HW3_decision_tree_teamwork

February 13, 2022

1 HW3

1.1 Decision Trees

1.1.1 Team members

Team 3

- Anjali Sebastian
- Rupansh Phutela
- Yesha Sharma

1.1.2 Setup

First, let's import a few common modules, ensure MatplotLib plots figures inline and prepare a function to save the figures. Let's check that Python 3.5 or later is installed, as well as Scikit-Learn 0.20.

[1]:
```
# Python 3.5 is required
import sys
assert sys.version_info >= (3, 5)

# Scikit-Learn 0.20 is required
import sklearn
assert sklearn.__version__ >= "0.20"

# Common imports
import numpy as np
import os
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

# To plot pretty figures
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)
```

```
# to make this notebook's output stable across runs
np.random.seed(42)

# Where to save the figures
PROJECT_ROOT_DIR = "."
CHAPTER_ID = "decision_trees"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID)

os.makedirs(IMAGES_PATH, exist_ok=True)

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

1.2 Part 0

- Run and check the outputs.

1.2.1 Confusion matrix plot

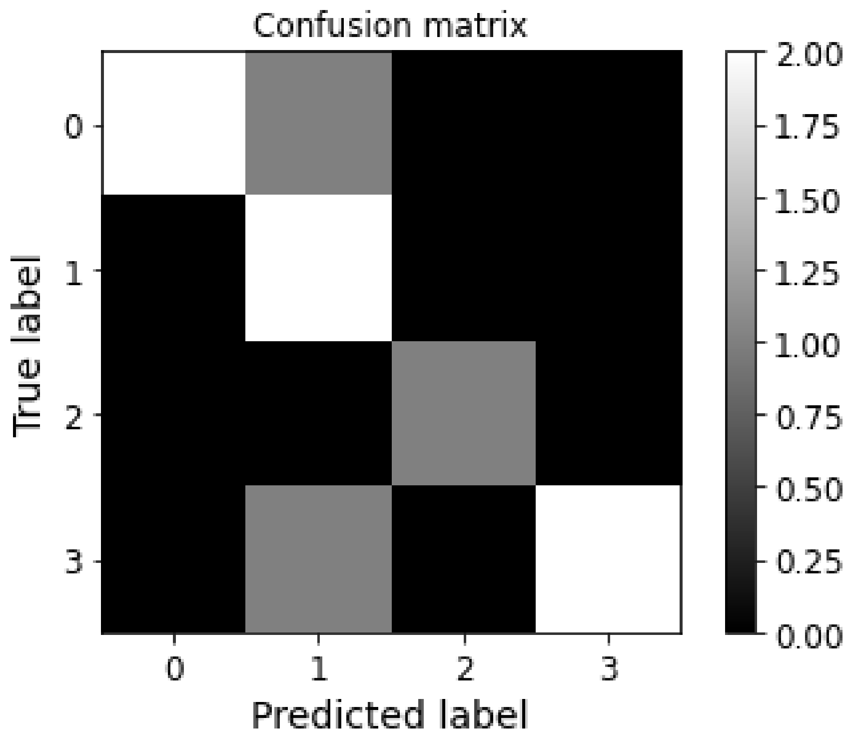
[2]:
```
# Show confusion matrix
def plot_confusion_matrix(confusion_mat, cln):
    plt.imshow(confusion_mat, interpolation='nearest', cmap=plt.cm.gray)
    plt.title('Confusion matrix')
    plt.colorbar()
    tick_marks = np.arange(cln)
    plt.xticks(tick_marks, tick_marks)
    plt.yticks(tick_marks, tick_marks)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()
```

1.2.2 Confusion matrix simple example 1

[3]:
```
y_true = [1, 0, 0, 2, 1, 0, 3, 3, 3]
y_pred = [1, 1, 0, 2, 1, 0, 1, 3, 3]
confusion_mat = confusion_matrix(y_true, y_pred)

print(confusion_mat)
plot_confusion_matrix(confusion_mat, 4)
```

```
[[2 1 0 0]
 [0 2 0 0]
```

```
 [0 0 1 0]
 [0 1 0 2]]
```



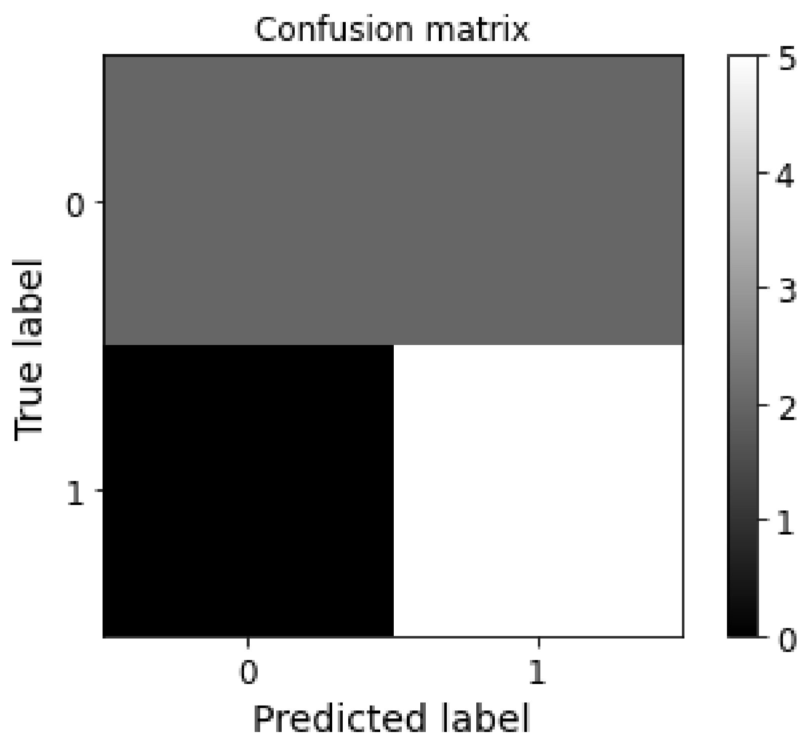
[4]:
```python
# Print classification report
target_names = ['Class-0', 'Class-1', 'Class-2', 'Class-3']

result_metrics = classification_report(y_true, y_pred,␣
 ↪target_names=target_names)

print(result_metrics)
```

```
              precision    recall  f1-score   support

     Class-0       1.00      0.67      0.80         3
     Class-1       0.50      1.00      0.67         2
     Class-2       1.00      1.00      1.00         1
     Class-3       1.00      0.67      0.80         3

    accuracy                           0.78         9
   macro avg       0.88      0.83      0.82         9
weighted avg       0.89      0.78      0.79         9
```

1.2.3 Confusion matrix simple example 2

[5]:
```python
y_true2 = [1, 0, 0, 1, 1, 0, 1, 1, 0]
y_pred2 = [1, 1, 0, 1, 1, 0, 1, 1, 1]
confusion_mat2 = confusion_matrix(y_true2, y_pred2)

print(confusion_mat2)
plot_confusion_matrix(confusion_mat2, 2)
```

```
[[2 2]
 [0 5]]
```



[6]:
```python
# Print classification report
target_names2 = ['Class-0', 'Class-1']

result_metrics = classification_report(y_true2, y_pred2,␣
 ↪target_names=target_names2)

print(result_metrics)
```

```
              precision    recall  f1-score   support

     Class-0       1.00      0.50      0.67         4
     Class-1       0.71      1.00      0.83         5
```

```
    accuracy                           0.78         9
   macro avg       0.86      0.75      0.75         9
weighted avg       0.84      0.78      0.76         9
```

1.3 Data Visualization

1.3.1 iris dataset before we start training and testing a model

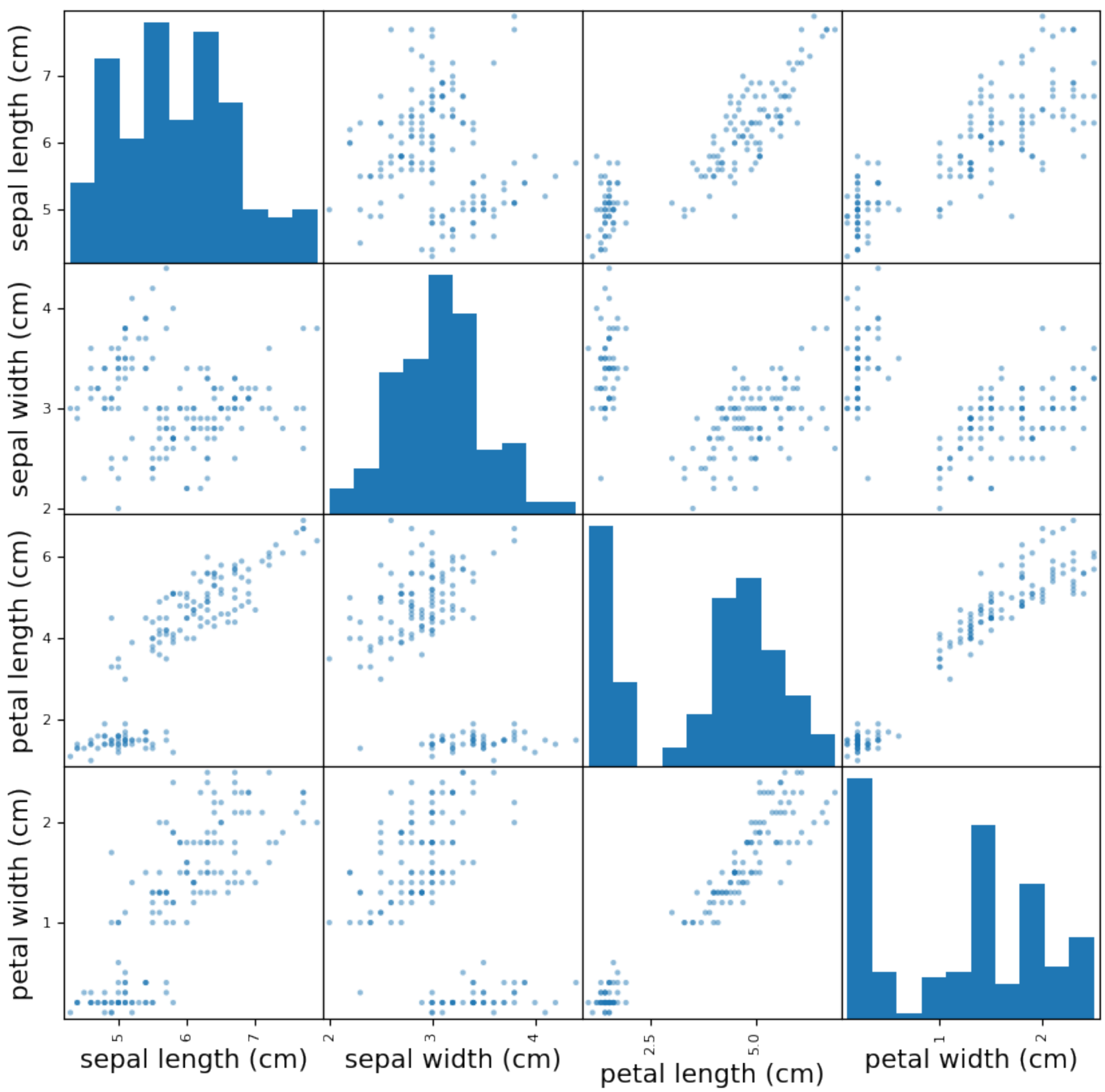
use pandas pd.plotting.scatter_matrix

[7]:
```
import matplotlib.pyplot as plt
import pandas as pd

# read data from CSV file to dataframe
iris = pd.read_csv('iris.csv')
print(iris.head())
print(iris.tail())
from sklearn import datasets
import pandas as pd
import matplotlib.pyplot as plt
# Load some data
iris = datasets.load_iris()
print(iris['feature_names'])
iris_df = pd.DataFrame(iris['data'], columns=iris['feature_names'])
# scatter matrix plot
fig, ax = plt.subplots(figsize=(10,10), dpi=100)

_ = pd.plotting.scatter_matrix(iris_df[[c for c in iris_df.columns if c !=␣
↪'y']], ax=ax)
_ = ax.set_title('Scatter matrix')
plt.show()
```

```
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa
      Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
145  146            6.7           3.0            5.2           2.3
146  147            6.3           2.5            5.0           1.9
147  148            6.5           3.0            5.2           2.0
148  149            6.2           3.4            5.4           2.3
149  150            5.9           3.0            5.1           1.8

            Species
145  Iris-virginica
146  Iris-virginica
```

```
147   Iris-virginica
148   Iris-virginica
149   Iris-virginica
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width
(cm)']
```

```
/var/folders/wz/88qmbx0907b9r93k2czdqckw0000gn/T/ipykernel_40817/1488047156.py:1
8: UserWarning: To output multiple subplots, the figure containing the passed
axes is being cleared
  _ = pd.plotting.scatter_matrix(iris_df[[c for c in iris_df.columns if c !=
'y']], ax=ax)
```



2 Decision Trees

2.0.1 Load data

- For the following code, we use sklearn.datasets package for loading a dataset instead of reading a data file stored on a local machine.

[8]:
```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

iris = load_iris()
#print(iris)
```

2.0.2 Split the data to training and testing

[9]:
```
X = iris.data[:, 2:] # petal length and width
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)
```

2.1 Training

2.1.1 Learing using training data

- use Gini index measure

*** Notes: you can also use gain information (entropy) measure by setting criterion="entropy" in the model

[10]:
```
tree_clf = DecisionTreeClassifier(max_depth=2, criterion="gini",␣
↪random_state=42)
tree_clf.fit(X_train, y_train)
```

[10]:
```
DecisionTreeClassifier(max_depth=2, random_state=42)
```

2.2 Testing

2.2.1 Evaluating the model using testing data

[11]:
```
y_pred = tree_clf.predict(X_test)
```

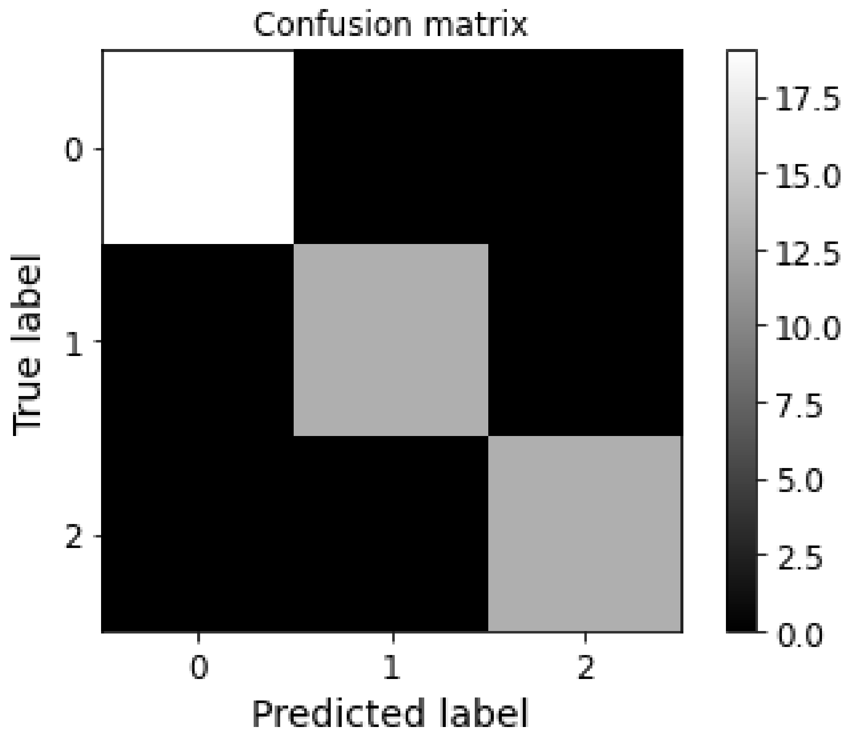
3 Visualization

3.1 Confusion matrix

[12]:
```
# plot a confusion matrix
confusion_mat = confusion_matrix(y_test, y_pred)

