Santanu_Mukherjee_zes254_HW3

Santanu Mukherjee, zes254

07/27/2022

R Markdown

Chapter 12 - E-Book - Applied Predictive Modelling - Exercises pages 327:

Q 12.2

In Exercise 4.4, we described a data set which contained 96 oil samples each from one of seven types of oils (pumpkin, sunflower, peanut, olive, soybean, rapeseed, and corn). Gas chromatography was performed on each sample and the percentage of each type of 7 fatty acids was determined. We would like to use these data to build a model that predicts the type of oil based on a sample's fatty acid percentages.

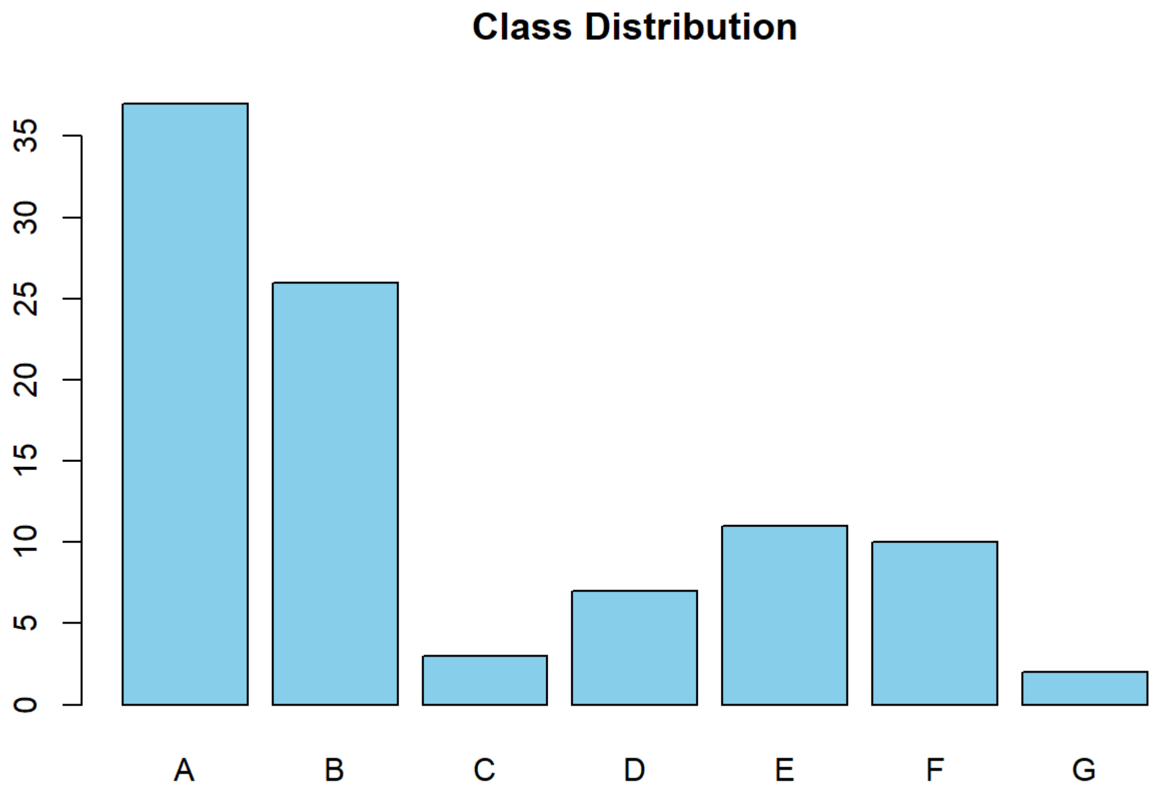
Q 12.2 a

a. Like the hepatic injury data, these data suffer from extreme imbalance. Given this imbalance, should the data be split into training and test sets?

Answer (12.2 a)

Yes, we would still split the data into training and test data sets.

```
data(oil)

library(MASS)
set.seed(123)

barplot(table(oilType),col=c("skyblue"), main="Class Distribution")
```



```
# Identifying predictors with zero-variance
nzv = nearZeroVar(fattyAcids,saveMetrics =TRUE)
nzv
```

```
##            freqRatio percentUnique zeroVar   nzv
## Palmitic    1.333333      46.87500   FALSE FALSE
## Stearic     1.500000      42.70833   FALSE FALSE
## Oleic       1.000000      78.12500   FALSE FALSE
## Linoleic    1.500000      84.37500   FALSE FALSE
## Linolenic   1.000000      37.50000   FALSE FALSE
## Eicosanoic  1.033333      12.50000   FALSE FALSE
## Eicosenoic  3.176471      14.58333   FALSE FALSE
```

```
#
zv_cols = nearZeroVar(fattyAcids)
print( sprintf("Dropping %d zero variance columns from %d (fraction=%10.6f)", length(zv_cols), dim(fattyAcids)[2], length(zv_cols)/dim(fattyAcids)[2]) );
```

```
## [1] "Dropping 0 zero variance columns from 7 (fraction=  0.000000)"
```

```
X = fattyAcids

# There are no linearly dependent columns remaining (or to start with)
print( findLinearCombos(X) )
```

```
## $linearCombos
## list()
##
## $remove
## NULL
```

```
# Remove the correlation between the predictors

high.Corr.M<-findCorrelation(cor(fattyAcids),cutoff = .75)
no.high.corr <- fattyAcids[,-high.Corr.M]

# So, after removing the highly correlated predictor, we split the data into 80% training and 20% test (using stratified random sampling)

set.seed(234)
training.Rows =  createDataPartition(oilType, p = .80, list= FALSE)

train.FattyAcids <- no.high.corr[ training.Rows, ]
test.FattyAcids <- no.high.corr[-training.Rows, ]

train.OilType <- oilType[training.Rows]
test.OilType <- oilType[-training.Rows]

ctrl <- trainControl(summaryFunction = defaultSummary)
```

Q 12.2 b

b. Which classification statistic would you choose to optimize for this exercise and why?

Answer (12.2 b)

The classification statistic that I have used here is the **“Accuracy”** rate. This is the simplest statistic as it reflects the agreement between the observed and predicted classes and so has the most straight forward interpretation.

Q 12.2 c

Of the models presented in this chapter, which performs best on these data? Which oil type does the model most accurately predict? Least accurately predict?

Answer (12.2 c)

