

✓ Assignment: Decision Tree

In this assignment, we will first learn how to train decision trees and we will use `sklearn` python package. It is recommended to use python 3.5 version or greater. Your `sklearn` version should be v0.21.3 or greater. Some functions will not run in older versions.

We will use `sklearn` package in `ipython jupyter` notebook for this assignment.

✓ Decision Trees : sklearn

Go to <https://scikit-learn.org/stable/modules/tree.html>. The page contains nice description of decision trees. The try to run the following code and modify the `ipython` notebook to answer few questions. You can insert markdown cells for answers, or code cells for programming. Run the cells one by one. Inspect the code. Write code and answers following the instructions.

```
# Import necessary Packages
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import sklearn
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
print('The scikit-learn version is {}'.format(sklearn.__version__))
```

→ The scikit-learn version is 1.5.2.

```
#Load iris data
iris = load_iris()

# check what are the features
print("Iris data Feature Names : ", iris.feature_names)

# print the detailed description
print(iris.DESCR)

# load the data and target
X = iris.data
y = iris.target
```

→ Iris data Feature Names : ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
.. _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.

```
.. dropdown:: 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

Question 1. Study the code above. Write code to know how many classes are there in target y. Print the number of classes in the dataset in the cell. Print how many instances are there for each class. Print the number of total instances.

```
classes,counts=np.unique(y,return_counts=True)
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X,y)
print(f"the number of classes in the dataset is {len(classes)} with len")
print(f"the number of classes in the dataset is {clf.n_classes_} with n_classes_.")
for class_label, count in zip(classes,counts):
    print(f"the number of instances for class {class_label} is {count}")
print(f"the number of total instances is {len(y)}")
```

```
→ the number of classes in the dataset is 3 with len
the number of classes in the dataset is 3 with n_classes_.
the number of instances for class 0 is 50
the number of instances for class 1 is 50
the number of instances for class 2 is 50
the number of total instances is 150
```

The following code takes first two features and creates the decision boundary. Study the code, try to understand each line, run the code and answer the questions. You may modify the code (print a few lines) to understand it better.

```
n_classes = 3
plot_colors = "ryb"

feature_pair = [0,1]
X = iris.data[:, feature_pair]
y = iris.target

# instantiate the decision tree
clf = DecisionTreeClassifier()
clf.fit(X, y)

# Plot the decision boundary
plt.plot()

x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.05),
                     np.arange(y_min, y_max, 0.05))


Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
cs = plt.contourf(xx, yy, Z, cmap=plt.cm.Paired)

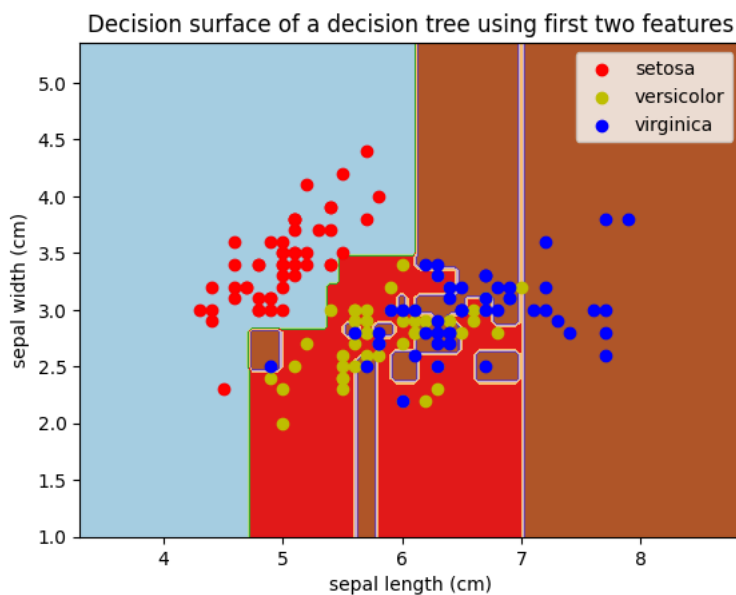
plt.xlabel(iris.feature_names[feature_pair[0]])
plt.ylabel(iris.feature_names[feature_pair[1]])

# Plot the training points
for i, color in zip(range(n_classes), plot_colors):
```

```
idx = np.where(y == i)
plt.scatter(X[idx, 0], X[idx, 1], c=color, label=iris.target_names[i],
            cmap=plt.cm.Paired)
```

```
plt.title("Decision surface of a decision tree using first two features")
plt.legend()
plt.show()
```

 <ipython-input-22-f788d25b60d0>:31: UserWarning: No data for colormapping provided via 'c'. Parameters 'cmap' will be ignored



Question 2. Copy the above code below and modify it to train and display decision boundaries for the last two features.

```
# Answer for question 2 :
# Copy the above code below and modify it
# write your code here
n_classes = 3
plot_colors = "ryb"

feature_pair = [-2,-1]
X = iris.data[:, feature_pair]
y = iris.target

# instantiate the decision tree
clf = DecisionTreeClassifier()
clf.fit(X, y)

# Plot the decision boundary
plt.plot()

x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.05),
                     np.arange(y_min, y_max, 0.05))


Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
cs = plt.contourf(xx, yy, Z, cmap=plt.cm.Paired)

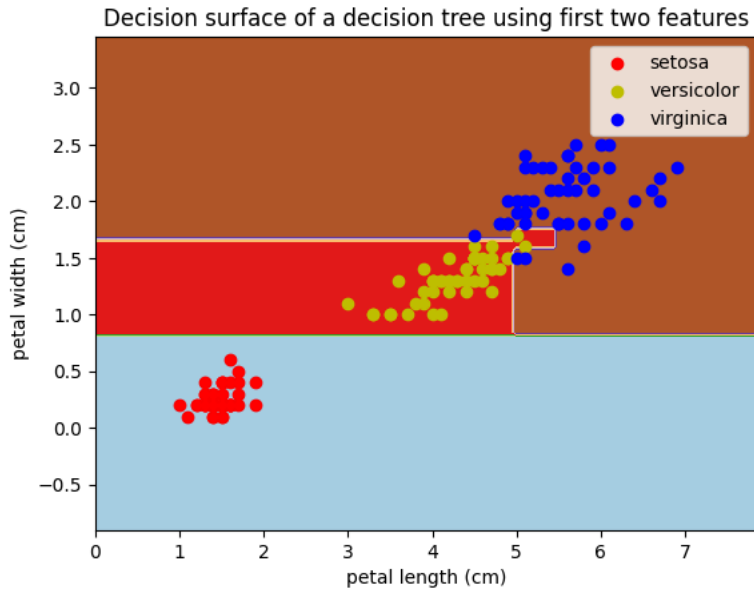
plt.xlabel(iris.feature_names[feature_pair[0]])
plt.ylabel(iris.feature_names[feature_pair[1]])

# Plot the training points
for i, color in zip(range(n_classes), plot_colors):
    idx = np.where(y == i)
    plt.scatter(X[idx, 0], X[idx, 1], c=color, label=iris.target_names[i],
                cmap=plt.cm.Paired)

plt.title("Decision surface of a decision tree using first two features")
```


```
plt.legend()
plt.show()
```

 <ipython-input-25-e1e071e4ed17>:34: UserWarning: No data for colormapping provided via 'c'. Parameters 'cmap' will be ignored
plt.scatter(X[idx, 0], X[idx, 1], c=color, label=iris.target_names[i],



Question 3. Write code to train the decision tree using all 4 attributes.

```
X=iris.data
y=iris.target
clf=DecisionTreeClassifier()
clf.fit(X,y)
accuracy=clf.score(X,y)
print(f"Training accuracy of decision tree using all 4 features--{accuracy}")
```

 Training accuracy of decision tree using all 4 features--1.0

▼ Printing Decision Trees

Let us again train decision tree using the first two features and print it. Run the following code. Your `sklearn` version should be v0.21.3 or greater

```
import sklearn
print('The scikit-learn version is {}'.format(sklearn.__version__))

from sklearn import tree

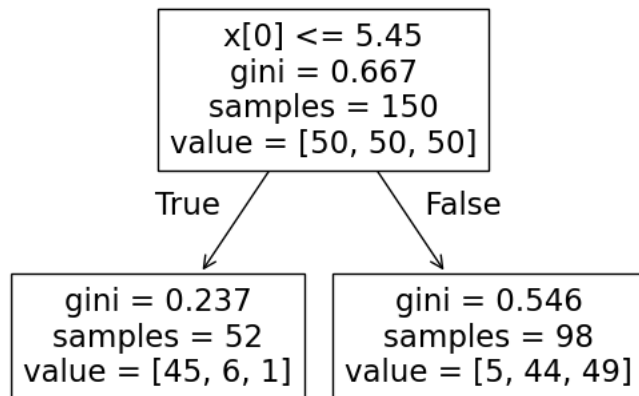
feature_pair = [0,1]
X = iris.data[:, feature_pair]
y = iris.target

# instantiate the decision tree
clf = DecisionTreeClassifier(max_depth=1)
tree.plot_tree(clf.fit(X, y))
```

```

The scikit-learn version is 1.5.2.
[Text(0.5, 0.75, 'x[0] <= 5.45\ngini = 0.667\nsamples = 150\nvalue = [50, 50, 50]'),
 Text(0.25, 0.25, 'gini = 0.237\nsamples = 52\nvalue = [45, 6, 1]'),
 Text(0.375, 0.5, 'True '),
 Text(0.75, 0.25, 'gini = 0.546\nsamples = 98\nvalue = [5, 44, 49]'),
 Text(0.625, 0.5, ' False')]

```



Question 4. (a) What does the max_depth parameter do? Go to <https://scikit-learn.org/stable/modules/tree.html> to study different algorithms :ID3, C4.5, C5.0 and CART (b) Which algorithm is used in sklearn? (c) Which criterion is used to build the decision tree?

Answer: max depth controls how deep the tree will be. Lower the max depth to prevent overfitting and increase the max depth to prevent underfitting.

CART (Classification and Regression Trees) is used in sklearn. The page states, "scikit-learn uses an optimized version of the CART algorithm; however, the scikit-learn implementation does not support categorical variables for now."

sk learn uses the gini criterion to build a decision tree. <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html> states "criterion{'gini', 'entropy', 'log_loss'}, default='gini' The function to measure the quality of a split. Supported criteria are 'gini' for the Gini impurity and 'log_loss' and 'entropy' both for the Shannon information gain, see Mathematical formulation."

Question 5 Modify the following code to change the following : maximum depth = 1 and entropy criterion.

