Navigation

Start Here Blog Books About Contact

Search...

Want help with machine learning? Take the FREE Crash-Course.

How to Implement Random Forest From Scratch in Python

by **Jason Brownlee** on November 14, 2016 in **Algorithms From Scratch**

6 310

Decision trees can suffer from high variance which makes their results fragile to the specific training data used.

Building multiple models from samples of your training data, called bagging, can reduce this variance, but the trees are highly correlated.

Random Forest is an extension of bagging that in addition to building trees based on multiple samples of your training data, it also constrains the features that can be used to build the trees, forcing trees to be different. This, in turn, can give a lift in performance.

In this tutorial, you will discover how to implement the Random Forest algorithm from scratch in Python.

After completing this tutorial, you will know:

- The difference between bagged decision trees and
- How to construct bagged decision trees with more
- How to apply the random forest algorithm to a prec

Let's get started.

- **Update Jan/2017**: Changed the calculation of fold ger. Fixes issues with Python 3.
- **Update Feb/2017**: Fixed a bug in build_tree.

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

How to Implement Random Forest From Scratch in Python
Photo by InspireFate Photography, some rights reserved.

Description

This section provides a brief introduction to the Random Forest algorithm and the Sonar dataset used in this tutorial.

Random Forest Algorithm

Decision trees involve the greedy selection of the best split point from the dataset at each step.

This algorithm makes decision trees susceptible to high variance if they are not pruned. This high variance can be harnessed and reduced by creating multiple trees wi
views of the problem) and combining their predictions. T
bagging for short.

A limitation of bagging is that the same greedy algorithm
that the same or very similar split points will be chosen i ees
will be correlated). This, in turn, makes their predictions

We can force the decision trees to be different by limiting
evaluate at each split point when creating the tree. This

Like bagging, multiple samples of the training dataset ar
difference is that at each point a split is made in the data

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

attributes can be considered.

For classification problems, the type of problems we will look at in this tutorial, the number of attributes to be considered for the split is limited to the square root of the number of input features.

```
num_features_for_split = sqrt(total_input_features)
```

The result of this one small change are trees that are more different from each other (uncorrelated) resulting predictions that are more diverse and a combined prediction that often has better performance that single tree or bagging alone.

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 variables are the strength of the returns at different angles. It is a binary classification problem that requires a model to differentiate rocks from metal cylinders. There are 208 observations.

It is a well-understood dataset. All of the variables are continuous and generally in the range of 0 to 1. The output variable is a string "M" for mine and "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 steps.

1. Calculating Splits.
2. Sonar Dataset Case Study.

These steps provide the foundation that you need to im
your own predictive modeling problems.

1. Calculating Splits

In a decision tree, split points are chosen by finding the
the lowest cost.

For classification problems, this cost function is often th
data created by the split point. A Gini index of 0 is perfe
into two groups, in the case of a two-class classification

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Finding the best split point in a decision tree involves evaluating the cost of each value in the training dataset for each input variable.

For bagging and random forest, this procedure is executed upon a sample of the training dataset, made with replacement. Sampling with replacement means that the same row may be chosen and added to the sample more than once.

We can update this procedure for Random Forest. Instead of enumerating all values for input attributes in search if the split with the lowest cost, we can create a sample of the input attributes to consider.

This sample of input attributes can be chosen randomly and without replacement, meaning that each input attribute needs only be considered once when looking for the split point with the lowest cost.

Below is a function name **get_split()** that implements this procedure. It takes a dataset and a fixed number of input features from to evaluate as input arguments, where the dataset may be a sample of the actual training dataset.

The helper function **test_split()** is used to split the dataset by a candidate split point and **gini_index()** is used to evaluate the cost of a given split by the groups of rows created.