Building various models

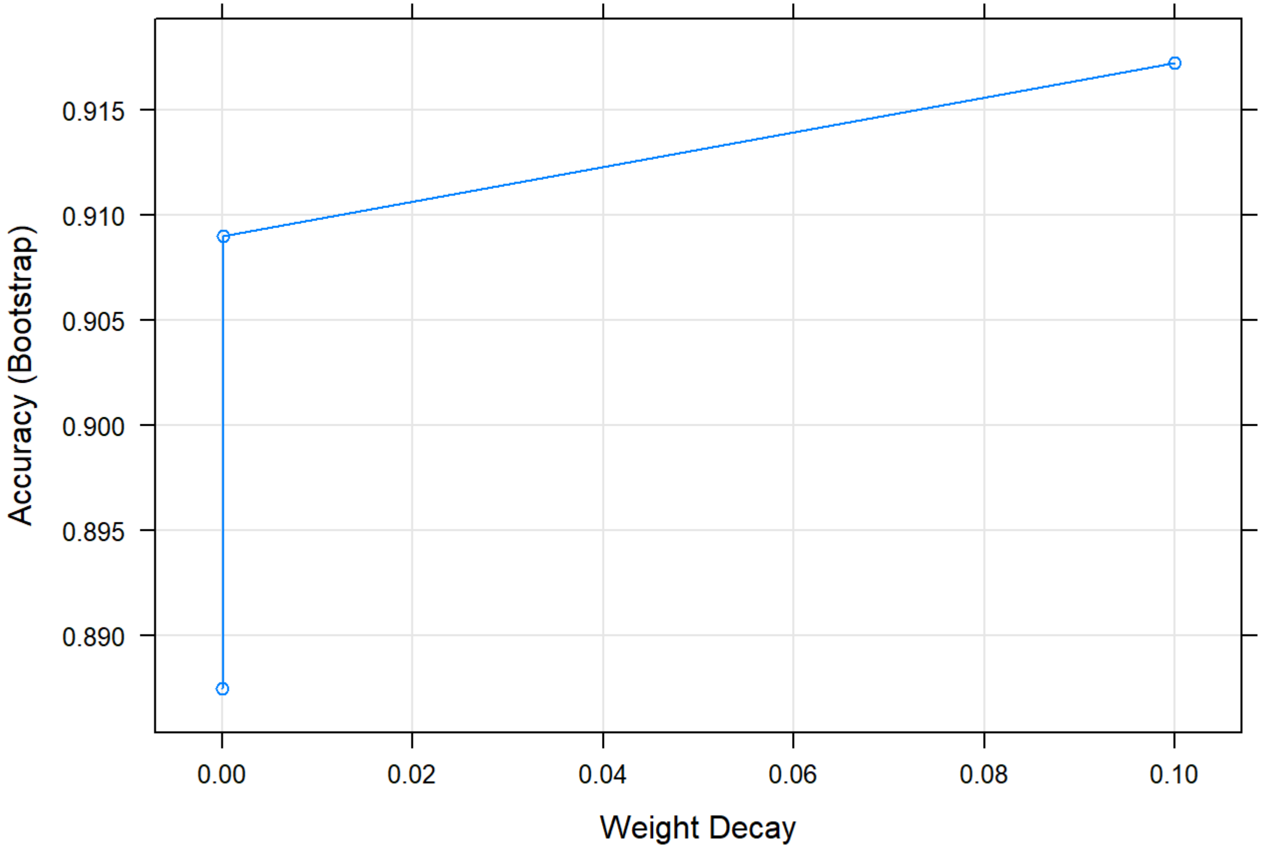
```
# Build models with this data:
#
############ Logistic Regression Analysis #############
# logistic regression

library(caret)
set.seed(456)
lr.FattyAcids <- train(x=train.FattyAcids,
               y = train.OilType,
               method = "multinom",
               metric = "Accuracy",
               trControl = ctrl)
```

```
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 33.774638
## iter  20 value 4.872133
## iter  30 value 0.021525
## final  value 0.000084
## converged
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 34.182768
## iter  20 value 12.320164
## iter  30 value 9.641936
## iter  40 value 9.541117
## iter  50 value 9.533587
## iter  60 value 9.532916
## iter  70 value 9.532845
## final  value 9.532838
## converged
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 33.775016
## iter  20 value 4.881610
## iter  30 value 0.180168
## iter  40 value 0.151977
## iter  50 value 0.142520
## iter  60 value 0.133913
## iter  70 value 0.130058
## iter  80 value 0.129193
## iter  90 value 0.127817
## iter 100 value 0.125855
## final  value 0.125855
## stopped after 100 iterations
## # weights:  48 (35 variable)
## initial  value 141.548998
## iter  10 value 51.073659
## iter  20 value 3.868819
## iter  30 value 0.098740
## iter  40 value 0.000605
## final  value 0.000076
## converged
## # weights:  48 (35 variable)
## initial  value 141.548998
## iter  10 value 51.463724
## iter  20 value 10.897481
## iter  30 value 8.302658
## iter  40 value 8.210548
## iter  50 value 8.187258
## iter  60 value 8.179259
## iter  70 value 8.175739
## iter  80 value 8.174449
## final  value 8.174335
## converged
## # weights:  48 (35 variable)
## initial  value 141.548998
## iter  10 value 51.074051
## iter  20 value 3.892800
## iter  30 value 0.207268
## iter  40 value 0.161671
## iter  50 value 0.150492
## iter  60 value 0.136441
## iter  70 value 0.131783
## iter  80 value 0.125323
## iter  90 value 0.118324
## iter 100 value 0.115793
## final  value 0.115793
## stopped after 100 iterations
## # weights:  48 (35 variable)
## initial  value 141.548998
## iter  10 value 19.945112
## iter  20 value 1.771753
## iter  30 value 0.012231
## iter  40 value 0.000145
## final  value 0.000050
## converged
## # weights:  48 (35 variable)
## initial  value 141.548998
## iter  10 value 22.510838
## iter  20 value 10.993928
## iter  30 value 9.882727
## iter  40 value 9.765650
## iter  50 value 9.729180
## iter  60 value 9.723955
## iter  70 value 9.722478
## iter  80 value 9.722314
## final  value 9.722312
## converged
## # weights:  48 (35 variable)
## initial  value 141.548998
## iter  10 value 19.947726
## iter  20 value 1.809113
## iter  30 value 0.276144
## iter  40 value 0.252201
## iter  50 value 0.187651
## iter  60 value 0.174233
## iter  70 value 0.163847
## iter  80 value 0.153294
## iter  90 value 0.149032
## iter 100 value 0.145494
## final  value 0.145494
## stopped after 100 iterations
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 34.294019
## iter  20 value 2.656412
## iter  30 value 0.013691
## iter  40 value 0.000141
## iter  40 value 0.000088
## iter  40 value 0.000040
## final  value 0.000040
## converged
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 35.778911
## iter  20 value 11.326897
## iter  30 value 9.793150
## iter  40 value 9.714824
## iter  50 value 9.703318
## iter  60 value 9.699520
## iter  70 value 9.697806
## iter  80 value 9.697734
## final  value 9.697729
## converged
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 34.295515
## iter  20 value 2.670047
## iter  30 value 0.199169
## iter  40 value 0.170569
## iter  50 value 0.160670
## iter  60 value 0.145198
## iter  70 value 0.139963
## iter  80 value 0.137289
## iter  90 value 0.135890
## iter 100 value 0.134680
## final  value 0.134680
## stopped after 100 iterations
## # weights:  48 (35 variable)
## initial  value 141.548998
## iter  10 value 29.449743
## iter  20 value 1.257273
## iter  30 value 0.004362
## final  value 0.000020
## converged
## # weights:  48 (35 variable)
## initial  value 141.548998
## iter  10 value 30.230615
## iter  20 value 9.382785
## iter  30 value 8.464242
## iter  40 value 8.320560
## iter  50 value 8.299227
## iter  60 value 8.295410
## iter  70 value 8.295003
## iter  80 value 8.294962
## final  value 8.294959
## converged
## # weights:  48 (35 variable)
## initial  value 141.548998
## iter  10 value 29.450523
## iter  20 value 1.282222
## iter  30 value 0.149428
## iter  40 value 0.133304
## iter  50 value 0.124125
## iter  60 value 0.116010
## iter  70 value 0.113996
## iter  80 value 0.111558
## iter  90 value 0.108976
## iter 100 value 0.106956
## final  value 0.106956
## stopped after 100 iterations
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 26.975726
## iter  20 value 1.730693
## iter  30 value 0.035611
## iter  40 value 0.007637
## iter  50 value 0.001302
## iter  60 value 0.000137
## final  value 0.000086
## converged
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 30.673762
## iter  20 value 13.146077
## iter  30 value 11.720518
## iter  40 value 11.604553
## iter  50 value 11.571565
## iter  60 value 11.560868
## iter  70 value 11.560240
## iter  80 value 11.560104
## final  value 11.560067
## converged
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 26.979458
## iter  20 value 1.763946
## iter  30 value 0.237614
## iter  40 value 0.191283
## iter  50 value 0.176492
## iter  60 value 0.165236
## iter  70 value 0.161341
## iter  80 value 0.157678
## iter  90 value 0.153806
## iter 100 value 0.147464
## final  value 0.147464
## stopped after 100 iterations
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 35.178503
## iter  20 value 2.954793
## iter  30 value 0.030551
## final  value 0.000055
## converged
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 36.969077
## iter  20 value 11.748119
## iter  30 value 7.203886
## iter  40 value 7.008908
## iter  50 value 6.986803
## iter  60 value 6.984883
## iter  70 value 6.984646
## iter  80 value 6.984537
## final  value 6.984524
## converged
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 35.180314
## iter  20 value 2.965878
## iter  30 value 0.107137
## iter  40 value 0.095150
## iter  50 value 0.084606
## iter  60 value 0.072508
## iter  70 value 0.059575
## iter  80 value 0.056207
## iter  90 value 0.055349
## iter 100 value 0.053071
## final  value 0.053071
## stopped after 100 iterations
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 36.548769
## iter  20 value 8.807080
## iter  30 value 0.127748
## iter  40 value 0.006449
## iter  50 value 0.000324
## final  value 0.000034
## converged
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 37.798708
## iter  20 value 10.770266
## iter  30 value 8.491106
## iter  40 value 8.277970
## iter  50 value 8.244160
## iter  60 value 8.236133
## iter  70 value 8.235100
## iter  80 value 8.234959
## final  value 8.234955
## converged
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 36.550028
## iter  20 value 8.848907
## iter  30 value 0.205380
## iter  40 value 0.125465
## iter  50 value 0.115790
## iter  60 value 0.103035
## iter  70 value 0.090301
## iter  80 value 0.086076
## iter  90 value 0.081370
## iter 100 value 0.077920
## final  value 0.077920
## stopped after 100 iterations
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 52.249885
## iter  20 value 10.914258
## iter  30 value 0.120758
## iter  40 value 0.000151
## iter  40 value 0.000076
## iter  40 value 0.000072
## final  value 0.000072
## converged
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 52.458058
## iter  20 value 16.576746
## iter  30 value 8.165289
## iter  40 value 7.644348
## iter  50 value 7.588415
## iter  60 value 7.556668
## iter  70 value 7.547752
## iter  80 value 7.541614
## iter  90 value 7.540154
## iter 100 value 7.539249
## final  value 7.539249
## stopped after 100 iterations
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 52.250094
## iter  20 value 10.924811
## iter  30 value 0.198563
## iter  40 value 0.114267
## iter  50 value 0.105391
## iter  60 value 0.096164
## iter  70 value 0.091426
## iter  80 value 0.087504
## iter  90 value 0.081995
## iter 100 value 0.076734
## final  value 0.076734
## stopped after 100 iterations
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 43.943724
## iter  20 value 7.171333
## iter  30 value 0.036433
## iter  40 value 0.000132
## iter  40 value 0.000071
## iter  40 value 0.000067
## final  value 0.000067
## converged
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 44.604555
## iter  20 value 13.492270
## iter  30 value 8.541689
## iter  40 value 8.372394
## iter  50 value 8.342366
## iter  60 value 8.339582
## iter  70 value 8.339273
## iter  80 value 8.339203
## final  value 8.339181
## converged
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 43.944389
## iter  20 value 7.178910
## iter  30 value 0.175603
## iter  40 value 0.141547
## iter  50 value 0.116412
## iter  60 value 0.098397
## iter  70 value 0.096145
## iter  80 value 0.086185
## iter  90 value 0.080860
## iter 100 value 0.078996
## final  value 0.078996
## stopped after 100 iterations
## # weights:  48 (35 variable)
## initial  value 141.548998
## iter  10 value 43.987489
## iter  20 value 1.341118
## iter  30 value 0.008164
## final  value 0.000059
## converged
## # weights:  48 (35 variable)
## initial  value 141.548998
## iter  10 value 44.205922
## iter  20 value 9.534398
## iter  30 value 8.205814
## iter  40 value 8.108572
## iter  50 value 8.103823
## iter  60 value 8.102657
## iter  70 value 8.102376
## final  value 8.102361
## converged
## # weights:  48 (35 variable)
## initial  value 141.548998
## iter  10 value 43.987707
## iter  20 value 1.370928
## iter  30 value 0.186875
## iter  40 value 0.158170
## iter  50 value 0.144267
## iter  60 value 0.128111
## iter  70 value 0.113659
## iter  80 value 0.109765
## iter  90 value 0.107594
## iter 100 value 0.105170
## final  value 0.105170
## stopped after 100 iterations
## # weights:  48 (35 variable)
## initial  value 141.548998
## iter  10 value 13.989891
## iter  20 value 1.635363
## iter  30 value 0.036504
## iter  40 value 0.005661
## iter  50 value 0.001108
## final  value 0.000052
## converged
## # weights:  48 (35 variable)
## initial  value 141.548998
## iter  10 value 16.124110
## iter  20 value 8.826168
## iter  30 value 8.608842
## iter  40 value 8.543402
## iter  50 value 8.537668
## iter  60 value 8.537236
## iter  70 value 8.537229
## final  value 8.537228
## converged
## # weights:  48 (35 variable)
## initial  value 141.548998
## iter  10 value 13.992060
## iter  20 value 1.365848
## iter  30 value 0.195606
## iter  40 value 0.156195
## iter  50 value 0.142751
## iter  60 value 0.127753
## iter  70 value 0.124441
## iter  80 value 0.121744
## iter  90 value 0.117968
## iter 100 value 0.114592
## final  value 0.114592
## stopped after 100 iterations
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 59.362947
## iter  20 value 12.403577
## iter  30 value 0.088979
## iter  40 value 0.000322
## final  value 0.000049
## converged
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 59.666592
## iter  20 value 25.151614
## iter  30 value 11.922385
## iter  40 value 11.533296
## iter  50 value 11.498393
## iter  60 value 11.487007
## iter  70 value 11.483174
## iter  80 value 11.482588
## iter  90 value 11.482533
## final  value 11.482504
## converged
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 59.363252
## iter  20 value 12.418956
## iter  30 value 0.263417
## iter  40 value 0.223912
## iter  50 value 0.201216
## iter  60 value 0.172529
## iter  70 value 0.155515
## iter  80 value 0.143611
## iter  90 value 0.140029
## iter 100 value 0.132368
## final  value 0.132368
## stopped after 100 iterations
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 71.670844
## iter  20 value 10.605977
## iter  30 value 0.110888
## iter  40 value 0.003531
## iter  50 value 0.000523
## iter  60 value 0.000191
## final  value 0.000050
## converged
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 47.735745
## iter  20 value 14.071774
## iter  30 value 10.099830
## iter  40 value 9.734830
## iter  50 value 9.677227
## iter  60 value 9.662201
## iter  70 value 9.655346
## iter  80 value 9.653147
## iter  90 value 9.652196
## iter 100 value 9.651920
## final  value 9.651920
## stopped after 100 iterations
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 71.672621
## iter  20 value 10.627082
## iter  30 value 0.225871
## iter  40 value 0.147171
## iter  50 value 0.133213
## iter  60 value 0.127329
## iter  70 value 0.122468
## iter  80 value 0.112722
## iter  90 value 0.107423
## iter 100 value 0.102005
## final  value 0.102005
## stopped after 100 iterations
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 38.944571
## iter  20 value 8.074028
## iter  30 value 0.056396
## iter  40 value 0.000570
## final  value 0.000055
## converged
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 39.390736
## iter  20 value 11.318098
## iter  30 value 8.108949
## iter  40 value 7.998510
## iter  50 value 7.983754
## iter  60 value 7.979945
## iter  70 value 7.979388
## iter  80 value 7.979313
## final  value 7.979308
## converged
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 38.945016
## iter  20 value 8.088802
## iter  30 value 0.173701
## iter  40 value 0.151142
## iter  50 value 0.132294
## iter  60 value 0.114318
## iter  70 value 0.092336
## iter  80 value 0.073099
## iter  90 value 0.070574
## iter 100 value 0.067382
## final  value 0.067382
## stopped after 100 iterations
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 34.246976
## iter  20 value 3.129732
## iter  30 value 0.014994
## final  value 0.000063
## converged
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 35.004769
## iter  20 value 10.713633
## iter  30 value 8.611485
## iter  40 value 8.484129
## iter  50 value 8.458486
## iter  60 value 8.446048
## iter  70 value 8.442135
## iter  80 value 8.441803
## iter  90 value 8.441763
## final  value 8.441750
## converged
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 34.247741
## iter  20 value 3.146451
## iter  30 value 0.159068
## iter  40 value 0.145749
## iter  50 value 0.131540
## iter  60 value 0.106770
## iter  70 value 0.097574
## iter  80 value 0.094669
## iter  90 value 0.089187
## iter 100 value 0.084939
## final  value 0.084939
## stopped after 100 iterations
## # weights:  48 (35 variable)
## initial  value 141.548998
## iter  10 value 16.997445
## iter  20 value 0.818301
## iter  30 value 0.017500
## iter  40 value 0.002761
## iter  50 value 0.000307
## final  value 0.000023
## converged
## # weights:  48 (35 variable)
## initial  value 141.548998
## iter  10 value 21.407550
## iter  20 value 9.195885
## iter  30 value 8.782977
## iter  40 value 8.681808
## iter  50 value 8.660961
## iter  60 value 8.656798
## iter  70 value 8.654281
## iter  80 value 8.654146
## final  value 8.654138
## converged
## # weights:  48 (35 variable)
## initial  value 141.548998
## iter  10 value 17.001971
## iter  20 value 0.856613
## iter  30 value 0.248509
## iter  40 value 0.187528
## iter  50 value 0.151156
## iter  60 value 0.141511
## iter  70 value 0.128400
## iter  80 value 0.124364
## iter  90 value 0.121915
## iter 100 value 0.121104
## final  value 0.121104
## stopped after 100 iterations
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 51.154626
## iter  20 value 5.358411
## iter  30 value 0.021845
## iter  40 value 0.000784
## final  value 0.000027
## converged
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 52.290418
## iter  20 value 12.835297
## iter  30 value 10.580419
## iter  40 value 10.417425
## iter  50 value 10.383851
## iter  60 value 10.377611
## iter  70 value 10.375046
## iter  80 value 10.374303
## iter  90 value 10.374041
## iter 100 value 10.373819
## final  value 10.373819
## stopped after 100 iterations
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 51.155769
## iter  20 value 5.369209
## iter  30 value 0.226549
## iter  40 value 0.188863
## iter  50 value 0.164981
## iter  60 value 0.159444
## iter  70 value 0.145893
## iter  80 value 0.141674
## iter  90 value 0.137021
## iter 100 value 0.132393
## final  value 0.132393
## stopped after 100 iterations
## # weights:  40 (28 variable)
## initial  value 127.145595
## iter  10 value 32.807826
## iter  20 value 0.769667
## iter  30 value 0.001656
## final  value 0.000052
## converged
## # weights:  40 (28 variable)
## initial  value 127.145595
## iter  10 value 33.497478
## iter  20 value 7.110354
## iter  30 value 6.746339
## iter  40 value 6.564488
## iter  50 value 6.548644
## iter  60 value 6.536083
## iter  70 value 6.522318
## final  value 6.522317
## converged
## # weights:  40 (28 variable)
## initial  value 127.145595
## iter  10 value 32.808526
## iter  20 value 0.797600
## iter  30 value 0.154865
## iter  40 value 0.123289
## iter  50 value 0.097058
## iter  60 value 0.087226
## iter  70 value 0.079771
## iter  80 value 0.077196
## iter  90 value 0.074960
## iter 100 value 0.067176
## final  value 0.067176
## stopped after 100 iterations
## # weights:  48 (35 variable)
## initial  value 141.548998
## iter  10 value 56.162320
## iter  20 value 3.237015
## iter  30 value 0.058886
## iter  40 value 0.004859
## iter  50 value 0.001388
## final  value 0.000076
## converged
## # weights:  48 (35 variable)
## initial  value 141.548998
## iter  10 value 37.006711
## iter  20 value 10.005032
## iter  30 value 8.432321
## iter  40 value 8.338280
## iter  50 value 8.319770
## iter  60 value 8.316393
## iter  70 value 8.315619
## iter  80 value 8.315259
## final  value 8.315237
## converged
## # weights:  48 (35 variable)
## initial  value 141.548998
## iter  10 value 56.164370
## iter  20 value 3.249091
## iter  30 value 0.238977
## iter  40 value 0.175146
## iter  50 value 0.157407
## iter  60 value 0.148771
## iter  70 value 0.138645
## iter  80 value 0.133566
## iter  90 value 0.128546
## iter 100 value 0.124754
## final  value 0.124754
## stopped after 100 iterations
## # weights:  48 (35 variable)
## initial  value 141.548998
## iter  10 value 10.695706
## iter  20 value 0.340744
## iter  30 value 0.006566
## iter  40 value 0.000905
## final  value 0.000069
## converged
## # weights:  48 (35 variable)
## initial  value 141.548998
## iter  10 value 13.280168
## iter  20 value 8.936366
## iter  30 value 8.346602
## iter  40 value 8.297244
## iter  50 value 8.289910
## iter  60 value 8.289539
## iter  70 value 8.289368
## iter  70 value 8.289367
## iter  70 value 8.289367
## final  value 8.289367
## converged
## # weights:  48 (35 variable)
## initial  value 141.548998
## iter  10 value 10.698322
## iter  20 value 0.404135
## iter  30 value 0.178476
## iter  40 value 0.164870
## iter  50 value 0.148268
## iter  60 value 0.135913
## iter  70 value 0.128346
## iter  80 value 0.120697
## iter  90 value 0.114981
## iter 100 value 0.113369
## final  value 0.113369
## stopped after 100 iterations
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 51.995007
## iter  20 value 7.529170
## iter  30 value 0.044836
## final  value 0.000037
## converged
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 52.469232
## iter  20 value 13.335203
## iter  30 value 9.465995
## iter  40 value 9.318174
## iter  50 value 9.295935
## iter  60 value 9.291938
## iter  70 value 9.291218
## iter  80 value 9.291090
## final  value 9.291066
## converged
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 51.995484
## iter  20 value 7.541295
## iter  30 value 0.227916
## iter  40 value 0.195094
## iter  50 value 0.159839
## iter  60 value 0.141458
## iter  70 value 0.132195
## iter  80 value 0.128237
## iter  90 value 0.116466
## iter 100 value 0.112507
## final  value 0.112507
## stopped after 100 iterations
## # weights:  48 (35 variable)
## initial  value 141.548998
## iter  10 value 26.814274
## iter  20 value 3.337491
## iter  30 value 0.026058
## final  value 0.000051
## converged
## # weights:  48 (35 variable)
## initial  value 141.548998
## iter  10 value 28.179253
## iter  20 value 8.194786
## iter  30 value 6.927215
## iter  40 value 6.901047
## iter  50 value 6.873601
## iter  60 value 6.858475
## iter  70 value 6.855689
## iter  80 value 6.854679
## final  value 6.854626
## converged
## # weights:  48 (35 variable)
## initial  value 141.548998
## iter  10 value 26.815655
## iter  20 value 3.347469
## iter  30 value 0.158008
## iter  40 value 0.139610
## iter  50 value 0.108112
## iter  60 value 0.093077
## iter  70 value 0.074678
## iter  80 value 0.069742
## iter  90 value 0.066699
## iter 100 value 0.061804
## final  value 0.061804
## stopped after 100 iterations
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 31.838450
## iter  20 value 1.507277
## iter  30 value 0.040427
## iter  40 value 0.002707
## iter  50 value 0.000836
## final  value 0.000085
## converged
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 34.900601
## iter  20 value 11.864816
## iter  30 value 10.792848
## iter  40 value 10.695758
## iter  50 value 10.687698
## iter  60 value 10.686885
## iter  70 value 10.686761
## final  value 10.686753
## converged
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 31.841555
## iter  20 value 1.540988
## iter  30 value 0.309636
## iter  40 value 0.196821
## iter  50 value 0.160429
## iter  60 value 0.151257
## iter  70 value 0.142794
## iter  80 value 0.132554
## iter  90 value 0.128000
## iter 100 value 0.123995
## final  value 0.123995
## stopped after 100 iterations
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 23.900110
## iter  20 value 1.447643
## iter  30 value 0.007902
## final  value 0.000078
## converged
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 25.003954
## iter  20 value 7.495253
## iter  30 value 6.532706
## iter  40 value 6.476606
## iter  50 value 6.474762
## iter  60 value 6.474685
## iter  70 value 6.474670
## iter  80 value 6.474668
## iter  80 value 6.474668
## iter  80 value 6.474668
## final  value 6.474668
## converged
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 23.901196
## iter  20 value 1.461418
## iter  30 value 0.099445
## iter  40 value 0.088551
## iter  50 value 0.079829
## iter  60 value 0.069404
## iter  70 value 0.059109
## iter  80 value 0.045252
## iter  90 value 0.044396
## iter 100 value 0.042700
## final  value 0.042700
## stopped after 100 iterations
## # weights:  56 (42 variable)
## initial  value 153.726902
## iter  10 value 38.701008
## iter  20 value 11.445468
## iter  30 value 10.395524
## iter  40 value 10.316670
## iter  50 value 10.313653
## iter  60 value 10.313589
## iter  70 value 10.313569
## final  value 10.313567
## converged
```

```
prediction.LR.FattyAcids<-predict(lr.FattyAcids,test.FattyAcids)

confusionMatrix(data =prediction.LR.FattyAcids,
                reference = test.OilType)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction A B C D E F G
##          A 5 1 0 0 0 0 0
##          B 2 4 0 0 0 0 0
##          C 0 0 0 0 0 0 0
##          D 0 0 0 1 0 0 0
##          E 0 0 0 0 2 0 0
##          F 0 0 0 0 0 2 0
##          G 0 0 0 0 0 0 0
## 
## Overall Statistics
##                                           
##                Accuracy : 0.8235          
##                  95% CI : (0.5657, 0.962) 
##     No Information Rate : 0.4118          
##     P-Value [Acc > NIR] : 0.0006427       
##                                           
##                   Kappa : 0.7548          
##                                           
##  Mcnemar's Test P-Value : NA              
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity            0.7143   0.8000       NA  1.00000   1.0000   1.0000
## Specificity            0.9000   0.8333        1  1.00000   1.0000   1.0000
## Pos Pred Value         0.8333   0.6667       NA  1.00000   1.0000   1.0000
## Neg Pred Value         0.8182   0.9091       NA  1.00000   1.0000   1.0000
## Prevalence             0.4118   0.2941        0  0.05882   0.1176   0.1176
## Detection Rate         0.2941   0.2353        0  0.05882   0.1176   0.1176
## Detection Prevalence   0.3529   0.3529        0  0.05882   0.1176   0.1176
## Balanced Accuracy      0.8071   0.8167       NA  1.00000   1.0000   1.0000
##                      Class: G
## Sensitivity                NA
## Specificity                 1
## Pos Pred Value             NA
## Neg Pred Value             NA
## Prevalence                0
## Detection Rate            0
## Detection Prevalence      0
## Balanced Accuracy        NA
```

```
plot(lr.FattyAcids)
```



