P8106: Data Science II, Homework #3

Zachary Katz (UNI: zak2132)

3/25/2022

Contents

Set-Up and Data Preprocessing

```r
set.seed(2132)

# Load data, clean column names, eliminate rows containing NA entries
data_raw = read_csv("./Data/auto.csv") %>%
  janitor::clean_names() %>%
  na.omit() %>%
  as.data.frame()

data = data_raw %>%
  mutate(
    year = as.factor(year),
    origin = as.factor(origin),
    mpg_cat = as.factor(mpg_cat)
  )

# Partition data into training/test sets (70% split)
indexTrain = createDataPartition(y = data$mpg_cat,
                                 p = 0.7,
                                 list = FALSE)
```

Part (a): Exploratory Data Analysis

```r
# Summary statistics
summary(data)
```

```
##    cylinders      displacement     horsepower        weight      acceleration  
##  Min.   :3.000   Min.   : 68.0   Min.   : 46.0   Min.   :1613   Min.   : 8.00  
##  1st Qu.:4.000   1st Qu.:105.0   1st Qu.: 75.0   1st Qu.:2225   1st Qu.:13.78  
##  Median :4.000   Median :151.0   Median : 93.5   Median :2804   Median :15.50  
##  Mean   :5.472   Mean   :194.4   Mean   :104.5   Mean   :2978   Mean   :15.54  
##  3rd Qu.:8.000   3rd Qu.:275.8   3rd Qu.:126.0   3rd Qu.:3615   3rd Qu.:17.02  
##  Max.   :8.000   Max.   :455.0   Max.   :230.0   Max.   :5140   Max.   :24.80  
##                                                                                
##       year     origin  mpg_cat  
##  73     : 40   1:245   high:196  
##  78     : 36   2: 68   low :196  
##  76     : 34   3: 79             
##  75     : 30                     
##  82     : 30                     
##  70     : 29                     
##  (Other):193
```

```r
skimr::skim(data)
```

Table 1: Data summary

Name	data
Number of rows	392
Number of columns	8
Column type frequency:	
factor	3
numeric	5
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
year	0	1	FALSE	13	73: 40, 78: 36, 76: 34, 75: 30
origin	0	1	FALSE	3	1: 245, 3: 79, 2: 68
mpg_cat	0	1	FALSE	2	hig: 196, low: 196

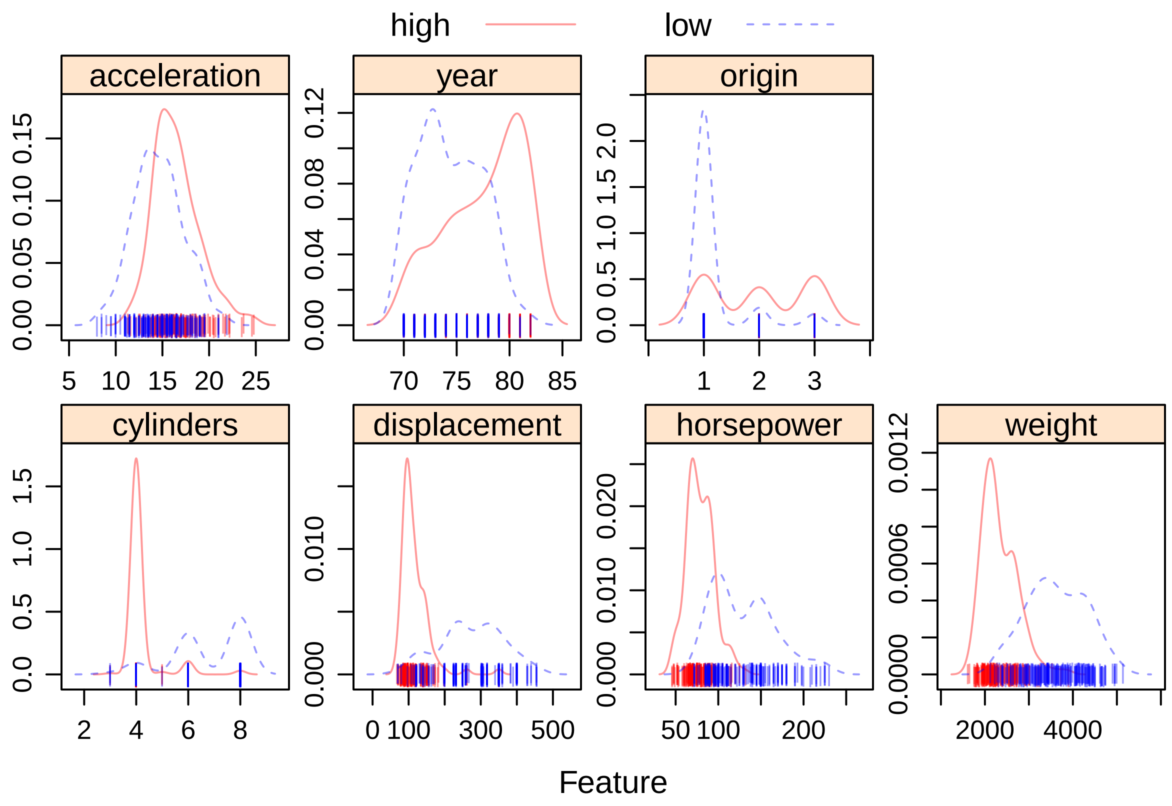
Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
cylinders	0	1	5.47	1.71	3	4.00	4.0	8.00	8.0	
displacement	0	1	194.41	104.64	68	105.00	151.0	275.75	455.0	
horsepower	0	1	104.47	38.49	46	75.00	93.5	126.00	230.0	

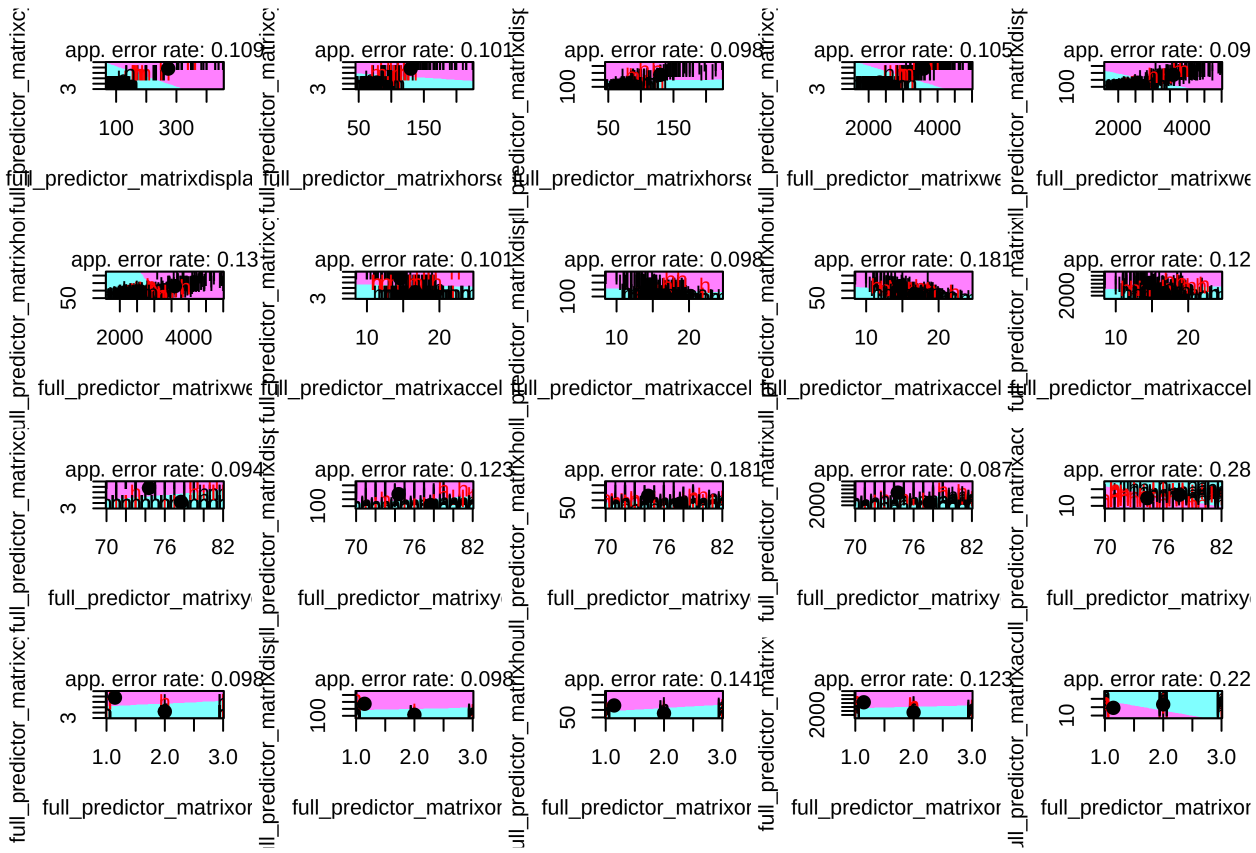
skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
weight	0	1	2977.58	849.40	1613	2225.25	2803.5	3614.75	5140.0	
acceleration	0	1	15.54	2.76	8	13.78	15.5	17.02	24.8	