```

n_classes = 3
plot_colors = "ryb"

feature_pair = [0,1]
X = iris.data[:, feature_pair]
y = iris.target

# instantiate the decision tree
clf = DecisionTreeClassifier(max_depth=1,criterion="entropy")
clf.fit(X, y)

# Plot the decision boundary
plt.plot()

x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                     np.arange(y_min, y_max, 0.01))


Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
cs = plt.contourf(xx, yy, Z, cmap=plt.cm.Paired)

plt.xlabel(iris.feature_names[feature_pair[0]])
plt.ylabel(iris.feature_names[feature_pair[1]])

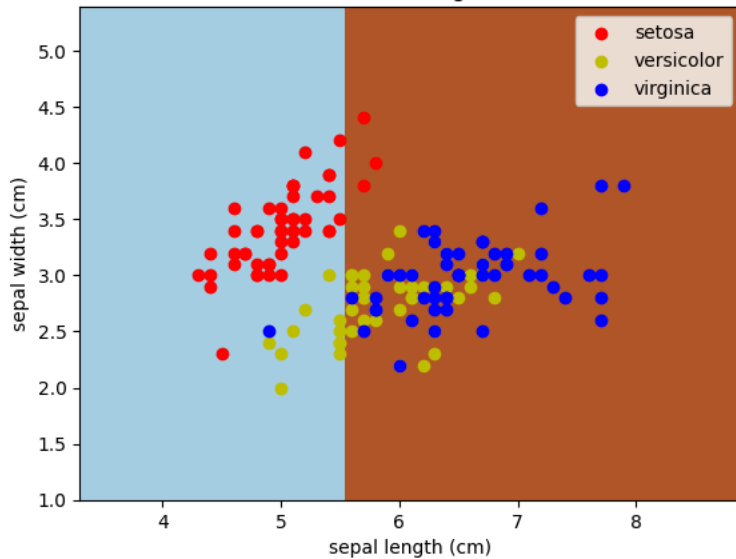
```

```
# Plot the training points
for i, color in zip(range(n_classes), plot_colors):
    idx = np.where(y == i)
    plt.scatter(X[idx, 0], X[idx, 1], c=color, label=iris.target_names[i],
               cmap=plt.cm.Paired)
```

```
plt.title("Decision surface of a decision tree using first two features and entropy")
plt.legend()
plt.show()
```


 <ipython-input-39-93ac45cb1877>:31: UserWarning: No data for colormapping provided via 'c'. Parameters 'cmap' will be ignored
plt.scatter(X[idx, 0], X[idx, 1], c=color, label=iris.target_names[i],

Decision surface of a decision tree using first two features and entropy



Question 6 Write a code to print the decision tree using all the four attributes in the training set use : max_depth = 2 and entropy criterion

```
# Answer for question 6 :
# write your code here
X=iris.data
y=iris.target
clf=DecisionTreeClassifier(max_depth=2,criterion="entropy")
clf.fit(X,y)
accuracy=clf.score(X,y)
print(f"Training accuracy of decision tree using all 4 features--{accuracy}")
```

 Training accuracy of decision tree using all 4 features--0.96

✓ Train-Test Split

Study the code and understand what is being done.

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

iris = load_iris()
X, y = iris.data, iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

clf = tree.DecisionTreeClassifier()
clf = clf.fit(X_train, y_train)

predicted = clf.predict(X_train)
```

Question 7 Write a code to to print accuracy of the model.

```

train_accuracy=clf.score(X_train,y_train)
test_accuracy=clf.score(X_test,y_test)
print(f"Training accuracy of decision tree using all 4 features--{train_accuracy}")
print(f"Testing accuracy of decision tree using all 4 features--{test_accuracy}")

```

↗ Training accuracy of decision tree using all 4 features--1.0
Testing accuracy of decision tree using all 4 features--0.98

✓ Finding the best Tree

Question 8. Perform a comparative study of a decision tree on Iris data using different parameters. Split the data into train and test. Experiment with `criterion` (default is `gini`, try to train using `entropy`), `max_depth` (try to set it to 2, 3, 4), `min_samples_leaf` (default is 1, try with 5, 15). Objective is to find a combination of parameters given below that improves accuracy on the test set. Report what you found. Print the decision boundaries, and the tree in each case. You have to use the given structure to use different parameters (Read about List, dictionary and Tuple in python). Run the following code and try to understand how to iterate such data structure.

```

#
parameters = {
    'criterion' : ('gini', 'entropy'),
    'max_depth' : [2,3,4],
    'min_samples_leaf' : [1,5,15]
}

print("The variable parameter is a dictionary with keys : ", parameters.keys())
print("-----Printing keys-----")
for key in parameters.keys():
    print(key)

print("-----Printing Criterion : a Tuple -----")
for cr in parameters["criterion"]:
    print(cr)

print("-----Printing max_depth : a List -----")
for d in parameters["max_depth"]:
    print(d)

print("-----Printing min_samples_leaf : a List -----")
for l in parameters["min_samples_leaf"]:
    print(l)

↗ The variable parameter is a dictionary with keys : dict_keys(['criterion', 'max_depth', 'min_samples_leaf'])
-----Printing keys-----
criterion
max_depth
min_samples_leaf
-----Printing Criterion : a Tuple -----
gini
entropy
-----Printing max_depth : a List -----
2
3
4
-----Printing min_samples_leaf : a List -----
1
5
15

```

Experiments

This code will output the decision boundary plot, the tree, and the accuracy for all combinations. The accuracy is part of the decision boundary plot. There is a lot of output, so the output is hidden. Click on the button in the upper left that says show/hide output to see the output.

```

# Importing necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score
from itertools import product

```

```

# Loading Iris data
iris = load_iris()
X = iris.data[:, :2] # We use only the first two features for visualization
y = iris.target

# Splitting data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Defining the parameter grid
parameters = {
    'criterion': ('gini', 'entropy'),
    'max_depth': [2, 3, 4],
    'min_samples_leaf': [1, 5, 15]
}

# To store accuracy and corresponding parameters
results = []

# Modify the function to print accuracy along with plotting decision boundaries and trees
def plot_decision_boundary_with_accuracy(clf, X, y, title, accuracy):
    # Decision boundary plot
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                          np.arange(y_min, y_max, 0.02))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.contourf(xx, yy, Z, alpha=0.8)
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', marker='o')
    plt.title(f'{title} (Accuracy: {accuracy:.2f})')
    plt.show()

# Iterate through all combinations of parameters and plot along with accuracy
for criterion, max_depth, min_samples_leaf in product(parameters['criterion'], parameters['max_depth'], parameters['min_samples_leaf']):

    # Initialize the Decision Tree Classifier
    clf = DecisionTreeClassifier(criterion=criterion, max_depth=max_depth, min_samples_leaf=min_samples_leaf, random_state=42)

    # Train the classifier
    clf.fit(X_train, y_train)

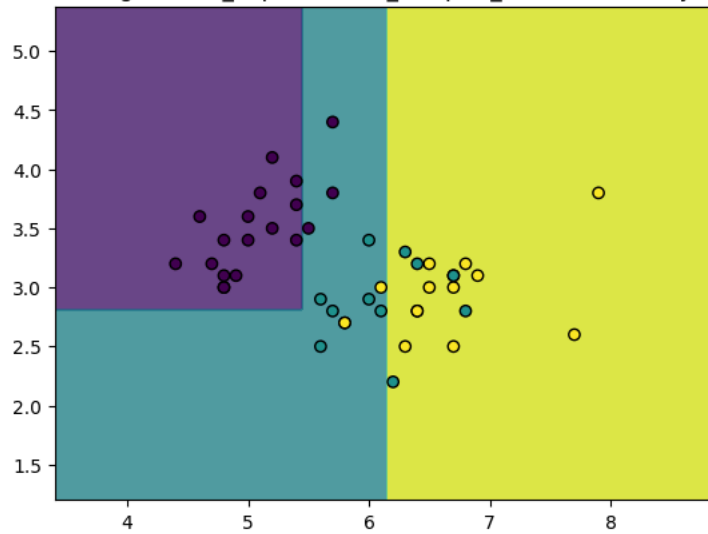
    # Predict and compute accuracy on test set
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)

    # Plot the decision boundary with accuracy
    plot_decision_boundary_with_accuracy(clf, X_test, y_test, f'criterion={criterion}, max_depth={max_depth}, min_samples_leaf={min_samples_leaf}', accuracy)

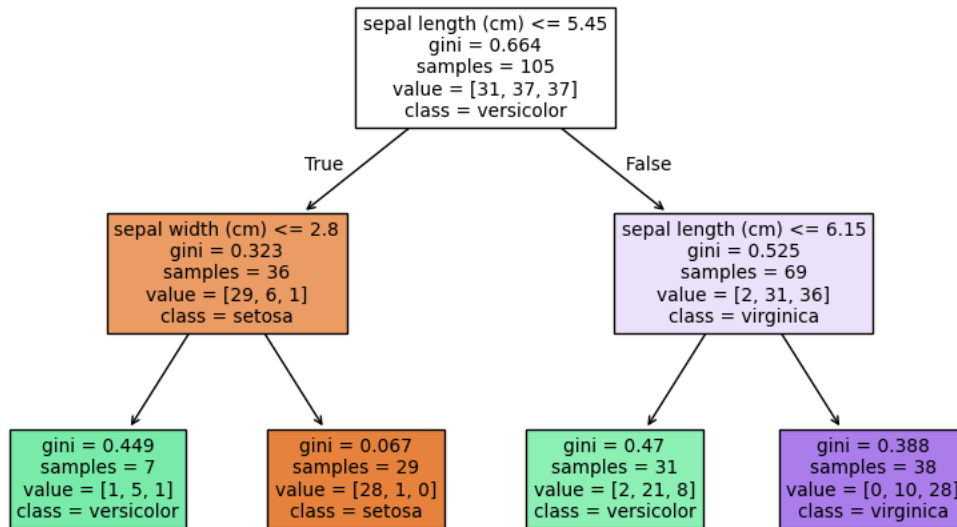
    # Plot the tree
    plt.figure(figsize=(10, 6))
    plot_tree(clf, filled=True, feature_names=iris.feature_names[:2], class_names=iris.target_names)
    plt.title(f'Tree with criterion={criterion}, max_depth={max_depth}, min_samples_leaf={min_samples_leaf}')
    plt.show()

