ML 23 - Decision Tree Classifier With Post-Prunning By Virat Tiwari

December 13, 2023

1 Decision Tree Classifier With Post Prunning By Virat Tiwari

[1]:

```
# Here we import some important libraries that useful for visualization and for␣
  ↪imoprting pre built datasets

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

[2]:

```
# We import " iris " dataset from sklearn

from sklearn.datasets import load_iris
```

[3]:

```
# we store iris dataset in variable " dataset "

dataset=load_iris()
```

[4]:

```
# print( dataset . DESCR ) function is used for understand the description of␣
  ↪dataset
# . DESCR fucntion describe about the dataset

print(dataset.DESCR)
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica
```

:Summary Statistics:

	Min	Max	Mean	SD	Class Correlation
sepal length:	4.3	7.9	5.84	0.83	0.7826
sepal width:	2.0	4.4	3.05	0.43	-0.4194
petal length:	1.0	6.9	3.76	1.76	0.9490 (high!)
petal width:	0.1	2.5	1.20	0.76	0.9565 (high!)

:Missing Attribute Values: None
:Class Distribution: 33.3% for each of 3 classes.
:Creator: R.A. Fisher
:Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
:Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken from Fisher's paper. Note that it's the same as in R, but not as in the UCI Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the pattern recognition literature. Fisher's paper is a classic in the field and is referenced frequently to this day. (See Duda & Hart, for example.) The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.

.. topic:: References

- Fisher, R.A. "The use of multiple measurements in taxonomic problems" Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to Mathematical Statistics" (John Wiley, NY, 1950).
- Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis. (Q327.D83) John Wiley & Sons. ISBN 0-471-22361-1. See page 218.
- Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System Structure and Classification Rule for Recognition in Partially Exposed Environments". IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-2, No. 1, 67-71.

```
  - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
    on Information Theory, May 1972, 431-433.
  - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
    conceptual clustering system finds 3 classes in the data.
  - Many, many more …
```

[5]:
```
# Another way to import same dataset using seaborn
# we can also import the " iris " through the seaborn library

df=sns.load_dataset("iris")
```

[6]:
```
# . head ( ) function gives the initial 5 datapoints from the entire dataset

df.head()
```

[6]:
```
   sepal_length  sepal_width  petal_length  petal_width species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa
```

[7]:
```
# . target function is used for understanding the dataset more clearly in the␣
 ↪form of 0's and 1's and so on

dataset.target
```

[7]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

[8]:
```
# Independent and dependent features
# . iloc [ ] fucntion is used for defing the dataset that how much we want␣
 ↪independent data or dependent data

x=df.iloc[:,:-1]
y=dataset.target
```

[9]:
```
x
```

[9]:
```
     sepal_length  sepal_width  petal_length  petal_width
0             5.1          3.5           1.4          0.2
1             4.9          3.0           1.4          0.2
```

```
2         4.7       3.2       1.3       0.2
3         4.6       3.1       1.5       0.2
4         5.0       3.6       1.4       0.2
..        …       …       …       …
145       6.7       3.0       5.2       2.3
146       6.3       2.5       5.0       1.9
147       6.5       3.0       5.2       2.0
148       6.2       3.4       5.4       2.3
149       5.9       3.0       5.1       1.8

[150 rows x 4 columns]
```

[10]:
```
y
```

[10]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

[11]:
```
# Train Test Split
# import train_test_split function Train and Test the dataset
# train_test_split(x,y,test_size=0.33,random_state=42) - We pass x , y , test␣
 ↪size and random state for accuracy inseide the train_test_split function
# This function require four parameters - x_train,x_test,y_train,y_test