print(confusion_mat)
```

```
plot_confusion_matrix(confusion_mat, 3)
```

```
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
```



3.1.1 Model performance summary

[13]:
```
# Print classification report
target_names = iris.target_names

result_metrics = classification_report(y_test, y_pred,␣
↪target_names=target_names)

print(result_metrics)
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
```

```
weighted avg       1.00      1.00      1.00        45
```

[14]:
```
# you can access each class's metrics from result_metrics
result_metrics_dict = classification_report(y_test, y_pred,␣
 ↪target_names=target_names, output_dict=True)

print(result_metrics_dict['setosa']['precision'])
```

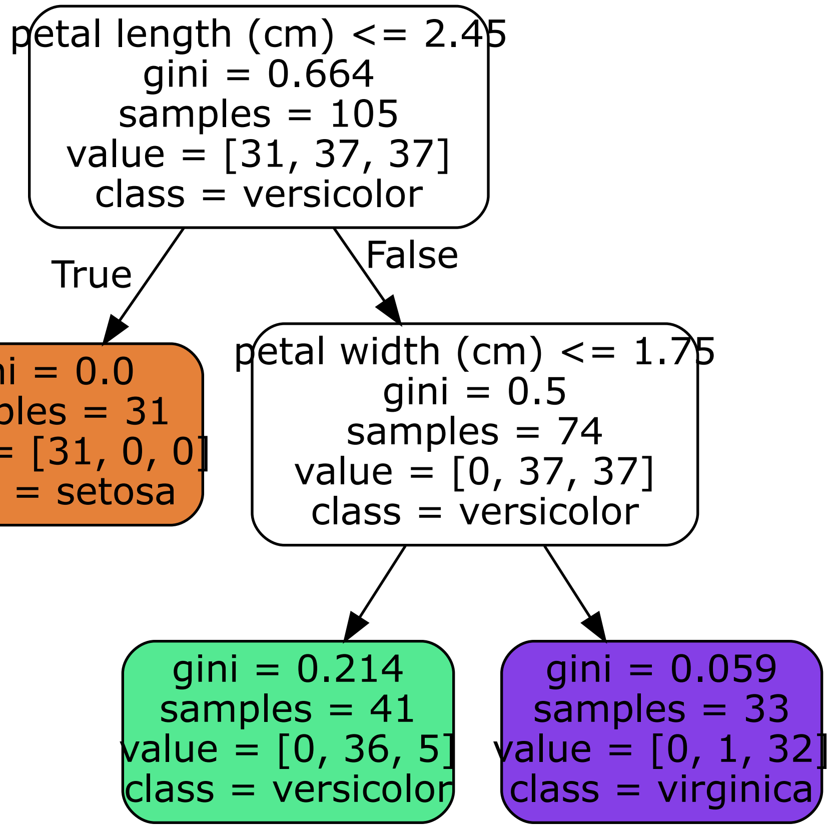
```
1.0
```

3.1.2 Draw a decision tree

notice that using graphviz is not the only method to draw decision tree. You can also use sklearn.tree.plot_tree

[15]:
```
from graphviz import Source
from sklearn.tree import export_graphviz

export_graphviz(
        tree_clf,
        out_file=os.path.join(IMAGES_PATH, "iris_tree.dot"),
        feature_names=iris.feature_names[2:],
        class_names=iris.target_names,
        rounded=True,
        filled=True
    )

Source.from_file(os.path.join(IMAGES_PATH, "iris_tree.dot"))
```

[15]:



3.2 k-Cross Validation

- using sklearn corss_val_score() function

[16]:
```
from sklearn.model_selection import cross_val_score

cross_val_score(tree_clf, iris.data, iris.target, cv=3)
```

[16]:
```
array([0.96, 0.92, 0.92])
```

3.3 k-Cross Validation

- using KFold function with freedom

[17]:
```
from sklearn.model_selection import KFold # import k-fold validation

kf = KFold(n_splits=3, random_state=None, shuffle=True) # Define the split -
 ↪into 2 folds

kf.get_n_splits(X) # returns the number of splitting iterations in the
 ↪cross-validator
```

```
print(kf)
```

```
KFold(n_splits=3, random_state=None, shuffle=True)
```

3.3.1 Applying k-Cross Validation

[18]:
```
tree_clf = DecisionTreeClassifier(max_depth=2, criterion="gini",␣
↪random_state=42)

for train_index, test_index in kf.split(X):
    #print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    tree_clf.fit(X_train, y_train)

    y_pred = tree_clf.predict(X_test)

    # Print classification report
    target_names = iris.target_names
    print(classification_report(y_test, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        21
  versicolor       1.00      0.92      0.96        12
   virginica       0.94      1.00      0.97        17

    accuracy                           0.98        50
   macro avg       0.98      0.97      0.98        50
weighted avg       0.98      0.98      0.98        50

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        16
  versicolor       0.87      1.00      0.93        20
   virginica       1.00      0.79      0.88        14

    accuracy                           0.94        50
   macro avg       0.96      0.93      0.94        50
weighted avg       0.95      0.94      0.94        50

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        13
  versicolor       0.89      0.94      0.92        18
   virginica       0.94      0.89      0.92        19
```

```
    accuracy                           0.94        50
   macro avg       0.95      0.95      0.95        50
weighted avg       0.94      0.94      0.94        50
```

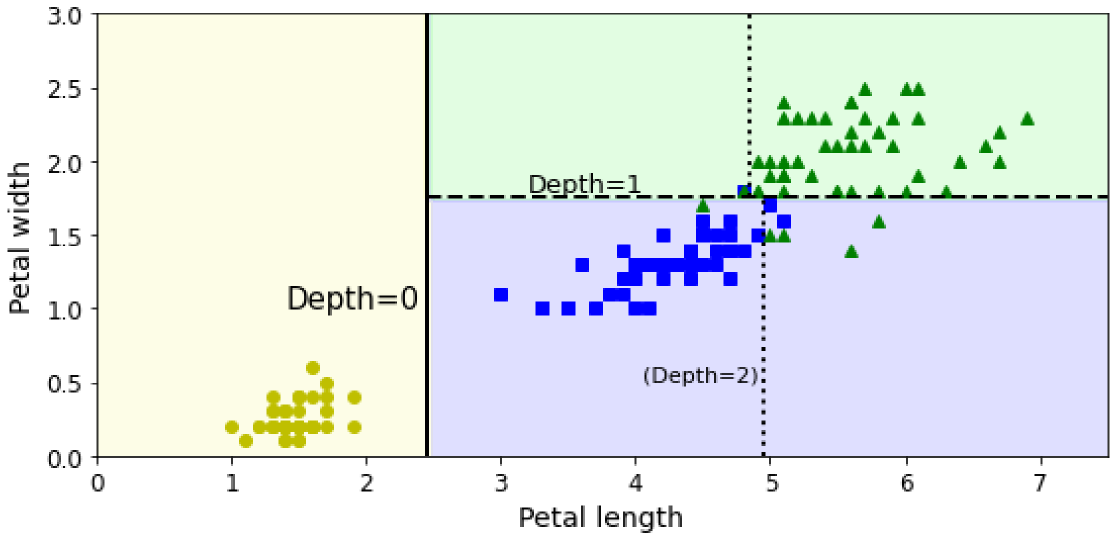
4 Decision Tree boundary Visualization

[19]:
```
## Example  This function is meant to be used for other data besides iris.
```

[20]:
```
from matplotlib.colors import ListedColormap

def plot_decision_boundary(clf, X, y, axes=[0, 7.5, 0, 3], iris=True,␣
 ↪legend=False, plot_training=True):
    x1s = np.linspace(axes[0], axes[1], 100)       # Return evenly spaced␣
 ↪numbers over a specified interval.
    x2s = np.linspace(axes[2], axes[3], 100)
    x1, x2 = np.meshgrid(x1s, x2s)
    X_new = np.c_[x1.ravel(), x2.ravel()]
    y_pred = clf.predict(X_new).reshape(x1.shape)
    custom_cmap = ListedColormap(['#fafab0','#9898ff','#a0faa0'])
    plt.contourf(x1, x2, y_pred, alpha=0.3, cmap=custom_cmap)
    if not iris:
        custom_cmap2 = ListedColormap(['#7d7d58','#4c4c7f','#507d50'])
        plt.contour(x1, x2, y_pred, cmap=custom_cmap2, alpha=0.8)
    if plot_training:
        plt.plot(X[:, 0][y==0], X[:, 1][y==0], "yo", label="Iris setosa")
        plt.plot(X[:, 0][y==1], X[:, 1][y==1], "bs", label="Iris versicolor")
        plt.plot(X[:, 0][y==2], X[:, 1][y==2], "g^", label="Iris virginica")
        plt.axis(axes)
    if iris:
        plt.xlabel("Petal length", fontsize=14)
        plt.ylabel("Petal width", fontsize=14)
    else:
        plt.xlabel(r"$x_1$", fontsize=18)
        plt.ylabel(r"$x_2$", fontsize=18, rotation=0)
    if legend:
        plt.legend(loc="lower right", fontsize=14)

plt.figure(figsize=(8, 4))
plot_decision_boundary(tree_clf, X, y)
plt.plot([2.45, 2.45], [0, 3], "k-", linewidth=2)
plt.plot([2.45, 7.5], [1.75, 1.75], "k--", linewidth=2)
plt.plot([4.95, 4.95], [0, 1.75], "k:", linewidth=2)
plt.plot([4.85, 4.85], [1.75, 3], "k:", linewidth=2)
plt.text(1.40, 1.0, "Depth=0", fontsize=15)
```

```
plt.text(3.2, 1.80, "Depth=1", fontsize=13)
plt.text(4.05, 0.5, "(Depth=2)", fontsize=11)

save_fig("decision_tree_decision_boundaries_plot")
plt.show()
```

```
Saving figure decision_tree_decision_boundaries_plot
```



5 Predicting classes and class probabilities

[21]:
```
tree_clf.predict_proba([[5, 1.5]])
```

[21]:
```
array([[0.        , 0.91428571, 0.08571429]])
```

[22]:
```
tree_clf.predict([[5, 1.5]])
```

[22]:
```
array([1])
```

6 Sensitivity to training set details

[23]:
```
X[(X[:, 1]==X[:, 1][y==1].max()) & (y==1)] # widest Iris versicolor flower
```

[23]:
```
array([[4.8, 1.8]])
```

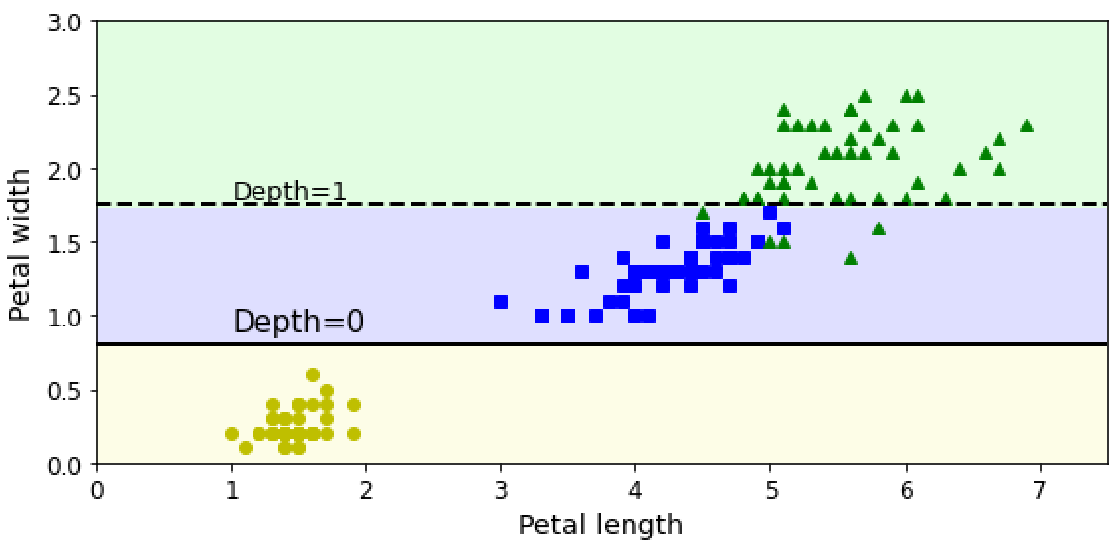
[24]:
```
not_widest_versicolor = (X[:, 1]!=1.8) | (y==2)
X_tweaked = X[not_widest_versicolor]
y_tweaked = y[not_widest_versicolor]
```

```
tree_clf_tweaked = DecisionTreeClassifier(max_depth=2, random_state=40)
tree_clf_tweaked.fit(X_tweaked, y_tweaked)
```

[24]: DecisionTreeClassifier(max_depth=2, random_state=40)

[25]:
```
plt.figure(figsize=(8, 4))
plot_decision_boundary(tree_clf_tweaked, X_tweaked, y_tweaked, legend=False)
plt.plot([0, 7.5], [0.8, 0.8], "k-", linewidth=2)
plt.plot([0, 7.5], [1.75, 1.75], "k--", linewidth=2)
plt.text(1.0, 0.9, "Depth=0", fontsize=15)
plt.text(1.0, 1.80, "Depth=1", fontsize=13)

