# Lab 2 Report

**Palaskar Adarsh Mahesh**
B20EE087

February 1, 2022

## 1 Question 1

### Q1 .1 Pre-processing and Visualization:

Checked for null or missing values in the dataset and dropped the rows having null values. 11 such rows were dropped. Plots for some possible pairs of features were plotted and we can clearly observe that data from the year column provides no information about the distinction of penguin species, since all of graphs completely overlap each other. Therefore, we will drop this column from the data frame to avoid unnecessary trouble for the cost function, while determining the splits.(refer Figure 1)
After label encoding all the categorical attributes, now plots for all possible pairs of features are plotted which can be observed in Figure 2:

### Q1 .2 Cost function(odd roll number – gini index)

Implemented the function for calculating gini index for a particular attribute using formula for gini index. Then a function for calculating information gain was written using gini index.

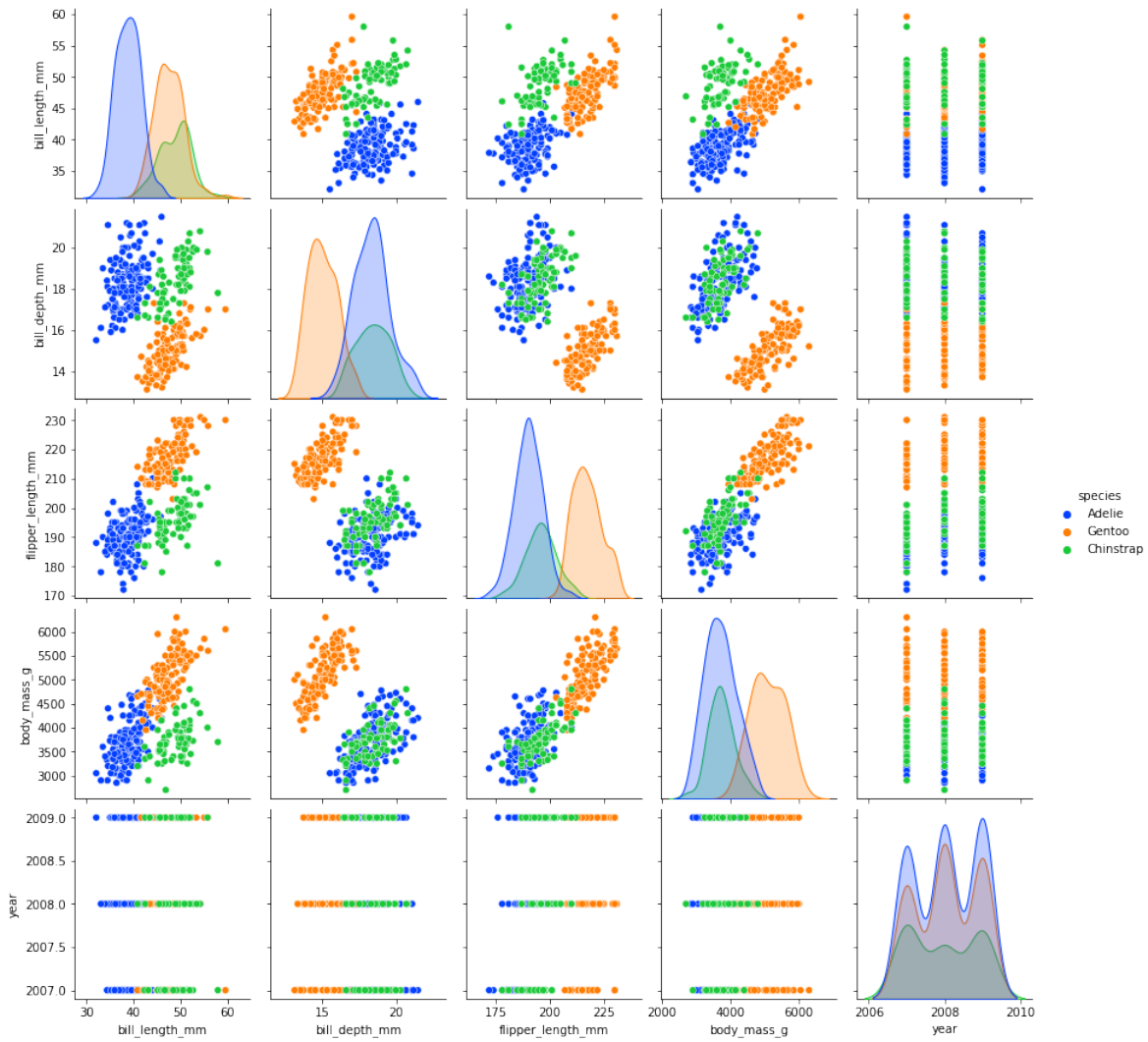


Figure 1: Visualization of given dataset


Figure 2: Visualization after removing year column

## Q1 .3 Converting continuous variables to categorical variables:

All continuous values are split into two categories, and the threshold at which the split will take place is determined by calculating the information gain for each value and assigning the threshold value to the value with maximum information gain. This is done by looping through all the values from all the continuous attribute columns. The graph for a continuous attribute is attached. We store their threshold values in the array 'thresholds_contvals'

and then encode these attributes by assigning 0's and 1's corresponding to each split.

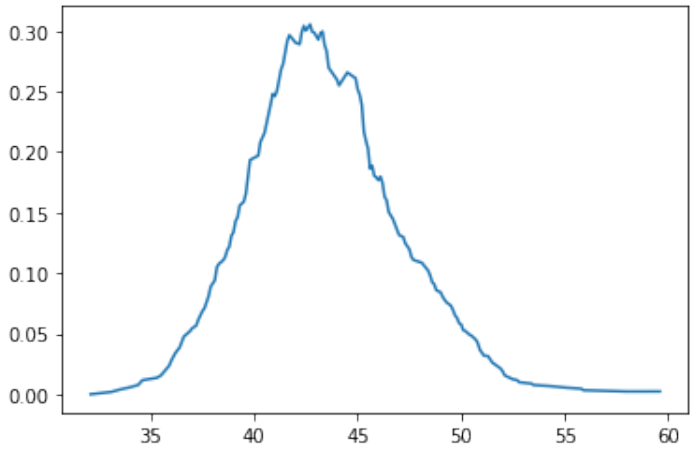


Figure 3: Calculating the threshold corresponding to highest information gain for bill_length (information gain v/s bill_length)

### Q1 .4 Training function:

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):
   (answer for Q1.4.(a)) 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):
   (answer for Q1.4.(b) and (c)) 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) The function also has two conditional statements:
   1. the split takes place only when the depth of the tree is less than the defined max_depth. It can access the value of max_depth defined in the constructor of the training function. (answer for Q1.5.(a))

