HW02_Chenxin

2024-02-20

```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.3     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```r
library(ggplot2)
library(modelr)
library(rsample)
library(mosaic)
```

```
## Registered S3 method overwritten by 'mosaic':
##   method                           from
##   fortify.SpatialPolygonsDataFrame ggplot2
##
## The 'mosaic' package masks several functions from core packages in order to add
## additional features.  The original behavior of these functions should not be affected by this.
##
## Attaching package: 'mosaic'
##
## The following object is masked from 'package:Matrix':
##
##     mean
##
## The following object is masked from 'package:modelr':
##
##     resample
##
## The following objects are masked from 'package:dplyr':
##
##     count, do, tally
##
## The following object is masked from 'package:purrr':
##
##     cross
##
## The following object is masked from 'package:ggplot2':
##
##     stat
```

```
##
## The following objects are masked from 'package:stats':
##
##     binom.test, cor, cor.test, cov, fivenum, IQR, median, prop.test,
##     quantile, sd, t.test, var
##
## The following objects are masked from 'package:base':
##
##     max, mean, min, prod, range, sample, sum
```

```r
library(MASS)
```

```
##
## Attaching package: 'MASS'
##
## The following object is masked from 'package:dplyr':
##
##     select
```

```r
library(caret) # For data splitting and preprocessing
```

```
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:mosaic':
##
##     dotPlot
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```r
library(pROC) # For ROC curve analysis
```

```
## Type 'citation("pROC")' for a citation.
##
## Attaching package: 'pROC'
##
## The following objects are masked from 'package:mosaic':
##
##     cov, var
##
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
library(class)
library(kknn)
```

```
##
## Attaching package: 'kknn'
##
## The following object is masked from 'package:caret':
##
##     contr.dummy
```

```r
library(foreach)
```

```
##
## Attaching package: 'foreach'
##
## The following objects are masked from 'package:purrr':
##
##     accumulate, when
```

```r
library(doParallel)
```

```
## Loading required package: iterators
## Loading required package: parallel
```

```r
library(ModelMetrics)
```

```
##
## Attaching package: 'ModelMetrics'
##
## The following object is masked from 'package:pROC':
##
##     auc
##
## The following objects are masked from 'package:caret':
##
##     confusionMatrix, precision, recall, sensitivity, specificity
##
## The following objects are masked from 'package:modelr':
##
##     mae, mse, rmse
##
## The following object is masked from 'package:base':
##
##     kappa
```

```r
library(glmnet) # For lasso regression
```

```
## Loaded glmnet 4.1-8
```

