Decision Tree

A Decision Tree Classifier is a type of machine learning algorithm used for classification problems. It works by constructing a tree-like model of decisions and their possible consequences. The algorithm starts at the root of the tree and splits the data based on the most significant feature. The splitting process continues recursively until the data is split into homogeneous groups, called leaves. Each leaf represents a prediction class for the instances assigned to it. The final result is a tree of decisions and predictions, where the prediction for a new instance can be made by following the path from the root to a leaf, based on the values of its features.

Gini Index and Entropy are two measures used to determine the impurity of a set in the context of Decision Tree algorithms.

Gini Index measures the probability of a random sample being classified incorrectly if it were randomly labeled according to the class distribution in a group of examples. It ranges between 0 (perfectly pure) and 1 (perfectly impure).

Entropy is a measure of the amount of uncertainty or randomness in the set. It ranges between 0 (perfectly pure) and log2(n) (perfectly impure, where n is the number of classes).

Both Gini Index and Entropy are used to determine the best split point while building a decision tree. The split that results in the lowest impurity (highest homogeneity) is selected as the best split.

The formula for Gini Index is:

Gini(S) = 1 - Σ(p_i)^2

where p_i is the proportion of instances in class i in set S.

The formula for Entropy is:

Entropy(S) = - Σ(p_i * log2(p_i))

where p_i is the proportion of instances in class i in set S.

In [ ]:

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [ ]:

```
data = pd.read_csv('car_evaluation.csv')
```

In [2]:

```
data.head()
```

Out[2]:

	vhigh	vhigh.1	2	2.1	small	low	unacc
0	vhigh	vhigh	2	2	small	med	unacc
1	vhigh	vhigh	2	2	small	high	unacc
2	vhigh	vhigh	2	2	med	low	unacc
3	vhigh	vhigh	2	2	med	med	unacc
4	vhigh	vhigh	2	2	med	high	unacc

In [3]:

```
data.shape
```

Out[3]:

```
(1727, 7)
```

In [4]:

```
col_names = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class']
data.columns = col_names
```

In [5]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1727 entries, 0 to 1726
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   buying    1727 non-null   object
 1   maint     1727 non-null   object
 2   doors     1727 non-null   object
 3   persons   1727 non-null   object
 4   lug_boot  1727 non-null   object
 5   safety    1727 non-null   object
 6   class     1727 non-null   object
dtypes: object(7)
memory usage: 94.6+ KB
```

In [6]:

```
data.isna().any()
```

Out[6]:

```
buying      False
maint       False
doors       False
persons     False
lug_boot    False
safety      False
class       False
dtype: bool
```

In [7]:

```
data.dtypes
```

Out[7]:

```
buying      object
maint       object
doors       object
persons     object
lug_boot    object
safety      object
class       object
dtype: object
```

In [8]:

```
data.head()
```

Out[8]:

	buying	maint	doors	persons	lug_boot	safety	class
0	vhigh	vhigh	2	2	small	med	unacc
1	vhigh	vhigh	2	2	small	high	unacc
2	vhigh	vhigh	2	2	med	low	unacc
3	vhigh	vhigh	2	2	med	med	unacc
4	vhigh	vhigh	2	2	med	high	unacc

In [9]:

```
data.tail()
```

Out[9]:

	buying	maint	doors	persons	lug_boot	safety	class
1722	low	low	5more	more	med	med	good
1723	low	low	5more	more	med	high	vgood
1724	low	low	5more	more	big	low	unacc
1725	low	low	5more	more	big	med	good
1726	low	low	5more	more	big	high	vgood

In [10]:

```
x = data.drop(['class'], axis=1)
y = data['class']
```

In [11]:

```
data.isnull().any()
```

Out[11]:

```
buying      False
maint       False
doors       False
persons     False
lug_boot    False
safety      False
class       False
dtype: bool
```

In [12]:

```
data.dtypes
```

Out[12]:

```
buying      object
maint       object
doors       object
persons     object
lug_boot    object
safety      object
class       object
dtype: object
```

In [13]:

```
x.head()
```

Out[13]:

	buying	maint	doors	persons	lug_boot	safety
0	vhigh	vhigh	2	2	small	med
1	vhigh	vhigh	2	2	small	high
2	vhigh	vhigh	2	2	med	low
3	vhigh	vhigh	2	2	med	med
4	vhigh	vhigh	2	2	med	high

This code imports the OrdinalEncoder class from the category_encoders library and applies it to a Pandas DataFrame named x. The OrdinalEncoder class is used to encode categorical features in the DataFrame into ordinal (integer) values.

The cols argument specifies the columns in the DataFrame that should be encoded, and in this case it is a list of column names ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety']. The fit_transform method fits the encoder to the data and returns the encoded DataFrame, which can then be used as input to machine learning models.

In [14]:

```
import category_encoders as ce
encoder = ce.OrdinalEncoder(cols=['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety'])
x = encoder.fit_transform(x)
```

In [15]:

```
pip install category_encoders
```

```
Requirement already satisfied: category_encoders in c:\users\user\anaconda3\lib\site-packages (2.6.0)
Requirement already satisfied: scipy>=1.0.0 in c:\users\user\anaconda3\lib\site-packages (from category_encoders) (1.7.1)
Requirement already satisfied: scikit-learn>=0.20.0 in c:\users\user\anaconda3\lib\site-packages (from category_encoders) (0.24.2)
Requirement already satisfied: patsy>=0.5.1 in c:\users\user\anaconda3\lib\site-packages (from category_encoders) (0.5.2)
Requirement already satisfied: statsmodels>=0.9.0 in c:\users\user\anaconda3\lib\site-packages (from category_encoders) (0.12.2)
Requirement already satisfied: pandas>=1.0.5 in c:\users\user\anaconda3\lib\site-packages (from category_encoders) (1.3.4)
Requirement already satisfied: numpy>=1.14.0 in c:\users\user\anaconda3\lib\site-packages (from category_encoders) (1.22.4)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\user\anaconda3\lib\site-packages (from pandas>=1.0.5->category_encoders) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in c:\users\user\anaconda3\lib\site-packages (from pandas>=1.0.5->category_encoders) (2021.3)
Requirement already satisfied: six in c:\users\user\anaconda3\lib\site-packages (from patsy>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: joblib>=0.11 in c:\users\user\anaconda3\lib\site-packages (from scikit-learn>=0.20.0->category_encoders) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\user\anaconda3\lib\site-packages (from scikit-learn>=0.20.0->category_encoders) (2.2.0)
Note: you may need to restart the kernel to use updated packages.
```

In [16]:

```
x.head()
```

Out[16]:

	buying	maint	doors	persons	lug_boot	safety
0	1	1	1	1	1	1
1	1	1	1	1	1	2
2	1	1	1	1	2	3
3	1	1	1	1	2	1
4	1	1	1	1	2	2

This code imports the LabelEncoder class from the sklearn.preprocessing module and applies it to a target variable y. The LabelEncoder is used to encode categorical target variables into numerical values.

The fit_transform method fits the label encoder to the target variable y and returns the encoded values. After fitting, the label encoder can be used to transform new, unseen data into the same encoding, by calling the transform method.

In [17]:

```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
label_encoder.fit_transform(y)
```

Out[17]:

```
array([2, 2, 2, ..., 2, 1, 3])
```

In [ ]:

In [ ]:

In [ ]:

In [18]:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.30, random_state = 42)
```

This code imports the DecisionTreeClassifier class from the sklearn.tree module and creates an instance of the classifier. The created classifier is a decision tree model for binary or multi-class classification.

The classifier is initialized with the following arguments:

criterion: The impurity measure to use while building the tree. In this case, it is set to 'entropy', meaning that the entropy impurity measure will be used.

max_depth: The maximum depth of the tree. In this case, it is set to 10, meaning that the tree can have at most 10 levels.

random_state: The random seed used for the random number generator. In this case, it is set to 0.

Finally, the classifier is fitted to the training data (x_train, y_train) using the fit method. The decision tree model is built based on the training data and the provided impurity measure and other parameters.

In [19]:

```
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(criterion='entropy', max_depth=10, random_state=0)

clf.fit(x_train, y_train)
```

Out[19]:

```
DecisionTreeClassifier(criterion='entropy', max_depth=10, random_state=0)
```

In [20]:

```
clf.score(x_test, y_test)
```

Out[20]:

```
0.9595375722543352
```

In [ ]:

In [21]:

```
from sklearn.model_selection import StratifiedKFold
```

In [ ]:

In [22]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score, make_scorer, confusion_matrix
model = DecisionTreeClassifier(criterion='entropy', max_depth=10, random_state=0)
fold = StratifiedKFold(n_splits=10, shuffle=True)
lst_accu_stratified = []
for train_index, test_index in fold.split(x, y):
    x_train_fold, x_test_fold = x.iloc[train_index], x.iloc[test_index]
    y_train_fold, y_test_fold = y[train_index], y[test_index]
    model.fit(x_train_fold, y_train_fold)
    lst_accu_stratified.append(model.score(x_test_fold, y_test_fold))
    y_pre = model.predict(x_test_fold)
    cd = confusion_matrix(y_test_fold, y_pre)
lst_accu_stratified
```

Out[22]:

```
[0.9826589595375722,
 0.9710982658959537,
 0.9826589595375722,
 0.9710982658959537,
 0.9826589595375722,
 0.9653179190751445,
 0.9710982658959537,
 0.9709302325581395,
 0.9651162790697675,
 0.9651162790697675]
```

In [23]:

```
np.average(lst_accu_stratified)
```

Out[23]:

```
0.9727752386073398
```

In [ ]:

In [24]:

```
def classification_report_with_accuracy_score(y_true, y_pred):

    print(classification_report(y_true, y_pred))
    return accuracy_score(y_true, y_pred)
```

In [25]:

```
print(classification_report(y_test_fold, y_pre))
```

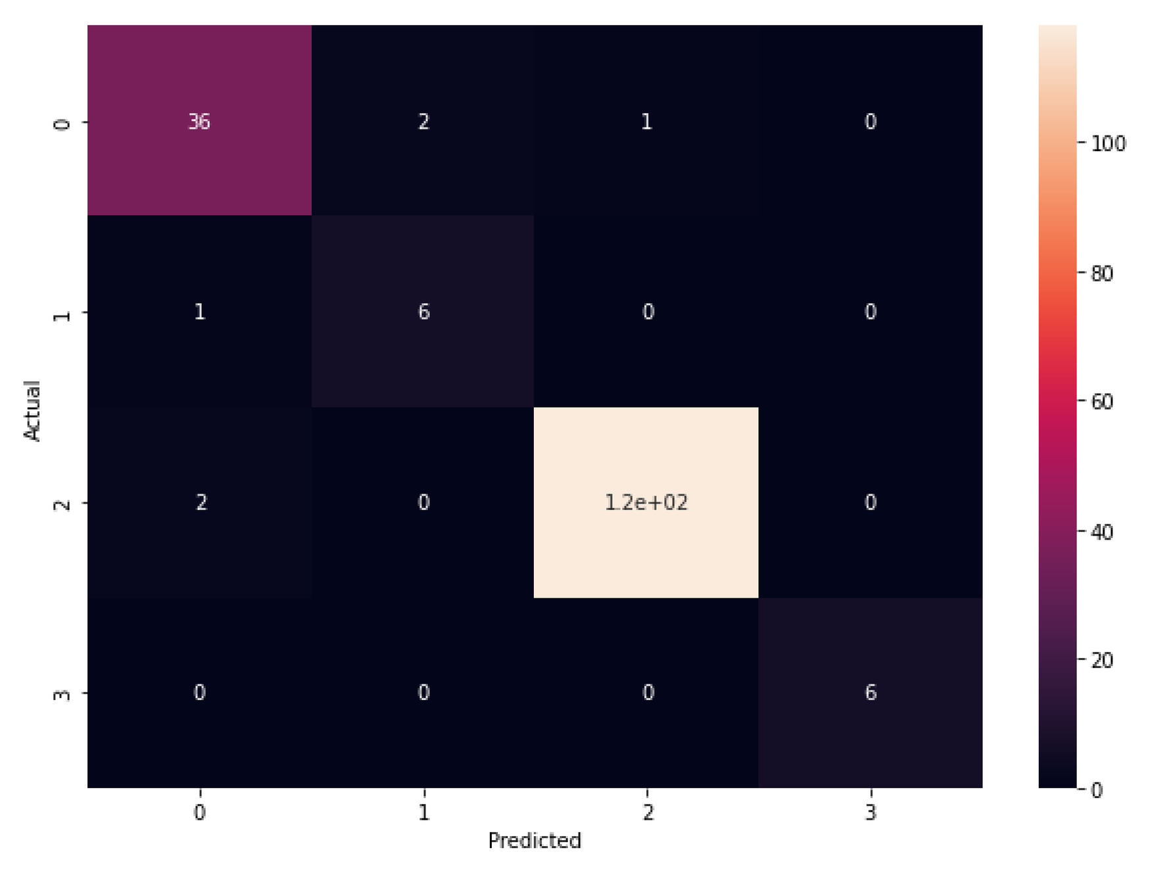
```
              precision    recall  f1-score   support

         acc       0.92      0.92      0.92        39
        good       0.75      0.86      0.80         7
       unacc       0.99      0.98      0.99       120
       vgood       1.00      1.00      1.00         6

    accuracy                           0.97       172
   macro avg       0.92      0.94      0.93       172
weighted avg       0.97      0.97      0.97       172
```

In [26]:

```
import seaborn as sn
plt.figure(figsize = (10, 7))
sn.heatmap(cd, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Actual')
```

Out[26]:

```
Text(69.0, 0.5, 'Actual')
```



In [ ]:

In [ ]:

In [ ]:

In [ ]: