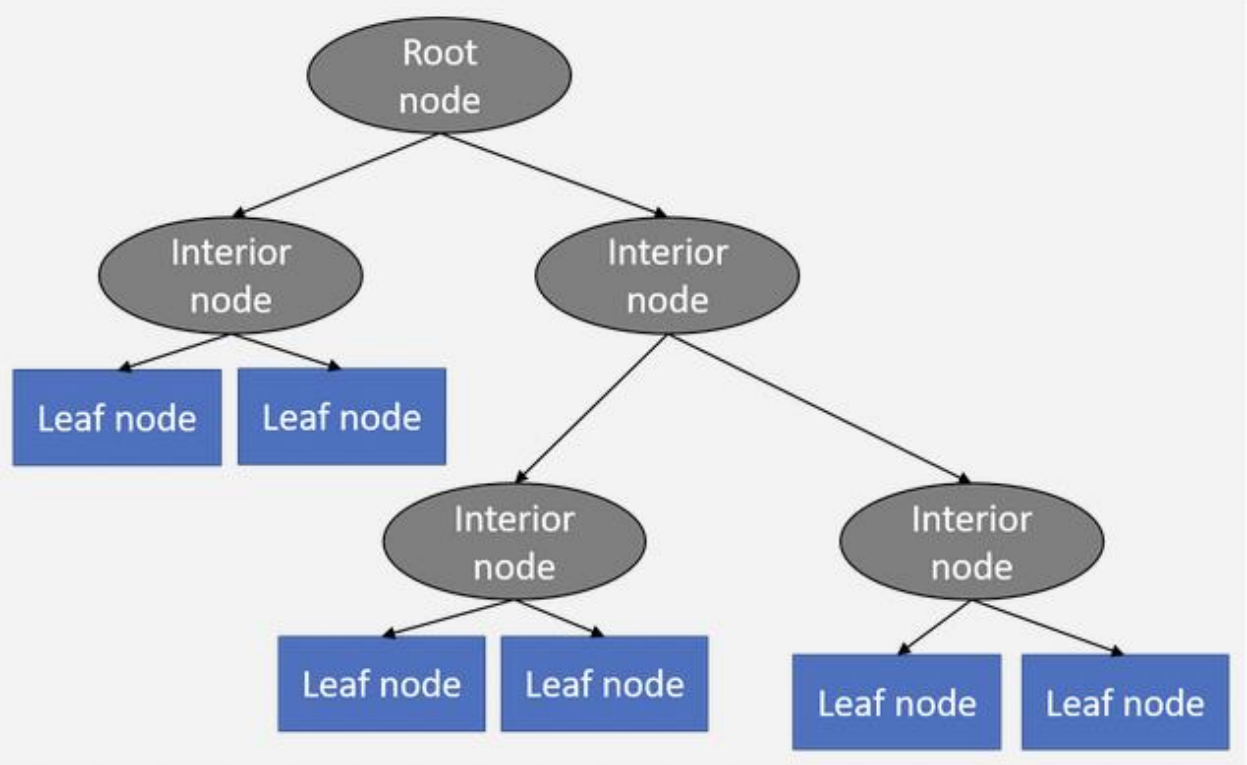
Using Top-Down Greedy Strategy algorithms to build, compare, and apply decision trees to different problem sets.

Project Final Report



[1]

[1] [1] Tobias Schlagenhauf, "31. Decision Trees in Python," 31. Decision Trees in Python | Machine Learning, accessed December 13, 2022, https://python-course.eu/machine-learning/decision-trees-in-python.php.

INTRODUCTION

Project Context:

After a few weeks of thought process, our team members gathered a wide range of intriguing subjects for our algorithm's final project. During the discussion, we talked about each of the topics, and in the end, we agreed that a core understanding we wanted to leave with from this course was recursion, trees, graph traversals, and greedy approaches. A natural fit that falls right within this domain includes Decision Trees. Therefore, we will take a deep dive into what a decision tree is, different algorithms commonly used, and their applications.

1. Karan Shah: My background is Data Science at Carnegie Mellon University. I worked in industry as a Data Scientist as well, and in order to take many machine learning algorithms from conception and deploy them in end-to-end production with MLOPs requires having a core understanding of algorithms (aside from knowledge needed in machine learning theory/application, distributed systems, API's, containerization, cloud vs On-Prem architecture, etc.). Algorithms will be a solid stepping stone for more advanced areas I plan to go into which may include Advanced Algorithms, Robotics, Comp. Vision, etc. Therefore, in this project, I will supervise the rest of the group with my background knowledge in this domain along with the material from the Machine Learning course I am currently taking at NEU. I will provide the mathematical foundations needed to understand CART/ID3 (precursor of C4.5 algorithm) and attempt to bridge the gap between the math and implementation.

2. Ameya Santosh Gidh: I have a background in mechanical engineering. I love the subject of algorithms and how it is used to solve the daily tasks. Finding the appropriate data structures and solving complex problems efficiently is what I have learnt from this project. I am interested in probabilities and statistics and on how the kind of algorithms I learnt in this course, I can use to solve real life problems. One of the fundamental work of a computer is to make decisions and thus learning one of the fundamental algorithm of decision trees is important for us to understand how the trees we studied in our algorithm class can help us in real life situations. For me I love to analyze data.

3. Mattia Contestabile: My previous studies were concentrated on Philosophy and Political Economy, and my interests for algorithms that optimizes a wide range of tasks has grown naturally during this algorithms class. The core understanding how we choose the best set of actions has real implications not only on our individual life but as also in our collective lives. Using graph traversal algorithms such as Breadth First Search, Dijkstra's algorithm, Kruskal's Minimum Spanning Tree provides us powerful tools to explore solutions that can optimize real life scenarios like increasing delivery efficiency, finding the shortest greedy algorithms for allocating just enough resources to achieve that. My aim is to be able to delve into the decision tree application by understanding how to build a CART algorithm that is required for building a decision tree. This is not only for the sake of this project but also for my interest decision analysis. At the same time, I aim to work with the team knowing that we must be time sensitive due to the ongoing homework deadlines for CS5800 and other courses coupled with the upcoming synthesis.

4. Jose Lou: I have a degree in Business Management (with a focus on finance) and MIS. From the time that I graduated back in 2007, I have held many different posts and wore many different hats in various industries. I had worked as a business developer, financial planning and analysis, and risk analysis for multinational corporations and banks. When I finally made enough money and had an opportunity to create my own business, I did so at the age of 25. Two years after business has stabilized, I felt like I still could do more with my time and went to Med School while running my business. Now, I am yet again embarking on a new academic journey, which could be my last stint in academia as I am not getting any younger and the older you get, the more responsibilities and obligations life thrusts at you. This time in the field of technology, Computer Science to be more precise. With my background in various fields, I have gleaned a unique understanding and perspective of their unique characteristics. Thus, I believe that I am in a unique position to use these various angles to provide unconventional methods and solutions to problems and hopefully use and leverage the newfound skills I am gaining from this degree and tie it in to my previous work and life experiences. Algorithms are essential to CS because they provide us with various techniques or approaches to solving a specific problem in the most efficient time. This is very important to aspiring software engineers or computer scientists as we develop software that is powering the modern world, and research ways to further improve upon existing knowledge. I am very interested in working and developing AI products after graduation. Novel things from self-driving cars, autonomous robots, and AI assistants. I feel like automation is our future, and I want to be part or a catalyst to that change. The topic my group wanted to investigate is Dijkstra and Kruskal's algorithm which I agreed with because it fit in with my goal. These algorithms are one of the best and most used when it comes to finding the Shortest

Path. The concept is simplistic yet very powerful and has a lot of practical applications in the real world.

TOP- DOWN GREEDY APPROACH

What is greedy approach?

The greedy approach is designed to either maximize or minimize the heuristic function.
In doing so, it behaves by selecting the current best option at every step of the function.
However, this does not mean it will always lead to an optimal solution. It is important to mention that this greedy algorithm only moves in one direction. What it does mean is that it does not retract to its previous position.

When it comes to analyzing a top-down approach, we go from a more general case to a specific case. In a top-down fashion, we build a tree starting from a root node, and then we expand the tree at every single level. The criterion for expanding the tree is from choosing what is best at the current instance.

In this case, we will essentially be building binary decision trees but that is not always the case. Furthermore, as aforementioned above, we will investigate what a decision tree truly is composed of, how these algorithms work, their benefits, and their drawbacks. Aside from trying to answer these questions, our aim also is to explore possible direct applications of decision trees on real world datasets. We originally thought to use politician's/education's dataset, but we ended up changing our dataset to a car evaluation dataset to better fit our approach. While we evaluate these techniques on the dataset, we will consider whether one was better or worse ultimately and suggest other alternatives.

Deep Analysis

The time complexity of decision tree is O(depth) = O(nlogn*d) + O(n*d) which is asymptotically O(nlogn*d). The space complexity is O(nlogn*d).

The space complexity is O(n) where n is the number of nodes. In worst case, all the nodes are visited, and we visit all the nodes.

WHAT IS A DECISION TREE?

