Report for the ML Assignment-1

Developing a Decision Tree Classifier in Python

Prepared by

Shobhit Gupta
19EC10058

Shreyash Vaish
19EE30030

Dataset - Indian Liver Patient Dataset

url - https://www.kaggle.com/jeevannagaraj/indian-liver-patient-dataset

This data set contains 416 liver patient records and 167 non-liver patient records. The data set was collected from test samples in North East of Andhra Pradesh, India. 'is_patient' is a class label used to divide into groups(liver patient or not).

Sample from the data-

	age	gender	tot_bilirubin	direct_bilirubin	tot_proteins	albumin	ag_ratio	sgpt	sgot	alkphos	is_patient
0	65	Female	0.7	0.1	187	16	18	6.8	3.3	0.90	1
1	62	Male	10.9	5.5	699	64	100	7.5	3.2	0.74	1
2	62	Male	7.3	4.1	490	60	68	7.0	3.3	0.89	1
3	58	Male	1.0	0.4	182	14	20	6.8	3.4	1.00	1
4	72	Male	3.9	2.0	195	27	59	7.3	2.4	0.40	1

Attributes

1. *age* Age of the patient
2. *gender* Gender of the patient
3. *tot_bilirubin* Total Bilirubin
4. *direct_bilirubin* Direct Bilirubin
5. *tot_proteins* Total Proteins
6. *albumin* Albumin
7. *ag_ratio* Albumin and Globulin Ratio
8. *sgpt* Alamine Aminotransferase
9. *sgot* Aspartate Aminotransferase
10. *alkphos* Alkaline Phosphotase
11. *is_patient* Selector field used to split the data into two sets (labeled by the experts)