We have 392 observations with 8 parameters: 7 predictors, including 5 continuous variables (`cylinders`, `displacement`, `horsepower`, `weight`, `acceleration`) and 2 categorical variables (`year`, `origin`), along with one binary outcome variable, `mpg_cat`, which takes values "high" and "low." Half our observations have the "high" label while the other half have the "low" label.

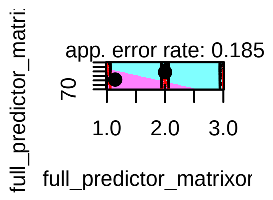
```
# Simple visualizations of the data

# Feature plot for all data (training and test)
theme1 = transparentTheme(trans = 0.4)
trellis.par.set(theme1)

full_predictor_matrix = model.matrix(mpg_cat ~., data_raw)[, -1]
full_outcome_vector = as.factor(data_raw$mpg_cat)

featurePlot(x = full_predictor_matrix,
            y = full_outcome_vector,
            scales = list(x = list(relation = "free"),
                          y = list(relation = "free")),
            plot = "density", pch = "|",
            auto.key = list(columns = 2))
```



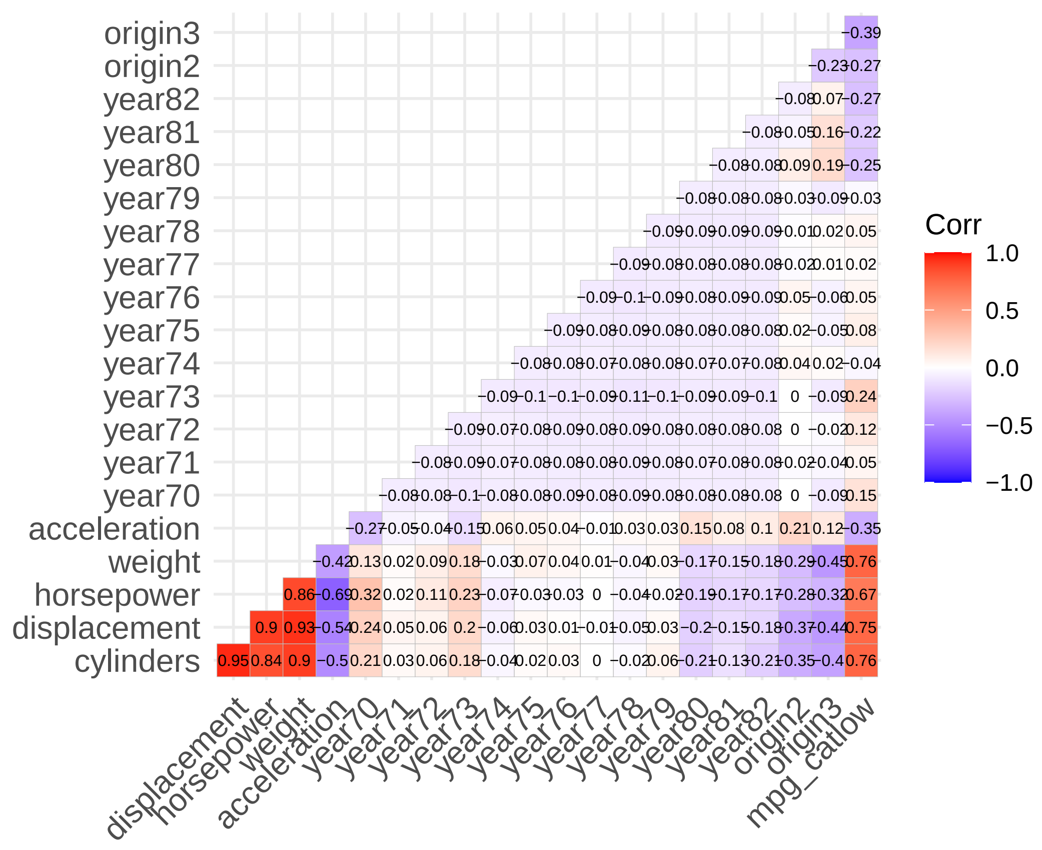
```
# Partition plot (LDA based on every combination of two variables) for training data only
partimat(full_outcome_vector ~ full_predictor_matrix, subset = indexTrain, method = "lda")
```



Partition Plot



```
# Correlation plot for all data
model.matrix(~0+., data = data) %>%
  cor(use = "pairwise.complete.obs") %>%
  ggcorrplot(show.diag = F, type = "lower", lab = TRUE, lab_size = 2)
```



We conduct three basic exploratory analyses: we construct a feature plot showing the probability density distribution of each predictor variable split by class of `mpg_cat`, partition plots for each pair of predictor variables (training data only), and a correlation plot for our parameters as well.

From the feature plot, we see that cars with low mpg are disproportionately from Origin 1; that cars with low mpg tend to be heavier, have more cylinders, higher displacement, and higher horsepower; and that cars with lower mpg tend to be from earlier years and have lower acceleration.

From the partition plots using LDA, we see how we would partition the classes based on every combination of two variables, giving us the decision boundary. Red points are considered misclassified. Our error rate is lowest for the following combinations of two predictors: `weight` and `displacement`; `year` and `cylinders`; and `year` and `weight`. On the other hand, our error rate is highest for `year` and `acceleration`, as well as for `origin` and acceleration. However, in this case, we haven't yet treated `year` and `origin` as factor variables so the decision boundaries are somewhat misleading.

Finally, from the correlation plot, we see that our outcome variable has the highest correlation with `cylinders`, `weight`, and `displacement`. Moreover, `cylinders`, `displacemebt`, and `horsepower` all have high collinearity, potentially leading to some redundancy in a model that includes all three covariates.