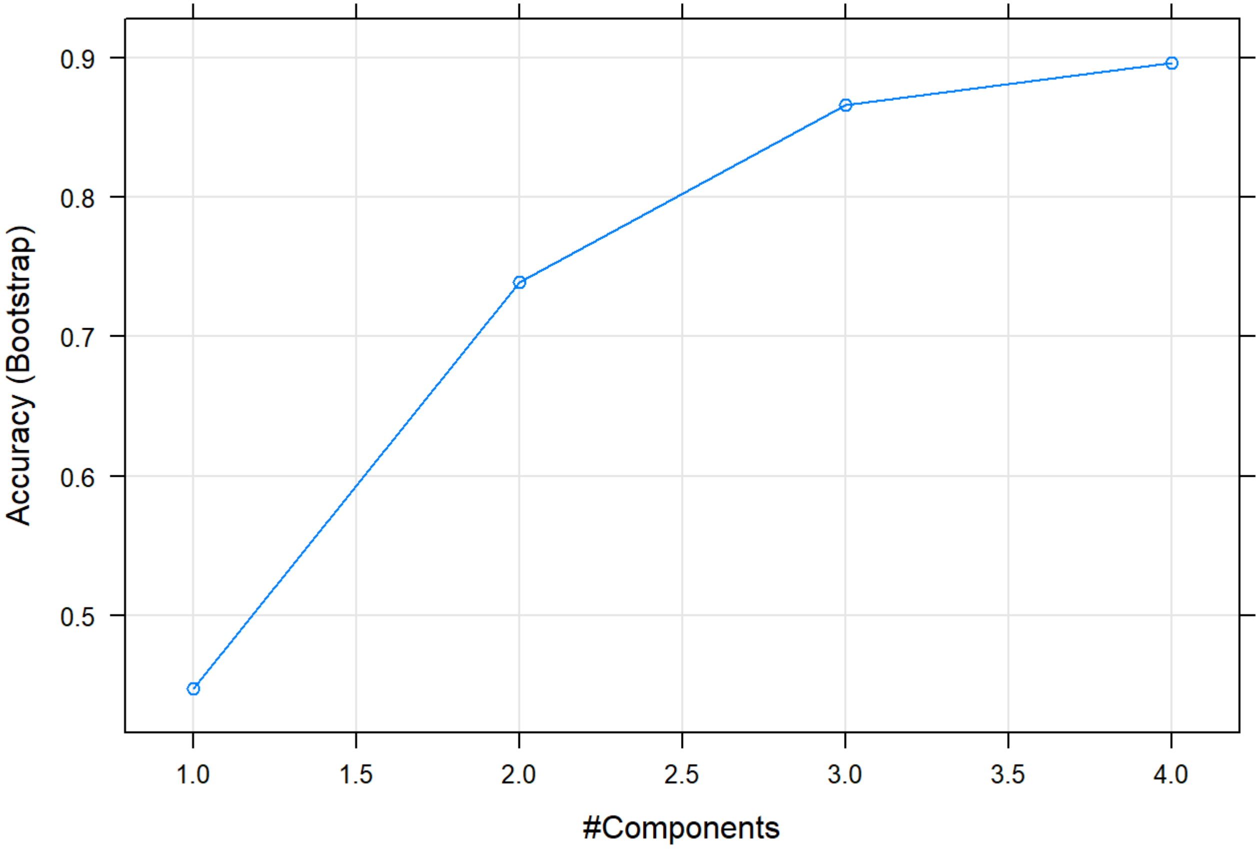
```r
############ Linear Discriminant Analysis #############

# LDA Analysis
library(MASS)
set.seed(678)


lda.FattyAcids <- train(x = train.FattyAcids,
                y = train.OilType,
                method = "lda",
                metric = "Accuracy",
                trControl = ctrl)

prediction.LDA.FattyAcids <-predict(lda.FattyAcids,test.FattyAcids)
confusionMatrix(data =prediction.LDA.FattyAcids,
                reference = test.OilType)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction A B C D E F G
##          A 5 0 0 0 0 0 0
##          B 2 5 0 0 0 0 0
##          C 0 0 0 0 0 0 0
##          D 0 0 0 1 0 0 0
##          E 0 0 0 0 2 0 0
##          F 0 0 0 0 0 2 0
##          G 0 0 0 0 0 0 0
## 
## Overall Statistics
## 
##                Accuracy : 0.8824          
##                  95% CI : (0.6356, 0.9854)
##     No Information Rate : 0.4118          
##     P-Value [Acc > NIR] : 8.516e-05       
## 
##                   Kappa : 0.8381          
## 
##  Mcnemar's Test P-Value : NA              
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity            0.7143   1.0000       NA  1.00000   1.0000   1.0000
## Specificity            1.0000   0.8333        1  1.00000   1.0000   1.0000
## Pos Pred Value         1.0000   0.7143       NA  1.00000   1.0000   1.0000
## Neg Pred Value         0.8333   1.0000       NA  1.00000   1.0000   1.0000
## Prevalence             0.4118   0.2941        0  0.05882   0.1176   0.1176
## Detection Rate         0.2941   0.2941        0  0.05882   0.1176   0.1176
## Detection Prevalence   0.2941   0.4118        0  0.05882   0.1176   0.1176
## Balanced Accuracy      0.8571   0.9167       NA  1.00000   1.0000   1.0000
##                      Class: G
## Sensitivity                NA
## Specificity                 1
## Pos Pred Value             NA
## Neg Pred Value             NA
## Prevalence                  0
## Detection Rate              0
## Detection Prevalence        0
## Balanced Accuracy          NA
```

```
############## Partial Least Squares Discriminant Analysis ###############

library(MASS)
set.seed(123)
pls.FattyAcids <- train(x = train.FattyAcids,
                y = train.OilType,
                method = "pls",
                tuneGrid = expand.grid(.ncomp = 1:4),
                metric = "Accuracy",
                trControl = ctrl)

prediction.PLS.FattyAcids <-predict(pls.FattyAcids,test.FattyAcids)
confusionMatrix(data =prediction.PLS.FattyAcids,
                reference = test.OilType)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction A B C D E F G
##          A 5 0 0 0 0 0 0
##          B 2 5 0 0 0 0 0
##          C 0 0 0 0 0 0 0
##          D 0 0 0 1 0 0 0
##          E 0 0 0 0 2 0 0
##          F 0 0 0 0 0 2 0
##          G 0 0 0 0 0 0 0
## 
## Overall Statistics
## 
##                Accuracy : 0.8824          
##                  95% CI : (0.6356, 0.9854)
##     No Information Rate : 0.4118          
##     P-Value [Acc > NIR] : 8.516e-05       
## 
##                   Kappa : 0.8381          
## 
##  Mcnemar's Test P-Value : NA              
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity            0.7143   1.0000       NA  1.00000   1.0000   1.0000
## Specificity            1.0000   0.8333        1  1.00000   1.0000   1.0000
## Pos Pred Value         1.0000   0.7143       NA  1.00000   1.0000   1.0000
## Neg Pred Value         0.8333   1.0000       NA  1.00000   1.0000   1.0000
## Prevalence             0.4118   0.2941        0  0.05882   0.1176   0.1176
## Detection Rate         0.2941   0.2941        0  0.05882   0.1176   0.1176
## Detection Prevalence   0.2941   0.4118        0  0.05882   0.1176   0.1176
## Balanced Accuracy      0.8571   0.9167       NA  1.00000   1.0000   1.0000
##                      Class: G
## Sensitivity                NA
## Specificity                 1
## Pos Pred Value             NA
## Neg Pred Value             NA
## Prevalence                  0
## Detection Rate              0
## Detection Prevalence        0
## Balanced Accuracy          NA
```

