



Want help with machine learning? [Take the FREE Crash-Course.](#)



# How To Implement The Decision Tree Algorithm From Scratch In Python

by Jason Brownlee on [November 9, 2016](#) in [Code Machine Learning Algorithms From Scratch](#)



Decision trees are a powerful prediction method and extremely popular.

They are popular because the final model is so easy to understand by practitioners and domain experts alike. The final decision tree can explain exactly why a specific prediction was made, making it very attractive for operational use.

Decision trees also provide the foundation for more advanced ensemble methods such as bagging, random forests and gradient boosting.

In this tutorial, you will discover how to implement the [Classification And Regression Tree algorithm](#) from scratch with Python.

After completing this tutorial, you will know:

- How to calculate and evaluate candidate split points in a data.
- How to arrange splits into a decision tree structure.
- How to apply the classification and regression tree algorithm to a real problem.

Let's get started.

- **Update Jan/2017:** Changed the calculation of fold\_size in cross\_validation\_split() to always be an integer. Fixes issues with Python 3.
- **Update Feb/2017:** Fixed a bug in build\_tree.
- **Update Aug/2017:** Fixed a bug in Gini calculation, added the missing weighting of group Gini scores by group size (thanks Michael!).
- **Update Aug/2018:** Tested and updated to work with Python 3.6.



How To Implement The Decision Tree Algorithm From Scratch In Python  
Photo by [Martin Cathrae](#), some rights reserved.

## Descriptions

This section provides a brief introduction to the Classification and Regression Tree algorithm and the Banknote dataset used in this tutorial.

## Classification and Regression Trees

Classification and Regression Trees or CART for short is an acronym introduced by Leo Breiman to refer to Decision Tree algorithms that can be used for classification or regression predictive modeling problems.

We will focus on using CART for classification in this tutorial.

The representation of the CART model is a binary tree. This is the same binary tree from algorithms and data structures, nothing too fancy (each node can have zero, one or two child nodes).

A node represents a single input variable (X) and a split point on that variable, assuming the variable is numeric. The leaf nodes (also called terminal nodes) of the tree contain an output variable (y) which is used to make a prediction.

Once created, a tree can be navigated with a new row of data following each branch with the splits until a final prediction is made.

Creating a binary decision tree is actually a process of dividing up the input space. A greedy approach is used to divide the space called recursive binary splitting. This is a numerical procedure where all the values are lined up and different split points are tried and tested using a cost function.

The split with the best cost (lowest cost because we minimize cost) is selected. All input variables and all possible split points are evaluated and chosen in a greedy manner based on the cost function.

- **Regression:** The cost function that is minimized to choose split points is the sum squared error across all training samples that fall within the rectangle.
- **Classification:** The Gini cost function is used which provides an indication of how pure the nodes are, where node purity refers to how mixed the training data assigned to each node is.

Splitting continues until nodes contain a minimum number of training examples or a maximum tree depth is reached.

## Banknote Dataset

The banknote dataset involves predicting whether a given banknote is authentic given a number of measures taken from a photograph.

The dataset contains 1,372 rows with 5 numeric variables. It is a classification problem with two classes (binary classification).

Below provides a list of the five variables in the dataset.

1. variance of Wavelet Transformed image (continuous).
2. skewness of Wavelet Transformed image (continuous).

3. kurtosis of Wavelet Transformed image (continuous).
4. entropy of image (continuous).
5. class (integer).

Below is a sample of the first 5 rows of the dataset

1	3.6216,8.6661,-2.8073,-0.44699,0
2	4.5459,8.1674,-2.4586,-1.4621,0
3	3.866,-2.6383,1.9242,0.10645,0
4	3.4566,9.5228,-4.0112,-3.5944,0
5	0.32924,-4.4552,4.5718,-0.9888,0
6	4.3684,9.6718,-3.9606,-3.1625,0

Using the Zero Rule Algorithm to predict the most common class value, the baseline accuracy on the problem is about 50%.

You can learn more and download the dataset from the [UCI Machine Learning Repository](#).

Download the dataset and place it in your current working directory with the filename **data\_banknote\_authentication.csv**.

## Tutorial

This tutorial is broken down into 5 parts:

1. Gini Index.
2. Create Split.
3. Build a Tree.
4. Make a Prediction.
5. Banknote Case Study.

These steps will give you the foundation that you need to implement the CART algorithm from scratch and apply it to your own predictive modeling problems.

### 1. Gini Index

The Gini index is the name of the cost function used to evaluate splits in the dataset.

A split in the dataset involves one input attribute and one value for that attribute. It can be used to divide training patterns into two groups of rows.

A Gini score gives an idea of how good a split is by how mixed the classes are in the two groups created by the split. A perfect separation results in a Gini score of 0, whereas the worst case split that results in 50/50 classes in each group result in a Gini score of 0.5 (for a 2 class problem).

Calculating Gini is best demonstrated with an example.

We have two groups of data with 2 rows in each group. The rows in the first group all belong to class 0 and the rows in the second group belong to class 1, so it's a perfect split.

We first need to calculate the proportion of classes in each group.

```
1 proportion = count(class_value) / count(rows)
```

The proportions for this example would be:

```
1 group_1_class_0 = 2 / 2 = 1
2 group_1_class_1 = 0 / 2 = 0
3 group_2_class_0 = 0 / 2 = 0
4 group_2_class_1 = 2 / 2 = 1
```

Gini is then calculated for each child node as follows:

```
1 gini_index = sum(proportion * (1.0 - proportion))
2 gini_index = 1.0 - sum(proportion * proportion)
```

The Gini index for each group must then be weighted by the size of the group, relative to all of the samples in the parent, e.g. all samples that are currently being grouped. We can add this weighting to the Gini calculation for a group as follows:

```
1 gini_index = (1.0 - sum(proportion * proportion)) * (group_size/total_samples)
```

In this example the Gini scores for each group are calculated as follows:

```
1 Gini(group_1) = (1 - (1*1 + 0*0)) * 2/4
2 Gini(group_1) = 0.0 * 0.5
```

```

3 Gini(group_1) = 0.0
4 Gini(group_2) = (1 - (0*0 + 1*1)) * 2/4
5 Gini(group_2) = 0.0 * 0.5
6 Gini(group_2) = 0.0

```

The scores are then added across each child node at the split point to give a final Gini score for the split point that can be compared to other candidate split points.

The Gini for this split point would then be calculated as 0.0 + 0.0 or a perfect Gini score of 0.0.

Below is a function named **gini\_index()** the calculates the Gini index for a list of groups and a list of known class values.

You can see that there are some safety checks in there to avoid a divide by zero for an empty group.

```

1 # Calculate the Gini index for a split dataset
2 def gini_index(groups, classes):
3     # count all samples at split point
4     n_instances = float(sum([len(group) for group in groups]))
5     # sum weighted Gini index for each group
6     gini = 0.0
7     for group in groups:
8         size = float(len(group))
9         # avoid divide by zero
10        if size == 0:
11            continue
12        score = 0.0
13        # score the group based on the score for each class
14        for class_val in classes:
15            p = [row[-1] for row in group].count(class_val) / size
16            score += p * p
17        # weight the group score by its relative size
18        gini += (1.0 - score) * (size / n_instances)
19    return gini

```

We can test this function with our worked example above. We can also test it for the worst case of a 50/50 split in each group. The complete example is listed below.

```

1 # Calculate the Gini index for a split dataset
2 def gini_index(groups, classes):
3     # count all samples at split point
4     n_instances = float(sum([len(group) for group in groups]))

```

```

5     # sum weighted Gini index for each group
6     gini = 0.0
7     for group in groups:
8         size = float(len(group))
9         # avoid divide by zero
10        if size == 0:
11            continue
12        score = 0.0
13        # score the group based on the score for each class
14        for class_val in classes:
15            p = [row[-1] for row in group].count(class_val) / size
16            score += p * p
17        # weight the group score by its relative size
18        gini += (1.0 - score) * (size / n_instances)
19    return gini
20
21 # test Gini values
22 print(gini_index([[[1, 1], [1, 0]], [[1, 1], [1, 0]], [0, 1]]))
23 print(gini_index([[[1, 0], [1, 0]], [[1, 1], [1, 1]], [0, 1]]))

```

Running the example prints the two Gini scores, first the score for the worst case at 0.5 followed by the score for the best case at 0.0.

```

1 0.5
2 0.0

```

Now that we know how to evaluate the results of a split, let's look at creating splits.

## 2. Create Split

A split is comprised of an attribute in the dataset and a value.

We can summarize this as the index of an attribute to split and the value by which to split rows on that attribute. This is just a useful shorthand for indexing into rows of data.

Creating a split involves three parts, the first we have already looked at which is calculating the Gini score. The remaining two parts are:

1. Splitting a Dataset.
2. Evaluating All Splits.

Let's take a look at each.

## 2.1. Splitting a Dataset

Splitting a dataset means separating a dataset into two lists of rows given the index of an attribute and a split value for that attribute.

Once we have the two groups, we can then use our Gini score above to evaluate the cost of the split.

Splitting a dataset involves iterating over each row, checking if the attribute value is below or above the split value and assigning it to the left or right group respectively.

Below is a function named **test\_split()** that implements this procedure.

```
1 # Split a dataset based on an attribute and an attribute value
2 def test_split(index, value, dataset):
3     left, right = list(), list()
4     for row in dataset:
5         if row[index] < value:
6             left.append(row)
7         else:
8             right.append(row)
9     return left, right
```

Not much to it.

Note that the right group contains all rows with a value at the index above or equal to the split value.

## 2.2. Evaluating All Splits

With the Gini function above and the test split function we now have everything we need to evaluate splits.

Given a dataset, we must check every value on each attribute as a candidate split, evaluate the cost of the split and find the best possible split we could make.

Once the best split is found, we can use it as a node in our decision tree.

This is an exhaustive and greedy algorithm.



We will use a dictionary to represent a node in the decision tree as we can store data by name. When selecting the best split and using it as a new node for the tree we will store the index of the chosen attribute, the value of that attribute by which to split and the two groups of data split by the chosen split point.

Each group of data is its own small dataset of just those rows assigned to the left or right group by the splitting process. You can imagine how we might split each group again, recursively as we build out our decision tree.

Below is a function named **get\_split()** that implements this procedure. You can see that it iterates over each attribute (except the class value) and then each value for that attribute, splitting and evaluating splits as it goes.

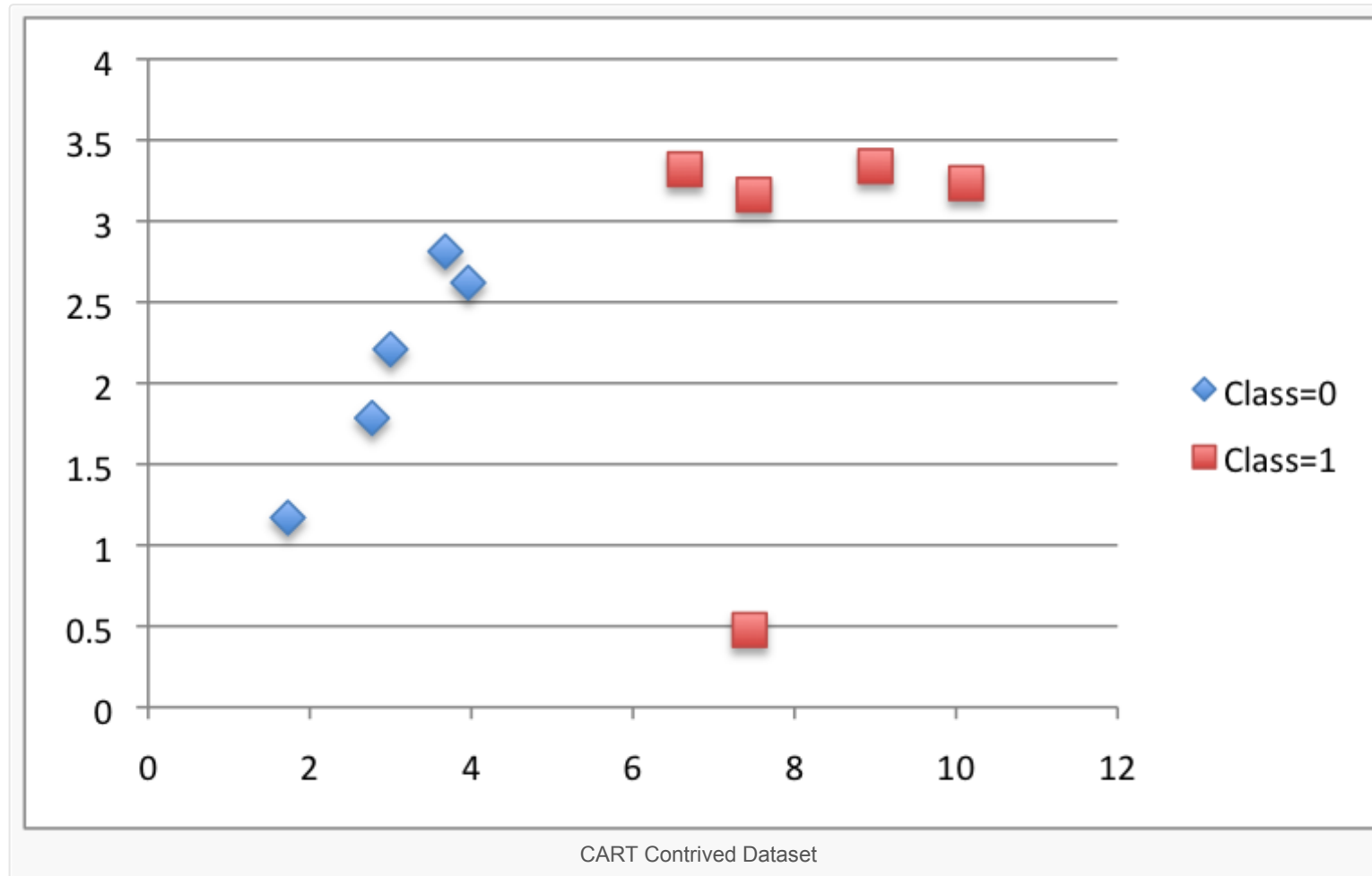
The best split is recorded and then returned after all checks are complete.

```
1 # Select the best split point for a dataset
2 def get_split(dataset):
3     class_values = list(set(row[-1] for row in dataset))
4     b_index, b_value, b_score, b_groups = 999, 999, 999, None
5     for index in range(len(dataset[0])-1):
6         for row in dataset:
7             groups = test_split(index, row[index], dataset)
8             gini = gini_index(groups, class_values)
9             if gini < b_score:
10                 b_index, b_value, b_score, b_groups = index, row[index], gini, groups
11     return {'index':b_index, 'value':b_value, 'groups':b_groups}
```

We can contrive a small dataset to test out this function and our whole dataset splitting process.

	X1	X2	Y
1	2.771244718	1.784783929	0
2	1.728571309	1.169761413	0
3	3.678319846	2.81281357	0
4	3.961043357	2.61995032	0
5	2.999208922	2.209014212	0
6	7.497545867	3.162953546	1
7	9.00220326	3.339047188	1
8	7.444542326	0.476683375	1
9	10.12493903	3.234550982	1
10	6.642287351	3.319983761	1
11			

We can plot this dataset using separate colors for each class. You can see that it would not be difficult to manually pick a value of  $X_1$  (x-axis on the plot) to split this dataset.



The example below puts all of this together.

```
1 # Split a dataset based on an attribute and an attribute value
2 def test_split(index, value, dataset):
3     left, right = list(), list()
4     for row in dataset:
```

```

5         if row[index] < value:
6             left.append(row)
7         else:
8             right.append(row)
9     return left, right
10
11 # Calculate the Gini index for a split dataset
12 def gini_index(groups, classes):
13     # count all samples at split point
14     n_instances = float(sum([len(group) for group in groups]))
15     # sum weighted Gini index for each group
16     gini = 0.0
17     for group in groups:
18         size = float(len(group))
19         # avoid divide by zero
20         if size == 0:
21             continue
22         score = 0.0
23         # score the group based on the score for each class
24         for class_val in classes:
25             p = [row[-1] for row in group].count(class_val) / size
26             score += p * p
27         # weight the group score by its relative size
28         gini += (1.0 - score) * (size / n_instances)
29     return gini
30
31 # Select the best split point for a dataset
32 def get_split(dataset):
33     class_values = list(set(row[-1] for row in dataset))
34     b_index, b_value, b_score, b_groups = 999, 999, 999, None
35     for index in range(len(dataset[0])-1):
36         for row in dataset:
37             groups = test_split(index, row[index], dataset)
38             gini = gini_index(groups, class_values)
39             print('X%d < %.3f Gini=%.3f' % ((index+1), row[index], gini))
40             if gini < b_score:
41                 b_index, b_value, b_score, b_groups = index, row[index], gini, groups
42     return {'index':b_index, 'value':b_value, 'groups':b_groups}
43
44 dataset = [[2.771244718,1.784783929,0],
45            [1.728571309,1.169761413,0],
46            [3.678319846,2.81281357,0],
47            [3.961043357,2.61995032,0],
48            [2.999208922,2.209014212,0],
49            [7.497545867,3.162953546,1],

```

```

50 [9.00220326,3.339047188,1],
51 [7.444542326,0.476683375,1],
52 [10.12493903,3.234550982,1],
53 [6.642287351,3.319983761,1]]
54 split = get_split(dataset)
55 print('Split: [X%d < %.3f]' % ((split['index']+1), split['value']))

```

The **get\_split()** function was modified to print out each split point and it's Gini index as it was evaluated.

Running the example prints all of the Gini scores and then prints the score of best split in the dataset of  $X_1 < 6.642$  with a Gini Index of 0.0 or a perfect split.

```

1 X1 < 2.771 Gini=0.444
2 X1 < 1.729 Gini=0.500
3 X1 < 3.678 Gini=0.286
4 X1 < 3.961 Gini=0.167
5 X1 < 2.999 Gini=0.375
6 X1 < 7.498 Gini=0.286
7 X1 < 9.002 Gini=0.375
8 X1 < 7.445 Gini=0.167
9 X1 < 10.125 Gini=0.444
10 X1 < 6.642 Gini=0.000
11 X2 < 1.785 Gini=0.500
12 X2 < 1.170 Gini=0.444
13 X2 < 2.813 Gini=0.320
14 X2 < 2.620 Gini=0.417
15 X2 < 2.209 Gini=0.476
16 X2 < 3.163 Gini=0.167
17 X2 < 3.339 Gini=0.444
18 X2 < 0.477 Gini=0.500
19 X2 < 3.235 Gini=0.286
20 X2 < 3.320 Gini=0.375
21 Split: [X1 < 6.642]

```

Now that we know how to find the best split points in a dataset or list of rows, let's see how we can use it to build out a decision tree.

### 3. Build a Tree

Creating the root node of the tree is easy.

We call the above **get\_split()** function using the entire dataset.

Adding more nodes to our tree is more interesting.

Building a tree may be divided into 3 main parts:

1. Terminal Nodes.
2. Recursive Splitting.
3. Building a Tree.

### 3.1. Terminal Nodes

We need to decide when to stop growing a tree.

We can do that using the depth and the number of rows that the node is responsible for in the training dataset.

- **Maximum Tree Depth.** This is the maximum number of nodes from the root node of the tree. Once a maximum depth of the tree is met, we must stop splitting adding new nodes. Deeper trees are more complex and are more likely to overfit the training data.
- **Minimum Node Records.** This is the minimum number of training patterns that a given node is responsible for. Once at or below this minimum, we must stop splitting and adding new nodes. Nodes that account for too few training patterns are expected to be too specific and are likely to overfit the training data.

These two approaches will be user-specified arguments to our tree building procedure.

There is one more condition. It is possible to choose a split in which all rows belong to one group. In this case, we will be unable to continue splitting and adding child nodes as we will have no records to split on one side or another.

Now we have some ideas of when to stop growing the tree. When we do stop growing at a given point, that node is called a terminal node and is used to make a final prediction.

This is done by taking the group of rows assigned to that node and selecting the most common class value in the group. This will be used to make predictions.

Below is a function named **to\_terminal()** that will select a class value for a group of rows. It returns the most common output value in a list of rows.

```

1 # Create a terminal node value
2 def to_terminal(group):
3     outcomes = [row[-1] for row in group]
4     return max(set(outcomes), key=outcomes.count)

```

### 3.2. Recursive Splitting

We know how and when to create terminal nodes, now we can build our tree.

Building a decision tree involves calling the above developed **get\_split()** function over and over again on the groups created for each node.

New nodes added to an existing node are called child nodes. A node may have zero children (a terminal node), one child (one side makes a prediction directly) or two child nodes. We will refer to the child nodes as left and right in the dictionary representation of a given node.

Once a node is created, we can create child nodes recursively on each group of data from the split by calling the same function again.

Below is a function that implements this recursive procedure. It takes a node as an argument as well as the maximum depth, minimum number of patterns in a node and the current depth of a node.

You can imagine how this might be first called passing in the root node and the depth of 1. This function is best explained in steps:

1. Firstly, the two groups of data split by the node are extracted for use and deleted from the node. As we work on these groups the node no longer requires access to these data.
2. Next, we check if either left or right group of rows is empty and if so we create a terminal node using what records we do have.
3. We then check if we have reached our maximum depth and if so we create a terminal node.
4. We then process the left child, creating a terminal node if the group of rows is too small, otherwise creating and adding the left node in a depth first fashion until the bottom of the tree is reached on this branch.
5. The right side is then processed in the same manner, as we rise back up the constructed tree to the root.

```

1 # Create child splits for a node or make terminal
2 def split(node, max_depth, min_size, depth):
3     left, right = node['groups']
4     del(node['groups'])
5     # check for a no split
6     if not left or not right:
7         node['left'] = node['right'] = to_terminal(left + right)

```

```

8         return
9     # check for max depth
10    if depth >= max_depth:
11        node['left'], node['right'] = to_terminal(left), to_terminal(right)
12    return
13    # process left child
14    if len(left) <= min_size:
15        node['left'] = to_terminal(left)
16    else:
17        node['left'] = get_split(left)
18        split(node['left'], max_depth, min_size, depth+1)
19    # process right child
20    if len(right) <= min_size:
21        node['right'] = to_terminal(right)
22    else:
23        node['right'] = get_split(right)
24        split(node['right'], max_depth, min_size, depth+1)

```

### 3.3. Building a Tree

We can now put all of the pieces together.

Building the tree involves creating the root node and calling the **split()** function that then calls itself recursively to build out the whole tree.

Below is the small **build\_tree()** function that implements this procedure.

```

1 # Build a decision tree
2 def build_tree(train, max_depth, min_size):
3     root = get_split(train)
4     split(root, max_depth, min_size, 1)
5     return root

```

We can test out this whole procedure using the small dataset we contrived above.

Below is the complete example.

Also included is a small **print\_tree()** function that recursively prints out nodes of the decision tree with one line per node. Although not as striking as a real decision tree diagram, it gives an idea of the tree structure and decisions made throughout.

```

1 # Split a dataset based on an attribute and an attribute value

```

```

2 def test_split(index, value, dataset):
3     left, right = list(), list()
4     for row in dataset:
5         if row[index] < value:
6             left.append(row)
7         else:
8             right.append(row)
9     return left, right
10
11 # Calculate the Gini index for a split dataset
12 def gini_index(groups, classes):
13     # count all samples at split point
14     n_instances = float(sum([len(group) for group in groups]))
15     # sum weighted Gini index for each group
16     gini = 0.0
17     for group in groups:
18         size = float(len(group))
19         # avoid divide by zero
20         if size == 0:
21             continue
22         score = 0.0
23         # score the group based on the score for each class
24         for class_val in classes:
25             p = [row[-1] for row in group].count(class_val) / size
26             score += p * p
27         # weight the group score by its relative size
28         gini += (1.0 - score) * (size / n_instances)
29     return gini
30
31 # Select the best split point for a dataset
32 def get_split(dataset):
33     class_values = list(set(row[-1] for row in dataset))
34     b_index, b_value, b_score, b_groups = 999, 999, 999, None
35     for index in range(len(dataset[0])-1):
36         for row in dataset:
37             groups = test_split(index, row[index], dataset)
38             gini = gini_index(groups, class_values)
39             if gini < b_score:
40                 b_index, b_value, b_score, b_groups = index, row[index], gini, groups
41     return {'index':b_index, 'value':b_value, 'groups':b_groups}
42
43 # Create a terminal node value
44 def to_terminal(group):
45     outcomes = [row[-1] for row in group]
46     return max(set(outcomes), key=outcomes.count)

```



```

47
48 # Create child splits for a node or make terminal
49 def split(node, max_depth, min_size, depth):
50     left, right = node['groups']
51     del(node['groups'])
52     # check for a no split
53     if not left or not right:
54         node['left'] = node['right'] = to_terminal(left + right)
55         return
56     # check for max depth
57     if depth >= max_depth:
58         node['left'], node['right'] = to_terminal(left), to_terminal(right)
59         return
60     # process left child
61     if len(left) <= min_size:
62         node['left'] = to_terminal(left)
63     else:
64         node['left'] = get_split(left)
65         split(node['left'], max_depth, min_size, depth+1)
66     # process right child
67     if len(right) <= min_size:
68         node['right'] = to_terminal(right)
69     else:
70         node['right'] = get_split(right)
71         split(node['right'], max_depth, min_size, depth+1)
72
73 # Build a decision tree
74 def build_tree(train, max_depth, min_size):
75     root = get_split(train)
76     split(root, max_depth, min_size, 1)
77     return root
78
79 # Print a decision tree
80 def print_tree(node, depth=0):
81     if isinstance(node, dict):
82         print('%s[X%d < %.3f]' % ((depth*' ', (node['index']+1), node['value'])))
83         print_tree(node['left'], depth+1)
84         print_tree(node['right'], depth+1)
85     else:
86         print('%s[%s]' % ((depth*' ', node)))
87
88 dataset = [[2.771244718, 1.784783929, 0],
89            [1.728571309, 1.169761413, 0],
90            [3.678319846, 2.81281357, 0],
91            [3.961043357, 2.61995032, 0],

```

```

92 [2.999208922,2.209014212,0],
93 [7.497545867,3.162953546,1],
94 [9.00220326,3.339047188,1],
95 [7.444542326,0.476683375,1],
96 [10.12493903,3.234550982,1],
97 [6.642287351,3.319983761,1]]
98 tree = build_tree(dataset, 1, 1)
99 print_tree(tree)

```

We can vary the maximum depth argument as we run this example and see the effect on the printed tree.

With a maximum depth of 1 (the second parameter in the call to the **build\_tree()** function), we can see that the tree uses the perfect split we discovered in the previous section. This is a tree with one node, also called a decision stump.

```

1 [X1 < 6.642]
2 [0]
3 [1]

```

Increasing the maximum depth to 2, we are forcing the tree to make splits even when none are required. The **X1** attribute is then used again by both the left and right children of the root node to split up the already perfect mix of classes.

```

1 [X1 < 6.642]
2 [X1 < 2.771]
3 [0]
4 [0]
5 [X1 < 7.498]
6 [1]
7 [1]

```

Finally, and perversely, we can force one more level of splits with a maximum depth of 3.

```

1 [X1 < 6.642]
2 [X1 < 2.771]
3 [0]
4 [X1 < 2.771]
5 [0]
6 [0]
7 [X1 < 7.498]
8 [X1 < 7.445]
9 [1]
10 [1]

```

```
11 [X1 < 7.498]
12 [1]
13 [1]
```

These tests show that there is great opportunity to refine the implementation to avoid unnecessary splits. This is left as an extension.

