Need 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 **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.

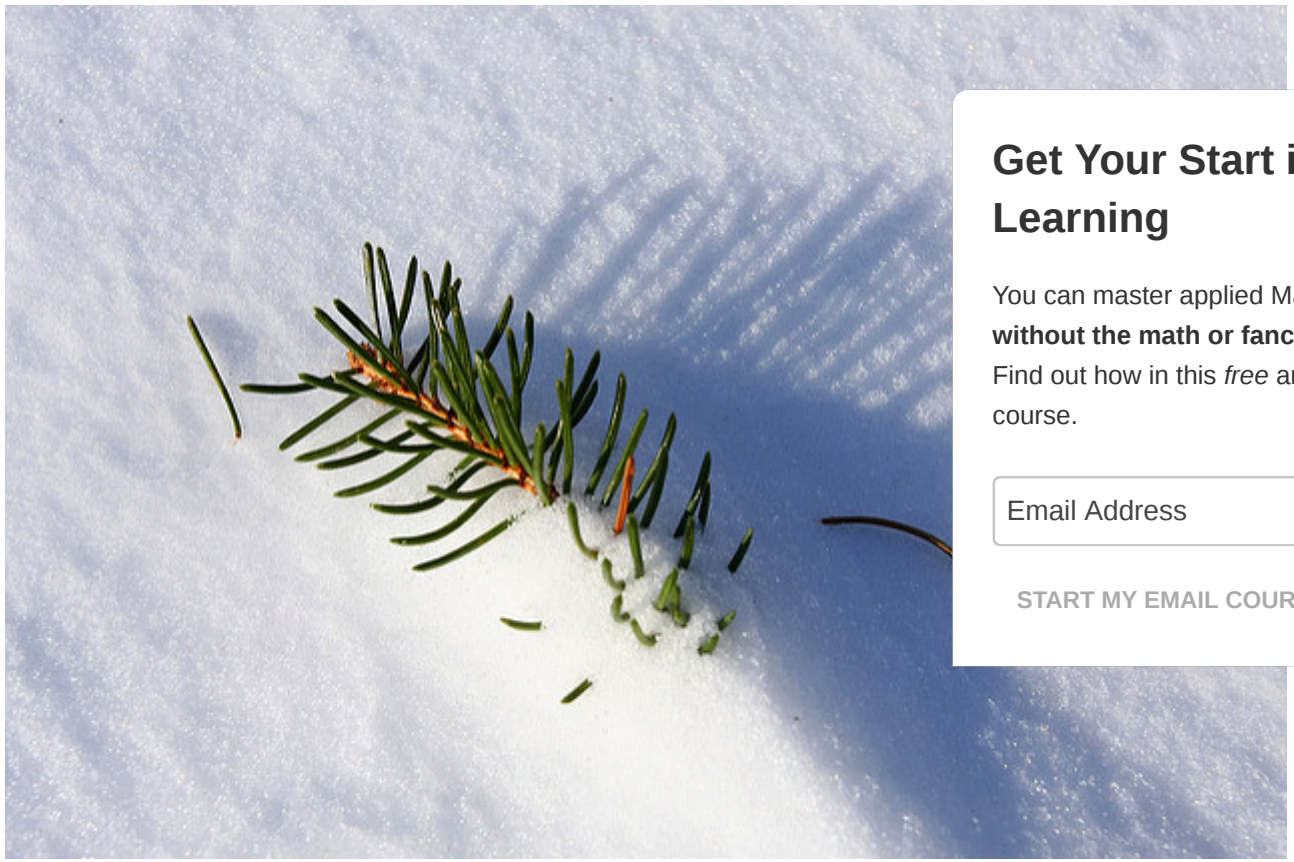
In this tutorial, you will discover how to implement the Classification And Regression Tree algorithm from scratch with Python.

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!).



