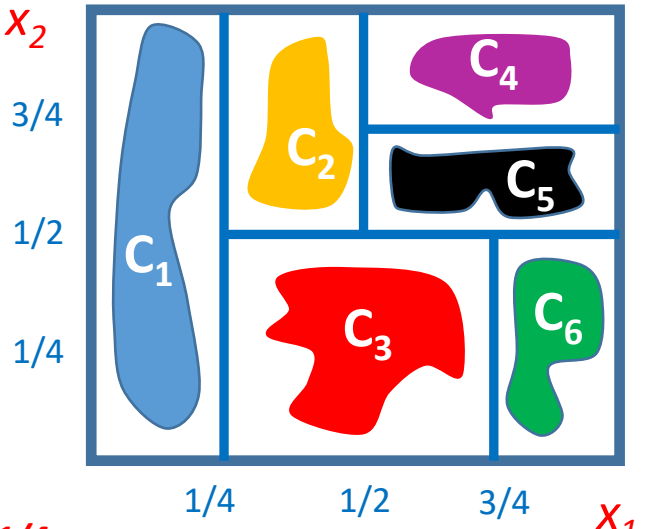
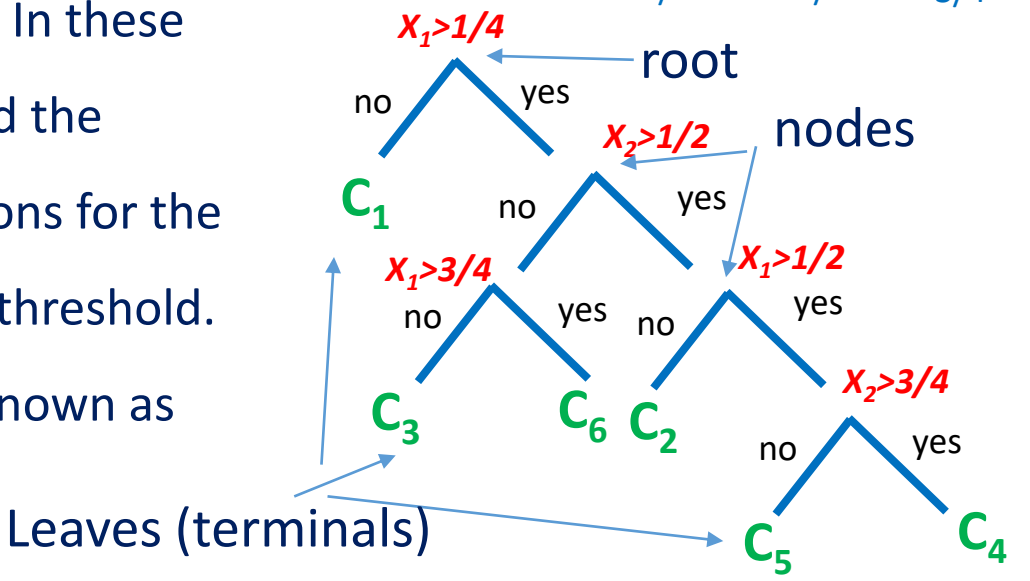
Decision Trees (DT)

Decision Trees (DT)

- Decision trees represent a class of supervised learning, non-linear classifiers/regressors.
- The features space is split into unique regions of specific classes sequentially.
- Most popular decision trees those that can split the feature space into hyperrectangles with sides are parallel to the axes. In these algorithms, a tree is constructed with several nodes and the decision at each node is taken by asking a set of questions for the individual features such as is feature $x_i > t$, where t is a threshold. Since the answer is 'yes' or 'no', these type of trees is known as ordinary binary classification trees (OBCTs).





Node Impurity

- There are several measures that can be used to determine the information impurity such as:
  - Information Impurity (or Entropy Impurity)
  - Gini Impurity
  - Misclassification Impurity

Node Impurity

- The information impurity i(n) at node n is defined as:

$$i(n) = -\sum_j P(C_j)\, log_2 P(C_j)$$

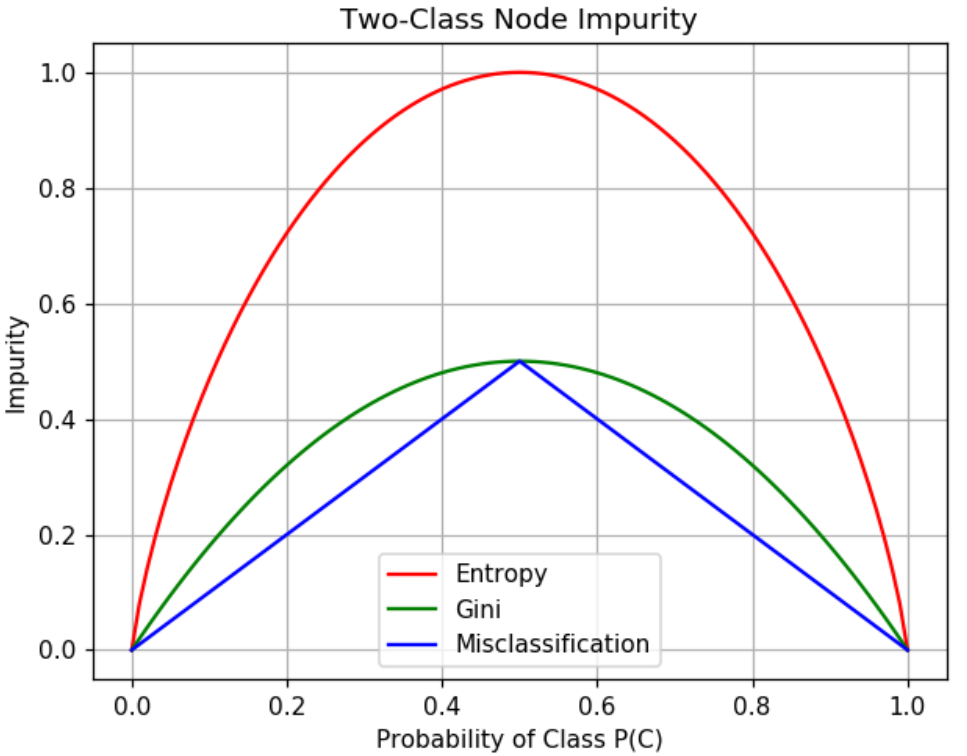
Where $P(C_j)$ is the fraction of data at node n that belongs to class C_j

- Gini impurity i(n) at node n is defined as:

$$i(n) = \sum_j P(C_j)(1-P(C_j))$$

- Misclassification impurity i(n) at node n is defined as:

$$i(n) = 1 - \max_j P(C_j)$$



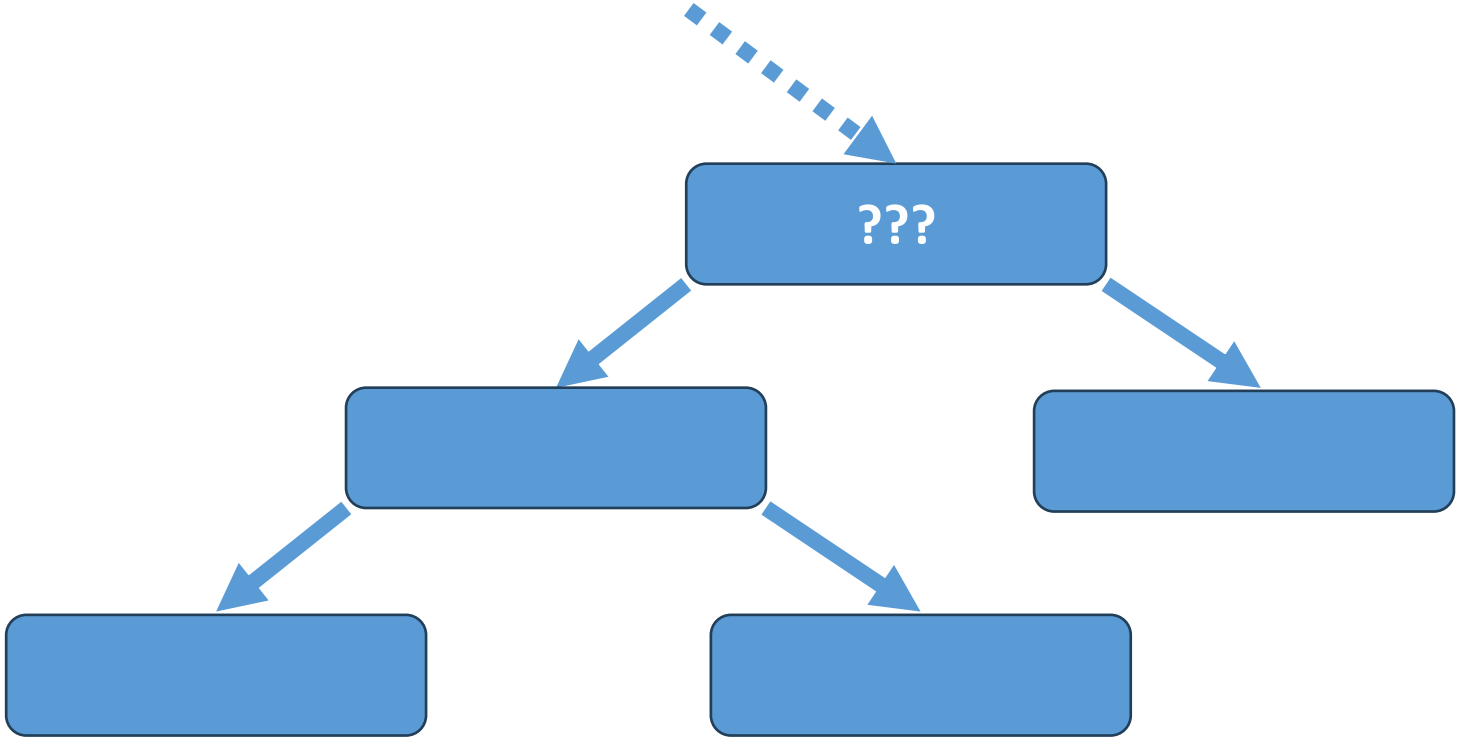
Example: Building a classification Tree Step-by-Step

- Given this training dataset, we want to build a classification tree that uses Loves Popcorn, Loves Soda, and Age to predict whether or not someone will love Music 1.

