How to Fix k-Fold Cross-Validation for Imbalanced Classification

by **Jason Brownlee** on January 13, 2020 in **Imbalanced Classification**

Last Updated on May 29, 2020

Model evaluation involves using the available dataset to fit a model and estimate its performance when making predictions on unseen examples.

It is a challenging problem as both the training dataset used to fit the model and the test set used to evaluate it must be sufficiently large and representative of the underlying problem so that the resulting estimate of model performance is not too optimistic or pessimistic.

The two most common approaches used for model evaluation are the train/test split and the k-fold cross-validation procedure. Both approaches can be very effective in general, although they can result in misleading results and potentially fail when used on classification problems with a severe class imbalance.

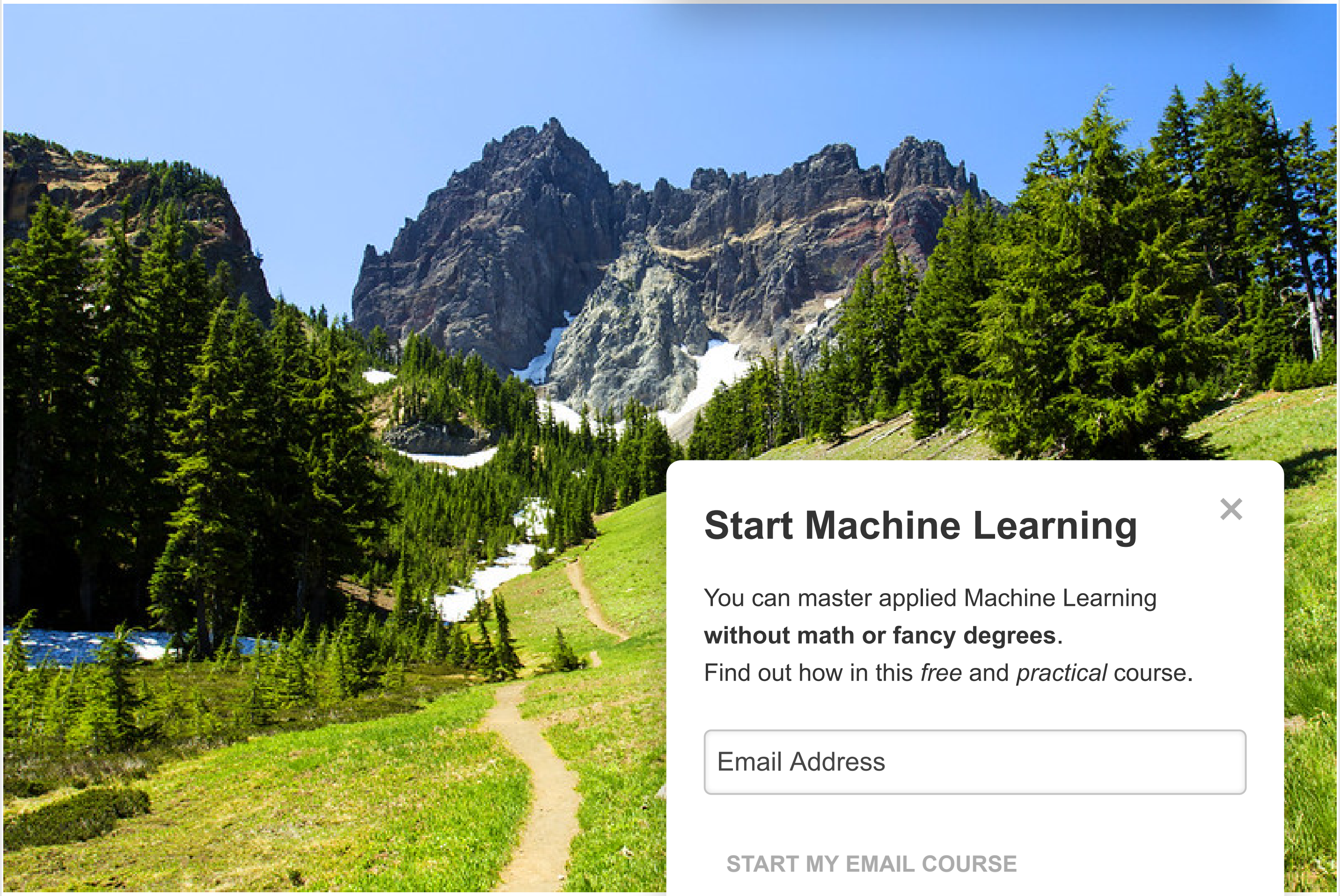
In this tutorial, you will discover how to evaluate classifier models on imbalanced datasets.