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
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 u

Creating a binary decision tree is actually a process of dividing up the input space. A greedy approac ary
splitting. This is a numerical procedure where all the values are lined up and different split points are

The split with the best cost (lowest cost because we minimize cost) is selected. All input variables ar en
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
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 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

```
3.6216,8.6661,-2.8073,-0.44699,0
4.5459,8.1674,-2.4586,-1.4621,0
3.866,-2.6383,1.9242,0.10645,0
3.4566,9.5228,-4.0112,-3.5944,0
0.32924,-4.4552,4.5718,-0.9888,0
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

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_banknot**

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.

Get Your Start in Machine Learning

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 all belong to class 1, so it's a perfect split.

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

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

The proportions for this example would be:

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

Gini is then calculated for each child node as follows:

```
gini_index = sum(proportion * (1.0 - proportion))
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:

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

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

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

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

```
# Calculate the Gini index for a split dataset
def gini_index(groups, classes):
    # count all samples at split point
    n_instances = float(sum([len(group) for group in groups]))
    # sum weighted Gini index for each group
    gini = 0.0
    for group in groups:
        size = float(len(group))
        # avoid divide by zero
        if size == 0:
            continue
        score = 0.0
        # score the group based on the score for each class
        for class_val in classes:
            p = [row[-1] for row in group].count(class_val) / size
            score += p * p
        # weight the group score by its relative size
        gini += (1.0 - score) * (size / n_instances)
    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.

```
# Calculate the Gini index for a split dataset
```

```
def gini_index(groups, classes):
    # count all samples at split point
    n_instances = float(sum([len(group) for group in groups]))
    # sum weighted Gini index for each group
    gini = 0.0
    for group in groups:
        size = float(len(group))
        # avoid divide by zero
        if size == 0:
            continue
        score = 0.0
        # score the group based on the score for each class
        for class_val in classes:
            p = [row[-1] for row in group].count(class_val) / size
            score += p * p
        # weight the group score by its relative size
        gini += (1.0 - score) * (size / n_instances)
    return gini

# test Gini values
print(gini_index([[[1, 1], [1, 0]], [[1, 1], [1, 0]]], [0, 1]))
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 t

```
0.5
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.

```
# Split a dataset based on an attribute and an attribute value
def test_split(index, value, dataset):
    left, right = list(), list()
    for row in dataset:
        if row[index] < value:
            left.append(row)
        else:
            right.append(row)
    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

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.