Loves Popcorn	Loves Soda	Age	Music 1
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes
Yes	No	50	No
No	No	83	No

Example

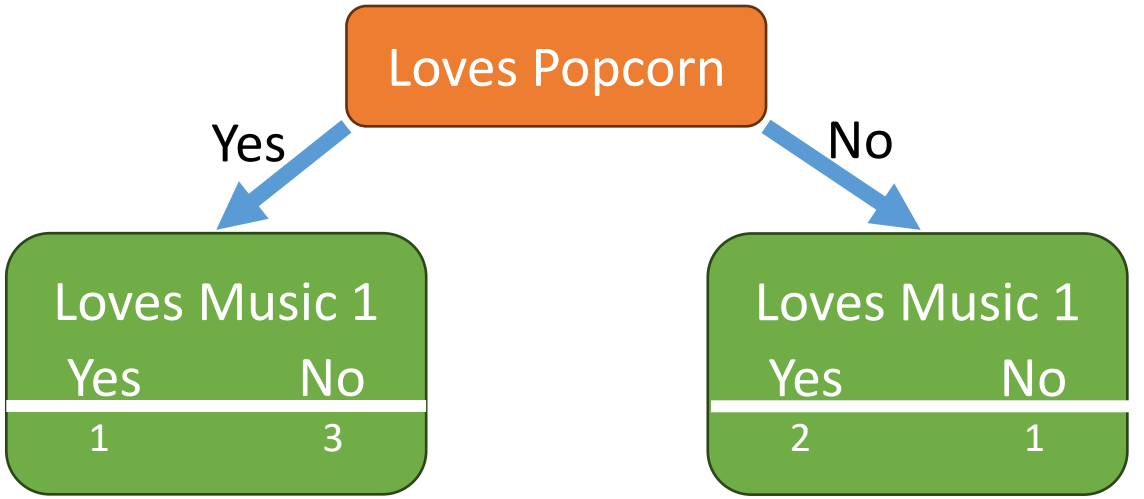
- The first thing we do is to decide whether Loves Popcorn, Loves Soda, or Age should be the question we ask at the very top of the tree.



Example

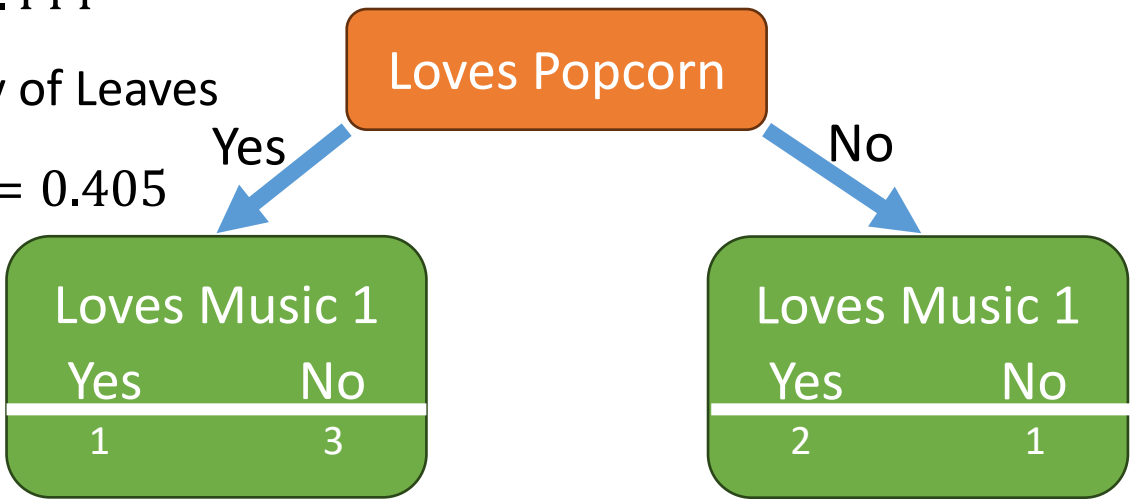
- To make that decision, we will start by looking at how well Loves Popcorn predicts whether or not someone loves Music 1

Loves Popcorn	Loves Soda	Age	Music 1
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes
Yes	No	50	No
No	No	83	No



Example

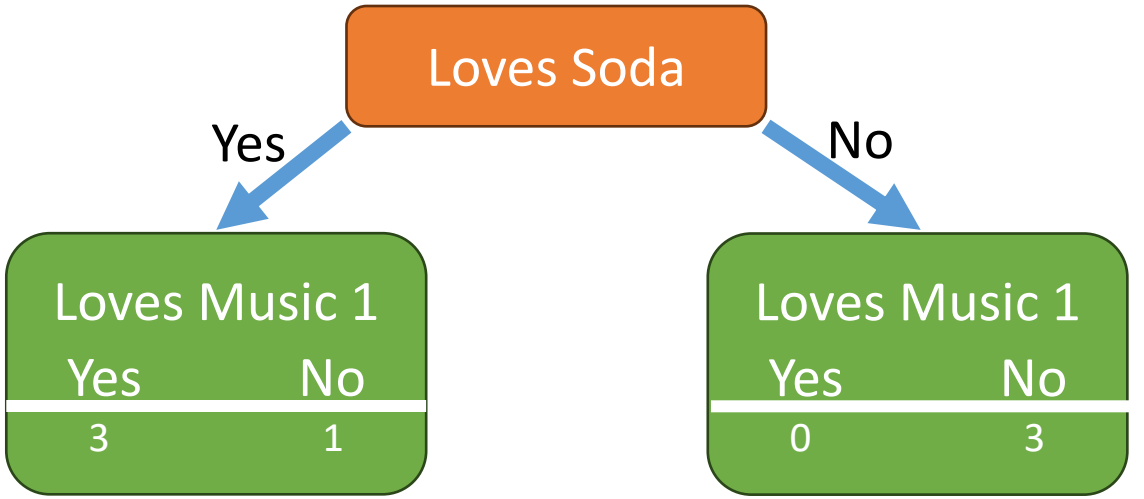
- Calculate the Gini Impurity for Loves Popcorn, first we calculate the Gini Impurity for each individual Leaf.
- Gini Impurity of a Leaf =1 – $(the\ probability\ of\ "yes"\ )^2 - (the\ probability\ of\ "\text{no}")^2$
- Gini Impurity of Left Leaf = $1 - \left(\frac{1}{1+3}\right)^2 - \left(\frac{3}{1+3}\right)^2 = 0.375$
- Gini Impurity of Right Leaf = $1 - \left(\frac{2}{2+1}\right)^2 - \left(\frac{1}{2+1}\right)^2 = 0.444$
- Total Gini Impurity = weighted average of Gini Impurity of Leaves
- Total Gini Impurity $= \left(\frac{4}{4+3}\right) * 0.375 + \left(\frac{3}{4+3}\right) * 0.444 = 0.405$



Example

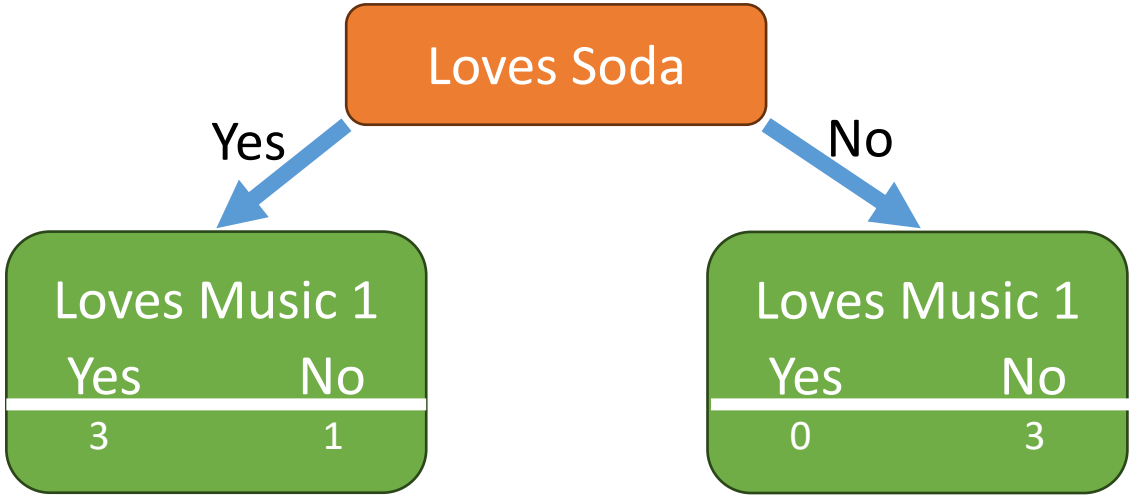
- Then, we do the same with Loves Soda

Loves Popcorn	Loves Soda	Age	Music 1
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes
Yes	No	50	No
No	No	83	No



Example

- In a similar way, we can calculate the Total Gini Impurity for Loves Soda.
- Total Gini Impurity = 0.214



Example

- Now, we need to calculate the Gini Impurity for the Age.
- First, Sort the rows by the value of Age. Then, Calculate the average Age for each adjacent rows.

