Classification

ECE30007 Intro to AI Project

한동대학교
HANDONG GLOBAL UNIVERSITY

Contents

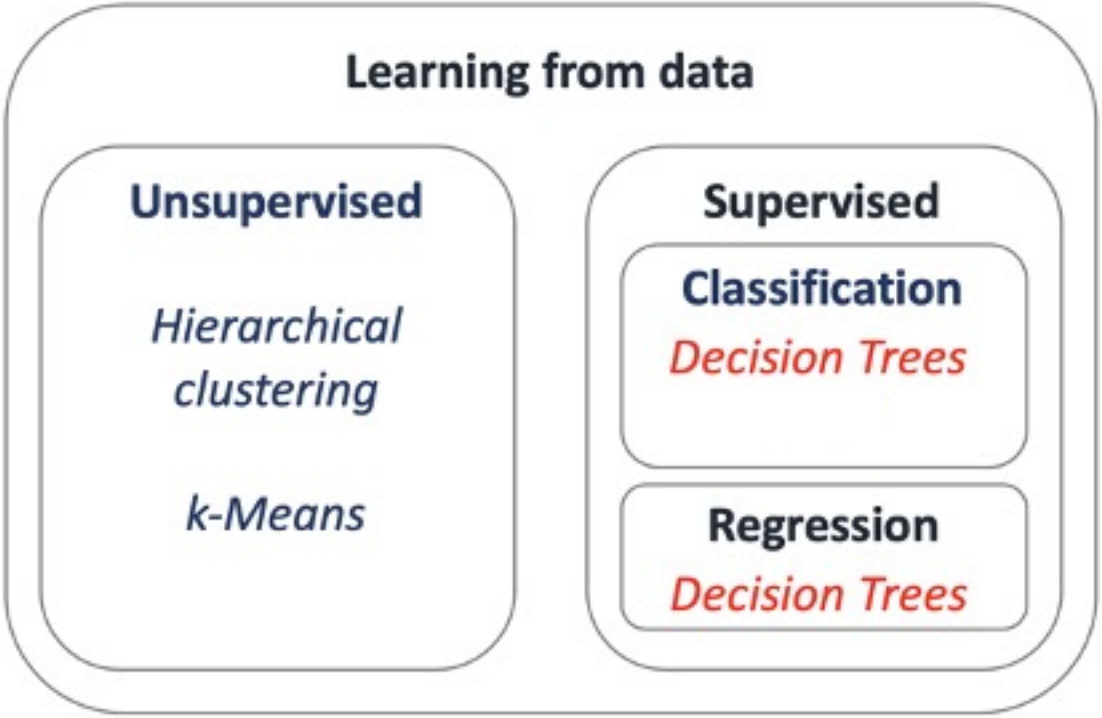
1. Recap on learning strategies
2. Decision Trees
3. Random Forests
4. Scikit-Learn with MNIST data

Learning Strategies

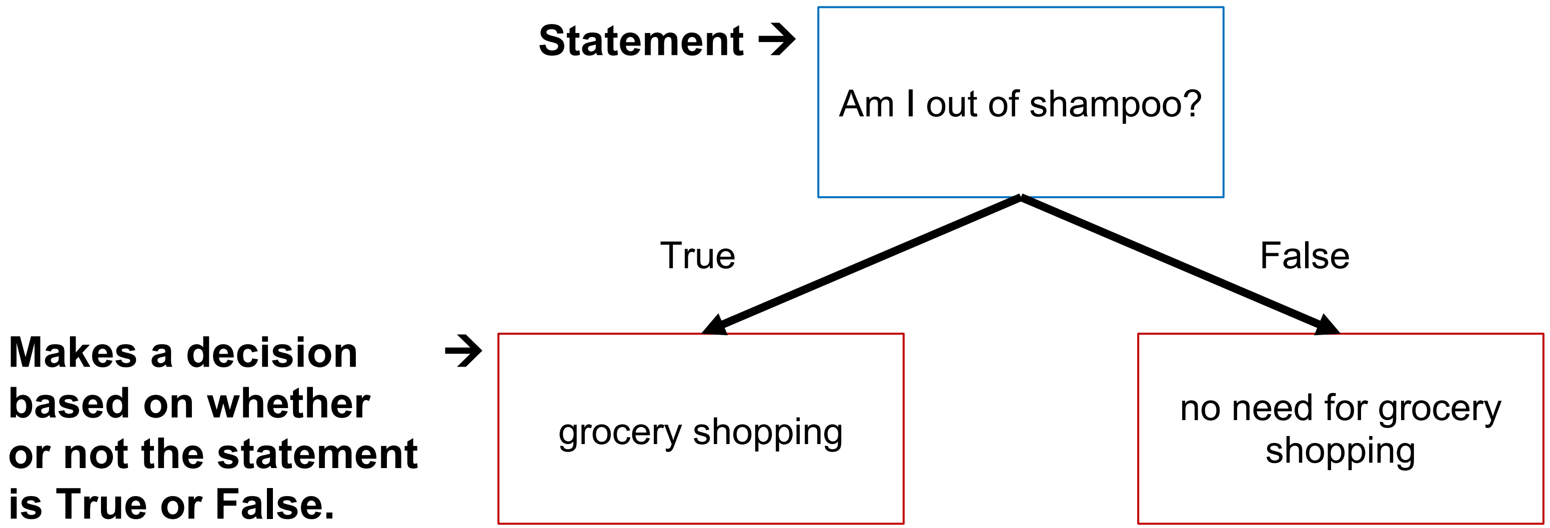
- Unsupervised learning
  - e.g., clustering and dimension reduction
- Supervised learning
  - **Classification**
    - Problems with categorical solutions like yes/no, apple/orange/mango
  - **Regression**
    - Problems wherein continuous value needs to be predicted, such as ”product price” or “profits”

