How to Implement Bagging From Scratch With Python

by **Jason Brownlee** on November 11, 2016 in **Algorithms From Scratch**

Decision trees are a simple and powerful predictive modeling technique, but they suffer from high-variance.

This means that trees can get very different results given different training data.

A technique to make decision trees more robust and to achieve better performance is called bootstrap aggregation or bagging for short.

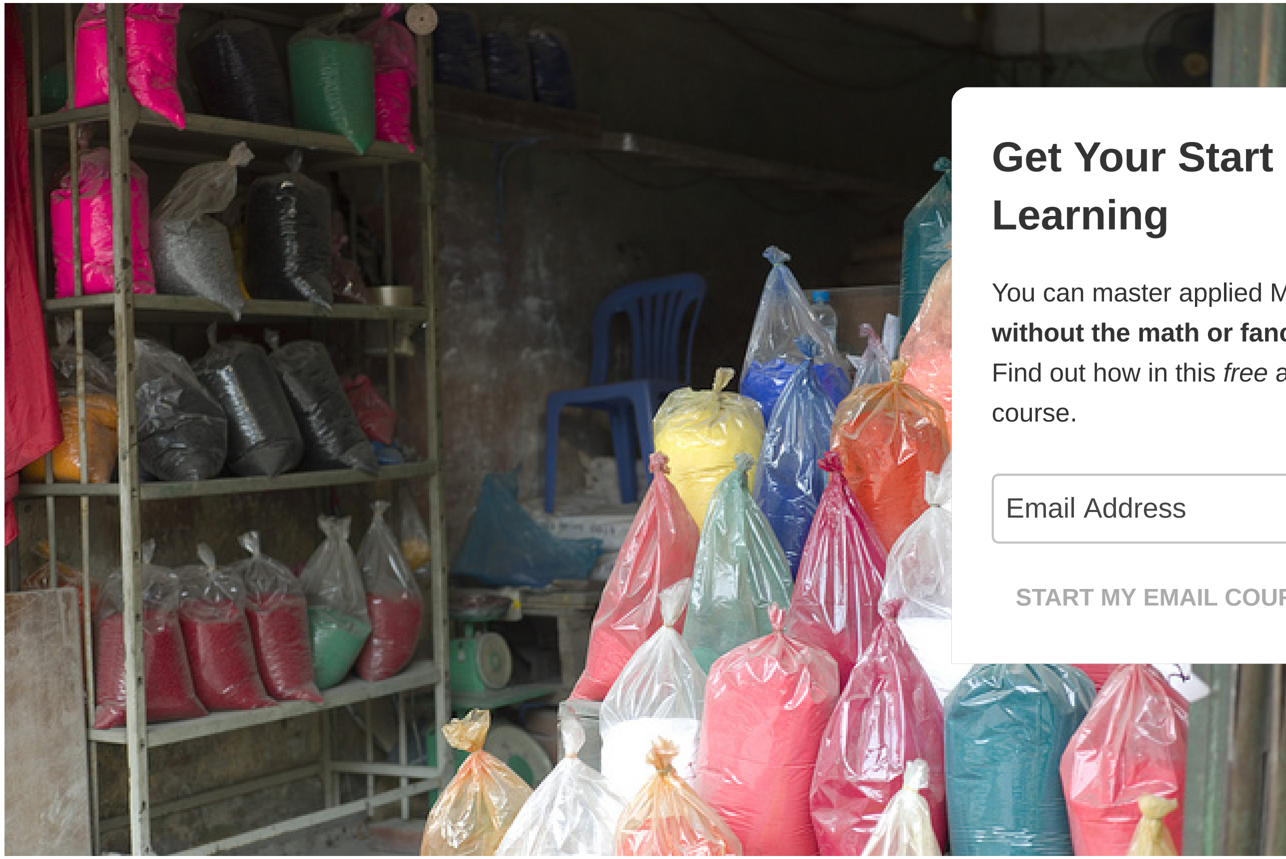
In this tutorial, you will discover how to implement the bagging procedure with decision trees from scratch with Python.

After completing this tutorial, you will know:

- How to create a bootstrap sample of your dataset.
- How to make predictions with bootstrapped models.
- How to apply bagging to your own predictive modeling problems.

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 Bagging From Scratch With Python
Photo by Michael Cory, some rights reserved.