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)
```

[12]:
```
# Here we import DecisionTreeClassifier
# sklearn provide built in function for decision tree

from sklearn.tree import DecisionTreeClassifier
```

[13]:
```
# Post Prunning - It Use for reduce the Overfitting
# We store the DecisionTreeClassifier in variable " treeclassifier "

treeclassifier=DecisionTreeClassifier()
```

[14]:
```
# Now finally we fit the x_train and y_train in treeclassifier (␣
 ↪DecisionTreeClassifier)

treeclassifier.fit(x_train,y_train)
```

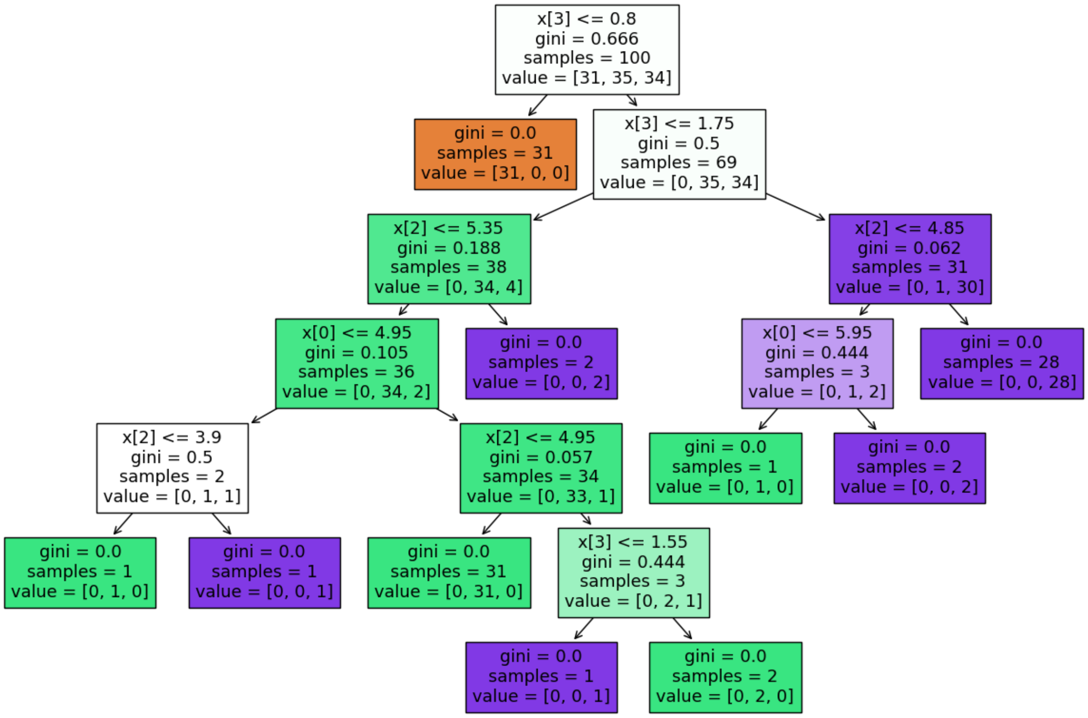
```
[14]: DecisionTreeClassifier()
```

```
[15]: # Here we see the x_train data after fit in DecisionTreeClassifier

      x_train.head()
```

```
[15]:      sepal_length  sepal_width  petal_length  petal_width
      96            5.7          2.9           4.2          1.3
      105           7.6          3.0           6.6          2.1
      66            5.6          3.0           4.5          1.5
      0             5.1          3.5           1.4          0.2
      122           7.7          2.8           6.7          2.0
```

```
[16]: # We import " tree " from sklearn for making the tree in graphical form
      # tree.plot_tree ( ) function is used for visualing the data in Decision tree␣
       ↪format

      from sklearn import tree
      plt.figure(figsize=(15,10))
      tree.plot_tree(treeclassifier,filled=True)
```

