How to Implement Random Forest From Scratch in Python

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

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 the random forest algorithm.
- How to construct bagged decision trees with more variance.
- How to apply the random forest algorithm to a predictive modeling problem.

Let's get started.

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

How to Implement 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 with different samples of the training dataset (different views of the problem) and combining their predictions. This approach is called bootstrap aggregation or bagging for short.

A limitation of bagging is that the same greedy algorithm is used to create each tree, meaning that it is likely that the same or very similar split points will be chosen in each tree making the different trees very similar (trees will be correlated). This, in turn, makes their predictions similar, mitigating the variance originally sought.

We can force the decision trees to be different by limiting the features (rows) that the greedy algorithm can evaluate at each split point when creating the tree. This is called the Random Forest algorithm.

Like bagging, multiple samples of the training dataset are taken and a different tree trained on each. The difference is that at each point a split is made in the data and added to the tree, only a fixed subset of 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 implement and apply the Random Forest algorithm to your own predictive modeling problems.

1. Calculating Splits

In a decision tree, split points are chosen by finding the attribute and the value of that attribute that results in the lowest cost.

For classification problems, this cost function is often the Gini index, that calculates the purity of the groups of data created by the split point. A Gini index of 0 is perfect purity where class values are perfectly separated into two groups, in the case of a two-class classification problem.

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

Now that we know how a decision tree algorithm can be modified for use with the Random Forest algorithm, we can piece this together with an implementation of bagging and apply it to a real-world dataset.

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 name **sonar.all-data.csv**.

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].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
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, 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_features, 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)

# 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(2)
# 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_size, n_trees, n_features)
    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 and a minimum number of training rows at each node of 1. Samples of the training dataset were created with the same size as the original dataset, which is a default expectation for the Random Forest algorithm.

The number of features considered at each split point was set to sqrt(num_features) or sqrt(60)=7.74 rounded to 7 features.

A suite of 3 different numbers of trees were evaluated for comparison, showing the increasing skill as more trees are added.

Running the example prints the scores for each fold and mean score for each configuration.

```
Trees: 1
Scores: [56.09756097560976, 63.41463414634146, 60.97560975609756, 58.536585365853654, 73.17073170731707]
Mean Accuracy: 62.439%

Trees: 5
Scores: [70.73170731707317, 58.536585365853654, 85.36585365853658, 75.60975609756098, 63.41463414634146]
Mean Accuracy: 70.732%

Trees: 10
Scores: [75.60975609756098, 80.48780487804879, 92.6829268292683, 73.17073170731707, 70.73170731707317]
Mean Accuracy: 78.537%
```

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.

Tweet Share Share

About Jason Brownlee

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

View all posts by Jason Brownlee →

‹ How to Implement Bagging From Scratch With Python

How to Implement Stacked Generalization (Stacking) From Scratch With Python ›

94 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_folds, max_depth, min_size, sample_size, n_trees, 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 randrange
raise ValueError("empty range for randrange()")
ValueError: empty range for randrange()

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

Dionysis June 4, 2017 at 4:05 am #

REPLY

Hi,

Is it simple to adapt this implementation in order to accommodate tuples of feature vectors?

Thanks,
D.

Jeffrey Grover July 28, 2017 at 8:30 am #

REPLY

Hi Jason, I was able to get the code to run and got the results as posted on this page. My question what next? How do you use these results to make classification on new data?

Thanks Jeff

Jason Brownlee July 28, 2017 at 8:41 am #

REPLY

You can fit a final model on all training data and start making predictions.

See this post about developing a final model:
http://machinelearningmastery.com/train-final-machine-learning-model/

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_features)
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

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 #

REPLY

Hi Jake, using pickle on the learned object would be a good starting point.

Alessandro February 25, 2017 at 12:25 am #

REPLY

Hi Jason, great tutorial! Just a question about the function build_tree: when you evaluate the root of the tree, shouldn't you use the train sample and not the whole dataset?

I mean:

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 #

REPLY

Hello Jason, I like the approach that allows a person to 'look under the hood' of these machine learning methods. I look forward to learning more of the machine learning methods this way.

Random forest is completely new to me. I have a dataset that could use random forest regression. I would like to know what changes are needed to make random forest classification code (above) into random forest regression. This was asked earlier by Alessandro but I didn't understand the reply. Random forest regression is not explained well as far as I can tell.

Thanks.