Now that we can create a decision tree, let's see how we can use it to make predictions on new data.

## 4. Make a Prediction

Making predictions with a decision tree involves navigating the tree with the specifically provided row of data.

Again, we can implement this using a recursive function, where the same prediction routine is called again with the left or the right child nodes, depending on how the split affects the provided data.

We must check if a child node is either a terminal value to be returned as the prediction, or if it is a dictionary node containing another level of the tree to be considered.

Below is the **predict()** function that implements this procedure. You can see how the index and value in a given node

You can see how the index and value in a given node is used to evaluate whether the row of provided data falls on the left or the right of the split.

```
1 # Make a prediction with a decision tree
2 def predict(node, row):
3     if row[node['index']] < node['value']:
4         if isinstance(node['left'], dict):
5             return predict(node['left'], row)
6         else:
7             return node['left']
8     else:
9         if isinstance(node['right'], dict):
10            return predict(node['right'], row)
11        else:
12            return node['right']
```

We can use our contrived dataset to test this function. Below is an example that uses a hard-coded decision tree with a single node that best splits the data (a decision stump).

The example makes a prediction for each row in the dataset.

```
1 # Make a prediction with a decision tree
2 def predict(node, row):
3     if row[node['index']] < node['value']:
4         if isinstance(node['left'], dict):
5             return predict(node['left'], row)
6         else:
7             return node['left']
8     else:
9         if isinstance(node['right'], dict):
10            return predict(node['right'], row)
11        else:
12            return node['right']
13
14 dataset = [[2.771244718, 1.784783929, 0],
15            [1.728571309, 1.169761413, 0],
16            [3.678319846, 2.81281357, 0],
17            [3.961043357, 2.61995032, 0],
18            [2.999208922, 2.209014212, 0],
19            [7.497545867, 3.162953546, 1],
20            [9.00220326, 3.339047188, 1],
21            [7.444542326, 0.476683375, 1],
22            [10.12493903, 3.234550982, 1],
23            [6.642287351, 3.319983761, 1]]
24
25 # predict with a stump
26 stump = {'index': 0, 'right': 1, 'value': 6.642287351, 'left': 0}
27 for row in dataset:
28     prediction = predict(stump, row)
29     print('Expected=%d, Got=%d' % (row[-1], prediction))
```

Running the example prints the correct prediction for each row, as expected.

```
1 Expected=0, Got=0
2 Expected=0, Got=0
3 Expected=0, Got=0
4 Expected=0, Got=0
5 Expected=0, Got=0
6 Expected=1, Got=1
7 Expected=1, Got=1
8 Expected=1, Got=1
9 Expected=1, Got=1
```

We now know how to create a decision tree and use it to make predictions. Now, let's apply it to a real dataset.

## 5. Banknote Case Study

This section applies the CART algorithm to the Bank Note dataset.

The first step is to load the dataset and convert the loaded data to numbers that we can use to calculate split points. For this we will use the helper function **load\_csv()** to load the file and **str\_column\_to\_float()** to convert string numbers to floats.

We will evaluate the algorithm using k-fold cross-validation with 5 folds. This means that  $1372/5=274.4$  or just over 270 records will be used in each fold. We will use the helper functions **evaluate\_algorithm()** to evaluate the algorithm with cross-validation and **accuracy\_metric()** to calculate the accuracy of predictions.

A new function named **decision\_tree()** was developed to manage the application of the CART algorithm, first creating the tree from the training dataset, then using the tree to make predictions on a test dataset.

The complete example is listed below.

```
1 # CART on the Bank Note dataset
2 from random import seed
3 from random import randrange
4 from csv import reader
5
6 # Load a CSV file
7 def load_csv(filename):
8     file = open(filename, "rb")
9     lines = reader(file)
10    dataset = list(lines)
11    return dataset
12
13 # Convert string column to float
14 def str_column_to_float(dataset, column):
15     for row in dataset:
16         row[column] = float(row[column].strip())
17
18 # Split a dataset into k folds
```

```

19 def cross_validation_split(dataset, n_folds):
20     dataset_split = list()
21     dataset_copy = list(dataset)
22     fold_size = int(len(dataset) / n_folds)
23     for i in range(n_folds):
24         fold = list()
25         while len(fold) < fold_size:
26             index = randrange(len(dataset_copy))
27             fold.append(dataset_copy.pop(index))
28         dataset_split.append(fold)
29     return dataset_split
30
31 # Calculate accuracy percentage
32 def accuracy_metric(actual, predicted):
33     correct = 0
34     for i in range(len(actual)):
35         if actual[i] == predicted[i]:
36             correct += 1
37     return correct / float(len(actual)) * 100.0
38
39 # Evaluate an algorithm using a cross validation split
40 def evaluate_algorithm(dataset, algorithm, n_folds, *args):
41     folds = cross_validation_split(dataset, n_folds)
42     scores = list()
43     for fold in folds:
44         train_set = list(folds)
45         train_set.remove(fold)
46         train_set = sum(train_set, [])
47         test_set = list()
48         for row in fold:
49             row_copy = list(row)
50             test_set.append(row_copy)
51             row_copy[-1] = None
52         predicted = algorithm(train_set, test_set, *args)
53         actual = [row[-1] for row in fold]
54         accuracy = accuracy_metric(actual, predicted)
55         scores.append(accuracy)
56     return scores
57
58 # Split a dataset based on an attribute and an attribute value
59 def test_split(index, value, dataset):
60     left, right = list(), list()
61     for row in dataset:
62         if row[index] < value:
63             left.append(row)

```

```

64         else:
65             right.append(row)
66         return left, right
67
68 # Calculate the Gini index for a split dataset
69 def gini_index(groups, classes):
70     # count all samples at split point
71     n_instances = float(sum([len(group) for group in groups]))
72     # sum weighted Gini index for each group
73     gini = 0.0
74     for group in groups:
75         size = float(len(group))
76         # avoid divide by zero
77         if size == 0:
78             continue
79         score = 0.0
80         # score the group based on the score for each class
81         for class_val in classes:
82             p = [row[-1] for row in group].count(class_val) / size
83             score += p * p
84         # weight the group score by its relative size
85         gini += (1.0 - score) * (size / n_instances)
86     return gini
87
88 # Select the best split point for a dataset
89 def get_split(dataset):
90     class_values = list(set(row[-1] for row in dataset))
91     b_index, b_value, b_score, b_groups = 999, 999, 999, None
92     for index in range(len(dataset[0])-1):
93         for row in dataset:
94             groups = test_split(index, row[index], dataset)
95             gini = gini_index(groups, class_values)
96             if gini < b_score:
97                 b_index, b_value, b_score, b_groups = index, row[index], gini, groups
98     return {'index':b_index, 'value':b_value, 'groups':b_groups}
99
100 # Create a terminal node value
101 def to_terminal(group):
102     outcomes = [row[-1] for row in group]
103     return max(set(outcomes), key=outcomes.count)
104
105 # Create child splits for a node or make terminal
106 def split(node, max_depth, min_size, depth):
107     left, right = node['groups']
108     del(node['groups'])

```

```

109 # check for a no split
110 if not left or not right:
111     node['left'] = node['right'] = to_terminal(left + right)
112     return
113 # check for max depth
114 if depth >= max_depth:
115     node['left'], node['right'] = to_terminal(left), to_terminal(right)
116     return
117 # process left child
118 if len(left) <= min_size:
119     node['left'] = to_terminal(left)
120 else:
121     node['left'] = get_split(left)
122     split(node['left'], max_depth, min_size, depth+1)
123 # process right child
124 if len(right) <= min_size:
125     node['right'] = to_terminal(right)
126 else:
127     node['right'] = get_split(right)
128     split(node['right'], max_depth, min_size, depth+1)
129
130 # Build a decision tree
131 def build_tree(train, max_depth, min_size):
132     root = get_split(train)
133     split(root, max_depth, min_size, 1)
134     return root
135
136 # Make a prediction with a decision tree
137 def predict(node, row):
138     if row[node['index']] < node['value']:
139         if isinstance(node['left'], dict):
140             return predict(node['left'], row)
141         else:
142             return node['left']
143     else:
144         if isinstance(node['right'], dict):
145             return predict(node['right'], row)
146         else:
147             return node['right']
148
149 # Classification and Regression Tree Algorithm
150 def decision_tree(train, test, max_depth, min_size):
151     tree = build_tree(train, max_depth, min_size)
152     predictions = list()
153     for row in test:

```



```

154     prediction = predict(tree, row)
155     predictions.append(prediction)
156     return(predictions)
157
158 # Test CART on Bank Note dataset
159 seed(1)
160 # load and prepare data
161 filename = 'data_banknote_authentication.csv'
162 dataset = load_csv(filename)
163 # convert string attributes to integers
164 for i in range(len(dataset[0])):
165     str_column_to_float(dataset, i)
166 # evaluate algorithm
167 n_folds = 5
168 max_depth = 5
169 min_size = 10
170 scores = evaluate_algorithm(dataset, decision_tree, n_folds, max_depth, min_size)
171 print('Scores: %s' % scores)
172 print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))

```

The example uses the max tree depth of 5 layers and the minimum number of rows per node to 10. These parameters to CART were chosen with a little experimentation, but are by no means are they optimal.

Running the example prints the average classification accuracy on each fold as well as the average performance across all folds.

You can see that CART and the chosen configuration achieved a mean classification accuracy of about 97% which is dramatically better than the Zero Rule algorithm that achieved 50% accuracy.

```

1 Scores: [96.35036496350365, 97.08029197080292, 97.44525547445255, 98.17518248175182, 97.44525547445255]
2 Mean Accuracy: 97.299%

```

## Extensions

This section lists extensions to this tutorial that you may wish to explore.

- **Algorithm Tuning.** The application of CART to the Bank Note dataset was not tuned. Experiment with different parameter values and see if you can achieve better performance.

- **Cross Entropy.** Another cost function for evaluating splits is cross entropy (logloss). You could implement and experiment with this alternative cost function.
- **Tree Pruning.** An important technique for reducing overfitting of the training dataset is to prune the trees. Investigate and implement tree pruning methods.
- **Categorical Dataset.** The example was designed for input data with numerical or ordinal input attributes, experiment with categorical input data and splits that may use equality instead of ranking.
- **Regression.** Adapt the tree for regression using a different cost function and method for creating terminal nodes.
- **More Datasets.** Apply the algorithm to more datasets on the UCI Machine Learning Repository.

**Did you explore any of these extensions?**

Share your experiences in the comments below.

## Review

In this tutorial, you discovered how to implement the decision tree algorithm from scratch with Python.

Specifically, you learned:

- How to select and evaluate split points in a training dataset.
- How to recursively build a decision tree from multiple splits.
- How to apply the CART algorithm to a real world classification predictive modeling problem.

**Do you have any questions?**

Ask your questions in the comments below and I will do my best to answer them.

---

## Want to Code Algorithms in Python Without Math?

**Code Your First Algorithm in Minutes**



...with step-by-step tutorials on real-world datasets

Discover how in my new Ebook:

[Machine Learning Algorithms From Scratch](#)

It covers **18 tutorials** with all the code for **12 top algorithms**, like:  
Linear Regression, k-Nearest Neighbors, Stochastic Gradient Descent and much more...

## Finally, Pull Back the Curtain on Machine Learning Algorithms

Skip the Academics. Just Results.

[Click to learn more.](#)



### About Jason Brownlee

Jason Brownlee, PhD is a machine learning specialist who teaches developers how to get results with modern machine learning methods via hands-on tutorials.

[View all posts by Jason Brownlee →](#)

[← How to Implement the Backpropagation Algorithm From Scratch In Python](#)

[How to Implement Bagging From Scratch With Python →](#)

## 243 Responses to *How To Implement The Decision Tree Algorithm From Scratch In Python*



**steve** November 23, 2016 at 4:02 am #

REPLY ↩

Super good .. Thanks a lot for sharing



**Jason Brownlee** November 23, 2016 at 9:01 am #

REPLY ↩

I'm glad you found it useful steve.



**MAk** August 26, 2017 at 1:14 pm #

REPLY ↩

Hi , Could you explain what do it mean ?

[X1 < 6.642]

[X1 < 2.771]

[0]

[0]

[X1 < 7.498]

[1]

[1]

Does it mean if X1 <6.642 and X1 <2.771, it belongs to class 0, if X1 <6.642 and X1 < 7.498, it belong to class 1 ?

Thanks for your help.

Mak



**Jason Brownlee** August 27, 2017 at 5:46 am #

REPLY ↩

Yes,  $[X_1 < 6.642]$  is the root node with two child nodes, each leaf node has a classification label.



**Furkon** April 13, 2018 at 9:17 pm #

Hello, can you help me with this problem?

This project is expecting you to write two functions. The first will take the training data including the type of features and returns a decision tree best modeling the classification problem. This should not do any pruning of the tree. An optional part of the homework will require the pruning step. The second function will just apply the decision tree to a given set of data points.

