Lab 5: Factor Selection

Sarah Wright & Tyusha

Introduction

In predictive modeling, there are often many factors (i.e., attributes) for which data are available. It is possible to build a model on any combination of these, so if there are n factors, there are $2^n - 1$ non-trivial subsets of factors. If we include the possibility of interaction terms involving the product of two or more factors, this number can get even larger. One straightforward approach would be to use all n factors in a linear or generalized linear model. However, some factors may be inconsequential, and others may lack predictive value.

A complicated model with many factors is likely to *overfit* the data. Therefore, it will have low *training error* but high *test error*. All other things being equal, a simpler model is generally considered a better model. Thus, it is desirable to keep the number of factors as small as possible while minimizing predictive error.

Factor selection is a rich topic, on which a graduate-level statistics course could spend weeks. In one day, we will only scratch the surface, considering some simple ideas for factor selection that are easily implemented in R.

Factor Selection Lab

In this lab, you will practice factor selection by working with data for NFL field goal attempts. (For those of you unfamiliar with American football, this is when the kicker tries to kick the ball through the uprights to score three points, as in this video.)

This data set contains information about 3000 NFL field goal attempts over three seasons. The column we will be interested in predicting is `Made`, which has value `1` or `0` indicating if the field goal was made or missed. If you want explanations for the other variables, you can look at the notes in the Excel file (the `.xlsx` file).

1) Load the data from `nfl-fg-data.csv` into R. Call this data `FG`. Then, partition the data into a "training" set called `FGtrain` consisting of the first 80% of the data and a "test" set called `FGtest` consisting of the last 20%. (Hint: setting `row.names = 1` in `read.csv` will make your first column row names, not data.)

```r
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.0.2
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
FG<- read.csv("/Users/sewii/Documents/CLASSES_Spring2022/Data325_AppliedDataScience/nfl_fg_data.csv", h

percentTr = 3000*0.8
FGtrain <-  slice(FG, n = 1:percentTr)


percentTe = 3000*0.2
FGtest <- slice_tail(FG, n = percentTe )
```

2) What is the mean of the `Made` column in the training set? What does this mean in context and how could it be used to create a baseline model?

```
mean(FGtrain$Made)
```

```
## [1] 0.8241667
```

```
#FGtrain$Made
```

```
        The Made column  has value '1' or '0' indicating if the field goal was made or missed.This ave

    Each shot made has a probabily of 0.8241667 of success.
```

The best way to evaluate a probabilistic model is with *likelihood*. However, to avoid introducing the theory of likelihood, we will use MSE in this lab, which will still give us a good sense of how our models perform.

3) What is the MSE of the simple baseline model on the training and testing sets? HINT: Use an `ifelse()` statement to convert the variable `Made` into a vector of probabilities. Calculate the MSE based on this new vector.

```
set.seed(12)
base_mse_train <- ifelse(FGtrain$Made ==1, 1 - mean(FGtrain$Made), mean(FGtrain$Made))
mean(base_mse_train^2)
```

```
## [1] 0.144916
```

```
#if made = 1, then we have a probability of failure printed below(1-mean())
#if they missed the ball we will have a probabily of 0.8241667

base_mse_test <- ifelse(FGtest$Made ==1, 1-mean(FGtest$Made), mean(FGtest$Made))
mean(base_mse_test^2)
```

```
## [1] 0.1421972
```

4) Use `glm()` to build a logistic model predicting the probability of a particular field goal attempt being converted using the 23 other columns of the training data. In R, the formula `Y ~ .` will predict the variable `Y` as a function of all the other variables in the data frame.

