Navigation

Machine Learning Mastery
Making Developers Awesome at Machine Learning

Click to Take the FREE Statistics Crash-Course

Search...

How to Calculate Bootstrap Confidence Intervals For Machine Learning Results in Python

by Jason Brownlee on June 5, 2017 in Statistics

Tweet Tweet Share Share

Last Updated on August 14, 2020

It is important to both present the expected skill of a machine learning model a well as confidence intervals for that model skill.

Confidence intervals provide a range of model skills and a likelihood that the model skill will fall between the ranges when making predictions on new data. For example, a 95% likelihood of classification accuracy between 70% and 75%.

A robust way to calculate confidence intervals for machine learning algorithms is to use the bootstrap. This is a general technique for estimating statistics that can be used to calculate empirical confidence intervals, regardless of the distribution of skill scores (e.g. non-Gaussian)

In this post, you will discover how to use the bootstrap to calculate confidence intervals for the performance of your machine learning algorithms.

After reading this post, you will know:

- How to estimate confidence intervals of a statistic using the bootstrap.
- How to apply this method to evaluate machine learning algorithms.
- How to implement the bootstrap method for estimating confidence intervals in Python.

Kick-start your project with my new book Statistics for Machine Learning, including *step-by-step tutorials* and the *Python source code* files for all examples.

Let's get started.

- **Update June/2017**: Fixed a bug where the wrong values were provided to numpy.percentile(). Thanks Elie Kawerk.
- **Update March/2018**: Updated link to dataset file.

How to Calculate Bootstrap Confidence Intervals For Machine Learning Results in Python
Photo by Hendrik Wieduwilt, some rights reserved.

Bootstrap Confidence Intervals

Calculating confidence intervals with the bootstrap involves two steps:

1. Calculate a Population of Statistics
2. Calculate Confidence Intervals

1. Calculate a Population of Statistics

The first step is to use the bootstrap procedure to resample the original data a number of times and calculate the statistic of interest.

The dataset is sampled with replacement. This means that each time an item is selected from the original dataset, it is not removed, allowing that item to possibly be selected again for the sample.

The statistic is calculated on the sample and is stored so that we build up a population of the statistic of interest.

The number of bootstrap repeats defines the variance of the estimate, and more is better, often hundreds or thousands.

We can demonstrate this step with the following pseudocode.

```
statistics = []
for i in bootstraps:
    sample = select_sample_with_replacement(data)
    stat = calculate_statistic(sample)
    statistics.append(stat)
```

2. Calculate Confidence Interval

Now that we have a population of the statistics of interest, we can calculate the confidence intervals.

This is done by first ordering the statistics, then selecting values at the chosen percentile for the confidence interval. The chosen percentile in this case is called alpha.

For example, if we were interested in a confidence interval of 95%, then alpha would be 0.95 and we would select the value at the 2.5% percentile as the lower bound and the 97.5% percentile as the upper bound on the statistic of interest.

For example, if we calculated 1,000 statistics from 1,000 bootstrap samples, then the lower bound would be the 25th value and the upper bound would be the 975th value, assuming the list of statistics was ordered.

In this, we are calculating a non-parametric confidence interval that does not make any assumption about the functional form of the distribution of the statistic. This confidence interval is often called the empirical confidence interval.

We can demonstrate this with pseudocode below.

```
ordered = sort(statistics)
lower = percentile(ordered, (1-alpha)/2)
upper = percentile(ordered, alpha+((1-alpha)/2))
```

Bootstrap Model Performance

The bootstrap can be used to evaluate the performance of machine learning algorithms.

The size of the sample taken each iteration may be limited to 60% or 80% of the available data. This will mean that there will be some samples that are not included in the sample. These are called out of bag (OOB) samples.

A model can then be trained on the data sample each bootstrap iteration and evaluated on the out of bag samples to give a performance statistic, which can be collected and from which confidence intervals may be calculated.

We can demonstrate this process with the following pseudocode.

```
statistics = []
for i in bootstraps:
    train, test = select_sample_with_replacement(data, size)
    model = train_model(train)
    stat = evaluate_model(test)
    statistics.append(stat)
```

Calculate Classification Accuracy Confidence Interval

This section demonstrates how to use the bootstrap to calculate an empirical confidence interval for a machine learning algorithm on a real-world dataset using the Python machine learning library scikit-learn.

This section assumes you have Pandas, NumPy, and Matplotlib installed. If you need help setting up your environment, see the tutorial:

- How to Setup a Python Environment for Machine Learning and Deep Learning with Anaconda

First, download the Pima Indians dataset and place it in your current working directory with the filename "pima–indians-*diabetes.data.csv*" (update: download here).

We will load the dataset using Pandas.

```
# load dataset
data = read_csv('pima-indians-diabetes.data.csv', header=None)
values = data.values
```

Next, we will configure the bootstrap. We will use 1,000 bootstrap iterations and select a sample that is 50% the size of the dataset.

```
# configure bootstrap
n_iterations = 1000
n_size = int(len(data) * 0.50)
```

Next, we will iterate over the bootstrap.

The sample will be selected with replacement using the resample() function from sklearn. Any rows that were not included in the sample are retrieved and used as the test dataset. Next, a decision tree classifier is fit on the sample and evaluated on the test set, a classification score calculated, and added to a list of scores collected across all the bootstraps.

```
# run bootstrap
stats = list()
for i in range(n_iterations):
    # prepare train and test sets
    train = resample(values, n_samples=n_size)
    test = numpy.array([x for x in values if x.tolist() not in train.tolist()])
    # fit model
    model = DecisionTreeClassifier()
    model.fit(train[:,:-1], train[:,-1])
    # evaluate model
    predictions = model.predict(test[:,:-1])
    score = accuracy_score(test[:,-1], predictions)
```

Once the scores are collected, a histogram is created to give an idea of the distribution of scores. We would generally expect this distribution to be Gaussian, perhaps with a skew with a symmetrical variance around the mean.

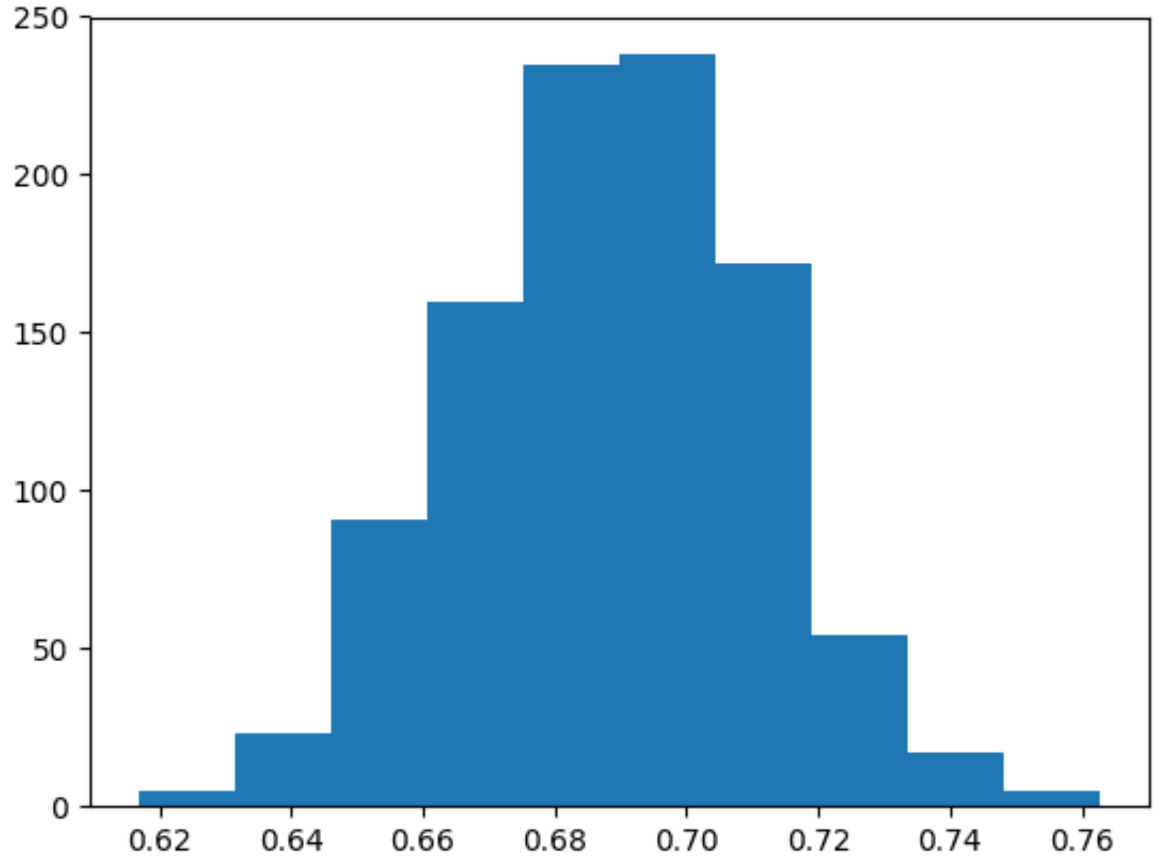
Finally, we can calculate the empirical confidence intervals using the percentile() NumPy function. A 95% confidence interval is used, so the values at the 2.5 and 97.5 percentiles are selected.

Putting this all together, the complete example is listed below.

```
import numpy
from pandas import read_csv
from sklearn.utils import resample
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from matplotlib import pyplot
# load dataset
data = read_csv('pima-indians-diabetes.data.csv', header=None)
values = data.values
# configure bootstrap
n_iterations = 1000
n_size = int(len(data) * 0.50)
# run bootstrap
stats = list()
for i in range(n_iterations):
    # prepare train and test sets
    train = resample(values, n_samples=n_size)
    test = numpy.array([x for x in values if x.tolist() not in train.tolist()])
    # fit model
    model = DecisionTreeClassifier()
    model.fit(train[:,:-1], train[:,-1])
    # evaluate model
    predictions = model.predict(test[:,:-1])
    score = accuracy_score(test[:,-1], predictions)
    print(score)
    stats.append(score)
# plot scores
pyplot.hist(stats)
pyplot.show()
# confidence intervals
alpha = 0.95
p = ((1.0-alpha)/2.0) * 100
lower = max(0.0, numpy.percentile(stats, p))
p = (alpha+((1.0-alpha)/2.0)) * 100
upper = min(1.0, numpy.percentile(stats, p))
print('%.1f confidence interval %.1f%% and %.1f%%' % (alpha*100, lower*100, upper*100))
```

Running the example prints the classification accuracy each bootstrap iteration.

A histogram of the 1,000 accuracy scores is created showing a Gaussian-like distribution.



Distribution of Classification Accuracy Using the Bootstrap

Finally, the confidence intervals are reported, showing that there is a 95% likelihood that the confidence interval 64.4% and 73.0% covers the true skill of the model.

```
...
0.646288209607
0.682203389831
0.668085106383
0.673728813559
0.686021505376
95.0 confidence interval 64.4% and 73.0%
```

This same method can be used to calculate confidence intervals of any other errors scores, such as root mean squared error for regression algorithms.

Further Reading

This section provides additional resources on the bootstrap and bootstrap confidence intervals.

- An Introduction to the Bootstrap, 1996
- Bootstrap Confidence Intervals, Statistical Science, 1996
- Section 5.2.3, Bootstrap Confidence Intervals, Empirical Methods for Artificial Intelligence
- Bootstrapping on Wikipedia
- Section 4.4 Resampling Techniques, Applied Predictive Modeling

Summary

In this post, you discovered how to use the bootstrap to calculate confidence intervals for machine learning algorithms.

Specifically, you learned:

- How to calculate the bootstrap estimate of confidence intervals of a statistic from a dataset.
- How to apply the bootstrap to evaluate machine learning algorithms.
- How to calculate bootstrap confidence intervals for machine learning algorithms in Python.

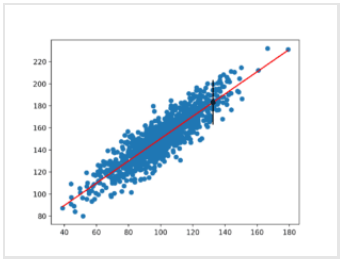
Do you have any questions about confidence intervals?
Ask your questions in the comments below.

Tweet Tweet Share Share

More On This Topic

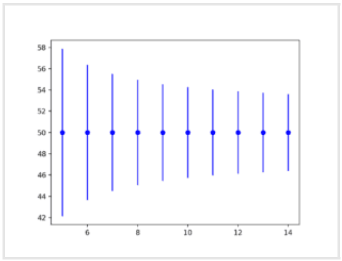
Confidence Intervals for Machine Learning

Statistics for Evaluating Machine Learning Models



Prediction Intervals for Machine Learning

A Gentle Introduction to the Bootstrap Method



A Gentle Introduction to Statistical Tolerance…

Prediction Intervals for Deep Learning Neural Networks

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 Report Classifier Performance with Confidence Intervals

The 5 Step Life-Cycle for Long Short-Term Memory Models in Keras >

132 Responses to *How to Calculate Bootstrap Confidence Intervals For Machine Learning Results in Python*

benson dube June 5, 2017 at 6:40 pm # REPLY

Thank you Jason

Jason Brownlee June 6, 2017 at 9:23 am # REPLY

I'm glad you found the post useful.

Alassane June 9, 2017 at 4:11 am # REPLY

Where can i get your book "Mastering machine learning with python"

Jason Brownlee June 9, 2017 at 6:30 am # REPLY

Here:
https://machinelearningmastery.com/machine-learning-with-python/

Divya September 18, 2019 at 3:10 pm # REPLY

Sir ,how we can apply bootstrap resampling for software effort interval prediction and what is the procedure and initial steps that are to be followed.

Jason Brownlee September 19, 2019 at 5:50 am # REPLY

Collect your data, load it in Python and apply the procedure to estimate the desired quantity.

Which part are you struggling with exactly?

Elie Kawerk June 5, 2017 at 8:25 pm # REPLY

Thank you Jason,

Are you considering to include the posts on confidence intervals in a previous (or new) book?

Regards
Elie

Jason Brownlee June 6, 2017 at 9:33 am # REPLY

Hi Elie,

Not especially. I have a soft idea of "statistical methods for machine learning" or something to that effect that could cover topics like sampling theory and confidence intervals. Posts like this are a test to see if there is interest.

Why do you ask? Is this a topic for which you're looking for more help?

Elie Kawerk June 6, 2017 at 3:31 pm # REPLY

Yes,

I actually find posts like these very useful for reporting statistically meaningful results for machine learning. Here's a nice free book link on the subject: https://www.otexts.org/book/sfml

Regards

Elie June 6, 2017 at 5:22 pm # REPLY

Jason,

I have another question. How do we report confidence intervals on an evaluation done on a hold-out set? We cannot apply the bootstrap with training here since this would contaminate our results.

Regards

Jason Brownlee June 7, 2017 at 7:09 am #

Generally, you can repeat the holdout process many times with different random samples and use the outcomes as your population of results.

Jason Brownlee June 7, 2017 at 7:08 am # REPLY

I agree, it's a great book!

Elie Kawerk June 7, 2017 at 3:21 pm #

Wouldn't it be suitable to apply the bootstrap by sampling from one hold out set for a certain number of iterations without any training involved?

Jason Brownlee June 8, 2017 at 7:38 am #

As in only resampling the set used to evaluate the skill of the model?

No, I don't think this would be valid.

Elie Kawerk June 10, 2017 at 4:09 am # REPLY

Jason,

Something seems not to make sense in the confidence interval you obtain. The one you calculate is [61.5, 63.9]. However the mean (that I obtain) is about 69% (similar to the one you get from the graph) and visually one can inspect and estimate the 95% CI to be [64, 74]. I think there is something wrong.

Please correct me if I am wrong.

Regards
Elie

Jason Brownlee June 10, 2017 at 8:40 am # REPLY

You are correct Elie, thank you so much.

The values provided to numpy.percentile() must be in [0,100] and I was providing them in [0,1].

I have fixed the example and made it clearer how the percentile values are calculated (so they can be debugged by the skeptical developers we should be).

I really appreciate you finding this and the comment. It makes the example better for everyone!

AH December 11, 2019 at 8:35 pm # REPLY

np.quantile(q) = np.percentile(p) for q=p/100

Dawit August 24, 2017 at 7:41 am # REPLY

I love the piece of cake notes about ML.

Jason Brownlee August 24, 2017 at 4:24 pm # REPLY

Thanks Dawit.

Malcolm August 27, 2017 at 6:31 am # REPLY

Hi Jason, thanks for this post. Is the concept of fitting your classifier using a training set and calculating it's accuracy on unseen data (a holdout set) now outdated? It seems to me that CV is testing how good your algorithm (with optimized parameters) is at building a model, in that the model is re-fitted in each fold prior to testing it on the held-out fold. It doesn't seem to predict how a trained model will work once it is put into production and starts having to classify completely new instances. For that, I can't see any option other than training on the full training set and testing against the hold out set – am I missing something ?

Jason Brownlee August 28, 2017 at 6:45 am # REPLY

Cross Validation (CV) is doing the same thing, just repeated 10 times, for 10-fold CV.

CV and train/test splits are both resampling methods intended to estimate the skill of the model on unseen data. Perhaps this post will clear things up for you:
https://machinelearningmastery.com/train-final-machine-learning-model/

On many problems where we have millions of examples (e.g. deep learning) we often do not have the resources for CV.

Malcolm August 28, 2017 at 8:50 pm # REPLY

Thanks for the link – that clarifies a lot of things ! In summary, is it correct to say where you have the processing power and time to perform it, k-fold validation is always going to be the superior option as it provides means, standard deviations and confidence intervals ?

Jason Brownlee August 29, 2017 at 5:04 pm # REPLY

CV will give a less biased estimate of model skill, in general. This is the key objective of the method.

I often use a bootstrap to then present the final confidence interval for the chosen configuration.

Rob February 15, 2018 at 2:55 am # REPLY

Hi, thanks for this article. I'm after calculating confidence intervals for sensitivity and specificity. I've used your code, but changed the prep step to use the 'train_test_split' (with a random seed), to create samples of the data. Then, once per iteration, I've fitted a model and made predictions, then created a confusion matrix, and then worked out the sensitivity and specificity. Do you think that sounds reasonable? The results look sensible.

Thanks.

Jason Brownlee February 15, 2018 at 8:48 am # REPLY

I think the bootstrap approach would work for other accuracy related measures.

Ian March 28, 2018 at 10:28 pm # REPLY

Nice post (:

Might be useful to mention the bootsrapped package in case people are feeling lazy 😉

https://pypi.python.org/pypi/bootstrapped/0.0.1

Jason Brownlee March 29, 2018 at 6:35 am # REPLY

Thanks for the link Ian.

Sabrina April 17, 2018 at 8:11 am # REPLY

Great post Jason. I was wondering if you think it might be feasible to use bootstrap method for neural nets to estimate confidence level?

Jason Brownlee April 17, 2018 at 2:47 pm # REPLY

Absolutely!

Vladislav Gladkikh May 25, 2018 at 5:48 pm # REPLY

How can I estimate a 95% likelihood of classification accuracy for a particular unseen data point?

Suppose, I have three new data points x1, x2, x3. How can I estimate that my method has a 95% likelihood of classification accuracy between 80% and 85% for predicting, say, x2, but between 60% and 65% for predicting x1, and between 90% and 95% for predicting x3?

Jason Brownlee May 26, 2018 at 5:50 am # REPLY

It sounds like you might be asking about a prediction interval instead of a confidence interval.

I cover this in my book on stats and I have a post on the topic scheduled for soon.

Until then see this article:
https://en.wikipedia.org/wiki/Prediction_interval

Vladislav Gladkikh May 26, 2018 at 12:05 pm # REPLY

Thanks for the link!

Anthony The Koala June 3, 2018 at 11:30 pm # REPLY

Dear Dr Jason,
While I understand the general concept presented above, I am lost on one fine detail on lines 21 and 23. It is about selection array [:,:-1] and [:,-1].

```
   model = DecisionTreeClassifier()
21 model.fit(train[:,:-1], train[:,-1])
   # evaluate model
23 predictions = model.predict(test[:,:-1])
```

It is about

```
train[:,:-1]
```

and

```
train[:,-1]
```

In other words, when there is an array, what is the difference between the selection methods.

```
[:,:-1]
```

and

```
[:,-1]
```

Thank you,
Anthony of Sydney

Jason Brownlee June 4, 2018 at 6:29 am # REPLY

Yes, we are selecting columns.

You can learn more about slicing arrays here:
https://machinelearningmastery.com/index-slice-reshape-numpy-arrays-machine-learning-python/

Anthony The Koala June 6, 2018 at 10:57 pm # REPLY

Dear Dr Jason,
Thank you for that reference. I admit I was having difficulty then. Today I understand.
Regards
Anthony of Sydney

Jason Brownlee June 7, 2018 at 6:27 am # REPLY

Glad to hear it.

VK July 31, 2018 at 1:24 am # REPLY

Hello,
can you please point to a reference for the expressions in line 32 and 34 of the complete example. frame?

p = ((1.0-alpha)/2.0) * 100
p = (alpha+((1.0-alpha)/2.0)) * 100

Thanks

Jason Brownlee July 31, 2018 at 6:09 am # REPLY

Perhaps see the references at the end of the post.

vk July 31, 2018 at 2:56 am # REPLY

Please ignore my previous comment. I just observed that alpha was set to .95.
They now make sense.

Jason Brownlee July 31, 2018 at 6:11 am # REPLY

No problem.

Santiago September 15, 2018 at 2:49 am # REPLY

Hi! Thanks very much for your post!
However I don’t understand your confidence intervals. If you wanted a 95% confidence interval shouldn’t you be setting your alpha to 0.05?
Cheers,
Santiago

Jason Brownlee September 15, 2018 at 6:17 am # REPLY

Perhaps re-read the post. Remember that we want both sides, e.g. 2.5% gap on both sides of the mean.

Perhaps this will help:
https://machinelearningmastery.com/confidence-intervals-for-machine-learning/

khanh ha December 12, 2018 at 1:35 am # REPLY

it’s very intuitive. Thanks for writing it

Jason Brownlee December 12, 2018 at 5:55 am # REPLY

Thanks.

Chris December 23, 2018 at 4:56 pm # REPLY

Jason,

Thank you for writing this article!!

During my PhD in Physics, we had to fit unwieldy (highly) nonlinear functions to real data and it was essential for us (as “budding” scientists) to at least attempt to report on both the standard and systematic errors of our results.

While learning about and using propagation of error methods throughout school, it surprised me to learn so late in my studies that the bootstrap method was a (kosher) go-to method for (a simulated) estimate of standard error of parameters from such complicated functions.

When nature's constituent parts at various scales appear to often exhibit non-linear interaction dynamics, one would think our engineer and science teaching forefathers (e.g. for engineers, physicists, chemists, biologists, etc.) would have emphasized merit and further study of this method, particularly in an era now heavily (re)focused more on phenomenological studies and less so on pure analytical theories.

To end on a high note, I'm still thankful for the advances in DS and ML for making such phenomenological studies, and basic predictive analytics accessible to the masses, and namely a laymen like me.

Thank you again for another great article!!

Jason Brownlee December 24, 2018 at 5:26 am # REPLY

Thanks for sharing, I could not agree with you more.

In fact, more simulation/monte carlo methods should be taught in general.

Michael June 15, 2019 at 8:58 am # REPLY

Hi Jason,

Great article on this. It's helped me immensely. I've been trying to apply this to getting CIs for AUC and after visualizing the sampling distribution it looks like it is a truncated normal distribution. Have you worked with these type of distributions when calculating CIs? Any advice would be much appreciated.

Jason Brownlee June 16, 2019 at 7:08 am # REPLY

I have not sorry.

Perhaps try repeating the experiment 100+ times and then review the distribution, e.g. histogram and perhaps a statistical test.

Manoj June 26, 2019 at 1:18 pm # REPLY

I find this article very useful. I have been working on a data set where I am using

RandomForestRegressor to predict 2 outputs (multi output) and there are 17 predictors. I made the model and prediction but I want prediction intervals from them, I read papers jackknife and infinitesimal jackknife and quantile regression, and found out about foresci package which makes interval based on jackknife paper, but none of them explained the multi output case like how to find the interval for multivariate case. It will be helpful if you could give me some advice.

Jason Brownlee June 26, 2019 at 2:35 pm # REPLY

Thanks for the suggestion, I may cover the topic in the future.

Manuel Zanutelli July 10, 2019 at 2:38 pm # REPLY

Hi Jason, thank you very much, I love this article, is super helpful and clear.

I have one question: I'm currently interested in just the confidence intervals, I've noticed that varying the size of the sample gives me different intervals. I came across the Seaborn package that has a bootstrap function that takes samples that are 100% the size of the dataset [1].

Is it Ok to sample with a size equal to 100% of the dataset? Why do the intervals vary?

Thanks for any guide you could provide me.

[1]
https://github.com/mwaskom/seaborn/blob/b9551aff1e2b020542a5fb610fec468b69b87c6e/seaborn/algorithms.py#L86

Jason Brownlee July 11, 2019 at 9:43 am # REPLY

Yes, I believe using bootstrap samples that are 100% is a standard practice.

Mariia Havrylovych July 19, 2019 at 5:46 am # REPLY

Hello!
Really helpful article, thanks a lot!
I want to ask: if we made not bootstrap, but some shufflesplit CV. And have a list of some metric values after it. Can we test the normality of this values (with shapiro test for example) and if we don't reject null

hypothesis, build a confidence interval for mean over this values with unknown standard deviation (x.mean +- t*s/sqrt(n))? I was trying to do something like that, but I don't really confident in correctness of this computations.
Thanks for any advice you could give me!

Jason Brownlee July 19, 2019 at 9:26 am # REPLY

Might be a bit messy, e.g. k mean values for m repeats with different splits.

Not sure about the implications sorry. Perhaps check in with a statistician?

Prerna Singhal October 20, 2019 at 1:07 am # REPLY

Very nice article! Can you please suggest how can we use Bootstrapping with logistic regression? I want to change the C value in every iteration. How this can be done?

Thank you!

Jason Brownlee October 20, 2019 at 6:19 am # REPLY

Sounds straightforward, what problem are you having exactly?

Prerna Singhal October 20, 2019 at 11:53 am # REPLY

I am not sure how can I send series of C for logistic regression and which value will give me best solution. Can you provide me the syntax of doing this? Currently I am using LogReg.fit to do the logistic regression but not sure how can I allocate different C. Also, if C can be optimized?

Thank you!

Jason Brownlee October 21, 2019 at 6:13 am # REPLY

Perhaps try a range of values for C on a log scale?

Luis November 26, 2019 at 11:04 am # REPLY

Hi, is there are reason why you don't use replacement? And what is the ideal percentage of your original dataset that we should keep on each bootstrap? Is it just random?
Thanks

Jason Brownlee November 26, 2019 at 1:31 pm # REPLY

The bootstrap does use replacement.

The bootstrap size should match the original dataset size, that is a good heuristic.

Luis November 28, 2019 at 11:29 am # REPLY

Thanks
And should we care about stratification when resampling?

Jason Brownlee November 28, 2019 at 1:33 pm # REPLY

Yes, probably.

Beau Hilton January 5, 2020 at 9:10 pm # REPLY

Thanks Jason! Great article, as always.

The mlxtend Python package has a nice implementation of the bootstrap, as well as the .632 and .632+ methods (bootstrap_point632.py). My only complaint is that it only takes one scoring function at a time as of now, but I'm working on a PR to make it more efficient for multiple scores. 🙂

Jason Brownlee January 6, 2020 at 7:11 am # REPLY

Thanks for sharing.

shafi March 29, 2020 at 6:49 am #

REPLY

Hello Dr. Jason Bronwlee,

It is very nice stuff. I want to calculate the Confidence intervals for XGBoost of H2o. I do not understand a few steps. I am applying data to XGBoost by splitting into train, valid and test assuming time series case. How can we put it into iterations? I would appreciate your guidance.

import numpy
from pandas import read_csv
from sklearn.utils import resample
from sklearn.metrics import accuracy_score
from matplotlib import pyplot
from xgboost import XGBRegressor
# load dataset
#data = read_csv('pima-indians-diabetes.data.csv', header=None)
#values = data.values
# configure bootstrap
n_iterations = 100
n_size = int(len(hf_train) * 0.50)
# run bootstrap
stats = list()
for i in range(n_iterations):
# prepare train and test sets
#train = resample(values, n_samples=n_size)
#test = numpy.array([x for x in values if x.tolist() not in train.tolist()])
# fit model
import h2o
from h2o.estimators.xgboost import H2OXGBoostEstimator
h2o.init()

param = {
"ntrees" : 10000
, "nfolds" : 0
,"stopping_rounds" : 40
, "max_depth" : 25
,"stopping_tolerance" : 0.005
, "learn_rate" : 0.05

```
, “eta” : 0.3

, “sample_rate” : 0.8
, “subsample” : 1
, “col_sample_rate” : 1
, “col_sample_rate_per_tree” : 1
, “colsample_bylevel” : 1
, “min_rows” : 0.06
,”min_child_weight”: 1
, “sample_type” : “uniform”
, “normalize_type” : “tree”
,”nthread” : -1
, “max_bins” : 256
, “max_leaves” : 0
, “seed”: 1601
, “score_tree_interval”: 0
,”reg_lambda” : 0.1
,”reg_alpha” : 1/1
,”grow_policy” : “depthwise”
, “booster” : “gbtree”

}
from h2o.estimators import H2OXGBoostEstimator
model = H2OXGBoostEstimator(**param)
model.train(x = X, y = y, training_frame = hf_train, validation_frame = hf_valid)
# evaluate model
predictions = model.predict(hf_test)
score = model.model_performance(test_data= hf_test)
print(score)
stats.append(score)
# plot scores
pyplot.hist(stats)
pyplot.show()
# confidence intervals
alpha = 0.95
p = ((1.0-alpha)/2.0) * 100
lower = max(0.0, numpy.percentile(stats, p))
p = (alpha+((1.0-alpha)/2.0)) * 100
upper = min(1.0, numpy.percentile(stats, p))
print(‘%.1f confidence interval %.1f%% and %.1f%%’ % (alpha*100, lower*100, upper*100))
```

Jason Brownlee March 30, 2020 at 5:26 am # REPLY

This is a common question that I answer here:
https://machinelearningmastery.com/faq/single-faq/can-you-read-review-or-debug-my-code

shafi March 30, 2020 at 7:02 am # REPLY

Thanks: That,s make sense.

Jason Brownlee March 30, 2020 at 7:53 am # REPLY

You're welcome.

shafi March 31, 2020 at 2:06 am # REPLY

Hello Dr. Jason,

I followed the step given the link above by "cutting the problem" into a small level. I have now been able to apply to the XGBoost H2o. I need still your guidance for a few things.

first: I want to divide the data into train,valid and test. The models are trained using the training data and scored using the validation data and obtain a final score of the model using test data. How can we change it.?

train = resample(values, n_samples=n_size)

test = numpy.array([x for x in values if x.tolist() not in train.tolist()])

Second: I do not understand why lower and upper confidence interval are same i.e.(4.4% and 4.4%). I am doing it for regression .For score I am using:

score = model.rmse()

Jason Brownlee March 31, 2020 at 8:14 am # REPLY

You can use this function to split data into subsets:

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

shafi April 1, 2020 at 5:23 am # REPLY

Thank a lot: I have now posted my second question at Stackoverflow with reproducible codes.

https://stackoverflow.com/questions/60956655/bootstrap-confidence-intervals-for-xgboost-h2o-regression-python

Jason Brownlee April 1, 2020 at 5:54 am # REPLY

Well done.

shafi April 2, 2020 at 5:18 am # REPLY

Hello Dr Jason: I am still looking for the answer my second question (in comments 31March).

In your example. When we print(score), we will the score equal to the number of n_iterations.This is how we can plot the histogram.

score = accuracy_score(test[:,-1], predictions)

print(score)

IN my case: XGBoost h2o. (whole codes can be seen in Stackoverflow). I get the single value for the score when we do n_iterations. This is the problem. How can we solve it?

predictions = model.predict(test_hf1)
score = model.rmse()
print(score)

Jason Brownlee April 2, 2020 at 6:04 am # REPLY

Sorry, I don't know about the h20 api, not sure I can give you advice. Perhaps contact their support?

shafi April 2, 2020 at 6:08 am # REPLY

Thanks a lot for the always prompt response. I have contacted with H2O. I am also working on it.

Jason Brownlee April 2, 2020 at 6:40 am # REPLY

I'm happy to hear that, good luck!

shafi April 2, 2020 at 9:58 am # REPLY

I checked it with using LightGBM library it seem to have the same problem for the regression as I am getting with XGBoost H2o. It seems to me the codes does not work for the regression. It works well with the XGBoost classifier. Also it means that the problem is not pertain to specific API such H2o rather to applying to regression or classification. It works fine with classification.

I shall be very thankful to you if you can check it for regression.

Jason Brownlee April 2, 2020 at 1:30 pm # REPLY

Yes, the code example calculates accuracy, you must change this for regression to an error score like MSE.

shafi April 2, 2020 at 2:36 pm # REPLY

Probably I could not explain my question. I did change to MSE/RMSE but the problem is not solved. It works perfectly only for classification case.

score = np.sqrt(mean_squared_error(yt, y_pred))
print(score)
stats.append(score)
# plot scores
pyplot.hist(stats)
pyplot.show()

# confidence intervals
alpha = 0.95
p = ((1.0-alpha)/2.0) * 100
lower = max(0.0, numpy.percentile(stats, p))
p = (alpha+((1.0-alpha)/2.0)) * 100
upper = min(1.0, numpy.percentile(stats, p))
print('%.1f confidence interval %.1f%% and %.1f%%' % (alpha*100, lower*100, upper*100))

Jason Brownlee April 3, 2020 at 6:51 am # REPLY

Sorry to hear that, why not? What is the problem you are having precisely?

shafi April 3, 2020 at 1:45 am # REPLY

I have now posted the issue at https://github.com/dmlc/xgboost/issues/5475

Jason Brownlee April 3, 2020 at 6:56 am # REPLY

Try plotting the data to confirm there is a distribution. Perhaps there is not.

If there is, try changing the number of bins in the histogram plot.

shafi April 3, 2020 at 7:19 am # REPLY

Yes, I tried that too but did not work:

import seaborn as sns; sns.set()

sns.distplot(stats, hist=True, kde=False, bins=int(10), color = 'blue', hist_kws={'edgecolor':'black'})

Jason Brownlee April 3, 2020 at 7:56 am # REPLY

Print the predicted values and look at them, ensure that there is some spread. Perhaps there isn't in which case everything is working correctly.

shafi April 3, 2020 at 8:39 am # REPLY

Thanks a lot once again!! I have now plotted the prediction, the spread looks fine. It can be found here : https://github.com/dmlc/xgboost/issues/5475

Jason Brownlee April 3, 2020 at 9:40 am # REPLY

Nice.

How about print the values of mean error from each run?

shafi April 3, 2020 at 9:02 am # REPLY

The problem is related to (I think) when we run iteration for bootstrapping and using print(score) we can get as many values for score as number of iteration for classification however, we do not get for regression rather we get single value for score during the iteration. But I do not know how to fix it.

Jason Brownlee April 3, 2020 at 9:41 am # REPLY

It may not be a bug if the mean performance across runs does not change. It means the problem is trivial and the model is very stable.

shafi April 4, 2020 at 12:14 am # REPLY

I can not get the mean error from run when use print(error).

error = np.mean(predictions-y_test)

Jason Brownlee April 4, 2020 at 6:19 am # REPLY

Use the sklearn library to calculate it:
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html#sklearn.metrics.mean_squared_error

shafi April 4, 2020 at 7:29 am # REPLY

Thanks! : I did already with different ways including sklearn

score = mean_squared_error(y_test,y_pred, squared= True)

with this functrion.

```
def rmse_calculator(predicted, actual):

    assert len(predicted) == len(actual)
    return np.sqrt(
                np.mean(
                    np.power(predicted- actual, 2)))
    score = rmse_calculator(y_test , predictions)
```

But nothing work. Is there any way we can incorporate rmse into loop like rmse[i] in your codes. Though I am trying myself too.

I did in R with XGBoost H2o. it works fine on small number of iteration. But I get problem with large iteration. I posted question at Stackoverflow (https://stackoverflow.com/questions/61019185/boostrapped-histogram-of-rmse-for-xgboost-regression-h2o-r-how-to-make-it-more).

Jason Brownlee April 4, 2020 at 9:00 am # REPLY

Yes.

Sorry, I don't have the capacity to prepare custom code for you.

Carlos Alexandre April 25, 2020 at 10:03 pm # REPLY

Dr Jason, I have a question. If I happen to use a Random Forest Regressor, for instance, which already contains the bootstrap as a default parameter, would it be the case to not manually apply a bootstrap sampling, right? Thanks !

Jason Brownlee April 26, 2020 at 6:12 am # REPLY

No, I don't think so. They are different.

The model uses the bootstrap to ensemble trees.

You can use the bootstrap to get an estimate of model performance.

Maria May 8, 2020 at 3:13 am # REPLY

Hello! I found this post very useful, as many others:) I just have a quick question.

I am wondering at what stage in your pipeline you perform hyperparameter optimization (using 10 fold CV on the training set).

In your example, you fit a simple Decision Tree classifier on the whole training data at each bootstrap iteration (with default hyperparameters I suppose).

But what if I wanted to do report the performance of the best optimized model (e.g.SVM) with the confidence intervals as you describe here with this bootstrap procedure?

Would I need to repeat hyperparameter tuning to find the best C and gamma parameters (using 10 fold CV) on the training set at each bootstrapping iteration?
Or instead, would I do it only once as a preliminary step during the search for the best model before the bootstrapping resampling? and then use the best hyperparameters found, to fit my model iteratively during the bootstrapping in all iterations?

Thanks!

Jason Brownlee May 8, 2020 at 6:40 am # REPLY

You're welcome.

Within the resampling process, e.g. so-called nested cross-validation or nested bootstrap.

Or select model+config first, then use the bootstrap on the final model to estimate performance.

Anisha May 15, 2020 at 9:06 pm # REPLY

hi Jason,

Thank you for the post.. It was very useful and explanatory. I get the following error while trying to execute the code. Can you please direct me how to overcome this?

AttributeError: 'DataFrame' object has no attribute 'tolist'

Jason Brownlee May 16, 2020 at 6:10 am # REPLY

You're welcome.

Sorry, to hear that, perhaps this will help:
https://machinelearningmastery.com/faq/single-faq/why-does-the-code-in-the-tutorial-not-work-for-me

Anisha May 15, 2020 at 9:30 pm # REPLY

Also, pandas version is 0.25.1. No upgrade required. What is the alternative to the tolist()

Jason Brownlee May 16, 2020 at 6:11 am # REPLY

Update to pandas 1.0.3 or better.

Himanshu May 29, 2020 at 6:45 pm # REPLY

Hi Jason,
Great stuff 🙂

I had a doubt about the test sizes in such bootstrapping-with-replacement methods. The approach pretty much means we are going to have different test set sizes for the same size of training data in every iteration. For example, I find that in my case, test size varies between 444 to 486.

Is it acceptable in general to compare/make use of accuracy scores (or other metrics) which have been arrived at using different test set sizes?

Or is it something we accept to live with, only in case of bootstrapping-with-replacement approach, since no better options are available?

Thanks

Jason Brownlee May 30, 2020 at 5:55 am # REPLY

Typically the bootstrap sample size is configured to be the same size as the raw dataset.

Kaike July 22, 2020 at 7:41 am # REPLY

Hello Jason, I have a question:
What I need to change if my metrics distribution is not a gaussian? I.E. the metrics that I get for each bootstrap created a list of metrics, let's say 3000 metrics for each bootstrap dataset, but that list not follow a gaussian distribution (fail in shapiro test), how can I create a confidence interval?

Best regards from Brazil,

Jason Brownlee July 22, 2020 at 1:40 pm # REPLY

You can use the bootstrap directly, it does not assume a distribution.

Kaike July 23, 2020 at 6:42 am # REPLY

Thanks Jason, but I don't understand why isn't necessary to assume a normal distribution, can you explain more or point out some links to read more about it? I was reading about non-parametric CI and they have a robust mathematical background about the distribution, why for this specific case is not necessary.

Jason Brownlee July 23, 2020 at 2:36 pm # REPLY

Yes, the whole idea of using the boostrap is because it is a nonparametric method that does not assume a distribution.

If you need further information, see the "Further Reading" section.

Kaike July 31, 2020 at 1:20 am # REPLY

Jason, one last question: Can I evaluate a BCI in the test set (doing resampling over this set) to create a 95%CI for my prediction?
I make this question because I developed an HP optimization before the BCI.

Jason Brownlee July 31, 2020 at 6:21 am # REPLY

It is calculated on one dataset with in and out samples each run. There is no train and test sets per se.

Kaike July 31, 2020 at 8:11 am # REPLY

That is my problem Jason. I will briefly explain my pipeline:
– Hold out TRAIN/TEST split
– Hyper parameter optimization CV 5-Fold in TRAIN split
– Generate BCI intervals based in my the model's prediction for TEST split. I.E. I generate N bootstrap sets from test, calculate a metric and then in the calculate the BCI.

I was reading more about and this could be a prediction interval problem, but still I don't know a good method to get it. I was reading other post of yours:
https://machinelearningmastery.com/report-classifier-performance-confidence-intervals/
And I don't know if it is suitable for my problem. I don't want to train all my models again… So somehow it is possible to generate a CI based in test set prediction?

Jason Brownlee July 31, 2020 at 1:40 pm # REPLY

Thanks for sharing.

If it works for you, that is great, it is not the approach I would take as I stated earlier.

gonconist August 27, 2020 at 12:16 am # REPLY

Hello,

In a multi-classfication problem I'm interested in knowing if the performance of my model (micro f1-score) is statistically significant over the baseline. Thus, I am looking for confidence intervals for the f1-score.

I want to use bootstrap, but since my model is large enough, I cannot afford to re-train and compute the f1-score each time.

What I have at my disposal is a prediction-file, that contains all predictions of my model for each test-example.

Does it make sense to apply bootstrap directly by sampling with replacement from these predictions and calculating f1-scores? Essentially, I will be leaving out some test-samples, calculating the f1-score on the subset and repeating until I have enough statistics to compute confidence intervals.

If not, do you have any suggestion on how I can proceed with this task?

Thank you in advance!

Jason Brownlee August 27, 2020 at 6:17 am # REPLY

Off the cuff, no. You would be sampling predictions, not sampling the capability of the model.

Drop the confidence interval and use a different statistical test, e.g.:
https://machinelearningmastery.com/mcnemars-test-for-machine-learning/

gonconist September 2, 2020 at 7:16 pm # REPLY

Thank you for your quick reply!

I've tried the McNemar's test although I'm still a bit confused. Is this test only applicable in a binary classification scenario? To my understanding the 2×2 contingency table is built using the predictions of both models (regardless of how many classes we have) and the ground truth.

I have found on the internet that Stuart-Maxwell (or generalised McNemar's) test can be used on a multi-classification setting, although I didn't find it implemented in any python library (hence I'm concerned about it's validity).

Now, I know that model A has a higher f1-score than model B and McNemar's test yields a p-value which is less than the critical value, e.g. p < 0.05.

Can I conclude that the difference between the performance of the two models is statistically significant, i.e. model A is better than model B?

Jason Brownlee September 3, 2020 at 6:06 am # REPLY

Yes, if the result is better and significant, you can start to make claims.

You might also want to look at 2×5 cv:
https://machinelearningmastery.com/hypothesis-test-for-comparing-machine-learning-algorithms/

Nadaa November 3, 2020 at 6:47 am # REPLY

Hi Jason!