Q1

```r
data(SaratogaHouses)
head(SaratogaHouses)
```

```
##    price lotSize age landValue livingArea pctCollege bedrooms fireplaces
## 1 132500    0.09  42     50000        906         35        2          1
## 2 181115    0.92   0     22300       1953         51        3          0
## 3 109000    0.19 133      7300       1944         51        4          1
## 4 155000    0.41  13     18700       1944         51        3          1
## 5  86060    0.11   0     15000        840         51        2          0
## 6 120000    0.68  31     14000       1152         22        4          1
##   bathrooms rooms          heating     fuel              sewer waterfront
## 1       1.0     5         electric electric             septic         No
## 2       2.5     6 hot water/steam      gas             septic         No
## 3       1.0     8 hot water/steam      gas public/commercial         No
```

```
## 4       1.5     5         hot air      gas           septic         No
## 5       1.0     3         hot air      gas public/commercial         No
## 6       1.0     8         hot air      gas           septic         No
##   newConstruction centralAir
## 1              No         No
## 2              No         No
## 3              No         No
## 4              No         No
## 5             Yes        Yes
## 6              No         No
```

```
# Data preprocessing
SaratogaHouses_1 <- SaratogaHouses %>%
  mutate(
    log_price = log(price),
    sqr_bedrooms = bedrooms^2,
    interaction_term = bedrooms * bathrooms
  )

# Split into training and testing sets
set.seed(6666) # for reproducibility
saratoga_split <- initial_split(SaratogaHouses_1, prop = 0.8)
saratoga_train <- training(saratoga_split)
saratoga_test <- testing(saratoga_split)

# (1) baseline medium model
lm_medium = lm(price ~ lotSize + age + livingArea + pctCollege + bedrooms +
        fireplaces + bathrooms + rooms + heating + fuel + centralAir, data=saratoga_train)
coef(lm_medium) %>% round(0)
```

```
##            (Intercept)                lotSize                    age
##                  24001                   9477                     45
##             livingArea             pctCollege               bedrooms
##                     97                    368                 -17127
##             fireplaces              bathrooms                  rooms
##                  -1252                  20756                   3284
## heatinghot water/steam        heatingelectric            fuelelectric
##                 -12537                  -1541                 -12714
##                fueloil           centralAirNo
##                  -6381                 -15691
```

```
# Predictions out of sample
# Root mean squared error
rmse(lm_medium, saratoga_test)
```

```
## [1] 65444.27
```

```
# (2) forward selection
lm_forward <- step(lm(price ~ 1, data = saratoga_train),
                   scope = ~ .^2 + landValue + sewer + newConstruction + waterfront + log_price + sqr_be
                   direction = 'forward',
                   trace = 0)
rmse_forward <- rmse(lm_forward, saratoga_test)
print(rmse_forward)
```

```
## [1] 33320.17
```

```r
# (3) backward selection
lm0 = lm(price ~ 1, data=saratoga_train)
lm_backward = step(lm0, direction='backward',
    scope=~(lotSize + age + livingArea + pctCollege + bedrooms +
              fireplaces + bathrooms + rooms + heating + fuel + centralAir)^2 +
        landValue + sewer + newConstruction + waterfront + log_price + sqr_bedrooms + interaction_term)
```

```
## Start:  AIC=31785.61
## price ~ 1
```

```r
rmse(lm_backward, saratoga_test)
```

```
## [1] 98630.43
```

```r
# Average
n = nrow(SaratogaHouses_1)
# Preallocate the vector to store the RMSE values
rmse_vals = do(100)*{
  # re-split into train and test cases
  n_train = round(0.8*n)  # round to nearest integer
  n_test = n - n_train
  train_cases = sample.int(n, n_train, replace=FALSE)
  test_cases = setdiff(1:n, train_cases)
  saratoga_train = SaratogaHouses_1[train_cases,]
  saratoga_test = SaratogaHouses_1[test_cases,]
  # fit to this training set
  lm_s1=lm(price ~ lotSize + age + livingArea + pctCollege + bedrooms +
        fireplaces + bathrooms + rooms + heating + fuel + centralAir, data=saratoga_train)
  # predict on this testing set
  yhat_test_s1 = predict(lm_s1, saratoga_test)
  c(rmse(saratoga_test$price, yhat_test_s1))
}
```

```
## Using parallel package.
##   * Set seed with set.rseed().
##   * Disable this message with options(`mosaic:parallelMessage` = FALSE)
```

```r
colMeans(rmse_vals)
```

```
##   result
## 66978.25
```

```r
# Standardize features, not including the price (target variable)
set.seed(6666)

preProcValues <- preProcess(SaratogaHouses[, -which(names(SaratogaHouses) == "price")], method = c("cen
SaratogaHouses_processed <- predict(preProcValues, SaratogaHouses)

# Create k-fold cross-validation folds
SaratogaHouses_folds <- crossv_kfold(SaratogaHouses_processed, k = 5)

# Range of k values for KNN
k_grid <- c(2, 4, 6, 8, 10, 15, 20, 25, 30, 35, 40, 45, 50, 60, 70, 80, 90, 100, 125, 150, 175, 200, 25

# Register parallel backend
registerDoParallel(cores = parallel::detectCores())
```

```r
# Perform KNN and calculate RMSE
# Perform KNN across different values of k and calculate RMSE
cv_grid = foreach(k=k_grid, .combine = 'rbind') %dopar% {
  models=map(SaratogaHouses_folds$train, ~ knnreg(price ~ lotSize + age + livingArea + pctCollege + bedi

  errs=map2_dbl(models, SaratogaHouses_folds$test, modelr::rmse)
  c(k=k, err = mean(errs), std_err=sd(errs)/sqrt(5))
} %>% as.data.frame

mink_saratogahouse=cv_grid%>%
  arrange(err) %>%
  head(1)

mink_saratogahouse
```

```
##          k      err  std_err
## result.4 8 62202.19 1653.298
```

Which model seems to do better at achieving lower out-of-sample mean-squared error?