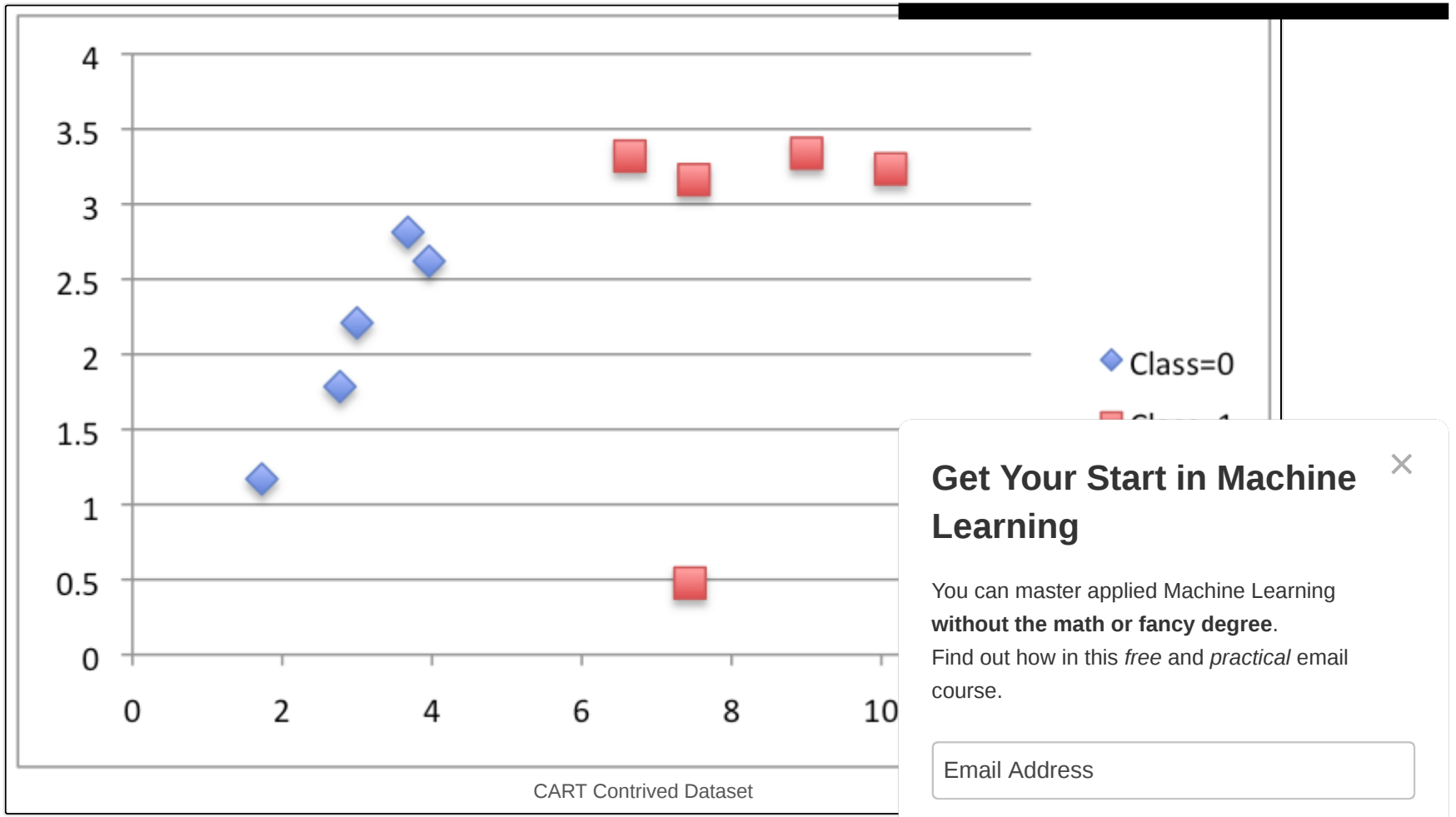
The best split is recorded and then returned after all checks are complete.

```
# Select the best split point for a dataset
def get_split(dataset):
    class_values = list(set(row[-1] for row in dataset))
    b_index, b_value, b_score, b_groups = 999, 999, 999, None
    for index in range(len(dataset[0])-1):
        for row in dataset:
            groups = test_split(index, row[index], dataset)
            gini = gini_index(groups, class_values)
            if gini < b_score:
                b_index, b_value, b_score, b_groups = index, row[index], gini, groups
    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
2.771244718     1.784783929     0
1.728571309     1.169761413     0
3.678319846     2.81281357      0
3.961043357     2.61995032      0
2.999208922     2.209014212     0
7.497545867     3.162953546     1
9.00220326      3.339047188     1
7.444542326     0.476683375     1
10.12493903     3.234550982     1
6.642287351     3.319983761     1
```

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 X1 (x-axis on the plot) to split this dataset.



The example below puts all of this together.

```
# Split a dataset based on an attribute and an attribute value
def test_split(index, value, dataset):
    left, right = list(), list()
    for row in dataset:
        if row[index] < value:
            left.append(row)
        else:
            right.append(row)
    return left, right

# Calculate the Gini index for a split dataset
```

```
def gini_index(groups, classes):
    # count all samples at split point
    n_instances = float(sum([len(group) for group in groups]))
    # sum weighted Gini index for each group
    gini = 0.0
    for group in groups:
        size = float(len(group))
        # avoid divide by zero
        if size == 0:
            continue
        score = 0.0
        # score the group based on the score for each class
        for class_val in classes:
            p = [row[-1] for row in group].count(class_val) / size
            score += p * p
        # weight the group score by its relative size
        gini += (1.0 - score) * (size / n_instances)
    return gini

# Select the best split point for a dataset
def get_split(dataset):
    class_values = list(set(row[-1] for row in dataset))
    b_index, b_value, b_score, b_groups = 999, 999, 999, None
    for index in range(len(dataset[0])-1):
        for row in dataset:
            groups = test_split(index, row[index], dataset)
            gini = gini_index(groups, class_values)
            print('X%d < %.3f Gini=%.3f' % ((index+1), row[index], gini))
            if gini < b_score:
                b_index, b_value, b_score, b_groups = index, row[index], gini, groups
    return {'index':b_index, 'value':b_value, 'groups':b_groups}

dataset = [[2.771244718,1.784783929,0],
    [1.728571309,1.169761413,0],
    [3.678319846,2.81281357,0],
    [3.961043357,2.61995032,0],
    [2.999208922,2.209014212,0],
    [7.497545867,3.162953546,1],
    [9.00220326,3.339047188,1],
    [7.444542326,0.476683375,1],
    [10.12493903,3.234550982,1],
    [6.642287351,3.319983761,1]]
split = get_split(dataset)
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 evalua

Get Your Start in Machine Learning

Running the example prints all of the Gini scores and then prints the score of best split in the dataset of X1 < 6.642 with a Gini index of 0.0 of a perfect split.

