1. Construct a Decision tree using entropy as criterion. Write a function in python for entropy computation at each node. Using this, write a program to construct a decision tree for above data. Also, explain each step in the report. You can use only Numpy, panda and plotting libraries.

I have used OOP methodology to build the decision tree and its related helper methods.

The definition of Node of a decision tree is as follows.

Node:

1. node_type : a string which is either “internal”(for internal node) or “label”(for label nodes)
2. label : a string which represents the class of the target variable(used for reporting prediction)
3. entropy : a numerical value representing the entropy at this node when the feature “feature_name” is used for splitting. It is only available for nodes of “internal” type
4. feature_name : a string representing the name of a feature which is used to split the data at this node. Only available for nodes of type “internal”
5. children : a dictionary where key = attribute of the feature “feature_name” and values = a list of subtrees(nodes)
6. parent : it references the parent of this node

A brief description of all the methods used are as follows.

1. __init__(self, data, target_feature):

It just initializes the DT class by providing the dataset “data” and the target feature “target_feature”.

2. predict(self, record):

It takes a dict object “record” representing an instance of the dataset and it returns a list of string values that represent the path taken to make the decision. The mechanism is very simple as it just compares the value of the features at each node and chooses the respective branches. It does this until a leaf node is encountered.

3. _build_tree(self, data, features, parent):

This method contains the core logic to build the decision tree. It is a recursive method.

It takes the dataset “data”, all input features “features” and parent node “parent”. The mechanism of building the tree is the same as the algorithm of the ID3 algorithm. Its pseudocode is as follows.

ID3(*Examples*, *Target_attribute*, *Attributes*)

Examples are the training examples. Target_attribute is the attribute whose value is to be predicted by the tree. Attributes is a list of other attributes that may be tested by the learned decision tree. Returns a decision tree that correctly classifies the given Examples.

- Create a *Root* node for the tree
- If all *Examples* are positive, Return the single-node tree *Root*, with label = +
- If all *Examples* are negative, Return the single-node tree *Root*, with label = −
- If *Attributes* is empty, Return the single-node tree *Root*, with label = most common value of *Target_attribute* in *Examples*
- Otherwise Begin
    - $A \leftarrow$ the attribute from *Attributes* that best* classifies *Examples*
    - The decision attribute for *Root* $\leftarrow A$
    - For each possible value, v_i, of A,
        - Add a new tree branch below *Root*, corresponding to the test $A = v_i$
        - Let $Examples_{v_i}$ be the subset of *Examples* that have value v_i for A
        - If $Examples_{v_i}$ is empty
            - Then below this new branch add a leaf node with label = most common value of *Target_attribute* in *Examples*
            - Else below this new branch add the subtree ID3($Examples_{v_i}$, *Target_attribute*, *Attributes* − {A}))
- End
- Return *Root*

Source: link

4. _get_features(self):

It returns a dictionary object in which keys are the name of the features and the value is a list of attributes of that feature.

5. _get_entropy(self, data):

This method calculates and returns the entropy of the given subset “data” of the original dataset.

6. Main method:

The main method loads the data and trains the decision tree model. Finally, I have used this model to predict the class of a new record as follows.

```python
# load the data
data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/data.csv')

# initialize the tree
dt = DT(data, 'profit')

# predict the class of a new record
print(dt.predict({'price': 'low', 'maintenance': 'high', 'capacity': 5, 'airbag': 'no'}))
```

```
['capacity=5', 'maintenance=high', '[prediction=no]']
```

2. Use the gini index criterion and repeat all the tasks as in 1.

I have used OOP methodology to build the decision tree and its related helper methods.

The definition of Node of a decision tree is as follows.

Node:
1. node_type : a string which is either “internal”(for internal node) or “label”(for label nodes)
2. label : a string which represents the class of the target variable(used for reporting prediction)
3. gini_index : a numerical value representing the gini_index at this node when the feature “feature_name” is used for splitting. It is only available for nodes of “internal” type
4. feature_name : a string representing the name of a feature which is used to split the data at this node. Only available for nodes of type “internal”
5. left : a reference to the left subtree
6. right : a reference to the right subtree
7. left_attr_collection : a set of attributes of the feature “feature_name” which is used to confirm whether to go to left subtree
8. right_attr_collection : a set of attributes of the feature “feature_name” which is used to confirm whether to go to right subtree

A brief description of all the methods used are as follows.

