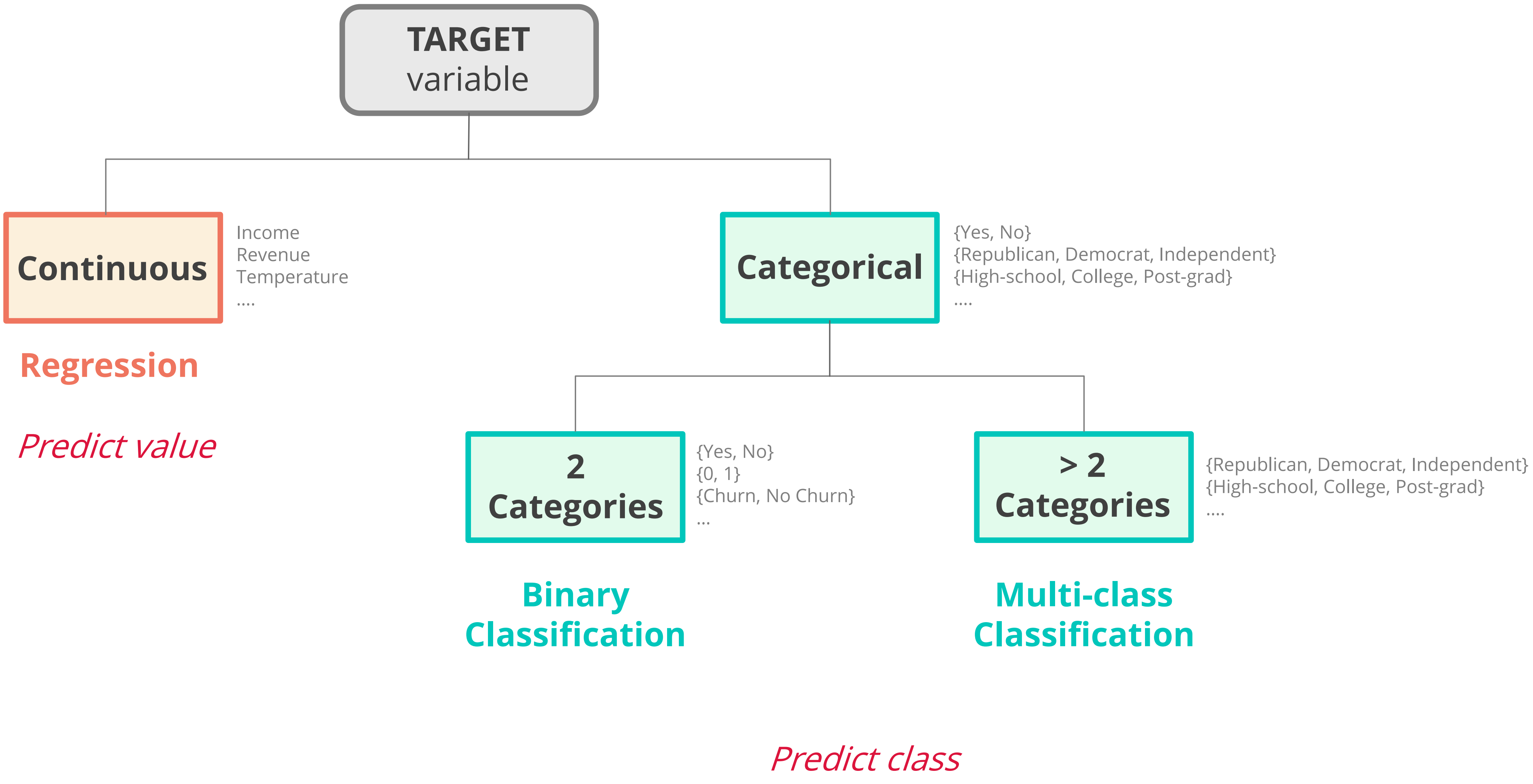
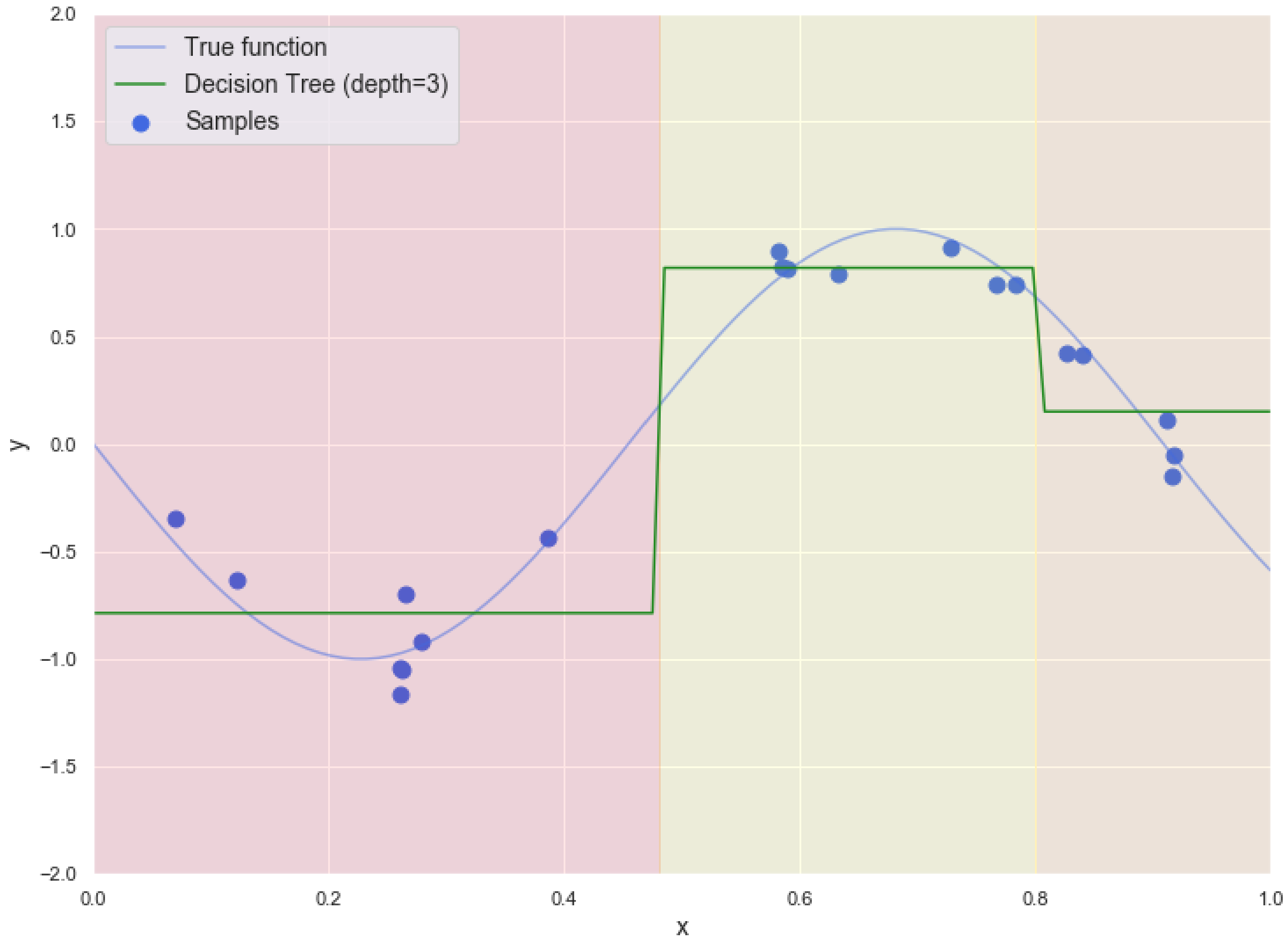
Supervised Learning: Classification

Spring 2022

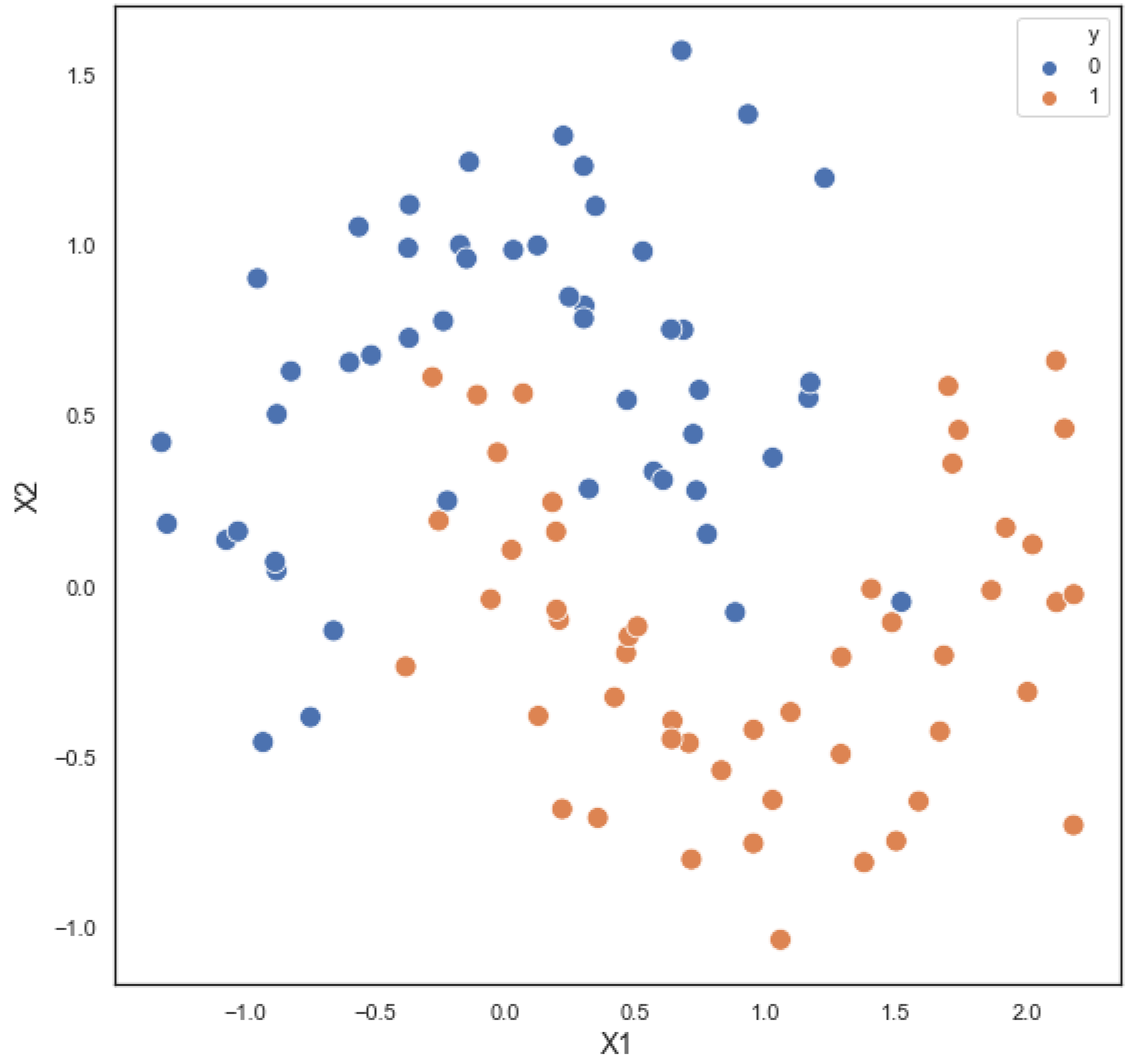
1. Introduction
2. The Data Science Process
3. **Supervised Learning: Classification**
4. Unsupervised Learning
5. The Grunt Work
6. Wrap Up


TARGET
variable
Continuous
Income
Revenue
Temperature
....
Regression
Predict value
Categorical
{Yes, No}
{Republican, Democrat, Independent}
{High-school, College, Post-grad}
....
2
Categories
{Yes, No}
{0, 1}
{Churn, No Churn}
...
Binary
Classification
> 2
Categories
{Republican, Democrat, Independent}
{High-school, College, Post-grad}
....
Multi-class
Classification
Predict class

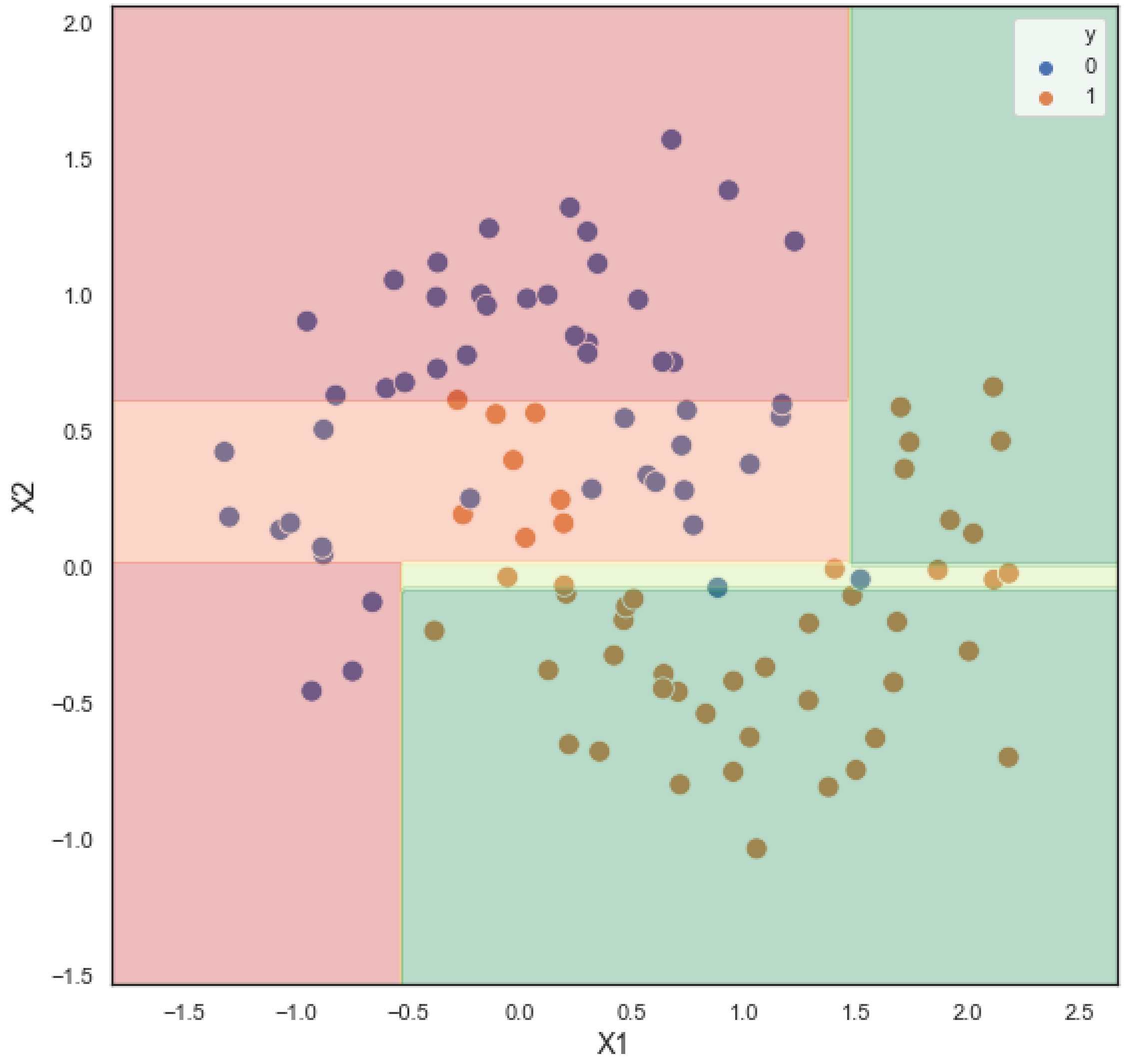
Classification Trees



Recursive partitioning on continuous data (outcome)



Binary outcome (y)



Recursive partitioning

1. How to partition the data?
2. When to stop?

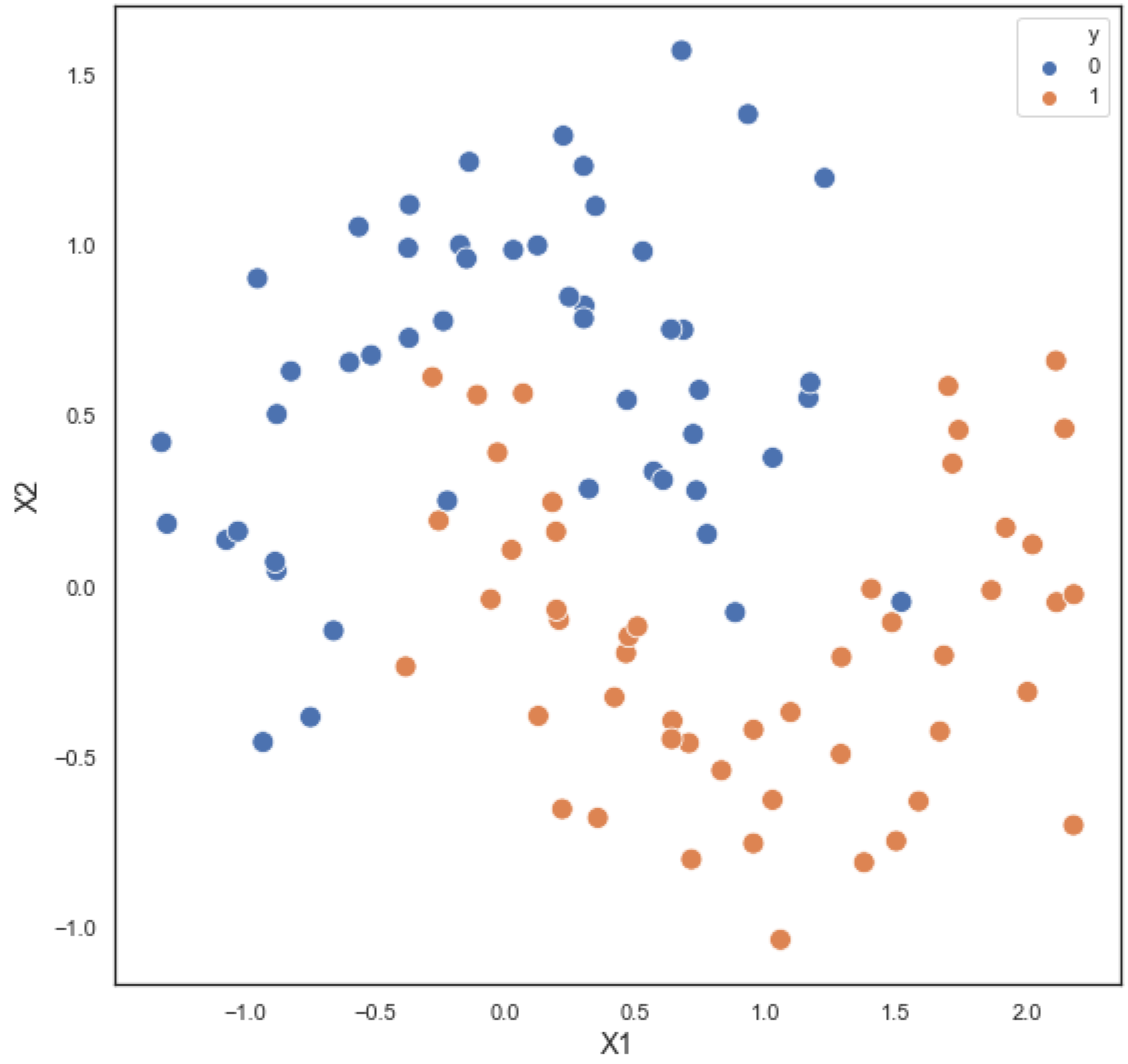
REGRESSION TREES

$$MSE = \frac{1}{n} \sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

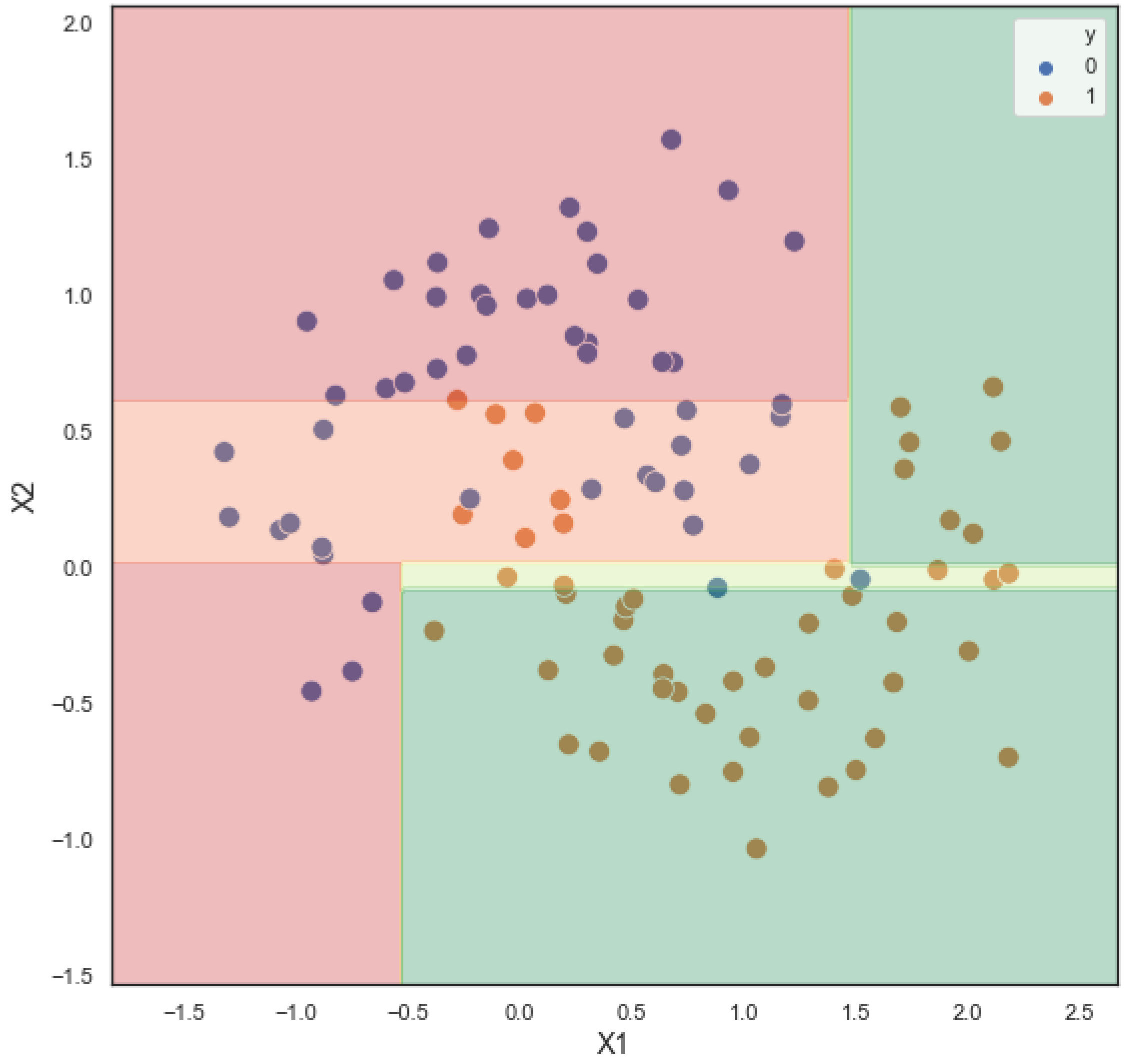
CLASSIFICATION TREES

?

