# module3_part2

Andrew Estes

4/2/2022

```
library(tidyverse)
library(caret)
library(pROC)
```

```
set.seed(3287)
# Skip the first line of the file.
df.raw <- read.csv("default_of_credit_card_clients.csv", skip=1)
# We'll take out SEX and MARRIAGE, which seem like things you shouldn't base
# a decision on. If we don't take out ID, then we'll be training a model
# to basically use your "name" to predict your likelihood to default.
# Renaming and changing Default to a factor to make other code you might
# write happier later on.
df.raw <- df.raw %>%
  rename(Default = default.payment.next.month) %>% select(-ID) %>%
  select(-SEX, -MARRIAGE) %>%
  mutate(Default=factor(if_else(Default==1, "Yes", "No")))
# First, taking random 20% of entire data set, mostly to speed up your
# training runs. My test show we still have enough to get good prediction
# even with fewer data points.
df <- df.raw[sample(1:nrow(df.raw), round(0.2*nrow(df.raw))), ]
# The `downSample()` command from `caret` balances the data.
df <- downSample(x=select(df, -Default), y=df$Default, yname="Default")
# Then we make a training/testing split. You can do the whole assignment just
# with the training data set and cross-validation, but I put this in just in
# case you like to do one more verification of model accuracy.
ind <- createDataPartition(df$Default, p=0.70, list=FALSE)
df.train <- df[ind, ]
df.test <- df[-ind, ]
```

## 1

Use the train command with method="glm" to create a logistic regression model using your credit.train data set. Logistic regression is fast, so you can use repeated cross-validation to get a fairly reliable accuracy estimate. What is the accuracy you obtained? (Note: Since our SVM and neural network methods are, in a sense, both elaborations on what logistic regression does, this result is useful as a baseline to see whether those methods offer any benefit.)

```
fit.glm.train <-
  train(Default ~ .,
```

```
        data=df.train,
        method="glm",
        trControl=trainControl(method="repeatedcv",
                               number=100,
                               savePredictions="final",
                               classProbs=TRUE)
  )

#fit.glm.train$finalModel
fit.glm.train$results$Accuracy
```