1. __init__(self, data, target_feature):

It just initializes the DT class by providing the dataset “data” and the target feature “target_feature”.

2. predict(self, record):

It takes a dict object “record” representing an instance of the dataset and it returns a list of string values that represent the path taken to make the decision. The mechanism is very simple as it just compares the value of the features at each node and chooses the respective branches. It does this until a leaf node is encountered.

3. _build_tree(self, data, features, parent):

This method contains the core logic to build the decision tree. It is a recursive method.

It takes the dataset “data”, all input features “features” and parent node “parent”. The mechanism of building the tree is the same as the algorithm of the ID3 algorithm. The only difference is that it uses gini index rather than entropy to choose the branch.

4. _get_feature_attr_sets(self, attrs):

It returns a list of all possible combinations of the attributes of the features. Mathematically, if a feature has n different attributes then it returns all possible combinations of attributes taken 1 at a time, 2 at a time, …, n - 1 taken at a time. So, there will be a total of 2^n - 2 combinations.

5. _get_gini_index(self, data):

It calculates and returns the gini index of the given subset “data”.

6. Main method:

The main method loads the data and trains the decision tree model. Finally, I have used this model to predict the class of a new record as follows.

```
# load the data
data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/data.csv')

# initialize the tree
dt = DT(data, 'profit')

# predict the class of a new record
print(dt.predict({'price': 'low', 'maintenance': 'high', 'capacity': 5, 'airbag': 'no'}))
```

```
['maintenance=high', 'capacity=5', 'price=low', '[prediction=no]']
```

3. Scikit–learn based assignment.

Use data set https://archive.ics.uci.edu/ml/datasets/car+evaluation.

a) By using the DecisionTreeClassifier , train your model and report accuracy (F1) and confusion matrix. Plot the tree. Also, do analysis by varying parameters of DecisionTreeClassifier (at least with criterion and depth of tree). Add in a report.

b) By using sklearn.ensemble.RandomForestClassifier, train your model and report accuracy (F1) and confusion matrix.Also, do analysis by varying parameters of sklearn.ensemble.RandomForestClassifier (at least with criterion, depth of tree and number of tree equal to 10 and 20). Add in a report.

Description of the dataset is as follows.

Number of Instances: 1728

Number of input Attributes(features): 6

Attribute Values:

buying	v-high, high, med, low
maint	v-high, high, med, low
doors	2, 3, 4, 5-more
persons	2, 4, more
lug_boot	small, med, big
safety	low, med, high

a)

The tree built with default parameters is as shown in page no. 10. The default configuration uses the “criterion” parameter value as “gini”. And no restriction is placed upon the depth of the tree.

Here, the depth is 12. We can easily see that the tree is overfitting the data.

The accuracy, confusion matrix and F1 score is as follows.

Confusion matrix
```
 [[110  12   7   0]
 [  0  20   0   0]
 [  6   0 391   0]
 [  3   0   0  22]]
```

	precision	recall	f1-score	support
acc	0.92	0.85	0.89	129
good	0.62	1.00	0.77	20
unacc	0.98	0.98	0.98	397
vgood	1.00	0.88	0.94	25
accuracy			0.95	571
macro avg	0.88	0.93	0.89	571
weighted avg	0.96	0.95	0.95	571

By varying the “criterion” parameter to “entropy” and “max_depth” to 6, new tree is as shown in the page no 11.

This new tree is not overfitted as the max depth is not very high. It’s only 6.

The accuracy, confusion matrix and F1 score is as follows.

Confusion matrix

```
[[ 92   8  29   0]
 [  5  15   0   0]
 [  6   0 391   0]
 [  3   9   0  13]]
```

	precision	recall	f1-score	support
acc	0.87	0.71	0.78	129
good	0.47	0.75	0.58	20
unacc	0.93	0.98	0.96	397
vgood	1.00	0.52	0.68	25
accuracy			0.89	571
macro avg	0.82	0.74	0.75	571
weighted avg	0.90	0.89	0.89	571

b)

The default configuration in RandomForestClassifier are as follows.

