Lab Notebook

Shannon Williams & Daniel Ward & Euan Enticott

22/01/2021

Introduction

This lab notebook contains the analyses undertaken as part of the COMPASS TB1 group project. We will be working on the Chicago crime data set, which contains information on incidents reported in Chicago.

In general, the analysis can be broken up into 4 sections.

The notebook will be broken up into four sections:

1. Classification: Predicting if an incident led to an arrest
2. Predicting the number of crimes using interpolation
3. Forecasting the number of crimes
4. Work flow and Computational Techniques

Workflow and computational techniques

In the development of the package for this project, we utilised many computational techniques, that are outlined below.

Object-oriented approach

We decided to use an object oriented approach to constructing models. Specifically, we made use of the `R6Class` from the `R6` package. `R6` objects are mutable and use an object oriented approach similar to most programming languages (unlike the more basic `S3` and `S4` classes). `R6` classes are very similar to reference classes, although generally `R6` classes are preferred. This is for a variety of reasons, including the fact that `R6` classes are much faster, and they handle fields in a neater way by putting them in a separate environment. More information on this can be found here. When building models, a model could be fitted using `model$fit(X, y)`, and predictions made with `model$predict(X)`. It is worth noting that this use of `fit` and `predict` methods in object-oriented machine learning packages is used elsewhere, particularly in the Python package `scikit-learn`. The key advantage of this approach is that it abstracts away the underlying complexities, and creates a consistent structure that makes it easier to compare different models.

Parallel

Parallel programming was used to speed up slow code, utilising the packages `doParallel`, `parallel` and `foreach`. In particular we implemented a parallel cross-validation function `kfold_cv`, which facilitated more accurate and faster assessment of model performance.

Documentation

In order to document functions in the package, we made use of `roxygen2`. This package automatically creates the `.Rd` documentation files for functions (or classes), from comments added above the function definitions.

As well as being efficient, this allows the code and documentation to coexist, meaning it is easier to remember to update the documentation.

Testing and continuous integration

For tests, we used the `testthat` package. This simply creates a new directory `./tests/testthat/` in which the user can define tests for the package functions. Although the tests can be run alone, generally, we made use of the package `rcmdcheck`. This contains a function `rcmdcheck::rcmdcheck()`, which not only runs the tests, but also carries out a more sophisticated check. Some examples of what is checked are listed below:

- Checks the package installs correctly
- Checks for missing documentation
- Automatically runs examples in documentation, to check they run successfully.
- Checks for undefined variables
- Checks for missing dependencies

In order to make sure that tests and checks were run on a regular basis, continuous integration was set up using github actions. This ran `rcmdcheck::rcmdcheck()`, for each pull request and push to the main/master branch. This limits the possibility of unintentionally introducing breaking changes.