Blood Donation Classification Prediction

using Cross Validation with Random Forest

Andey Nunes

March 07, 2019

Data Description

Our group is entered into a competitiion for a classification problem: https://www.drivendata.org/competitions/2/warm-up-predict-blood-donations/

> The business question is: using data about student blood donations, can we predict if a student will donate blood next time?

The competition specifies the training and test files, so we do not need to set aside test data. First we load and inspect the training set for structure, variable types, and pairwise correlations.

```
training <- read_csv("projectdata.csv")
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
## Parsed with column specification:
## cols(
##   X1 = col_double(),
##   `Months since Last Donation` = col_double(),
##   `Number of Donations` = col_double(),
##   `Total Volume Donated (c.c.)` = col_double(),
##   `Months since First Donation` = col_double(),
##   `Made Donation in March 2007` = col_double()
## )
```

```
glimpse(training)
```

```
## Observations: 576
## Variables: 6
## $ X1                            <dbl> 619, 664, 441, 160, 358, 335, 47...
## $ `Months since Last Donation`  <dbl> 2, 0, 1, 2, 1, 4, 2, 1, 5, 0, 2,...
## $ `Number of Donations`         <dbl> 50, 13, 16, 20, 24, 4, 7, 12, 46...
## $ `Total Volume Donated (c.c.)` <dbl> 12500, 3250, 4000, 5000, 6000, 1...
## $ `Months since First Donation` <dbl> 98, 28, 35, 45, 77, 4, 14, 35, 9...
## $ `Made Donation in March 2007` <dbl> 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1,...
```

The variable `X1` has 576 unique entries and is therefore likely to be a student or patient ID number. We can ignore it for modeling purposes. I'm also going to shorten the variable names as follows:

- `Months since Last Donation` is renamed `recency` to indicate how many months ago the last donation event was
- `Months since First Donation` is renamed `time` to indicate the total time span in months since the first donation event
- `Number of Donations` is renamed `frequency`
- `Total Volume Donated (c.c.)` is renamed `volume` the original data set library makes note that this field indicates the monetary measure being used for the business case
- `Made Donation in March 2007` is renamed `target` and is the binary variable that we are trying to predict/classify

** Correlation Matrix of Blood Donation Data **

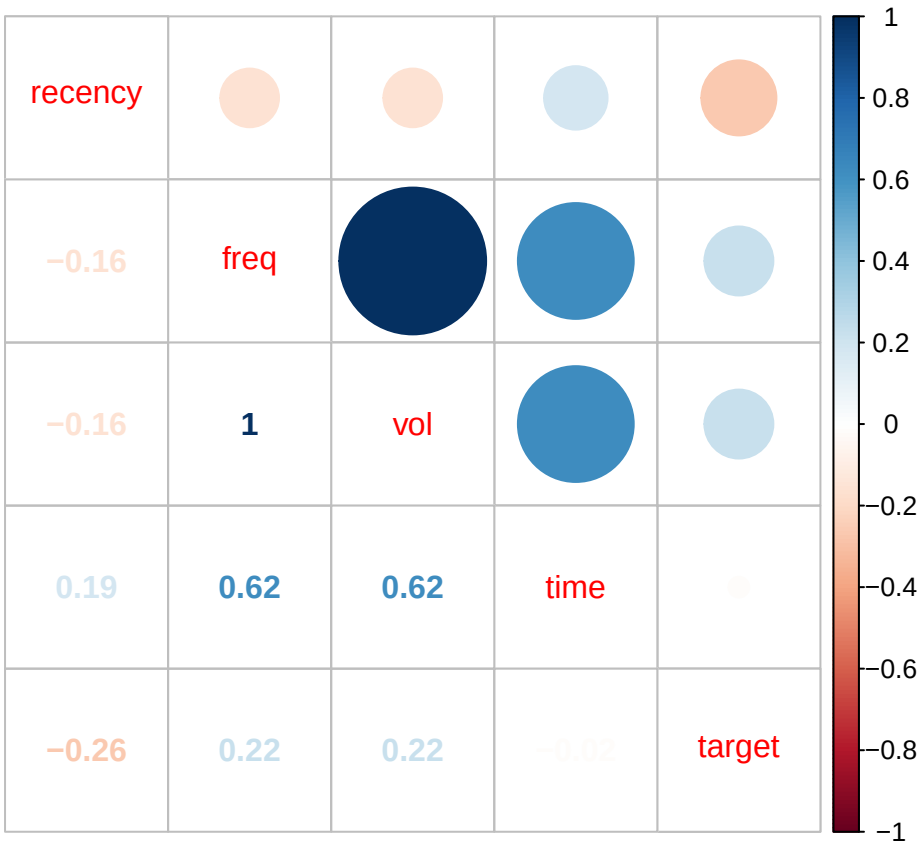
```
training <- training[,-1]

names(training) <- c("recency", "freq", "vol", "time", "target")

training_mat <- data.matrix(training, rownames.force = T)

corr_mat <- cor(training_mat)

corrplot.mixed(corr_mat)
```