1. n_estimators = 100, the number of trees in the forest
2. criterion = “gini”, The function to measure the quality of a split.
3. max_depth = None, so no restriction on the depth of tree

The confusion matrix, accuracy and F1 score of this random forest is as follows.

Confusion matrix
```
 [[113   8   7   1]
 [  2  17   0   1]
 [  8   0 389   0]
 [  3   3   0  19]]
```

	precision	recall	f1-score	support
acc	0.90	0.88	0.89	129
good	0.61	0.85	0.71	20
unacc	0.98	0.98	0.98	397
vgood	0.90	0.76	0.83	25
accuracy			0.94	571
macro avg	0.85	0.87	0.85	571
weighted avg	0.95	0.94	0.94	571

Now, the parameters are changed to the following configurations.

1. n_estimators = 20, the number of trees in the forest
2. criterion = “entropy”, The function to measure the quality of a split.
3. max_depth = 4, depth of tree

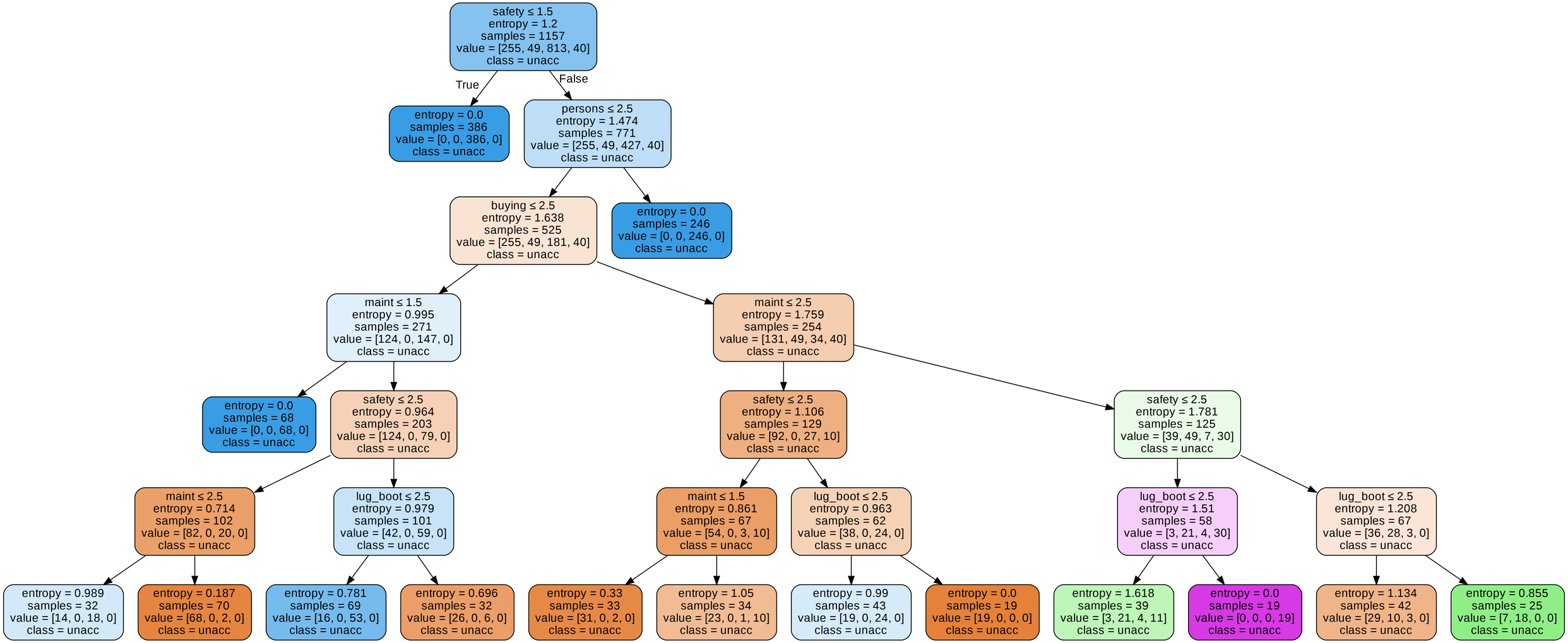
Its confusion matrix, accuracy and F1 score are as follows.

