Decision Tree

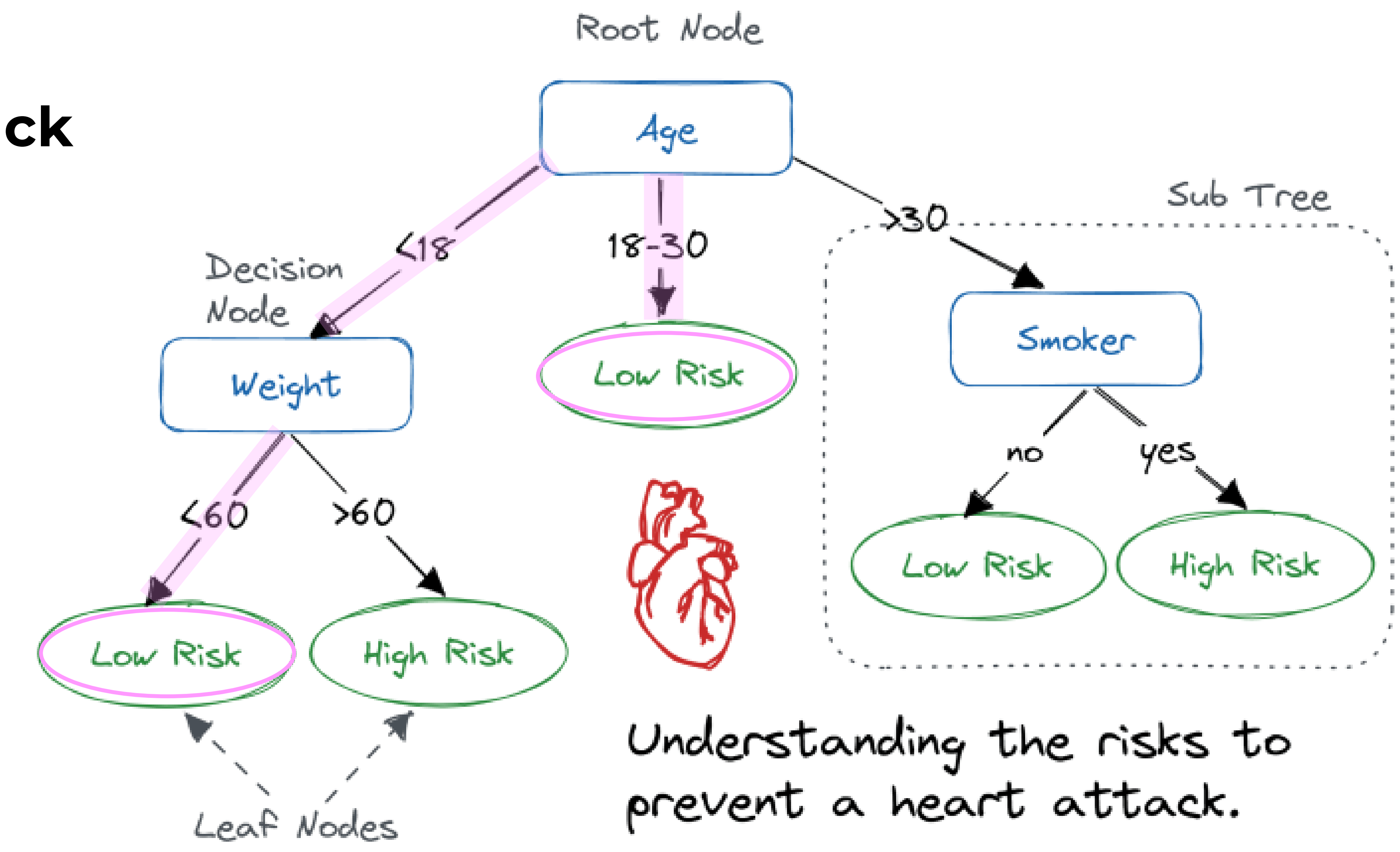
Kanokkorn Pimcharoen

Decision Tree Structure

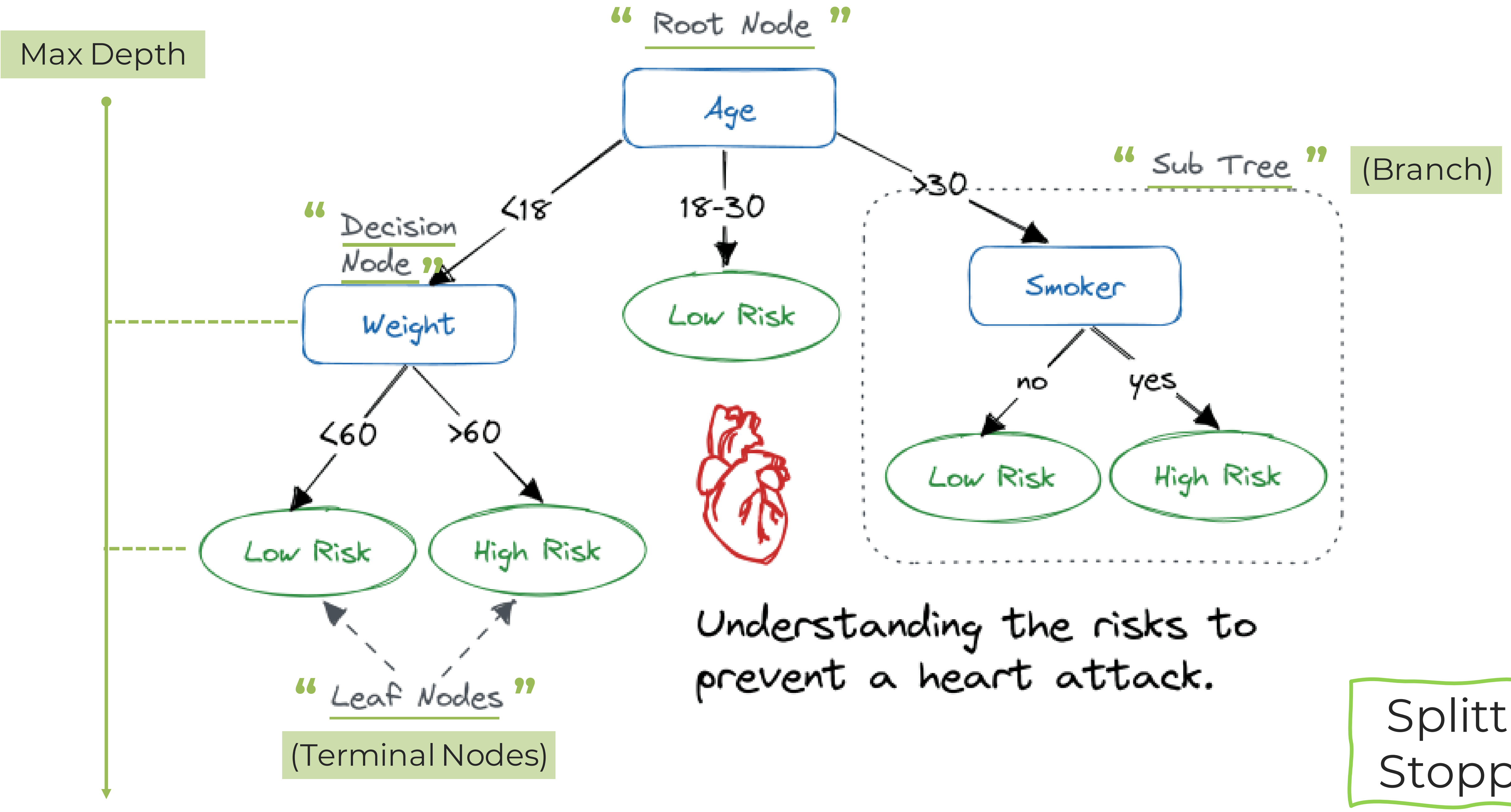
Risk for heart attack

Features:
- Age
- Weight
- Smoker

Labels/classes:
- Low risk
- High risk



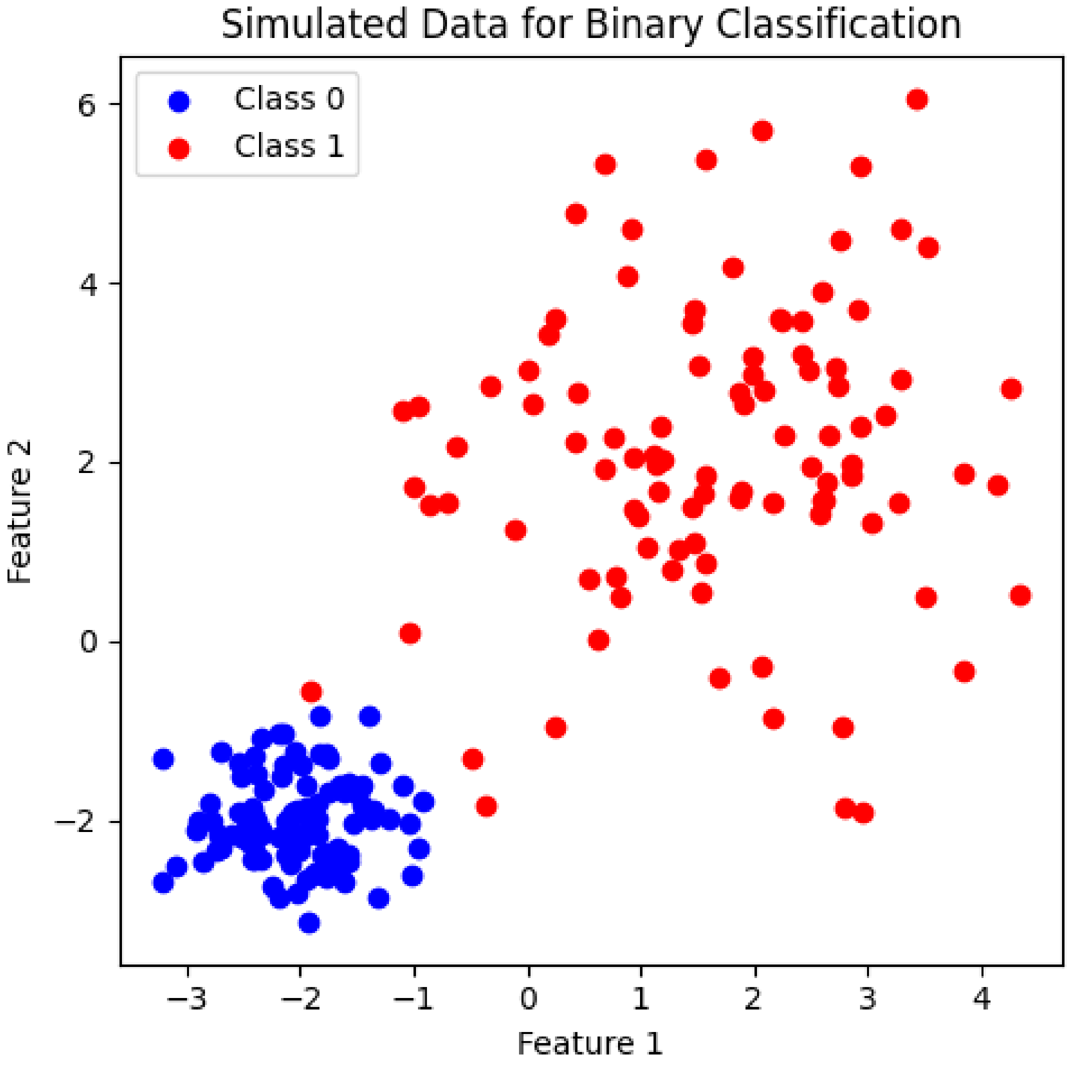
Decision Tree Structure



Splitting Criteria?
Stopping Criteria?

Brain
Code
Camp

Decision Tree



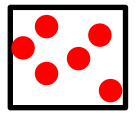
Dataset
Class 0: 100 data
Class 1 : 100 data

Gini Impurity $gini = \sum_i p_i(1 - p_i)$

where p_i is the proportion of class i in a node, such that $\sum_i p_i = 1$.

For binary classification (2 classes: $p_0 + p_1 = 1$),

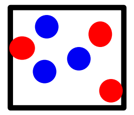
$gini$ is minimum at $p_i = 0$ or $p_i = 1$ → Perfect split



$p_0 = 0,\ p_1 = 6/6 = 1$

$gini = 0 \times (1 - 0) + 1 \times (1 - 1) = 0$

$gini$ is maximum at $p_i = 0.5$ → Equally split



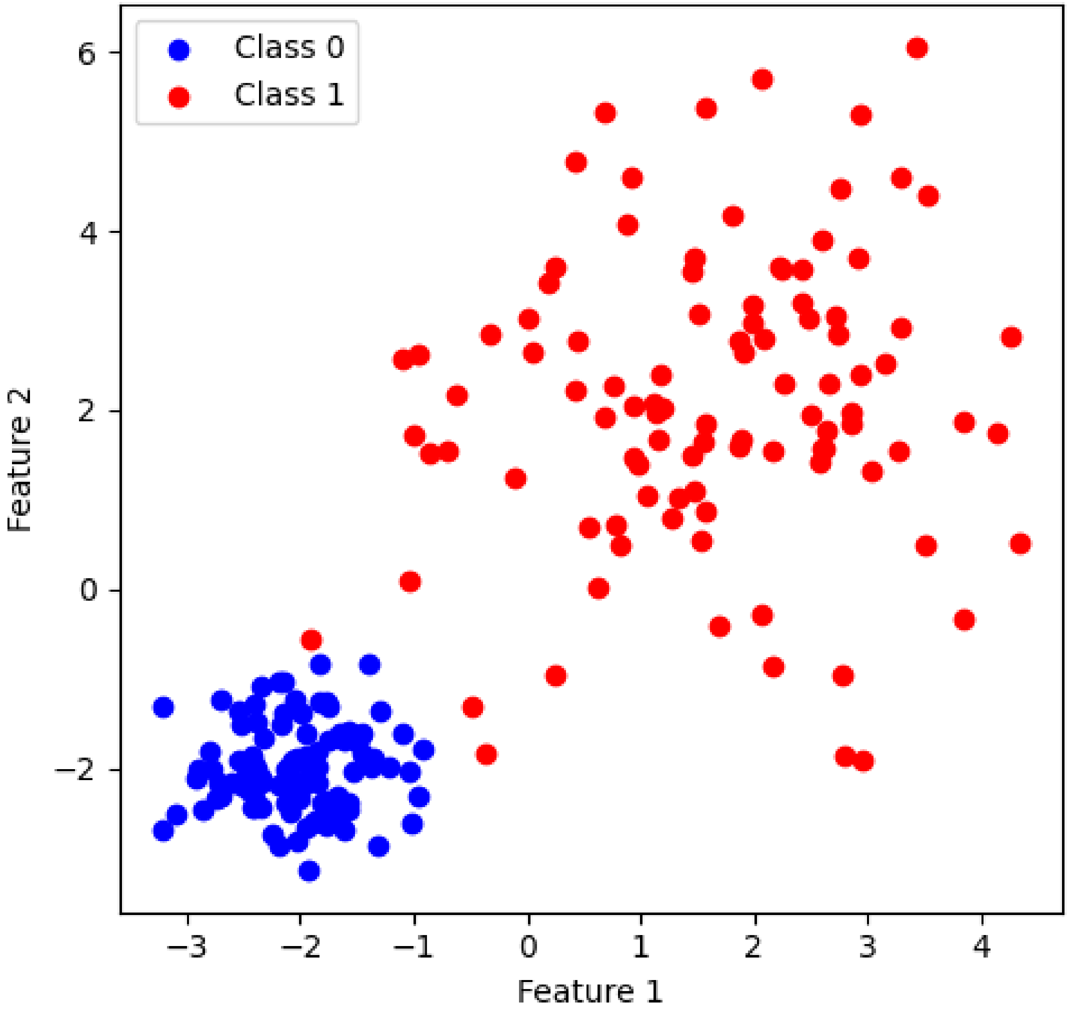
$p_0 = p_1 = 3/6 = 0.5$

$gini = 0.5 \times (1 - 0.5) + 0.5 \times (1 - 0.5) = 0.5$

Decision Tree

Brain Code Camp

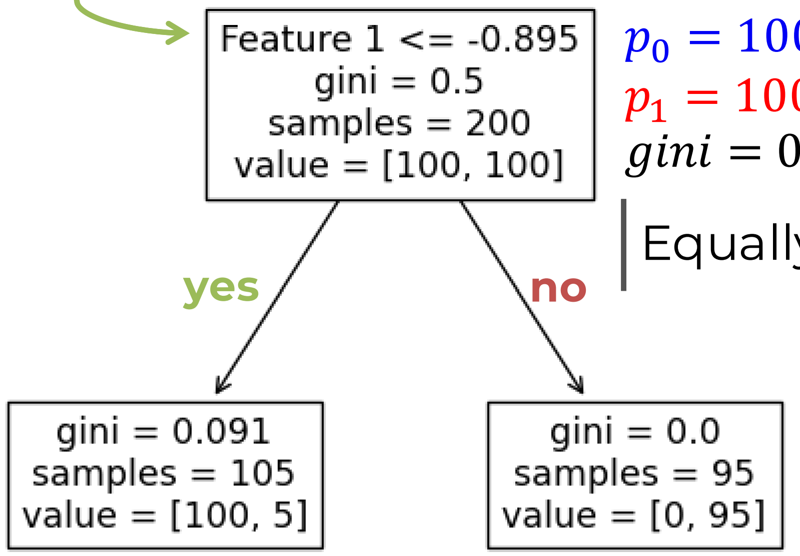
Simulated Data for Binary Classification