```
X1 < 2.771 Gini=0.444
X1 < 1.729 Gini=0.500
X1 < 3.678 Gini=0.286
X1 < 3.961 Gini=0.167
X1 < 2.999 Gini=0.375
X1 < 7.498 Gini=0.286
X1 < 9.002 Gini=0.375
X1 < 7.445 Gini=0.167
X1 < 10.125 Gini=0.444
X1 < 6.642 Gini=0.000
X2 < 1.785 Gini=0.500
X2 < 1.170 Gini=0.444
X2 < 2.813 Gini=0.320
X2 < 2.620 Gini=0.417
X2 < 2.209 Gini=0.476
X2 < 3.163 Gini=0.167
X2 < 3.339 Gini=0.444
X2 < 0.477 Gini=0.500
X2 < 3.235 Gini=0.286
X2 < 3.320 Gini=0.375
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 u

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.

Get Your Start in Machine Learning

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 nd 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 poi to make a final prediction.

This is done by taking the group of rows assigned to that node and selecting the most common class predictions.

Below is a function named **to_terminal()** that will select a class value for a group of rows. It returns t

```
# Create a terminal node value
def to_terminal(group):
    outcomes = [row[-1] for row in group]
    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.

Get Your Start in Machine Learning

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 u
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, otherwi st 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

```
# Create child splits for a node or make terminal
def split(node, max_depth, min_size, depth):
    left, right = node['groups']
    del(node['groups'])
    # check for a no split
    if not left or not right:
        node['left'] = node['right'] = to_terminal(left + right)
        return
    # check for max depth
    if depth >= max_depth:
        node['left'], node['right'] = to_terminal(left), to_terminal(right)
        return
    # process left child
    if len(left) <= min_size:
        node['left'] = to_terminal(left)
    else:
        node['left'] = get_split(left)
        split(node['left'], max_depth, min_size, depth+1)
    # process right child
    if len(right) <= min_size:
        node['right'] = to_terminal(right)
    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.

```
# 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
```

