hw3

Johnstone Tcheou

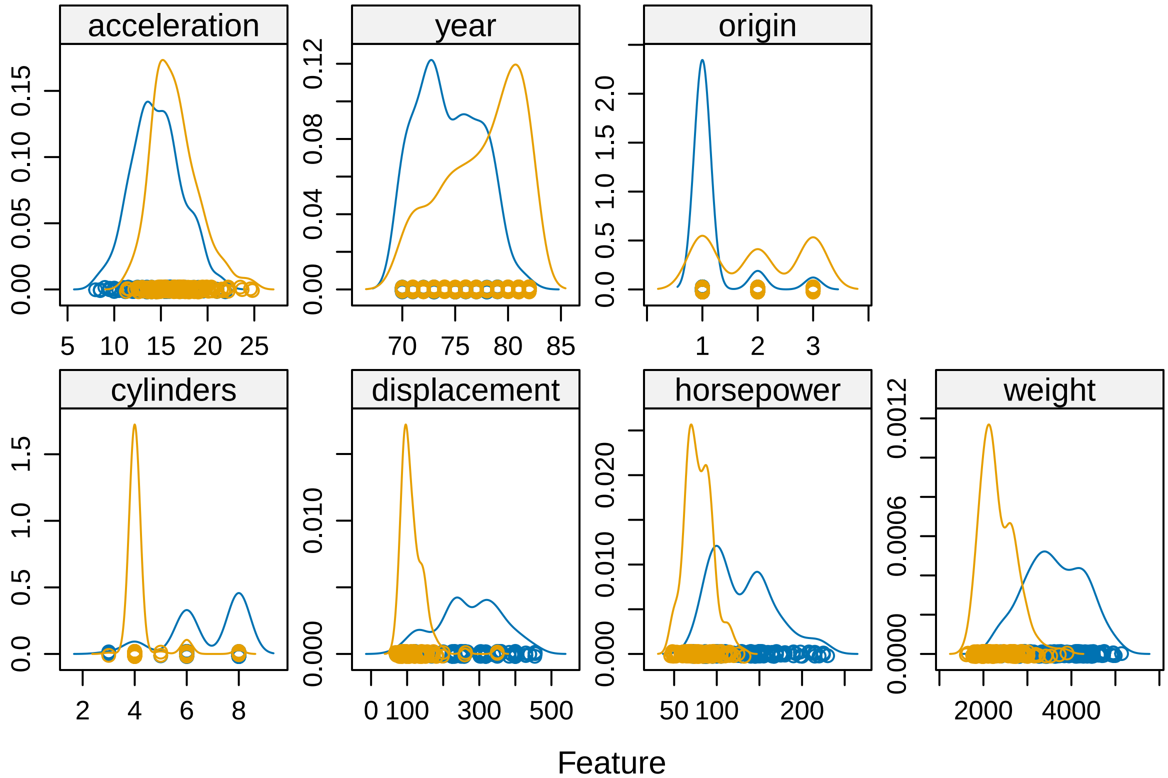
2025-03-24

```r
library(tidyverse)
library(caret)
library(glmnet)
library(mlbench)
library(tidymodels)
library(pROC)
library(pdp)
library(vip)
library(MASS)
library(earth)
library(plotmo)
```

Data import, exploration, and split

Since there are no `NA` observations, we do not need `na.omit`. We do need to coerce the response variable `mpg_cat` to be a factor before visually exploring the data with `featurePlot` prior to any model fitting. Since most of the predictors are continuous, we can use density plots to best visualize their distributions stratified by levels of the response variable with y axes scaled to each predictor.

```r
set.seed(81063)

auto <-
  read.csv("auto.csv") |>
  na.omit() |>
  mutate(
    mpg_cat = factor(mpg_cat, levels = c("low", "high"))
  )

featurePlot(
  x = auto[, 1:7],
  y = auto$mpg_cat,
  scales = list(x = list(relation = "free"),
                        y = list(relation = "free")),
            plot = "density"
  )
```



When stratified by `mpg_cat`, most variables have pretty different distributions, except for `acceleration`. These may indicate potential variable informativeness towards predicting `mpg_cat`.

```
set.seed(81063)
auto_split <- initial_split(auto, prop = 0.70)

training_data <- training(auto_split)
testing_data <- testing(auto_split)

training_predictors <- training_data[, -ncol(training_data)]
training_response <- training_data$mpg_cat

testing_predictors <- testing_data[, -ncol(testing_data)]
testing_response <- testing_data$mpg_cat
```

Question a

Logistic regression

We can use the `contrasts` function to ensure we are using the correct predictor labels. Afterwards, we can fit a logistic regression model to the training data and get predicted probabilities with the testing data to evaluate the model.

```r
set.seed(81063)
contrasts(auto$mpg_cat)
```

```
##      high
## low     0
## high    1
```

```r
ctrl <- trainControl(
  method = "cv",
  number = 10,
  summaryFunction = twoClassSummary,
  classProbs = TRUE
)

logit <- train(
  x = training_data[, -ncol(training_data)],
  y = training_response,
  method = "glm",
  metric = "ROC",
  trControl = ctrl
)

summary(logit)
```

```
## 
## Call:
## NULL
## 
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)    
## (Intercept)  -13.883316   7.133612  -1.946  0.05163 .  
## cylinders     -0.092058   0.525410  -0.175  0.86091    
## displacement   0.005974   0.014922   0.400  0.68891    
## horsepower    -0.061502   0.031976  -1.923  0.05443 .  
## weight        -0.004089   0.001411  -2.898  0.00376 ** 
## acceleration  -0.125432   0.188102  -0.667  0.50488    
## year           0.416420   0.090016   4.626 3.73e-06 ***
## origin         0.695208   0.467909   1.486  0.13734    
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## (Dispersion parameter for binomial family taken to be 1)
## 
##     Null deviance: 379.48  on 273  degrees of freedom
## Residual deviance: 109.21  on 266  degrees of freedom
## AIC: 125.21
## 
## Number of Fisher Scoring iterations: 8
```

```r
coef(logit$finalModel)
```

```
##   (Intercept)     cylinders  displacement    horsepower        weight
```

```
## -13.883315553  -0.092057931   0.005973898  -0.061502176  -0.004089208
##  acceleration          year        origin
##  -0.125432046   0.416420203   0.695208015
```

```r
logit_pred_prob <- predict(
  logit,
  newdata = testing_data,
  type = "raw"
)

(logit_confusion_matrix <- confusionMatrix(
  data = logit_pred_prob,
  reference = testing_data$mpg_cat,
  positive = "high"
))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction low high
##       low   46    3
##       high   8   61
##
##                Accuracy : 0.9068
##                  95% CI : (0.8393, 0.9525)
##     No Information Rate : 0.5424
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8108
##
##  Mcnemar's Test P-Value : 0.2278
##
##             Sensitivity : 0.9531
##             Specificity : 0.8519
##          Pos Pred Value : 0.8841
##          Neg Pred Value : 0.9388
##              Prevalence : 0.5424
##          Detection Rate : 0.5169
##    Detection Prevalence : 0.5847
##       Balanced Accuracy : 0.9025
##
##        'Positive' Class : high
##
```

```r
# logit <- glm(
#   mpg_cat ~ .,
#   data = training_data,
#   family = binomial(link = "logit")
# )
#
# logit_pred_prob <- predict(logit, newdata = testing_data, type = "response")
# logit_pred <- rep("low", length(logit_pred_prob))
# logit_pred[logit_pred_prob > 0.5] <- "high"
```