Dataset
Class 0: 100 data
Class 1 : 100 data

Split on Feature 1

Decision Condition



$p_0 = 100/200 = 0.5$
$p_1 = 100/200 = 0.5$
$gini = 0.5$

Equally split

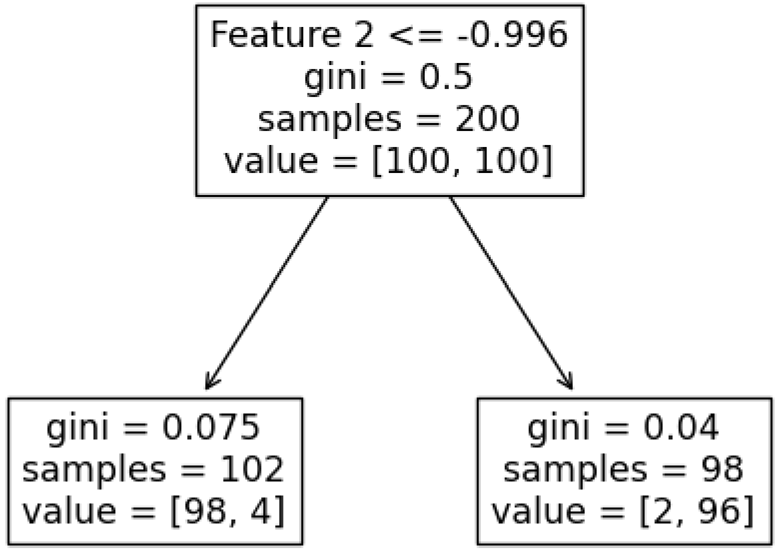
$p_0 = 100/105 = 0.952$
$p_1 = 5/105 = 0.0476$
$gini = 0.952 \times (1 - 0.952) + 0.0475 \times (1 - 0.0476) = 0.091$

$p_0 = 0/95 = 0$
$p_1 = 95/95 = 1$
$gini = 0$

Perfect split

$$\boxed{weighted\ gini = \left(\frac{105}{200}\right) \times 0.091 + \left(\frac{95}{200}\right) \times 0 = 0.048}$$

Split on Feature 2



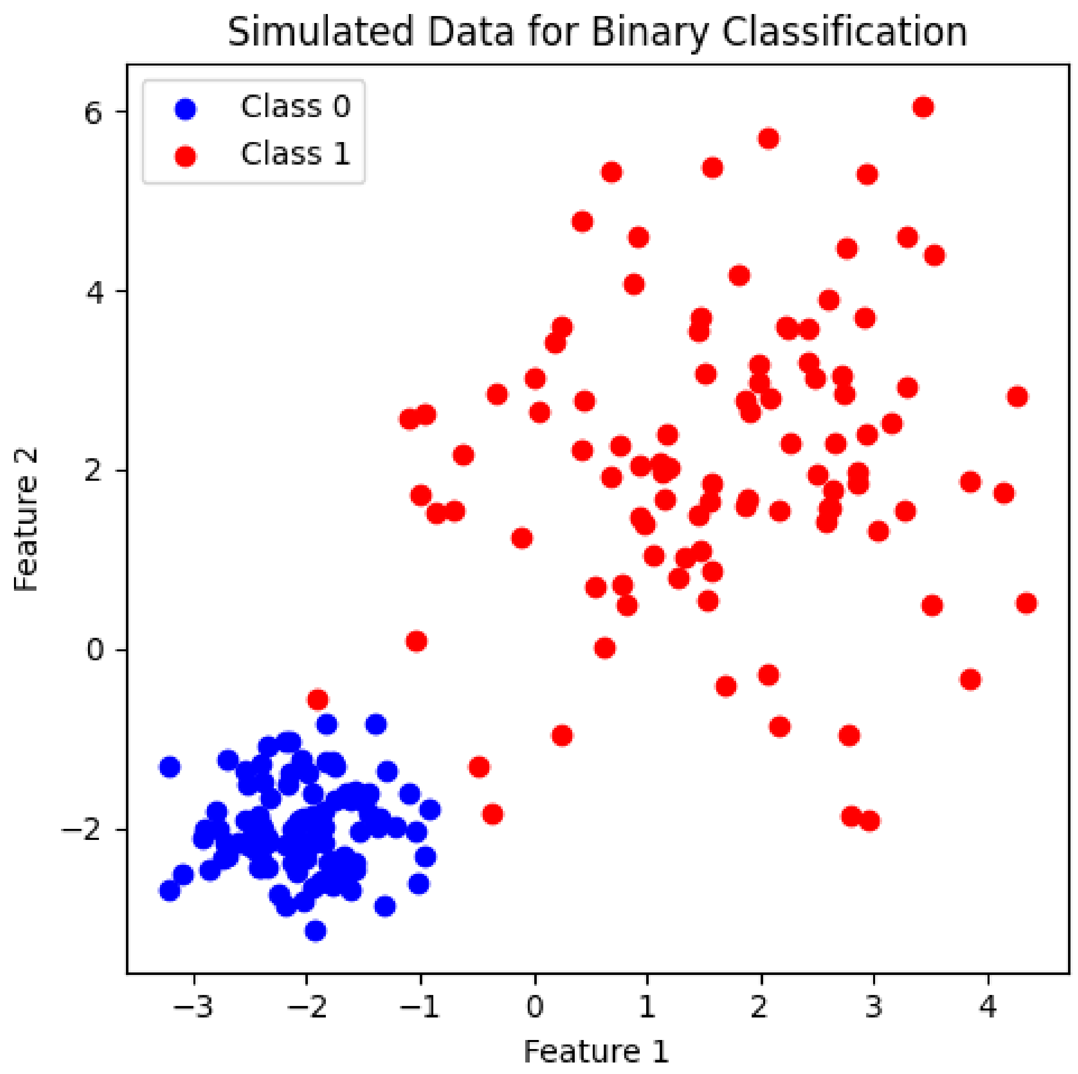
$$weighted\ gini = \left(\frac{102}{200}\right) \times 0.075 + \left(\frac{98}{200}\right) \times 0.04 = 0.058$$

Splitting Criteria

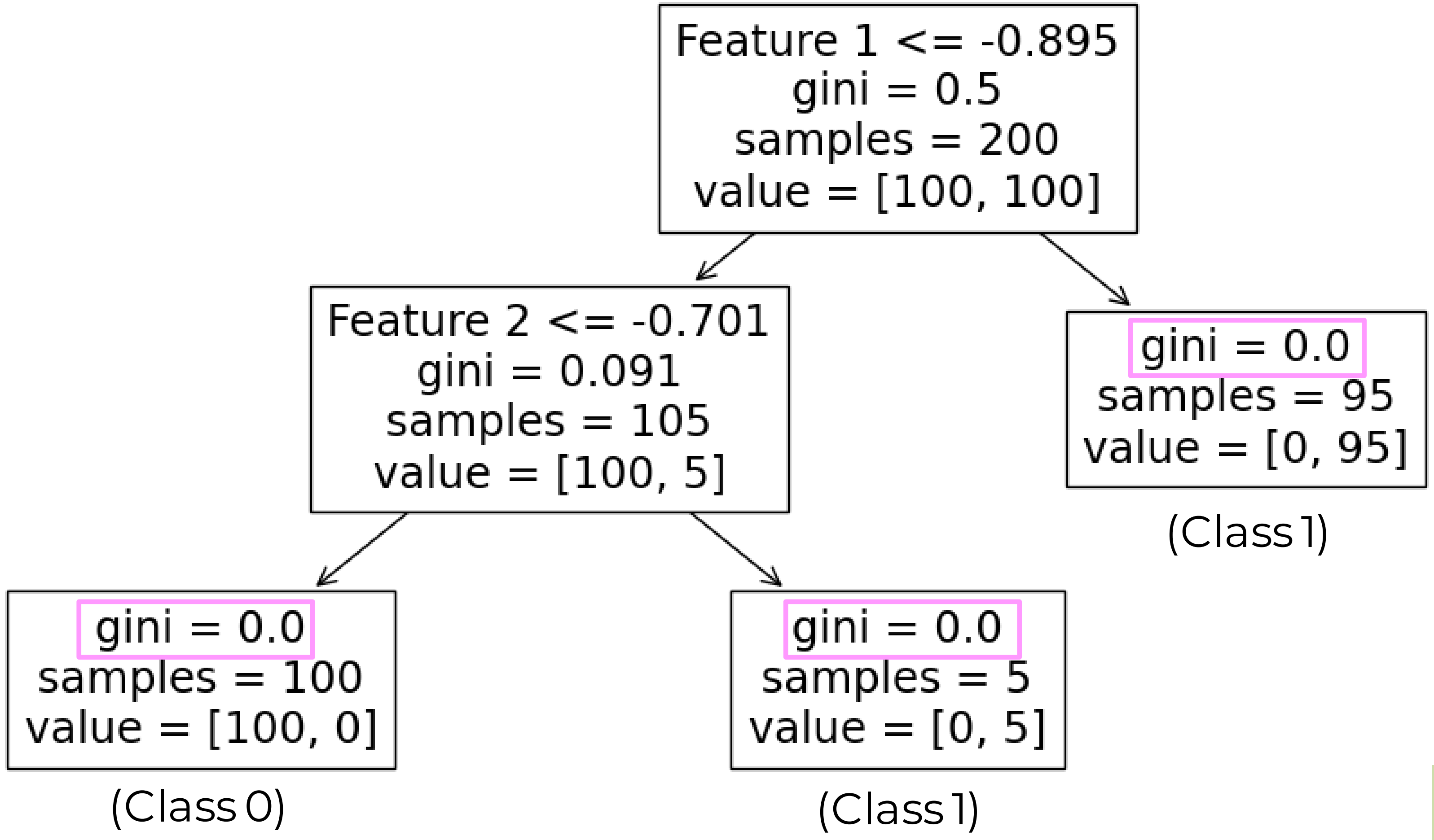
Gini Impurity $gini = \sum_i p_i(1 - p_i)$ where p_i is the proportion of class i in a node, such that $\sum_i p_i = 1$.

Brain Code Camp

Decision Tree



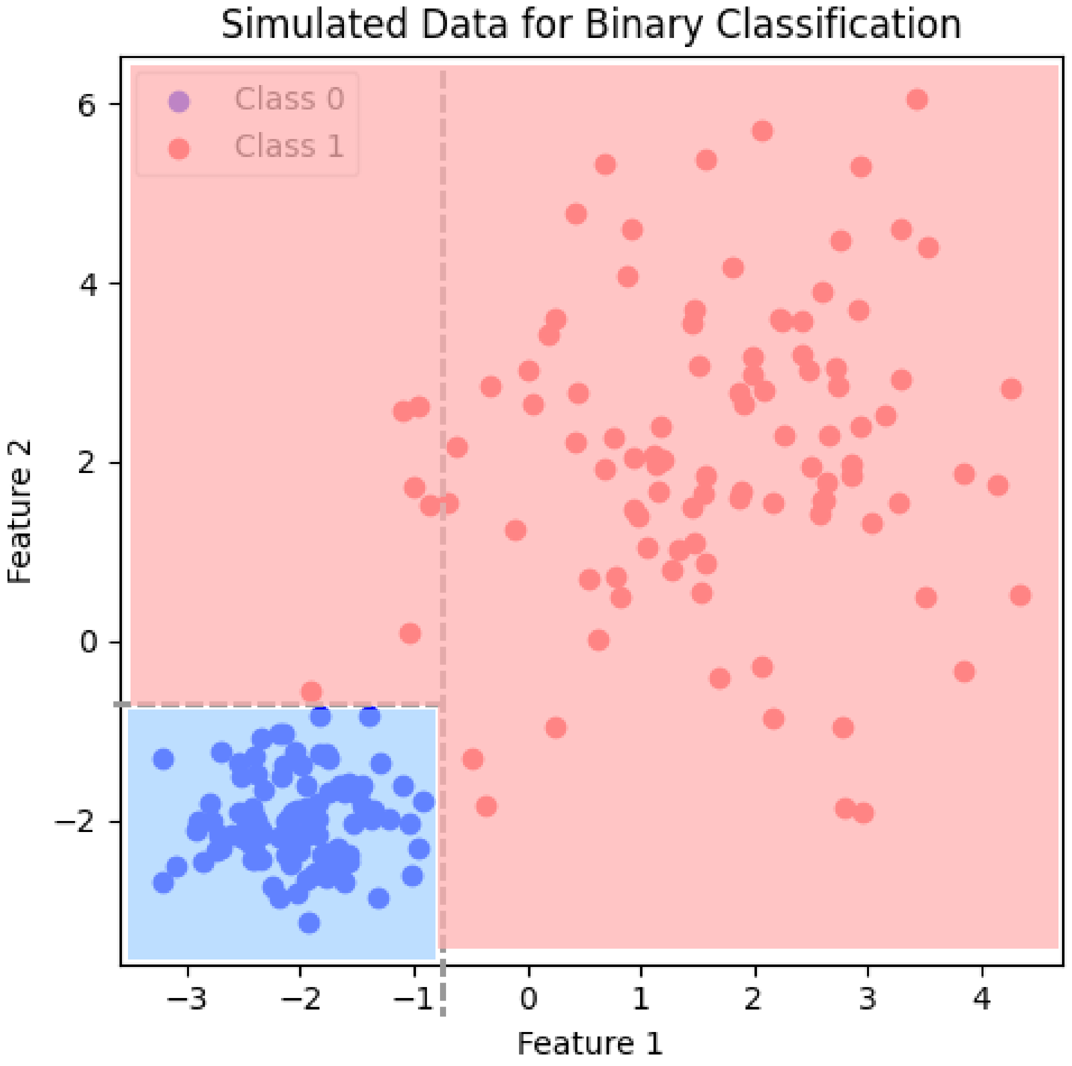
Dataset
Class 0: 100 data
Class 1: 100 data

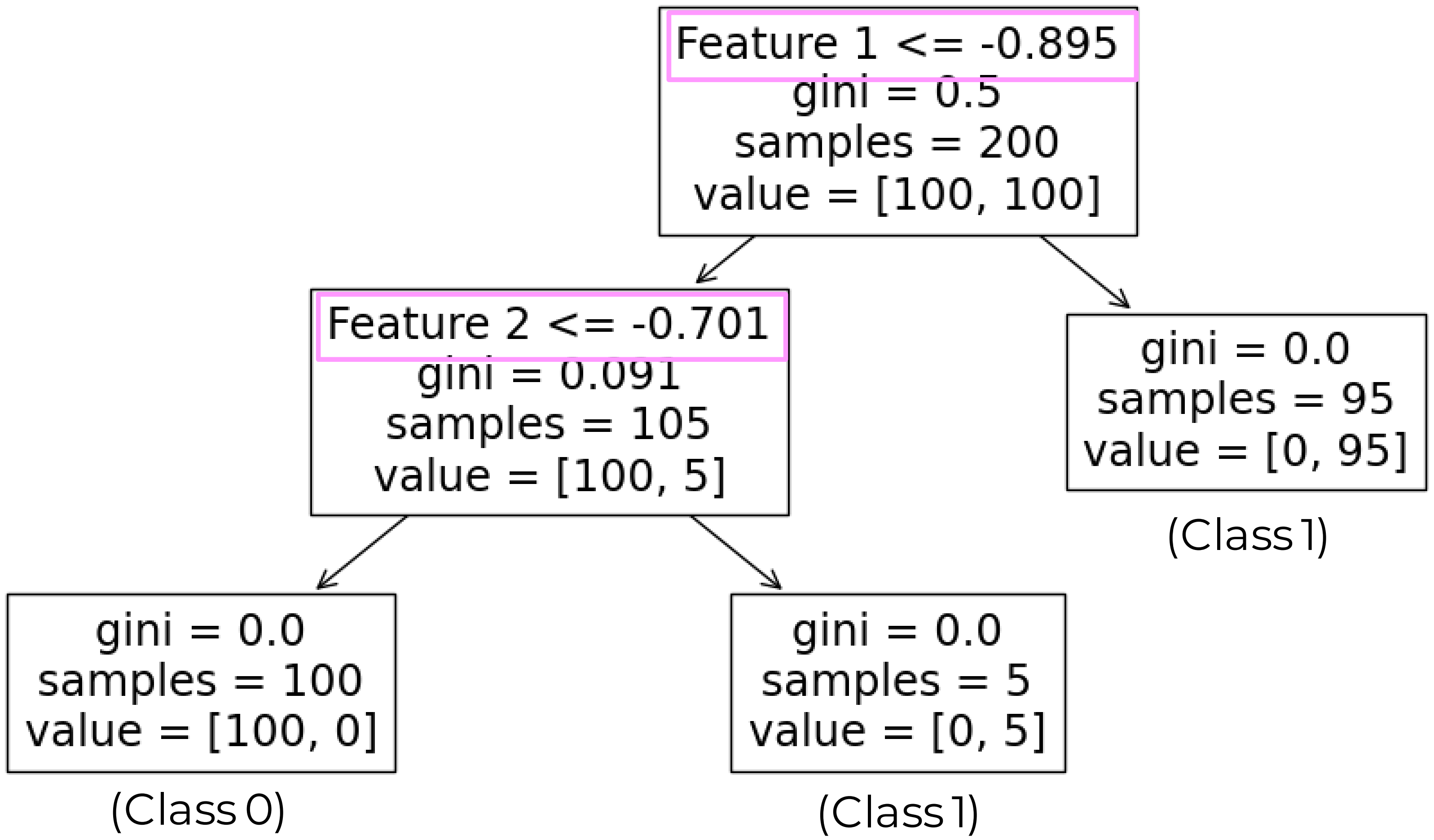


Stopping Criteria

Gini Impurity $gini = \sum_i p_i(1-p_i)$ where p_i is the proportion of class i in a node, such that $\sum_i p_i = 1$.

