Sagar Pathak

Mandar Shivaji Hanchate

https://colab.research.google.com/drive/11wZGACVknc14B-veqi7bdmEZk6PfeQ3K?usp=sharing

Q1 >>The Iris Dataset contains four features i.e. length and width of sepals and petals (in cm) and these features are of three species of Iris i.e Iris setosa, Iris virginica and Iris versicolor. Base on this length and width of sepals and petals, this dataset tells which specises is this of Iris.

+ Code + Text

Dataset Loading and Making more presentable

```
# Loading the Iris Dataset

from sklearn.datasets import load_iris

import numpy as np
import pandas as pd

iris = load_iris()
iris
```

```
{'DESCR': '.. _iris_dataset:\n\nIris plants dataset\n--------------------\n\n**Data S
 'data': array([[5.1, 3.5, 1.4, 0.2],
        [4.9, 3. , 1.4, 0.2],
        [4.7, 3.2, 1.3, 0.2],
        [4.6, 3.1, 1.5, 0.2],
        [5. , 3.6, 1.4, 0.2],
        [5.4, 3.9, 1.7, 0.4],
        [4.6, 3.4, 1.4, 0.3],
        [5. , 3.4, 1.5, 0.2],
        [4.4, 2.9, 1.4, 0.2],
        [4.9, 3.1, 1.5, 0.1],
```

```
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
       [5. , 3. , 1.6, 0.2],
       [5. , 3.4, 1.6, 0.4],
       [5.2, 3.5, 1.5, 0.2],
       [5.2, 3.4, 1.4, 0.2],
       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
       [5.4, 3.4, 1.5, 0.4],
       [5.2, 4.1, 1.5, 0.1],
       [5.5, 4.2, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.2],
       [5. , 3.2, 1.2, 0.2],
       [5.5, 3.5, 1.3, 0.2],
       [4.9, 3.6, 1.4, 0.1],
       [4.4, 3. , 1.3, 0.2],
       [5.1, 3.4, 1.5, 0.2],
       [5. , 3.5, 1.3, 0.3],
       [4.5, 2.3, 1.3, 0.3],
       [4.4, 3.2, 1.3, 0.2],
       [5. , 3.5, 1.6, 0.6],
       [5.1, 3.8, 1.9, 0.4],
       [4.8, 3. , 1.4, 0.3],
       [5.1, 3.8, 1.6, 0.2],
       [4.6, 3.2, 1.4, 0.2],
       [5.3, 3.7, 1.5, 0.2],
       [5. , 3.3, 1.4, 0.2],
       [7. , 3.2, 4.7, 1.4],
       [6.4, 3.2, 4.5, 1.5],
       [6.9, 3.1, 4.9, 1.5],
       [5.5, 2.3, 4. , 1.3],
       [6.5, 2.8, 4.6, 1.5],
```

```
X = iris.data
Y = iris.target
```

```
X
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
```

```
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
       [5. , 3. , 1.6, 0.2],
       [5. , 3.4, 1.6, 0.4],
       [5.2, 3.5, 1.5, 0.2],
       [5.2, 3.4, 1.4, 0.2],
       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
       [5.4, 3.4, 1.5, 0.4],
       [5.2, 4.1, 1.5, 0.1],
       [5.5, 4.2, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.2],
       [5. , 3.2, 1.2, 0.2],
       [5.5, 3.5, 1.3, 0.2],
       [4.9, 3.6, 1.4, 0.1],
       [4.4, 3. , 1.3, 0.2],
       [5.1, 3.4, 1.5, 0.2],
       [5. , 3.5, 1.3, 0.3],
       [4.5, 2.3, 1.3, 0.3],
       [4.4, 3.2, 1.3, 0.2],
       [5. , 3.5, 1.6, 0.6],
       [5.1, 3.8, 1.9, 0.4],
       [4.8, 3. , 1.4, 0.3],
       [5.1, 3.8, 1.6, 0.2],
       [4.6, 3.2, 1.4, 0.2],
       [5.3, 3.7, 1.5, 0.2],
       [5. , 3.3, 1.4, 0.2],
       [7. , 3.2, 4.7, 1.4],
       [6.4, 3.2, 4.5, 1.5],
       [6.9, 3.1, 4.9, 1.5],
       [5.5, 2.3, 4. , 1.3],
       [6.5, 2.8, 4.6, 1.5],
       [5.7, 2.8, 4.5, 1.3],
       [6.3, 3.3, 4.7, 1.6],
       [4.9, 2.4, 3.3, 1. ],
```

```
Y
```

```
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])
```

```
column_names = iris.feature_names
column_names
```

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

```
Target_Column_Name = iris.target_names

Target_Column_Name
```

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

```
#Making Data is ready with column names

Data = pd.DataFrame(X, columns=column_names)

Data['species'] = Y

Data
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0

Q2 >> Here in this problem, Base on this length and width of sepals and petals, we have to tell that specises is blongs to which catagory i.e. 'setosa', 'versicolor', 'virginica' so output variable/dependent variable/label is catagorical and we need to classifiy each row of the dataset into 3 class so this is classsification problem. Hence we have to use logistic regression or decision tree. I am prefering deciosn tree as this dataset has very less no of features so decision tree for this dataset will be easily understandable for non-technical person.

Q3 >> Splitting The dataset in Train and Test

as we don't have enough rows in the dataset we will be performing cross validation and we will not be having validation dataset

```
X = Data.iloc[:,[0,1,2,3]]
```

```
Y = Data.iloc[:,4]
```

```
from sklearn.model_selection import train_test_split
```

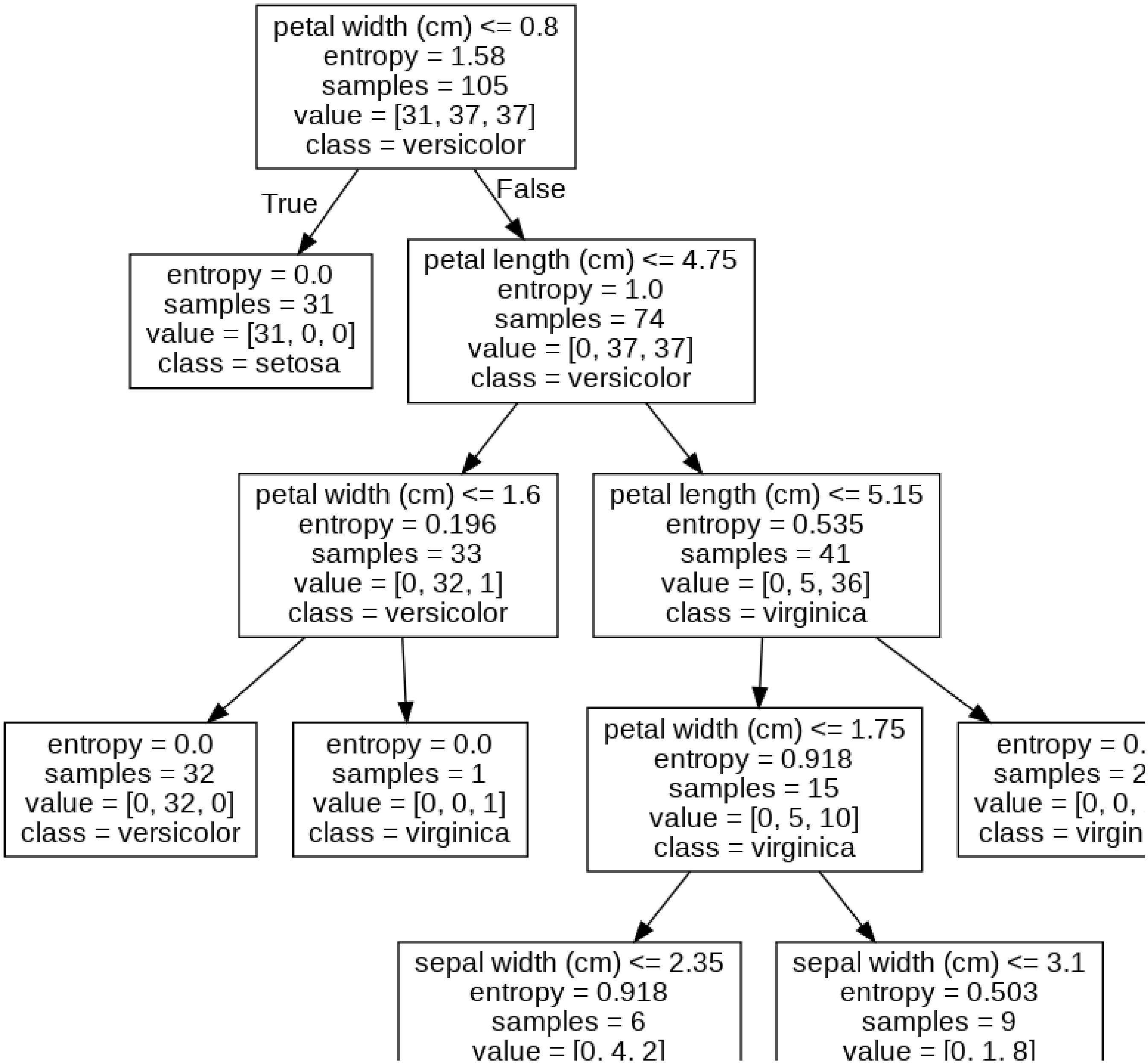
```
X_train,X_test,Y_train,Y_test = train_test_split( X , Y , test_size = 0.3 , random_state = 42
```

```
X_train.shape,X_test.shape,Y_train.shape,Y_test.shape
```

```
((105, 4), (45, 4), (105,), (45,))
```

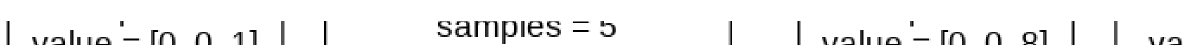
Q4 >> Making Decision Tree Model and fitting Train Dataset

```
# Making DecisionTree the Model without any hyperparameters

from sklearn.tree import DecisionTreeClassifier

D_Tree_Object = DecisionTreeClassifier(random_state=0, criterion ="entropy")

D_Tree_Model = D_Tree_Object.fit(X_train,Y_train)


# Plotting the Model


from sklearn.tree import export_graphviz
from IPython.display import Image

# export_graphviz create image in ".dot" format

export_graphviz( D_Tree_Model, out_file = 'Retree.dot', feature_names = X_train.columns, clas

# converting ".dot" format into ".png" format

! dot -Tpng Retree.dot -o Retree.png

# display the image

Image("Retree.png")
```




```
# Here we used all the Features

D_Tree_Model.max_features_
```

```
4
```




```
# Depth of our Model

print(D_Tree_Model.tree_.max_depth)
```

```
7
```


```
# Accuracy we got for Train Dataset

D_Tree_Model.score(X_train,Y_train)
```

```
1.0
```

samples = 1 samples = 1

Q5 >> (a) Doing cross validation with k = 5 and We are tuning 2 hyperparameters max_depth and max_feature

max_depth = It is used to decide How long/deep tree you want.

max_feature = How many features you want while making the tree

```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
import random

random.seed(10)

D_Tree_Object_1 = DecisionTreeClassifier(random_state=0,criterion ="entropy")

parameters = { 'max_depth' : [1,2,3,4,5,6,7,8,9,10], 'max_features':[1,2,3,4]}

cross_Valid_model = GridSearchCV(D_Tree_Object_1, parameters, cv = 5)


# cross Validation on Train dataset >>

cross_Valid_model.fit(X_train,Y_train)

# we got best Accuracy for below model where 'max_depth': 6, 'max_features': 2
cross_Valid_model.best_params_
```

```
{'max_depth': 3, 'max_features': 3}
```

```
# Entire Result of clasification problem

cross_Valid_model.cv_results_
```

```
{'mean_fit_time': array([0.00379691, 0.00292611, 0.00307202, 0.00392017, 0.0024127 ,
        0.00283904, 0.00285854, 0.00384727, 0.00357199, 0.00306945,
        0.00249386, 0.00308833, 0.00258346, 0.00230408, 0.00270591,
        0.00238233, 0.00239449, 0.00259428, 0.0023706 , 0.00243931,
        0.00262375, 0.00234756, 0.00244956, 0.00257382, 0.00233192,
        0.00367432, 0.00292902, 0.00235643, 0.0024034 , 0.00245337,
        0.00259399, 0.00230174, 0.00235481, 0.00322738, 0.0025713 ,
        0.00251946, 0.00233622, 0.00229301, 0.00234632, 0.0023983 ]),
 'mean_score_time': array([0.00215349, 0.00203967, 0.00174766, 0.00238252, 0.00148396
        0.00180182, 0.00153179, 0.00253348, 0.00214634, 0.00190673,
        0.00155115, 0.00205026, 0.00151157, 0.00162411, 0.00200806,
        0.00147405, 0.00149126, 0.00159326, 0.00144691, 0.00155315,
        0.00183496, 0.00145469, 0.00148549, 0.00151405, 0.00154614,
```

```
        0.00233774, 0.00176678, 0.00145483, 0.00154352, 0.00155082,
        0.00151291, 0.00145273, 0.00144777, 0.00193424, 0.00153437,
        0.0015214 , 0.00160456, 0.00143723, 0.00146236, 0.00148358]),
 'mean_test_score': array([0.51428571, 0.62857143, 0.62857143, 0.62857143, 0.71428571
        0.92380952, 0.91428571, 0.91428571, 0.78095238, 0.92380952,
        0.93333333, 0.93333333, 0.82857143, 0.91428571, 0.92380952,
        0.92380952, 0.87619048, 0.91428571, 0.91428571, 0.92380952,
        0.88571429, 0.91428571, 0.92380952, 0.93333333, 0.9047619 ,
        0.91428571, 0.92380952, 0.92380952, 0.91428571, 0.91428571,
        0.92380952, 0.93333333, 0.91428571, 0.91428571, 0.92380952,
        0.93333333, 0.91428571, 0.91428571, 0.92380952, 0.93333333]),
 'param_max_depth': masked_array(data=[1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4
                    5, 5, 6, 6, 6, 6, 7, 7, 7, 7, 8, 8, 8, 8, 9, 9, 9, 9,
                    10, 10, 10, 10],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_max_features': masked_array(data=[1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3
                    3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4,
                    1, 2, 3, 4],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'max_depth': 1, 'max_features': 1},
  {'max_depth': 1, 'max_features': 2},
  {'max_depth': 1, 'max_features': 3},
  {'max_depth': 1, 'max_features': 4},
  {'max_depth': 2, 'max_features': 1},
  {'max_depth': 2, 'max_features': 2},
  {'max_depth': 2, 'max_features': 3},
  {'max_depth': 2, 'max_features': 4},
  {'max_depth': 3, 'max_features': 1},
  {'max_depth': 3, 'max_features': 2},
  {'max_depth': 3, 'max_features': 3},
  {'max_depth': 3, 'max_features': 4},
  {'max_depth': 4, 'max_features': 1},
```

```
# During Cross Validation we got best Accuracy i.e. 0.9333333333333333 and It is for {'max_de
```

```
cross_Valid_model.best_score_
```

```
0.9333333333333333
```

```
depth=[]
feature=[]
```

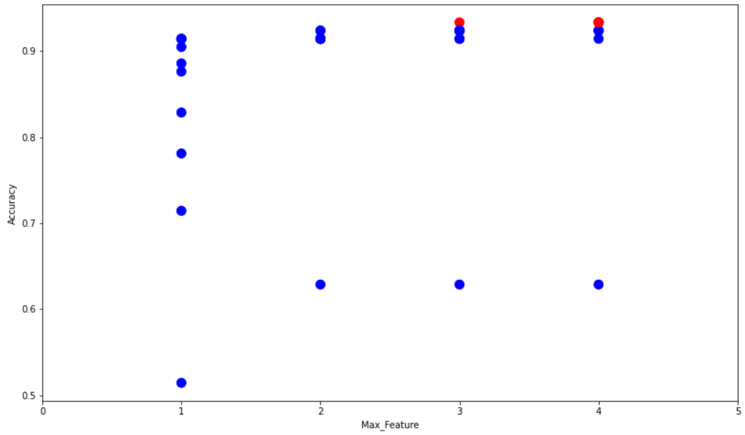
```
Score=[]

for i in cross_Valid_model.cv_results_['params']:

  depth.append(i['max_depth'])
  feature.append(i['max_features'])


for i in cross_Valid_model.cv_results_['mean_test_score']:

  Score.append(i)
```

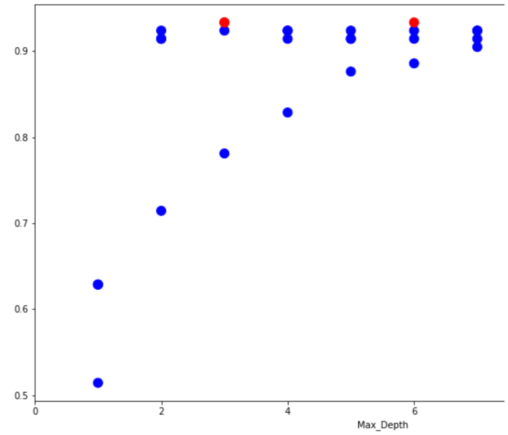
Q5 >> (B) Generate plot of hyperparameter values w.r.t performance metric

Plot of Max_Depth VS Accuracy and Max_Feature VS Accuracy

```
import matplotlib.pyplot as plt

color = ['red' if i == 0.9333333333333333 else 'blue' for i in Score]

plt.figure(figsize = (30 , 8))

ax1 = plt.subplot(1,2,1)
ax1.set_xlabel('Max_Feature')
ax1.set_ylabel('Accuracy')

plt.scatter( feature, Score , c = color, s= 100)

ax2 = plt.subplot(1,2,2, sharey=ax1)

ax2.set_xlabel('Max_Depth')

plt.scatter( depth, Score, c = color, s= 100)

ax1.set_xlim(left=0, right=5)
ax2.set_xlim(left=0, right=11)
```

```
(0.0, 11.0)
```

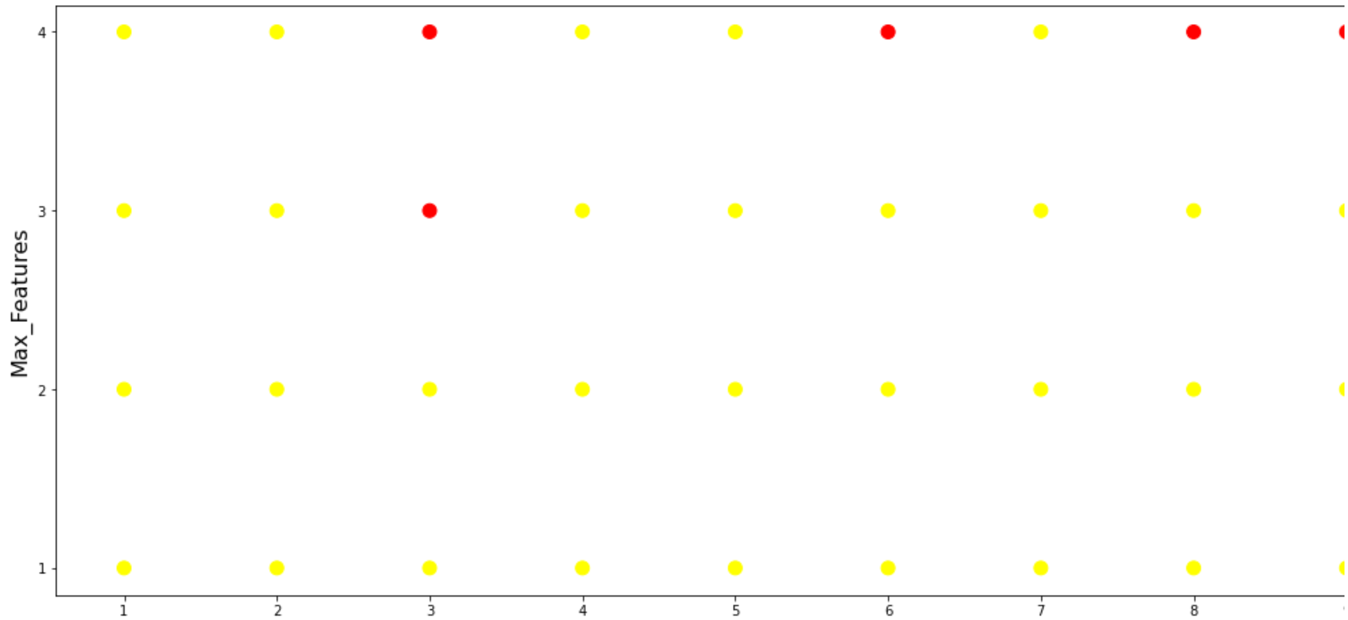




Plot of Max_Depth, Max_Feature and Accuracy (as data points color)