Check the MSE on both data sets (train and test) using the `predict` function. Don't forget to use use `binomial` when training your glm and `type = response` when predicting. Which model (the baseline model or the glm with all 23 factors) performs better on the *testing* data?

```
base_model_train <- glm(Made ~ ., data = FGtrain, family = "binomial")

mse_model_train <- predict(base_model_train, FGtrain, type = "response")
mse_model_test <- predict(base_model_train, FGtest, type = "response")

#perdicted - actual
Train_Model <- mean((mse_model_train-FGtrain$Made)^2)
Test_Model <- mean((mse_model_test-FGtest$Made)^2)

Train_Model
```

```
## [1] 0.1220446
```

```
Test_Model
```

```
## [1] 0.1223552
```

```
      The model that performs best on the testing data is the glm model with a smaller MSE of 0.1223552
```

5) One simple way to see what factors might be useful is to see how they correlate with `Made`. Calculate the correlation coefficients between the kick results and each of the other attributes in the training set, using the function, `cor()`. As a reminder, correlation coefficients can take on values between -1 <= r <= 1. Correlations farthest from zero (with the greatest absolute value) indicate the most promising attributes.

Which column is (by far) the best by this measure, and what does it represent?

```
cor(FGtrain$Made, FGtrain[,-1]) #could find abs value  and sort
```

```
##             Dist        Home  Stadium..       Indoor       Grass    Playoff
## [1,] -0.3547574 -0.01229919 0.01747059 -0.002913097 -0.01616234 -0.0414827
##            Time          Qtr         Marg     X.Marg.       Temp    Precip
## [1,] 0.0151261 -0.002780346 0.0007426963 0.01008999 0.03371169 0.0145577
##       Precip.Bin      Dewpt     Humidity    H.T.wind      C.wind  Altitude
## [1,] 0.007759441 0.03236847 -0.002847443 -0.02107868 -0.04965941 0.0122677
##             Wind     Denver       FG.ID      Backup     Def.PA
## [1,] -0.0437419 0.01359194 -0.04346899 -0.04825543 0.05561767
```

```
              Distance aka Dist (how far away you are from the Feild Goal) has the strongest correlatio
```

6) Build a model using only this one attribute, and compute the error values on the testing and training data. How is this model predictively compared to the baseline model and the model with all 23 attributes when assessing the test set?

```
DistanceModel <- glm(Made ~ Dist, data = FGtrain, family = "binomial")
DistanceModelMseTest <- predict(DistanceModel, FGtest, type = "response")
DistanceModelMseTrain <- predict(DistanceModel, FGtrain, type = "response")

mean((DistanceModelMseTrain-FGtrain$Made)^2)
```

```
## [1] 0.1255616
```

```
mean((DistanceModelMseTest-FGtest$Made)^2)
```

```
## [1] 0.1210686
```

```
       This model does slightly better than the model above. It does better on the testing set but is we
```

Important notes about factor selection

The model that uses all 23 columns is hindered by *overfitting*. It includes too many attributes, some of which lack predictive value, and so the model picks up on patterns created by *random noise* in the data. As we have seen in previous labs, overfitting is an ever-present issue in any kind of data-driven modeling.

Improving our model one factor at a time

Having a decent model with only one factor, we now seek to improve the model by adding other useful factors. If we go back to our list of correlations, we might select the attribute that is the second most correlated with our targets.

However, this will add little value if the factor includes much of the same information as the factor already included (e.g., the distance of each field goal in feet would correlate well to `Made`, but is predictively useless once we include distance in yards).

For this reason, it would be better to compute the correlations between each attribute and the *residuals* (errors) of our existing model. If a factor correlates well with a model's residuals (the difference between targets and model predictions), then it likely contains information not already included in the model.

7) Use this method to find a second attribute to include in the model. What is this attribute?

```
#DistanceModelMseTrain
cor((FGtrain$Made - DistanceModelMseTrain),FGtrain)
```

```
##             Made          Dist         Home Stadium..      Indoor        Grass
## [1,] 0.9284373 -1.108753e-11 -0.009931728  0.015183 0.004644238 -0.003818055
##           Playoff        Time          Qtr        Marg    X.Marg.        Temp
## [1,] -0.04140599 0.009494518 -0.004873104 -0.02102945 0.01890864 0.05147201
##           Precip  Precip.Bin      Dewpt    Humidity     H.T.wind      C.wind
## [1,] 0.01479813 -0.01214519 0.04475721 0.01013922 -0.03791322 -0.05618582
##         Altitude        Wind     Denver       FG.ID      Backup     Def.PA
## [1,] 0.01977663 -0.05809813 0.02335025 -0.04036201 -0.05467173 0.06205962
```

```
        The attribute Def.PA has the strongest correlation 0.06205962.
```