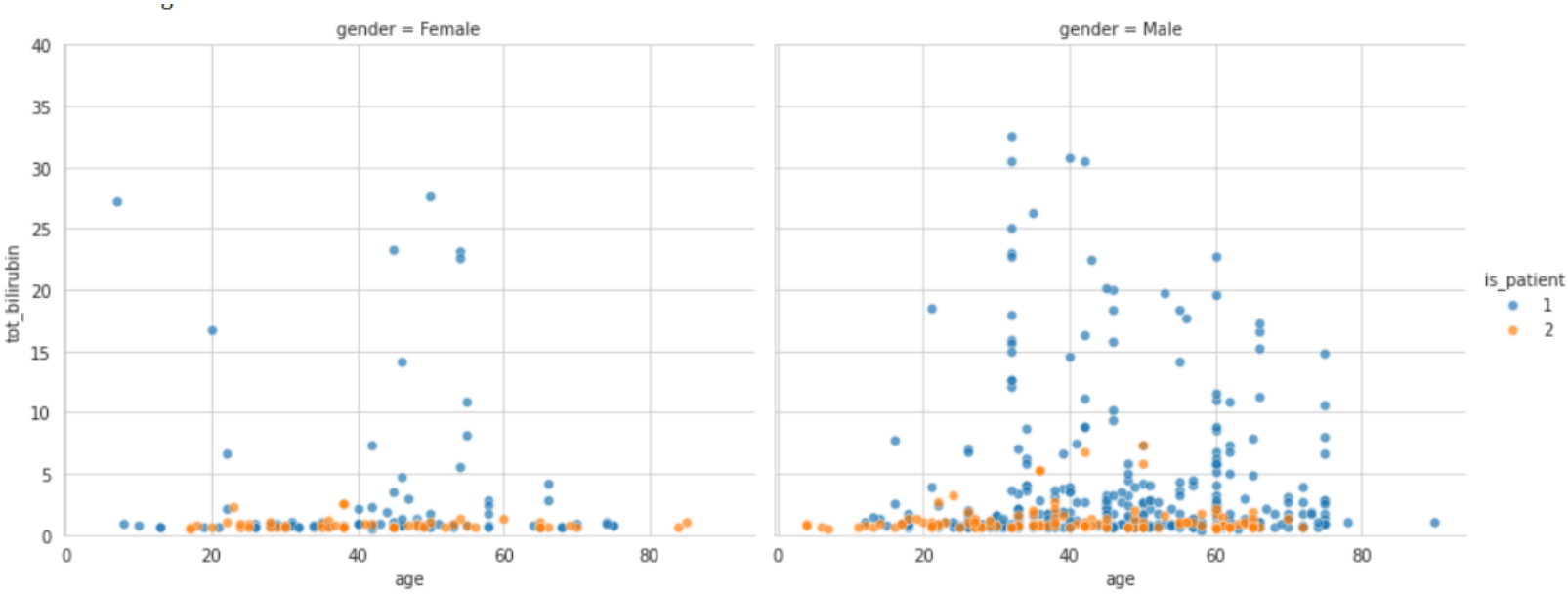
Description of the data

	age	tot_bilirubin	direct_bilirubin	tot_proteins	albumin	ag_ratio	sgpt	sgot	alkphos	is_patient
count	583.000000	583.000000	583.000000	583.000000	583.000000	583.000000	583.000000	583.000000	579.000000	583.000000
mean	44.746141	3.298799	1.486106	290.576329	80.713551	109.910806	6.483190	3.141852	0.947064	1.286449
std	16.189833	6.209522	2.808498	242.937989	182.620356	288.918529	1.085451	0.795519	0.319592	0.452490
min	4.000000	0.400000	0.100000	63.000000	10.000000	10.000000	2.700000	0.900000	0.300000	1.000000
25%	33.000000	0.800000	0.200000	175.500000	23.000000	25.000000	5.800000	2.600000	0.700000	1.000000
50%	45.000000	1.000000	0.300000	208.000000	35.000000	42.000000	6.600000	3.100000	0.930000	1.000000
75%	58.000000	2.600000	1.300000	298.000000	60.500000	87.000000	7.200000	3.800000	1.100000	2.000000
max	90.000000	75.000000	19.700000	2110.000000	2000.000000	4929.000000	9.600000	5.500000	2.800000	2.000000

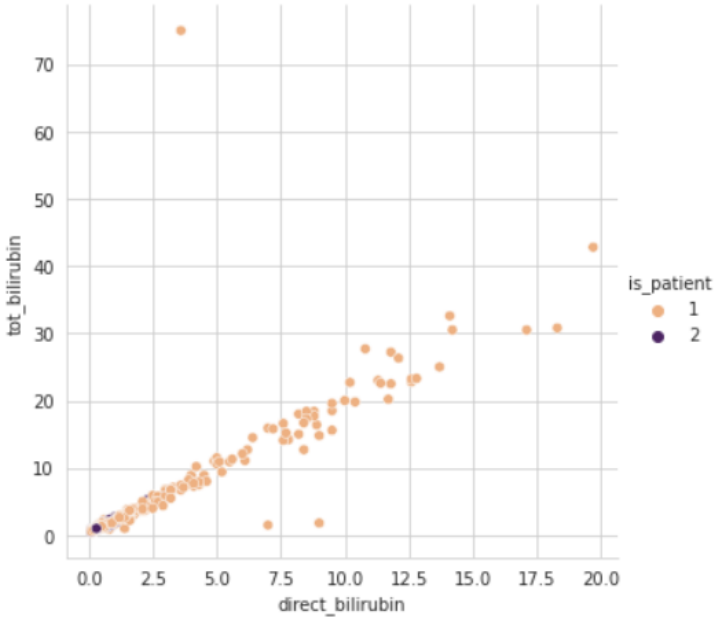
The attributes associated with blood composition are continuous-valued. Hence, all the attributes except age, gender and is_patient are continuous-valued.
Number of unique values of each attribute are given below -

```
Name: is_patient, dtype: int64
age                  72
gender                2
tot_bilirubin       113
direct_bilirubin     80
tot_proteins        263
albumin             152
ag_ratio            177
sgpt                 58
sgot                 40
alkphos              69
is_patient            2
dtype: int64
```

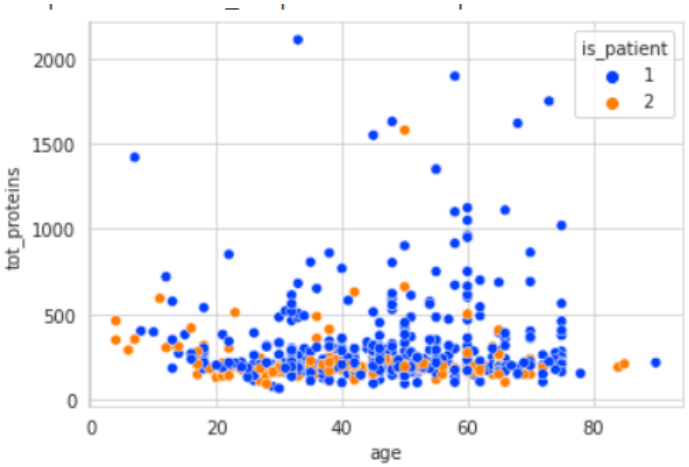
Age and lifestyle habits affect the liver condition of a human.
The blood composition of the patient can be used to predict if he has a liver problem.



Based on the above plot, we can say that most of the liver patients have higher total bilirubin content in the blood. Also, for the same amount of bilirubin content, people with higher age(of both genders) are more susceptible to liver problems.

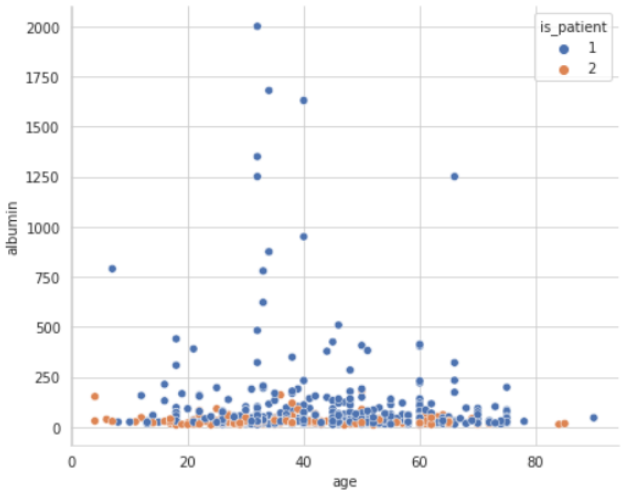


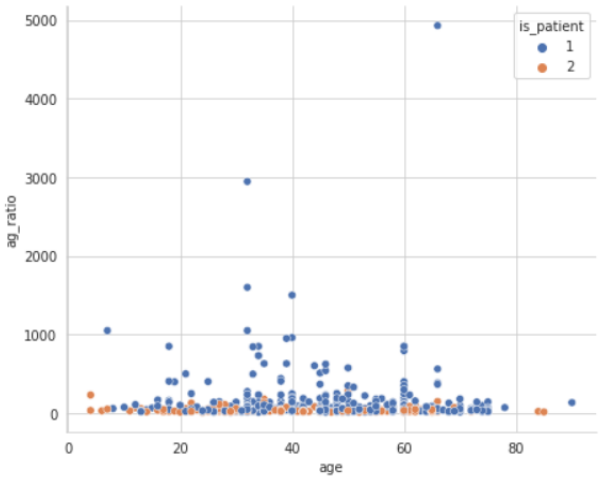
This plot shows that almost all the liver patients have higher amounts of bilirubin. Also, total bilirubin and direct bilirubin are linearly related.