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 wi
real decision tree diagram, it gives an idea of the tree structure and decisions made throughout.

```
# Split a dataset based on an attribute and an attribute value
def test_split(index, value, dataset):
    left, right = list(), list()
    for row in dataset:
        if row[index] < value:
            left.append(row)
        else:
            right.append(row)
    return left, right

# Calculate the Gini index for a split dataset
def gini_index(groups, classes):
    # count all samples at split point
    n_instances = float(sum([len(group) for group in groups]))
    # sum weighted Gini index for each group
    gini = 0.0
    for group in groups:
        size = float(len(group))
```

```
        # avoid divide by zero
        if size == 0:
            continue
        score = 0.0
        # score the group based on the score for each class
        for class_val in classes:
            p = [row[-1] for row in group].count(class_val) / size
            score += p * p
        # weight the group score by its relative size
        gini += (1.0 - score) * (size / n_instances)
    return gini

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

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

# Create child splits for a node or make terminal
def split(node, max_depth, min_size, depth):
    left, right = node['groups']
    del(node['groups'])
    # check for a no split
    if not left or not right:
        node['left'] = node['right'] = to_terminal(left + right)
        return
    # check for max depth
    if depth >= max_depth:
        node['left'], node['right'] = to_terminal(left), to_terminal(right)
        return
    # process left child
    if len(left) <= min_size:
        node['left'] = to_terminal(left)
    else:
        node['left'] = get_split(left)
        split(node['left'], max_depth, min_size, depth+1)
```

```
    # process right child
    if len(right) <= min_size:
        node['right'] = to_terminal(right)
    else:
        node['right'] = get_split(right)
        split(node['right'], max_depth, min_size, depth+1)

# 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

# Print a decision tree
def print_tree(node, depth=0):
    if isinstance(node, dict):
        print('%s[X%d < %.3f]' % ((depth*' ', (node['index']+1), node['value'])))
        print_tree(node['left'], depth+1)
        print_tree(node['right'], depth+1)
    else:
        print('%s[%s]' % ((depth*' ', node)))

dataset = [[2.771244718,1.784783929,0],
    [1.728571309,1.169761413,0],
    [3.678319846,2.81281357,0],
    [3.961043357,2.61995032,0],
    [2.999208922,2.209014212,0],
    [7.497545867,3.162953546,1],
    [9.00220326,3.339047188,1],
    [7.444542326,0.476683375,1],
    [10.12493903,3.234550982,1],
    [6.642287351,3.319983761,1]]
tree = build_tree(dataset, 1, 1)
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.

```
[X1 < 6.642]
 [0]
 [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.

```
[X1 < 6.642]
 [X1 < 2.771]
  [0]
  [0]
 [X1 < 7.498]
  [1]
  [1]
```

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

```
[X1 < 6.642]
 [X1 < 2.771]
  [0]
  [X1 < 2.771]
   [0]
   [0]
 [X1 < 7.498]
  [X1 < 7.445]
   [1]
   [1]
  [X1 < 7.498]
   [1]
   [1]
```

These tests show that there is great opportunity to refine the implementation to avoid unnecessary s

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.

Get Your Start in Machine Learning

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.

```
# Make a prediction with a decision tree
def predict(node, row):
    if row[node['index']] < node['value']:
        if isinstance(node['left'], dict):
            return predict(node['left'], row)
        else:
            return node['left']
    else:
        if isinstance(node['right'], dict):
            return predict(node['right'], row)
        else:
            return node['right']
```

We can use our contrived dataset to test this function. Below is an example that uses a hard-coded
data (a decision stump).

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

```
# Make a prediction with a decision tree
def predict(node, row):
    if row[node['index']] < node['value']:
        if isinstance(node['left'], dict):
            return predict(node['left'], row)
        else:
            return node['left']
    else:
        if isinstance(node['right'], dict):
            return predict(node['right'], row)
        else:
            return node['right']

dataset = [[2.771244718,1.784783929,0],
    [1.728571309,1.169761413,0],
    [3.678319846,2.81281357,0],
    [3.961043357,2.61995032,0],
    [2.999208922,2.209014212,0],
    [7.497545867,3.162953546,1],
    [9.00220326,3.339047188,1],
    [7.444542326,0.476683375,1],
    [10.12493903,3.234550982,1],
```

```
    [6.642287351,3.319983761,1]]

#  predict with a stump
stump = {'index': 0, 'right': 1, 'value': 6.642287351, 'left': 0}
for row in dataset:
    prediction = predict(stump, row)
    print('Expected=%d, Got=%d' % (row[-1], prediction))
```

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

```
Expected=0, Got=0
Expected=0, Got=0
Expected=0, Got=0
Expected=0, Got=0
Expected=0, Got=0
Expected=1, Got=1
Expected=1, Got=1
Expected=1, Got=1
Expected=1, Got=1
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 re

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 calcu 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 ld. 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.