Confusion matrix
```
 [[ 98   0  31   0]
 [ 20   0   0   0]
 [ 12   0 385   0]
 [ 25   0   0   0]]
```

	precision	recall	f1-score	support
acc	0.63	0.76	0.69	129
good	0.00	0.00	0.00	20
unacc	0.93	0.97	0.95	397
vgood	0.00	0.00	0.00	25

accuracy			0.85	571
macro avg	0.39	0.43	0.41	571
weighted avg	0.79	0.85	0.81	571


safety ≤ 1.5
gini = 0.455
samples = 1157
value = [255, 49, 813, 40]
class = unacc
True
False
gini = 0.0
samples = 386
value = [0, 0, 386, 0]
class = unacc
persons ≤ 2.5
gini = 0.577
samples = 771
value = [255, 49, 427, 40]
class = unacc
gini = 0.0
samples = 246
value = [0, 0, 246, 0]
class = unacc


safety ≤ 1.5
entropy = 1.2
samples = 1157
value = [255, 49, 813, 40]
class = unacc
True
False
entropy = 0.0
samples = 386
value = [0, 0, 386, 0]
class = unacc
persons ≤ 2.5
entropy = 1.474
samples = 771
value = [255, 49, 427, 40]
class = unacc
buying ≤ 2.5
entropy = 1.638
samples = 525
value = [255, 49, 181, 40]
class = unacc
entropy = 0.0
samples = 246
value = [0, 0, 246, 0]
class = unacc
maint ≤ 1.5
entropy = 0.995
samples = 271
value = [124, 0, 147, 0]
class = unacc
maint ≤ 2.5
entropy = 1.759
samples = 254
value = [131, 49, 34, 40]
class = unacc
entropy = 0.0
samples = 68
value = [0, 0, 68, 0]
class = unacc
safety ≤ 2.5
entropy = 0.964
samples = 203
value = [124, 0, 79, 0]
class = unacc
safety ≤ 2.5
entropy = 1.106
samples = 129
value = [92, 0, 27, 10]
class = unacc
safety ≤ 2.5
entropy = 1.781
samples = 125
value = [39, 49, 7, 30]
class = unacc
maint ≤ 2.5
entropy = 0.714
samples = 102
value = [82, 0, 20, 0]
class = unacc
lug_boot ≤ 2.5
entropy = 0.979
samples = 101
value = [42, 0, 59, 0]
class = unacc
maint ≤ 1.5
entropy = 0.861
samples = 67
value = [54, 0, 3, 10]
class = unacc
lug_boot ≤ 2.5
entropy = 0.963
samples = 62
value = [38, 0, 24, 0]
class = unacc
lug_boot ≤ 2.5
entropy = 1.51
samples = 58
value = [3, 21, 4, 30]
class = unacc
lug_boot ≤ 2.5
entropy = 1.208
samples = 67
value = [36, 28, 3, 0]
class = unacc
entropy = 0.989
samples = 32
value = [14, 0, 18, 0]
class = unacc
entropy = 0.187
samples = 70
value = [68, 0, 2, 0]
class = unacc
entropy = 0.781
samples = 69
value = [16, 0, 53, 0]
class = unacc
entropy = 0.696
samples = 32
value = [26, 0, 6, 0]
class = unacc
entropy = 0.33
samples = 33
value = [31, 0, 2, 0]
class = unacc
entropy = 1.05
samples = 34
value = [23, 0, 1, 10]
class = unacc
entropy = 0.99
samples = 43
value = [19, 0, 24, 0]
class = unacc
entropy = 0.0
samples = 19
value = [19, 0, 0, 0]
class = unacc
entropy = 1.618
samples = 39
value = [3, 21, 4, 11]
class = unacc
entropy = 0.0
samples = 19
value = [0, 0, 0, 19]
class = unacc
entropy = 1.134
samples = 42
value = [29, 10, 3, 0]
class = unacc
entropy = 0.855
samples = 25
value = [7, 18, 0, 0]
class = unacc