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
```

```r
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

```r
# 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.

```r
# 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")
```

```r
# 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
```

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

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

```r
sensitivity(caps_matrix)
```

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

```r
specificity(caps_matrix)
```

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

```r
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
```

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

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

```r
sensitivity(selected_matrix)
```

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

```r
specificity(selected_matrix)
```

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

```r
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
```

```r
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)

Excercize 2

```r
bank = read.csv("bank.csv")
table(bank$y)
```

```
##
##   no  yes
## 4000  521
```

```r
bank_idx = sample(nrow(bank), 4000)
bank_trn = bank[bank_idx, ]
bank_tst = bank[-bank_idx, ]
fit = glm(y ~ age + balance + campaign + previous +loan +duration + housing, data = bank_trn, family = b
# Run cross fold validation
set.seed(1)
cv.glm(bank_trn, fit, K = 10)$delta[1]
```

```
## [1] 0.085372
```

```r
bank_pred = ifelse(predict(fit, bank_tst) > 0,
                       "yes",
                       "no")

# Create confusion matrices for each
fit_matrix = make_conf_mat(predicted = bank_pred, actual = bank_tst$y)
mean(bank_pred != bank_tst$y)
```

```
## [1] 0.0940499
```

```r
sensitivity(fit_matrix)
```

```
## [1] 0.9768421
```

```r
specificity(fit_matrix)
```

```
## [1] 0.173913
```

```r
summary(fit)
```

```
##
## Call:
## glm(formula = y ~ age + balance + campaign + previous + loan +
##     duration + housing, family = binomial, data = bank_trn)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.2108  -0.4433  -0.3279  -0.2363   2.7235
##
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.937e+00  2.503e-01 -11.732  < 2e-16 ***
```

```
## age          3.916e-03  5.001e-03   0.783  0.43363
## balance      2.608e-05  1.841e-05   1.417  0.15647
## campaign    -8.910e-02  2.718e-02  -3.278  0.00104 **
## previous     1.664e-01  2.399e-02   6.936 4.04e-12 ***
## loanyes     -8.741e-01  1.976e-01  -4.423 9.73e-06 ***
## duration     3.884e-03  1.941e-04  20.012  < 2e-16 ***
## housingyes  -8.256e-01  1.162e-01  -7.103 1.22e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2915.4  on 3999  degrees of freedom
## Residual deviance: 2269.5  on 3992  degrees of freedom
## AIC: 2285.5
##
## Number of Fisher Scoring iterations: 6
```

We created a model using age,balance,campaign,previous,loan,duration,housing. It does out perform classifying all observations as the majority case, but it is a somewhat weak model with low sensitivity. Since it has fairly high specificity it could be useful for a marketing campaign since clients predicted postive are very likely to sign with the bank, but it still lets quite a few yes's through the cracks. It would probably be more useful to bias the models predictions upwards and focus on the most likely candidates(even though the model predicts them as no).

The coeefecients on the model are:

Intercept -2.911e+00 This drives the model down, basically it is saying if everything else is 0 we should say this client will not subscribe.

age 5.079e-03 The coeeficent on age is .00078 as people grow older they are slightly more likely to subscribe. This is however not a statistically significant prediction.

balance 2.473e-05 As balance increased people are slightly more likely to subscribe.

campaign -9.836e-01 The number of contacts has a fairly strong negative relationship with likelihood to subscribe.

previous 3.757e-01 On the other hand the number of contacts performed before this campaign has a fairly strong positive effect.

duration 8.879e-03 This is the strongest predictor in our data, as duration increases it becomes more and more likely the result will be yes.

housingyes -7.976e-01 If the person has a housing loan they are significantly less likely to subscribe.

```r
fit_matrix
```

```
##          actual
## predicted  no yes
##       no  464  38
##       yes  11   8
```

```r
bank_pred = ifelse(predict(fit, bank_tst) > -2,
                       "yes",
                       "no")
```

```r
# Create confusion matrices for each
fit_matrix = make_conf_mat(predicted = bank_pred, actual = bank_tst$y)
mean(bank_pred != bank_tst$y)
```

```
## [1] 0.2072937
```

```r
sensitivity(fit_matrix)
```

```
## [1] 0.7978947
```

```r
specificity(fit_matrix)
```

```
## [1] 0.7391304
```

```r
summary(fit)
```

```
##
## Call:
## glm(formula = y ~ age + balance + campaign + previous + loan +
##     duration + housing, family = binomial, data = bank_trn)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.2108  -0.4433  -0.3279  -0.2363   2.7235
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.937e+00  2.503e-01 -11.732  < 2e-16 ***
## age          3.916e-03  5.001e-03   0.783  0.43363
## balance      2.608e-05  1.841e-05   1.417  0.15647
## campaign    -8.910e-02  2.718e-02  -3.278  0.00104 **
## previous     1.664e-01  2.399e-02   6.936 4.04e-12 ***
## loanyes     -8.741e-01  1.976e-01  -4.423 9.73e-06 ***
## duration     3.884e-03  1.941e-04  20.012  < 2e-16 ***
## housingyes  -8.256e-01  1.162e-01  -7.103 1.22e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2915.4  on 3999  degrees of freedom
## Residual deviance: 2269.5  on 3992  degrees of freedom
## AIC: 2285.5
##
## Number of Fisher Scoring iterations: 6
```

```r
fit_matrix
```

```
##          actual
## predicted  no yes
##       no  379  12
##       yes  96  34
```

By biasing up like this I sacrifice specificity for sensitivity, but since a bank is mostly only concerned with people who say yes. I think this is a more valuable model, despite its worse absolute performance.