You should follow the following for this homework:

1. Load the data and perform a quick analysis of what it is and what features it has. You will need to construct a vector indicating the type (values) of each of the features. In this case, you can assume that you have numeric (real or integer) and categorical values.
2. Implement the function “`build_dt(X, y, attribute_types, options)`”.
  - a. `X`: is the matrix of features/attributes for the training data. Each row includes a data sample.
  - b. `y`: The vector containing the class labels for each sample in the rows of `X`.
  - c. `attribute_types`: The vector containing (1: integer/real) or (2: categorical) indicating the type of each attributes (the columns of `X`).
  - d. `options`: Any options you might want to pass to your decision tree builder.
  - e. Returns a decision tree of the structure of your choice.
3. Implement the function “`predict_dt(dt, X, options)`”.
  - a. `dt`: The decision tree modeled by “`build_dt`” function.
  - b. `X`: is the matrix of features/attributes for the test data.
  - c. Returns a vector for the predicted class labels.
4. Report the performance of your implementation using an appropriate k-fold cross validation using confusion matrices on the given dataset.  
[Optional] Implement the pruning strategy discussed in the class. Repeat the steps 4 above. Indicate any assumptions you might have made.

**Jason Brownlee** April 14, 2018 at 6:44 am #



I think you should complete your own homework assignments.

If you have questions about your homework assignment, ask your teacher. That is why you're paying them.



**Edgar Panganiban** July 23, 2018 at 2:44 pm #

Hi can you help me interpret the model the I created..thanks

[X1 < 2.000]

[X10 < 0.538]

[0.0]

[1.0]

[X15 < 10.200]

[X20 < 2.700]

[X10 < 0.574]

[X6 < 14.000]

[1.0]

[X8 < 256.000]

[1.0]

[X16 < 5.700]

[X18 < 0.600]

[X2 < 82.000]

[X15 < 4.100]

[0.0]

[0.0]

[1.0]

[X4 < 112.000]

[X1 < 3.000]

[0.0]

[0.0]

[0.0]

[0.0]  
[0.0]  
[0.0]  
[1.0]



**Railot Railot** November 17, 2018 at 6:45 pm #

REPLY ↩

hello, super jor let me translate this article in russian language with link on this resource?



**Jason Brownlee** November 18, 2018 at 6:39 am #

REPLY ↩

No thanks. This is a common question that I answer here:

<https://machinelearningmastery.com/faq/single-faq/can-i-translate-your-posts-books-into-another-language>



**jonathan** November 27, 2016 at 9:22 am #

REPLY ↩

can this code be used for a multinomial Decision tree dataset?



**Jason Brownlee** November 27, 2016 at 10:22 am #

REPLY ↩

It can with some modification.

**STORM RICK** November 29, 2016 at 1:02 am #

REPLY ↩



What modifications would you recommend?



**Jason Brownlee** November 29, 2016 at 8:53 am #

REPLY ↩

Specifically the handling of evaluating and selecting nominal values at split points.



**Mike** December 24, 2016 at 2:48 pm #

REPLY ↩

Thanks for detailed description and code.

I tried to run and got 'ValueError: empty range for randrange()' in line 26:

```
index = randrange(len(dataset_copy))
```

if replace dataset\_copy to list(dataset) and run this line manually it works.



**Jason Brownlee** December 26, 2016 at 7:39 am #

REPLY ↩

Sounds like a Python 3 issue Mike.

Replace

```
1 fold_size = len(dataset) / n_folds
```

With:

```
1 fold_size = int(len(dataset) / n_folds)
```





**Jason Brownlee** January 3, 2017 at 9:53 am #

REPLY ↩

I have updated the `cross_validation_split()` function in the above example to address issues with Python 3.



**Mohendra Roy** January 4, 2017 at 12:32 am #

REPLY ↩

How about to use of euclidian distance instead of calculating for each element in the dataset?



**Jason Brownlee** January 4, 2017 at 8:55 am #

REPLY ↩

What do you mean exactly? Are you able to elaborate?



**Selva Rani B** January 12, 2017 at 4:43 pm #

REPLY ↩

Thank you very much



**Jason Brownlee** January 13, 2017 at 9:09 am #

REPLY ↩

You're welcome.



**Sokrates** January 21, 2017 at 4:10 am #

REPLY ↩

Hi Jason,

Great tutorial on CART!

The results of decision trees are quite dependent on the training vs test data. With this in mind, how do I set the amount of training vs test data in the code right now to changes in the result? From what I can see, it looks like they are being set in the `evaluate_algorithm` method.

//Kind regards

Sokrates



**Jason Brownlee** January 21, 2017 at 10:37 am #

REPLY ↩

That is correct Sokrates.

The example uses k-fold cross validation to evaluate the performance of the algorithm on the dataset.

You can change the number of folds by setting the “`n_folds`” variable.

You can use a different resampling method, like train/test splits, see this post:

<http://machinelearningmastery.com/implement-resampling-methods-scratch-python/>



**Adeshina Alani** January 27, 2017 at 3:52 am #

REPLY ↩

Nice Post. I will like to ask if i this implementation can be used for time series data with only one feature



**Jason Brownlee** January 27, 2017 at 12:13 pm #

REPLY ↩

Yes it could, but the time series data would have to be re-framed as a supervised learning problem.

See this post for more information:

<http://machinelearningmastery.com/time-series-forecasting-supervised-learning/>



**vishal** January 30, 2017 at 5:06 am #

REPLY ↩

Really helpful. Thanks a lot for sharing.



**Jason Brownlee** February 1, 2017 at 10:18 am #

REPLY ↩

I'm glad you found the post useful vishal.



**elberiver** February 3, 2017 at 6:02 pm #

REPLY ↩

Hi Jason,

there is a minor point in your code. Specifically, in the following procedure:

```
_____  
# Build a decision tree  
def build_tree(train, max_depth, min_size):  
    root = get_split(dataset)  
    split(root, max_depth, min_size, 1)  
    return root  
_____
```

I think it should be `root = get_split(train)`, even though your code is still running correctly since `dataset` is the global variable.

Thank you for your nice posts.

I like your blog very much.

**Jason Brownlee** February 4, 2017 at 9:59 am #

REPLY ↩



I think you're right, nice catch!

I'll investigate and fix up the example.



**from Thailand** March 8, 2017 at 2:35 pm #

REPLY ↩

Thanks a lot Jason, really helpful



**Jason Brownlee** March 9, 2017 at 9:52 am #

REPLY ↩

I'm glad to hear that.



**eve** November 5, 2018 at 1:32 am #

REPLY ↩

Excuse me, I'm confused about the `get_split` function in your work. Actually, I encountered a problem that the function always returns a tree with only a depth of one even if I have two more features. As a novice, I found it a little difficult to work out the problem. So would you please answer the question for me? Just show me a direction is OK. Thanks.



**Jason Brownlee** November 5, 2018 at 6:18 am #

REPLY ↩

Perhaps your data does not benefit from more than one split?

Perhaps try a decision tree as part of the scikit-learn library?



**Amit Moondra** April 2, 2017 at 6:31 am #

REPLY ↩

I'm slowly going through your code and I'm confused about a line in your `get_split` function

```
groups = test_split(index, row[index], dataset)
```

Doesn't this only return the left group? It seems we need both groups to calculate the gini\_index?

Thank you.



**Jason Brownlee** April 2, 2017 at 6:34 am #

REPLY ↩

Hi Amit,

The `get_split()` function evaluates all possible split points and returns a dict of the best found split point, gini score, split data.



**Amit Moondra** April 2, 2017 at 12:15 pm #

REPLY ↩

After playing around with the code for a bit, I realized that function returns both groups (left and right) under one variable.



**Amit Moondra** April 2, 2017 at 9:59 am #

REPLY ↩

In the function `split`

if not left or not right:

```
node['left'] = node['right'] = to_terminal(left + right)
```

```
return
```

Why do you add (left + right)? Are you adding the two groups together into one group?

Thank you.



**Jason Brownlee** April 4, 2017 at 9:05 am #

REPLY ↩

Yes.



**Amit Moondra** April 2, 2017 at 12:30 pm #

REPLY ↩

Another question (line 132)

```
if isinstance(node['left'], dict):  
    return predict(node['left'], row)
```

isinstance is just checking if we have already created such a dictionary instance?

Thank you.



**Jason Brownlee** April 4, 2017 at 9:05 am #

REPLY ↩

It is checking if the type of the variable is a dict.



**Ann** April 3, 2017 at 9:25 am #

REPLY ↩

Hello,

I've been trying some stuff out with this code and I thought I was understanding what was going on but when I tried it on a dataset with binary values it doesn't seem to work and I can't figure out why. Could you help me out please?

Thanks.



**Jason Brownlee** April 4, 2017 at 9:11 am #

REPLY ↩

The example assumes real-valued inputs, binary or categorical inputs should be handled differently.

I don't have an example at hand, sorry.



**Dimple** April 17, 2017 at 1:37 am #

REPLY ↩

Hi

Could you tell me how decision trees are used for predicting an unknown function when a sample dataset is given. What i mean how it is used for regression?



**Jason Brownlee** April 17, 2017 at 5:15 am #

REPLY ↩

Good question, sorry, I don't have an example of decision trees for regression from scratch.



**Dimple** April 17, 2017 at 10:18 am #

REPLY ↩

How can we use weka for regression using decision trees?



**Jason Brownlee** April 18, 2017 at 8:28 am #

REPLY ↩

Consider using the search function of this blog.

See this post:

<http://machinelearningmastery.com/use-regression-machine-learning-algorithms-weka/>



**Joe** April 18, 2017 at 1:31 am #

REPLY ↩

Great article, this is exactly what I was looking for!



**Jason Brownlee** April 18, 2017 at 8:33 am #

REPLY ↩

I'm really glad to hear that Joe!



**ansar** April 19, 2017 at 3:13 am #

REPLY ↩

I am new to machine learning ... successfully ran the code with the given data set

Now I want to run it for my own data set ... will the algo always treat the last column as the column to classify?

thanks



**Jason Brownlee** April 19, 2017 at 7:54 am #

REPLY ↩

Yes, that is how it was coded.





**Ansar** April 20, 2017 at 2:53 am #

REPLY ↩

Thank you! It works beautifully



**Jason Brownlee** April 20, 2017 at 9:32 am #

REPLY ↩

Well done!



**Ansar** April 21, 2017 at 10:22 pm #

REPLY ↩

Apologies if I am taking too much time but I tried to run this algo on the below scenario with 10 folds

#

#[https://www.youtube.com/watch?v=eKD5gxPPeY0&list=PLBv09BD7ez\\_4temBw7vLA19p3tdQH6FY0](https://www.youtube.com/watch?v=eKD5gxPPeY0&list=PLBv09BD7ez_4temBw7vLA19p3tdQH6FY0)

#

#outlook

#

#1 = sunny

#2 = overcast

#3 = rain

#

#humidity

#

#1 = high

#2 = normal

#

#wind

#

#1 = weak  
#2 = strong  
#  
#play  
#  
#0 = no  
#1 = yes

The tree generated does not cater to x2, x3 variables (for some reason), just generates for x1 (what am I doing wrong?) ... the accuracy has dropped to 60%

[X1 < 1.000]  
[1.0]  
[1.0]  
[X1 < 1.000]  
[1.0]  
[1.0]  
[X1 < 1.000]  
[1.0]  
[1.0]  
[X1 < 1.000]  
[1.0]  
[1.0]  
[X1 < 3.000]  
[X1 < 1.000]  
[1.0]  
[1.0]  
[0.0]  
[X1 < 1.000]  
[1.0]  
[1.0]  
[X1 < 1.000]  
[1.0]  
[1.0]

```
[X1 < 1.000]
[1.0]
[1.0]
[X1 < 1.000]
[1.0]
[1.0]
[X1 < 1.000]
[1.0]
[1.0]
Scores: [100.0, 100.0, 0.0, 100.0, 0.0, 100.0, 0.0, 0.0, 100.0, 100.0]
Mean Accuracy: 60.000%
```



**Jason Brownlee** April 22, 2017 at 9:26 am #

REPLY ↩

Ensure that you have loaded your data correctly.



**Ansar** April 25, 2017 at 3:01 am #

REPLY ↩

Yes, working fine now 😊

Would love to get my hands on a script that would print the tree in a more graphical (readable) format. The current format helps but does get confusing at times.

Thanks a lot!



**Greg** May 6, 2017 at 10:15 am #

REPLY ↩

This will generate a graphviz dot file that you can use to generate a .png, jpeg etc.

e.g:

```
dot -Tpng graph1.dot > graph.png
```

Note it generates a new file each time it is called – graph1.dot ... graphN.dot

# code begin

```
def graph_node(f, node):
    if isinstance(node, dict):
        f.write(' %d [label=\"%X%d %d;\n' % ((id(node), id(node['left']))))
        f.write(' %d -> %d;\n' % ((id(node), id(node['right']))))
        graph_node(f, node['left'])
        graph_node(f, node['right'])
    else:
        f.write(' %d [label=\"%s\"]; \n' % ((id(node), node)))

def graph_tree(node):
    if not hasattr(graph_tree, 'n'): graph_tree.n = 0
    graph_tree.n += 1
    fn = 'graph' + str(graph_tree.n) + '.dot'
    f = open(fn, 'w')
    f.write('digraph {\n')
    f.write(' node[shape=box];\n')
    graph_node(f, node)
    f.write('}\n')
```



**Edgar Panganiban** July 22, 2018 at 10:41 pm #

REPLY ↩

Hello dude, Do I only need to put this codes at the end part?? You only define functions...How can I run this and on what parts of the code....Thanks



**katana** April 28, 2017 at 1:27 am <#>

REPLY

Thanks a lot for this, Dr. Brownlee!



**Jason Brownlee** April 28, 2017 at 7:47 am <#>

REPLY

I'm glad you found it useful.



**godavari** May 4, 2017 at 11:08 pm <#>

REPLY

excellent explanation



**Jason Brownlee** May 5, 2017 at 7:31 am <#>

REPLY

I'm glad you found it useful.



**King Deng** May 14, 2017 at 7:15 am <#>

REPLY

I'm implementing AdaBoost from scratch now, and I have a tough time understanding how to apply the sample weights calculated in each iteration to build the decision tree? I guess I should modify the gini index in this regard, but I'm not specifically sure how to do that. Could you shed some light? Thanks!



**Jason Brownlee** May 14, 2017 at 7:35 am <#>

REPLY

I offer a step-by-step example in this book:

<https://machinelearningmastery.com/master-machine-learning-algorithms/>

I would also recommend this book for a great explanation:

<http://www-bcf.usc.edu/~gareth/ISL/>



**Pavithra** May 19, 2017 at 7:10 pm #

REPLY ↩

Part of the code: `predicted = algorithm(train_set, test_set, *args)`

TypeError: 'int' object is not callable

Issue: I'm getting error like this. Please help me



**Jason Brownlee** May 20, 2017 at 5:36 am #

REPLY ↩

I'm sorry to hear that.

Ensure that you have copied all of the code without any extra white space.



**Luis Ilabaca** May 25, 2017 at 9:23 am #

REPLY ↩

hey jason

honestly dude stuff like this is no joke man.

I did BA in math and one year of MA in math

then MA in Statistical Computing and Data Mining

and then sas certifications and a lot of R and man let me tell you,

when I read your work and see how you have such a strong understanding of the unifications of all the different fields needed to be successful at applying machine learning.

you my friend, are a killer.



**Jason Brownlee** June 2, 2017 at 11:40 am #

REPLY ↩

Thanks.



**Saurabh** May 25, 2017 at 7:16 pm #

REPLY ↩

Hello Sir!!

First of all Thank You for such a great tutorial.

I would like to make a suggestion for function `get_split()`-

In this function instead of calculating gini index considering every value of that attribute in data set , we can just use the mean of that attribute as the `split_value` for `test_split` function.

This is just my idea please do correct me if this approach is wrong.

Thank You!!



**Jason Brownlee** June 2, 2017 at 11:43 am #

REPLY ↩

Try it and see.

**Habiba** June 4, 2017 at 6:36 am #

REPLY ↩



Hello Sir,

I am a student and i need to develop an algorithm for both Decision Tree and Ensemble(Preferably,Random Forest) both using python and R. i really need the book that contains everything that is,the super bundle.

Thank you so very much for the post and the tutorials. They have been really helpful.



**Jason Brownlee** June 4, 2017 at 7:55 am #

REPLY ↩

You can grab the super bundle here:

<https://machinelearningmastery.com/super-bundle/>



**Dev** June 21, 2017 at 9:17 pm #

REPLY ↩

Hello Sir,

First of all Thank You for such a great tutorial

I am new to machine learning and python as well.I'm slowly going through your code

I have a doubt in the below section

# Build a decision tree

```
def build_tree(train, max_depth, min_size):  
    root = get_split(train)  
    split(root, max_depth, min_size, 1)  
    return root
```

In this section the “split” function returns “none”,Then how the changes made in “split” function are reflecting in the variable “root”

To know what values are stored in “root” variable, I run the code as below

# Build a decision tree

```
def build_tree(train, max_depth, min_size):  
    root = get_split(train)  
    print(root) — Before calling split function
```



```
split(root, max_depth, min_size, 1)
print(root) — after calling split function
```

The values in both cases are different. But I'm confused how it happens  
Can anybody help me out please?  
Thank you.



**Jason Brownlee** June 22, 2017 at 6:05 am #

REPLY ↩

The split function adds child nodes to the passed in root node to build the tree.



**Xinglong Li** June 24, 2017 at 12:48 pm #

REPLY ↩

Hello sir

Thanks for this tutorial.

I guess that when sum the Gini indexes of subgroups into a total one, their corresponding group sizes should be taken into consideration, that is, I think it should be a weighted sum.



**Michael** August 5, 2017 at 9:07 am #

REPLY ↩

I think I have the same question. The Gini Index computed in the above examples are not weighted by the size of the partitions. Shouldn't they be? I posted a detailed example, but it has not been excepted (yet?).



**sanju** June 27, 2017 at 10:33 pm #

REPLY ↩

Hello Sir,

Assume we have created a model using the above algorithm.

Then if we go for prediction using this model ,will this model process on the whole training data.?

Thank you.



**Jason Brownlee** June 28, 2017 at 6:25 am #

REPLY ↩

We do not need to evaluate the model when we want to make predictions. We fit it on all the training data then start using it.

See this post:

<http://machinelearningmastery.com/train-final-machine-learning-model/>



**Rohit** June 28, 2017 at 7:38 pm #

REPLY ↩

Hello Sir,

Assume I have 1 million training data,and I created and saved a model to predict whether the patient is diabetic or not ,Every thing is ok with the model and the model is deployed in client side(hospital).consider the case below

1. If a new patient come then based on some input from the patient the model will predict whether the patient is diabetic or not.
- 2.If another patient come then also our model will predict

my doubt is in the both case will the model process on one million training data.?

that is if 100 patient come at different time, will this model process 100 times over one million data(training data).?

Thank you



**Jason Brownlee** June 29, 2017 at 6:33 am #

REPLY ↩

No, after you fit the model you save it. Later you load it and make a prediction.

See this post on creating a final model for making predictions:

<http://machinelearningmastery.com/train-final-machine-learning-model/>



**Michael Shparber** June 29, 2017 at 7:26 am #

REPLY ↩

Excellent, thank you for sharing!

Are there tools that allow to do this without ANY coding?

I mean drag-n-drop / right-click-options.

It is very useful to understand behind-the-scenes but much faster in many cases to use some sort of UI.

What would you recommend?

Thank you,

Michael



**Jason Brownlee** June 29, 2017 at 7:48 am #

REPLY ↩

Yes, Weka:

<http://machinelearningmastery.com/start-here/#weka>



**Ali Mesbahi** July 5, 2017 at 12:35 am #

REPLY ↩

Hello,

Excellent post. It has been really helpful for me!

Is there a similar article/tutorial for the C4.5 algorithm?

Is there any implementations in R?

Thank you,  
Ali.



**Jason Brownlee** July 6, 2017 at 10:20 am #

REPLY ↩

See here for R examples:

<http://machinelearningmastery.com/non-linear-classification-in-r-with-decision-trees/>



**Jeet** July 9, 2017 at 9:35 pm #

REPLY ↩

```
dataset = list(lines)
```

Error: iterator should return strings, not bytes (did you open the file in text mode?)

How can i solve this issue ?



**Jason Brownlee** July 11, 2017 at 10:17 am #

REPLY ↩

This might be a Python version issue.

Try changing the loading of the file to 'rt' format.



**AJENG SHILVIE NURLATIFAH** July 12, 2017 at 6:03 pm #

REPLY ↩

Hi jason, thanks for sharing about this algorithm,  
i have a final task to use some classification model like decision tree to fill missing values in data set, is it possible ?

Thank you



**Jason Brownlee** July 13, 2017 at 9:50 am #

REPLY ↩

Yes, see this post for ideas:

<http://machinelearningmastery.com/handle-missing-data-python/>



**Nghi** July 21, 2017 at 12:43 am #

REPLY ↩

Hi Jason,

I tried running the modified version by adding labels for the data

#to add label but not part of the training

```
data = pd.read_csv(r'H:\Python\Tree\data_banknote_authentication.txt', header = None )
```

```
dataset = pd.DataFrame(data)
```

```
dataset.columns = ['var', 'skew', 'curt', 'ent', 'bin']
```

so on the last step, i only ran

```
n_folds = 5
```

```
max_depth = 5
```

```
min_size = 10
```

```
scores = evaluate_algorithm(dataset, decision_tree, n_folds, max_depth, min_size)
```

```
print('Scores: %s' % scores)
```

```
print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))
```

and i got the error message saying

```
ValueError: empty range for randrange()
```

Could you help explain why?

Also on this section to load the CSV file

```
# Load a CSV file
def load_csv(filename):
    file = open(filename, "rb")
    lines = reader(file)
    dataset = list(lines)
    return dataset
```

could you help explain how to add the location of the file if it's on local drive?

Thank you very much,



**preet shah** July 24, 2017 at 9:04 pm #

REPLY ↩

What changes would you suggest i should make to solve the following problem:

A premium payer wants to improve the medical care using Machine Learning. They want to predict next events of Diagnosis, Procedure or Treatment that is going to be happen to patients. They've provided with patient journey information coded using ICD9.

You have to predict the next 10 events reported by patient in order of occurrence in 2014.

Data Description

There are three files given to download: train.csv, test.csv and sample\_submission.csv

The train data consists of patients information from Jan 2011 to Dec 2013. The test data consists of Patient IDs for the year 2014.

Variable Description

ID Patient ID

Date Period of Diagnosis

Event Event ID (ICD9 Format) – Target Variable



**Jason Brownlee** July 25, 2017 at 9:42 am #

REPLY ↩

Sounds like homework.

If not, I recommend this process for working through new predictive modeling problems:

<http://machinelearningmastery.com/start-here/#process>



**Ikib Kilam** July 26, 2017 at 10:14 pm #

REPLY ↩

Jason,

Thanks for this wonderful example. I rewrote your code, exactly as you wrote it, but cannot replicate your scores and mean accuracy. I get the following:

Scores: [28.467153284671532, 37.591240875912405, 76.64233576642336, 68.97810218978103, 70.43795620437956]

Mean Accuracy: 56.42335766423357

As another run, I changed nFolds to be 7, maxDepth to be 4 and minSize to be 4 and got:

Scores: [40.30612244897959, 38.775510204081634, 37.755102040816325, 71.42857142857143, 70.91836734693877, 70.91836734693877, 69.89795918367348]

Mean Accuracy: 57.142857142857146

I have tried all types of combinations for nFolds, minSize and maxDepth, and even tried stopping randomizing the selection of data instances into Folds. However, my scores do not change, and my mean accuracy has never exceeded 60%, and in fact consistently is in the 55%+ range. Strangely, my first two scores are low and then they increase, though not to 80%. I am at my wits end, why I cannot replicate your high (80%+) scores/accuracy.

Any ideas on what might be happening? Thanks for taking the time to read my comment and again thanks for the wonderful illustration.

Ikib



**Jason Brownlee** July 27, 2017 at 8:05 am #

REPLY ↩

That is odd.

Perhaps a copy-paste error somewhere? That would be my best guess.



**buzznizz** August 4, 2017 at 9:22 pm #

REPLY ↩

Hi Jason

It's not usefull splitting on groups with size 1 (as you do now with groups size 0) and you can make them directly terminal.

BR



**Michael** August 5, 2017 at 8:22 am #

REPLY ↩

Thanks Jason for a very well put together tutorial!

Read through all the replies here to see if someone had asked the question I had. I think the question Xinglong Li asked on June 24, 2017 is the same one I have, but it wasn't answered so I'll rephrase:

Isn't the calculation of the Gini Index suppose to be weighted by the count in each group?

For example, in the table just above section "3. Build a Tree", on line 15, the output lists:

X2 < 2.209 Gini=0.934

When I compute this by hand, I get this exact value if I do NOT weight the calculation by the size of the partitions. When I do this calculation the way I think it's supposed to be done (weighting by partition size), I get Gini=0.4762. This is how I compute this value:

1) Start with the test data sorted by X1 so we can easily do the split (expressed in csv format):

X1,X2,Y

7.444542326,0.476683375,1

1.728571309,1.169761413,0

2.771244718,1.784783929,0



— group 1 above this line, group 2 below this line —

2.999208922,2.209014212,0

3.961043357,2.61995032,0

3.678319846,2.81281357,0

7.497545867,3.162953546,1

10.12493903,3.234550982,1

6.642287351,3.319983761,1

9.00220326,3.339047188,1

2) Compute the proportions (computing to 9 places, displaying 4):

$P(1, 0) = \text{group 1 (above line), class 0} = 0.6667$

$P(1, 1) = \text{group 1 (above line), class 1} = 0.3333$

$P(2, 0) = \text{group 2 (below line), class 0} = 0.4286$

$P(2, 1) = \text{group 2 (below line), class 1} = 0.5714$

3) Compute Gini Score WITHOUT weighting (computing to 9 places, displaying 4):

Gini Score =

$[P(1, 0) \times (1 - P(1, 0))] +$

$[P(1, 1) \times (1 - P(1, 1))] +$

$[P(2, 0) \times (1 - P(2, 0))] +$

$[P(2, 1) \times (1 - P(2, 1))] =$

$0.2222 + 0.2222 + 0.2449 + 0.2449 = 0.934$  (just what you got)

4) Compute Gini Score WITH weighting (computing to 9 places, displaying 4):

$[(3/10) \times (0.2222 + 0.2222)] + [(7/10) \times (0.2449 + 0.2449)] = 0.4762$

Could you explain why you don't weight by partition size when you compute your Gini Index?

Thanks!

---

Jason Brownlee August 6, 2017 at 7:34 am #

REPLY ↩



Hi Michael, I believe the counts are weighted.

See “proportion” both in the description of the algorithm and the code calculation of gini for each class group.

See “an introduction to statistical learning with applications in r” pages 311 onwards for the equations.

Does that help?



**Michael** August 9, 2017 at 7:41 am #

REPLY ↩

Hi Jason, Thanks for the reply. After reading and re-reading pages 311 and 312 several times, it seems to me that equation (8.6) in the ISL (love this book BTW, but in this rare case, it is lacking some important details) should really have a subscript  $m$  on the  $G$  because it is computing the Gini score for a particular region. Notice that in this equation: 1) the summation is over  $K$  (the total number of classes) and 2) the  $m_k$  subscript on  $\hat{p}$ . These imply that both the proportion ( $\hat{p}_{m_k}$ ) and  $G$  are with respect to a particular region  $m$ .

Eqn. (8.6) in the ISL is correct, but it's not the final quantity that should be used for determining the quality of the split. There is an additional step (not described in the ISL) which needs to be done: weighting the Gini scores by the size of the proposed split regions as shown in this example:

[https://www.researchgate.net/post/How\\_to\\_compute\\_impurity\\_using\\_Gini\\_Index](https://www.researchgate.net/post/How_to_compute_impurity_using_Gini_Index)

I think the `gini_index` function should look something like what is shown below. This version gives me the values I expect and is consistent with how the gini score of a split is computed in the above example:

```
1 def gini_index2(groups, class_values):
2     group_sizes = [len(group) for group in groups]
3     sample_count = sum(group_sizes)
4     gini_groups = []
5     for idx, group in enumerate(groups):
6         gini_group = 0.0
7         group_size = len(group)
8         for class_value in class_values:
9             if group_size == 0:
10                continue
11             proportion = [row[-1] for row in group].count(class_value) / \
12                           float(group_size)
13             gini_group += (proportion * (1.0 - proportion))
14             # weight the group gini score by the size of the group
15             partition_weight = group_sizes[idx] / sample_count
```

```
16 gini_groups.append(partition_weight * gini_group)
17
18 return sum(gini_groups)
```

Thoughts?



**Michael** August 9, 2017 at 7:44 am #

REPLY ↩

Sorry about the loss of indentation in the code... Seems like the webapp parses these out before posting.



**Jason Brownlee** August 10, 2017 at 6:35 am #

I added some pre tags for you.



**Jason Brownlee** August 10, 2017 at 6:35 am #

REPLY ↩

I'll take a look, thanks for sharing.



**Michael** August 10, 2017 at 11:56 am #

Thanks for your eyeballs. The original code I proposed works, but it has some unnecessary stuff in it. This is a cleaner (less lines) implementation (hopefully the pre tags I insert work...):

```
1 # groups - tuple or list of samples corresponding to a split group
2 # class_values - tuple or list of unique class labels
3 # TODO class_values can be obtained from groups - no need to pass in
4 def gini_index2(groups, class_values):
5     sample_count = sum([len(group) for group in groups])
```

```

6     gini_groups = []
7     for group in groups:
8         gini_group = 0.0
9         group_size = len(group)
10        for class_value in class_values:
11            if group_size == 0:
12                continue
13            proportion = [row[-1] for row in group].count(class_value) / \
14                          float(group_size)
15            gini_group += (proportion * (1.0 - proportion))
16            # weight the group gini score by the size of the group
17            partition_weight = group_size / sample_count
18            gini_groups.append(partition_weight * gini_group)
19
20    return sum(gini_groups)

```



**Jason Brownlee** August 10, 2017 at 4:42 pm #

Thanks. Added to my trello to review.

Update: Yes, you are 100% correct. Thank you so much for pointing this out and helping me see the fault Michael! I really appreciate it!

I did some homework on the calculation (checked some textbooks and read sklearn's source) and wrote a new version of the gini calculation function from scratch. I then update the tutorial.

I think it's good now, but shout if you see anything off.



**nehasharma** August 18, 2017 at 6:24 am #

REPLY ↩

Hi Jason, In your code:

```
print(gini_index([[[1, 1], [1, 0]], [[1, 1], [1, 0]], [0, 1]]))
```

how should i interpret the number of ZEROES and ONES(distribution of classes in group)?I am new to this..



**Jason Brownlee** August 18, 2017 at 6:56 am #

REPLY ↩

Great question!

No, each array is a pattern, the final value in each array is a class value. The final array are the valid classes.



**John Sobola** October 22, 2017 at 10:36 pm #

REPLY ↩

I did not notice any difference in the split point and prediction when I used both the old and revised gini codes



**aaaaa** August 28, 2017 at 12:08 am #

REPLY ↩

how to apply this for discrete value ?



**scssek** September 15, 2017 at 1:33 pm #

REPLY ↩

Do you recommend this decision tree model for binary file based data?



**Jason Brownlee** September 16, 2017 at 8:37 am #

REPLY ↩

I do not recommend an algorithm. I recommend testing a suite of algorithms to see what works best.

See this post:

<http://machinelearningmastery.com/a-data-driven-approach-to-machine-learning/>



**Ann** September 18, 2017 at 12:30 am #

REPLY ↩

Good Day sir!

I would like to ask why does my code run around 30-46 minutes to get the mean accuracy? I am running about 24100 rows of data with 3 columns. Is it normally this slow? Thank you very much!



**Jason Brownlee** September 18, 2017 at 5:47 am #

REPLY ↩

Yes, because this is just a demonstration.

For a more efficient implementation, use the implementation is scikit-learn.

For more on why you should only code machine learning algorithms from scratch for learning, see this post:

<http://machinelearningmastery.com/dont-implement-machine-learning-algorithms/>



**Jarich** September 19, 2017 at 12:12 am #

REPLY ↩

Can you leave an example with a larger hard-coded decision tree, with 2 or even 3 stages?

I can't seem to get the syntax right to work with larger decision trees.



**Jason Brownlee** September 19, 2017 at 7:47 am #

REPLY ↩

Thanks for the suggestion. Note that the code can develop such a tree.

**jarich** September 19, 2017 at 5:42 pm #

REPLY ↩



Yeah I just found that out myself, might be better to not even give the example and let us think for some time.



**Maria** September 22, 2017 at 1:16 pm #

REPLY ↩

I would like to ask if there is a way to store the already learned decision tree? so that when the predict function is called for one test data only, it can be run at a much faster speed. Thank you very much!



**Jarich** September 22, 2017 at 5:54 pm #

REPLY ↩

Hey Maria, you could save it to a .txt file and then read it back in for the prediction. I did it that way and it works flawlessly!



**Maria** September 22, 2017 at 11:53 pm #

REPLY ↩

oh okay. I will save the result for build\_tree to a text file? Thank you very much for this information! 😊



**Jason Brownlee** September 23, 2017 at 5:37 am #

REPLY ↩

Great tip Jarich.



**Jason Brownlee** September 23, 2017 at 5:35 am #

REPLY ↩

Yes, I would recommend using the sklearn library then save the fit model using pickle.

I have examples of this on my blog, use the search.



**chakri** September 26, 2017 at 2:01 am #

REPLY ↩

Jason : Thank you for a great article. Can you provide me the code for generating the tree. I have got scores and accuracy and would like to view the decision tree. Please provide the code related to generating tree



**chakri** September 26, 2017 at 2:02 am #

REPLY ↩

I would need to code using graphviz



**Jason Brownlee** September 26, 2017 at 5:39 am #

REPLY ↩

Thanks for the suggestion.



**Maria** September 30, 2017 at 11:45 pm #

REPLY ↩

I would like to know if this tree can also handle multiple classification aside from 0 or 1? for example I need to classify if it is 1,2,3,4,5 Thank you very much!



**Jason Brownlee** October 1, 2017 at 9:07 am #

REPLY ↩

It could be extended to multiple classes. I do not have an example sorry.





**Leo** October 4, 2017 at 3:45 am #

REPLY ↩

I can't get started.

I use Python 3.5 on Spyder 3.0.0.

Your code doesn't read the dataset and an error comes out:

Error: iterator should return strings, not bytes (did you open the file in text mode?)

I'm a beginner in both Python and ML.

Can you help?

Thank you, Leo



**Jason Brownlee** October 4, 2017 at 5:48 am #

REPLY ↩

The code was developed for Python 2.7. I will update it for Python 3 in the future.

Perhaps start with Weka where no programming is required:

<https://machinelearningmastery.com/start-here/#weka>



**Andisheh** July 4, 2018 at 11:48 am #

REPLY ↩

Hello, I am a beginner and I have the same problem, too. Why are codes different for Python 2 and 3? Shall I save the file as text or csv if I am using your codes? What should I do now? Thank you



**Jason Brownlee** July 4, 2018 at 2:56 pm #

REPLY ↩

Python 2 and 3 are different programming language. The example is intended for Python2 and requires modification to work with Python3.

I will make those changes in coming months.



**lightbandit** October 12, 2017 at 4:18 am #

REPLY ↩

I have a dataset with titles on both axes (will post example below) and they have 1's and 0's signifying if the characteristic is present or not. Where would you suggest I start with altering your code in order to fit this dataset? Right now I am getting an error with the converting string to float function.

white blue tall

ch1 0 1 1

ch2 1 0 0

ch3 1 0 1



**Jason Brownlee** October 12, 2017 at 5:36 am #

REPLY ↩

I'm not sure I understand your problem, sorry.

Perhaps this process will help you define and work through your problem end to end:

<https://machinelearningmastery.com/start-here/#process>



**Liu Yong** October 15, 2017 at 7:32 pm #

REPLY ↩

This is great!

May I know do you plan to introduce cost-complexity pruning for CART?

Many thanks!



**Jason Brownlee** October 16, 2017 at 5:42 am #

REPLY ↩

Not at this stage. Perhaps you could post some links?



**Revi** October 19, 2017 at 3:27 am #

REPLY ↩

Hello sir, i was trying to use your code on my data, but i find a problem which said ValueError: invalid literal for float():  
1;0.766126609;45;2;0.802982129;9120;13;0;6;0;2 . How could i fix this?



**Frackson** October 21, 2017 at 10:27 pm #

REPLY ↩

Hello Sir,

Thanks for the tutorial. It is quite clear and concise. However, how best can you advise on altering it to suit the following data set:  
my\_data=[['slashdot','USA','yes',18,'None'], ['google','France','yes',23,'Premium'], ['digg','USA','yes',24,'Basic'], ['kiwitobes','France','yes',23,'Basic'],  
['google','UK','no',21,'Premium'], ['(direct)','New Zealand','no',12,'None'], ['(direct)','UK','no',21,'Basic'], ['google','USA','no',24,'Premium'],  
['slashdot','France','yes',19,'None'], ['digg','USA','no',18,'None'], ['google','UK','no',18,'None'], ['kiwitobes','UK','no',19,'None'], ['digg','New  
Zealand','yes',12,'Basic'], ['slashdot','UK','no',21,'None'], ['google','UK','yes',18,'Basic'], ['kiwitobes','France','yes',19,'Basic']



**Jason Brownlee** October 22, 2017 at 5:20 am #

REPLY ↩

This tutorial is advanced.

Perhaps you would be better suited starting with Weka:  
<https://machinelearningmastery.com/start-here/#weka>

Or scikit-learn library in Python:  
<https://machinelearningmastery.com/start-here/#python>



**John Sobola** October 22, 2017 at 10:40 pm <#>

REPLY

Jason,

Thanks so much for this code. I was able to adapt for my application but can you suggest how to plot the tree diagram in place of the build tree code?

Thanks

John



**Jason Brownlee** October 23, 2017 at 5:43 am <#>

REPLY

Sorry John, I don't have code to plot the tree.



**Steven Lee** October 25, 2017 at 8:22 am <#>

REPLY

Hey Jason,

Thank you very much for this tutorial. It has been extremely helpful in my understanding of decision trees. I have one point of inquiry which requires clarification. In the 'def split (node, max\_depth, min\_size, depth)' method I can see that you recursively split the left nodes until the max depth or min size condition is met. When you perform the split you add 1 to the current depth to but once you move to the next if statement to process the first right child/partition there is no function that resets the depth.

Is this because in a recursive function you are saving new depth values for every iteration? Meaning unique depth values are saved at each recursion?



**Jason Brownlee** October 25, 2017 at 3:58 pm <#>

REPLY

Depth is passed down each line of the recursion.



**Steven Lee** October 25, 2017 at 9:00 am #

REPLY ↩

Also how is the return of build tree read?

i.e.:

in:

```
tree = build_tree(dataset, 10, 1)
print tree
```

out:

```
{'index': 0, 'right': {'index': 0, 'right': {'index': 0, 'right': 1, 'value': 7.497545867, 'left': 1}, 'value': 7.497545867, 'left': {'index': 0, 'right': 1, 'value': 7.444542326, 'left': 1}}, 'value': 6.642287351, 'left': {'index': 0, 'right': {'index': 0, 'right': 0, 'value': 2.771244718, 'left': 0}, 'value': 2.771244718, 'left': 0}}
```



**Jason Brownlee** October 25, 2017 at 4:01 pm #

REPLY ↩

Each node has references to left and right nodes.



**Henrik** October 31, 2017 at 12:03 am #

REPLY ↩

Hey,

I have a dataset with numerical and categorical values, so I modified some of the code in load\_csv and main to convert categorical values to numerical:

```
#load_csv
def load_csv(filename):
    #file = open(filename, "r")
    headers = ["location", "w", "final_margin", "shot_number", "period", "game_clock", "shot_clock", "dribbles", "touch_time",
```

```

"shot_dist", "pts_type", "close_def_dist", "target"]
df = pd.read_csv(filename, header=None, names=headers, na_values="?")

cleanup_nums = {"location": {"H": 1, "A": 0},
"w": {"W": 1, "L": 0},
"target": {"made": 1, "missed": 0}}
df.replace(cleanup_nums, inplace=True)
df.head()

obj_df=list(df.values.flatten())
return obj_df

#main:
# Test CART on Basketball dataset
seed(1)
# load and prepare data
filename='data/basketball.train.csv'
dataset = load_csv(filename)
i=0
new_list=[]
while i<len(dataset):
new_list.append(dataset[i:i+13])
i+=13
n_folds = 5
max_depth = 5
min_size = 10
scores = evaluate_algorithm(new_list, decision_tree, n_folds, max_depth, min_size)
print('Scores: %s' % scores)
print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))

```

The program is still working for your dataset, but when i try to run it on my own, the program freezes in the listcomp in the line `p = [row[-1] for row in group].count(class_val) / size`, in the method `gini_index`.