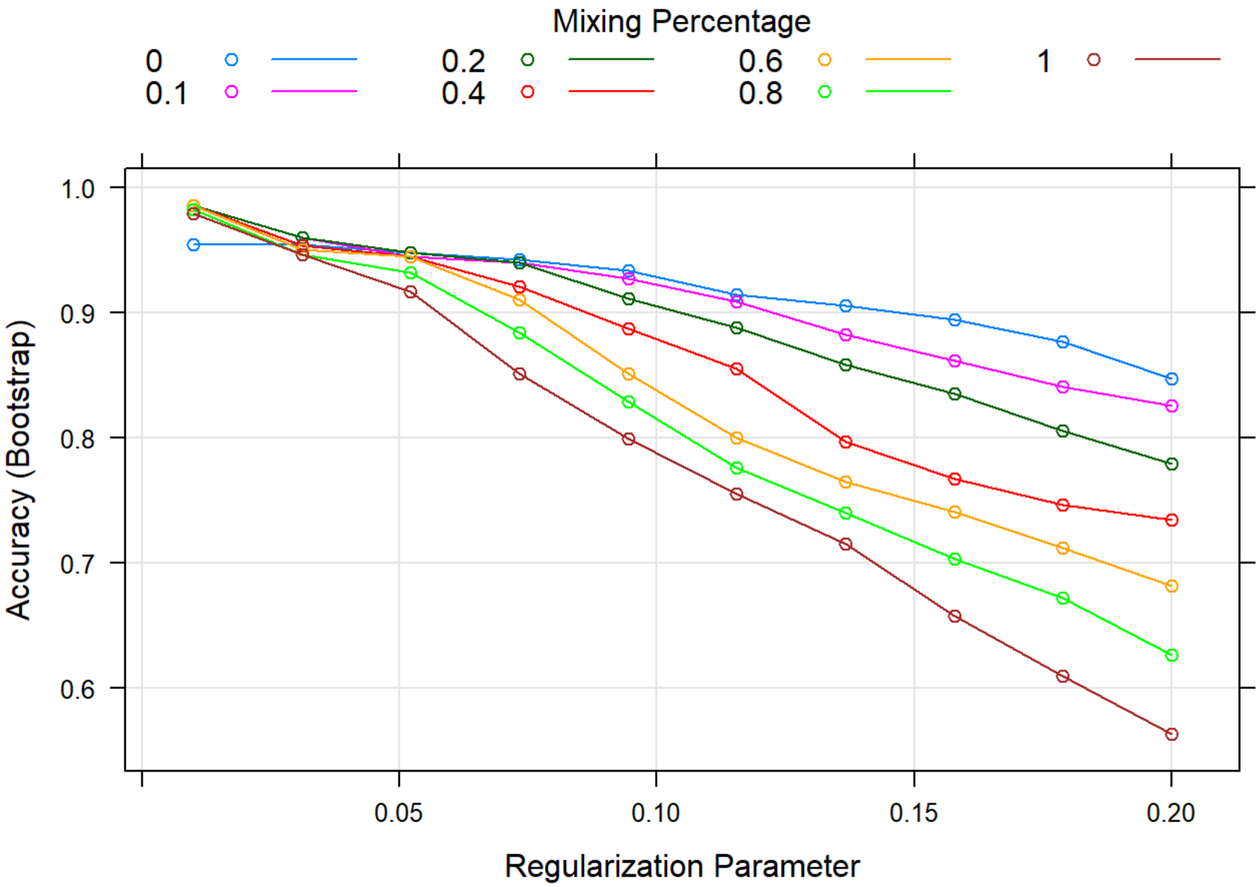
```
plot(pls.FattyAcids)
```



```
########### Penalized Models ###########

########### Penalized Models for Logistic Regression ###########
glmnGrid <- expand.grid(.alpha = c(0, .1, .2, .4, .6, .8, 1),
                        .lambda = seq(.01, .2, length = 10))
set.seed(123)

glmn.Tuned.LR.FattyAcids<- train(x=train.FattyAcids,
                        y =train.OilType,
                        method = "glmnet",
                        tuneGrid = glmnGrid,
                        metric = "Accuracy",
                        trControl = ctrl)

prediction.Glmnet.FattyAcids <-  predict(glmn.Tuned.LR.FattyAcids,test.FattyAcids)
confusionMatrix(data =prediction.Glmnet.FattyAcids,reference = test.OilType)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction A B C D E F G
##          A 5 0 0 0 0 0 0
##          B 2 5 0 0 0 0 0
##          C 0 0 0 0 0 0 0
##          D 0 0 0 1 0 0 0
##          E 0 0 0 0 2 0 0
##          F 0 0 0 0 0 2 0
##          G 0 0 0 0 0 0 0
## 
## Overall Statistics
## 
##                Accuracy : 0.8824
##                  95% CI : (0.6356, 0.9854)
##     No Information Rate : 0.4118
##     P-Value [Acc > NIR] : 8.516e-05
## 
##                   Kappa : 0.8381
## 
##  Mcnemar's Test P-Value : NA
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity            0.7143   1.0000       NA  1.00000   1.0000   1.0000
## Specificity            1.0000   0.8333        1  1.00000   1.0000   1.0000
## Pos Pred Value         1.0000   0.7143       NA  1.00000   1.0000   1.0000
## Neg Pred Value         0.8333   1.0000       NA  1.00000   1.0000   1.0000
## Prevalence             0.4118   0.2941        0  0.05882   0.1176   0.1176
## Detection Rate         0.2941   0.2941        0  0.05882   0.1176   0.1176
## Detection Prevalence   0.2941   0.4118        0  0.05882   0.1176   0.1176
## Balanced Accuracy      0.8571   0.9167       NA  1.00000   1.0000   1.0000
##                      Class: G
## Sensitivity                NA
## Specificity                 1
## Pos Pred Value             NA
## Neg Pred Value             NA
## Prevalence                 0
## Detection Rate             0
## Detection Prevalence       0
## Balanced Accuracy         NA
```

```
plot(glmn.Tuned.LR.FattyAcids)
```



```
########### Penalized Models for LDA ###########

library(sparseLDA)
set.seed(123)
sparse.Lda.ModelFattyAcids <- sda(x=train.FattyAcids,
                         y =train.OilType,
                         lambda = 0.01,
                         stop = -7)
## the ridge parameter called lambda.

prediction.Sparse.LDA.FattyAcids <-  predict(sparse.Lda.ModelFattyAcids,test.FattyAcids)
confusionMatrix(data =prediction.Sparse.LDA.FattyAcids$class, reference = test.OilType)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction A B C D E F G
##          A 0 2 0 0 0 0 0
##          B 0 0 0 0 0 0 0
##          C 0 0 0 0 0 0 0
##          D 7 3 0 1 2 2 0
##          E 0 0 0 0 0 0 0
##          F 0 0 0 0 0 0 0
##          G 0 0 0 0 0 0 0
## 
## Overall Statistics
## 
##                Accuracy : 0.0588          
##                  95% CI : (0.0015, 0.2869)
##     No Information Rate : 0.4118          
##     P-Value [Acc > NIR] : 0.9999          
## 
##                   Kappa : -0.0462         
## 
##  Mcnemar's Test P-Value : NA              
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity            0.0000   0.0000       NA  1.00000   0.0000   0.0000
## Specificity            0.8000   1.0000        1  0.12500   1.0000   1.0000
## Pos Pred Value         0.0000      NaN       NA  0.06667      NaN      NaN
## Neg Pred Value         0.5333   0.7059       NA  1.00000   0.8824   0.8824
## Prevalence             0.4118   0.2941        0  0.05882   0.1176   0.1176
## Detection Rate         0.0000   0.0000        0  0.05882   0.0000   0.0000
## Detection Prevalence   0.1176   0.0000        0  0.88235   0.0000   0.0000
## Balanced Accuracy      0.4000   0.5000       NA  0.56250   0.5000   0.5000
##                      Class: G
## Sensitivity                NA
## Specificity                 1
## Pos Pred Value             NA
## Neg Pred Value             NA
## Prevalence                 0
## Detection Rate             0
## Detection Prevalence       0
## Balanced Accuracy         NA
```

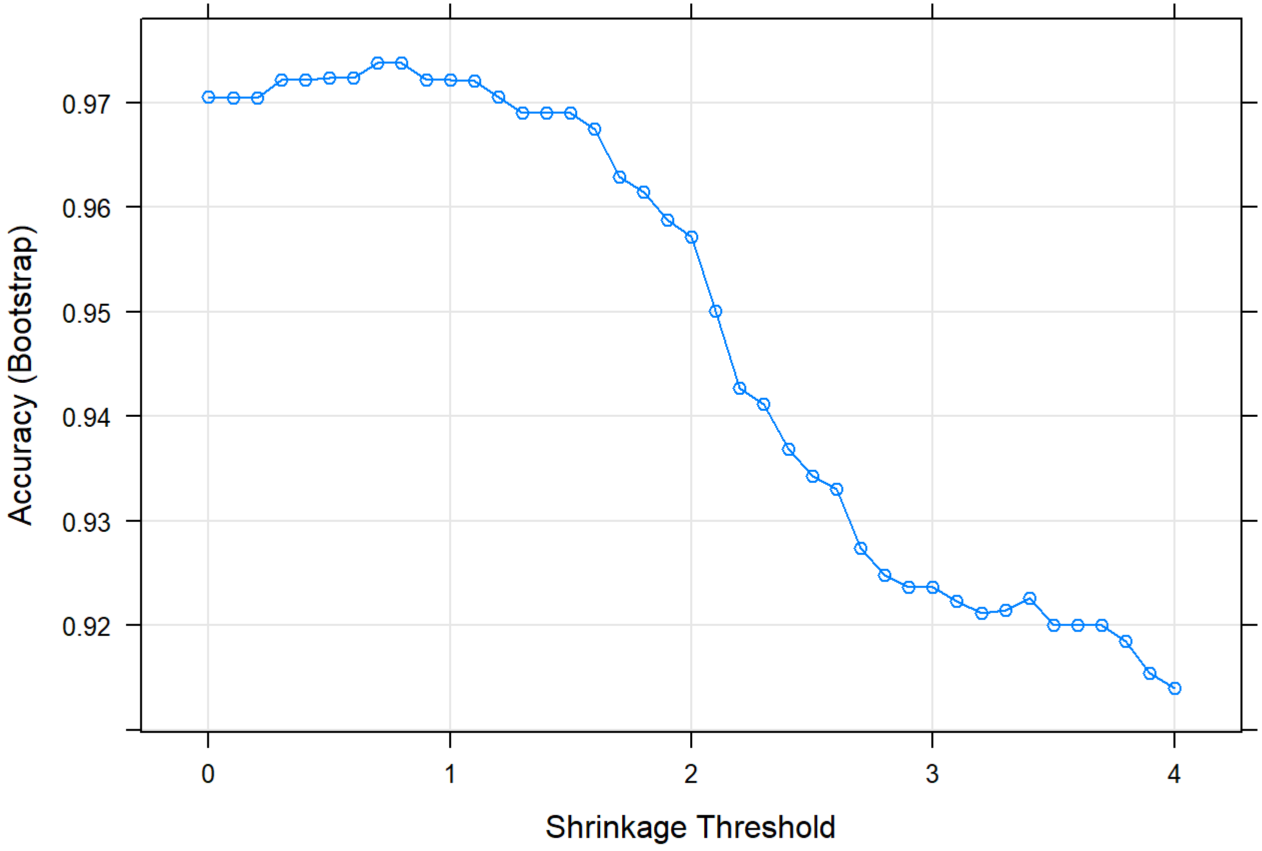
```
########### Nearest Shrunken Centroids ###########

library(pamr)
nsc.Grid.FattyAcids <- data.frame(.threshold = seq(0,4, by=0.1))
set.seed(123)
nsc.Tuned.FattyAcids <- train(x = train.FattyAcids,
                     y = train.OilType,
                     method = "pam",
                     tuneGrid = nsc.Grid.FattyAcids,
                     metric = "Accuracy",
                     trControl = ctrl)
```

```
## 11Warning: a class contains only 1 sample1Warning: a class contains only 1 sample111Warning: a class contains only 1 samp
le111Warning: a class contains only 1 sample1Warning: a class contains only 1 sample1111111Warning: a class contains only 1
sample1Warning: a class contains only 1 sample1Warning: a class contains only 1 sample111Warning: a class contains only 1 sa
mple1Warning: a class contains only 1 sample1Warning: a class contains only 1 sample11
```

