Machine Learning (CS60050)

Assignment-1

Parth Jindal, Pranav Rajput

Group : 59

Abstract

This report contains an analysis for a decision tree classifier trained on the **UCI Car dataset**.The following tasks are covered in the report:

1. Build a decision tree classifier using gini-index and information gain as the impurity measures
2. Find the average accuracy over 10 random 80/20 splits
3. Vary the depth of the decision tree and plot accuracy vs depth graph
4. Prune the decision tree using cross validation
5. Print the decision tree

Data Analysis

The UCI car dataset is a categorical dataset classifying a car into 4 categories **[‘unacc’, ‘acc’, ‘good’, ‘v-good’]** The frequency table for each class is given below:

class	N	%
unacc	1210	70.023 %
acc	384	22.222 %
good	69	3.992 %
v-good	65	3.762 %

Each car variable has 6 attributes which are:

1. buying cost (buying)
2. maintenance cost (maint)
3. number of doors (doors)
4. number of seats (persons)
5. boot space (lug_boot)
6. relative safety of passengers in the vehicle (safety).

The data was randomly splitted in 80:20 ratio and the tree was built thereafter. The ID3 algorithm was used to form the decision tree. The data is the evaluation of cars based on certain attributes.

Comparison of impurity measures

Using Gini-index and Information gain as the two impurity measures, various statistical analysis results were produced on a random splitting of the dataset as 80/20 split.
Note: 10 random splits were used to find the mean accuracies and confidence intervals for both the criteria.

Gini Index: Gini index measures the expected value of error under a class distribution

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

Gini Gain: It measures the change in Gini index under the new distribution of classification derived upon an attribute variable

$$Gain(S, A) = Gini(S) - \sum_{a \in A} P(a) \cdot Gini(S \mid a)$$

Mean accuracy	95% C.I for accuracy
91.33 %	86.24%-96.42%

The following table contains metrics such as **'Precision', 'Recall', 'F1'** for each class label for a decision tree trained using Gini-Gain as the impurity measure

Class	Precision	Recall	F1
unacc	0.956	0.984	0.970
acc	0.947	0.826	0.882
vgood	0.778	0.778	0.778
good	0.545	0.857	0.667

The confusion matrix for the decision tree trained on using gini-gain is as follows
The columns indicate the predicted label by the classifier whereas the rows are the true labels in the test dataset

True/Prediction	unacc	acc	vgood	good
unacc	240	11	0	0
acc	4	71	0	0
vgood	0	1	7	1
good	0	3	2	6

Entropy: Entropy measures the average no. of bits required to encode a distribution of a random variable X

$$E(X) = -\sum_{x \in X} P(x) \cdot \log_2 (x)$$

Information Gain: Information gain measures the gain in entropy of the class label distribution condition on the attribute value under consideration

$$T|(X, A) = E(X) - \sum_{a \in A} P(a) \cdot E(X \mid a)$$

Mean accuracy	**95% C.I for accuracy**
92.31 %	89.86%-94.76%

The following table contains metrics such as **'Precision', 'Recall', 'F1'** for each class label for a decision tree trained using Information gain as the impurity measure

Class	**Precision**	**Recall**	**F1**
unacc	0.956	0.984	0.970
acc	0.947	0.835	0.887
vgood	0.778	0.778	0.778
good	0.636	0.875	0.737

The confusion matrix for the decision tree trained on using gini-gain is as follows
The columns indicate the predicted label by the classifier whereas the rows are the true labels in the test dataset

True/Prediction	unacc	acc	vgood	good
unacc	240	11	0	0
acc	4	71	0	0
vgood	0	1	7	1
good	0	2	2	7

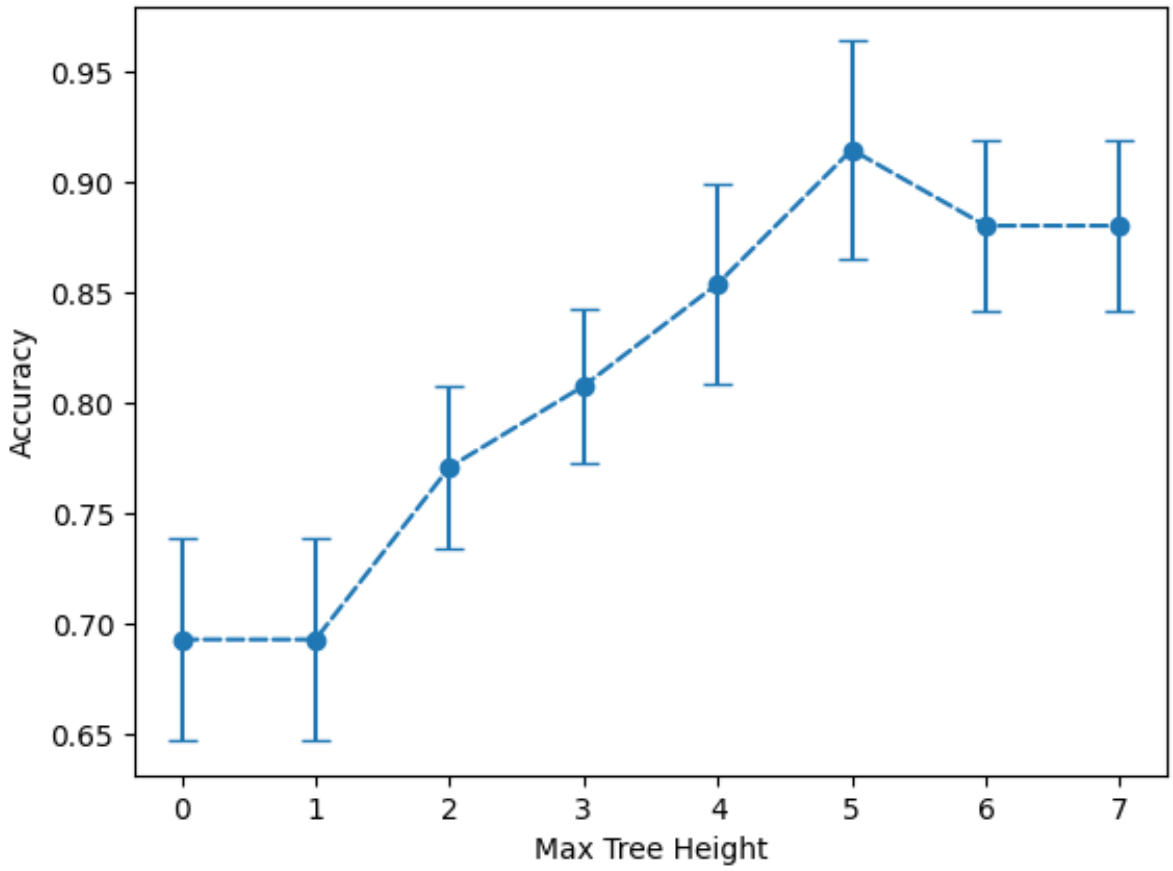
Observations: Both information-gain and gini-gain have similar accuracies and other metric evaluations. Information gain produces a tighter bound on the mean accuracy achieved as perceived from the 95% C.I interval whereas Gini-gain might produce a higher accuracy in some cases over information-gain but suffers from high variance under this dataset

Using 10 Random splits for evaluating a Decision-Tree

Mean accuracy	95% C.I for accuracy
92.31 %	89.86%-94.76%

Note: All trees built under this experiment had a **max-height limit of 5** and **minimum-samples per leaf as 1** with the impurity measure as **Information gain**

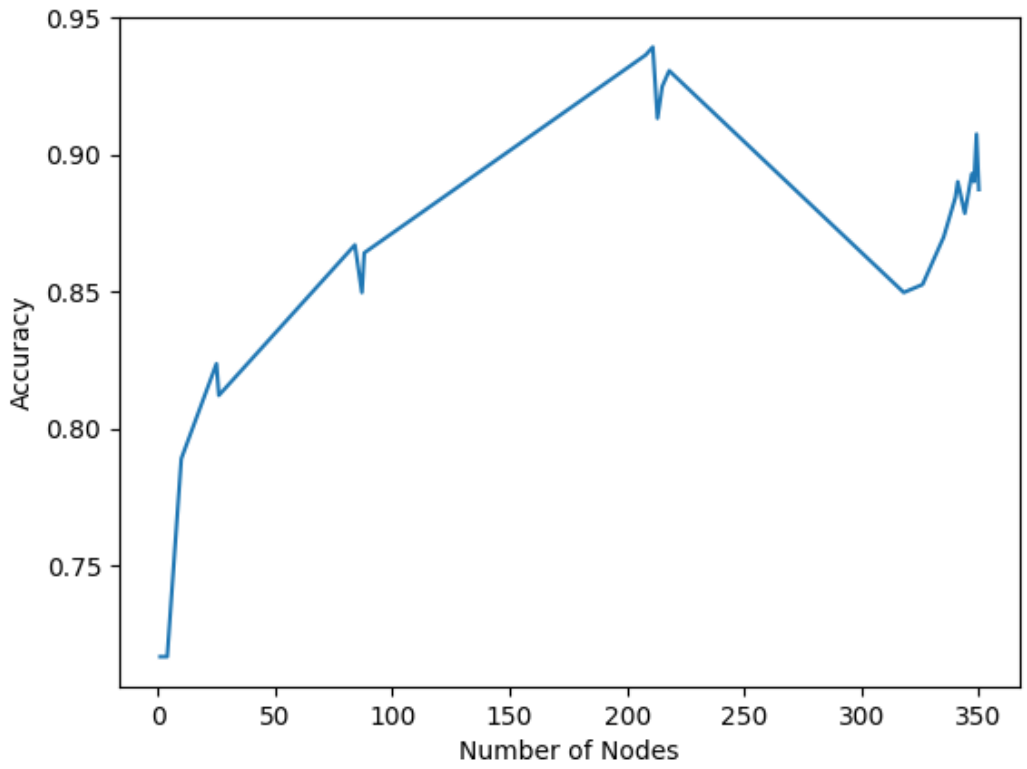
Plotting Height vs Accuracy



For comparing heights, **10 random splits** of the dataset were done in 80/20 splits and the test accuracies evaluated. Here the error bars show the **95% C.I** on the test accuracy

As seen from the graph, the best possible height is at **max-height = 5** for which the accuracy ranges between **87%-96%**

Plotting Number of nodes vs Accuracy



The above graph shows how accuracy changes with the number of nodes in a graph. The data is collected by varying the height of a tree from 0 to 7 and measuring the number of nodes in 10 random runs for each height-tree.

We can clearly see the direct correspondence of the height and number of nodes along with the accuracy in the above two graphs.

Pruning the decision tree

The decision tree is pruned using two different methods:

1. Chi-square test as the statistical test
2. Reduced error pruning

Chi-square Test: This test is used to check the validity of the hypothesis under consideration with respect to the null-hypothesis i.e pruning the deduced rules. The chi-square test is performed under the 95 % confidence that the hypothesis is not a null-hypothesis.

$$P(H \neq Null - Hypothesis) > 0.95$$

Pruning-algorithm:

Construct the entire tree as before and store the frequency array of class labels under each node
Starting at the leaf, eliminate the splits recursively:

- At each leaf node N
- Compute the chi value for N and its parent P along with all other children of P
- Make P a leaf
- Recursively until completion

Results:

$$\chi_c^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

Status	Number of nodes	Accuracy
Before pruning	213	93.93 %
After pruning	146	94.04 %

Reduced error pruning:

In this method, a validation set is used to evaluate the accuracy of a decision tree at a node with and without its children. If the accuracy increase on pruning the children, the pruning is made permanent and the process continues

Pruning-algorithm:

Construct the entire tree as before
Starting at the leaf node eliminate the splits recursively:
- At each parent node P temporarily prune the children and calculate the test accuracy on the validation set
- If prune-accuracy > non-prune-accuracy:
    - Make the pruning permanent, continue
- else:
    - Return

Results:

Status	Number of nodes	Accuracy
Before pruning	213	93.93 %
After pruning	126	94.80 %

Note: Even though reduced error pruning is producing a higher accuracy over chi-square testing, Reduced error pruning is prone to suffer from injecting validation-set bias into the decision tree unlike chi-square testing.

Printing the best decision trees obtained in Q2.

The image obtained was very large in size and hence has not been embedded in the document. The zip file contains the image required under the name '**tree.gv.png**'.

Procedure and Implementation details

The following packages have been used in building the decision tree:

Numpy: for reading and writing arrays

Pandas: for reading csv dataset files

Matplotlib: for creating graphs and plotting

Major Classes:

DecisionTree

This is the main class to build the decision tree with the following arguments:

max-height: maximum height for the decision tree

min-samples: minimum samples for splitting a node further

criterion: The impurity measure used

Functions:

fit : this function fits the decision tree model to the input dataset. Present in class *DecisionTree*

build_tree : It is a private method of the class decision_tree which uses the other helper functions (get_leaf, get_best_split etc.) to construct the decision tree. It is a recursive function, which for every node at a certain height, chooses an attribute which is then used to split the node in children nodes. The attribute chosen is finalised on the basis of information gain and the comparison of gini gain values

get_best_split : for a certain node at a fixed height, this function decides the best possible split by considering the gini gain or information gain of all attributes. Present in class *DecisionTree*

print_tree : Prints the decision tree formed by our program. Makes use of the graphviz package for the visual presentation of the decision tree. It is a recursive function and stores the tree in '.gv' (graphviz) format. Present in class *DecisionTree*

prune_tree : Given the decision tree built by build_tree, this function performs the pruning operation on the tree. It uses the chi-square or reduced-error method for pruning the decision tree. Present in class *DecisionTree*

predict : this function predicts the output upon training the provided decision tree (with root) and training data. It returns the predicted values. Present in class *DecisionTree*

CarDataset

Class to download and load the dataset from a given directory. Contains the functions **sanity_check**, **download**, and **read_data**.

Major functions:

compare_heights : function to plot height vs accuracy with error bars and to plot number of nodes v accuracy.
average_runs : function to run all the functions to form the decision tree from the dataset and number of runs given as input. Returns the best decision tree formed and the result.

compare_criteria : wrapper function to compare the different criteria given for the decision tree.

Helper functions:

information_gain : function to calculate the information gain for an attribute and target labels given as input. It is used by the **get_best_split** function.

gini_gain : function to calculate the gini gain for an attribute and target labels given as input. It is used by the **get_best_split** function.

entropy : function to calculate the entropy for the target label given as input. It is used by the **information_gain** function.

gini_index : function to compute gini index for the target label given as input. It is used by the **gini_gain** function.

split_dataset : function to randomly split the given dataset in 80:20 ratio for training : validation. Returns the training dataset and validation dataset

shuffle_dataset : function to shuffle the dataset and target labels. Returns the shuffled dataset**.**

precision_score : function to compute the precision score for a set of labels given as input. Returns the precision score.

recall_score : function to compute the recall score for a set of labels given as input. Returns the recall score.

get_metrics : function to return the attribute values for an instance given as input present in the dataset. Returns the attribute values for the instance

confusion_matrix : function to compute the confusion matrix for a set of labels given as input. Returns the confusion matrix.

find_ci_interval : function to find the confidence interval for a given dataset of an estimator. Returns the mid value of the interval and the confidence interval.

f1_score : function to compute the F1 score for a set of labels given as input. Returns the F1 score.

Steps to run the code:

1. Inside the submission directory, use pip install -e requirements.txt to install the dependencies
2. run python main.py --max_height 5 --min_samples 1 --criterion ig --verbose to run all the experiments
3. Since there are lots of experiments and numerous runs for each, the procedure might take 5 minutes to complete