Blood Donation Classification Prediction

using Cross Validation with Random Forest

Andey Nunes

3/2/2019

Data Description

Our group is entered into a competitiion for a classification problem: https://www.drivendata.org/competitions/2/warm-up-predict-blood-donations/ (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, 1…
## $ `Number of Donations`          <dbl> 50, 13, 16, 20, 24, 4, 7, 12, 46, …
## $ `Total Volume Donated (c.c.)` <dbl> 12500, 3250, 4000, 5000, 6000, 100…
## $ `Months since First Donation` <dbl> 98, 28, 35, 45, 77, 4, 14, 35, 98,…
## $ `Made Donation in March 2007` <dbl> 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0…
```

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 **

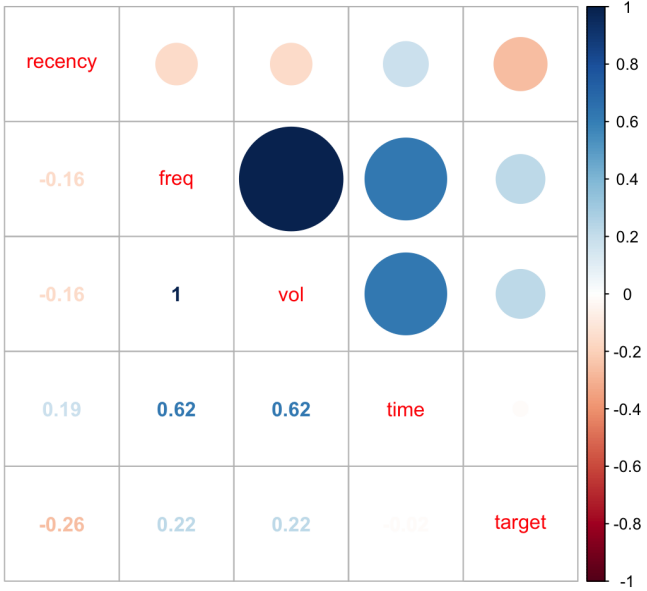
```
training <- training[,-1]

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

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 57…
## $ id       <chr> "Fold1", "Fold2", "Fold3", "Fold4", "Fold5"
## $ train    <list> [<tbl_df[460 x 4]>, <tbl_df[461 x 4]>, <tbl_df[461 x 4…
## $ validate <list> [<tbl_df[116 x 4]>, <tbl_df[115 x 4]>, <tbl_df[115 x 4…
```

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)))
```

knn ?

random forest

```
# 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 = 100, seed = 123)))
```

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

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
## 0.32 0.32 0.35 0.33 0.32
```

The average mean absolute error across all linear models was 0.33.

Lets see how other models compared.

knn ?

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
## 0.30 0.31 0.30 0.31 0.29
```

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

```
## [1] 0.3
```

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 = 100, seed = 123)))

glimpse(cv_model_tunerf)
```

```
## Observations: 15
## Variables: 6
## $ splits   <list> [<rsplit[460 x 116 x 576 x 4]>, <rsplit[460 x 116 x 57…
## $ id       <chr> "Fold1", "Fold1", "Fold1", "Fold2", "Fold2", "Fold2", "…
## $ train    <list> [<tbl_df[460 x 4]>, <tbl_df[460 x 4]>, <tbl_df[460 x 4…
## $ validate <list> [<tbl_df[116 x 4]>, <tbl_df[116 x 4]>, <tbl_df[116 x 4…
## $ mtry     <int> 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3
## $ rf_model <list> [<0.2360, 0.4114, 0.5433, 0.7181, 0.6184, 0.5692, 0.67…
```

```
# 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 = .x, predic
ted = .y)))

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

mtry <int>	rf_mean_mae <dbl>
1	0.3
2	0.3
3	0.3

3 rows

Selected random forest model with the lowest average mae

```
best <- filter(cv_eval_tunerf, rf_validate_mae == min(cv_eval_tunerf$rf_validate_mae))
```

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 = 3, num.trees = 100, seed = 123)
```

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_data <- test_data[,-1]
```

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

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/datasets/Blood+Transfusion+Service+Center))

https://archive.ics.uci.edu/ml/machine-learning-databases/blood-transfusion/transfusion.names (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 (https://www.datacamp.com/courses/machine-learning-in-the-tidyverse)

Appendix