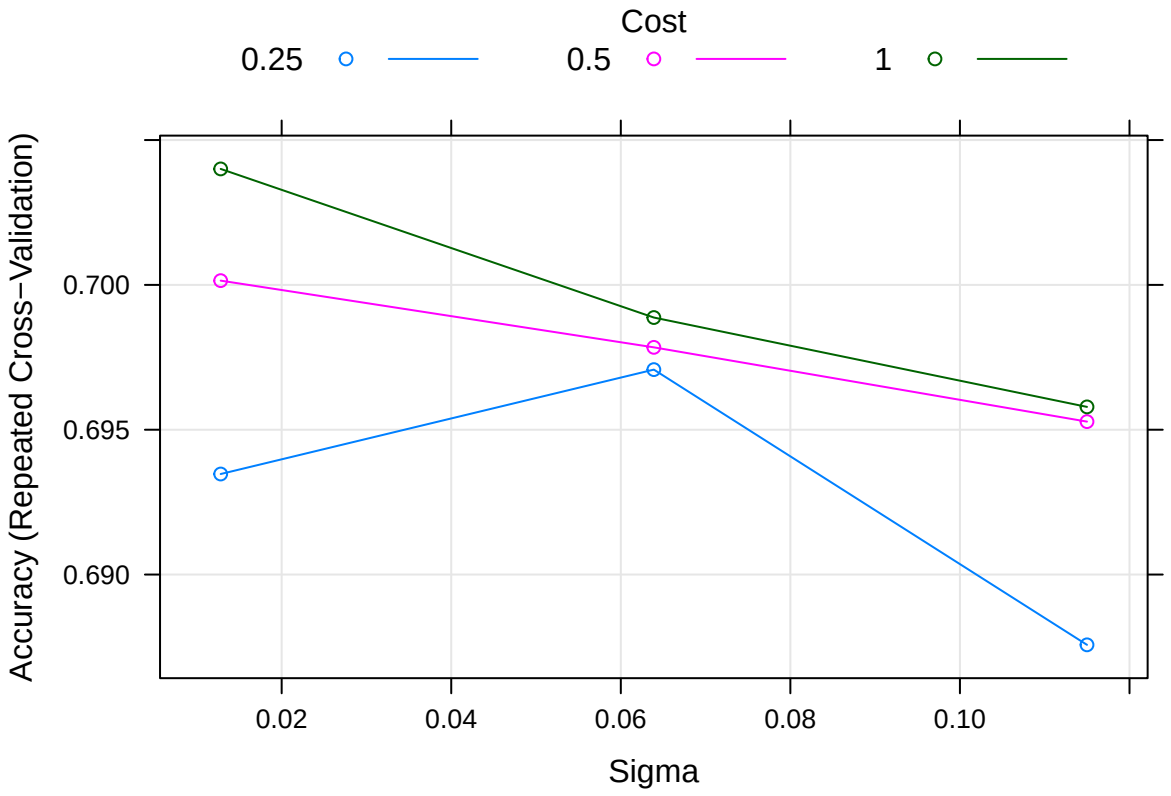
```
## [1] 0.6744708
```

Accuracy is 67.45%.

## 2

Use the train command with method="svmRadialSigma" to tune a support vector machine model. You may need to try several runs, but try to end up with a tuning grid that seems to capture a set of optimal parameters and shows drop-off on either side. Because of variation in cross-validation, you may not every find "exactly" perfect optimal parameters, but try to get a range that shows significant drop-off from an optimal zone. Output should include a plot of accuracy vs. tuning parameters as well as a statement of which tuning parameters give the best accuracy and what that accuracy was. (Note: To get ready for Question 4 below, you might want to add the appropriate options to trControl at this stage.)

```
fit.svm.train <-
  train(Default ~ .,
        data=df.train,
        method="svmRadialSigma",
        trControl=trainControl(method="repeatedcv",
                               number=3,
                               repeats=2,
                               savePredictions="final",
                               classProbs=TRUE)
  )

plot(fit.svm.train)
```




```
fit.svm.train$bestTune
```

```
##        sigma C
```

```
## 3 0.01280831 1
```

```
#fit.svm.train$finalModel
```

The best tuning parameters was at a cost of 0.50 and a sigma between .069. The actual number depends on how the CV divvied up the dataset. The accuracy was 70.9%.

This represents a 3% increase in accuracy over the GLM method even though the GLM had 50-fold CV repeated 50 times, while the SVM has 5-fold CV repeated 3 times.

## 3

Repeat the process above using method="nnet" and appropriate tuning parameters.

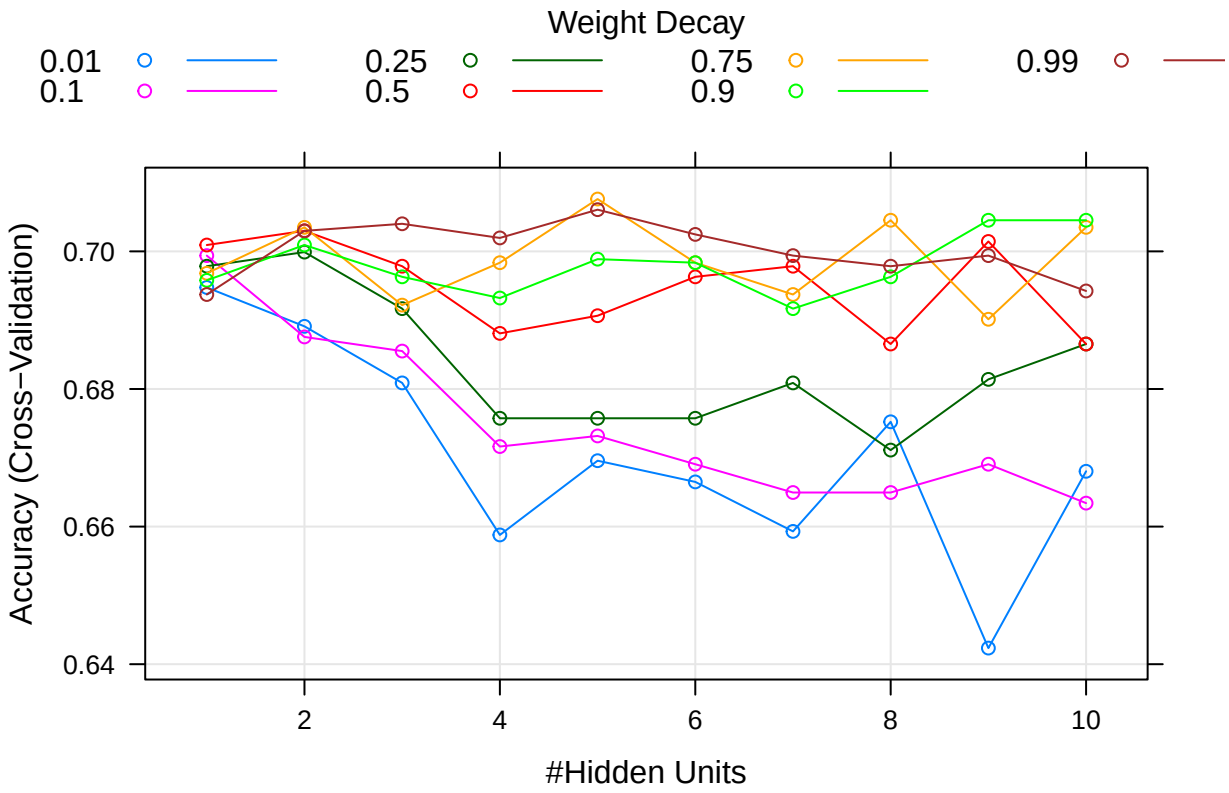
```
fit.nnet.train <-
  train(Default ~ .,
        data=df.train,
        method="nnet",
        trControl=trainControl(method="cv",
                               number=2,
                               savePredictions="final",
                               classProbs=TRUE),
        preProcess=c("scale", "center"),
        tuneGrid=expand.grid(size=c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10),
                    decay=c(.01, .1, .25, .5, .75, .9, .99)),
        trace=FALSE
  )

fit.nnet.train$bestTune
```

```
##    size decay
## 33    5  0.75
```

```
max(fit.nnet.train$results$Accuracy)
```

```
## [1] 0.7076053
```

```
plot(fit.nnet.train)
```