Descriptions

This section provides a brief description to Bootstrap Aggregation and the Sonar dataset that will be used in this tutorial.

Bootstrap Aggregation Algorithm

A bootstrap is a sample of a dataset with replacement.

This means that a new dataset is created from a random sample of an existing dataset where a given row may be selected and added more than once to the sample.

It is a useful approach to use when estimating values such as the mean for a broader dataset, when you only have a limited dataset available. By creating samples of your dataset and estimating the mean from those samples, you can take the ave… …ne true mean of the underlying problem.

This same approach can be used with machine learning algorithms that have a high variance, such a… each bootstrap sample of data and the average output of those models used to make predictions. Th… bagging for short.

Variance means that an algorithm's performance is sensitive to the training data, with high variance s… changed, the more the performance of the algorithm will vary.

The performance of high variance machine learning algorithms like unpruned decision trees can be i… average of their predictions. Results are often better than a single decision tree.

Another benefit of bagging in addition to improved performance is that the bagged decision trees ca… added until a maximum in performance is achieved.

Sonar Dataset

The dataset we will use in this tutorial is the Sonar dataset.

This is a dataset that describes sonar chirp returns bouncing off different surfaces. The 60 input varia… angles. It is a binary classification problem that requires a model to differentiate rocks from metal cyl…

It is a well-understood dataset. All of the variables are continuous and generally in the range of 0 to [redacted] "R" for rock, which will need to be converted to integers 1 and 0.

By predicting the class with the most observations in the dataset (M or mines) the Zero Rule Algorithm can achieve an accuracy of 53%.

You can learn more about this dataset at the UCI Machine Learning repository.

Download the dataset for free and place it in your working directory with the filename **sonar.all-data.csv**.

Tutorial

This tutorial is broken down into 2 parts:

1. Bootstrap Resample.
2. Sonar Dataset Case Study.

These steps provide the foundation that you need to implement and apply bootstrap aggregation wit
problems.

1. Bootstrap Resample

Let's start off by getting a strong idea of how the bootstrap method works.

We can create a new sample of a dataset by randomly selecting rows from the dataset and adding tl
number of rows or until the size of the new dataset matches a ratio of the size of the original dataset

We can allow sampling with replacement by not removing the row that was selected so that it is available for future selections.

Below is a function named **subsample()** that implements this procedure. The **randrange()** function from the random module is used to select a random row index to add to the sample each iteration of the loop. The default size of the sample is the size of the original dataset.

```
# Create a random subsample from the dataset with replacement
def subsample(dataset, ratio=1.0):
    sample = list()
    n_sample = round(len(dataset) * ratio)
    while len(sample) < n_sample:
```

```
        index = randrange(len(dataset))
        sample.append(dataset[index])
    return sample
```

We can use this function to estimate the mean of a contrived dataset.

First, we can create a dataset with 20 rows and a single column of random numbers between 0 and 9 and calculate the mean value.

We can then make bootstrap samples of the original dataset, calculate the mean, and repeat this process until we have a list of means. Taking the average of these sample means should give us a robust estimate of the mean of the entire dataset.

The complete example is listed below.

Each bootstrap sample is created as a 10% sample of the original 20 observation dataset (or 2 obser ... 100 bootstrap samples of the original dataset, calculate their mean value, then average all of those estim

```
from random import seed
from random import random
from random import randrange


# Create a random subsample from the dataset with replacement
def subsample(dataset, ratio=1.0):
    sample = list()
    n_sample = round(len(dataset) * ratio)
    while len(sample) < n_sample:
        index = randrange(len(dataset))
        sample.append(dataset[index])
    return sample


# Calculate the mean of a list of numbers
def mean(numbers):
    return sum(numbers) / float(len(numbers))


seed(1)
# True mean
dataset = [[randrange(10)] for i in range(20)]
print('True Mean: %.3f' % mean([row[0] for row in dataset]))
# Estimated means
ratio = 0.10
for size in [1, 10, 100]:
```

```
    sample_means = list()
    for i in range(size):
        sample = subsample(dataset, ratio)
        sample_mean = mean([row[0] for row in sample])
        sample_means.append(sample_mean)
    print('Samples=%d, Estimated Mean: %.3f' % (size, mean(sample_means)))
```

Running the example prints the original mean value we aim to estimate.