We can see that a list of features is created by randomly selecting feature indices and adding them to a list (called **features**), this list of features is then enumerated and specific values in the training dataset evaluated as split points.

```
# Select the best split point for a dataset
def get_split(dataset, n_features):
    class_values = list(set(row[-1] for row in dataset))
    b_index, b_value, b_score, b_groups = 999, 999, 999, None
    features = list()
    while len(features) < n_features:
        index = randrange(len(dataset[0])-1)
        if index not in features:
            features.append(index)
    for index in features:
        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_(
    return {'index':b_index, 'value':b_value,
```

Now that we know how a decision tree algorithm can be ... n, we can piece this together with an implementation of ba

2. Sonar Dataset Case Study

In this section, we will apply the Random Forest algorith

The example assumes that a CSV copy of the dataset is **sonar.all-data.csv**.

×
Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

The dataset is first loaded, the string values converted to numeric and the output column is converted from strings to the integer values of 0 and 1. This is achieved with helper functions **load_csv()**, **str_column_to_float()** and **str_column_to_int()** to load and prepare the dataset.

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, our modified **get_split()** function discussed in the previous step, **to_terminal()**, **split()** and **build_tree()** used to create a single decision tree, **predict()** to make a prediction with a decision tree, **subsample()** to make a subsample of the training dataset and **bagging_predict()** to make a prediction with a list of decision trees.

A new function name **random_forest()** is developed that first creates a list of decision trees from subsamples of the training dataset and then uses them to make predictions.

As we stated above, the key difference between Random Forest and bagged decision trees is the one small change to the way that trees are created, here in the **get_split()** function.

The complete example is listed below.

```
# Random Forest Algorithm on Sonar Dataset
from random import seed
from random import randrange
from csv import reader
from math import sqrt

# 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].stri

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

Get Your Start in Machine Learning

×

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

```
    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 datase
def gini_index(groups, class_values):
    gini = 0.0
    for class_value in class_values:
        for group in groups:
            size = len(group)
            if size == 0:
                continue
            proportion = [row[-1] for row in
            gini += (proportion * (1.0 - prop
    return gini

# Select the best split point for a dataset
```

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

```
def get_split(dataset, n_features):
    class_values = list(set(row[-1] for row in dataset))
    b_index, b_value, b_score, b_groups = 999, 999, 999, None
    features = list()
    while len(features) < n_features:
        index = randrange(len(dataset[0])-1)
        if index not in features:
            features.append(index)
    for index in features:
        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, n_features, 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, n_features)
        split(node['left'], max_depth, min_size, n_features, depth+1)
    # process right child
    if len(right) <= min_size:
        node['right'] = to_terminal(right)
    else:
        node['right'] = get_split(right, n_features)
        split(node['right'], max_depth, min_size, n_features, depth+1)

# Build a decision tree
def build_tree(train, max_depth, min_size, n_features):
    root = get_split(train, n_features)
    split(root, max_depth, min_size, n_featu
    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']
```

×

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

```
# 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)

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

# Test the random forest algorithm
seed(1)
# load and prepare data
filename = 'sonar.all-data.csv'
dataset = load_csv(filename)
# convert string attributes to integers
for i in range(0, 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 = 10
min_size = 1
sample_size = 1.0
n_features = int(sqrt(len(dataset[0])-1))
for n_trees in [1, 5, 10]:
    scores = evaluate_algorithm(dataset, random_forest, n_folds, max_depth, min_size, sample_s
    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 be evaluated upon each iteration.

Deep trees were constructed with a max depth of 10 ar … of 1. Samples of the training dataset were created with the … lt expectation for the Random Forest algorithm.

The number of features considered at each split point w … ed to 7 features.

A suite of 3 different numbers of trees were evaluated fc … trees are added.

Running the example prints the scores for each fold anc

×
Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

```
Trees: 1
Scores: [51.21951219512195, 78.04878048780488, 58.536585365853654, 65.85365853658537, 53.658536
Mean Accuracy: 61.463%

Trees: 5
Scores: [63.41463414634146, 60.97560975609756, 56.09756097560976, 60.97560975609756, 56.0975609
Mean Accuracy: 59.512%

Trees: 10
Scores: [63.41463414634146, 53.65853658536586, 70.73170731707317, 56.09756097560976, 68.2926829
Mean Accuracy: 62.439%
```

Extensions

This section lists extensions to this tutorial that you may be interested in exploring.