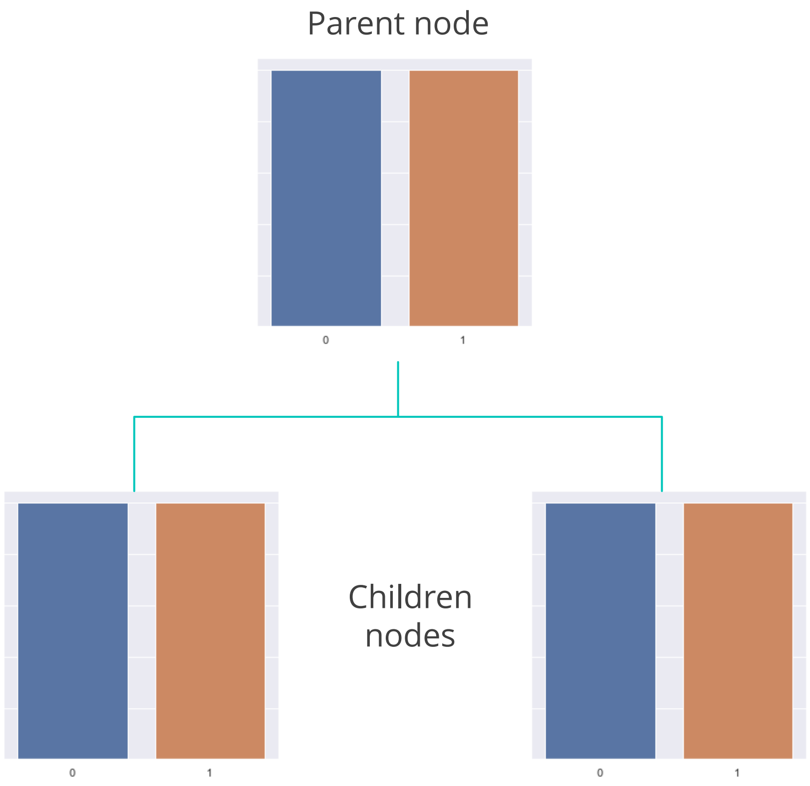
Measure of Impurity



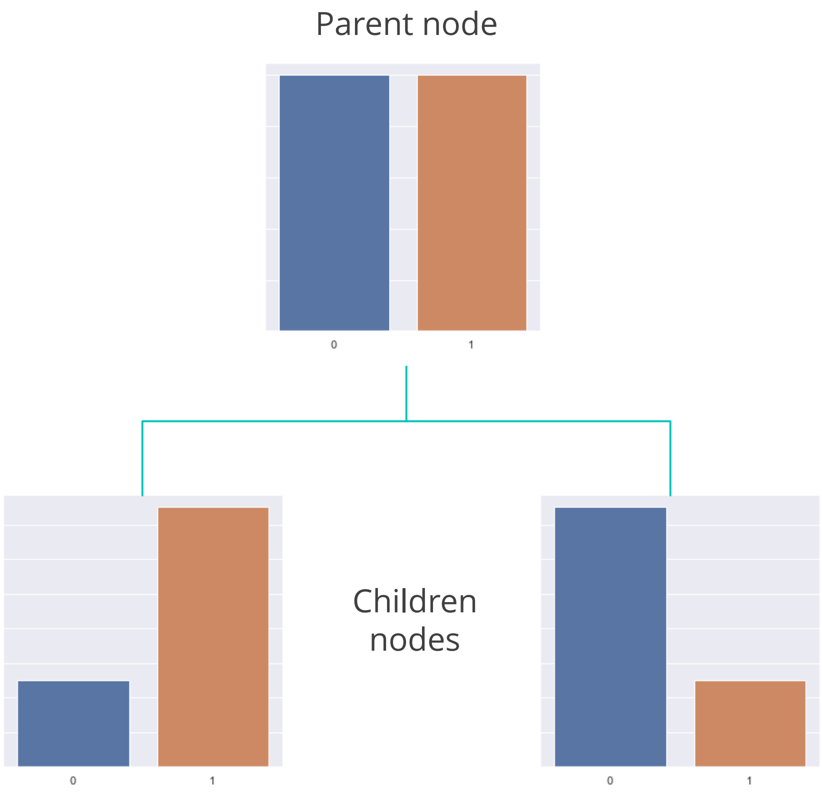
Binary outcome (y)



Recursive partitioning

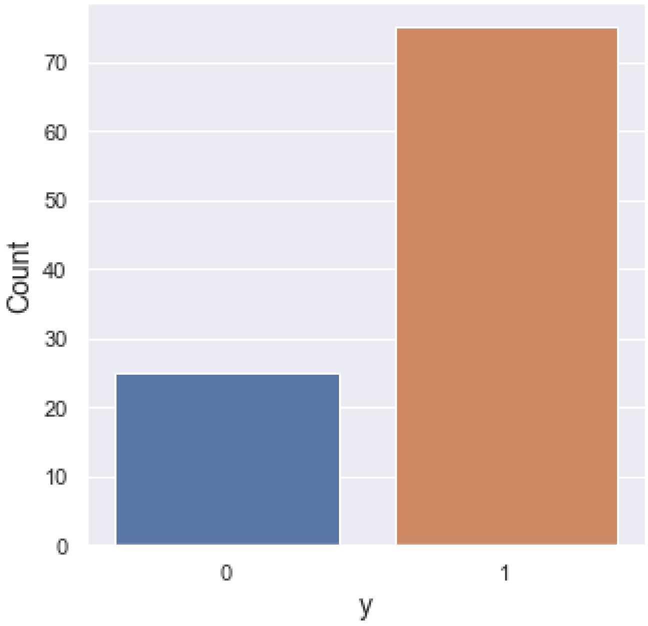


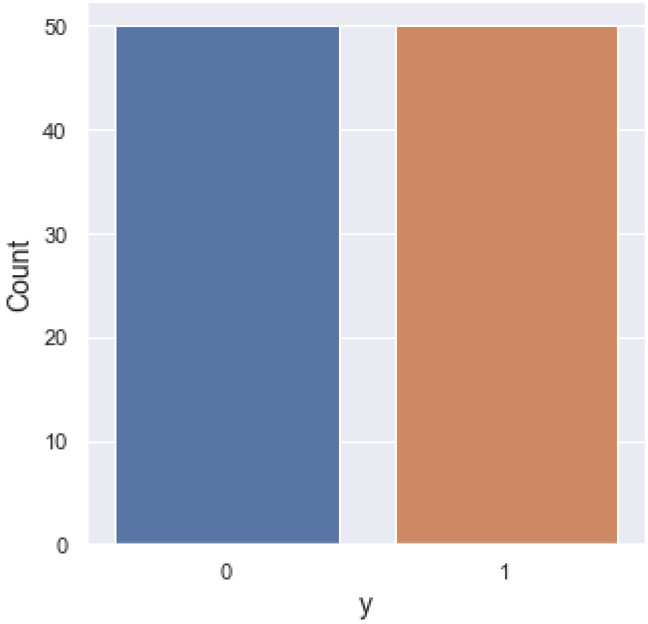
Split #1

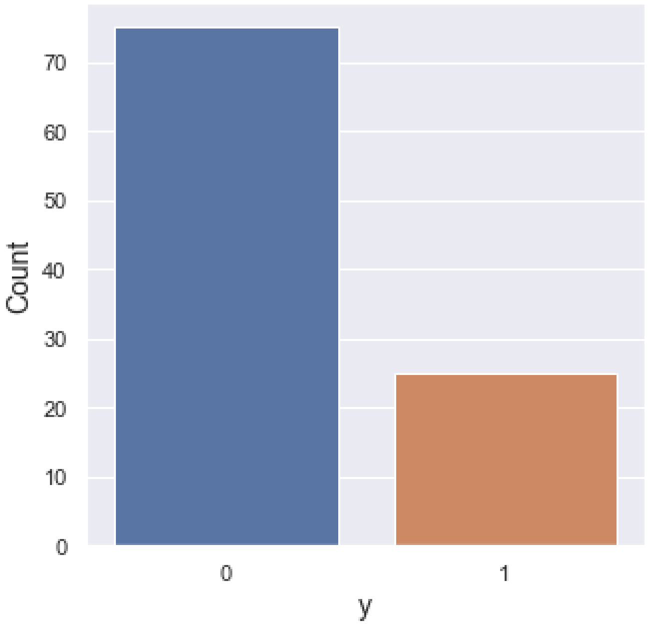


Split #2

How to determine which split is better?



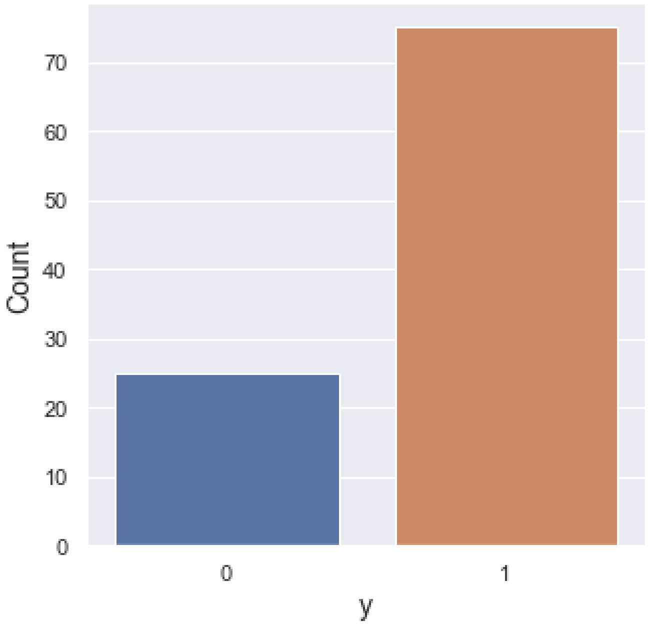


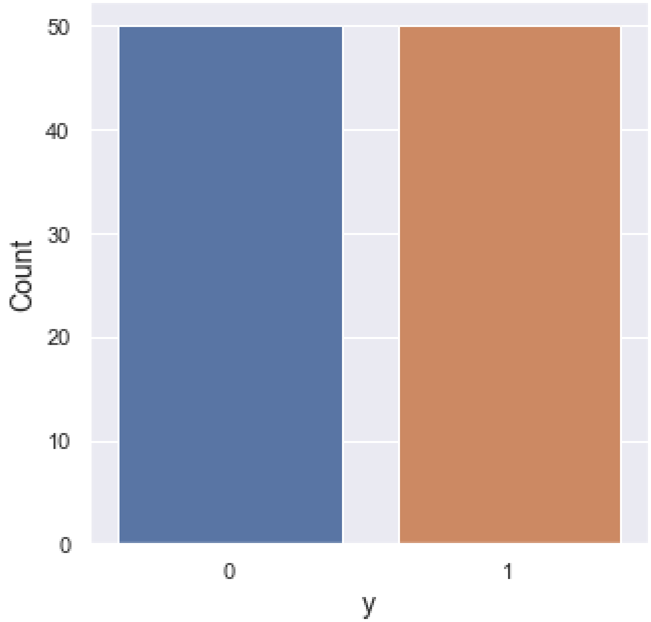


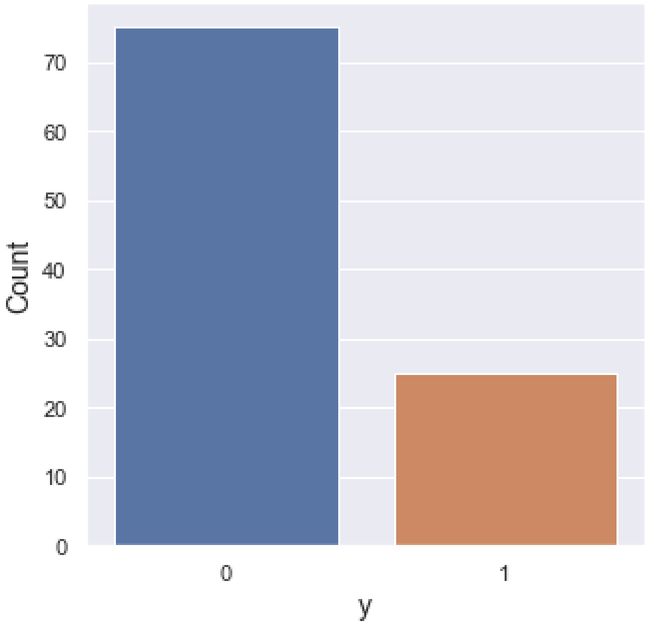
$$p_{y=0} = \frac{25}{100} = \mathbf{0.25} \qquad p_{y=0} = \frac{50}{100} = \mathbf{0.50} \qquad p_{y=0} = \frac{75}{100} = \mathbf{0.75}$$

$$\boxed{p_{y=1} = \frac{75}{100} = \mathbf{0.75} \qquad p_{y=1} = \frac{50}{100} = \mathbf{0.50} \qquad p_{y=1} = \frac{25}{100} = \mathbf{0.25}}$$

$p_{y=1}$ is not symmetrical.







$$p_{y=0} = \frac{25}{100} = \mathbf{0.25}$$

$$p_{y=1} = \frac{75}{100} = \mathbf{0.75}$$

$$p_{y=0} = \frac{50}{100} = \mathbf{0.50}$$

$$p_{y=1} = \frac{50}{100} = \mathbf{0.50}$$

$$p_{y=0} = \frac{75}{100} = \mathbf{0.75}$$

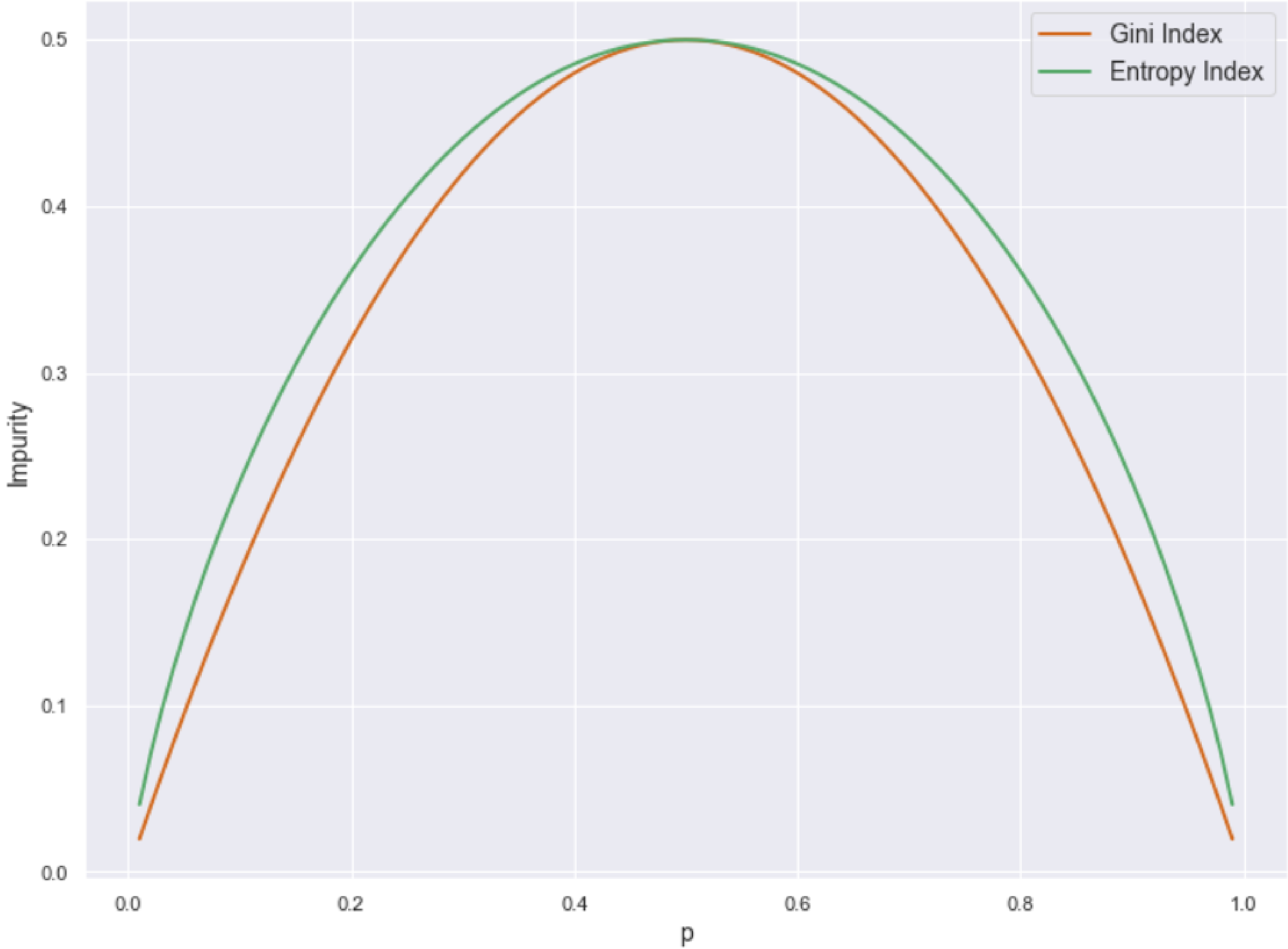
$$p_{y=1} = \frac{25}{100} = \mathbf{0.25}$$