save_fig("decision_tree_instability_plot")
plt.show()
```

Saving figure decision_tree_instability_plot



7 ========== HW3 ===============

7.1 ==================================

7.2 Construct decision trees

1. Construct a decision tree using the following parameters

- Use information gain (entropy) measure
- Apply k=10 cross validation and print a summary of statistics (performance evaluation) for each fold

2. Compare the performance results with those of the decision tree using Gini index measure in the above example

3. For both trees, change the following parameters and observe the changes:

- The depth of tree: currently max_depth=2 in the model training step. Change the depth 3, 4, 5 and check if this affects the overall results.
- The k value for cross validation is currently set to 3. Change k value, k = 5, 7, 10 and check if this affects the overall results.

7.2.1 1. LOAD DATASET AND IMPORTS

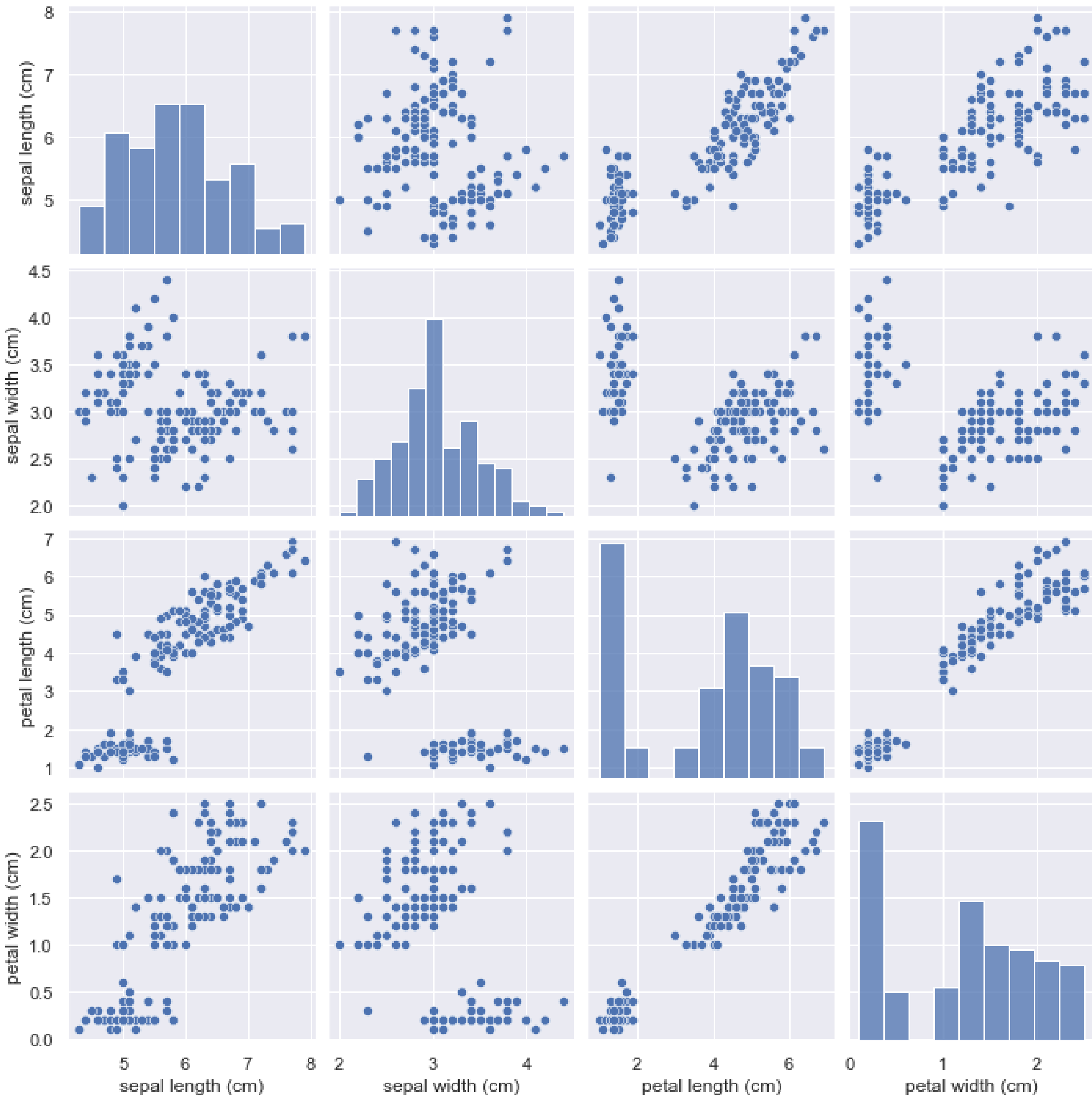
[26]:
```
# Import the data, classifier, and metrics libraries similar to what we did␣
↪above
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

#Import sns for scatter plots
import seaborn as sns

#import data
iris = pd.read_csv('iris.csv')
print('Head for CSV : \n ',iris.head())
print('Tail for CSV : \n ',iris.tail())

#Train Test Split
from sklearn.model_selection import train_test_split

# import k-fold validation
from sklearn.model_selection import KFold

#To visualize the decision tree
from graphviz import Source
from sklearn.tree import export_graphviz

#to determine the cross validation score
from sklearn.model_selection import cross_val_score

# load iris dataset and confirm data has been loaded by printing first few lines
from sklearn.datasets import load_iris

iris = load_iris()
print('Feature Names for Dataset : \n ',iris['feature_names'])
iris_df = pd.DataFrame(iris['data'], columns=iris['feature_names'])

print('Head for Dataset : \n ',iris_df.head())
```

```
print('Tail for Dataset : \n ',iris_df.tail())
```

```
Head for CSV :
     Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa
Tail for CSV :
         Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
145  146            6.7           3.0            5.2           2.3
146  147            6.3           2.5            5.0           1.9
147  148            6.5           3.0            5.2           2.0
148  149            6.2           3.4            5.4           2.3
149  150            5.9           3.0            5.1           1.8

            Species
145  Iris-virginica
146  Iris-virginica
147  Iris-virginica
148  Iris-virginica
149  Iris-virginica
Feature Names for Dataset :
  ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width
(cm)']
Head for Dataset :
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
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
Tail for Dataset :
       sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
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
```

[27]:
```
#Scatters plots
sns.set()
sns.pairplot(iris_df)
```

[27]: `<seaborn.axisgrid.PairGrid at 0x7fc480adfca0>`



Train test split

```
[28]: # Split data into testing and training
X = iris.data[:, 2:] # petal length and petal width
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)
```

7.2.2 2. DECISION TREE WITH ENTROPY

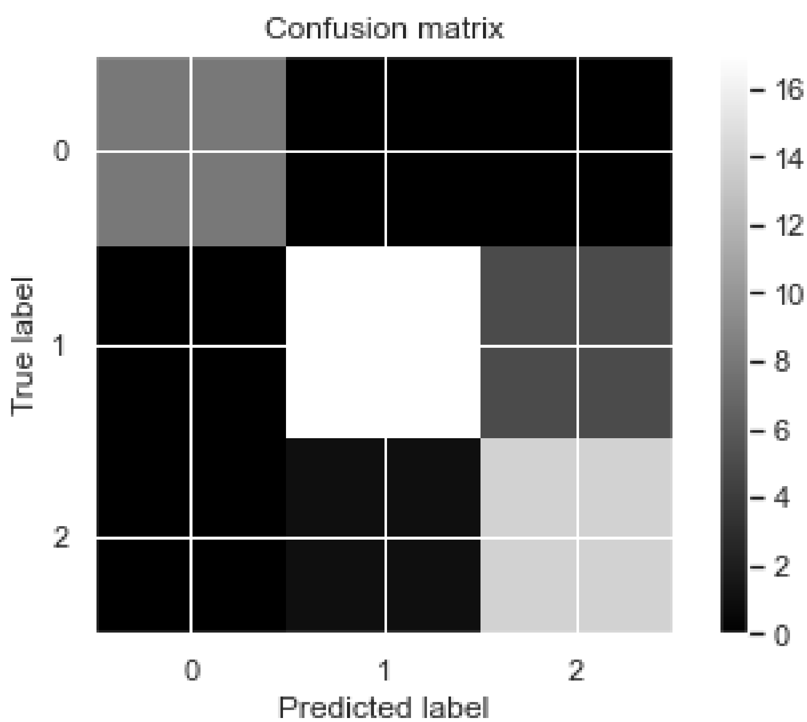
```
[29]: # Create decision tree and set criterion to information gain (entropy)
tree_clf_md2_ent = DecisionTreeClassifier(max_depth=2, criterion="entropy",␣
↪random_state=42)
tree_clf_md2_ent.fit(X_train, y_train)
```

[29]: DecisionTreeClassifier(criterion='entropy', max_depth=2, random_state=42)

[30]:
```
# Predict
y_pred = tree_clf_md2_ent.predict(X_test)
```

[31]:
```
# plot a confusion matrix
confusion_mat_md2_ent = confusion_matrix(y_test, y_pred)
print(confusion_mat_md2_ent)
plot_confusion_matrix(confusion_mat_md2_ent, 3)
```

```
[[ 8  0  0]
 [ 0 17  5]
 [ 0  1 14]]
```



[32]:
```
# Print classification report
target_names = iris.target_names

result_metrics = classification_report(y_test, y_pred,␣
↪target_names=target_names)

print(result_metrics)
```

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

```
      setosa       1.00      1.00      1.00         8
  versicolor       0.94      0.77      0.85        22
   virginica       0.74      0.93      0.82        15

    accuracy                           0.87        45
   macro avg       0.89      0.90      0.89        45
weighted avg       0.89      0.87      0.87        45
```

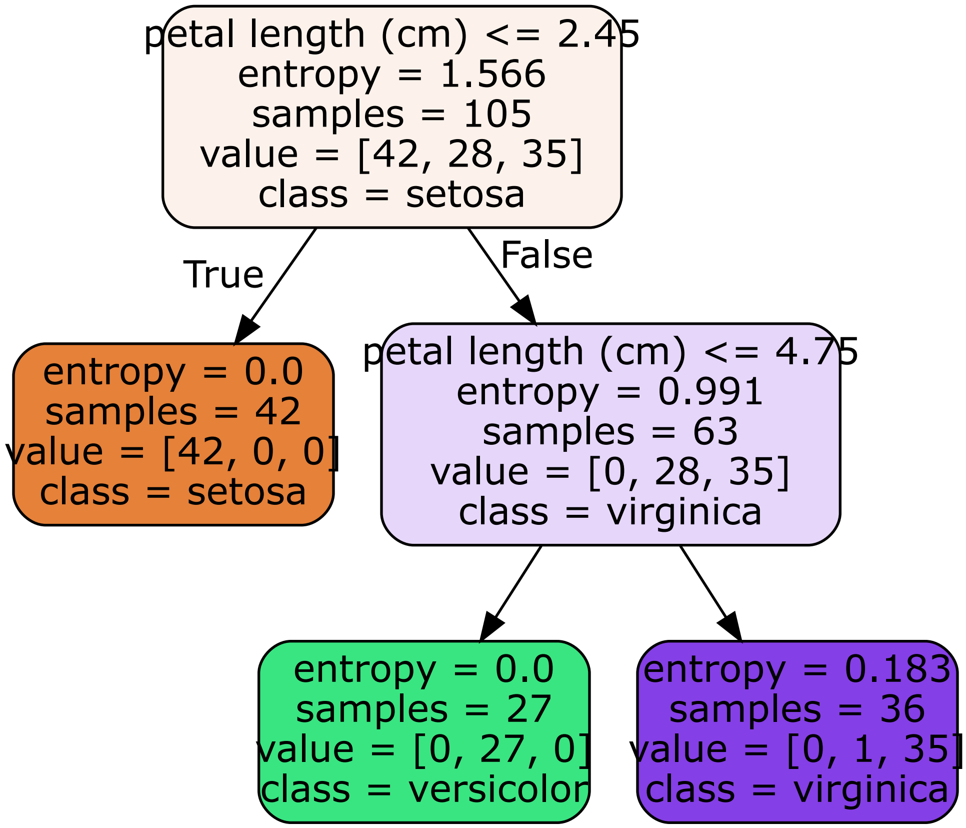
[33]:
```
# you can access each class's metrics from result_metrics
result_metrics_dict = classification_report(y_test, y_pred,␣
 ↪target_names=target_names, output_dict=True)
print(result_metrics_dict['setosa']['precision'])
```

```
1.0
```

7.2.3 3. VISUALIZE THE TREE (ENTROPY)

[34]:
```
#Graphviz tree

export_graphviz(
        tree_clf_md2_ent,
        out_file=os.path.join(IMAGES_PATH, "iris_tree_md2_entropy.dot"),
        feature_names=iris.feature_names[2:],
        class_names=iris.target_names,
        rounded=True,
        filled=True
    )

Source.from_file(os.path.join(IMAGES_PATH, "iris_tree_md2_entropy.dot"))
```

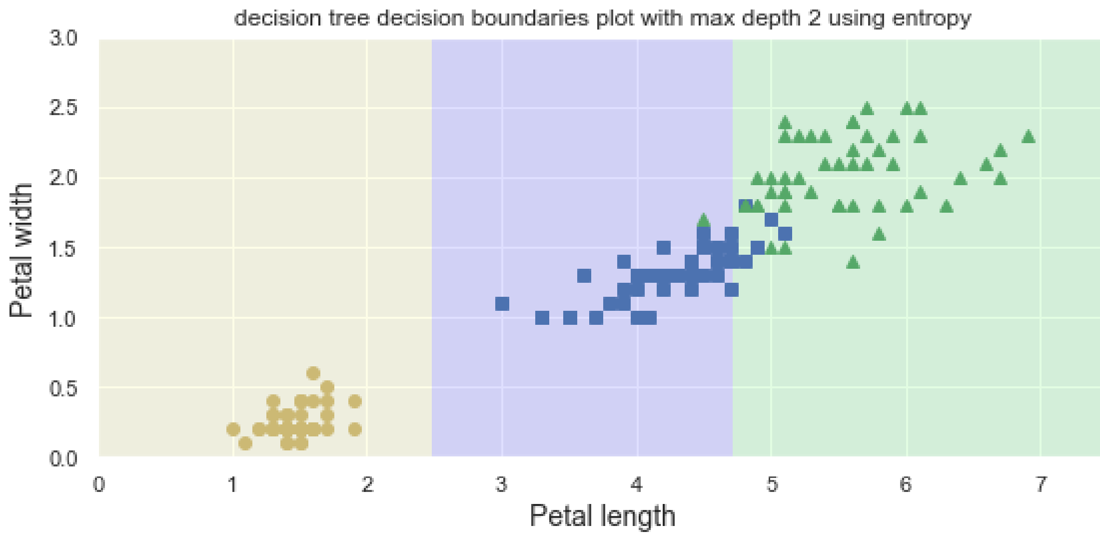
[34]:



Plot the decision boundry (entropy)

[35]:
```
# Plot the decision boundry graph for the entropy model
plt.figure(figsize=(8, 4))
plot_decision_boundary(tree_clf_md2_ent, X, y)
plt.title('decision tree decision boundaries plot with max depth 2 using␣
 ↪entropy')
save_fig("decision_tree_decision_boundaries_plot_10f_md2_entropy")
plt.show()
```

```
Saving figure decision_tree_decision_boundaries_plot_10f_md2_entropy
```



7.2.4 4. CROSS VALIDATION SCORE 10-FOLD (ENTROPY)

[36]:
```
# Apply 10-fold cross validation score
cross_val_score(tree_clf_md2_ent, iris.data, iris.target, cv=10)
```

[36]:
```
array([0.93333333, 0.93333333, 1.        , 0.93333333, 0.93333333,
       0.86666667, 0.86666667, 1.        , 1.        , 1.        ])
```

7.2.5 Explain what is the relationship between the drawn decision boundaries and gini index.

We are using the petal length and petal width to distinguish among the three flowers species types. In the first example we used the gini index and later entropy to create the two decision trees . The decision boundaries suffice the purpose of dividing different flowers with different hue. - Setosa boundaries are extremely clear which is also reflected by its classification report which is usually 1 for precision and recall scores. - While the other two species boundaries are also defined well by the decision tree using gini index, but there are certain points which either overlap or are very close to each other, so they are in the wrong color region which is expected as there are cases which have been incorrectly classified.

7.2.6 Compare the performance results with those of the decision tree using Gini index measure in the above example

- K-Fold =3, Max Depth = 2, Criterion = gini (above example)
- Cross val score - 0.96, 0.92, 0.92 in each fold respectively
- K-Fold = 10, Max Depth =2 , Criterion = entropy
- Cross val score - 0.93333333, 0.93333333, 1, 0.93333333, 0.93333333, 0.86666667, 0.86666667, 1, 1, 1 in each fold respectively Similarities

- Setosa is correctly classified in both the cases with 100% accuracy
- The decision boundaries are similar (almost the same) for versicolor and virginica
- entropy and gini use two different approaches i.e. Gini - calculates the amount of probability of a specific feature that is classified incorrectly when selected randomly, between 0 and 0.5, Entropy - it is the measurement of the impurity or randomness in the data points, calculated between 0 and 1
- However, the final decision boundaries are very similar in both the scenarios
- Differences: When we increase the number of folds, gini seems to overfit less and performs better in this particular use case.

8 Section 1: 10 Fold Cross Validation Trees

- Tree with max depth 2
- Tree with max depth 3
- Tree with max depth 4
-

8.1 Tree with max depth 5

8.1.1 Setup for K-fold = 10

[37]:
```
# Parse and bin the data into 10 folds for validation

# Define the split - into 10 folds
kf_10_all = KFold(n_splits=10, random_state=None, shuffle=True)

# returns the number of splitting iterations in the cross-validator
kf_10_all.get_n_splits(X)

print(kf_10_all)
```

```
KFold(n_splits=10, random_state=None, shuffle=True)
```

8.1.2 K-FOLD Cross Validation (10-folds - Max Depth = 2 Default) with Entropy

Apply K-fold and Classification Report

[38]:
```
# Apply the 10-fold cross validation we've created
tree_clf_10f_md2_ent = DecisionTreeClassifier(max_depth=2, criterion="entropy",␣
↪random_state=42)

for train_index, test_index in kf_10_all.split(X):
    #print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
```