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/

Steve Hansen June 9, 2017 at 10:29 am #

REPLY

Jason,

Thanks for the advice with random forest regression.

On the sonar dataset, I plotted a 60 x 60 correlation matrix from the data. Many of the successive rows, and even not so close rows, are highly correlated. For instance, row 17 and column 18 have the following correlation:

Number of Observations: 131
Number of Degrees of Freedom: 2

R-squared: 0.870
Rmse: 0.1046
F statistic 863.

and columns 14 and 15 have the correlation

Number of Observations: 131
Number of Degrees of Freedom: 2

R-squared: 0.8554
Rmse: 0.0708
F statistic 763.

What impact does this correlation have on the use of random forest? What can be done to remove or measure the effect of the correlation?

Also, for this dataset I was able to get the following results:

n_folds = 5
max_depth = 12
min_size = 1
sample_size = 0.75
for n_trees in [1, 5, 19]:

71.875%, 73.438%, 82.031%

Thanks for the great work. I am trying to absorb it all.

Jason Brownlee June 10, 2017 at 8:12 am #

REPLY

Nice results.

I don't think RF is too affected by highly corrected features. Nevertheless, try removing some and see how it impacts model skill. I'd love to hear what you discover.

dhrumil January 30, 2018 at 7:15 pm #

REPLY

how did you find correlation and why would it create a problem.I am kinda new to this so I would like to know these things from experts like you.Thank you.

danniel June 17, 2017 at 12:52 am #

REPLY

Hello Jason,thanks for awesome tutorial,can you please explain following things>
1.what is function of this line : row_copy[-1] = None : because it works perfectly without this line
2.When i tried n_trees=[3,5,10] it returned following result in which accuracy decreases with more trees>
Trees: 3
Scores: [63.41463414634146, 51.21951219512195, 68.29268292682927, 68.29268292682927, 63.41463414634146]
Mean Accuracy: 62.927%
Trees: 5
Scores: [65.85365853658537, 60.97560975609756, 60.97560975609756, 60.97560975609756, 58.536585365853654]
Mean Accuracy: 61.463%
Trees: 10
Scores: [48.78048780487805, 60.97560975609756, 58.536585365853654, 70.73170731707317, 53.65853658536586]
Mean Accuracy: 58.537%

Jason Brownlee June 17, 2017 at 7:32 am #

REPLY

1. To clear the output value so the algorithm/developer cannot accidentally cheat.
2. Yes, it is important to tune an algorithm to a problem.

Danniel June 17, 2017 at 9:39 pm #

REPLY

Would you like to help me?I am a student and I am using this for a problem that I found online
>https://github.com/barotdhrumil21/road_sign_prediction_using_random_forest_classifier/tree/master

Jason Brownlee June 18, 2017 at 6:30 am #

REPLY

I would recommend contacting the author of that code.

danniel June 18, 2017 at 12:55 am #

REPLY

how do you suggest I should use this :https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/learn/random_forest_mnist.py
or can I use it and is it same what you've done?

Jason Brownlee June 18, 2017 at 6:32 am #

REPLY

Use whatever code you like.

chris June 19, 2017 at 7:02 pm #

REPLY

nice job! what kind of cost function should i use when doing regression problems?

Jason Brownlee June 20, 2017 at 6:36 am #

REPLY

Great question, consider mean squared error or mean absolute error.

joe June 20, 2017 at 12:30 am #

REPLY

test_split has return two values but here groups = test_split(index, row[index], dataset) just one variable, can anyone explain that, please, thanks a lot

Jason Brownlee June 20, 2017 at 6:37 am #

REPLY

The returned array is assigned a variable named groups.

Chiky July 6, 2017 at 5:13 pm #

REPLY

Hi Jason,
I am trying to learn RF through your sample example. But while running the code I am getting an error. I am using Ipython Notebook.

in split(node, max_depth, min_size, n_features, depth)
6 # Create child splits for a node or make terminal
7 def split(node, max_depth, min_size, n_features, depth):

—-> 8 left, right = node['groups']
9 del(node['groups'])
10 # check for a no split

TypeError: 'NoneType' object is not iterable

Please help.