Decision Trees

- A member in supervised learning

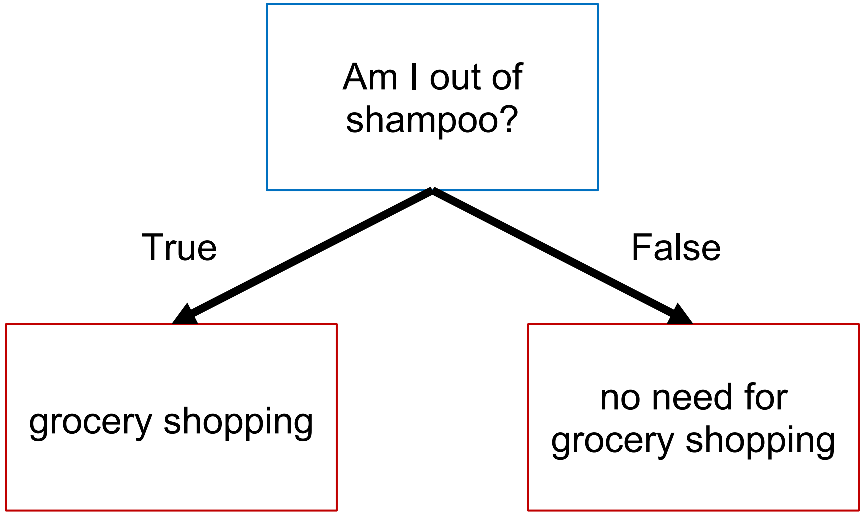


Example of Decision Tree

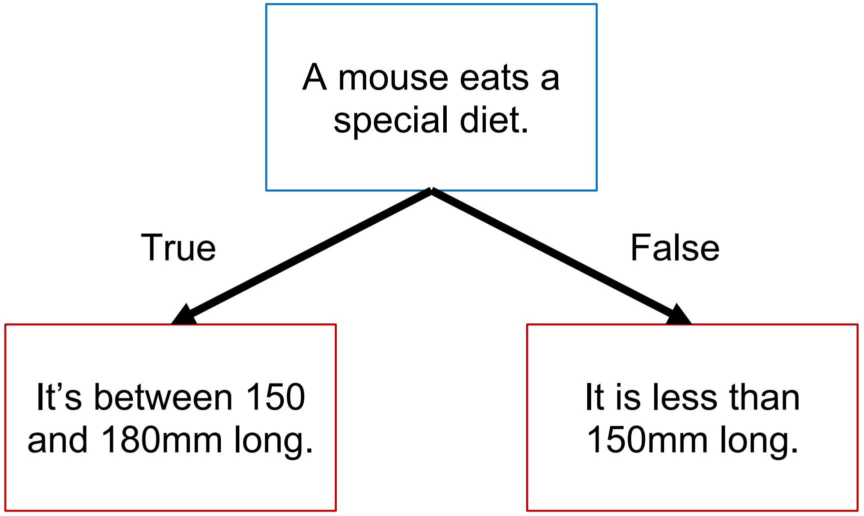


Decision Tree for classification/regression

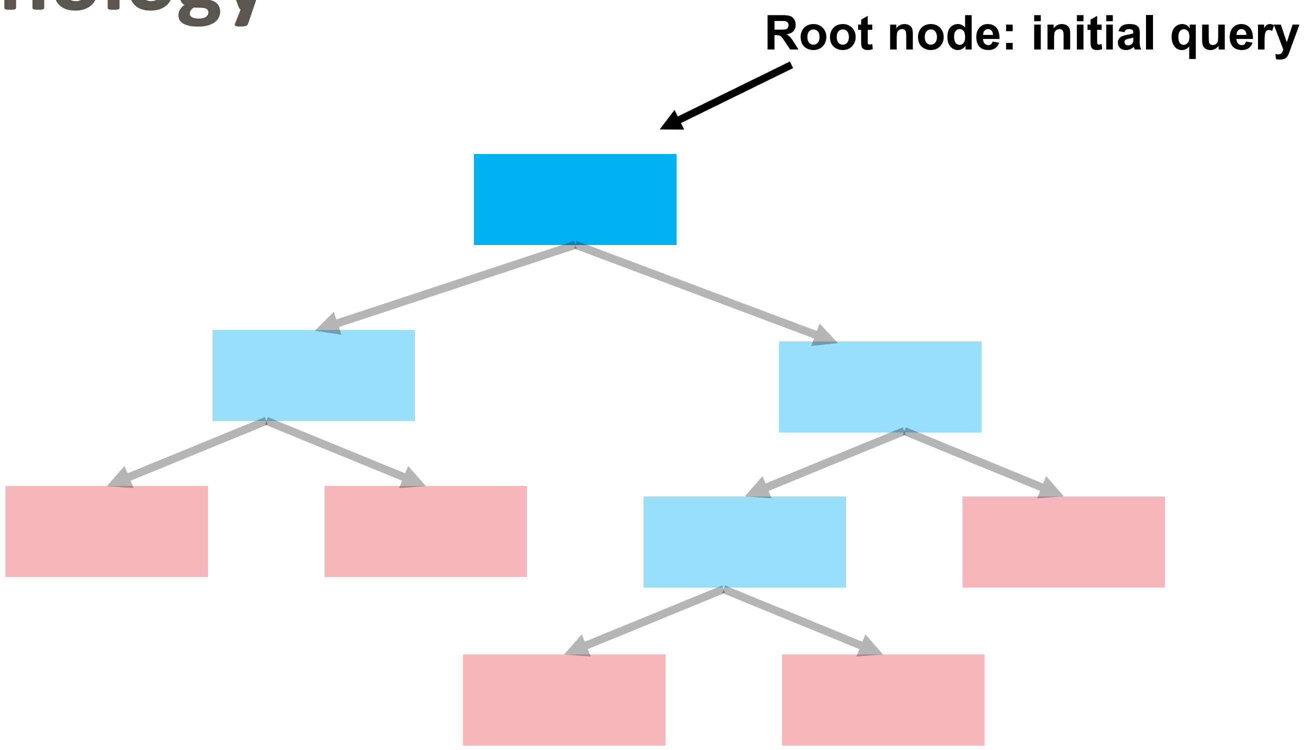
- Classification Tree



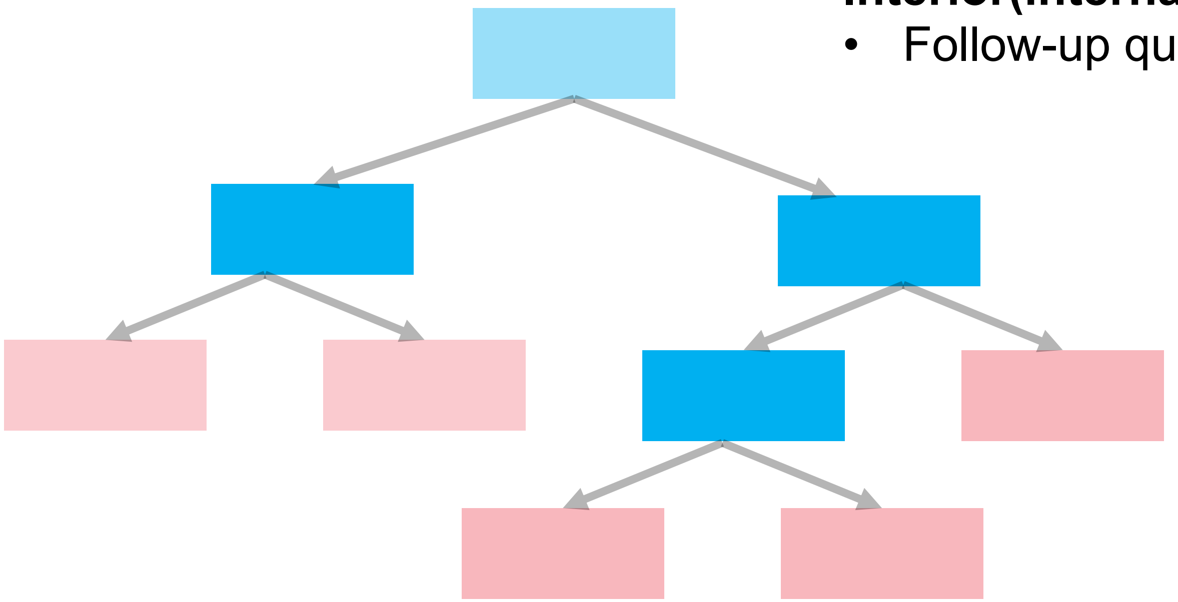
- Regression Tree



Terminology



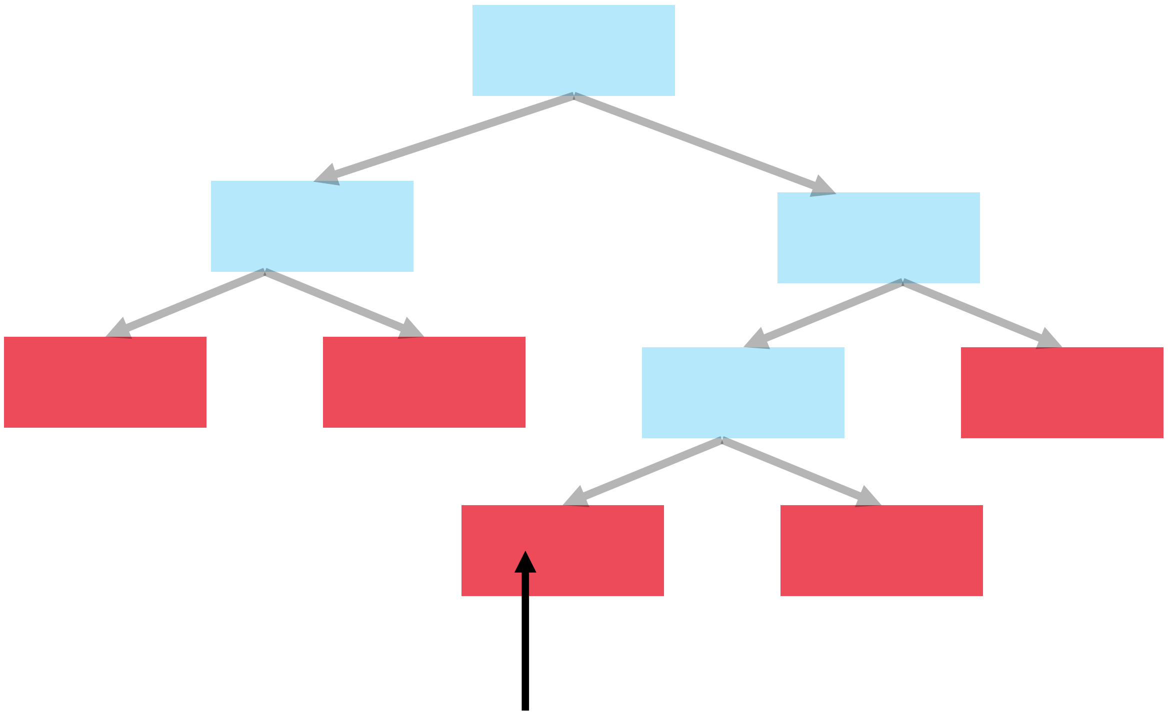
Terminology



Interior(internal) nodes

- Follow-up queries

Terminology



Leaf nodes

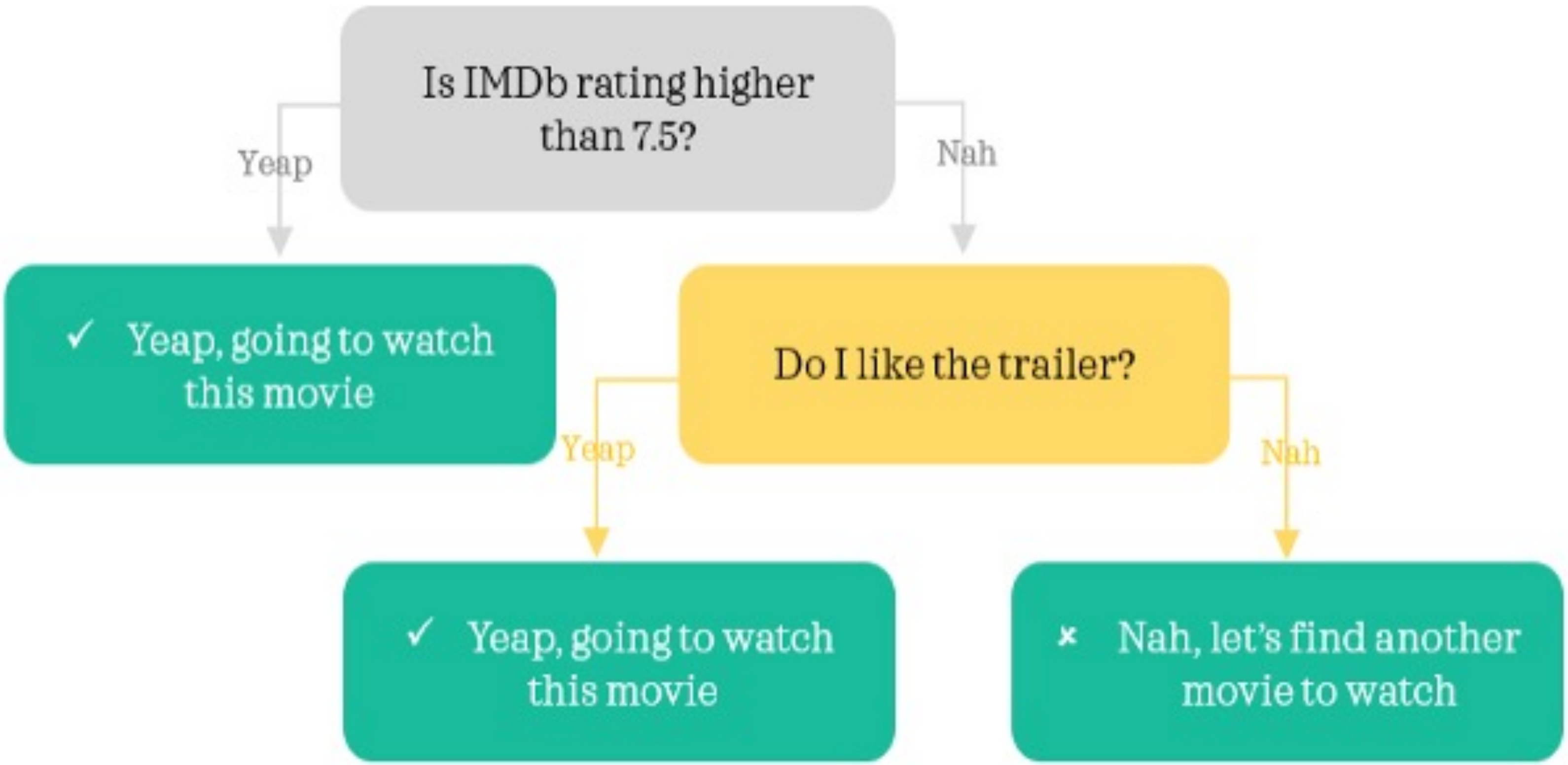
- Decision nodes
- Carry out predictions

Key questions

- **Prediction**
  - Given a decision tree, how to use it to make predictions?
- **Training (Learning)**
  - How to train (learn) a decision tree from data?
  - Which questions to ask, and when?