```python
tree_clf_10f_md2_ent.fit(X_train, y_train)

y_pred = tree_clf_10f_md2_ent.predict(X_test)

# Print classification report
target_names = iris.target_names
print(classification_report(y_test, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         2
  versicolor       0.50      1.00      0.67         2
   virginica       1.00      0.82      0.90        11

    accuracy                           0.87        15
   macro avg       0.83      0.94      0.86        15
weighted avg       0.93      0.87      0.88        15

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         5
  versicolor       1.00      0.86      0.92         7
   virginica       0.75      1.00      0.86         3

    accuracy                           0.93        15
   macro avg       0.92      0.95      0.93        15
weighted avg       0.95      0.93      0.94        15

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         4
  versicolor       1.00      1.00      1.00         6
   virginica       1.00      1.00      1.00         5

    accuracy                           1.00        15
   macro avg       1.00      1.00      1.00        15
weighted avg       1.00      1.00      1.00        15

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         8
  versicolor       1.00      1.00      1.00         4
   virginica       1.00      1.00      1.00         3

    accuracy                           1.00        15
   macro avg       1.00      1.00      1.00        15
weighted avg       1.00      1.00      1.00        15
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         5
  versicolor       0.71      0.83      0.77         6
   virginica       0.67      0.50      0.57         4

    accuracy                           0.80        15
   macro avg       0.79      0.78      0.78        15
weighted avg       0.80      0.80      0.79        15

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         4
  versicolor       0.86      1.00      0.92         6
   virginica       1.00      0.80      0.89         5

    accuracy                           0.93        15
   macro avg       0.95      0.93      0.94        15
weighted avg       0.94      0.93      0.93        15

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         5
  versicolor       1.00      1.00      1.00         5
   virginica       1.00      1.00      1.00         5

    accuracy                           1.00        15
   macro avg       1.00      1.00      1.00        15
weighted avg       1.00      1.00      1.00        15

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         4
  versicolor       0.67      0.80      0.73         5
   virginica       0.80      0.67      0.73         6

    accuracy                           0.80        15
   macro avg       0.82      0.82      0.82        15
weighted avg       0.81      0.80      0.80        15

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         7
  versicolor       1.00      1.00      1.00         4
   virginica       1.00      1.00      1.00         4

    accuracy                           1.00        15
   macro avg       1.00      1.00      1.00        15
```

```
weighted avg       1.00      1.00      1.00        15

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         6
  versicolor       1.00      1.00      1.00         5
   virginica       1.00      1.00      1.00         4

    accuracy                           1.00        15
   macro avg       1.00      1.00      1.00        15
weighted avg       1.00      1.00      1.00        15
```

Cross Validation Score

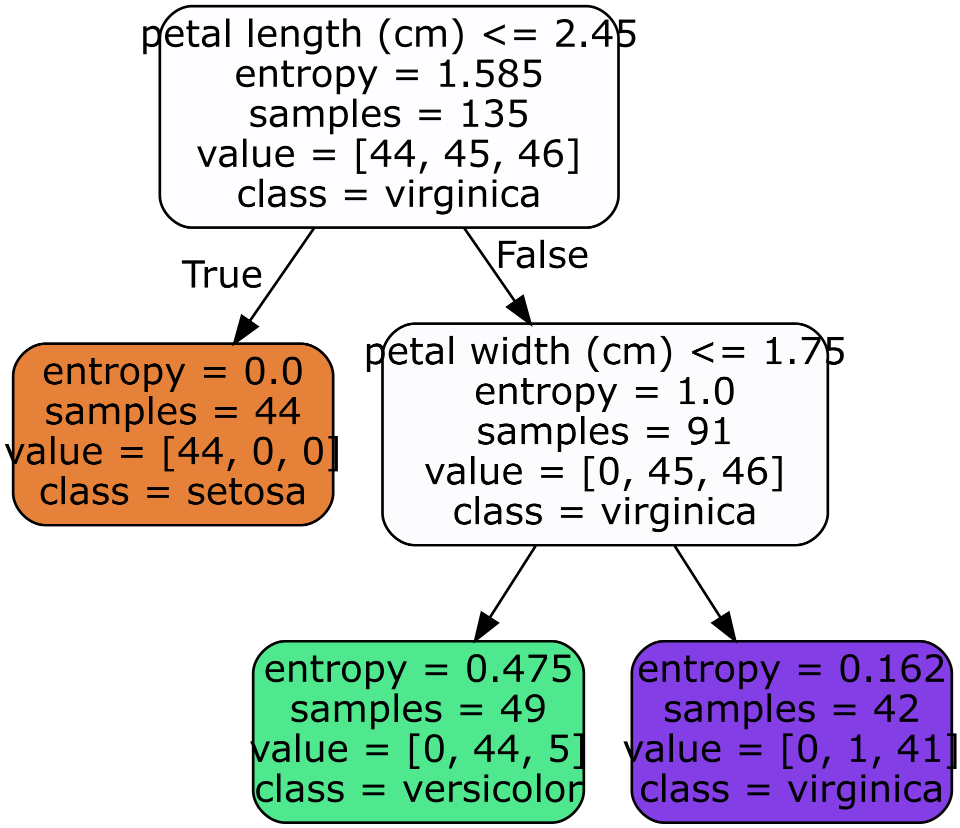
[39]:
```
cross_val_score(tree_clf_10f_md2_ent, iris.data, iris.target, cv=10)
```

[39]:
```
array([0.93333333, 0.93333333, 1.        , 0.93333333, 0.93333333,
       0.86666667, 0.86666667, 1.        , 1.        , 1.        ])
```

Graphviz

[40]:
```
export_graphviz(
        tree_clf_10f_md2_ent,
        out_file=os.path.join(IMAGES_PATH, "iris_tree_10f_md2_entropy.dot"),
        feature_names=iris.feature_names[2:],
        class_names=iris.target_names,
        rounded=True,
        filled=True
    )

Source.from_file(os.path.join(IMAGES_PATH, "iris_tree_10f_md2_entropy.dot"))
```

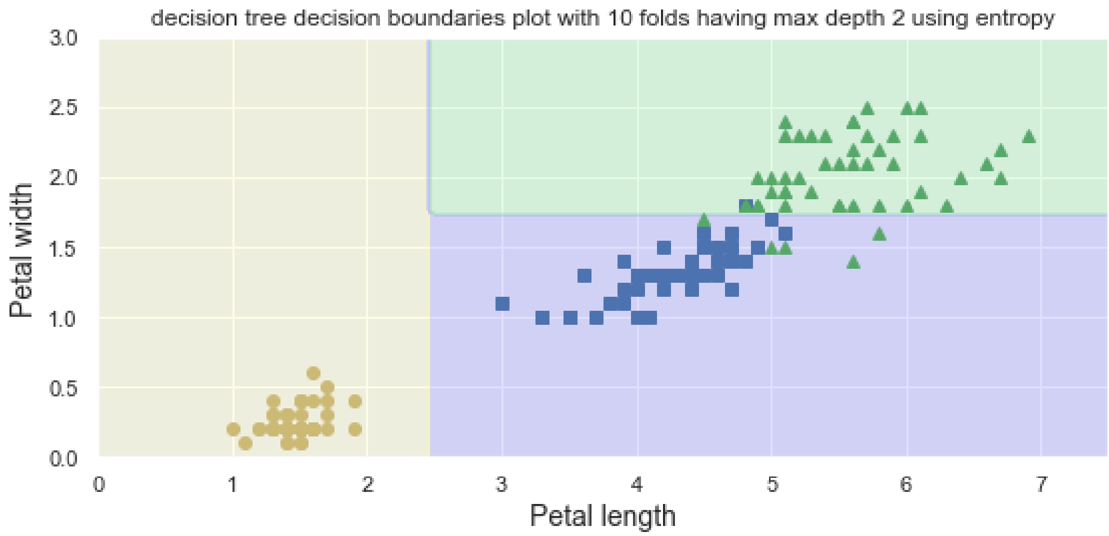
[40]:



Plot the decision boundry (entropy)

[41]:
```
# Plot the decision boundry graph for the entropy model
plt.figure(figsize=(8, 4))
plot_decision_boundary(tree_clf_10f_md2_ent, X, y)
plt.title('decision tree decision boundaries plot with 10 folds having max␣
 ↪depth 2 using entropy')
save_fig("decision_tree_decision_boundaries_plot_10f_md2_entropy")
plt.show()
```

```
Saving figure decision_tree_decision_boundaries_plot_10f_md2_entropy
```



8.1.3 K-FOLD Cross Validation (10-folds - Max Depth = 3) with Entropy

Apply K-fold and Classification Report

[42]:
```
# Apply the 10-fold cross validation we've created
tree_clf_10f_md3_ent = DecisionTreeClassifier(max_depth=3, criterion="entropy",␣
↪random_state=42)

for train_index, test_index in kf.split(X):
    #print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    tree_clf_10f_md3_ent.fit(X_train, y_train)

    y_pred = tree_clf_10f_md3_ent.predict(X_test)

    # Print classification report
    target_names = iris.target_names
    print(classification_report(y_test, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        13
  versicolor       0.95      0.82      0.88        22
   virginica       0.78      0.93      0.85        15

    accuracy                           0.90        50
   macro avg       0.91      0.92      0.91        50
weighted avg       0.91      0.90      0.90        50
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        14
   virginica       1.00      1.00      1.00        17

    accuracy                           1.00        50
   macro avg       1.00      1.00      1.00        50
weighted avg       1.00      1.00      1.00        50

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        18
  versicolor       0.81      0.93      0.87        14
   virginica       0.94      0.83      0.88        18

    accuracy                           0.92        50
   macro avg       0.92      0.92      0.92        50
weighted avg       0.93      0.92      0.92        50
```

Cross Validation Score

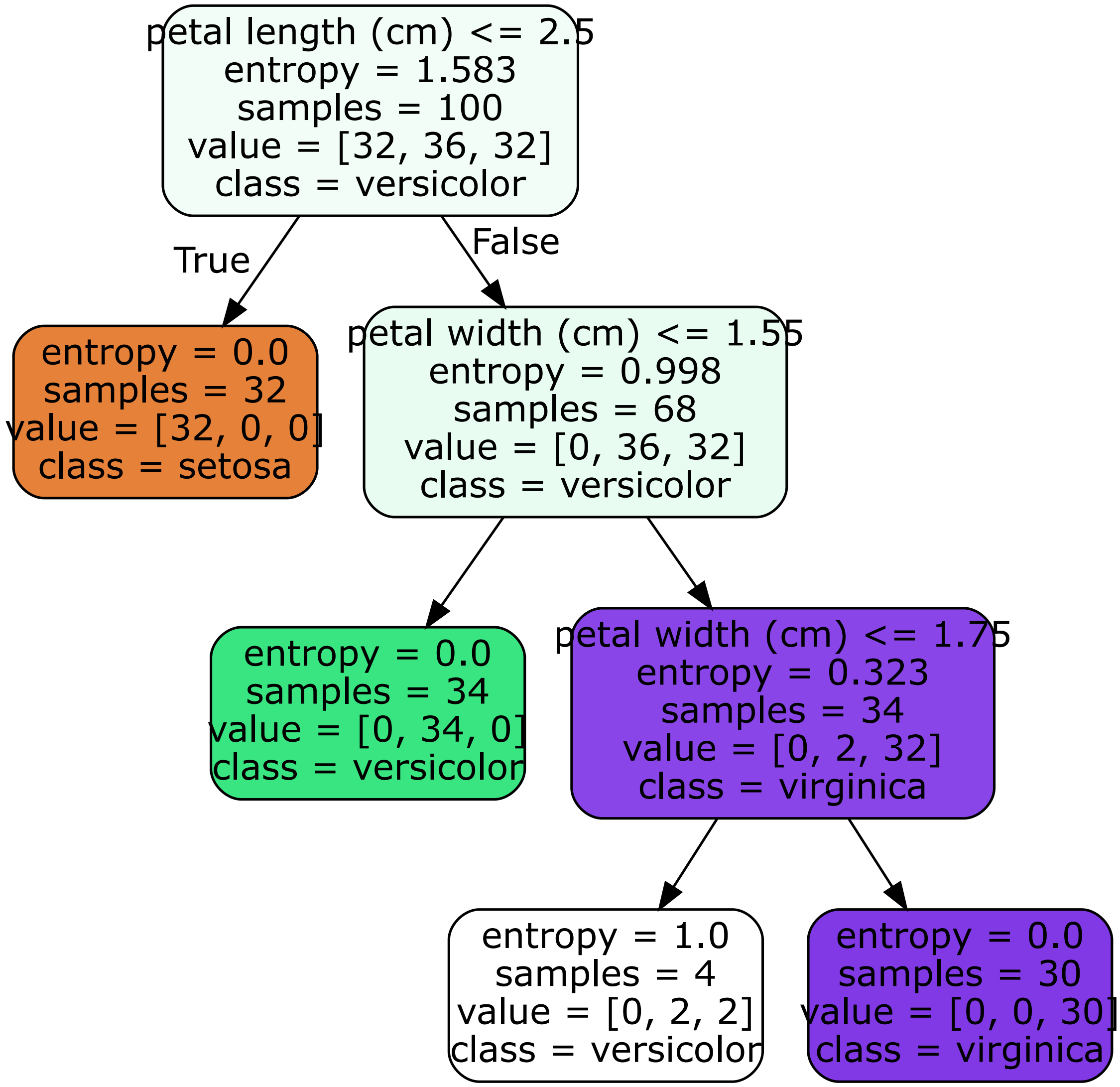
[43]:
```
cross_val_score(tree_clf_10f_md3_ent, iris.data, iris.target, cv=10)
```

[43]:
```
array([1.        , 0.93333333, 1.        , 0.93333333, 0.93333333,
       0.93333333, 0.93333333, 0.93333333, 1.        , 1.        ])
```

Graphviz

[44]:
```
export_graphviz(
        tree_clf_10f_md3_ent,
        out_file=os.path.join(IMAGES_PATH, "iris_tree_10f_md3_entropy.dot"),
        feature_names=iris.feature_names[2:],
        class_names=iris.target_names,
        rounded=True,
        filled=True
    )

Source.from_file(os.path.join(IMAGES_PATH, "iris_tree_10f_md3_entropy.dot"))
```

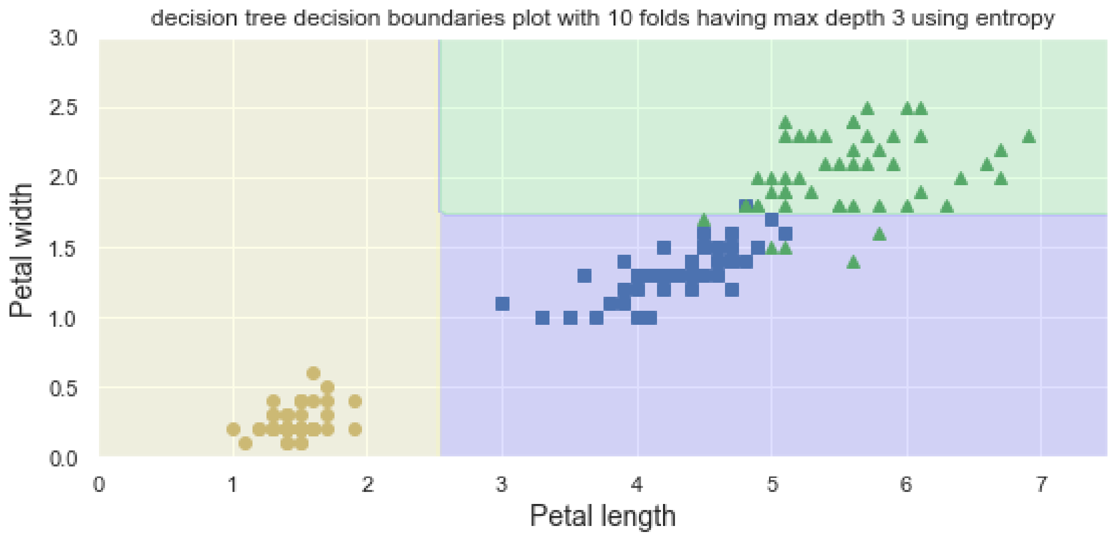
[44]:



Plot the decision boundry (entropy)

[45]:
```
# Plot the decision boundry graph for the model
plt.figure(figsize=(8, 4))
plot_decision_boundary(tree_clf_10f_md3_ent, X, y)
plt.title('decision tree decision boundaries plot with 10 folds having max␣
↪depth 3 using entropy')
save_fig("decision_tree_decision_boundaries_plot_10f_md3_entropy")
plt.show()
```

```
Saving figure decision_tree_decision_boundaries_plot_10f_md3_entropy
```



8.1.4 K-FOLD Cross Validation (10-folds - Max Depth = 4) with Entropy

Apply K-fold and Classification Report

[46]:
```python
# Apply the 10-fold cross validation we've created
tree_clf_10f_md4_ent = DecisionTreeClassifier(max_depth=4, criterion="entropy",␣
↪random_state=42)

for train_index, test_index in kf.split(X):
    #print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    tree_clf_10f_md4_ent.fit(X_train, y_train)

    y_pred = tree_clf_10f_md4_ent.predict(X_test)

    # Print classification report
    target_names = iris.target_names
    print(classification_report(y_test, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        18
  versicolor       1.00      0.89      0.94        19
   virginica       0.87      1.00      0.93        13

    accuracy                           0.96        50
   macro avg       0.96      0.96      0.96        50
weighted avg       0.97      0.96      0.96        50
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        18
  versicolor       0.69      1.00      0.81        11
   virginica       1.00      0.76      0.86        21

    accuracy                           0.90        50
   macro avg       0.90      0.92      0.89        50
weighted avg       0.93      0.90      0.90        50

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        14
  versicolor       1.00      0.95      0.97        20
   virginica       0.94      1.00      0.97        16

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

Cross Validation Score

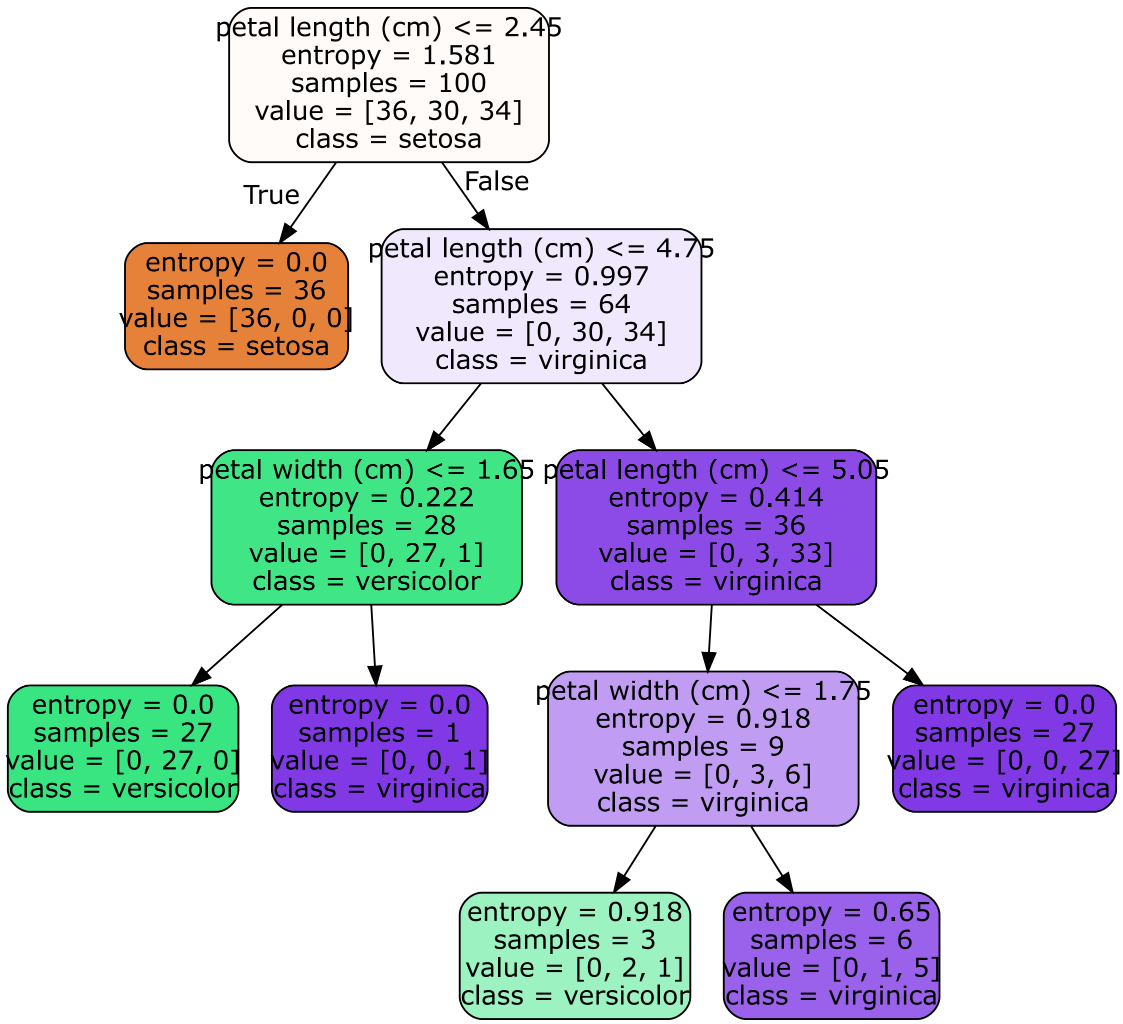
[47]:
```python
cross_val_score(tree_clf_10f_md4_ent, iris.data, iris.target, cv=10)
```

[47]:
```
array([1.        , 0.93333333, 1.        , 0.93333333, 0.93333333,
       0.86666667, 0.93333333, 1.        , 1.        , 1.        ])
```

Graphviz

[48]:
```python
export_graphviz(
        tree_clf_10f_md4_ent,
        out_file=os.path.join(IMAGES_PATH, "iris_tree_10f_md4_entropy.dot"),
        feature_names=iris.feature_names[2:],
        class_names=iris.target_names,
        rounded=True,
        filled=True
    )

Source.from_file(os.path.join(IMAGES_PATH, "iris_tree_10f_md4_entropy.dot"))
```

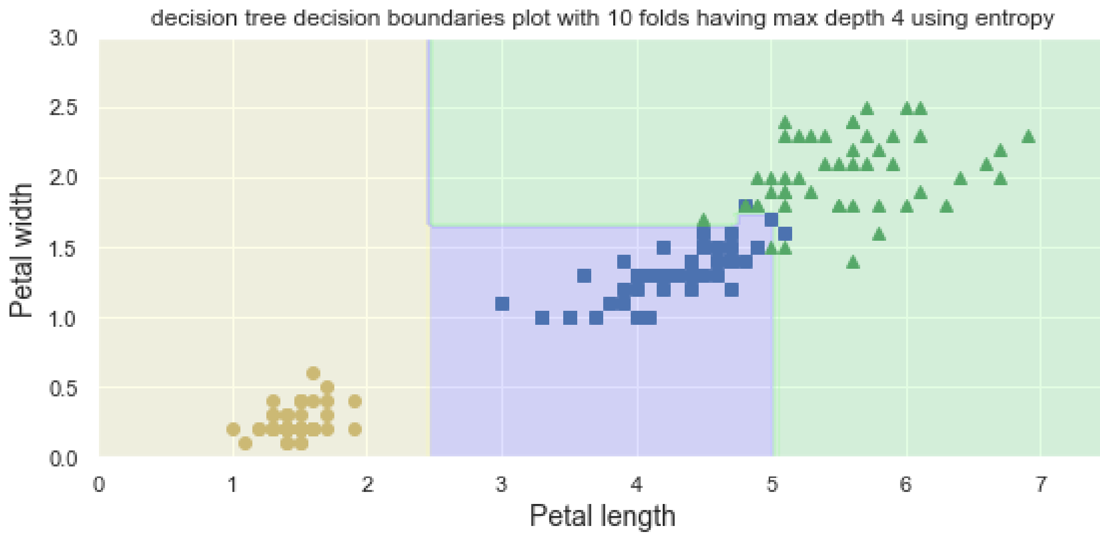
[48]:



Plot the decision boundry (entropy)

[49]:
```python
# Plot the decision boundry graph for the model
plt.figure(figsize=(8, 4))
plot_decision_boundary(tree_clf_10f_md4_ent, X, y)
plt.title('decision tree decision boundaries plot with 10 folds having max␣
 ↪depth 4 using entropy')
save_fig("decision_tree_decision_boundaries_plot_10f_md4_entropy")
plt.show()
```

```
Saving figure decision_tree_decision_boundaries_plot_10f_md4_entropy
```



8.1.5 K-FOLD Cross Validation (10-folds - Max Depth = 5) with Entropy

Apply K-fold and Classification Report

[50]:
```
# Apply the 10-fold cross validation we've created
tree_clf_10f_md5_ent = DecisionTreeClassifier(max_depth=5, criterion="entropy",␣
↪random_state=42)

for train_index, test_index in kf.split(X):
    #print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    tree_clf_10f_md5_ent.fit(X_train, y_train)

    y_pred = tree_clf_10f_md5_ent.predict(X_test)

    # Print classification report
    target_names = iris.target_names
    print(classification_report(y_test, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        15
  versicolor       0.89      0.94      0.91        17
   virginica       0.94      0.89      0.91        18

    accuracy                           0.94        50
   macro avg       0.94      0.94      0.94        50
weighted avg       0.94      0.94      0.94        50
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        22
  versicolor       0.93      0.76      0.84        17
   virginica       0.71      0.91      0.80        11

    accuracy                           0.90        50
   macro avg       0.88      0.89      0.88        50
weighted avg       0.91      0.90      0.90        50

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        13
  versicolor       0.79      0.94      0.86        16
   virginica       0.94      0.81      0.87        21

    accuracy                           0.90        50
   macro avg       0.91      0.92      0.91        50
weighted avg       0.91      0.90      0.90        50
```

Cross Validation Score

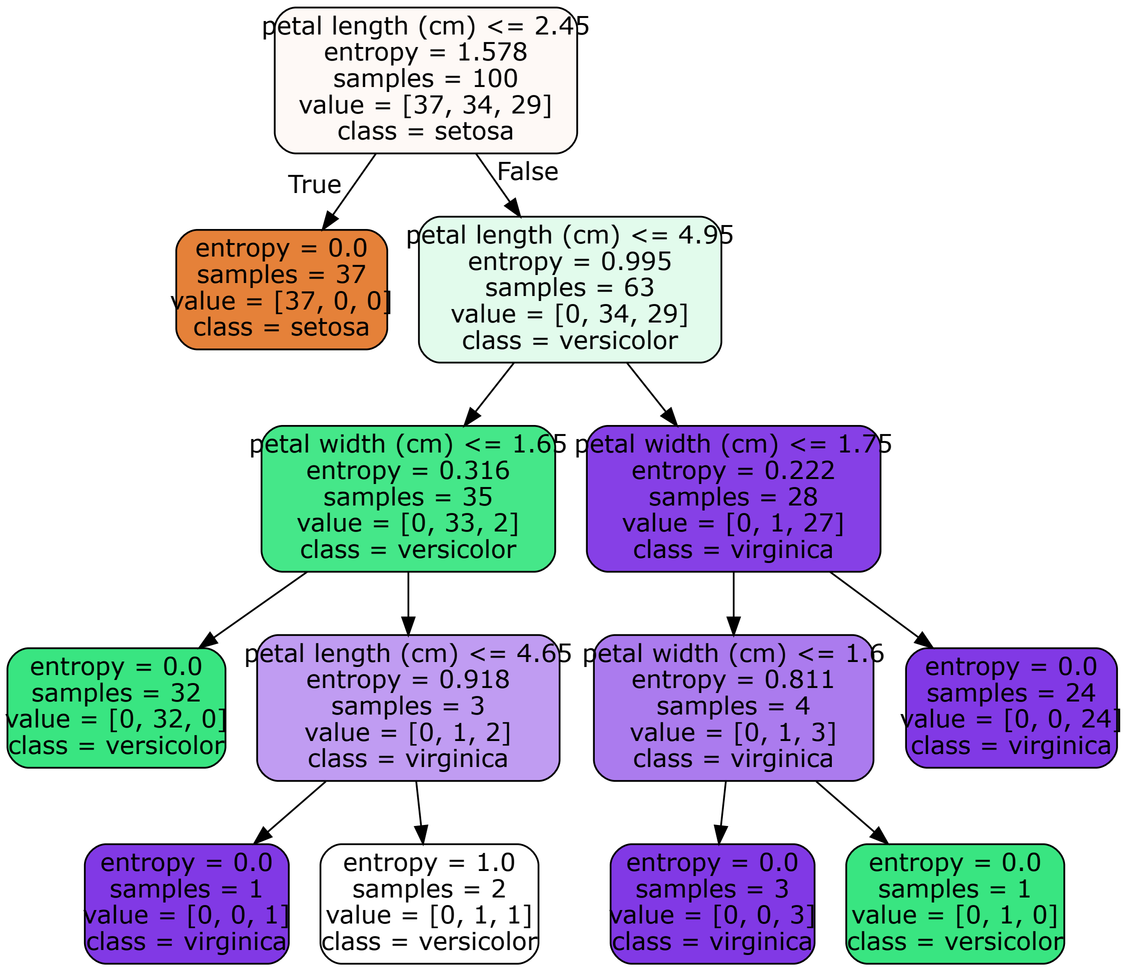
[51]:
```
cross_val_score(tree_clf_10f_md5_ent, iris.data, iris.target, cv=10)
```

[51]:
```
array([1.        , 0.93333333, 1.        , 0.93333333, 0.93333333,
       0.86666667, 0.93333333, 0.93333333, 1.        , 1.        ])
```

Graphviz

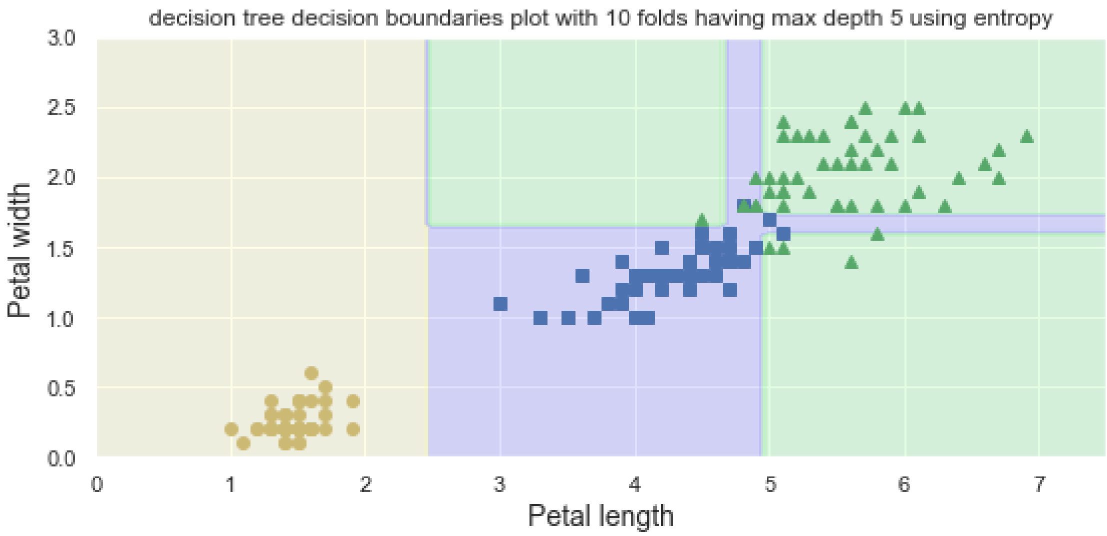
[52]:
```
export_graphviz(
        tree_clf_10f_md5_ent,
        out_file=os.path.join(IMAGES_PATH, "iris_tree_10f_md5_entropy.dot"),
        feature_names=iris.feature_names[2:],
        class_names=iris.target_names,
        rounded=True,
        filled=True
    )

Source.from_file(os.path.join(IMAGES_PATH, "iris_tree_10f_md5_entropy.dot"))
```

[52]:



[53]:
```python
# Plot the decision boundry graph for the model
plt.figure(figsize=(8, 4))
plot_decision_boundary(tree_clf_10f_md5_ent, X, y)
plt.title('decision tree decision boundaries plot with 10 folds having max␣
 ↪depth 5 using entropy')
save_fig("decision_tree_decision_boundaries_plot_10f_md5_entropy")
plt.show()
```

```
Saving figure decision_tree_decision_boundaries_plot_10f_md5_entropy
```



9 Section 2 : Changing Depths and K-Folds

- Tree with max depth 3, 5 K-Folds and Criterion as Entropy
- Tree with max depth 3, 5 K-Folds and Criterion as Gini
- Tree with max depth 4, 7 K-Folds and Criterion as Entropy
- Tree with max depth 4, 7 K-Folds and Criterion as Gini
- Tree with max depth 5, 10 K-Folds and Criterion as Gini(Entropy already performed above)

9.0.1 Setup for K-fold = 5

[54]:
```
# Create the 5 fold cv
kf_5f = KFold(n_splits=5, random_state=None, shuffle=True)
kf_5f.get_n_splits(X)
print(kf_5f)
```

```
KFold(n_splits=5, random_state=None, shuffle=True)
```

9.0.2 K-FOLD Cross Validation (K-Folds = 5, Max Depth = 3) with Entropy

Apply K-fold and Classification Report

[55]:
```
# Validate output on the tree with depth 3 entropy
tree_clf_5f_md3_ent = DecisionTreeClassifier(max_depth=3, criterion="entropy",␣
↪random_state=42)

for train_index, test_index in kf_5f.split(X):
    #print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
```

```python
    y_train, y_test = y[train_index], y[test_index]

    tree_clf_5f_md3_ent.fit(X_train, y_train)

    y_pred = tree_clf_5f_md3_ent.predict(X_test)

    # Print classification report
    target_names = iris.target_names
    print(classification_report(y_test, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        12
  versicolor       1.00      0.67      0.80         9
   virginica       0.75      1.00      0.86         9

    accuracy                           0.90        30
   macro avg       0.92      0.89      0.89        30
weighted avg       0.93      0.90      0.90        30

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        11
  versicolor       0.91      0.91      0.91        11
   virginica       0.88      0.88      0.88         8

    accuracy                           0.93        30
   macro avg       0.93      0.93      0.93        30
weighted avg       0.93      0.93      0.93        30

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         7
  versicolor       1.00      0.93      0.97        15
   virginica       0.89      1.00      0.94         8

    accuracy                           0.97        30
   macro avg       0.96      0.98      0.97        30
weighted avg       0.97      0.97      0.97        30

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       0.88      1.00      0.93         7
   virginica       1.00      0.92      0.96        13

    accuracy                           0.97        30
   macro avg       0.96      0.97      0.96        30
```

```
weighted avg       0.97      0.97      0.97        30

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         8
   virginica       1.00      1.00      1.00        12

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

Cross Validation Score

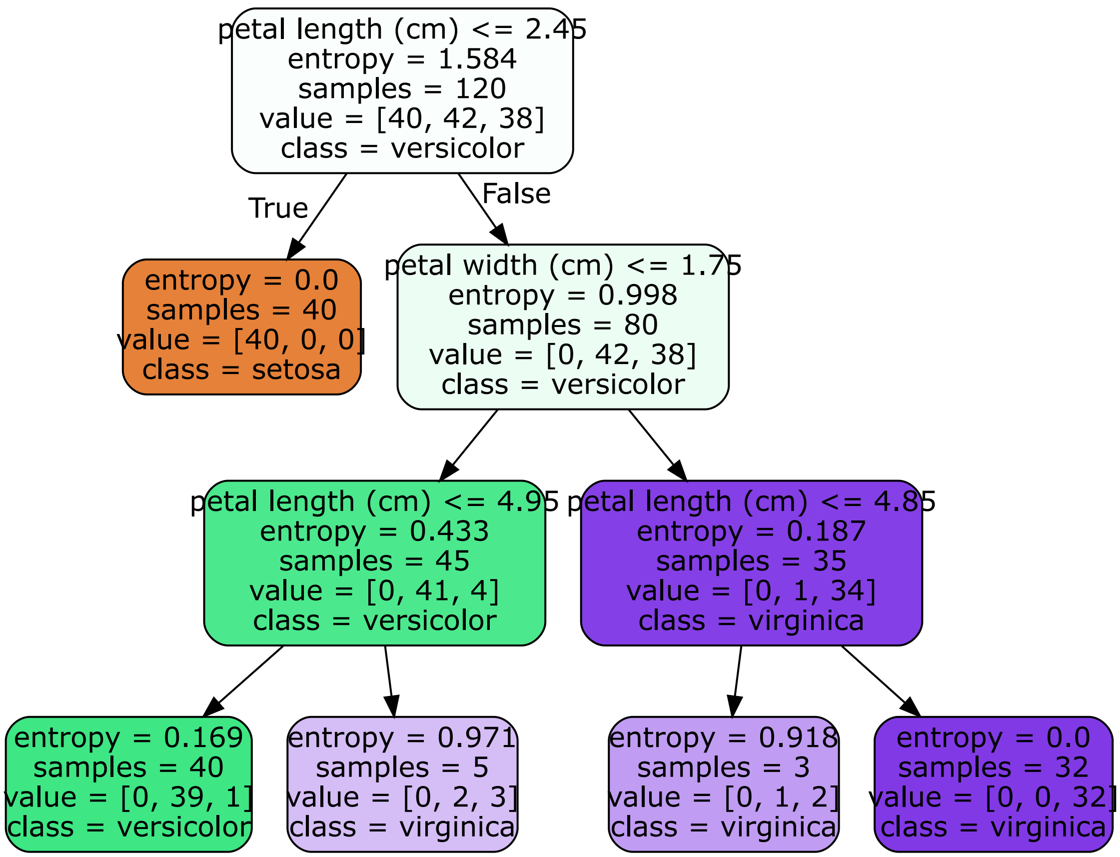
[56]:
```
cross_val_score(tree_clf_5f_md3_ent, iris.data, iris.target, cv=5)
```

[56]:
```
array([0.96666667, 0.96666667, 0.93333333, 0.93333333, 1.        ])
```

Graphviz

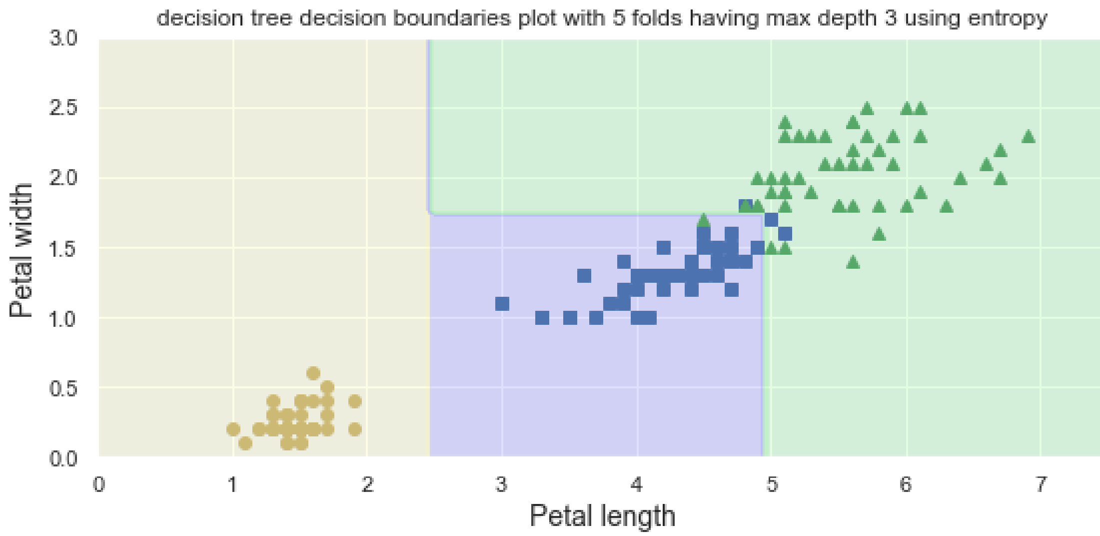
[57]:
```
export_graphviz(
        tree_clf_5f_md3_ent,
        out_file=os.path.join(IMAGES_PATH, "iris_tree_5f_md3_entropy.dot"),
        feature_names=iris.feature_names[2:],
        class_names=iris.target_names,
        rounded=True,
        filled=True
    )

Source.from_file(os.path.join(IMAGES_PATH, "iris_tree_5f_md3_entropy.dot"))
```

[57]:



[58]:
```
#Decision Boundary
plt.figure(figsize=(8, 4))
plot_decision_boundary(tree_clf_5f_md3_ent, X, y)
plt.title('decision tree decision boundaries plot with 5 folds having max depth␣
 ↪3 using entropy')
save_fig("decision_tree_decision_boundaries_plot_5f_md3_entropy")
plt.show()
```

```
Saving figure decision_tree_decision_boundaries_plot_5f_md3_entropy
```



9.0.3 K-FOLD Cross Validation (K-Folds = 5, Max Depth = 3) with Gini

Apply K-fold and Classification Report

[59]:
```
# Validate output on the tree with depth 3 gini

tree_clf_5f_md3_gini = DecisionTreeClassifier(max_depth=3, criterion="gini",␣
↪random_state=42)

for train_index, test_index in kf_5f.split(X):
    #print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    tree_clf_5f_md3_gini.fit(X_train, y_train)

    y_pred = tree_clf_5f_md3_gini.predict(X_test)

    # Print classification report
    target_names = iris.target_names
    print(classification_report(y_test, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         8
  versicolor       1.00      1.00      1.00        10
   virginica       1.00      1.00      1.00        12

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
```

```
weighted avg       1.00      1.00      1.00        30

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         6
  versicolor       1.00      0.92      0.96        12
   virginica       0.92      1.00      0.96        12

    accuracy                           0.97        30
   macro avg       0.97      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        15
  versicolor       0.88      1.00      0.93         7
   virginica       1.00      0.88      0.93         8

    accuracy                           0.97        30
   macro avg       0.96      0.96      0.96        30
weighted avg       0.97      0.97      0.97        30

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        15
  versicolor       0.78      1.00      0.88         7
   virginica       1.00      0.75      0.86         8

    accuracy                           0.93        30
   macro avg       0.93      0.92      0.91        30
weighted avg       0.95      0.93      0.93        30

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         6
  versicolor       0.93      0.93      0.93        14
   virginica       0.90      0.90      0.90        10

    accuracy                           0.93        30
   macro avg       0.94      0.94      0.94        30
weighted avg       0.93      0.93      0.93        30
```

Cross Validation Score

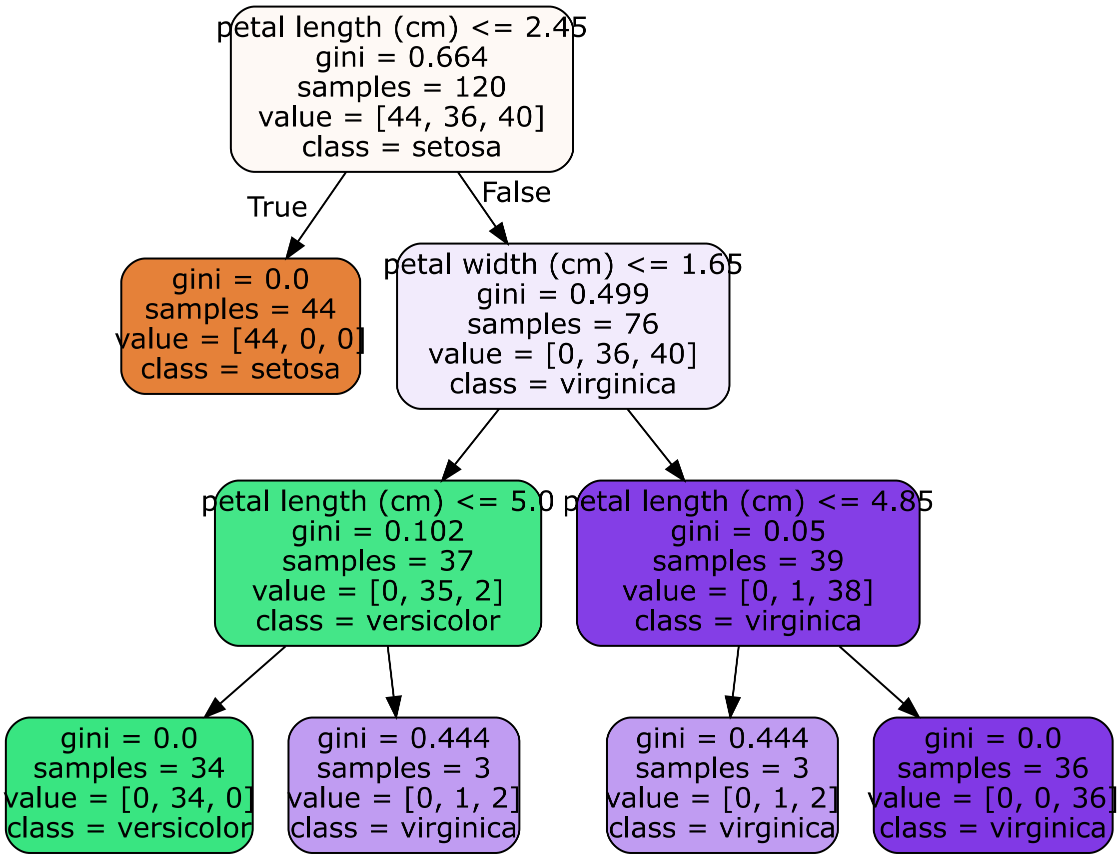
[60]:
```
cross_val_score(tree_clf_5f_md3_gini, iris.data, iris.target, cv=5)
```

[60]:
```
array([0.96666667, 0.96666667, 0.93333333, 1.        , 1.        ])
```

Graphviz

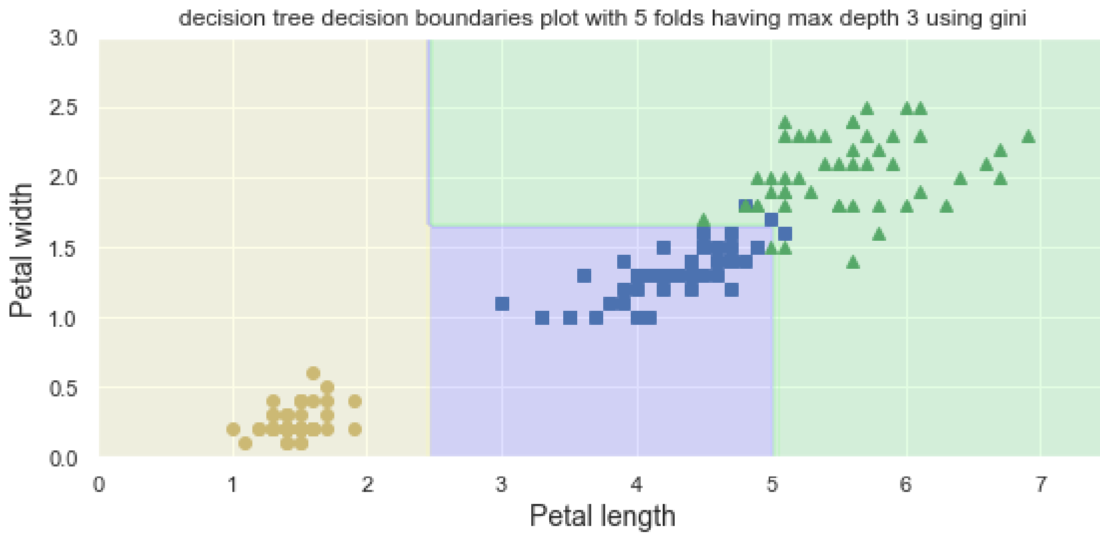
[61]:
```
export_graphviz(
        tree_clf_5f_md3_gini,
        out_file=os.path.join(IMAGES_PATH, "iris_tree_5f_md3_gini.dot"),
        feature_names=iris.feature_names[2:],
        class_names=iris.target_names,
        rounded=True,
        filled=True
    )