Part (b): Logistic Regression

```
set.seed(2132)

# Logistic regression using the training data (note: not using penalized logistic regression in this ca
glm.fit = glm(mpg_cat ~ .,
              data = data,
              subset = indexTrain,
              family = binomial(link = "logit"))

# Check for statistically significant predictors
summary(glm.fit)
```

```
##
## Call:
## glm(formula = mpg_cat ~ ., family = binomial(link = "logit"),
##     data = data, subset = indexTrain)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -3.01633  -0.17393  -0.00459   0.10393   2.06717
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -15.035632   5.381823  -2.794  0.00521 **
## cylinders      0.347589   0.605498   0.574  0.56593
## displacement  -0.008974   0.016080  -0.558  0.57677
## horsepower     0.015978   0.034629   0.461  0.64451
## weight         0.005764   0.001782   3.234  0.00122 **
## acceleration  -0.084525   0.189690  -0.446  0.65589
## year71         0.838433   2.132469   0.393  0.69419
## year72         1.601259   2.084661   0.768  0.44242
## year73         2.362207   2.124220   1.112  0.26612
## year74        -0.885310   3.035134  -0.292  0.77053
## year75        -1.108047   2.164668  -0.512  0.60874
## year76        -0.616430   2.215381  -0.278  0.78082
## year77        -1.228545   2.312780  -0.531  0.59528
## year78        -0.292928   2.127214  -0.138  0.89047
## year79        -4.247770   2.292743  -1.853  0.06393 .
## year80        -4.192358   2.444933  -1.715  0.08640 .
## year81        -4.457980   2.399826  -1.858  0.06322 .
## year82        -3.732246   2.255748  -1.655  0.09802 .
## origin2       -1.712525   1.038138  -1.650  0.09902 .
## origin3       -0.902381   0.929884  -0.970  0.33184
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 382.617  on 275  degrees of freedom
## Residual deviance:  89.034  on 256  degrees of freedom
## AIC: 129.03
##
## Number of Fisher Scoring iterations: 8
```

```
# Alternatively, using caret (to compare cross-validation with other models, rather than tuning the mod
ctrl = trainControl(method = "repeatedcv",
                    repeats = 5,
                    summaryFunction = twoClassSummary,
                    classProbs = TRUE)

set.seed(2132)

glm.logit.caret = train(x = data[indexTrain, 1:7],
                        y = data$mpg_cat[indexTrain],
                        method = "glm",
                        metric = "ROC",
                        trControl = ctrl)

summary(glm.logit.caret)
```

```
##
## Call:
## NULL
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -3.01633  -0.17393  -0.00459   0.10393   2.06717
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -15.035632   5.381823  -2.794  0.00521 **
## cylinders     0.347589   0.605498   0.574  0.56593
## displacement -0.008974   0.016080  -0.558  0.57677
## horsepower    0.015978   0.034629   0.461  0.64451
## weight        0.005764   0.001782   3.234  0.00122 **
## acceleration -0.084525   0.189690  -0.446  0.65589
## year71        0.838433   2.132469   0.393  0.69419
## year72        1.601259   2.084661   0.768  0.44242
## year73        2.362207   2.124220   1.112  0.26612
## year74       -0.885310   3.035134  -0.292  0.77053
## year75       -1.108047   2.164668  -0.512  0.60874
## year76       -0.616430   2.215381  -0.278  0.78082
## year77       -1.228545   2.312780  -0.531  0.59528
## year78       -0.292928   2.127214  -0.138  0.89047
## year79       -4.247770   2.292743  -1.853  0.06393 .
## year80       -4.192358   2.444933  -1.715  0.08640 .
## year81       -4.457980   2.399826  -1.858  0.06322 .
## year82       -3.732246   2.255748  -1.655  0.09802 .
## origin2      -1.712525   1.038138  -1.650  0.09902 .
## origin3      -0.902381   0.929884  -0.970  0.33184
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 382.617  on 275  degrees of freedom
## Residual deviance:  89.034  on 256  degrees of freedom
## AIC: 129.03
##
## Number of Fisher Scoring iterations: 8
```

With two methods, we build a logistic regression model (without penalization) from our training data. At the 0.01 significance level, `weight` is a significant predictor of our outcome `mpg_cat`. At the 0.1 significance level, i.e. less significantly than `weight`, our indicator variables `year79`, `year80`, `year81`, `year82`, and `origin2` are significant predictors of our outcome as well.

```r
# Check performance on test data (use simple classifier with cut-off of 0.5)
test.pred.prob = predict(glm.fit, newdata = data[-indexTrain,],
                           type = "response")

test.pred = rep("high", length(test.pred.prob))

test.pred[test.pred.prob>0.5] = "low"

confusionMatrix(data = as.factor(test.pred),
                reference = data$mpg_cat[-indexTrain],
                positive = "high")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction high low
##       high   56   7
##       low     2  51
##
##                Accuracy : 0.9224
##                  95% CI : (0.8578, 0.9639)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8448
##
##  Mcnemar's Test P-Value : 0.1824
##
##             Sensitivity : 0.9655
##             Specificity : 0.8793
##          Pos Pred Value : 0.8889
##          Neg Pred Value : 0.9623
##              Prevalence : 0.5000
##          Detection Rate : 0.4828
##    Detection Prevalence : 0.5431
##       Balanced Accuracy : 0.9224
##
##        'Positive' Class : high
##
```

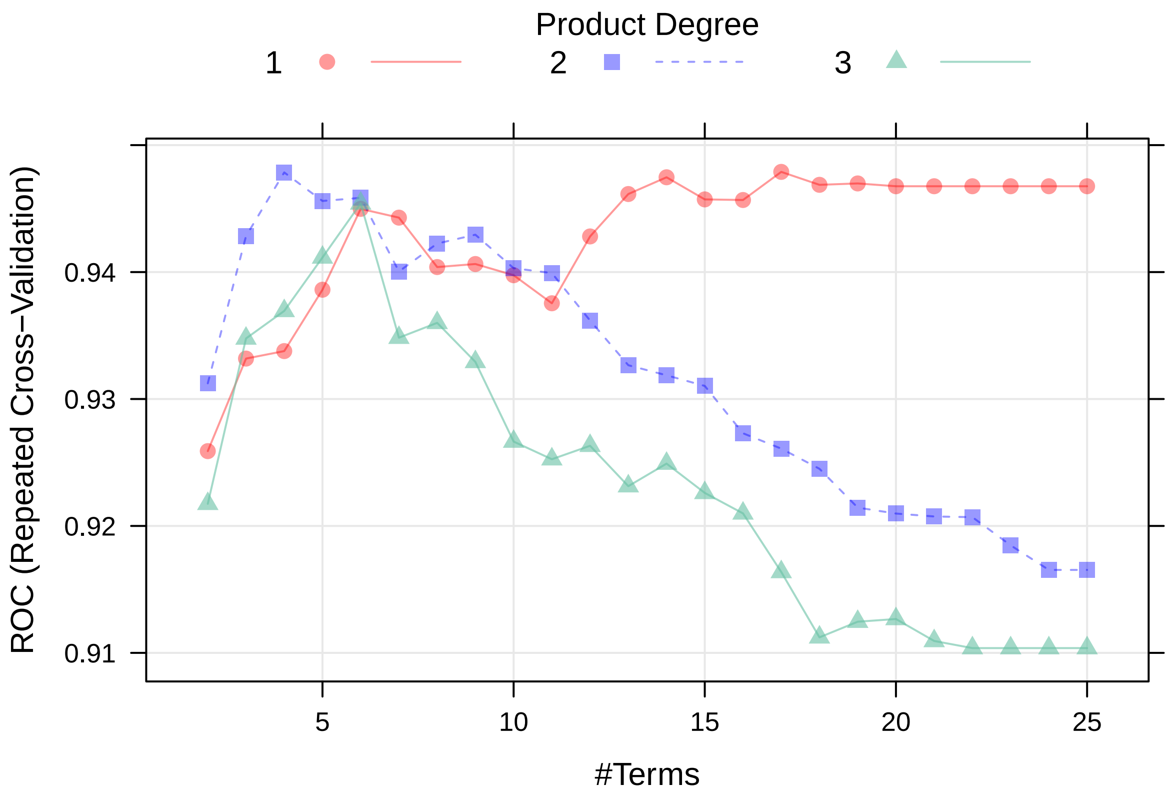
Our confusion matrix shows that our accuracy, or overall fraction of correct predictions, is roughly 92% (95% CI: 86% to 96%) once our model is applied to test data. The confusion matrix also tells us that our no information rate is 50%, which means that if we had no information and made the same class prediction for all observations, our model would be 50% accurate. Our p-value near 0 tells us that our accuracy is statistically significantly better than our no information rate. The model' is 96.6% sensitive (true detected positives out of all actual positives) and 87.9% specific (true detected negatives out of all actual negatives), with a positive predictive value of 88.9% (true detected positives out of all predicted positives) and a negative predictive value of 96.2% (true detected negatives out of all predicted negatives). Our sensitivity and specificity average to 92.2%, which is our balanced accuracy. Our kappa, at 0.8448, means that our interrater agreement is quite high, even accounting for the possibility of agreement by chance.

Part (c): MARS Model

```r
# Train MARS model using the training data
set.seed(2132)

model.mars = train(x = data[indexTrain, 1:7],
                   y = data$mpg_cat[indexTrain],
                   method = "earth",
                   tuneGrid = expand.grid(degree = 1:3,
                                          nprune = 2:25),
                   metric = "ROC",
                   trControl = ctrl)

summary(model.mars)
```

```
## Call: earth(x=data.frame[276,7], y=factor.object, keepxy=TRUE,
##             glm=list(family=function.object, maxit=100), degree=1, nprune=17)
##
## GLM coefficients
##                            low
## (Intercept)          2.2013294
## year73               2.6071990
## year79              -3.9580757
## year80              -5.4302372
## year81              -4.0910934
## year82              -6.4827284
## h(4-cylinders)      18.2394325
## h(cylinders-4)       0.4722001
## h(cylinders-6)      -2.3513657
## h(displacement-119)  1.1432420
## h(displacement-122) -1.2689170
## h(displacement-151)  0.2081505
## h(displacement-232) -0.1090955
## h(horsepower-80)     0.0499698
## h(4055-weight)      -0.0033184
##
## GLM (family binomial, link logit):
##  nulldev  df       dev  df   devratio     AIC iters converged
##  382.617 275    66.626 261      0.826   96.63    14         1
##
## Earth selected 15 of 24 terms, and 9 of 19 predictors (nprune=17)
## Termination condition: Reached nk 39
## Importance: displacement, year82, year73, year81, cylinders, year80, ...
## Number of terms at each degree of interaction: 1 14 (additive model)
## Earth GCV 0.06594421    RSS 14.57678    GRSq 0.7381311    RSq 0.7887424
```

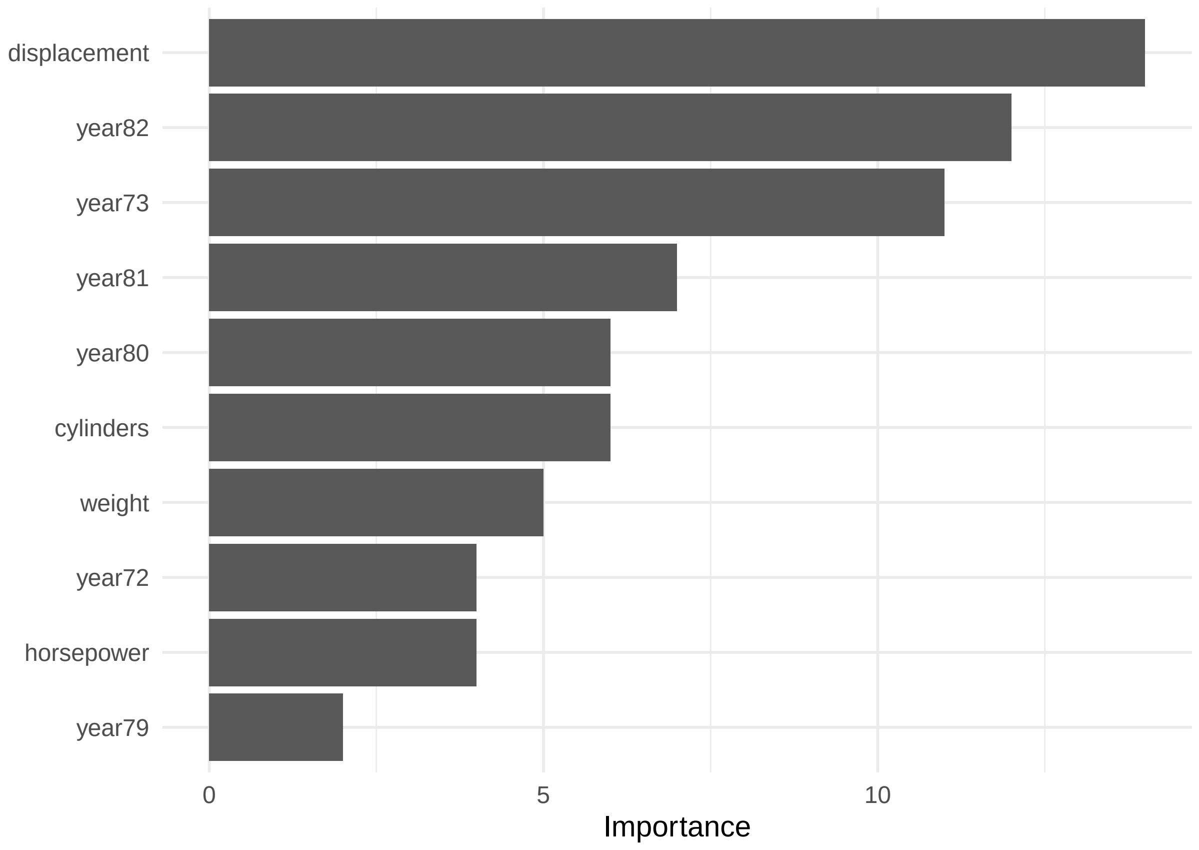
```r
plot(model.mars)
```



```
coef(model.mars$finalModel) %>% knitr::kable(col.names = "Coefficient")
```

	Coefficient
(Intercept)	2.2013294
h(displacement-232)	-0.1090955
year82	-6.4827284
year73	2.6071990
h(horsepower-80)	0.0499698
h(displacement-151)	0.2081505
year81	-4.0910934
year80	-5.4302372
h(cylinders-4)	0.4722001
h(4-cylinders)	18.2394325
h(displacement-122)	-1.2689170
h(cylinders-6)	-2.3513657
h(displacement-119)	1.1432420
year79	-3.9580757
h(4055-weight)	-0.0033184

