Part 6 - Holdout Predictions

Ibrahim Yazici

Overview

This report demonstrates how to read in the hold-out test, make predictions, and organize the predictions in the necessary format. The compiled predictions are then saved to a CSV file.

This report uses `caret`.

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.3     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

Read training data

The training data set is read in the code chunk below assuming you have downloaded the data from Canvas.

```
df <- readr::read_csv("paint_project_train_data.csv", col_names = TRUE)
```

```
## Rows: 835 Columns: 8
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): Lightness, Saturation
## dbl (6): R, G, B, Hue, response, outcome
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Regression problem

The data associated with the regression task is created below. Note that the logit-transformed response is calculated and assigned to the variable `y`.

```
dfii <- df %>%
  mutate(y = boot::logit( (response - 0) / (100 - 0) ) ) %>%
  select(R, G, B,
         Lightness, Saturation, Hue,
         y)
```

Let's train and assess a linear model with `caret`. We add Categorical Inputs to Interactions from 3 DOF spline from input R and All Pairwise Interactions of Continuous Inputs G, B, Hue (This is our best performing model in regression).

We will use 5-fold cross-validation with 3 repeats.

```
my_ctrl_regress <- trainControl(method = 'repeatedcv', number = 5, repeats = 3)
```

Next, define the primary performance metric of the model.

```
my_metrics_regress <- 'RMSE'
```

Let's now train and assess our linear model.

```
set.seed(2023)

mod_regress <- train(y ~ splines::ns(R, 3) * (G + B + Hue)^2 + Lightness + Saturation,
                     data = dfii,
                     method = "lm",
                     metric = my_metrics_regress,
                     preProcess = c("center", "scale"),
                     trControl = my_ctrl_regress)
```

Classification problem

The data associated with the binary classification task are assembled below. The binary outcome is set as a `factor` variable with levels `'event'` and `'non_event'` with the first level set to be `'event'`. This is the format required by `caret` and is the same data created for part iiiD) for the project.

```
dfiiiD <- df %>%
  select(-response) %>%
  mutate(outcome = ifelse(outcome == 1, 'event', 'non_event'),
         outcome = factor(outcome, levels = c('event', 'non_event')))
```

The `caret` package requires specifying a primary performance. It can only tune models for one type of metric at a time. This means that we must train and tune a model twice in order to consider the impact of tuning for maximizing Accuracy vs tuning for maximizing ROC AUC. This is unfortunate, but is just how `caret` is constructed to operate. This report first sets up training and tuning for Accuracy and then repeats the training/tuning a second time to maximize ROC AUC.

Accuracy

The code chunk below specifies the resampling scheme that we will use for the model associated with the Accuracy metric.

```
my_ctrl_acc <- trainControl(method = 'repeatedcv', number = 5, repeats = 3)
```

Next, define the primary performance metric.

```r
my_metrics_acc <- "Accuracy"
```

Let's now train and assess our model.

```r
set.seed(2022)

mod_binary_acc <- train(outcome ~ .,
                        data = dfiiiD,
                        method = "xgbTree",
                        metric = my_metrics_acc,
                        trControl = my_ctrl_acc,
                        verbosity = 0,
                        nthread = 1)
```

ROC AUC

Next, let's setup the resampling scheme control options associated with maximizing the ROC AUC. We do not need to modify the resampling itself, we can use the same 5-fold cross-validation with 3 repeats that we used previously. However, `caret` requires that we modify how the predictions will be stored and summarized in order to calculate the ROC AUC. The metric is also set such that `caret` will calculate the ROC AUC.

```r
my_ctrl_roc <- trainControl(method = 'repeatedcv', number = 5, repeats = 3,
                            summaryFunction = twoClassSummary,
                            classProbs = TRUE,
                            savePredictions = TRUE)

my_metrics_roc <- 'ROC'
```

Let's now train and assess our model.

```r
set.seed(2022)

mod_binary_roc <- train(outcome ~ .,
                        data = dfiiiD,
                        method = "xgbTree",
                        metric = my_metrics_roc,
                        trControl = my_ctrl_roc,
                        verbosity = 0,
                        nthread = 1)
```

Hold-out set predictions

Now that we have trained two models, it's time to setup the predictions! First, the hold-out test set is loaded in the code chunk below.

```r
holdout <- readr::read_csv('paint_project_holdout_data.csv', col_names = TRUE)
```

```
## Rows: 844 Columns: 6
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): Lightness, Saturation
## dbl (4): R, G, B, Hue
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Note that the holdout test set only consists of input variables! The number of columns is therefore different from the `df` object!

```
sprintf("columns in df: %d vs columns in holdout: %d", ncol(df), ncol(holdout))
```

```
## [1] "columns in df: 8 vs columns in holdout: 6"
```

Displaying the names of the `holdout` tibble shows that the `response` and `outcome` columns are NOT present!

```
holdout %>% names()
```

```
## [1] "R"          "G"          "B"          "Lightness"  "Saturation"
## [6] "Hue"
```

The `holdout` tibble therefore only consists of inputs.

It is easy to make predictions with `caret` trained models. We simply call the `predict()` function! The first argument to `predict()` is the model object and the second argument is the data we wish to predict. Let's start by making predictions with the regression model. As shown below the result is a numeric vector!

```
predict(mod_regress, holdout) %>% class()
```

```
## [1] "numeric"
```

The length of the returned vector is equal to the number of rows in the test set.

```
length( predict(mod_regress, holdout) ) == nrow(holdout)
```

```
## [1] TRUE
```

It should be noted though, that predicting a different data set will result in a different number of predictions. For example, if we predicted the training set, the length of the returned predictions equals the number of rows in the training set!

```
length( predict(mod_regress, df) ) == nrow(df)
```

```
## [1] TRUE
```

The values of the returned predicted vector are the predictions of the continuous output. However, in lecture we learned that these are really the trend or predictions of the **mean** response! The head of the predicted vector is displayed below to show a few of the values. **IMPORTANT**: please remember that the regression models are predicting the LOGIT-TRANSFORMED `response`. Thus, the predictions provided by the `predict()` function are the MEAN or EXPECTED logit-transformed `response`!

```
predict( mod_regress, holdout ) %>% head()
```

```
##            1            2            3            4            5            6 
##  0.827752130  1.628962881  0.005202726 -0.353617523  0.234051233  0.115945894
```

Next, let's make predictions of the binary output, `outcome`. As with regression, `caret` trained classification models are "complete" models. We can make predictions with them! Classification model objects can return several types of predictions. The default option from `caret` is different from the default option from `glm()`. By default, `caret` predictions return the outcome class level or label. The returned object is a regular vector associated with the `factor` data type. The code chunk below makes predictions with the logistic regression model trained using the Accuracy resampling control options. The second argument to `predict()`, the `newdata` argument, is explicitly named in the `predict()` call. We will see why it is useful to use the name of the argument shortly. The predictions are pipped to the `class()` function to show the vector is a `factor` data type.

```
predict(mod_binary_acc, newdata = holdout) %>% class()
```

```
## [1] "factor"
```

The `factor` data type is `R`'s categorical data type. We can check the finite values, levels, labels, or categories using the `levels()` function. Notice that our predictions have just two levels, `'event'` and `'non_event'`. Thus, by default the `predict()` function returns the classifications rather than the predicted probability!

```
predict(mod_binary_acc, newdata = holdout) %>% levels()
```

```
## [1] "event"     "non_event"
```

The number of elements in the vector equals the number of rows in the test set.

```
length( predict(mod_binary_acc, newdata = holdout) ) == nrow(holdout)
```

```
## [1] TRUE
```

The same holds true whether we use the `caret` object associated with maximizing Accuracy or maximizing ROC AUC.

```
length( predict(mod_binary_roc, newdata = holdout) ) == nrow(holdout)
```

```
## [1] TRUE
```

The `caret` object associated with maximizing ROC AUC by default also returns the classifications.

```
predict(mod_binary_roc, newdata = holdout) %>% levels()
```

```
## [1] "event"     "non_event"
```

The first few elements of the `caret` trained logistic regression models are printed to the screen below.

```
predict(mod_binary_acc, newdata = holdout) %>% head()
```

```
## [1] event event event event event event
## Levels: event non_event
```

```
predict(mod_binary_roc, newdata = holdout) %>% head()
```

```
## [1] event event event event event event
## Levels: event non_event
```

The predictions of the two models are the same.

```
all.equal(predict(mod_binary_acc, newdata = holdout),
          predict(mod_binary_roc, newdata = holdout))
```

```
## [1] TRUE
```

The returned predicted classifications assume the default threshold of 50%. We are also interested in knowing the predicted event probability. We can instruct a `caret` trained model to return the probability associated with each class by setting the `type` argument to `type='prob'` within the `predict()` call. However, by doing so the result is no longer a regular vector! Instead, a data.frame is returned!

```
predict(mod_binary_acc, newdata = holdout, type = 'prob') %>% class()
```

```
## [1] "data.frame"
```

This is the case with the `caret` object associated with maximizing the ROC AUC as well.

```
predict(mod_binary_roc, newdata = holdout, type = 'prob') %>% class()
```

```
## [1] "data.frame"
```

The returned datatypes because the predicted probability for each class is provided. This allows `caret` to scale to multi-class situations when there are more than 2 classes associated with the categorical output. The head of the probability predictions are shown below.

```r
predict(mod_binary_acc, newdata = holdout, type = 'prob') %>% head()
```

```
##       event   non_event
## 1 0.9829868 0.017013192
## 2 0.8537184 0.146281600
## 3 0.8447922 0.155207753
## 4 0.9952430 0.004756987
## 5 0.9032172 0.096782804
## 6 0.9904079 0.009592116
```

Compile predictions

We must upload your hold-out set predictions to an RShiny app. This app will calculate the performance of your models on the hold-out test set. We will organize your predictions into a single tibble with the following column names:

`id, y, outcome, probability`

The `id` column is a row index for the predictions, the `y` is the logit-transformed continuous output, the `outcome` column is the binary output level, and the `probability` column is the event probability.

The code chunk below organizes the hold-out test set predictions.

```r
my_preds <- tibble::tibble(
  y = predict(mod_regress, newdata = holdout),
  outcome = predict(mod_binary_acc, newdata = holdout)
) %>%
  bind_cols(
    predict(mod_binary_acc, newdata = holdout, type = 'prob') %>%
      select(probability = event)
  ) %>%
  tibble::rowid_to_column('id')
```

A glimpse of the predictions is shown below.

```r
my_preds %>% glimpse()
```

The head of the compiled predictions is shown below.

```r
my_preds %>% head()
```

```
## # A tibble: 6 x 4
##      id        y outcome probability
##   <int>    <dbl> <fct>         <dbl>
## 1     1  0.828   event         0.983
## 2     2  1.63    event         0.854
## 3     3  0.00520 event         0.845
## 4     4 -0.354   event         0.995
## 5     5  0.234   event         0.903
## 6     6  0.116   event         0.990
```

The compiled hold-out test set predictions are saved to a CSV file in the code chunk below.

```r
my_preds %>%
  readr::write_csv('holdout_set_preds.csv', col_names = TRUE)
```