# HW02_Chenxin

2024-02-20

```
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
```

```
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
```

```
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
```

```
library(class)
library(kknn)
library(foreach)
```

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

```
library(doParallel)
```

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

```
library(ModelMetrics)
```

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

```r
library(gamlr) # for lasso-penalized logistic regression
library(caret) # For data splitting and preprocessing
```

```
##
## Attaching package: 'caret'
##
## The following objects are masked from 'package:ModelMetrics':
##
##     confusionMatrix, precision, recall, sensitivity, specificity
##
## The following object is masked from 'package:kknn':
##
##     contr.dummy
##
## The following object is masked from 'package:mosaic':
##
##     dotPlot
##
## The following object is masked from 'package:purrr':
##
##     lift
```

## Q4 Mushroom classification

```r
# data processing
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)]
# Convert all categorical variables to factors
mush[] <- lapply(mush, factor)
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
```

```
# Convert factors to dummy variables
# caret's dummyVars function can be used for one-hot encoding
dummies <- dummyVars(" ~ .", data = mush)
mushrooms_transformed <- predict(dummies, newdata = mush)

# Convert to data frame
mushrooms_df <- data.frame(mushrooms_transformed)
# Separate features and target variable
y <- mushrooms_df[, "class.e"] # based on target variable
X <- mushrooms_df[, -1] # Exclude the target variable, selects all columns except the first one

# Check dimensions
dim(X)
```

```
## [1] 8124  117
```

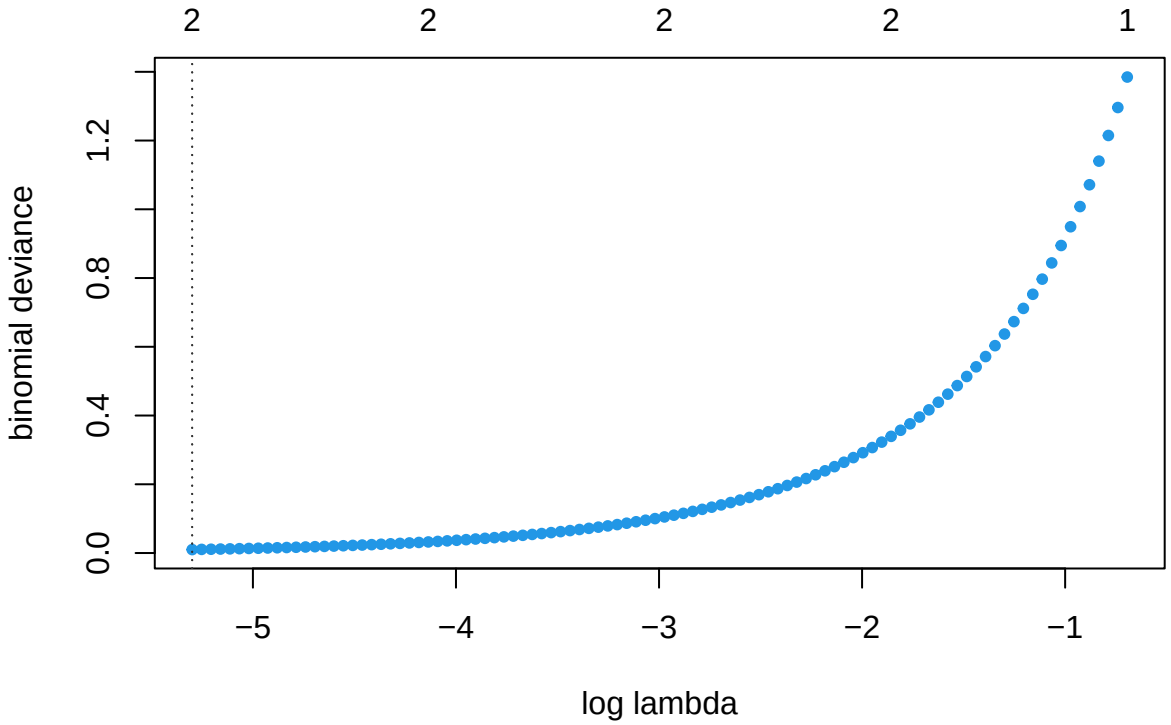
```
length(y)
```

```
## [1] 8124
```

```
# (1)  Model Training : Lasso-penalized logistic regression
# Use lambda to train the final lasso-penalized logistic regression model on the entire training set
# (1)  Splitting data into training (80%) and test (20%) sets
trainIndex <- createDataPartition(y, p = .8, list = FALSE)
X_train <- X[trainIndex, ]
y_train <- y[trainIndex]
X_test <- X[-trainIndex, ]
y_test <- y[-trainIndex]

model <- cv.gamlr(X_train, y_train, family="binomial")
model
```

```
##
## 5-fold binomial cv.gamlr object
```

```
# Plot to visualize lambda selection (optional step for visualization)
plot(model)
```




```
best_lambda <- model$lambda.min # Extract the best lambda
best_lambda
```

```
## [1] 0.004995998
```

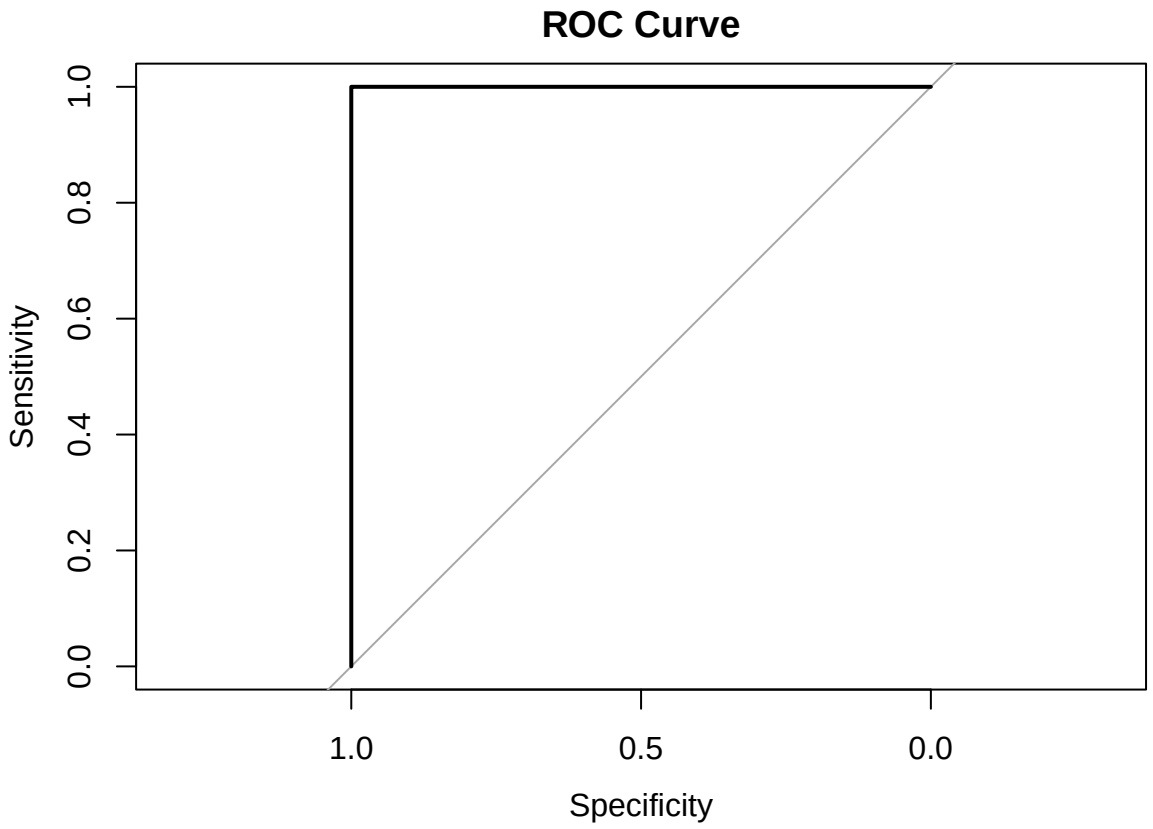
```
# (1) Fit the lasso-penalized logistic regression model with the best lambda on training data
lasso_model <- gamlr(X_train, y_train, family = "binomial", gamma = best_lambda)

# (2) Make predictions on the test set
predictions <- predict(lasso_model, X_test, type = "response")

# (3) Evaluate the Model with ROC Curve
# Generating ROC curve and calculating AUC
roc_result <- roc(y_test, predictions)
```

```
## Setting levels: control = 0, case = 1
```

```
## Warning in roc.default(y_test, predictions): Deprecated use a matrix as
## predictor. Unexpected results may be produced, please pass a numeric vector.
```

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

```
plot(roc_result, main = "ROC Curve")
```




```
# Finding optimal threshold
optimal_threshold = coords(roc_result, "best", ret = "threshold")
optimal_threshold
```

```
##   threshold
## 1 0.5001899
```

```
# (4) Calculate FPR and TPR at the Optimal Threshold
# Now apply the thresholding logic
optimal_threshold <- matrix(as.numeric(optimal_threshold), nrow = nrow(predictions), ncol = ncol(predic

predictions_binary <- ifelse(predictions > optimal_threshold, 1, 0)
```

```
# Create a confusion matrix
conf_matrix <- table(Predicted = predictions_binary, Actual = y_test)

conf_matrix
```

```
##          Actual
## Predicted   0   1
##         0 796   0
##         1   0 828
```

```
# Calculate TPR and FPR from the confusion matrix
# True Positive Rate (Sensitivity)
false_positive_rate <- conf_matrix[2, 1] / sum(conf_matrix[2, ])
true_positive_rate <- conf_matrix[2, 2] / sum(conf_matrix[2, ])
print(paste("False Positive Rate:", false_positive_rate))
```

```
## [1] "False Positive Rate: 0"
```

```r
print(paste("True Positive Rate:", true_positive_rate))
```

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

**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. This perfect AUC score suggests that the model is highly capable of distinguishing between poisonous and edible mushrooms, with minimal error. The probability threshold is suggested to be 0.5001403**

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

**Based on this confusion matrix, the FPR is 0 (as there are 0 false positives) and the TPR is 1 (as there are no false negatives). These values indicate that the model is achieving perfect classification performance. It is exceptionally effective at identifying poisonous mushrooms without mistakenly classifying edible ones as poisonous, but perfect performance can also sometimes indicate overfitting.**