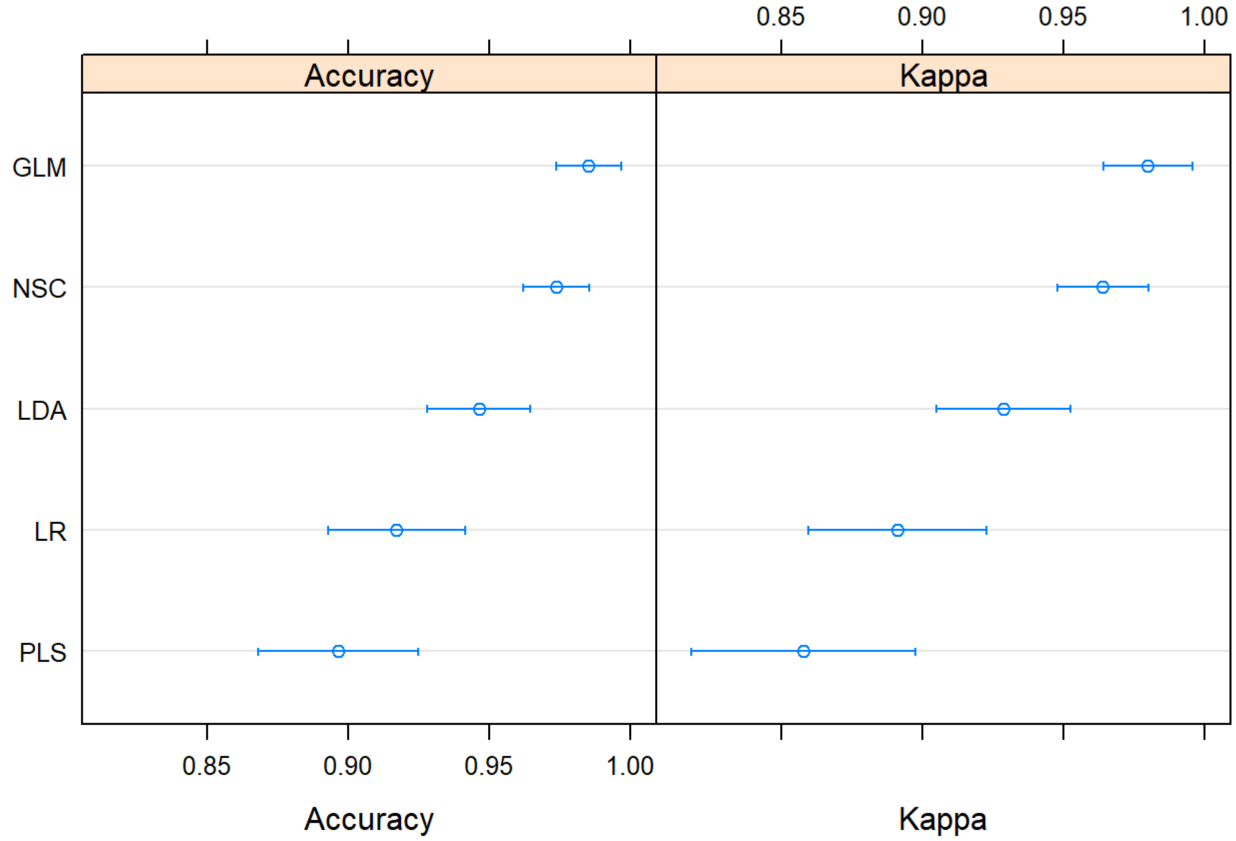
```
prediction.NSC.FattyAcids <-predict(nsc.Tuned.FattyAcids,test.FattyAcids)
confusionMatrix(data =prediction.NSC.FattyAcids, reference = test.OilType)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction A B C D E F G
##          A 5 0 0 0 0 0 0
##          B 2 5 0 0 0 0 0
##          C 0 0 0 0 0 0 0
##          D 0 0 0 1 0 0 0
##          E 0 0 0 0 2 0 0
##          F 0 0 0 0 0 2 0
##          G 0 0 0 0 0 0 0
## 
## Overall Statistics
## 
##                Accuracy : 0.8824          
##                  95% CI : (0.6356, 0.9854)
##     No Information Rate : 0.4118          
##     P-Value [Acc > NIR] : 8.516e-05       
## 
##                   Kappa : 0.8381          
## 
##  Mcnemar's Test P-Value : NA              
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity            0.7143   1.0000       NA  1.00000   1.0000   1.0000
## Specificity            1.0000   0.8333        1  1.00000   1.0000   1.0000
## Pos Pred Value         1.0000   0.7143       NA  1.00000   1.0000   1.0000
## Neg Pred Value         0.8333   1.0000       NA  1.00000   1.0000   1.0000
## Prevalence             0.4118   0.2941        0  0.05882   0.1176   0.1176
## Detection Rate         0.2941   0.2941        0  0.05882   0.1176   0.1176
## Detection Prevalence   0.2941   0.4118        0  0.05882   0.1176   0.1176
## Balanced Accuracy      0.8571   0.9167       NA  1.00000   1.0000   1.0000
##                      Class: G
## Sensitivity                NA
## Specificity                 1
## Pos Pred Value             NA
## Neg Pred Value             NA
## Prevalence                 0
## Detection Rate             0
## Detection Prevalence       0
## Balanced Accuracy         NA
```

```
plot(nsc.Tuned.FattyAcids)
```



```
# Combining the models from Question 12
res12 = resamples(list(LR=lr.FattyAcids,LDA=lda.FattyAcids,PLS=pls.FattyAcids,GLM=glmn.Tuned.LR.FattyAcids,NSC=nsc.Tuned.Fat
tyAcids ))
dotplot(res12)
```



Confidence Level: 0.95

Based on the output of the resamples function on the training data and using the classification statistic **"Accuracy"**, and also matching with the Confusion Matrix data for test data, we can say that **GLM (Penalized Model for Logistic Regression)** is the best model as far as accuracy of prediction is concerned.

And from the data, we can also see that **PLS (Partial Least Square Discriminant Analysis)** is the model with the least accurate prediction.

Chapter 13 - E-Book - Applied Predictive Modelling - Exercises pages 367:

Q 13.2

Use the fatty acid data from the previous exercise set (Exercise 12.2).

Q 13.2 a

a. Use the same data splitting approach (if any) and pre-processing steps that you did in the previous chapter. Using the same classification statistic as before, build models described in this chapter for these data. Which model has the best predictive ability? How does this optimal model's performance compare to the best linear model's performance? Would you infer that the data have nonlinear separation boundaries based on this comparison?

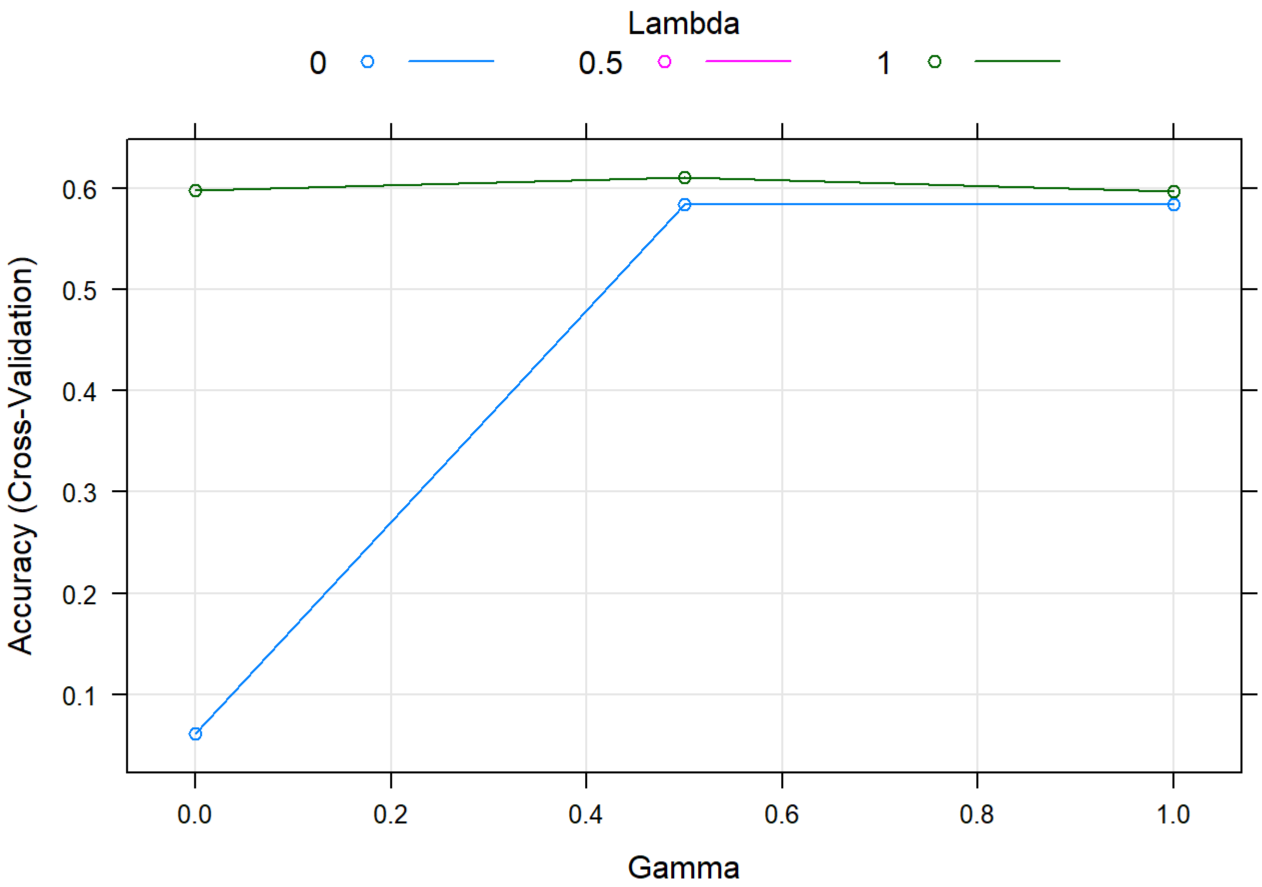
Answer (13.2 a)

So, the classification statistic that I will be using here is the same as the one I used in the previous question which is "Accuracy" rate. This is the simplest statistic as it reflects the agreement between the observed and predicted classes and so has the most straight forward interpretation.

```
### Create a control function that will be used across models.
#set.seed(100)

ctrl <- trainControl(summaryFunction = defaultSummary, classProbs = TRUE)



######################### Regularized discriminant analysis (RDA) ##################################

 set.seed(476)
 library(klaR)

 rda_grid = trainControl(method = "cv", number = 5)

 RDATune <- train(x=train.FattyAcids,
 y = train.OilType,
 method = "rda",
 metric = "Accuracy",
 trControl = rda_grid)

 RDATune
```

```
## Regularized Discriminant Analysis
## 
## 79 samples
##  6 predictor
##  7 classes: 'A', 'B', 'C', 'D', 'E', 'F', 'G' 
## 
## No pre-processing
## Resampling: Cross-Validated (5 fold) 
## Summary of sample sizes: 63, 64, 64, 62, 63 
## Resampling results across tuning parameters:
## 
##   gamma  lambda  Accuracy    Kappa    
##   0.0    0.0     0.06066176  0.0000000
##   0.0    0.5     0.59843137  0.5653438
##   0.0    1.0     0.59843137  0.5657895
##   0.5    0.0     0.58509804  0.5469163
##   0.5    0.5     0.61093137  0.5824561
##   0.5    1.0     0.61093137  0.5824561
##   1.0    0.0     0.58509804  0.5469163
##   1.0    0.5     0.59759804  0.5639376
##   1.0    1.0     0.59759804  0.5653179
## 
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were gamma = 0.5 and lambda = 1.
```

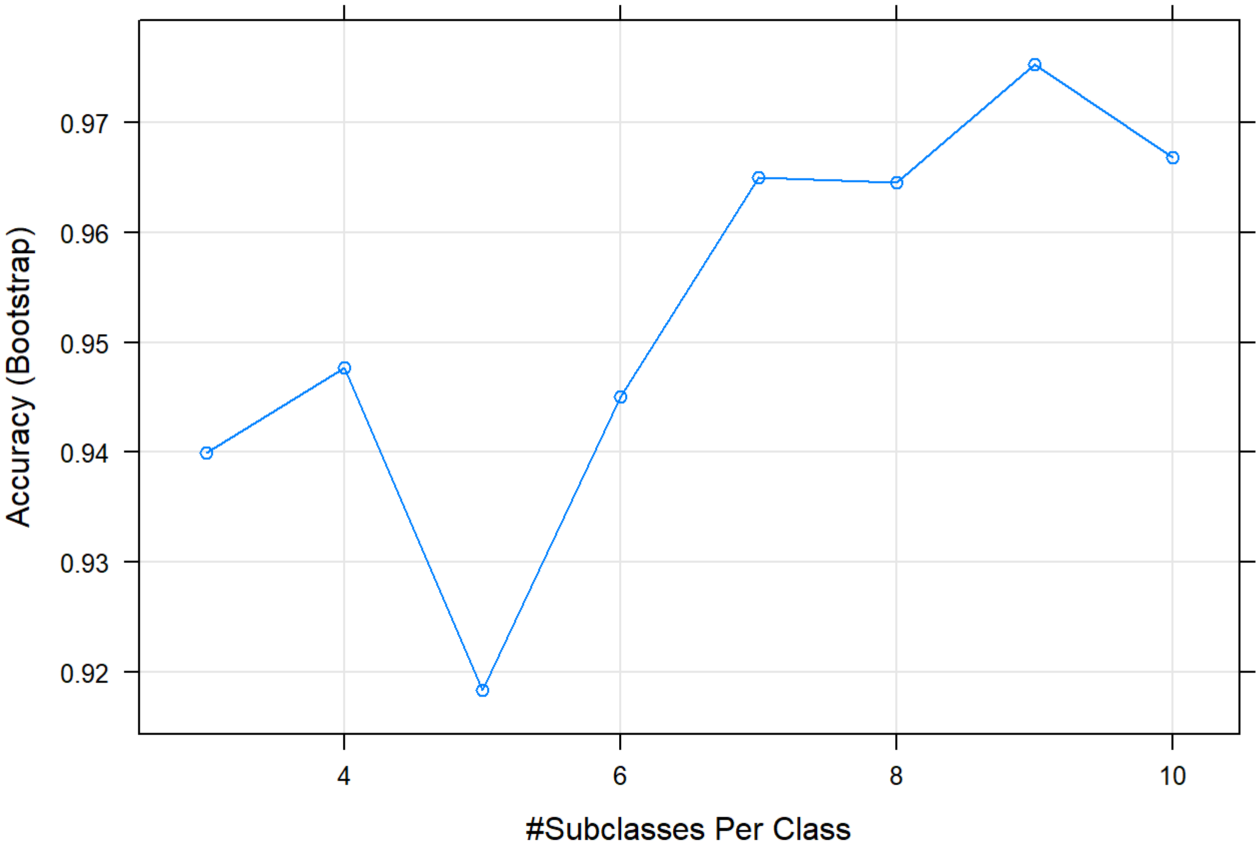
```
plot(RDATune)
```



```
#################################### Mixture discriminant analysis (MDA) #######################
set.seed(476)

library(mda)
MDATune <- train(as.matrix(train.FattyAcids),
y =  train.OilType,
method = "mda",
tuneGrid = expand.grid(.subclasses = 3:10),
metric = "Accuracy",
trControl = ctrl)

MDATune
```

```
## Mixture Discriminant Analysis
##
## 79 samples
##  6 predictor
##  7 classes: 'A', 'B', 'C', 'D', 'E', 'F', 'G'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 79, 79, 79, 79, 79, 79, ...
## Resampling results across tuning parameters:
##
##   subclasses  Accuracy   Kappa
##    3          0.9399608  0.9177847
##    4          0.9477165  0.9277604
##    5          0.9183273  0.8860572
##    6          0.9450660  0.9227933
##    7          0.9649807  0.9533588
##    8          0.9645560  0.9535669
##    9          0.9753166  0.9665862
##   10          0.9668271  0.9544511
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was subclasses = 9.
```

```
plot(MDATune)
```