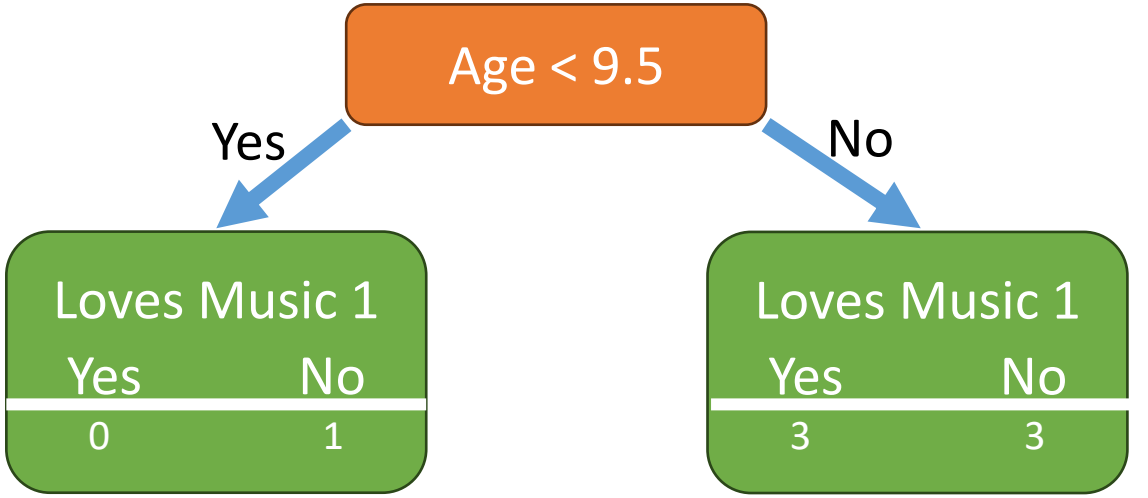
Loves Popcorn	Loves Soda	Age	Music 1
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes
Yes	No	50	No
No	No	83	No

	Age	Music 1
	7	No
9.5		
	12	No
15		
	18	Yes
26.5		
	35	Yes
36.5		
	38	Yes
44		
	50	No
66.5		
	83	No

Example

- Calculate the Gini Impurity for each average value.
- The first average age is 9.5, so we use 9.5 as the threshold for splitting the rows into 2 leaves.
- Total Gini Impurity = 0.429

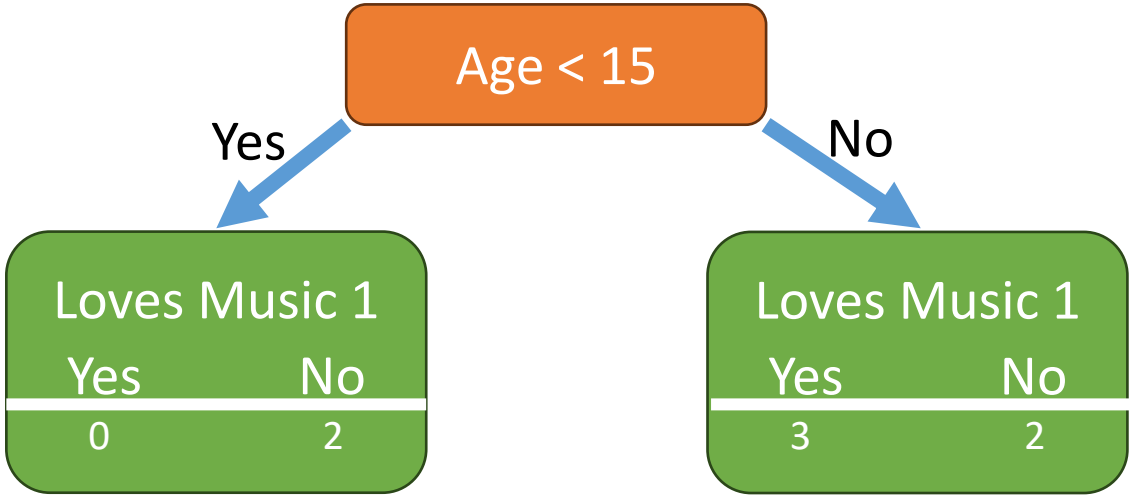
	Age	Music 1
	7	No
9.5	12	No
15	18	Yes
26.5	35	Yes
36.5	38	Yes
44	50	No
66.5	83	No



Example

- Ultimately, we end up with a Gini Impurity for each potential threshold for Age.
- Identify the threshold with lowest Impurity and pick it as the root.

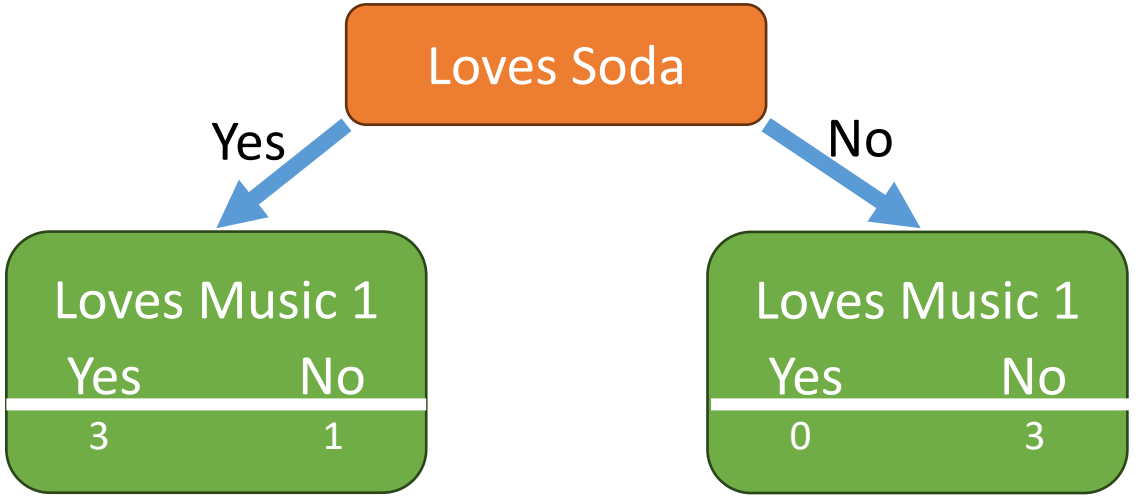
	Age	Music 1	
9.5	7	No	GI = 0.429
15	12	No	GI = 0.343
26.5	18	Yes	GI = 0.476
36.5	35	Yes	GI = 0.476
44	38	Yes	GI = 0.343
66.5	50	No	GI = 0.429
	83	No	



Example

- Gini Impurity for Loves Popcorn = 0.405
- Gini Impurity for Loves Soda = 0.214
- Gini Impurity for Age < 15 = 0.343
- Choose the lowest Gini Impurity as the root node -> Loves Soda is the root.

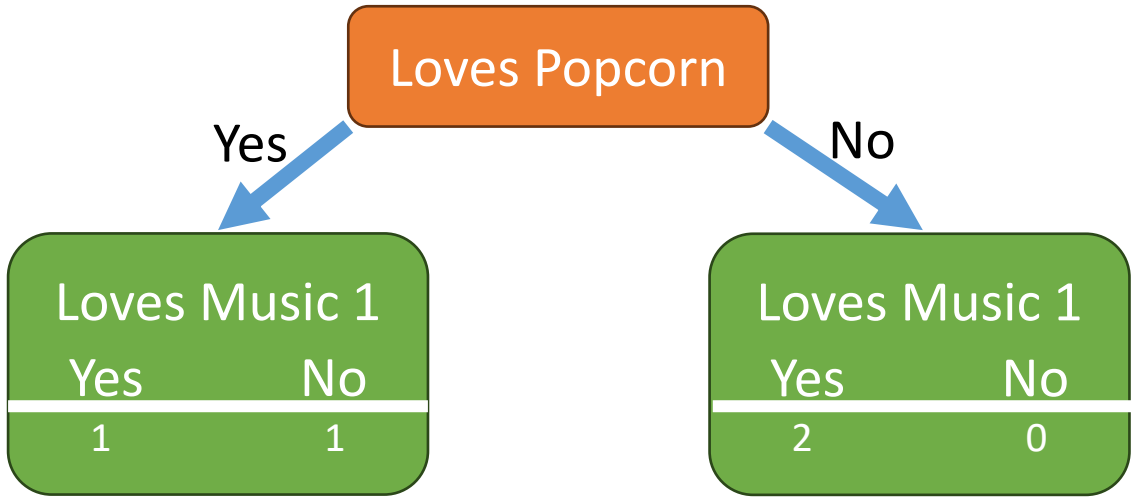
- The node on the right is pure. It becomes a leaf.
- The node on the left is Impure, we can split the 4 people in it based on Loves Popcorn or Age and calculate Gini Impurity.