Decision Tree

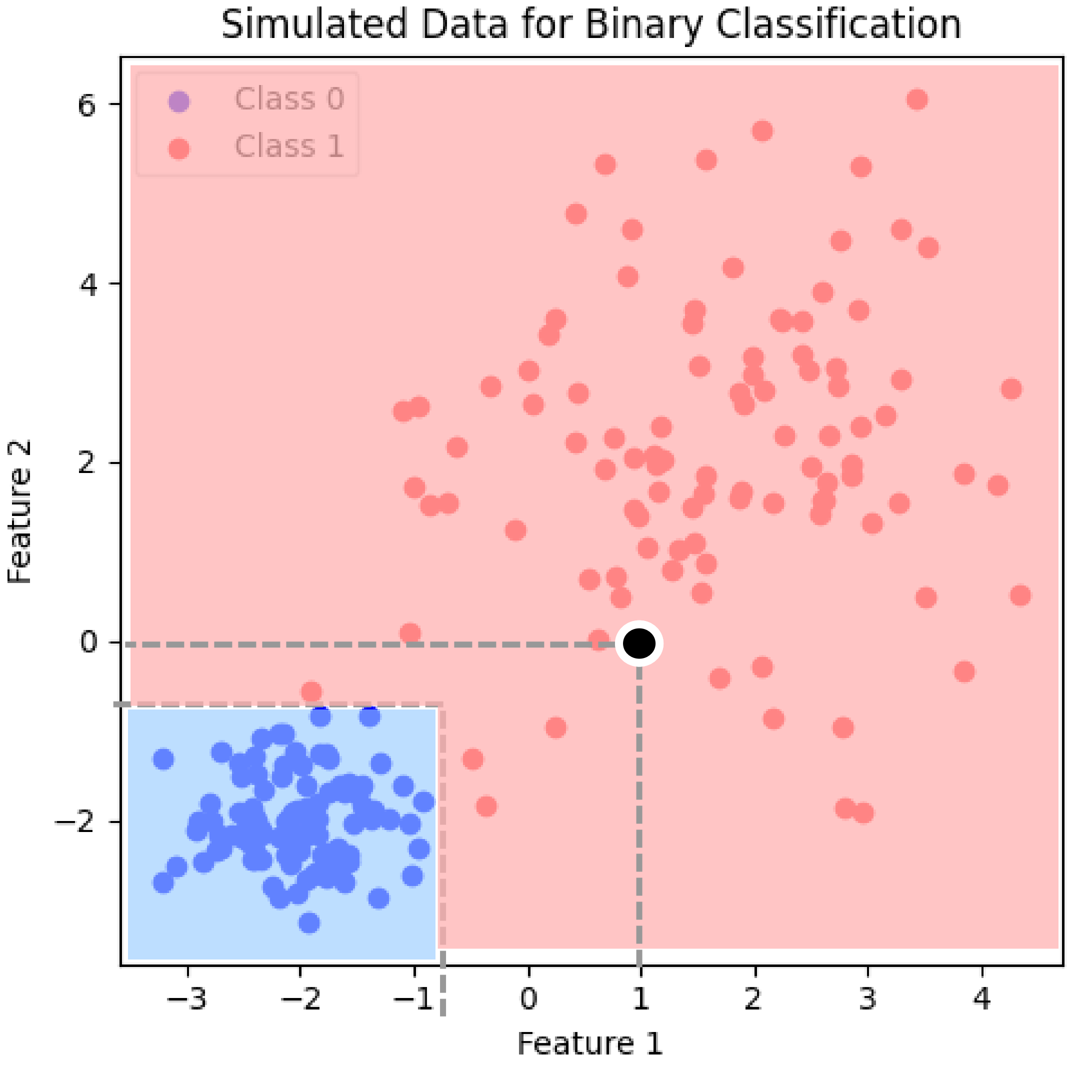




Dataset
Class 0: 100 data
Class 1: 100 data

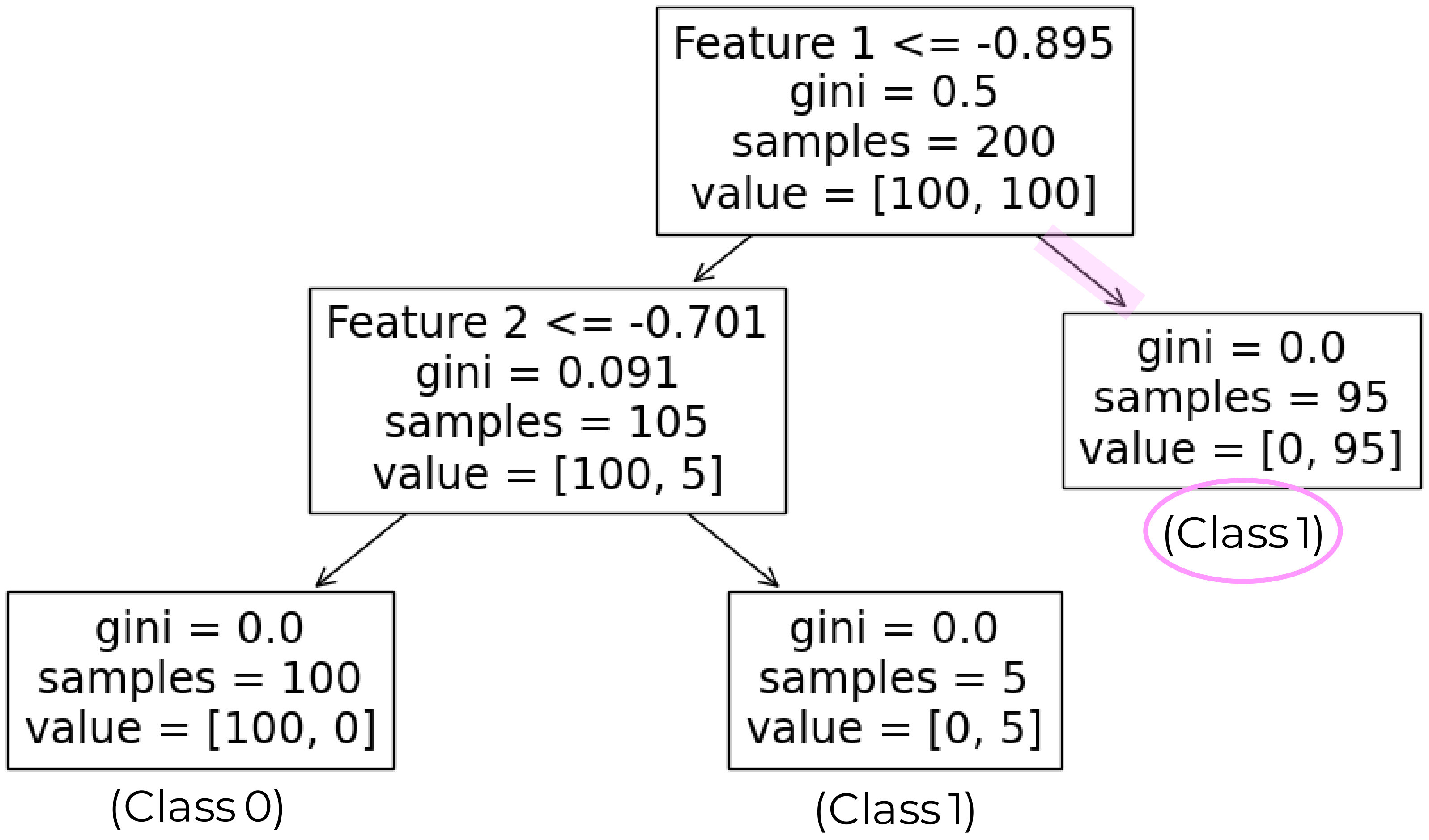
Gini Impurity $gini = \sum_i p_i(1 - p_i)$ where p_i is the proportion of class i in a node, such that $\sum_i p_i = 1$.

Decision Tree



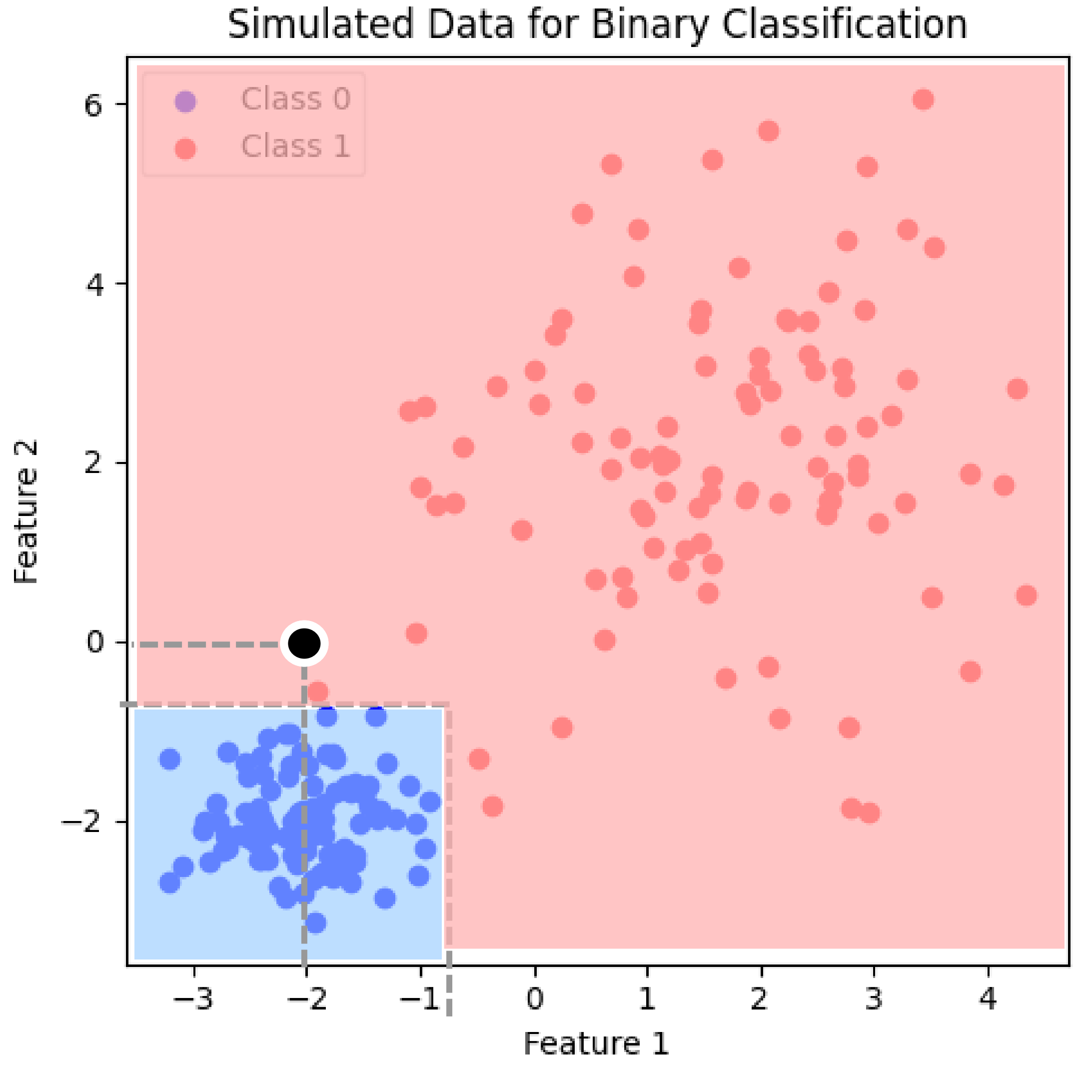
Dataset
Class 0: 100 data
Class 1 : 100 data

New data: Feature 1 = 1, Feature 2 = 0, Class?



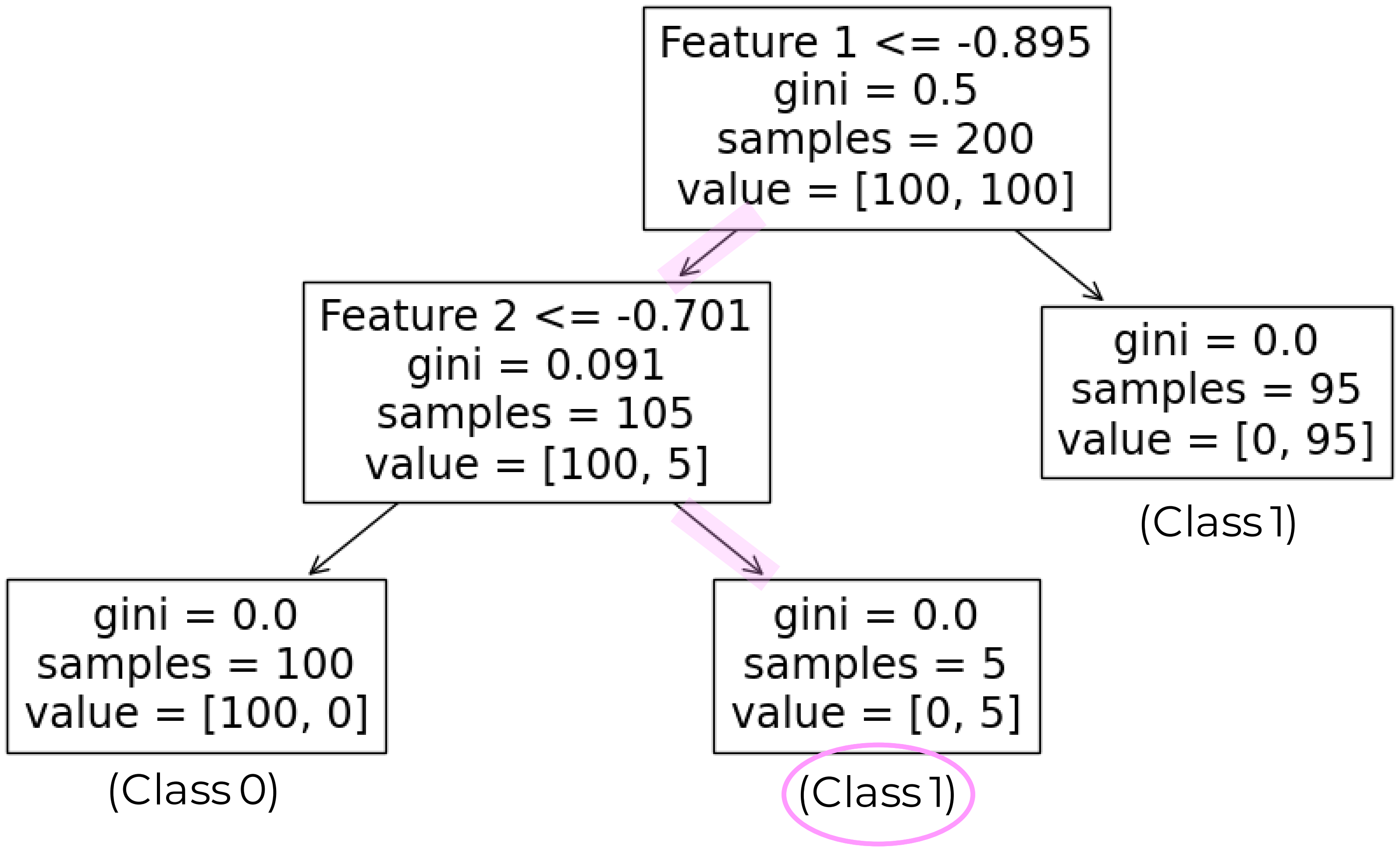
Gini Impurity $$gini = \sum_i p_i(1 - p_i)$$ where p_i is the proportion of class i in a node, such that $\sum_i p_i = 1$.

Decision Tree



Dataset
Class 0: 100 data
Class 1 : 100 data

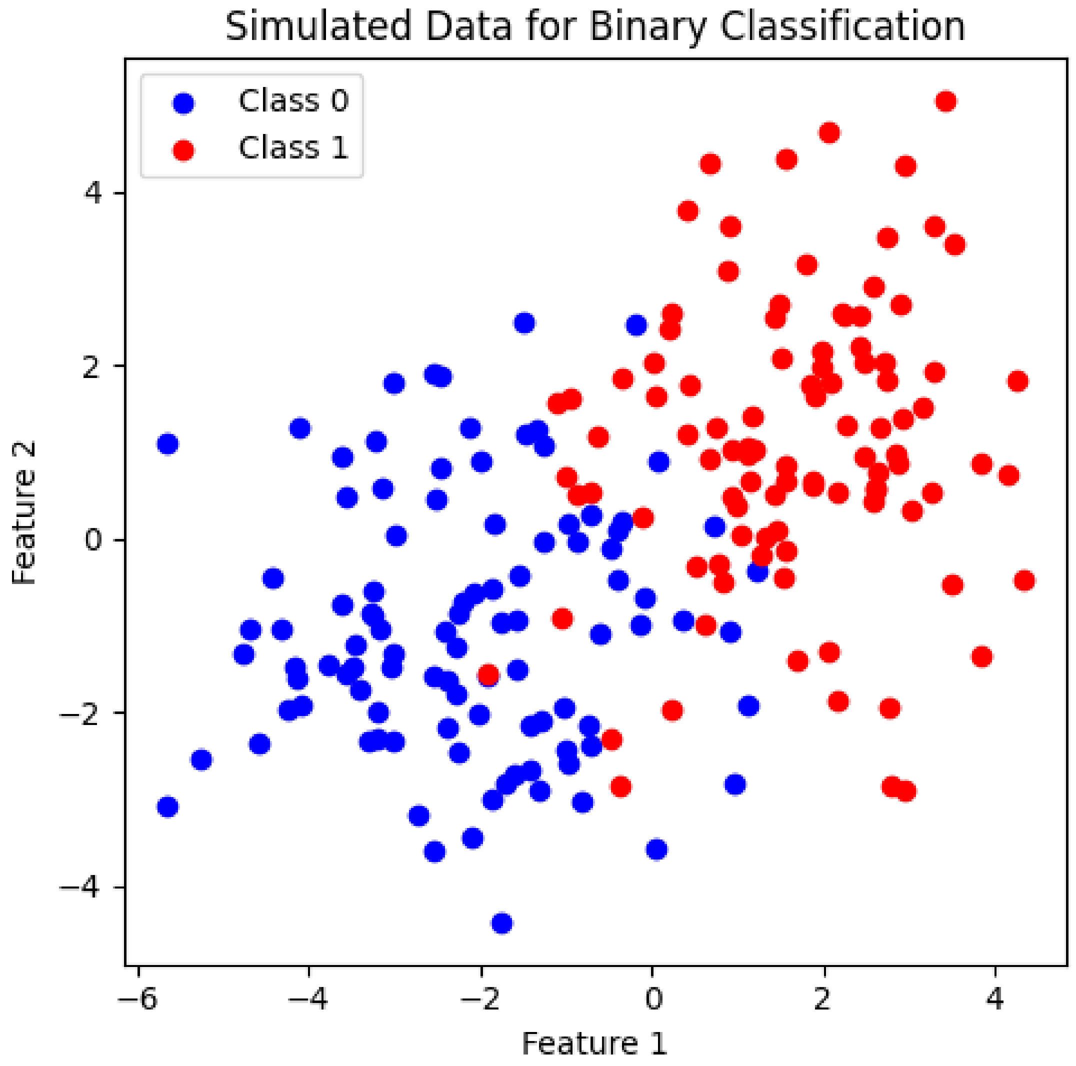
New data: Feature 1 = -2, Feature 2 = 0, Class?