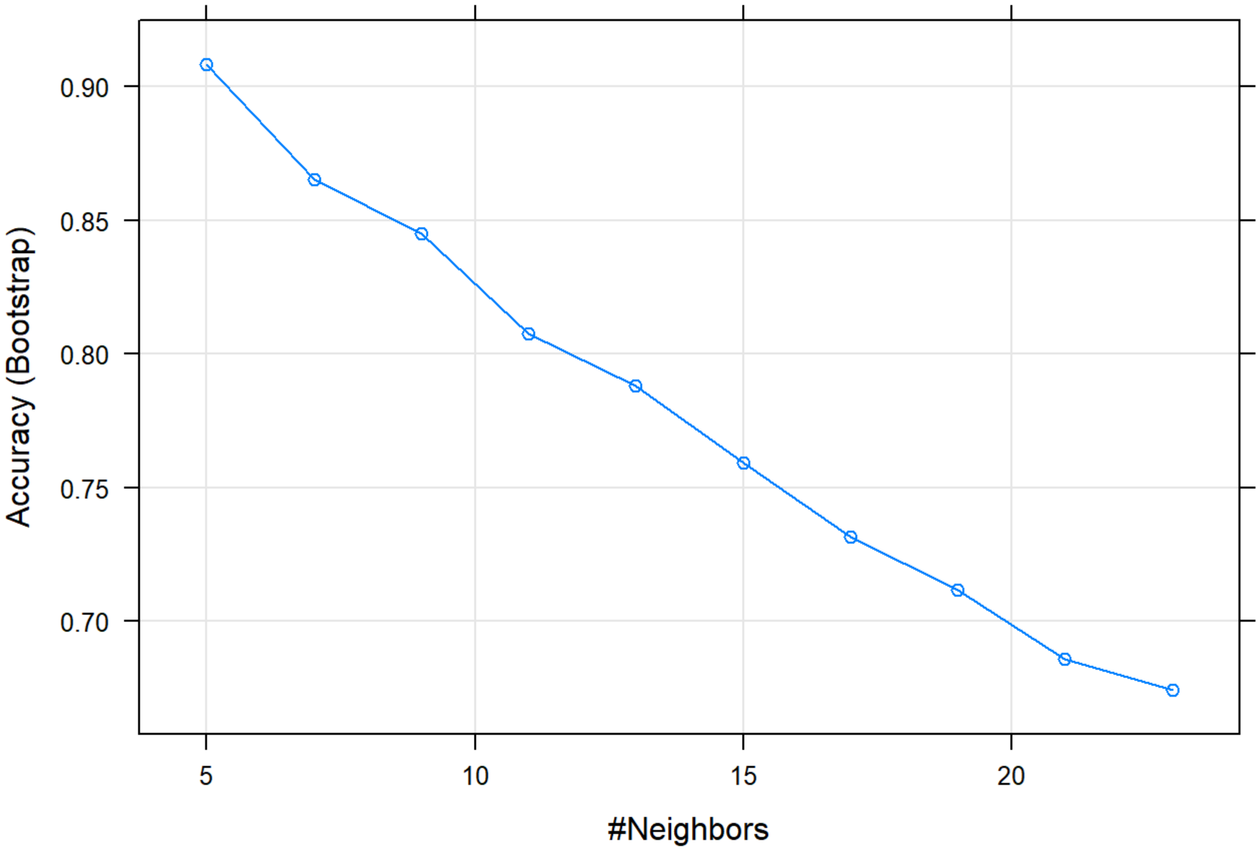
```
############################### Naive Bayes (NB) ################################################

 set.seed(476)

 NBTune <- train(x = as.matrix(train.FattyAcids),
 y = train.OilType,
 method = "nb",
 preProc = c('center', 'scale'),
 metric = "Accuracy",
 trControl = ctrl)

 NBTune
```

```
## Naive Bayes
##
## 79 samples
##  6 predictor
##  7 classes: 'A', 'B', 'C', 'D', 'E', 'F', 'G'
##
## Pre-processing: centered (6), scaled (6)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 79, 79, 79, 79, 79, 79, ...
## Resampling results across tuning parameters:
##
##   usekernel  Accuracy   Kappa
##   FALSE            NaN        NaN
##    TRUE      0.9317812  0.9070348
##
## Tuning parameter 'fL' was held constant at a value of 0
## Tuning
##  parameter 'adjust' was held constant at a value of 1
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were fL = 0, usekernel = TRUE and adjust
##  = 1.
```

```
############################### K-nearest neighbors (KNN) #########################################

set.seed(1)
KNNTune = train(x=train.FattyAcids, y=train.OilType, method="knn", metric = "Accuracy",preProcess=c("center","scale"), tuneL
ength=10)
KNNTune
```

```
## k-Nearest Neighbors
##
## 79 samples
##  6 predictor
##  7 classes: 'A', 'B', 'C', 'D', 'E', 'F', 'G'
##
## Pre-processing: centered (6), scaled (6)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 79, 79, 79, 79, 79, 79, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    5  0.9085544  0.8781189
##    7  0.8653627  0.8216568
##    9  0.8449985  0.7938198
##   11  0.8076641  0.7417653
##   13  0.7881356  0.7134347
##   15  0.7593568  0.6698862
##   17  0.7314335  0.6275864
##   19  0.7116438  0.5961364
##   21  0.6856207  0.5549648
##   23  0.6740550  0.5377973
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 5.
```

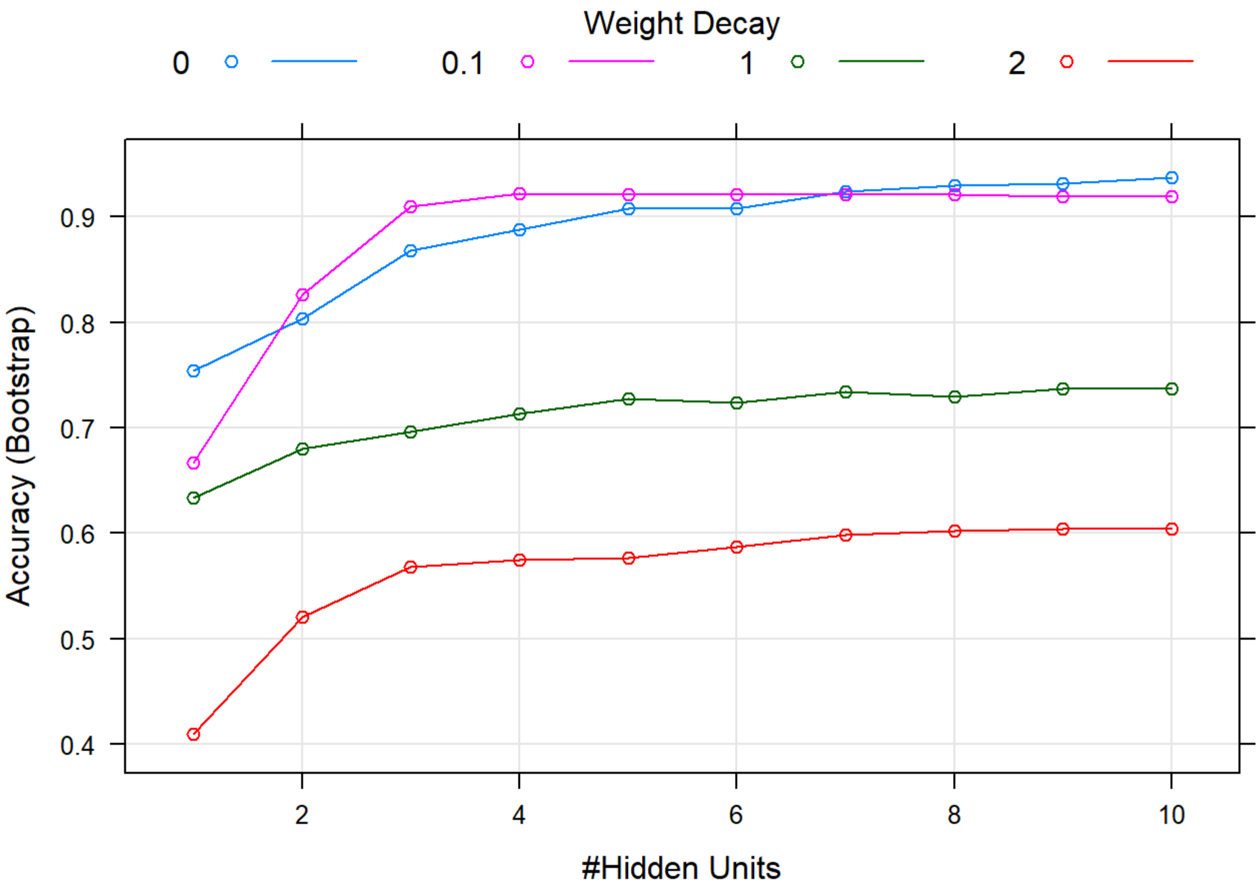
```
plot(KNNTune)
```


Accuracy (Bootstrap)
0.90
0.85
0.80
0.75
0.70
5
10
15
20
#Neighbors

```
############################### Neural networks (NN) ##############################################
 set.seed(476)

 nnetGrid <- expand.grid(.size = 1:10,
 .decay = c(0, .1, 1, 2))
 maxSize <- max(nnetGrid$.size)
 numWts <-200

 NNTune <- train(x = as.matrix(train.FattyAcids),
               y = train.OilType,
 method = "nnet",
 metric = "Accuracy",
 preProc = c("center", "scale", "spatialSign"),
 tuneGrid = nnetGrid,
 trace = FALSE,
 maxit = 2000,
 MaxNWts = numWts,
 trControl = ctrl)

 NNTune
```

```
## Neural Network
##
## 79 samples
##  6 predictor
##  7 classes: 'A', 'B', 'C', 'D', 'E', 'F', 'G'
##
## Pre-processing: centered (6), scaled (6), spatial sign transformation (6)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 79, 79, 79, 79, 79, 79, ...
## Resampling results across tuning parameters:
##
##   size  decay  Accuracy   Kappa
##    1    0.0    0.7538554  0.65573062
##    1    0.1    0.6668914  0.51397841
##    1    1.0    0.6332390  0.44049879
##    1    2.0    0.4093663  0.08997442
##    2    0.0    0.8037841  0.73740077
##    2    0.1    0.8258184  0.75985316
##    2    1.0    0.6800462  0.51411136
##    2    2.0    0.5207266  0.25701899
##    3    0.0    0.8680357  0.82080082
##    3    0.1    0.9098924  0.87692567
##    3    1.0    0.6965723  0.54180210
##    3    2.0    0.5682829  0.33650389
##    4    0.0    0.8874056  0.84721102
##    4    0.1    0.9217128  0.89361252
##    4    1.0    0.7131944  0.57107632
##    4    2.0    0.5745016  0.34335711
##    5    0.0    0.9075085  0.87678433
##    5    0.1    0.9206029  0.89198672
##    5    1.0    0.7276461  0.59302754
##    5    2.0    0.5767849  0.34907932
##    6    0.0    0.9073718  0.87539184
##    6    0.1    0.9206459  0.89191039
##    6    1.0    0.7233038  0.58726371
##    6    2.0    0.5868525  0.36141344
##    7    0.0    0.9241425  0.89731282
##    7    0.1    0.9206459  0.89190913
##    7    1.0    0.7341276  0.60370926
##    7    2.0    0.5987207  0.37874697
##    8    0.0    0.9294742  0.90538081
##    8    0.1    0.9206918  0.89223841
##    8    1.0    0.7293657  0.59787680
##    8    2.0    0.6018021  0.38402564
##    9    0.0    0.9317742  0.90813314
##    9    0.1    0.9193125  0.89015029
##    9    1.0    0.7371736  0.60977713
##    9    2.0    0.6044586  0.38798309
##   10    0.0    0.9367121  0.91421819
##   10    0.1    0.9193125  0.89033655
##   10    1.0    0.7369355  0.60990750
##   10    2.0    0.6045608  0.38853294
## 
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were size = 10 and decay = 0.
```