We can then see the estimated mean from the various different numbers of bootstrap samples. We can see that with 100 samples we achieve a good estimate of the mean.

```
True Mean: 4.450
Samples=1, Estimated Mean: 4.500
Samples=10, Estimated Mean: 3.300
Samples=100, Estimated Mean: 4.480
```

Instead of calculating the mean value, we can create a model from each subsample.

Next, let's see how we can combine the predictions from multiple bootstrap models.

2. Sonar Dataset Case Study

In this section, we will apply the Random Forest algorithm to the Sonar dataset.

The example assumes that a CSV copy of the dataset is in the current working directory with the file

The dataset is first loaded, the string values converted to numeric and the output column is converte is achieved with helper functions **load_csv()**, **str_column_to_float()** and **str_column_to_int()** to load

We will use k-fold cross validation to estimate the performance of the learned model on unseen data. This means that we will construct and evaluate k models and estimate the performance as the mean model error. Classification accuracy will be used to evaluate each model. These behaviors are provided in the **cross_validation_split()**, **accuracy_metric()** and **evaluate_algorithm()** helper functions.

We will also use an implementation of the Classification and Regression Trees (CART) algorithm adapted for bagging including the helper functions **test_split()** to split a dataset into groups, **gini_index()** to evaluate a split point, **get_split()** to find an optimal split point, **to_terminal()**, **split()** and **build_tree()** used to create a single decision tree, **predict()** to make a prediction with a decision tree previous step to make a subsample of the training dataset

A new function named **bagging_predict()** is developed that is responsible for making a prediction w[redacted]ns into a single return value. This is achieved by selecting the most common prediction from the list of predictions made by the bagged trees.

Finally, a new function named **bagging()** is developed that is responsible for creating the samples of the training dataset, training a decision tree on each, then making predictions on the test dataset using the list of bagged trees.

The complete example is listed below.

```
# Bagging Algorithm on the Sonar dataset
from random import seed
from random import randrange
from csv import reader

# Load a CSV file
def load_csv(filename):
    dataset = list()
    with open(filename, 'r') as file:
        csv_reader = reader(file)
        for row in csv_reader:
            if not row:
                continue
            dataset.append(row)
    return dataset

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

# Convert string column to integer
def str_column_to_int(dataset, column):
    class_values = [row[column] for row in dataset]
    unique = set(class_values)
    lookup = dict()
    for i, value in enumerate(unique):
        lookup[value] = i
    for row in dataset:
        row[column] = lookup[row[column]]
    return lookup

# 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
```

```python
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:
        # for i in range(len(dataset)):
        #   row = dataset[randrange(len(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']

# Create a random subsample from the dataset with replacement
def subsample(dataset, ratio):
    sample = list()
    n_sample = round(len(dataset) * ratio)
    while len(sample) < n_sample:
        index = randrange(len(dataset))
        sample.append(dataset[index])
    return sample

# Make a prediction with a list of bagged trees
def bagging_predict(trees, row):
    predictions = [predict(tree, row) for tree in trees]
```

```
    return max(set(predictions), key=predictions.count)

# Bootstrap Aggregation Algorithm
def bagging(train, test, max_depth, min_size, sample_size, n_trees):
    trees = list()
    for i in range(n_trees):
        sample = subsample(train, sample_size)
        tree = build_tree(sample, max_depth, min_size)
        trees.append(tree)
    predictions = [bagging_predict(trees, row) for row in test]
    return(predictions)

# Test bagging on the sonar dataset
seed(1)
# load and prepare data
filename = 'sonar.all-data.csv'
dataset = load_csv(filename)
# convert string attributes to integers
for i in range(len(dataset[0])-1):
    str_column_to_float(dataset, i)
# convert class column to integers
str_column_to_int(dataset, len(dataset[0])-1)
# evaluate algorithm
n_folds = 5
max_depth = 6
min_size = 2
sample_size = 0.50
for n_trees in [1, 5, 10, 50]:
    scores = evaluate_algorithm(dataset, bagging, n_folds, max_depth, min_size, sample_
    print('Trees: %d' % n_trees)
    print('Scores: %s' % scores)
    print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))
```


A k value of 5 was used for cross-validation, giving each fold 208/5 = 41.6 or just over 40 records to

Deep trees were constructed with a max depth of 6 and a minimum number of training rows at each node of 2. Samples of the training dataset were created with 50% the size of the original dataset. This was to force some variety in the dataset subsamples used to train each tree. The default for bagging is to have the size of sample datasets match the size of the original training dataset.

A series of 4 different numbers of trees were evaluated to show the behavior of the algorithm.

The accuracy from each fold and the mean accuracy for each configuration are printed. We can see a trend of some minor lift in performance as the number of trees is increased.

```
Trees: 1
Scores: [87.8048780487805, 65.85365853658537, 65.85365853658537, 65.85365853658537, 73.17073170731707]
Mean Accuracy: 71.707%

Trees: 5
Scores: [60.97560975609756, 80.48780487804879, 78.04878048780488, 82.92682926829268, 63.41463414634146]
Mean Accuracy: 73.171%

Trees: 10
Scores: [60.97560975609756, 73.17073170731707, 82.92682926829268, 80.48780487804879, 68.29268292682927]
Mean Accuracy: 73.171%

Trees: 50
Scores: [63.41463414634146, 75.60975609756098, 80.48780487804879, 75.60975609756098, 85.36585365853658]
Mean Accuracy: 76.098%
```

A difficulty of this method is that even though deep trees are constructed, the bagged trees that are created are very similar. In turn, the predictions made by these trees are also similar, and the high variance we desire among the trees trained on different

This is because of the greedy algorithm used in the construction of the trees selecting the same or s

The tutorial tried to re-inject this variance by constraining the sample size used to train each tree. A r that may be evaluated when creating each split point. This is the method used in the Random Fores

Extensions

- **Tune the Example**. Explore different configurations for the number of trees and even individual ve results.
- **Bag Another Algorithm**. Other algorithms can be used with bagging. For example, a k-nearest a high variance and is a good candidate for bagging.
- **Regression Problems**. Bagging can be used with regression trees. Instead of predicting the most common class value from the set of predictions, you can return the average of the predictions from the bagged trees. Experiment on regression problems.

Did you try any of these extensions?
Share your experiences in the comments below.

Review

In this tutorial, you discovered how to implement bootstrap aggregation from scratch with Python.

Specifically, you learned:

- How to create a subsample and estimate bootstrap quantities.
- How to create an ensemble of decision trees and use them to make predictions.
- How to apply bagging to a real world predictive modeling problem.

Do you have any questions?
Ask your questions in the comments below and I will do my best to answer.

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 Decision Tree Algorithm From Scratch In Python

How to Implement Random Forest From Scratch in Python ›

6 Responses to *How to Implement Bagging From Scratch With Python*

skorzec January 18, 2017 at 4:30 am #

Thanks for this example in python from scratch. I think the reason your score stays the same is
your split attributes. This leads to similar trees and thus a small variance of the ensemble set. If you cha
in the trees, you will see an increase in performance of the ensemble. One solution is to alter this code s

# 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

Jason Brownlee January 18, 2017 at 10:17 am #

Thanks for the tip!

Jovan Sardinha June 25, 2017 at 9:41 am #

REPLY

thanks for the great tutorial!

I have re-written this script using sklearn for easy implementation:
https://gist.github.com/JovanSardinha/2c58bd1e7e3aa4c02affedfe7abe8a29

Jason Brownlee June 26, 2017 at 6:04 am #

Nice work!

Alex Godfrey August 31, 2017 at 6:43 am #

Hi Jason,

Minor tip – In the string to integer conversion – I found that the unique set gets created in somev
scripts that use this function. To avoid this I have changed the line to read-

unique = sorted(set(class_values))

This results in creating the same lookup dictionary every time. I ran across this when I was using the lookup dictionary to create nicely mapped printouts of the intermediate data.

Jason Brownlee September 1, 2017 at 6:38 am #

REPLY

Great tip, thanks Alex!

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

POPULAR

Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras
JULY 21, 2016

Your First Machine Learning Project in Python Step-By-Step
JUNE 10, 2016

Develop Your First Neural Network in Python With Keras Step-By-Step
MAY 24, 2016

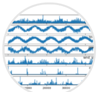

Sequence Classification with LSTM Recurrent Neural Networks in Python with Keras
JULY 26, 2016

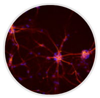
How to Setup a Python Environment for Machine Learning and Deep Learning with Anaconda
MARCH 13, 2017


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