Homework2

Gabriel Fei

10/26/2021

```
## -- Attaching packages ----------------------------------------------------------------------------

## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.2
## v tidyr   1.1.1     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0

## Warning: package 'ggplot2' was built under R version 4.0.1

## Warning: package 'tibble' was built under R version 4.0.2

## Warning: package 'tidyr' was built under R version 4.0.2

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

## -- Conflicts --------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Linear Regression

1.

```
linmod1 = lm(mpg~cylinders + displacement + horsepower + weight + acceleration + year + origin, data = A
summary(linmod1)
```

```
##
## Call:
## lm(formula = mpg ~ cylinders + displacement + horsepower + weight +
##     acceleration + year + origin, data = Auto)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.5903 -2.1565 -0.1169  1.8690 13.0604
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -17.218435   4.644294  -3.707  0.00024 ***
## cylinders     -0.493376   0.323282  -1.526  0.12780
## displacement   0.019896   0.007515   2.647  0.00844 **
## horsepower    -0.016951   0.013787  -1.230  0.21963
## weight        -0.006474   0.000652  -9.929  < 2e-16 ***
## acceleration   0.080576   0.098845   0.815  0.41548
## year           0.750773   0.050973  14.729  < 2e-16 ***
## origin         1.426141   0.278136   5.127 4.67e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.328 on 384 degrees of freedom
## Multiple R-squared:  0.8215, Adjusted R-squared:  0.8182
## F-statistic: 252.4 on 7 and 384 DF,  p-value: < 2.2e-16
```

Using our threshold of 0.01, we can take a look at the p-values for each of the coefficients to see if there's any linear association between those predictors and mpg by checking if those values are greater or less than 1. Cylinders' p-value is greater than 0.01, thus we fail to reject the null hypothesis, hence cylinders has no linar association with mpg. Displacement's p-value is smaller than 0.01 thus we can reject the null hypothesis and conclude that there is a linear association between displacement and mpg. Horsepower has a p-value that's greater than 0.01, so like before, we fail to reject the null hypothesis and can conclude that horsepower has no linear association with mpg. Weight has a p-value that is a lot smaller than 0.01 and so we can reject the null and say that weight and mpg have a linear association. Acceleration has a p-value that's bigger than 0.01 so we fail to reject the null hypothesis and thus acceleration and mpg have no linear association. Year has a p-value that's less than 0.01 so we reject the null and say there's a linear association between year and mpg. Finally origin has a p-value much smaller than 0.01, thus we can reject the null and say that origin has linear association with mpg.
Have linear association with mpg: Displacement, Weight, Year, Origin
Don't have linear association with mpg: Cylinders, Horsepower, Acceleration

2.

```
auto_training = Auto
print(mean((auto_training$mpg - linmod1$fitted.values)^2))
```

```
## [1] 10.84748
```

The training mean squared error of this model is 10.84748

3.

```
new_auto = data.frame(cylinders = c(4), displacement = c(122.0), horsepower = c(1045), weight = c(3100)
auto_pred = predict(linmod1, newdata = new_auto)
auto_pred
```

```
##        1
## 19.20287
```

The gas mileage that's predicted for a European car with 4 cylinders, displacement 122, horsepower of 105, weight of 3100, acceleration of 32, and built in the year 1991 is **19.20287 mpg**.

4.

```
ja = mean(Auto[Auto['origin'] == 3, ]$mpg)
am = mean(Auto[Auto['origin'] == 1, ]$mpg)
ja_diff = ja - am
ja_diff
```

```
## [1] 10.41716
```

On average, the difference between the mpg of a Japanese car and the mpg of an American car is **10.41716.**

```r
eu = mean(Auto[Auto['origin'] == 2, ]$mpg)
ea_diff = eu - am
ea_diff
```

```
## [1] 7.569472
```

On average, the difference between the mpg of a European car and the mpg of an American car is **7.569472.**

5.

```r
linmod1$coefficients['displacement'] * 10
```

```
## displacement
##    0.1989564
```

The change in mpg associated with a 10 unit increase in displacement is 0.1989564.

Algae Classification using Logistic Regression

1.

```
## Parsed with column specification:
## cols(
##   season = col_character(),
##   size = col_character(),
##   speed = col_character(),
##   mxPH = col_double(),
##   mnO2 = col_double(),
##   Cl = col_double(),
##   NO3 = col_double(),
##   NH4 = col_double(),
##   oPO4 = col_double(),
##   PO4 = col_double(),
##   Chla = col_double(),
##   a1 = col_double(),
##   a2 = col_double(),
##   a3 = col_double(),
##   a4 = col_double(),
##   a5 = col_double(),
##   a6 = col_double(),
##   a7 = col_double()
## )
```

```
## Warning: `funs()` is deprecated as of dplyr 0.8.0.
## Please use a list of either functions or lambdas:
##
##   # Simple named list:
##   list(mean = mean, median = median)
##
##   # Auto named with `tibble::lst()`:
##   tibble::lst(mean, median)
##
##   # Using lambdas
##   list(~ mean(., trim = .2), ~ median(., na.rm = TRUE))
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

Showing that the inverse of a logistic function is the logit function:

$$
\begin{aligned}
p(z) &= \frac{e^z}{1+e^z} \\
p \cdot (1+e^z) &= e^z \\
p \cdot 1 + p \cdot e^z &= e^z \\
e^z - pe^z &= p \\
e^z(1-p) &= p \\
e^z &= \frac{p}{1-p} \\
log(e^z) &= log_e(\frac{p}{1-p}) \\
z(p) &= ln(\frac{p}{1-p})
\end{aligned}
$$

2.

We assume that $z = \beta_0 + \beta_1 x_1$ and $p = logistic(z)$. If we increase x_1 by two, **the odds of the outcome will increase by e_1^{β} by 2-units**. If we assumed β_1 to be negative, as $x_1 \to \infty$, p would approach a **value of 0**, and as $x_1 \to -\infty$, p would approach a **value of infinity**.

3.

```
glm.fit = glm(a1 ~ season + size + speed + mxPH + mnO2 + Cl + NO3 + NH4 + oPO4 + PO4 + Chla, data = alga
              family = binomial)
summary(glm.fit)
```

```
##
## Call:
## glm(formula = a1 ~ season + size + speed + mxPH + mnO2 + Cl +
##     NO3 + NH4 + oPO4 + PO4 + Chla, family = binomial, data = algae.train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.9032  -0.6381   0.1222   0.5772   1.9807
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)   2.37083   11.30121   0.210  0.83384
## seasonspring -0.65822    0.78301  -0.841  0.40056
## seasonsummer  0.88654    0.84198   1.053  0.29237
## seasonwinter  0.61594    0.67731   0.909  0.36314
## sizemedium    0.60433    0.75668   0.799  0.42449
## sizesmall     1.91977    0.86716   2.214  0.02684 *
## speedlow      1.44044    0.84675   1.701  0.08892 .
## speedmedium   0.07744    0.61573   0.126  0.89992
## mxPH         -0.24681    5.41014  -0.046  0.96361
## mnO2          1.16712    0.91867   1.270  0.20393
## Cl           -0.36364    0.37655  -0.966  0.33418
## NO3          -0.15681    0.37182  -0.422  0.67321
## NH4           0.38282    0.26290   1.456  0.14535
## oPO4         -0.97837    0.48170  -2.031  0.04225 *
## PO4          -0.15579    0.58560  -0.266  0.79021
## Chla         -0.83758    0.28916  -2.897  0.00377 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 202.69  on 149  degrees of freedom
## Residual deviance: 113.73  on 134  degrees of freedom
## AIC: 145.73
##
## Number of Fisher Scoring iterations: 6
```

```
a1_test = predict(glm.fit, algae.test, type="response") %>% round
a1_train = predict(glm.fit, algae.train, type = "response") %>% round
calc_error_rate(a1_test, algae.test$a1)
```

```
## [1] 0.3
```

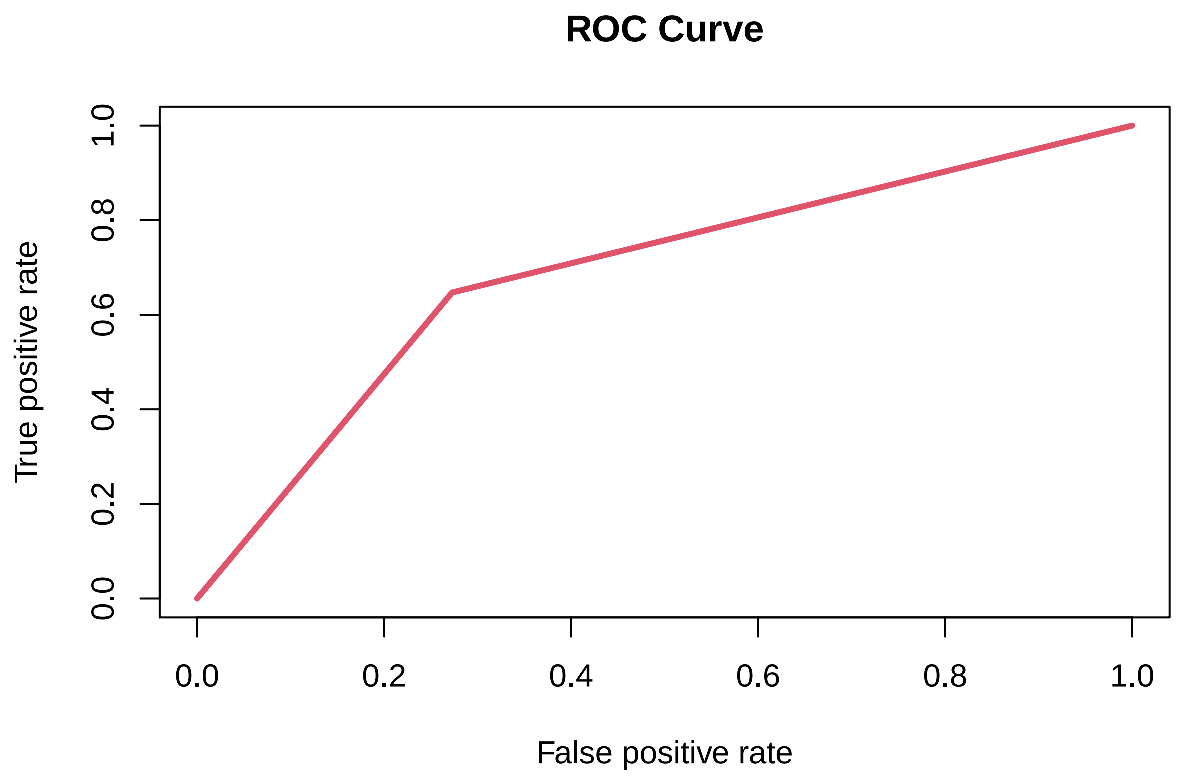
```
calc_error_rate(a1_train, algae.train$a1)
```

```
## [1] 0.2
```

The test and training errors for our model respectively are 0.3 and 0.2. The class labels can be found in the a1_test and a1_train variables.

4.

```
a1_test_pred = prediction(a1_test, algae.test$a1)
perf = performance(a1_test_pred, measure = "tpr", x.measure = "fpr")
plot(perf, col = 2, lwd = 3, main = "ROC Curve")
```



```r
auc = performance(a1_test_pred, "auc")@y.values
auc
```

```
## [[1]]
## [1] 0.6871658
```

The AUC is 0.6871658.

Fundamentals of the bootstrap

1.
   Given a sample of size n, the probability that any observation j is not in a bootstrap sample is $(1-\frac{1}{n})^n$. Let's say that S is our sample that was given with size n. We draw bootstrap samples B from this sample with replacement. So any of the observations of B can be any of the n observations in S, and they can be each be picked n^n different ways (n observation, n amount of times) each given an equal likely chance to be picked. If we say observation j is an observation that is in S but not in B, that means we didn't pick it. Thus instead of n observations being able to be picked from S, we now have n - 1 obseravtions that are able to be picked from S for B. So we have $(n-1)^n$ possible ways to pick observations because we have n-1 observations that can be picked an n number of times (size of B). And so the probabilty that j won't be in B is the number of ways we won't pick j over number of total ways we can pick any observation in S which is $(1-\frac{1}{n})^n$.