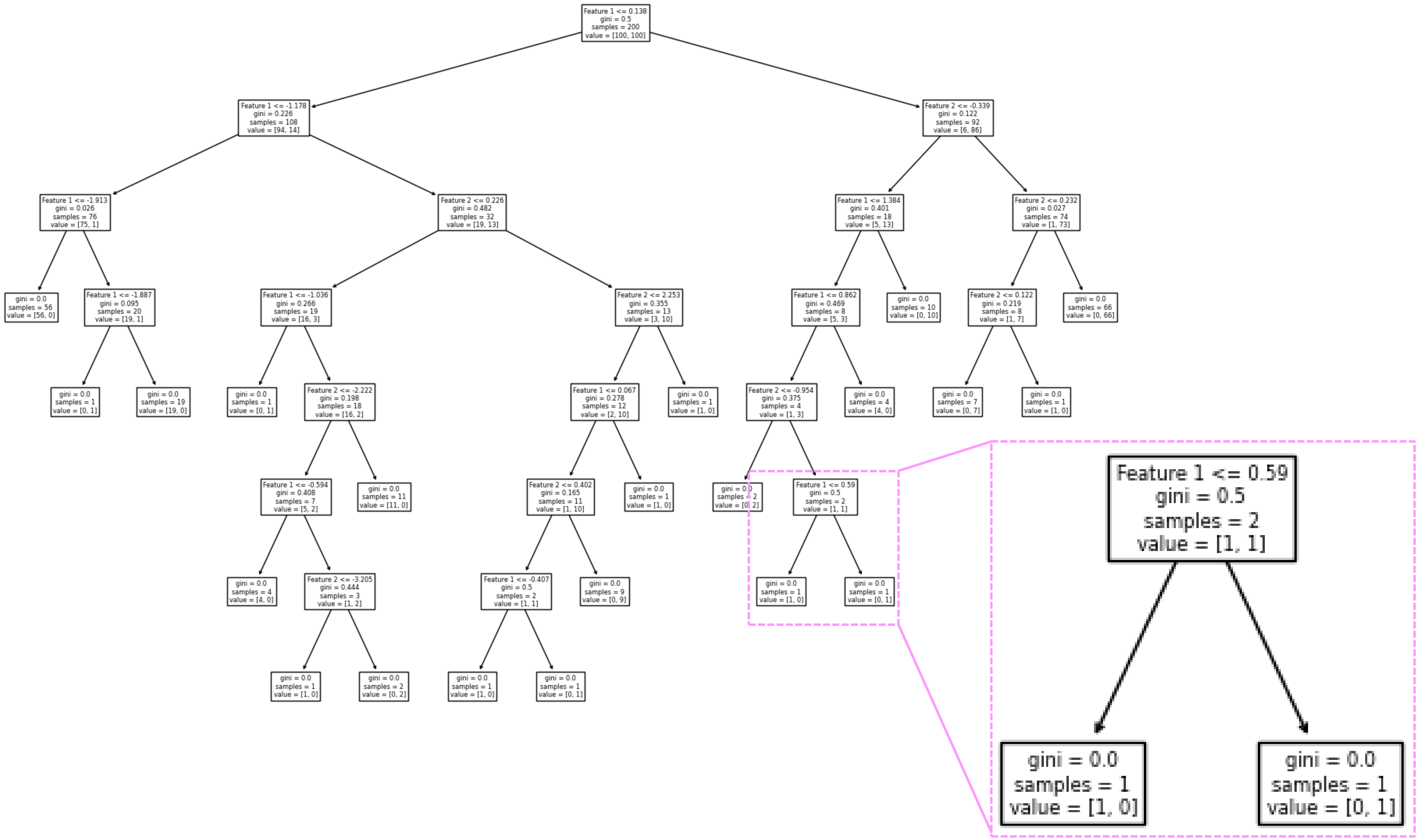
Gini Impurity $gini = \sum_i p_i(1 - p_i)$ where p_i is the proportion of class i in a node, such that $\sum_i p_i = 1$.

Brain
Code
Camp

Decision Tree

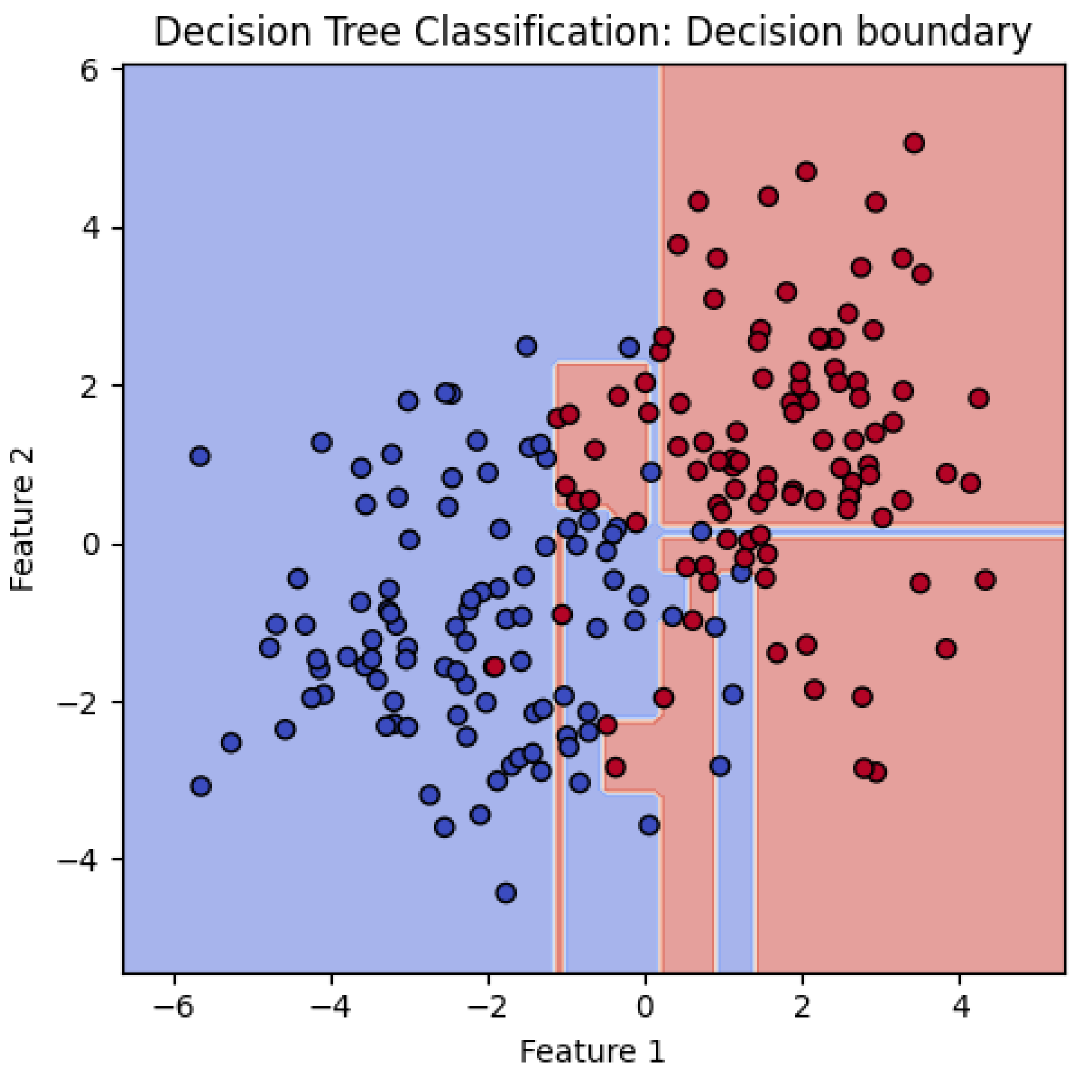


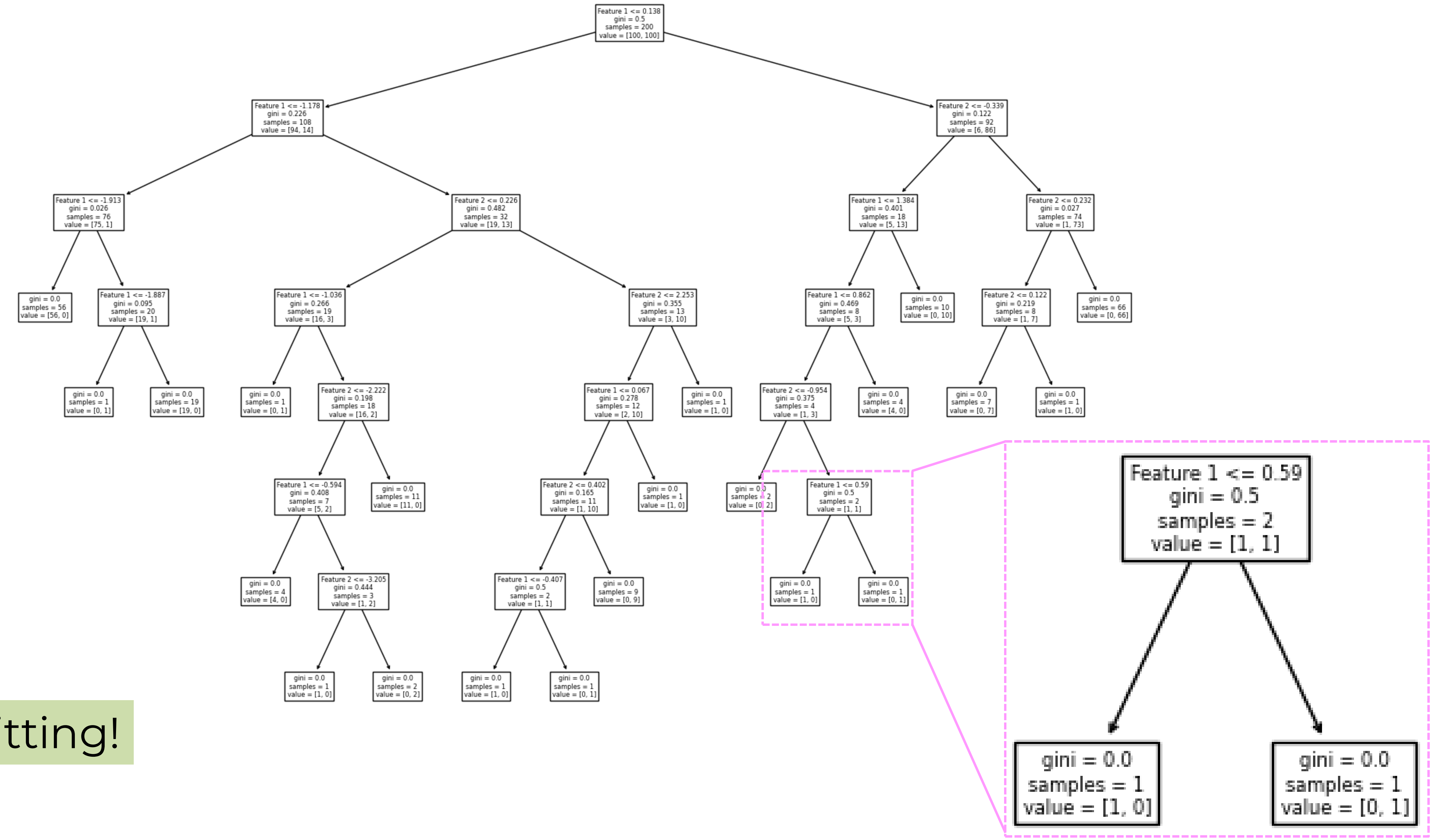
Dataset
Class 0: 100 data
Class 1: 100 data



Brain Code Camp

Decision Tree





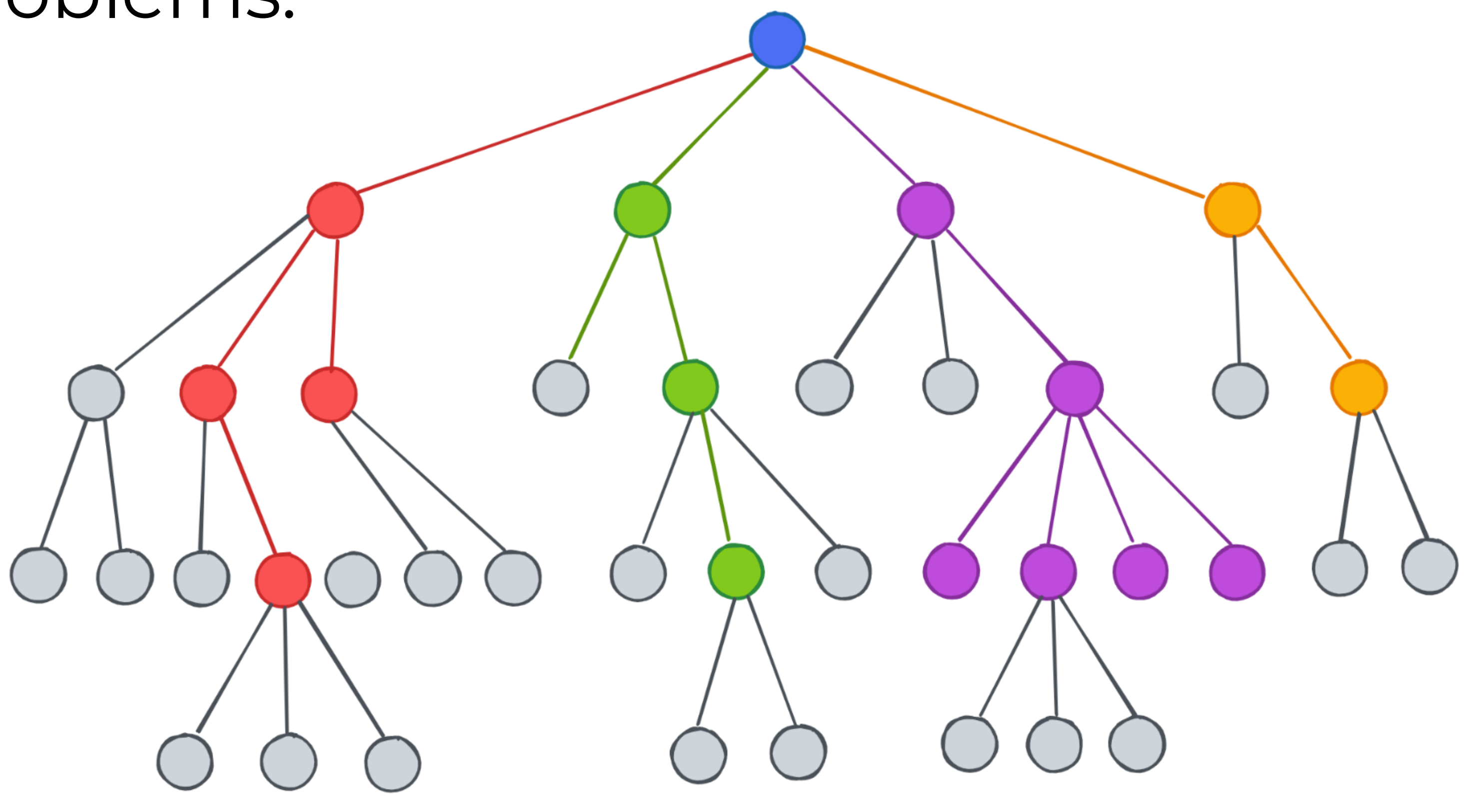
Dataset
Class 0: 100 data
Class 1: 100 data

Overfitting!

สร้างคน ข้ามพรมแดน

Decision Tree

Pros:
- Can solve both linear and non-linear problems.
- Can ignore redundant features.
- Easy to visualize and explain.



Cons:
- Easy to overfit.
- Does not generalize well.
- Large trees are hard to interpret.

Brain Code Camp

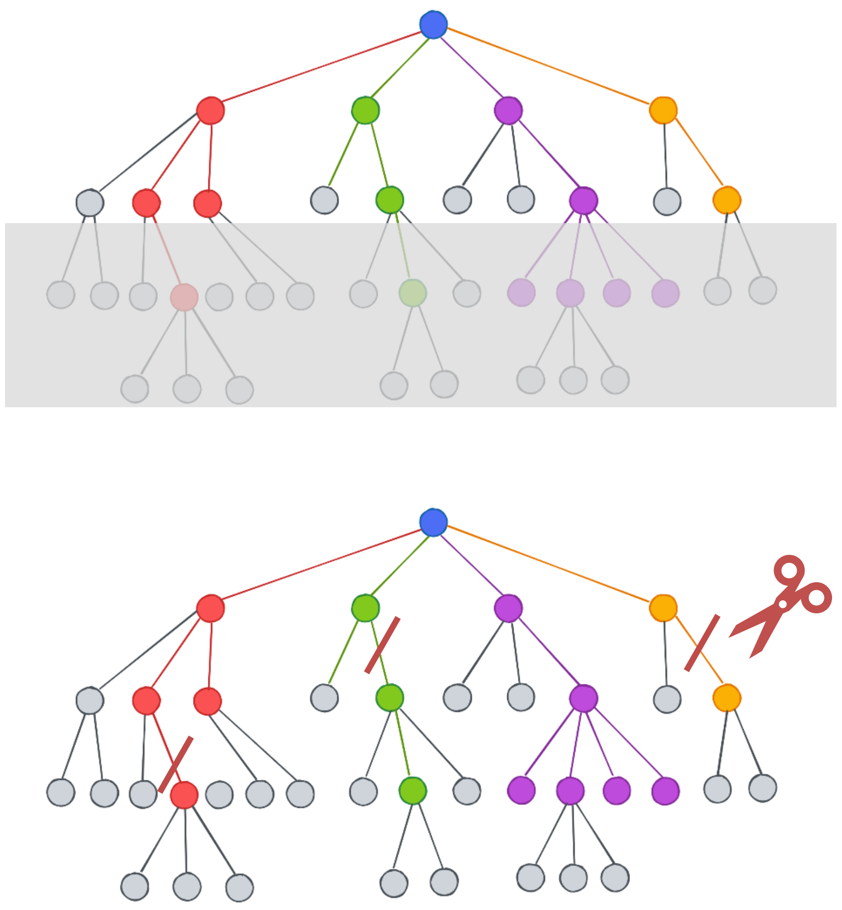
Decision Tree

Kanokkorn Pimcharoen

Overfitting

How to avoid overfitting in decision tree?