Chiky July 6, 2017 at 5:19 pm #

REPLY

The chain error list:

TypeError Traceback (most recent call last)
in ()
16 n_features = int(sqrt(len(dataset[0])-1))
17 for n_trees in [1, 5, 10]:
—> 18 scores = evaluate_algorithm(dataset, random_forest, n_folds, max_depth, min_size, sample_size, n_trees, n_features)
19 print('Trees: %d' % n_trees)
20 print('Scores: %s' % scores)

in evaluate_algorithm(dataset, algorithm, n_folds, *args)
12 test_set.append(row_copy)
13 row_copy[-1] = None
—> 14 predicted = algorithm(train_set, test_set, *args)
15 actual = [row[-1] for row in fold]
16 accuracy = accuracy_metric(actual, predicted)

in random_forest(train, test, max_depth, min_size, sample_size, n_trees, n_features)
18 for i in range(n_trees):
19 sample = subsample(train, sample_size)
—> 20 tree = build_tree(sample, max_depth, min_size, n_features)
21 trees.append(tree)
22 predictions = [bagging_predict(trees, row) for row in test]

in build_tree(train, max_depth, min_size, n_features)
2 def build_tree(train, max_depth, min_size, n_features):
3 root = get_split(train, n_features)
—-> 4 split(root, max_depth, min_size, n_features, 1)
5 return root
6

in split(node, max_depth, min_size, n_features, depth)
6 # Create child splits for a node or make terminal
7 def split(node, max_depth, min_size, n_features, depth):
—-> 8 left, right = node['groups']
9 del(node['groups'])
10 # check for a no split

TypeError: 'NoneType' object is not iterable

Jason Brownlee July 9, 2017 at 10:25 am #

REPLY

Sorry, I don't use notebooks. Confirm Python version 2.

Danny Shterman July 13, 2017 at 11:53 pm #

REPLY

Shouldn't dataset be sorted by a feature before calculating gini?

Tatiana July 14, 2017 at 9:50 am #

REPLY

Hello, Jason
Thank you very much for your lessons. You code worked perfectly.

Now I am trying to use different dataset, which has also string values. And having difficulty with it. Is it even possible? I keep getting errors that cannot convert string to integer.

Jason Brownlee July 15, 2017 at 9:34 am #

REPLY

Thanks.

You must convert the strings to integers or real values. Perhaps you need to use a one hot encoding?

Danny July 17, 2017 at 5:40 pm #

REPLY

Hi,
is there a need to perform a sum of the the weighted gini indexes for each split?
Thanks
Danny

Jeffrey Grover July 29, 2017 at 6:19 am #

REPLY

Hi Jason, I have posted this protocol on YouTube as a reference @ https://youtu.be/Appc0Hpnado

Thanks for taking the time to teach us this method!

Jeff

Jason Brownlee July 29, 2017 at 8:13 am #

REPLY

Nice one Jeffrey!

Wells August 31, 2017 at 2:47 pm #

REPLY

Hi Jason, your implementation helps me a lot! However, I have a question here: on each split, the algorithm randomly selects a subset of features from the total features and then pick the best feature with the best gini score. Then, is it possible for a tree that a single feature is used repeatedly during different splits? since in get_split(), the line index = randrange(len(dataset[0])-1) basically pick features from the whole pool. Could you explain this? Thanks!

Jason Brownlee September 1, 2017 at 6:41 am #

REPLY

It does not choose the best split, but a random split from among the best.

You can split a single feature many times, if it makes sense from a gini-score perspective.

Wells September 1, 2017 at 1:36 pm #

REPLY

Yeah I realized this point. Thanks!

Ria September 22, 2017 at 10:51 am #

REPLY

rf_model = training(training_data2,RandomForestClassifier())
print rf_model
test(rf_model,test_data2)

RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
max_depth=None, max_features='auto', max_leaf_nodes=None,
min_impurity_split=1e-07, min_samples_leaf=1,

min_samples_split=2, min_weight_fraction_leaf=0.0,
n_estimators=10, n_jobs=1, oob_score=False, random_state=None,
verbose=0, warm_start=False)
I tried using number of trees =1,5,10 as per your example but not working could you pls say me where shld i need to make changes and moreover when i set randomstate = none each time i execute my accuracy keeps on changing but when i set a value for the random state giving me same accuracy.

DATAEXPERT October 15, 2017 at 4:58 am #

REPLY

Hi,
I would like to change the code so it will work for 90% of data for train and 10% for test, with no folds. If I use n_folds = 1, I get an error. How can I change the code so it will work?

Jason Brownlee October 15, 2017 at 5:23 am #

REPLY

Perhaps you would be better served by using scikit-learn to fit your model:
https://machinelearningmastery.com/ensemble-machine-learning-algorithms-python-scikit-learn/

DATAEXPERT October 15, 2017 at 6:54 am #

REPLY

Thanks a lot. I would like to use your code since I made another internal change of the algorithm that can't be done using scikit-learn. I think the major (may be the only) change is in the evaluate_algorithm function.

Ali October 28, 2017 at 2:23 am #

REPLY

Can we implement random forest using fitctree in matlab?

There is a function call TreeBagger that can implement random forest. However, if we use this function, we have no control on each individual tree. Can we use the MATLAB function fitctree, which build a decision tree, to implement random forest? Thanks a lot.

Jason Brownlee October 28, 2017 at 5:15 am #

REPLY

Sorry, I don't have material on matlab.

Kuber Jain November 22, 2017 at 12:06 pm #

REPLY

Hi Jason,

I am trying to solve classification problem using RF, and each time I run RandomForestClassifier on my training data, feature importance shows different features everytime I run it. How can I make sure it gives me same top 5 features everytime I run the model ? Please let me know.

model_rc = RandomForestClassifier(n_estimators=10,max_depth=None,min_samples_split=2,random_state=0)
rc_fit=model_rc.fit(X_train, y_train.values.ravel())

Jason Brownlee November 23, 2017 at 10:23 am #

REPLY

Yes, this is a feature of the algorithm.

See this post:
https://machinelearningmastery.com/randomness-in-machine-learning/

Khin January 3, 2018 at 2:15 am #

REPLY

I would like to know the difference between sklearn randomforest and random forest algorithm implemented by oneself. Is there any weakness or something in sklearn randomforest?

Jason Brownlee January 3, 2018 at 5:39 am #

REPLY

I would recommend only implementing the algorithm yourself for learning, I would expect the sklearn implementation will be more robust and efficient.

PSN January 3, 2018 at 5:43 pm #

REPLY

Could you implement rotation forest algorithm ?

Jason Brownlee January 4, 2018 at 8:06 am #

REPLY

Thanks for the suggestion.

Sterling January 4, 2018 at 11:01 am #

REPLY

Your blogs and tutorials have aided me throughout my PhD. Thank you for putting so much time and effort into sharing this information.

Jason Brownlee January 4, 2018 at 3:26 pm #

REPLY

Thanks, I'm really glad to hear that!

Ioannis February 5, 2018 at 3:50 am #

REPLY

Dear Jason,

thank you very much for this implementation, fantastic work!

Is it possible to know which features are most discriminative
for the task at hand and maybe the degree of importance
for each of these features?

Many thanks in advance !!!

Jason Brownlee February 5, 2018 at 7:47 am #

REPLY

Yes, you can use feature selection methods:
http://machinelearningmastery.com/an-introduction-to-feature-selection/

K February 28, 2018 at 4:22 pm #

REPLY

Hi Jason,

Thanks for sharing! I wonder how fast is your implementation. (I guess I should try it out myself. 🙂

I was a master student in biostatistics and doing a thesis project which applied a modified random forest (no existing implementation) to solve a problem. I was and still I am only comfortable with R. I implemented the modified random forest from scratch in R. Although I tried hard to improve my code and implement some parts in C++ (via Rcpp package), it was still so slow… I noticed random forests packages in R or Python were all calling codes writing in C at its core.

So, would you mind estimate how fast is your implementation comparing to mainstream implementation (e.g. 10 times slower than Scikit-learn) ?
I am new to Python. I should really try it myself but just can't help ask for a quick answer for this to inspire me to learn Python! 🙂

Thanks!

Kehao

Jason Brownlee March 1, 2018 at 6:07 am #

REPLY

It is slow. It is for learning purposes only.

Kirtika Bhatt March 23, 2018 at 6:55 am #

REPLY

I am new to python and doing a mini project for self learning. It will be helpful if you guide that how can I use this algorithm to predict the class of some test data.

Jason Brownlee March 23, 2018 at 8:28 am #

REPLY

A good place to start is here:
https://machinelearningmastery.com/start-here/#python

niloofar April 6, 2018 at 7:02 pm #

REPLY

Hi Jason,
I might send another message but I am not sure if it had sent or not.
I just wanted to say thank you for your informative website.
this post was also and very comprehensive with full of integrated ideas and topics.
If the python project is available I would appreciate if you send it.

best regards,
niloofar

Jason Brownlee April 7, 2018 at 6:22 am #

REPLY

Thanks, I'm glad it helped!

samiha May 24, 2018 at 10:32 pm #

REPLY

hi
please how can i evaluate the algorithme !?

Jason Brownlee May 25, 2018 at 9:26 am #

REPLY

Great question, I answer it here:
https://machinelearningmastery.com/faq/single-faq/how-do-i-evaluate-a-machine-learning-algorithm

Adeola May 30, 2018 at 9:30 pm #

REPLY

Good Dr Brownlee,

Kudos for the good work sir, I have a quick question sir. How long did it take you to write such a wonderful piece of code up and what are the resources you used to help you sir?
Thank you sir and kind regards.

Jason Brownlee May 31, 2018 at 6:16 am #

REPLY

Perhaps a day or two. I used some textbooks.

Piyasi Choudhury June 30, 2018 at 12:16 pm #

REPLY

Great work Jason..wonder if I can use this to conceptualize a 3 way split tree – a tree that can have 3 classes, instead of binary?

Jason Brownlee July 1, 2018 at 6:22 am #

REPLY

I don't see why not.

Then September 24, 2018 at 6:15 pm #

REPLY

Hi…
How can I implement this code for multiclass classification?.

Jason Brownlee September 25, 2018 at 6:18 am #

REPLY

Perhaps this tutorial is a bit advanced, I would recommend using scikit-learn to get started:
https://machinelearningmastery.com/start-here/#python

Then September 25, 2018 at 4:14 pm #

REPLY

Hi!
How can I implement your code for multi-class classification?
Thanks.

Jason Brownlee September 26, 2018 at 6:10 am #

REPLY

I would encourage you to use scikit-learn instead, as modifying this example for multi-class classification is not for beginners.

Fraser October 24, 2018 at 2:50 am #

REPLY

Hello Jason,

You might never see this because its been so long since posted this article.

I am running your code with python 3.6 in PyCharm and I noticed that if I comment out the

def str_column_to_int(dataset, column)

function then the code runs just fine but the accuracy scores change to this:

Trees: 1
Scores: [56.09756097560976, 63.41463414634146, 60.97560975609756, 58.536585365853654, 73.17073170731707]
Mean Accuracy: 62.439%
Trees: 5
Scores: [70.73170731707317, 58.536585365853654, 85.36585365853658, 75.60975609756098, 63.41463414634146]
Mean Accuracy: 70.732%
Trees: 10

Scores: [82.92682926829268, 75.60975609756098, 97.5609756097561, 80.48780487804879, 68.29268292682927]
Mean Accuracy: 80.976%

Process finished with exit code 0

The accuracy increases for the 10 trees.

Any idea whats going on?

Jason Brownlee October 24, 2018 at 6:32 am #

REPLY

More trees is better for this problem!

Shipika Singh October 24, 2018 at 6:12 am #

REPLY

hi,
very nice explanation! but I am thinking what if I create a random forest from a dataset and then pass a single document to test it. what will be the method to pass a single document in the clf of random forest?

Jason Brownlee October 24, 2018 at 6:34 am #

REPLY

This tutorial is for learning how random forest works. If you are working on a project, I'd recommend that you use random forest in sklearn.

Elizabeth October 24, 2018 at 12:22 pm #

REPLY

HI Jason,
This is a great tutorial. I've been working on a random forest project in R and have been reading alot about using this method. I'm confused because some

articles note that RF will NOT overfit, yet there seems to be a constant discussion about overfitting with RF in stackoverflow. Do RF models overfit? My second question pertains to the Gini decrease scores–are these impacted by correlated variables ? (I know RF handles correlated predictor variables fairly well). Final question– if using CV in caret, is train/test sample necessary? I have a very unbalanced outcome classifier and not a ton of data, so I didn't want to split it further, unless absolutely necessary.
Thank you!

Jason Brownlee October 24, 2018 at 2:46 pm #

REPLY

Generally, bagged trees don't overfit. I've read this and observed this, it might even be true.

Trees are invariant to correlated inputs.

Probability just CV or train/test would be sufficient, probably not both.

amjad December 16, 2018 at 9:22 pm #

REPLY

i have ten variables one dependent and nine independent first i will take sample of independent then random sample of observation and after that of preductive model

Jason Brownlee December 17, 2018 at 6:20 am #

REPLY

Random forest will choose split points using independent variables only.

Julie January 4, 2019 at 3:23 am #

REPLY

Hi Jason,

Thank you for your great work !

I am running your code with python 3.7 in Spyder but I have this error :
"left, right = node['groups']
TypeError: cannot unpack non-iterable NoneType object".

I don't understand why… Do you have an idea ?

Thanks.

Jason Brownlee January 4, 2019 at 6:33 am #

REPLY

Perhaps try saving all code to a file and running from the command line instead:
https://machinelearningmastery.com/faq/single-faq/how-do-i-run-a-script-from-the-command-line

bbrighttaer January 15, 2019 at 5:35 pm #

REPLY

Hello Dr. Jason,
Thanks so much for this wonderful website and the amazing work you do over here.
I went through your tutorial and had the same accuracy as found in it the tutorial. I realized that the attributes are selected with replacement so I made the modification and applied cross entropy loss for n_trees = [1, 5, 10, 15, 20]. I had the following accuracy metrics:

Trees: 1
Scores: [68.29268292682927, 63.41463414634146, 65.85365853658537, 73.17073170731707, 75.60975609756098]
Mean Accuracy: 69.268%

Trees: 5
Scores: [73.17073170731707, 82.92682926829268, 70.73170731707317, 70.73170731707317, 75.60975609756098]
Mean Accuracy: 74.634%

Trees: 10
Scores: [80.48780487804879, 75.60975609756098, 65.85365853658537, 75.60975609756098, 87.8048780487805]

Mean Accuracy: 77.073%

Trees: 15
Scores: [90.2439024390244, 70.73170731707317, 78.04878048780488, 73.17073170731707, 80.48780487804879]
Mean Accuracy: 78.537%

Trees: 20
Scores: [65.85365853658537, 75.60975609756098, 85.36585365853658, 87.8048780487805, 85.36585365853658]
Mean Accuracy: 80.000%

Jason Brownlee January 16, 2019 at 5:42 am #

REPLY

Well done!

Marcin January 25, 2019 at 1:48 am #

REPLY

Hello Jason,

it looks like I wrote a comment to not proper article before 🙂

I am inspired and wrote the python random forest classifier from this site. I go one more step further and decided to implement Adaptive Random Forest algorithm. But I faced with many issues.
I implemented the window, where I store examples. But unfortunately, I am unable to perform the classification. I cannot translate the learning step to be a little adaptive. I'm stuck.

Could you give me some advices, examples, how to overcome this issues ?

Jason Brownlee January 25, 2019 at 8:45 am #

REPLY

Sorry, I don’t have an example of adaptive random forest, I’ve not heard of it before.

Marcin January 25, 2019 at 11:20 pm #

REPLY

This is an random forest which is able to learn from streams. It includes some concept drift detection method. This is not common topic unfortunately.

Jason Brownlee January 26, 2019 at 6:14 am #

REPLY

Intersting. Good luck with your project!

Marcin February 2, 2019 at 1:31 am #

REPLY

Hello Jason,

Could you explain me how is it possible, that every time I am running your script I always receive the same scores ?
This means that in fact we do not implement random mechanism.

Jason Brownlee February 2, 2019 at 6:23 am #

REPLY

I fix the random number seed.

You can learn more here:
https://machinelearningmastery.com/introduction-to-random-number-generators-for-machine-learning/

Leave a Reply

Name (required)

Email (will not be published) (required)

Website

SUBMIT COMMENT

Welcome to Machine Learning Mastery!

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

Read More

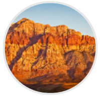
POPULAR


How to Develop LSTM Models for Multi-Step Time Series Forecasting of Household Power Consumption
OCTOBER 10, 2018


So, You are Working on a Machine Learning Problem…
APRIL 4, 2018


11 Classical Time Series Forecasting Methods in Python (Cheat Sheet)
AUGUST 6, 2018

How to Make Predictions with Keras
APRIL 9, 2018



How to Develop LSTM Models for Time Series Forecasting
NOVEMBER 14, 2018

How to Make Predictions with scikit-learn
APRIL 6, 2018

You're Doing it Wrong. Why Machine Learning Does Not Have to Be So Hard
MARCH 28, 2018

Top beginner tutorials:

- How to Install Python for Machine Learning
- Your First Machine Learning Project in Python
- Your First Neural Network in Python
- Your First Classifier in Weka
- Your First Time Series Forecasting Project

Top project tutorials:

- Photo Captioning Project
- Neural Machine Translation Project
- Sentiment Analysis Project
- Power Forecasting Project