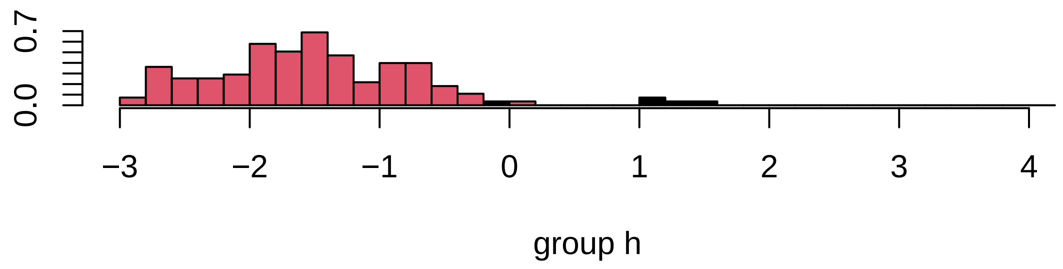
```
vip(model.mars$finalModel)
```

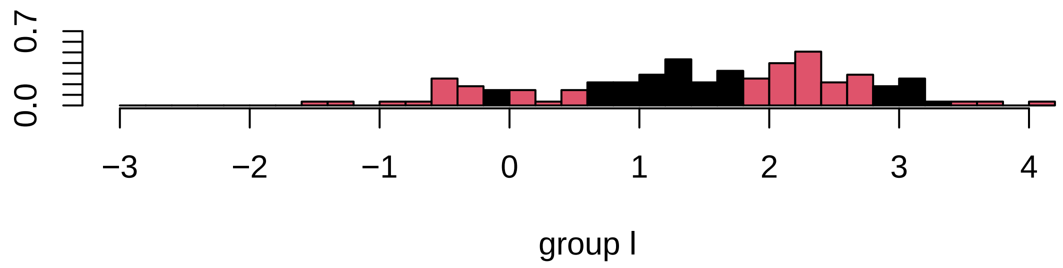


Overall, our MARS model tells us that `displacement` is the most important continuous variable, with indicators `year82` and `year73` following closely behind, based on the overall impact of each variable on our regression function following a backward elimination procedure. Using `earth`, our model selects 15 out of 24 terms, representing 9 of 19 predictors (nprune terms = 17). The model is optimized with and has an R-squared of 0.789.

Part (d): LDA & QDA

```
# LDA using the training data
lda.fit = lda(mpg_cat ~ ., data = data, subset = indexTrain)

# Plot the linear discriminants from LDA
plot(lda.fit, col = as.numeric(data$mpg_cat), abbrev = TRUE)
```





```r
# Obtain scaling matrix
lda.fit$scaling
```

```
##                       LD1
## cylinders     0.356652644
## displacement  0.001776196
## horsepower   -0.013036833
## weight        0.001121857
## acceleration -0.022844593
## year71       -0.321252141
## year72        0.074598052
## year73        0.291079641
## year74       -0.660521789
## year75       -0.353799843
## year76       -0.508935831
## year77       -0.679418153
## year78        0.064990148
## year79       -1.198259514
## year80       -1.598036344
## year81       -1.694074260
## year82       -1.741020869
## origin2      -0.561045699
## origin3      -0.405598966
```

LDA has no tuning parameters, and allows us to classify by nearest centroid. Because we have two classes, we have k = 2-1 = 1 linear discriminants, and so our linear discriminant plot gives us the histogram of our transformed X (predictors) for both classes. In this case, when our "X" is lower, we tend to classify in the high `mpg_cat` group, whereas when our "X" is higher, we tend to classify in the low `mpg_cat` group. Finally, the scaling object gives us our matrix A, which is (k-1) x p matrix, or in this case, a simple column vector with one entry per predictor, given we only have two outcome classes. This matrix allows us to build our x-tilde for each observation / data point.

```
# Alternatively, use caret for LDA
set.seed(2132)

model.lda = train(x = data_raw[indexTrain, 1:7],
                  y = data$mpg_cat[indexTrain],
                  method = "lda",
                  metric = "ROC",
                  trControl = ctrl)

model.lda$results
```

```
##   parameter       ROC      Sens      Spec      ROCSD     SensSD     SpecSD
## 1      none 0.9578481 0.9696703 0.8421978 0.03446146 0.04836388 0.07414189
```