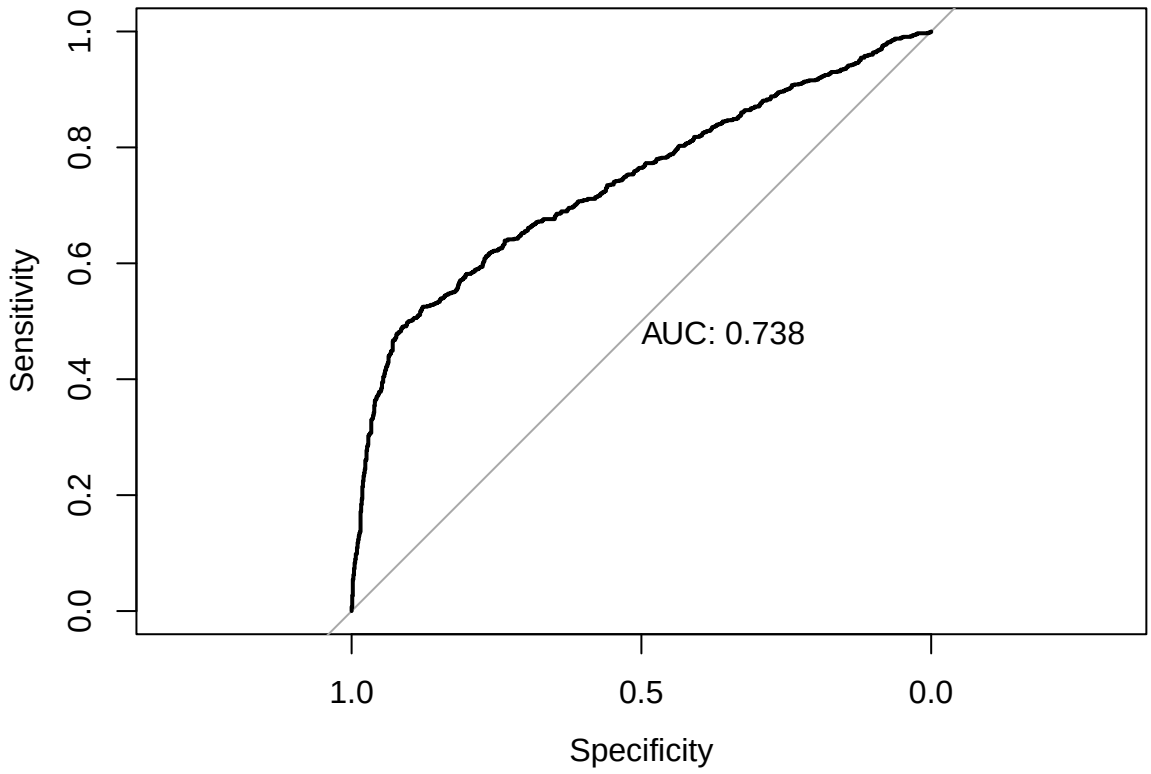



If doing it this way, increase the number of decay. .5 and .6 = 87% accuracy .5 and 5 = 95% accuracy .9 and 4 = 95% accuracy .4 and 4 = 95% accuracy .3 and 4 = 95% accuracy .3 and 5 = 95% accuracy .5 and 5 = 95% accuracy .9 and 5 = 95% accuracy .4 and 5 = 96% accuracy .9 and 5 = 96% accuracy .9 and 4.75 = 95% accuracy .7 and 4.25 = 94% accuracy .5 and 4.5 = 96% accuracy .1 and 4.2 = 95% accuracy .6 and 4.4 = 96% accuracy .7 and 4.6

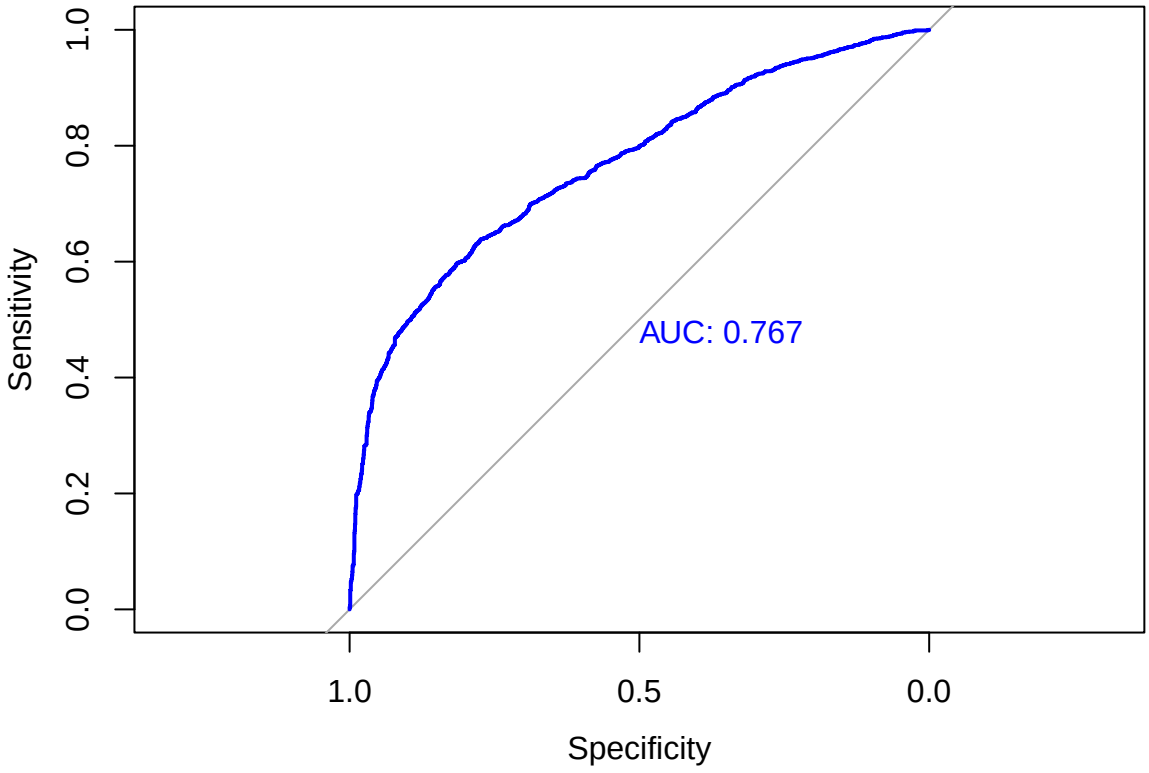
# 4

Use the roc() command from the pROC package (along with plot() command) to create plots of the ROC curves for each model. (You can use the add=TRUE option to the plot command to superimpose the curves on each other. There are also ggplot-based solutions if you want to find them.)