After completing this tutorial, you will know:

- The challenge of evaluating classifiers on datasets using train/test splits and cross-validation.
- How a naive application of k-fold cross-validation and train-test splits will fail when evaluating classifiers on imbalanced datasets.
- How modified k-fold cross-validation and train-test splits can be used to preserve the class distribution in the dataset.

Discover SMOTE, one-class classification, cost-sensitive learning, threshold moving, and much more in my new book, with 30 step-by-step tutorials and full Python source code.

Let's get started.



How to Use k-Fold Cross-Validation for Imbalanced Classification
Photo by Bonnie Moreland, some rights reserved.

Tutorial Overview

This tutorial is divided into three parts; they are:

1. Challenge of Evaluating Classifiers
2. Failure of k-Fold Cross-Validation
3. Fix Cross-Validation for Imbalanced Classification

Challenge of Evaluating Classifiers

Evaluating a classification model is challenging because we won’t know how good a model is until it is used.

Instead, we must estimate the performance of a model using available data where we already have the target or outcome.

Model evaluation involves more than just evaluating a model; it includes testing different data preparation schemes, different learning algorithms, and different hyperparameters for well-performing learning algorithms.

- Model = Data Preparation + Learning Algorithm + Hyperparameters

Ideally, the model construction procedure (data preparation, learning algorithm, and hyperparameters) with the best score (with your chosen metric) can be selected and used.

The simplest model evaluation procedure is to split a dataset into two parts and use one part for training a model and the second part for testing the model. As such, the parts of the dataset are named for their function, train set and test set respectively.

This is effective if your collected dataset is very large and representative of the problem. The number of examples required will differ from problem to problem, but may be thousands, hundreds of thousands, or millions of examples to be sufficient.

A split of 50/50 for train and test would be ideal, althou 33 or 80/20 for train and test sets.

We rarely have enough data to get an unbiased estim on of a model. Instead, we often have a much smaller da strategies must be used on this dataset.

The most used model evaluation scheme for classifier

The k-fold cross-validation procedure involves splitting s are used to train a model, and the holdout *k*th fold is u each of the folds is given an opportunity to be used as the holdout test set. A total of *k* models are fit and evaluated, and the performance of the model is calculated as the mean of these runs.

The procedure has been shown to give a less optimistic estimate of model performance on small training datasets than a single train/test split. A value of *k=10* has been shown to be effective across a wide range of dataset sizes and model types.

Failure of k-Fold Cross-Validation

Sadly, the k-fold cross-validation is not appropriate for

> *A 10-fold cross-validation, in particular, the most commonly used error-estimation method in machine learning, can easily break down in the case of class imbalances, even if the skew is less extreme than the one previously considered.*

— Page 188, Imbalanced Learning: Foundations, Algorithms, and Applications, 2013.

The reason is that the data is split into *k*-folds with a uniform probability distribution.

This might work fine for data with a balanced class distribution, but when the distribution is severely skewed, it is likely that one or more folds will have few or no examples from the minority class. This means that some or perhaps many of the model evaluations will be misleading, as the model need only predict the majority class correctly.

We can make this concrete with an example.

First, we can define a dataset with a 1:100 minority to

This can be achieved using the make_classification() f g
the number of examples (1,000), the number of classe %).

```
# generate 2 class dataset
X, y = make_classification(n_samples=1000, n_c om_s
```

The example below generates the synthetic binary cla
distribution.


```
# create a binary classification dataset
from numpy import unique
from sklearn.datasets import make_classification
# generate 2 class dataset
X, y = make_classification(n_samples=1000, n_classes=2, weights=[0.99, 0.01], flip_y=0, random_
# summarize dataset
classes = unique(y)
total = len(y)
for c in classes:
    n_examples = len(y[y==c])
    percent = n_examples / total * 100
    print('> Class=%d : %d/%d (%.1f%%)' % (c, n_examples, total, percent))
```

Running the example creates the dataset and summarizes the number of examples in each class.