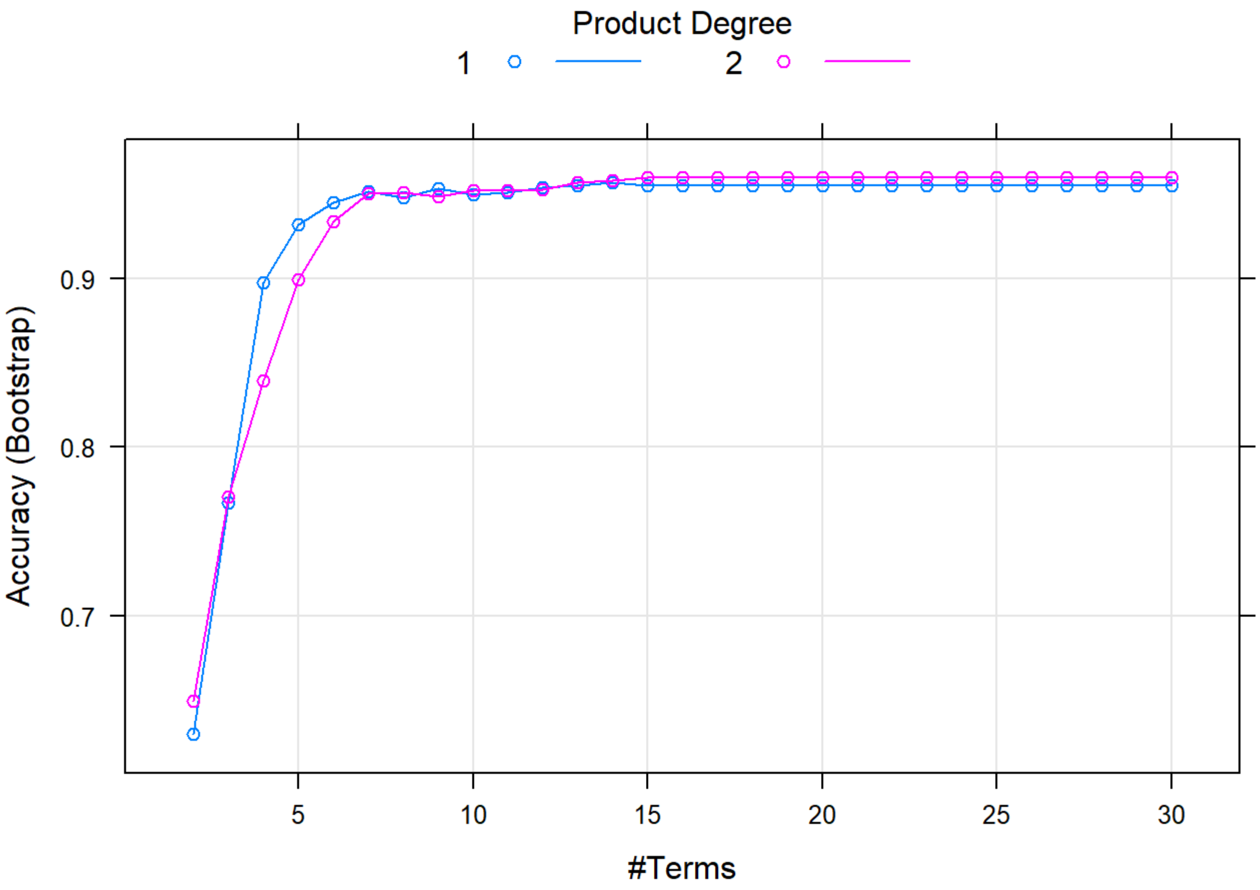
```
plot(NNTune)
```


Weight Decay
0
0.1
1
2
Accuracy (Bootstrap)
0.4
0.5
0.6
0.7
0.8
0.9
#Hidden Units
2
4
6
8
10

```
############################### Flexible discriminant analysis (FDA) ###############################

 set.seed(476)

 library(mda)
 FDATune <- train(as.matrix(train.FattyAcids),
 y =  train.OilType,
 method = "fda",
 tuneGrid = expand.grid(.nprune = 2:30,.degree = 1:2),
 metric = "Accuracy",
 trControl = ctrl)

 FDATune
```

```
## Flexible Discriminant Analysis
##
## 79 samples
##  6 predictor
##  7 classes: 'A', 'B', 'C', 'D', 'E', 'F', 'G'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 79, 79, 79, 79, 79, 79, ...
## Resampling results across tuning parameters:
##
##   nprune  degree  Accuracy   Kappa
##    2      1       0.6298498  0.4501024
##    2      2       0.6494450  0.4758130
##    3      1       0.7667217  0.6691438
##    3      2       0.7708481  0.6745045
##    4      1       0.8977735  0.8563852
##    4      2       0.8395508  0.7754163
##    5      1       0.9316972  0.9068045
##    5      2       0.8992119  0.8585319
##    6      1       0.9448666  0.9248773
##    6      2       0.9336017  0.9069360
##    7      1       0.9517139  0.9339442
##    7      2       0.9504849  0.9321043
##    8      1       0.9476869  0.9279072
##    8      2       0.9506034  0.9321027
##    9      1       0.9532736  0.9355339
##    9      2       0.9486826  0.9294467
##   10      1       0.9493888  0.9304827
##   10      2       0.9519720  0.9339302
##   11      1       0.9511271  0.9331036
##   11      2       0.9522983  0.9345158
##   12      1       0.9536868  0.9361965
##   12      2       0.9523873  0.9348404
##   13      1       0.9551656  0.9383075
##   13      2       0.9565676  0.9403595
##   14      1       0.9566471  0.9402144
##   14      2       0.9579709  0.9423340
##   15      1       0.9552185  0.9384904
##   15      2       0.9595053  0.9440537
##   16      1       0.9552185  0.9384904
##   16      2       0.9595053  0.9441142
##   17      1       0.9552185  0.9384904
##   17      2       0.9595053  0.9442221
##   18      1       0.9552185  0.9384904
##   18      2       0.9595053  0.9442566
##   19      1       0.9552185  0.9384904
##   19      2       0.9595053  0.9442954
##   20      1       0.9552185  0.9384904
##   20      2       0.9595053  0.9442954
##   21      1       0.9552185  0.9384904
##   21      2       0.9595053  0.9442954
##   22      1       0.9552185  0.9384904
##   22      2       0.9595053  0.9442954
##   23      1       0.9552185  0.9384904
##   23      2       0.9595053  0.9442954
##   24      1       0.9552185  0.9384904
##   24      2       0.9595053  0.9442954
##   25      1       0.9552185  0.9384904
##   25      2       0.9595053  0.9442954
##   26      1       0.9552185  0.9384904
##   26      2       0.9595053  0.9442954
##   27      1       0.9552185  0.9384904
##   27      2       0.9595053  0.9442954
##   28      1       0.9552185  0.9384904
##   28      2       0.9595053  0.9442954
##   29      1       0.9552185  0.9384904
##   29      2       0.9595053  0.9442954
##   30      1       0.9552185  0.9384904
##   30      2       0.9595053  0.9442954
## 
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were degree = 2 and nprune = 15.
```

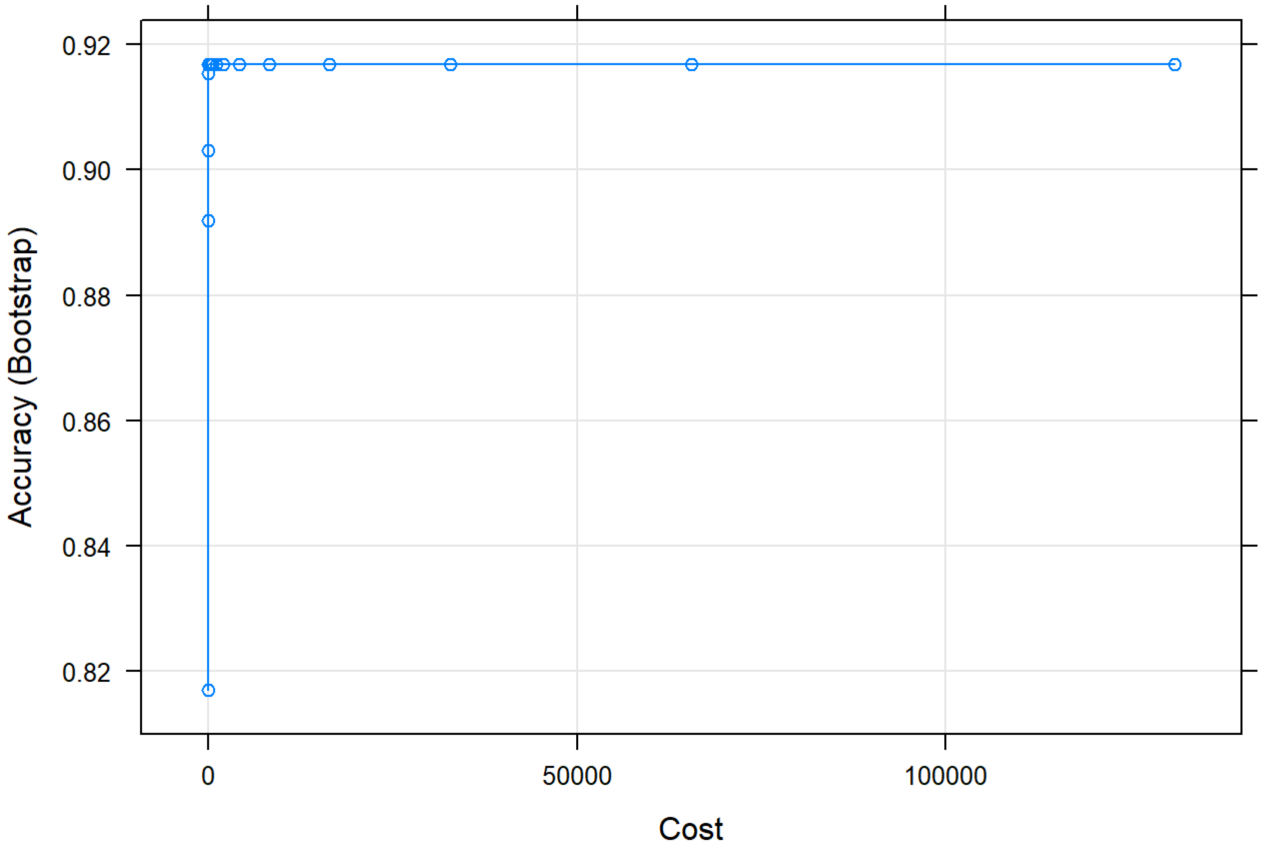
```
plot(FDATune)
```



```
################################ Support Vector Machines (SVM) ################################

set.seed(0)

library(kernlab)
SVMTune = train(x=train.FattyAcids, y=train.OilType, method="svmRadial", metric = "Accuracy", preProcess=c("center","scale"
), tuneLength=20)

SVMTune
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 79 samples
##  6 predictor
##  7 classes: 'A', 'B', 'C', 'D', 'E', 'F', 'G'
##
## Pre-processing: centered (6), scaled (6)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 79, 79, 79, 79, 79, 79, ...
## Resampling results across tuning parameters:
##
##   C          Accuracy   Kappa
##        0.25  0.8169850  0.7509460
##        0.50  0.8919323  0.8561479
##        1.00  0.9031009  0.8712838
##        2.00  0.9153426  0.8882733
##        4.00  0.9168811  0.8901886
##        8.00  0.9168811  0.8901886
##       16.00  0.9168811  0.8901886
##       32.00  0.9168811  0.8901886
##       64.00  0.9168811  0.8901886
##      128.00  0.9168811  0.8901886
##      256.00  0.9168811  0.8901886
##      512.00  0.9168811  0.8901886
##     1024.00  0.9168811  0.8901886
##     2048.00  0.9168811  0.8901886
##     4096.00  0.9168811  0.8901886
##     8192.00  0.9168811  0.8901886
##    16384.00  0.9168811  0.8901886
##    32768.00  0.9168811  0.8901886
##    65536.00  0.9168811  0.8901886
##   131072.00  0.9168811  0.8901886
##
## Tuning parameter 'sigma' was held constant at a value of 0.2143859
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.2143859 and C = 4.
```

```
plot(SVMTune)
```


