**Decision Tree Classification with Python and Scikit-Learn**

In this project, I build a Decision Tree Classifier to predict the safety of the car. I build two models, one with criterion gini index and another one with criterion entropy. I implement Decision Tree Classification with Python and Scikit-Learn. I have used the **Car Evaluation Data Set** for this project, downloaded from the UCI Machine Learning Repository website.

**Table of Contents**

1. Introduction to Decision Tree algorithm
2. Classification and Regression Trees
3. Decision Tree algorithm intuition
4. Attribute selection measures
   * Information gain
   * Gini index
5. The problem statement
6. Dataset description
7. Import libraries
8. Import dataset
9. Exploratory data analysis
10. Declare feature vector and target variable
11. Split data into separate training and test set
12. Feature engineering
13. Decision Tree classifier with criterion gini-index
14. Decision Tree classifier with criterion entropy
15. Confusion matrix
16. Classification report
17. Results and conclusion

**1. Introduction to Decision Tree algorithm**

A Decision Tree algorithm is one of the most popular machine learning algorithms. It uses a tree like structure and their possible combinations to solve a particular problem. It belongs to the class of supervised learning algorithms where it can be used for both classification and regression purposes.

A decision tree is a structure that includes a root node, branches, and leaf nodes. Each internal node denotes a test on an attribute, each branch denotes the outcome of a test, and each leaf node holds a class label. The topmost node in the tree is the root node.

**2. Classification and Regression Trees (CART)**

Nowadays, Decision Tree algorithm is known by its modern name **CART** which stands for **Classification and Regression Trees**. Classification and Regression Trees or **CART** is a term introduced by Leo Breiman to refer to Decision Tree algorithms that can be used for classification and regression modeling problems.The CART algorithm provides a foundation for other important algorithms like bagged decision trees, random forest and boosted decision trees.

In this project, I will solve a classification problem. So, I will refer the algorithm also as Decision Tree Classification problem.

**3. Decision Tree algorithm intuition**

The Decision-Tree algorithm is one of the most frequently and widely used supervised machine learning algorithms that can be used for both classification and regression tasks. The intuition behind the Decision-Tree algorithm is very simple to understand.

The Decision Tree algorithm intuition is as follows:-

1. For each attribute in the dataset, the Decision-Tree algorithm forms a node. The most important attribute is placed at the root node.
2. For evaluating the task in hand, we start at the root node and we work our way down the tree by following the corresponding node that meets our condition or decision.
3. This process continues until a leaf node is reached. It contains the prediction or the outcome of the Decision Tree.

**4. Attribute selection measures**

The primary challenge in the Decision Tree implementation is to identify the attributes which we consider as the root node and each level. This process is known as the **attributes selection**. There are different attributes selection measure to identify the attribute which can be considered as the root node at each level.

There are 2 popular attribute selection measures. They are as follows:-

* **Information gain**
* **Gini index**

While using **Information gain** as a criterion, we assume attributes to be categorical and for **Gini index** attributes are assumed to be continuous. These attribute selection measures are described below.

**Information gain**

By using information gain as a criterion, we try to estimate the information contained by each attribute. To understand the concept of Information Gain, we need to know another concept called **Entropy**.

Entropy measures the impurity in the given dataset. In Physics and Mathematics, entropy is referred to as the randomness or uncertainty of a random variable X. In information theory, it refers to the impurity in a group of examples. **Information gain** is the decrease in entropy. Information gain computes the difference between entropy before split and average entropy after split of the dataset based on given attribute values.

The ID3 (Iterative Dichotomiser) Decision Tree algorithm uses entropy to calculate information gain. So, by calculating decrease in **entropy measure** of each attribute we can calculate their information gain. The attribute with the highest information gain is chosen as the splitting attribute at the node.

**Gini index**

Another attribute selection measure that **CART (Categorical and Regression Trees)** uses is the **Gini index**. It uses the Gini method to create split points.

Gini index says, if we randomly select two items from a population, they must be of the same class and probability for this is 1 if the population is pure.