By setting the *random_state* argument, it ensures that we get the same randomly generated examples each time the code is run.

```
> Class=0 : 990/1000 (99.0%)
> Class=1 : 10/1000 (1.0%)
```

A total of 10 examples in the minority class is not many. If we used 10-folds, we would get one example in each fold in the ideal case, which is not enough to train a model. For demonstration purposes, we will use 5-folds.

In the ideal case, we would have 10/5 or two examples in each fold, meaning 4*2 (8) folds worth of examples in a training dataset and 1*2 folds (2) in a given test dataset.

First, we will use the KFold class to randomly split the dataset into 5-folds and check the composition of each train and test set. The complete example is listed below.

```
# example of k-fold cross-validation with an imbalanced dataset
from sklearn.datasets import make_classification
from sklearn.model_selection import KFold
# generate 2 class dataset
X, y = make_classification(n_samples=1000, n_classes=2, weights=[0.99, 0.01], flip_y=0, random_
kfold = KFold(n_splits=5, shuffle=True, random_state=1)
# enumerate the splits and summarize the distributions
for train_ix, test_ix in kfold.split(X):
    # select rows
    train_X, test_X = X[train_ix], X[test_ix]
    train_y, test_y = y[train_ix], y[test_ix]
    # summarize train and test composition
    train_0, train_1 = len(train_y[train_y==0]
    test_0, test_1 = len(test_y[test_y==0]),
    print('>Train: 0=%d, 1=%d, Test: 0=%d, 1=
```


Running the example creates the same dataset and e ss distribution for both the train and test sets.

We can see that in this case, there are some splits tha s, and others that are much worse, such as 6/4 (optimist

Evaluating a model on these splits of the data would not give a reliable estimate of performance.

```
>Train: 0=791, 1=9, Test: 0=199, 1=1
>Train: 0=793, 1=7, Test: 0=197, 1=3
>Train: 0=794, 1=6, Test: 0=196, 1=4
>Train: 0=790, 1=10, Test: 0=200, 1=0
>Train: 0=792, 1=8, Test: 0=198, 1=2
```

We can demonstrate a similar issue exists if we use a simple train/test split of the dataset, although the issue is less severe.

We can use the train_test_split() function to create a 50/50 split of the dataset and, on average, we would expect five examples from the minority class to appear in each dataset if we performed this split many times.

The complete example is listed below.

```
# example of train/test split with an imbalanced dataset
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
# generate 2 class dataset
X, y = make_classification(n_samples=1000, n_classes=2, weights=[0.99, 0.01], flip_y=0, random_
# split into train/test sets with same class ratio
trainX, testX, trainy, testy = train_test_split(X, y, test_size=0.5, random_state=2)
# summarize
train_0, train_1 = len(trainy[trainy==0]), len(trainy[trainy==1])
test_0, test_1 = len(testy[testy==0]), len(te
print('>Train: 0=%d, 1=%d, Test: 0=%d, 1=%d' %
```

Running the example creates the same dataset as before and splits it into a random train and test split.

In this case, we can see only three examples of the minority class are present in the training set, with seven in the test set.

Evaluating models on this split would not give them enough examples to learn from, too many to be evaluated on, and likely give poor performance. You can imagine how the situation could be worse with an even more severe random spit.

```
>Train: 0=497, 1=3, Test: 0=493, 1=7
```

Fix Cross-Validation for Imbalanced Classification

The solution is to not split the data randomly when usi

Specifically, we can split a dataset randomly, although distribution in each subset. This is called stratification ne class, is used to control the sampling process.

For example, we can use a version of k-fold cross-vali distribution in each fold. It is called stratified k-fold cro in each split of the data to match the distribution in the c

> *… it is common, in the case of class imbalances in particular, to use stratified 10-fold cross-validation, which ensures that the proportion of positive to negative examples found in the original distribution is respected in all the folds.*

— Page 205, Imbalanced Learning: Foundations, Algorithms, and Applications, 2013.

We can make this concrete with an example.

We can stratify the splits using the StratifiedKFold class that supports stratified k-fold cross-validation as its name suggests.