Thank you for this excellent post. We are trying to create a confidence interval for AUC. We have a dataset of 120k plus observations. Could we just resample the dataset with 80% of the data and run a train/test split instead of doing bootstrap with replacement? Or could we just run a train/test split on all the data 1000s of times? I am not clear why, from a stats theory standpoint, we should do bootstrap with replacement for the training set. After all, this results in a training set with duplicates whereas the method below does not.

auc_list = list()
# Begin bootstrap sampling
for i in range(num_iter):
df_resampled = df.sample(frac=0.8, replace=False)
y, X = df_resampled.pop('DEATH_90'), df_resampled
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
clf.fit(X_train, y_train)
y_prob = clf.predict_proba(X_test)[:,1]
auc_score = roc_auc_score(y_test, y_prob)
auc_list.append(auc_score)
# Calculate lower & upper confidence interval
lower_ci = np.percentile(auc_list, ((1.0-alpha)/2.0) * 100)

upper_ci = np.percentile(auc_list, (alpha+((1.0-alpha)/2.0)) * 100)

Thanks for your help!

Jason Brownlee November 3, 2020 at 6:57 am # REPLY

Yes, perhaps try a bootstrap as a first step.

Sorry, I don't have the capacity to debug your code, these tips may help:
https://machinelearningmastery.com/faq/single-faq/can-you-read-review-or-debug-my-code

Nadaa November 3, 2020 at 9:00 am # REPLY

Thanks, Jason. Perhaps, I can rephrase the question without using any code. I have a dataset with 120k rows. Can I just run a train/test split 10,000 times without a random seed and with a data shuffle, so that the split is different every time, instead of using the bootstrap sample with replacement method? Using my method, there would be no duplicates in the training set and both the train and test sets would be the same size.

Nadaa

Jason Brownlee November 3, 2020 at 10:09 am # REPLY

Sure.

The bootstrap simulates multiple samples when you don't have much data.

Sam McDonald December 3, 2020 at 2:28 am # REPLY

Why use min and max when calculating the bounds?

Jason Brownlee December 3, 2020 at 8:20 am # REPLY

The min and max are the very definition of "bounds".

Ali January 22, 2021 at 3:49 am # REPLY

Hi.thanks for good post.
I have a question.
How i can apply this method in deep learning binary classfication using keras?
Please help me.
Thank you

Jason Brownlee January 22, 2021 at 7:25 am # REPLY

Yes, sure.

SaHaR January 26, 2021 at 7:04 am # REPLY

Hi Jason,

This article was very useful for me as always. I have a question. when I use resample method it produces some NaNs. How can I handle this problem? My dataset do not have NaNs

Jason Brownlee January 26, 2021 at 9:31 am # REPLY

Thanks.

Sorry, I don’t know why you are getting NaNs.

Perhaps try posting your code and error to stackoverflow.com

Alejandra March 26, 2021 at 8:27 am # REPLY

Hi Jason,
really good explanation. I was wondering how could I find a prediction interval for a multivariable nonlinear model. I want to show a plot of predicted data with the corresponding prediction interval bands. Should I use the percentile function in the predicted data for each X?.

Jason Brownlee March 29, 2021 at 5:45 am # REPLY

Thanks.

Perhaps this would be a good starting point:
https://machinelearningmastery.com/prediction-intervals-for-machine-learning/

And this:
https://machinelearningmastery.com/prediction-intervals-for-deep-learning-neural-networks/

Chandril August 5, 2021 at 12:32 am # REPLY

Hi, thanks for this article. I am an absolute beginner and have a few questions. It would be a great help to get your advice and ideas.

1. You mentioned "95.0 confidence interval 64.4% and 73.0%" – is it the same as saying 95% CI [64.4, 73.0]?

2. The "bootstrapped Python library" (https://pypi.org/project/bootstrapped/) allows you to build confidence intervals from data. It does not considers or asks for including the ML model. But in your approach, you included DecisionTreeClassifier. I am curious to know what are the differences between these two approaches of finding the CI? Is your approach finding something different?

3. It seems to me that you are choosing percentile for the confidence interval. But the bootstrap python lbirary authors are using Mean or SD for the confidence interval. Are these different options available? [honestly I am just starting to learn C.I.]

4. You said "A 95% confidence interval is used, so the values at the 2.5 and 97.5 percentiles are selected." – could you please explain what exactly this means? From where you are getting these values of 2.5 and 97.5? and more importantly, I could not find these values on your codes. So how or where were these values used?

Jason Brownlee August 5, 2021 at 5:23 am # REPLY

1. yes I think so.

2. I've not seen that lib. Here we are determining the CI for a model, that is why the model is included in the code.

3. Yes. Choose the representation you require.

4. Yes, see further reading section.

Chandril August 5, 2021 at 2:41 am # REPLY

The following is WRONG, right?

"Area Under ROC Curve performance of the model X is 0.59, the 95% confidence interval calculated using bootstrapped re-sampling is [0.92-0.96]."

I used your codes on my data and this is what I got. What I might be doing wrong?

Jason Brownlee August 5, 2021 at 5:25 am # REPLY

Perhaps this will help:
https://machinelearningmastery.com/faq/single-faq/why-does-the-code-in-the-tutorial-not-work-for-me

Or this:
https://machinelearningmastery.com/faq/single-faq/why-do-i-get-different-results-each-time-i-run-the-code

Stepan August 25, 2021 at 3:03 am # REPLY

Not sure if this has been mentioned before, but a cleaner and a faster way would be to use a bagging regressor from sklearn. It can do the resampling internally for you in a more optimized fashion than with a for loop.
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingRegressor.html

Adrian Tam August 25, 2021 at 6:16 am # REPLY

Thanks for sharing.

Leave a Reply

Name (required)

Email (will not be published) (required)

Website

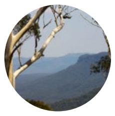
SUBMIT COMMENT

Welcome!
I'm *Jason Brownlee* PhD
and I **help developers** get results with **machine learning**.
Read more

Never miss a tutorial:

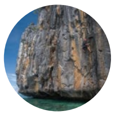
Picked for you:

Statistics for Machine Learning (7-Day Mini-Course)


A Gentle Introduction to k-fold Cross-Validation

How to Calculate Bootstrap Confidence Intervals For Machine Learning Results in Python

Statistical Significance Tests for Comparing Machine Learning Algorithms


A Gentle Introduction to Normality Tests in Python