Prediction

Should I watch this video?



https://towardsdatascience.com/how-are-decision-trees-built-a8e5af57ce8

Prediction

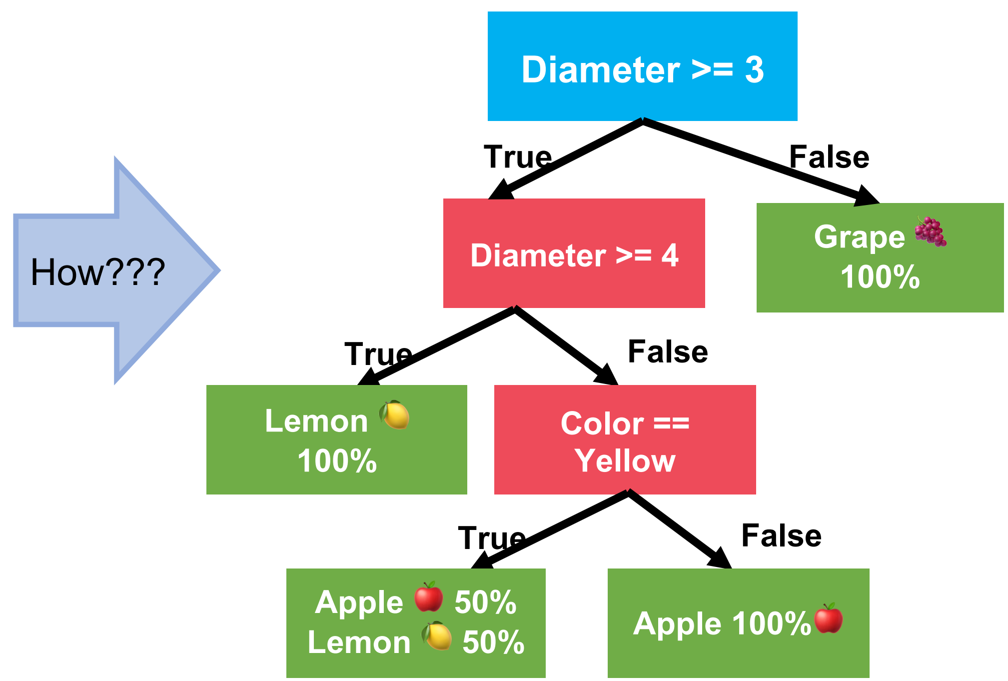
- A decision tree
  - Assuming that a trained decision tree is given
    - For each data instance, answer to the query on each node and take the branch with the answer
    - When the instance arrives at a leaf node, make a prediction according to the node's decision

  - This could be efficiently implemented as a recursive program

Training

- Training data example

Color	Diameter	Label
Green	3	Apple
Yellow	3	Apple
Red	1	Grape
Red	1	Grape
Yellow	3	Lemon
Yellow	4	Lemon



https://www.youtube.com/watch?v=LDRbO9a6XPU

Training

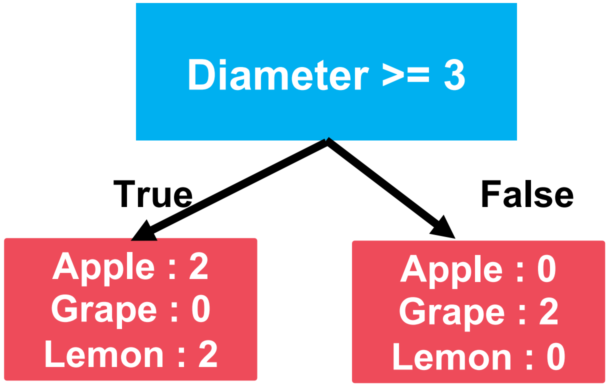
- Idea
  - Start from an empty decision tree, with all train data
  - Split on next best attribute(feature)
    - Use, for example, information gain to decide the splitting condition

How to make splits?
How to split the data into multiple nodes?

Training example

assuming that diameter is the best feature now.

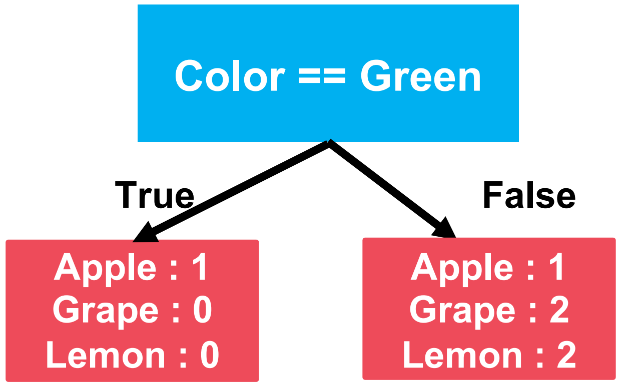
Color	Diameter	Label
Green	3	Apple
Yellow	3	Apple
Red	1	Grape
Red	1	Grape
Yellow	3	Lemon
Yellow	4	Lemon



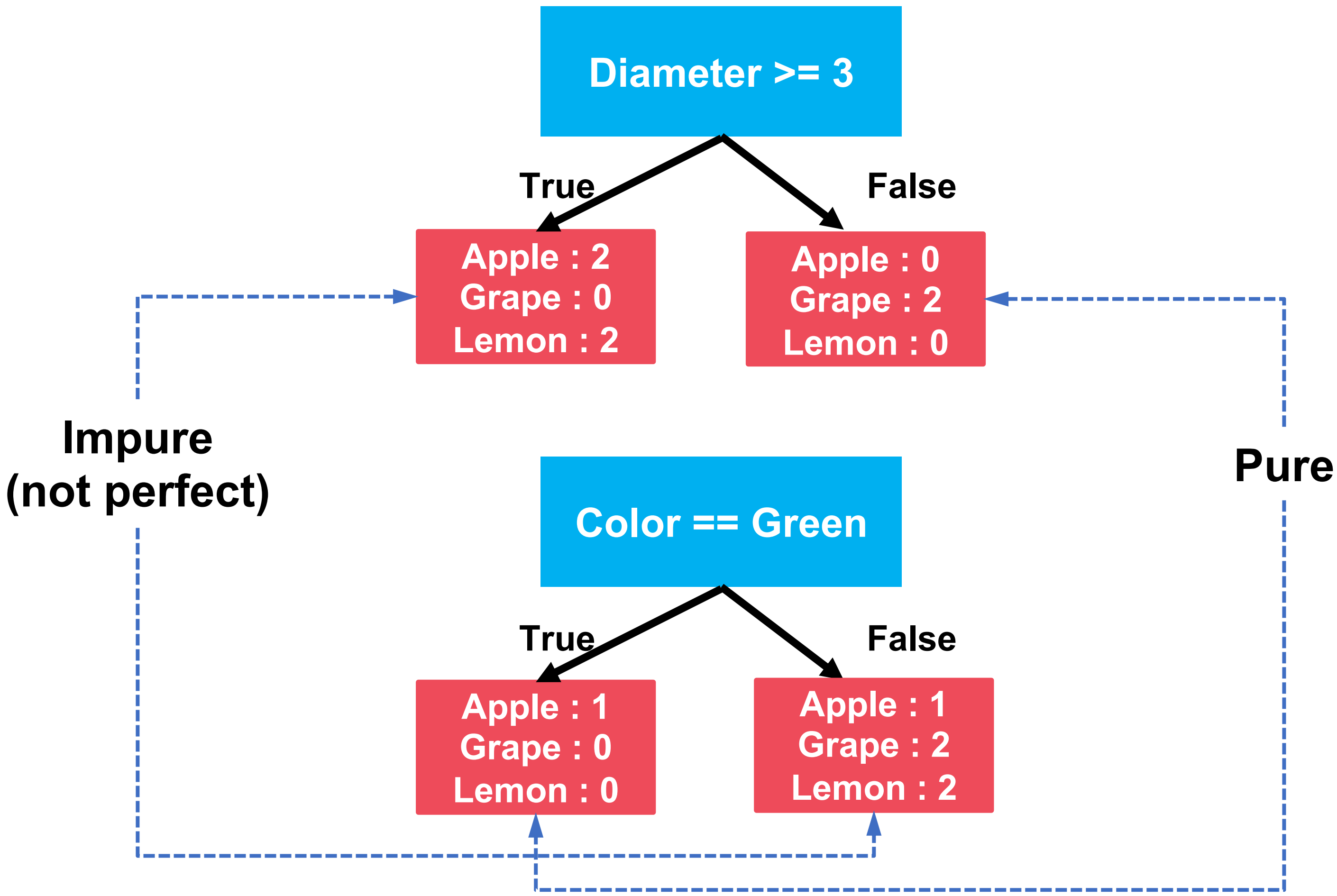
Training example

assuming that color is the best feature now.

Color	Diameter	Label
Green	3	Apple
Yellow	3	Apple
Red	1	Grape
Red	1	Grape
Yellow	3	Lemon
Yellow	4	Lemon



Training example



How to find the best feature?

