Pattern Recognition and Machine Learning

(Winter 2022)

Assignment 2: Decision Trees

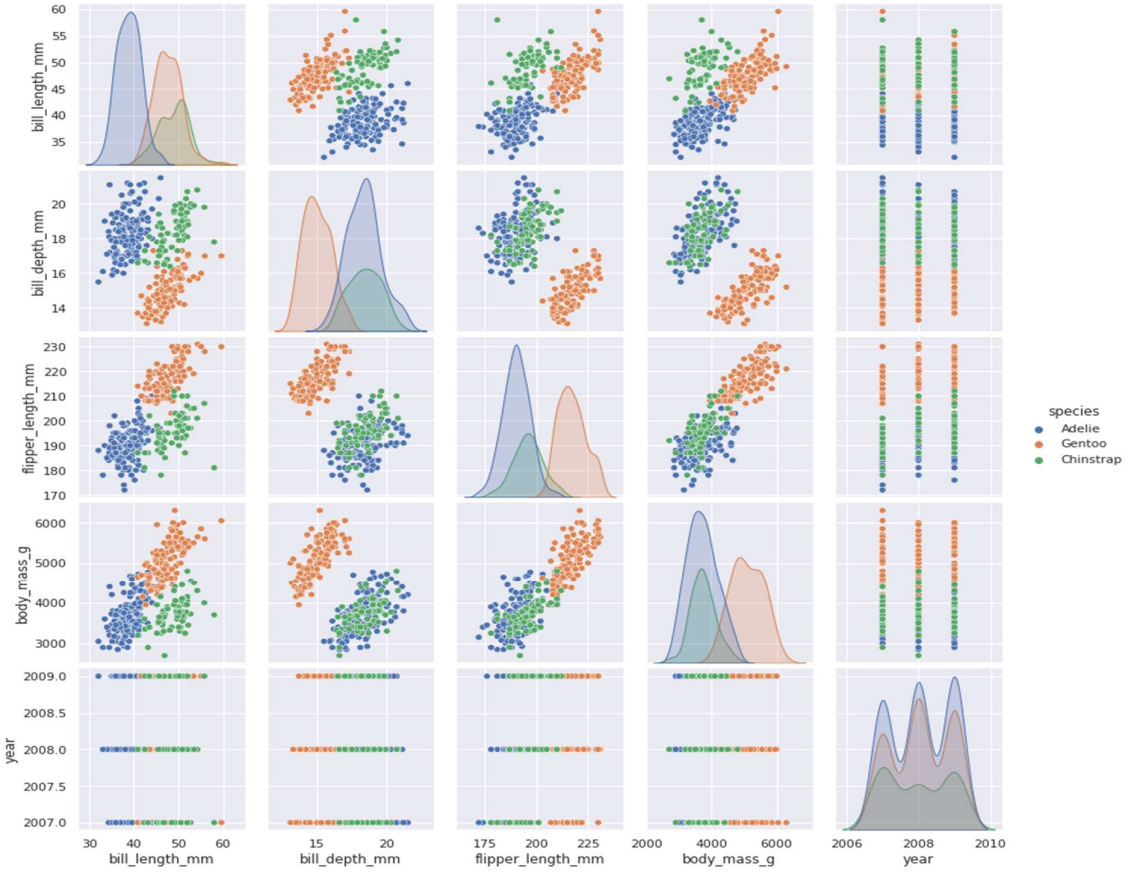
Report

Question - 1:

<u>Task - 1:</u> Perform pre-processing and visualization of the dataset. Perform categorical encoding wherever applicable and split the data into train and test sets

In first question, I first took the input of dataset. Then I performed the data pre-processing on the given dataset. I first checked if there is any null values present in the dataset, it turns out that there are some null/Nan/missing values in the data set. So, I just used to dropna () function to remove the Nan values from the dataset.

For the visualisation of the dataset. I used the seaborn library. I got these results:



From the visualization of the dataset, I could able to see the distribution of different attributes of the dataset.

Next, we have to perform categorical encoding wherever applicable in the dataset. So, for that I just made a list to store all the attributes that can be classified into categorical data. And, then I applied the LableEncoder () function from sklearn to encode the categorical data. Then, I split the data into train and test samples.

Task - 2: Implement the cost function as per your roll num.

In Next part we were asked to calculate cost function entropy/Gini on the basis of roll number. Since, my roll number is B20CS090. I calculated the entropy cost function. For that I just applied the concepts that I have learned in the class. I stored the occurrence of elements in particular feature/attribute of which the entropy is to be calculated. Then I divided the stored list (occurrence) by length of that list. Since in entropy formula we have to calculate proportions first. Then finally I just applied the formula of entropy to calculate it and returned the entropy calculated.