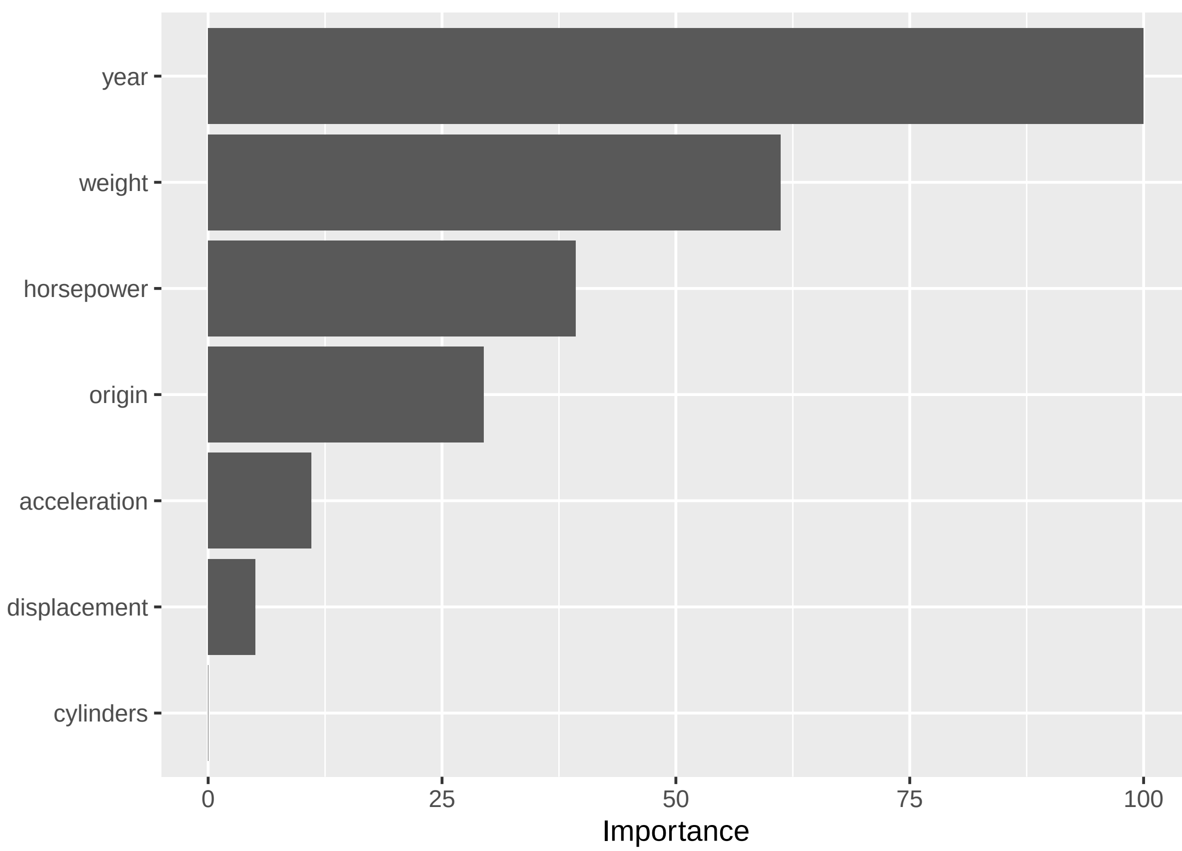
```
#
#
# confusionMatrix(data = factor(logit_pred, levels = c("low", "high")),
#                 reference = testing_data$mpg_cat,
#                 positive = "high")
#
# roc_logit <- roc(training$mpg_cat, logit_pred_prob)
#
# plot(roc_logit, legacy.axes = TRUE, print.auc = TRUE)
# #plot(smooth(roc_logit), col = 4, add = TRUE)
#
# vip(logit)
```

The fitted logistic regression model has 7 predictors, for `cylinders`, `displacement`, `horsepower`, `weight`, `acceleration`, `year`, and `origin`. When estimating predictions against the training dataset, we can get the confusion matrix to assess the robustness of the model's classification.

With an accuracy of 0.9067797, it is greater than the no information rate, which means that this classifier is meaningful. Additionally, the kappa is 0.8108423. Being greater than 0.6, it indicates good agreement.