To understand what a decision tree is, it is first imperative to understand its significance in Machine Learning (ML). ML itself is divided in 3 parts:

For the scope of this project, we are using the supervised technique for building our decision tree model. A metaphor for 'building' decision tree is like creating a tree like structure, but just using conditionals. In this effort, the algorithm will try to split each node based on some feature and criterion. Decision trees are made of individual internal nodes, and once built through training, they can be used to make a prediction on a new sample instance. While we are using them in the classification setting, they can be used in the regression setting as well. Reason for why we are using decision trees in this paper is to understand about algorithms we learned in this course, and how we can then apply them to classify the car evaluation data set problem available at UCI Machine Learning Repository[2].

[2] Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

Components of a Decision Tree

To build a decision tree, we first elaborate about the active components that make a tree. These components are:

1. Root Node: It is the beginning of the splitting process. We can use a certain criterion to understand which variable target is best to split on which is a theme persistent throughout greedy approaches.
2. Node Purity: Most of the nodes can be classified either as *pure* or *impure*. Purity is defined as the measure of homogeneity of the labels at the node in hand. Specifically, the distinction between the two is given by the fact that *pure* nodes are made of only one label. On the other hand, *impure* nodes are made of more than one node. Ultimately, we want to partition the data by maximizing the purity of the split or minimize the impurity.
3. Decision Nodes: These are intermediate nodes in which feature to use to divide the feature space.
4. Leaf Node: This is the terminal node. Used during the prediction part of machine learning to define the outcome for some prediction sample.[3]

[3] Shailey Dash, "Decision Trees Explained-Entropy, Information Gain, Gini Index, CCP Pruning..," Medium (Towards Data Science, November 2, 2022), https://towardsdatascience.com/decision-trees-explained-entropy-information-gain-gini-index-ccp-pruning-4d78070db36c.

WHAT DID YOU DO?

In this project, we created a tree classifier on a car evaluation dataset for which we created two commonly used algorithms which are namely **CART** and **ID3** algorithm. We compared the results with the inbuilt sklearn library. Both these algorithms use different criteria for printing the node, which is *Entropy* for ID3 and *GINI Index* for CART. Entropy measures the randomness of the dataset. Once Entropy is calculated for each feature, we then use it to calculate the *Information Gain provided by each feature. T*hus, the greater value of information gain for a certain feature, the better. The car dataset is made the target variable which is a binary feature taking values such as unacceptable denoted as *unacc, and* acceptable denoted as *acc.* Other features in our dataset include the feature of purchasing a car. This includes the price of the car, which is a combination of the *maintenance* price and the *buying* price. Also, comfort features are included such as the size of the luggage boot, denoted *lug_boot.* The number of doors, and the number of people that can be carried along with the safety of the car is included. The dataset is obtained from UCI Open Source, and it is generated by using data collected from various sources. To make this dataset, pre-dominant features which influence the user decision in buying a car were analyzed on historical data.

WHY DID WE DO IT?

A lot was taught in the algorithm class, and one area we wanted to focus on was recursion, trees, and greedy. We have currently used Binary trees for our decision tree problem. A binary tree is "composed of a parent node and at most two children"[4] and its time complexity would be given by its height. Additionally, we wanted to explore more of the greedy strategy. During the class, we learned a lot of the greedy algorithms such as Prim, Huffman, Dijkstra, Kruskal's algorithm, etc. We wanted to go deeper into this, and thus we decided to explore this topic. On top of that, we learned about recursion, which has an internal stack like data structure. Now in our algorithm we are traversing through the tree here, and the same operation is performed throughout. So, we need a base condition for when the max depth is reached, or other things like the node is no completely *pure*

[4] Mohd Ehtesham Uddin Qureshi, "Time & Space Complexity of Binary Tree Operations," OpenGenus IQ: Computing Expertise & Legacy (OpenGenus IQ: Computing Expertise & Legacy, December 8, 2021), https://iq.opengenus.org/time-complexity-of-binary-tree/.

ID3 (Iterative Dichotomiser 3)

The hypothesis space of all decision spaces is a complete space of finite discrete-valued functions, relative to the available attributes. In addition, the inductive bias ID3 is that it prefers *short trees,* more precisely ID3 elects short trees that have 'high information gain attributes near the root'.[5] When dealing with ID3, it is important to familiarize yourself with the concept of *Entropy which* is used for measuring the data sample's impurity. For statistical measures, ID3 will use *Entropy* throughout. A node is pure when the entropy value takes its minimum value to 0 and is impure when it takes the highest value 1. Meanwhile, Information gainis used to understand and classify which of the feature supplies the most amount of information[6] by always referring to the concept of purity given by *Entropy*. For the time complexity of ID3, the determining "factors are given by the *d* attributes and *m* training instances." Since the depth of the ID3 is O(log m) we can evaluate that the Time Complexity for ID3 is O(d m log m)[7]

C4.5

[5] Tom Michael Mitchell, "Chapter 3," in *Machine Learning* (New York: McGraw-Hill, 1997).
[6] Neelam Tyagi, "Understanding the Gini Index and Information Gain in Decision Trees," Medium (Analytics Steps, September 30, 2020), https://medium.com/analytics-steps/understanding-the-gini-index-and-information-gain-in-decision-trees-ab4720518ba8.
[7] Tom Michael Mitchell, "Chapter 3," in *Machine Learning* (New York: McGraw-Hill, 1997).

As mentioned in the above paragraph, C4_5 is an extension of the ID3 algorithm. But differently than ID3, C4_5 does not need to have its input dataset be discrete and it can handle both continous (numerical) and discrete features.

CART

Next, CART can be found to follow a similar structure to C4_5. However, the difference that distinguishes these two algorithms can be found in the "splitting creation for selection attribute."[8] As mentioned, CART uses GINI index as a reference for the splits it will execute. Its time complexity is calculated also as $O(mn \log_2 n)$ "where m refers to the attributes and n to the observations."[9]

IMPLEMENTATION

In this project we primarily used the following data structure:

1. Node class
2. Arrays to calculate and store our entropy/gini index values

For implementation, we imported necessary libraries such as Pandas, NumPy, MATPLOTLIB, and SKLEARN).

[8] Max A. Bramer, *Artificial Intelligence XXXV: 38th SGAI International Conference on Artificial Intelligence, AI 2018, Cambridge, UK, December 11-13, 2018: Proceedings* (Cham: Springer, 2018).
[9] Max A. Bramer, *Artificial Intelligence XXXV: 38th SGAI International Conference on Artificial Intelligence, AI 2018, Cambridge, UK, December 11-13, 2018: Proceedings* (Cham: Springer, 2018).

To use the data for training, we first operate a pre-processing step, where the car dataset has been completely *binarized including the features*. The given features were transformed accordingly:

1. Buying -> Med to low & VHIGH -> HIGH
2. Maint -> Med to low & VHIGH -> HIGH
3. Doors -> All cars -> 2 Doors one bin, and cars with 2 doors is in the other bin.
4. People's car can hold half of 4 persons goes into 2 persons and other half into more persons.
5. lug_boot: Half of med goes into small and other half of med into big.
6. Safety: Half of medium goes into low and other half into high.
7. Target: Unacceptable vs Acceptable car

```
Out[35]: [low     660
          high    636
          Name: 0, dtype: int64,
          low     648
          high    648
          Name: 1, dtype: int64,
          >2    963
          2     333
          Name: 2, dtype: int64,
          2       655
          more    641
          Name: 3, dtype: int64,
          big      657
          small    639
          Name: 4, dtype: int64,
          low     654
          high    642
          Name: 5, dtype: int64,
          unacc    910
          acc      386
          Name: 6, dtype: int64]
```

MINI STEP

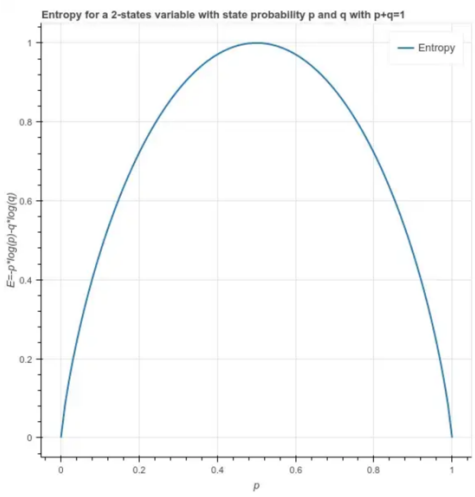
Part A

As per the ID3 algorithm, we split on safety = low. This is the first split, then the second is buying = low since we have 59 upon 272 as the ratio of acceptability to unacceptability. Then we split doors when the value is greater equal to 2,(*doors => 2*), since we have 48 upon 200 as the ratio of acceptability to unacceptability. After doing that, we proceed with splitting buying when *buying = high* since we have 24 upon 299 as the ratio of acceptability to unacceptability. We proceed with splitting lug_boot when *lug_boot = small* since we have 5 upon 173 as the ratio of acceptability to unacceptability. We continue with our split, when *lug_boot = big* since 19 upon 126 as the ratio of acceptability to unacceptability. Then we split

safety when safety = high since we have 303 upon 339 as the ratio of acceptability to unacceptability. Then we split persons when *persons = more, since* we have 239 upon 111 as the ratio of acceptability to unacceptability. Moving on, then we split buying when *buying = low* since we have 160 upon 20 as the ratio of acceptability to unacceptability. Then we continue splitting, when *buying = high* since we have 79 upon 91 as the ratio of acceptability to unacceptability. Then we split persons when *persons =2*, since we have 64 upon 228 as the ratio of acceptability to unacceptability. And finally, we split lug_boot *when lug_boot = small* since we have 45 upon 103 as the ratio of acceptability to unacceptability.