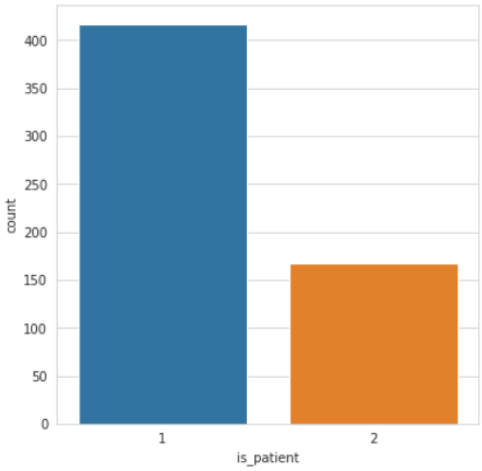


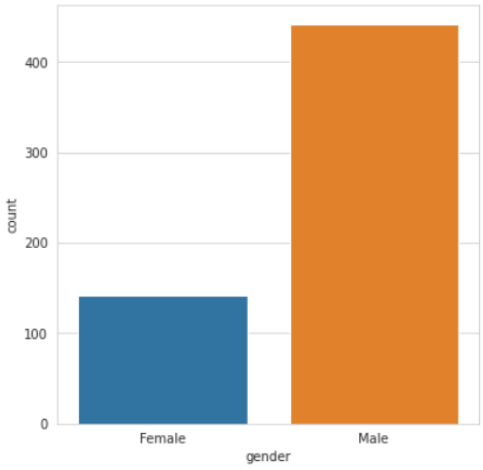
From the plot, we see that most liver patients had higher total protein levels in blood.

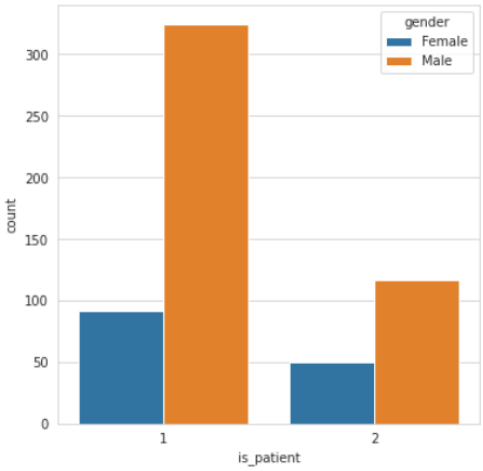
Similarly, we can observe from the data that high levels of albumin and albumin-to-globulin ratio is dangerous for the liver.



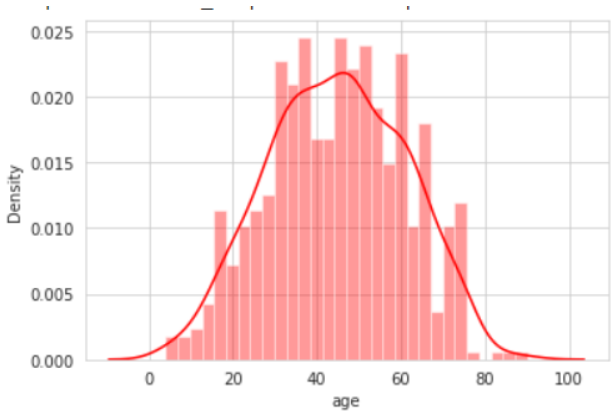








The first plot shows the count of patients and non-patients. The dataset has 416 records of patients and 167 records of non-patients. The second plot shows the data distribution by gender. The number of patients and non-patients grouped by each gender are shown in the third plot.



age	0
gender	0
tot_bilirubin	0
direct_bilirubin	0
tot_proteins	0
albumin	0
ag_ratio	0
sgpt	0
sgot	0
alkphos	4
is_patient	0

dtype: int64

This plot shows the age distribution of the entries in the dataset. The data closely follows the normal distribution with respect to age. Hence, the data is symmetrically distributed about the mean age.

Originally we can see that the data contains null values.
Since, the number of null values are very less and exists only for alkphos attribute, we replaced the null values in this attribute by the value with most occurrence.

How to compile and run the code?

FOR WINDOWS & LINUX-

1. Extract the "Group31_Assg1.zip".
2. Make sure Python3 is installed on the system.
3. Install the dependencies stored in the "requirements.txt" file using the following command - **pip install -r requirements.txt**
4. For running any python file, use the following command - **python filename.py**
    i) For viewing the data analysis part, run the "analyse.py" file.
    ii) For each sub task i = 1,2,3,4,5., run the corresponding "task_i.py" file.