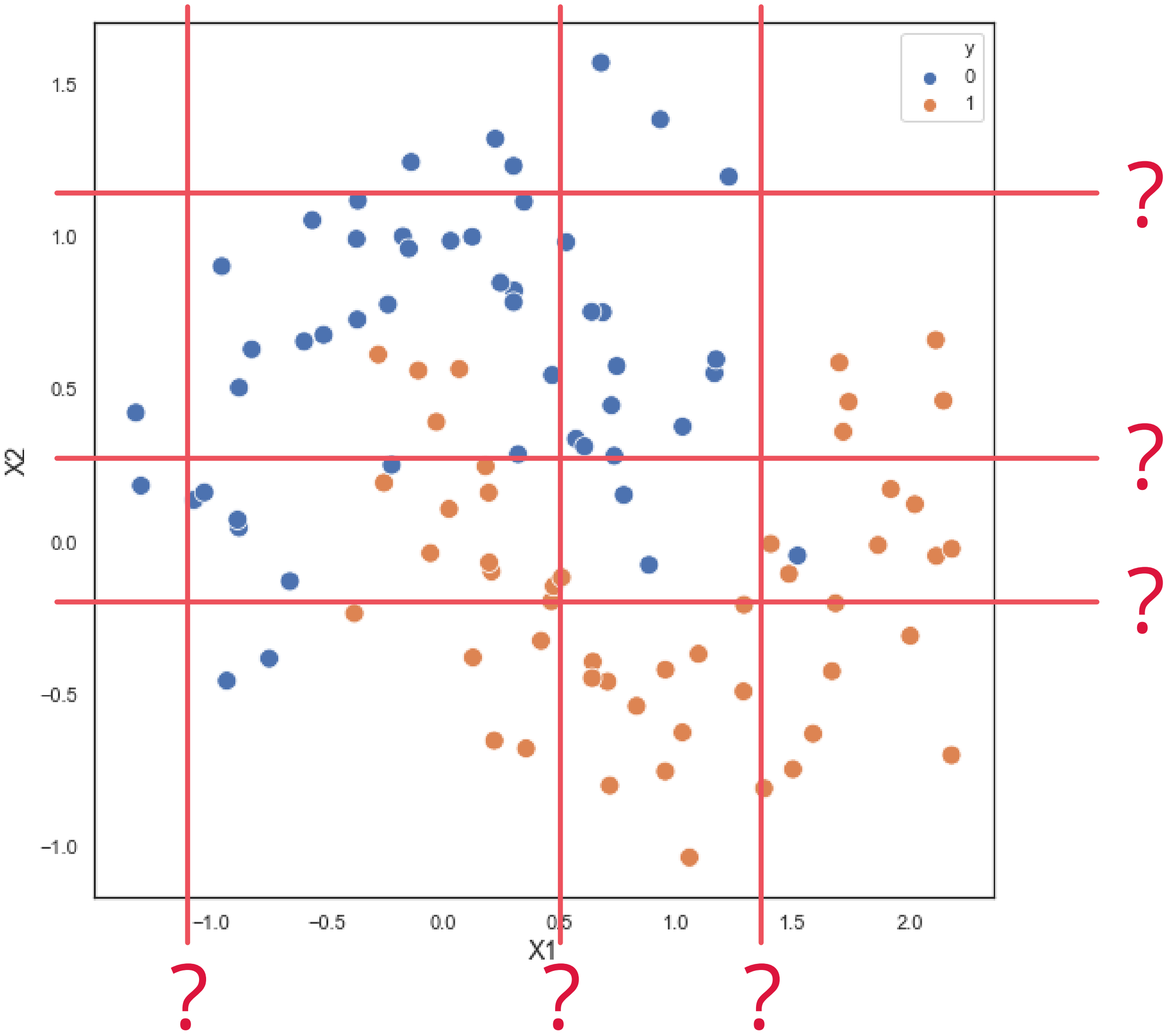
0.375	**0.50**	**0.375**

$$p_{y=1} * \left(1 - p_{y=1}\right) + p_{y=0} * \left(1 - p_{y=0}\right)$$

$$Entropy = -\sum_k p_k \log_2 p_k$$

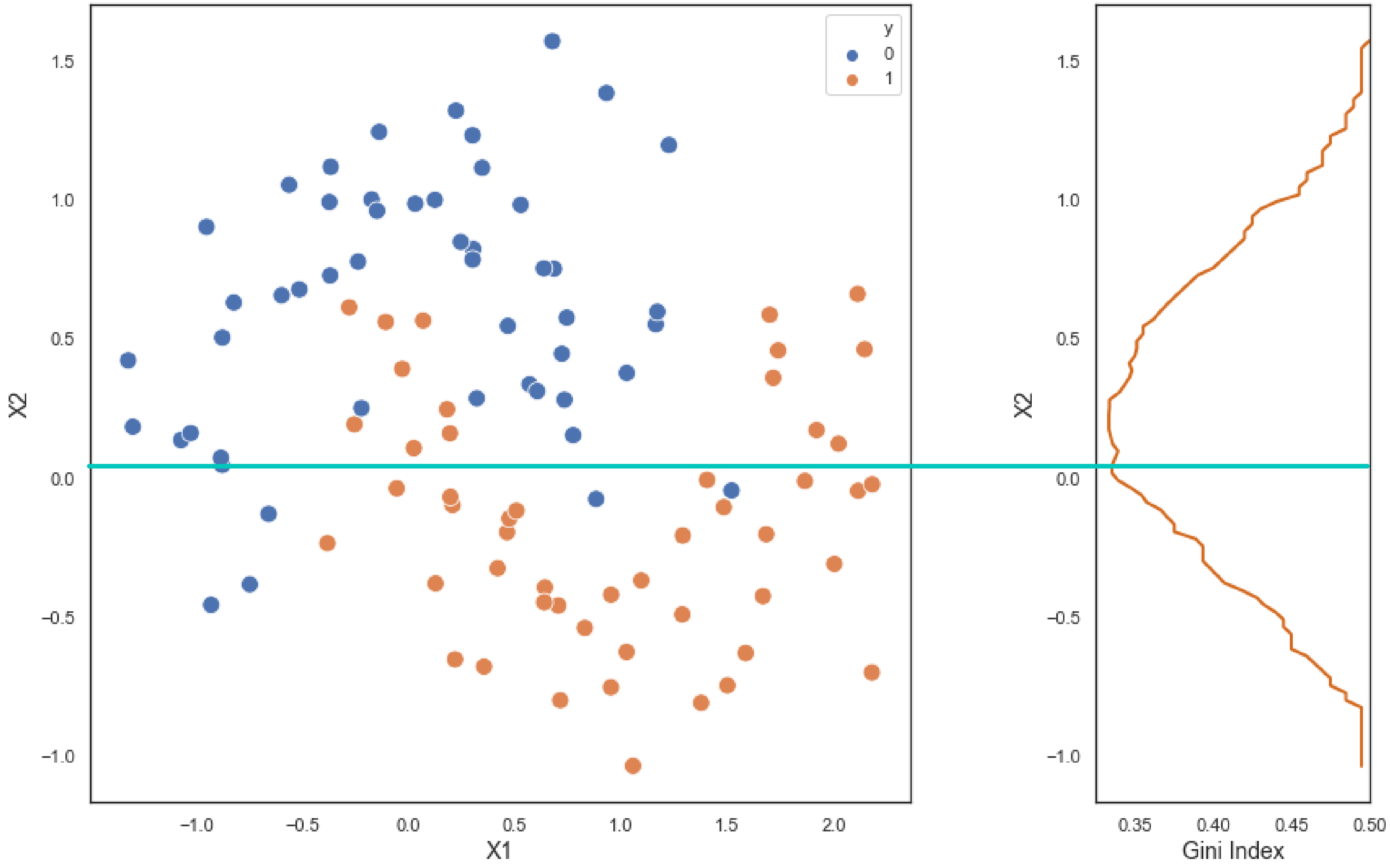
$$Gini = \sum_k p_k(1 - p_k)$$





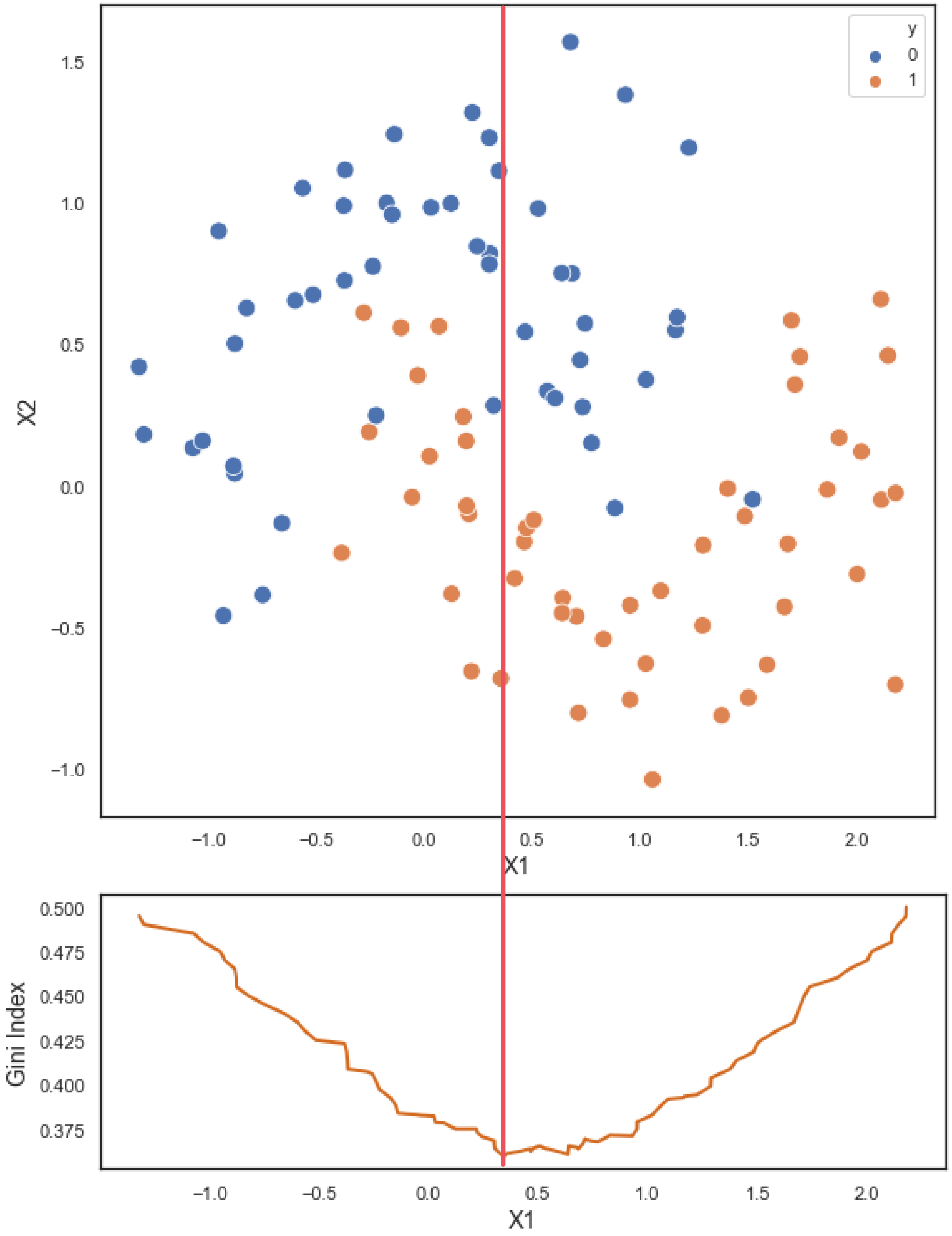
Where is the optimal cut-off (partition)

that would yield the lowest impurity?

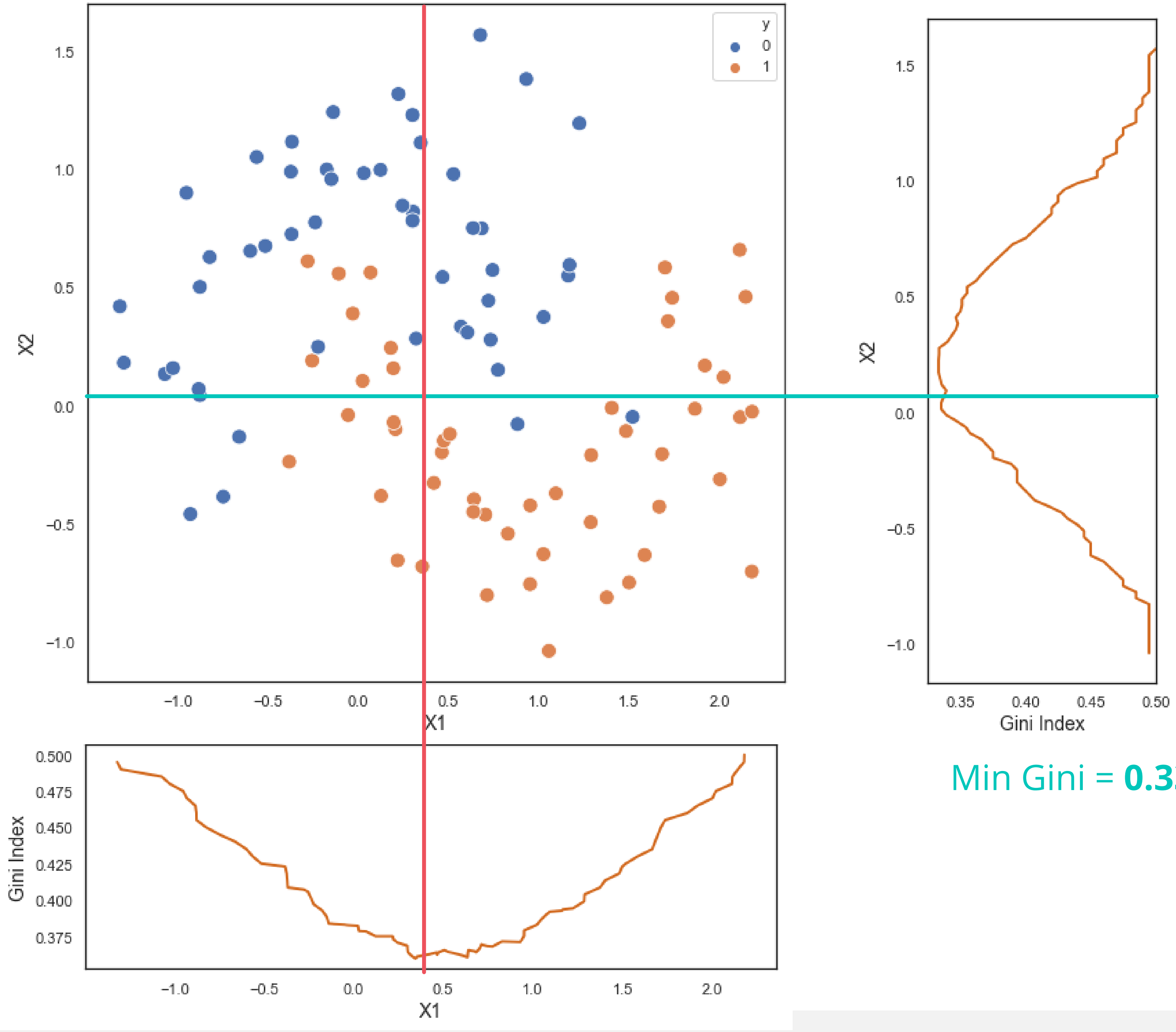


Min Gini = **0.33**

Min Gini = **0.36**

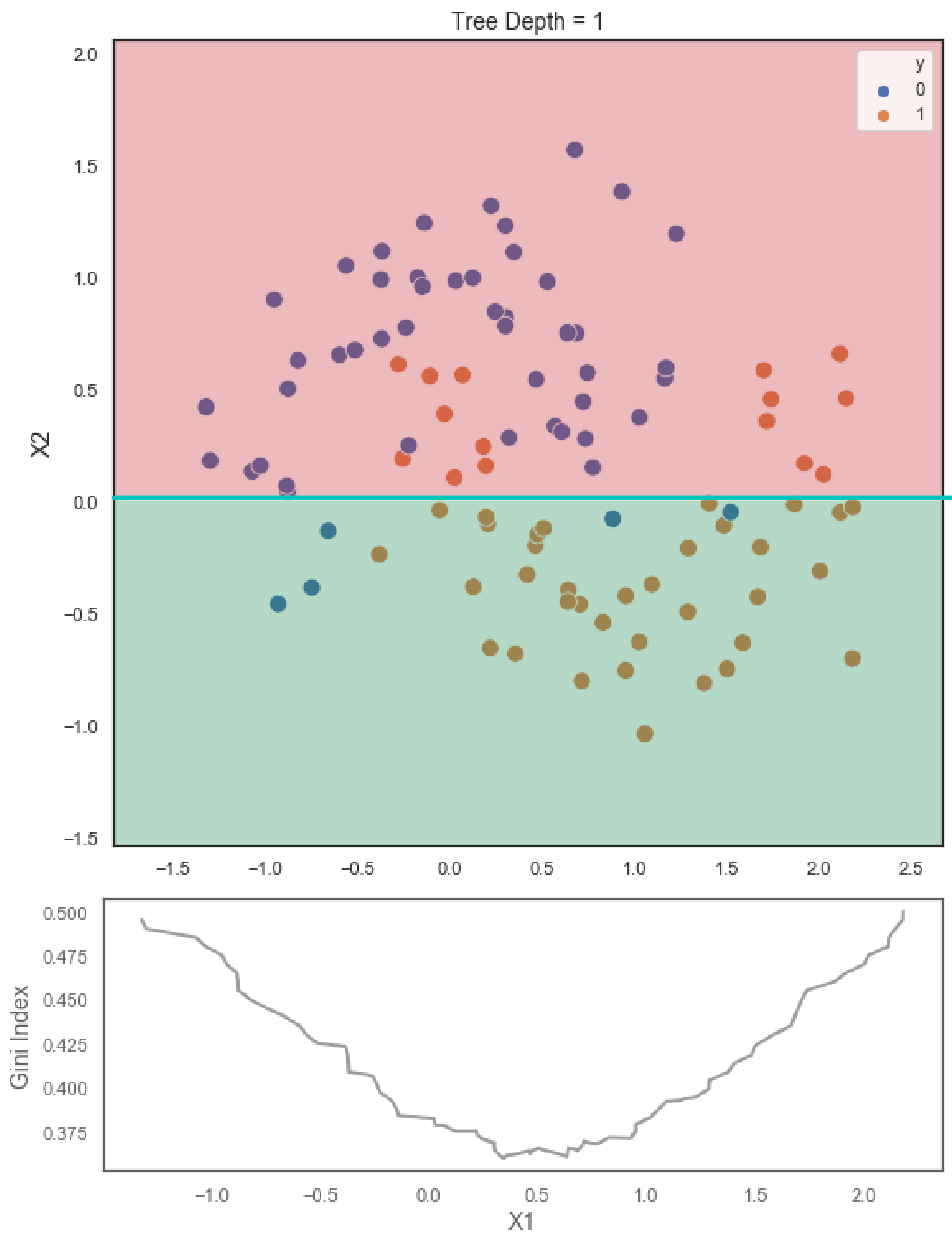


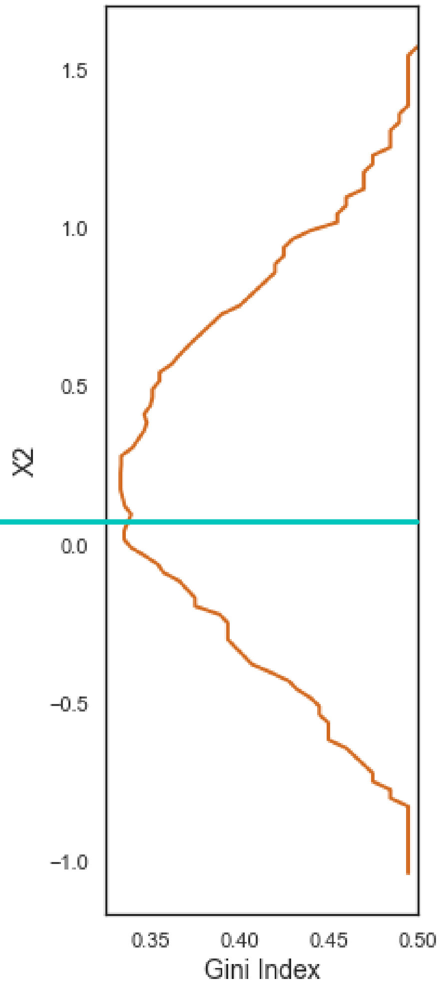
y
0
1
1.5
1.0
0.5
0.0
−0.5
−1.0
X2
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
X1
0.500
0.475
0.450
0.425
0.400
0.375
Gini Index
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
X1



Min Gini = **0.33**

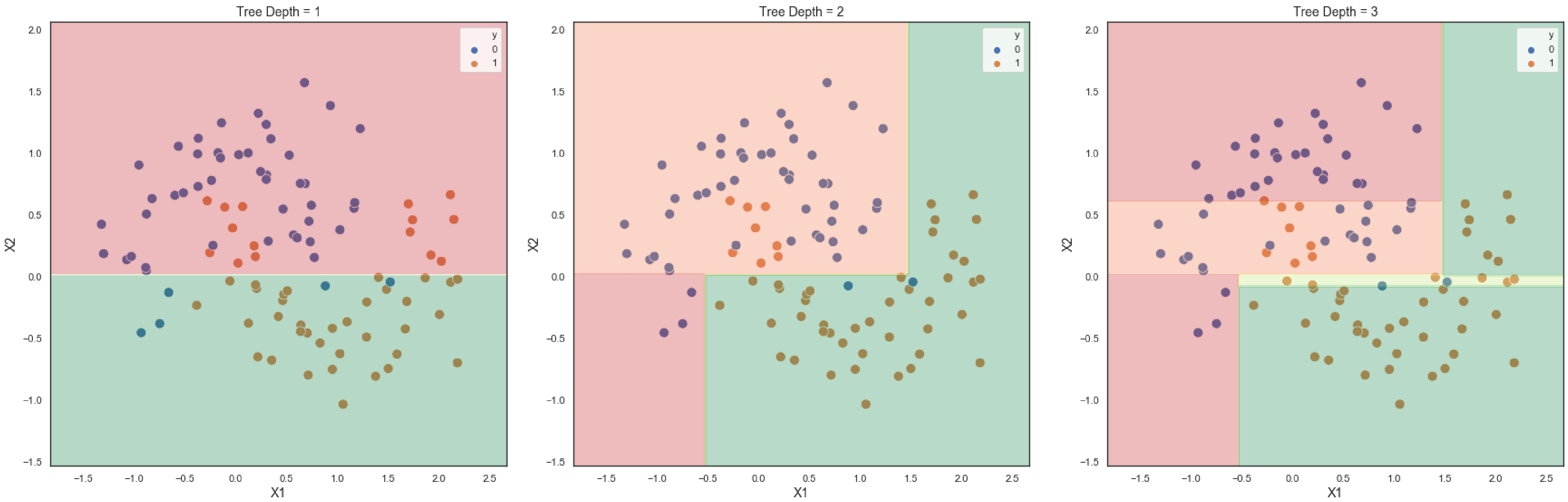
Min Gini = **0.36**



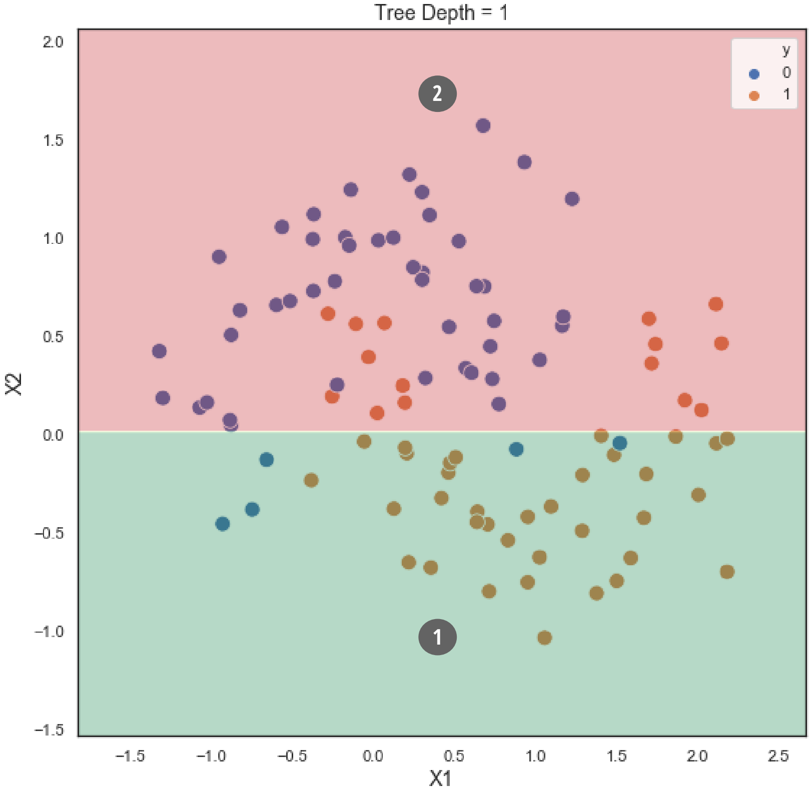


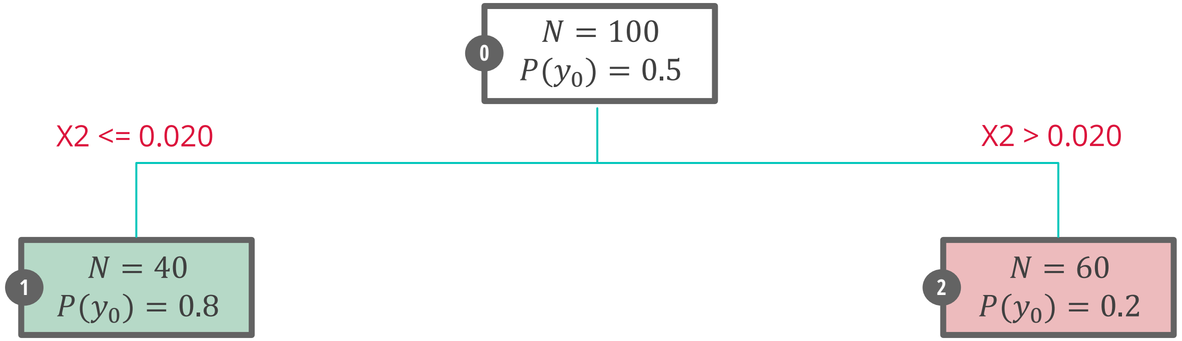
Min Gini = **0.33**

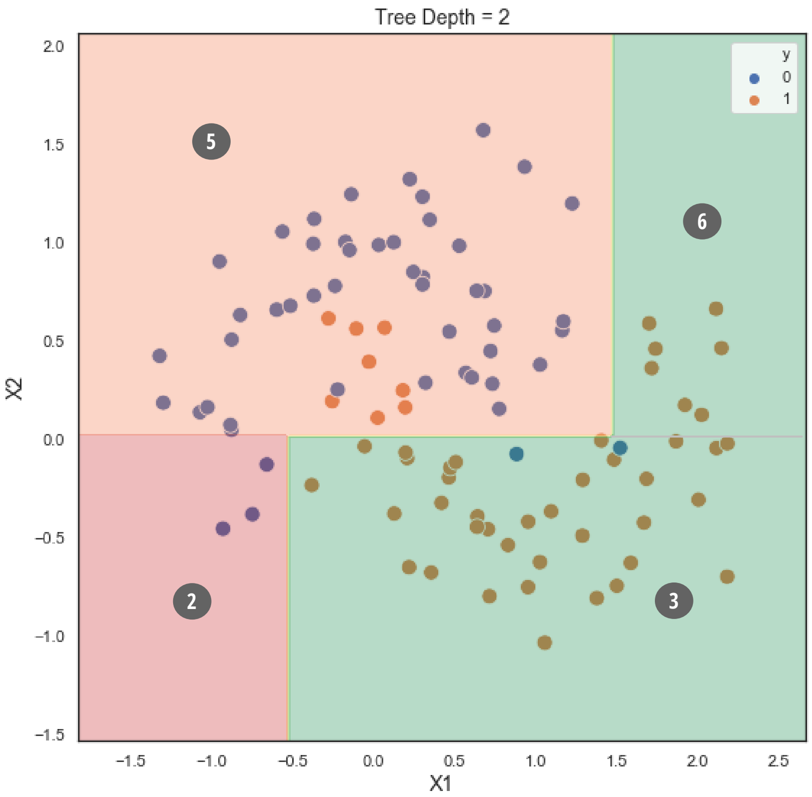
Min Gini = **0.36**



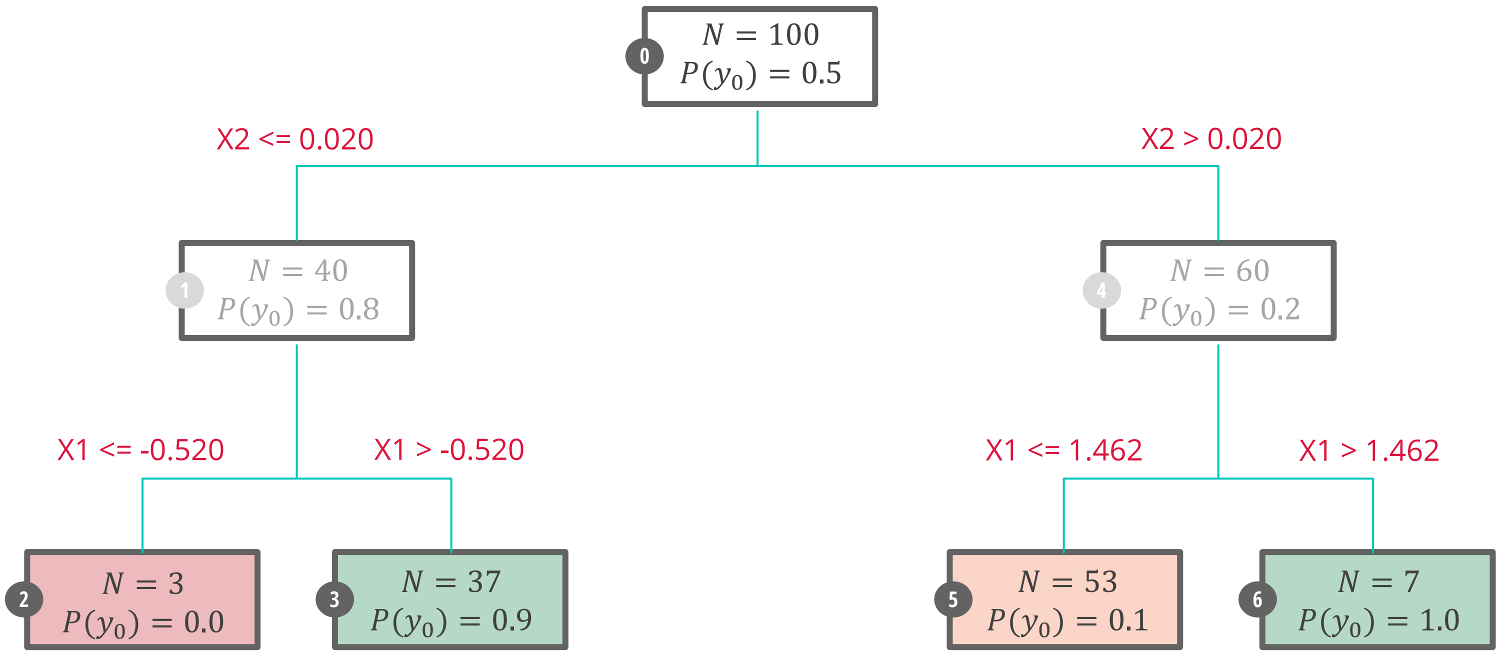
Reduction in impurity → "Information Gain"


Tree Depth = 1
y
0
1
2
1
X1
X2

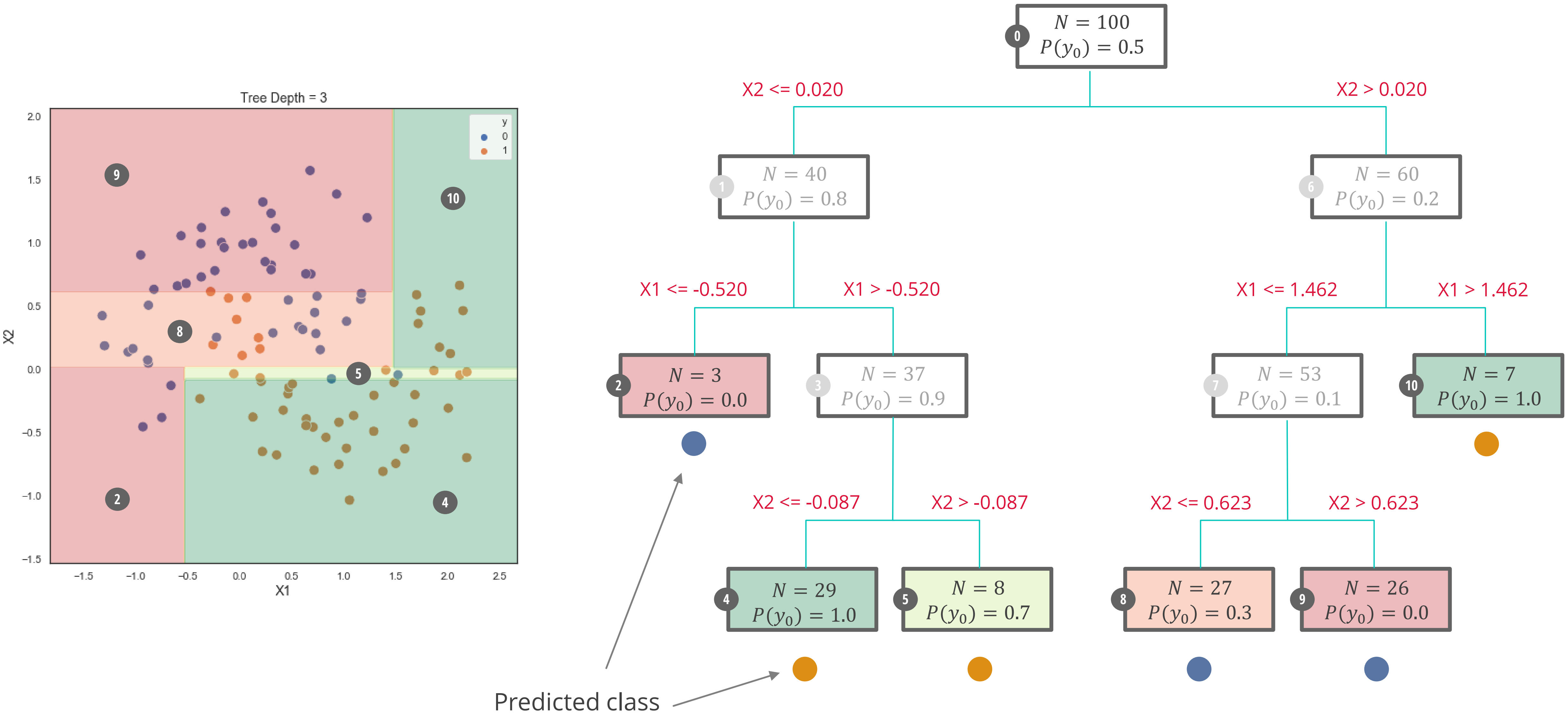

0
$N = 100$
$P(y_0) = 0.5$
X2 <= 0.020
X2 > 0.020
1
$N = 40$
$P(y_0) = 0.8$
2
$N = 60$
$P(y_0) = 0.2$



Tree Depth = 2
y
0
1
5
6
2
3
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
X2
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
X1

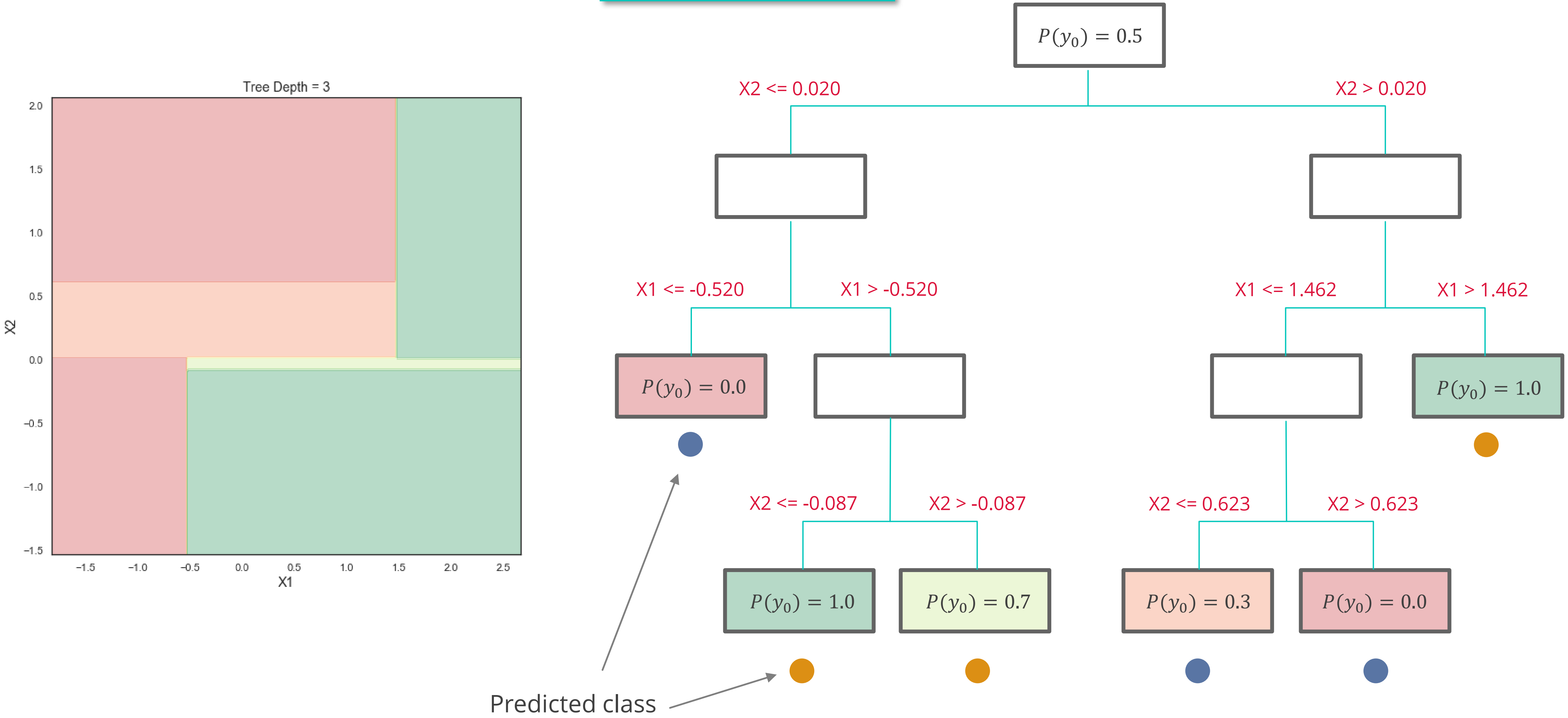


0
$N = 100$
$P(y_0) = 0.5$
X2 <= 0.020
X2 > 0.020
1
$N = 40$
$P(y_0) = 0.8$
4
$N = 60$
$P(y_0) = 0.2$
X1 <= -0.520
X1 > -0.520
X1 <= 1.462
X1 > 1.462
2
$N = 3$
$P(y_0) = 0.0$
3
$N = 37$
$P(y_0) = 0.9$
5
$N = 53$
$P(y_0) = 0.1$
6
$N = 7$
$P(y_0) = 1.0$

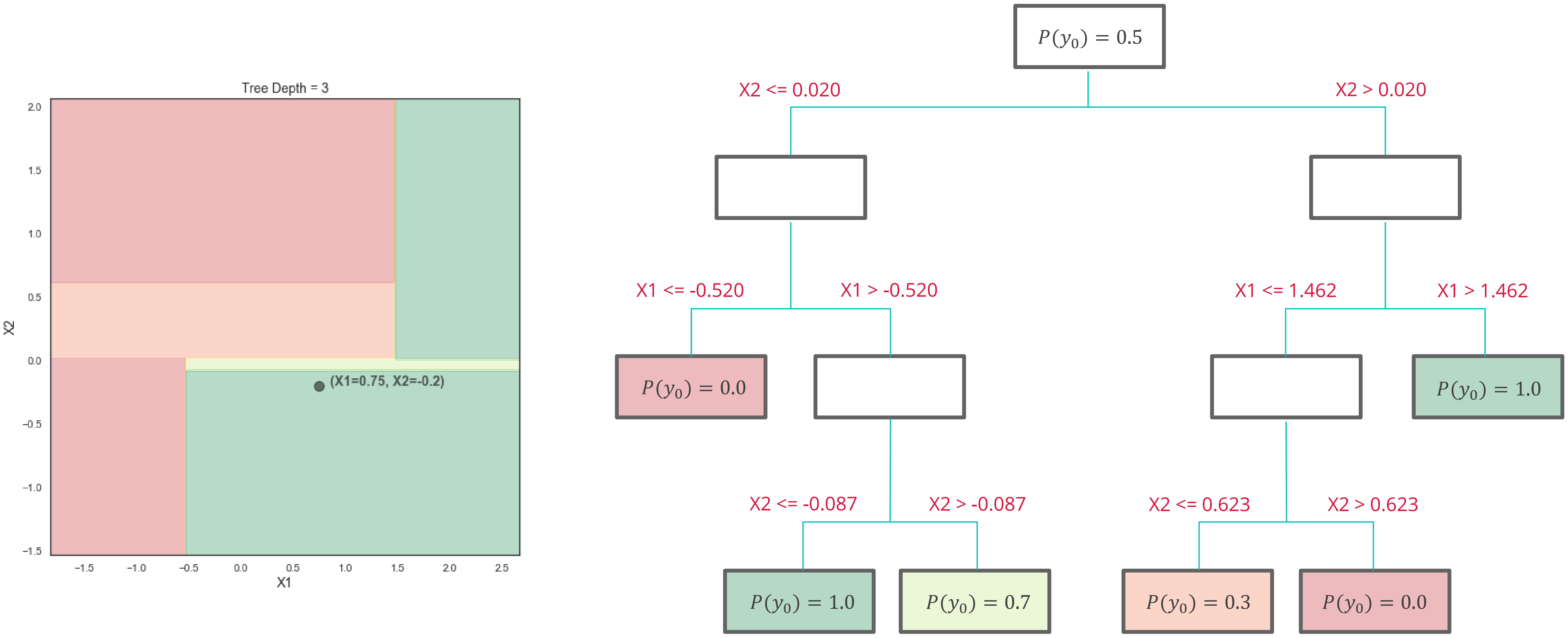


Tree Depth = 3
y
0
1
9
10
8
5
2
4
X1
X2
N = 100
P(y0) = 0.5
X2 <= 0.020
X2 > 0.020
N = 40
P(y0) = 0.8
N = 60
P(y0) = 0.2
X1 <= -0.520
X1 > -0.520
X1 <= 1.462
X1 > 1.462
N = 3
P(y0) = 0.0
N = 37
P(y0) = 0.9
N = 53
P(y0) = 0.1
N = 7
P(y0) = 1.0
X2 <= -0.087
X2 > -0.087
X2 <= 0.623
X2 > 0.623
N = 29
P(y0) = 1.0
N = 8
P(y0) = 0.7
N = 27
P(y0) = 0.3
N = 26
P(y0) = 0.0
Predicted class

The Model

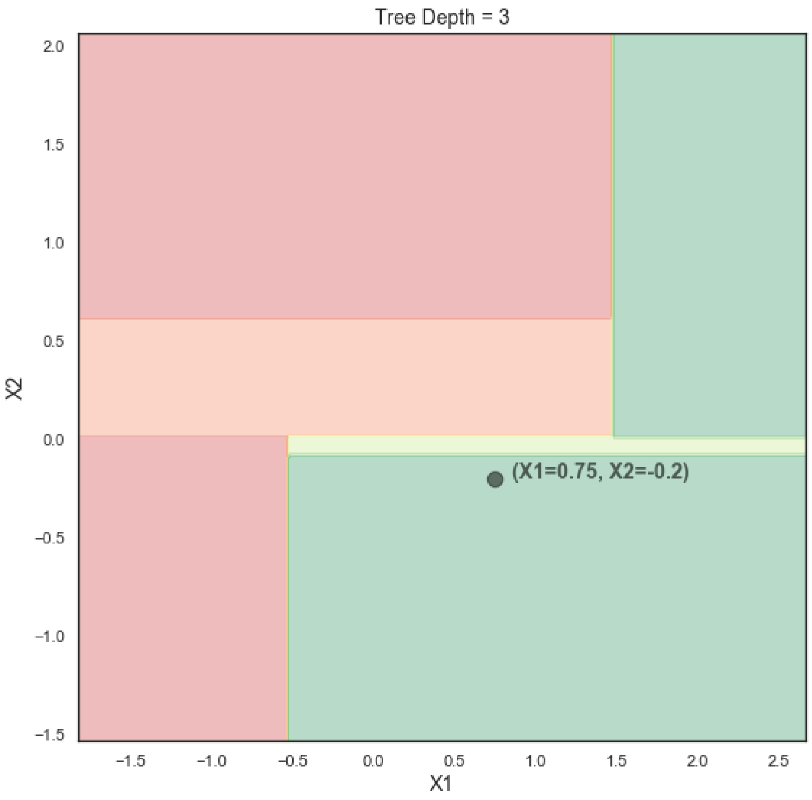


Tree Depth = 3
X1
X2
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
$P(y_0) = 0.5$
X2 <= 0.020
X2 > 0.020
X1 <= -0.520
X1 > -0.520
X1 <= 1.462
X1 > 1.462
$P(y_0) = 0.0$
$P(y_0) = 1.0$
X2 <= -0.087
X2 > -0.087
X2 <= 0.623
X2 > 0.623
$P(y_0) = 1.0$
$P(y_0) = 0.7$
$P(y_0) = 0.3$
$P(y_0) = 0.0$
Predicted class

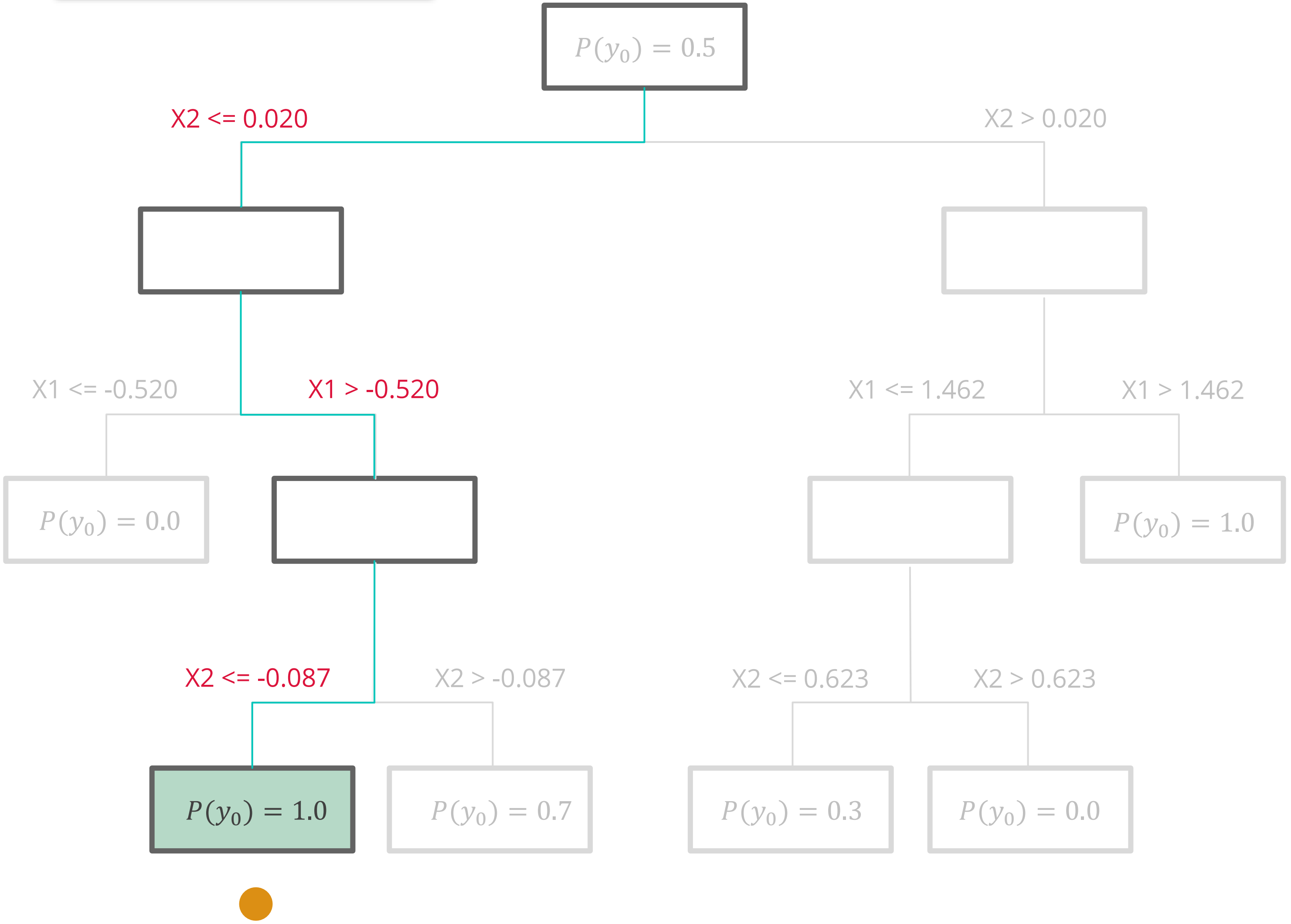


Tree Depth = 3
X1
X2
(X1=0.75, X2=-0.2)
P(y0) = 0.5
X2 <= 0.020
X2 > 0.020
X1 <= -0.520
X1 > -0.520
X1 <= 1.462
X1 > 1.462
P(y0) = 0.0
P(y0) = 1.0
X2 <= -0.087
X2 > -0.087
X2 <= 0.623
X2 > 0.623
P(y0) = 1.0
P(y0) = 0.7
P(y0) = 0.3
P(y0) = 0.0