Are there redundant predictors in your model?

```
vip(logit)
```

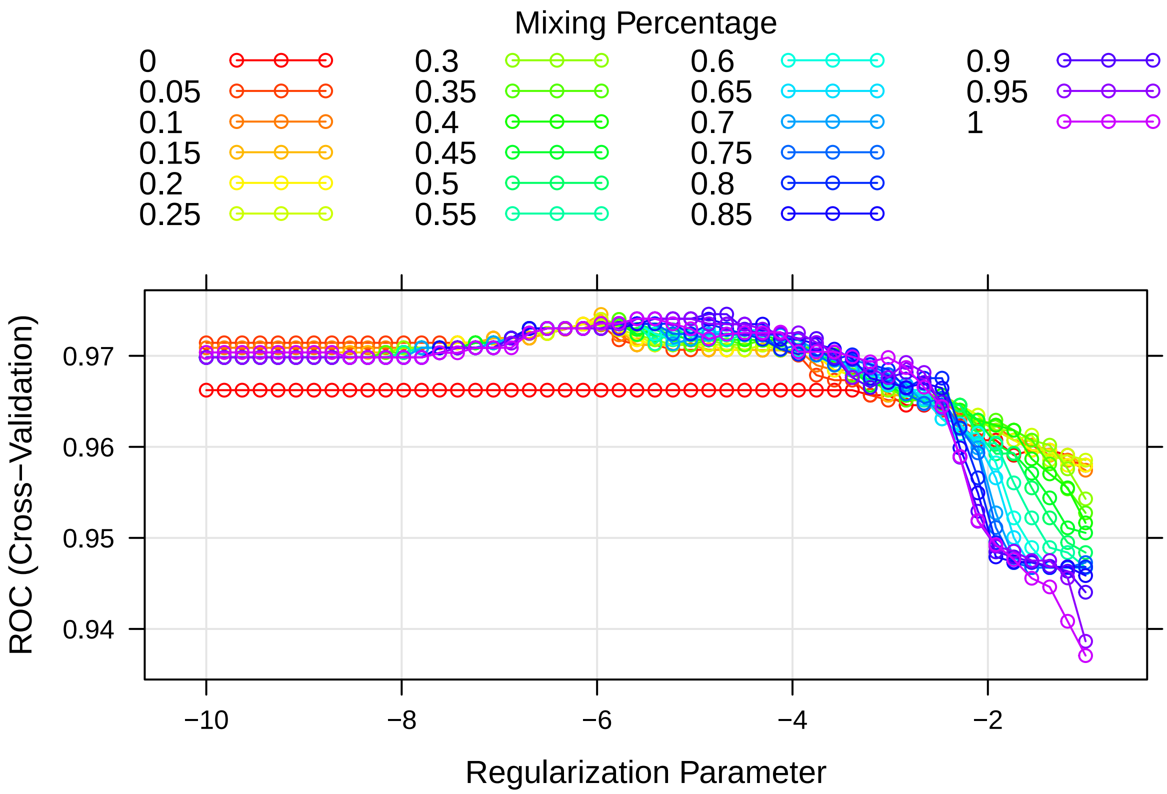


Variable importance is a metric assigned to each predictor, where the higher the number indicates a more important variable for predicting an outcome in a model. Variables with an importance of 0 are not actually included in the regression function and are unimportant/redundant. Based on the graph of variables and their importance, `cylinders` appears to be a redundant variable in the predictor, with an importance of 0.

```r
set.seed(81063)

glmnGrid <- expand.grid(
  .alpha = seq(0, 1, length = 21),
  .lambda = exp(seq(-10, -1, length = 50))
)

penalized_logit <- train(
  x = training_data[, -ncol(training_data)],
  y = training_response,
  method = "glmnet",
  tuneGrid = glmnGrid,
  metric = "ROC",
  trControl = ctrl
)

penalized_logit$bestTune
```

```
##     alpha     lambda
## 930   0.9 0.00933981
```

```r
myCol <- rainbow(25)
myPar <- list(superpose.symbol = list(col = myCol),
              superpose.line = list(col = myCol))

plot(penalized_logit, par.settings = myPar, xTrans = function(x) log(x))
```




```r
coef(penalized_logit$finalModel)
```

```
## 8 x 88 sparse Matrix of class "dgCMatrix"

##    [[ suppressing 88 column names 's0', 's1', 's2' ... ]]

##
## (Intercept)  -0.07302514  0.3547372658  0.7502171822  1.1065478750
## cylinders     .          -0.0161963825 -0.0455163960 -0.0690438652
## displacement  .           .            -0.0001563238 -0.0004276565
## horsepower    .           .             .             .
## weight        .          -0.0001130509 -0.0001810620 -0.0002395066
## acceleration  .           .             .             .
## year          .           .             .             .
## origin        .           .             .             .
##
## (Intercept)   1.4456748759  1.7704440757  2.0831121837  2.3854835749
## cylinders    -0.0909734004 -0.1114024390 -0.1304080580 -0.1480575648
## displacement -0.0006786052 -0.0009134881 -0.0011354979 -0.0013471482
## horsepower    .             .             .             .
## weight       -0.0002969440 -0.0003538833 -0.0004107361 -0.0004678168
## acceleration  .             .             .             .
## year          .             .             .             .
## origin        .             .             .             .
##
```

```
## (Intercept)   2.6788176523  2.964690291  3.2438760430  3.5171560699
## cylinders    -0.1643274977 -0.179442251 -0.1933770629 -0.2061867841
## displacement -0.0015517426 -0.001748328 -0.0019394257 -0.0021260532
## horsepower    .             .            .             .
## weight       -0.0005253669 -0.000583551 -0.0006424932 -0.0007022676
## acceleration  .             .            .             .
## year          .             .            .             .
## origin        .             .            .             .
##
## (Intercept)   3.6490783141  2.9870904058  2.3396373881  1.7054619609
## cylinders    -0.2173615602 -0.2238302784 -0.2291742670 -0.2335574666
## displacement -0.0022929557 -0.0023697570 -0.0024469397 -0.0025240618
## horsepower    .             .             .             .
## weight       -0.0007638416 -0.0008324814 -0.0009035406 -0.0009769163
## acceleration  .             .             .             .
## year          0.0017457801  0.0136309147  0.0253233650  0.0368476001
## origin        .             .             .             .
##
## (Intercept)   1.083274318  0.472067931 -0.128930000 -0.720295891 -1.302438422
## cylinders    -0.237035737 -0.239664971 -0.241500525 -0.242596899 -0.243007482
## displacement -0.002601275 -0.002678595 -0.002755931 -0.002833094 -0.002909822
## horsepower    .            .            .            .            .
## weight       -0.001052576 -0.001130463 -0.001210494 -0.001292565 -0.001376554
## acceleration  .            .            .            .            .
## year          0.048225087  0.059472220  0.070601074  0.081619999  0.092534093
## origin        .            .            .            .            .
##
## (Intercept)  -1.875620971 -2.439979688 -2.995538617 -3.542222878 -4.079870648
## cylinders    -0.242784372 -0.241978293 -0.240638577 -0.238813219 -0.236548966
## displacement -0.002985788 -0.003060619 -0.003133909 -0.003205232 -0.003274151
## horsepower    .            .            .            .            .
## weight       -0.001462319 -0.001549703 -0.001638534 -0.001728625 -0.001819780
## acceleration  .            .            .            .            .
## year          0.103345587  0.114054163  0.124657243  0.135150235  0.145526770
## origin        .            .            .            .            .
##
## (Intercept)  -4.554162893 -5.025681019 -5.526355998 -6.015392623 -6.492940059
## cylinders    -0.232076987 -0.226165158 -0.219670149 -0.212982228 -0.206079668
## displacement -0.003287307 -0.003200974 -0.003042472 -0.002876503 -0.002704605
## horsepower   -0.000644693 -0.001773801 -0.003048859 -0.004370160 -0.005730665
## weight       -0.001903990 -0.001983049 -0.002061044 -0.002139484 -0.002218281
## acceleration  .            .            .            .            .
## year          0.155346791  0.165029254  0.174833879  0.184530055  0.194110423
## origin        .            0.009531249  0.028985082  0.048521597  0.068137868
##
## (Intercept)  -6.959026683 -7.413622989 -7.856652894 -8.288002473 -8.707528713
## cylinders    -0.199015526 -0.191843592 -0.184616519 -0.177385277 -0.170198666
## displacement -0.002526545 -0.002342023 -0.002150732 -0.001952411 -0.001746883
## horsepower   -0.007123235 -0.008540674 -0.009975858 -0.011421834 -0.012871895
## weight       -0.002297363 -0.002376652 -0.002456062 -0.002535496 -0.002614845
## acceleration  .            .            .            .            .
## year          0.203567139  0.212891379  0.222073512  0.231103281  0.239969976
## origin        0.087842865  0.107635754  0.127506519  0.147436615  0.167399745
##
```

```
## (Intercept)  -9.115068169 -9.510445411 -9.893481167 -1.026400e+01 -1.062184e+01
## cylinders    -0.163102855 -0.156140978 -0.149352754 -1.427742e-01 -1.364372e-01
## displacement -0.001534085 -0.001314092 -0.001087129 -8.535698e-04 -6.139375e-04
## horsepower   -0.014319641 -0.015759022 -0.017184360 -1.859038e-02 -1.997222e-02
## weight       -0.002693985 -0.002772777 -0.002851070 -2.928705e-03 -3.005511e-03
## acceleration  .            .            .            .             .
## year          0.248662614  0.257170118  0.265481488  2.735860e-01  2.814733e-01
## origin        0.187362750  0.207286610  0.227127519  2.468380e-01  2.663682e-01
##
## (Intercept)  -10.966849032 -1.129891e+01 -11.618582257 -11.922613244
## cylinders     -0.130369389 -1.245941e-01  -0.115868708  -0.104023620
## displacement  -0.000368891 -1.192111e-04   .             .
## horsepower    -0.021325433 -2.264603e-02  -0.023879847  -0.025088514
## weight        -0.003081314 -3.155939e-03  -0.003222245  -0.003282671
## acceleration   .             .             .             .
## year           0.289133646  2.965581e-01   0.303550534   0.310193107
## origin         0.285666697  3.046821e-01   0.320993965   0.334827531
##
## (Intercept)  -12.213907298 -12.492865907 -12.758240945 -13.010168390
## cylinders     -0.092390237  -0.080832851  -0.069626835  -0.058800536
## displacement   .             .             .             .
## horsepower    -0.026263416  -0.027397484  -0.028488384  -0.029534654
## weight        -0.003340894  -0.003397456  -0.003451921  -0.003504218
## acceleration   .             .             .             .
## year           0.316568764   0.322685647   0.328528396   0.334096303
## origin         0.348312435   0.361411042   0.374009305   0.386091105
##
## (Intercept)  -13.248849238 -13.474529660 -13.687498286 -13.888082915
## cylinders     -0.048375281  -0.038368160  -0.028792061  -0.019655783
## displacement   .             .             .             .
## horsepower    -0.030535222  -0.031489414  -0.032396921  -0.033257783
## weight        -0.003554292  -0.003602109  -0.003647647  -0.003690904
## acceleration   .             .             .             .
## year           0.339390179   0.344412179   0.349165737   0.353655478
## origin         0.397645506   0.408665591   0.419148329   0.429094378
##
## (Intercept)  -13.987967118 -13.986268365 -1.397413e+01 -13.934048367
## cylinders     -0.012018120  -0.005794019 -4.025193e-04   .
## displacement   .             .            .              .
## horsepower    -0.034559958  -0.036380052 -3.815406e-02  -0.039803388
## weight        -0.003716820  -0.003723704 -3.728244e-03  -0.003724232
## acceleration  -0.004231384  -0.012700162 -2.098745e-02  -0.029257487
## year           0.357719506   0.361363979  3.647572e-01   0.367681940
## origin         0.439404010   0.450617902  4.611963e-01   0.469321754
##
## (Intercept)  -13.893300406 -13.856826491 -13.819221798 -13.782764140
## cylinders      .             .             .             .
## displacement   .             .             .             .
## horsepower    -0.041359817  -0.042796777  -0.044168731  -0.045458365
## weight        -0.003719557  -0.003715738  -0.003711534  -0.003707441
## acceleration  -0.037082390  -0.044292187  -0.051157311  -0.057596508
## year           0.370403139   0.372946215   0.375314427   0.377518554
## origin         0.476849724   0.483791047   0.490385869   0.496565489
##
```

```
## (Intercept)  -13.751180806 -13.718224935 -13.686491556 -13.656404765
## cylinders      .             .             .             .
## displacement   .             .             .             .
## horsepower    -0.046633939  -0.047758818  -0.048812144  -0.049793496
## weight        -0.003704265  -0.003700618  -0.003697068  -0.003693703
## acceleration  -0.063459216  -0.069055837  -0.074287509  -0.079154487
## year           0.379568476   0.381469787   0.383232310   0.384864315
## origin         0.502199973   0.507564768   0.512575219   0.517233755
##
## (Intercept)  -13.631218144 -13.604950730 -13.579815236 -13.55620636
## cylinders      .             .             .             .
## displacement   .             .             .             .
## horsepower    -0.050674575  -0.051517231  -0.052303135  -0.05303158
## weight        -0.003691222  -0.003688339  -0.003685532  -0.00368289
## acceleration  -0.083523118  -0.087692198  -0.091575549  -0.09517118
## year           0.386373030   0.387767256   0.389054110   0.39024072
## origin         0.521424507   0.525412977   0.529124951   0.53256043
##
## (Intercept)  -13.53709880 -1.355033e+01 -1.358289e+01 -1.361713e+01
## cylinders      .            .             .             .
## displacement   .            2.656235e-04  5.632122e-04  8.463602e-04
## horsepower    -0.05367562 -5.447059e-02 -5.511513e-02 -5.569161e-02
## weight        -0.00368104 -3.702409e-03 -3.729209e-03 -3.755272e-03
## acceleration  -0.09835298 -1.011972e-01 -1.032075e-01 -1.049416e-01
## year           0.39133121  3.931057e-01  3.948990e-01  3.965883e-01
## origin         0.53560935  5.461486e-01  5.569091e-01  5.669991e-01
##
## (Intercept)  -13.648176983 -13.678364930 -13.706673692 -13.731395600
## cylinders      .             .             .             .
## displacement   0.001100387   0.001341465   0.001564076   0.001760489
## horsepower    -0.056215396  -0.056701203  -0.057147259  -0.057552965
## weight        -0.003778925  -0.003801378  -0.003822213  -0.003840739
## acceleration  -0.106525452  -0.107975252  -0.109300804  -0.110526521
## year           0.398131609   0.399581612   0.400924118   0.402136467
## origin         0.576077580   0.584654251   0.592566195   0.599595559
##
## (Intercept)  -13.755445377 -13.777985602 -1.379616e+01 -13.805514794
## cylinders      .             .            -4.606826e-04  -0.008699485
## displacement   0.001947717   0.002120632  2.279664e-03   0.002626361
## horsepower    -0.057930117  -0.058275715 -5.858631e-02  -0.058849047
## weight        -0.003858364  -0.003874705 -3.889128e-03  -0.003908025
## acceleration  -0.111649408  -0.112672860 -1.136317e-01  -0.114641746
## year           0.403276401   0.404329807  4.052713e-01   0.406279149
## origin         0.606261856   0.612408910  6.178446e-01   0.625179400
##
## (Intercept)  -13.811379908 -13.816863319 -13.821853006 -13.826600750
## cylinders     -0.015402707  -0.021422861  -0.027319930  -0.032447779
## displacement   0.002901360   0.003150163   0.003388772   0.003599172
## horsepower    -0.059078376  -0.059290336  -0.059483666  -0.059660610
## weight        -0.003922969  -0.003936570  -0.003949408  -0.003960904
## acceleration  -0.115565430  -0.116416912  -0.117201775  -0.117915323
## year           0.407118543   0.407888547   0.408606496   0.409257397
## origin         0.631017096   0.636337198   0.641353924   0.645833023
##
```

```
## (Intercept)  -13.831047411 -13.835140300 -13.839000953
## cylinders     -0.037022540  -0.041139181  -0.045253521
## displacement   0.003788345   0.003959205   0.004126248
## horsepower    -0.059822975  -0.059971691  -0.060107658
## weight        -0.003971323  -0.003980775  -0.003989850
## acceleration  -0.118567921  -0.119165360  -0.119715472
## year           0.409851522   0.410393635   0.410904044
## origin         0.649888767   0.653567461   0.657092444
```

