homework

Lab 4 : Report

> **Use Logistic Regression In Making Binary Predictionsr** is different from the linear regression model that is often used in daily learning or work. When predicting something, such as predicting house prices, height, GDP, student performance, etc., it is found that these predicted variables are continuous variables.
>
> However, in some cases, the predicted variable may be a binary variable, that is, success or failure, loss or not loss, rise or fall, etc. For such problems, linear regression will be helpless. At this time, another regression method is required for prediction, that is, Logistic regression.

Exercise 1

First Part

Question 1

Since the code has been shown, only the output results and analysis process will be shown here.

```
# install.packages("kernlab")
library(kernlab)
data("spam")
tibble::as.tibble(spam)
```

```
## Warning: `as.tibble()` is deprecated, use `as_tibble()` (but mind th
e new semantics).
## This warning is displayed once per session.
```

```
## # A tibble: 4,601 x 58
##     make address   all num3d   our  over remove internet order  mail
 receive
##    <dbl>   <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>    <dbl> <dbl> <dbl>
   <dbl>
##  1  0       0.64  0.64     0  0.32  0      0        0     0     0
    0
##  2  0.21    0.28  0.5      0  0.14  0.28   0.21     0.07  0     0.94
    0.21
##  3  0.06    0     0.71     0  1.23  0.19   0.19     0.12  0.64  0.25
    0.38
##  4  0       0     0        0  0.63  0      0.31     0.63  0.31  0.63
    0.31
##  5  0       0     0        0  0.63  0      0.31     0.63  0.31  0.63
```

```
    0.31
##   6  0       0     0         0  1.85  0      0        1.85  0     0
    0
##   7  0       0     0         0  1.92  0      0        0     0     0.64
    0.96
##   8  0       0     0         0  1.88  0      0        1.88  0     0
    0
##   9  0.15    0     0.46      0  0.61  0      0.3      0     0.92  0.76
    0.76
## 10  0.06    0.12  0.77      0  0.19  0.32   0.38     0     0.06  0
    0
## # ... with 4,591 more rows, and 47 more variables: will <dbl>, peopl
e <dbl>,
## #   report <dbl>, addresses <dbl>, free <dbl>, business <dbl>, email
 <dbl>,
## #   you <dbl>, credit <dbl>, your <dbl>, font <dbl>, num000 <dbl>, m
oney <dbl>,
## #   hp <dbl>, hpl <dbl>, george <dbl>, num650 <dbl>, lab <dbl>, labs
 <dbl>,
## #   telnet <dbl>, num857 <dbl>, data <dbl>, num415 <dbl>, num85 <db
l>,
## #   technology <dbl>, num1999 <dbl>, parts <dbl>, pm <dbl>, direct <
dbl>,
## #   cs <dbl>, meeting <dbl>, original <dbl>, project <dbl>, re <dbl>,
## #   edu <dbl>, table <dbl>, conference <dbl>, charSemicolon <dbl>,
## #   charRoundbracket <dbl>, charSquarebracket <dbl>, charExclamation
 <dbl>,
## #   charDollar <dbl>, charHash <dbl>, capitalAve <dbl>, capitalLong
<dbl>,
## #   capitalTotal <dbl>, type <fct>
```

```r
is.factor(spam$type)
```

```
## [1] TRUE
```

```r
levels(spam$type)
```

```
## [1] "nonspam" "spam"
```

```r
set.seed(42)
# spam_idx = sample(nrow(spam), round(nrow(spam) / 2))
spam_idx = sample(nrow(spam), 1000)
spam_trn = spam[spam_idx, ]
spam_tst = spam[-spam_idx, ]

fit_caps = glm(type ~ capitalTotal,
               data = spam_trn, family = binomial)
fit_selected = glm(type ~ edu + money + capitalTotal + charDollar,
                   data = spam_trn, family = binomial)
```

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

```r
fit_additive = glm(type ~ .,
                   data = spam_trn, family = binomial)
```

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

```r
fit_over = glm(type ~ capitalTotal * (.),
               data = spam_trn, family = binomial, maxit = 50)
```

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

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

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

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

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

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

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

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

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

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

and

```r
library(boot)
set.seed(1)
cv.glm(spam_trn, fit_caps, K = 5)$delta[1]
```

```
## [1] 0.2166961
```

```r
cv.glm(spam_trn, fit_selected, K = 5)$delta[1]
```

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

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

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

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

```
## [1] 0.1587043
```

```r
cv.glm(spam_trn, fit_additive, K = 5)$delta[1]
```

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

## [1] 0.08684467

cv.glm(spam_trn, fit_over, K = 5)$delta[1]

## [1] 0.137
```

According to the outcome, the misclassification rate of each model when the k-fold cross-validated is not used are `0.339 0.224 0.066 0.136`, and the misclassification rate of each model when the k-fold cross-validated is used are `0.217 0.159 0.087 0.137`.

So the answer of the first question of exercise1 is: **the model `fit_caps` is the most underfit model**, for its misclassification rate when the k-fold cross-validated is not used and misclassification rate when the k-fold cross-validated is used are both the highest in these four models.

And, **the model `fit_additive` is the most overfit model**, for its misclassification rate when the k-fold cross-validated is not used and misclassification rate when the k-fold cross-validated is used are both the lowest in these four models.

Question 2

Re-run the code above with 100 folds and a different seed of 2 as required.

```
set.seed(2)
cv.glm(spam_trn, fit_caps, K = 100)$delta[1]

## [1] 0.2168058

cv.glm(spam_trn, fit_selected, K = 100)$delta[1]

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

## [1] 0.1588852

cv.glm(spam_trn, fit_additive, K = 100)$delta[1]

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

## [1] 0.08098914

cv.glm(spam_trn, fit_over, K = 100)$delta[1]

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

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

## [1] 0.136
```

and our conclusion is nothing different from before.

Second Part

Question 1

Using the function given named `make_conf_mat` , we can generate fout confusion matrix as:

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

spam_tst_pred = ifelse(predict(fit_additive, spam_tst) > 0,
                       "spam",
                       "nonspam")
#spam_tst_pred = ifelse(predict(fit_additive, spam_tst, type = "respons
e") > 0.5,
#                      "spam",
#                      "nonspam")

(conf_mat_50 = make_conf_mat(predicted = spam_tst_pred, actual = spam_t
st$type))
```

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

```
table(spam_tst$type) / nrow(spam_tst)
```

```
##
##   nonspam      spam
## 0.6064982 0.3935018
```

to predict

```
spam_tst_pred1 = ifelse(predict(fit_caps, spam_tst) > 0,
                        "spam",
                        "nonspam")
spam_tst_pred2 = ifelse(predict(fit_selected, spam_tst) > 0,
                        "spam",
                        "nonspam")
spam_tst_pred3 = ifelse(predict(fit_additive, spam_tst) > 0,
                        "spam",
                        "nonspam")
spam_tst_pred4 = ifelse(predict(fit_over, spam_tst) > 0,
                        "spam",
                        "nonspam")

(conf_mat_caps<-make_conf_mat(predicted = spam_tst_pred1, actual = spam
_tst$type))
```

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

```
(conf_mat_selected<-make_conf_mat(predicted = spam_tst_pred2, actual =
spam_tst$type))
```

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

```
(conf_mat_additive<-make_conf_mat(predicted = spam_tst_pred3, actual =
spam_tst$type))
```

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

```
(conf_mat_over<-make_conf_mat(predicted = spam_tst_pred4, actual = spam
_tst$type))
```

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

Question 2

As for the overall accuracy, we can use function to calculate Prev value as:

```
prev_calcu<-function(mat){
  prev<-sum(diag(mat))/sum(mat)
  prev
}
```

Using this fuction, we can generate four overall accuracy like:

```
prev_calcu(conf_mat_caps)
```

```
## [1] 0.6589836
```

```
prev_calcu(conf_mat_selected)
```

```
## [1] 0.7983893
```

```
prev_calcu(conf_mat_additive)
```

```
## [1] 0.921133
```

```
prev_calcu(conf_mat_over)
```

```
## [1] 0.8439322
```

so the overall accuracy for each model are about `0.66`,`0.80`,`0.92`,`0.84` .

And the function below can calculate the sensitivity value and specificity value:

```
sens_calcu<-function(mat){
  prev<-mat[1,1]/sum(mat[,1])
  prev
}

spec_calcu<-function(mat){
  prev<-mat[2,2]/sum(mat[,2])
  prev
}

sens_calcu(conf_mat_caps)

## [1] 0.9258242

sens_calcu(conf_mat_selected)

## [1] 0.9491758

sens_calcu(conf_mat_additive)

## [1] 0.9418498

sens_calcu(conf_mat_over)

## [1] 0.7898352

spec_calcu(conf_mat_caps)

## [1] 0.2477064

spec_calcu(conf_mat_selected)

## [1] 0.5659845

spec_calcu(conf_mat_additive)

## [1] 0.8892025

spec_calcu(conf_mat_over)

## [1] 0.9273112
```

Considering the sensitivity value and specificity value, the first three models have good ability in identify spam emails from all the spam emails. But for identify no-spam emails from all no-spam emails, the first two models seems too strict that may let the user of email account miss some inportant information, which is worse than identify spam emails to good ones.

Above all, as far as I'm concerned, the third model, which names `fit_additive` is the best among four models for it's high sensitivity value and more importantly, the specificity value. The model `fit_over` also do well in specificity value but the improvment on specificity is not obvious and is too lenient in filtering spam, which will cause a lot of spam to flood the user homepage.

Exercise 2

Create Split

Holding 4521 observations, we split the data in 2:1, that is:

```
dat<-read.csv('bank.csv')

yes_idx = sample(nrow(dat), 3014)
yes_trn = dat[yes_idx, ]
yes_tst = dat[-yes_idx, ]
```

Logistic Regression

Using the `cv.glm()` function and set the data and K value, we got:

```
fit_yes<-glm(y ~ .,data = yes_trn, family = binomial)
summary(fit_yes)

##
## Call:
## glm(formula = y ~ ., family = binomial, data = yes_trn)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.8399  -0.4205  -0.2790  -0.1649   3.0347
##
## Coefficients:
##                      Estimate Std. Error z value Pr(>|z|)
## (Intercept)        -2.033e+00  5.884e-01  -3.454 0.000551 ***
## age                -4.223e-03  8.258e-03  -0.511 0.609082
## jobblue-collar     -3.301e-01  2.807e-01  -1.176 0.239490
## jobentrepreneur    -6.711e-02  4.460e-01  -0.150 0.880384
## jobhousemaid       -1.032e-01  4.685e-01  -0.220 0.825650
## jobmanagement       4.368e-02  2.822e-01   0.155 0.877004
## jobretired          6.675e-01  3.624e-01   1.842 0.065514 .
## jobself-employed   -4.498e-02  4.183e-01  -0.108 0.914387
## jobservices        -2.929e-01  3.255e-01  -0.900 0.368199
## jobstudent          1.926e-01  4.520e-01   0.426 0.670085
## jobtechnician      -1.140e-01  2.696e-01  -0.423 0.672408
## jobunemployed      -6.288e-01  5.086e-01  -1.236 0.216367
## jobunknown          1.248e-01  6.965e-01   0.179 0.857778
## maritalmarried     -1.960e-01  2.081e-01  -0.942 0.346279
## maritalsingle      -1.373e-01  2.459e-01  -0.558 0.576728
## educationsecondary  7.861e-02  2.312e-01   0.340 0.733843
```

```
## educationtertiary   1.514e-01  2.701e-01   0.561 0.574975    
## educationunknown   -3.767e-01  4.278e-01  -0.881 0.378491    
## defaultyes          6.252e-01  5.031e-01   1.243 0.214052    
## balance             7.742e-06  2.373e-05   0.326 0.744264    
## housingyes         -3.792e-01  1.592e-01  -2.382 0.017216 *  
## loanyes            -4.886e-01  2.223e-01  -2.198 0.027975 *  
## contacttelephone   -3.576e-02  2.699e-01  -0.133 0.894588    
## contactunknown     -1.507e+00  2.549e-01  -5.911 3.39e-09 ***
## day                -1.576e-03  9.416e-03  -0.167 0.867100    
## monthaug           -4.208e-01  2.785e-01  -1.511 0.130828    
## monthdec            5.524e-01  6.578e-01   0.840 0.401094    
## monthfeb           -2.594e-01  3.466e-01  -0.748 0.454192    
## monthjan           -1.377e+00  5.214e-01  -2.641 0.008277 ** 
## monthjul           -9.322e-01  2.818e-01  -3.309 0.000938 ***
## monthjun            3.751e-01  3.304e-01   1.135 0.256285    
## monthmar            1.436e+00  4.487e-01   3.200 0.001375 ** 
## monthmay           -6.154e-01  2.636e-01  -2.335 0.019554 *  
## monthnov           -9.950e-01  3.108e-01  -3.201 0.001368 ** 
## monthoct            1.223e+00  3.785e-01   3.232 0.001231 ** 
## monthsep            4.417e-01  4.928e-01   0.896 0.370115    
## duration            3.999e-03  2.354e-04  16.984  < 2e-16 ***
## campaign           -9.659e-02  3.558e-02  -2.715 0.006631 ** 
## previous            1.189e-01  3.271e-02   3.637 0.000276 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## (Dispersion parameter for binomial family taken to be 1)
## 
##     Null deviance: 2181  on 3013  degrees of freedom
## Residual deviance: 1578  on 2975  degrees of freedom
## AIC: 1656
## 
## Number of Fisher Scoring iterations: 6
```

```r
cv_yes<-cv.glm(yes_trn, fit_yes, K = 10)
cv_yes$delta[1]
```

```
## [1] 0.08402389
```