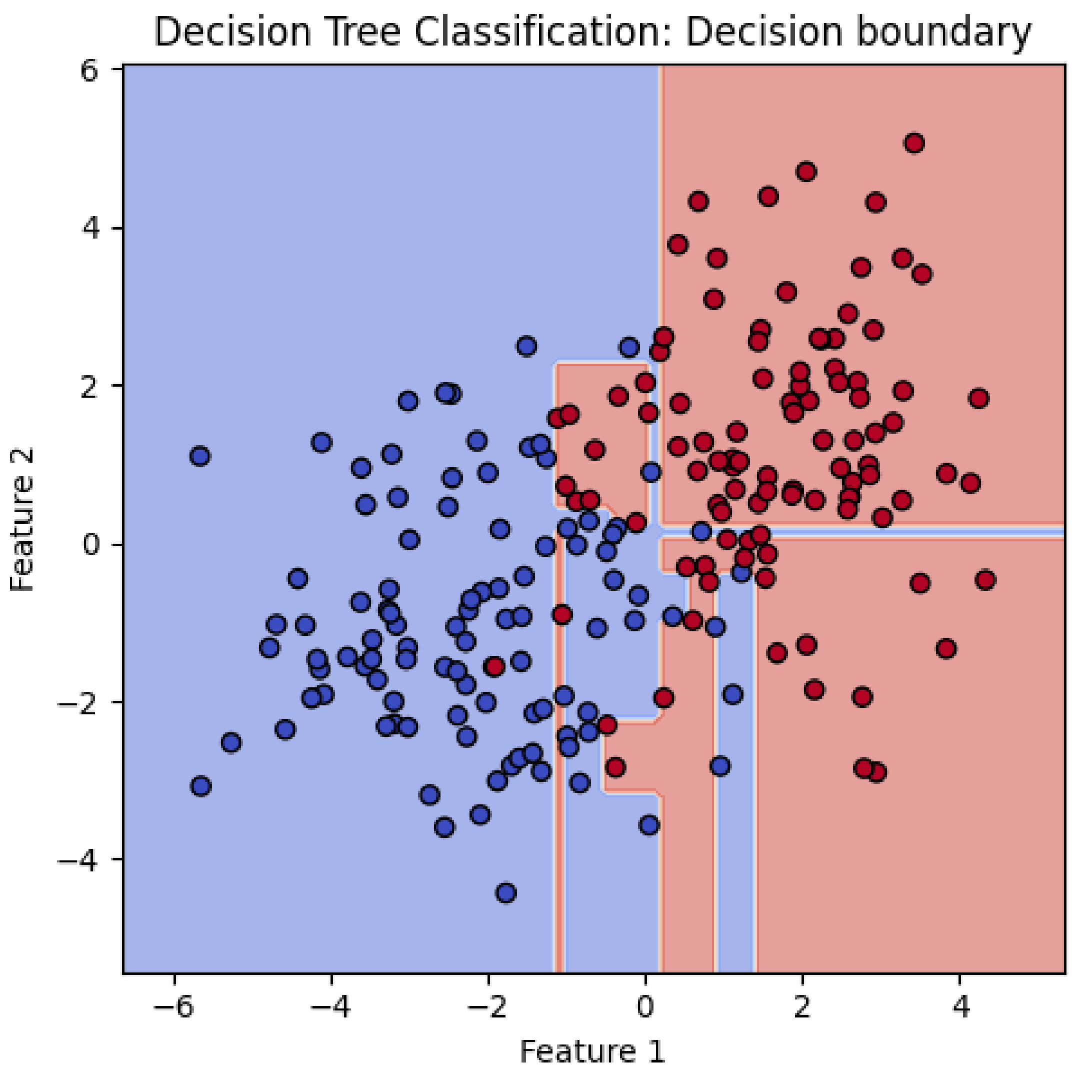
Early Stopping: stop growing before a tree becomes too complex.
- Limit tree depth.
- Do not split nodes which contain too few data points.
- Do not split nodes which do not significantly decrease impurity.



Post Pruning: grow full tree (overfitting), then later simplify the tree.

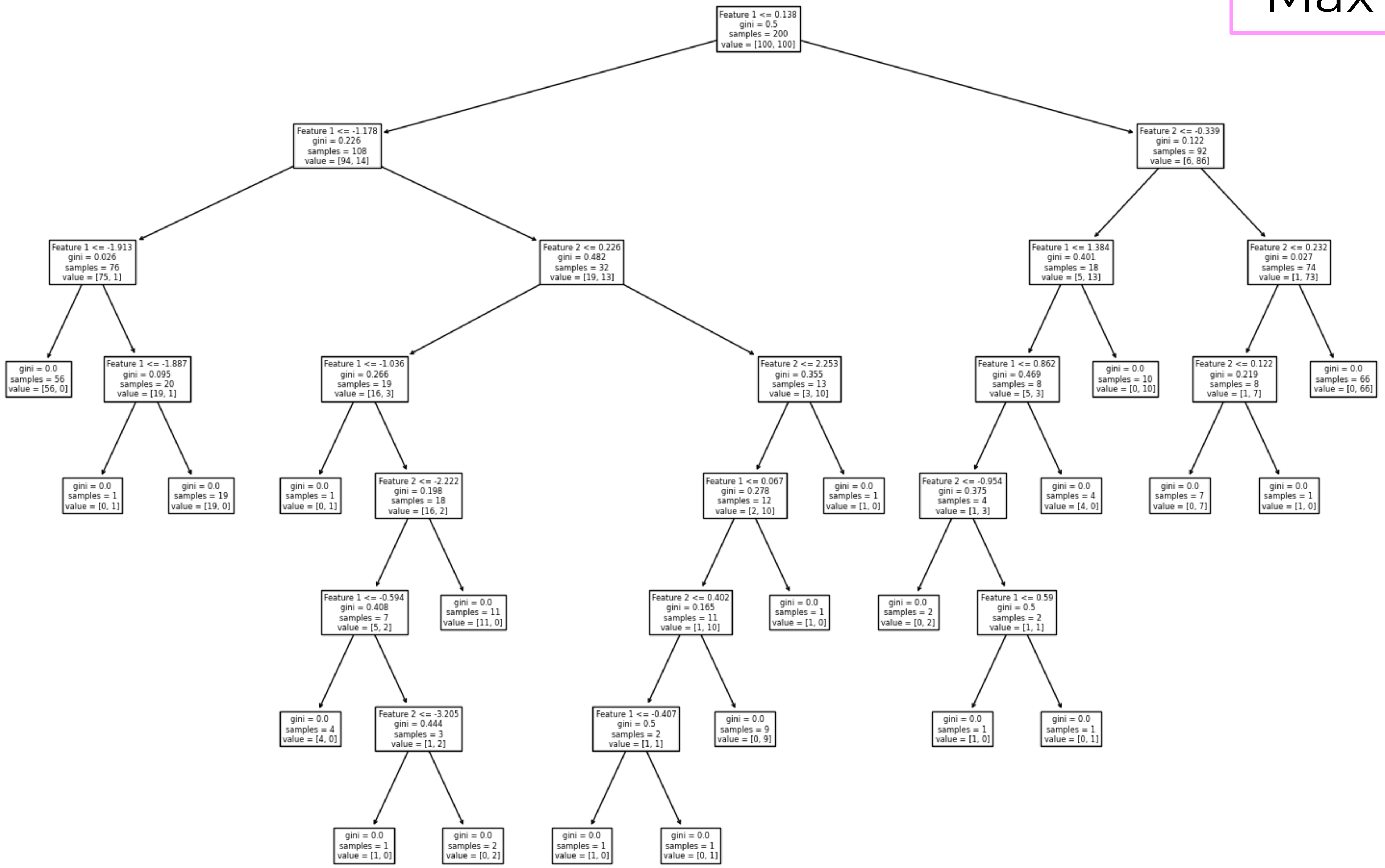
Ensemble Methods: Use multiple trees to obtain better predictive performance. Wisdom of the crowds.
(see random forest video)

Early Stopping

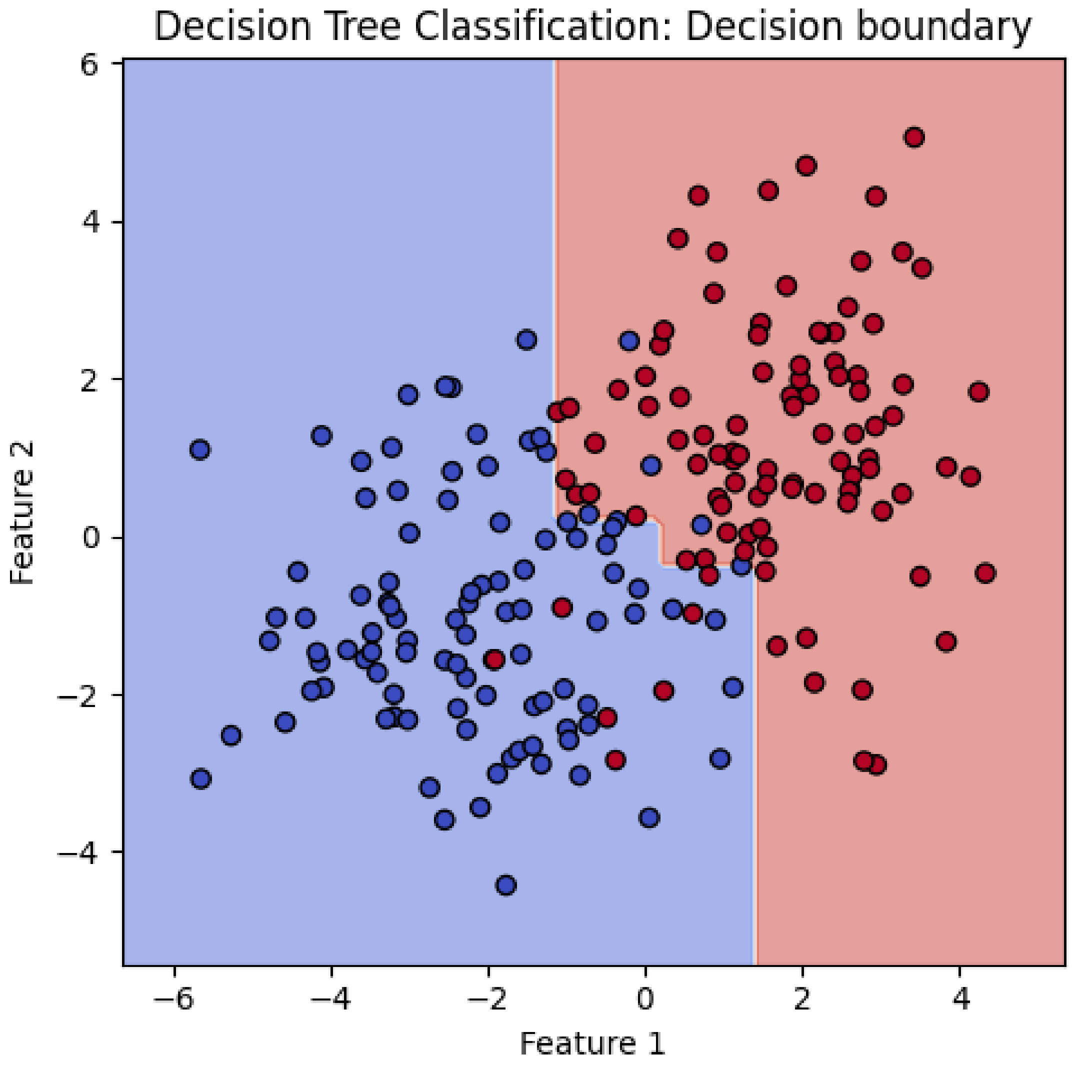


Dataset
Class 0: 100 data
Class 1: 100 data

Max Depth = 7

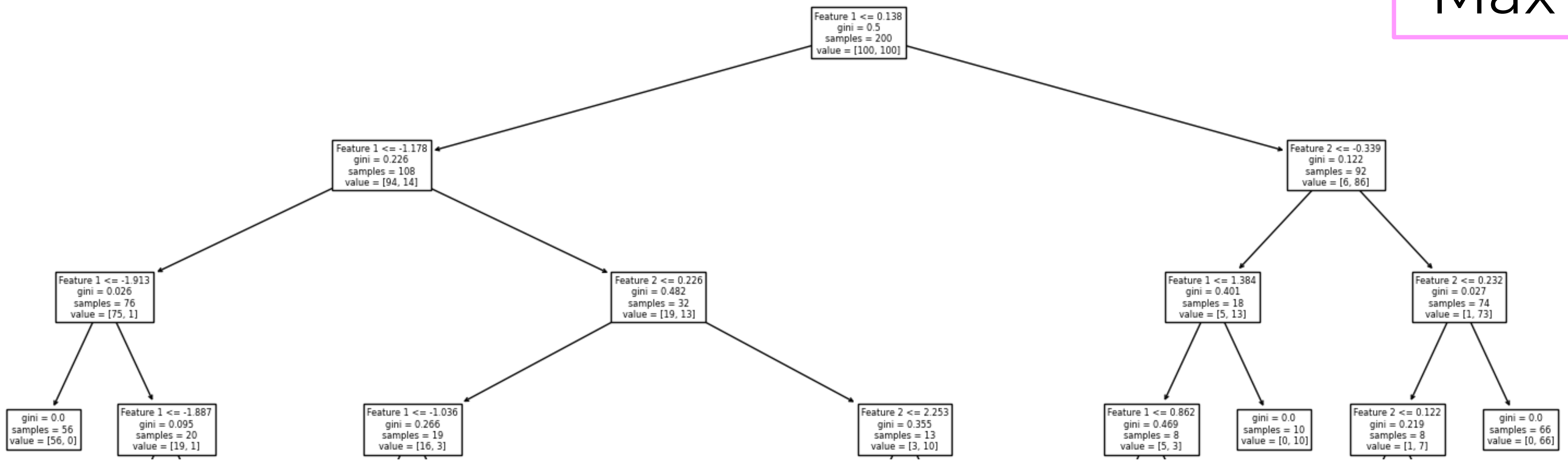


Early Stopping

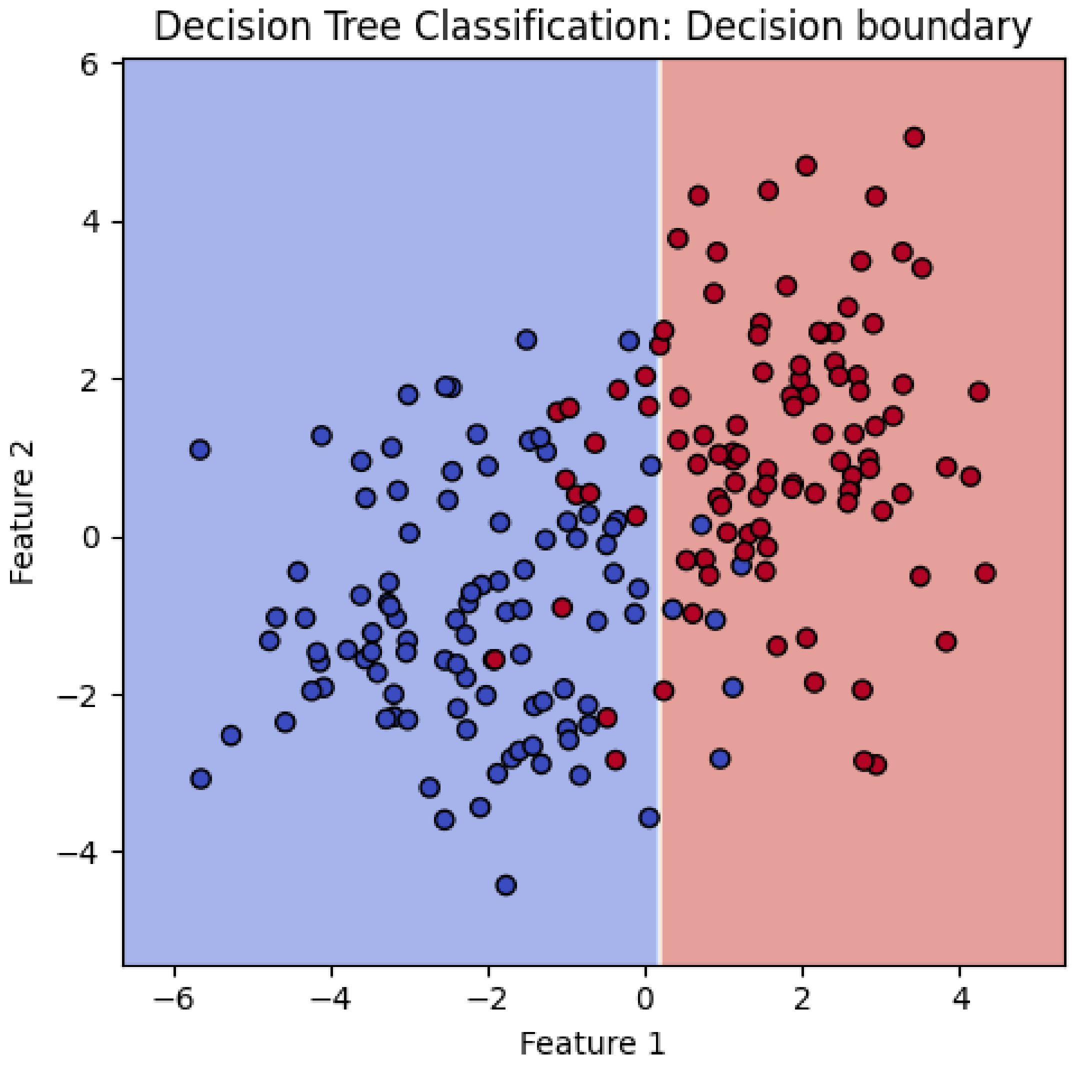


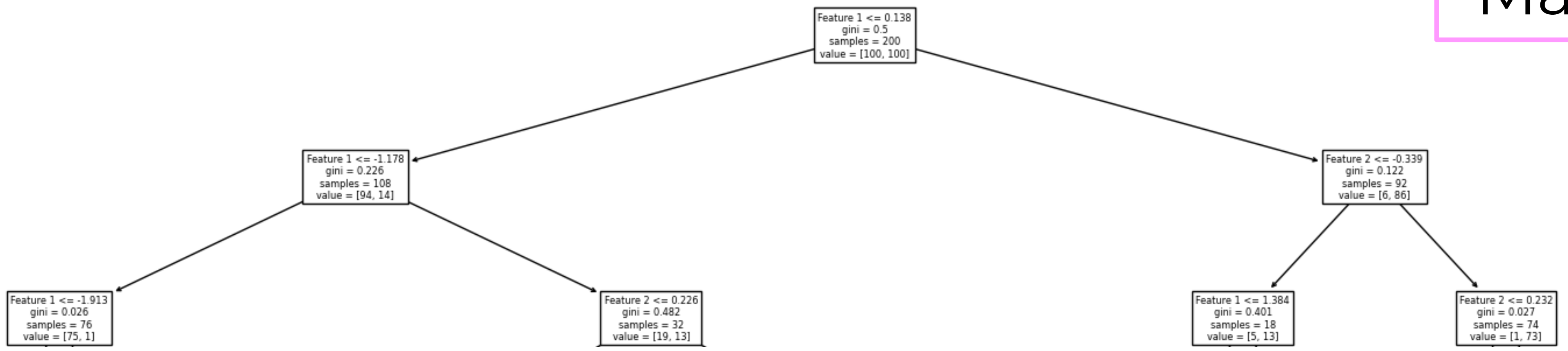
Dataset
Class 0: 100 data
Class 1 : 100 data

Max Depth = 3



Early Stopping





Max Depth = 2

Dataset
Class 0: 100 data
Class 1 : 100 data

Brain Code Camp

Post pruning

The objective of pruning is to balance model performance and model complexity.

$$\mathcal{R}(T) + \alpha|\tilde{T}|$$

Given decision tree T,

$\mathcal{R}(T)$ is the weighted gini impurity $\rightarrow$ $Model\ Performance$,

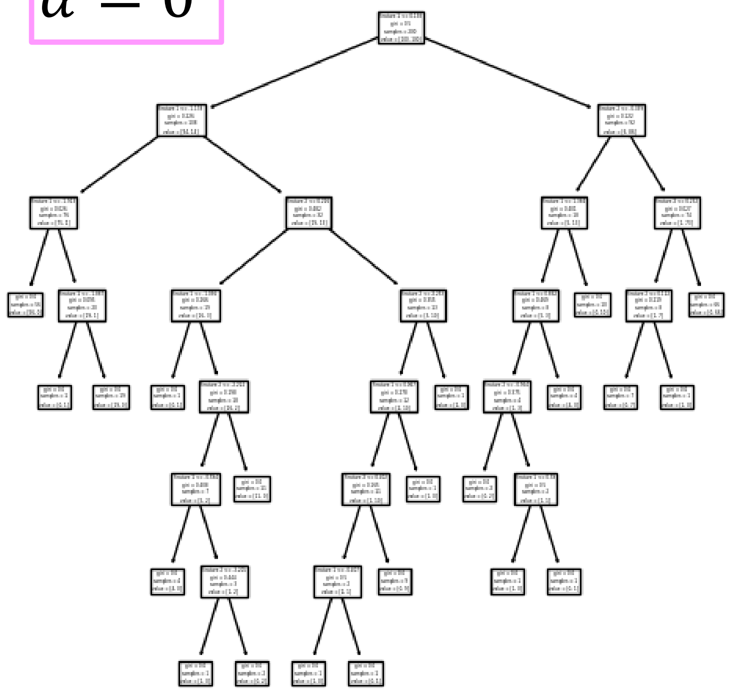
$|\tilde{T}|$ is the number of terminal nodes $\rightarrow$ $Model\ Complexity$,

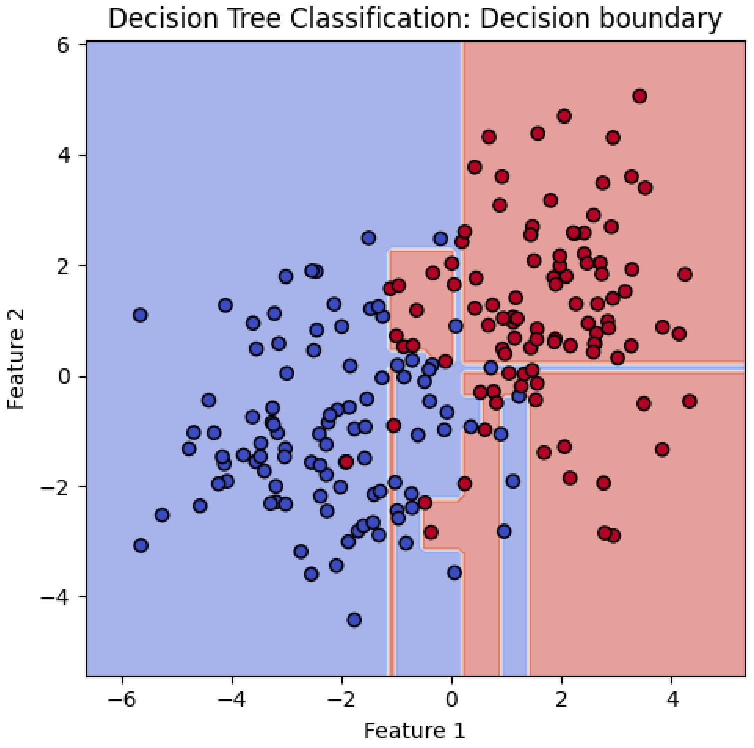
α is the complexity parameter ($\alpha \geq 0$).

- $\alpha = 0$ → No pruning.
- Increase α → activate post-pruning.
- Too high α → Too much pruning. → Over-simplified model→ Underfitting

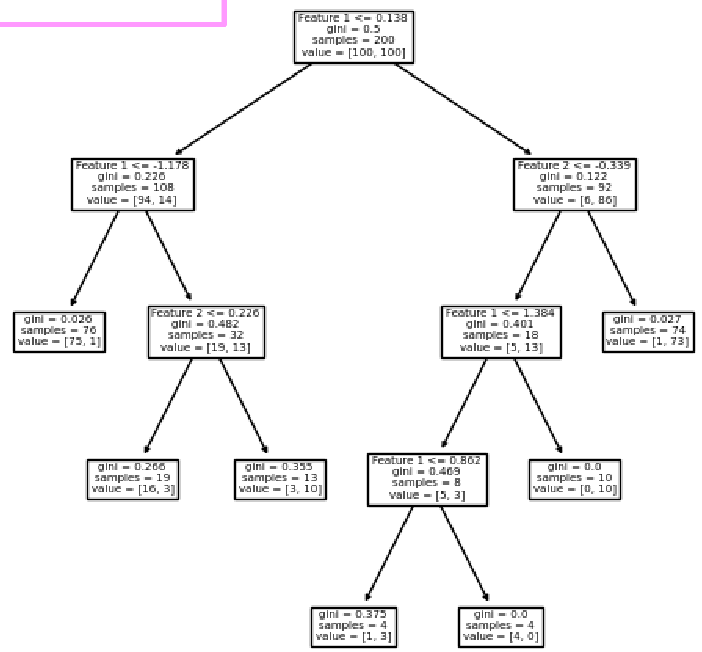
Post pruning

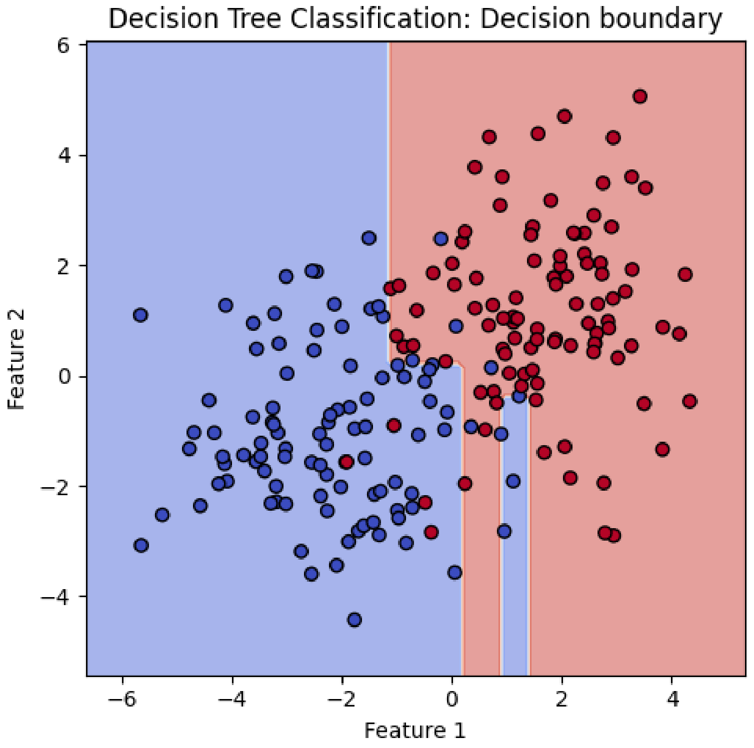
$\alpha = 0$



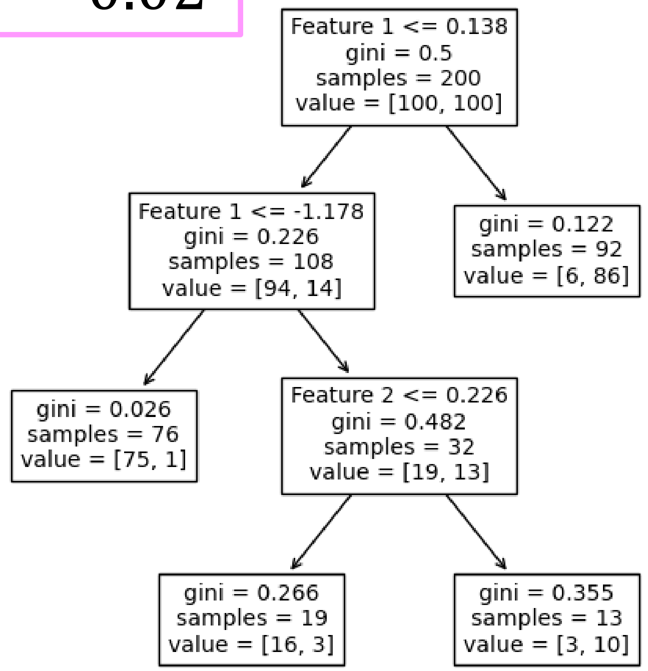


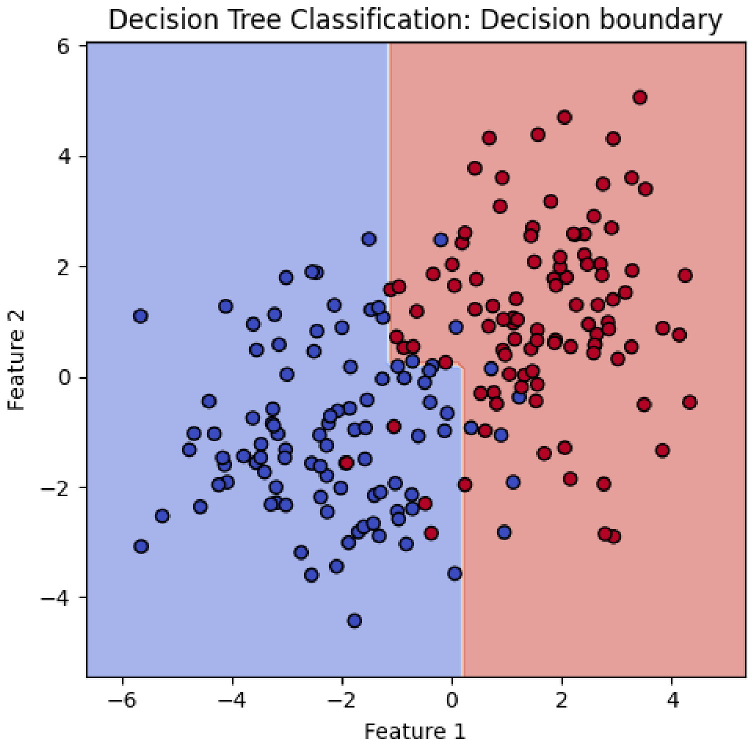
$\alpha = 0.01$



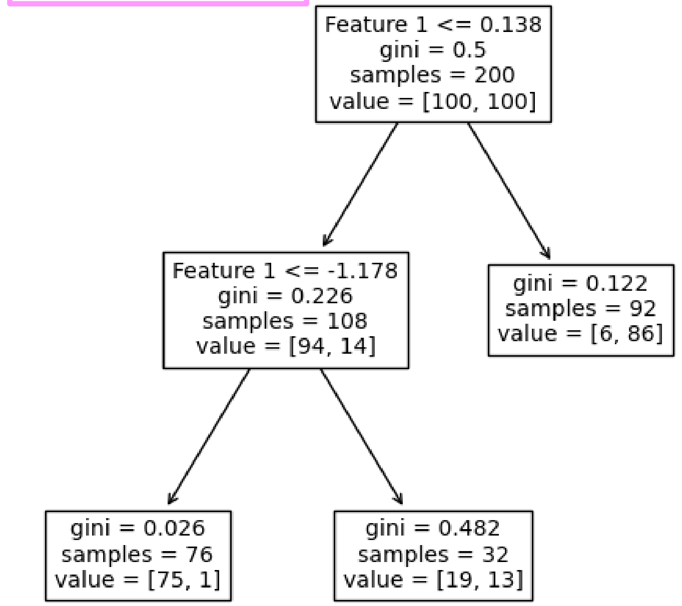


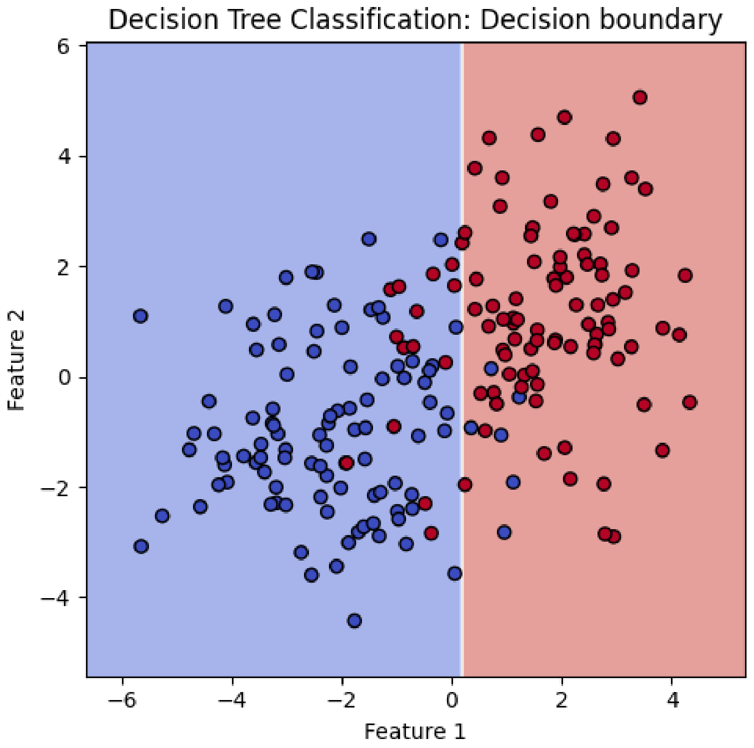
$\alpha = 0.02$





$\alpha = 0.03$





Brain
Code
Camp

sklearn.tree.DecisionTreeClassifier

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, class_weight=None, ccp_alpha=0.0*)

[source]

Parameters:

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.

max_depth : *int, default=None*

The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

min_samples_split : *int or float, default=2*

The minimum number of samples required to split an internal node:

min_impurity_decrease : *float, default=0.0*

A node will be split if this split induces a decrease of the impurity greater than or equal to this value.

ccp_alpha : *non-negative float, default=0.0*

Complexity parameter used for Minimal Cost-Complexity Pruning.

Early Stopping

Post-Pruning

sklearn.tree.DecisionTreeClassifier

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, class_weight=None, ccp_alpha=0.0*)

[source]

Attributes:

classes_ : *ndarray of shape (n_classes,) or list of ndarray*

The classes labels (single output problem), or a list of arrays of class labels (multi-output problem).

feature_importances_ : *ndarray of shape (n_features,)*

Return the feature importances.

Impurity-based Feature Importance
(Gini Importances - mean decrease in Gini impurity)

tree_ : *Tree instance*

The underlying Tree object.