The logistic regression model that maximizes the AUC has a lambda of 0.0093398 and an alpha of 0.9.

Of the 7 predictors,

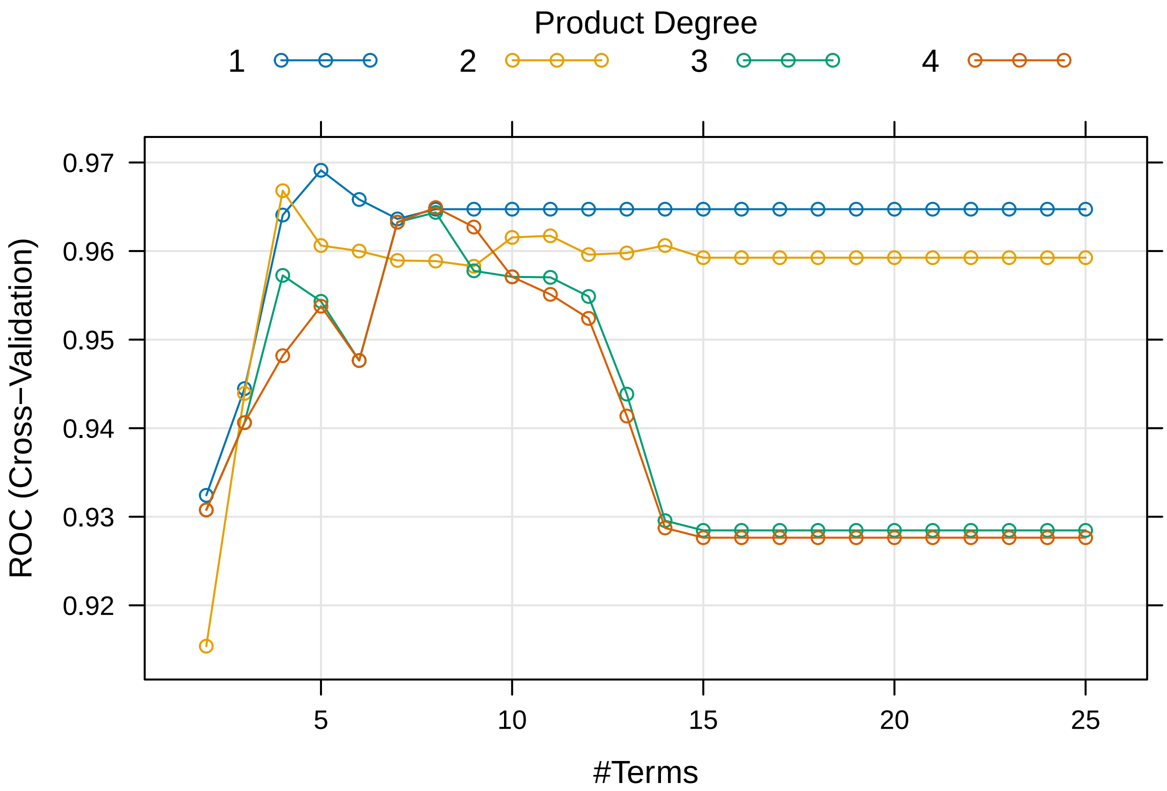
question b

MARS model

Next, we can fit a MARS model to the training data, passing the `preProcess` argument `"scale"` to scale the data.

fitting MARS model gets Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred message

```r
set.seed(81063)

mars <- train(
  x = training_data[1:7],
  y = training_data$mpg_cat,
  method = "earth",
  tuneGrid = expand.grid(
    degree = 1:4,
    nprune = 2:25
  ),
  preProcess = "scale",
  metric = "ROC",
  trControl = ctrl
)

plot(mars)
```




```r
coef(mars$finalModel)
```

```
##              (Intercept)          h(year-19.4294)         h(4.015-weight)
##                -4.561637                 1.949816                4.060890
## h(displacement-1.40162) h(displacement-2.15348)
##                -4.389948                 6.293828
```

```r
mars_pred <- predict(
  mars,
  newdata = training_data
)

(mars_confusion_matrix <- confusionMatrix(
  data = mars_pred,
  reference = training_data$mpg_cat,
  positive = "high"
))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction low high
##       low  127    8
##       high  15  124
##
```

```
##                Accuracy : 0.9161
##                  95% CI : (0.8767, 0.946)
##     No Information Rate : 0.5182
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8322
##
##  Mcnemar's Test P-Value : 0.2109
##
##             Sensitivity : 0.9394
##             Specificity : 0.8944
##          Pos Pred Value : 0.8921
##          Neg Pred Value : 0.9407
##              Prevalence : 0.4818
##          Detection Rate : 0.4526
##    Detection Prevalence : 0.5073
##       Balanced Accuracy : 0.9169
##
##        'Positive' Class : high
##
```

Does the MARS model improve prediction performance compared to logistic regression?

Yes, it does. For one thing, the accuracy is higher - 0.9160584 compared to 0.9067797. Secondly, the kappa is also much higher - 0.8322062 vs 0.8108423, indicating great agreement. However, the ROC AUC is slightly lower, with the best fit MARS with the highest ROC had an ROC (`nprune` = 12, `degree` = 2) of 0.9691209 compared to 0.9703715.

```
mars
```

```
## Multivariate Adaptive Regression Spline
##
## 274 samples
##   7 predictor
##   2 classes: 'low', 'high'
##
## Pre-processing: scaled (7)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 247, 247, 245, 247, 247, 247, ...
## Resampling results across tuning parameters:
##
##   degree  nprune  ROC        Sens       Spec
##   1        2      0.9324176  0.8519048  0.9104396
##   1        3      0.9444689  0.8519048  0.9395604
##   1        4      0.9640659  0.8947619  0.9472527
##   1        5      0.9691209  0.9076190  0.9318681
##   1        6      0.9658242  0.9076190  0.9318681
##   1        7      0.9636264  0.9076190  0.9318681
##   1        8      0.9647253  0.9147619  0.9318681
##   1        9      0.9647253  0.9147619  0.9241758
##   1       10      0.9647253  0.9147619  0.9241758
##   1       11      0.9647253  0.9147619  0.9241758
```

```
##   1        12       0.9647253  0.9147619  0.9241758
##   1        13       0.9647253  0.9147619  0.9241758
##   1        14       0.9647253  0.9147619  0.9241758
##   1        15       0.9647253  0.9147619  0.9241758
##   1        16       0.9647253  0.9147619  0.9241758
##   1        17       0.9647253  0.9147619  0.9241758
##   1        18       0.9647253  0.9147619  0.9241758
##   1        19       0.9647253  0.9147619  0.9241758
##   1        20       0.9647253  0.9147619  0.9241758
##   1        21       0.9647253  0.9147619  0.9241758
##   1        22       0.9647253  0.9147619  0.9241758
##   1        23       0.9647253  0.9147619  0.9241758
##   1        24       0.9647253  0.9147619  0.9241758
##   1        25       0.9647253  0.9147619  0.9241758
##   2         2       0.9153846  0.8161905  0.8873626
##   2         3       0.9439194  0.8519048  0.9472527
##   2         4       0.9668132  0.8876190  0.9620879
##   2         5       0.9606227  0.9080952  0.9401099
##   2         6       0.9600026  0.9076190  0.9324176
##   2         7       0.9589377  0.8938095  0.9543956
##   2         8       0.9588645  0.9290476  0.9467033
##   2         9       0.9582810  0.9290476  0.9472527
##   2        10       0.9615411  0.9295238  0.9395604
##   2        11       0.9617216  0.9223810  0.9472527
##   2        12       0.9595997  0.9157143  0.9472527
##   2        13       0.9597828  0.9157143  0.9395604
##   2        14       0.9606253  0.9157143  0.9395604
##   2        15       0.9592517  0.9157143  0.9395604
##   2        16       0.9592517  0.9157143  0.9395604
##   2        17       0.9592517  0.9157143  0.9395604
##   2        18       0.9592517  0.9157143  0.9395604
##   2        19       0.9592517  0.9157143  0.9395604
##   2        20       0.9592517  0.9157143  0.9395604
##   2        21       0.9592517  0.9157143  0.9395604
##   2        22       0.9592517  0.9157143  0.9395604
##   2        23       0.9592517  0.9157143  0.9395604
##   2        24       0.9592517  0.9157143  0.9395604
##   2        25       0.9592517  0.9157143  0.9395604
##   3         2       0.9307692  0.8519048  0.9181319
##   3         3       0.9406227  0.8519048  0.9472527
##   3         4       0.9572527  0.8800000  0.9406593
##   3         5       0.9543223  0.8871429  0.9549451
##   3         6       0.9476374  0.8938095  0.9395604
##   3         7       0.9632601  0.8938095  0.9318681
##   3         8       0.9643590  0.9009524  0.9164835
##   3         9       0.9577656  0.9080952  0.9164835
##   3        10       0.9570879  0.9080952  0.9010989
##   3        11       0.9570330  0.9080952  0.9087912
##   3        12       0.9548718  0.9080952  0.9087912
##   3        13       0.9438462  0.8938095  0.9087912
##   3        14       0.9295604  0.9009524  0.8934066
##   3        15       0.9284615  0.9009524  0.8934066
##   3        16       0.9284615  0.9009524  0.8934066
##   3        17       0.9284615  0.9009524  0.8934066
```

```
## 3 18 0.9284615 0.9009524 0.8934066
## 3 19 0.9284615 0.9009524 0.8934066
## 3 20 0.9284615 0.9009524 0.8934066
## 3 21 0.9284615 0.9009524 0.8934066
## 3 22 0.9284615 0.9009524 0.8934066
## 3 23 0.9284615 0.9009524 0.8934066
## 3 24 0.9284615 0.9009524 0.8934066
## 3 25 0.9284615 0.9009524 0.8934066
## 4 2 0.9307692 0.8519048 0.9181319
## 4 3 0.9406227 0.8519048 0.9472527
## 4 4 0.9481868 0.8800000 0.9406593
## 4 5 0.9537729 0.8871429 0.9549451
## 4 6 0.9476374 0.8938095 0.9395604
## 4 7 0.9632601 0.8938095 0.9318681
## 4 8 0.9649084 0.9009524 0.9164835
## 4 9 0.9627106 0.9080952 0.9164835
## 4 10 0.9570879 0.9080952 0.9010989
## 4 11 0.9551099 0.9080952 0.9087912
## 4 12 0.9523993 0.9080952 0.9087912
## 4 13 0.9413736 0.8938095 0.9087912
## 4 14 0.9287363 0.9009524 0.8934066
## 4 15 0.9276374 0.9009524 0.8934066
## 4 16 0.9276374 0.9009524 0.8934066
## 4 17 0.9276374 0.9009524 0.8934066
## 4 18 0.9276374 0.9009524 0.8934066
## 4 19 0.9276374 0.9009524 0.8934066
## 4 20 0.9276374 0.9009524 0.8934066
## 4 21 0.9276374 0.9009524 0.8934066
## 4 22 0.9276374 0.9009524 0.8934066
## 4 23 0.9276374 0.9009524 0.8934066
## 4 24 0.9276374 0.9009524 0.8934066
## 4 25 0.9276374 0.9009524 0.8934066
##
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were nprune = 5 and degree = 1.
```

```r
mars$bestTune
```

```
##   nprune degree
## 4      5      1
```

```r
max(mars$results$ROC)
```

```
## [1] 0.9691209
```

Question c

Linear discriminant analysis

We can also fit the data with linear discriminant analysis.

```r
set.seed(81063)

lda <- train(
  x = training_predictors,
  y = training_response,
  method = "lda",
  metric = "ROC",
  trControl = ctrl
)

lda$results$ROC
```

```
## [1] 0.9496337
```

```r
lda_pred <- predict(
  lda,
  newdata = training_data,
  type = "raw"
)

(lda_confusion_matrix <- confusionMatrix(
  data = lda_pred,
  reference = training_data$mpg_cat,
  positive = "high"
))
```