Example

- When we split the 4 people who Love Soda based on whether or not they Love Popcorn, the Gini Impurity = 0.25.

Loves Popcorn	Loves Soda	Age	Music 1
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes
Yes	No	50	No
No	No	83	No



Example

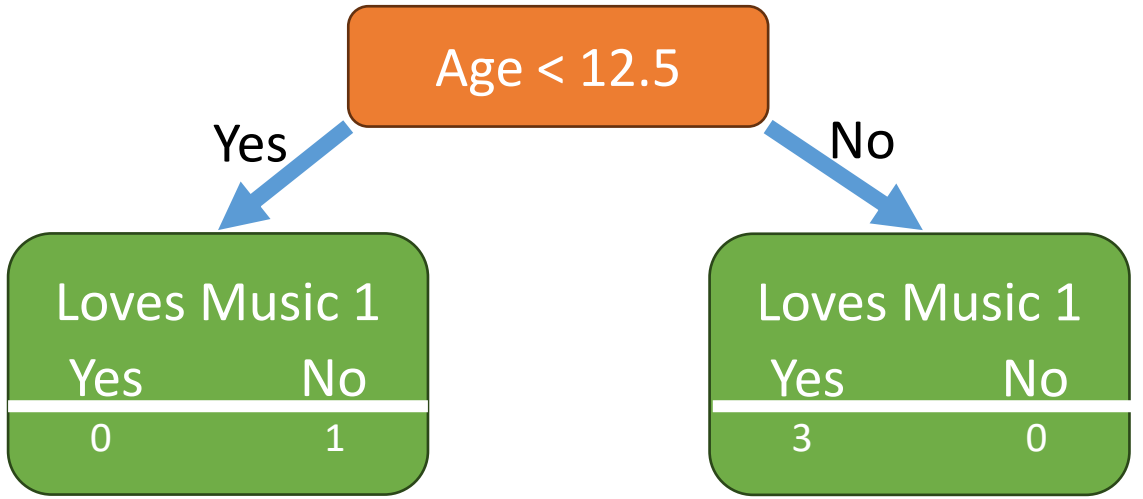
- However, when we split the 4 people based on Age < 12.5, the Gini Impurity = 0.

Loves Popcorn	Loves Soda	Age	Music 1
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes
Yes	No	50	No
No	No	83	No

12.5

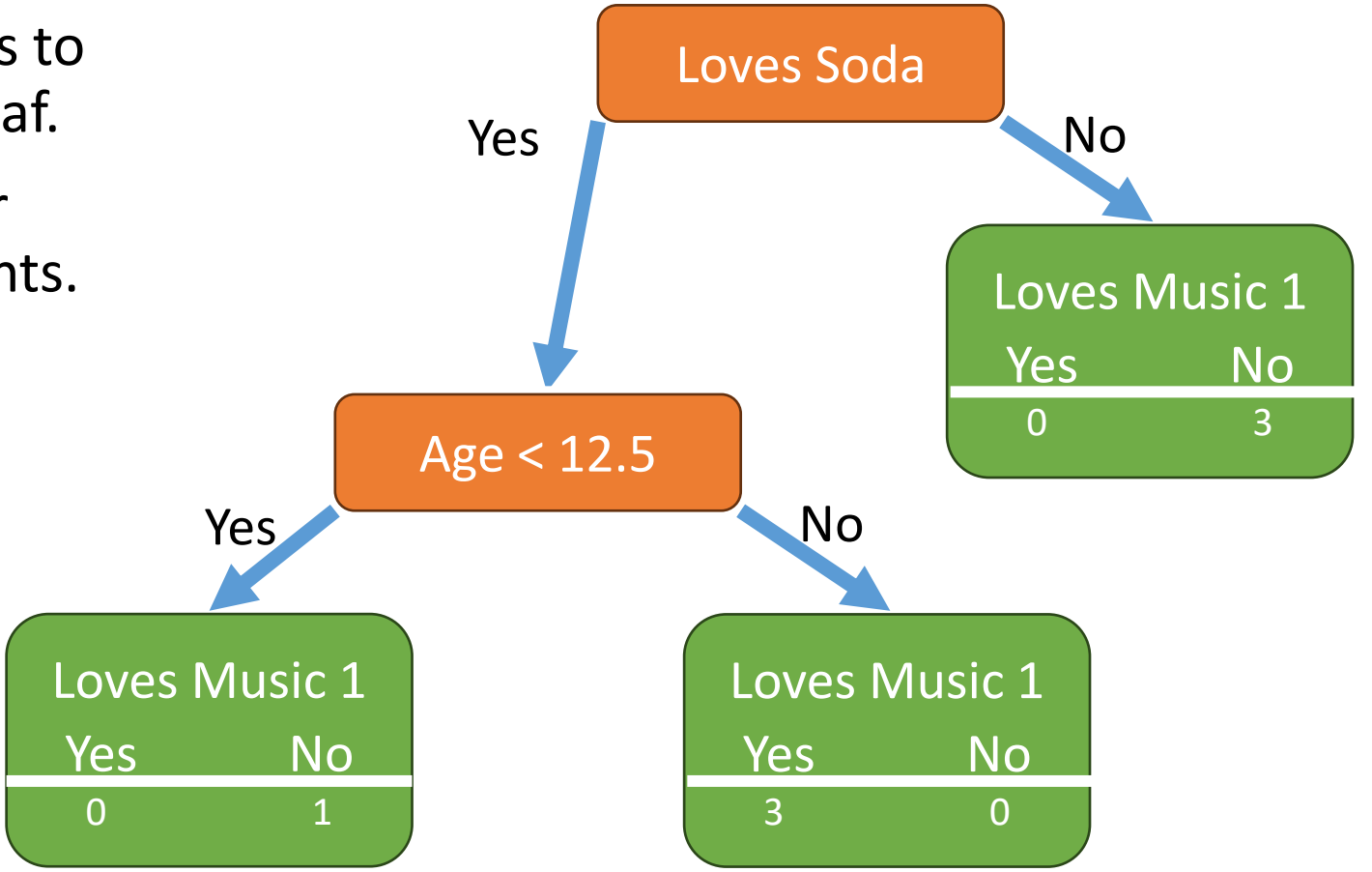
26.5

36.5

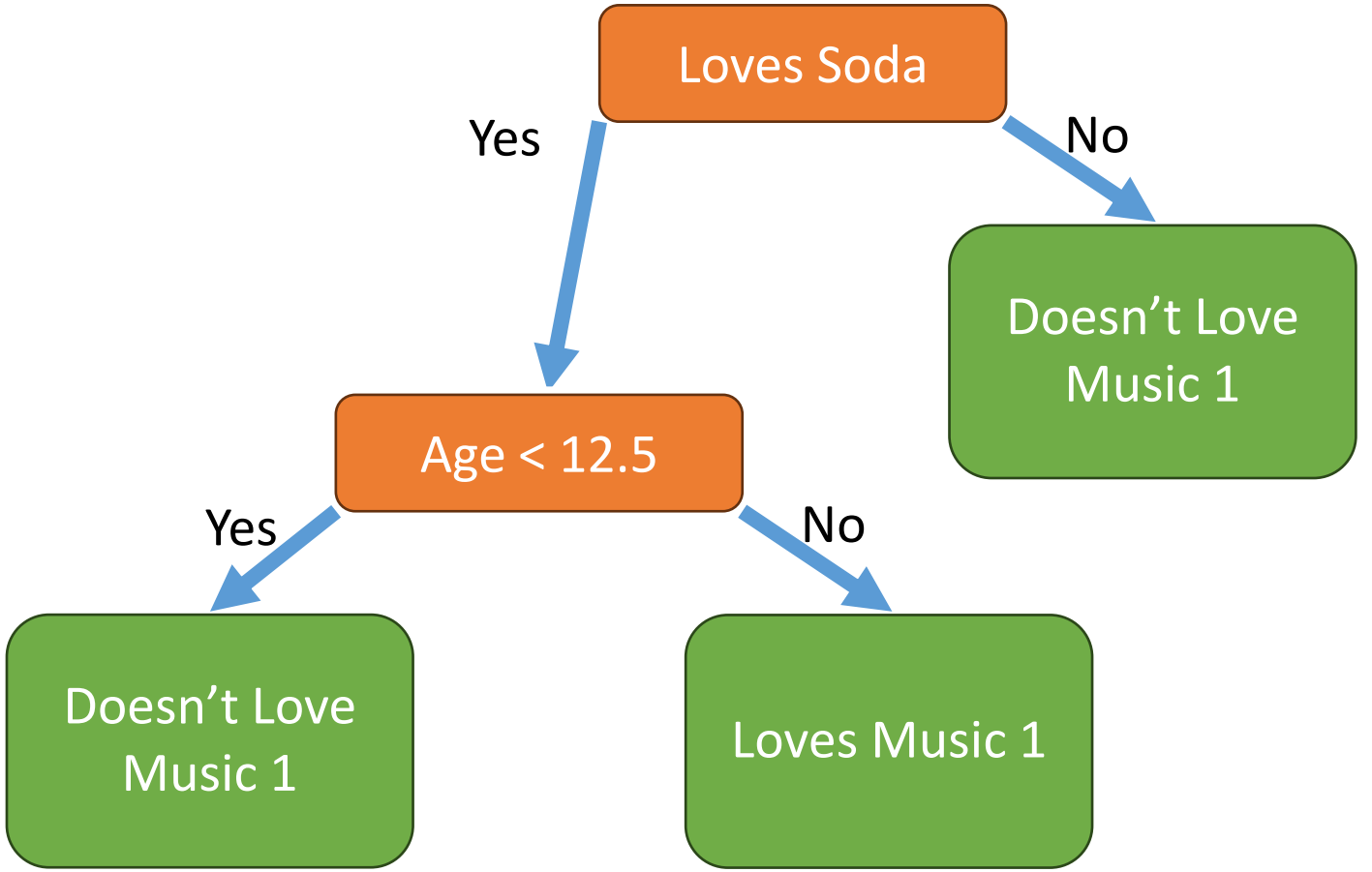


Example

- Now, the only thing remaining is to assign output values for each leaf.
- The output of a leaf is whatever category that has the most counts.



Example: Final Tree



Decision Tree in sklearn

class sklearn.tree.**DecisionTreeClassifier**(**, criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort='deprecated', ccp_alpha=0.0*)

- criterion: either ‘gini’ or ‘entropy’ for impurity measure
- min_samples_split: do splitting if samples are >= min_samples_split. Also, used to determine the depth of the tree if max_depth is none.
- min_samples_leaf: split will not be considered if it results in a number of samples less than min_samples_leaf (if no split then the node is a leaf).
- max_features: if none, all feature will be used when consider splitting, otherwise use ‘sqrt’ as square root of the number of features, or ‘log2’ or ‘auto’ = ‘sqrt’.
- max_leaf_nodes: if specified, tree cannot grow to more than this maximum value, otherwise no limit on the number of leaf nodes.
- min_impurity_decrease: to consider splitting, impurity should decrease by at least min_impurity_decrease.
- min_impurity_split: minimum value for impurity to consider splitting, otherwise the node is considered a leaf.
- ccp_alpha: a regularization factor, the larger the value the more we prune nodes based on minimal cost-complexity approach for pruning.

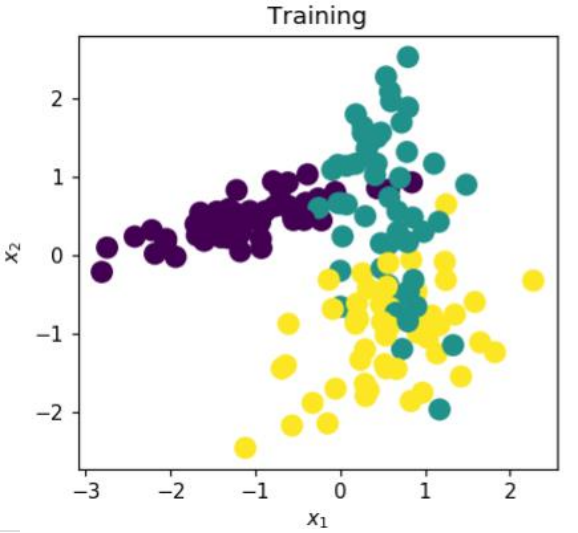
Decision Tree in sklearn, Example 1

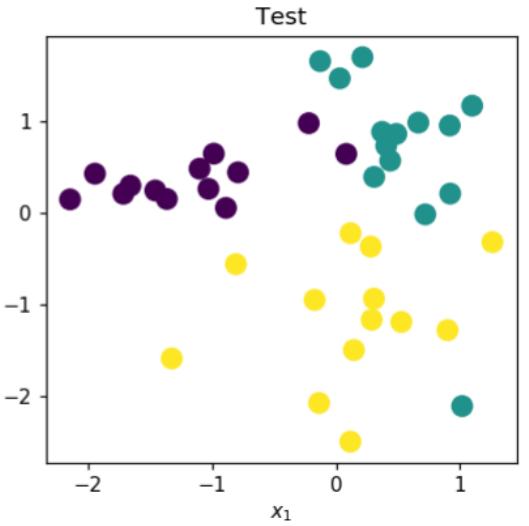
```
%matplotlib notebook
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.datasets import make_classification
X, y = make_classification(n_samples = 200, n_features=2,
                           n_redundant=0, n_informative=2,
                           n_clusters_per_class=1,
                           n_classes=3, random_state = 0)
```

```
X_train, X_test, y_train,  y_test = train_test_split(X, y,
                                                     test_size = 0.2,
                                                     random_state = 0)
scaler_1 = StandardScaler().fit(X_train)
X_train = scaler_1.transform(X_train)
X_test = scaler_1.transform(X_test)
```

```
plt.figure(1)
plt.subplot(1,2,1)
plt.scatter(X_train[:, 0], X_train[:, 1], marker='o', c=y_train,s=100)
plt.xlabel("$x_1$")
plt.ylabel("$x_2$")
plt.title('Training')
plt.subplot(1,2,2)
plt.scatter(X_test[:, 0], X_test[:, 1], marker='o', c=y_test,s=100)
plt.xlabel("$x_1$")
plt.ylabel("$x_2$")
plt.title('Test')
plt.show()
```





Decision Tree in sklearn, Example 1

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

C1 = DecisionTreeClassifier(random_state=0,ccp_alpha=0.0) #You can change ccp_alpha
```

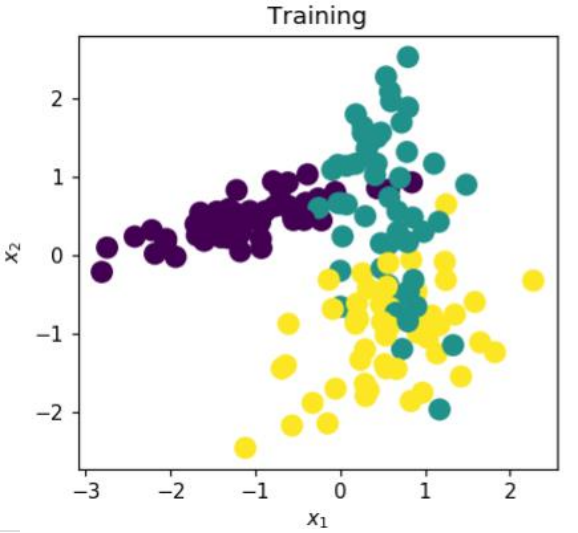
```
C1.fit(X_train,y_train)
```

```
DecisionTreeClassifier(random_state=0)
```

```
yp1 = C1.predict(X_test)
```

```
print(accuracy_score(y_test,yp1))
```

```
0.8
```




```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

C1 = DecisionTreeClassifier(random_state=0,ccp_alpha=0.05) #You can change ccp_alpha
```

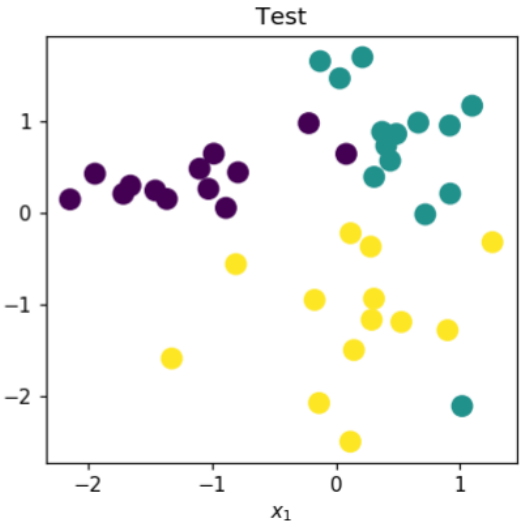
```
C1.fit(X_train,y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.05, random_state=0)
```