8) Build a new glm on the training set with the these two attributes and record the MSE on the test set. Is this better or worse than the one factor model?

```
DefPAModel <- glm(Made ~ Def.PA + Dist , data = FGtrain, family = "binomial")
DefPAModelMseTest <- predict(DefPAModel, FGtest, type = "response")
DefPAModelMseTrain <- predict(DefPAModel, FGtrain, type = "response")

mean((DefPAModelMseTrain-FGtrain$Made)^2)
```

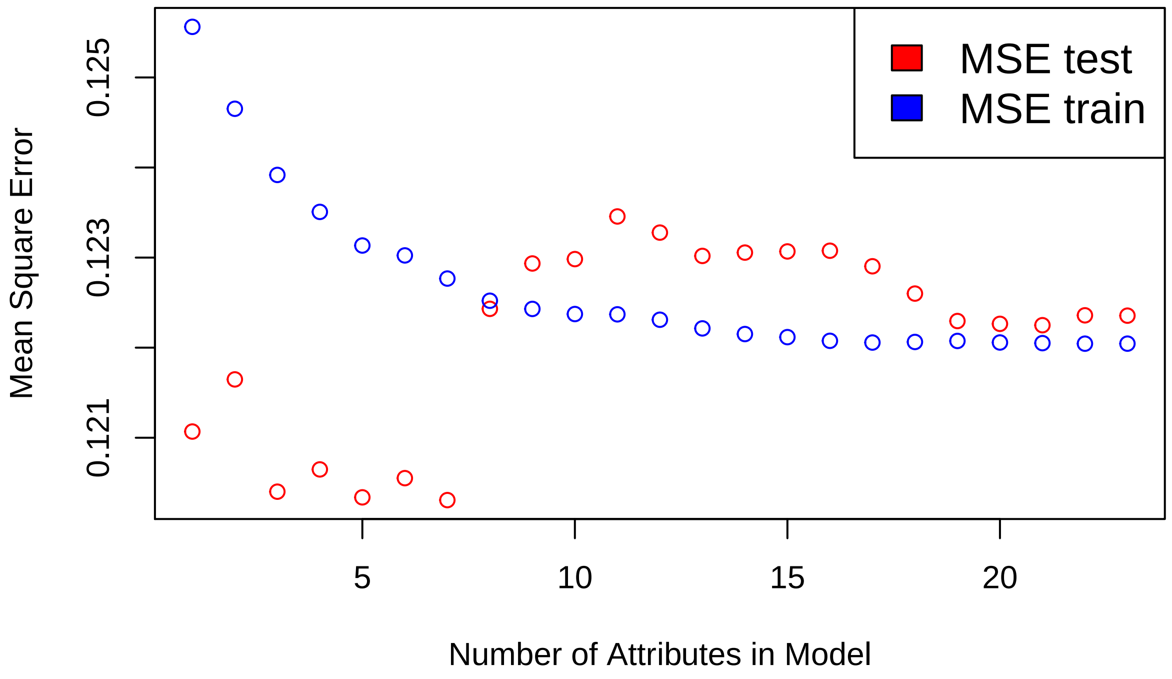
```
## [1] 0.1246522
```

```
mean((DefPAModelMseTest-FGtest$Made)^2)
```

```
## [1] 0.1216476
```

```
    This model does better than the previous model, however it is still not as good on the training set
```

If we continue the above process (using a for loop, rather than manually), we obtain the error values in the plot below. *Please read through this code carefully to understand the for loop!*



Notice, that this matches the problems with overfitting we described above. The best model on the *training data* does include all the factors. In fact, every time we add another variable to our model we are **guaranteed to improve the model fit to the training data** (evaluated by whatever metric we are optimizing when training our model). But, the best model on the *testing* data turns out to be a more simple model that avoids overfitting.