Prediction



Tree Depth = 3
(X1=0.75, X2=-0.2)
X1
X2



$P(y_0) = 0.5$
X2 <= 0.020
X2 > 0.020
X1 <= -0.520
X1 > -0.520
X1 <= 1.462
X1 > 1.462
$P(y_0) = 0.0$
$P(y_0) = 1.0$
X2 <= -0.087
X2 > -0.087
X2 <= 0.623
X2 > 0.623
$P(y_0) = 1.0$
$P(y_0) = 0.7$
$P(y_0) = 0.3$
$P(y_0) = 0.0$

Decision Trees for Classification: Summary

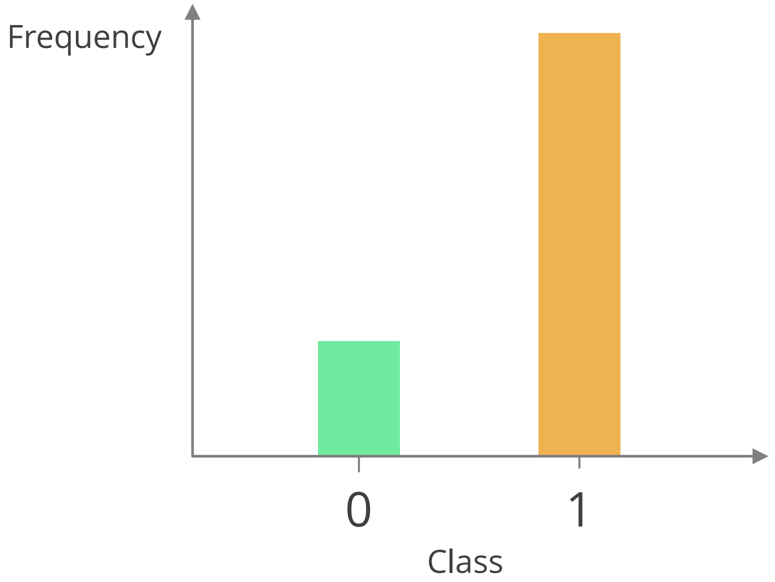
Impurity measure	Formula
Entropy (Cross-entropy, deviance)	$-\sum_k p_k \log_2 p_k$
Gini index	$\sum_k p_k(1-p_k)$
Mis-classification error	$\sum_k (1-p_k)$
Chi-Square	$\sum_i \frac{(x_i - m_i)^2}{m_i}$

- **CHAID** (Chi-Square Automatic Interaction Detection), based on adjusted significant testing
  - Can perform two or more splits
  - Not available in `scikit-learn`

Entropy $$H = -\sum_k p_k \log_2 p_k$$

Low Entropy

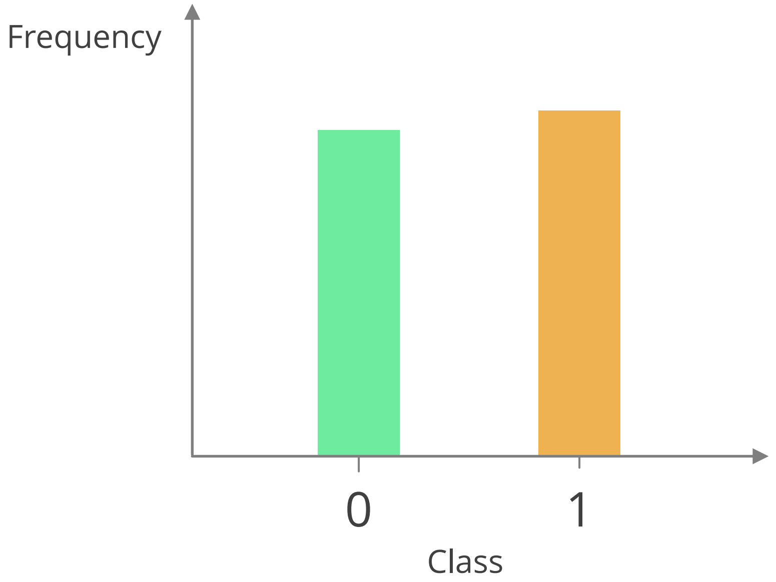
One class is more likely than the other.



High purity

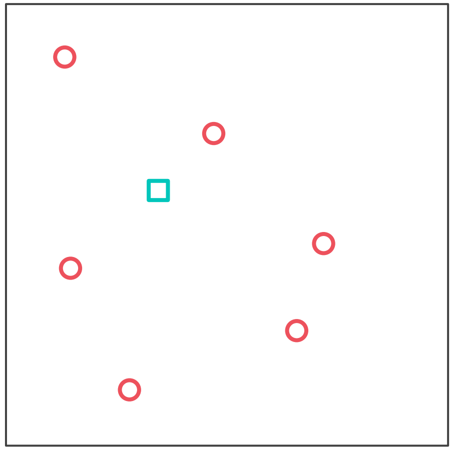
High Entropy

Both classes are nearly equally likely.



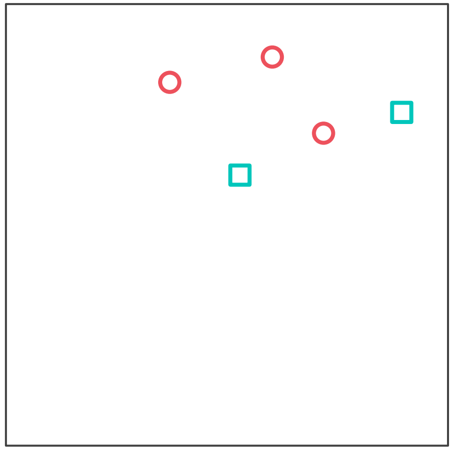
Low purity

A



$p_{\square} = \frac{1}{7}$ $\quad$ $p_{\circ} = \frac{6}{7}$

B



$p_{\square} = \frac{2}{5}$ $\quad$ $p_{\circ} = \frac{3}{5}$

$$H = -\sum_{k} p_k \log_2 p_k$$

$$H = -\,p_{\square} \log_2 p_{\square} - p_{\circ} \log_2 p_{\circ}$$

Low Entropy $\quad$ $\boldsymbol{H_A = 0.59}$

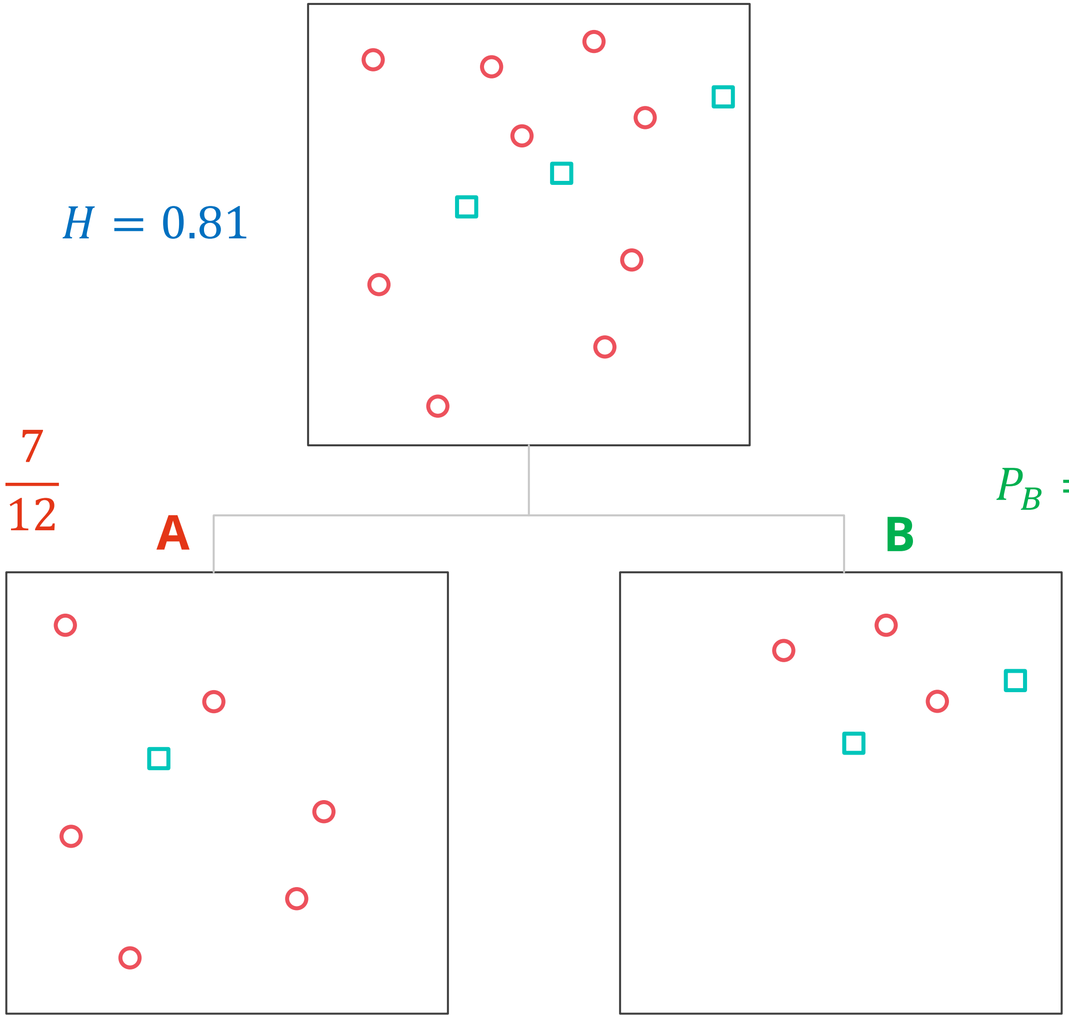
$\boldsymbol{H_B = 0.97}$ $\quad$ **High Entropy**

High purity

Low purity

Information Gain

$H = 0.81$



$P_A = \frac{7}{12}$ $P_B = \frac{5}{12}$

$H_A = 0.59$ $H_B = 0.97$

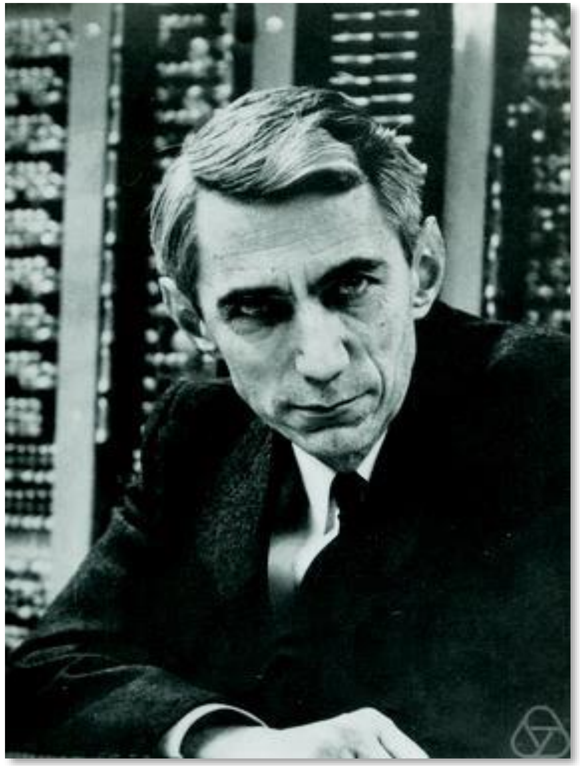
$$IG = H - (H_A * P_A + H_B * P_B)$$

$$IG = 0.81 - \left(0.59 * \frac{7}{12} + 0.97 * \frac{5}{12}\right)$$

$$IG = 0.81 - 0.75 = \mathbf{0.06}$$

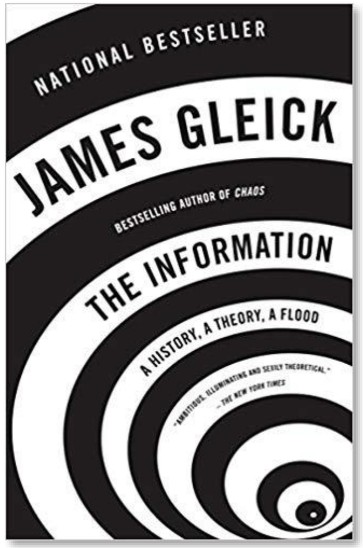
- **Information Gain**: The amount by which the ambiguity (entropy) decreases due to the split.
- The goal is to find the split that maximizes the information gain.

Information Entropy



Claude Shannon
(1916 –2001)
The father of Information Theory

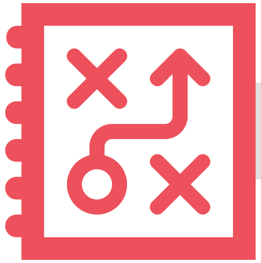
$$H = -\sum_k p_k \log_2 p_k$$



Khan Academy

Information Entropy
(7-minute video)

1



A B A C B D B D C

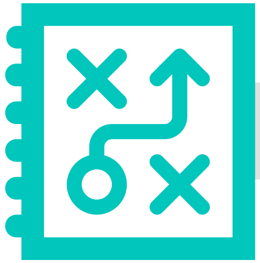
$P_A = 0.25$

$P_B = 0.25$

$P_C = 0.25$

$P_D = 0.25$

2



D A C A B A A C A

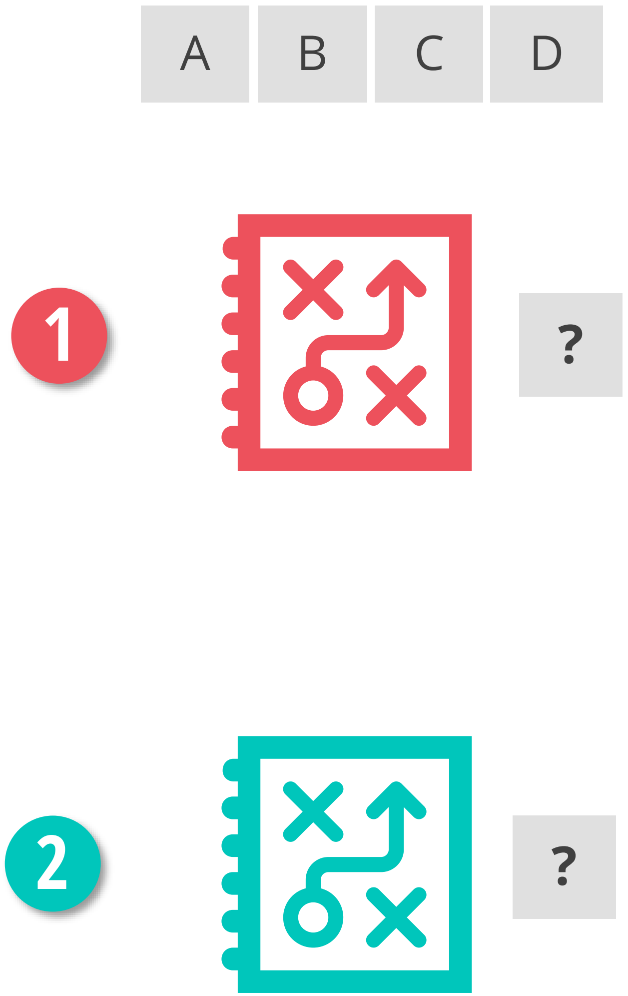
$P_A = 0.50$

$P_B = 0.125$

$P_C = 0.125$

$P_D = 0.25$

Which machine is producing more **information**?



Which machine is producing more **information**?

If you had to predict the **next symbol** from a machine, **how many (yes/no) questions** you would have to ask?

1

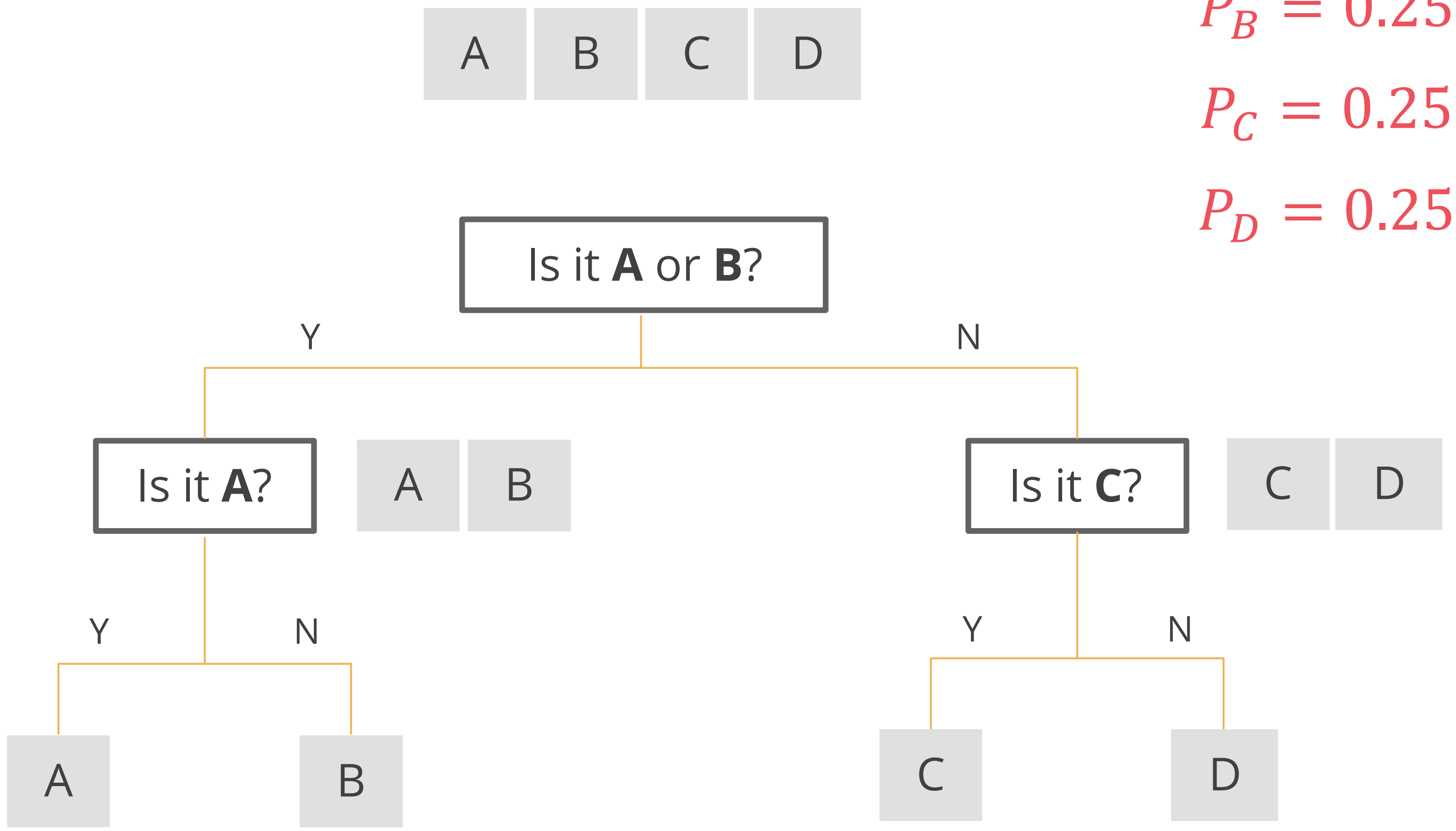
?

$P_A = 0.25$

$P_B = 0.25$

$P_C = 0.25$

$P_D = 0.25$



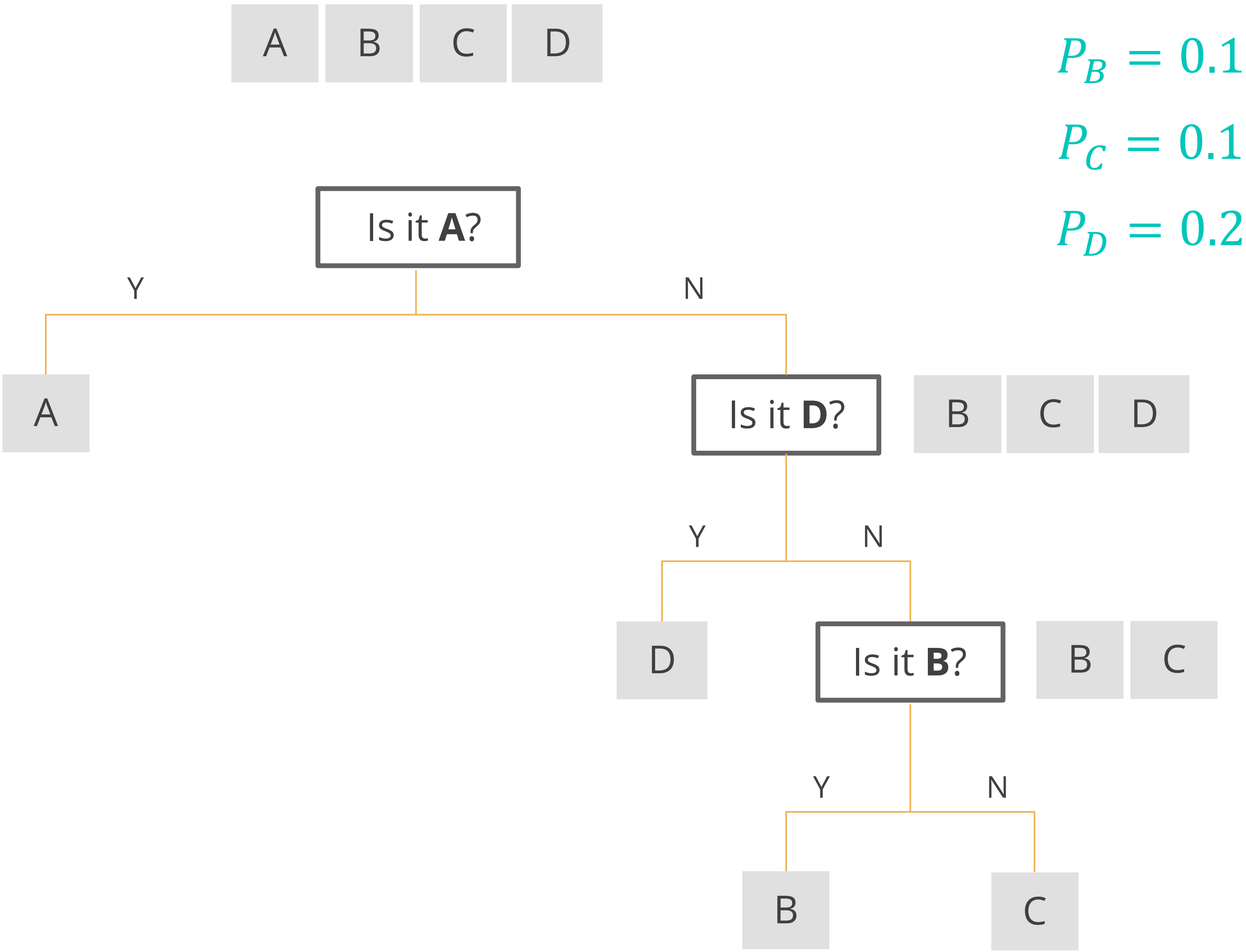
The uncertainty in **machine 1** is **two questions** per symbol.

$$P_A = 0.50$$

$$P_B = 0.125$$

$$P_C = 0.125$$

$$P_D = 0.25$$

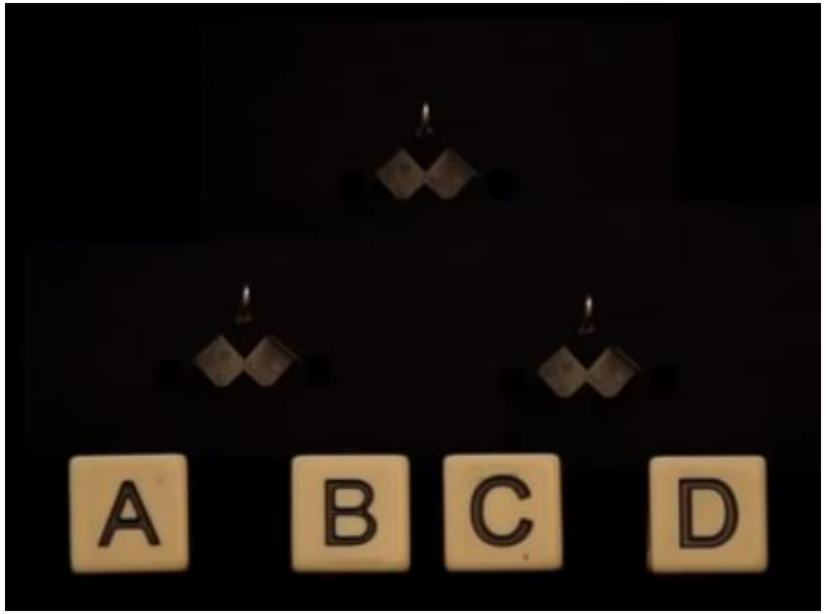


2

?