2.

For n = 1000 we get:

$$\begin{aligned}(1-\frac{1}{n})^n &= (1-\frac{1}{1000})^{1000} \\ &= 0.999^{1000} \\ &= 0.3676954\end{aligned}$$

3.

```
set.seed(174)
n = 1:1000
boostrap_sample = list()
for (i in 1:1000) {
  boostrap_sample[[i]] = sample(n, size = 1000, replace = TRUE)
}
j = c()
for (i in 1:1000) {
  j[i] = (45 %in% unique(boostrap_sample[[i]]))
}

(prob_non = (1 - sum(j)/(length(boostrap_sample))))
```

```
## [1] 0.367
```

```
prob_non
```

```
## [1] 0.367
```

Because the ratio of missing observations in our simulation is relatively similar to our theoretically calculated probability, we can say that our calculation in part 2 is reasonable.

Cross-validation estimate of test error

1.

```
set.seed(123)
dat_indices = sample(1:nrow(dat), 700)
dat.train = dat[dat_indices,]
dat.test = dat[-dat_indices,]
dat.train.glm.fit = glm(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume, data = dat.train, family
dat.prob.test = predict(dat.train.glm.fit, dat.test, type = "response") %>% round
calc_error_rate(dat.prob.test, dat.test)
```

```
## [1] 0.9244156
```

The error rate of this model on the test data is 0.9244156.

2.

```
library(reshape2)
```

```
##
## Attaching package: 'reshape2'
## The following object is masked from 'package:tidyr':
##
##     smiths
```

```
do.chunk <- function(chunkid, folddef, dat, ...){
  # Get training index
  train = (folddef!=chunkid)

  # Get training set and validation set
  dat.train = dat[train, ]
  dat.val = dat[-train, ]

  # Train logistic regression model on training data
  fit.train = glm(Direction ~ ., family = binomial, data = dat.train)

  # get predicted value on the validation set
  pred.val = predict(fit.train, newdata = dat.val, type = "response")
  pred.val = ifelse(pred.val > .5, 1,0)

  data.frame(fold = chunkid,
             val.error = mean(pred.val != dat.val$Direction))
}
# Specify we want a 10-fold CV
nfold = 10

# cut: divides all training observations into 3 intervals;
# labels = FALSE instructs R to use integers to code different intervals
folds = cut(1:nrow(dat.train), breaks = nfold, labels = FALSE) %>% sample()

# Set error.folds (a vector) to save validation errors in future
error.folds = NULL

# Give possible number of nearest neighbours to be considered
allK = 1:50

set.seed(123)

# Loop through different number of neighbors
for (k in allK) {
  # Loop through different chunk id
  for (j in seq(10)){
    tmp = do.chunk(chunkid = j, folddef = folds, dat = dat, k = k)
    tmp$neighbors = k
    error.folds = rbind(error.folds, tmp)
  }
}
```

```
# Transform the format of error.folds for further convenience
errors = melt(error.folds, id.vars=c('fold', 'neighbors'), value.name = "error")

# Choose the number of neighbors which minimizes validation error
#val.error.means = errors %>%
  # Select all rows of validation errors
#  filter(variable=='val.error') %>%
  # Group the selected data frame by neighbors
#  group_by(neighbors, variable) %>%
#  print(val.error.means)
  # Calculate CV error rate for each k
#  summarise_each(funs(mean), error) %>%
  # Remove existing group
#  ungroup() %>%
#  filter(error==min(error))

# Best number of neighbors
# if there is a tie, pick larger number of neighbors for simpler model
#numneighbor = max(val.error.means$neighbors)
#numneighbor
```