Part B

Entropy and Gini Index Calculation



When using Decision Trees, it is crucial that the nodes are positioned so that entropy reduces with downward splitting. This basically suggests that it gets simpler to reach a firm choice the more appropriately dividing is done.

Entropy is of fundamental importance when calculating Information Gain within ID3. Entropy values range from 0 to 1 while Gini index values range from 0 to 0.5. As we already discussed, the use of *Information Gain* is to "maximize information about classification, based on the notion of entropy"[10]; this is done with the intention of limiting and subtracting the value of entropy that is present from the root node to the leaf's node.

[10] Neelam Tyagi, "Understanding the Gini Index and Information Gain in Decision Trees," Medium (Analytics Steps, September 30, 2020), https://medium.com/analytics-steps/understanding-the-gini-index-and-information-gain-in-decision-trees-ab4720518ba8.

The formula of Entropy is listed below:

$$\text{Entropy} = -\sum_{i=1}^{n} p_i * \text{Log}_2(p_i)$$

For reference, *p* refers to the probability that is a function of entropy"[11]

Comparatively, the Gini Index is "calculated by subtracting the total of each class squared probabilities from 1[12]". As discussed before, the algorithm takes advantage of the Gini Index is CART meanwhile ID3 relies on *Entropy*. The formula of the Gini Index is listed below:

$$\text{Gini Index} = 1 - \sum_{i=1}^{n} (P_i)^2$$

[13]

[11] Neelam Tyagi, "Understanding the Gini Index and Information Gain in Decision Trees," Medium (Analytics Steps, September 30, 2020), https://medium.com/analytics-steps/understanding-the-gini-index-and-information-gain-in-decision-trees-ab4720518ba8.

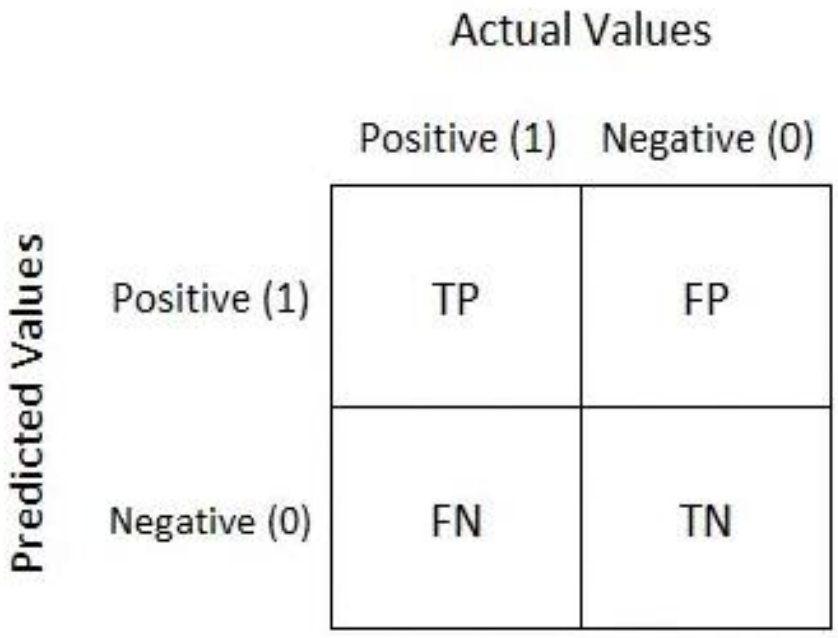
[12] Neelam Tyagi, "Understanding the Gini Index and Information Gain in Decision Trees," Medium (Analytics Steps, September 30, 2020), https://medium.com/analytics-steps/understanding-the-gini-index-and-information-gain-in-decision-trees-ab4720518ba8.

[13] Neelam Tyagi, "Understanding the Gini Index and Information Gain in Decision Trees," Medium (Analytics Steps, September 30, 2020), https://medium.com/analytics-steps/understanding-the-gini-index-and-information-gain-in-decision-trees-ab4720518ba8.

Measuring Performance

Confusion Matrix

$$\begin{bmatrix} 160 & 226 \\ 20 & 890 \end{bmatrix}$$



True Positive:

Interpretation: You predicted positive and it's true.

True Negative:

Interpretation: You predicted negative and it's true.

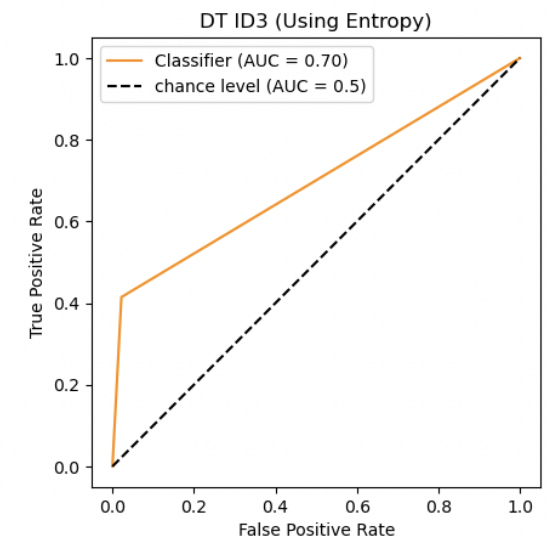
False Positive: (Type 1 Error)

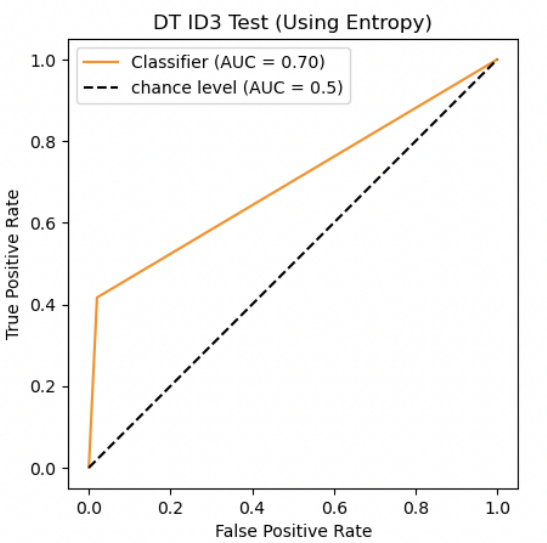
Interpretation: You predicted positive and it's false.

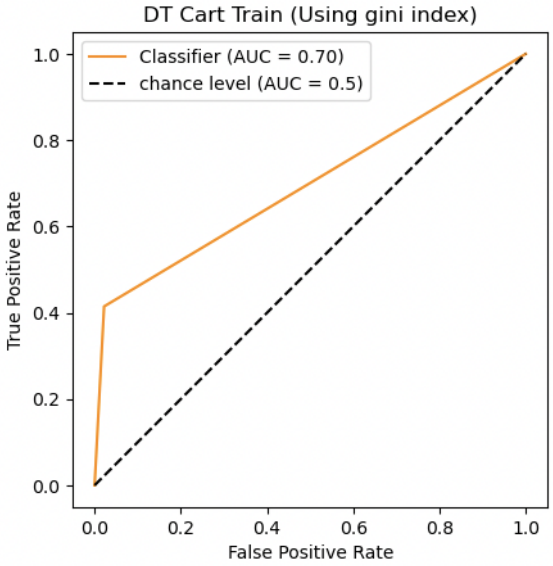
False Negative: (Type 2 Error)

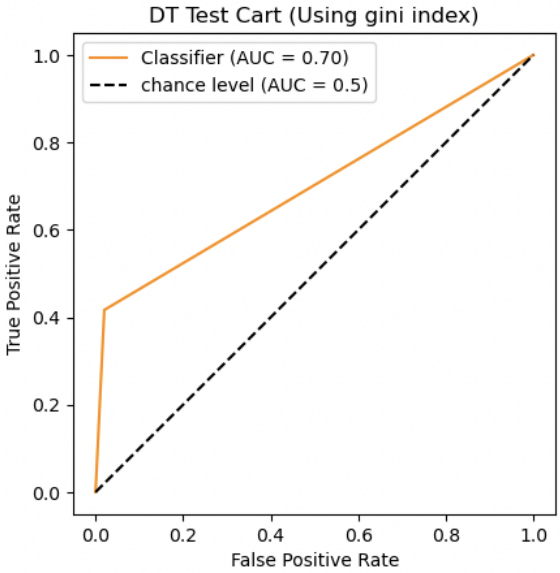
Interpretation: You predicted negative and it's false.[14]

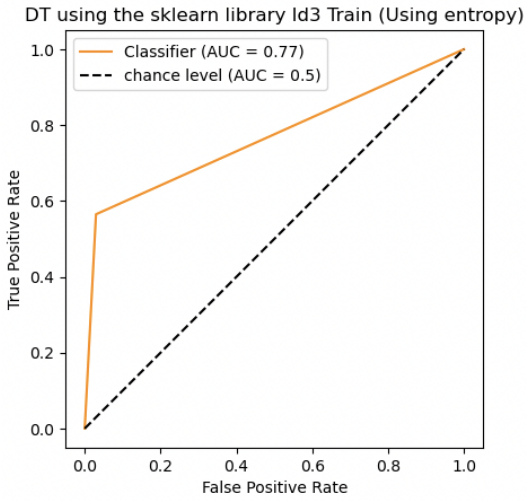
[14] Sarang Narkhede, "Understanding Confusion Matrix," Medium (Towards Data Science, June 15, 2021), https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62.