A	B	C	D

1 2 questions



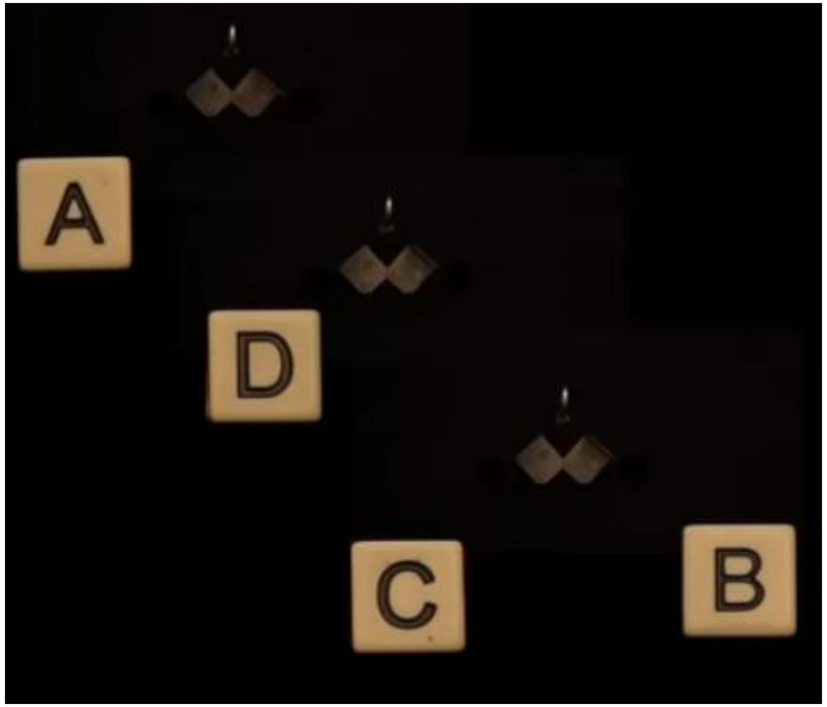
$P_A = 0.25$

$P_B = 0.25$

$P_C = 0.25$

$P_D = 0.25$

2 ?



$P_A = 0.50$

$P_B = 0.125$

$P_C = 0.125$

$P_D = 0.25$

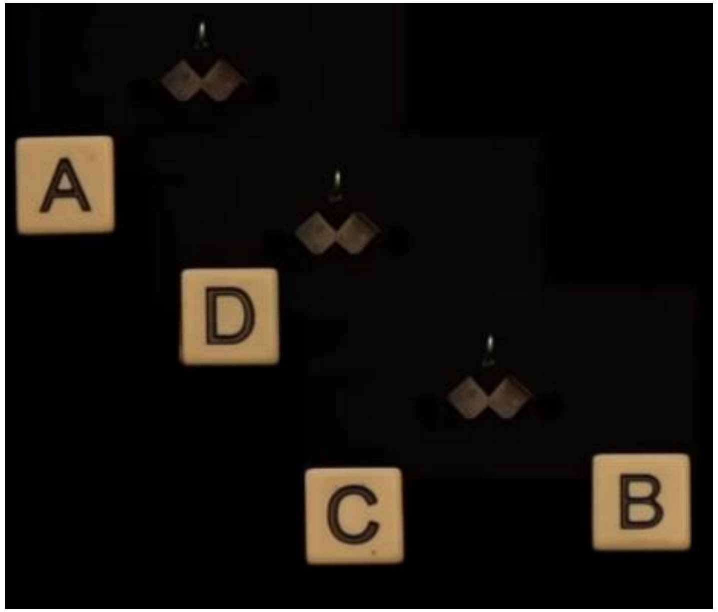
$$Expected\ \#\ of\ bounces = P_A * 1 + P_B * 3 + P_c * 3 + P_D * 2$$

$$= 0.5 * 1 + 0.125 * 3 + 0.125 * 3 + 0.25 * 2$$

$$= 1.75$$

$$= Expected\ \#\ of\ questions$$

2 ?



$P_A = 0.50$

$P_B = 0.125$

$P_C = 0.125$

$P_D = 0.25$

A	B	C	D

1 **2 questions**

2 **1.75 questions**

Machine 2 is producing **less information**, because there's less uncertainty (or surprise) about its output.

A Mathematical Theory of Communication

By C. E. SHANNON

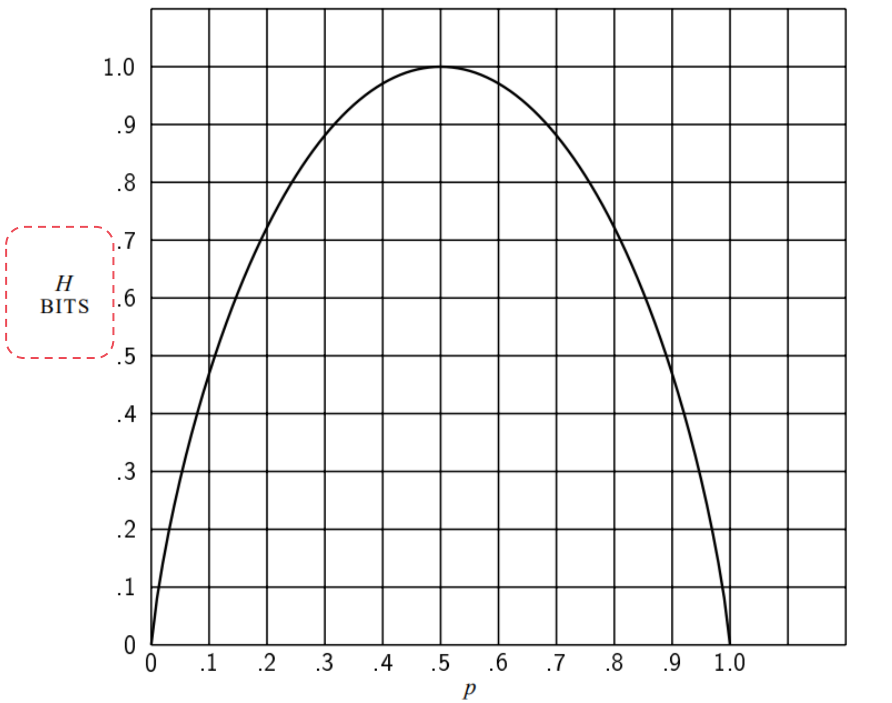
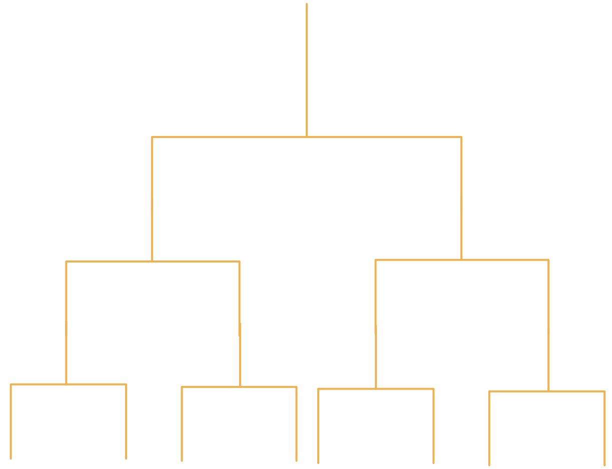


Fig. 7—Entropy in the case of two possibilities with probabilities p and $(1-p)$.

$$H = -\sum_k p_k \log_2 p_k$$

$$H = -\sum_k p_k * \#\, of\ bounces_k$$



$$\#\, of\ bounces = log_2(\#\, of\ outcomes)$$

$$\#\, of\ outcomes = \frac{1}{p}$$

$$\#\, of\ bounces = log_2\left(\frac{1}{p}\right) = -log_2 p$$

Classification Trees in `scikit-learn`

```
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,
    ccp_alpha=0.0)
```

1

```
# Import
from sklearn.tree import DecisionTreeClassifier
```

2

```
# Define
clf = DecisionTreeClassifier()
```

3

```
# Fit
clf.fit(x_train, y_class)
```

4

```
# Predict
clf.predict(x_test)
```

https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

```
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,
    ccp_alpha=0.0)
```

The function to measure the quality of a split.

Supported criteria are "`gini`" for the Gini impurity and "`entropy`" for the information gain.

```
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,
        ccp_alpha=0.0)
```

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.

Recommendation: `max_depth` between 6 and 10

```
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,
    ccp_alpha=0.0)
```

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

If `int`, then consider `min_samples_split` as the minimum number.

If `float`, then `ceil(min_samples_split * n_samples)` are the minimum number of samples for each split.

Recommendation: `min_samples_split = 0.05`

```
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=False)
```

The minimum number of samples required to be at a leaf node.

A split point at any depth will only be considered if it leaves at least `min_samples_leaf` training samples in each of the left and right branches.

Recommendation: `min_samples_leaf = 0.02`

```
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,
    ccp_alpha=0.0)
```

Set a user-defined seed for reproducible results.

If `int`, `random_state` is the seed used by the random number generator.

Recommendation: Always set a seed (e.g., 314) to ensure reproducible results.

```
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,
    ccp_alpha=0.0)
```

Weights associated with classes in the form {`class_label`: `weight`}.

If not given, all classes are supposed to have weight one.

Recommendation: `class_weight = 'balanced'`

when class imbalance is high

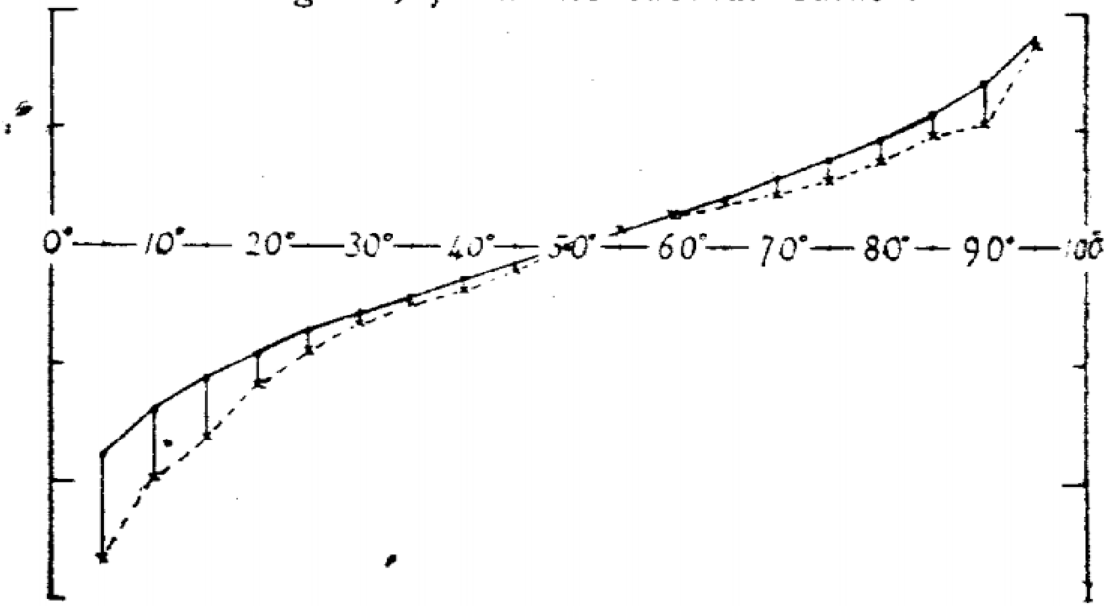
Random Forests

Distribution of the estimates of the dressed weight of a particular living ox, made by 787 different persons.

Degrees of the length of Array 0°—100°	Estimates in lbs.	* Centiles: Observed deviates from 1207 lbs.	* Centiles: Normal p.e =37	Excess of Observed over Normal
5	1074	− 133	− 90	+ 43
10	1109	− 98	− 70	+ 28
15	1126	− 81	− 57	+ 24
20	1148	− 59	− 46	+ 13
q_1 25	1162	− 45	− 37	+ 8
30	1174	− 33	− 29	+ 4
35	1181	− 26	− 21	+ 5
40	1188	− 19	− 14	+ 5
45	1197	− 10	− 7	+ 3
m 50	1207	0	0	0
55	1214	+ 7	+ 7	0
60	1219	+ 12	+ 14	− 2
65	1225	+ 18	+ 21	− 3
70	1230	+ 23	+ 29	− 6
q_3 75	1236	+ 29	+ 37	− 8
80	1243	+ 36	+ 46	− 10
85	1254	+ 47	+ 57	− 10
90	1267	+ 52	+ 70	− 18
95	1293	+ 86	+ 90	− 4

q_1, q_3, the first and third quartiles, stand at 25° and 75° respectively.
m, the median or middlemost value, stands at 50°.
The dressed weight proved to be 1198 lbs.

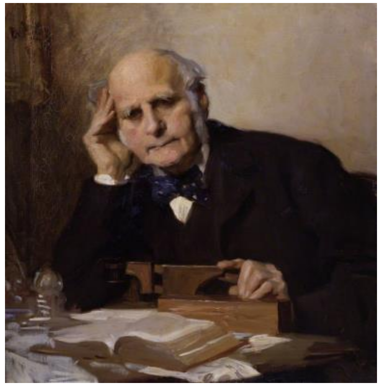
Diagram, from the tabular values.



The continuous line is the normal curve with p.e.=37.
The broken line is drawn from the observations.
The lines connecting them show the differences between the observed and the normal.

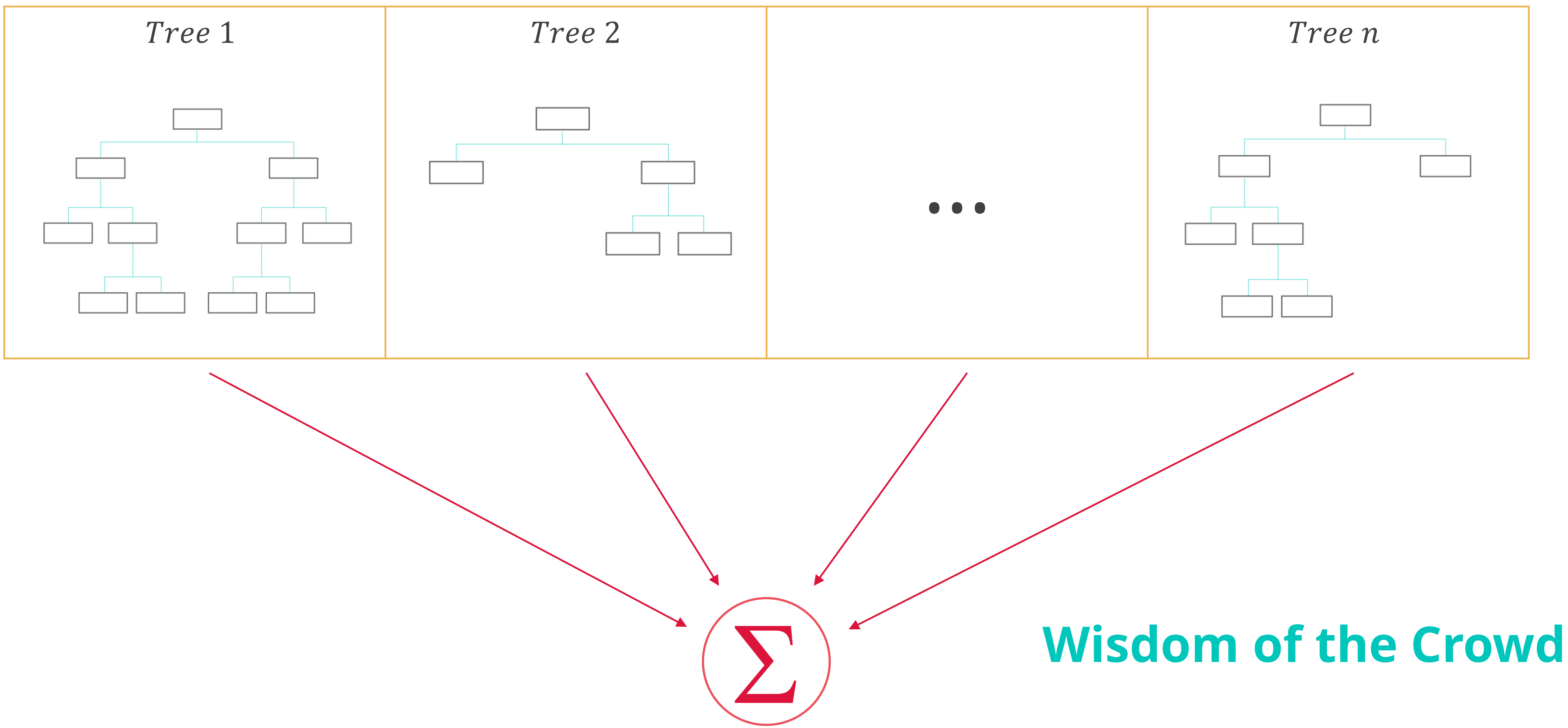
Wisdom of the Crowd

Vox populi



Sir Francis Galton

http://galton.org/essays/1900-1911/galton-1907-vox-populi.pdf



The final prediction (of the ensemble) is

the averaged prediction of the individual classifiers.

A **random forest** is a meta estimator

that fits a number of decision tree classifiers

on various sub-samples of the dataset

and uses averaging

to improve the predictive accuracy

and control over-fitting.

[scikit-learn]

Random Forests in `scikit-learn`

```
class sklearn.ensemble.RandomForestClassifier (
    n_estimators='warn',
    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)
```

1

```
# Import
from sklearn.ensemble import RandomForestClassifier
```

2

```
# Define
clf = RandomForestClassifier()
```

3

```
# Fit
clf.fit(x_train, y_class)
```

4

```
# Predict
clf.predict(x_test)
```

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

```
class sklearn.ensemble.RandomForestClassifier (
    n_estimators='warn',
    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)
```

The function to measure the quality of a split.

Supported criteria are "`gini`" for the Gini impurity and "`entropy`" for the information gain.

```
class sklearn.ensemble.RandomForestClassifier (
    n_estimators='warn',
    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)
```

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.

Recommendation: `max_depth` between 6 and 10

```
class sklearn.ensemble.RandomForestClassifier (
    n_estimators='warn',
    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)
```

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

If `int`, then consider `min_samples_split` as the minimum number.

If `float`, then `ceil(min_samples_split * n_samples)` are the minimum number of samples for each split.

Recommendation: `min_samples_split = 0.05`

```
class sklearn.ensemble.RandomForestClassifier (
    n_estimators='warn',
    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)
```

The minimum number of samples required to be at a leaf node.

A split point at any depth will only be considered if it leaves at least `min_samples_leaf` training samples in each of the left and right branches.

Recommendation: `min_samples_leaf = 0.02`

```
class sklearn.ensemble.RandomForestClassifier (
    n_estimators='warn',
    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)
```

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 `int(max_features * n_features)` features are considered at each split.

If "`auto`", then `max_features=sqrt(n_features)`.

If "`log2`", then `max_features=log2(n_features)`.

If `None`, then `max_features=n_features`.

Recommendation: `max_features = 'auto'`

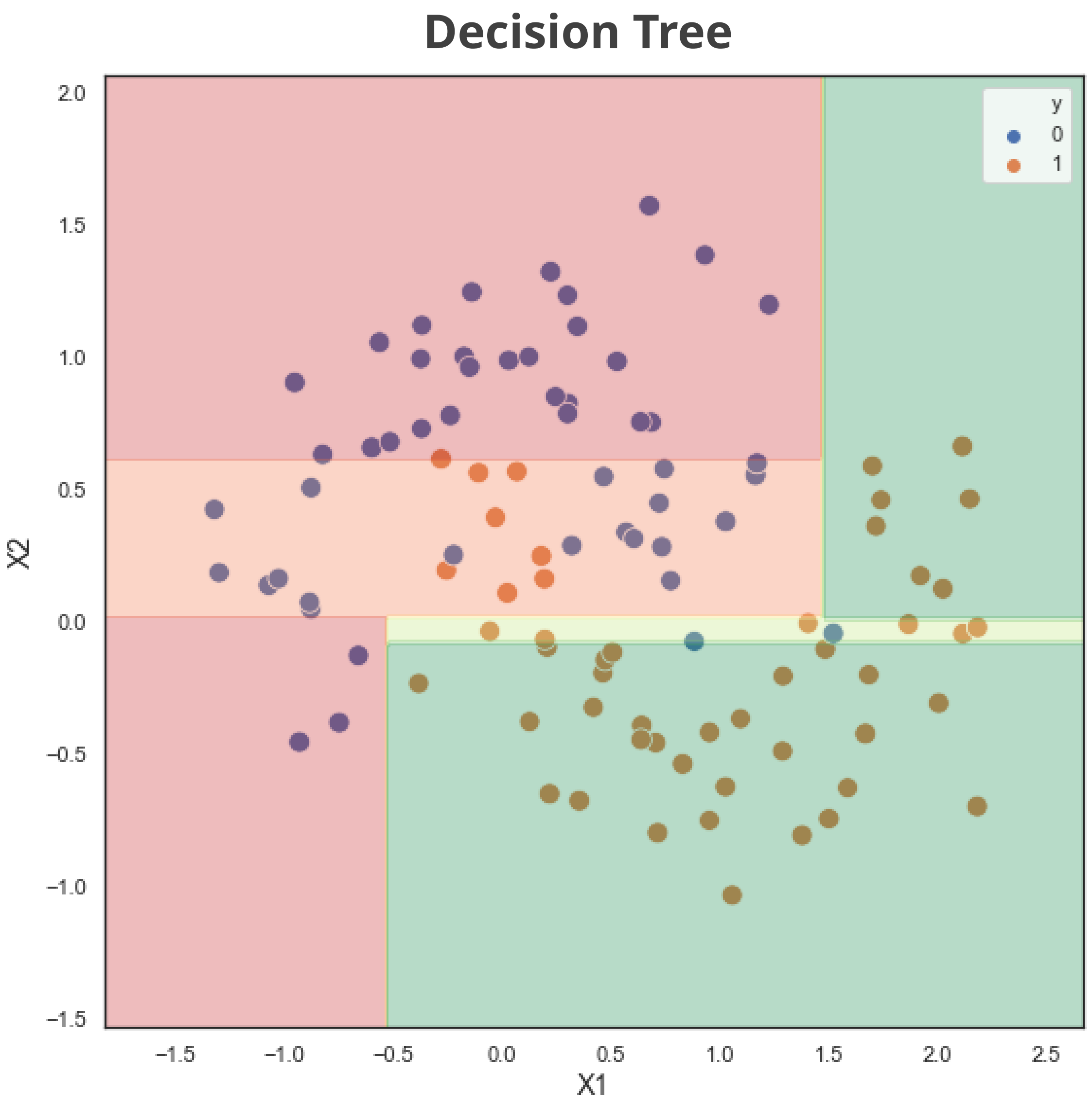
```
class sklearn.ensemble.RandomForestClassifier (
    n_estimators='warn',
    criterion='gini',
    max_depth=None,
    min_samples_split=2,
    min_samples_leaf=1,
    min_weight_fraction_leaf=0.0,
    max_features='auto',
    max_leaf_nodes=None,
    min_impurity_decrease=0.0,
    min_impurity_split=None,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

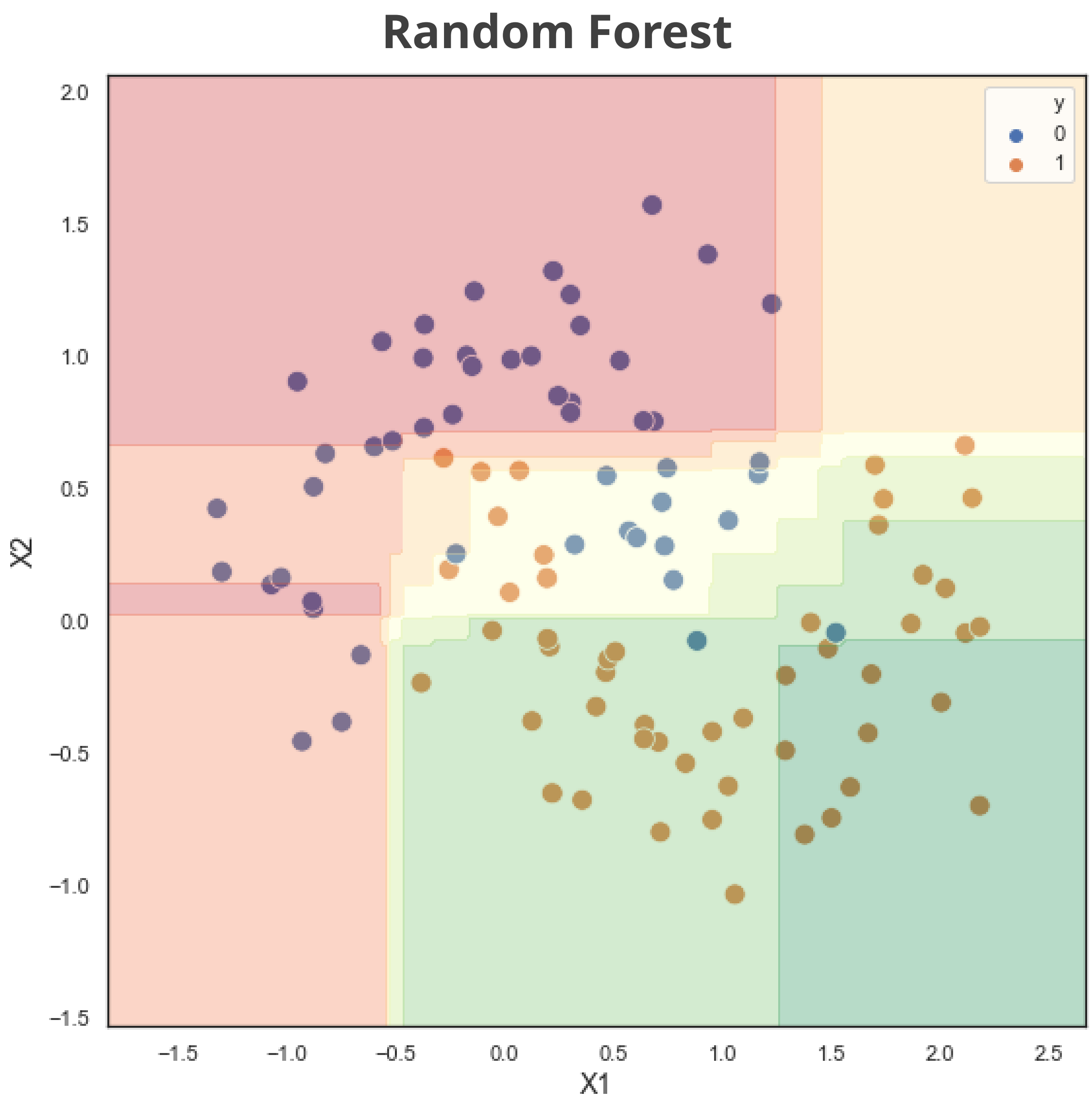
Whether bootstrap samples are used when building trees.

Recommendation: `bootstrap = True`

Bagging (Bootstrap Aggregation)

- **Random Forest:** For each tree, it uses a random subset of data as well as a random subset of all available features.
- The goal is to reduce the variance by doing this.


Decision Tree
y
0
1
X1
X2
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5


Random Forest
y
0
1
X1
X2
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5

Feature Importance

```
from sklearn.datasets import make_classification

# Build a classification task using 3 informative features
X, y = make_classification(n_samples=1000,
                           n_features=10,
                           n_informative=3,
                           n_redundant=0,
                           n_repeated=0,
                           n_classes=2,
                           random_state=314,
                           shuffle=False)
```

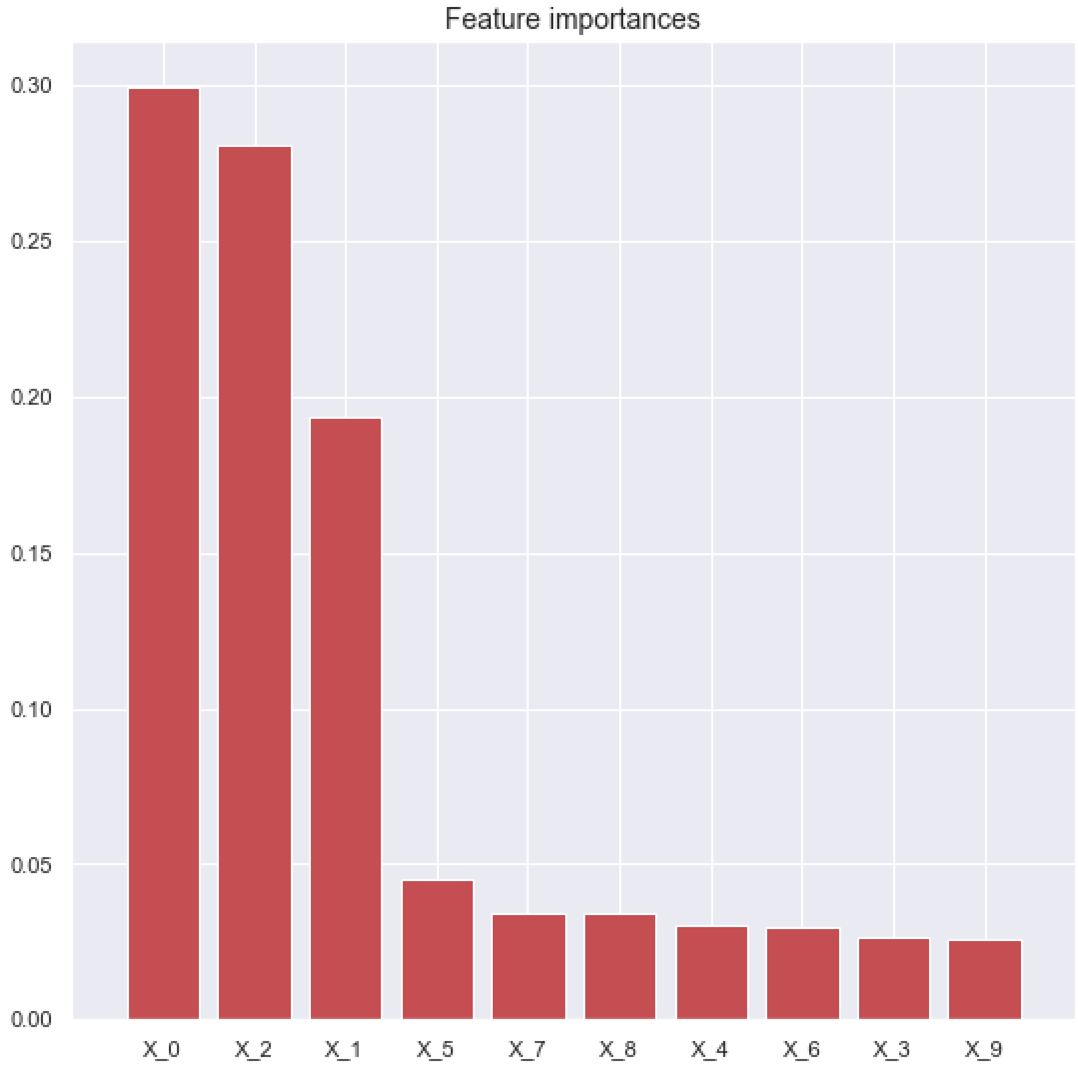
```
# Build a forest and compute the feature importances
from sklearn.ensemble import RandomForestClassifier

forest = RandomForestClassifier(random_state=314)

forest.fit(X, y)
```

```
importances = forest.feature_importances_

indices = np.argsort(importances)[::-1]
```



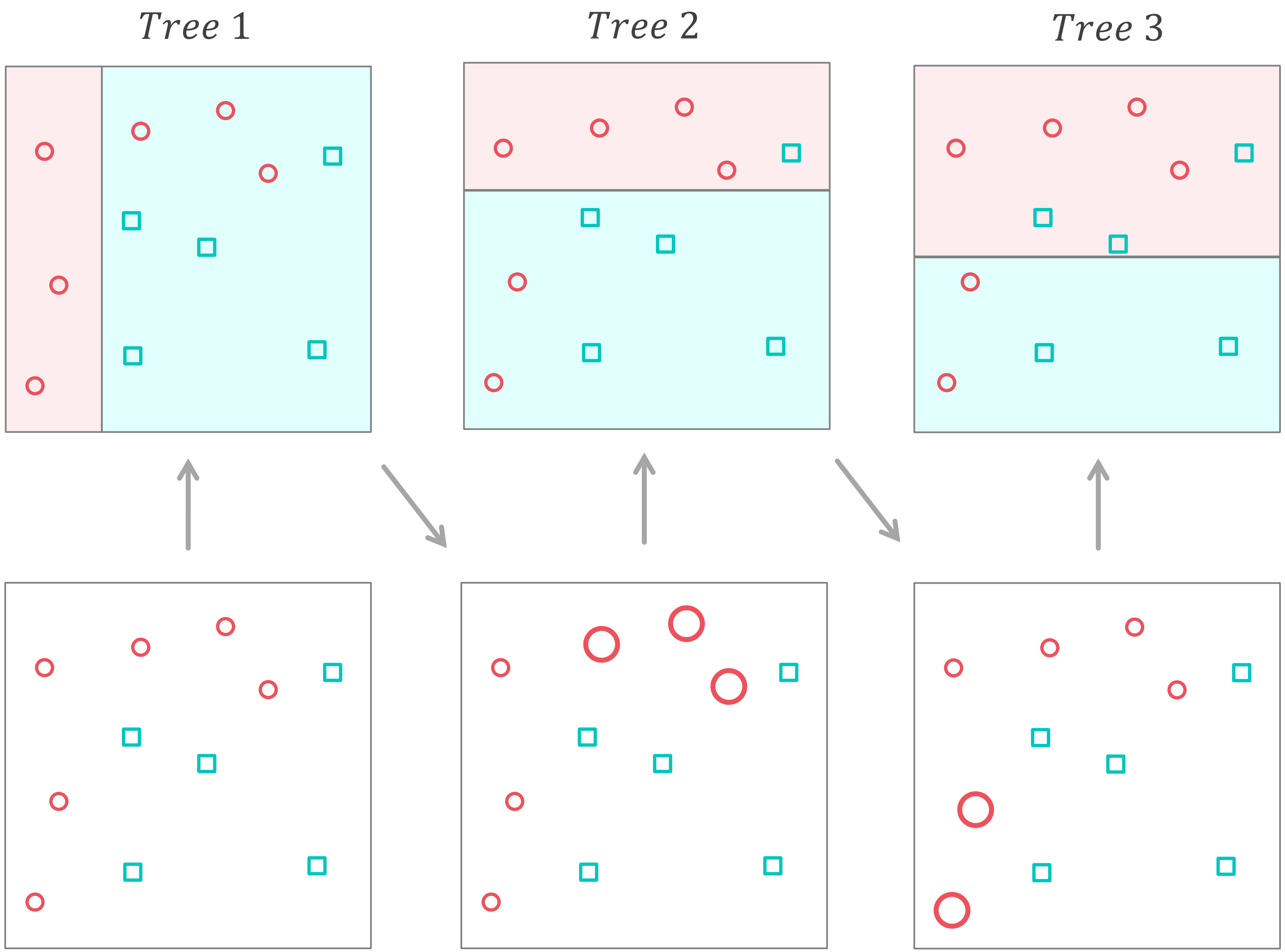
https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html

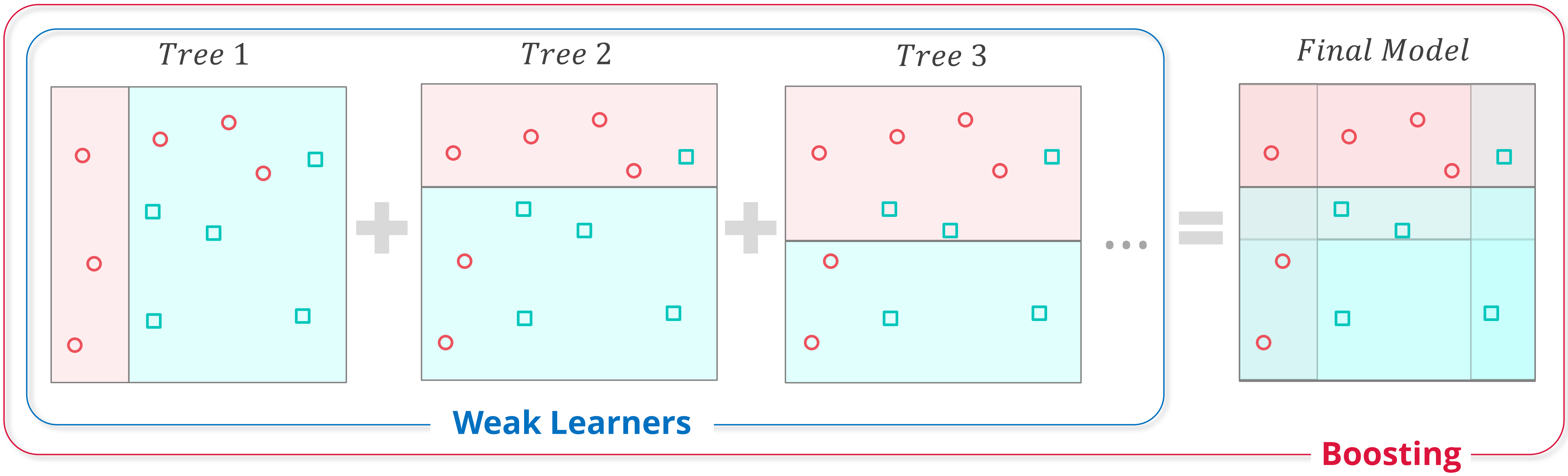
Feature Importance

- Features used at the top of the tree contribute to the final prediction decision of a larger fraction of the input samples.
- The expected fraction of the samples they contribute to can thus be used as an estimate of the relative importance of the features.
- In `scikit-learn`, the fraction of samples a feature contributes to is combined with the decrease in impurity from splitting them to create a normalized estimate of the predictive power of that feature.

Gradient Boosting Classifiers

1993


Tree 1
Tree 2
Tree 3

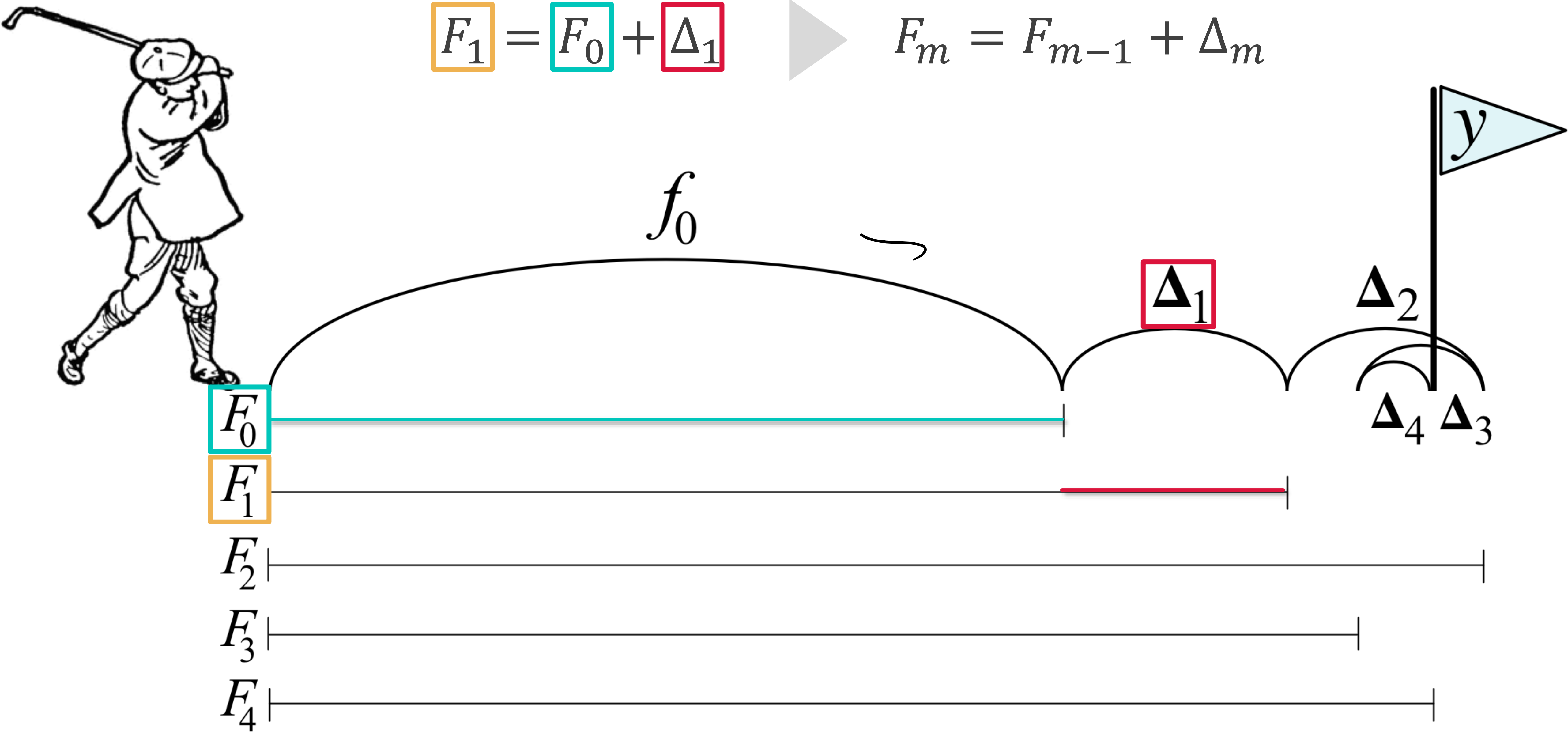


Gradient Boosting = Boosting + Gradient Descent

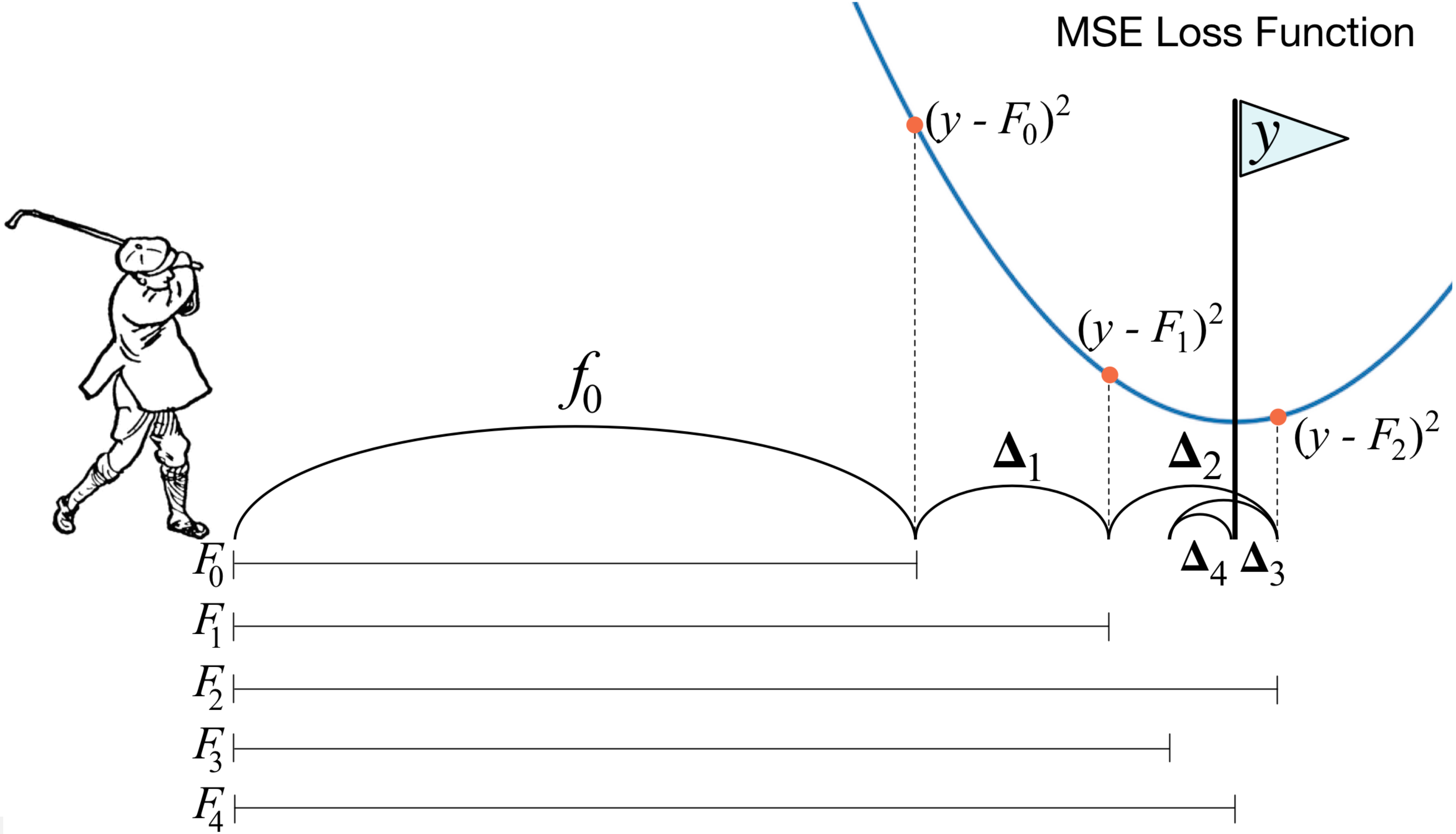
Greedy Function Approximation: A Gradient Boosting Machine – Jerome Friedman (1999)

A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting – Y. Freund, and R. Schapire (1997)

$$\boxed{F_1} = \boxed{F_0} + \boxed{\Delta_1} \qquad F_m = F_{m-1} + \Delta_m$$



$$\hat{y} = f_0(\mathbf{x}) + \Delta_1(\mathbf{x}) + \Delta_2(\mathbf{x}) + \ldots + \Delta_M(\mathbf{x})$$



MSE Loss Function
$(y - F_0)^2$
y
$(y - F_1)^2$
$(y - F_2)^2$
f_0
$\mathbf{\Delta}_1$
$\mathbf{\Delta}_2$
$\mathbf{\Delta}_4$
$\mathbf{\Delta}_3$
F_0
F_1
F_2
F_3
F_4

Stage m	Boosted Model	Model Output $\hat{y}$	Actual Error	Predicted Error Δ_m
0	F_0	70		
1	$F_1 = F_0 + \Delta_1$	$70 + 15 = 85$	$100 - 70 = 30$	$\Delta_1 = 15$
2	$F_2 = F_1 + \Delta_2$	$85 + 20 = 105$	$100 - 85 = 15$	$\Delta_2 = 20$
3	$F_3 = F_2 + \Delta_3$	$105 - 10 = 95$	$100 - 105 = -5$	$\Delta_3 = -10$
4	$F_4 = F_3 + \Delta_4$	$95 + 5 = 100$	$100 - 95 = 5$	$\Delta_4 = 5$

Algorithm: $l2boost(X, y, M, \eta)$ **returns** model F_M

1. Let $F_0(X) = \frac{1}{N}\sum_{i=1}^{N} y_i$, mean of target y across all observations
2. **for** $m = 1$ **to** M **do**

   Let $r_{m-1} = y - F_{m-1}(X)$ be the residual vector

   Train regression tree Δ_m on r_{m-1}, minimizing squared error

   $$F_m(X) = F_{m-1}(X) + \eta\Delta_m(X)$$

   end
3. **return** F_M

HYPER-PARAMETERS → GRID SEARCH

M = number of steps
η = learning rate

GBM algorithm to minimize L_2 loss, adapted from Friedman's LS_Boost and LAD_TreeBoost: https://explained.ai/gradient-boosting/L2-loss.html


Linear Regression
Gradient Boosting

Gradient Boosting scikit-learn