It works with the categorical target variable “Success” or “Failure”. It performs only binary splits. The higher the value of Gini, higher the homogeneity. CART (Classification and Regression Tree) uses the Gini method to create binary splits.

Steps to Calculate Gini for a split

1. Calculate Gini for sub-nodes, using formula sum of the square of probability for success and failure (p^2+q^2).
2. Calculate Gini for split using weighted Gini score of each node of that split.

In case of a discrete-valued attribute, the subset that gives the minimum gini index for that chosen is selected as a splitting attribute. In the case of continuous-valued attributes, the strategy is to select each pair of adjacent values as a possible split-point and point with smaller gini index chosen as the splitting point. The attribute with minimum Gini index is chosen as the splitting attribute.

**5. The problem statement**

The problem is to predict the safety of the car. In this project, I build a Decision Tree Classifier to predict the safety of the car. I implement Decision Tree Classification with Python and Scikit-Learn. I have used the **Car Evaluation Data Set** for this project, downloaded from the UCI Machine Learning Repository website.

**6. Dataset description**

I have used the **Car Evaluation Data Set** downloaded from the Kaggle website. I have downloaded this data set from the Kaggle website. The data set can be found at the following url:-

<http://archive.ics.uci.edu/ml/datasets/Car+Evaluation>

Car Evaluation Database was derived from a simple hierarchical decision model originally developed for expert system for decision making. 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, lug\_boot, safety.

It was donated by Marko Bohanec.

**7. Import libraries**

In [1]:

**import** pandas **as** pd

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**import** seaborn **as** sns

**%matplotlib** inline

In [2]:

**import** warnings

warnings**.**filterwarnings('ignore')

**8. Import dataset**

In [3]:

data **=** 'C:/datasets/car.data'

df **=** pd**.**read\_csv(data, header**=None**)

**9. Exploratory data analysis**

Now, I will explore the data to gain insights about the data.

In [4]:

*# view dimensions of dataset*

df**.**shape

Out[4]:

(1728, 7)

We can see that there are 1728 instances and 7 variables in the data set.

**View top 5 rows of dataset**

In [5]:

*# preview the dataset*

df**.**head()

Out[5]:

|  | **0** | **1** | **2** | **3** | **4** | **5** | **6** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | vhigh | vhigh | 2 | 2 | small | low | unacc |
| **1** | vhigh | vhigh | 2 | 2 | small | med | unacc |
| **2** | vhigh | vhigh | 2 | 2 | small | high | unacc |
| **3** | vhigh | vhigh | 2 | 2 | med | low | unacc |
| **4** | vhigh | vhigh | 2 | 2 | med | med | unacc |

**Rename column names**

We can see that the dataset does not have proper column names. The columns are merely labelled as 0,1,2.... and so on. We should give proper names to the columns. I will do it as follows:-

In [6]:

col\_names **=** ['buying', 'maint', 'doors', 'persons', 'lug\_boot', 'safety', 'class']

df**.**columns **=** col\_names

col\_names

Out[6]:

['buying', 'maint', 'doors', 'persons', 'lug\_boot', 'safety', 'class']

In [7]:

*# let's again preview the dataset*

df**.**head()

Out[7]:

|  | **buying** | **maint** | **doors** | **persons** | **lug\_boot** | **safety** | **class** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | vhigh | vhigh | 2 | 2 | small | low | unacc |
| **1** | vhigh | vhigh | 2 | 2 | small | med | unacc |
| **2** | vhigh | vhigh | 2 | 2 | small | high | unacc |
| **3** | vhigh | vhigh | 2 | 2 | med | low | unacc |
| **4** | vhigh | vhigh | 2 | 2 | med | med | unacc |

We can see that the column names are renamed. Now, the columns have meaningful names.

**View summary of dataset**

In [8]:

df**.**info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 1728 entries, 0 to 1727

Data columns (total 7 columns):

buying 1728 non-null object

maint 1728 non-null object

doors 1728 non-null object

persons 1728 non-null object

lug\_boot 1728 non-null object

safety 1728 non-null object

class 1728 non-null object

dtypes: object(7)

memory usage: 94.6+ KB

**Frequency distribution of values in variables**

Now, I will check the frequency counts of categorical variables.