For completeness, we also run an LDA model using `caret`, which has a 0.957 ROC, with 97% sensitivity and 84% specificity. We also run a QDA model as follows using both methods as well:

```
# Train a QDA model on the training data
qda.fit = qda(mpg_cat~., data = data,
               subset = indexTrain)

# Obtain scaling matrix
qda.fit$scaling
```

```
## , , high
##
##                     1           2           3            4            5
## cylinders    1.352392  1.92543199 -0.23070614  0.455255833 -0.508466282
## displacement 0.000000 -0.04123078  0.02426365 -0.041263638  0.011247954
## horsepower   0.000000  0.00000000 -0.08464518 -0.027031712  0.090019121
## weight       0.000000  0.00000000  0.00000000  0.004975879 -0.002913385
## acceleration 0.000000  0.00000000  0.00000000  0.000000000  0.580884192
## year71       0.000000  0.00000000  0.00000000  0.000000000  0.000000000
## year72       0.000000  0.00000000  0.00000000  0.000000000  0.000000000
## year73       0.000000  0.00000000  0.00000000  0.000000000  0.000000000
## year74       0.000000  0.00000000  0.00000000  0.000000000  0.000000000
## year75       0.000000  0.00000000  0.00000000  0.000000000  0.000000000
## year76       0.000000  0.00000000  0.00000000  0.000000000  0.000000000
## year77       0.000000  0.00000000  0.00000000  0.000000000  0.000000000
## year78       0.000000  0.00000000  0.00000000  0.000000000  0.000000000
## year79       0.000000  0.00000000  0.00000000  0.000000000  0.000000000
## year80       0.000000  0.00000000  0.00000000  0.000000000  0.000000000
## year81       0.000000  0.00000000  0.00000000  0.000000000  0.000000000
## year82       0.000000  0.00000000  0.00000000  0.000000000  0.000000000
## origin2      0.000000  0.00000000  0.00000000  0.000000000  0.000000000
## origin3      0.000000  0.00000000  0.00000000  0.000000000  0.000000000
##                          6             7             8             9
## cylinders    -0.196902069  0.0939981678  0.1663484856 -0.2826075111
## displacement  0.008917219  0.0017292803 -0.0009918236  0.0059667197
## horsepower    0.039419481 -0.0345837554  0.0020209630 -0.0010522927
## weight       -0.002178845  0.0008959304 -0.0001786234  0.0003884445
## acceleration  0.151576862 -0.1383127484  0.0327351571 -0.0020699353
## year71       -4.162696457  0.6130896980 -0.2422804868  0.4994817176
## year72        0.000000000  4.7304027393 -0.2260785198  0.4104944693
## year73        0.000000000  0.0000000000 -5.9875205573  0.3800961999
## year74        0.000000000  0.0000000000  0.0000000000  4.1909219276
## year75        0.000000000  0.0000000000  0.0000000000  0.0000000000
## year76        0.000000000  0.0000000000  0.0000000000  0.0000000000
## year77        0.000000000  0.0000000000  0.0000000000  0.0000000000
## year78        0.000000000  0.0000000000  0.0000000000  0.0000000000
## year79        0.000000000  0.0000000000  0.0000000000  0.0000000000
## year80        0.000000000  0.0000000000  0.0000000000  0.0000000000
## year81        0.000000000  0.0000000000  0.0000000000  0.0000000000
## year82        0.000000000  0.0000000000  0.0000000000  0.0000000000
## origin2       0.000000000  0.0000000000  0.0000000000  0.0000000000
## origin3       0.000000000  0.0000000000  0.0000000000  0.0000000000
##                         10           11           12            13
## cylinders    -0.0973378885  0.060103629  0.127404524  0.2368881609
## displacement -0.0028955652 -0.002200546 -0.003462691 -0.0022029954
## horsepower   -0.0074844202 -0.013406960 -0.014369933 -0.0180981363
## weight        0.0006246983  0.001030240  0.001096337  0.0006770417
## acceleration -0.0326212067 -0.063870225 -0.056433481 -0.0776006924
## year71       -0.1332599878  0.776130828  0.848325790  0.7027853310
## year72       -0.2250509682  0.629695745  0.688001091  0.6544387590
## year73       -0.2157729144  0.478909644  0.515354796  0.4353812010
## year74       -0.2440777489  0.613915290  0.662750921  0.5008367376
## year75       -4.9575841636  0.378608068  0.443416676  0.4382958155
## year76        0.0000000000  3.690993459  0.659689474  0.5462106248
## year77        0.0000000000  0.000000000  3.865270245  0.5386009975
## year78        0.0000000000  0.000000000  0.000000000  4.7082177622
## year79        0.0000000000  0.000000000  0.000000000  0.0000000000
## year80        0.0000000000  0.000000000  0.000000000  0.0000000000
## year81        0.0000000000  0.000000000  0.000000000  0.0000000000
## year82        0.0000000000  0.000000000  0.000000000  0.0000000000
## origin2       0.0000000000  0.000000000  0.000000000  0.0000000000
## origin3       0.0000000000  0.000000000  0.000000000  0.0000000000
##                         14            15           16            17
## cylinders     4.873408e-01 -0.0321809988  0.690266936  0.0527057849
## displacement  9.579536e-04  0.0075256835 -0.011589081  0.0020678313
## horsepower   -1.495169e-02  0.0099482073 -0.031907503 -0.0332720446
## weight       -7.054495e-05 -0.0009518663  0.001153067  0.0008313543
## acceleration -9.060382e-02  0.0161351759 -0.116899628 -0.0485541565
## year71       -4.512180e-01  0.7708762384 -1.504401987 -7.9484311008
## year72       -3.434292e-01  0.9291289694 -1.580678434 -8.0087470051
## year73       -6.494051e-01  0.9426778952 -1.944895900 -8.3249350680
## year74       -5.821126e-01  0.9602359665 -1.950817492 -8.2291808987
## year75       -4.672983e-01  1.0526926979 -1.847896667 -8.3246553799
## year76       -5.090095e-01  0.9171402993 -1.764240258 -8.1976941126
## year77       -5.118814e-01  0.9127727933 -1.747945605 -8.1898341906
## year78       -4.169272e-01  0.9571074163 -1.670640245 -8.1803234233
## year79       -4.017407e+00  0.8751741801 -2.095923005 -8.3750377510
## year80        0.000000e+00  3.3399992250 -1.896787147 -8.3590921375
## year81        0.000000e+00  0.0000000000 -3.756922171 -8.3785892302
## year82        0.000000e+00  0.0000000000  0.000000000 -9.0230492649
## origin2       0.000000e+00  0.0000000000  0.000000000  0.0000000000
## origin3       0.000000e+00  0.0000000000  0.000000000  0.0000000000
##                        18           19
## cylinders    -0.400698648  0.8344750850
## displacement  0.017187880 -0.0189511862
## horsepower    0.043335189  0.0160211986
## weight       -0.002297014 -0.0010274535
## acceleration  0.144459166  0.0230771790
## year71        1.675728286 -0.3792936609
## year72        2.242080775 -0.0001245589
## year73        1.314587722 -0.3310653147
## year74        2.188562738  0.0237365232
## year75        2.579122469  0.7472501546
## year76        2.070652634 -0.2731663414
## year77        2.437655626 -0.1062107747
## year78        3.148642483  0.2793387608
## year79        2.839962122 -0.4565790840
## year80        2.596477611  0.5885101463
## year81        3.058666832  0.2659065753
## year82        2.888847069 -0.0454375619
## origin2       2.606518717 -1.5578729376
## origin3       0.000000000 -2.8421198589
##
## , , low
##
##                      1         2           3            4            5
## cylinders    -0.7119051  1.521557  0.04640311 -0.388044147  0.428670544
## displacement  0.0000000 -0.027415  0.01759079 -0.010602870  0.007949339
## horsepower    0.0000000  0.000000 -0.05074176 -0.009801541  0.036616160
## weight        0.0000000  0.000000  0.00000000  0.002995610 -0.001950806
## acceleration  0.0000000  0.000000  0.00000000  0.000000000  0.700304775
## year71        0.0000000  0.000000  0.00000000  0.000000000  0.000000000
## year72        0.0000000  0.000000  0.00000000  0.000000000  0.000000000
## year73        0.0000000  0.000000  0.00000000  0.000000000  0.000000000
## year74        0.0000000  0.000000  0.00000000  0.000000000  0.000000000
## year75        0.0000000  0.000000  0.00000000  0.000000000  0.000000000
## year76        0.0000000  0.000000  0.00000000  0.000000000  0.000000000
## year77        0.0000000  0.000000  0.00000000  0.000000000  0.000000000
## year78        0.0000000  0.000000  0.00000000  0.000000000  0.000000000
## year79        0.0000000  0.000000  0.00000000  0.000000000  0.000000000
## year80        0.0000000  0.000000  0.00000000  0.000000000  0.000000000
## year81        0.0000000  0.000000  0.00000000  0.000000000  0.000000000
## year82        0.0000000  0.000000  0.00000000  0.000000000  0.000000000
## origin2       0.0000000  0.000000  0.00000000  0.000000000  0.000000000
## origin3       0.0000000  0.000000  0.00000000  0.000000000  0.000000000
##                          6             7             8             9
## cylinders     0.3220180222  0.1458940445  3.971281e-01 -0.0463456408
## displacement -0.0065188014  0.0049086520 -1.790995e-03 -0.0029921438
## horsepower    0.0164625003 -0.0013418795 -6.084760e-03 -0.0056756806
## weight       -0.0002120439 -0.0005074439 -7.912014e-05  0.0007319096
## acceleration  0.1242716606  0.0980883021  7.737309e-02 -0.0243939302
## year71        4.0347659958  0.3964160944  7.462218e-01 -0.3907267026
## year72        0.0000000000  3.2164902328  7.996544e-01 -0.4366891445
## year73        0.0000000000  0.0000000000  2.672355e+00 -0.3397785276
## year74        0.0000000000  0.0000000000  0.000000e+00 -4.7424992543
## year75        0.0000000000  0.0000000000  0.000000e+00  0.0000000000
## year76        0.0000000000  0.0000000000  0.000000e+00  0.0000000000
## year77        0.0000000000  0.0000000000  0.000000e+00  0.0000000000
## year78        0.0000000000  0.0000000000  0.000000e+00  0.0000000000
## year79        0.0000000000  0.0000000000  0.000000e+00  0.0000000000
## year80        0.0000000000  0.0000000000  0.000000e+00  0.0000000000
## year81        0.0000000000  0.0000000000  0.000000e+00  0.0000000000
## year82        0.0000000000  0.0000000000  0.000000e+00  0.0000000000
## origin2       0.0000000000  0.0000000000  0.000000e+00  0.0000000000
## origin3       0.0000000000  0.0000000000  0.000000e+00  0.0000000000
##                         10            11            12            13
## cylinders    -0.1351953364  0.0562646595  0.2243052451 -0.0203908542
## displacement  0.0030641824  0.0016391713  0.0002485723  0.0053470742
## horsepower   -0.0157785035  0.0112167260  0.0077210829  0.0130357716
## weight        0.0004572943 -0.0006765931 -0.0007059479 -0.0006989961
## acceleration -0.0312404602  0.0832200867  0.0612987050  0.0781205937
## year71       -0.7858077315  0.9145517672  0.8969945779  1.5970286520
## year72       -0.5240809303  0.8364911389  0.8479653091  1.5634841643
## year73       -0.4810760060  0.7516849328  0.7827047872  1.3689293597
## year74       -0.7490459345  0.9806074697  0.9519243362  1.7136940877
## year75       -3.9688916626  0.9371577530  0.8908040419  1.7035674277
## year76        0.0000000000  3.6002669240  0.8336066087  1.5692861051
## year77        0.0000000000  0.0000000000  4.5934800865  1.5399166713
## year78        0.0000000000  0.0000000000  0.0000000000  3.9701063895
## year79        0.0000000000  0.0000000000  0.0000000000  0.0000000000
## year80        0.0000000000  0.0000000000  0.0000000000  0.0000000000
## year81        0.0000000000  0.0000000000  0.0000000000  0.0000000000
## year82        0.0000000000  0.0000000000  0.0000000000  0.0000000000
## origin2       0.0000000000  0.0000000000  0.0000000000  0.0000000000
## origin3       0.0000000000  0.0000000000  0.0000000000  0.0000000000
##                         14            15            16            17
## cylinders    -0.1037628058  0.0302319836 -0.0674333214  4.935639e-02
## displacement -0.0005218163 -0.0012210643 -0.0005622487 -1.587547e-03
## horsepower    0.0193755125  0.0119437175 -0.0278835658  9.981436e-03
## weight       -0.0005983406 -0.0004075201  0.0010441270 -7.396633e-05
## acceleration -0.0032870385 -0.0347363273 -0.0777162776  2.485565e-02
## year71        2.0489829259  1.0986551650 -2.1032598611  1.226551e+00
## year72        1.8040477662  0.9344700590 -1.7822969259  1.043294e+00
## year73        1.7096228752  0.8649038294 -1.5968267233  1.003697e+00
## year74        2.2139997466  1.2186817567 -2.2041697350  1.121244e+00
## year75        2.2587470400  1.2599194366 -2.2733338464  1.235490e+00
## year76        2.0360082360  1.0630220559 -1.9649217235  1.094182e+00
## year77        1.9786309208  1.0579008744 -1.9193378219  1.077293e+00
## year78        2.0160527201  1.0856446384 -1.9885936831  1.156774e+00
## year79        4.8805817778  1.0832018873 -1.8804781900  1.083089e+00
## year80        0.0000000000 12.4663832632 -2.4452461470  1.308049e+00
## year81        0.0000000000  0.0000000000 -9.6605825209  1.360358e+00
## year82        0.0000000000  0.0000000000  0.0000000000  1.229833e+01
## origin2       0.0000000000  0.0000000000  0.0000000000  0.000000e+00
## origin3       0.0000000000  0.0000000000  0.0000000000  0.000000e+00
##                       18            19
## cylinders    -0.518924026 -0.3683383172
## displacement -0.014873849 -0.0211000455
## horsepower    0.010981707  0.0221310662
## weight        0.001277774  0.0006636918
## acceleration -0.179376648 -0.2235825301
## year71       -0.255502827  0.0361876750
## year72       -0.129649275  0.0569432137
## year73       -0.094205647  0.5809522877
## year74       -0.432651555  0.0161855693
## year75       -0.129793900  0.2693150706
## year76        0.136381283  0.7290845053
## year77        0.277490643  0.2875354428
## year78        0.277721247  0.4589472272
## year79        0.279066412  0.6737171440
## year80       -0.012016416  0.5020264567
## year81       -0.339934189 -0.0192455854
## year82       -0.169339245 -0.3691110912
## origin2      -5.519706754 -3.1536056900
## origin3       0.000000000 -6.6136917305
```

```
# Alternatively, use caret for QDA
set.seed(2132)

model.qda = train(x = data_raw[indexTrain, 1:7],
                  y = data$mpg_cat[indexTrain],
                  method = "qda",
                  metric = "ROC",
                  trControl = ctrl)

model.qda$results
```

```
##   parameter       ROC      Sens      Spec      ROCSD     SensSD     SpecSD
## 1      none 0.9452276 0.9061538 0.8883516 0.0403078 0.07709781 0.07687052
```

Our QDA model in `caret` using training data gives us a slightly worse ROC (0.945) and sensitivity (90%), but a bit better specificity (89%) compared to the LDA model, at first glance.

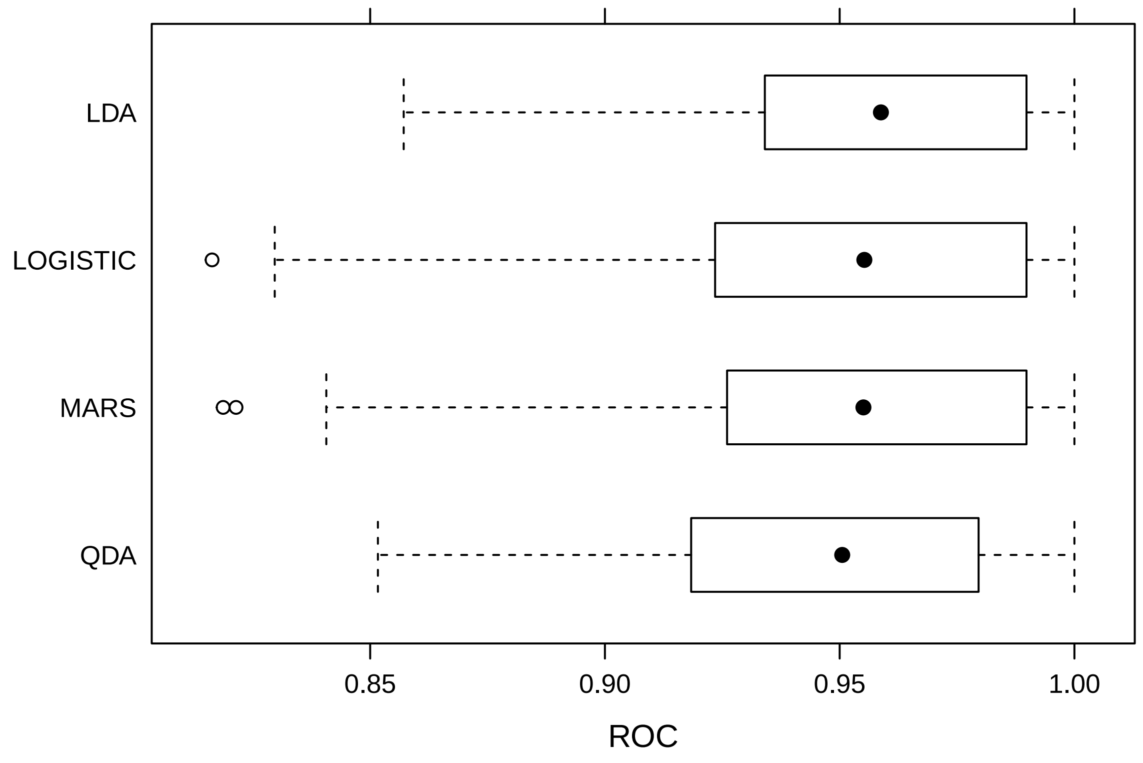
Part (e): Model Comparison and AUC/ROC

```
# Model comparison based on ROC (training data)
res = resamples(list(LOGISTIC = glm.logit.caret,
                     MARS = model.mars,
                     LDA = model.lda,
                     QDA = model.qda))

summary(res)
```

```
##
## Call:
## summary.resamples(object = res)
```

```
##
## Models: LOGISTIC, MARS, LDA, QDA
## Number of resamples: 50
##
## ROC
##               Min.   1st Qu.    Median      Mean   3rd Qu. Max. NA's
## LOGISTIC 0.8163265 0.9260204 0.9552590 0.9513634 0.9897959    1    0
## MARS     0.8186813 0.9266582 0.9550628 0.9478925 0.9895997    1    0
## LDA      0.8571429 0.9354396 0.9587912 0.9578481 0.9893884    1    0
## QDA      0.8516484 0.9183673 0.9505495 0.9452276 0.9791994    1    0
##
## Sens
##               Min.   1st Qu.    Median      Mean   3rd Qu. Max. NA's
## LOGISTIC 0.5714286 0.8571429 0.9285714 0.9031868 0.9285714    1    0
## MARS     0.6428571 0.8571429 0.9285714 0.9021978 1.0000000    1    0
## LDA      0.8571429 0.9285714 1.0000000 0.9696703 1.0000000    1    0
## QDA      0.7142857 0.8571429 0.9285714 0.9061538 0.9285714    1    0
##
## Spec
##               Min.   1st Qu.    Median      Mean   3rd Qu. Max. NA's
## LOGISTIC 0.6923077 0.8571429 0.9285714 0.8923077 0.9285714    1    0
## MARS     0.6923077 0.8489011 0.8571429 0.8836264 0.9285714    1    0
## LDA      0.6428571 0.7857143 0.8571429 0.8421978 0.8571429    1    0
## QDA      0.7142857 0.8571429 0.8901099 0.8883516 0.9285714    1    0
```