```

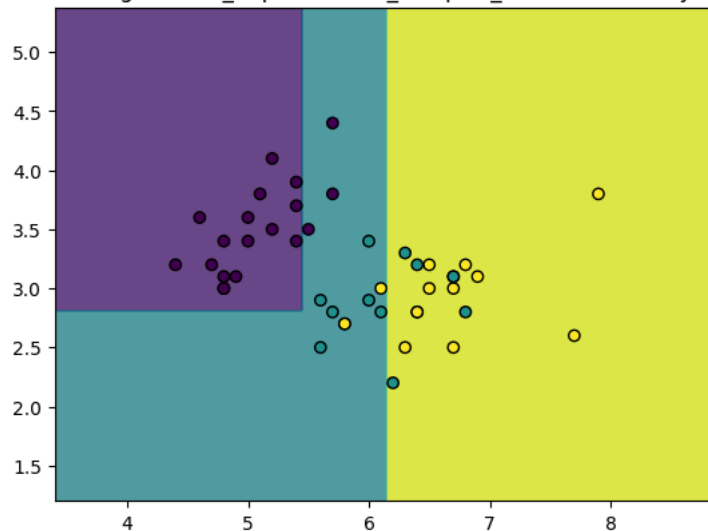

↻ criterion=gini, max_depth=2, min_samples_leaf=1 (Accuracy: 0.76)



Tree with criterion=gini, max_depth=2, min_samples_leaf=1

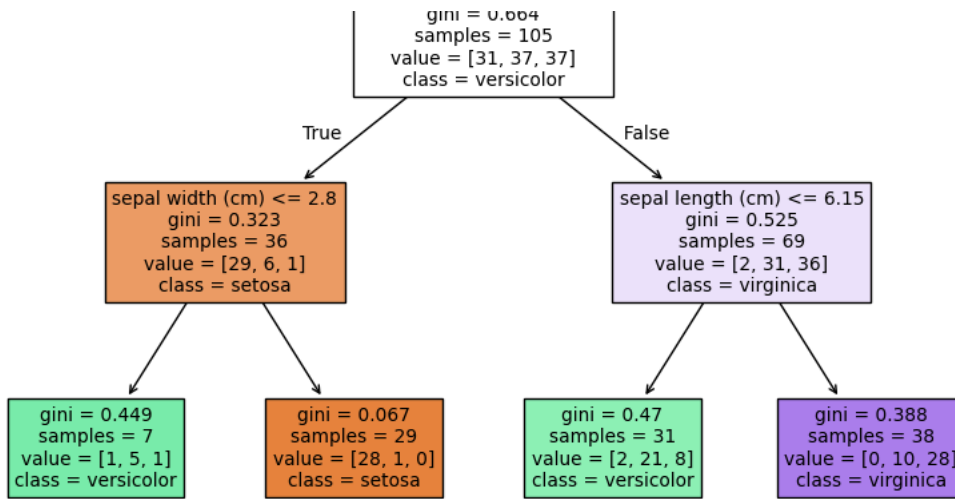


criterion=gini, max_depth=2, min_samples_leaf=5 (Accuracy: 0.76)

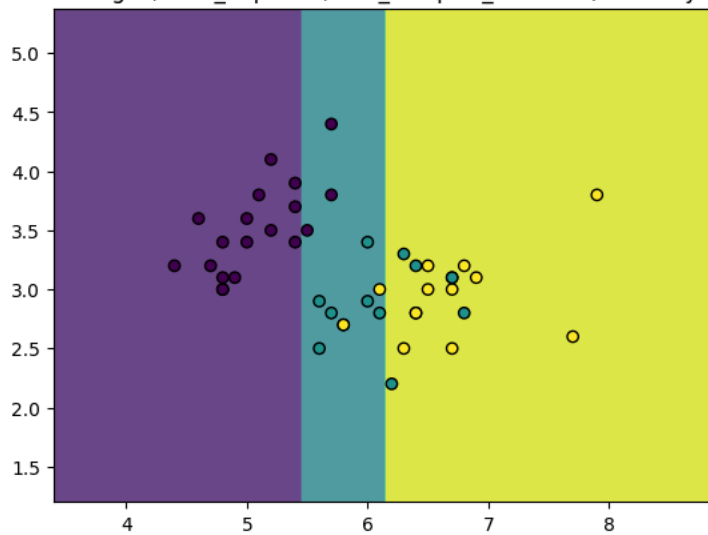


Tree with criterion=gini, max_depth=2, min_samples_leaf=5

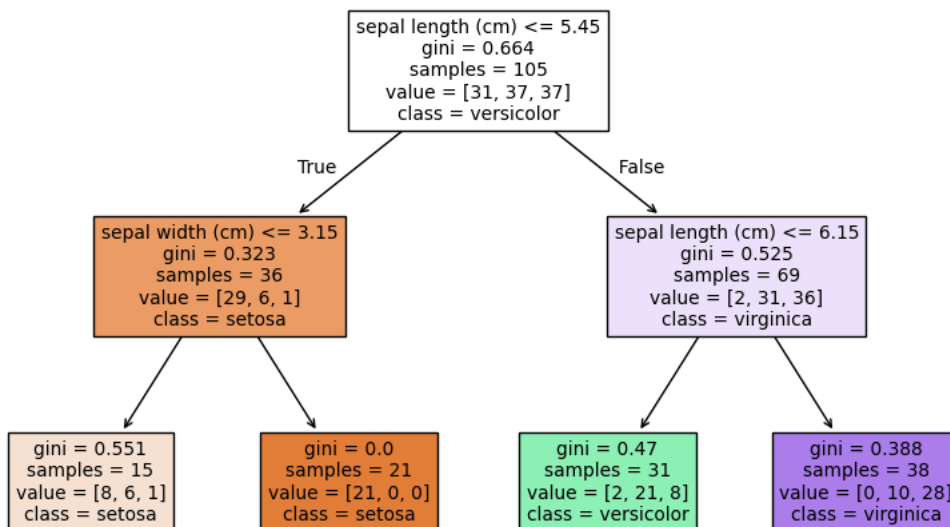
sepal length (cm) <= 5.45



criterion=gini, max_depth=2, min_samples_leaf=15 (Accuracy: 0.76)

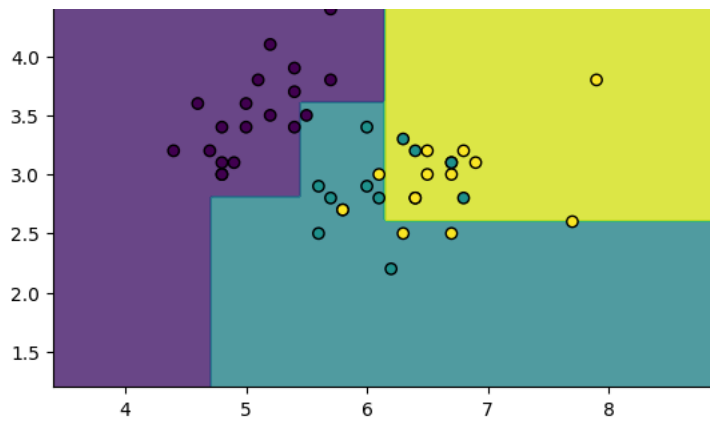


Tree with criterion=gini, max_depth=2, min_samples_leaf=15

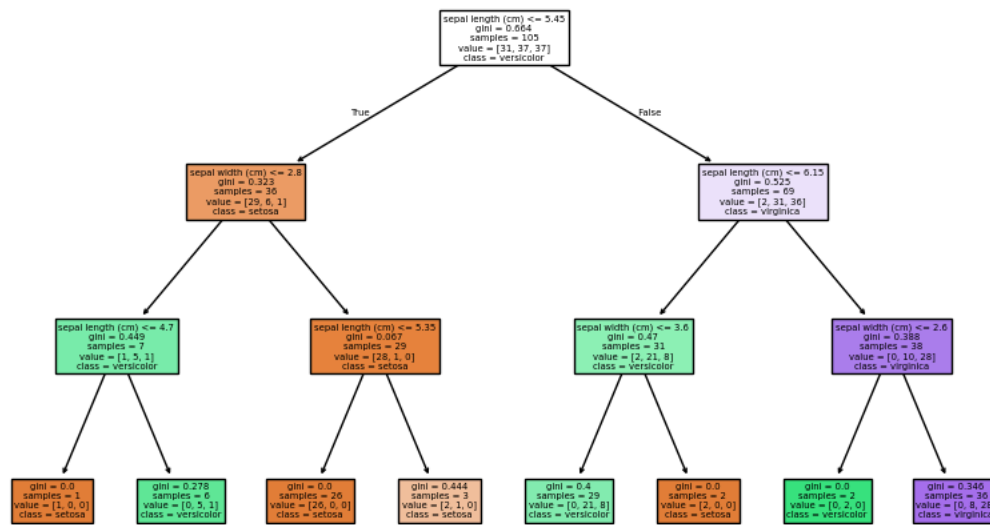


criterion=gini, max_depth=3, min_samples_leaf=1 (Accuracy: 0.76)

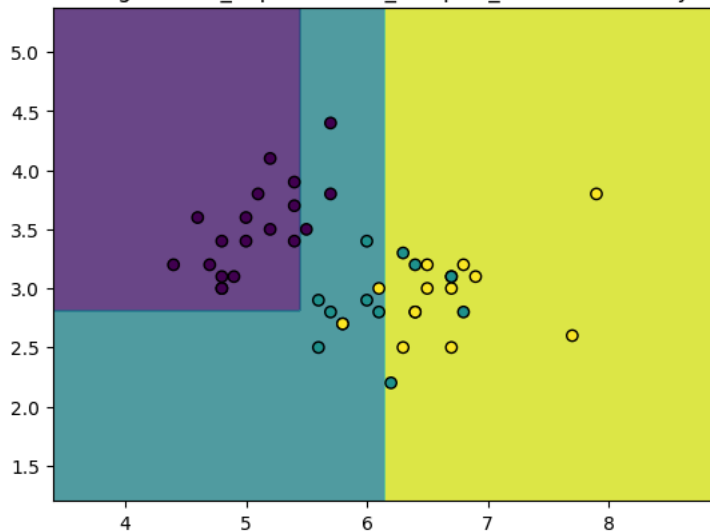




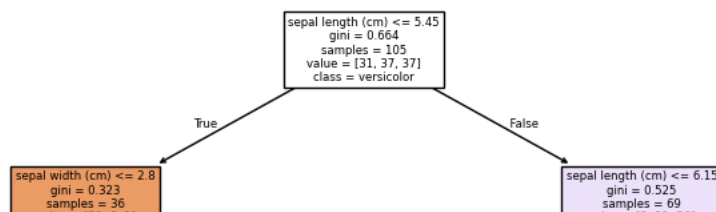
Tree with criterion=gini, max_depth=3, min_samples_leaf=1

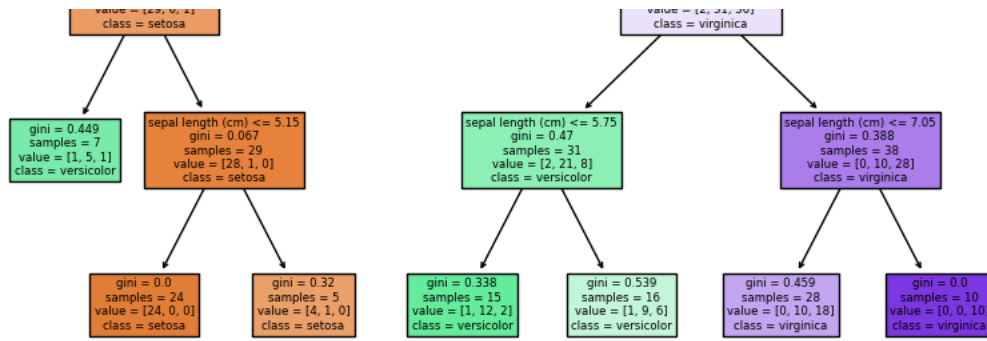


criterion=gini, max_depth=3, min_samples_leaf=5 (Accuracy: 0.76)

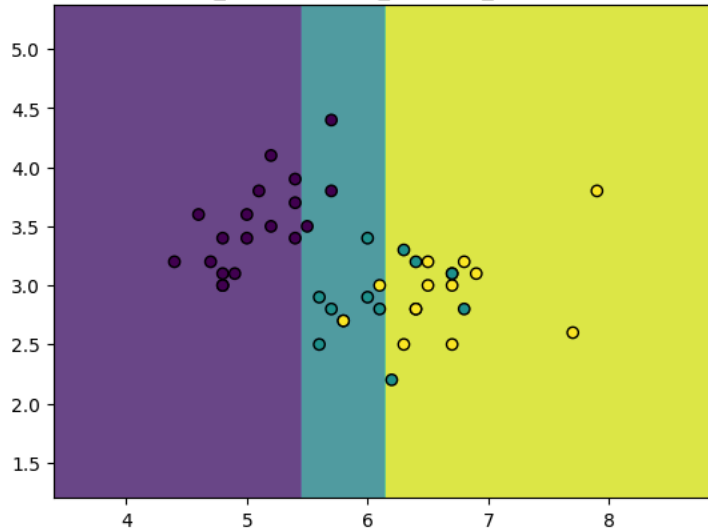


Tree with criterion=gini, max_depth=3, min_samples_leaf=5

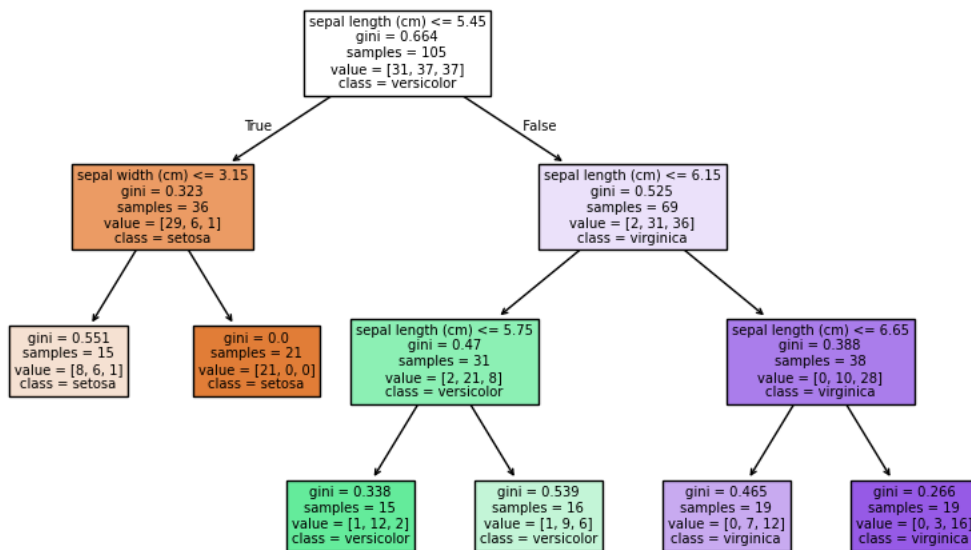




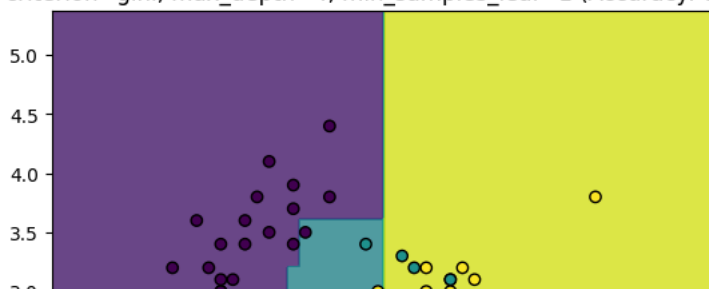
criterion=gini, max_depth=3, min_samples_leaf=15 (Accuracy: 0.76)

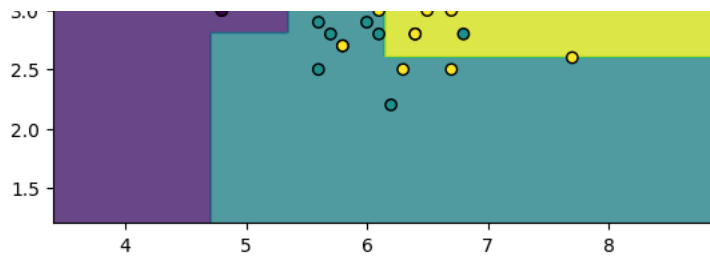


Tree with criterion=gini, max_depth=3, min_samples_leaf=15

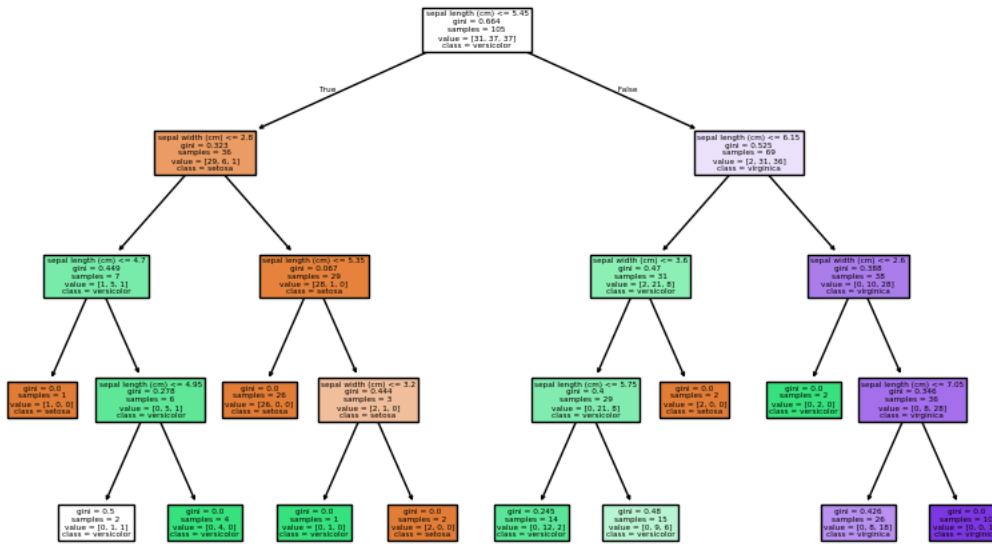


criterion=gini, max_depth=4, min_samples_leaf=1 (Accuracy: 0.76)

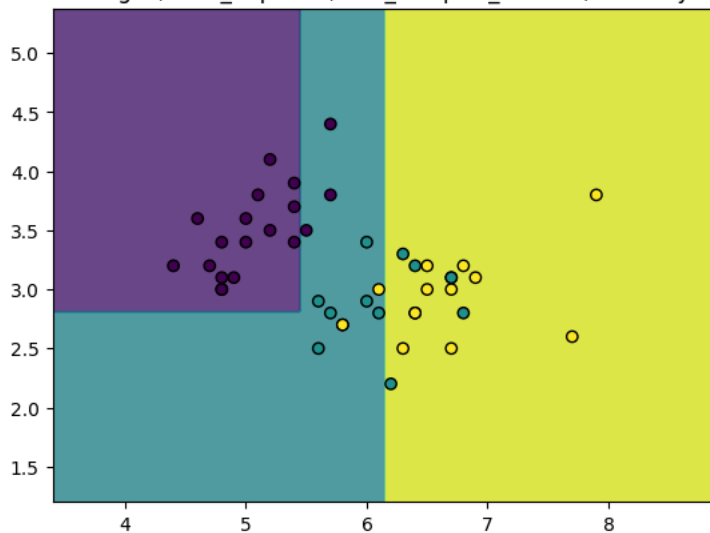




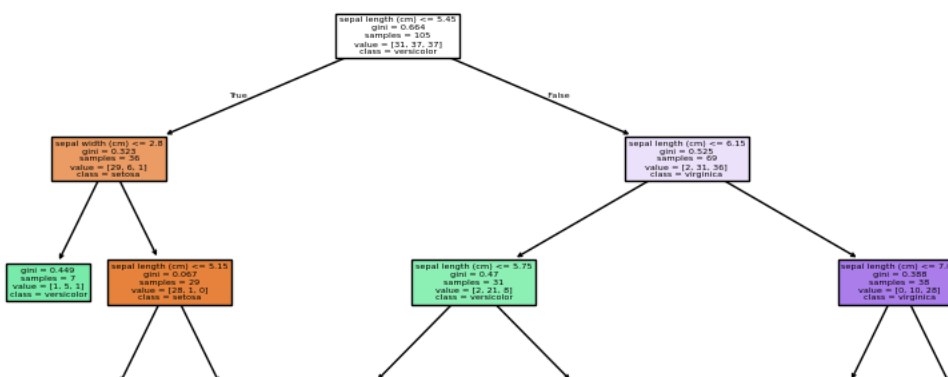
Tree with criterion=gini, max_depth=4, min_samples_leaf=1

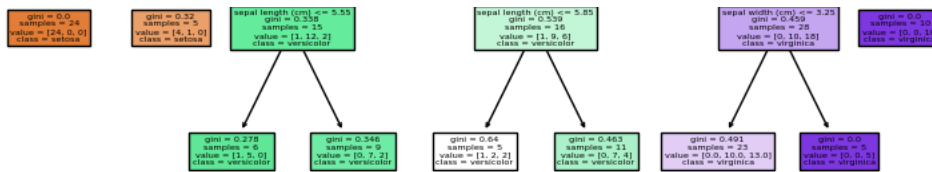


criterion=gini, max_depth=4, min_samples_leaf=5 (Accuracy: 0.76)

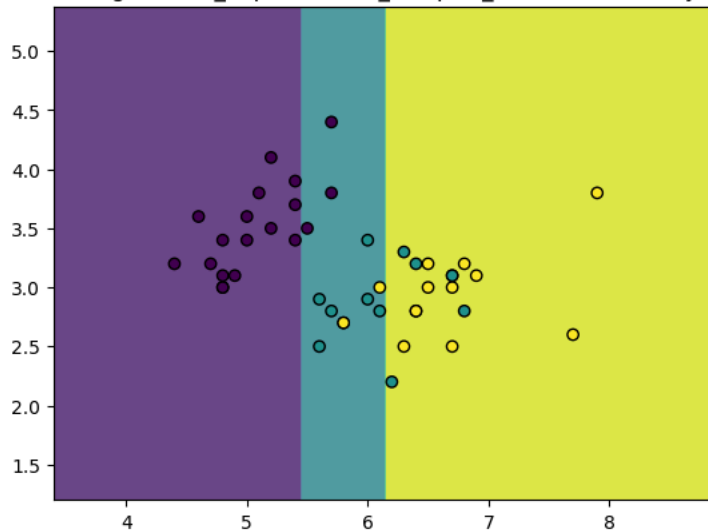


Tree with criterion=gini, max_depth=4, min_samples_leaf=5

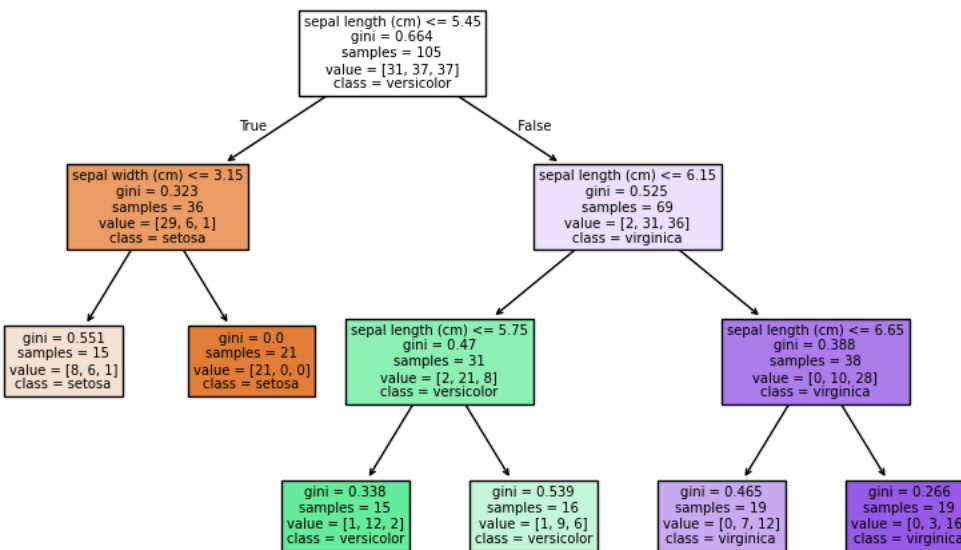




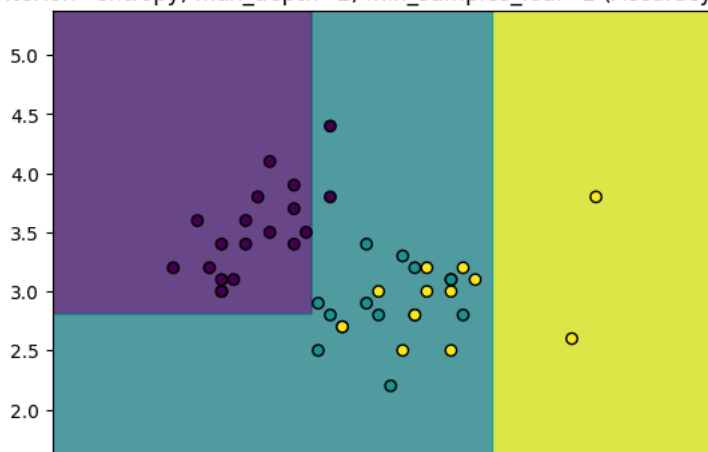
criterion=gini, max_depth=4, min_samples_leaf=15 (Accuracy: 0.76)

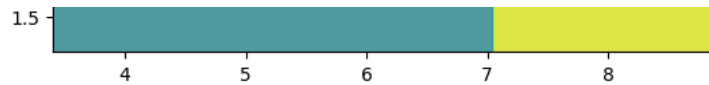


Tree with criterion=gini, max_depth=4, min_samples_leaf=15

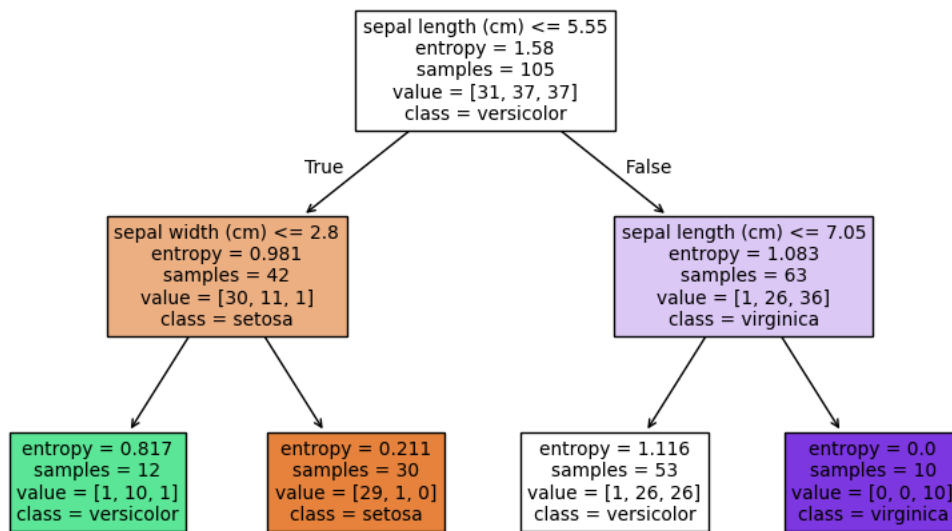


criterion=entropy, max_depth=2, min_samples_leaf=1 (Accuracy: 0.71)

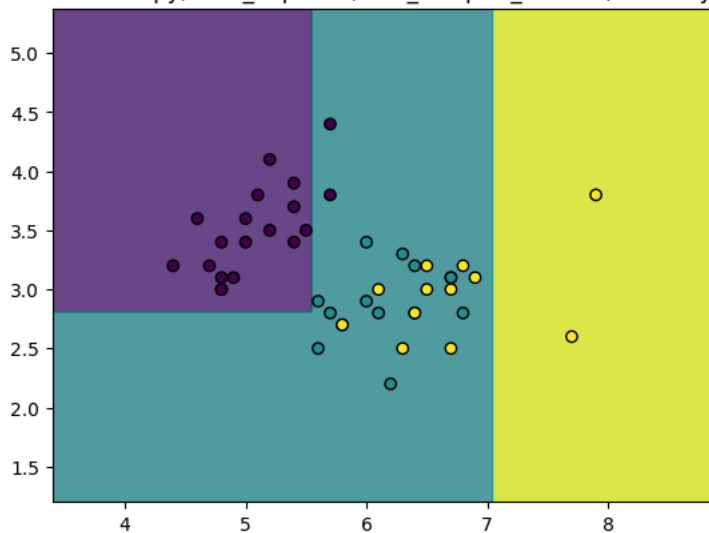




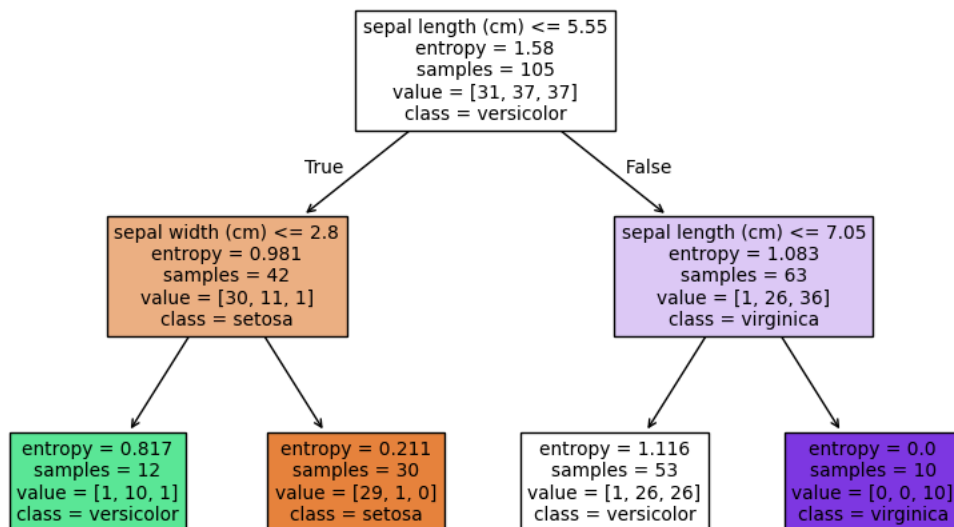
Tree with criterion=entropy, max_depth=2, min_samples_leaf=1



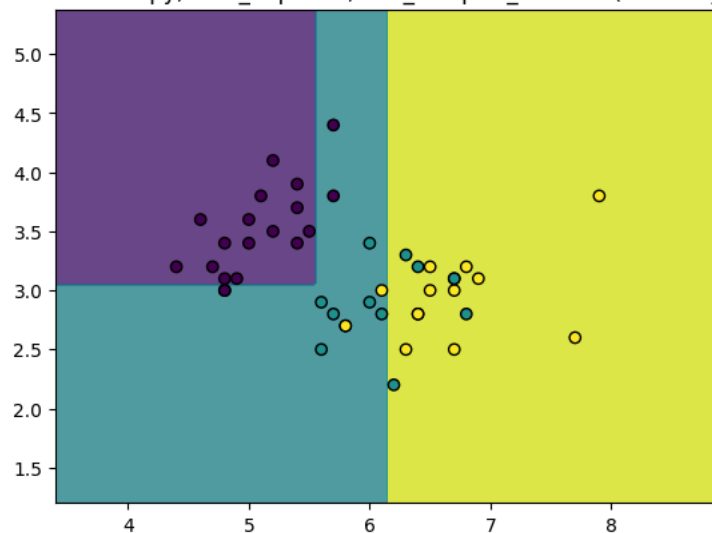
criterion=entropy, max_depth=2, min_samples_leaf=5 (Accuracy: 0.71)



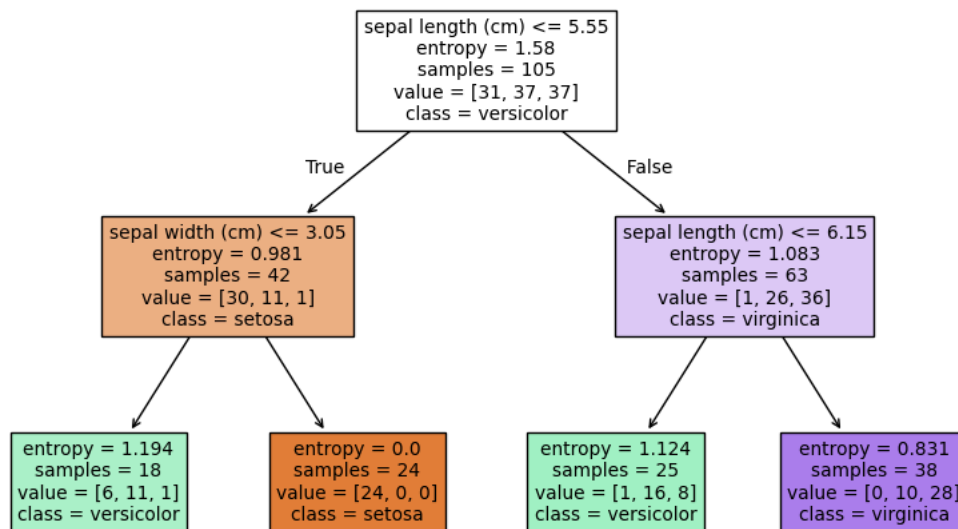
Tree with criterion=entropy, max_depth=2, min_samples_leaf=5



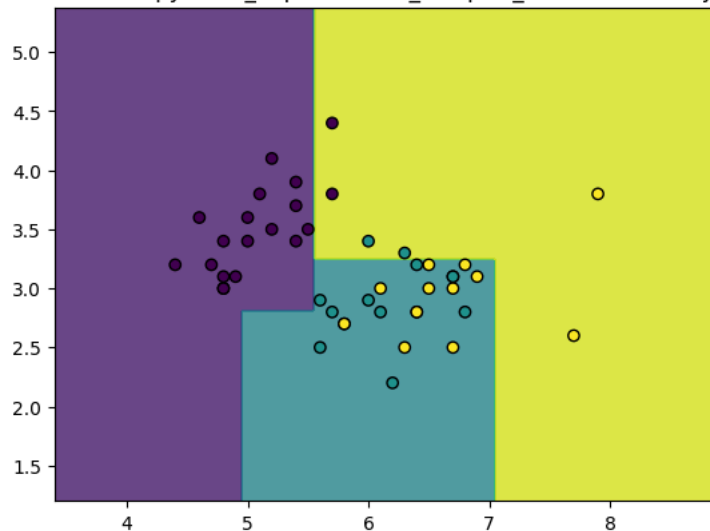
criterion=entropy, max_depth=2, min_samples_leaf=15 (Accuracy: 0.73)



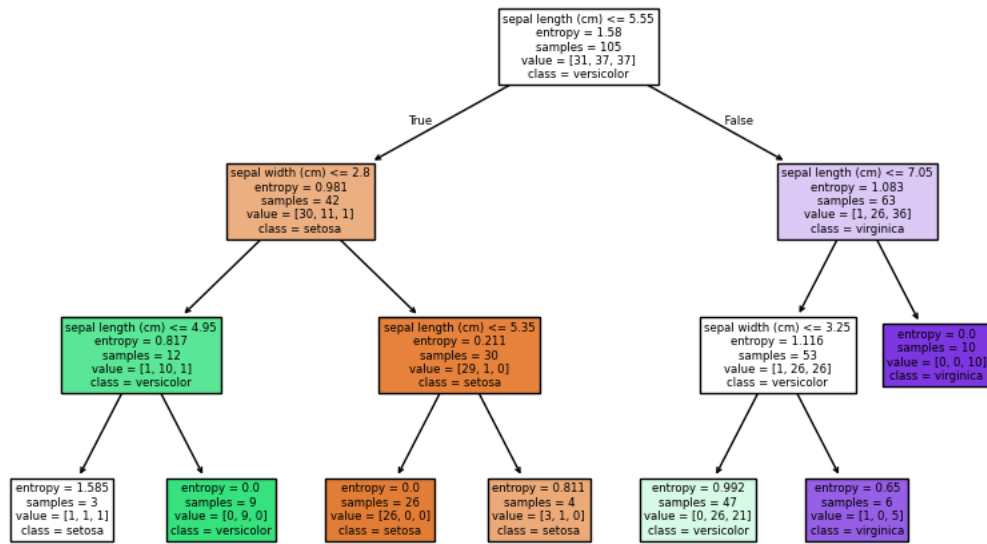
Tree with criterion=entropy, max_depth=2, min_samples_leaf=15



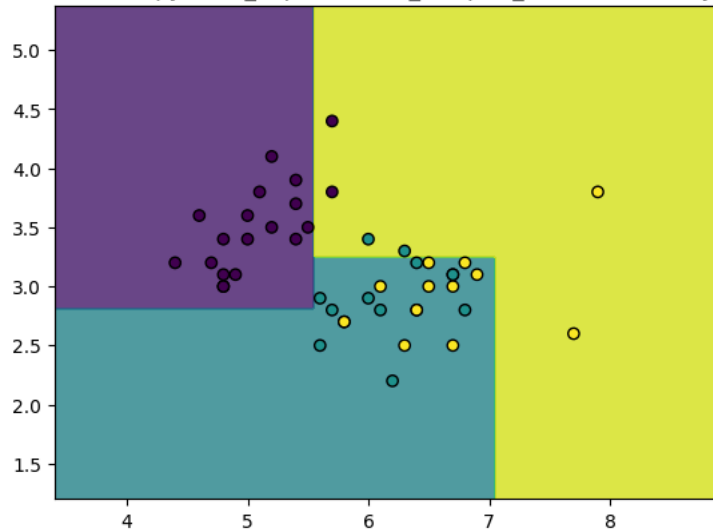
criterion=entropy, max_depth=3, min_samples_leaf=1 (Accuracy: 0.67)



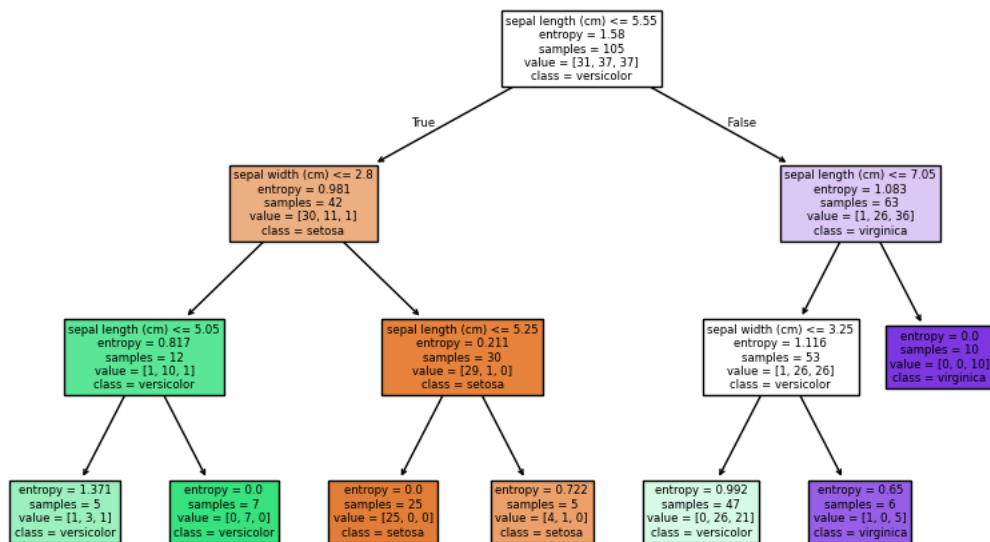
Tree with criterion=entropy, max_depth=3, min_samples_leaf=1



criterion=entropy, max_depth=3, min_samples_leaf=5 (Accuracy: 0.67)

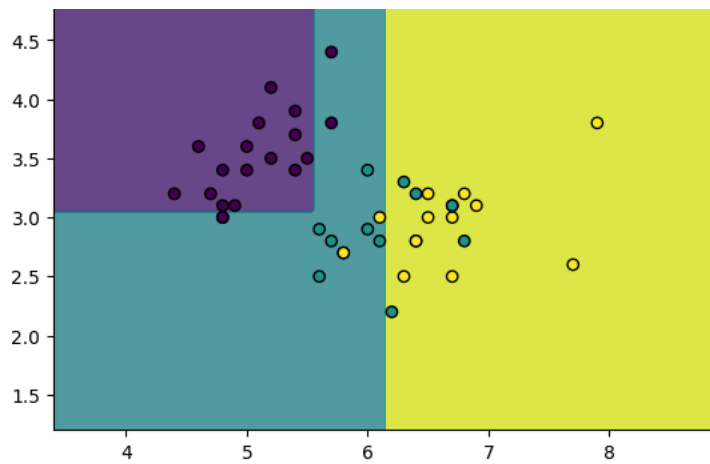


Tree with criterion=entropy, max_depth=3, min_samples_leaf=5

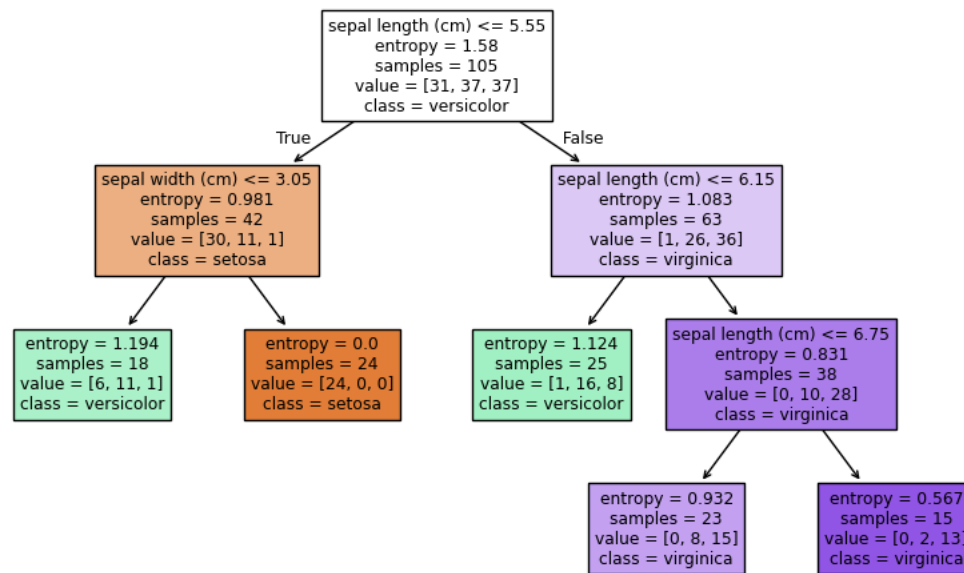


criterion=entropy, max_depth=3, min_samples_leaf=15 (Accuracy: 0.73)

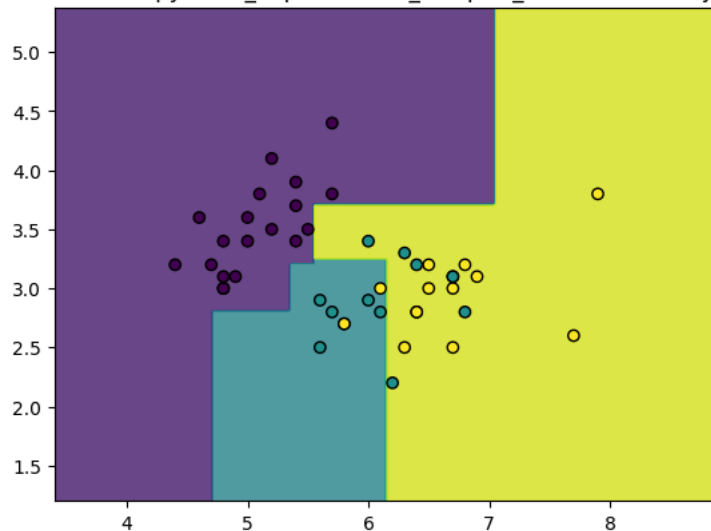




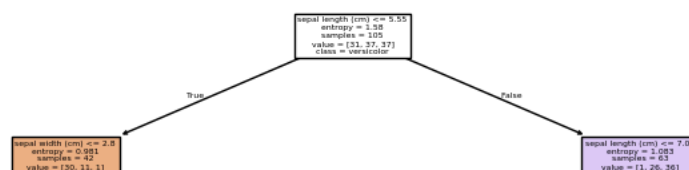
Tree with criterion=entropy, max_depth=3, min_samples_leaf=15

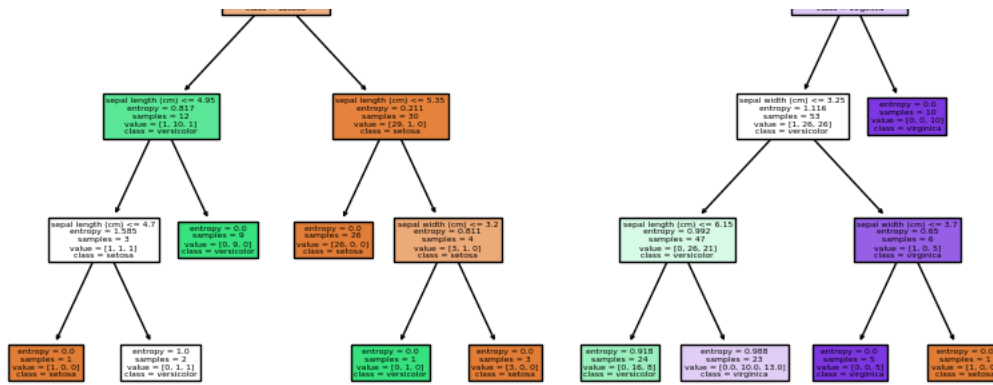


criterion=entropy, max_depth=4, min_samples_leaf=1 (Accuracy: 0.80)

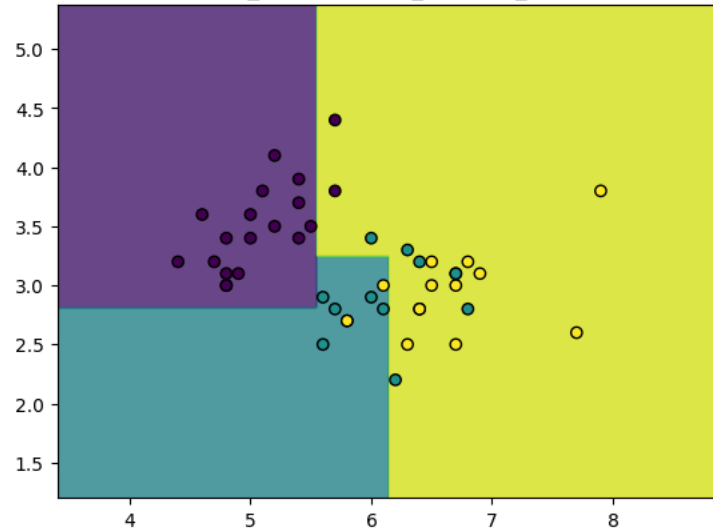


Tree with criterion=entropy, max_depth=4, min_samples_leaf=1

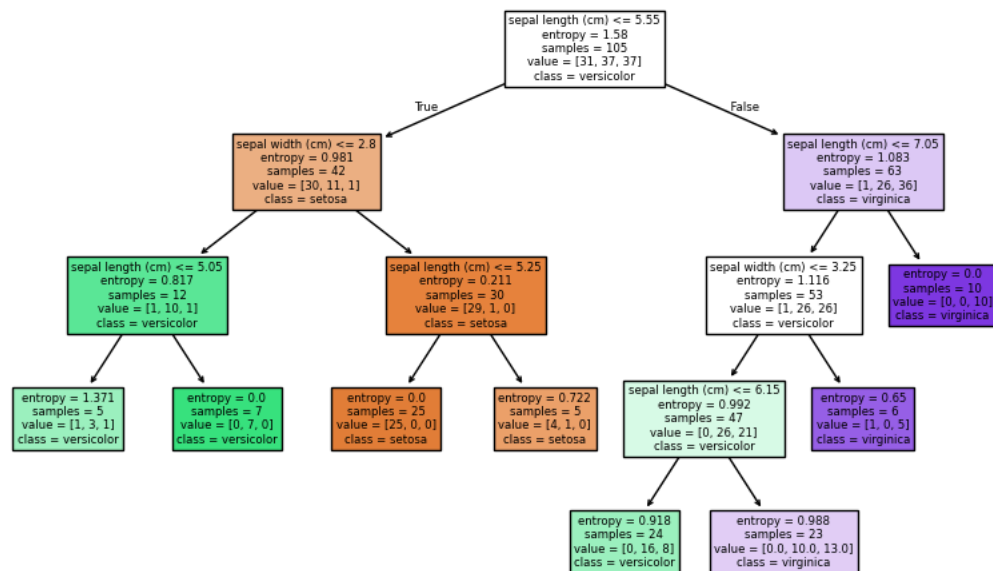




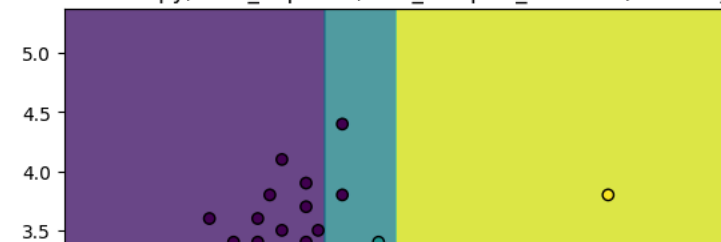
criterion=entropy, max_depth=4, min_samples_leaf=5 (Accuracy: 0.76)

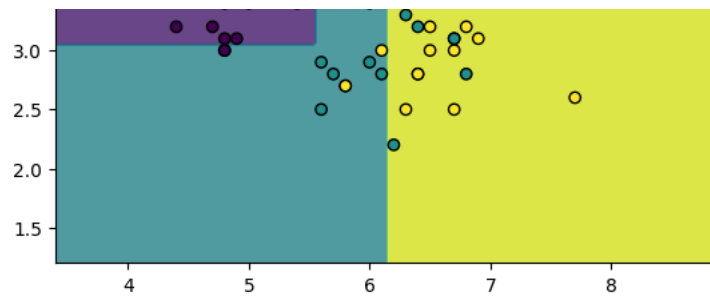


Tree with criterion=entropy, max_depth=4, min_samples_leaf=5

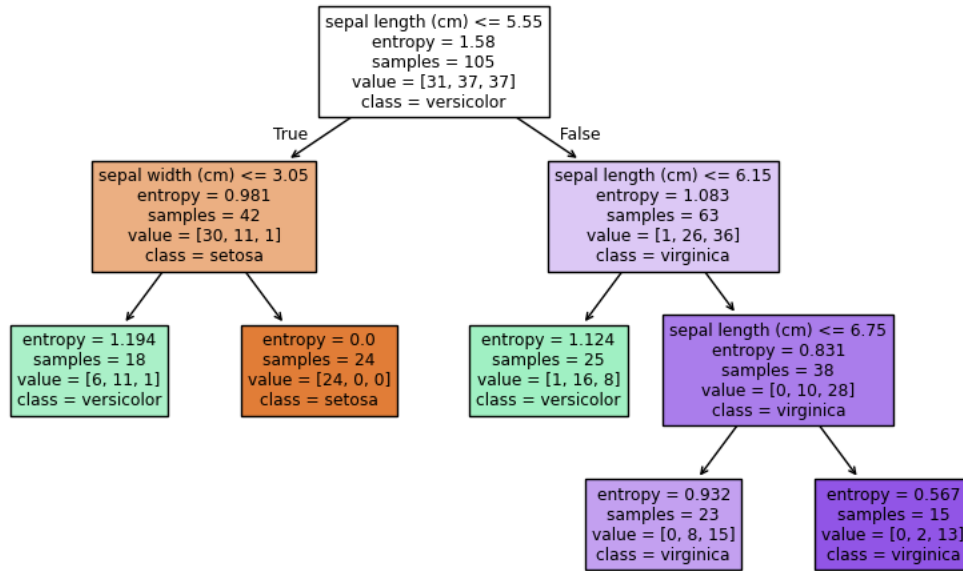


criterion=entropy, max_depth=4, min_samples_leaf=15 (Accuracy: 0.73)





Tree with criterion=entropy, max_depth=4, min_samples_leaf=15



This shows the accuracy. The key is an association and the value is how accurate. That model is. Interestingly, the key is an association/dictionary. You can look and see that the most accurate model was with the criterion set to entropy, the max depth set to 4, and the minimum samples per leaf set to 1. This yielded an accuracy of 0.8. There were no other combinations that were that accurate (so no ties for first place).

```
# Creating a dictionary with tuples of parameters as keys and accuracy as values
results_dict = {}

for result in results:
    key = (result['criterion'], result['max_depth'], result['min_samples_leaf'])
    value = result['accuracy']
    results_dict[key] = value

# Sorting the dictionary by accuracy in descending order
sorted_results_dict = dict(sorted(results_dict.items(), key=lambda item: item[1], reverse=True))

# Printing the sorted dictionary
for key, value in sorted_results_dict.items():
    print(f"criterion={key[0]}, max_depth={key[1]}, min_samples_leaf={key[2]} => {value}")

criterion=entropy, max_depth=4, min_samples_leaf=1 => 0.8
criterion=gini, max_depth=2, min_samples_leaf=1 => 0.7555555555555555
criterion=gini, max_depth=2, min_samples_leaf=5 => 0.7555555555555555
criterion=gini, max_depth=2, min_samples_leaf=15 => 0.7555555555555555
criterion=gini, max_depth=3, min_samples_leaf=1 => 0.7555555555555555
criterion=gini, max_depth=3, min_samples_leaf=5 => 0.7555555555555555
criterion=gini, max_depth=3, min_samples_leaf=15 => 0.7555555555555555
criterion=gini, max_depth=4, min_samples_leaf=1 => 0.7555555555555555
criterion=gini, max_depth=4, min_samples_leaf=5 => 0.7555555555555555
criterion=gini, max_depth=4, min_samples_leaf=15 => 0.7555555555555555
criterion=entropy, max_depth=4, min_samples_leaf=5 => 0.7555555555555555
criterion=entropy, max_depth=2, min_samples_leaf=15 => 0.7333333333333333
criterion=entropy, max_depth=3, min_samples_leaf=15 => 0.7333333333333333
criterion=entropy, max_depth=4, min_samples_leaf=15 => 0.7333333333333333
criterion=entropy, max_depth=2, min_samples_leaf=1 => 0.7111111111111111
criterion=entropy, max_depth=2, min_samples_leaf=5 => 0.7111111111111111
criterion=entropy, max_depth=3, min_samples_leaf=1 => 0.6666666666666666
criterion=entropy, max_depth=3, min_samples_leaf=5 => 0.6666666666666666
```

This plots the decision boundary, tree, and accuracy for the most accurate tree.

```
# First, identify the keys with the highest accuracy from the dictionary
highest_accuracy = max(results_dict.values())

# Select the keys that correspond to the highest accuracy
best_combinations = [key for key, value in results_dict.items() if value == highest_accuracy]

# Iterate over the best combinations and plot decision boundaries, trees, and accuracies for those
for criterion, max_depth, min_samples_leaf in best_combinations:

    # Initialize the Decision Tree Classifier
    clf = DecisionTreeClassifier(criterion=criterion, max_depth=max_depth, min_samples_leaf=min_samples_leaf, random_state=42)

    # Train the classifier
    clf.fit(X_train, y_train)

    # Predict and compute accuracy on test set
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)

    # Plot the decision boundary with accuracy
    plot_decision_boundary_with_accuracy(clf, X_test, y_test, f'criterion={criterion}, max_depth={max_depth}, min_samples_leaf={min_samples_leaf}')

    # Plot the tree
    plt.figure(figsize=(10, 6))
    plot_tree(clf, filled=True, feature_names=iris.feature_names[:2], class_names=iris.target_names)
    plt.title(f'Tree with criterion={criterion}, max_depth={max_depth}, min_samples_leaf={min_samples_leaf}')
    plt.show()
```



criterion=entropy, max_depth=4, min_samples_leaf=1 (Accuracy: 0.80)

5.0

