# mlr3 Resampling: An Introduction

Natalie Foss

2022-09-19

## Introduction

The purpose of this post is to introduce you to `mlr3` resampling and give an example of how to train a model with resampling (showing the best practice methods), using the `palmer penguins` dataset. `palmer penguins` is a built in task described as "classification data to predict the species of penguins". Read more about the dataset here.

This blog post picks up from the previous post If you do not wish to read that post, this code chunk will get you up to speed (though we recommend you skim it).

```
library("mlr3")
task = tsk("penguins")
learner = lrn("classif.rpart")
measure = msr("classif.ce")
```

### Review

In the last post we saw how depending on the random seed we can get performance estimates that vary widely:

```
set.seed(3)
splits = partition(task, ratio = 0.8, cat_col="species")
learner$train(task, splits$train)
prediction = learner$predict(task, splits$test)
prediction$score(measure)
```

```
## classif.ce
##  0.1014493
```

```
set.seed(22)
splits = partition(task, ratio = 0.8, cat_col="species")
learner$train(task, splits$train)
prediction = learner$predict(task, splits$test)
prediction$score(measure)
```

```
## classif.ce
## 0.01449275
```

In order to report the accurate predictive performance of your model, you need resampling.

## Resampling methods

Similar to learners and performance measures, there are many popular resampling method implemented. You read about the options here, or just view them:

```r
as.data.table(mlr_resamplings)
```

```
##             key                         label         params iters
## 1:    bootstrap                     Bootstrap ratio,repeats    30
## 2:       custom                 Custom Splits                  NA
## 3:    custom_cv Custom Split Cross-Validation                  NA
## 4:           cv              Cross-Validation          folds    10
## 5:      holdout                       Holdout          ratio     1
## 6:     insample           Insample Resampling                   1
## 7:          loo                 Leave-One-Out                  NA
## 8: repeated_cv     Repeated Cross-Validation folds,repeats   100
## 9: subsampling                   Subsampling ratio,repeats    30
```

Each resampling method has a set of parameters (0, 1, 2). To view the parameters and other metaa-information:

```r
rsmp("bootstrap")$param_set
```

```
## <ParamSet>
##         id    class lower upper nlevels        default value
## 1:   ratio ParamDbl     0     1     Inf <NoDefault[3]>     1
## 2: repeats ParamInt     1   Inf     Inf <NoDefault[3]>    30
```

Cross Validation is the resampling method we will use in this example, read about it here (called k-fold cross validation).

To initialize the resampling object we use the sugar function `rsmp()`. We specify any parameters that we want different than the defaults here.

```r
cv = rsmp("cv", folds=10)
```

Now lets perform the resampling. We will set `store_models` to TRUE so we can look at individual models later (this defaults to FALSE to limit memory consumption).

```r
rr = resample(task, learner, cv, store_models = TRUE)
```

```
## INFO  [10:11:29.928] [mlr3] Applying learner 'classif.rpart' on task 'penguins' (iter 1/10)
## INFO  [10:11:29.984] [mlr3] Applying learner 'classif.rpart' on task 'penguins' (iter 2/10)
## INFO  [10:11:30.005] [mlr3] Applying learner 'classif.rpart' on task 'penguins' (iter 3/10)
## INFO  [10:11:30.027] [mlr3] Applying learner 'classif.rpart' on task 'penguins' (iter 4/10)
## INFO  [10:11:30.044] [mlr3] Applying learner 'classif.rpart' on task 'penguins' (iter 5/10)
## INFO  [10:11:30.060] [mlr3] Applying learner 'classif.rpart' on task 'penguins' (iter 6/10)
## INFO  [10:11:30.100] [mlr3] Applying learner 'classif.rpart' on task 'penguins' (iter 7/10)
## INFO  [10:11:30.119] [mlr3] Applying learner 'classif.rpart' on task 'penguins' (iter 8/10)
## INFO  [10:11:30.137] [mlr3] Applying learner 'classif.rpart' on task 'penguins' (iter 9/10)
## INFO  [10:11:30.154] [mlr3] Applying learner 'classif.rpart' on task 'penguins' (iter 10/10)
```

```
rr
```

```
## <ResampleResult> of 10 iterations
## * Task: penguins
## * Learner: classif.rpart
## * Warnings: 0 in 0 iterations
## * Errors: 0 in 0 iterations
```

Now for the moment of truth, we can look at the performance **aggregated** over all resamplings.

```
rr$aggregate(measure)
```

```
## classif.ce
## 0.05260504
```

Compare this classification error to the ones from the review section (0.1014493 and 0.01449275). It turns out that the true performance is between these values.

Learn about how to use pipelines to make resampling simpler here.

Or learn about benchmarking and how it can help you compare the performance of different learners and different tasks here.