```
yp1 = C1.predict(X_test)
```

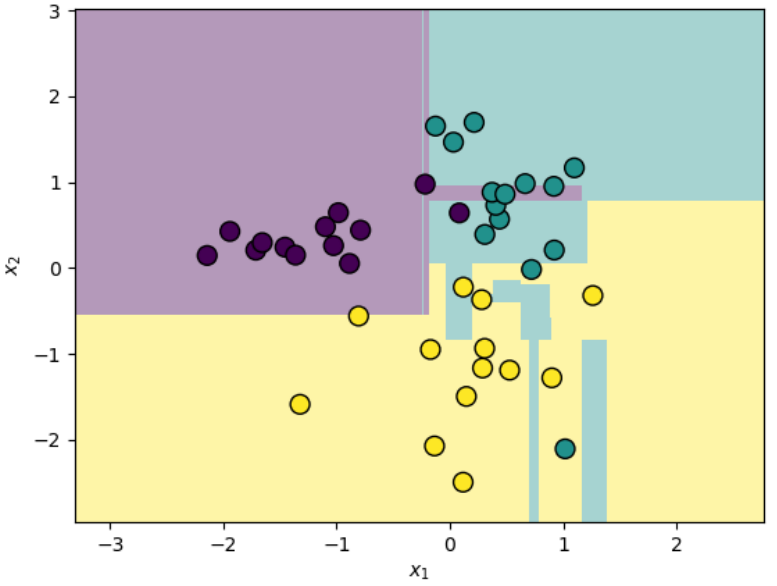
```
print(accuracy_score(y_test,yp1))
```

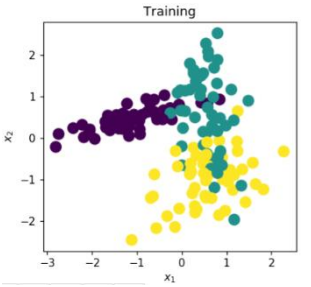
```
0.925
```



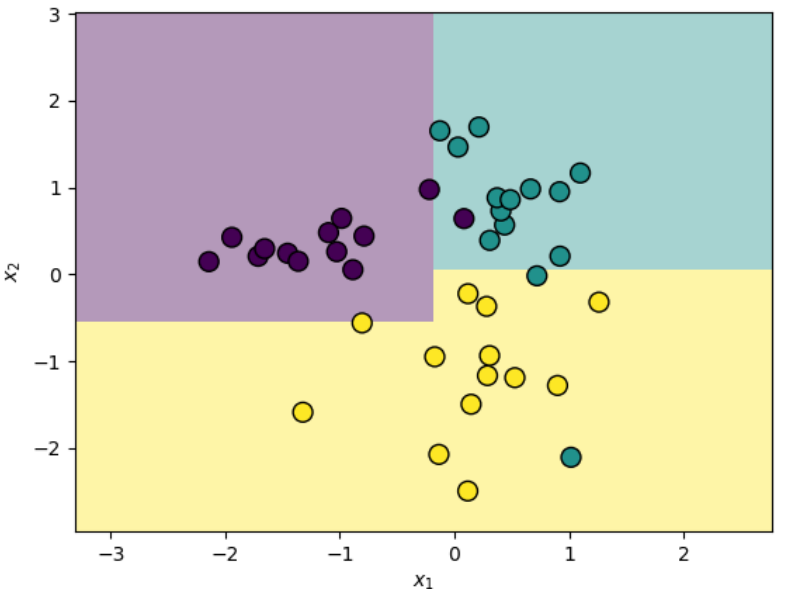
Decision Tree in sklearn, Example 1

ccp_alpha=0.0





ccp_alpha=0.05



Decision Tree in sklearn, Example 1

```
from sklearn.tree import export_text
text_tree = export_text(C1)
print(text_tree)
```

```
|--- feature_0 <= -0.18
|   |--- feature_1 <= -0.55
|   |   |--- class: 2
|   |--- feature_1 >  -0.55
|   |   |--- feature_0 <= -0.26
|   |   |   |--- class: 0
|   |   |--- feature_0 >  -0.26
|   |   |   |--- feature_0 <= -0.24
|   |   |   |   |--- class: 1
|   |   |   |--- feature_0 >  -0.24
|   |   |   |   |--- class: 0
|--- feature_0 >  -0.18
|   |--- feature_1 <= 0.04
|   |   |--- feature_1 <= -0.85
|   |   |   |--- feature_0 <= 1.17
|   |   |   |   |--- feature_0 <= 0.70
|   |   |   |   |   |--- class: 2
|   |   |   |   |--- feature_0 >  0.70
|   |   |   |   |   |--- feature_0 <= 0.78
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- feature_0 >  0.78
|   |   |   |   |   |   |--- class: 2
|   |   |   |--- feature_0 >  1.17
|   |   |   |   |--- feature_0 <= 1.38
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- feature_0 >  1.38
|   |   |   |   |   |--- class: 2
```

ccp_alpha=0.0

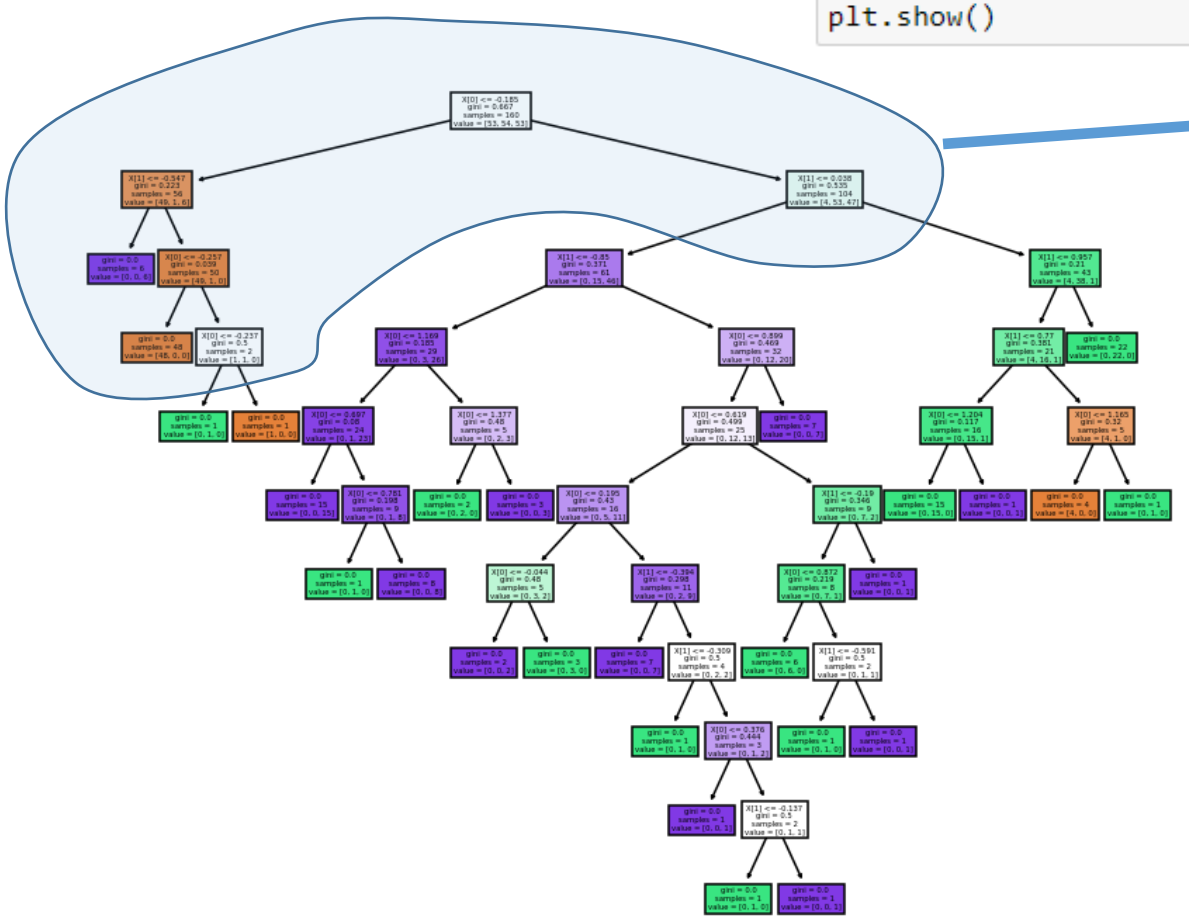
```
|   |   |--- feature_1 >  -0.85
|   |   |   |--- feature_0 <= 0.90
|   |   |   |   |--- feature_0 <= 0.62
|   |   |   |   |   |--- feature_0 <= 0.19
|   |   |   |   |   |   |--- feature_0 <= -0.04
|   |   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |   |--- feature_0 >  -0.04
|   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- feature_0 >  0.19
|   |   |   |   |   |   |--- feature_1 <= -0.39
|   |   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |   |--- feature_1 >  -0.39
|   |   |   |   |   |   |   |--- feature_1 <= -0.31
|   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |--- feature_1 >  -0.31
|   |   |   |   |   |   |   |   |--- feature_0 <= 0.38
|   |   |   |   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |   |   |   |--- feature_0 >  0.38
|   |   |   |   |   |   |   |   |   |--- feature_1 <= -0.14
|   |   |   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |   |   |--- feature_1 >  -0.14
|   |   |   |   |   |   |   |   |   |   |--- class: 2
```

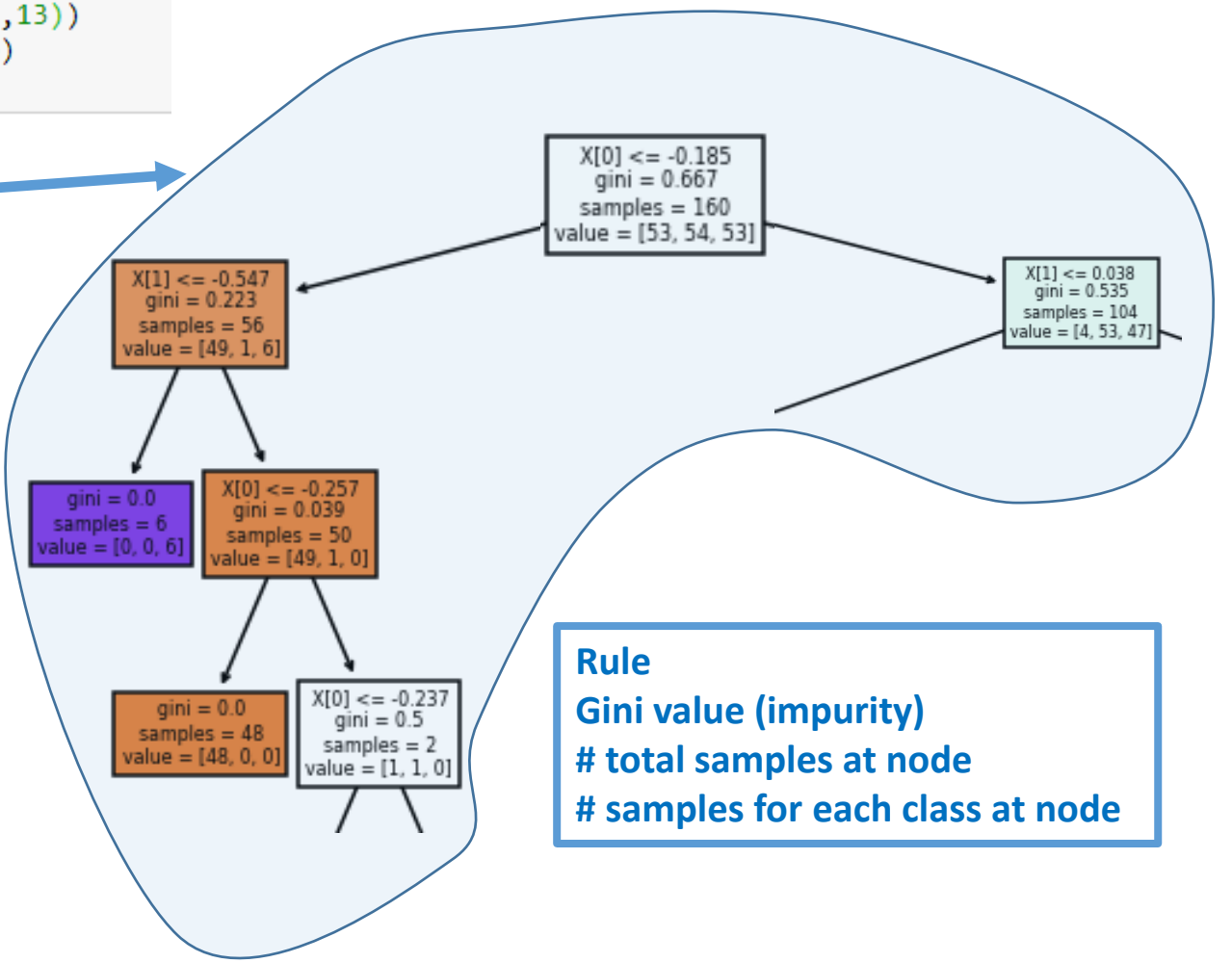
```
|   |   |   |   |--- feature_0 >  0.62
|   |   |   |   |   |--- feature_1 <= -0.19
|   |   |   |   |   |   |--- feature_0 <= 0.87
|   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |--- feature_0 >  0.87
|   |   |   |   |   |   |   |--- feature_1 <= -0.59
|   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |--- feature_1 >  -0.59
|   |   |   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |--- feature_1 >  -0.19
|   |   |   |   |   |   |--- class: 2
|   |   |   |--- feature_0 >  0.90
|   |   |   |   |--- class: 2
|   |--- feature_1 >  0.04
|   |   |--- feature_1 <= 0.96
|   |   |   |--- feature_1 <= 0.77
|   |   |   |   |--- feature_0 <= 1.20
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- feature_0 >  1.20
|   |   |   |   |   |--- class: 2
|   |   |   |--- feature_1 >  0.77
|   |   |   |   |--- feature_0 <= 1.16
|   |   |   |   |   |--- class: 0
|   |   |   |   |--- feature_0 >  1.16
|   |   |   |   |   |--- class: 1
|   |   |--- feature_1 >  0.96
|   |   |   |--- class: 1
```

Decision Tree in sklearn, Example 1

ccp_alpha=0.0