- **Algorithm Tuning**. The configuration used in the tutorial was found with a little trial and error but was not optimized. Experiment with larger numbers of trees, different numbers of features and even different tree configurations to improve performance.
- **More Problems**. Apply the technique to other classification problems and even adapt it for regression with a new cost function and a new method for combining the predictions from trees.

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

Review

In this tutorial, you discovered how to implement the Random Forest algorithm from scratch.

Specifically, you learned:

- The difference between Random Forest and Bagged Decision Trees.
- How to update the creation of decision trees to accommodate the Random Forest procedure.
- How to apply the Random Forest algorithm 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.

Want to Code Algorithms

Code Your First Alg

...with step-by-step tutori

Discover how in my new Ebook: Mach

Get Your Start in Machine Learning

×

You can master applied Machine Learning
without the math or fancy degree.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

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

Finally, Pull Back the Curtain on
Machine Learning Algorithms

Skip the Academics. Just Results.

Click to learn more.

About Jason Brownlee

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 Bagging From Scratch With Python   How to Implement Stacked Generalization From Scratch With Python >

16 Responses to *How to Implement Random Forest From Scratch in Python*

Marco December 3, 2016 at 7:06 am # REPLY

Hi Jason,
Firstly, thanks for your work on this site – I'm finding it to be a great resource to start my exploration in python machine learning!

Now, I'm working through your python machine learning mini course and I'm up to Lesson 09: spot checking algorithms. You suggest testing the random forest which has lead me to this blog post where I'm tyring to run the recipe but get thrown the following:

Traceback (most recent call last):
File "test.py", line 203, in
scores = evaluate_algorithm(dataset, random_forest, n_f
n_features)
File "test.py", line 57, in evaluate_algorithm
folds = cross_validation_split(dataset, n_folds)
File "test.py", line 42, in cross_validation_split
index = randrange(len(dataset_copy))
File "//anaconda/lib/python3.5/random.py", line 186, in r
raise ValueError("empty range for randrange()")
ValueError: empty range for randrange()

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

I've spent the better part of the last hour trying to work out what I may be doing wrong.. unfortunately I'm really new to coding so I'm finding it very difficult. I think i've narrowed to the following possibilities:
1. possibly a problem with the evaluate_algorithm function that has been defined..?
2. possibly an issue using randrange in python 3.5.2?
3. possibly a problem with the definition of "dataset"?

I think it's either #1 because I can run the code without issue up until line 202 or #3 because dataset is the common thread in each of the returned lines from the error..?

Your guidance would be greatly appreciated!

thanks again!
marco

Marco December 4, 2016 at 10:09 pm # REPLY

Figured it out! It was a problem with using Python 3.5.2. I switched to 2.7 and it worked!

thanks
marco

Jason Brownlee December 5, 2016 at 6:49 am # REPLY

Glad to hear it Marco.

srikanth December 8, 2016 at 9:42 pm # REPLY

Traceback (most recent call last):
File "rf2.py", line 203, in
scores = evaluate_algorithm(dataset, random_forest, n_folds, max_depth, min_size, sample_size, n_trees, n_features)
File "rf2.py", line 68, in evaluate_algorithm
predicted = algorithm(train_set, test_set, *args)
File "rf2.py", line 181, in random_forest
tree = build_tree(sample, max_depth, min_size, n_featur
File "rf2.py", line 146, in build_tree
split(root, max_depth, min_size, n_features, 1)
File "rf2.py", line 120, in split
left, right = node['groups']
TypeError: 'NoneType' object is not iterable

Get Your Start in Machine Learning

×

You can master applied Machine Learning
without the math or fancy degree.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

beedotkiran December 21, 2016 at 7:00 am # REPLY

Works in python 3.x also. The division in line 45 :
fold_size = len(dataset) / n_folds
renders a float which remains valid when length of dataset_copy goes to zero. randrange(0) gives this error.