Here we can see that frequency of donation and the total blood volume are perfectly positively correlated. We can therefore ignore the total blood volume variable since all of the information will be captured and passed along to our algorithms in the using the variables measured in months and the donation frequency.

```
training <- select(training, -vol)
```

Cross Validation Folds & Splits

The workflow and code used here is adapted from the *Machine Learning in the Tidyverse* course by Dmitriy Gorenshteyn (DataCamp 2019). The approach involves using list-columns to store list objects (such as model output) in a tibble, which is a special R data frame. These lists can then be iterated over with specialized `map` functions from the `purrr` package to calculate and extract information.

Using the training data, we will create a 5-fold cross-validation split tibble. This will set us up to itertively generate multiple models on our new train/validation subsets.

```
# using the rsample library
cv_split <- vfold_cv(training, v = 5)
# cv_split  # uncomment the front of this line if you want to preview
```

The `cv_split` object has five rows and two columns. The first column, `splits`, is a list column containing the training and validation data. The second column is a character vector containing the fold id generated by the `vfold_cv()` function. We can iterate over the `splits` column and extract the train and validation columns

```
cv_data <- cv_split %>%
   mutate(train = map(splits, ~training(.x)),
          validate = map(splits, ~testing(.x)))
glimpse(cv_data)
```

```
## Observations: 5
## Variables: 4
## $ splits   <list> [<rsplit[460 x 116 x 576 x 4]>, <rsplit[461 x 115 x ...
## $ id       <chr> "Fold1", "Fold2", "Fold3", "Fold4", "Fold5"
## $ train    <list> [<tbl_df[460 x 4]>, <tbl_df[461 x 4]>, <tbl_df[461 x...
## $ validate <list> [<tbl_df[116 x 4]>, <tbl_df[115 x 4]>, <tbl_df[115 x...
```

We've just created two new list columns containing the data that we can now train and validate models over.

Model preparation

linear model

```
cv_models_lm <- cv_data %>%
   mutate(lm_model = map(train, ~lm(formula = target~., data = .x)))
```

other models

use the linear model and random forest model code chunks as examples for making another object holding the cross-validation data sets and the specified models.

random forest

```
# set forest parameter
n_trees <- 500
# Build a random forest model for each fold
cv_models_rf <- cv_data %>%
  mutate(rf_model = map(train, ~ranger(formula = target~., data = .x,
                                       num.trees = n_trees,)))
```

Model evaluation

The *cross-validation model evaluation process* follows the same general set of steps iterated over each fold: 1. extract the actual target values from the validation set 2. use the models to make target predictions that will

be compared to the actual target 3. for each fold, calculate the Mean Absolute Error (`MAE`) where

$$MAE = \frac{\sum_{i=1}^{n} |Actual_i - Predicted_i|}{n}$$

4. Take the average over all MAE values to determine which model performs best on these sets of training & validation splits

In the following subsections, the eval code chunk follows a similar pattern:

- the *actual* and *predicted* values are extracted from the validation sets into a `cv_prep_`

Lets see how other models and randocompared.

linear model

```
# extract actual values
cv_prep_lm <- cv_models_lm %>%
   mutate(validate_actual = map(validate, ~.x$target),
          validate_predicted = map2(.x = lm_model, .y = validate,
                                    ~predict(.x, .y)))
```

```
# the function mae() is from library(Metrics)
# Calculate the mean absolute error for each validate fold
cv_eval_lm <- cv_prep_lm %>%
  mutate(validate_mae = map2_dbl(.x = validate_actual,
                                 .y = validate_predicted,
                                 ~mae(actual = .x, predicted = .y)))

# Print the validate_mae column
cv_eval_lm$validate_mae
```

```
##      1      2      3      4      5
## 0.3233 0.3242 0.3536 0.3350 0.3155
```