```
from sklearn.tree import plot_tree
plt.figure(3,figsize=(15,13))
plot_tree(C1,filled=True)
plt.show()
```





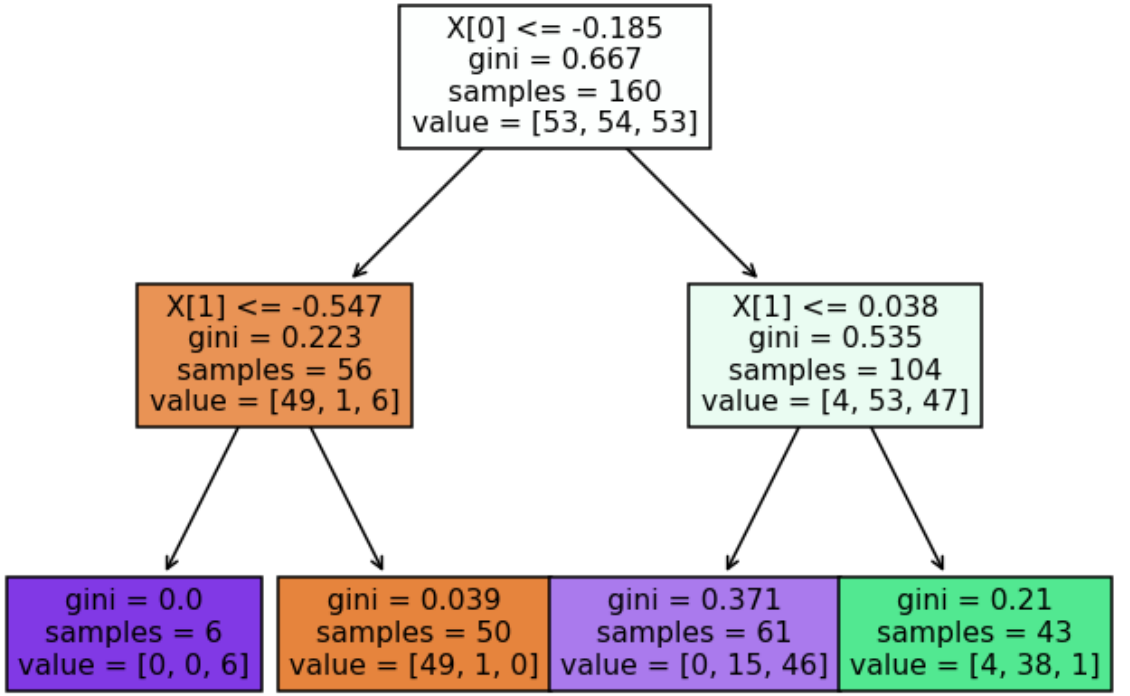
Rule
Gini value (impurity)
 # total samples at node
 # samples for each class at node

Decision Tree in sklearn, Example 1

ccp_alpha=0.05

```
from sklearn.tree import export_text
text_tree = export_text(C1)
print(text_tree)
```

```
|--- feature_0 <= -0.18
|   |--- feature_1 <= -0.55
|   |   |--- class: 2
|   |--- feature_1 >  -0.55
|   |   |--- class: 0
|--- feature_0 >  -0.18
|   |--- feature_1 <= 0.04
|   |   |--- class: 2
|   |--- feature_1 >  0.04
|   |   |--- class: 1
```



Random Forest Classifiers in sklearn

- A random forest is a supervised learning approach that fits a set of decision tree classifiers/regressors on different sub-samples (randomly drawn with replacement) of the input dataset.
- It averages the individual predictions to improve the predictive accuracy and control over-fitting.

class sklearn.ensemble.**RandomForestClassifier**(*n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None*)

- n_estimators: the number of the trees in the forest.
- bootstrap: if True, random samples from the whole data is drown randomly with replacement, otherwise the whole data is used to grow the tree.
- oob_score: if True, the out-of-bag approach is used with the bootstrap as a validation approach. The samples left out of the bootstrap sampling are used on trees that didn't see such examples while training. The class assigned to the oob samples is the majority vote of the classifications from all trees that are used to test these samples.

Random Forest Classifiers in sklearn

```
from sklearn.ensemble import RandomForestClassifier
RF1 = RandomForestClassifier(random_state=0,ccp_alpha=0.0)
```