   2. After the split we check whether the information gain due to the split is higher than the minimum value specified in the training function call, i.e it identifies automatically whether the information gain is very less (less than the specified value) and then stops the recursion (growth of the node where information gain is less). (answer for Q1.5.(b))

   (d) Classify():
   (answer for Q1.6)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.

The overall accuracy on the test data = 0.9402985074626866
The class wise accuracy is:
Accuracy for class Adelie = 0.9354838709677419
Accuracy for class Chinstrap = 0.8888888888888888
Accuracy for class Gentoo = 1.0

# 2 Question 2

### Q2 .1 Data Pre-processing:

Checked for null or missing values in the data, but no such values were found. Checked for outliers using box plots. No values lying outside the box were found. Thus the data does not contain any outliers. The data was split into a 70:10:20 ratio

### Q2 .2

Trained the decision tree regressor using inbuilt DecisionTreeRegressor and trained it on the data on default hyper parameters.
These default hyper parameters were:
$('ccp\_alpha' : 0.0,' criterion' :' squared\_error',' max\_depth' : None,' max\_features' : None,' max\_leaf\_nodes' : None,' min\_impurity\_decrease' : 0.0,' min\_samples\_leaf' : 1,' min\_samples\_split' : 2,' min\_weight\_fraction\_leaf' : 0.0,' random\_state' : 42,' splitter' :' best')$

The model was validated and and a very low mean squared error of 0.3569 was observed, meaning that the model was performing extremely well on the validation data. But in such cases we need to check if the model is overfittng to the train and and since the validation set does not have a lot of inputs, we cannot say that our model will perform extremely well on any data.
Checked for tree depth and number of leaf nodes, and they were found to be:
Depth = 14
Number of leaf nodes = 528
Given the training data had only 537 rows, we observe that the tree has entirely over fitted on the training data and is thus not generalized.

Hyperparameters to be tuned to obtained a generalized tree:

1. min_impurity_decrease: In the above model the since the min_impurity decrease is zero, the nodes will branch all the way until it becomes completely pure, which leads to over fitting. We can thus determine a value below which the node does not split.

2. max_depth: We can set a max_depth to the tree so that the tree does not grow beyond a certain depth, which helps to generalize it.