Below is the same dataset and the same example with the stratified version of cross-validation.

```
# example of stratified k-fold cross-validation with an imbalanced dataset
from sklearn.datasets import make_classification
from sklearn.model_selection import StratifiedKFold
# generate 2 class dataset
X, y = make_classification(n_samples=1000, n_classes=2, weights=[0.99, 0.01], flip_y=0, random_
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)
# enumerate the splits and summarize the distributions
for train_ix, test_ix in kfold.split(X, y):
    # select rows
    train_X, test_X = X[train_ix], X[test_ix]
    train_y, test_y = y[train_ix], y[test_ix]
    # summarize train and test composition
    train_0, train_1 = len(train_y[train_y==0
    test_0, test_1 = len(test_y[test_y==0]),
```

```
    print('>Train: 0=%d, 1=%d, Test: 0=%d, 1=%d' % (train_0, train_1, test_0, test_1))
```

Running the example generates the dataset as before and summarizes the class distribution for the train and test sets for each split.

In this case, we can see that each split matches what we expected in the ideal case.

Each of the examples in the minority class is given one opportunity to be used in a test set, and each train and test set for each split of the data has the same class distribution.

```
>Train: 0=792, 1=8, Test: 0=198, 1=2
>Train: 0=792, 1=8, Test: 0=198, 1=2
>Train: 0=792, 1=8, Test: 0=198, 1=2
>Train: 0=792, 1=8, Test: 0=198, 1=2
>Train: 0=792, 1=8, Test: 0=198, 1=2
```

This example highlights the need to first select a value e
are a sufficient number of examples in the train and te
from the minority class in the test set is probably too fe

It also highlights the requirement to use stratified *k*-fol
preserve the class distribution in the train and test sets

We can also use a stratified version of a train/test split

This can be achieved by setting the "*stratify*" argumen e
"*y*" variable containing the target variable from the dataset. From this, the function will determine the desired class distribution and ensure that the train and test sets both have this distribution.

We can demonstrate this with a worked example, listed below.

```
# example of stratified train/test split with an imbalanced dataset
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
# generate 2 class dataset
X, y = make_classification(n_samples=1000, n_classes=2, weights=[0.99, 0.01], flip_y=0, random_
# split into train/test sets with same class ratio
trainX, testX, trainy, testy = train_test_split(X, y, test_size=0.5, random_state=2, stratify=y
# summarize
train_0, train_1 = len(trainy[trainy==0]), len(trainy[trainy==1])
test_0, test_1 = len(testy[testy==0]), len(testy[testy==1])
print('>Train: 0=%d, 1=%d, Test: 0=%d, 1=%d' % (train_0, train_1, test_0, test_1))
```

Running the example creates a random split of the dataset into training and test sets, ensuring that the class distribution is preserved, in this case leaving five examples in each dataset.

```
>Train: 0=495, 1=5, Test: 0=495, 1=5
```

Further Reading

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

Tutorials

- A Gentle Introduction to k-fold Cross-Validation

Books

- Imbalanced Learning: Foundations, Algorithms, and Applications, 2013.

API

- sklearn.model_selection.KFold API.
- sklearn.model_selection.StratifiedKFold API.
- sklearn.model_selection.train_test_split API.

Summary

In this tutorial, you discovered how to evaluate classifi

Specifically, you learned:

- The challenge of evaluating classifiers on dataset
- How a naive application of k-fold cross-validation classifiers on imbalanced datasets.
- How modified k-fold cross-validation and train-tes ion in the dataset.

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

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 →

‹ A Gentle Introduction to Probability Metrics for Imbalanced Cla

What Is t ric? ›

26 Responses to *How to Fix k-Fold Cros Classification*

Sadegh Arefnezhad January 14, 2020 at 1:05 am # REPLY

Thank you!

Jason Brownlee January 14, 2020 at 7:24 am # REPLY

You're welcome!

Markus January 15, 2020 at 10:53 pm # REPLY

As always, KUDOS!

Jason Brownlee January 16, 2020 at 6:16 am # REPLY

Thanks!

avinash January 25, 2020 at 1:13 am # REPLY

train_X, test_y = X[train_ix], X[test_ix] should be train_X, test_X = X[train_ix], X[test_ix]

Jason Brownlee January 25, 2020 at 8:38 am # REPLY

Thanks, fixed!

Sachin February 15, 2020 at 1:31 am #

Hello, Great article. Thanks.

My problem is that in my original dataset, I have 1% re responders and 50% non responders. After this I have validation on sampled data (not real data)? Also, I am test dataset.

Jason Brownlee February 15, 2020 at 6:34 a

No.

The resampling must happen within the cross-validation folds. See examples here:
https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/

Sachin February 16, 2020 at 2:57 am # REPLY

Thanks. Even in that example the SMOTE and undersampling of majority class is done before stratified k-fold validation is applied on X and y. This means the validation is done on balanced data. Please correct if I am wrong. Thanks again.

Jason Brownlee February 16, 2020 at 6:13 am # REPLY

Not true.

We use a pipeline to ensure that data sampling occurs within the cross-validation procedure.

Perhaps re-read the tutorial?

Sachin February 17, 2020 at 9:08 pm # REPLY

According to the example, this is the pipeline:

steps = [('over', over), ('under', under), ('model', model)]
pipeline = Pipeline(steps=steps)

and then this step does the cross validation:

scores = cross_val_score(pipeline, X, y, scoring='roc_auc', cv=cv, n_jobs=-1)

doesn't this mean that pipeline is applied to original dataset (X and y) – which means first
first oversampling by SMOTE, then random undersampling and then the model is applied and validated with ROC curve?

Jason Brownlee February 18, 2020 at 6:19 a

Yes, but the over/under sampling is applie

faezeh March 25, 2020 at 2:08 am #

Hello,
What is the disadvantage(s) of the method s
Stratified cross validation ?

Jason Brownlee March 25, 2020 at 6:34 am # REPLY

It is computationally more expensive, slightly.

It is only for classification, not regression.

faezeh March 25, 2020 at 4:39 pm # REPLY

Thank you

Jason Brownlee March 26, 2020 at 7:49 am # REPLY

You're welcome.

Daniele April 22, 2020 at 12:30 am # REPLY

Hi, great post! In the case of GridSearchCV, is (Stratified)KFolds implicit? This is an example:

gs_clf = GridSearchCV(clf_pipe, param_grid=params, verbose=0, cv=5, n_jobs=-1)

Thanks for your reply!

Jason Brownlee April 22, 2020 at 5:59 am # REPLY

Thanks.

I think so. It is better to be explicit with you cv meth

ghizlan April 26, 2020 at 12:24 pm #

Hello Jason, great post thank you !
One question
I have a data of 100 samples and 10 features. I want t
(variables)

method 1: I divide my data into 80% for training using k-fold cross validation and then validate the model on the unseen data(20%)

Method 2: I use all my data to fit the model using k-fold cross validation.

Between the two methods, which is the true one?

Jason Brownlee April 27, 2020 at 5:25 am # REPLY

They are both viable/true, use the method that you believe will give you the most robust estimate of model performance for your specific project.

ghizlan April 27, 2020 at 7:17 am # REPLY

Thank you, Jason

Jason Brownlee April 27, 2020 at 7:34 am # REPLY

You're welcome.

Grzegorz Kępisty May 7, 2020 at 4:53 pm #

REPLY

It seems that "stratify" flag from train_test_split() is very useful. What is more, almost any real classification problem is imbalanced. Then I guess that this strategy shall be always used as a default? (as there is nothing to lose, right?)

Jason Brownlee May 8, 2020 at 6:23 am #

REPLY

Exactly right on all points!

I guess the only practical down side is a slight com

Saily Shah May 21, 2020 at 1:18 am #

Hi Jason. Wonderful article:-). I just had one c splitting of the imbalanced data in train and test datase maintained in training and test dataset. In stratified k-fo "training" dataset is divided into n_folds, it maintains th we are handling imbalanced data, we need to use "stra fold cv" . Am i correct?

Jason Brownlee May 21, 2020 at 6:20 am #

REPLY

Thanks.

Correct.

Leave a Reply

Name (required)

Email (will not be published) (required)

Website

SUBMIT COMMENT

Welcome!
My name is *Jason Brownlee* PhD, and I **hel**
Read more

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

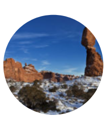
START MY EMAIL COURSE

Never miss a tutorial:

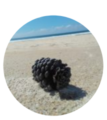
Picked for you:

8 Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset

SMOTE for Imbalanced Classification with Python



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



A Gentle Introduction to Threshold-Moving for Imbalanced Classification

Step-By-Step Framework for Imbalanced Classification Projects

Loving the

Start Machine Learning

Start Machine Learning