In [9]:

col\_names **=** ['buying', 'maint', 'doors', 'persons', 'lug\_boot', 'safety', 'class']

**for** col **in** col\_names:

print(df[col]**.**value\_counts())

med 432

low 432

vhigh 432

high 432

Name: buying, dtype: int64

med 432

low 432

vhigh 432

high 432

Name: maint, dtype: int64

5more 432

4 432

2 432

3 432

Name: doors, dtype: int64

4 576

2 576

more 576

Name: persons, dtype: int64

med 576

big 576

small 576

Name: lug\_boot, dtype: int64

med 576

low 576

high 576

Name: safety, dtype: int64

unacc 1210

acc 384

good 69

vgood 65

Name: class, dtype: int64

We can see that the doors and persons are categorical in nature. So, I will treat them as categorical variables.

**Summary of variables**

* There are 7 variables in the dataset. All the variables are of categorical data type.
* These are given by buying, maint, doors, persons, lug\_boot, safety and class.
* class is the target variable.

**Explore class variable**

In [10]:

df['class']**.**value\_counts()

Out[10]:

unacc 1210

acc 384

good 69

vgood 65

Name: class, dtype: int64

The class target variable is ordinal in nature.

**Missing values in variables**

In [11]:

*# check missing values in variables*

df**.**isnull()**.**sum()

Out[11]:

buying 0

maint 0

doors 0

persons 0

lug\_boot 0

safety 0

class 0

dtype: int64

We can see that there are no missing values in the dataset. I have checked the frequency distribution of values previously. It also confirms that there are no missing values in the dataset.

**10. Declare feature vector and target variable**

In [12]:

X **=** df**.**drop(['class'], axis**=**1)

y **=** df['class']

**11. Split data into separate training and test set**

In [13]:

*# split X and y into training and testing sets*

**from** sklearn.model\_selection **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size **=** 0.33, random\_state **=** 42)

In [14]:

*# check the shape of X\_train and X\_test*

X\_train**.**shape, X\_test**.**shape

Out[14]:

((1157, 6), (571, 6))

**12. Feature Engineering**

**Feature Engineering** is the process of transforming raw data into useful features that help us to understand our model better and increase its predictive power. I will carry out feature engineering on different types of variables.

First, I will check the data types of variables again.

In [15]:

*# check data types in X\_train*

X\_train**.**dtypes

Out[15]:

buying object

maint object

doors object

persons object

lug\_boot object

safety object

dtype: object

**Encode categorical variables**

Now, I will encode the categorical variables.

In [16]:

X\_train**.**head()

Out[16]:

|  | **buying** | **maint** | **doors** | **persons** | **lug\_boot** | **safety** |
| --- | --- | --- | --- | --- | --- | --- |
| **48** | vhigh | vhigh | 3 | more | med | low |
| **468** | high | vhigh | 3 | 4 | small | low |
| **155** | vhigh | high | 3 | more | small | high |
| **1721** | low | low | 5more | more | small | high |
| **1208** | med | low | 2 | more | small | high |

We can see that all the variables are ordinal categorical data type.

In [17]:

*# import category encoders*

**import** category\_encoders **as** ce

In [18]:

*# encode variables with ordinal encoding*

encoder **=** ce**.**OrdinalEncoder(cols**=**['buying', 'maint', 'doors', 'persons', 'lug\_boot', 'safety'])

X\_train **=** encoder**.**fit\_transform(X\_train)

X\_test **=** encoder**.**transform(X\_test)

In [19]:

X\_train**.**head()

Out[19]:

|  | **buying** | **maint** | **doors** | **persons** | **lug\_boot** | **safety** |
| --- | --- | --- | --- | --- | --- | --- |
| **48** | 1 | 1 | 1 | 1 | 1 | 1 |
| **468** | 2 | 1 | 1 | 2 | 2 | 1 |
| **155** | 1 | 2 | 1 | 1 | 2 | 2 |
| **1721** | 3 | 3 | 2 | 1 | 2 | 2 |
| **1208** | 4 | 3 | 3 | 1 | 2 | 2 |