```
[16]: [Text(0.5416666666666666, 0.9285714285714286, 'x[3] <= 0.8\ngini =
      0.666\nsamples = 100\nvalue = [31, 35, 34]'),
       Text(0.4583333333333333, 0.7857142857142857, 'gini = 0.0\nsamples = 31\nvalue =
      [31, 0, 0]'),
       Text(0.625, 0.7857142857142857, 'x[3] <= 1.75\ngini = 0.5\nsamples = 69\nvalue
      = [0, 35, 34]'),
       Text(0.4166666666666667, 0.6428571428571429, 'x[2] <= 5.35\ngini =
      0.188\nsamples = 38\nvalue = [0, 34, 4]'),
       Text(0.3333333333333333, 0.5, 'x[0] <= 4.95\ngini = 0.105\nsamples = 36\nvalue
      = [0, 34, 2]'),
       Text(0.16666666666666666, 0.35714285714285715, 'x[2] <= 3.9\ngini =
      0.5\nsamples = 2\nvalue = [0, 1, 1]'),
       Text(0.08333333333333333, 0.21428571428571427, 'gini = 0.0\nsamples = 1\nvalue
      = [0, 1, 0]'),
       Text(0.25, 0.21428571428571427, 'gini = 0.0\nsamples = 1\nvalue = [0, 0, 1]'),
       Text(0.5, 0.35714285714285715, 'x[2] <= 4.95\ngini = 0.057\nsamples = 34\nvalue
      = [0, 33, 1]'),
       Text(0.4166666666666667, 0.21428571428571427, 'gini = 0.0\nsamples = 31\nvalue
      = [0, 31, 0]'),
       Text(0.5833333333333334, 0.21428571428571427, 'x[3] <= 1.55\ngini =
      0.444\nsamples = 3\nvalue = [0, 2, 1]'),
       Text(0.5, 0.07142857142857142, 'gini = 0.0\nsamples = 1\nvalue = [0, 0, 1]'),
       Text(0.6666666666666666, 0.07142857142857142, 'gini = 0.0\nsamples = 2\nvalue =
      [0, 2, 0]'),
       Text(0.5, 0.5, 'gini = 0.0\nsamples = 2\nvalue = [0, 0, 2]'),
       Text(0.8333333333333334, 0.6428571428571429, 'x[2] <= 4.85\ngini =
      0.062\nsamples = 31\nvalue = [0, 1, 30]'),
       Text(0.75, 0.5, 'x[0] <= 5.95\ngini = 0.444\nsamples = 3\nvalue = [0, 1, 2]'),
       Text(0.6666666666666666, 0.35714285714285715, 'gini = 0.0\nsamples = 1\nvalue =
      [0, 1, 0]'),
       Text(0.8333333333333334, 0.35714285714285715, 'gini = 0.0\nsamples = 2\nvalue =
      [0, 0, 2]'),
       Text(0.9166666666666666, 0.5, 'gini = 0.0\nsamples = 28\nvalue = [0, 0, 28]')]
```

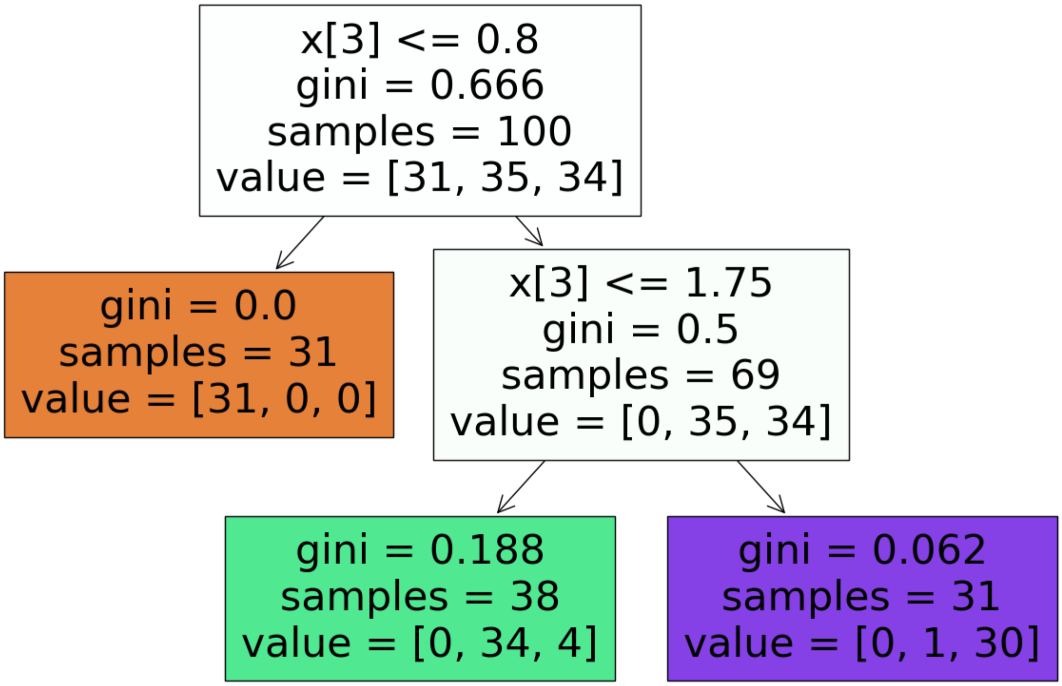


[17]:
```
# Post Prunning - It Use for reduce the Overfitting
# Here we pass " max_depth = 2 " for cutting the tree and for getting the tree␣
 ↪from the second node only