Outcome Interpretation

On the basis of the outcome and summary, the `Intercept` and variables `contactunknown`,`monthmar`,`monthoct`,`monthsep`,`duration`,`previous`,`loanyes`,`monthjul`,`campaign` has significant impact on `y` if we set the significant code to `0.01`. Among these variables, `contactunknown` ,`loanyes`,`campaign`,`monthjul` has a negative coefficient, which means that if the contact is unkown or the user is in debt or it's in July, or more campiagh there are, it's more likely to have `y = no`.

And variables `monthmar`,`monthoct`,`monthsep`,`previous` has a positive coefficient, which means that the higher previous are, or if it's in March,September or October, it's more likely to have `y = yes`.

Confusion Matrix and Evaluation

As same as in exercise 1, use:

```
yes_tst_pred = ifelse(predict(fit_yes, yes_tst) > 0,
                      "yes",
                      "no")

(conf_mat_yes<-make_conf_mat(predicted = yes_tst_pred, actual = yes_tst
$y))

##          actual
## predicted   no  yes
##       no  1312  120
##       yes   28   47

prev_calcu(conf_mat_yes)

## [1] 0.9017916

sens_calcu(conf_mat_yes)

## [1] 0.9791045

spec_calcu(conf_mat_yes)

## [1] 0.2814371
```

the Prev value and sensitivity is good but the specificity is awful, obviously the model is kind of overfitted. So I first set the training set smaller with `1507` observation and use same fit model and it got:

```
> prev_calcu(conf_mat_yes)
[1] 0.8865295
> sens_calcu(conf_mat_yes)
[1] 0.9710961
> spec_calcu(conf_mat_yes)
[1] 0.2428571
```

but the specificity is still awful. And I set the K value to 100 while keep training set of 3014, still got:

```
> prev_calcu(conf_mat_yes)
[1] 0.8984738
> sens_calcu(conf_mat_yes)
[1] 0.9768311
> spec_calcu(conf_mat_yes)
[1] 0.2781065
```

And I fit another small model:

```
fit_yes_small<-glm(y~housing+contact+month+duration,data = yes_trn, fam
ily = binomial)

cv_yes_small<-cv.glm(yes_trn, fit_yes_small, K = 10)
cv_yes_small$delta[1]

## [1] 0.08187217

yes_tst_pred_small = ifelse(predict(fit_yes_small, yes_tst) > 0,
                      "yes",
                      "no")

(conf_mat_yes_small<-make_conf_mat(predicted = yes_tst_pred_small, actu
al = yes_tst$y))

##          actual
## predicted   no  yes
##       no  1313  127
##       yes   27   40

prev_calcu(conf_mat_yes_small)

## [1] 0.8978102

sens_calcu(conf_mat_yes_small)

## [1] 0.9798507

spec_calcu(conf_mat_yes_small)

## [1] 0.239521
```

So according to the jod done, I think the problem is not the model or K value or the scale of training set. Every outcome generated is reasonable.