In [20]:

X\_test**.**head()

Out[20]:

|  | **buying** | **maint** | **doors** | **persons** | **lug\_boot** | **safety** |
| --- | --- | --- | --- | --- | --- | --- |
| **599** | 2 | 2 | 4 | 3 | 1 | 2 |
| **1201** | 4 | 3 | 3 | 2 | 1 | 3 |
| **628** | 2 | 2 | 2 | 3 | 3 | 3 |
| **1498** | 3 | 2 | 2 | 2 | 1 | 3 |
| **1263** | 4 | 3 | 4 | 1 | 1 | 1 |

We now have training and test set ready for model building.

**13. Decision Tree Classifier with criterion gini index**

In [21]:

*# import DecisionTreeClassifier*

**from** sklearn.tree **import** DecisionTreeClassifier

In [22]:

*# instantiate the DecisionTreeClassifier model with criterion gini index*

clf\_gini **=** DecisionTreeClassifier(criterion**=**'gini', max\_depth**=**3, random\_state**=**0)

*# fit the model*

clf\_gini**.**fit(X\_train, y\_train)

Out[22]:

DecisionTreeClassifier(class\_weight=None, criterion='gini', max\_depth=3,

max\_features=None, max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, presort=False, random\_state=0,

splitter='best')

**Predict the Test set results with criterion gini index**

In [23]:

y\_pred\_gini **=** clf\_gini**.**predict(X\_test)

**Check accuracy score with criterion gini index**

In [24]:

**from** sklearn.metrics **import** accuracy\_score

print('Model accuracy score with criterion gini index: {0:0.4f}'**.** format(accuracy\_score(y\_test, y\_pred\_gini)))

Model accuracy score with criterion gini index: 0.8021

Here, **y\_test** are the true class labels and **y\_pred\_gini** are the predicted class labels in the test-set.

**Compare the train-set and test-set accuracy**

Now, I will compare the train-set and test-set accuracy to check for overfitting.

In [25]:

y\_pred\_train\_gini **=** clf\_gini**.**predict(X\_train)

y\_pred\_train\_gini

Out[25]:

array(['unacc', 'unacc', 'unacc', ..., 'unacc', 'unacc', 'acc'],

dtype=object)

In [26]:

print('Training-set accuracy score: {0:0.4f}'**.** format(accuracy\_score(y\_train, y\_pred\_train\_gini)))

Training-set accuracy score: 0.7865

**Check for overfitting and underfitting**

In [27]:

*# print the scores on training and test set*

print('Training set score: {:.4f}'**.**format(clf\_gini**.**score(X\_train, y\_train)))

print('Test set score: {:.4f}'**.**format(clf\_gini**.**score(X\_test, y\_test)))

Training set score: 0.7865

Test set score: 0.8021

Here, the training-set accuracy score is 0.7865 while the test-set accuracy to be 0.8021. These two values are quite comparable. So, there is no sign of overfitting.

**14. Decision Tree Classifier with criterion entropy**

In [28]:

*# instantiate the DecisionTreeClassifier model with criterion entropy*

clf\_en **=** DecisionTreeClassifier(criterion**=**'entropy', max\_depth**=**3, random\_state**=**0)

*# fit the model*

clf\_en**.**fit(X\_train, y\_train)

Out[28]:

DecisionTreeClassifier(class\_weight=None, criterion='entropy', max\_depth=3,

max\_features=None, max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, presort=False, random\_state=0,

splitter='best')

**Predict the Test set results with criterion entropy**

In [29]:

y\_pred\_en **=** clf\_en**.**predict(X\_test)

**Check accuracy score with criterion entropy**

In [30]:

**from** sklearn.metrics **import** accuracy\_score

print('Model accuracy score with criterion entropy: {0:0.4f}'**.** format(accuracy\_score(y\_test, y\_pred\_en)))

Model accuracy score with criterion entropy: 0.8021

**Compare the train-set and test-set accuracy**

Now, I will compare the train-set and test-set accuracy to check for overfitting.