The average mean absolute error across all linear models was 0.3303.

other models

use the linear model and random forest model code chunks as examples for making another object holding the cross-validation data sets and the specified models.

random forest

```
# Generate predictions using the random forest model
cv_prep_rf <- cv_models_rf %>%
  mutate(validate_actual = map(validate, ~.x$target),
         validate_predicted = map2(.x = rf_model, .y = validate, ~predict(.x, .y)$predictions))
```

```
# Calculate validate MAE for each fold
cv_eval_rf <- cv_prep_rf %>%
  mutate(validate_mae = map2_dbl(validate_actual, validate_predicted, ~mae(actual = .x, predicted = .y)))

# Print the validate_mae column
cv_eval_rf$validate_mae
```

```
##      1      2      3      4      5
## 0.3048 0.3116 0.3001 0.3101 0.2878
```

```
# Calculate the mean of validate_mae column
mean(cv_eval_rf$validate_mae)
```

```
## [1] 0.3029
```

Tune hyper-parameters

```
# Prepare for tuning cross validation folds by varying mtry
cv_tune <- cv_data %>%
   crossing(mtry = 1:3)

# Build a model for each fold & mtry combination
cv_model_tunerf <- cv_tune %>%
  mutate(rf_model =
           map2(.x = train, .y = mtry,
                ~ranger(formula = target~., data = .x,
                        mtry = .y, num.trees = n_trees)))

glimpse(cv_model_tunerf)
```

```
## Observations: 15
## Variables: 6
## $ splits   <list> [<rsplit[460 x 116 x 576 x 4]>, <rsplit[460 x 116 x ...
## $ id       <chr> "Fold1", "Fold1", "Fold1", "Fold2", "Fold2", "Fold2",...
## $ train    <list> [<tbl_df[460 x 4]>, <tbl_df[460 x 4]>, <tbl_df[460 x...
## $ validate <list> [<tbl_df[116 x 4]>, <tbl_df[116 x 4]>, <tbl_df[116 x...
## $ mtry     <int> 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3
## $ rf_model <list> [<0.201877, 0.267314, 0.570314, 0.631848, 0.661379, ...
```

```
# Generate validate predictions for each model
cv_prep_tunerf <- cv_model_tunerf %>%
  mutate(rf_validate_actual = map(validate, ~.x$target),
     rf_validate_predicted = map2(.x = rf_model, .y = validate, ~predict(.x, .y)$predictions))

# Calculate validate MAE for each fold and mtry combination
cv_eval_tunerf <- cv_prep_tunerf %>%
  mutate(rf_validate_mae = map2_dbl(.x = rf_validate_actual, .y = rf_validate_predicted, ~mae(actual =

# Calculate the mean validate_mae for each mtry used
cv_eval_tunerf %>%
  group_by(mtry) %>%
  summarise(rf_mean_mae = mean(rf_validate_mae))
```

```
## # A tibble: 3 x 2
##    mtry rf_mean_mae
##   <int>       <dbl>
## 1     1       0.303
## 2     2       0.300
## 3     3       0.301
```

Select random forest model with the lowest average mae.

```
best <- filter(cv_eval_tunerf, rf_validate_mae == min(cv_eval_tunerf$rf_validate_mae))
kable(tibble(id = best$id, mtry = best$mtry))
```

id	mtry
Fold5	2

Use these parameters to traing our best rf model.

```
# Build the model using all training data and the best performing parameter
best_model <- ranger(formula = target~., data = training,
                     mtry = best$mtry, num.trees = n_trees)
```

Model Output on Test Data

```
test_data <- read_csv("project test data.csv")
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
## Parsed with column specification:
## cols(
##   X1 = col_double(),
##   `Months since Last Donation` = col_double(),
##   `Number of Donations` = col_double(),
##   `Total Volume Donated (c.c.)` = col_double(),
##   `Months since First Donation` = col_double()
## )
```

```
names(test_data) <- c("id", "recency", "freq", "vol", "time")
# remove id
test <- select(test_data,-id, -vol)
```

```
# Predict life_expectancy for the testing_data
test_predicted <- predict(best_model, test)$predictions
rf_test_prediction <- cbind(test_data$id, test_predicted)
```