Do you have any idea how i can break this infinite loop?

Thanks



**Jason Brownlee** October 31, 2017 at 5:34 am #

REPLY ↩

Perhaps get it working with sklearn first and better understand your data:

<https://machinelearningmastery.com/start-here/#python>



**amitabh** November 30, 2017 at 6:28 pm #

REPLY ↩

Thanks it helps!



**Jason Brownlee** December 1, 2017 at 7:27 am #

REPLY ↩

You're welcome.



**adan** December 11, 2017 at 5:12 pm #

REPLY ↩

I don't get this part row [-1] for row in group. please explain what's happening here



**Jason Brownlee** December 12, 2017 at 5:21 am #

REPLY ↩

You can learn more about Python array indexing in this post:

<https://machinelearningmastery.com/index-slice-reshape-numpy-arrays-machine-learning-python/>



**Khalida** December 15, 2017 at 2:22 am <#>

REPLY

Just thanks <3



**Jason Brownlee** December 15, 2017 at 5:37 am <#>

REPLY

You're welcome.



**John Wang** January 3, 2018 at 3:45 pm <#>

REPLY

Really helpful, thanks a lot !



**Jason Brownlee** January 3, 2018 at 3:46 pm <#>

REPLY

Thanks, I'm glad to hear that.



**Chetan** February 1, 2018 at 9:59 pm <#>

REPLY

Easy to understand, no need of any background knowledge. Excellent Article

**Jason Brownlee** February 2, 2018 at 8:19 am <#>

REPLY





Thanks,



**Nandit Khosa** February 10, 2018 at 3:45 am #

REPLY ↩

Great tutorial, can you help me using entropy criteria for splitting rather than gini index. Any example for that. I tried but not working for me



**Jason Brownlee** February 10, 2018 at 8:59 am #

REPLY ↩

Sorry, I don't have an example.



**Kevin Sequeira** October 17, 2018 at 5:24 pm #

REPLY ↩

Hi Nandit,

I tried out Jason's algorithm along with Entropy as cost function. Might be able to help you.



**Hitesh** January 21, 2019 at 11:31 am #

REPLY ↩

@Kevin Sequeira

Hi Kevin,

I tried using entropy – information gain criteria in Jason's algorithm, but it's not working for me. Please, can you provide an example of its implementation?



**Mala Gupta** February 14, 2018 at 1:06 pm #

REPLY ↩

Hello Sir,

Is it possible to visualize the above tree? is it possible implement using Graphviz? Can you please provide an example to visualize the above decision tree?



**Jason Brownlee** February 14, 2018 at 2:44 pm #

REPLY ↩

Sure you can. Sorry, I don't have an example.

Perhaps you can use a third-part CART library with viz builtin, such as R or Weka.



**dpk** March 2, 2018 at 4:33 am #

REPLY ↩

Hi –

I notice that `to_terminal` is basically the Zero Rule algorithm. Is that just a coincidence (i.e. it might just be the least stupid thing to do with small data sets)? Or is there a deeper idea here that when you cut down the tree to whatever minimum data size you choose, you might want to apply some other predictive algorithm to the remaining subset (sort of using a decision tree to prefilter inputs to some other method)?

(If it's just a coincidence, then I guess I'm guilty of over-fitting... 😊 )

Thanks.



**Jason Brownlee** March 2, 2018 at 5:36 am #

REPLY ↩

Just a coincidence, well done on noting it!



**David Urive** March 8, 2018 at 2:36 pm <#>

REPLY

Im having trouble trying to classify the iris.csv dataset, how would you suggest modifying the code to to load in the attributes and a non numeric class?



**Jason Brownlee** March 8, 2018 at 2:56 pm <#>

REPLY

Load the data as a mixed numpy array, then covert the strings to integers via a label encoding.

I have many examples on the blog.



**harry** March 8, 2018 at 4:39 pm <#>

REPLY

I am having loading dataset it says the following error " iterator should return strings, not bytes (did you open the file in text mode?"



**Jason Brownlee** March 9, 2018 at 6:19 am <#>

REPLY

Are you able to confirm that you are using Python 2.7?



**Suyog** March 15, 2018 at 2:31 pm <#>

REPLY

A typo under the subheading Banknote Dataset...

{The dataset contains 1,372 with 5 numeric variables.} should be

{The dataset contains 1,372 rows with 5 numeric variables.}



**Jason Brownlee** March 15, 2018 at 2:52 pm #

REPLY ↩

Thanks, fixed.



**AKSHAY TENKALE** March 17, 2018 at 2:38 am #

REPLY ↩

outcomes = [row[-1] for row in group]  
what does this code snippet does.?

for Regression tree, to\_terminal(group) function should return mean of target attribute values, but I am getting group variable having all NaN values.  
Please suggest any alternatives



**Jason Brownlee** March 17, 2018 at 8:42 am #

REPLY ↩

Collects all the output values.

Perhaps this post will help you understand lists and array indexing in Python:

<https://machinelearningmastery.com/index-slice-reshape-numpy-arrays-machine-learning-python/>

The example was written for Python 2.7, confirm you are using this version.



**Arvind** March 18, 2018 at 6:28 pm #

REPLY ↩

Thanks, This is extremely useful.

We are trying to make a decision based on the attributes found in a WebElement on a WebPage for generating dynamic test cases.

for example, we want to feed some properties like type = "text" and name ="email" etc... and then come to a decision that its a text filed and generate test cases applicable for textfield. An example of what we want to read properties from

Any pointers, how to begin ? Any help would be appreciated. Basically how to start preparing the training dataset for links, images and other element types..



**Jason Brownlee** March 19, 2018 at 6:06 am #

REPLY ↩

Perhaps this post will help define the inputs and outputs to the model:

<http://machinelearningmastery.com/how-to-define-your-machine-learning-problem/>



**Kelvin** April 6, 2018 at 5:58 am #

REPLY ↩

Hi Jason,

I must be blind. Once an attribute is used in a split, I don't see you remove it from the next recursive split/branching.

Thanks!



**Jason Brownlee** April 6, 2018 at 6:38 am #

REPLY ↩

It can be reused if it can add value, but note the subset of data at the next level down will be different – e.g. having been split.



**Sergey Kojoian** April 7, 2018 at 5:40 am #

REPLY ↩

Hello Jason,

Thanks for the great tutorial.

I am a bit confused by the definition of gini index.

$\text{gini\_index} = \sum(\text{proportion} * (1.0 - \text{proportion}))$  is just  $2 * p * q$  where  $p$  is the proportion of one class in the group and  $q$  is the proportion of the other, i.e.  
 $\text{gini\_index} = \sum(\text{proportion} * (1.0 - \text{proportion})) == 2p * (1 - p)$

Correct?



**Jason Brownlee** April 7, 2018 at 6:37 am #

REPLY ↩

What makes you say that?



**Adez** April 28, 2018 at 8:53 pm #

REPLY ↩

Nice tutorial sir. I really enjoy going through your work sir. It has helped me to understand what is happening under the hood better. How can someone plot the logloss for this tutorial?



**Jason Brownlee** April 29, 2018 at 6:25 am #

REPLY ↩

Thanks.

How would a plot of logloss work for a decision tree exactly? As it is being constructed? It might not be appropriate.



**eye\_water** May 9, 2018 at 1:18 pm #

REPLY ↩

Hi Jason,  
thanks for your tutorial, i want to translate your essay to Chinese not for business just for share.

Can i?

If i can, there are something i should notice?



**Jason Brownlee** May 9, 2018 at 2:57 pm #

REPLY ↩

Please do not translate and republish my content.



**eye\_water** May 9, 2018 at 4:57 pm #

REPLY ↩

OK, thanks for your reply.



**Parikshit Bhide** June 2, 2018 at 5:30 pm #

REPLY ↩

I'd recommend that for the recursive split explanation, you may add a note regarding Python creating a separate copy of the 'node' dictionary on each recursive call. It is not trivially known to people without extensive programming experience and can confuse them if they try to evaluate the expressions manually for the small 'dataset'. Thanks otherwise for such a great article.



**Jason Brownlee** June 3, 2018 at 6:21 am #

REPLY ↩

Thanks for the suggestion.



**Parikshit Bhide** June 3, 2018 at 1:39 pm #

REPLY ↩





Welcome. I ran through the whole recursive code for the small dataset manually myself just to get a hang of the flow. Used a different variable name for 'node's at each depth level. Here's the code, incase it is helpful to anyone:

```
root = get_split(dataset)
max_depth = 3
min_size = 1
depth = 1
node = root.copy()
left, right = node['groups']
del(node['groups'])
node['left'] = l1 = get_split(left)
depth = depth + 1
l1left, l1right = l1['groups']
del(l1['groups'])
len(l1left) <= min_size
l1['left'] = to_terminal(l1left)
l1['right'] = l1r2 = get_split(l1right)
depth = depth + 1
l1r2left, l1r2right = l1r2['groups']
del(l1r2['groups'])
not l1r2left or not l1r2right
l1r2['left'] = l1r2['right'] = to_terminal(l1r2left + l1r2right)
depth = depth - 1
node['right'] = r1 = get_split(right)
r1left, r1right = r1['groups']
del(r1['groups'])
r1['left'] = r1l2 = get_split(r1left)
depth = depth + 1
r1l2left, r1l2right = r1l2['groups']
del(r1l2['groups'])
r1l2['left'], r1l2['right'] = to_terminal(r1l2left), to_terminal(r1l2right)
depth = depth - 1
r1['right'] = r1r2 = get_split(r1right)
```

```
depth = depth + 1
r1r2left, r1r2right = r1r2['groups']
del(r1r2['groups'])
not r1r2left or not r1r2right
r1r2['left'] = r1r2['right'] = to_terminal(r1r2left + r1r2right)
```



**Jason Brownlee** June 4, 2018 at 6:22 am #

REPLY ↩

Nice work!



**Vincent** June 10, 2018 at 8:54 am #

REPLY ↩

What does the number after each X mean? (X1, X2, etc) In the `print_tree` function it stands for `node['index']`. Is this the gini-index?

Why do only two indexes appear around your example-data?

I tried some other dataset and this is the result:

```
[X4 < 23.570]
[X1 < 1143.631]
[X1 < 1096.228]
[X1 < 905.490]
[X1 < 714.853]
[X1 < 524.248]
[0]
[0]
[0]
[0]
[0]
[X1 < 1143.631]
[0]
```

[0]  
[X1 < 1373.940]  
[X1 < 1292.075]  
[X1 < 1276.251]  
[X1 < 1141.360]  
[X1 < 912.083]  
[1]  
[1]  
[1]  
[1]  
[1]  
[X1 < 1373.940]  
[1]  
[1]



**Jason Brownlee** June 11, 2018 at 6:02 am #

REPLY ↩

Xn is the variable number, the floating point value number after is the observation of that variable chosen to be the split point.



**Vincent** June 10, 2018 at 8:58 am #

REPLY ↩

Another question:

How exactly do you have to modify your python-code to solve an n-dimensional problem with a decision tree?

Wouldn't this be a good update for this page here? 😊

I would be more than happy and thankful.

Anyway, thank you so much for your work here.



**Jason Brownlee** June 11, 2018 at 6:04 am #

REPLY ↩

The code is to teach you how the algorithm works.

To solve real problems, I recommend using the sklearn library. I have many examples, search the blog.



**Shivan** June 12, 2018 at 7:49 pm #

REPLY ↩

Really nice writeup ! Can I know what these “groups” are exactly ? I know they’re 3d arrays having 2 groups of data but what do they hold ? Thanks !



**Jason Brownlee** June 13, 2018 at 6:18 am #

REPLY ↩

Thanks. Which groups?



**Shivan** June 14, 2018 at 2:56 am #

REPLY ↩

I was referring to the “groups” of data but I understood what they were after I read the code. Thank you for your time !



**Jason Brownlee** June 14, 2018 at 6:11 am #

REPLY ↩

No problem.



**Soumyodipta Karmakar** June 14, 2018 at 2:52 pm #

REPLY ↩

Hello,

I am working on decision tree classifier. Could you please share the coding of SLIQ – decision tree classifier in python language?



**Jason Brownlee** June 14, 2018 at 4:09 pm #

REPLY ↩

What is SLIQ?



**Soumyodipta Karmakar** June 29, 2018 at 4:13 pm #

REPLY ↩

Hello,

SLIQ is a Decision Tree classifier for Data Mining. I am sending you the file of SLIQ. Please check it and if possible please share its coding in Python.

<http://sci2s.ugr.es/keel/pdf/algorithm/congreso/SLIQ.pdf>

Thank you, sir.



**Jason Brownlee** June 30, 2018 at 6:04 am #

REPLY ↩

Thanks. Sorry, I don't know anything about it.



**Vincent** June 22, 2018 at 9:55 pm #

REPLY ↩

I highly recommend publishing this code here on github since it makes contributing very easy. I would then suggest a simple pruning-function. And you already suggest other improvements/features in your text. Plus: you as the owner get more insights and numbers concerning clones/usages etc. What do you think?



**Jason Brownlee** June 23, 2018 at 6:17 am #

REPLY ↩

Thanks for the suggestion, but I prefer to not have my code on github.



**Vincent** June 26, 2018 at 12:37 am #

REPLY ↩

Why?



**Jason Brownlee** June 26, 2018 at 6:41 am #

REPLY ↩

Because I control this blog and all aspects of the experience, I have zero control over github, it is someone else's platform.

I write code tutorials, I don't just dump source code online. Context is critical.

Also sell books, which allows me to keep writing more tutorials.



