are given below, respectively:

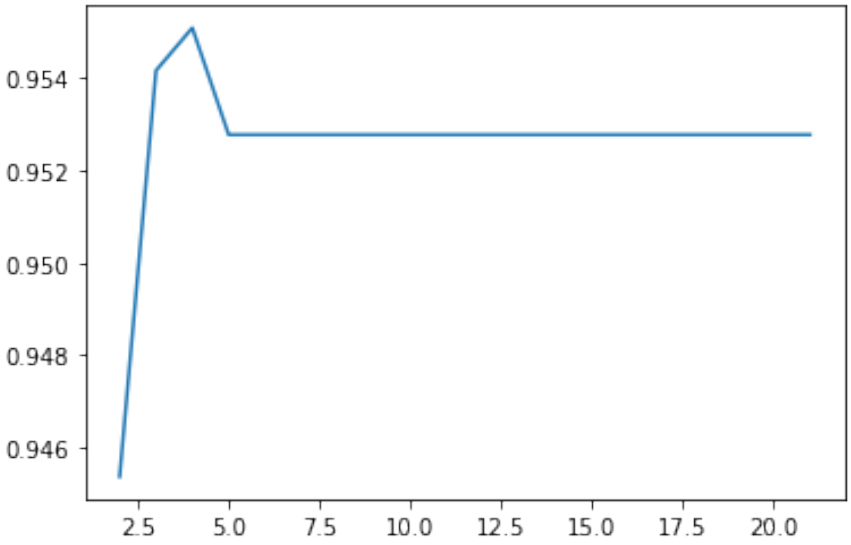

Figure 5: accuracy vs num_leave(max_depth = 3)

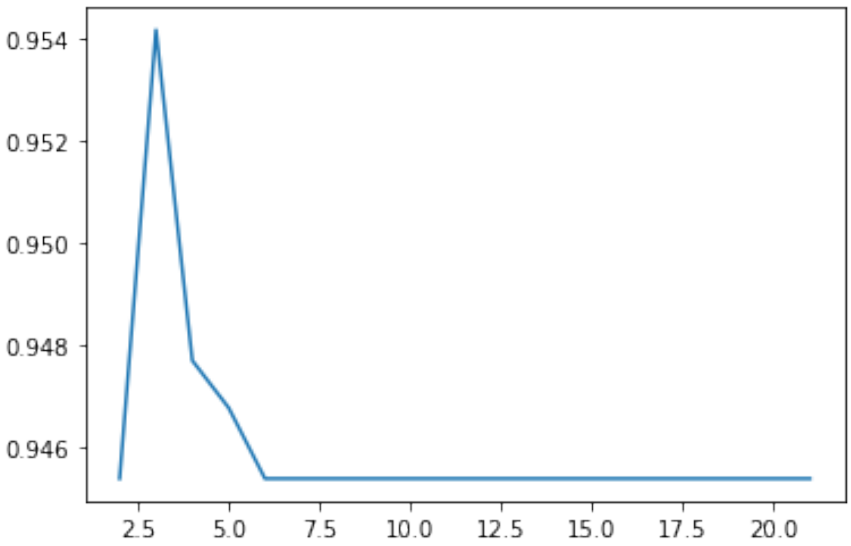

Figure 6: accuracy vs num_leave(max_depth = 5)

Using num_leaves, we can set the maximum number of leaves each weak learner has. Increasing num_leaves can increase the accuracy, but also pos-

sesses a risk of over-fitting above a certain value. In a LightGBM model, leaf-wise tree is deeper than a level-wise tree, there are even higher chances of over-fitting and thus we cannot direct use the theoretical value of num_leaves $= 2\hat{(}$max_depth) and instead, we should follow num_leaves $<= 2\hat{(}$max_depth). Considering the above two graphs, we can clearly see that the highest accuracy for test data is obtained when num_leaves $<= 2\hat{(}$max_depth), the test accuracy does not improve any further when we increase num_leaves but the accuracy on train sets increases in the later iterations. This indicates that the model starts over-fitting approximately when num_leaves $> 2\hat{(}$max_depth)

Q2.8 Parameter tuning:

1. num_leaves: As discussed in the above question, num_leaves $<= 2\hat{(}$max_depth), to avoid over-fitting and maximize accuracy.

2. max_depth: Using larger values of max_depth can cause over-fitting. It will also impact the best value for num_leaves parameter.

3. subsample: It indicates the fraction of rows from the original dataset that can be used to train the weak learners in each iteration. Smaller values increase the speed of training, moderate ones can improve generalization and the very high values(close to 1) can cause the model to over-fit.

4. feature_fraction: It indicates the fraction of features(columns) from the original dataset that can be used to train the weak learners in each iteration. It's behaviour is similar to subsample.

5. max_bin: Binning is used to convert continuous data to discrete values. The parameter indicates how many maximum bins the dataset can be converted into. When the value of max_bin is increased, the training of the model becomes slow but the accuracy increases and very high values can also cause the model to over-fit with respect to the training data.

3 References

- SciKit Learn Website - For details regarding inbuilt functions.
- Stack Overflow - For debugging errors.