Discussion

References

Data is courtesy of Yeh, I-Cheng via the UCI Machine Learning repository (https://archive.ics.uci.edu/ml/datasets/Blood+Transfusion+Service+Center)

https://archive.ics.uci.edu/ml/machine-learning-databases/blood-transfusion/transfusion.names

Code examples were borrowed from DataCamp course material presented by Dmitriy Gorenshteyn, “Machine Learning in the Tidyverse” https://www.datacamp.com/courses/machine-learning-in-the-tidyverse

Appendix

```
kable(rf_test_prediction)
```

	test_predicted
659	0.6560
276	0.0916
263	0.0949
303	0.1577
83	0.4062
500	0.7503
530	0.3464
244	0.0039
249	0.0412
728	0.0083
129	0.6609
534	0.0538
317	0.1891
401	0.0949
696	0.5771
192	0.2137
176	0.0220
571	0.5404
139	0.1115
423	0.1817
563	0.9306
56	0.5721
528	0.5254
101	0.0949
467	0.0532
382	0.2061
466	0.3980
294	0.0039
512	0.1427
659	0.6560
389	0.1769
487	0.2360
701	0.1701
419	0.0542
536	0.1336
240	0.0357
508	0.0752
515	0.0039
283	0.3135
650	0.0949
65	0.4895
228	0.4427
741	0.0934
297	0.3001
464	0.0352
63	0.1701
231	0.3861
28	0.0686
248	0.0039
357	0.1022
300	0.4830
726	0.7164

	test_predicted
680	0.2304
520	0.7066
254	0.0293
582	0.3021
143	0.1099
98	0.8073
1	0.1108
221	0.3238
352	0.0949
64	0.2078
138	0.2030
745	0.3858
64	0.2078
688	0.0532
623	0.0615
289	0.1214
174	0.3663
690	0.0949
105	0.1803
427	0.1361
48	0.1055
14	0.0949
657	0.0063
301	0.0518
455	0.1411
579	0.2300
722	0.0548
98	0.8073
491	0.1055
303	0.1577
466	0.3980
65	0.4895
300	0.4830
9	0.0686
622	0.2137
323	0.0597
289	0.1214
568	0.1847
290	0.0949
156	0.3084
464	0.0352
426	0.0364
306	0.1747
4	0.3604
12	0.3210
187	0.2242
406	0.0647
96	0.3464
509	0.1701
733	0.0475
548	0.0264
478	0.3190

	test_predicted
501	0.1258
127	0.5613
199	0.0747
299	0.7218
162	0.0949
235	0.1708
23	0.0259
473	0.0949
487	0.2360
683	0.8202
303	0.1577
309	0.0172
569	0.0710
34	0.7018
686	0.6022
84	0.0069
733	0.0475
537	0.5518
181	0.1038
453	0.3145
67	0.4001
161	0.2415
307	0.1528
703	0.0384
181	0.1038
246	0.1574
316	0.3141
278	0.2137
346	0.3176
545	0.0949
419	0.0542
694	0.3464
622	0.2137
663	0.0641
262	0.5580
461	0.4494
373	0.6480
233	0.0949
466	0.3980
207	0.1099
263	0.0949
16	0.0710
513	0.0074
449	0.4238
429	0.0000
701	0.1701
632	0.0309
529	0.4161
245	0.0704
344	0.3937
353	0.3818
241	0.3884

	test_predicted
633	0.0000
624	0.1701
726	0.7164
189	0.2137
138	0.2030
402	0.0039
511	0.2808
590	0.2775
334	0.2264
447	0.0389
119	0.3980
389	0.1769
644	0.0949
423	0.1817
131	0.0710
405	0.1142
82	0.0949
643	0.6526
156	0.3084
617	0.0154
574	0.5933
272	0.2444
613	0.0511
545	0.0949
685	0.0039
570	0.0640
537	0.5518
691	0.0000
85	0.4387
483	0.1574
455	0.1411
93	0.2137
744	0.1701
33	0.1574
321	0.7583
523	0.3578
426	0.0364
196	0.2392
301	0.0518
103	0.0000
224	0.0361
454	0.0681
585	0.0058
154	0.0686