Cost-Sensitive Logistic Regression for Imbalanced Classification

by **Jason Brownlee** on January 27, 2020 in **Imbalanced Classification**

Tweet Share Share

Last Updated on February 1, 2020

Logistic regression does not support imbalanced classification directly.

Instead, the training algorithm used to fit the logistic regression model must be modified to take the skewed distribution into account. This can be achieved by specifying a class weighting configuration that is used to influence the amount that logistic regression coefficients are updated during training.

The weighting can penalize the model less for errors made on examples from the majority class and penalize the model more for errors made on examples from the minority class. The result is a version of logistic regression that performs better on imbalanced classification tasks, generally referred to as cost-sensitive or weighted logistic regression.

In this tutorial, you will discover cost-sensitive logistic regression for imbalanced classification.

After completing this tutorial, you will know:

- How standard logistic regression does not suppor
- How logistic regression can be modified to weight coefficients.
- How to configure class weight for logistic regressi configurations.

Discover SMOTE, one-class classification, cost-sensit my new book, with 30 step-by-step tutorials and full Pytho

Let's get started.

Start Machine Learning

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

Email Address

START MY EMAIL COURSE

- **Update Feb/2020**: Fixed typo in weight calculation.

Cost-Sensitive Logistic Regression for Imbalanced Classification
Photo by Naval S, some rights reserved.

Tutorial Overview

This tutorial is divided into five parts; they are:

1. Imbalanced Classification Dataset
2. Logistic Regression for Imbalanced Classification
3. Weighted Logistic Regression With Scikit-Learn
4. Grid Search Weighted Logistic Regression

Imbalanced Classification Data

Before we dive into the modification of logistic regress an imbalanced classification dataset.

We can use the make_classification() function to defin dataset. We will generate 10,000 examples with an ap

```
1 ...
2 # define dataset
```


Start Machine Learning

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

Email Address

START MY EMAIL COURSE

```
X, y = make_classification(n_samples=10000, n_features=2, n_redundant=0,
    n_clusters_per_class=1, weights=[0.99], flip_y=0, random_state=2)
```

Once generated, we can summarize the class distribution to confirm that the dataset was created as we expected.

```
...
# summarize class distribution
counter = Counter(y)
print(counter)
```

Finally, we can create a scatter plot of the examples and color them by class label to help understand the challenge of classifying examples from this dataset.

```
...
# scatter plot of examples by class label
for label, _ in counter.items():
    row_ix = where(y == label)[0]
    pyplot.scatter(X[row_ix, 0], X[row_ix, 1], label=str(label))
pyplot.legend()
pyplot.show()
```

Tying this together, the complete example of generating the synthetic dataset and plotting the examples is listed below.

```
# Generate and plot a synthetic imbalanced classification dataset
from collections import Counter
from sklearn.datasets import make_classification
from matplotlib import pyplot
from numpy import where
# define dataset
X, y = make_classification(n_samples=10000, n_features=2, n_redundant=0,
    n_clusters_per_class=1, weights=[0.99], flip_y=0, random_state=2)
# summarize class distribution
counter = Counter(y)
print(counter)
# scatter plot of examples by class label
for label, _ in counter.items():
    row_ix = where(y == label)[0]
    pyplot.scatter(X[row_ix, 0], X[row_ix, 1], label=str(label))
pyplot.legend()
pyplot.show()
```

Running the example first creates the dataset and summarizes the class distribution.

We can see that the dataset has an approximate 1:10
examples in the majority class and 100 in the minority

```
Counter({0: 9900, 1: 100})
```

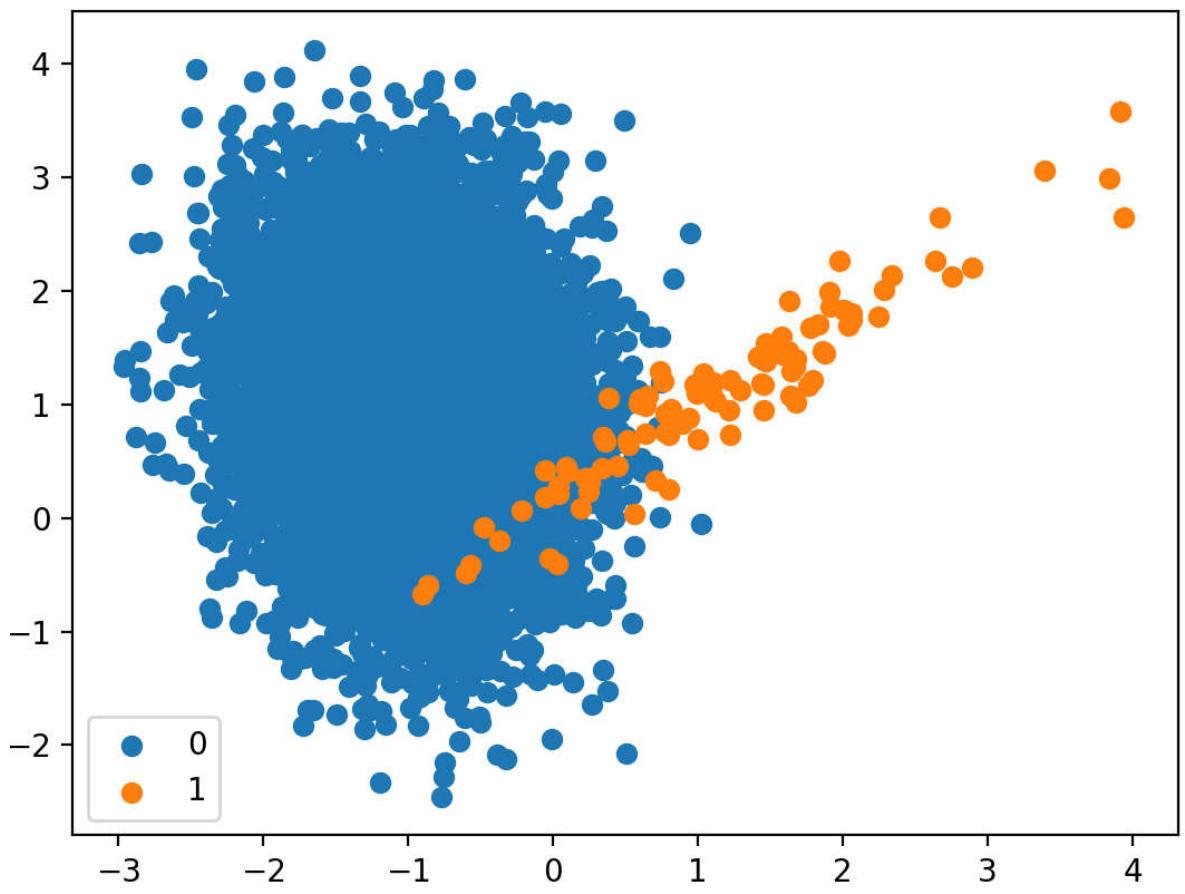
Next, a scatter plot of the dataset is created showing t
(blue) and a small number of examples for the minorit

Start Machine Learning

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

Email Address

START MY EMAIL COURSE



Scatter Plot of Binary Classification Dataset With 1 to 100 Class Imbalance

Next, we can fit a standard logistic regression model on the dataset.

We will use repeated cross-validation to evaluate the model, with three repeats of 10-fold cross-validation. The mode performance will be reported using the mean ROC area under curve (ROC AUC) averaged over repeats and all folds.

```
...
# define evaluation procedure
cv = RepeatedStratifiedKFold(n_splits=10, n_re
# evaluate model
scores = cross_val_score(model, X, y, scoring=
# summarize performance
print('Mean ROC AUC: %.3f' % mean(scores))
```

Tying this together, the complete example of evaluate
classification problem is listed below.

```
# fit a logistic regression model on an imbal
from numpy import mean
from sklearn.datasets import make_classificat
from sklearn.model_selection import cross_val
from sklearn.model_selection import RepeatedS
from sklearn.linear_model import LogisticRegr
```


Start Machine Learning

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

Email Address

START MY EMAIL COURSE

```
# generate dataset
X, y = make_classification(n_samples=10000, n_features=2, n_redundant=0,
    n_clusters_per_class=1, weights=[0.99], flip_y=0, random_state=2)
# define model
model = LogisticRegression(solver='lbfgs')
# define evaluation procedure
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate model
scores = cross_val_score(model, X, y, scoring='roc_auc', cv=cv, n_jobs=-1)
# summarize performance
print('Mean ROC AUC: %.3f' % mean(scores))
```

Running the example evaluates the standard logistic regression model on the imbalanced dataset and reports the mean ROC AUC.

We can see that the model has skill, achieving a ROC AUC above 0.5, in this case achieving a mean score of 0.985.

```
Mean ROC AUC: 0.985
```

This provides a baseline for comparison for any modifications performed to the standard logistic regression algorithm.

Want to Get Started With Imbalance Classification?

Take my free 7-day email crash course now (with sample code).

Click to sign-up and also get a free PDF Ebook version of the course.

Download Your FREE Mini-Course

Logistic Regression for Imbalanced Classification

Logistic regression is an effective model for binary classification tasks, although by default, it is not effective at imbalanced classification.

Logistic regression can be modified to be better suited

The coefficients of the logistic regression algorithm are
the negative log likelihood (loss) for the model on the

- minimize sum i to n -(log(yhat_i) * y_i + log(1 – yh

This involves the repeated use of the model to make p
coefficients in a direction that reduces the loss of the m

Start Machine Learning

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

Email Address

START MY EMAIL COURSE

The calculation of the loss for a given set of coefficients can be modified to take the class balance into account.

By default, the errors for each class may be considered to have the same weighting, say 1.0. These weightings can be adjusted based on the importance of each class.

- minimize sum i to n -(w0 * log(yhat_i) * y_i + w1 * log(1 – yhat_i) * (1 – y_i))

The weighting is applied to the loss so that smaller weight values result in a smaller error value, and in turn, less update to the model coefficients. A larger weight value results in a larger error calculation, and in turn, more update to the model coefficients.

- **Small Weight**: Less importance, less update to the model coefficients.
- **Large Weight**: More importance, more update to the model coefficients.

As such, the modified version of logistic regression is referred to as Weighted Logistic Regression, Class-Weighted Logistic Regression or Cost-Sensitive Logistic Regression.

The weightings are sometimes referred to as importance weightings.

Although straightforward to implement, the challenge of weighted logistic regression is the choice of the weighting to use for each class.

Weighted Logistic Regression with Scikit-Learn

The scikit-learn Python machine learning library provides an implementation of logistic regression that supports class weighting.

The LogisticRegression class provides the class_weight argument that can be specified as a model hyperparameter. The class_weight is a dictionary that defines each class label (e.g. 0 and 1) and the weighting to apply in the calculation of the negative log likelihood when fitting the model.

For example, a 1 to 1 weighting for each class 0 and 1 can be defined as follows:

```
...
# define model
weights = {0:1.0, 1:1.0}
model = LogisticRegression(solver='lbfgs', cla
```

The class weighing can be defined multiple ways; for

- **Domain expertise**, determined by talking to subj
- **Tuning**, determined by a hyperparameter search
- **Heuristic**, specified using a general best practice

A best practice for using the class weighting is to use
training dataset.

Start Machine Learning

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

Email Address

START MY EMAIL COURSE

For example, the class distribution of the test dataset is a 1:100 ratio for the minority class to the majority class. The inversion of this ratio could be used with 1 for the majority class and 100 for the minority class; for example:

```
...
# define model
weights = {0:1.0, 1:100.0}
model = LogisticRegression(solver='lbfgs', class_weight=weights)
```

We might also define the same ratio using fractions and achieve the same result; for example:

```
...
# define model
weights = {0:0.01, 1:1.0}
model = LogisticRegression(solver='lbfgs', class_weight=weights)
```

We can evaluate the logistic regression algorithm with a class weighting using the same evaluation procedure defined in the previous section.

We would expect that the class-weighted version of logistic regression to perform better than the standard version of logistic regression without any class weighting.

The complete example is listed below.

```
# weighted logistic regression model on an imbalanced classification dataset
from numpy import mean
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.linear_model import LogisticRegression
# generate dataset
X, y = make_classification(n_samples=10000, n_features=2, n_redundant=0,
    n_clusters_per_class=1, weights=[0.99], flip_y=0, random_state=2)
# define model
weights = {0:0.01, 1:1.0}
model = LogisticRegression(solver='lbfgs', class_weight=weights)
# define evaluation procedure
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate model
scores = cross_val_score(model, X, y, scoring='roc_auc', cv=cv, n_jobs=-1)
# summarize performance
print('Mean ROC AUC: %.3f' % mean(scores))
```

Running the example prepares the synthetic imbalanced classification dataset, then evaluates the class-weighted version of logistic regression using repeated

The mean ROC AUC score is reported, in this case sh of logistic regression, 0.989 as compared to 0.985.

```
Mean ROC AUC: 0.989
```

The scikit-learn library provides an implementation of

It is implemented via the compute_class_weight() func

- n_samples / (n_classes * n_samples_with_class)

Start Machine Learning

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

Email Address

START MY EMAIL COURSE

We can test this calculation manually on our dataset. For example, we have 10,000 examples in the dataset, 9900 in class 0, and 100 in class 1.

The weighting for class 0 is calculated as:

- weighting = n_samples / (n_classes * n_samples_with_class)
- weighting = 10000 / (2 * 9900)
- weighting = 10000 / 19800
- weighting = 0.05

The weighting for class 1 is calculated as:

- weighting = n_samples / (n_classes * n_samples_with_class)
- weighting = 10000 / (2 * 100)
- weighting = 10000 / 200
- weighting = 50

We can confirm these calculations by calling the *compute_class_weight()* function and specifying the *class_weight* as "*balanced*." For example:

```
# calculate heuristic class weighting
from sklearn.utils.class_weight import compute_class_weight
from sklearn.datasets import make_classification
# generate 2 class dataset
X, y = make_classification(n_samples=10000, n_features=2, n_redundant=0,
    n_clusters_per_class=1, weights=[0.99], flip_y=0, random_state=2)
# calculate class weighting
weighting = compute_class_weight('balanced', [0,1], y)
print(weighting)
```

Running the example, we can see that we can achieve a weighting of about 0.5 for class 0 and a weighting of 50 for class 1.

These values match our manual calculation.

```
[ 0.50505051 50. ]
```

The values also match our heuristic calculation above for inverting the ratio of the class distribution in the training dataset; for example:

- 0.5:50 == 1:100

We can use the default class balance directly with the ight argument to 'balanced.' For example:

```
...
# define model
model = LogisticRegression(solver='lbfgs', cla
```

The complete example is listed below.

```
# weighted logistic regression for class imba
```


Start Machine Learning

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

Email Address

START MY EMAIL COURSE

```
from numpy import mean
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.linear_model import LogisticRegression
# generate dataset
X, y = make_classification(n_samples=10000, n_features=2, n_redundant=0,
    n_clusters_per_class=1, weights=[0.99], flip_y=0, random_state=2)
# define model
model = LogisticRegression(solver='lbfgs', class_weight='balanced')
# define evaluation procedure
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate model
scores = cross_val_score(model, X, y, scoring='roc_auc', cv=cv, n_jobs=-1)
# summarize performance
print('Mean ROC AUC: %.3f' % mean(scores))
```

Running the example gives the same mean ROC AUC as we achieved by specifying the inverse class ratio manually.

```
Mean ROC AUC: 0.989
```

Grid Search Weighted Logistic Regression

Using a class weighting that is the inverse ratio of the training data is just a heuristic.

It is possible that better performance can be achieved with a different class weighting, and this too will depend on the choice of performance metric used to evaluate the model.

In this section, we will grid search a range of different class weightings for weighted logistic regression and discover which results in the best ROC AUC score.

We will try the following weightings for class 0 and 1:

- {0:100,1:1}
- {0:10,1:1}
- {0:1,1:1}
- {0:1,1:10}
- {0:1,1:100}

These can be defined as grid search parameters for th

```
...
# define grid
balance = [{0:100,1:1}, {0:10,1:1}, {0:1,1:1}, {0
param_grid = dict(class_weight=balance)
```

We can perform the grid search on these parameters el performance using ROC AUC:

```
...
# define evaluation procedure
cv = RepeatedStratifiedKFold(n_splits=10, n_re
# define grid search
grid = GridSearchCV(estimator=model, param_gri uc'
```


Start Machine Learning

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

Email Address

START MY EMAIL COURSE

Once executed, we can summarize the best configuration as well as all of the results as follows:

```
...
# report the best configuration
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
# report all configurations
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

Tying this together, the example below grid searches five different class weights for logistic regression on the imbalanced dataset.

We might expect that the heuristic class weighing is the best performing configuration.

```
# grid search class weights with logistic regression for imbalance classification
from numpy import mean
from sklearn.datasets import make_classification
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.linear_model import LogisticRegression
# generate dataset
X, y = make_classification(n_samples=10000, n_features=2, n_redundant=0,
    n_clusters_per_class=1, weights=[0.99], flip_y=0, random_state=2)
# define model
model = LogisticRegression(solver='lbfgs')
# define grid
balance = [{0:100,1:1}, {0:10,1:1}, {0:1,1:1}, {0:1,1:10}, {0:1,1:100}]
param_grid = dict(class_weight=balance)
# define evaluation procedure
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# define grid search
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=cv, scoring='roc_auc'
# execute the grid search
grid_result = grid.fit(X, y)
# report the best configuration
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
# report all configurations
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

Running the example evaluates each class weighting using repeated k-fold cross-validation and reports the best configuration and the associated mean ROC AUC

In this case, we can see that the 1:100 majority to min C score. This matches the configuration for the general

It might be interesting to explore even more severe cla C AUC score.

```
Best: 0.989077 using {'class_weight': {0: 1, 1
0.982498 (0.016722) with: {'class_weight': {0:
0.983623 (0.015760) with: {'class_weight': {0:
0.985387 (0.013890) with: {'class_weight': {0:
```


Start Machine Learning

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

Email Address

START MY EMAIL COURSE

```
0.988044 (0.010384) with: {'class_weight': {0: 1, 1: 10}}
0.989077 (0.006865) with: {'class_weight': {0: 1, 1: 100}}
```

Further Reading

This section provides more resources on the topic if you are looking to go deeper.

Papers

- Logistic Regression in Rare Events Data, 2001.
- The Estimation of Choice Probabilities from Choice Based Samples, 1977.

Books

- Learning from Imbalanced Data Sets, 2018.
- Imbalanced Learning: Foundations, Algorithms, and Applications, 2013.

APIs

- sklearn.utils.class_weight.compute_class_weight API.
- sklearn.linear_model.LogisticRegression API.
- sklearn.model_selection.GridSearchCV API.

Summary

In this tutorial, you discovered cost-sensitive logistic regression for imbalanced classification.

Specifically, you learned:

- How standard logistic regression does not support imbalanced classification.
- How logistic regression can be modified to weight model error by class weight when fitting the coefficients.
- How to configure class weight for logistic regression and how to grid search different class weight configurations.

Do you have any questions?
Ask your questions in the comments below and I will d

Get a Handle on Imba

Develop Imbalanced Lea

...with just a few lir

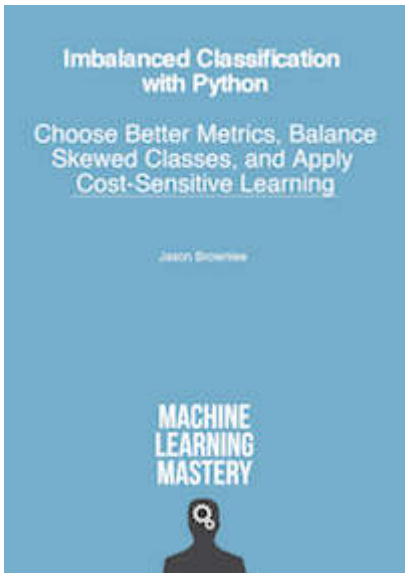
Discover how in
Imbalanced Classif

Start Machine Learning

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

Email Address

START MY EMAIL COURSE



It provides **self-study tutorials** and **end-to-end projects** on:
Performance Metrics, *Undersampling Methods*, *SMOTE*, *Threshold Moving*, *Probability Calibration*, *Cost-Sensitive Algorithms*
and much more...

Bring Imbalanced Classification Methods to Your Machine Learning Projects

SEE WHAT'S INSIDE

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 →

‹ Tour of Data Sampling Methods for Imbalanced Classification

Cost-Sensitive Decision Trees for Imbalanced Classification ›

18 Responses to *Cost-Sensitive Logistic Regression for Imbalanced Classification*

Elie January 27, 2020 at 7:11 am #

REPLY

Jason, almost done from reading the book.

Really great piece of work!

One minor recommendation: I'd like to see more expla

Jason Brownlee January 27, 2020 at 7:40 am

Well done on your progress!