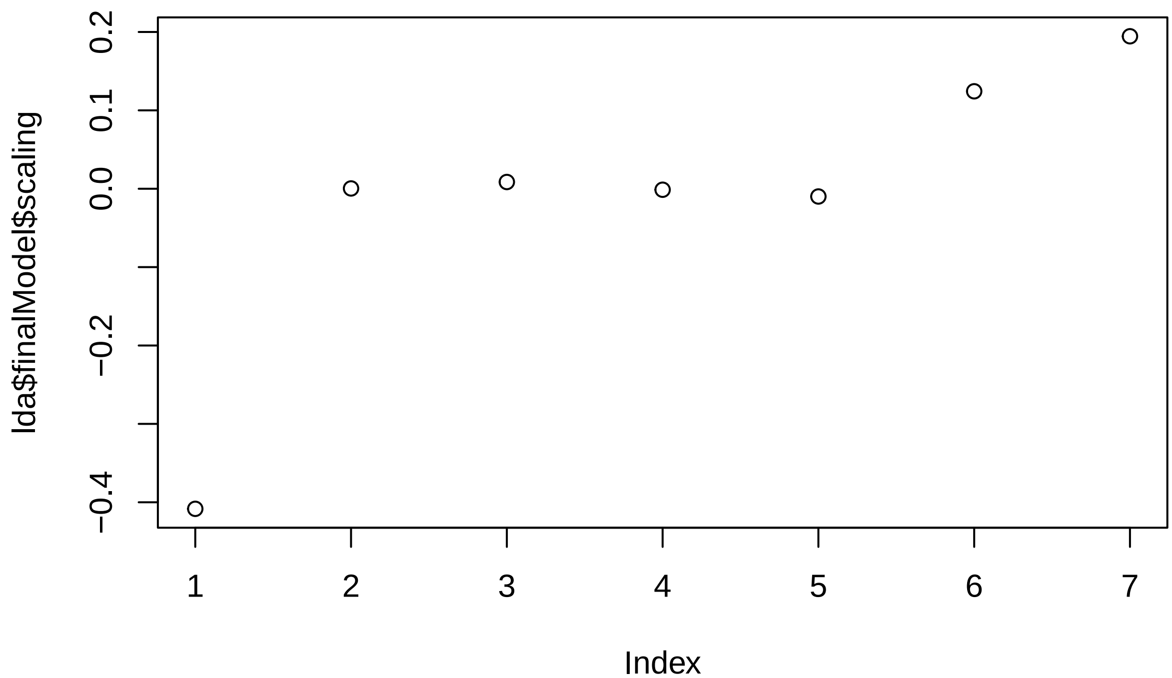
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction low high
##       low  122    5
##       high  20  127
##
##                Accuracy : 0.9088
##                  95% CI : (0.8683, 0.9401)
##     No Information Rate : 0.5182
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.818
##
##  Mcnemar's Test P-Value : 0.00511
##
##             Sensitivity : 0.9621
##             Specificity : 0.8592
##          Pos Pred Value : 0.8639
##          Neg Pred Value : 0.9606
##              Prevalence : 0.4818
##          Detection Rate : 0.4635
##    Detection Prevalence : 0.5365
##       Balanced Accuracy : 0.9106
##
##        'Positive' Class : high
##
```

Getting predictions against the training dataset again, we see that the LDA model has an accuracy of 0.9087591. It also has a good kappa of 0.8180031. Though a good model, it still does not have as high agreement or accuracy as MARS, and also has a lower ROC with 0.9496337.

Plot the linear discriminants

Below are the discriminant coordinates for the LDA model.

```
plot(lda$finalModel$scaling)
```



Question d

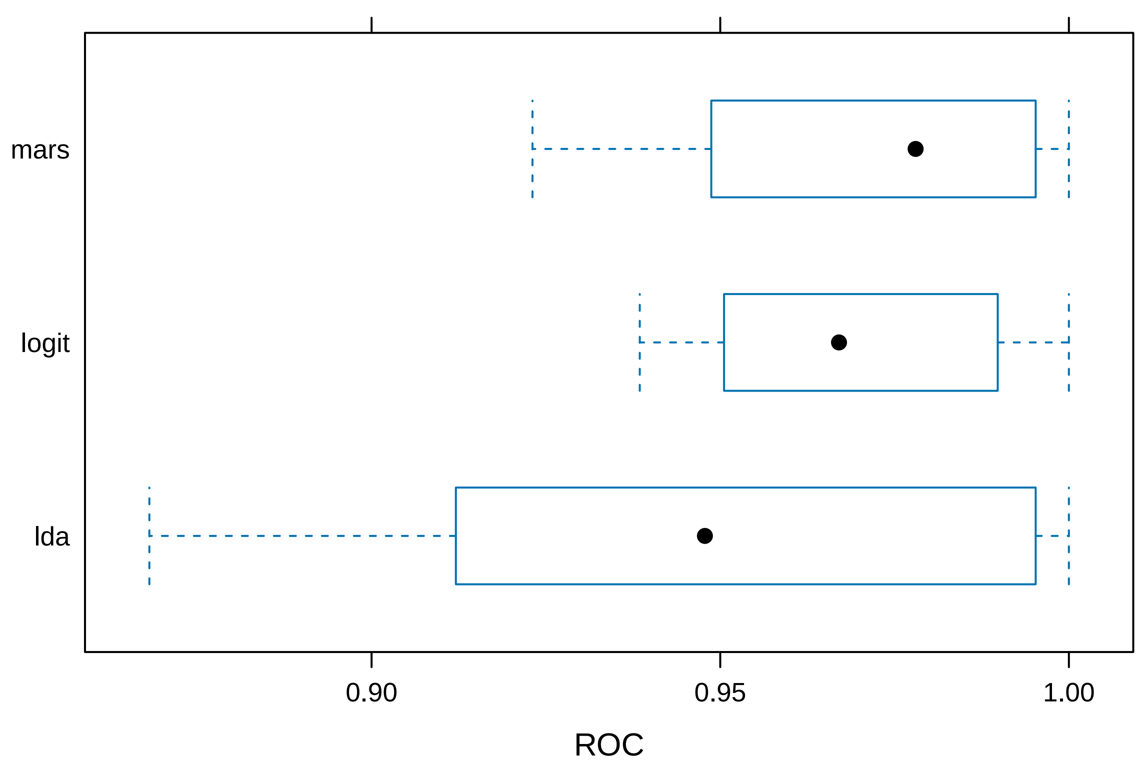
Which model will you choose to predict the response variable?

To select our best model, we should **evaluate it based on CV and not on test data** - hence, a boxplot of the CV-ROC is shown below.

```
res <- resamples(list(logit = logit,
                      mars = mars,
                      lda = lda))

summary(res)
```

```
##
## Call:
## summary.resamples(object = res)
##
## Models: logit, mars, lda
## Number of resamples: 10
##
## ROC
##            Min.   1st Qu.    Median      Mean   3rd Qu. Max. NA's
## logit 0.9384615 0.9546703 0.9670330 0.9703715 0.9882261    1    0
## mars  0.9230769 0.9505495 0.9780220 0.9691209 0.9923077    1    0
## lda   0.8681319 0.9175824 0.9478022 0.9496337 0.9923077    1    0
##
## Sens
##            Min.   1st Qu.    Median      Mean   3rd Qu. Max. NA's
## logit 0.7857143 0.7857143 0.8976190 0.8938095 1.0000000    1    0
## mars  0.7857143 0.8571429 0.9285714 0.9076190 0.9833333    1    0
## lda   0.6428571 0.7321429 0.8642857 0.8447619 0.9321429    1    0
##
## Spec
##            Min.   1st Qu.    Median      Mean   3rd Qu. Max. NA's
## logit 0.7692308 0.8489011 0.9230769 0.9016484 0.9271978    1    0
## mars  0.8461538 0.8736264 0.9230769 0.9318681 1.0000000    1    0
## lda   0.9230769 0.9230769 0.9285714 0.9549451 1.0000000    1    0
```

```
bwplot(res, metric = "ROC")
```



```r
median_logit_roc <- median(res$values$`logit~ROC`)

median_lda_roc <- median(res$values$`lda~ROC`)

median_mars_roc <- median(res$values$`mars~ROC`)
```

This illustrates that between LDA, logistic regression, and MARS, LDA has the highest median ROC with CV, with 0.9478022 compared to 0.967033 for logistic regression and 0.978022 for MARS.

Plot its ROC curve

We can compare the ROC curves generated when fitting each model's predicted values against the training dataset.

```r
logit_pred <- predict(logit, newdata = training_data, type = "prob")[,2]

mars_pred <- predict(mars, newdata = training_data, type = "prob")[,2]

lda_pred <- predict(lda, newdata = training_data, type = "prob")[,2]