DT ID3 (Using Entropy)
Classifier (AUC = 0.70)
chance level (AUC = 0.5)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

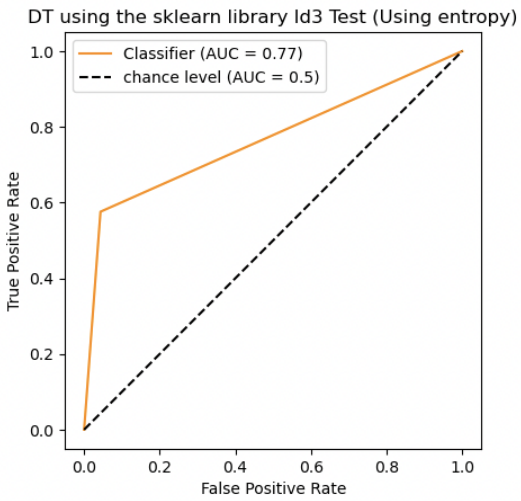

DT ID3 Test (Using Entropy)
Classifier (AUC = 0.70)
chance level (AUC = 0.5)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0


DT Cart Train (Using gini index)
Classifier (AUC = 0.70)
chance level (AUC = 0.5)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

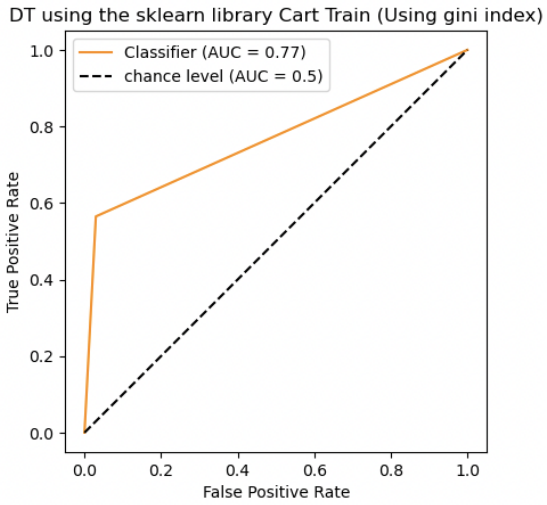

DT Test Cart (Using gini index)
Classifier (AUC = 0.70)
chance level (AUC = 0.5)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0



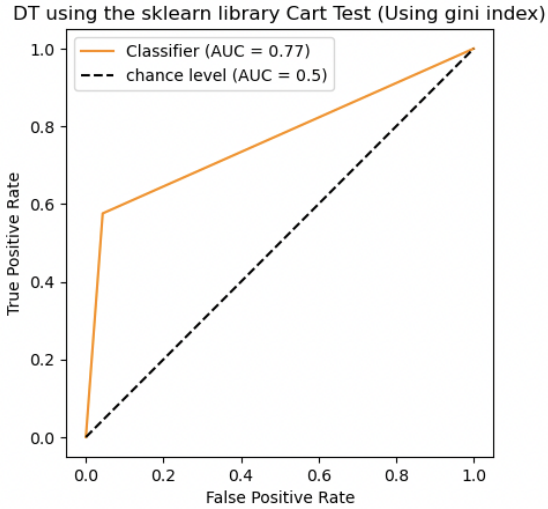
(Decision tree using Sklearn)



(Decision tree using Sklearn)



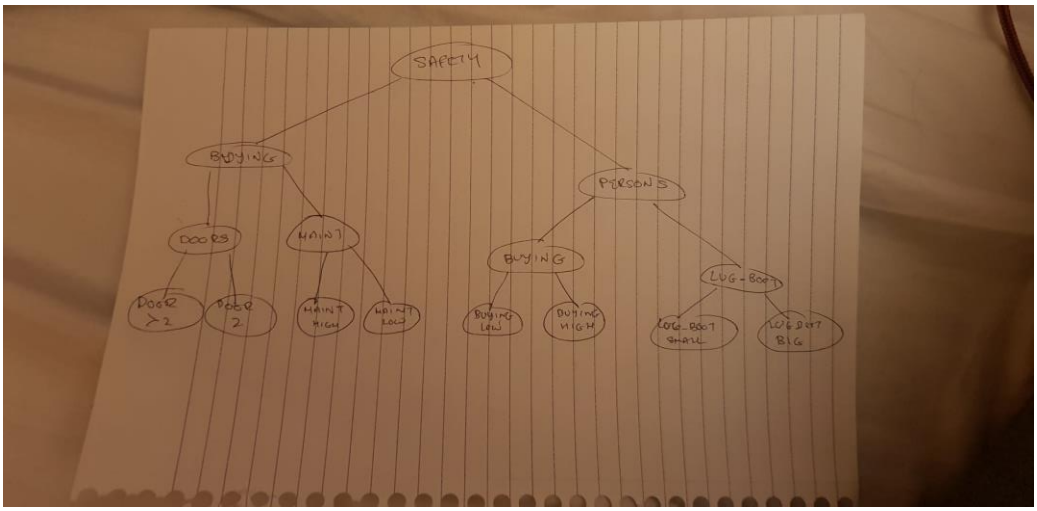
(Decision tree using Sklearn)



(Decision tree using Sklearn)

Algorithm Benchmarks

We observe that even though the different algorithms such as CART and ID3 had different criterion for evaluation they produced the same output. The value of the criterion was different, but the roc curves were almost identical. This is the result of the features having similar number of instances which can be seen by checking the counts of each feature.



The decision tree generated from our algorithm is this one.

Results

Given the data we have finally elaborated, the avenues for future research concentrate on reducing the size of the Decision Tree for more robust results. For achieving that we use pruning, that reduces the size of a Decision Tree and its complexity. Better approaches include using Random Forest algorithm, that uses a combination of many decision trees and depending on the number of votes it marks the right label to choose. In a random forest, we make use of *bagging* which is just

bootstrap aggregation. Then each model is trained on the sample with replacement also known as row sampling. Thus, since we combine many results together to reach the output of majority voting, that is the reason why it is also known as aggregation. This reduces the variance in the model which is typically much higher for decision trees. This means that when we make predictions on new test sets, the results tend to differ and results are not as reproducible.

What did we learn:

Notsky:

I learned about variations of the greedy algorithms, namely: Iterative Dichotomiser 3 (id3), c4.5, and CART (Classification and Regression Trees). I learned about the differences, pros and cons of each approach and how these algorithms had various applications in Machine Learning. This would be very helpful for me as I am taking the Machine Learning class here at Northeastern University next semester. I am very much interested in AI. I am just getting started but I am very excited to learn more about its applications in the real world, and how I can hopefully make a contribution in the field. Sometimes the insane amounts of math and the complications of the code get me bogged down for a significant amount of time.

But, if you really want to achieve something or be relatively good at something, you definitely have to undergo some growing/learning pains.

Ameya Santosh Gidh:- After studying the project in depth I learnt more about how some algorithms are related to machine learning and what are the analysis we perform to evaluate them. Data is the new fuel and this project helped me gain more information about the data preprocessing and prediction. I learnt about the ID3, CART algorithm and learnt how to code this from scratch.

Mattia:

Researching Decisions Trees such as ID3, CART and C4.5 has been beneficial to me for exposing me to more complex algorithms models that I was not aware of before. The question about how to get the right set of questions for the right set of data truly fascinates me and reminds me of my previous philosophical studies. However, I feel that in order to master the Decision Tree algorithm knowledge at this level requires a much deeper mathematical understanding that is preventing me to fully embrace the underlining context. I remained fascinated with the concept and usage of greedy algorithms that we explored in this semester but aware about its limitations. Overall, I had the chance to explore Decisions Trees, Binary Trees and Greedy algorithms in much more depth that I was expecting.

Karan Shah:- I enjoyed the overall experience and supervising the team as a whole.

Appendix

https://github.com/ameyagidh/algorithm-Decision-tree-from-scartch

References:

Bramer, Max A. *Artificial Intelligence XXXV: 38th SGAI International Conference on Artificial Intelligence, AI 2018, Cambridge, UK, December 11-13, 2018: Proceedings*. Cham: Springer, 2018.

Dash, Shailey. "Decision Trees Explained - Entropy, Information Gain, Gini Index, CCP Pruning.." Medium. Towards Data Science, November 2, 2022. https://towardsdatascience.com/decision-trees-explained-entropy-information-gain-gini-index-ccp-pruning-4d78070db36c.

Kanade, Vijay. "What Is a Decision Tree? Algorithms, Template, Examples, and Best Practices." Decision Tree Algorithms, Template, Best Practices, May 30, 2022. https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-decision-tree/.

Mitchell, Tom Michael. "3." Essay. In *Machine Learning*. New York: McGraw-Hill, 1997.

Narkhede, Sarang. "Understanding Confusion Matrix." Medium. Towards Data Science, June 15, 2021. https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62.