Source.from_file(os.path.join(IMAGES_PATH, "iris_tree_5f_md3_gini.dot"))
```

[61]:



[62]:
```
#Decision Boundary
plt.figure(figsize=(8, 4))
plot_decision_boundary(tree_clf_5f_md3_gini, X, y)
plt.title('decision tree decision boundaries plot with 5 folds having max depth␣
 ↪3 using gini')
save_fig("decision_tree_decision_boundaries_plot_5f_md3_gini")
plt.show()
```

```
Saving figure decision_tree_decision_boundaries_plot_5f_md3_gini
```



9.0.4 Setup for K-fold = 7

[63]:
```
# Create the 7 fold cv
kf_7f = KFold(n_splits=7, random_state=None, shuffle=True)
kf_7f.get_n_splits(X)
print(kf_7f)
```

```
KFold(n_splits=7, random_state=None, shuffle=True)
```

9.0.5 K-FOLD Cross Validation (K-Folds = 7, Max Depth = 4) with Entropy

Apply K-fold and Classification Report

[64]:
```
# Validate output on the tree with depth 4 Entropy
tree_clf_7f_md4_ent = DecisionTreeClassifier(max_depth=4, criterion="entropy",␣
↪random_state=42)

for train_index, test_index in kf_7f.split(X):
    #print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    tree_clf_7f_md4_ent.fit(X_train, y_train)

    y_pred = tree_clf_7f_md4_ent.predict(X_test)

    # Print classification report
    target_names = iris.target_names
    print(classification_report(y_test, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         8
  versicolor       0.57      1.00      0.73         4
   virginica       1.00      0.70      0.82        10

    accuracy                           0.86        22
   macro avg       0.86      0.90      0.85        22
weighted avg       0.92      0.86      0.87        22

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         6
  versicolor       0.91      0.91      0.91        11
   virginica       0.80      0.80      0.80         5

    accuracy                           0.91        22
   macro avg       0.90      0.90      0.90        22
weighted avg       0.91      0.91      0.91        22

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         6
  versicolor       0.89      1.00      0.94         8
   virginica       1.00      0.88      0.93         8

    accuracy                           0.95        22
   macro avg       0.96      0.96      0.96        22
weighted avg       0.96      0.95      0.95        22

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         9
  versicolor       1.00      0.75      0.86         4
   virginica       0.89      1.00      0.94         8

    accuracy                           0.95        21
   macro avg       0.96      0.92      0.93        21
weighted avg       0.96      0.95      0.95        21

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         8
  versicolor       1.00      0.88      0.93         8
   virginica       0.83      1.00      0.91         5

    accuracy                           0.95        21
   macro avg       0.94      0.96      0.95        21
```

```
weighted avg       0.96      0.95      0.95        21

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         3
  versicolor       0.92      1.00      0.96        11
   virginica       1.00      0.86      0.92         7

    accuracy                           0.95        21
   macro avg       0.97      0.95      0.96        21
weighted avg       0.96      0.95      0.95        21

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         4
   virginica       1.00      1.00      1.00         7

    accuracy                           1.00        21
   macro avg       1.00      1.00      1.00        21
weighted avg       1.00      1.00      1.00        21
```

Cross Validation Score

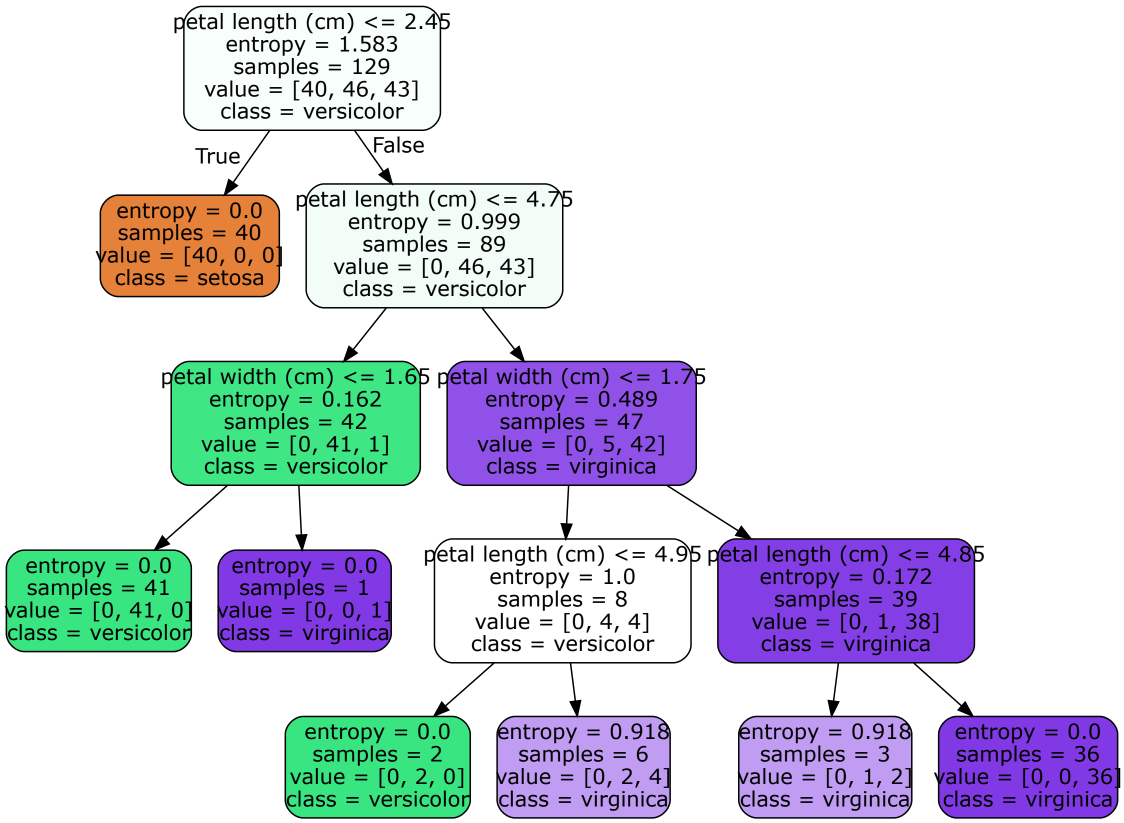
[65]:
```
cross_val_score(tree_clf_7f_md4_ent, iris.data, iris.target, cv=7)
```

[65]:
```
array([0.95454545, 0.95454545, 0.90909091, 0.85714286, 0.95238095,
       1.        , 1.        ])
```

Graphviz

[66]:
```
export_graphviz(
        tree_clf_7f_md4_ent,
        out_file=os.path.join(IMAGES_PATH, "iris_tree_7f_md4_entropy.dot"),
        feature_names=iris.feature_names[2:],
        class_names=iris.target_names,
        rounded=True,
        filled=True
    )

Source.from_file(os.path.join(IMAGES_PATH, "iris_tree_7f_md4_entropy.dot"))
```

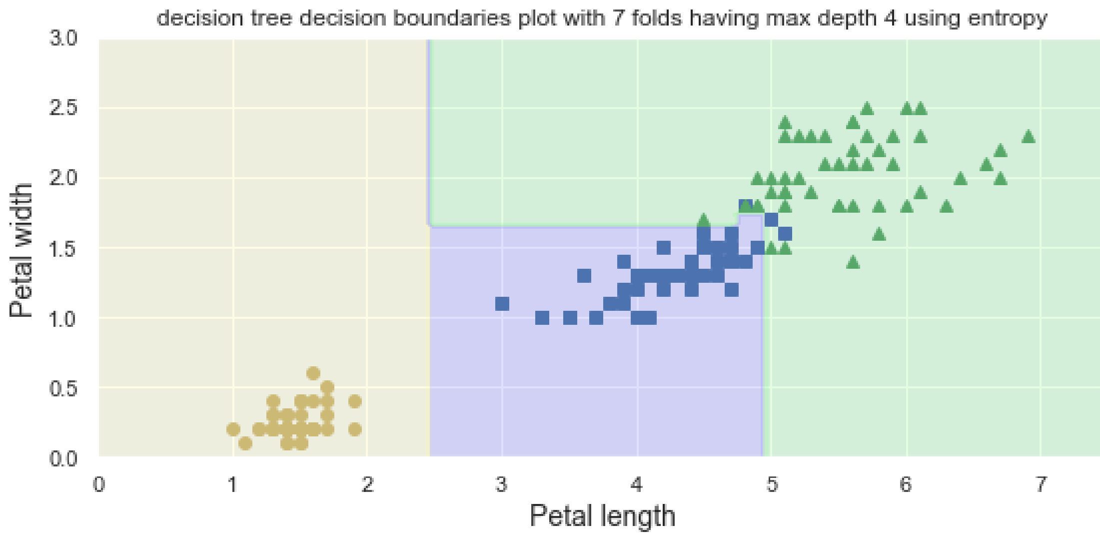
[66]:



Decision Boundary

[67]:
```
plt.figure(figsize=(8, 4))
plot_decision_boundary(tree_clf_7f_md4_ent, X, y)
plt.title('decision tree decision boundaries plot with 7 folds having max depth␣
 ↪4 using entropy')
save_fig("decision_tree_decision_boundaries_plot_7f_md4_entropy")
plt.show()
```

```
Saving figure decision_tree_decision_boundaries_plot_7f_md4_entropy
```



9.0.6 K-FOLD Cross Validation (K-Folds = 7, Max Depth = 4) with Gini

Apply K-fold and Classification Report

[68]:
```
# Validate output on the tree with depth 4 gini

tree_clf_7f_md4_gini = DecisionTreeClassifier(max_depth=4, criterion="gini",␣
↪random_state=42)

for train_index, test_index in kf_7f.split(X):
    #print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    tree_clf_7f_md4_gini.fit(X_train, y_train)

    y_pred = tree_clf_7f_md4_gini.predict(X_test)

    # Print classification report
    target_names = iris.target_names
    print(classification_report(y_test, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         7
  versicolor       0.88      1.00      0.93         7
   virginica       1.00      0.88      0.93         8

    accuracy                           0.95        22
   macro avg       0.96      0.96      0.96        22
```

```
weighted avg       0.96      0.95      0.95        22

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      0.75      0.86         4
   virginica       0.89      1.00      0.94         8

    accuracy                           0.95        22
   macro avg       0.96      0.92      0.93        22
weighted avg       0.96      0.95      0.95        22

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         9
  versicolor       0.86      0.75      0.80         8
   virginica       0.67      0.80      0.73         5

    accuracy                           0.86        22
   macro avg       0.84      0.85      0.84        22
weighted avg       0.87      0.86      0.87        22

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         6
  versicolor       1.00      1.00      1.00         8
   virginica       1.00      1.00      1.00         7

    accuracy                           1.00        21
   macro avg       1.00      1.00      1.00        21
weighted avg       1.00      1.00      1.00        21

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         6
  versicolor       0.88      1.00      0.93         7
   virginica       1.00      0.88      0.93         8

    accuracy                           0.95        21
   macro avg       0.96      0.96      0.96        21
weighted avg       0.96      0.95      0.95        21

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         6
  versicolor       0.88      1.00      0.93         7
   virginica       1.00      0.88      0.93         8
```

```
    accuracy                           0.95        21
   macro avg       0.96      0.96      0.96        21
weighted avg       0.96      0.95      0.95        21

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         6
  versicolor       0.82      1.00      0.90         9
   virginica       1.00      0.67      0.80         6

    accuracy                           0.90        21
   macro avg       0.94      0.89      0.90        21
weighted avg       0.92      0.90      0.90        21
```

Cross Validation Score

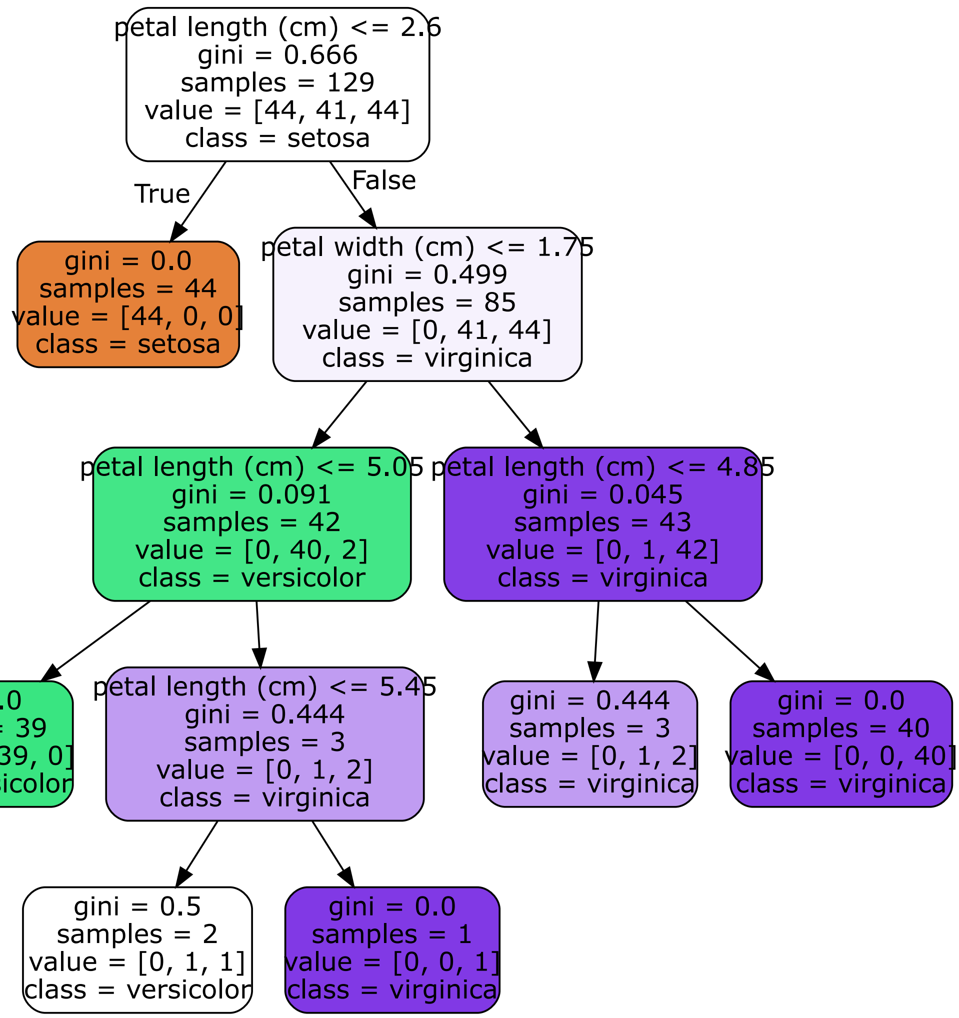
[69]:
```
cross_val_score(tree_clf_7f_md4_gini, iris.data, iris.target, cv=7)
```

[69]:
```
array([0.95454545, 0.95454545, 0.90909091, 0.85714286, 0.95238095,
       1.        , 1.        ])
```

Graphviz

[70]:
```
export_graphviz(
        tree_clf_7f_md4_gini,
        out_file=os.path.join(IMAGES_PATH, "iris_tree_7f_md4_gini.dot"),
        feature_names=iris.feature_names[2:],
        class_names=iris.target_names,
        rounded=True,
        filled=True
    )


