Lab4

Bushong Boys

3/30/2020

Exercise 1

Using the spam data from the kernlab library, we looked to created classifier using a logistically to determine if an email was spam or not. The following four regression models were compared.

```
fit_caps = glm(type ~ capitalTotal,
               data = spam_trn, family = binomial)
fit_selected = glm(type ~ edu + money + capitalTotal + charDollar,
                   data = spam_trn, family = binomial)
fit_additive = glm(type ~ .,
                   data = spam_trn, family = binomial)
fit_over = glm(type ~ capitalTotal * (.),
               data = spam_trn, family = binomial, maxit = 50)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

After fitting the data, we wanted to get a glimps into how the models performed. We used the following code, but noticed a similar issue as with ordinary linear regression that the misclassification rate goes down as you add more predictors despite the fact that model may be overfitting the to the training data. The results are below.

```
mean(ifelse(predict(fit_caps) > 0, "spam", "nonspam") != spam_trn$type)
```

```
## [1] 0.339
```

```
mean(ifelse(predict(fit_selected) > 0, "spam", "nonspam") != spam_trn$type)
```

```
## [1] 0.224
```

```
mean(ifelse(predict(fit_additive) > 0, "spam", "nonspam") != spam_trn$type)
```

```
## [1] 0.066
```

```
mean(ifelse(predict(fit_over) > 0, "spam", "nonspam") != spam_trn$type)
```

```
## [1] 0.136
```

To combat this problem, we decided to use a cross validation method using the cv.glm() function. We orginally ran a 5 fold validation with the seed set to one, then switched it to 100 fold with seed 90. I will not run the code due to the large number of messages it dispalys but below is the code run and summary of its output

```
# First Case
set.seed(1)
cv.glm(spam_trn, fit_caps, K = 5)$delta[1]
cv.glm(spam_trn, fit_selected, K = 5)$delta[1]
cv.glm(spam_trn, fit_additive, K = 5)$delta[1]
cv.glm(spam_trn, fit_over, K = 5)$delta[1]

#Second Case
set.seed(90)
cv.glm(spam_trn, fit_caps, K = 100)$delta[1]
cv.glm(spam_trn, fit_selected, K = 100)$delta[1]
cv.glm(spam_trn, fit_additive, K = 100)$delta[1]
cv.glm(spam_trn, fit_over, K = 100)$delta[1]
```

First Case .216 .159 .087 .14

Second Case .216 .158 .081 .14

Models fit from most underfit to overfit are: caps, selected, additive, over Models from best to worst are: additive, over, selected, caps This does not change when the seed or K-folds are altered. Now that we explored cross validation, its time to use confusion matrices on our training data to further explore the success of our models and evaluate the best one to use in this case.

```
# confusion matrix
make_conf_mat = function(predicted, actual) {
  table(predicted = predicted, actual = actual)
}

# Give us predicted values (same output, different ways)
caps_tst_pred = ifelse(predict(fit_caps, spam_tst) > 0,
                           "spam",
                           "nonspam")

selected_tst_pred = ifelse(predict(fit_selected, spam_tst) > 0,
                           "spam",
                           "nonspam")

additive_tst_pred = ifelse(predict(fit_additive, spam_tst) > 0,
                       "spam",
                       "nonspam")

over_tst_pred = ifelse(predict(fit_over, spam_tst) > 0,
                           "spam",
                           "nonspam")

# Create confusion matrices for each
caps_matrix = make_conf_mat(predicted = caps_tst_pred, actual = spam_tst$type)
caps_matrix
```

```
##          actual
```

```
## predicted nonspam spam
##   nonspam    2022 1066
##   spam        162  351
```

```
mean(caps_tst_pred != spam_tst$type)
```

```
## [1] 0.3410164
```

```
sensitivity(caps_matrix)
```

```
## [1] 0.9258242
```

```
specificity(caps_matrix)
```

```
## [1] 0.2477064
```

```
selected_matrix = make_conf_mat(predicted = selected_tst_pred, actual = spam_tst$type)
selected_matrix
```

```
##          actual
## predicted nonspam spam
##   nonspam    2073  615
##   spam        111  802
```

```
mean(selected_tst_pred != spam_tst$type)
```

```
## [1] 0.2016107
```

```
sensitivity(selected_matrix)
```

```
## [1] 0.9491758
```

```
specificity(selected_matrix)
```

```
## [1] 0.5659845
```

```
additive_matrix = make_conf_mat(predicted = additive_tst_pred, actual = spam_tst$type)
additive_matrix
```

```
##          actual
## predicted nonspam spam
##   nonspam    2057  157
##   spam        127 1260
```

```
mean(additive_tst_pred != spam_tst$type)
```

```
## [1] 0.07886698
```

```r
sensitivity(additive_matrix)
```

```
## [1] 0.9418498
```

```r
specificity(additive_matrix)
```

```
## [1] 0.8892025
```

```r
over_matrix = make_conf_mat(predicted = over_tst_pred, actual = spam_tst$type)
over_matrix
```

```
##           actual
## predicted nonspam spam
##   nonspam    1725  103
##   spam        459 1314
```

```r
mean(over_tst_pred != spam_tst$type)
```

```
## [1] 0.1560678
```

```r
sensitivity(over_matrix)
```

```
## [1] 0.7898352
```

```r
specificity(over_matrix)
```

```
## [1] 0.9273112
```

In making the decision on what is the best model to use, we should first evaluate the overall accuracy of each model using their misclassification rate. Since borht the caps and selected models have relatively high rates (.34 and .20 respectively), we can eliminate them from discussion. Instead, let's narrow ourselves down to the additive and over models with misclassification rates of .078 abd .156. It may be tempting to pick the additive model from this measure; however there is one additional factor we should still consider.

In this scenario, it is a much costly error to have actual spam be classified as predicted nonspam since the user will just have to delete the email. On the other hand if actual nonspam is classified as spam, important messages may be lost and the user will have to constantly dig through their spam folder. For this reason, sensitivity and specificity are valuable measures. With this scenario we want low false negatives. Since higher values of false negatives decrease the sensitivity measure, we want to have high sensitivity. Therefore since the sesitivity values for additive and over are .942 and .7898 we should choose the additive method. In another less logical scenario where we cared more about the case where actual spam is classified as nonspam (False positive) we might consider taking the model over since it has a much higher specificity ($.9273 > .8892$)