roc_logit <- roc(training_data$mpg_cat, logit_pred)
```

```
## Setting levels: control = low, case = high

## Setting direction: controls < cases
```

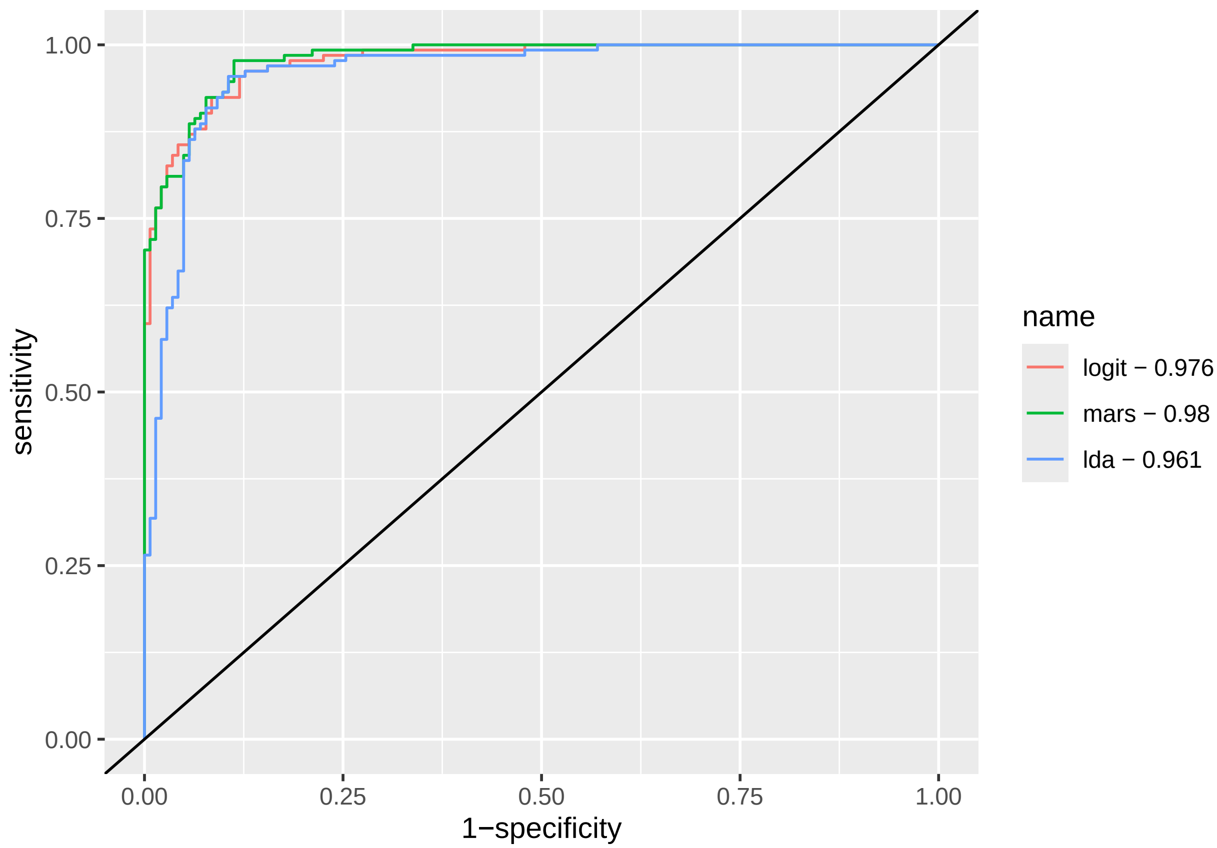
```r
roc_mars <- roc(training_data$mpg_cat, mars_pred)
```

```
## Setting levels: control = low, case = high
## Setting direction: controls < cases
```

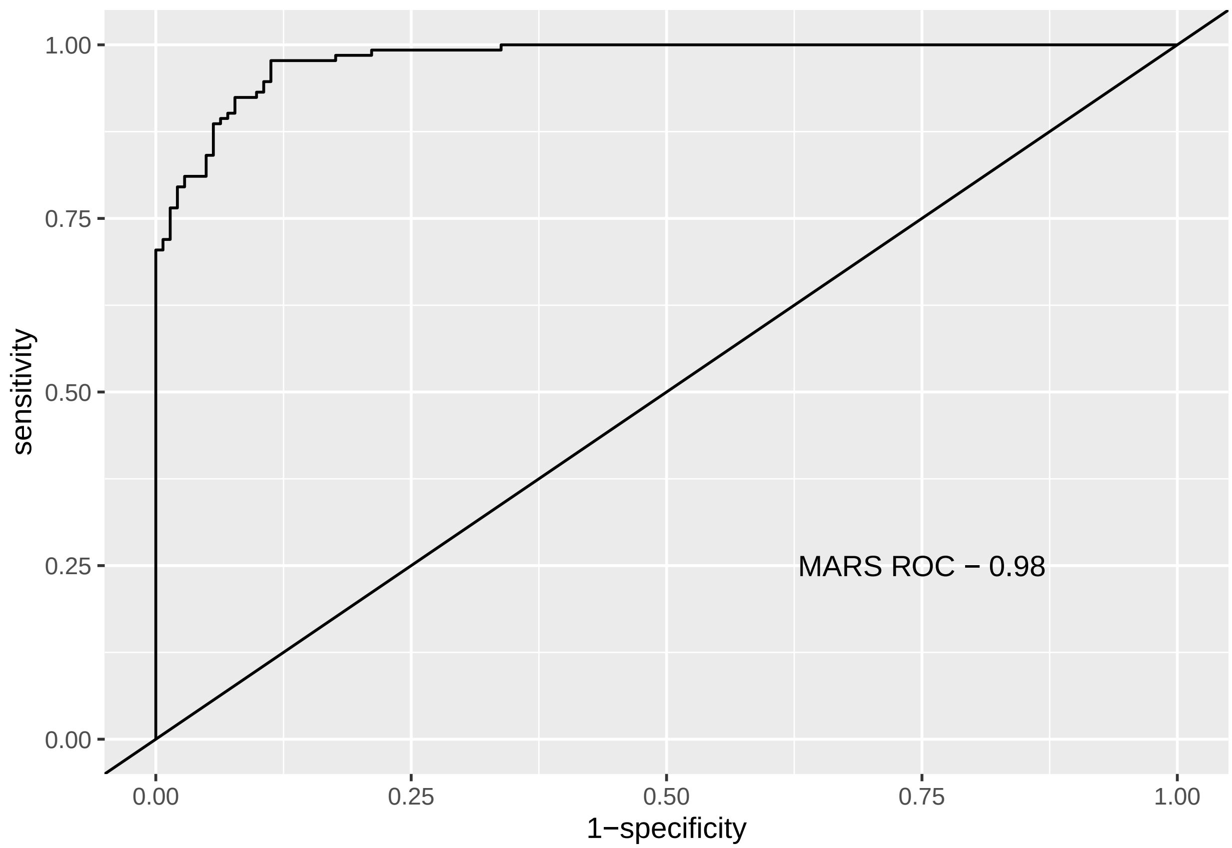
```r
roc_lda <- roc(training_data$mpg_cat, lda_pred)
```

```
## Setting levels: control = low, case = high
## Setting direction: controls < cases
```

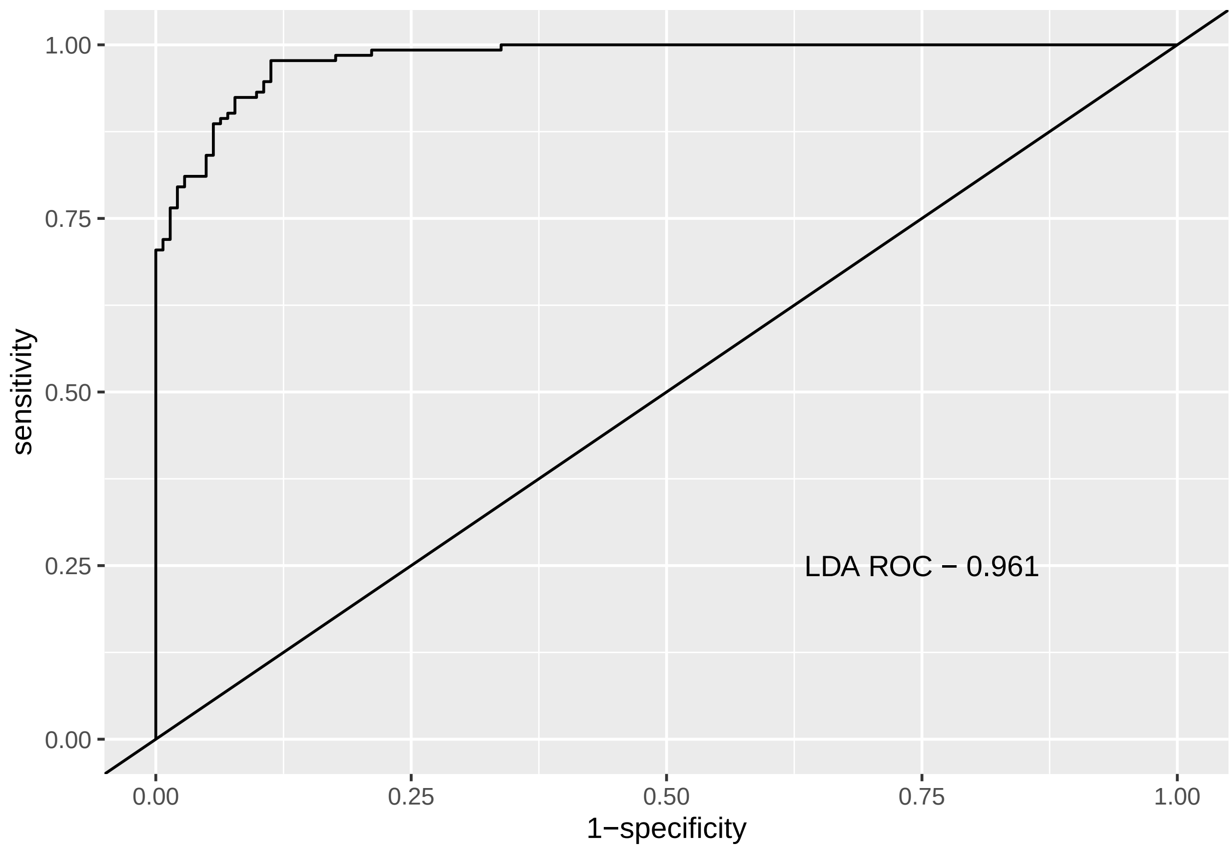
```r
auc <- c(roc_logit$auc[1], roc_mars$auc[1], roc_lda$auc[1])
model_names <- c("logit", "mars", "lda")