Replacing this line with
fold_size = len(dataset) // n_folds
gives an integer and the loop executes properly

Jason Brownlee December 21, 2016 at 8:50 am # REPLY

Thanks beedotkiran.

I'd recommend casting the result, in case python beginners are not familiar with the double slash operator:

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

Jason Brownlee January 3, 2017 at 9:54 am # REPLY

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

Jake Rage January 28, 2017 at 1:34 pm # REPLY

This was a fantastic tutorial thanks you for taking the time to do this! I was wondering if you had any suggestions for serialization or the tree for use against other similar data sets, would pickling working for this structure? Thanks for you help!

Jason Brownlee February 1, 2017 at 10:06 am

Hi Jake, using pickle on the learned object

Alessandro February 25, 2017 at 12:25 am #

Hi Jason, great tutorial! Just a question about tl
tree, shouldn't you use the train sample and not the whc

I mean:

Get Your Start in Machine Learning

You can master applied Machine Learning
without the math or fancy degree.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

root = get_split(train, n_features) rather than
root = get_split(dataset, n_features)

Can I ask also what are the main differences of this algorithm if you want adapt it to a regression problem rather than classification?

Thank you very much! Best regards

Alessandro February 25, 2017 at 12:28 am # REPLY

Sorry I didn't see that you had already settled the change

Jason Brownlee February 25, 2017 at 5:58 am # REPLY

No problem, nice catch!

Mike April 11, 2017 at 1:39 am # REPLY

Hello Jason great approach. I'm wondering if you have any tips about transforming the above code in order to support multi-label classification.
Thank you very much !!!

Jason Brownlee April 11, 2017 at 9:36 am # REPLY

Not off hand, sorry Mike. I would have to do some homework.

Consider a search on google scholar or consider some multi-label methods in sklearn:
http://scikit-learn.org/stable/modules/multiclass.html#multilabel-classification-format

Steve May 3, 2017 at 4:29 pm #

Hello Jason, I like the approach that allows a pe
learning methods. I look forward to learning more of the

Random forest is completely new to me. I have a datase
to know what changes are needed to make random fore
regression. This was asked earlier by Alessandro but I di
not explained well as far as I can tell.

Thanks.

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Jason Brownlee May 4, 2017 at 8:05 am # REPLY

Thanks Steve.

As a start, consider using random forest regression in the sklearn library:
http://machinelearningmastery.com/ensemble-machine-learning-algorithms-python-scikit-learn/

Leave a Reply

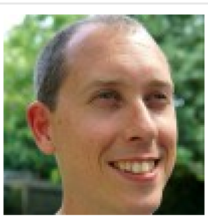
Name (required)

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 a

Read More

Get Your Start in Machine Learning

×

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Code Algorithms Fro

Discover how to code top machine learning

Code Machine Learning Algorithms From Scratch Today!

POPULAR


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

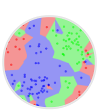

Your First Machine Learning Project in Python Step-By-Step
JUNE 10, 2016

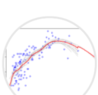

Develop Your First Neural Network in Python With Keras Step-By-Step
MAY 24, 2016


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


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


How to Run Your First Classifier in Weka
FEBRUARY 17, 2014


Tutorial To Implement k-Nearest Neighbors in Pytho
SEPTEMBER 12, 2014


A Tour of Machine Learning Algorithms
NOVEMBER 25, 2013

Get Your Start in Machine Learning

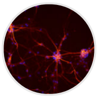
×

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

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



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

© 2017 Machine Learning Mastery. All Rights Reserved.

Privacy | Contact | About

Get Your Start in Machine Learning

×

You can master applied Machine Learning
without the math or fancy degree.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE