mlr3 Set-Up: An Introduction

Natalie Foss

2022-09-18

Introduction

This blog post will introduce you to `mlr3` and give an example of how to setup your `mlr3` project, 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.

Basic Setup

The first step to using the `mlr3` package is to load in the library. In this blog post whenever you see `library()` and you have not previously installed the package, run `install.packages()`.

```
# install.packages("mlr3")
library("mlr3")
```

The second step of this machine learning process is to load the dataset. In `mlr3` data sets are stored in objects called tasks. Tasks contain lots of meta information that is useful for the learners (like which column is the response). Initializing the task is very easy since `penguins` is a built in task. The sugar function `tsk()` can be called like so:

```
task = tsk("penguins")
```

Get to Know the Data

It is important to note several things about your data:

1. Response type - this is important because the learner you pick will need to be compatible with this type
2. Feature types - this is important because learners, performance measures, and more have to be compatible with the feature types of your data. We will will go over how to verify this later
3. Missing values - as with 1 and 2, this is important because certain learners, performance measures and more don't allow for missing values. Imputation can be used in these cases.

Penguins automatically picks the "species" column as the response. In `mlr3` the response/column you hope to predict is called the target.

```
task$col_roles$target
```

```
## [1] "species"
```

The following columns are identified as the features of the task.

```
task$col_roles$feature
```

```
## [1] "bill_depth"     "bill_length"    "body_mass"      "flipper_length"
## [5] "island"         "sex"            "year"
```

We can look at the data types of all of these columns by printing out the summary of the task:

```
task
```

```
## <TaskClassif:penguins> (344 x 8): Palmer Penguins
## * Target: species
## * Properties: multiclass
## * Features (7):
##   - int (3): body_mass, flipper_length, year
##   - dbl (2): bill_depth, bill_length
##   - fct (2): island, sex
```

Since the target data type is `factor` the task defaulted to `TaskClassif`, however, if the target data type was `numeric` it would have defaulted to `TaskRegr`. We go over how to change this in a post about encoding.

To see if there are missing values in the data set we use the function `missings()` that gives the number of missing values per column.

```
task$missings()
```

```
##        species     bill_depth    bill_length      body_mass flipper_length
##              0              2              2              2              2
##         island            sex           year
##              0             11              0
```

For the purposes of this post we won't worry about the missing values, however, this topic is discussed in a post about imputation.

Learner Setup

Learners are the `mlr3` object that encapsulates many popular machine learning algorithms. Each learner has a `$train()` and `$predict()` function that allows you to easily make use of these algorithms. You will want to select from the pool of learners that are specialized for the type of task you have (for this example recall the task type is `TaskClassif`. We can see all available classification learners like so:

```
mlr_learners$keys("classif")
```

```
## [1] "classif.debug"       "classif.featureless" "classif.rpart"
```

Note: to see other types of learners replace "classif" with one of: "clust", "dens", "regr", or "surv".

We will use the decision tree model `classif.rpart`, read about it in its documentation or this article. To initialize the learner use the sugar function `lrn()`.

```
learner = lrn("classif.rpart")
```

Important Notes on Data

You may be tempted train the learner on the data right now, however, it is important to consider how we will measure the performance of our model. If we were to train the model on all available data then we would not have leftover data to test the `predict()` function.

Thus, we use train and test splits. The train data is a subset of the total data that is used to train the model. The test data is a smaller subset of the data that is used to measure the performance of the model. This is the simplest way of measuring the performance of your model. Most data scientists use resampling methods because they can give a more accurate representation of the performance of models. We will go over resampling in this post.

Note: It is **very** bad practice to predict on data that was used in the training process, read about data leakage.

We will go over the best practice ways of training models, while preventing data leakage in the next post.