In [31]:

y\_pred\_train\_en **=** clf\_en**.**predict(X\_train)

y\_pred\_train\_en

Out[31]:

array(['unacc', 'unacc', 'unacc', ..., 'unacc', 'unacc', 'acc'],

dtype=object)

In [32]:

print('Training-set accuracy score: {0:0.4f}'**.** format(accuracy\_score(y\_train, y\_pred\_train\_en)))

Training-set accuracy score: 0.7865

**Check for overfitting and underfitting**

In [33]:

*# print the scores on training and test set*

print('Training set score: {:.4f}'**.**format(clf\_en**.**score(X\_train, y\_train)))

print('Test set score: {:.4f}'**.**format(clf\_en**.**score(X\_test, y\_test)))

Training set score: 0.7865

Test set score: 0.8021

We can see that the training-set score and test-set score is same as above. The training-set accuracy score is 0.7865 while the test-set accuracy to be 0.8021. These two values are quite comparable. So, there is no sign of overfitting.

Now, based on the above analysis we can conclude that our classification model accuracy is very good. Our model is doing a very good job in terms of predicting the class labels.

But, it does not give the underlying distribution of values. Also, it does not tell anything about the type of errors our classifer is making.

We have another tool called Confusion matrix that comes to our rescue.

**15. Confusion matrix**

A confusion matrix is a tool for summarizing the performance of a classification algorithm. A confusion matrix will give us a clear picture of classification model performance and the types of errors produced by the model. It gives us a summary of correct and incorrect predictions broken down by each category. The summary is represented in a tabular form.

Four types of outcomes are possible while evaluating a classification model performance. These four outcomes are described below:-

**True Positives (TP)** – True Positives occur when we predict an observation belongs to a certain class and the observation actually belongs to that class.

**True Negatives (TN)** – True Negatives occur when we predict an observation does not belong to a certain class and the observation actually does not belong to that class.

**False Positives (FP)** – False Positives occur when we predict an observation belongs to a certain class but the observation actually does not belong to that class. This type of error is called **Type I error.**

**False Negatives (FN)** – False Negatives occur when we predict an observation does not belong to a certain class but the observation actually belongs to that class. This is a very serious error and it is called **Type II error.**

These four outcomes are summarized in a confusion matrix given below.

In [34]:

*# Print the Confusion Matrix and slice it into four pieces*

**from** sklearn.metrics **import** confusion\_matrix

cm **=** confusion\_matrix(y\_test, y\_pred\_en)

print('Confusion matrix\n\n', cm)

Confusion matrix

[[ 73 0 56 0]

[ 20 0 0 0]

[ 12 0 385 0]

[ 25 0 0 0]]

**16. Classification Report**

**Classification report** is another way to evaluate the classification model performance. It displays the **precision**, **recall**, **f1** and **support** scores for the model. I have described these terms in later.

We can print a classification report as follows:-

In [35]:

**from** sklearn.metrics **import** classification\_report

print(classification\_report(y\_test, y\_pred\_en))

precision recall f1-score support

acc 0.56 0.57 0.56 129

good 0.00 0.00 0.00 20

unacc 0.87 0.97 0.92 397

vgood 0.00 0.00 0.00 25

micro avg 0.80 0.80 0.80 571

macro avg 0.36 0.38 0.37 571

weighted avg 0.73 0.80 0.77 571

**17. Results and conclusion**

1. In this project, I build a Decision-Tree Classifier model to predict the safety of the car. I build two models, one with criterion gini index and another one with criterion entropy. The model yields a very good performance as indicated by the model accuracy in both the cases which was found to be 0.8021.
2. In the model with criterion gini index, the training-set accuracy score is 0.7865 while the test-set accuracy to be 0.8021. These two values are quite comparable. So, there is no sign of overfitting.
3. Similarly, in the model with criterion entropy, the training-set accuracy score is 0.7865 while the test-set accuracy to be 0.8021.We get the same values as in the case with criterion gini. So, there is no sign of overfitting.
4. In both the cases, the training-set and test-set accuracy score is the same. It may happen because of small dataset.
5. The confusion matrix and classification report yields very good model performance.