Start Machine Learning

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

Email Address

START MY EMAIL COURSE

Great suggestion, thanks.

Rodney Silva January 27, 2020 at 10:18 am #

REPLY

I would like to see the improvement on the precision and recall for the minor class.

Jason Brownlee January 27, 2020 at 2:33 pm #

REPLY

Thanks, great suggestion.

marco January 29, 2020 at 2:42 am #

REPLY

Hello Jason,
a question about SVC and linearSVC.
What is the difference?
I'm trying a sentiment analysis with 1000 observations (750 training + 250 test).
Is it better to use SVC o linearSVC for the analysis?
What is the meaning of C hyperparameter in linearSVC (in simple words)?
I've found the C parameter is common also in other algorithms. The meaning is the same?
Thanks

Jason Brownlee January 29, 2020 at 6:45 am #

REPLY

SVC can do linear svc via a linear kernel. linear svc just uses the linear kernel and nothing else and is optimized for this use case – faster/more efficient.

More on C here:
https://machinelearningmastery.com/support-vector-machines-for-machine-learning/

Diane Halliwell January 30, 2020 at 6:58 am #

Hi

Does you book cover example-dependent cost-sensitiv

Jason Brownlee January 30, 2020 at 6:59 am

Start Machine Learning

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

Email Address

START MY EMAIL COURSE

No, just classes-based costs.

Temitope Mamukuyomi January 31, 2020 at 8:22 am #

REPLY

Thanks for the tutorial

Jason Brownlee January 31, 2020 at 2:04 pm #

REPLY

You're very welcome, I hope it helps you with your project!

macilane manjate January 31, 2020 at 3:43 pm #

REPLY

Dear Jason Browniee,
Good morning.
Whren will you have the book in R.
Kind regards,
Macilane

Jason Brownlee February 1, 2020 at 5:46 am #

REPLY

No plans at this stage. My focus is Python given that it is the most popular language for machine learning at the moment.

Sergio Garcia Garcia January 31, 2020 at 7:53 pm #

REPLY

"For example, we have 10,000 examples in the dataset, 9990 in class 0, and 100 in class 1."

Wouldn't be 9990-10 or 9900-100?

Good article, clear explanation

Jason Brownlee February 1, 2020 at 5:53 am

Thanks, fixed!

Start Machine Learning

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

Email Address

START MY EMAIL COURSE

Carlos February 7, 2020 at 3:20 am #

REPLY

Could you please provide the code for “Counter”
NameError: name ‘Counter’ is not defined
Thank you

Jason Brownlee February 7, 2020 at 8:24 am #

REPLY

You must copy the full code example that includes the important statement.

Atefeh April 12, 2020 at 4:19 am #

REPLY

Thank you for the great work. It was very helpful. I was looking to see how we can approach a specific problem both with classification and regression models? Let’s say I want to study the air temperature-soil temperature relation both with classification and regression models to get both the best fit and the decision boundary. Is there any article that can illustrate how steps will be different for these two approaches?

Jason Brownlee April 12, 2020 at 6:25 am #

REPLY

You’re welcome.

I don’t have an example, sorry.

Leave a Reply

Start Machine Learning

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

Email Address

START MY EMAIL COURSE

Name (required)

Email (will not be published) (required)

Website

SUBMIT COMMENT

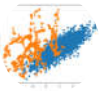
Welcome!
My name is *Jason Brownlee* PhD, and I **help developers** get results with **machine learning**.
Read more

Never miss a tutorial:

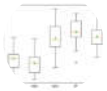
Picked for you:


8 Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset


Imbalanced Classification With Python (7-Day Mini-Course)


SMOTE for Imbalanced Classification with Python


Tour of Evaluation Metrics for Imbalanced Classification


Imbalanced Multiclass Classification with the Glass

Loving the

The Imbalanced Classification with Python E

Start Machine Learning

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

Email Address

START MY EMAIL COURSE

SEE WHAT'S INSIDE

© 2019 Machine Learning Mastery Pty. Ltd. All Rights Reserved.
Address: PO Box 206, Vermont Victoria 3133, Australia. | ACN: 626 223 336.
LinkedIn | Twitter | Facebook | Newsletter | RSS

Privacy | Disclaimer | Terms | Contact | Sitemap | Search

Start Machine Learning

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

Email Address

START MY EMAIL COURSE