ggroc(list(roc_logit, roc_mars, roc_lda), legacy.axes = TRUE) +
  scale_color_discrete(labels = paste(model_names, "-", round(auc,3), sep = " ")) +
  geom_abline(intercept = 0, slope = 1, color = "black")
```




```r
ggroc(roc_mars, legacy.axes = TRUE) +
  geom_abline(intercept = 0, slope = 1, color = "black") +
  annotate("text", x = 0.75, y = 0.25, label = paste("MARS ROC -", round(auc[2], 3), sep = " "))
```




```
ggroc(roc_mars, legacy.axes = TRUE) +
  geom_abline(intercept = 0, slope = 1, color = "black") +
  annotate("text", x = 0.75, y = 0.25, label = paste("LDA ROC -", round(auc[3], 3), sep = " "))
```



The first graph shows the ROC AUC for all 3 models against the training dataset, which gives that the MARS actually has the highest training ROC. However, since we are basing our model selection on CV, LDA is still the best model for our interests. The AUC of the LDA ROC is the last graph.

Select a probability threshold to classify observations and compute the confusion matrix.

lda_pred_prob generates 274 obs for each of the 2 classes, low and high, for a total of 548 obs while the training data is only 274 obs

```
lda_pred_prob <- predict(lda, newdata = training_data, type = "prob")
lda_pred_prob <- c(lda_pred_prob$low, lda_pred_prob$high)

threshold <- 0.5
lda_pred_0.5 <- rep("low", length(lda_pred_prob))
lda_pred_0.5[lda_pred_prob > threshold] <- "high"
lda_pred_0.5
```

```
##   [1] "low"  "high" "low"  "low"  "high" "low"  "high" "high" "high" "high"
##  [11] "low"  "high" "high" "high" "high" "low"  "high" "low"  "high" "high"
##  [21] "high" "low"  "low"  "high" "low"  "high" "low"  "high" "low"  "low"
##  [31] "high" "high" "high" "high" "high" "high" "low"  "low"  "high" "low"
##  [41] "low"  "high" "low"  "low"  "high" "high" "low"  "high" "low"  "low"
##  [51] "low"  "low"  "high" "low"  "high" "high" "high" "high" "low"  "high"
##  [61] "low"  "high" "high" "low"  "low"  "high" "high" "low"  "low"  "high"
```

```
##  [71] "high" "high" "high" "high" "low"  "high" "high" "low"  "low"  "low"
##  [81] "low"  "low"  "low"  "low"  "low"  "low"  "low"  "low"  "low"  "high"
##  [91] "high" "low"  "high" "low"  "low"  "high" "high" "high" "high" "high"
## [101] "low"  "high" "low"  "low"  "high" "low"  "high" "low"  "high" "high"
## [111] "low"  "high" "high" "low"  "low"  "high" "high" "low"  "high" "low"
## [121] "high" "high" "low"  "high" "low"  "high" "high" "low"  "high" "low"
## [131] "low"  "low"  "high" "low"  "low"  "high" "high" "high" "low"  "low"
## [141] "low"  "low"  "low"  "low"  "low"  "high" "high" "high" "high" "low"
## [151] "high" "low"  "low"  "low"  "low"  "low"  "low"  "high" "low"  "low"
## [161] "low"  "high" "low"  "high" "high" "low"  "high" "high" "low"  "low"
## [171] "high" "low"  "low"  "low"  "low"  "low"  "low"  "low"  "high" "low"
## [181] "high" "high" "low"  "low"  "high" "high" "high" "low"  "low"  "high"
## [191] "low"  "high" "high" "high" "high" "high" "low"  "low"  "low"  "high"
## [201] "low"  "low"  "high" "high" "high" "low"  "low"  "low"  "high" "high"
## [211] "high" "high" "low"  "low"  "low"  "low"  "high" "high" "high" "low"
## [221] "low"  "low"  "low"  "high" "low"  "low"  "high" "low"  "low"  "low"
## [231] "high" "low"  "low"  "low"  "high" "low"  "low"  "high" "low"  "high"
## [241] "low"  "high" "high" "low"  "high" "high" "low"  "high" "high" "low"
## [251] "low"  "low"  "low"  "high" "low"  "high" "high" "high" "high" "low"
## [261] "low"  "low"  "low"  "low"  "high" "low"  "low"  "low"  "low"  "low"
## [271] "high" "low"  "low"  "low"  "high" "low"  "high" "high" "low"  "high"
## [281] "low"  "low"  "low"  "low"  "high" "low"  "low"  "low"  "low"  "high"
## [291] "low"  "high" "low"  "low"  "low"  "high" "high" "low"  "high" "low"
## [301] "high" "low"  "high" "high" "low"  "low"  "low"  "low"  "low"  "low"
## [311] "high" "high" "low"  "high" "high" "low"  "high" "high" "low"  "low"
## [321] "high" "low"  "high" "high" "high" "high" "low"  "high" "low"  "low"
## [331] "low"  "low"  "high" "low"  "high" "low"  "low"  "high" "high" "low"
## [341] "low"  "high" "high" "low"  "low"  "low"  "low"  "low"  "high" "low"
## [351] "low"  "high" "high" "high" "high" "high" "high" "high" "high" "high"
## [361] "high" "high" "high" "low"  "low"  "high" "low"  "high" "high" "low"
## [371] "low"  "low"  "low"  "low"  "high" "low"  "high" "high" "low"  "high"
## [381] "low"  "high" "low"  "low"  "high" "low"  "low"  "high" "high" "low"
## [391] "low"  "high" "low"  "high" "low"  "low"  "high" "low"  "high" "low"
## [401] "low"  "high" "low"  "high" "high" "high" "low"  "high" "high" "low"
## [411] "low"  "low"  "high" "high" "high" "high" "high" "high" "high" "low"
## [421] "low"  "low"  "low"  "high" "low"  "high" "high" "high" "high" "high"
## [431] "high" "low"  "high" "high" "high" "low"  "high" "low"  "low"  "high"
## [441] "low"  "low"  "high" "high" "low"  "high" "high" "high" "high" "high"
## [451] "high" "high" "low"  "high" "low"  "low"  "high" "high" "low"  "low"
## [461] "low"  "high" "high" "low"  "high" "low"  "low"  "low"  "low"  "low"
## [471] "high" "high" "high" "low"  "high" "high" "low"  "low"  "low"  "high"
## [481] "high" "high" "low"  "low"  "low"  "low"  "high" "high" "high" "high"
## [491] "low"  "low"  "low"  "high" "high" "high" "high" "low"  "high" "high"
## [501] "low"  "high" "high" "high" "low"  "high" "high" "high" "low"  "high"
## [511] "high" "low"  "high" "low"  "high" "low"  "low"  "high" "low"  "low"
## [521] "high" "low"  "low"  "high" "high" "high" "high" "low"  "high" "low"
## [531] "low"  "low"  "low"  "high" "high" "high" "high" "high" "low"  "high"
## [541] "high" "high" "high" "high" "low"  "high" "high" "high"
```

```r
# confusionMatrix(
#   data = factor(lda_pred_0.5, levels = c("low", "high")),
#   reference = training_data$mpg_cat,
#   positive = "high"
# )
```