UCI Car Evaluation
Using Decision Tree Based Learning Model

Group F:
Abhinandan De(19CS10069)
Yashica Patodia(19CS10067)

1. Introduction

Car Evaluation Database was derived from a simple hierarchical decision model originally developed for the demonstration of DEX(M. Bohanec, V. Rajkovic: Expert system for decision making. Sistemica 1(1), pp. 145-157, 1990.). The Car Evaluation Database contains examples with the structural information removed, i.e., directly relates CAR to the six input attributes: buying, maint, doors, persons, lugboot, safety.Because of known underlying concept structure, this database may be particularly useful for testing constructive induction and structure discovery methods.

2. Overview of Decision Tree

Decision tree is the most powerful and popular tool for classification and prediction.A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label. It is a predictive machine-learning model that decides the target value (dependent variable) of a new sample based on various attribute values of the available data. The internal nodes of a decision tree denote different attributes; the branches between the nodes tell us the possible values that these attributes can have in the observed samples, while the terminal nodes tell us the final value (classification) of the dependent variable. The attribute that is to be predicted is known as the dependent variable, since its value depends upon, or is decided by, the values of all the other attributes. The other attributes, which help in predicting the value of the dependent variable, are known as the independent variables in the dataset.

2.1. Construction of Decision Trees

The tree is constructed by following a top down approach, where it starts with an empty root. Then it starts splitting based on the best decision attribute, where decision attribute is considered 'A'. After that, for each value of A, new descendant of node is created. Training examples(S) are sorted to leaf nodes and stopped unless all examples are classified and until then new leaves are created. To choose the best attribute, information content of the attributes are measured.

2.1.1. Information Gain as the Impurity Measure

Features are split based on the fact that less uncertain attribute will have more information gain. That can be calculated using Entropy, which is a measure of the uncertainty in S or randomness of S. The formulae of Entropy of S is as follows:

$$Entropy(S) \equiv -\sum_{i=1}^{n} p_i \log_2 p_i$$

where p_i is the proportion of examples of Class i in S. Information Gain based on an attribute A, shows that, by how much uncertainty is reduced. Here lies the mathematical formulation of Gain:

$$Information\_Gain(G, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

where $|S_v|$, $|S|$ and $Entropy(S_v)$ are number of training examples with value v of Attribute A, number of training examples and Entropy of S_v .

Hence the feature with the largest information gain is used as the root node of the decision tree ID3, since ID3 uses the method of entropy.

2.1.2. Gini Index as the Impurity Measure

Another such metric used to measure impurity is Gini Index. Gini index of S is given by the formulae:

$$Gini(S) \equiv 1 - \sum_{i=1}^{n} (p_i)^2$$

In case of this metric, the attribute which has lower Gini Index,is chosen for split. This metric is comparatively easy to implement and calculate, so it is favourable in case of larger partitions, whereas entropy and information gain favors smaller partitions with distinct values.

2.2. Overfitting in Decision Trees

It becomes a challenge to maintain, when an algorithm comes up with a drawback, just like here Decision Tree comes up with the issue of overfitting. When there is some noise in data, or when the number of training examples is too small then it can lead to difficulties since it becomes hard to produce a representative sample of the true target function. So, a simple decision tree algorithm can overfit the training examples. When the hypothesis, that fitted too much with training data, gives more error for test data whereas less error for train data then that phenomenon states overfitting. To check overfitting, we can split he train dataset into train and validation sets using cross validation.

2.2.1. Pruning

To avoid overfit situation, we can use pruning while splitting the tree. Pruning can be used to stop the growing tree earlier, letting to touch only till the point where it can be perfectly classified. That can be done using Reduced Order Pruning, by considering each nodes in the tree to be a candidate for pruning. It constitutes of removing the sub-tree rooted at that node, making it a leaf node, and assigning to it the most common classification of the training examples connected with that node. But removing the node is done only on the condition that the new tree does not make the classification worse than that of the cross-validation of original set. Pruning continues till further pruning is harmful. So to sum up, decision tree is one of the most preferred learning algorithms as it has the flexibility to include every possible situations like handling noise, pruning, handling discrete and continued-valued attributes.

3. Experimental Observation

We have used the ID3 for designing the Decision Tree.The data used for training was provided in .csv format.a. We considered the split of 80:20 a Train and Test set respectively.We have tried implementing all these things mentioned in the previous section, in python with and without using in-built library files of python.

3.1. Step-Wise Description of Implementation of Major Functions

3.1.1. data_handling.py

This python file contains helper functions which read the input file in csv format and split the data in training and test part respectively.

1) **gen_test_and_validation_set:** Generates training data and test data in the ratio 80:20.
2) **get_classification:** This will just return the classification of a list of indices in a dictionary.
3) **split_values:** This will split the values by the attribute,then it will return a dictionary of the list of indices supported by the new conjunction.

3.1.2. decision_tree.py

This python file contains the major functions which construct the Decision Tree.

1) **compute_values:** We are finding out the classifications of our input data.
2) **propagate:** Now we have to construct it further.We can stop if we have a pure split i.e. all out data points are in 1 category or we have exhausted all attributes.
3) **assign_index:** Dfs function to assign unique indices to each node.
4) **predict_cal:** Predicts the output value of the decision tree for a given conjunction.This also keeps track of the current depth in the dfs.This tracking comes in handy when we need to find the best possible depth for out decision tree.
5) **predict_value:** This function wraps around the predict_cal function.Given a validation set, it iterates over the set and gets the predictions from the predict_cal function.It then returns the list of predictions.
6) **get_locations:** This is another dfs function useful for pruning.This helps in quickly pruning a particular node instead of having to do a dfs everytime we need to prune a node.
7) **alter_prune:** Function to alter between a leaf and a non leaf node.Basically it will just set it to a leaf.This can be checked while printing the tree.

3.1.3. description.py

This file describes the content of our dataset.It contains the following attributes

1) **header_list:** The headers in the csv.
2) **attr_list:** The attributes that are to be tested.
3) **possible_values:** Possible values of the attributes.
4) **classifications:** Finally the possible target values.

3.1.4. impurity_calculators.py

This file calculates the impurity attribute for a particular node by both information gain and gini index.

1) **gini_index:** This takes in a list of the classifications and then computes the gini_index using the appropriate formula
2) **entropy:** This takes in a list of the classifications and then computes the entropy using the appropriate formula

3.1.5. solve.py

This is the main file which contains the implementations of all the parts mentioned in the assignment.

3.1.6. Part 1.Construction of Tree

Contained in the **construct_tree** method.This function takes in a dataset, makes it generate random splits,and then constructs a tree on the training set.

3.1.7. Part 2.Calculating accuracy

Contained in the **compute_accuracy** method.This function takes in a dataset and constructs 10 different trees by making 10 different 80/20 splits.It then returns the one which has the highest accuracy.It also returns the average accuracy over the 10 splits, the best accuracy among the 10 splits,the validation_set corresponding to the best split.

3.1.8. Part 3.Finding the best possible depth limit

Contained in the **get_depth_limit** method. This function iterates over the maximum height that can be attained by a node and then gets predictions from the model based on the height parameter.After evaluation, it returns the most feasible height and accuracy.

3.1.9. Part 4.Pruning operation to avoid overfitting

Here we will be using the reduced error pruning method.It is basically a post pruning method,while the accuracy for validation set doesnt decrease the function checks accuracy after removing each non leaf node.After that it removes the one that improves accuracy the most.

3.1.10. Part 5.Printing the final decision tree

This function prints the decision tree using the graphviz library.It basically does a bfs on the graph and then appends the nodes one by one.

3.2. Detailed Data Analysis

In this section, we have shown the accuracy of all the models on various kinds of metrics have been provided:

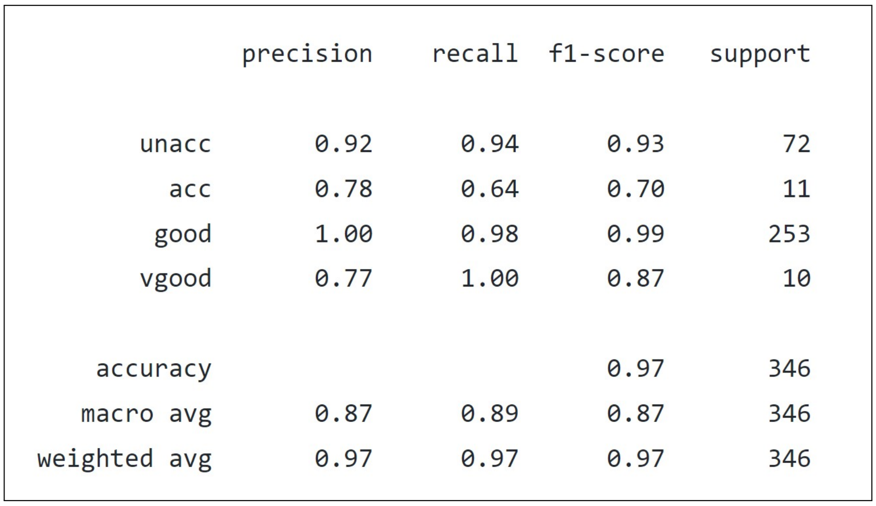
1) **Classification reports of different models:**

    a) Unpruned decision tree implementation using information gain from ID3 entropy. Following were the accuracies observed for 10 runs without using sklearn.

Unpruned Decision Tree using Information Gain	
No of iterations	Accuracy(%)
1	95.66
2	93.93
3	94.21
4	95.08
5	94.21
6	95.08
7	95.08
8	93.93
9	93.06
10	95.08

The average of the above accuracies is 94.53% and the best accuracy observed is 95.66%

Following was the classification report using sklearn



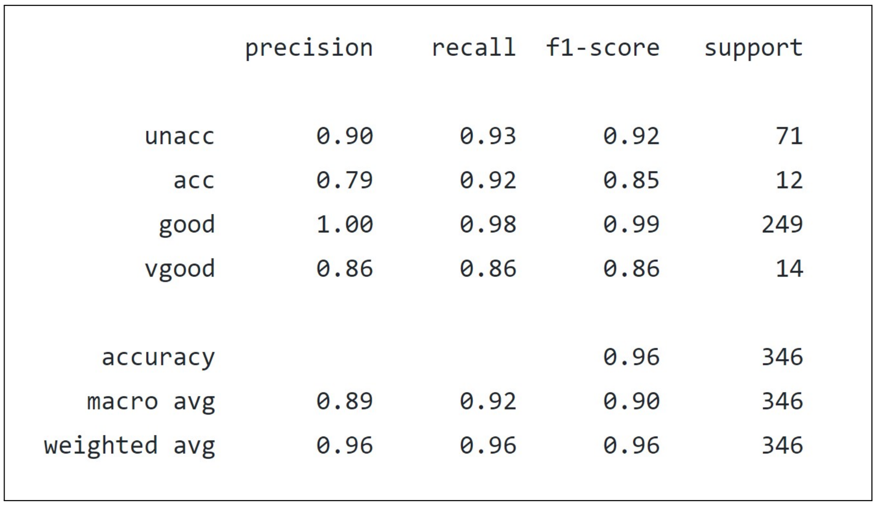
```
              precision    recall  f1-score   support

       unacc       0.92      0.94      0.93        72
         acc       0.78      0.64      0.70        11
        good       1.00      0.98      0.99       253
       vgood       0.77      1.00      0.87        10

    accuracy                           0.97       346
   macro avg       0.87      0.89      0.87       346
weighted avg       0.97      0.97      0.97       346
```

b) Unpruned decision tree implementation using information gain from GINI index
Following were the accuracies observed for 10 runs without using sklearn

Unpruned Decision Tree using GINI Index	
No of iterations	Accuracy(%)
1	96.24
2	94.21
3	91.32
4	94.50
5	94.79
6	95.37
7	93.35
8	94.50
9	94.50
10	93.64

The average of the above accuracies is 94.24% and the best accuracy observed is 96.24%

Following was the classification report using sklearn



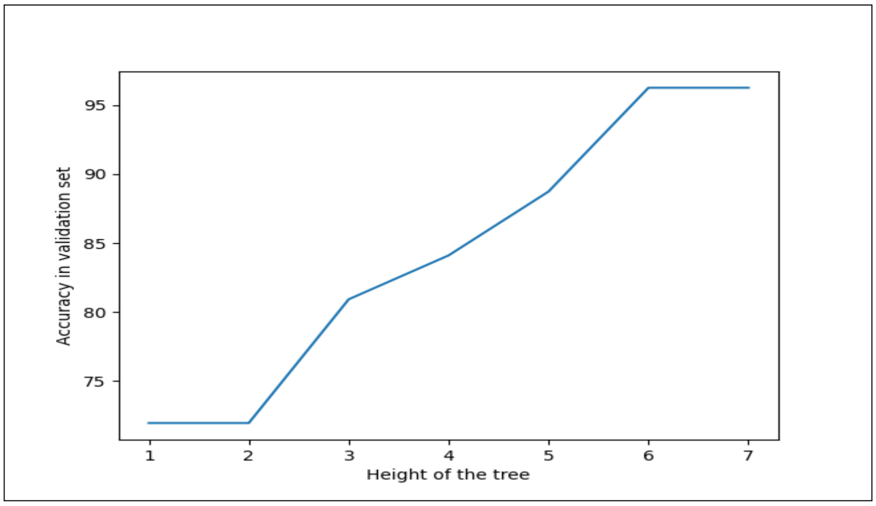
```
              precision    recall  f1-score   support

       unacc       0.90      0.93      0.92        71
         acc       0.79      0.92      0.85        12
        good       1.00      0.98      0.99       249
       vgood       0.86      0.86      0.86        14

    accuracy                           0.96       346
   macro avg       0.89      0.92      0.90       346
weighted avg       0.96      0.96      0.96       346
```

Hence the decision tree obtained from calculation obtained from **information gain as the impurity measure** has a better accuracy of **94.53%**.

Following are the analyzed difference between Information Gain and GINI index.

i) Gini index favours larger partitions (distributions) and is very easy to implement whereas information gain supports smaller partitions (distributions) with various distinct values, i.e there is a need to perform an experiment with data and splitting criterion.
ii) While working on categorical data variables, gini index gives results either in “success” or “failure” and performs binary splitting only, in contrast to this, information gain measures the entropy differences before and after splitting and depicts the impurity in class variables.

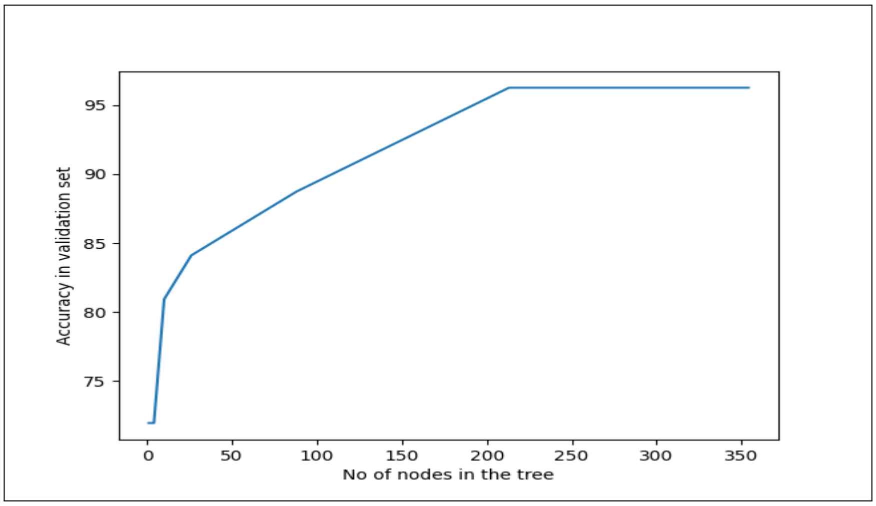
c) **Height v/s accuracy plot**



The best depth limit here is 6 with an accuracy of 95.66%

As the depth of the decision tree increases from 6 there is a slight decrease in accuracy due to overfitting and hence there exists a local maxima at 6 and not 7.

d) **No of nodes v/s accuracy plot**



The best number of nodes here is 220 with an accuracy of 95.66%

As we can observe from the plot the accuracy of the decision tree increases on increase in the number of nodes.This happens till the number of nodes is 220 after that it stabilizes with an accuracy of 95.6%.