```
# CART on the Bank Note dataset
```

```
from random import seed
from random import randrange
from csv import reader

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

# Convert string column to float
def str_column_to_float(dataset, column):
    for row in dataset:
        row[column] = float(row[column].strip())

# Split a dataset into k folds
def cross_validation_split(dataset, n_folds):
    dataset_split = list()
    dataset_copy = list(dataset)
    fold_size = int(len(dataset) / n_folds)
    for i in range(n_folds):
        fold = list()
        while len(fold) < fold_size:
            index = randrange(len(dataset_copy))
            fold.append(dataset_copy.pop(index))
        dataset_split.append(fold)
    return dataset_split

# Calculate accuracy percentage
def accuracy_metric(actual, predicted):
    correct = 0
    for i in range(len(actual)):
        if actual[i] == predicted[i]:
            correct += 1
    return correct / float(len(actual)) * 100.0

# Evaluate an algorithm using a cross validation split
def evaluate_algorithm(dataset, algorithm, n_folds, *args):
    folds = cross_validation_split(dataset, n_folds)
    scores = list()
    for fold in folds:
        train_set = list(folds)
        train_set.remove(fold)
        train_set = sum(train_set, [])
        test_set = list()
        for row in fold:
```

```
            row_copy = list(row)
            test_set.append(row_copy)
            row_copy[-1] = None
        predicted = algorithm(train_set, test_set, *args)
        actual = [row[-1] for row in fold]
        accuracy = accuracy_metric(actual, predicted)
        scores.append(accuracy)
    return scores

# Split a dataset based on an attribute and an attribute value
def test_split(index, value, dataset):
    left, right = list(), list()
    for row in dataset:
        if row[index] < value:
            left.append(row)
        else:
            right.append(row)
    return left, right

# Calculate the Gini index for a split dataset
def gini_index(groups, classes):
    # count all samples at split point
    n_instances = float(sum([len(group) for group in groups]))
    # sum weighted Gini index for each group
    gini = 0.0
    for group in groups:
        size = float(len(group))
        # avoid divide by zero
        if size == 0:
            continue
        score = 0.0
        # score the group based on the score for each class
        for class_val in classes:
            p = [row[-1] for row in group].count(class_val) / size
            score += p * p
        # weight the group score by its relative size
        gini += (1.0 - score) * (size / n_instances)
    return gini

# Select the best split point for a dataset
def get_split(dataset):
    class_values = list(set(row[-1] for row in dataset))
    b_index, b_value, b_score, b_groups = 999, 999, 999, None
    for index in range(len(dataset[0])-1):
        for row in dataset:
            groups = test_split(index, row[index], dataset)
            gini = gini_index(groups, class_values)
```

```
            if gini < b_score:
                b_index, b_value, b_score, b_groups = index, row[index], gini, groups
    return {'index':b_index, 'value':b_value, 'groups':b_groups}

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

# Create child splits for a node or make terminal
def split(node, max_depth, min_size, depth):
    left, right = node['groups']
    del(node['groups'])
    # check for a no split
    if not left or not right:
        node['left'] = node['right'] = to_terminal(left + right)
        return
    # check for max depth
    if depth >= max_depth:
        node['left'], node['right'] = to_terminal(left), to_terminal(right)
        return
    # process left child
    if len(left) <= min_size:
        node['left'] = to_terminal(left)
    else:
        node['left'] = get_split(left)
        split(node['left'], max_depth, min_size, depth+1)
    # process right child
    if len(right) <= min_size:
        node['right'] = to_terminal(right)
    else:
        node['right'] = get_split(right)
        split(node['right'], max_depth, min_size, depth+1)

# 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

# Make a prediction with a decision tree
def predict(node, row):
    if row[node['index']] < node['value']:
        if isinstance(node['left'], dict):
            return predict(node['left'], row)
        else:
            return node['left']
```

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

# Classification and Regression Tree Algorithm
def decision_tree(train, test, max_depth, min_size):
    tree = build_tree(train, max_depth, min_size)
    predictions = list()
    for row in test:
        prediction = predict(tree, row)
        predictions.append(prediction)
    return(predictions)

# Test CART on Bank Note dataset
seed(1)
# load and prepare data
filename = 'data_banknote_authentication.csv'
dataset = load_csv(filename)
# convert string attributes to integers
for i in range(len(dataset[0])):
    str_column_to_float(dataset, i)
# evaluate algorithm
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))))
```


The example uses the max tree depth of 5 layers and the minimum number of rows per node to 10. [illegible] tle 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 [illegible].

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.

```
Scores: [96.35036496350365, 97.08029197080292, 97.44525547445255, 98.17518248175182, 97.44525547445255]
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.

About Jason Brownlee

Dr. Jason Brownlee is a husband, proud father, academic researcher, author, professional developer and a machine learning practitioner. He is dedicated to helping developers get started and get good at applied machine learning. Learn more.

View all posts by Jason Brownlee →

‹ How to Implement the Backpropagation Algorithm From Scratch In Python

How to Implement Bagging From Scratch With Python ›

118 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 #

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, [X1 < 6.642] is the root node with two child nodes, each leaf node has a classification label.

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 #

It can with some modification.

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

What modifications would you recommend?

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

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:

Get Your Start in Machine Learning

index = randrange(len(dataset_copy))

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

REPLY

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

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 #

I have updated the cross_validation_split() function in the above example to address iss

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

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

REPLY

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

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 right now to changes in the result? From what I can see, it looks like they are being set in the evaluate_a

//Kind regards
Sokrates

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

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 #

Really helpful. Thanks a lot for sharing.

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

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 follwing procedure:
——
# Build a decision tree
def build_tree(train, max_depth, min_size):

Get Your Start in Machine Learning

root = get_split(dataset)
split(root, max_depth, min_size, 1)
return root
———
I think it should be root = get_split(train), eventhough 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 #

Thanks a lot Jason, really helpful

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

I'm glad to hear that.

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.

REPLY

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

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 #

After playing around with the code for a bit, I realized that function returns both groups (

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

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.

REPLY

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

Yes.

Get Your Start in Machine Learning

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 #

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

Ann April 3, 2017 at 9:25 am #

Hello,

I’ve been trying some stuff out with this code and I thought I was understanding what was going on but w 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.

Get Your Start in Machine Learning

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 #

How can we use weka for regression using decision trees?

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

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 #

I'm really glad to hear that Joe!

REPLY

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 #

Yes, that is how it was coded.

Ansar April 20, 2017 at 2:53 am #

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_4temBw7vLA19p3tdQH6FYO
#
#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 .pnj, 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')
```

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 #

I'm glad you found it useful.

godavari May 4, 2017 at 11:08 pm #

excellent explanation

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

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

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 #

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.

REPLY

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

Thanks.

Saurabh May 25, 2017 at 7:16 pm #

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 c
split_value for test_split function.

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

Thank You!!

REPLY

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

Try it and see.

REPLY

Habiba June 4, 2017 at 6:36 am #

Get Your Start in Machine Learning

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 #

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.

REPLY

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

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 #

Hello sir
Thanks for this tutorial.
I guess that when sum the Gini indexes of subgroups into a total one, their corresponding group sizes sh
should be a weighted sum.

Michael August 5, 2017 at 9:07 am #

I think I have the same question. The Gini Index computed in the above examples are not v
be? I posted a detailed example, but it has not been excepted (yet?).

REPLY

sanju June 27, 2017 at 10:33 pm #

Hello Sir,

Get Your Start in Machine Learning

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.

REPLY

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

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 #

Hello Sir,

Assume I have 1 million training data,and I created and saved a model to predict whether the patient is d
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 pa

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(traini

Thank you

REPLY

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

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

Get Your Start in Machine Learning

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 #

Yes, Weka:
http://machinelearningmastery.com/start-here/#weka

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

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.

Get Your Start in Machine Learning

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 #

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 #

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 Pati

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 #

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

Get Your Start in Machine Learning

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 y

Any ideas on what might be happening? Thanks for taking the time to read my comment and again than

Ikib

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

That is odd.

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

REPLY

buzznizz August 4, 2017 at 9:22 pm #

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) = group 1 (above line), class 0 = 0.6667
P(1, 1) = group 1 (above line), class 1 = 0.3333
P(2, 0) = group 2 (below line), class 0 = 0.4286
P(2, 1) = 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) x (1 – P(1, 0))] +
[P(1, 1) x (1 – P(1, 1))] +
[P(2, 0) x (1 – P(2, 0))] +
[P(2, 1) x (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) x (0.2222 + 0.2222)] + [(7/10) x (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 #

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 clas

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

Does that help?

Michael August 9, 2017 at 7:41 am #

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 mk subscript on p hat. These imply that both the proportion (p hat sub mk) and G are with respect to a particular region m.

Eqn. (8.6) in the ISL is correct, but it’s not the final quanitity 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 propo

Get Your Start in Machine Learning

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:

```
def gini_index2(groups, class_values):
    group_sizes = [len(group) for group in groups]
    sample_count = sum(group_sizes)
    gini_groups = []
    for idx, group in enumerate(groups):
        gini_group = 0.0
        group_size = len(group)
        for class_value in class_values:
            if group_size == 0:
                continue
            proportion = [row[-1] for row in group].count(class_value) / \
                         float(group_size)
            gini_group += (proportion * (1.0 - proportion))
        # weight the group gini score by the size of the group
        partition_weight = group_sizes[idx] / sample_count
        gini_groups.append(partition_weight * gini_group)

    return sum(gini_groups)
```

Thoughts?

Michael August 9, 2017 at 7:44 am #

Sorry about the loss of indentation in the code… Seems like the webapp parses the

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 #

Get Your Start in Machine Learning

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…):

```
# groups - tuple or list of samples corresponding to a split group
# class_values - tuple or list of unique class labels
# TODO class_values can be obtained from groups - no need to pass in
def gini_index2(groups, class_values):
    sample_count = sum([len(group) for group in groups])
    gini_groups = []
    for group in groups:
        gini_group = 0.0
        group_size = len(group)
        for class_value in class_values:
            if group_size == 0:
                continue
            proportion = [row[-1] for row in group].count(class_value)
                         float(group_size)
            gini_group += (proportion * (1.0 - proportion))
        # weight the group gini score by the size of the group
        partition_weight = group_size / sample_count
        gini_groups.append(partition_weight * gini_group)

    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 sklearns 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

aaaaa August 28, 2017 at 12:08 am #

how to apply this for discrete value ?

scssek September 15, 2017 at 1:33 pm #

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 #

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

Jarich September 19, 2017 at 12:12 am #

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 #

Hey Maria, you could save it to a .txt file and then read it back in for the prediction. I did it th

Maria September 22, 2017 at 11:53 pm #

oh okay. I will save the result for build_tree to a text file? Thank you very much for this i

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

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.

Get Your Start in Machine Learning

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 artcile. Can u 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 #

Thanks for the suggestion.

Maria September 30, 2017 at 11:45 pm #

I would like to know if this tree can also handle multiple classification aside from 0 or 1? for exa
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 #

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

lightbandit October 12, 2017 at 4:18 am #

I have a dataset with titles on both axes (will post example below) and they have 1's and 0's sig
would you suggest I start with altering your code in order to fit this dataset? Right now I am getting an er

white blue tall
ch1 0 1 1
ch2 1 0 0
ch3 1 0 1

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

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 #

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

Leave a Reply

Name (required)

Email (will not be published) (required)

Website

SUBMIT COMMENT

Welcome to Machine Learning Mastery

Hi, I'm Dr. Jason Brownlee.
My goal is to make practitioners like YOU awesome at applied machine learning.

Read More

Code Algorithms From Scratch in Python

Discover how to code top machine learning algorithms from first princi

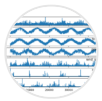
Code Machine Learning Algorithms From Scratch Toda

Get Your Start in Machine Learning

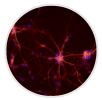
Time Series Forecasting with the Long Short-Term Memory Network in Python
APRIL 7, 2017

Multi-Class Classification Tutorial with the Keras Deep Learning Library
JUNE 2, 2016

Regression Tutorial with the Keras Deep Learning Library in Python
JUNE 9, 2016



Multivariate Time Series Forecasting with LSTMs in Keras
AUGUST 14, 2017



How to Implement the Backpropagation Algorithm From Scratch In Python
NOVEMBER 7, 2016

Privacy | Contact | About

Get Your Start in Machine Learning