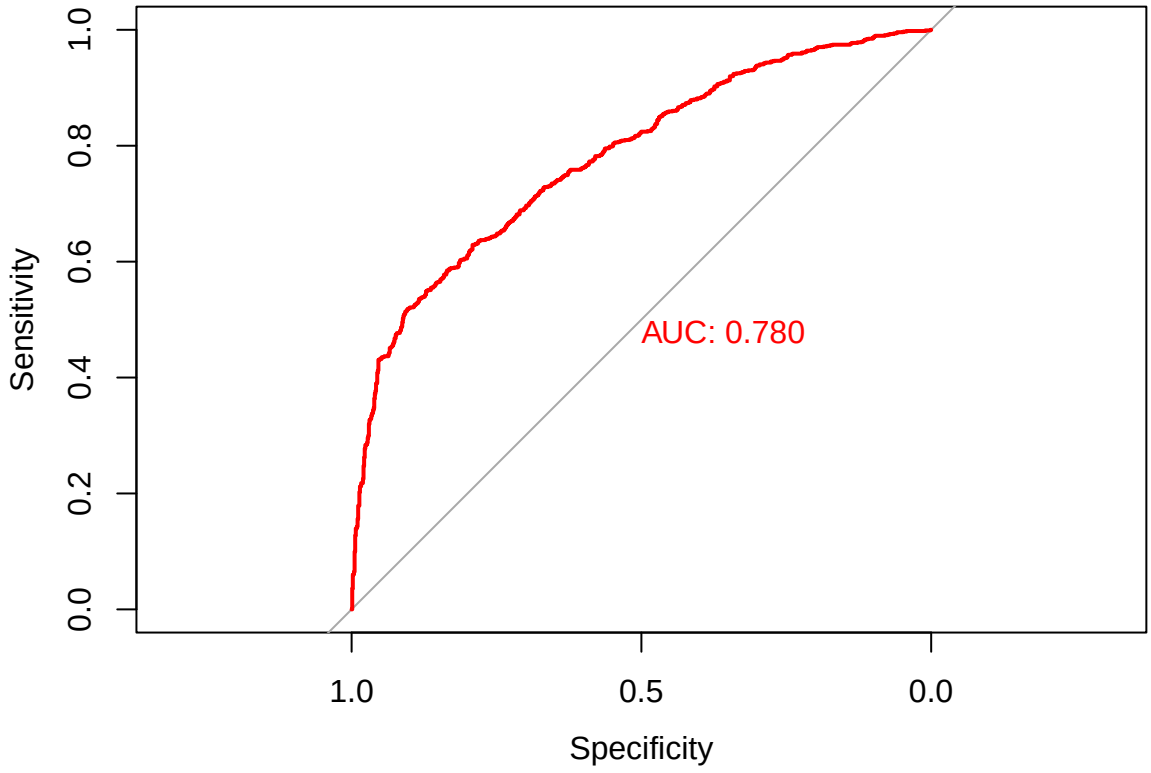
```
glm.plot.black <- plot(roc(fit.glm.train$pred$obs ~ fit.glm.train$pred$Yes),
                  print.auc=TRUE)
```




```
svm.plot.blue <- plot(roc(fit.svm.train$pred$obs ~ fit.svm.train$pred$Yes),
                  print.auc=TRUE,
                  col = "blue")
```




```
nnet.plot.red <- plot(roc(fit.nnet.train$pred$obs ~ fit.nnet.train$pred$Yes),
                      print.auc=TRUE,
                      col = "red")
```




```
?nnet
#I chose not to super-impose the graphs so we can see the AUC numbers
```

# 5

## Write up a final conclusion based on the accuracy and ROC curves for each model. Do the support ## vector machine and neural network do better than logistic regression? By how much?

The SVM did slightly better than the GLM method. I am withholding comment on the neural network as it is tremendously better than the other models - this likely has something to do with choosing a node amount of less than one. However this will take more research to ascertain.