Answer: The forward selection linear model has the lowest RMSE and therefore appears to be the best model.The KNN model with k=8 neighbors has a moderately high RMSE compared to the forward selection linear model but lower than the backward selection linear model.

```
# Q4 Mushroom classification

```r
mush = read.csv('/Users/vita/Desktop/mushrooms.csv')
mush = na.omit(mush)
# Remove columns with only one unique value (including factors with one level)
mush = mush[sapply(mush, function(x) length(unique(x)) > 1)]
head(mush,)
```

```
##   class cap.shape cap.surface cap.color bruises odor gill.attachment
## 1     p         x           s         n       t    p               f
## 2     e         x           s         y       t    a               f
## 3     e         b           s         w       t    l               f
## 4     p         x           y         w       t    p               f
## 5     e         x           s         g       f    n               f
## 6     e         x           y         y       t    a               f
##   gill.spacing gill.size gill.color stalk.shape stalk.root
## 1            c         n          k           e          e
## 2            c         b          k           e          c
## 3            c         b          n           e          c
## 4            c         n          n           e          e
```
```

```
## 5             w         b          k           t          e
## 6             c         b          n           e          c
##   stalk.surface.above.ring stalk.surface.below.ring stalk.color.above.ring
## 1                        s                        s                      w
## 2                        s                        s                      w
## 3                        s                        s                      w
## 4                        s                        s                      w
## 5                        s                        s                      w
## 6                        s                        s                      w
##   stalk.color.below.ring veil.color ring.number ring.type spore.print.color
## 1                      w          w           o         p                 k
## 2                      w          w           o         p                 n
## 3                      w          w           o         p                 n
## 4                      w          w           o         p                 k
## 5                      w          w           o         e                 n
## 6                      w          w           o         p                 k
##   population habitat
## 1          s       u
## 2          n       g
## 3          n       m
## 4          s       u
## 5          a       g
## 6          n       g
```

```r
# Remove columns with only one unique value (including factors with one level)
mush <- mush[sapply(mush, function(x) length(unique(x)) > 1)]

# dummy
dummy_vars <- model.matrix(~ . - 1, data = mush)

mush_encoded <- as.data.frame(dummy_vars)
mush_encoded$class <- mush$class # Add class variable back

predictors <- mush[, names(mush) != "class"]
predictors_encoded <- model.matrix(~ . + 0, data = predictors) # + 0 to exclude intercept

# Prepare the class vector for partitioning and modeling
class_vector <- mush$class

# Use createDataPartition to split the dataset
set.seed(123) # For reproducibility
training_indices <- createDataPartition(class_vector, p = 0.8, list = FALSE)
```

```r
# Split the encoded predictors and class vector into training and testing sets
x_train <- predictors_encoded[training_indices, ]
x_train <- as.matrix(x_train)
y_train <- class_vector[training_indices]
x_test <- predictors_encoded[-training_indices, ]
y_test <- class_vector[-training_indices]
```

```r
# Verify that predictors have non-zero variance for glmnet
apply(x_train, 2, function(column) var(column, na.rm = TRUE))
```

```
##               cap.shapeb                cap.shapec                cap.shapef
##             0.0508172618              0.0006151005              0.2380379467
```