```
import matplotlib.pyplot as plt

%matplotlib inline

color = ['red' if i == 0.9333333333333333 else 'yellow' for i in Score]

plt.figure( figsize = (20 , 8) )

plt.yticks(np.arange(0, 5, 1))
plt.ylabel('Max_Features',size=16)
plt.xticks(np.arange(0, 11, 1))
plt.xlabel('Max_Depth',size=16)

plt.scatter( depth, feature , c = color, s= 100)
```

```
<matplotlib.collections.PathCollection at 0x7f8845acd190>
```

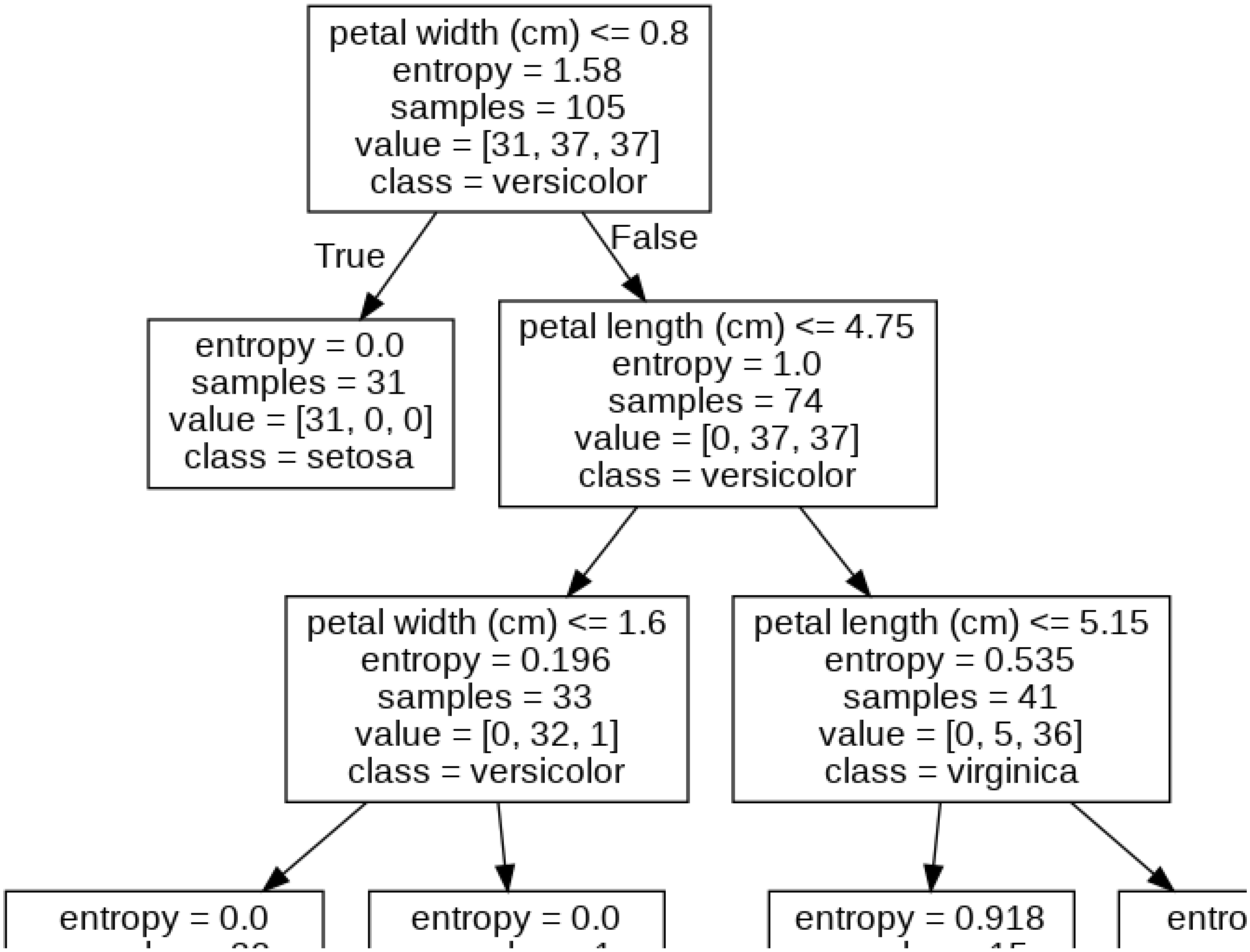


Making Decsion Tree with 'max_depth': 3, 'max_features': 3