treeclassifier=DecisionTreeClassifier(max_depth=2)
treeclassifier.fit(x_train,y_train)
```

[17]:
```
DecisionTreeClassifier(max_depth=2)
```

[18]:
```
# Now we visualise the dataset after applying the tunning or cutting on the tree

from sklearn import tree
plt.figure(figsize=(15,10))
tree.plot_tree(treeclassifier,filled=True)
```

[18]:
```
[Text(0.4, 0.8333333333333334, 'x[3] <= 0.8\ngini = 0.666\nsamples = 100\nvalue
= [31, 35, 34]'),
 Text(0.2, 0.5, 'gini = 0.0\nsamples = 31\nvalue = [31, 0, 0]'),
 Text(0.6, 0.5, 'x[3] <= 1.75\ngini = 0.5\nsamples = 69\nvalue = [0, 35, 34]'),
 Text(0.4, 0.16666666666666666, 'gini = 0.188\nsamples = 38\nvalue = [0, 34,
4]'),
 Text(0.8, 0.16666666666666666, 'gini = 0.062\nsamples = 31\nvalue = [0, 1,
30]')]
```



[19]:
```
# PREDICTION
# Now we predict the x_test through the . predict ( ) function

y_pred=treeclassifier.predict(x_test)
```

[20]:
```
y_pred
```

[20]:
```
array([1, 0, 2, 1, 1, 0, 1, 2, 1, 1, 2, 0, 0, 0, 0, 1, 2, 1, 1, 2, 0, 2,
       0, 2, 2, 2, 2, 2, 0, 0, 0, 0, 1, 0, 0, 2, 1, 0, 0, 0, 2, 1, 1, 0,
       0, 1, 1, 2, 1, 2])
```

[21]:
```
# Now we import accuracy_score , classification_report for getting the accuracy␣
 ↪or classification report of the dataset
```

```
from sklearn.metrics import accuracy_score,classification_report
```

[22]:

```
# accuracy_score(y_pred,y_test) - here we pass y_pred or y_test for getting the␣
 ↪accuracy

score=accuracy_score(y_pred,y_test)
print(score)
```

```
0.98
```

We get 98 % accuracy rate

[23]:

```
# Same thing we done here in case of " classification_report "

print(classification_report(y_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      0.94      0.97        16
           2       0.94      1.00      0.97        15

    accuracy                           0.98        50
   macro avg       0.98      0.98      0.98        50
weighted avg       0.98      0.98      0.98        50
```

Here we get classification Report

NOTE - IN DECISION TREE WE DON'T NEED TO DO FEATURE SCALING , STANDARDIZATION , NORMALISATION

NOTE - WE DONT USE " STANDARDIZATION " IN DECISION TREE ALGORITHM BECOUSE IT REDUCE THE DATASET SIZE AS PER THE STANDATRDIZATION OPERATION BUT IT DOEN'T REDUCE THE TIME COMPLEXITY AS MUCH AS WE WANT

THANK YOU SO MUCH !!

YOURS VIRAT TIWARI :)