Task - 3: In order for the decision tree to work successfully, continuous variables need to be converted to categorical variables first. To do this, you need to implement a decision function that makes this split. Let us call that cont_to_cat ().

For the Next part, we were told to convert continuous data to categorical data. So, for this, I approached it as, I first stored all the continuous variable in data in list by using a simple for loop and checking the dtype. Now, since it is mentioned in the question to perform the optimal split i.e., to slit the data into 2 categories, based in the value of the function you have been allotted. So, for this I implemented the function information_gain () using my previously implemented cost function “entropy”. I just applied the concept of calculating the information gain as discussed in the class slides to calculate (Gain (S, A)). I iterated over unique values in particular attribute of dataset and made each subset of the dataset corresponding to the value and calculated its entropy and hence calculated the information gain.

Now, since I have implemented the function to calculate the information gain. I also made a function which will help in splitting the dataset into two categories based on the threshold and also assign values 0s to the ones that are less than threshold and 1s to those values that are more than or equal to the threshold.

Hence, finally I implemented the function to convert continuous data to categorical data. For that I iterated over each possible values in a particular column and calculated its information gain based on those values as threshold. And, I constantly updated the best info gain and best spilt threshold and finally returned the best info gain and best spilt threshold. I also made a dictionary to store the best spilt threshold.

<u>Task - 4&5:</u> After step 2, all the attributes would have categorical values, so now you can go ahead and implement the training function. This would include implementing the following helper functions:

a. **Get the attribute that leads to the best split**
b. **Make that split**
c. **Repeat these steps for the newly-created split**

The DT should also include the following properties in the train function

a. **There should be a max depth that should be defined i.e., a depth after which the tree shouldn't be allowed to grow**
b. **The algorithm should self-identify when there is no information gain being done, i.e., the model has plateaued in its training and shouldn't grow further.**

Now for the next part of first question, I implemented the logic for part 4 and part 5 altogether as in part 5 we have to just add an extra condition to check max depth and condition of tree to not go further. So, for 4th part a, I implemented the function best attribute, the logic behind that was to iterate over each column in dataset except the target column and calculated its information gain. The attribute with highest information gain make split first. So, I made a variable to store best information gain, and best attribute and updated its values over each iteration. Hence, I got the best attribute and best information gain corresponding to it.

Also, I made a function to make split of each attribute and store the values in left, mid, right variables. I just stored the dataset corresponding to when dataset[attribute] = particular value. For values 1, I stored in left, similarly for 0 and 2, I stored them in right and mid respectively.

Now, I just made the class for leaf node and decision node. Next, I implemented the function to build tree. Again, I used the concepts taught in the class. I followed the recursive approach I calculated the best attribute using the function that I implemented before and then made the spilt using the function that I implemented before and checked if the data is categorised into two or three categories based

on that I build the tree using recursion by calling the function again and splitting into two or three based on attributes and also incremented the depth by 1 on each recursive call. This is how the tree was built. Also, I kept on drop the column that has been already used to make the split.

So, for the next part we have to ensure that there should be a max depth that should be defined after which the tree should not be allowed to grow and also it should self-identify if there is information gain being done the tree should not grow further. So, I kept a condition in the build tree function to check if depth >= max depth than return node and also, I checked if entropy = 0 that means the node is homogeneous and cannot be split more so returned the leaf node.

<u>Task - 6:</u> Write a function which is responsible for classification (i.e., at test time).

For the next part that is responsible for classification, I made the testing sample also split it into two categories based on the best threshold that we got for each attribute in previous part. Now, I just checked if it is a leaf node, I appended the predictions into the list. And if it is a decision node, I checked if it's categorized into two categories or three categories and based on that classified the data and appended the predictions in the main list.

<u>Task - 7:</u> Find out the accuracy you get on the test data.

Finally, I calculated the accuracy on the testing data and predictions that we got using inbuilt function.

<u>Question - 2:</u>

<u>Task - 1:</u> Pre-process the data. Split it using a 70:10:20 ratio, which represents training: validation: testing.

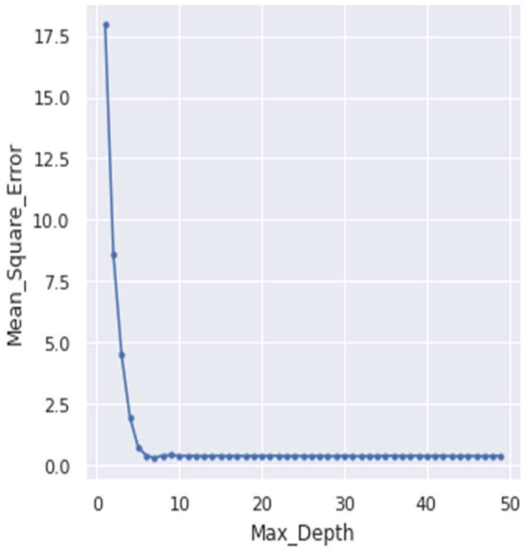
In the second question, I first took the input of dataset. Then I performed the data pre-processing on the given dataset. I used dropna () function to remove the Nan values from the dataset (if there are any) as a part of pre-processing of the data. Then, using the train test split function, I split the data into required ratios (i.e., 70: 10: 20).

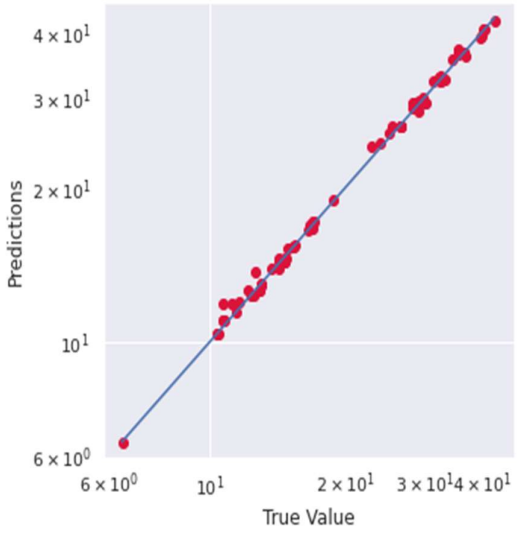
<u>Task - 2:</u> Write a function to train the data using a regression decision tree. The function varies hyper-parameters to find the tree that generalizes best (based on its performance on the validation set). So, you need to train on the 70% training data and check performance on the 10% validation data.

For this task I firstly initialized the decision tree regressor with random hyperparameters. I calculated the initial MSE of the decision tree with random hyperparameter. It came out to be huge. So, now as asked in the question, I varied different hyper-parameters in a defined range manually to get the best possible value of the each hyperparameter.

For Hyperparameter - 1 (max_depth):

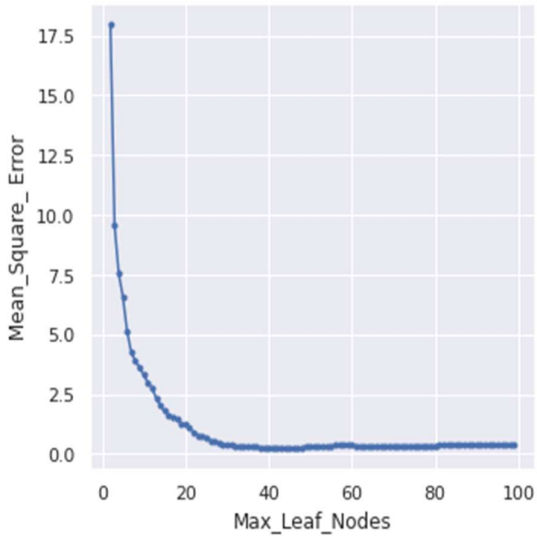
I iterated over a defined range of values for finding the best value for this hyper-parameter. I got at max_depth = 7. It shows lowest MSE = 0.2981042055579568.

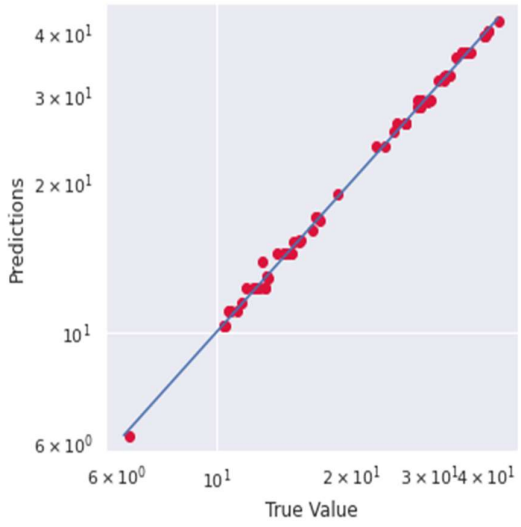




For Hyperparameter - 2 (max_leaf_nodes):