Source.from_file(os.path.join(IMAGES_PATH, "iris_tree_7f_md4_gini.dot"))
```

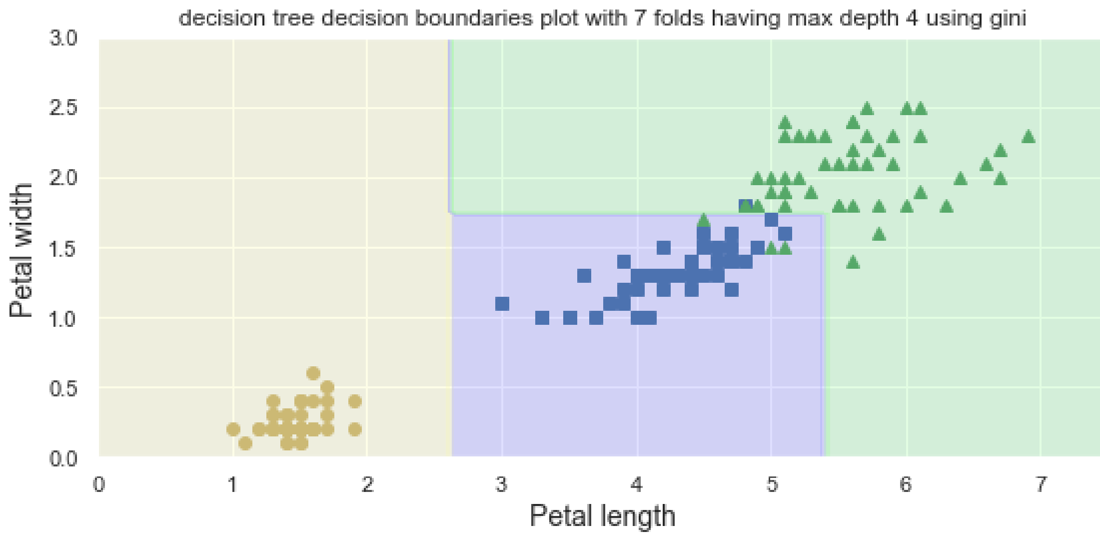
[70]:



Decision Boundary

[71]:
```
plt.figure(figsize=(8, 4))
plot_decision_boundary(tree_clf_7f_md4_gini, X, y)
plt.title('decision tree decision boundaries plot with 7 folds having max depth␣
↪4 using gini')
save_fig("decision_tree_decision_boundaries_plot_7f_md4_gini")
plt.show()
```

```
Saving figure decision_tree_decision_boundaries_plot_7f_md4_gini
```



9.0.7 Setup for K-fold = 10

[72]:
```
# Create the 10 fold cv
kf_10f = KFold(n_splits=10, random_state=None, shuffle=True)
kf_10f.get_n_splits(X)
print(kf_10f)
```

```
KFold(n_splits=10, random_state=None, shuffle=True)
```

9.0.8 K-FOLD Cross Validation (K-Folds = 10, Max Depth = 5) with Gini

Apply K-fold and Classification Report

[73]:
```
# Validate output on the tree with depth 5 gini
tree_clf_10f_md5_gini = DecisionTreeClassifier(max_depth=5, criterion="gini",␣
↪random_state=42)

for train_index, test_index in kf_10f.split(X):
    #print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    tree_clf_10f_md5_gini.fit(X_train, y_train)

    y_pred = tree_clf_10f_md5_gini.predict(X_test)

    # Print classification report
    target_names = iris.target_names
    print(classification_report(y_test, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         6
  versicolor       1.00      1.00      1.00         3
   virginica       1.00      1.00      1.00         6

    accuracy                           1.00        15
   macro avg       1.00      1.00      1.00        15
weighted avg       1.00      1.00      1.00        15

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         5
  versicolor       1.00      1.00      1.00         6
   virginica       1.00      1.00      1.00         4

    accuracy                           1.00        15
   macro avg       1.00      1.00      1.00        15
weighted avg       1.00      1.00      1.00        15

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         2
  versicolor       0.78      1.00      0.88         7
   virginica       1.00      0.67      0.80         6

    accuracy                           0.87        15
   macro avg       0.93      0.89      0.89        15
weighted avg       0.90      0.87      0.86        15

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         5
  versicolor       1.00      1.00      1.00         5
   virginica       1.00      1.00      1.00         5

    accuracy                           1.00        15
   macro avg       1.00      1.00      1.00        15
weighted avg       1.00      1.00      1.00        15

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         3
  versicolor       1.00      0.86      0.92         7
   virginica       0.83      1.00      0.91         5

    accuracy                           0.93        15
   macro avg       0.94      0.95      0.94        15
```

```
weighted avg       0.94      0.93      0.93        15

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         7
  versicolor       1.00      1.00      1.00         6
   virginica       1.00      1.00      1.00         2

    accuracy                           1.00        15
   macro avg       1.00      1.00      1.00        15
weighted avg       1.00      1.00      1.00        15

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         7
  versicolor       1.00      1.00      1.00         4
   virginica       1.00      1.00      1.00         4

    accuracy                           1.00        15
   macro avg       1.00      1.00      1.00        15
weighted avg       1.00      1.00      1.00        15

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         3
  versicolor       1.00      0.75      0.86         8
   virginica       0.67      1.00      0.80         4

    accuracy                           0.87        15
   macro avg       0.89      0.92      0.89        15
weighted avg       0.91      0.87      0.87        15

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         6
  versicolor       0.80      1.00      0.89         4
   virginica       1.00      0.80      0.89         5

    accuracy                           0.93        15
   macro avg       0.93      0.93      0.93        15
weighted avg       0.95      0.93      0.93        15

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         6
  versicolor       0.00      0.00      0.00         0
   virginica       1.00      0.78      0.88         9
```

```
    accuracy                           0.87        15
   macro avg       0.67      0.59      0.62        15
weighted avg       1.00      0.87      0.93        15
```

```
/Users/anjali/anaconda3/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1248: UndefinedMetricWarning: Recall
and F-score are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/anjali/anaconda3/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1248: UndefinedMetricWarning: Recall
and F-score are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/anjali/anaconda3/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1248: UndefinedMetricWarning: Recall
and F-score are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Cross Validation Score

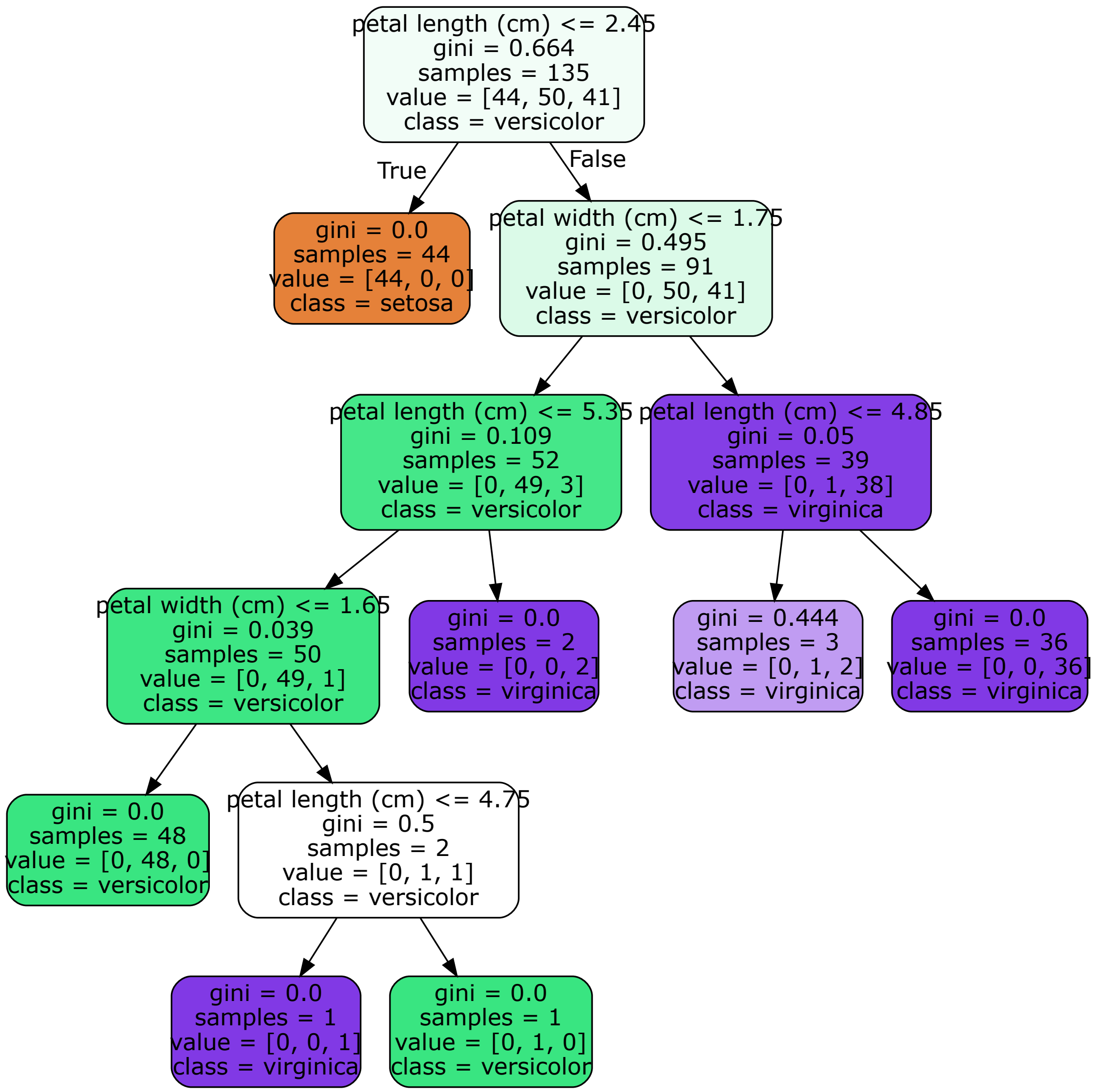
[74]:
```
cross_val_score(tree_clf_10f_md5_gini, iris.data, iris.target, cv=10)
```

[74]:
```
array([1.        , 0.93333333, 1.        , 0.93333333, 0.93333333,
       0.86666667, 0.93333333, 0.93333333, 1.        , 1.        ])
```

Graphviz

[75]:
```
export_graphviz(
        tree_clf_10f_md5_gini,
        out_file=os.path.join(IMAGES_PATH, "iris_tree_10f_md5_gini.dot"),
        feature_names=iris.feature_names[2:],
        class_names=iris.target_names,
        rounded=True,
        filled=True
    )

Source.from_file(os.path.join(IMAGES_PATH, "iris_tree_10f_md5_gini.dot"))
```

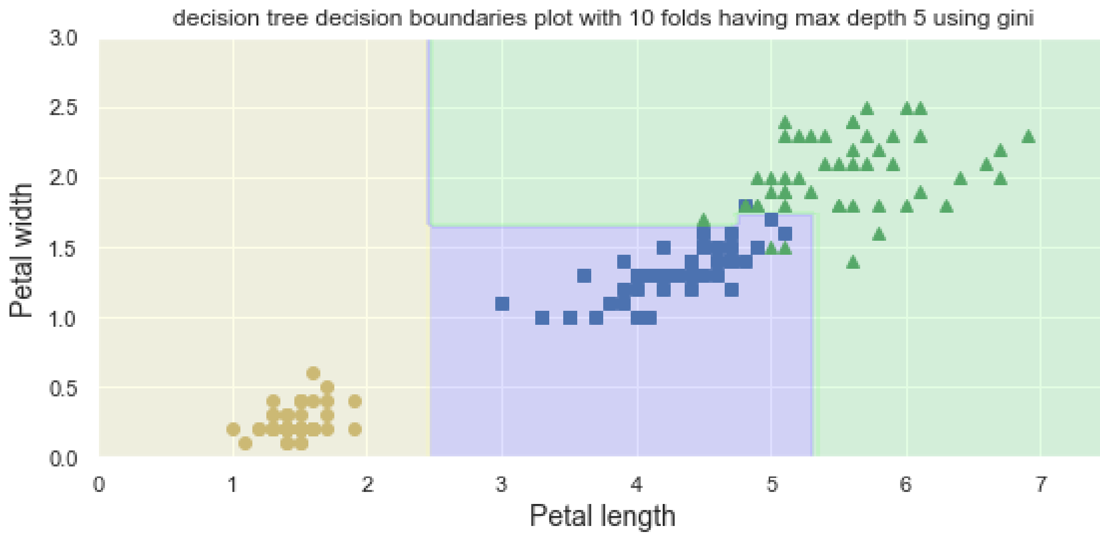
[75]:



Decision Boundary

[76]:
```
plt.figure(figsize=(8, 4))
plot_decision_boundary(tree_clf_10f_md5_gini, X, y)
plt.title('decision tree decision boundaries plot with 10 folds having max␣
 ↪depth 5 using gini')
save_fig("decision_tree_decision_boundaries_plot_10f_md5_gini")
plt.show()
```

```
Saving figure decision_tree_decision_boundaries_plot_10f_md5_gini
```



9.0.9 Explain your conclusions on increasing the depth and increasing the number of folds.

- We increased the number of folds from 5 -> 7 -> 10 and max depth from 3->4->5 for the criterion-entropy and gini separately
- Entropy
  - Yellow(Setosa) is correctly classified and isn't impacted by the increase in folds or max depth
  - When we increased the depth and fold from 5 to 7 folds and Depth from 3 to 4
  - Few points at the intersection of blue (versicolor) and green (virginica) are now correctly classified. But the increase is very slight compared to the computational resources being used to run the folds and go one step more in depth.
  - When we increased the folds from 7 to 10 and depth from 4 to 5, the decision boundaries for blue (versicolor) went overboard, and in order to correctly classify it, there are few green region points (virginica) which are now misclassified.
  - Also we are seeing a region where decision boundaries have become narrow and potential overfitting might occur.
- Gini
  - Yellow(Setosa) is correctly classified and isn't impacted by the increase in folds or max depth.
  - On increasing folds from 5 to 7 and max depth from 3->4 , there the misclassification between blue(versicolor) and green (virginica) has interchanged without any considerable information gain. For the change of folds from 7 to 10 and max depth from 4 to 5, there is a slightly better classification between versicolor and virginica.
- Overall : The overfitting seems to be less when using gini index in comparison to the entropy information gain method