```
##             cap.shapek              cap.shapes              cap.shapex
##           0.0892744209            0.0039846130            0.2477488608
##           cap.surfaceg            cap.surfaces            cap.surfacey
##           0.0004613964            0.2139913596            0.2402207203
##              cap.colorc              cap.colore              cap.colorg
##           0.0058128706            0.1505557068            0.1757078841
##              cap.colorn              cap.colorp              cap.colorr
##           0.2006785423            0.0181234983            0.0019963071
##              cap.coloru              cap.colorw              cap.colory
##           0.0024558571            0.1117476298            0.1151592789
##               bruisest                   odorc                   odorf
##           0.2428776025            0.0228412892            0.1951299253
##                  odorl                   odorm                   odorn
##           0.0452855232            0.0042897961            0.2459615089
##                  odorp                   odors                   odory
##           0.0298270740            0.0681410394            0.0639171470
##       gill.attachmentf           gill.spacingw              gill.sizen
##           0.0251820043            0.1347343615            0.2127540332
##            gill.colore             gill.colorg             gill.colorh
##           0.0112567377            0.0843013481            0.0830462438
##            gill.colork             gill.colorn             gill.coloro
##           0.0484755288            0.1128895806            0.0083912318
##            gill.colorp             gill.colorr             gill.coloru
##           0.1490945353            0.0029149810            0.0573553564
##            gill.colorw             gill.colory            stalk.shapet
##           0.1252472925            0.0109559104            0.2454346586
##            stalk.rootb             stalk.rootc             stalk.roote
##           0.2489644797            0.0629867080            0.1196418621
##            stalk.rootr stalk.surface.above.ringk stalk.surface.above.rings
##           0.0235740646            0.2070872679            0.2314977452
## stalk.surface.above.ringy stalk.surface.below.ringk stalk.surface.below.rings
##           0.0026089458            0.2037070555            0.2393802360
## stalk.surface.below.ringy   stalk.color.above.ringc   stalk.color.above.ringe
##           0.0344236628            0.0042897961            0.0117076237
##   stalk.color.above.ringg   stalk.color.above.ringn   stalk.color.above.ringo
##           0.0653745783            0.0528721342            0.0231345414
##   stalk.color.above.ringp   stalk.color.above.ringw   stalk.color.above.ringy
##           0.1779557802            0.2477782144            0.0010759288
##   stalk.color.below.ringc   stalk.color.below.ringe   stalk.color.below.ringg
##           0.0042897961            0.0117076237            0.0665627611
##   stalk.color.below.ringn   stalk.color.below.ringo   stalk.color.below.ringp
##           0.0587037296            0.0231345414            0.1780383728
##   stalk.color.below.ringw   stalk.color.below.ringy             veil.coloro
##           0.2487200161            0.0033736788            0.0120079776
##            veil.colorw             veil.colory            ring.numbero
##           0.0241594328            0.0010759288            0.0694510635
##           ring.numbert              ring.typef              ring.typel
##           0.0657710654            0.0065726325            0.1359848261
##             ring.typen              ring.typep       spore.print.colorh
##           0.0042897961            0.2497308225            0.1621355001
##     spore.print.colork      spore.print.colorn      spore.print.coloro
##           0.1798434315            0.1817864524            0.0058128706
##     spore.print.colorr      spore.print.coloru      spore.print.colorw
##           0.0088448164            0.0056607762            0.2059316818
```