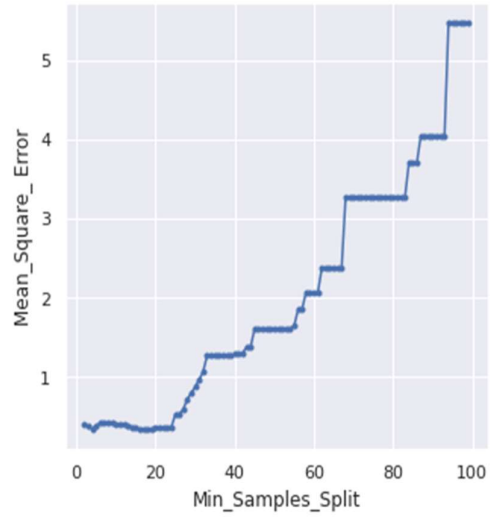
I iterated over a defined range of values for finding the best value for this hyper-parameter. I got at max_leaf_nodes = 47. It shows lowest MSE = 0.2507391939920361.

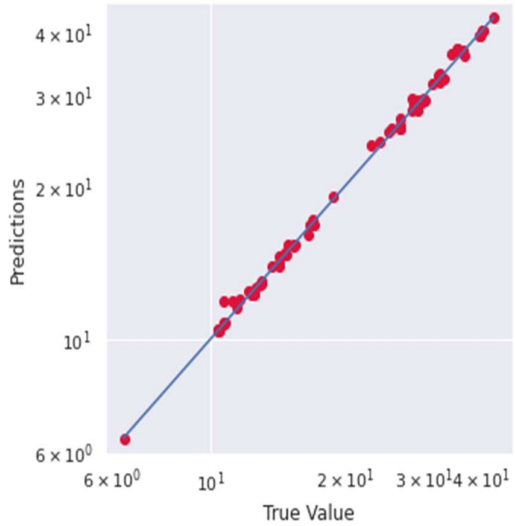




For Hyperparameter - 3 (min_samples_split):

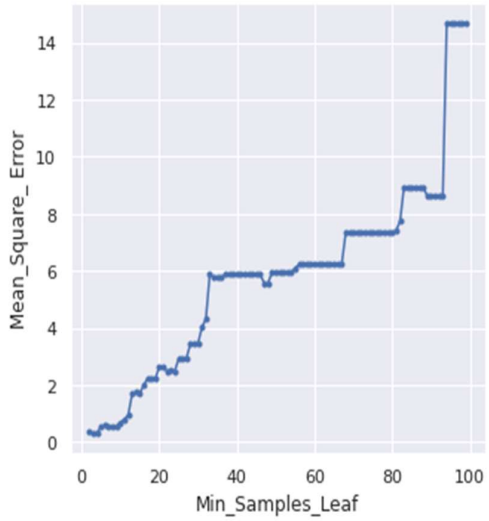
I iterated over a defined range of values for finding the best value for this hyper-parameter. I got at min_samples_split = 4. It shows lowest MSE = 0.3311442279942273.

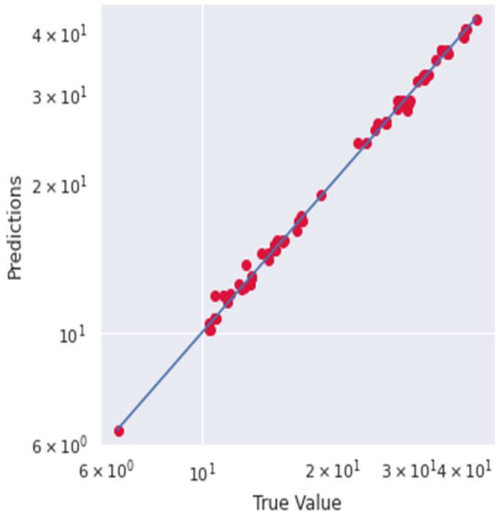




For Hyperparameter - 4 (min_samples_leaf):

I iterated over a defined range of values for finding the best value for this hyper-parameter. I got at min_samples_leaf = 3. It shows lowest MSE = 0.3110200292207795.





<u>Task - 3:</u> Perform 5-fold cross-validation using the optimal hyper-parameters decided in the previous question. Finally, calculate the mean squared error between the predicted and the ground-truth values in the test data for your best model. Also, plot the decision tree created.

For this task I used inbuilt function of sklearn to perform the 5-fold cross validation using optimal hyper-parameters that were found in previous task. For the calculation of MSE I again used inbuilt function and calculated Final MSE between the predicted and the ground-truth values in the test data for the best model. Hence, finally I plotted the decision tree using the inbuilt function.