3. max_features: Using an inbuilt feature, the importance for each feature was calculated in the above model. Importance of a feature roughly tells us about which feature was used the maximum number of times. We observe that feature X5 is not used at all and the X2,X4,X6 are rarely used. We can thus set a cap on the maximum number of features the training algorithm can use to classify the test data, to avoid unnecessary splits.

4. max_leaf_nodes: We can set a max number of leaves the tree can have to stop unnecessary growth.

5. min_samples_leaf: It indicates the minimum samples that must be present at a leaf node. Setting this parameter ensures the node does not split for a very less number of samples which can lead to loss of generalization.

Most of these hyper_parameters cannot be determined manually by just looking and visualizing the data. Thus, arrays for potential values for each hyperparameters were made and we test the on the validation dataset to check which set of parameters will provide us with the best results. But manually filling the dataset parameters in the arrays will reduce the chances of over fitting the tree to the training data.
Iterating over all the possible set of hyper parameters,we obtain these as out optimum set of hyper parameters which correspond to the minimum mean squared error obtained on the validation set among all the possible sets of parameters:
$('ccp\_alpha' : 0.0,' criterion' :' squared\_error',' max\_depth' : 6,' max\_features' : 5,' max\_leaf_nodes' : 21,' min\_impurity_decrease' : 0.1,' min\_samples\_leaf' : 12,' min\_samples\_split' : 2,' min\_weight\_fraction\_leaf' : 0.0,' random_state' :$

$42, 'splitter' : 'best')$

The graph corresponds to the mean squared error obtained for each set of hyperparameters.

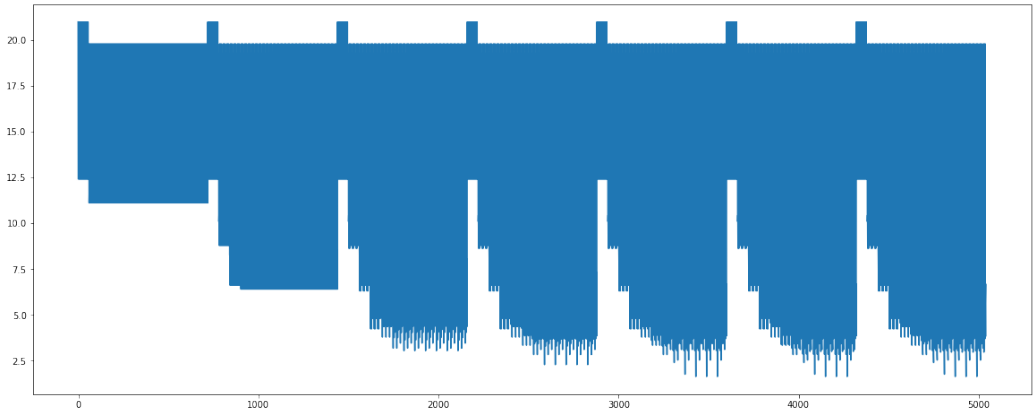


Figure 4: Calculating mean squared error corresponding to each set of hyper parameters(mse v/s index of parameters array)

**Q2 .3**

For performing 5-fold cross validation we merged the train and validation splits. The cross validation scores(mean_absolute_errors) for each fold were as follows:
$[1.33773806, 1.39862411, 1.13899925, 1.22774675, 1.38643595]$

Using the best set of hyper_parameters obtained above, another model is trained and mean squared error on the test data is = 4.111530878167731
The depth for the tree is reduced to 6 and the number of leaf nodes are now 21, which states that we avoided the tree from over fitting to the train data.
The plot for the best_model is given:

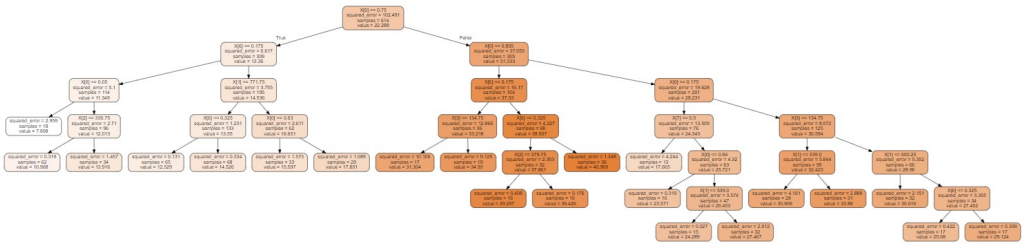

Figure 5: Decision Tree

# 3 References

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