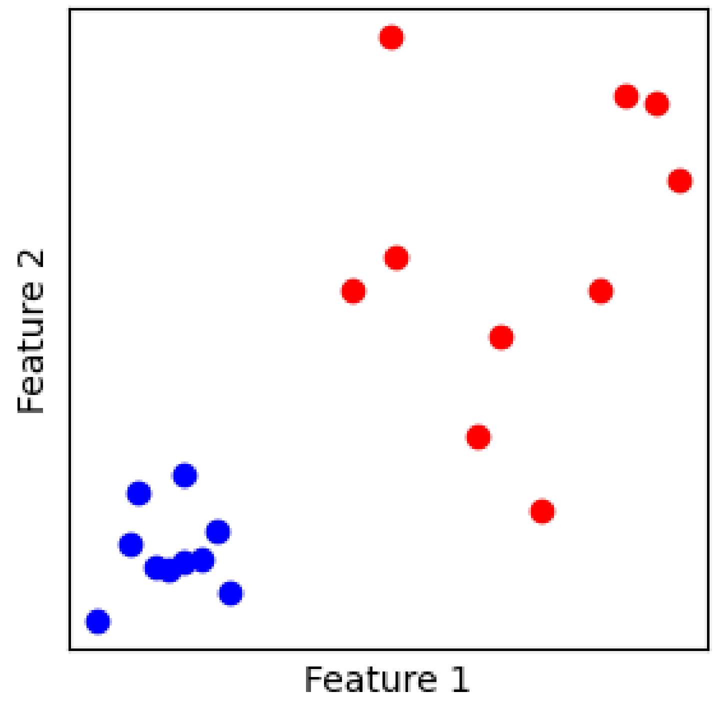
Tree structure

Brain Code Camp

Decision Tree Classifier

Dataset

$$(x_{1,1}, x_{1,2}, y_1), (x_{2,1}, x_{2,2}, y_2), (x_{3,1}, x_{3,2}, y_3), \ldots, (x_{n,1}, x_{n,2}, y_n)$$



$$\mathbf{X} = \begin{bmatrix} x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \\ x_{3,1} & x_{3,2} \\ \vdots & \vdots \\ x_{n,1} & x_{n,2} \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix}$$

$\text{shape} = (n, 2)$ $\qquad$ $\text{shape} = (n, 1)$

```python
# Import a necessary modules
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
import matplotlib.pyplot as plt

# Create the model
clf = DecisionTreeClassifier()

# Train the model
clf.fit(X,y)

# Make prediction
y_pred = clf.predict(X_test)

# Visualize tree
tree.plot_tree(clf, feature_names=['Feature 1','Feature 2'])
plt.tight_layout()
plt.show()

# Obtain feature importance
importances = clf.feature_importances_
```

Brain
Code
Camp

Random Forest

Kanokkorn Pimcharoen

Overfitting (recall)

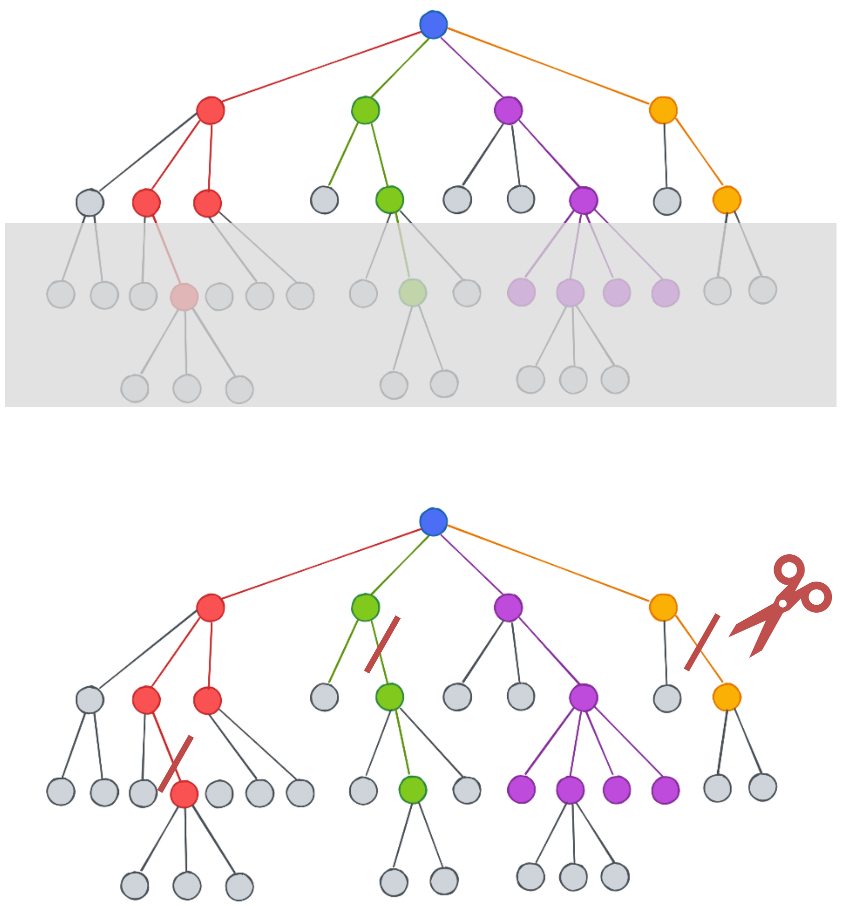
How to avoid overfitting in decision tree?

Early Stopping: stop growing before a tree becomes too complex.

- Limit tree depth.
- Do not split nodes which contain too few data points.
- Do not split nodes which do not significantly decrease impurity.

Post Pruning: grow full tree (overfitting), then later simplify the tree.

Ensemble Methods: Use multiple trees to obtain better predictive performance. Wisdom of the crowds.



Random Forest

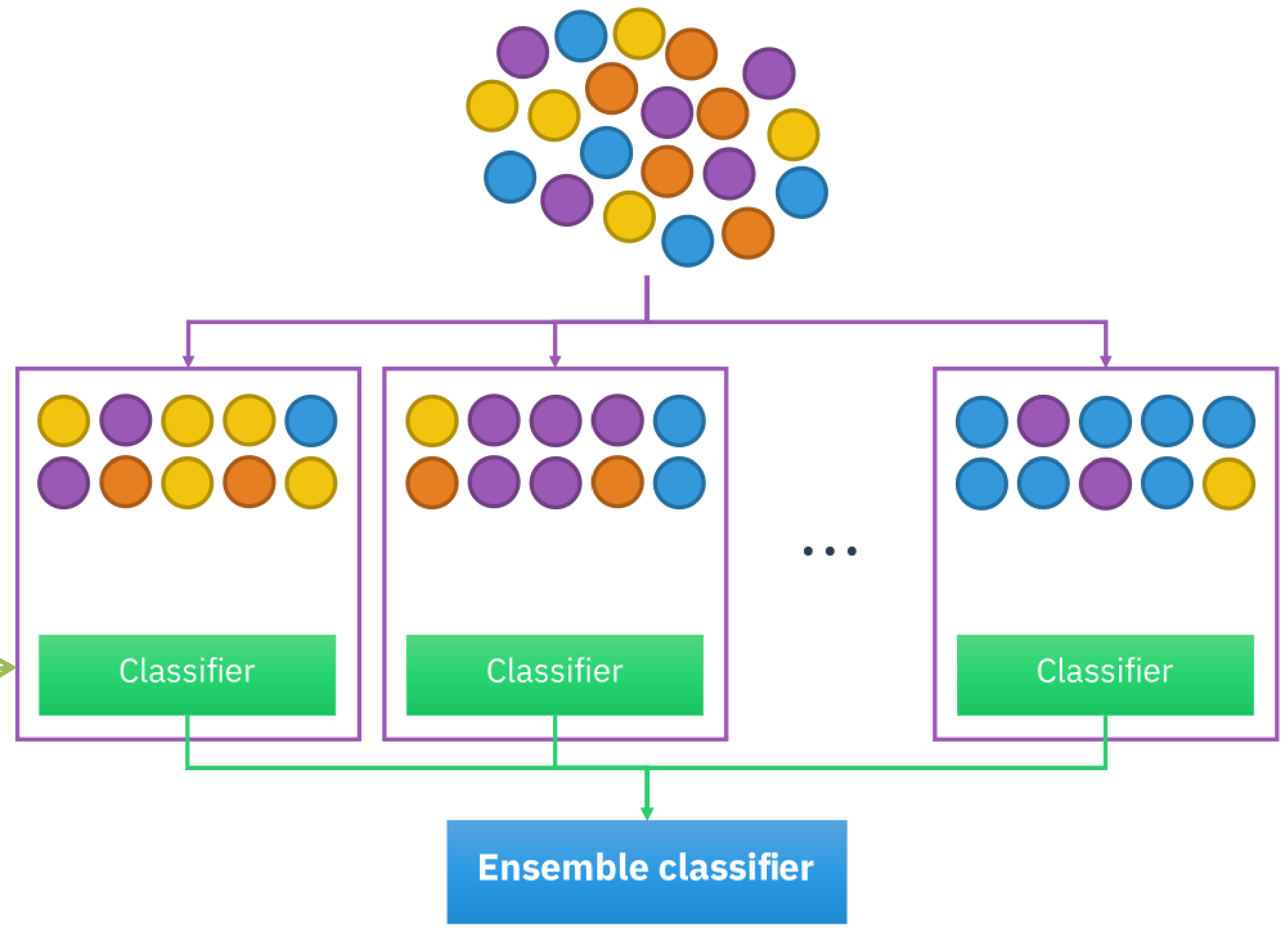
Bootstrap aggregating / bagging

Data sample randomly drawn from a training set with replacement.

Adding more diversity to the dataset.

Decorrelate the trees by randomly select subset of features.

Adding more diversity to the models.



Original Data

Bootstrapping

Aggregating
Let all decision trees vote for the result.

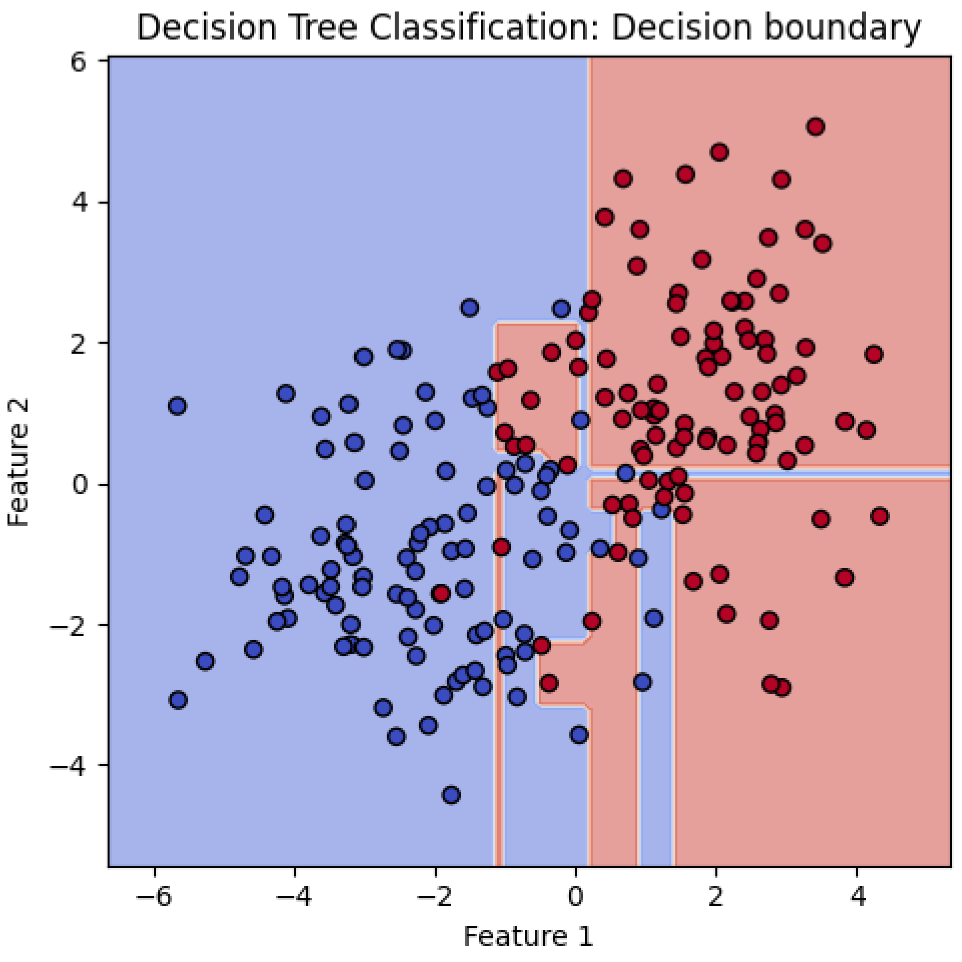
Bagging

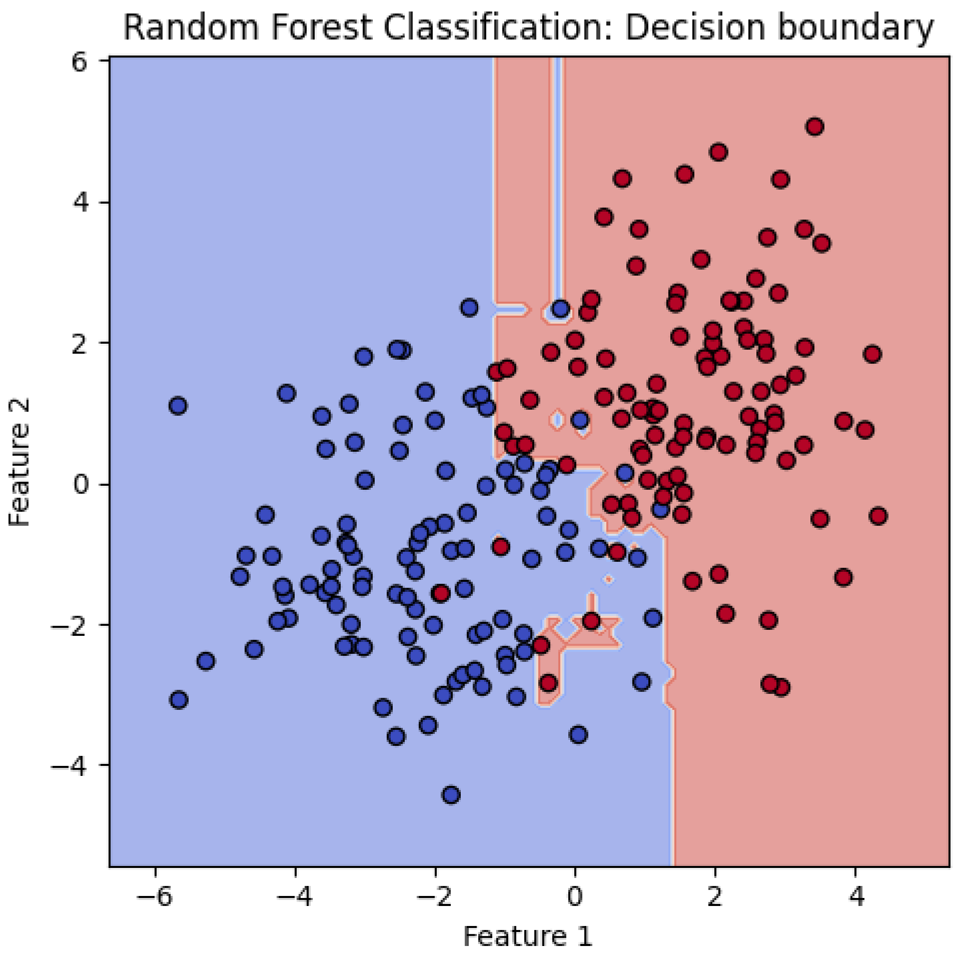
Wikipedia: https://en.wikipedia.org/wiki/Bootstrap_aggregating