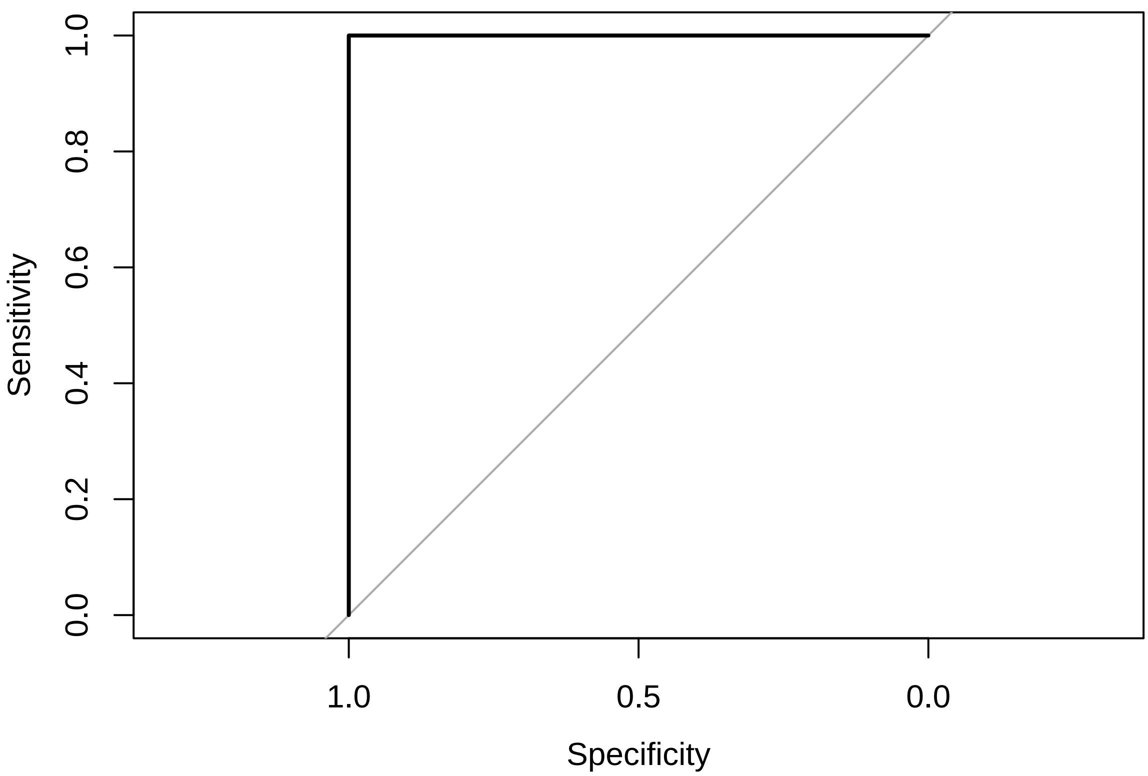
```
##       spore.print.colory            populationc             populationn
##             0.0064207748           0.0405239859            0.0443096808
##              populations            populationv             populationy
##             0.1303040468           0.2500126883            0.1684937091
##                 habitatg               habitatl                habitatm
##             0.1946964385           0.0909971001            0.0335654953
##                 habitatp               habitatu                habitatw
##             0.1201968587           0.0417897191            0.0238668434
```

```
# Lambda selection via cross-validation
cv_model <- cv.glmnet(x_train, y_train, family="binomial", alpha=1) # alpha=1 for lasso penalty

# Train the lasso model using the optimal lambda found
lasso_model <- glmnet(x_train, y_train, family = "binomial", alpha = 1, lambda = cv_model$lambda.min)

# Use the lambda that gives the minimum mean cross-validated error
best_lambda <- cv_model$lambda.min

# Predict probabilities on the test set
predictions <- predict(lasso_model, newx = x_test, s = cv_model$lambda.min, type = "response")

# Use the pROC package to generate a ROC curve
roc_curve <- roc(y_test, predictions[,1])
```

```
## Setting levels: control = e, case = p
```

```
## Setting direction: controls < cases
```

```
# Plot the ROC curve
plot(roc_curve)
```



```r
# Find optimal threshold (example method, consider your criteria)
optimal_threshold <- coords(roc_curve, "best", ret="threshold")

# Calculate performance metrics at the optimal threshold
predictions_binary <- ifelse(predictions[,1] > optimal_threshold, 1, 0)
```

```r
# Synthetic test to ensure the process works
synthetic_predictions <- sample(0:1, length(y_test), replace = TRUE)
conf_matrix <- table(Predicted = synthetic_predictions, Actual = y_test)
print(conf_matrix)
```

```
##          Actual
## Predicted   e   p
##         0 413 381
##         1 428 402
```

```r
# Calculate True Positive Rate and False Positive Rate
TPR <- conf_matrix[2,2] / sum(conf_matrix[2,])
FPR <- conf_matrix[1,2] / sum(conf_matrix[1,])

# Print the performance metrics
print(paste("True Positive Rate:", TPR))
```

```
## [1] "True Positive Rate: 0.484337349397590"
```

```r
print(paste("False Positive Rate:", FPR))
```

```
## [1] "False Positive Rate: 0.479848866498741"
```

Write a short report on the best-performing model you can find using lasso-penalized logistic regression. Evaluate the out-of-sample performance of your model using a ROC curve. Based on this ROC curve, recommend a probability threshold for declaring a mushroom poisonous.

Answer: The ROC curve to be a perfect diagonal line, which suggests that the model performs no better than random guessing. It would not be appropriate to recommend a probability threshold, as the model does not discriminate between the classes better than chance.

How well does your model perform at this threshold, as measured by false positive rate and true positive rate?

Answer: A True Positive Rate of approximately 48.13% indicates that the model correctly identifies about 48.13% of the poisonous mushrooms as poisonous. However, a False Positive Rate of approximately 48.29% suggests that roughly 48.29% of the edible mushrooms are incorrectly identified as poisonous. Model Evaluation:

The performance metrics suggest that the model is not performing well since the rates of correct and incorrect classification are nearly equivalent to tossing a coin, which gives no practical advantage over random guessing.