```
class sklearn.ensemble.GradientBoostingClassifier (
    loss='deviance',
    learning_rate=0.1,
    n_estimators=100,
    subsample=1.0,
    criterion='friedman_mse',
    min_samples_split=2,
    min_samples_leaf=1,
    min_weight_fraction_leaf=0.0,
    max_depth=3,
    min_impurity_decrease=0.0,
    min_impurity_split=None,
    init=None,
    random_state=None,
    max_features=None,
    verbose=0,
    max_leaf_nodes=None,
    warm_start=False,
    presort='auto',
    validation_fraction=0.1,
    n_iter_no_change=None,
    tol=0.0001)
```

The learning rate shrinks the contribution of each tree by `learning_rate`.

There is a trade-off between `learning_rate` and `n_estimators`.

η = learning rate

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

```
class sklearn.ensemble.GradientBoostingClassifier (
        loss='deviance',
        learning_rate=0.1,
        n_estimators=100,
        subsample=1.0,
        criterion='friedman_mse',
        min_samples_split=2,
        min_samples_leaf=1,
        min_weight_fraction_leaf=0.0,
        max_depth=3,
        min_impurity_decrease=0.0,
        min_impurity_split=None,
        init=None,
        random_state=None,
        max_features=None,
        verbose=0,
        max_leaf_nodes=None,
        warm_start=False,
        presort='auto',
        validation_fraction=0.1,
        n_iter_no_change=None,
        tol=0.0001)
```

The **number of boosting stages** to perform.

Gradient boosting is fairly robust to over-fitting so a large number usually results in better performance.

M= number of steps

```
class sklearn.ensemble.GradientBoostingClassifier (
        loss='deviance',
        learning_rate=0.1,
        n_estimators=100,
        subsample=1.0,
        criterion='friedman_mse',
        min_samples_split=2,
        min_samples_leaf=1,
        min_weight_fraction_leaf=0.0,
        max_depth=3,
        min_impurity_decrease=0.0,
        min_impurity_split=None,
        init=None,
        random_state=None,
        max_features=None,
        verbose=0,
        max_leaf_nodes=None,
        warm_start=False,
        presort='auto',
        validation_fraction=0.1,
        n_iter_no_change=None,
        tol=0.0001)
```

The **fraction of samples** to be used for fitting the individual base learners.

If smaller than 1.0 this results in Stochastic Gradient Boosting.

Choosing `subsample < 1.0` leads to a reduction of variance and an increase in bias.

```
class sklearn.ensemble.GradientBoostingClassifier (
        loss='deviance',
        learning_rate=0.1,
        n_estimators=100,
        subsample=1.0,
        criterion='friedman_mse',
        min_samples_split=2,
        min_samples_leaf=1,
        min_weight_fraction_leaf=0.0,
        max_depth=3,
        min_impurity_decrease=0.0,
        min_impurity_split=None,
        init=None,
        random_state=None,
        max_features=None,
        verbose=0,
        max_leaf_nodes=None,
        warm_start=False,
        presort='auto',
        validation_fraction=0.1,
        n_iter_no_change=None,
        tol=0.0001)
```

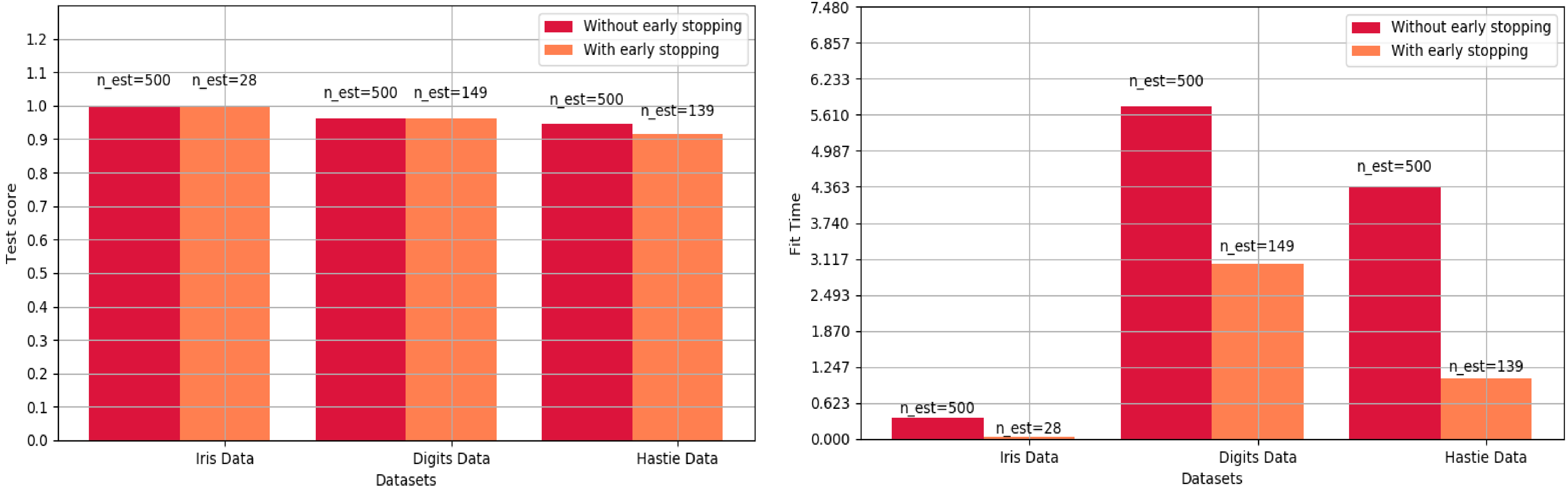
The **proportion of training data** to set aside as validation set for **early stopping**.

Must be between 0 and 1.

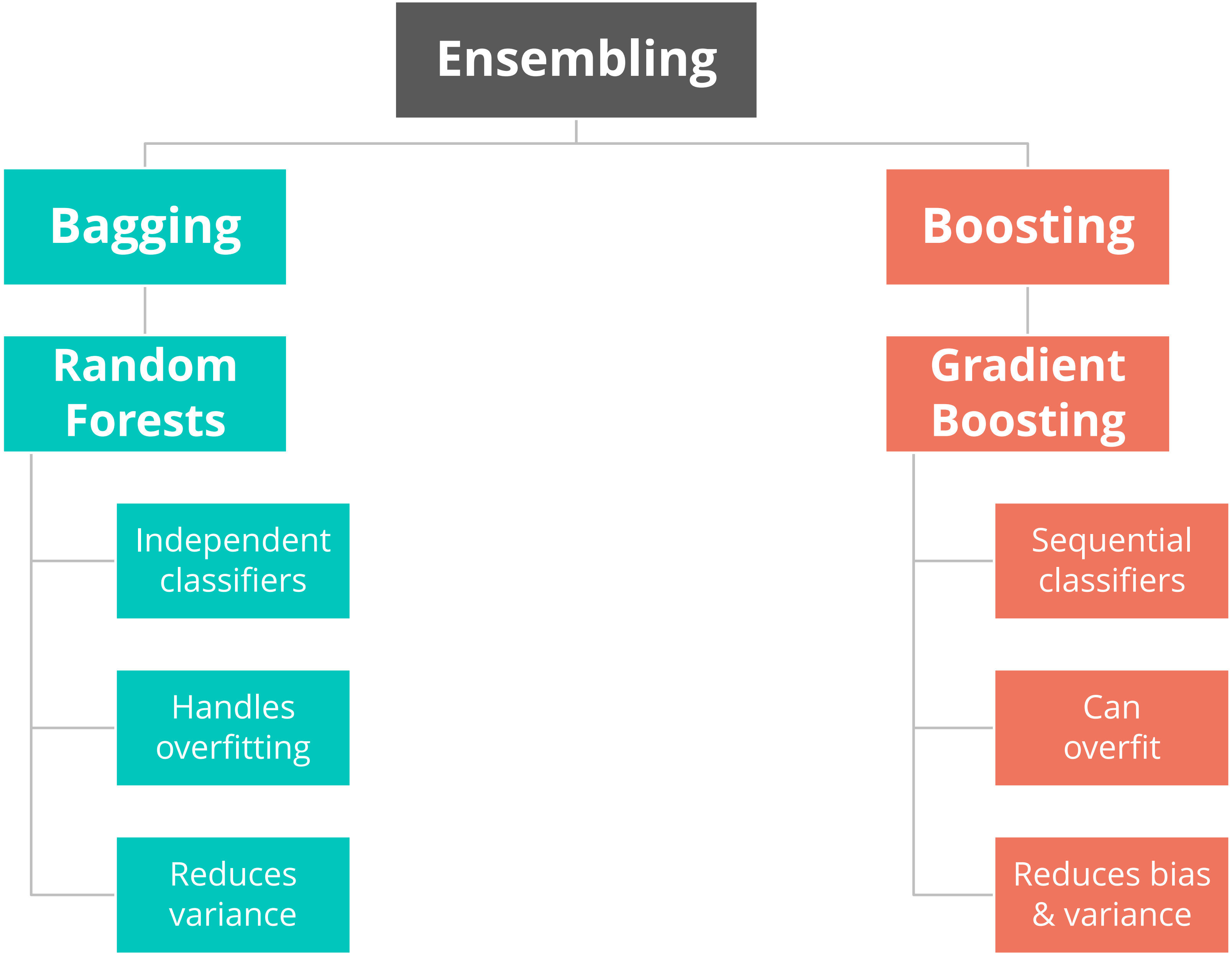
Only used if `n_iter_no_change` is set to an integer.

```
class sklearn.ensemble.GradientBoostingClassifier (
        loss='deviance',
        learning_rate=0.1,
        n_estimators=100,
        subsample=1.0,
        criterion='friedman_mse',
        min_samples_split=2,
        min_samples_leaf=1,
        min_weight_fraction_leaf=0.0,
        max_depth=3,
        min_impurity_decrease=0.0,
        min_impurity_split=None,
        init=None,
        random_state=None,
        max_features=None,
        verbose=0,
        max_leaf_nodes=None,
        warm_start=False,
        presort='auto',
        validation_fraction=0.1,
        n_iter_no_change=None,
        tol=0.0001)
```

`n_iter_no_change` is used to decide if **early stopping** will be used to terminate training when validation score is not improving.



https://scikit-learn.org/stable/auto_examples/ensemble/plot_gradient_boosting_early_stopping.html



https://medium.com/mlreview/gradient-boosting-from-scratch-1e317ae4587d

Further Reading: Gradient Boosting

Gradient boosting: Distance to target

Terence Parr and Jeremy Howard

https://explained.ai/gradient-boosting/L2-loss.html

3.2.4.3.5. `sklearn.ensemble`.GradientBoostingClassifier

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

Boosting algorithm: AdaBoost

SauceCat Follow

Apr 29, 2017 · 6 min read

https://towardsdatascience.com/boosting-algorithm-adaboost-b6737a9ee60c

XGBoost Documentation

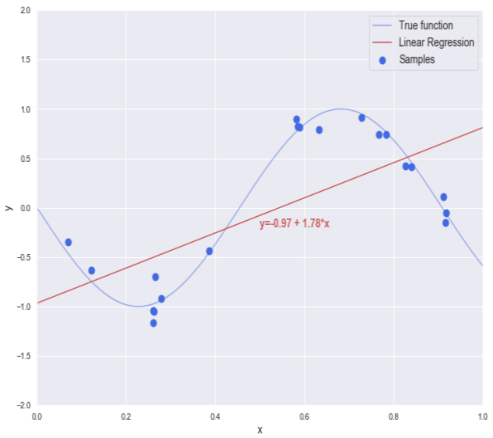
https://xgboost.readthedocs.io/en/latest/

Gradient Boosting Machine Learning

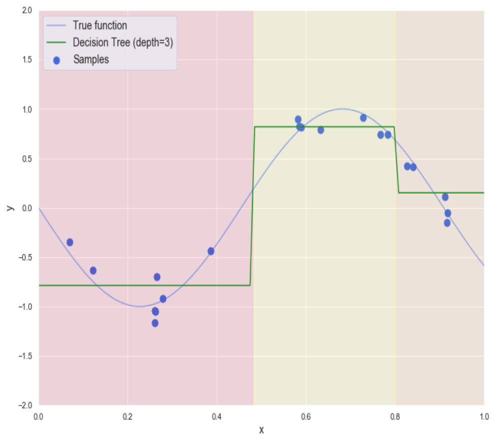
By Trevor Hastie

Summary

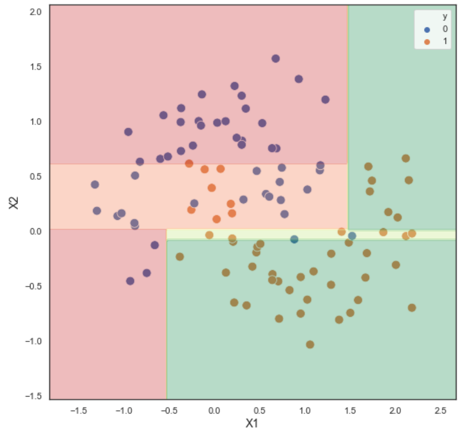
Linear Regression



Regression Trees

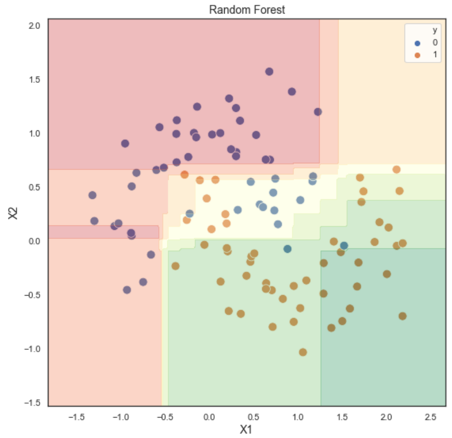


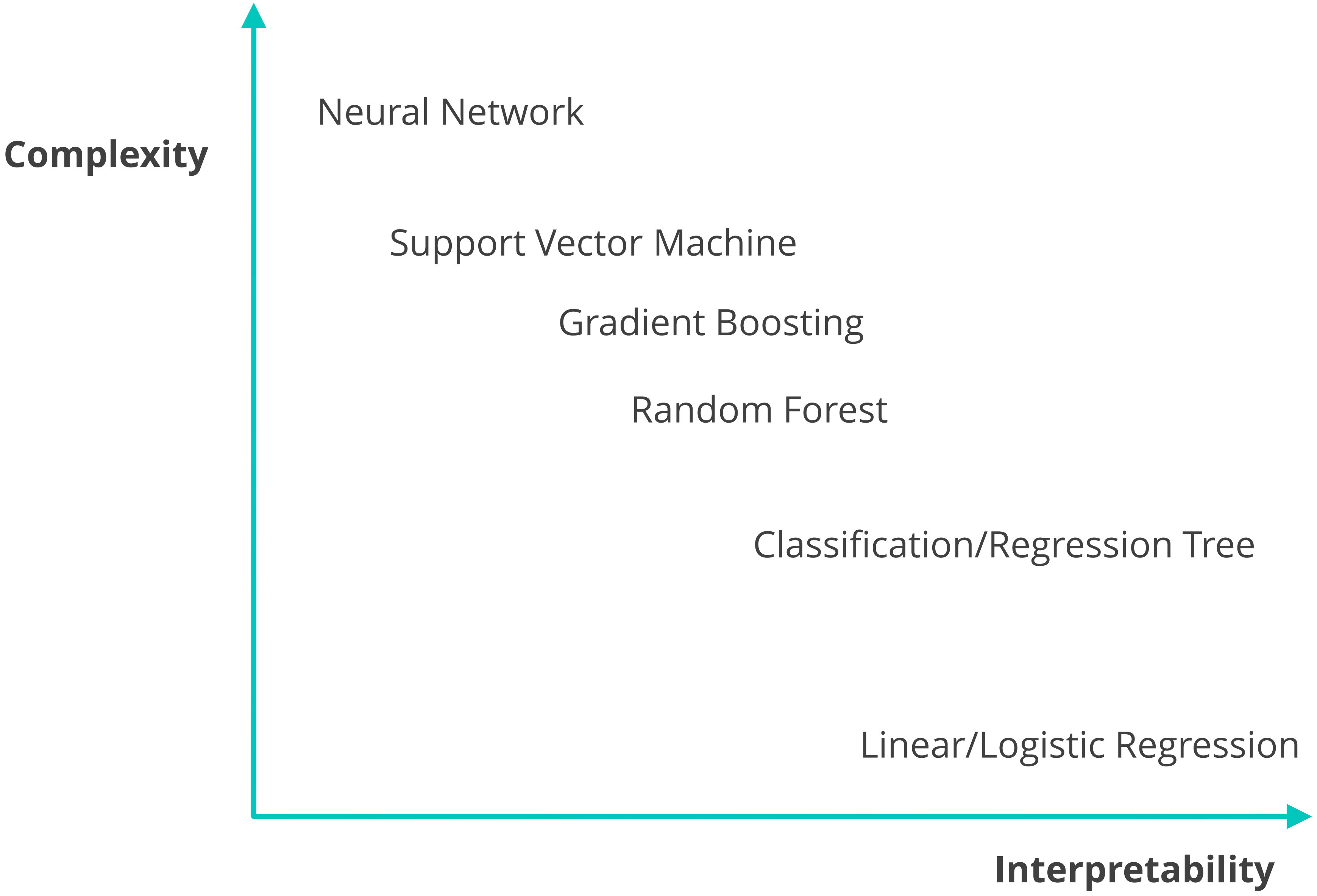
Classification Trees



Ensemble Trees

Random Forests & Gradient Boosting




Complexity
Neural Network
Support Vector Machine
Gradient Boosting
Random Forest
Classification/Regression Tree
Linear/Logistic Regression
Interpretability

Method	Advantages	Disadvantages
Linear / Logistic Regression	○ Model fit diagnostics ○ Interpretable coefficients, even for categorical predictors† ○ Tests for predictors	○ Simple/linear relationships ○ Unable to handle missing values ○ Binarize categorical predictors
Classification and Regression Trees	○ Categorical predictors† ○ Handles missing values and non-linear relationships ○ Visualization/interpretation ○ Variable importance	○ Prone to overfitting ○ Slow for categorical data with many levels
Random Forest & Gradient Boosting	○ Categorical predictors† & missing values ○ Variable importance ○ Controls overfitting (bias and/or variance)	○ No visualization ○ Slow for large datasets ○ Hyper-parameters is required
Support Vector Machine	○ Vary complexity by changing kernel/tuning parameters	○ Hard to visualize/interpret ○ Hyper-parameters is required ○ Binarize categorical predictors
Neural Network	○ Vary complexity by changing number of layers/tuning parameters ○ High accuracy	○ Hard to visualize/interpret, “black box” ○ Tuning parameters, stopping criteria ○ Binarize categorical predictors

Prof. Paul Brooks [DAPT631 Spring 2018]

† scikit-learn can’t handle categorical variables directly