Lab No. 11

Elizabeth Walter

```
knitr::opts_chunk$set(echo = TRUE,
                      fig.align = 'center')
```

```
library(caret)
library(tidyverse)
bank <- read.table("https://raw.githubusercontent.com/ajkirkpatrick/FS20/Spring2021/classdata/bank.csv"
                 header = TRUE,
                 sep = ",")
bank <- bank %>%
  mutate(y = as.factor(y),
         job = as.factor(job),
         marital = as.factor(marital),
         education = as.factor(education),
         default = as.factor(default),
         housing= as.factor(housing),
         loan= as.factor(loan),
         unemp = ifelse(job == "unemployed", 1, 0))
```

1. Split the data into an 80/20 train vs. test split. Make sure you explicitly set the seed for replicability, but do not share your seed with others in the class.

```
set.seed(21)

train_index = sample(nrow(bank), size = trunc(0.80 * nrow(bank)))
bank_trn = bank[train_index, ] #3616 rows
bank_tst = bank[-train_index, ] #905 rows
```

2. Run a series of logistic regressions with between 1 and 4 predictors of your choice (you can use interactions).

```
# duration, previous, age, job, education, housing, loan
fit_1 <- glm(y ~ duration, data = bank_trn, family = 'binomial')
fit_2 <- glm(y ~ duration + job, data = bank_trn, family = 'binomial')
fit_3 <- glm(y ~ duration + job + housing, data = bank_trn, family = 'binomial')
fit_4 <- glm(y ~ duration + job*housing, data = bank_trn, family = 'binomial')
```

3. Create eight total confusion matrices: four by applying your models to the training data, and four by applying your models to the test data. Briefly discuss your findings. How does the error rate, sensitivity, and specificity change as the number of predictors increases?

I changed the predictors used in my models a few times to try and reduce error rate and increase sensitivity and specificity. I focused on my first predictor in my one predictor model, as the first few tries it produced a sensitivity of 0.00 and some did not produce any 'yes' results and therefore could not produce a confusion matrix. My accuracy/error rate did not change much at all across predictors chosen or models, but I was able to see changes in sensitivity and small increases in specificity.

For the train data, the error decreases very slightly as the number of predictors increased until the model with 4 predictors, in which the error increases from the previous model's. The changes in error are similarly as small in the train data, but we see an increase in error rate from the first to second model, and then it decreases in the last two.

The sensitivity is incredibly low across all model complexities for both train and test data. For both the test and train data, the single predictor model has the lowest sensitivity, and increases very slightly from the 1- to 2-predictor and 2- to 3-predictor models and then does not increase further with the 4-predictor model, and even decreases in the test data. This indicates that there are few true positives - or many false negatives - with the single predictor model and while it does not get significantly better with the other models, the 3-predictor model may be the best fit for predictions in regards to sensitivity.

The specificity is high (greater than 0.98) for all models on both the train and test data. This means that there is a small number of false positives (classes predicted as positive that are actually negative). In the train data, it is highest in 1-predictor model and lowest in the 4-predictor model, while in the test data it is highest in the 4-predictor model.