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



Based on resampling / general cross-validation from how our models perform on the training data, having not seen the test data, I would choose the LDA model for classification of our response variable `mpg_cat`, as it has the highest ROC.

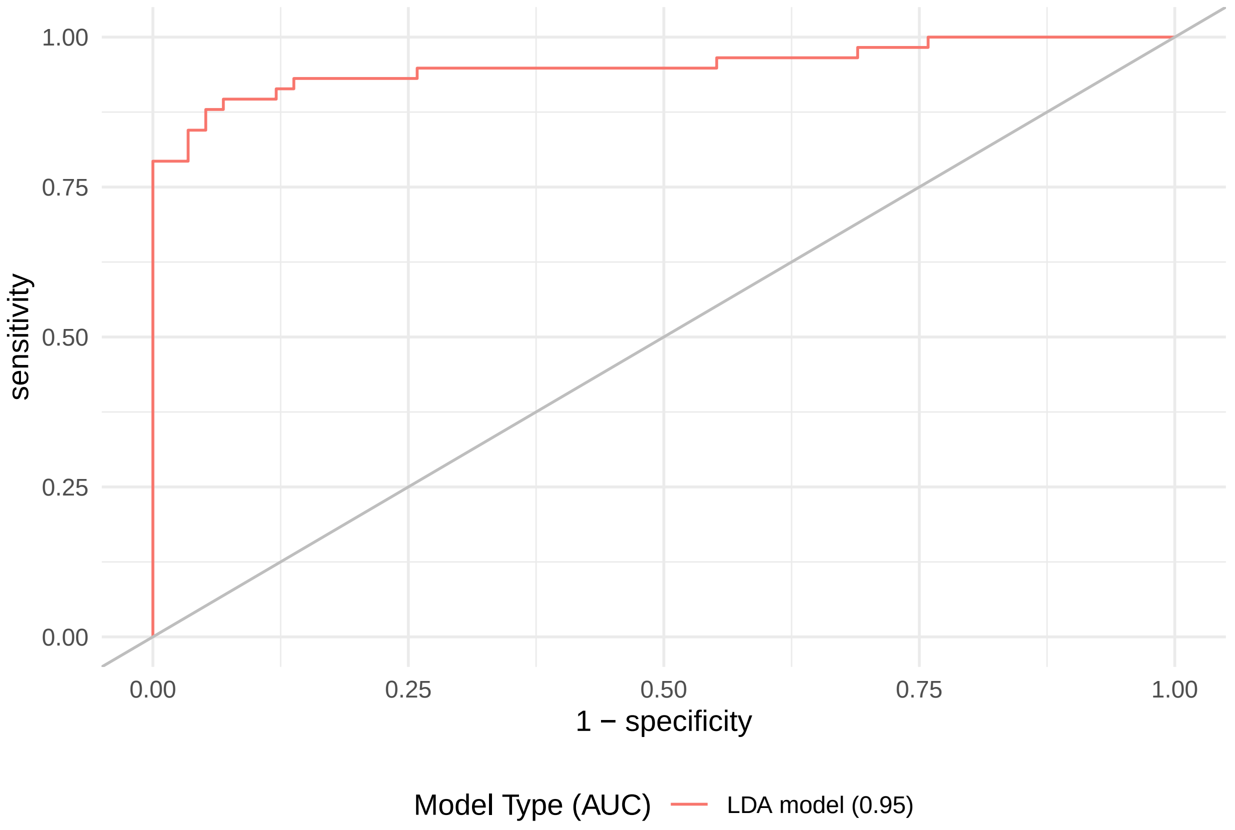
```r
# Predictions and ROC
lda.predict = predict(model.lda, newdata = data_raw[-indexTrain, 1:7], type = "prob")[,2]

roc.lda = roc(data$mpg_cat[-indexTrain], lda.predict)

# Report AUC and misclassification rate
auc_lda = roc.lda$auc[1]

auc_lda
```

```
## [1] 0.951843
```

```r
# Obtain classes
lda_class = lda.predict %>%
  as.data.frame() %>%
  mutate(
    class = case_when(. < 0.50 ~ "high",
                      . > 0.50 ~ "low")
  ) %>%
  dplyr::select(class) %>%
  as.matrix()

# Confusion matrix and misclassification error rate
confusionMatrix(data = as.factor(lda_class),
                reference = data$mpg_cat[-indexTrain],
                positive = "high")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction high low
##       high   56  10
##       low     2  48
##
##                Accuracy : 0.8966
##                  95% CI : (0.8263, 0.9454)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.7931
##
##  Mcnemar's Test P-Value : 0.04331
##
##             Sensitivity : 0.9655
##             Specificity : 0.8276
##          Pos Pred Value : 0.8485
##          Neg Pred Value : 0.9600
##              Prevalence : 0.5000
##          Detection Rate : 0.4828
##    Detection Prevalence : 0.5690
##       Balanced Accuracy : 0.8966
##
##        'Positive' Class : high
##
```

```
# Plot ROC curve for best model (LDA)
modelName = "LDA model"

pROC::ggroc(list(roc.lda), legacy.axes = TRUE) +
  scale_color_discrete(labels = paste0(modelName, " (", round(auc_lda, 2),")"),
                       name = "Model Type (AUC)") +
  geom_abline(intercept = 0, slope = 1, color = "grey")
```



When applied to the previously unseen test data, the LDA model has a misclassification rate of 1 - 0.8966, or ~10%, and an AUC of 0.95, as observed on our ROC plot above.