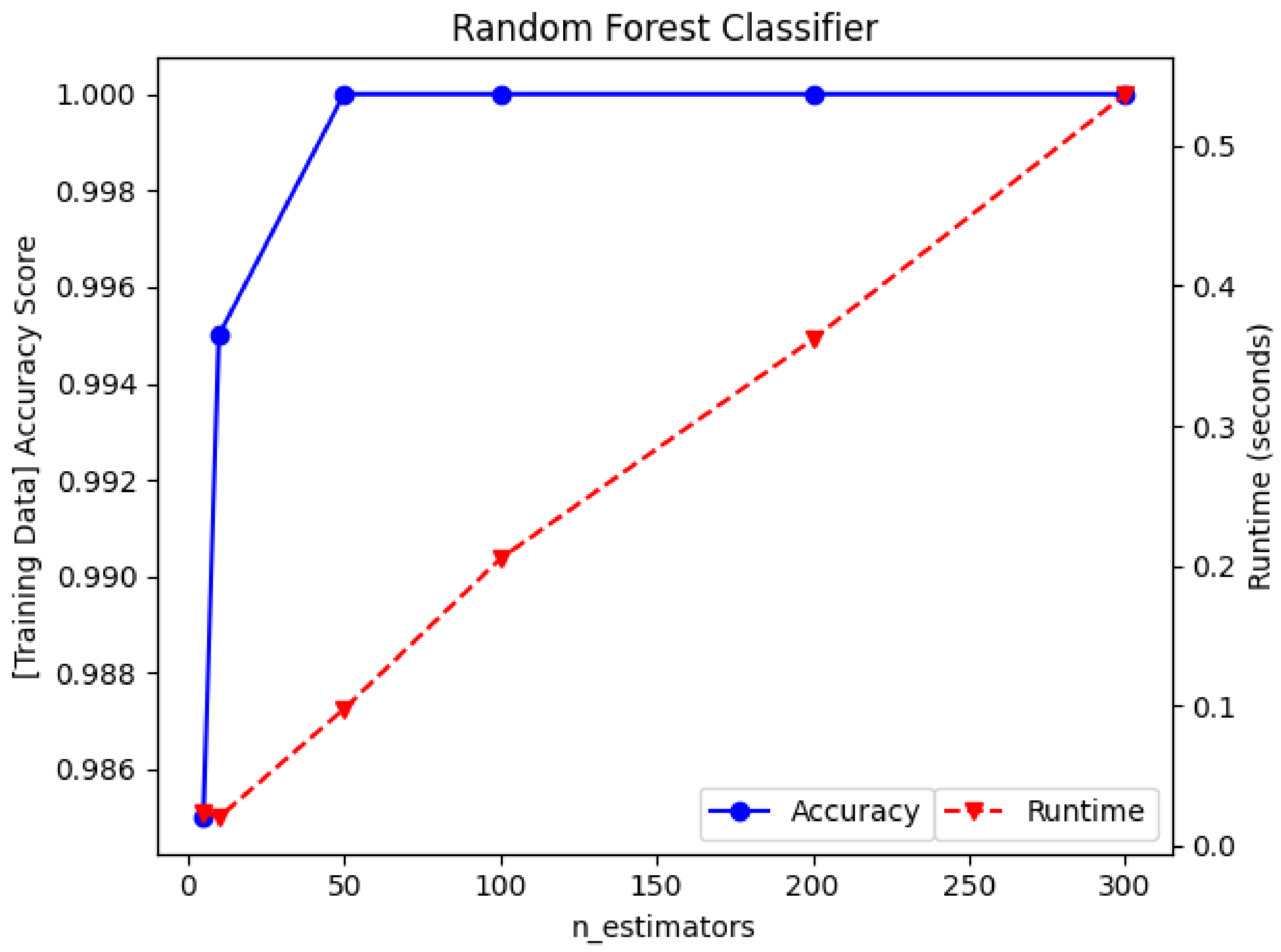
Random Forest

1 Tree

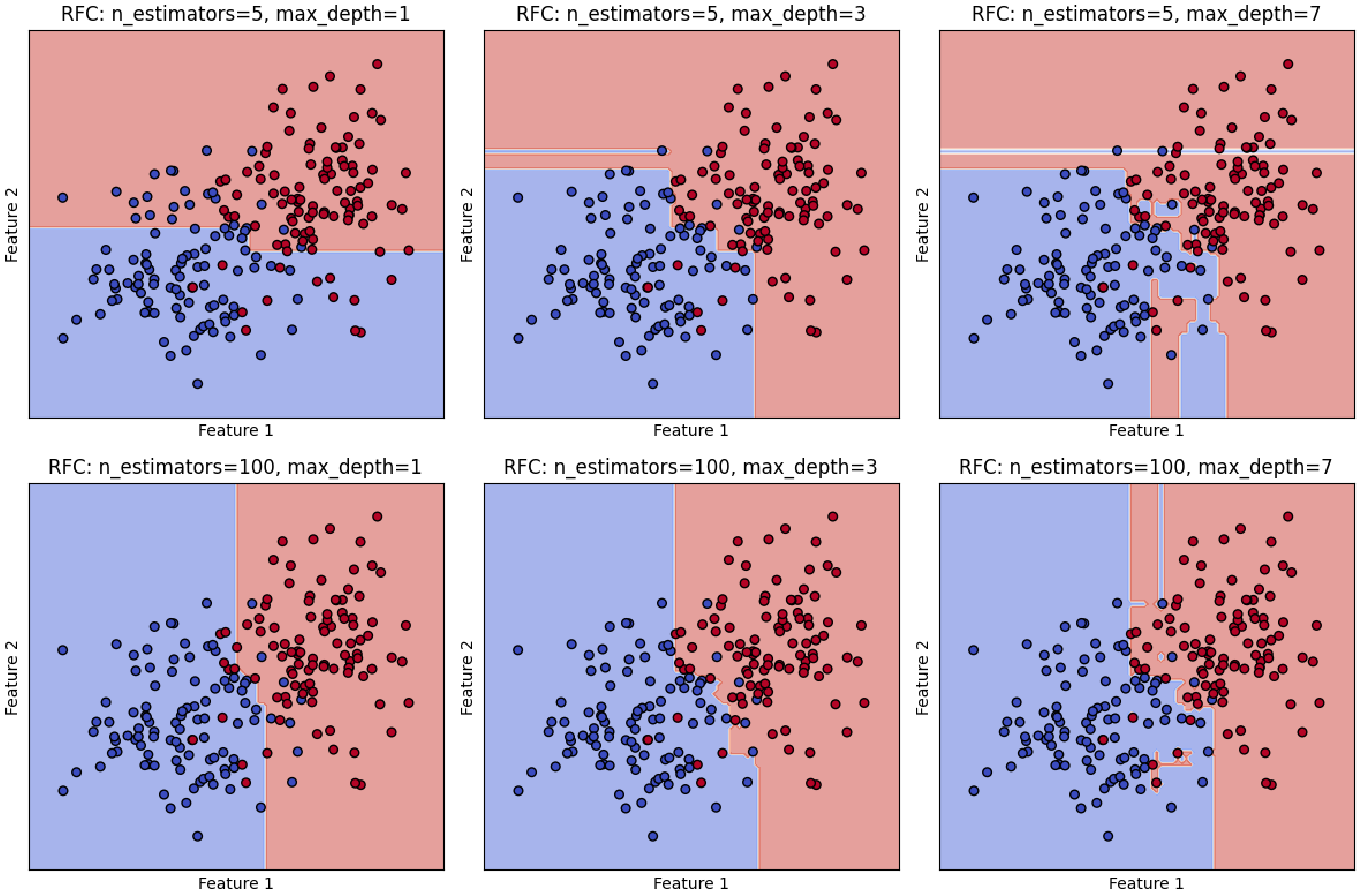
100 Trees



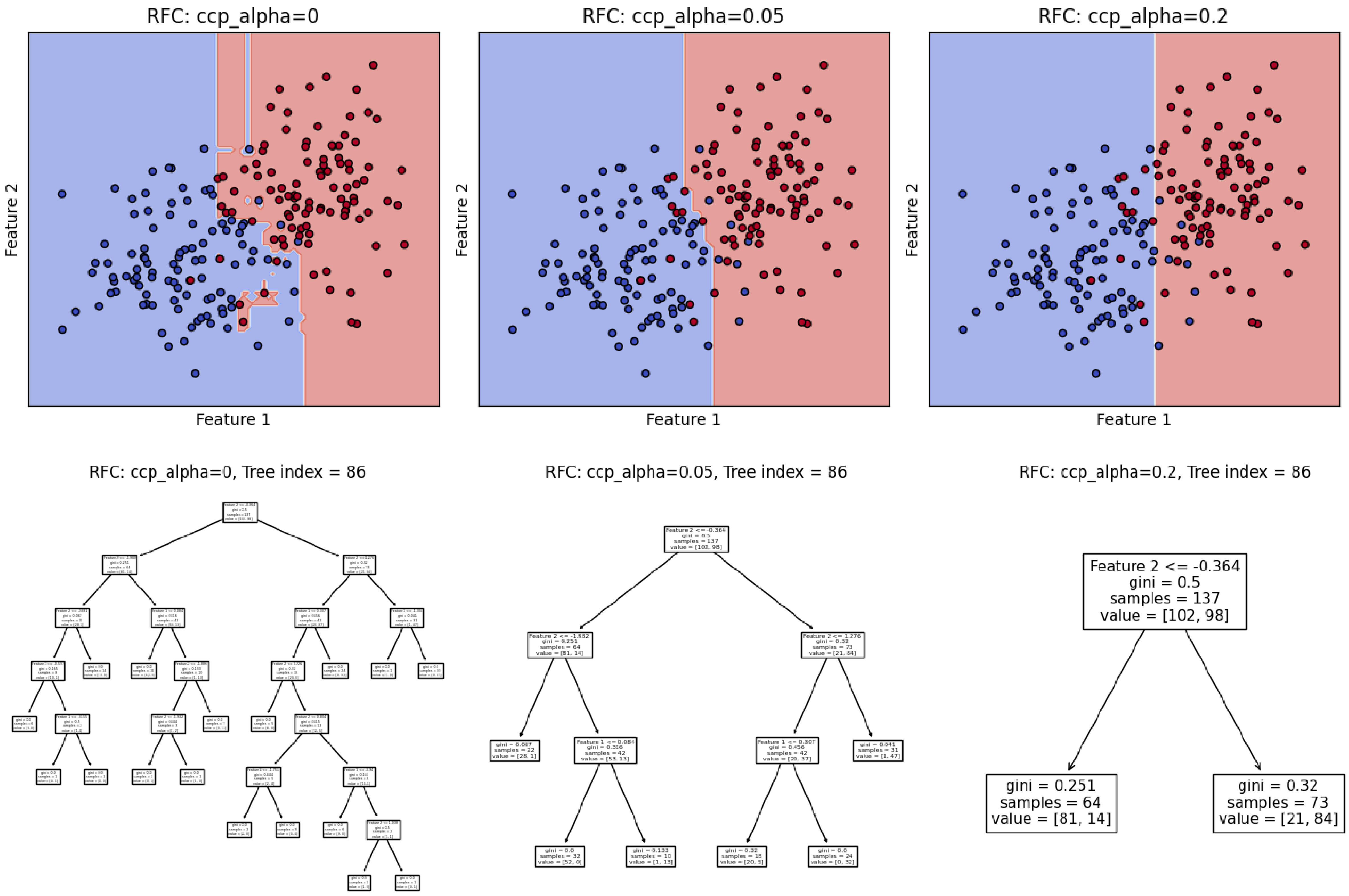




Random Forest + Early Stopping



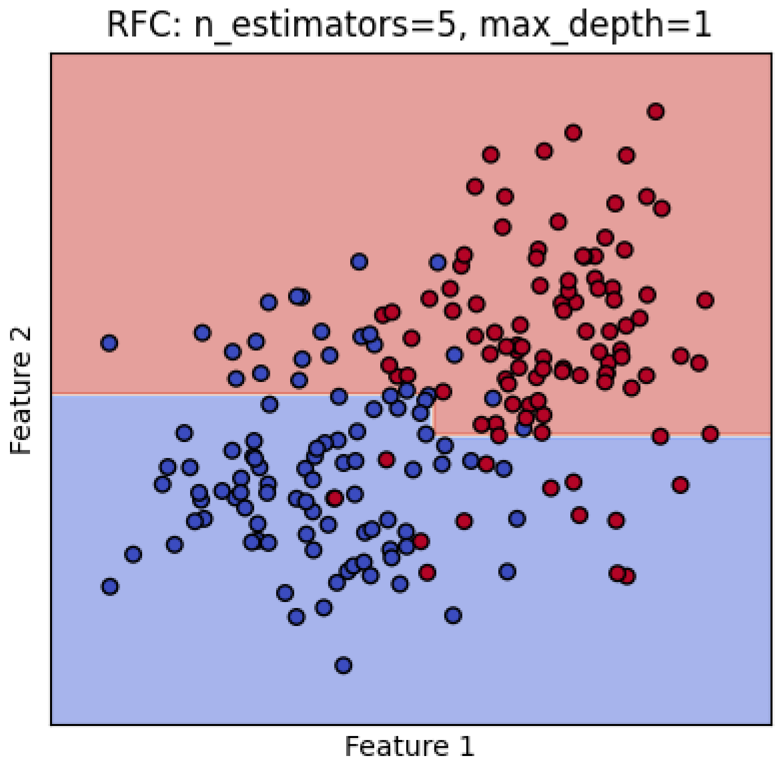
Random Forest + Post Pruning

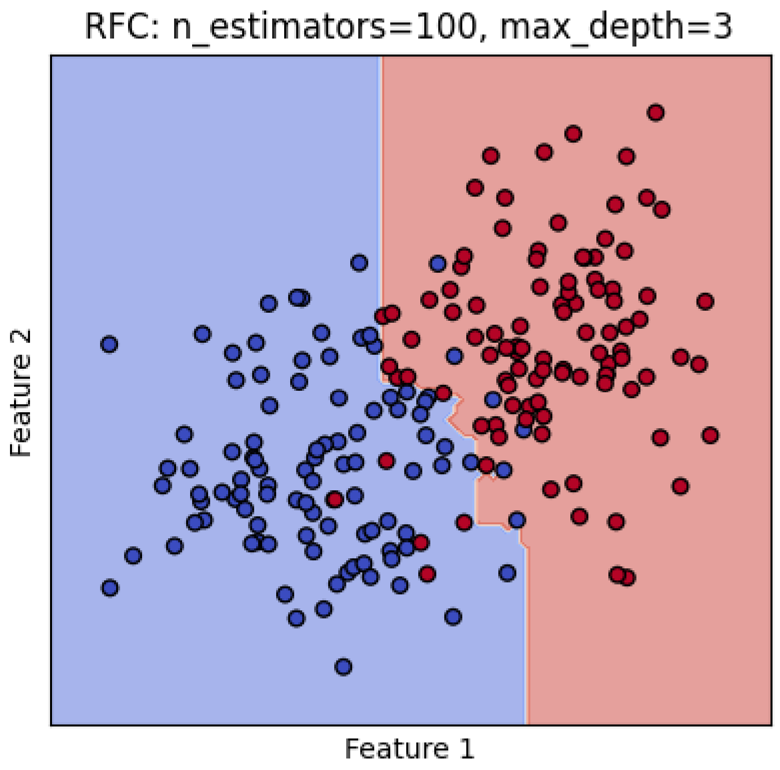


Brain
Code
Camp

Random Forest

Impurity-based Feature Importances (Gini Importances)






```
Relative Feature Importances:
Feature 1: 0.20
Feature 2: 0.80
```

```
Relative Feature Importances:
Feature 1: 0.68
Feature 2: 0.32
```

This is for the purpose of demonstrate the correlation between decision boundary and feature importance.

sklearn.ensemble.RandomForestClassifier

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='sqrt', max_leaf_nodes=None, min_impurity_decrease=0.0, 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*)

[source]

Parameters:

n_estimators : *int, default=100*

The number of trees in the forest.

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.

bootstrap : *bool, default=True*

Whether bootstrap samples are used when building trees. If False, the whole dataset is used to build each tree.

max_samples : *int or float, default=None*

If bootstrap is True, the number of samples to draw from X to train each base estimator.

- If None (default), then draw `X.shape[0]` samples.
- If int, then draw `max_samples` samples.
- If float, then draw `max(round(n_samples * max_samples), 1)` samples. Thus, `max_samples` should be in the interval `(0.0, 1.0]`.

Bootstrap sampling

sklearn.ensemble.RandomForestClassifier

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='sqrt', max_leaf_nodes=None, min_impurity_decrease=0.0, 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*)

[source]

Parameters:

max_features : *{"sqrt", "log2", None}, int or float, default="sqrt"*

The number of features to consider when looking for the best split:

- If int, then consider `max_features` features at each split.
- If float, then `max_features` is a fraction and `max(1, int(max_features * n_features_in_))` features are considered at each split.
- If "sqrt", then `max_features=sqrt(n_features)`.
- If "log2", then `max_features=log2(n_features)`.
- If None, then `max_features=n_features`.

Randomly select subset of features for trees to look for the best split.

sklearn.ensemble.RandomForestClassifier

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='sqrt', max_leaf_nodes=None, min_impurity_decrease=0.0, 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*)

[source]

Parameters:

max_depth : *int, default=None*

The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

min_samples_split : *int or float, default=2*

The minimum number of samples required to split an internal node:

min_impurity_decrease : *float, default=0.0*

A node will be split if this split induces a decrease of the impurity greater than or equal to this value.

Early Stopping

ccp_alpha : *non-negative float, default=0.0*

Complexity parameter used for Minimal Cost-Complexity Pruning.

Post-Pruning

sklearn.ensemble.RandomForestClassifier

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='sqrt', max_leaf_nodes=None, min_impurity_decrease=0.0, 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*)

[source]

Attributes:

estimator_ : `DecisionTreeClassifier`

The child estimator template used to create the collection of fitted sub-estimators.

estimators_ : ***list of DecisionTreeClassifier***

The collection of fitted sub-estimators.

`feature_importances_` **:** ***ndarray of shape (n_features,)***

The impurity-based feature importances.

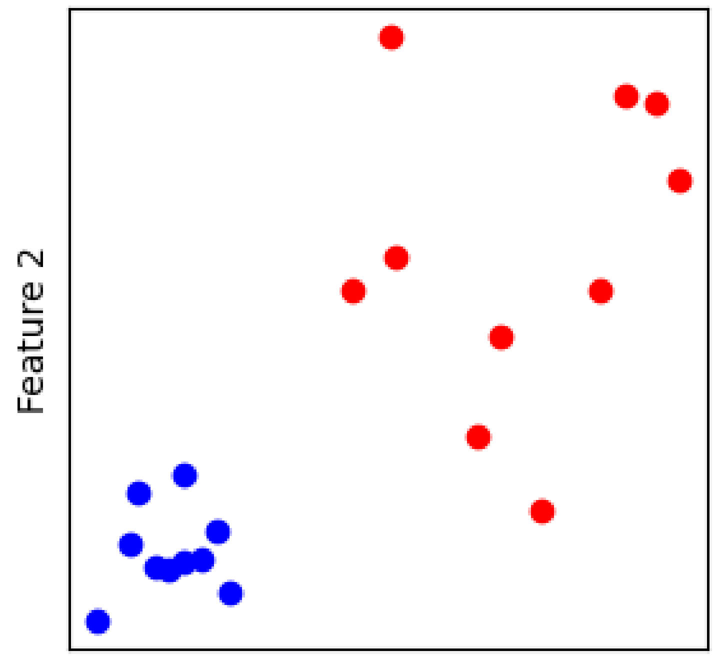
Impurity-based Feature Importance
(Gini Importance)

Brain Code Camp

Random Forest Classifier

Dataset

$$(x_{1,1}, x_{1,2}, y_1), (x_{2,1}, x_{2,2}, y_2), (x_{3,1}, x_{3,2}, y_3),..., (x_{n,1}, x_{n,2}, y_n)$$



$$\mathbf{x} = \begin{bmatrix} x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \\ x_{3,1} & x_{3,2} \\ \vdots & \vdots \\ x_{n,1} & x_{n,2} \end{bmatrix} \quad \text{shape} = (n, 2)$$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} \quad \text{shape} = (n, 1)$$

```
# Import a necessary modules
from sklearn.tree import RandomForestClassifier
from sklearn import tree
import matplotlib.pyplot as plt

# Create the model
clf = RandomForestClassifier()

# Train the model
clf.fit(X,y)

# Make prediction
y_pred = clf.predict(x_test)

# Visualize tree
i = 0  # Tree index
tree.plot_tree(clf.estimators_[i],
               feature_names=['Feature 1','Feature 2'])
plt.tight_layout()
plt.show()

# Obtain feature importance
importances = clf.feature_importances_
```