```
# Making Decsion Tree with 'max_depth': 3, 'max_features': 3

D_Tree_Object_after_cv = DecisionTreeClassifier(max_depth=3,max_features=3,random_state=0,cri

D_Tree_Model_after_cv = D_Tree_Object_after_cv.fit(X_train,Y_train)

D_Tree_Model_after_cv.score(X_train,Y_train)
```

```
0.9523809523809523
```

```
from sklearn.tree import export_graphviz
from IPython.display import Image

# export_graphviz create image in ".dot" format

export_graphviz( D_Tree_Model_after_cv, out_file = 'Retree.dot', feature_names = X_train.colu

# converting ".dot" format into ".png" format

! dot -Tpng Retree.dot -o Retree.png

# display the image

Image("Retree.png")
```



Q6 >> Evaluating Desicion Tree model (With cross validation Tuning) on Test dataset and generating Classification Report

```
#Test_DataSet Accuracy with old Desicion model (With cross validation tunning)

from sklearn.metrics import accuracy_score

Y_predict_test_after_CV = D_Tree_Object_after_cv.predict(X_test)

accuracy_score(Y_test , Y_predict_test_after_CV)
```

```
0.9777777777777777
```

```
from sklearn.metrics import classification_report

print(classification_report(Y_test,Y_predict_test_after_CV))
```

```
              precision    recall  f1-score   support
```

```
              0       1.00      1.00      1.00        19
              1       1.00      0.92      0.96        13
              2       0.93      1.00      0.96        13

       accuracy                           0.98        45
      macro avg       0.98      0.97      0.97        45
   weighted avg       0.98      0.98      0.98        45
```

Q7 >>

Observations

1. When we built the model without feeding any hyperparameter then machine learning algorithm made decision tree using all 4 feature and depth was 7 and for this gave us 100% accuracy for train dataset.

2. Later we did cross validation and in that process we got max_depth as 3, max_features as 3 then we built the model again using max_depth as 3, max_features as 3 , we got 95% accuracy for train dataset.

why we should say that decision tree with max_depth as 3, max_features as 3 is good becuase it is less complex and hence has very less chances to be overfit.

3. We checked accuracy of new model on test dataset and we got 98% accuracy.