Description of files in the root folder of assignment

1. data.py:
   Contains the data class and methods required to store/manipulate the data.

2. decision_tree.py:
   Contains the decision tree class and all its methods.

3. analyse.py:
   This file deals with the data analysis part. All the tables and plots shown previously can be generated by executing this file.

4. task_q.py (q = 1, 2, 3, 4) contains the code for sub task q (= 1, 2, 3, 4).

Pre-processing the data

The gender attribute was processed by assigning male as 0 and female as 1. Also, the 4 missing values in "alkphos" attribute were replaced by its most frequent value in the given dataset.

i) Building the decision-tree classifier

using both information and gini gain heuristics

Approach -

At first the training examples are assigned to the root of the tree, Then at each present leaf node of the tree, the following is done -

1. If the leaf node is pure, then do not split it and proceed to the next leaf node.
2. Else, select the best attribute for the split and make new leaf nodes and distribute the examples amongst them.

Selection of Attribute -

For each candidate attribute A present in the examples belonging to the node of tree, the examples are first sorted according to the value of the attribute A, then -

Let V_A = {v1, v2, ...} be the values possible for attribute A in the examples at present node and V_A be in increasing order. Then the threshold is varied as $(v_i + v_{i+1}) / 2$, the examples are split by this threshold and information gain is computed. The threshold with highest information gain is taken for the attribute A.

Now, after getting the threshold for each attribute A, the examples at present are split and either information or gini gain is calculated depending upon the choice. The attribute with highest gain is selected for splitting the examples at the present node and new leaf nodes are then created with their corresponding examples. The information of the splitting attribute and its threshold is also stored at each node.

Splitting the dataset -

For splitting the dataset, random indices are selected using the random library and then those examples are given to test data and the rest examples are assigned to training data.

Results -

The information gain and gini gain both seem to provide nearly the same accuracy with information gain taking the lead sometimes by little amount. Following are accuracy measures for some runs -

```
test_acc by training with information gain = 0.6724137931034483
test_acc by training with gini gain = 0.6551724137931034
```

```
test_acc by training with information gain = 0.7241379310344828
test_acc by training with gini gain = 0.6896551724137931
```

```
test_acc by training with information gain = 0.7241379310344828
test_acc by training with gini gain = 0.7413793103448276
```

The accuracy seems to have high variance though the decision tree trained with both the gains provides accuracy between 0.65-0.73 mostly.

The code for the above analysis can be found in task_1.py. The codes for building the tree and manipulating the data can be found in Decision_tree.py and Data.py respectively.

ii) Average performance of the classifier

finding avg. test accuracy over 10 random splits

Approach -

In this part, we randomly split the data into training and test data. Then, train the decision tree classifier over the chosen train and test sets by choosing both the heuristics - information gain and gini gain. We repeat the above process 10 times and each time, we add the test accuracy to its previous test accuracies for both the heuristics and store them in two separate variables.
In the end, we divide the accuracies by 10 to obtain average test accuracies over 10 random splits.

Results -

The average test accuracies for the information gain and the gini gain as given by the “task_2.py” file are reported below -

```
The average accuracy for information gain is 0.6724137931034483
The average accuracy for gini gain is 0.6534482758620689
```

Based on the above output, we can see that information gain has a slightly better average accuracy compared to the gini gain heuristic. Hence, we will choose information gain as our heuristic for building the decision tree from the next task onwards.

iii) Finding the best possible depth limit

plotting test accuracy vs depth and test accuracy vs #nodes

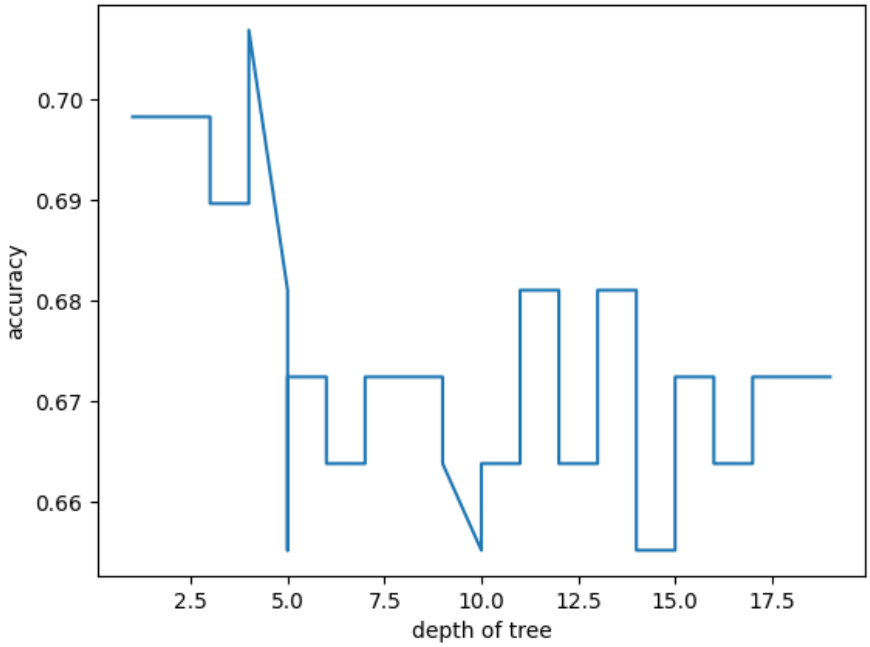
Approach -

After making the choice of heuristic as **information gain**, we again train the decision tree classifier model but this time, we also store the test accuracy at each depth and also the number of nodes at each depth. Here, every node has two children, since all the continuous valued attributes have been divided into two classes by choosing a threshold (as seen in task 1). The training of the tree is done in breadth first manner, hence depth is increasing by 1 at most at a time for efficiency purpose. As training of the tree continues, the test accuracy is measured every time and so we have the test accuracy corresponding to each depth and to the number of nodes. Hence, we can plot the test accuracy vs depth of tree and test accuracy vs number of nodes.

Results -

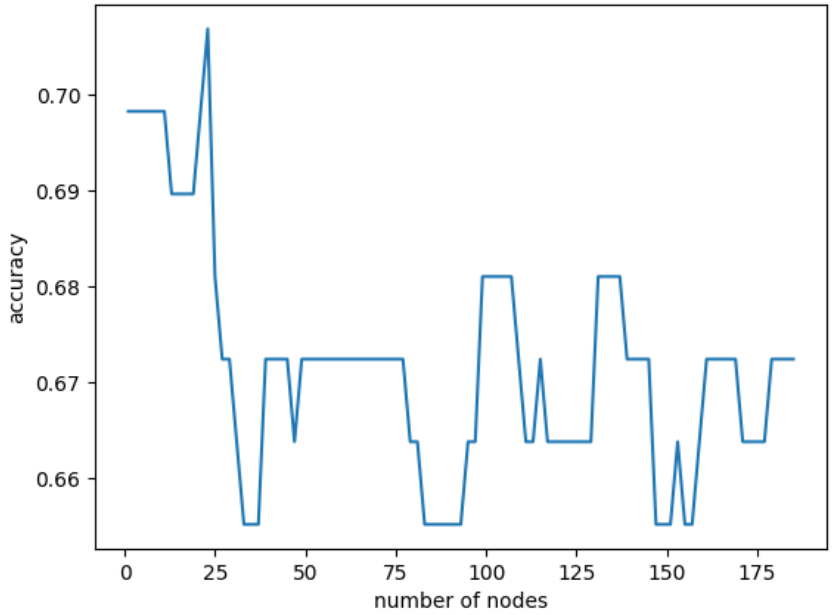
For plotting purposes, we used the python “matplotlib” library. By running the “task_3.py” file, we get the following plots in the files “acc_chang_with_depth.png” and “acc_chang_with_number_of_nodes.png”-

Plot for test accuracy vs depth of tree:



From the plot, maximum test accuracy occurs at depth = 4. This can be explained by the fact that as the depth increases, the classifier starts overfitting on the training data, hence, the test accuracy decreases.