```
RF1.fit(X_train,y_train)
```

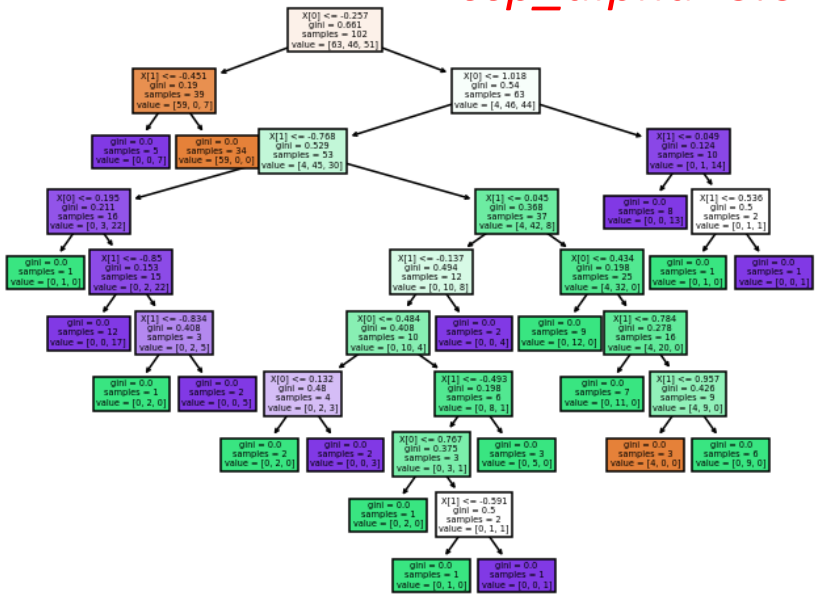
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=0, verbose=0,
                       warm_start=False)
```

```
y1 = RF1.predict(X_test)
```

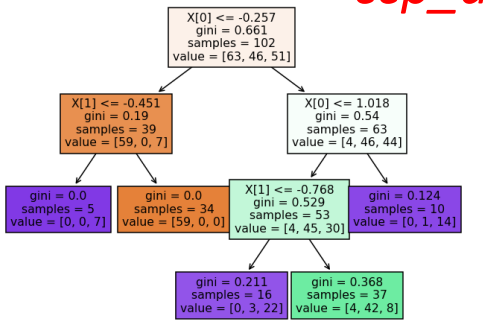
```
Z = RF1.predict(np.c_[x1grid.ravel(), x2grid.ravel()])
Z = Z.reshape(x1grid.shape)
```

```
plt.figure(6,figsize=(8,6))
print(len(RF1.estimators_))
plot_tree(RF1.estimators_[0],filled=True)
plt.show()
```

ccp_alpha=0.0

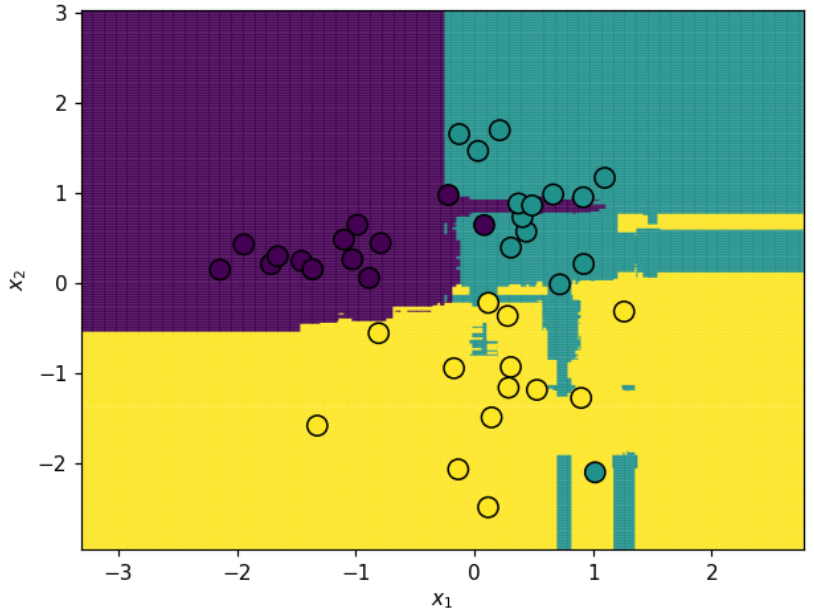


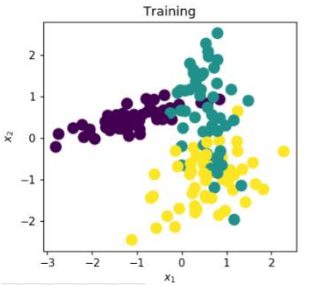
ccp_alpha=0.05



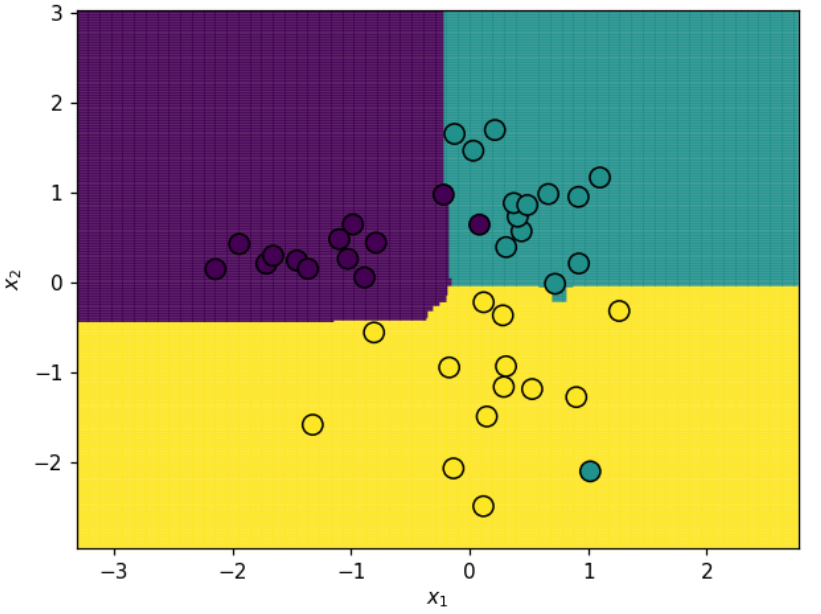
Random Forest Classifiers in sklearn

ccp_alpha=0.0





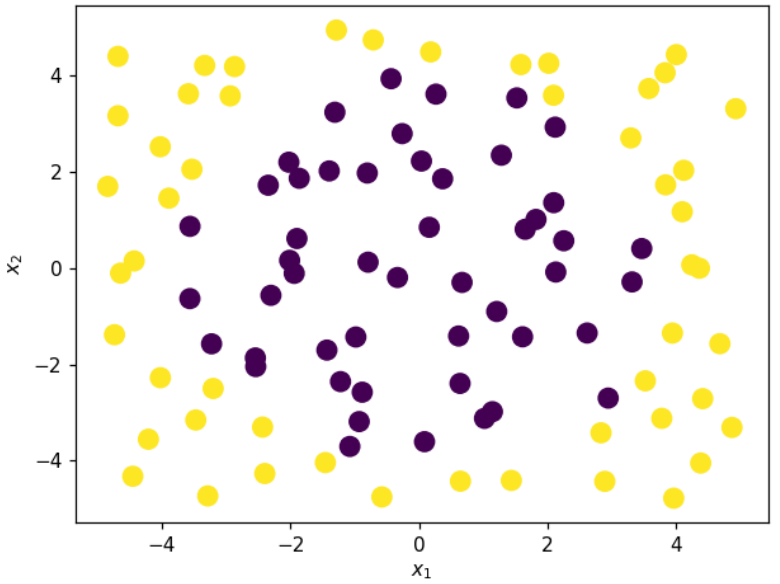
ccp_alpha=0.05



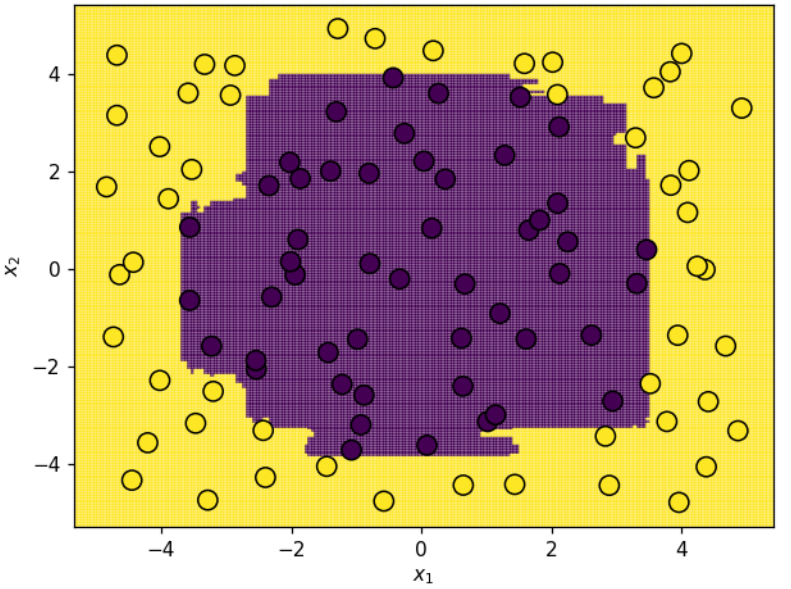
Accuracy = 0.85

Accuracy = 0.95

Random Forests Classifiers in sklearn



ccp_alpha=0.0



ccp_alpha=0.05