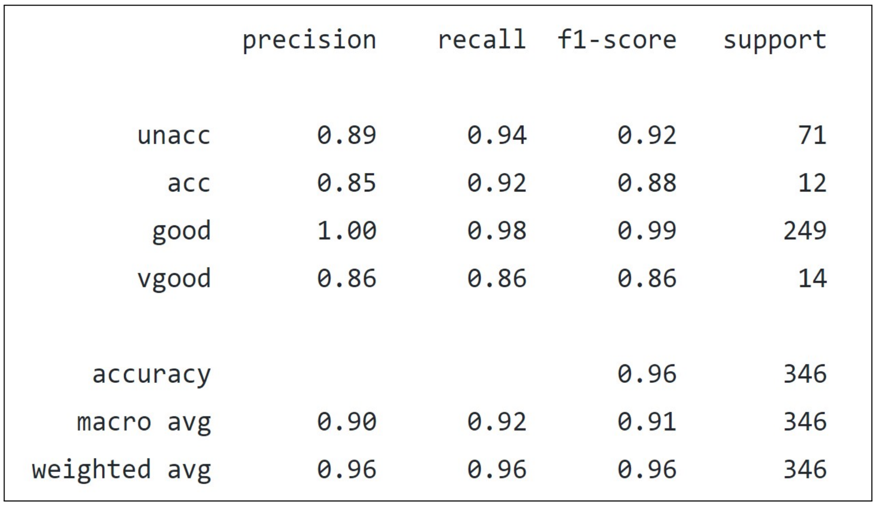
e) Pruned decision tree

Nodes removed while Pruning	
No of iterations	Node Number(%)
1	5
2	336
3	332
4	327
5	324
6	318
7	314
8	307
9	306
10	299
11	294
12	293
13	289
14	284
15	283
16	277
17	276
18	269
19	268
20	261
21	260
22	253
23	252
24	247
25	246
26	238
27	237
28	232
29	231
30	222
31	214
32	207
33	198
34	196
35	190
36	185
37	188
38	168
39	163
40	144
41	139
42	133
43	128
44	118
45	117
46	112
47	110
48	104

Nodes removed while Pruning	
No of iterations	Node Number(%)
49	95
50	93
51	88
52	78
53	71
54	69
55	62
56	53
57	46
58	38
59	25
60	18
61	16
62	10
63	8

63 nodes were removed from the decision tree.The accuracy obtained after pruning the decision tree with information gain as impurity measure is 95.95% which is better than accuracy without pruning 95.66%

Following was the classification report using sklearn after pruning.



```
              precision    recall  f1-score   support

       unacc       0.89      0.94      0.92        71
         acc       0.85      0.92      0.88        12
        good       1.00      0.98      0.99       249
       vgood       0.86      0.86      0.86        14

    accuracy                           0.96       346
   macro avg       0.90      0.92      0.91       346
weighted avg       0.96      0.96      0.96       346
```

3.3. Steps to Run The Code

- Download the code into your local machine.
- Ensure all the necessary dependencies with required version and latest version of Python3 are available (verify with requirements.txt) **pip3 install -r requirements.txt**
- Go into the src directory
- Run the solve.py python code using the following command: **python3 solve.py**

4. Conclusions

After doing this assignment we understand that significant advantage of a decision tree is that it forces the consideration of all possible outcomes of a decision and traces each path to a conclusion.It is often observed that decision trees are very catchy to understand because of their visual representation/interpretation. They can handle the pool of quality data that can be validated by statistical techniques and are cost-effective computationally.It can also handle high dimensional data with actual good accuracy. Besides that, various features selection methods are used in building the decision tree from root nodes to leaf nodes.This makes them a must have tool in the toolkit of every machine learning enthusiast or professional.

Acknowledgements

We are very grateful to Professor Jayanta Mukhopadhyay for giving us an opportunity to learn and apply the topics we have studied on some real-life problem. His guidance, support and teaching helped us a lot for making the skeleton of the project.