**TonyStorm** August 3, 2018 at 10:56 pm #

REPLY ↩

thanks alot!!

it really helps!!!

your book and your books are wonderful as well

I am a beginner of ML and I sort of have a language problem in reading those material , but not that difficult.

Q1:I wonder how to corectly use your book and your site tutorial and have a good combination of them?

Q2:there are 63 pages in your tutorial ,are they sorted by the random? or each of your blog is ordered ,eg. the blog right after this article is ,so next one is more advanced in the level of decision trees?last,what I wanna ask is that should I read the matiarial one by one if I wanna learning systematicly about ML or I just read your book ,after read it I look back on this site ?



**Jason Brownlee** August 4, 2018 at 6:10 am #

REPLY ↩

The blog is not very ordered. You can use the search feature.

For more help on a specific topic, I recommend getting a book or using a guide here:

<https://machinelearningmastery.com/start-here/>



**Jun Li** August 20, 2018 at 4:28 pm #

REPLY ↩

Hi, train\_set.remove(fold) line would raise an error.

ValueError: The truth value of an array with more than one element is ambiguous. Use a.any() or a.all()

I solve this problem by refer to this note.

<https://stackoverflow.com/questions/3157374/how-do-you-remove-a-numpy-array-from-a-list-of-numpy-arrays>

```
removearray(train_set,fold)
```

Hope this can help others.



**Jason Brownlee** August 21, 2018 at 6:10 am #

REPLY ↩

The code was written for Python 2.7. Are you using this version of Python?



**Gaurav Nakum** August 27, 2018 at 3:01 am #

REPLY ↩

Hi Jason!

First of all, wonderful article and thank you very much for writing it. I wanted to ask you regarding training a perceptron decision tree, where as you might know, each node is a perceptron. Now to build such a tree, ideally I should be training each node separately and depending on the output (say if its a binary classification problem with  $y = 0$  or  $1$ ), I would split the dataset into two child nodes, provided the impurity is greater than threshold.

Since the split at each node depends on the weights used for the perceptron, the entire tree structure also depends on the weights. Thus, during training I would be constantly changing the tree structure (which might be computationally intensive). Or should I train each perceptron separately starting from the root and do the same for the subsequent nodes?

Thanks!



**Jason Brownlee** August 27, 2018 at 6:14 am #

REPLY ↩

Sorry, I don't have an example of such an algorithm.



**mahi** September 20, 2018 at 4:05 am #

REPLY ↩

Sir I have to implement decision tree and linear svc from scratch on a data set of comments (string) fetched from twitter the csd will contain comments under the hashtags student refugee and trip ....



**Jason Brownlee** September 20, 2018 at 8:07 am #

REPLY ↩

Perhaps use a library like scikit-learn, it will be much easier!





**reynold** October 3, 2018 at 10:58 pm #

REPLY ↩

Sir I have implement this decision tree, but i want to visualize the tree, how can i do that?  
by using sklearn i can use pydotplus.graph\_from\_dot\_data, but i don't know how to do it by using this tutorial



**Jason Brownlee** October 4, 2018 at 6:18 am #

REPLY ↩

Sorry, I don't have an example of plotting a decision tree.



**Sajith Madusanka** October 4, 2018 at 4:13 pm #

REPLY ↩

How can i connect mysql database to get dataset



**Jason Brownlee** October 5, 2018 at 5:28 am #

REPLY ↩

I don't see why not.



**Tarun** October 5, 2018 at 7:30 pm #

REPLY ↩

Sir ,  
I am doing project that will take images as data and will process it such that has to take a decision "Yes" or "No".....suppose i give a data image as blue sky image.....it has to print yes only if its blue....else no.....this is to be done by Machine Learning .....Can you please help me telling a code for that....or any source .....please help me out



**Jason Brownlee** October 6, 2018 at 5:42 am #

REPLY ↩

Sounds like a big project.

Perhaps look into the use of CNN models and perhaps making use of pre-trained models. Also try this tutorial:

<https://machinelearningmastery.com/develop-a-deep-learning-caption-generation-model-in-python/>



**Lincy** October 8, 2018 at 9:46 am #

REPLY ↩

Hi Jason, This is a great article which you have posted. Really helped me a lot.

By any chance do you have links of the extension of this tutorial – Cross-Entropy and Tree Pruning?

Thank you so much!



**Jason Brownlee** October 9, 2018 at 8:30 am #

REPLY ↩

I don't, sorry.



**KK** October 15, 2018 at 1:39 pm #

REPLY ↩

Thank for your article. It is really helpful to my assignment. In your case, “class” has been generated; either 0 or 1. How would you generate class if you haven't done so? Any references on generating “class”. Thanks for your detailed explanation.



**Jason Brownlee** October 15, 2018 at 3:00 pm #

REPLY ↩

This is our goal. To train a model on data where we have the class, then use the trained model to make a prediction on data where we don't have the class.

I recommend starting here:

<https://machinelearningmastery.com/machine-learning-in-python-step-by-step/>



**Arslan** October 27, 2018 at 8:08 am #

REPLY ↩

I'm in a situation where i need to have a custom splitting function that splits on the basis of difference in rate (churn rates for different demographic groups) into two nodes, one with low difference and other with higher difference.

I worked out that function, but my issue now lies in splitting.If i have 3M rows of data, then i will have to calculate my metric 3M-1 times: for every combination of rows. is there a more efficient way to choose splits to evaluate?

Any thoughts would be appreciable. Thanks,



**Jason Brownlee** October 28, 2018 at 6:04 am #

REPLY ↩

This question sounds specific to your dataset, I'm not sure I can answer it sensibly sorry.



**Arslan** October 28, 2018 at 1:14 pm #

REPLY ↩

In the  $n \times m$  dataset :The `get_split(dataset)` function above is calculating gini index for  $n-1$  possible combinations of row-values for that column, before selecting the best split. Which means  $m \times (n-1)$  evaluations of `gini_index()`.

I was able bring down these gini evaluations to #of-unique-row-values in that column, but still is is pretty low.Is there a way to make it more efficient. I don't know how sklearn or rpart do i so quickly.



**Jason Brownlee** October 29, 2018 at 5:53 am #

REPLY ↩

I expect there is, I would recommend using an implementation from a robust library such as sklearn. This code is for learning how the algorithm works, not for operational use.



**Steve** October 30, 2018 at 7:28 pm #

REPLY ↩

Hi Jason, Great tutorial!! I ran your code on the small dataset and then replicated this process manually in Excel to gain a deeper understanding. Finally, I tried using the same data in sklearn but could not get the same results. What is sklearn in DecisionTreeClassifier doing differently/additionally from your code? Thanks!!!



**Jason Brownlee** October 31, 2018 at 6:25 am #

REPLY ↩

The sklearn library will be doing many more things to ensure that the results are robust and that the implementations use best practices. These differences add up to give slightly different resulting trees.



**kiv** November 19, 2018 at 7:30 am #

REPLY ↩

Hi. Can anyone help me what is “algorithm” in “predicted = algorithm(train\_set, test\_set, \*args)” line? Thanks a lot.



**Jason Brownlee** November 19, 2018 at 2:17 pm #

REPLY ↩

It is executing the algorithm function name passed in as an argument.

If this is challenging for you, perhaps start with sklearn:  
<https://machinelearningmastery.com/start-here/#python>



**Anant** November 22, 2018 at 1:25 am <#>

REPLY

Hi Jason,

I built a CART model for my case study in my project using the similar code. But when run it, it throws error `p=[row[-1] for row in group].count(class_val)/size`;invalid index to scalar variable.

Due to inter conversion of the dataset many times throughout the code,group finally becomes a list instead of a nd array, so row in group becomes a scalar variable for which we are trying to access `[-1]` index,resulting in that error.But how to fix it i am not able to figure out. Can you help me here ?



**Jason Brownlee** November 22, 2018 at 6:25 am <#>

REPLY

Ensure your data was loaded as you expect.



**Dphan** November 22, 2018 at 3:56 pm <#>

REPLY

Hi Jason,

I'm building a tree based on your code but I get stuck when I try to visualize the tree. Can you show me how to do that?? Thanks



**Jason Brownlee** November 23, 2018 at 7:42 am <#>

REPLY

I show how to visualize with a simple ascii output.

**AnnaFayn** January 8, 2019 at 1:27 am <#>

REPLY



Hi Jason! Thank you for the great tutorial.

I have a question regarding decision trees. Can I use label encoder for nominal categorical variables? I found a variety of answers for this question on the web by nothing unequivocal.



**Jason Brownlee** January 8, 2019 at 6:52 am #

REPLY ↩

You can, but no need with decision trees.

I also recommend using sklearn's implementation.



**Inbar** January 16, 2019 at 11:24 pm #

REPLY ↩

Excellent explanation! Very helpful

Thank you so much 😊



**Jason Brownlee** January 17, 2019 at 5:26 am #

REPLY ↩

I'm glad it helped.



**Lisa** January 23, 2019 at 8:14 pm #

REPLY ↩

Hello,

thanks a lot for the nice explanation.

I am wondering if I am able to add somewhere in the code an overall constraint regarding the test data.

For example, depending on to which group an instance is assigned, it has a specific value for a parameter. For all the test data the sum of these parameters have to equal something.

Is this somehow possible?



**Jason Brownlee** January 24, 2019 at 6:44 am #

REPLY ↩

Sorry, I don't follow, perhaps you can elaborate what you mean?



**Jugal** January 31, 2019 at 4:08 pm #

REPLY ↩

Hey Dr. Brownlee,

Thankful for your tutorial!

Some comments: I needed to change `rb` to `rt`, and I removed the `string to float` function for my dataset of >400000 obs (since they were already float). I know you didn't design this for such a large dataset, but it's just for fun for now! If my calculations are correct, this will take my machine ~ 200 minutes.

Just fyi, it's still working with Python 3.7 😊

Best,



**Jason Brownlee** February 1, 2019 at 5:33 am #

REPLY ↩

Thanks.



**Suresh** January 31, 2019 at 10:30 pm #

REPLY ↩

Hi Jason thanks for your material, in decision tree after every split we should remove the splitter column but your code is not doing that could you provide some sample code for the same



**Jason Brownlee** February 1, 2019 at 5:37 am <#>

REPLY

No, we do not remove the column used in the split in this algorithm.



**Raghav** February 13, 2019 at 9:43 am <#>

REPLY

Thank you . It was super helpful in understanding the algorithm



**Jason Brownlee** February 13, 2019 at 1:55 pm <#>

REPLY

Thanks, I'm glad it helped.



**Laksh** February 25, 2019 at 12:23 pm <#>

REPLY

Very nice Jason! Really clear explanation and well-written code. I really appreciate you putting out this content for free!



**Jason Brownlee** February 25, 2019 at 2:18 pm <#>

REPLY

Thanks. I hope that it helps.





**Daniel** March 4, 2019 at 8:09 pm #

REPLY ↩

Hi Jason,

Firstly I must say this website is an absolute life-saver when it comes to learning ML without any prior experience – amazing job creating and maintaining these articles!

I am a 3rd year student and I was given my Final year individual project to do which is implementing ML onto an affordable embedded system to recognise human activities – in my case walking and running to start with – the project is mainly proof-of-concept. Until I found this post I was very stuck but when I did I ran it on my collected data (from an on-board accelerometer) which I labeled and got a staggering 93-96% accuracy on my PC.

The microcontroller I am using runs MicroPython – I am using the LoPy by Pycom if you are curious – hence it is fairly straightforward to adapt it to run on it with a very small modified dataset. However I am new to ML and still learning python and I was wondering given the example above what would you suggest I should try to attempt to have a labelled dataset for training and an unlabelled dataset for testing, could that be done easily?

I tried examining your cross-validation function and it seems to need labels to work properly (I looked at your post about it – <https://machinelearningmastery.com/implement-resampling-methods-scratch-python/>). Correct me if I'm wrong of course. Do you have any suggestions? Thank you again for what you are doing here!



**Jason Brownlee** March 5, 2019 at 6:35 am #

REPLY ↩

Thanks.

Well done!

An unlabelled dataset means you are not testing, it means you are making a prediction on new data. No resampling is possible.

Leave a Reply

Name (required)

Email (will not be published) (required)

Website

SUBMIT COMMENT

Welcome to Machine Learning Mastery!



Hi, I'm **Jason Brownlee**, PhD.  
I write tutorials to help developers (*like you*) get results with machine learning.

[Read More](#)

Code Algorithms From Scratch

No libraries, just simple Python code.

[Click to Get Started Now!](#)



POPULAR



**How to Develop LSTM Models for Multi-Step Time Series Forecasting of Household Power Consumption**

OCTOBER 10, 2018



**So, You are Working on a Machine Learning Problem...**

APRIL 4, 2018



**11 Classical Time Series Forecasting Methods in Python (Cheat Sheet)**

AUGUST 6, 2018

**How to Make Predictions with Keras**

APRIL 9, 2018



### How to Develop LSTM Models for Time Series Forecasting

NOVEMBER 14, 2018



### How to Make Predictions with scikit-learn

APRIL 6, 2018



### You're Doing it Wrong. Why Machine Learning Does Not Have to Be So Hard

MARCH 28, 2018

#### Top beginner tutorials:

---

- [How to Install Python for Machine Learning](#)
- [Your First Machine Learning Project in Python](#)
- [Your First Neural Network in Python](#)
- [Your First Classifier in Weka](#)
- [Your First Time Series Forecasting Project](#)

#### Top project tutorials:

---

- [Photo Captioning Project](#)
- [Neural Machine Translation Project](#)
- [Sentiment Analysis Project](#)
- [Power Forecasting Project](#)

© 2019 Machine Learning Mastery. All Rights Reserved.

[Privacy](#) | [Disclaimer](#) | [Terms](#) | [Contact](#)