As mentioned in the lab 11 intro, defaults are relatively uncommon, and this is reflected in the data, where the positives occurances of y make up 11.7% and 10.9% in the train and test data, respectfully. We must be mindful of a low prevalence of positive cases when evaluating accuracy, as a bad classifier that misclassifies all of the true postives as negatives will still give a low error rate. Here, our error rates are not very different than the proportion of "yes" cases in the data, so I would not say they are great classifiers. But more importantly, as a bank suffers more from misclassifying someone who will default as someone who will not than the other way around, I believe the sensitivity is the more important metric to focus on here.

```r
# get classifications
get_predicted <- function(model, data){
  model_glm_pred = ifelse(predict(model, data, type = "response") > .5, "yes", "no")
}

# Train
train_1_pred <- get_predicted(fit_1, bank_trn)
train_2_pred <- get_predicted(fit_2, bank_trn)
train_3_pred <- get_predicted(fit_3, bank_trn)
train_4_pred <- get_predicted(fit_4, bank_trn)

train_tab_1 <- table(predicted = train_1_pred, actual = bank_trn$y)
train_tab_2 <- table(predicted = train_2_pred, actual = bank_trn$y)
train_tab_3 <- table(predicted = train_3_pred, actual = bank_trn$y)
train_tab_4 <- table(predicted = train_4_pred, actual = bank_trn$y)

train_conf_mat_1 <- confusionMatrix(train_tab_1, positive = "yes")
train_conf_mat_2 <- confusionMatrix(train_tab_2, positive = "yes")
train_conf_mat_3 <- confusionMatrix(train_tab_3, positive = "yes")
train_conf_mat_4 <- confusionMatrix(train_tab_4, positive = "yes")
```

```r
#Test
test_1_pred <- get_predicted(fit_1, bank_tst)
test_2_pred <- get_predicted(fit_2, bank_tst)
test_3_pred <- get_predicted(fit_3, bank_tst)
test_4_pred <- get_predicted(fit_4, bank_tst)

test_tab_1 <- table(predicted = test_1_pred, actual = bank_tst$y)
test_tab_2 <- table(predicted = test_2_pred, actual = bank_tst$y)
test_tab_3 <- table(predicted = test_3_pred, actual = bank_tst$y)
test_tab_4 <- table(predicted = test_4_pred, actual = bank_tst$y)

test_conf_mat_1 <- confusionMatrix(test_tab_1, positive = "yes")
test_conf_mat_2 <- confusionMatrix(test_tab_2, positive = "yes")
test_conf_mat_3 <- confusionMatrix(test_tab_3, positive = "yes")
test_conf_mat_4 <- confusionMatrix(test_tab_4, positive = "yes")

# Compare
metrics <- rbind(

  c(1 - train_conf_mat_1$overall["Accuracy"], #get error rate, but will still be labeled as "accuracy"
    train_conf_mat_1$byClass["Sensitivity"],
    train_conf_mat_1$byClass["Specificity"]),

  c(1 - train_conf_mat_2$overall["Accuracy"],
    train_conf_mat_2$byClass["Sensitivity"],
    train_conf_mat_2$byClass["Specificity"]),

  c(1 - train_conf_mat_3$overall["Accuracy"],
    train_conf_mat_3$byClass["Sensitivity"],
    train_conf_mat_3$byClass["Specificity"]),

  c(1 - train_conf_mat_4$overall["Accuracy"],
    train_conf_mat_4$byClass["Sensitivity"],
    train_conf_mat_4$byClass["Specificity"]),

  c(1 - test_conf_mat_1$overall["Accuracy"],
    test_conf_mat_1$byClass["Sensitivity"],
    test_conf_mat_1$byClass["Specificity"]),

  c(1 - test_conf_mat_2$overall["Accuracy"],
    test_conf_mat_2$byClass["Sensitivity"],
    test_conf_mat_2$byClass["Specificity"]),

  c(1 - test_conf_mat_3$overall["Accuracy"],
    test_conf_mat_3$byClass["Sensitivity"],
    test_conf_mat_3$byClass["Specificity"]),

  c(1 - test_conf_mat_4$overall["Accuracy"],
    test_conf_mat_4$byClass["Sensitivity"],
    test_conf_mat_4$byClass["Specificity"])

)
```

```r
rownames(metrics) = c("train1", "train2", "train3", "train4", "test1", "test2", "test3", "test4")
```

```r
metrics
```

```
##         Accuracy Sensitivity Specificity
## train1 0.1120022   0.1587678   0.9843456
## train2 0.1100664   0.1777251   0.9840326
## train3 0.1086836   0.1895735   0.9840326
## train4 0.1097898   0.1895735   0.9827802
## test1  0.1116022   0.1212121   0.9826303
## test2  0.1127072   0.1313131   0.9801489
## test3  0.1082873   0.1616162   0.9813896
## test4  0.1060773   0.1515152   0.9851117
```

```r
table(bank_trn$y) / nrow(bank_trn)
```

```
##
##        no       yes
## 0.8832965 0.1167035
```

```r
table(bank_tst$y) / nrow(bank_tst)
```

```
##
##        no       yes
## 0.8906077 0.1093923
```

4. Compare your results from Lab 11 to those of Lab 10 in a few sentences.

With the models from this lab using classification, I saw a much smaller error rate with less complex models than what the k-nearest neighbors models were able to produce. The train and test models seemed to produce more similar results with the classification method than with the knn method, but the knn method ultimately had about 4 models that produced very similar rmse's between train and test data. What is nice about the methods used in lab 11 is that we get more metrics than just error, which can reveal more nuanced indications about the predictive performance of our models.