Qureshi, Mohd Ehtesham Uddin. “Time & Space Complexity of Binary Tree Operations.” OpenGenus IQ: Computing Expertise & Legacy. OpenGenus IQ: Computing Expertise & Legacy, December 8, 2021. https://iq.opengenus.org/time-complexity-of-binary-tree/.

Sahin, Zehra. “Supervised Learning vs. Unsupervised Learning vs. Reinforcement Learning.” Medium. MLearning.ai, October 18, 2021. https://medium.com/mlearning-ai/supervised-learning-vs-unsupervised-learning-vs-reinforcement-learning-a2a7d89b6a99.

Schlagenhauf, Tobias. “31. Decision Trees in Python.” 31. Decision Trees in Python | Machine Learning. Accessed December 13, 2022. https://python-course.eu/machine-learning/decision-trees-in-python.php.

Tyagi, Neelam. “Understanding the Gini Index and Information Gain in Decision Trees.” Medium. Analytics Steps, September 30, 2020. https://medium.com/analytics-steps/understanding-the-gini-index-and-information-gain-in-decision-trees-ab4720518ba8.

In [1]:
```
#Entropy(S) = - ∑ pᵢ * log₂(pᵢ) ; i = 1 to n
#IG(S, A) = Entropy(S) - ∑((|Sᵥ| / |S|) * Entropy(Sᵥ))
# !pwd
# !ls ../Data
```

Importing Necessary Libraries

In [2]:
```
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_curve, roc_curve, confusion_matrix,auc
from sklearn.metrics import RocCurveDisplay
```

Performing data preprocessing to binarize the features present in the data.

In [3]:
```
fileInpTrain = "../Data/preProcess/car-data-train.csv"
fileInpTest = "../Data/preProcess/car-data-test.csv"


df = pd.read_csv(fileInpTrain, header=None)

df.iloc[:, 0] = df.iloc[:, 0].apply(lambda val: "low"
                                    if val == "med"
                                    else "high" if val == "vhigh"
                                    else val)

df.iloc[:, 1] = df.iloc[:, 1].apply(lambda val: "low"
                                    if val == "med"
                                    else "high" if val == "vhigh"
                                    else val)

# Created >2 other doors into one bin and leave 2 doors as 1 bin #uneven feature binning
# Check for other binning strategy
df.iloc[:, 2] = df.iloc[:, 2].apply(lambda val: val
                                    if val == '2'
                                    else '>2')


# Integer Division
FirstHalfIdx = df.iloc[:, 3][df.iloc[:, 3] == "4"].index[:len(df.iloc[:, 3][df.iloc[:, 3] == "4"].index)//2 + 1]
SecondHalfIdx = df.iloc[:, 3][df.iloc[:, 3] == "4"].index[len(df.iloc[:, 3][df.iloc[:, 3] == "4"].index)//2 + 1:]

# Number of person. Half of 4 in 2 and other half in more
df.iloc[:, 3] = df.apply(lambda row: "2" if row.name in FirstHalfIdx
                                         else "more"
                                         if row.name in SecondHalfIdx
                                         else row[3], axis=1)


# Integer Division
FirstHalfIdx = df.iloc[:, 4][df.iloc[:, 4] == "med"].index[:len(df.iloc[:, 4][df.iloc[:, 4] == "med"].index)//2 + 1
SecondHalfIdx = df.iloc[:, 4][df.iloc[:, 4] == "med"].index[len(df.iloc[:, 4][df.iloc[:, 4] == "med"].index)//2 + 1

#Lug Boot. Half of med added to small and other half med in more
df.iloc[:, 4] = df.apply(lambda row: "small" if row.name in FirstHalfIdx
                                             else "big"
                                             if row.name in SecondHalfIdx
                                             else row[4], axis=1)

# Integer Division
FirstHalfIdx = df.iloc[:, 5][df.iloc[:, 5] == "med"].index[:len(df.iloc[:, 5][df.iloc[:, 5] == "med"].index)//2 + 1
SecondHalfIdx = df.iloc[:, 5][df.iloc[:, 5] == "med"].index[len(df.iloc[:, 5][df.iloc[:, 5] == "med"].index)//2 + 1

# Number of person. Half of 4 in 2 and other half in more
df.iloc[:, 5] = df.apply(lambda row: "low" if row.name in FirstHalfIdx
                                           else "high"
                                           if row.name in SecondHalfIdx
                                           else row[5], axis=1)


# Count the binnings
value_counts = [df.iloc[:, i].value_counts() for i in range(len(df.columns))]

df.columns = ["buying", "maint", "doors", "persons", "lug_boot", "safety", "label"]

fileOutTrain = "../Data/postProcess/train.csv"
fileOutTest = "../Data/postProcess/test.csv"

df.to_csv(fileOutTrain, index=None)
```

Creating the Decision tree class and a complimentary node class.

We have shown how the two algorithms for making a decision tree work such as cart and Id3 which are the fundamental algorithms used for designing a decision tree.

In [4]:

```
'''
Node class acts as a supplimentary class
which creates the most basic element for
building a tree.

'''
class Node:
    def __init__(self, data, maxdepth):
        self.data = data
        self.maxdepth = maxdepth
        self.key = None #splitting attribute
        self.infogain = None
        self.label = None #classifier
        self.left = None
        self.right = None
        self.parent = None
        self.value = None #attribute value (ex. 'y' or 'n')

'''
Gini Index is a concise way to measure income disparity.
Incorporating the detailed share data into a single statistic,
the Gini coefficient captures the income distribution over the
full income distribution.

'''

def gini_index(labels):
    totalNumofLabels = len(labels)
    classes, countOfClasses = np.unique(labels, return_counts=True)
    if len(classes) <= 1:
        return 0
    ## computation
    probs = countOfClasses/totalNumofLabels
    gini_index = 0
    for p in probs:
        #calculation used to determine the gini index of the labels
        gini_index += p**2
    return 1 - gini_index

'''
Entropy is a metric used in information theory to
gauge how pure or uncertain a set of observations is.
It controls the way a decision tree decides how to
divide data.
'''

def entropy(labels):
    totalNumofLabels = len(labels)
    classes, countOfClasses = np.unique(labels, return_counts=True)
    #if arity of the labels is 1 then the entropy is 1
    if len(classes) <= 1:
        return 0
    ##computation
    probs = countOfClasses/totalNumofLabels
    ent = 0
    for p in probs:
        #calculation used to determine the entropy of the labels
        ent += p*math.log2(p)
    return -1 * ent

'''
These are helper functions. They are used to find mutual information value.

'''

def mutualInfo(data, position, switch):
    if switch == "entropy":
        ent = entropy(data[:,-1])
        lengthOfValues = len(data[:, position])
        classes, countOfClasses = np.unique(data[:, position], return_counts=True)
        if len(classes) <= 1:
            return 0
        #calculate probabilities used in conditional entropy
        probs = countOfClasses/lengthOfValues
        #calculate specific conditional entropies
        specCondEnts = []
        for c in classes:
            specCondEnt = 0
            boolean = data[:, position] == c
            subset = data[boolean]
            lengthOfSubset = len(subset)
            subsetClasses, subsetCountofClasses = np.unique(subset[:,-1], return_counts=True)
            subsetClasses, subsetCountofClasses
            probsOfSubset = subsetCountofClasses/lengthOfSubset
            for p in probsOfSubset:
                if p != 0:
```

```
87                      specCondEnt += p*math.log2(p)
                    else:
                        specCondEnt += 0
                specCondEnts.append(-1*specCondEnt)
            #calculate conditional entropy
            condEnt = sum(p*specCondEnts[idx] for idx, p in enumerate(probs))
            #calculate info gain
            infoGain = ent - condEnt
        elif switch == "gini" :
            ent = gini_index(data[:, -1])
            lengthOfValues = len(data[:, position])
            classes, countOfClasses = np.unique(data[:, position], return_counts=True)
            if len(classes) <= 1:
                return 0
            #calculate probabilities used in conditional entropy
            probs = countOfClasses/lengthOfValues
            #calculate specific conditional entropies
            specCondEnts = []
            for c in classes:
                specCondEnt = 0
                boolean = data[:, position] == c
                subset = data[boolean]
                lengthOfSubset = len(subset)
                subsetClasses, subsetCountofClasses = np.unique(subset[:,-1], return_counts=True)
                subsetClasses, subsetCountofClasses
                probsOfSubset = subsetCountofClasses/lengthOfSubset
                for p in probsOfSubset:
                    if p != 0:
                        specCondEnt += p**2
                    else:
                        specCondEnt += 0
                specCondEnts.append(1 - specCondEnt)
            #calculate conditional entropy
            condEnt = sum(p*specCondEnts[idx] for idx, p in enumerate(probs))
            #calculate info gain
            infoGain = ent - condEnt
        else:
            raise Exception("No other criterion. Only entropy or gini")

        return infoGain

    '''
    These are helper functions. They are used to split the
    data value at each Node.

    '''

    def splitData(data, idx):
        classes = np.unique(data[:,idx])
        subsets = []
        for c in classes:
            boolean = data[:, idx] == c
            subset = data[boolean]
            subsets.append(subset)

        return subsets

    '''
    These are helper functions. Function returns attribute that
    obtains the highest mutual information at each node.

    '''

    def bestMutualInfo(data, attributes, switch): #optimize this to work better
        if len(attributes) == 1: #unnecessary
            return attributes[0]
        elif len(attributes) == 0:
            return None
        else:
            atts = attributes.copy()[:len(attributes)-1]
            bestInfoGain = -np.inf
            bestatt = None
            for idx, att in enumerate(atts):
                if mutualInfo(data, idx, switch) >= bestInfoGain:
                    bestInfoGain = mutualInfo(data, idx, switch=switch)
                    bestatt = atts[idx]

            return bestInfoGain, bestatt

    '''
    These are helper functions. Function need to pass last column
    which is the target variable calculates the majority class
    for the labels.

    '''
```

```
173 def maj_classifer(data):
        labels = data.copy()
        classes, countOfClasses = np.unique(labels, return_counts=True)
        if len(classes) == 1:
            return classes[0]
        counter = 0
        maxValue = -np.inf
        best_label = None
        while counter < len(classes):
            for idx, count in enumerate(countOfClasses):
                if count >= maxValue:
                    maxValue = count
                    best_label = classes[idx]
                counter += 1
        return best_label

    '''
    These are helper functions. Recursive Algorithm to build the tree

    '''

    def buildTree(traindata,feats, maxdepth, switch):
        if maxdepth == 0:
            return maj_classifer(traindata[:,-1])
        #maxdepth is limited by number of features
        if maxdepth > len(feats) + 1:
            maxdepth = len(feats) + 1
        #create the root node with all the training data and an initial max depth
        root = Node(traindata, maxdepth)
        #calculate the information gain at that node and the attribute that best splits the data at that node
        infoGainVal, bestAtt = bestMutualInfo(root.data, feats, switch=switch)
        #base case
        if infoGainVal <= 0:
            root.key = 'leaf'
            root.label = maj_classifer(root.data[:, -1])
            return root
        root.key = bestAtt
        root.label = maj_classifer(root.data[:,-1])
        root.infogain = infoGainVal
        #split the root node data
        rightData, leftData = splitData(root.data, feats.index(bestAtt))

        #recurse to the left subtree
        if maxdepth != 1 and root.left == None :
            root.left = buildTree(leftData, feats, maxdepth - 1, switch=switch)
            root.left.value = leftData[0,feats.index(bestAtt)]
            root.left.parent = bestMutualInfo(root.data, feats, switch=switch)[1]

        #recurse to the right subtree
        if maxdepth != 1 and root.right == None :
            root.right = buildTree(rightData, feats, maxdepth - 1, switch=switch)
            root.right.value = rightData[0,feats.index(bestAtt)]
            root.right.parent = bestMutualInfo(root.data, feats, switch=switch)[1]

        return root

    '''
    These are helper functions. Recursive Algorithm to print the tree

    '''

    def printPreorder(root, classOne, classTwo, counter = 0):
        if root:
            classes = [classOne, classTwo]
            _, countOfClasses = np.unique(root.data[:,-1], return_counts=True)
            if counter == 0:
                print('[{} {} /{} {}]\n'.format(countOfClasses[0], classes[0], countOfClasses[1], classes[1]))
            else:
                if len(countOfClasses) == 2:
                    print('|'* counter + '{} = {}: [{} {} /{} {}] \n'.format(root.parent, root.value, countOfClasses[0]
                elif len(countOfClasses) == 1 and _ == classes[0]:
                    print('|' * counter +'{} = {}: [{} {} /{} {}] \n'.format(root.parent, root.value, countOfClasses[0]
                else:
                    print('|' * counter + '{} = {}: [{} {} /{} {}] \n'.format(root.parent, root.value, 0, classes[0], c

            # Then recur on left child
            printPreorder(root.left, classes[0], classes[1], counter+1)
            #Finally recur on right child
            printPreorder(root.right, classes[0], classes[1], counter+1)

    '''
    These are helper functions. Recursive Algorithm to print the tree

    '''
    def printPreorder2(tree=None, indent=" "):
        ''' function to print the tree '''
```

```
259     if tree:
            print("X_"+str(tree.key), " ? ", str(tree.infogain))
            print("%sleft:" % (indent), end="")
            printPreorder2(tree.left, indent + indent)
            print("%sright:" % (indent), end="")
            printPreorder2(tree.right, indent + indent)


    '''
    These are helper functions. Recursive function that traverses the tree
    and return the prediction of the query

    '''

    def prediction(tree, feats, row, maxdepth, currentdepth=1):
        #base case
        if tree.key == 'leaf':
            return tree.label
        #base case
        if maxdepth == currentdepth:
            return tree.label
        #recurse
        if any(tree.key == feat for feat in feats):
            idx = feats.index(tree.key)
            if row[idx] == tree.left.value:
                left = prediction(tree.left,feats,row, maxdepth, currentdepth + 1)
                return left
            if row[idx] == tree.right.value:
                right = prediction(tree.right,feats,row, maxdepth, currentdepth + 1)
                return right
```

Using the ID3 Algorithm

In [5]:
```
    # Setting up the train and test data.
    train = pd.read_csv("../Data/postProcess/train.csv")
    test = pd.read_csv("../Data/postProcess/test.csv")

    train = train.to_numpy()
    test = test.to_numpy()
    XTrain_Id3, yTrain_Id3 = train[: , :train.shape[1] - 1], train[:, -1]
    XTest_Id3, yTest_Id3 = test[: , :test.shape[1] - 1], test[:, -1]

    features = ["buying", "maint", "doors", "persons", "lug_boot", "safety", "label"]
    maxDepth = 4

    root = buildTree(train, features, maxDepth, switch="entropy")

    #printout of trained tree
    classes = np.unique(root.data[:,-1])
    printPreorder(root, classes[0], classes[1])
    # print()
    # print(printPreorder2(root))

    #Train on train and predict on Train
    yTrainPred_Id3 = []
    for row in XTrain_Id3:
        yTrainPred_Id3.append(prediction(root,  features, row, maxDepth))

    #Train on train and predict on Test
    yTestPred_Id3 = []
    for row in XTest_Id3:
        yTestPred_Id3.append(prediction(root,  features, row, maxDepth))

    yTrainPred_Id3 = np.array(yTrainPred_Id3)
    yTestPred_Id3 = np.array(yTestPred_Id3)
```

```
[386 acc /910 unacc]

|safety = low: [83 acc /571 unacc]

||buying = low: [59 acc /272 unacc]

|||doors = >2: [48 acc /200 unacc]

|||doors = 2: [11 acc /72 unacc]

||buying = high: [24 acc /299 unacc]

|||maint = low: [20 acc /140 unacc]

|||maint = high: [4 acc /159 unacc]

|safety = high: [303 acc /339 unacc]

||persons = more: [239 acc /111 unacc]

|||buying = low: [160 acc /20 unacc]

|||buying = high: [79 acc /91 unacc]

||persons = 2: [64 acc /228 unacc]

|||lug_boot = small: [45 acc /103 unacc]

|||lug_boot = big: [19 acc /125 unacc]
```

Accuracy

Calculating the accuracy for Train Data on ID3 Algorithm

In [6]:
```
    sum(yTrainPred_Id3 == yTrain_Id3)/len(yTrain_Id3)
```

Out[6]:
```
0.8101851851851852
```

Calculating the accuracy for Test Data on ID3

In [7]:
```
#Acc for Test Data on ID3
sum(yTestPred_Id3 == yTest_Id3)/len(yTest_Id3)
```

Out[7]:
```
0.8078703703703703
```

Confusion Matrix

A Confusion Matrix is a table called a confusion matrix is used to describe how well a classification system performs. the output of a classification algorithm is shown and summarized in a confusion matrix.

Confusion Matrix for ID3 Train data

In [8]:
```
confusionMatrix_Id3_Train = confusion_matrix(yTrain_Id3,yTrainPred_Id3)
confusionMatrix_Id3_Train
```

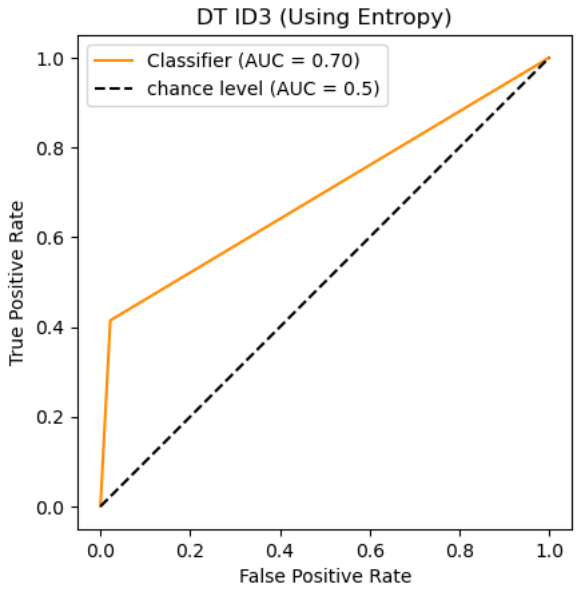
Out[8]:
```
array([[160, 226],
       [ 20, 890]])
```

ROC Curve for ID3 Train

In [9]:
```
binaryTransformation = lambda val: 0 if val == 'unacc' else 1
fn = np.vectorize(binaryTransformation)
yTrain_Id3 = fn(yTrain_Id3)
yTrainPred_Id3 = fn(yTrainPred_Id3)
```

In [10]:
```
rocCurve_ID3 = roc_curve(yTrain_Id3, yTrainPred_Id3, pos_label=1)
fpr, tpr, threshold = rocCurve_ID3
```

In [11]:
```
RocCurveDisplay.from_predictions(
    yTrain_Id3,
    yTrainPred_Id3,
    color="darkorange"
)
plt.plot([0, 1], [0, 1], "k--", label="chance level (AUC = 0.5)")
plt.axis("square")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("DT ID3 (Using Entropy)")
plt.legend()
plt.show()
```

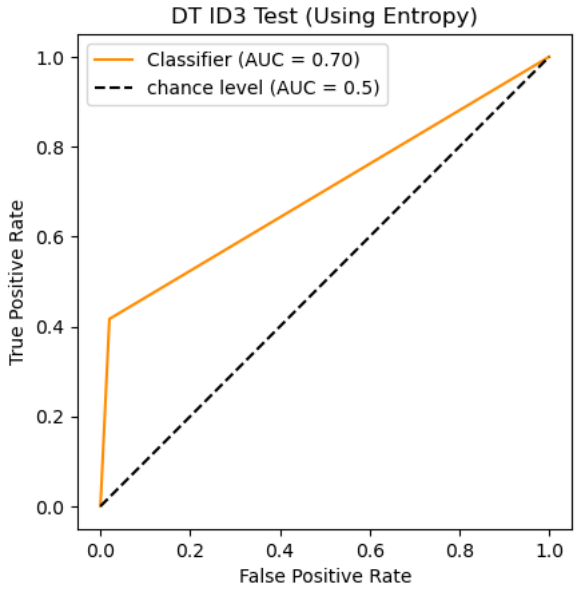


ROC Curve for ID3 Test

In [12]:
```
binaryTransformation = lambda val: 0 if val == 'unacc' else 1
fn = np.vectorize(binaryTransformation)
yTest_Id3 = fn(yTest_Id3)
yTestPred_Id3 = fn(yTestPred_Id3)
```

In [13]:
```
rocCurve_ID3Test = roc_curve(yTest_Id3, yTestPred_Id3, pos_label=1)
fpr, tpr, threshold = rocCurve_ID3Test
```

In [14]:
```
RocCurveDisplay.from_predictions(
    yTest_Id3,
    yTestPred_Id3,
    color="darkorange"
)
plt.plot([0, 1], [0, 1], "k--", label="chance level (AUC = 0.5)")
plt.axis("square")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("DT ID3 Test (Using Entropy)")
plt.legend()
plt.show()
```



Cart Algorithm on Car Evaluation

In [15]:
```
XTrain_Cart, yTrain_Cart = train[: , :train.shape[1] - 1], train[:, -1]
XTest_Cart, yTest_Cart = test[: , :test.shape[1] - 1], test[:, -1]

features = ["buying", "maint", "doors", "persons", "lug_boot", "safety", "label"]
maxDepth = 4

root = buildTree(train, features, maxDepth, switch="gini")

#printout of trained tree
classes = np.unique(root.data[:,-1])
printPreorder(root, classes[0], classes[1])

# printPreorder2(root)

#Train on train and predict on Train
yTrainPred_Cart = []
for row in XTrain_Cart:
    yTrainPred_Cart.append(prediction(root,  features, row, maxDepth))

#Train on train and predict on Test
yTestPred_Cart = []
for row in XTest_Cart:
    yTestPred_Cart.append(prediction(root,  features, row, maxDepth))

yTrainPred_Cart = np.array(yTrainPred_Cart)
yTestPred_Cart = np.array(yTestPred_Cart)
```

```
[386 acc /910 unacc]

|safety = low: [83 acc /571 unacc]

||buying = low: [59 acc /272 unacc]

|||doors = >2: [48 acc /200 unacc]

|||doors = 2: [11 acc /72 unacc]

||buying = high: [24 acc /299 unacc]

|||lug_boot = small: [5 acc /173 unacc]

|||lug_boot = big: [19 acc /126 unacc]

|safety = high: [303 acc /339 unacc]

||persons = more: [239 acc /111 unacc]

|||buying = low: [160 acc /20 unacc]

|||buying = high: [79 acc /91 unacc]

||persons = 2: [64 acc /228 unacc]

|||lug_boot = small: [45 acc /103 unacc]

|||lug_boot = big: [19 acc /125 unacc]
```

Calculating the accuracy for Train Data on CART Algorithm

In [16]:
```
#Acc for Train Data on CART
sum(yTrainPred_Cart == yTrain_Cart)/len(yTrain_Cart)
```

Out[16]: 0.8101851851851852

Calculating the accuracy for Test Data on CART Algorithm

In [17]:
```
#Acc for Test Data on CART
sum(yTestPred_Cart == yTest_Cart)/len(yTest_Cart)
```

Out[17]: 0.8078703703703703

In [18]:
```
df.head()
```

Out[18]:

	buying	maint	doors	persons	lug_boot	safety	label
0	high	low	2	more	small	high	unacc
1	low	low	>2	2	small	high	acc
2	low	high	>2	more	small	low	unacc
3	high	low	>2	2	big	high	acc
4	low	low	>2	more	small	high	acc

Confusion Matrix

A Confusion Matrix is a table called a confusion matrix is used to describe how well a classification system performs. the output of a classification algorithm is shown and summarized in a confusion matrix.

In [19]:
```
confusionMatrix_Cart_Train = confusion_matrix(yTrain_Cart,yTrainPred_Cart)
confusionMatrix_Cart_Train
```

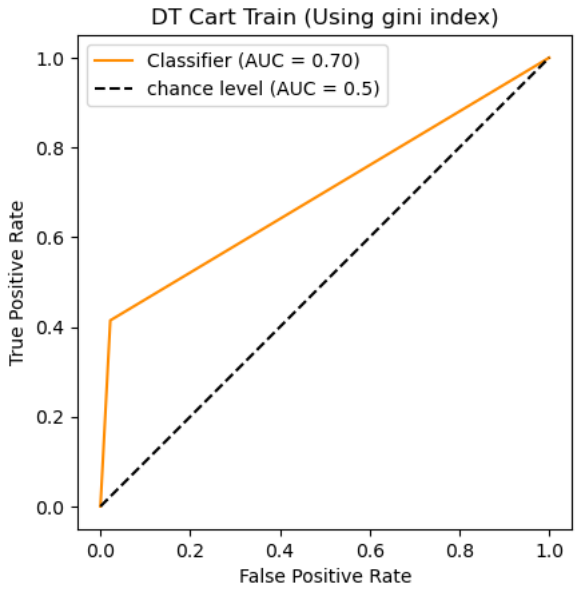
Out[19]:
```
array([[160, 226],
       [ 20, 890]])
```

ROC Curve for CART Train

In [20]:
```
binaryTransformation = lambda val: 0 if val == 'unacc' else 1
fn = np.vectorize(binaryTransformation)
yTrain_Cart = fn(yTrain_Cart)
yTrainPred_Cart = fn(yTrainPred_Cart)
```

In [21]:
```
rocCurve_Cart = roc_curve(yTrain_Cart, yTrainPred_Cart, pos_label=1)
fpr, tpr, threshold = rocCurve_Cart
```

In [22]:
```
RocCurveDisplay.from_predictions(
    yTrain_Cart,
    yTrainPred_Cart,
    color="darkorange"
)
plt.plot([0, 1], [0, 1], "k--", label="chance level (AUC = 0.5)")
plt.axis("square")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("DT Cart Train (Using gini index)")
plt.legend()
plt.show()
```

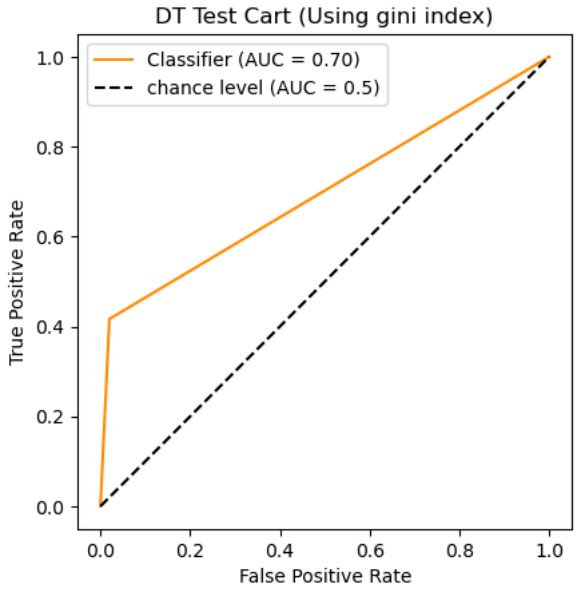


ROC for CART Test

In [23]:
```
binaryTransformation = lambda val: 0 if val == 'unacc' else 1
fn = np.vectorize(binaryTransformation)
yTest_Cart = fn(yTest_Cart)
yTestPred_Cart = fn(yTestPred_Cart)
```

In [24]:
```
rocCurve_CartTest = roc_curve(yTest_Id3, yTestPred_Id3, pos_label=1)
fpr, tpr, threshold = rocCurve_CartTest
```

In [25]:
```
RocCurveDisplay.from_predictions(
    yTest_Cart,
    yTestPred_Cart,
    color="darkorange"
)
plt.plot([0, 1], [0, 1], "k--", label="chance level (AUC = 0.5)")
plt.axis("square")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("DT Test Cart (Using gini index)")
plt.legend()
plt.show()
```