- **Entropy**: degree of complexity in data
  - $H(Y) = E[I(Y)] = E[-\log_2 P(Y)] = -\sum_i P(y_i) \log_2 P(y_i)$
  - High/low entropy: less/more predictable
- Conditional Entropy of Y given X
  - $H(Y|X) = \sum_j p(x_j) \left(-\sum_i p(y_i|x_j) \log_2 p(y_i|x_j)\right)$

- Splitting criterion: Information gain
  - Decrease in entropy (uncertainty) after splitting a node
  - IG(X) = H(Y) – H(Y|X)

Exercise(1) – Toy dataset

STEP ☝️. Make a toy dataset

```
[ ]  ## Toy dataset
     train_data = [
         ['Green', 3, 'Apple'],
         ['Yellow', 3, 'Apple'],
         ['Red', 1, 'Grape'],
         ['Red', 1, 'Grape'],
         ['Yellow', 3, 'Lemon'],
         ['Yellow', 4, 'Lemon']
     ]
```

```
[ ]  columns = ['color', 'diameter', 'label']
```

Exercise(2) – Useful functions

```
def unique_vals(rows, col):
  '''
  rows: list of row data
  col: the index of column that we want to find
  '''
  return set([row[col] for row in rows])
```

```
[8] unique_vals(train_data, 0)

    {'Green', 'Red', 'Yellow'}
```

```
def class_counts(rows):
    """Counts the number of each type of example in a dataset.
    rows: list of row data
    """
    counts = {}  # a dictionary of label -> count.

    for row in rows:
        # in our dataset format, the label is always the last column
        label = row[-1]

        if label not in counts:
            counts[label] = 0
        counts[label] += 1
    return counts
```

```
class_counts(train_data)

{'Apple': 2, 'Grape': 2, 'Lemon': 2}
```

Exercise(2) – Useful functions

```
def is_numeric(value):
  return isinstance(value, int) or isinstance(value, float)
```

```
print(is_numeric(7), is_numeric('Blue'))
```

```
True False
```

Exercise(3) – Question class

we need one question on each node

```
[106] class Question:
        ## Constructor
        def __init__(self, column, value):
          self.column = column
          self.value = value

        def compare_with_question(self, example):
          '''
          Arguments:
          example -- List of row data (EX. ['Blue', 2, 'Blueberry'])
          '''
          val = example[self.column]

          if is_numeric(val):
            return val >= self.value
          else:
            return val == self.value

        ## Python __repr__() function returns the object representation in string format.
        def __repr__(self):
          condition = "=="

          if is_numeric(self.value):
            condition = ">="
          return "Is {} {} {}?".format(columns[self.column], condition, str(self.value))
```

Question(1, 3) means
”is Diameter greater than 3?

we need this function to split the rows
based on the question

Exercise(3) – Question class

▾ 💁 : Let's implement **Quesetion** class!

```
[107] Question(1, 3)

      Is diameter >= 3?

[108] Question(0, 'Green')

      Is color == Green?
```

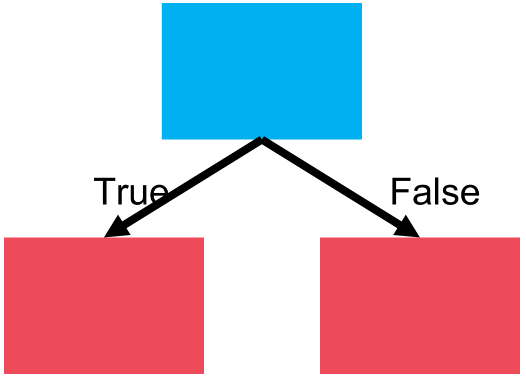
Color	Diameter	Label
Green	3	Apple
Yellow	3	Apple
Red	1	Grape
Red	1	Grape
Yellow	3	Lemon
Yellow	4	Lemon

Exercise(4) – Partition function

split the data based on the question

```
def partition(rows, question):
  """Partitions a dataset
  Arguments:
  rows -- List of row data
  question -- An object of Question class
  """

  true_rows, false_rows = [], []

  for row in rows:
    if question.compare_with_question(row):
      true_rows.append(row)
    else:
      false_rows.append(row)

  return true_rows, false_rows
```

Partition



Color	Diameter	Label
Green	3	Apple
Yellow	3	Apple
Red	1	Grape
Red	1	Grape
Yellow	3	Lemon
Yellow	4	Lemon

Exercise(4) – Partition function

- 💁 : If we partition **'train_data'** with a question "Is color == Red🖍", what is the true_rows and false_rows?

```
[26] true_rows, false_rows = partition(train_data, Question(0, 'Red'))
```

```
[27] print("The true_rows\n ===> ", true_rows, "\nThe false_rows\n ===> ", false_rows)
```

```
The true_rows
 ===>  [['Red', 1, 'Grape'], ['Red', 1, 'Grape']]
The false_rows
 ===>  [['Green', 3, 'Apple'], ['Yellow', 3, 'Apple'], ['Yellow', 3, 'Lemon'], ['Yellow', 4, 'Lemon']]
```

Color	Diameter	Label
Green	3	Apple
Yellow	3	Apple
Red	1	Grape
Red	1	Grape
Yellow	3	Lemon
Yellow	4	Lemon

Exercise(5) – Entropy calculation

```
def entropy(labels, base=None):
  """ Computes entropy of label distribution.
  Arguments:
  labels -- Lists of data's label
  """

  n_labels = len(labels)

  if n_labels <= 1:
    return 0

  counts = class_counts(labels) # return "dict{class_label : counts}"
  probs = [counts[key]/n_labels for key in counts.keys()]

  n_classes = np.count_nonzero(probs)

  if n_classes <= 1:
    return 0

  ent = 0.

  # Compute entropy
  # base = e if base is None else base
  for i in probs:
    ent -= i * log2(i)

  return ent
```

implemenation of entropy might be beyond this class.
but please understand how to use it
in the following slides

$$-\sum_i P(y_i)\log_2 P(y_i)$$

Exercise(5) – Entropy calculation

💁 : In ***pure dataset***, how much entropy value did you get?

```
[22] ## Pure case
     pure = [['Apple'],
             ['Apple']]

     # this will return 0
     entropy(pure)
```

```
0
```

💁 : In ***impure dataset***, how much entropy value did you get?

```
[23] ## Impure case
     impure = [['Apple'],
               ['Orange']]

     entropy(impure)
```

```
1.0
```

Exercise(5) – Entropy calculation

```
# Now, we'll look at a dataset with many different labels
impure = [['Apple'],
          ['Orange'],
          ['Grape'],
          ['Grapefruit'],
          ['Blueberry']]

# This will return 2.32
entropy(impure)

2.321928094887362
```

Exercise(6) – Information Gain

```python
def info_gain(left, right, current_uncertainty):
    """Information Gain.

    IG = The uncertainty of the starting node - the weighted impurity of two child nodes.
    """
    p = float(len(left)) / (len(left) + len(right))

    print("(1) Avg of Impurity = {:.4f} * {:.4f} + {:.4f} * {:.4f}".format(p, entropy(left),
                                                                      (1-p), entropy(right)))
    print("(2) Current uncertainty = {:.4f}".format(current_uncertainty))

    IG = current_uncertainty - (p * entropy(left) + (1 - p) * entropy(right))
    print("(3) Information gain = {:.4f} - ({:.4f} * {:.4f} + {:.4f} * {:.4f})= {:.4f}\n".
          format(current_uncertainty, p, entropy(left), 1-p, entropy(right), IG))

    return IG
```

$$\boxed{H(Y) = -\sum_i p(y_i)\log_2 p(y_i)}$$

$$\boxed{H(Y|X) = \sum_j p(x_j)\left(-\sum_i p(y_i|x_j)\log_2 p(y_i|x_j)\right)}$$

entropy at the current node

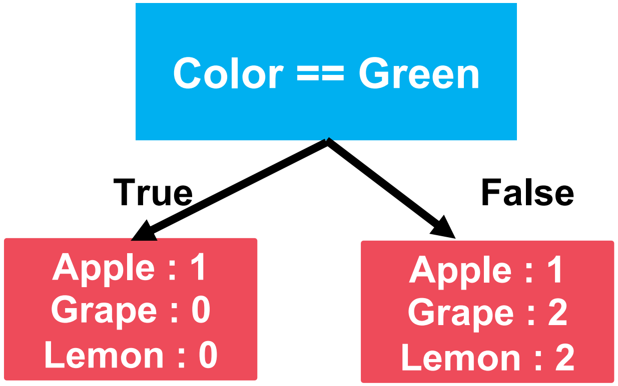
```
[26] current_uncertainty = entropy(train_data)
     print("\nCurrent uncertainty ===> {:.4f}".format(current_uncertainty))
```

```
Current uncertainty ===> 1.5850
```

Color	Diameter	Label
Green	3	Apple
Yellow	3	Apple
Red	1	Grape
Red	1	Grape
Yellow	3	Lemon
Yellow	4	Lemon

Current uncertainty $= -(\frac{1}{3}\log_2\frac{1}{3} + \frac{1}{3}\log_2\frac{1}{3} + \frac{1}{3}\log_2\frac{1}{3})$

$= 1.5850$



Exercise(6) – Information Gain

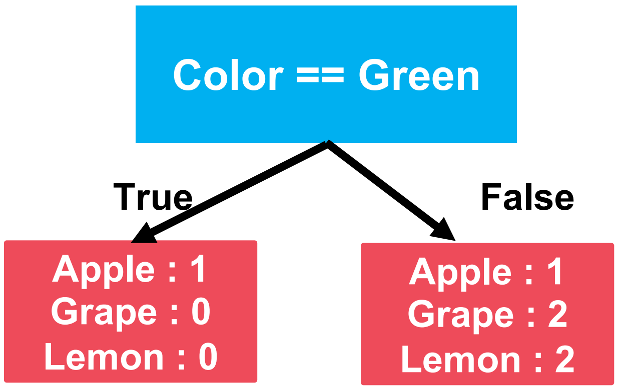
▾ 💁 : How much information do we gain by partitioning on 'Green'?

```
[44] print('Question? ', Question(0, 'Green'))

     true_rows, false_rows = partition(train_data, Question(0, 'Green'))
     print("True_rows ===> {}".format(true_rows))
     print("False rows ===> {}\n".format(false_rows))

     print("\nThe information gain by partitioning on \"Green\" is {:.4f}".format(
         info_gain(true_rows, false_rows, current_uncertainty)))
```

```
Question?  Is color == Green?
True_rows ===> [['Green', 3, 'Apple']]
False rows ===> [['Yellow', 3, 'Apple'], ['Red', 1, 'Grape'], ['Red', 1, 'Grape'], ['Yellow', 3, 'Lemon'], ['Yellow', 4, 'Lemon']]

(1) Avg of Impurity = 0.1667 * 0.0000 + 0.8333 * 1.5219
(2) Current uncertainty = 1.5850
(3) Information gain = 1.5850 - (0.1667 * 0.0000 + 0.8333 * 1.5219)= 0.3167

The information gain by partitioning on "Green" is 0.3167
```

information gain for “color==Green”

Current uncertainty = 1.5850



Impurity of True = 0

Impurity of False = 1.5219

Color	Diameter	Label
Green	3	Apple
Yellow	3	Apple
Red	1	Grape
Red	1	Grape
Yellow	3	Lemon
Yellow	4	Lemon

Avg Impurity $= \frac{1}{6} \times 0 + \frac{5}{6} \times 1.5219 = 1.268$

Information gain $= 1.5850 - 1.268 = $ **0.3167**

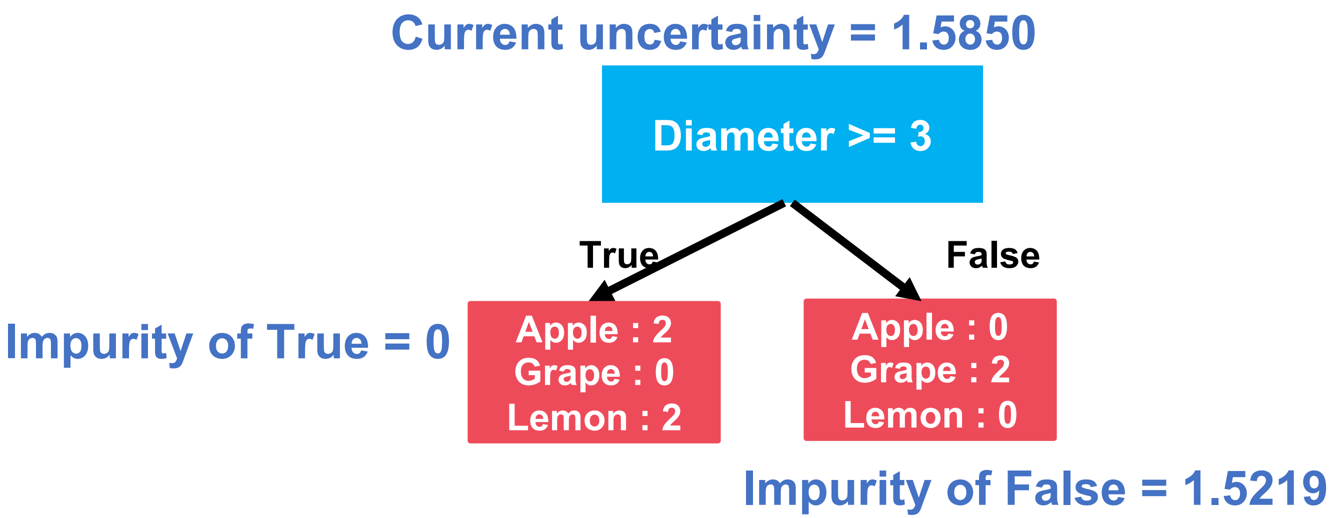
Exercise(6) – Information Gain

💁: How much information do we gain by partitioning on **diameter >= 3**?

```
[117] print('Question? ', Question(1, 3))

      true_rows, false_rows = partition(train_data, Question(1, 3))
      print("True_rows ===> {}".format(true_rows))
      print("False rows ===> {}\n".format(false_rows))

      print("\nThe information gain by partitioning on \"Green\" is {:.4f}".format(
          info_gain(true_rows, false_rows, current_uncertainty)))
```

```
Question?  Is diameter >= 3?
True_rows ===> [['Green', 3, 'Apple'], ['Yellow', 3, 'Apple'], ['Yellow', 3, 'Lemon'], ['Yellow', 4, 'Lemon']]
False rows ===> [['Red', 1, 'Grape'], ['Red', 1, 'Grape']]

(1) Avg of Impurity = 0.6667 * 1.0000 + 0.3333 * 0.0000
(2) Current uncertainty = 1.5850
(3) Information gain = 1.5850 - (0.6667 * 1.0000 + 0.3333 * 0.0000)= 0.9183

The information gain by partitioning on "diameter >= 3" is 0.9183
```

information gain for “Diameter >= 3”



Color	Diameter	Label
Green	3	Apple
Yellow	3	Apple
Red	1	Grape
Red	1	Grape
Yellow	3	Lemon
Yellow	4	Lemon

Avg Impurity $= \frac{4}{6}\times 1 + \frac{2}{6}\times 0 = 0.666$

Information gain = 1.5850 – 0.666 = **0.9183**

Exercise(7) – Find the best split!

```
[129] def find_best_split(rows):

        best_gain = 0
        best_question = None

        current_uncertainty = entropy(rows)
        n_features = len(rows[0]) - 1

        for col in range(n_features):

          values = set([row[col] for row in rows])

          for val in values:

            question = Question(col, val)

            true_rows, false_rows = partition(rows, question)

            if len(true_rows) == 0 or len(false_rows) == 0:
              continue

            print('Qustion ====>>> ', question)
            gain = info_gain(true_rows, false_rows, current_uncertainty)

            if gain >= best_gain:
              best_gain, best_question = gain, question


        return best_gain, best_question
```

for all features

for all values on the feature

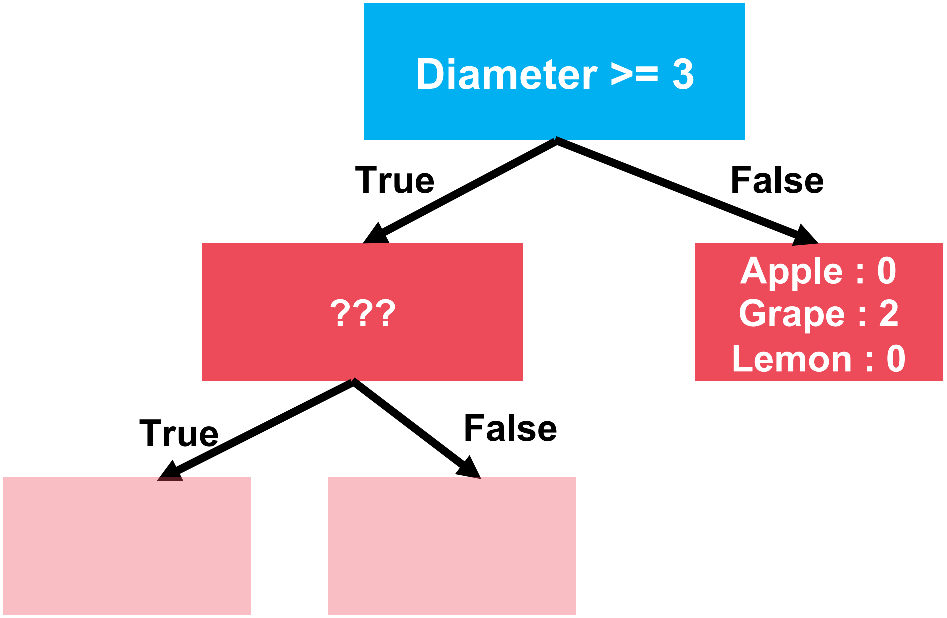
Exercise(7) – Find the best split!

```
Qustion ====>>>  Is color == Green?
(1) Avg of Impurity = 0.1667 * 0.0000 + 0.8333 * 1.5219
(2) Current uncertainty = 1.5850
(3) Information gain = 1.5850 - (0.1667 * 0.0000 + 0.8333 * 1.5219)= 0.3167

Qustion ====>>>  Is color == Yellow?
(1) Avg of Impurity = 0.5000 * 0.9183 + 0.5000 * 0.9183
(2) Current uncertainty = 1.5850
(3) Information gain = 1.5850 - (0.5000 * 0.9183 + 0.5000 * 0.9183)= 0.6667

Qustion ====>>>  Is color == Red?
(1) Avg of Impurity = 0.3333 * 0.0000 + 0.6667 * 1.0000
(2) Current uncertainty = 1.5850
(3) Information gain = 1.5850 - (0.3333 * 0.0000 + 0.6667 * 1.0000)= 0.9183

Qustion ====>>>  Is diameter >= 3?
(1) Avg of Impurity = 0.6667 * 1.0000 + 0.3333 * 0.0000
(2) Current uncertainty = 1.5850
(3) Information gain = 1.5850 - (0.6667 * 1.0000 + 0.3333 * 0.0000)= 0.9183

Qustion ====>>>  Is diameter >= 4?
(1) Avg of Impurity = 0.1667 * 0.0000 + 0.8333 * 1.5219
(2) Current uncertainty = 1.5850
(3) Information gain = 1.5850 - (0.1667 * 0.0000 + 0.8333 * 1.5219)= 0.3167

The best question ====>>>>>  Is diameter >= 3?
```

The information gain of 'Is diameter >= 3' is the biggest one.

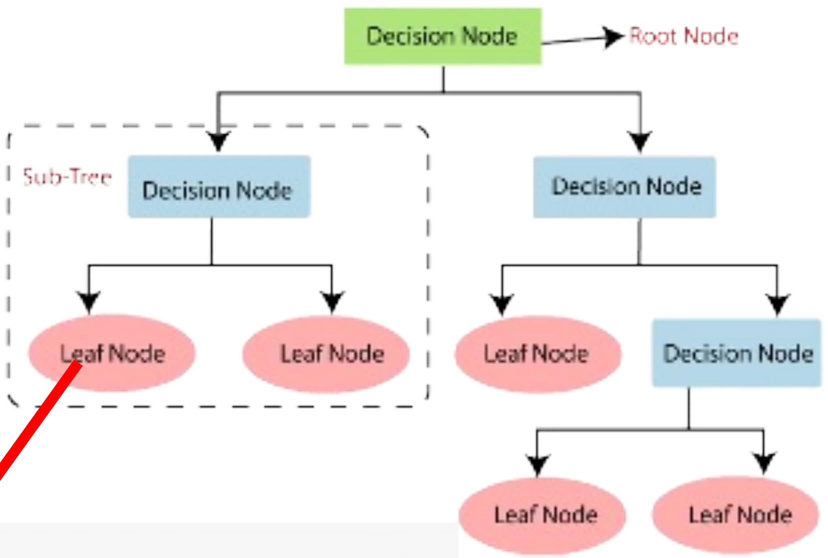
recursive training

Color	Diameter	Label
Green	3	Apple
Yellow	3	Apple
Red	1	Grape
Red	1	Grape
Yellow	3	Lemon
Yellow	4	Lemon



- Training continuously based on ‘True’ row data of statement ‘Diameter >= 3?’

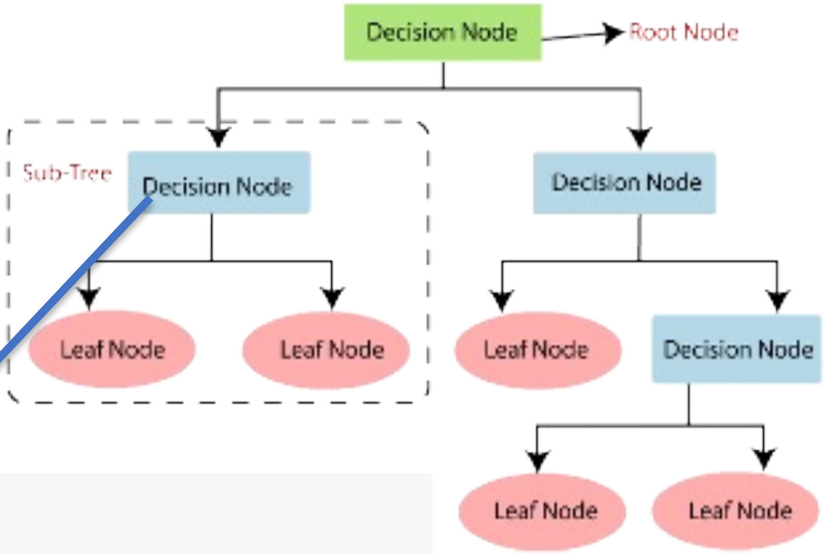
Exercise(8) – Make a Decision Tree!




```
[145] class Leaf:
          """A Leaf node classifies data.

          This holds a dictionary of class (e.g., "Apple") -> number of times
          it appears in the rows from the training data that reach this leaf.
          """

          def __init__(self, rows):
            ## 'self.predictions' is a dictionary of class counts.
            self.predictions = class_counts(rows)
```

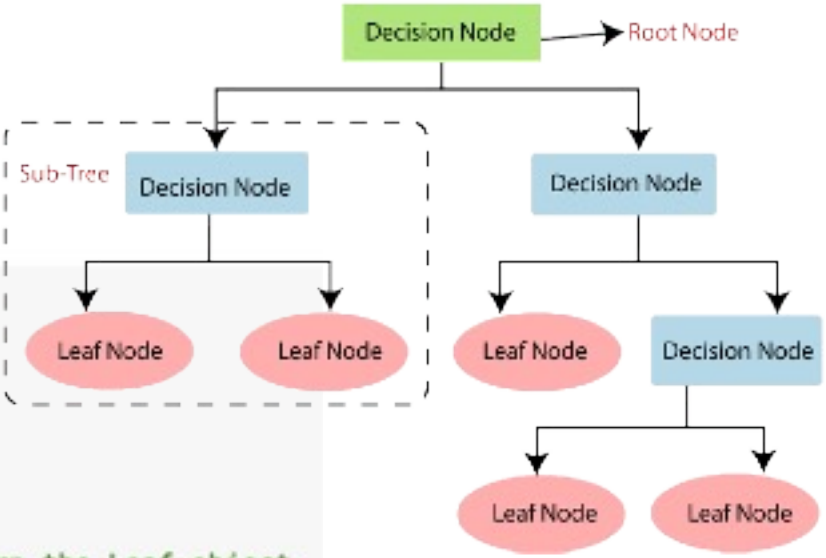
in Exercise(2)

Exercise(8) – Make a Decision Tree!




```
[146] class Decision_Node:
        """A Decision Node asks a question.

          This holds a reference to the question, and to the two child nodes.
          """

        def __init__(self,
                     question,
                     true_branch,
                     false_branch):
          self.question = question
          self.true_branch = true_branch
          self.false_branch = false_branch
```

Exercise(8) – Make a Decision Tree!




```
[147] def build_tree(rows):
        """Builds the tree.
        Arguments:
        rows --- List of row data
        """
        gain, question = find_best_split(rows)

        ## If information gain is equal to 0, just return the Leaf object.
        if gain == 0:
          return Leaf(rows)

        true_rows, false_rows = partition(rows, question)

        ## Make additional tree nodes
        true_branch = build_tree(true_rows)

        false_branch = build_tree(false_rows)

        return Decision_Node(question, true_branch, false_branch)
```

Exercise(8) – Make a Decision Tree!

```
[148] def print_tree(node, spacing=""):
          """Tree printing function."""

          # Base case: we've reached a leaf
          if isinstance(node, Leaf):
              print (spacing + "Predict", node.predictions)
              return

          # Print the question at this node
          print (spacing + str(node.question))

          # Call this function recursively on the true branch
          print (spacing + '--> True:')
          print_tree(node.true_branch, spacing + "  ")

          # Call this function recursively on the false branch
          print (spacing + '--> False:')
          print_tree(node.false_branch, spacing + "  ")
```

Exercise(8) – Make a Decision Tree!

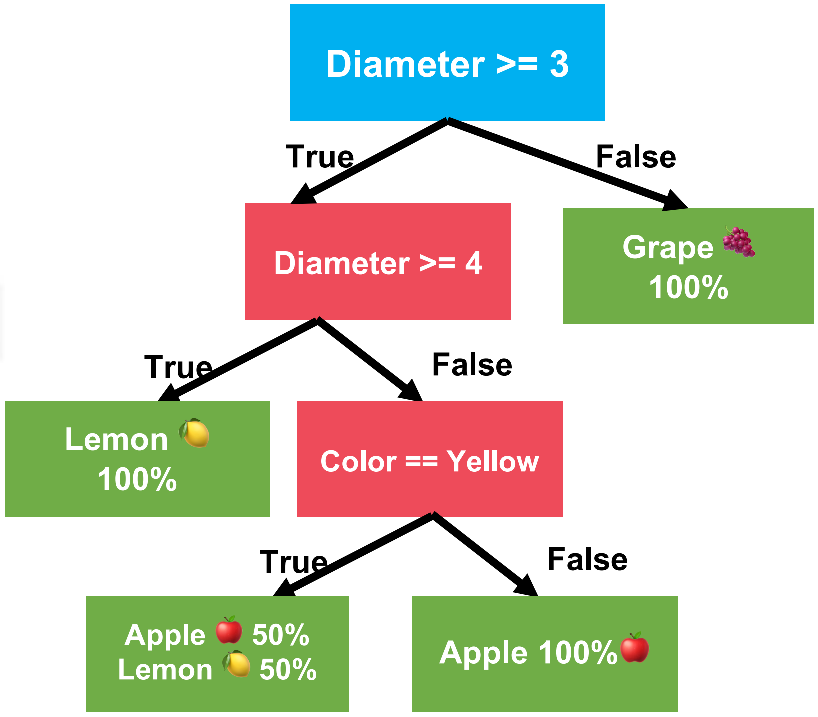
▾ 💁 : Let's build one *Decision Tree* !

```
[149] my_tree = build_tree(train_data)
      print(type(my_tree))
```

▾ 💁 : Let's print our ***Decision tree*** !

```
[150] print_tree(my_tree)
```

```
Is diameter >= 3?
--> True:
  Is diameter >= 4?
  --> True:
    Predict {'Lemon': 1}
  --> False:
    Is color == Yellow?
    --> True:
      Predict {'Apple': 1, 'Lemon': 1}
    --> False:
      Predict {'Apple': 1}
--> False:
  Predict {'Grape': 2}
```



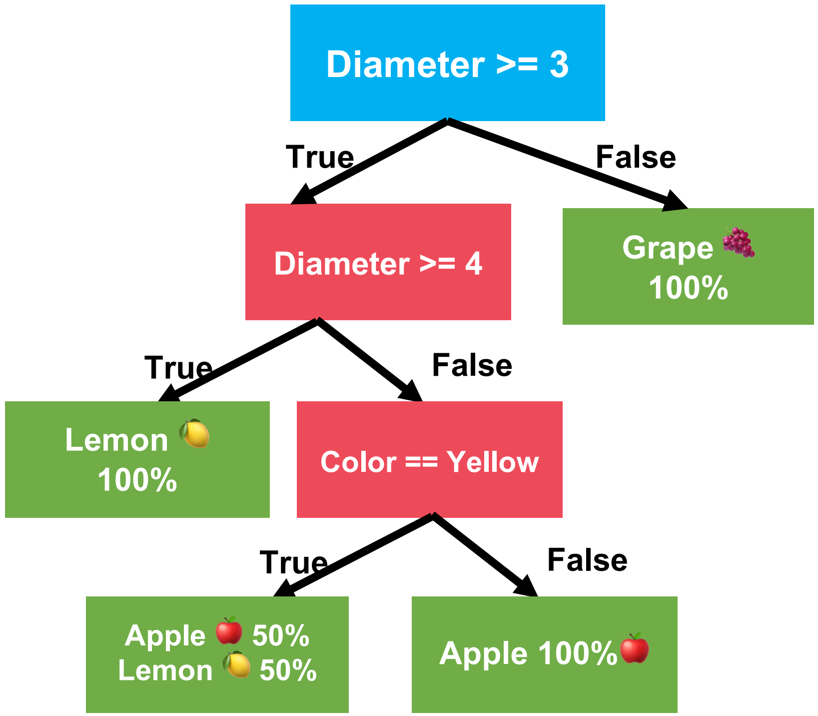
Exercise(8) – Make a Decision Tree!

Color	Diameter	Label
Green	3	Apple
Yellow	3	Apple
Red	1	Grape
Red	1	Grape
Yellow	3	Lemon
Yellow	4	Lemon

: Which results do you get when you classify ***'train_data[1]'***,?

```
[151] def classify(row, node):

        ## If this node is Leaf, return predicted results.
        if isinstance(node, Leaf):
          return node.predictions ## If you don't know what data

        if node.question.compare_with_question(row):
          return classify(row, node.true_branch)
        else:
          return classify(row, node.false_branch)
```

```
[155] ## Return the each class counts
      classify(train_data[0], my_tree)

      {'Apple': 1}
```



Exercise(8) – Make a Decision Tree!

▾ 🤷‍♀️ : Then, Which class is *'train_data[1]'* classified into?

```
[156] def print_leaf(counts):
          """A nicer way to print the predictions at a leaf."""
          total = sum(counts.values()) * 1.0
          probs = {}
          for lbl in counts.keys():
              probs[lbl] = str(int(counts[lbl] / total * 100)) + "%"
          return probs
```

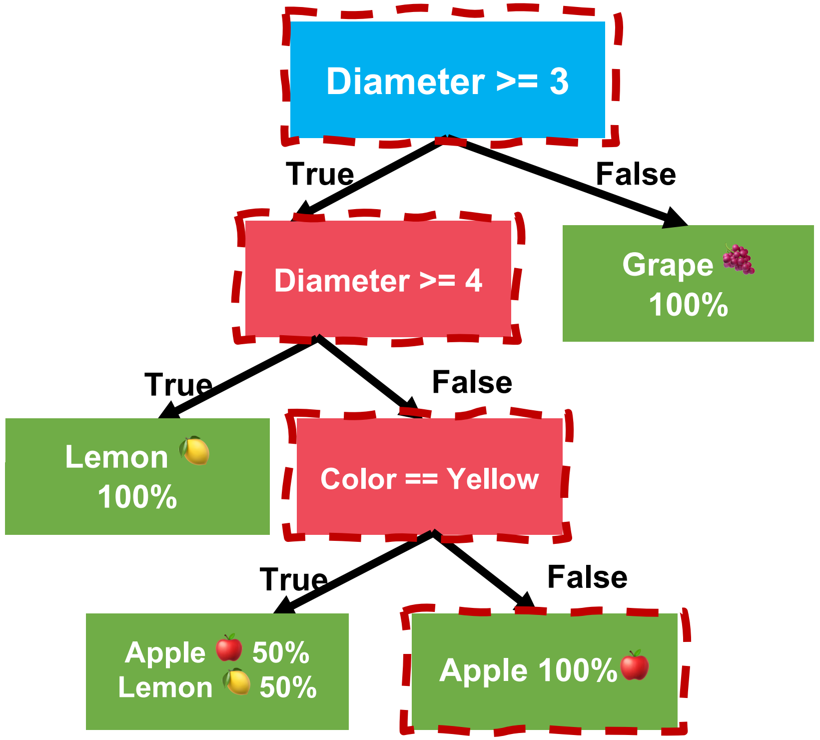
```
[157] print_leaf(classify(train_data[0], my_tree))
```

```
{'Apple': '100%'}
```

==> *If color is yellow and diameter is 3*, this fruit is predicted 100% 🍎.

```
print_leaf(classify(train_data[1], my_tree))
```

```
{'Apple': '50%', 'Lemon': '50%'}
```



Exercise(9) – Test our Decision Tree!

```
[158] testing_data = [
          ['Green', 3, 'Apple'],
          ['Yellow', 4, 'Apple'],
          ['Red', 2, 'Grape'],
          ['Red', 1, 'Grape'],
          ['Yellow', 3, 'Lemon'],
      ]
```

```
[159] for row in testing_data:
          print ("Actual: %s. Predicted: %s" %
                 (row[-1], print_leaf(classify(row, my_tree))))
```

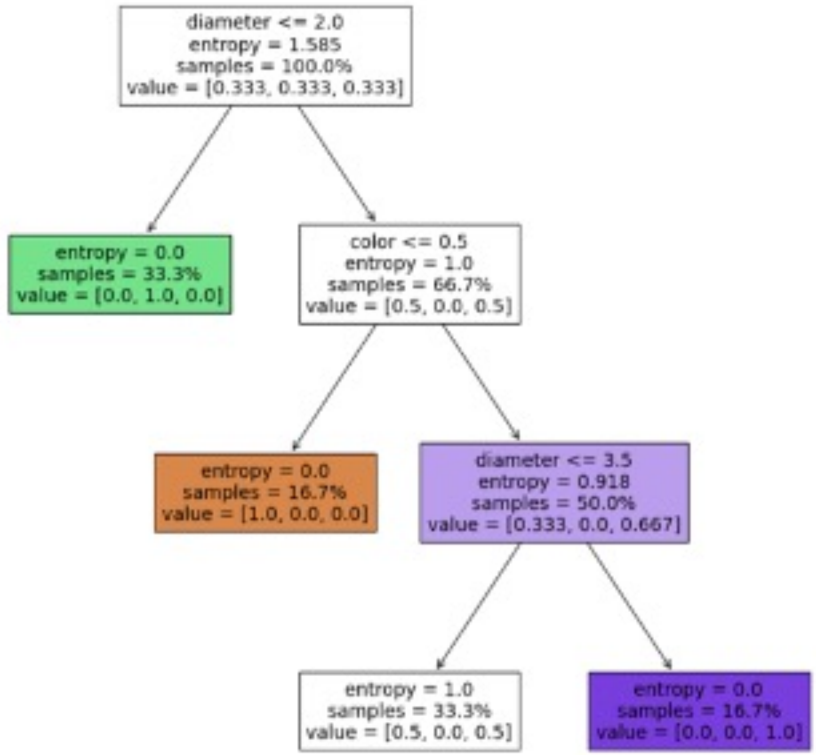
```
Actual: Apple. Predicted: {'Apple': '100%'}
Actual: Apple. Predicted: {'Lemon': '100%'}
Actual: Grape. Predicted: {'Grape': '100%'}
Actual: Grape. Predicted: {'Grape': '100%'}
Actual: Lemon. Predicted: {'Apple': '50%', 'Lemon': '50%'}
```

not correct!

Exercise(10) – Decision Tree with sklearn

```
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
import sklearn

decisionTree = DecisionTreeClassifier(random_state=0, criterion="entropy")
decisionTree.fit(train_df.iloc[:, 0:2], train_df.iloc[:, 2])

plt.figure(figsize=(12,12))
sklearn.tree.plot_tree(decisionTree, filled=True, feature_names=["color", "diameter"],proportion=True)
```

Color	Diameter	Label
Green	3	Apple
Yellow	3	Apple
Red	1	Grape
Red	1	Grape
Yellow	3	Lemon
Yellow	4	Lemon



Decision Trees

- Advantages
  - Simple to understand, interpret, and visualize
  - Little effort is required for data preparation
    - Normalization is not required
  - Able to obtain non-linear decision boundaries
  - Can handle both numeric and categorical data

- Disadvantages
  - Overfitting when the depth is too deep
  - High variance (models tends to be less stable)

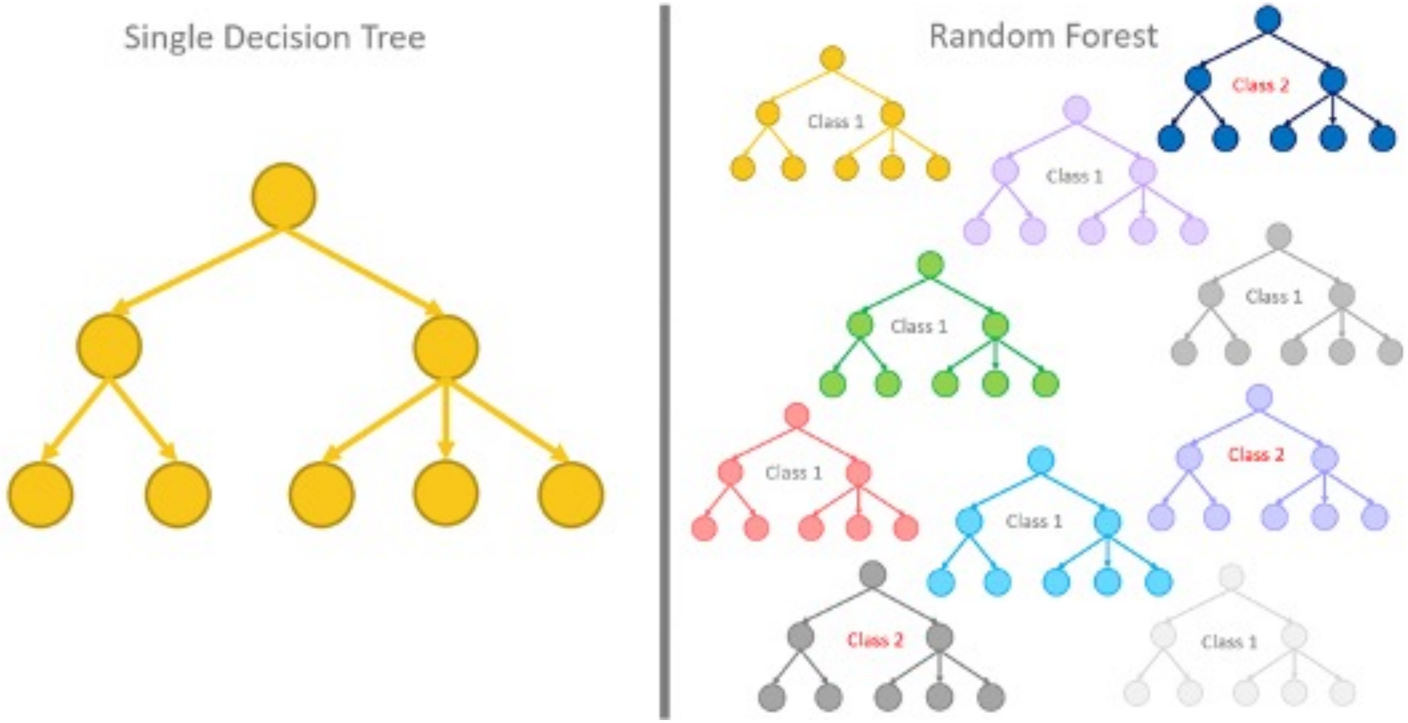
Background of Random Forests

- Issues with Decision Trees
  - Overfitting: occurs when the algorithm captures noise the data
  - Unstable: the model can get unstable with merely small variations in data(the model could get too much sensitive)

- Solutions
  - Must use tricks to find “simple trees”
    - Early stopping of training (learning)
      - Fixed depth: do not grow trees further than a specified depth
      - Minimum population per leaf node: do not split a node if the number of data instance fall in the node is smaller than a specified number
    - Readjusting/simplifying trained trees
      - Pruning: simplify trees by merging leaf nodes

Random Forests

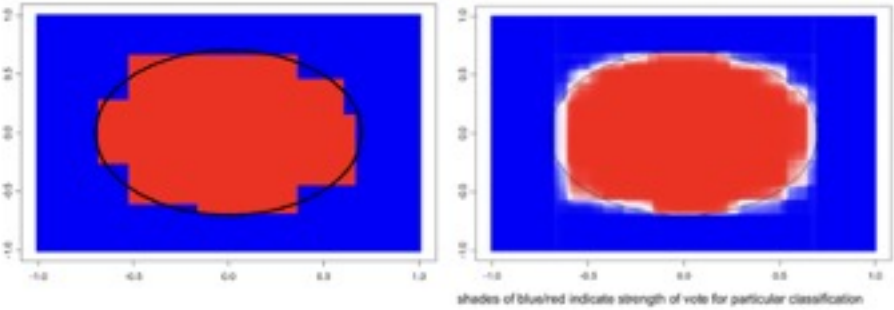
- Random Forest: Ensemble (mixture) of decision trees



https://towardsdatascience.com/from-a-single-decision-tree-to-a-random-forest-b9523be65147

Random Forests

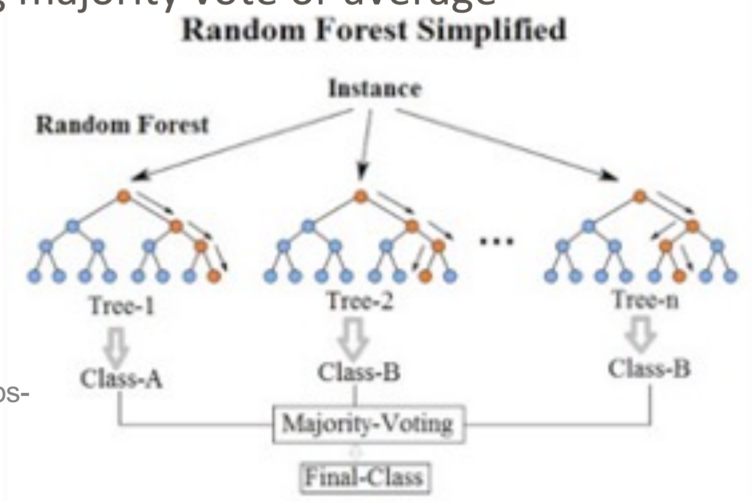
- Random Forest: Ensemble (mixture) of decision trees
  - Use multiple decision trees together to improve the predictive accuracy
    - To avoid over-fitting
    - To improve stability and accuracy
  - Combining weak classifiers in order to produce a strong classifier
    - Condition: diversity among the weak classifiers
  - Illustration: using 1 tree vs 100 trees



Random Forests

- How to achieve diversity
  - Bagging: **B**ootstrap **agg**regat**ing**
    - Bootstrap: Random resampling with repetition
    - Learn decision trees from different subsamples of original data

- How to make a group decision?
  - Each decision tree in a random forest makes a decision
  - Multiple decisions are aggregated using majority vote or average



https://levelup.gitconnected.com/a-noobs-guide-to-random-forest-d7398d56b01c

Scikit learn for Decision Tree

- Decision Tree
  - sklearn.tree.DecisionTreeClassfier

```
with open('../data/mnist.pkl', 'rb') as f:
    train_set, valid_set, test_set = pickle.load(f, encoding='latin1')
```

```
... loading data
```

```
train_x, train_y = train_set
test_x, test_y = test_set

train_x = pd.DataFrame(train_x)
train_y = pd.DataFrame(train_y, columns=['label'])
test_x = pd.DataFrame(test_x)
test_y = pd.DataFrame(test_y, columns=['label'])
```

```
from sklearn.tree import DecisionTreeClassifier

decisionTree = DecisionTreeClassifier(random_state=0, criterion="entropy")
decisionTree.fit(train_x, train_y)
```

{"gini", "entropy"}, default="gini"
The function to measure the quality of a split.

Scikit learn for Decision Tree

- Decision Tree
  - sklearn.tree.DecisionTreeClassfier

```
from sklearn.tree import DecisionTreeClassifier

decisionTree = DecisionTreeClassifier(random_state=0, criterion="entropy")
decisionTree.fit(train_x, train_y)
```

fit(*X*, *y*, *sample_weight=None*, *check_input=True*, *X_idx_sorted='deprecated'*)

X : *array-like, sparse matrix} of shape (n_samples, n_features)*
Y : *array-like of shape (n_samples,) or (n_samples, n_outputs)*

Scikit learn for Decision Tree

- Decision Tree
  - sklearn.tree.DecisionTreeClassfier

```
print("=== > Test set score : {:.2f}".format(decisionTree.score(test_x, test_y)))
=== > Test set score : 0.88
```

score(*X*, *y*, *sample_weight=None*)

X : *array-like of shape (n_samples, n_features)*
Y : *array-like of shape (n_samples,) or (n_samples, n_outputs)*

Scikit learn for Decision Tree

- Random Forest
  - sklearn.ensemble. RandomForestClassifier

```
from sklearn.ensemble import RandomForestClassifier

rforest = RandomForestClassifier(random_state=0)
rforest.fit(train_x, train_y)
```

fit(*X, y, sample_weight=None, check_input=True, X_idx_sorted='deprecated'*)

X : *array-like, sparse matrix} of shape (n_samples, n_features)*
Y : *array-like of shape (n_samples,) or (n_samples, n_outputs)*

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*) [source]

Scikit learn for Decision Tree

- Random Forest
  - sklearn.ensemble. RandomForestClassifier

```
print("Test set score : {:.2f}".format(rforest.score(test_x, test_y)))
```

```
Test set score : 0.97
```

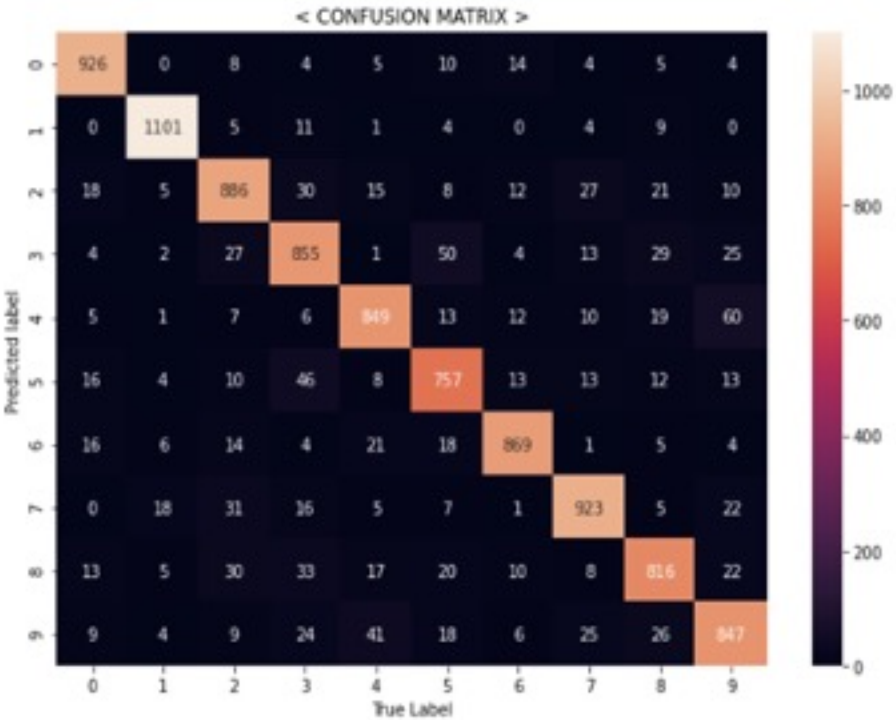
score(*X*, *y*, *sample_weight=None*)

X : *array-like of shape (n_samples, n_features)*
Y : *array-like of shape (n_samples,) or (n_samples, n_outputs)*

confusion matrix with heatmap

- decision tree

```
from sklearn.metrics import confusion_matrix
import seaborn as sn # if not available, “!pip install seaborn”

pred_y = decisionTree.predict(test_x)
cmdtree = confusion_matrix(test_y, pred_y)

plt.figure(figsize=(10,7))
sn.heatmap(cmdtree, annot=True, fmt='d')

plt.title(" < CONFUSION MATRIX > ")
plt.ylabel('Predicted label')
plt.xlabel('True Label')

plt.show()
```



confusion matrix with heatmap

- random forest

```
from sklearn.metrics import confusion_matrix
import seaborn as sn

pred_y = rforest.predict(test_x)
rtree_cmd = confusion_matrix(test_y, pred_y)

plt.figure(figsize = (10, 7))
sn.heatmap(rtree_cmd, annot=True, fmt='d')

plt.title(" < CONFUSION MATRIX > ")
plt.ylabel('Predicted label')
plt.xlabel('True Label')

plt.show()
```