XGBoost example

Environment set-up

Load libraries

```
reqd_pkgs <- c("tidyverse","readxl","xgboost","pROC") # we'll use pROC to calculate
the AUC measure
for (pkg in reqd_pkgs) {
  if(!(pkg %in% installed.packages())) install.packages(pkg)
  if(!(pkg %in% (.packages()))) library(pkg, character.only = TRUE) }
```

Reproducibility

Initialise random number generator.

```
set.seed(1234)
```

Confirm versions of all loaded packages

```
sessionInfo()
```

```
## R version 3.6.1 (2019-07-05)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 17134)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United Kingdom.1252
## [2] LC_CTYPE=English_United Kingdom.1252
## [3] LC_MONETARY=English_United Kingdom.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United Kingdom.1252
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] pROC_1.15.3      xgboost_0.90.0.2 readxl_1.3.1     forcats_0.4.0
##  [5] stringr_1.4.0    dplyr_0.8.3      purrr_0.3.2      readr_1.3.1
##  [9] tidyr_0.8.3      tibble_2.1.3     ggplot2_3.2.1    tidyverse_1.2.1
##
## loaded via a namespace (and not attached):
##  [1] tidyselect_0.2.5  xfun_0.9          haven_2.1.1
##  [4] lattice_0.20-38   colorspace_1.4-1  generics_0.0.2
##  [7] vctrs_0.2.0       htmltools_0.3.6   yaml_2.2.0
## [10] rlang_0.4.0       pillar_1.4.2      glue_1.3.1
## [13] withr_2.1.2       modelr_0.1.5      plyr_1.8.4
## [16] munsell_0.5.0     gtable_0.3.0      cellranger_1.1.0
## [19] rvest_0.3.4       evaluate_0.14     knitr_1.24
## [22] broom_0.5.2       Rcpp_1.0.2        scales_1.0.0
## [25] backports_1.1.4   jsonlite_1.6      hms_0.5.1
## [28] digest_0.6.20     stringi_1.4.3     grid_3.6.1
## [31] cli_1.1.0         tools_3.6.1       magrittr_1.5
## [34] lazyeval_0.2.2    crayon_1.3.4      pkgconfig_2.0.2
## [37] zeallot_0.1.0     Matrix_1.2-17     data.table_1.12.2
## [40] xml2_1.2.2        lubridate_1.7.4   assertthat_0.2.1
## [43] rmarkdown_1.15    httr_1.4.1        rstudioapi_0.10
## [46] R6_2.4.0          nlme_3.1-140      compiler_3.6.1
```

Data preparation and exploration

Load dataset

The dataset can be downloaded from:
https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients
(https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients)

Use setwd() to set your working folder to the download location.

```
credit_df <- read_excel("default of credit card clients.xls", sheet="Data", skip=1)
```

Prepare data

In practice, at this point we would typically do considerable amounts of:

- Data cleansing,
- Exploratory analysis (e.g. using R's `DataExplorer` package), and
- Feature engineering (dataset reshaping, imputation, variable normalisation, etc).

For simplicity, here we are going to adjust the data minimally to match XGBoost's needs.

```
default <- credit_df[["default payment next month"]] # split out the results column
credit_df <- select(credit_df, -one_of("ID", "default payment next month")) # leav
e only valid explanatory variables
```

ML framework set-up

Split off a testing (aka "hold-out") dataset.

```
split_rows <- sample(c(rep(1, 21000), rep(2, 9000))) # 70:30 split of 30,000 record
s
credit_train <- credit_df[which(split_rows==1),]
credit_test <- credit_df[which(split_rows==2),]
default_train <- default[which(split_rows==1)]
default_test <- default[which(split_rows==2)]
```

Split the existing training dataset into parameter training and hyperparameter validation subsets.

```
split_rows <- sample(c(rep(1, 14700), rep(2, 6300))) # 70:30 split of 21,000 record
s
credit_trn <- credit_train[which(split_rows==1),]
credit_val <- credit_train[which(split_rows==2),]
default_trn <- default_train[which(split_rows==1)]
default_val <- default_train[which(split_rows==2)]
```

This split is only relevant when we are tuning hyperparameters. In real-world applications, we would typically use K-fold cross-validation rather than constructing separate train/validate sets.

Benchmark model fitting (logistic regression)

Calibrate model.

```
lgt_model <- glm(default ~ . - default, data=cbind(credit_train, list(default=defau
lt_train)), family="binomial")
```

Determine model score.

```
lgt_preds <- predict(lgt_model, newdata=credit_test, type="response")
table(lgt_preds >= 0.5, default_test == 1) # confusion matrix
```

```
##
##         FALSE TRUE
##   FALSE  6859 1495
##   TRUE    189  457
```

```
print(paste("Logistic regression score:", auc(default_test, lgt_preds))) # 0.717
```

```
## [1] "Logistic regression score: 0.717372189354962"
```

XGBoost model fitting

Default hyperparameter choices

Calibrate model.

```
xgb_model_default <- xgboost(data=as.matrix(credit_train), label=default_train,
                             nrounds=100, early_stopping_rounds=10,
                             objective="binary:logistic", eval_metric="auc",
                             verbose=0)
```

Determine model score.

```
xgb_preds_default <- predict(xgb_model_default, newdata=as.matrix(credit_test))
table(xgb_preds_default >= 0.5, default_test == 1)
```

```
##
##         FALSE TRUE
##   FALSE  6613 1251
##   TRUE    435  701
```

```
print(paste("Raw XGBoost score:", auc(default_test, xgb_preds_default))) # 0.758
```

```
## [1] "Raw XGBoost score: 0.758342785012839"
```

Hyperparameter tuning

Tuning the eta (learning rate) and max_depth (maximum depth of individual trees) parameters.

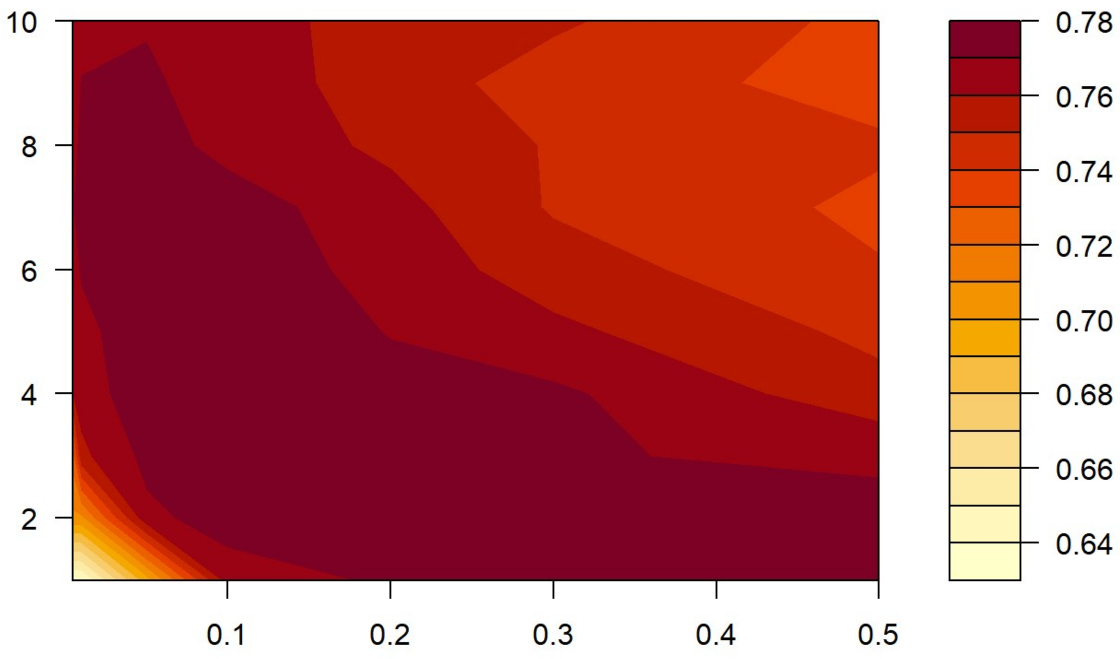
```
param_options <- list(eta=c(0.005, 0.01, 0.05, 0.1, 0.2, 0.3, 0.5),
                      max_depth=1:10)
param_sets <- cross_df(param_options) # each row is a different combination of para
meters

xgb_scores <- pmap(param_sets, function(eta, max_depth) { # test each combination i
n turn
  xgb_model <- xgboost(param=list(eta=eta, max_depth=max_depth),
                       data=as.matrix(credit_trn), label=default_trn,
                       nrounds=100, early_stopping_rounds=10,
                       objective="binary:logistic", eval_metric="auc",
                       verbose=0)
  xgb_preds <- predict(xgb_model, newdata=as.matrix(credit_val))
  return( as.numeric(auc(default_val, xgb_preds)) )
}) # takes a few minutes to run!
```

In real-world applications, the author prefers to use R’s `mlr` package, which handles a lot of the boilerplate for you.

Let’s take a look at the results.

```
filled.contour(x=param_options$eta, y=param_options$max_depth, z=matrix(xgb_scores,
nrow=length(param_options$eta)))
```



```
best_params <- as.list(param_sets[match(max(as.numeric(xgb_scores)),xgb_scores),])
print(paste("Best eta:", best_params$eta))
```

```
## [1] "Best eta: 0.3"
```

```
print(paste("Best max_depth:", best_params$max_depth))
```

```
## [1] "Best max_depth: 2"
```

```
print(paste("Best score:", max(as.numeric(xgb_scores))))
```

```
## [1] "Best score: 0.779132021332504"
```

The best hyperparameter pair appears to be eta=0.03 / max_depth=2, with AUC of 0.779 on the validation dataset.

Retrain with the best hyperparameters.

```
xgb_model_tuned <- xgboost(param=list(eta=best_params$eta, max_depth=best_params$ma
x_depth),
                     data=as.matrix(credit_train), label=default_train,
                     nrounds=100, early_stopping_rounds=10,
                     objective="binary:logistic", eval_metric="auc",
                     verbose=0)
```

Determine tuned model score.

```
xgb_preds_tuned <- predict(xgb_model_tuned, newdata=as.matrix(credit_test))
table(xgb_preds_tuned >= 0.5, default_test == 1)
```

```
## 
##          FALSE TRUE
##   FALSE  6695 1269
##   TRUE    353  683
```

```
print(paste("Tuned XGBoost score:", auc(default_test, xgb_preds_tuned))) # 0.779
```

```
## [1] "Tuned XGBoost score: 0.778736243336093"
```

This is a small but healthy improvement over the un-tuned score.