I’ve been working with UbiOps lately, a service that runs your data science models as a service. They have recently started supporting R next to python! So let’s see if we can deploy a tidymodels model to UbiOps! I am not going to tell you a lot about UbiOps, that is for another post. I presume you know what it is, you know what tidymodels means for R and you want to combine these things.

The usecase is this: I’ve developed a tidymodels model that predicts red wine quality given its chemical properties. Now I want to put that model online so the business can send requests to an endpoint and receive quality estimations. The business knows how to talk to APIs, they shouldn’t care about what language I’m using to make the predictions.

I’m going to use the UbiOps webinterface but there is also a CLI, a python library and I believe an R client version is coming as well.

**About ubiops**

Full disclosure, the company I work for (Ordina) has a partnership with UbiOps. I’m not getting paid for this post, but obviously I hope they do well! I think it can be a great product to reduce the time to production.

In short UbiOps wraps your code inside a container and makes it an API endpoint, all you have to do is supply the code, zip it up and drop it in the right place.

**Create a tidymodel-model**

Train a model and write it away into an RDS file.

library(recipes)

library(magrittr)

library(workflows)

library(rsample)

library(parsnip)

library(earth)

winequality <- read.csv("data/winequality-red.csv")

split <- initial\_split(winequality, prop = 0.8,strata = "quality")

train <- training(split)

test <- testing(split)

## make recipe

rec\_wine <-

training(split) %>%

recipe(quality~.) %>%

step\_corr(all\_predictors()) %>%

step\_nzv(all\_predictors()) %>%

step\_center(all\_predictors(), -all\_outcomes()) %>%

step\_scale(all\_predictors(), -all\_outcomes()) %>%

prep()

## make model

marsmodel <-

mars(mode = "regression") %>%

set\_engine("earth")

# make workflow

mars\_wf <-

workflow() %>%

add\_recipe(rec\_wine) %>%

add\_model(marsmodel)

## fit model

fit\_mars <- mars\_wf %>%

fit(data = train)

# evaluate

fit\_mars %>%

predict(test) %>%

bind\_cols(test) %>%

rmse(quality, .pred)

# 65%, pretty bad but it is a start

#save model

saveRDS(fit\_mars,'ubiops\_deployment/deployment\_package/fit\_mars.RDS')

Now we can wrap the saved model into a deployment package and drop that zip into UbiOps.

**deployment package**

You have to create a deployment package[1](https://blog.rmhogervorst.nl/blog/2021/06/05/tidymodels-on-ubiops/#fn1) (a zipfile) that contains an init and request method. For R deployments you need a deployment.R file.

The deployment\_package.zip I created contains

deployment\_package/renv.lock # contains the packages needed

deployment\_package/fit\_mars.RDS # my trained model

deployment\_package/deployment.R # the code that runs

the deployment.R file looks like this (I just modified the UbiOps [deployment template)](https://github.com/UbiOps/deployment-template/tree/master/r):

# This file (containing the deployment code) is required to be called 'deployment.R' and should contain an 'init'

# and 'request' method.

#' @title Init

#' @description Initialisation method for the deployment.

#' It can for example be used for loading modules that have to be kept in memory or setting up connections.

#' @param base\_directory (str) absolute path to the directory where the deployment.R file is located

#' @param context (named list) details of the deployment that might be useful in your code

init <- function(base\_directory, context) {

## Init runs only once during initialisation.

print("Initialising My Deployment")

library(recipes)

library(magrittr)

library(workflows)

library(rsample)

library(parsnip)

library(earth)

modelloc = file.path(base\_directory, 'fit\_mars.RDS')

print(paste0("loading model at ", modelloc))

## Assigns fit\_mars to the global namespace so that the request function

## can make use of this model.

fit\_mars <<- readRDS(modelloc)

}

#' @title Request

#' @description Method for deployment requests, called separately for each individual request.

#' @param input\_data (str or named list) request input data

#' - In case of structured input: a named list, with as keys the input fields as defined upon deployment creation

#' - In case of plain input: a string

#' @param base\_directory (str) absolute path to the directory where the deployment.R file is located

#' @param context (named list) details of the deployment that might be useful in your code

#' @return output data (str or named list) request result

#' - In case of structured output: a named list, with as keys the output fields as defined upon deployment creation

#' - In case of plain output: a string

request <- function(input\_data, base\_directory, context) {

# read csv

print('reading csv')

winequality <- read.csv(file.path(input\_data[["input\_data"]]))

# predict on results

print("predicting")

predictions <-

fit\_mars %>%

predict(winequality) %>%

bind\_cols(winequality)

predictions %>%

write.csv("predictions.csv")

# return the csv location

list( output = 'predictions.csv')

}

In the deployment specification I set the input to be a file and the output as well. I set the language to R4.0 and upload the zip.

There were few gotchas you should pay attention to:

The init() function is called during startup and the request() function at every API request to this service. The request() function has no access to the inside of the init() function. So if you want to initialize something (like loading a model) you should use the superassignment operator <<- to put it into the global namespace. The first versions of the R support had an example using R6 classes, more (encapsulating) object oriented programming (OOP). When you use R6, the init function and request function are related to the same object and so you don’t have to superassign. I believe most R programmers don’t want to use all that boilerplate of R6 and this is good enough.

Furthermore I am super excited that ubiops decided to support renv, you can just pin your specific package versions and know for sure that those will be installed. All you have to do to make that work is add an renv.lock file to the deployment-package

Finally installation of tidymodels takes a long time, UbiOps already installed tidyverse so that helps a lot! but I recommend you don’t set tidymodels as dependency, but only the packages you use {recipes},{parsnip},{workflows},etc

**Reproducibility**

At the moment of creation (when I knitted this document ) this was the state of my machine: **click here to expand**

sessioninfo::session\_info()

─ Session info ───────────────────────────────────────────────────────────────

setting value

version R version 4.0.2 (2020-06-22)

os macOS 10.16

system x86\_64, darwin17.0

ui X11

language (EN)

collate en\_US.UTF-8

ctype en\_US.UTF-8

tz Europe/Amsterdam

date 2021-06-03

─ Packages ───────────────────────────────────────────────────────────────────

package \* version date lib source

blogdown 1.3 2021-04-14 [1] CRAN (R 4.0.2)

bookdown 0.22 2021-04-22 [1] CRAN (R 4.0.2)

bslib 0.2.5.1 2021-05-18 [1] CRAN (R 4.0.2)

cli 2.5.0 2021-04-26 [1] CRAN (R 4.0.2)

digest 0.6.27 2020-10-24 [1] CRAN (R 4.0.2)

evaluate 0.14 2019-05-28 [1] CRAN (R 4.0.1)

glue 1.4.2 2020-08-27 [1] CRAN (R 4.0.2)

htmltools 0.5.1.1 2021-01-22 [1] CRAN (R 4.0.2)

jquerylib 0.1.4 2021-04-26 [1] CRAN (R 4.0.2)

jsonlite 1.7.2 2020-12-09 [1] CRAN (R 4.0.2)

knitr 1.33 2021-04-24 [1] CRAN (R 4.0.2)

magrittr 2.0.1 2020-11-17 [1] CRAN (R 4.0.2)

R6 2.5.0 2020-10-28 [1] CRAN (R 4.0.2)

rlang 0.4.11 2021-04-30 [1] CRAN (R 4.0.2)

rmarkdown 2.8 2021-05-07 [1] CRAN (R 4.0.2)

sass 0.4.0 2021-05-12 [1] CRAN (R 4.0.2)

sessioninfo 1.1.1 2018-11-05 [1] CRAN (R 4.0.2)

stringi 1.6.2 2021-05-17 [1] CRAN (R 4.0.2)

stringr 1.4.0 2019-02-10 [1] CRAN (R 4.0.2)

withr 2.4.2 2021-04-18 [1] CRAN (R 4.0.2)

xfun 0.23 2021-05-15 [1] CRAN (R 4.0.2)

yaml 2.2.1 2020-02-01 [1] CRAN (R 4.0.2)

[1] /Library/Frameworks/R.framework/Versions/4.0/Resources/library