Plot for test accuracy vs depth of tree:



As the number of nodes are increasing, the chances of overfitting increases, hence the test accuracy decreases.

iv) Pruning the decision tree

by using statistical testing method

Approach -

First, we train the decision tree classifier with **information gain.** Then, we perform pruning on the trained decision tree using statistical testing method.

Chi-square statistical testing method -

The problem in splitting with information gain is that we split whenever the information gain increases, but we never check if the change in entropy is statistically significant. So, this method is an attempt to remove useless splits from the decision tree.

First of all, we calculate the following value for each non-leaf node -

$$K = \sum_{\substack{\text{all classes } i \\ \text{children } j}} \frac{(N_{ij} - N'_{ij})^2}{N'_{ij}}$$

where N_{ij} = number of points of class i in child j, N_{ij}' = expected number of points of class i in child j using distribution of classes in the present node.

The non-leaf nodes where the value of K is less than the chosen threshold are made leaf nodes. The prediction at leaf node is given by the most frequent target value.

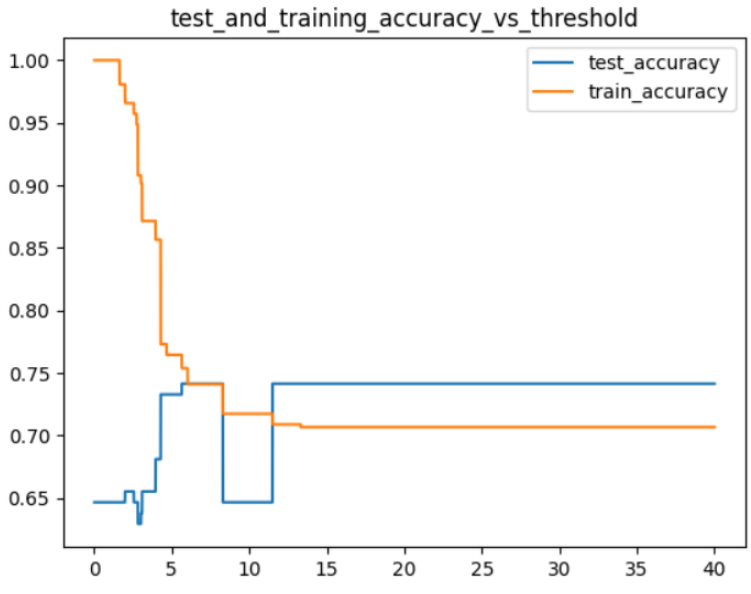
Choosing the threshold -

We will vary the threshold from 0 to 40 (taken by observation) and remove the useless splits by chi-square method. Then, we plot the training and test accuracy vs threshold curve and determine the optimal threshold where the training and test accuracy are reasonably high. Finally, print the training and test accuracy of the pruned decision tree.

Results -

After several runs of the “task_4.py” file, we find out that threshold = 5.5 works reasonably good for most train-test splits. Hence, we choose 5.5 as the threshold.

Plot generated by running the “task_4.py” file with threshold=5.5 is stored in “test_and_training_accuracy_vs_threshold.png” -



From the plot,
Training accuracy = 0.74088
Test accuracy = 0.74141

Other method for post-pruning -
The sub trees to be pruned can also be evaluated by testing accuracy with cross validation and then removing the subtree that increases the accuracy than present accuracy. This is done till the pruning increases accuracy.

The code for both the above methods can be found in DecisionTree class methods pruneByChiSquare and pruneByCrossValidation respectively.

v) Printing the decision tree

obtained from task 2

Approach -
We train the model in a similar way as we did in task 1. After that, we traverse the tree in the breadth first manner. We start by printing the root, and we push the left child and then the right child into the queue. Then, the depth of the child is also pushed in the queue and we print the elements present in the queue level by level i.e the nodes at the same level are printed in the same line.

Structure of the printed decision tree -
For each non-leaf node, we print in this manner : splitting attribute > threshold. The left child contains the examples splitting attribute <= threshold and the right child contains the examples with splitting attribute > threshold.
For each leaf node, we print the value of the target attribute with highest occurence for that node in the following manner: is_patient = 1 (or 2) and it has no child.
To get the child nodes of nodes at a level, keep the pointer at the first node of current level and next level. If a non-leaf node, The left child of the current node is the first node in the next level which is not assigned to any node in current level and the right child is next to this left child and Move the pointer in next level to next of right child and move the pointer in current level to next node. If a non-leaf node, just update the pointer in the current level to the next node.

Results-
The decision tree generated by running the “task_5.py” file is generated and stored in the “unpruned_decision_tree.txt”.

A small portion (starting from the root) of the decision tree generated is given below -

```
[ direct_bilirubin > 1.2 ]
[ albumin > 66.5 ]  [ direct_bilirubin > 3.6 ]
[ age > 72.0 ]  [ age > 39.0 ]  [ tot_proteins > 621.0 ]    [ is_patient = 1 ]
[ tot_proteins > 218.0 ]    [ is_patient = 1 ]  [ tot_proteins > 184.5 ]    [ is_patient = 1 ]  [ tot_proteins > 147.5 ]    [ sgot > 2.55 ]
```

Conclusion

With this assignment on implementation of a decision-tree classifier in python, we went through each and every step of building this machine learning model in detail. The dataset we used is “Indian Liver Patient Dataset”. To get insights and to understand the data, we start off with analyzing the data and the relationships among its various attributes and how our target attribute - “is_patient” is affected by these individual attributes.

The next step is to pre-process the data, that is, to use an effective strategy to remove the null values in the data, encoding any string-valued attributes to numeric values. Since, there are only 4 null values throughout the dataset, that too, in the same attribute - “alkphos”, we replaced these 4 null values by the most frequent value for that attribute. Also, “gender” was a string-valued attribute, so it had to be converted to numeric-values, where 0 = Male and 1 = Female.

Then, the next step is to build the decision-tree. We grow the tree from root to leaf in the breadth-first manner, that is, we define all the children of the parent node at present depth and then move to the next depth. We chose the attribute and its best possible threshold at the present node for splitting. The best threshold of the attribute is where the information gain after splitting the examples is maximum, then the required gain (either information gain or gini gain) is computed for the split(done by its best threshold) for present attribute and then the attribute whose split has maximum gain is selected for splitting the examples at present node.

In order to establish which heuristic is better for the given dataset, we train the model on 10 random 80/20 splits of the dataset and then find the average test accuracy and choose the tree with best test accuracy. After making the choice of heuristic, we once again build the tree but this time, we also note the test accuracy at each depth and find the best possible depth limit to which the tree doesn’t overfit the data. This will ensure better generalization over new data.

Another method to ensure good test accuracy and avoiding overfitting is post-pruning, that is, we first build the complete decision-tree, then we try to make the model simpler so that generalization is better. One method of post-pruning is the “Statistical Testing method”. Here, we go through all the splits present in the already built tree and then remove those splits which result in a very little or no information gain (or gini gain). We remove all the splits whose gain is less than a particular threshold. The threshold is varied over a range of values and the one with reasonably good training and test accuracies is chosen to build the final model.

Decision trees are very useful in interpreting the data in a highly visual way. To visualize our model, we have to print it. In order to print the tree, we traverse the model in the breadth-first manner and print all the non-leaf nodes in the form of a boolean function, and its value - true or false - determines the path in the tree. For any leaf node, we directly print the prediction. In this way, we get the complete visualization of the decision-tree.

After completing the assignment, we now have a better understanding of how a decision tree is built and the various techniques with which we can make the model robust and give a better performance on unseen test data. But at the same time, they are not very good for continuous-valued data and in case of imbalanced datasets. Despite these drawbacks, decision trees are still very popular in carrying out predictive analysis with practical applications across sectors from health, to finance, and technology.