Helper function for converting the values into binary

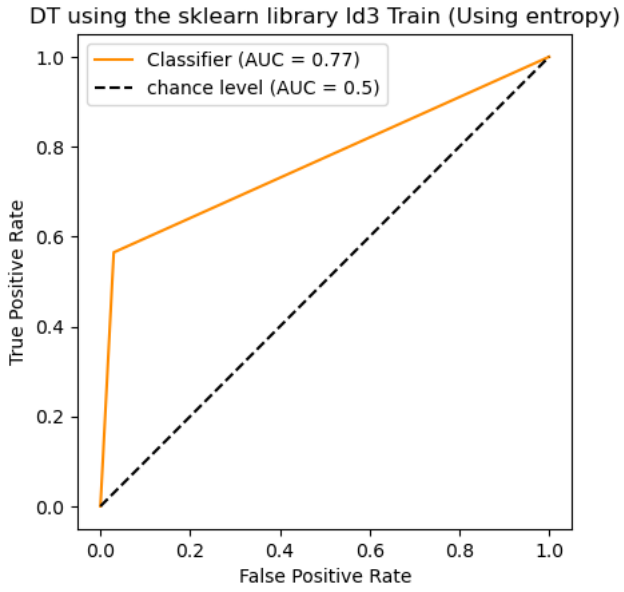
In [26]:
```
def conversion(XTrain_):
    for i in range(len(XTrain_)):
        for j in range(len(XTrain_[i])):
            if XTrain_[i][j] == 'high':
                XTrain_[i][j] = 1
            elif XTrain_[i][j] == 'low':
                XTrain_[i][j] = 0
            elif XTrain_[i][j] == '2':
                XTrain_[i][j] = 1
            elif XTrain_[i][j] == '>2':
                XTrain_[i][j] = 0
            elif XTrain_[i][j] == 'more':
                XTrain_[i][j] = 0
            elif XTrain_[i][j] == 'small':
                XTrain_[i][j] = 0
            elif XTrain_[i][j] == 'big':
                XTrain_[i][j] = 1
            elif XTrain_[i][j] == 'high':
                XTrain_[i][j] = 1
            elif XTrain_[i][j] == 'low':
                XTrain_[i][j] = 0
    return XTrain_
```

Decision tree using library of sklearn

Train ID3 data

In [27]:
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(criterion='entropy', splitter='best', max_depth=None, min_samples_split=2)
xt = conversion(XTrain_Id3)
dt.fit(xt,yTrain_Id3)
xtt = conversion(XTest_Id3)
yTrainPred_Id3= dt.predict(XTrain_Id3)
```

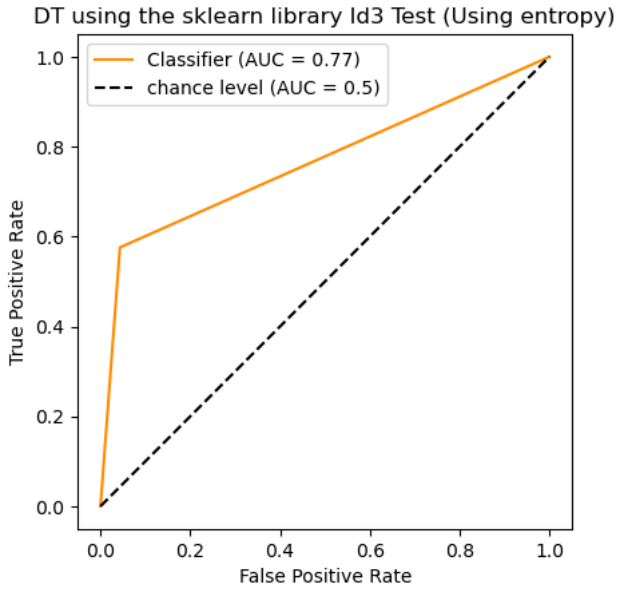
In [28]:
```
RocCurveDisplay.from_predictions(
#     yTest_Id3,y_pred,
    yTrain_Id3,yTrainPred_Id3,
    color="darkorange"
)
plt.plot([0, 1], [0, 1], "k--", label="chance level (AUC = 0.5)")
plt.axis("square")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("DT using the sklearn library Id3 Train (Using entropy)")
plt.legend()
plt.show()
```



Test ID3 data

In [29]:
```
dt = DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None, min_samples_split=2)
xt1 = conversion(XTest_Id3)
dt.fit(xt1,yTest_Id3)
# xtt = conversion(XTest_Id3)
yTestPred_Id3 = dt.predict(XTest_Id3)
```

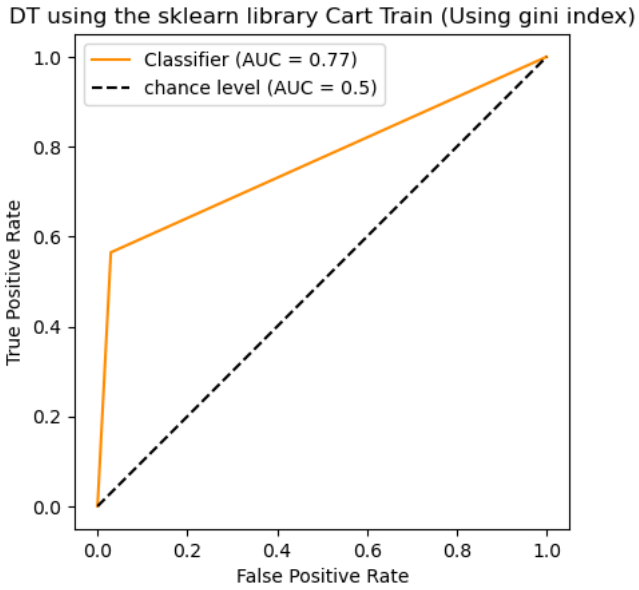
In [30]:
```
RocCurveDisplay.from_predictions(
    yTest_Id3,yTestPred_Id3,
    color="darkorange"
)
plt.plot([0, 1], [0, 1], "k--", label="chance level (AUC = 0.5)")
plt.axis("square")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("DT using the sklearn library Id3 Test (Using entropy)")
plt.legend()
plt.show()
```



Train data for Cart

In [31]:
```
dt = DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None, min_samples_split=2)
# xt2 = conversion(XTr_Cart)
dt.fit(XTrain_Cart,yTrain_Cart)
yTrainPred_Cart = dt.predict(XTrain_Cart)
```

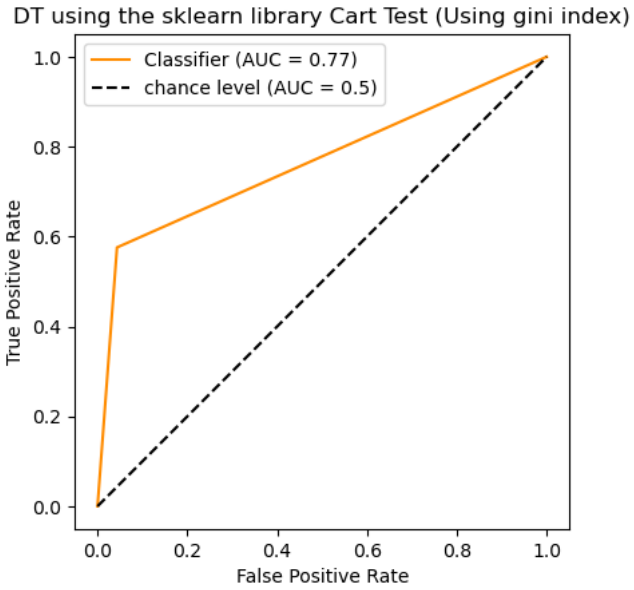
In [32]:
```
RocCurveDisplay.from_predictions(
    yTrain_Cart,yTrainPred_Cart,
    color="darkorange"
)
plt.plot([0, 1], [0, 1], "k--", label="chance level (AUC = 0.5)")
plt.axis("square")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("DT using the sklearn library Cart Train (Using gini index)")
plt.legend()
plt.show()
```



Test data for Cart

In [33]:
```
dt = DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None, min_samples_split=2)
# xt2 = conversion(XTr_Cart)
dt.fit(XTest_Cart,yTest_Cart)
yTestPred_Cart = dt.predict(XTest_Cart)
```

In [34]:
```
    RocCurveDisplay.from_predictions(
        yTest_Cart,yTestPred_Cart,
        color="darkorange"
    )
    plt.plot([0, 1], [0, 1], "k--", label="chance level (AUC = 0.5)")
    plt.axis("square")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("DT using the sklearn library Cart Test (Using gini index)")
    plt.legend()
    plt.show()
```



In [35]:
```
    value_counts
```

Out[35]:
```
[low     660
 high    636
 Name: 0, dtype: int64,
 low     648
 high    648
 Name: 1, dtype: int64,
 >2     963
 2      333
 Name: 2, dtype: int64,
 2       655
 more    641
 Name: 3, dtype: int64,
 big      657
 small    639
 Name: 4, dtype: int64,
 low     654
 high    642
 Name: 5, dtype: int64,
 unacc    910
 acc      386
 Name: 6, dtype: int64]
```