Accuracy (Bootstrap)
0.92
0.90
0.88
0.86
0.84
0.82
0
50000
100000
Cost

```
### Predict the test set based on eight models
#RDA
pred.rda <- predict(RDATune,test.FattyAcids, type = "prob")[,1]
#MDA
pred.mda <- predict(MDATune,test.FattyAcids, type = "prob")[,1]
#NB
pred.nb <- predict(NBTune,test.FattyAcids, type = "prob")[,1]
#KNN
pred.knn <- predict(KNNTune,test.FattyAcids, type = "prob")[,1]
#NN
pred.nn <- predict(NNTune,test.FattyAcids, type = "prob")[,1]
#FDA
pred.fda <- predict(FDATune, test.FattyAcids, type = "prob")[,1]
#SVM
pred.svm <- predict(SVMTune, test.FattyAcids, type = "prob")[,1]





#########################Create the confusion matrix from the test set######################

#Confusion Matrix of RDA
confusionMatrix(data = predict(RDATune, test.FattyAcids), reference = test.OilType)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction A B C D E F G
##          A 5 0 0 0 0 0 0
##          B 2 5 0 0 0 0 0
##          C 0 0 0 0 0 0 0
##          D 0 0 0 1 0 0 0
##          E 0 0 0 0 2 0 0
##          F 0 0 0 0 0 2 0
##          G 0 0 0 0 0 0 0
## 
## Overall Statistics
##                                           
##                Accuracy : 0.8824          
##                  95% CI : (0.6356, 0.9854)
##     No Information Rate : 0.4118          
##     P-Value [Acc > NIR] : 8.516e-05       
##                                           
##                   Kappa : 0.8381          
##                                           
##  Mcnemar's Test P-Value : NA              
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity            0.7143   1.0000       NA  1.00000   1.0000   1.0000
## Specificity            1.0000   0.8333        1  1.00000   1.0000   1.0000
## Pos Pred Value         1.0000   0.7143       NA  1.00000   1.0000   1.0000
## Neg Pred Value         0.8333   1.0000       NA  1.00000   1.0000   1.0000
## Prevalence             0.4118   0.2941        0  0.05882   0.1176   0.1176
## Detection Rate         0.2941   0.2941        0  0.05882   0.1176   0.1176
## Detection Prevalence   0.2941   0.4118        0  0.05882   0.1176   0.1176
## Balanced Accuracy      0.8571   0.9167       NA  1.00000   1.0000   1.0000
##                      Class: G
## Sensitivity                NA
## Specificity                 1
## Pos Pred Value             NA
## Neg Pred Value             NA
## Prevalence                 0
## Detection Rate             0
## Detection Prevalence       0
## Balanced Accuracy         NA
```

```
#Confusion matrix of MDA
confusionMatrix(data = predict(MDATune, test.FattyAcids), reference = test.OilType)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction A B C D E F G
##          A 6 0 0 0 0 0 0
##          B 1 5 0 0 0 0 0
##          C 0 0 0 0 0 0 0
##          D 0 0 0 1 0 0 0
##          E 0 0 0 0 2 0 0
##          F 0 0 0 0 0 2 0
##          G 0 0 0 0 0 0 0
## 
## Overall Statistics
## 
##                Accuracy : 0.9412 
##                  95% CI : (0.7131, 0.9985)
##     No Information Rate : 0.4118 
##     P-Value [Acc > NIR] : 7.111e-06
## 
##                   Kappa : 0.9183 
## 
##  Mcnemar's Test P-Value : NA 
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity            0.8571   1.0000       NA  1.00000   1.0000   1.0000
## Specificity            1.0000   0.9167        1  1.00000   1.0000   1.0000
## Pos Pred Value         1.0000   0.8333       NA  1.00000   1.0000   1.0000
## Neg Pred Value         0.9091   1.0000       NA  1.00000   1.0000   1.0000
## Prevalence             0.4118   0.2941        0  0.05882   0.1176   0.1176
## Detection Rate         0.3529   0.2941        0  0.05882   0.1176   0.1176
## Detection Prevalence   0.3529   0.3529        0  0.05882   0.1176   0.1176
## Balanced Accuracy      0.9286   0.9583       NA  1.00000   1.0000   1.0000
##                      Class: G
## Sensitivity                NA
## Specificity                 1
## Pos Pred Value             NA
## Neg Pred Value             NA
## Prevalence                 0
## Detection Rate             0
## Detection Prevalence       0
## Balanced Accuracy         NA
```

```
#Confusion matrix of NB
confusionMatrix(data = predict(NBTune, test.FattyAcids), reference = test.OilType)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction A B C D E F G
##          A 7 0 0 0 0 0 0
##          B 0 3 0 0 0 0 0
##          C 0 1 0 0 0 0 0
##          D 0 0 0 1 0 0 0
##          E 0 0 0 0 2 0 0
##          F 0 0 0 0 0 2 0
##          G 0 1 0 0 0 0 0
## 
## Overall Statistics
##                                           
##                Accuracy : 0.8824          
##                  95% CI : (0.6356, 0.9854)
##     No Information Rate : 0.4118          
##     P-Value [Acc > NIR] : 8.516e-05       
##                                           
##                   Kappa : 0.8426          
##                                           
##  Mcnemar's Test P-Value : NA              
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity            1.0000   0.6000       NA  1.00000   1.0000   1.0000
## Specificity            1.0000   1.0000  0.94118  1.00000   1.0000   1.0000
## Pos Pred Value         1.0000   1.0000       NA  1.00000   1.0000   1.0000
## Neg Pred Value         1.0000   0.8571       NA  1.00000   1.0000   1.0000
## Prevalence             0.4118   0.2941  0.00000  0.05882   0.1176   0.1176
## Detection Rate         0.4118   0.1765  0.00000  0.05882   0.1176   0.1176
## Detection Prevalence   0.4118   0.1765  0.05882  0.05882   0.1176   0.1176
## Balanced Accuracy      1.0000   0.8000       NA  1.00000   1.0000   1.0000
##                      Class: G
## Sensitivity                NA
## Specificity           0.94118
## Pos Pred Value             NA
## Neg Pred Value             NA
## Prevalence            0.00000
## Detection Rate        0.00000
## Detection Prevalence  0.05882
## Balanced Accuracy          NA
```

```
#Confusion matrix of KNN
confusionMatrix(data = predict(KNNTune, test.FattyAcids), reference = test.OilType)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction A B C D E F G
##          A 5 0 0 0 0 0 0
##          B 2 5 0 0 0 0 0
##          C 0 0 0 0 0 0 0
##          D 0 0 0 1 0 0 0
##          E 0 0 0 0 2 0 0
##          F 0 0 0 0 0 2 0
##          G 0 0 0 0 0 0 0
## 
## Overall Statistics
## 
##                Accuracy : 0.8824          
##                  95% CI : (0.6356, 0.9854)
##     No Information Rate : 0.4118          
##     P-Value [Acc > NIR] : 8.516e-05       
## 
##                   Kappa : 0.8381          
## 
##  Mcnemar's Test P-Value : NA              
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity            0.7143   1.0000       NA  1.00000   1.0000   1.0000
## Specificity            1.0000   0.8333        1  1.00000   1.0000   1.0000
## Pos Pred Value         1.0000   0.7143       NA  1.00000   1.0000   1.0000
## Neg Pred Value         0.8333   1.0000       NA  1.00000   1.0000   1.0000
## Prevalence             0.4118   0.2941        0  0.05882   0.1176   0.1176
## Detection Rate         0.2941   0.2941        0  0.05882   0.1176   0.1176
## Detection Prevalence   0.2941   0.4118        0  0.05882   0.1176   0.1176
## Balanced Accuracy      0.8571   0.9167       NA  1.00000   1.0000   1.0000
##                      Class: G
## Sensitivity                NA
## Specificity                 1
## Pos Pred Value             NA
## Neg Pred Value             NA
## Prevalence                  0
## Detection Rate              0
## Detection Prevalence        0
## Balanced Accuracy          NA
```

```
#Confusion matrix of NN
confusionMatrix(data = predict(NNTune, test.FattyAcids), reference = test.OilType)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction A B C D E F G
##          A 5 0 0 0 0 0 0
##          B 2 5 0 0 0 0 0
##          C 0 0 0 0 0 0 0
##          D 0 0 0 1 0 0 0
##          E 0 0 0 0 2 0 0
##          F 0 0 0 0 0 2 0
##          G 0 0 0 0 0 0 0
## 
## Overall Statistics
## 
##                Accuracy : 0.8824          
##                  95% CI : (0.6356, 0.9854)
##     No Information Rate : 0.4118          
##     P-Value [Acc > NIR] : 8.516e-05       
##                                           
##                   Kappa : 0.8381          
##                                           
##  Mcnemar's Test P-Value : NA              
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity            0.7143   1.0000       NA  1.00000   1.0000   1.0000
## Specificity            1.0000   0.8333        1  1.00000   1.0000   1.0000
## Pos Pred Value         1.0000   0.7143       NA  1.00000   1.0000   1.0000
## Neg Pred Value         0.8333   1.0000       NA  1.00000   1.0000   1.0000
## Prevalence             0.4118   0.2941        0  0.05882   0.1176   0.1176
## Detection Rate         0.2941   0.2941        0  0.05882   0.1176   0.1176
## Detection Prevalence   0.2941   0.4118        0  0.05882   0.1176   0.1176
## Balanced Accuracy      0.8571   0.9167       NA  1.00000   1.0000   1.0000
##                      Class: G
## Sensitivity                NA
## Specificity                 1
## Pos Pred Value             NA
## Neg Pred Value             NA
## Prevalence                  0
## Detection Rate              0
## Detection Prevalence        0
## Balanced Accuracy          NA
```

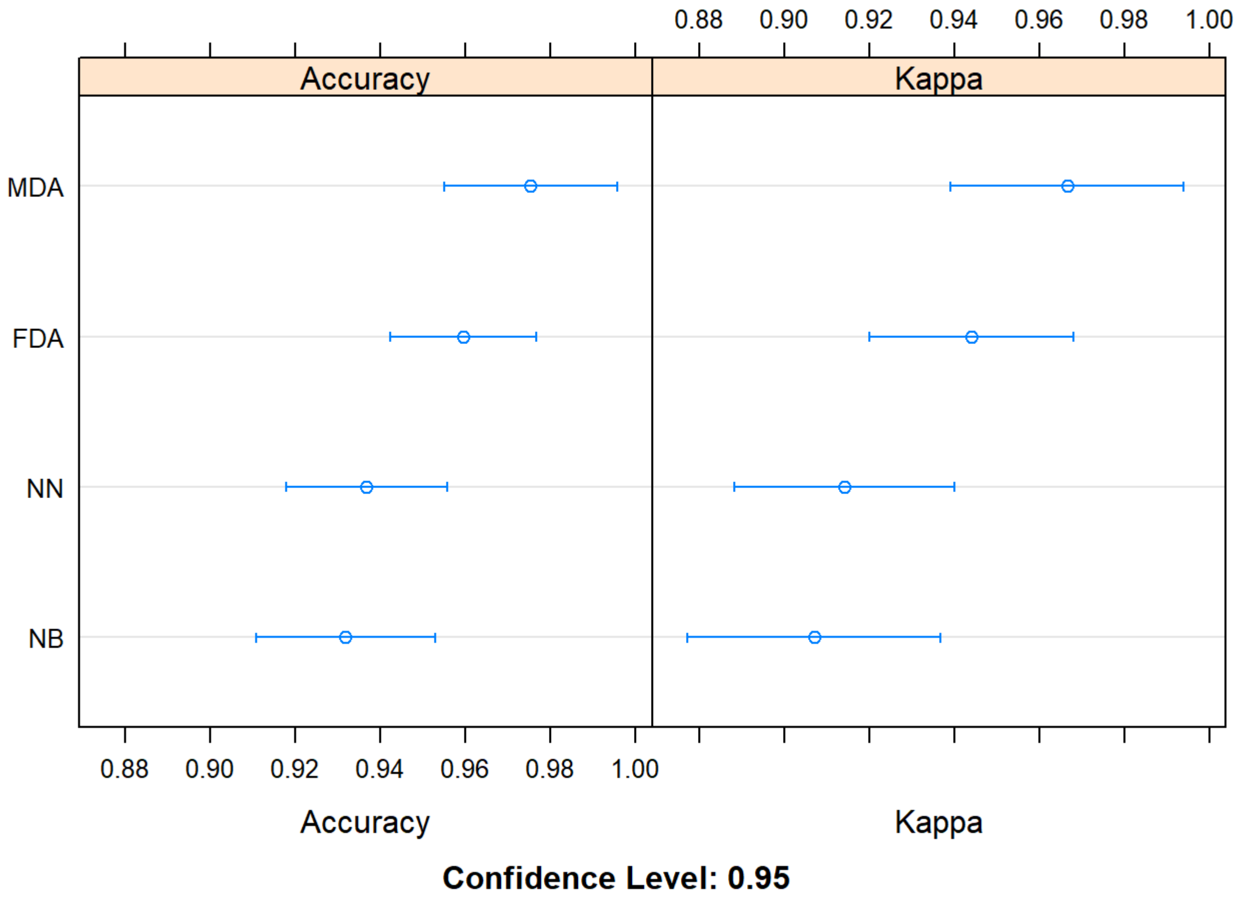
```
#Confusion matrix of FDA
confusionMatrix(data = predict(FDATune, test.FattyAcids), reference = test.OilType)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction A B C D E F G
##          A 5 0 0 0 0 0 0
##          B 2 5 0 0 0 0 0
##          C 0 0 0 0 0 0 0
##          D 0 0 0 1 0 0 0
##          E 0 0 0 0 2 0 0
##          F 0 0 0 0 0 2 0
##          G 0 0 0 0 0 0 0
## 
## Overall Statistics
## 
##                Accuracy : 0.8824          
##                  95% CI : (0.6356, 0.9854)
##     No Information Rate : 0.4118          
##     P-Value [Acc > NIR] : 8.516e-05       
## 
##                   Kappa : 0.8381          
## 
##  Mcnemar's Test P-Value : NA              
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity            0.7143   1.0000       NA  1.00000   1.0000   1.0000
## Specificity            1.0000   0.8333        1  1.00000   1.0000   1.0000
## Pos Pred Value         1.0000   0.7143       NA  1.00000   1.0000   1.0000
## Neg Pred Value         0.8333   1.0000       NA  1.00000   1.0000   1.0000
## Prevalence             0.4118   0.2941        0  0.05882   0.1176   0.1176
## Detection Rate         0.2941   0.2941        0  0.05882   0.1176   0.1176
## Detection Prevalence   0.2941   0.4118        0  0.05882   0.1176   0.1176
## Balanced Accuracy      0.8571   0.9167       NA  1.00000   1.0000   1.0000
##                      Class: G
## Sensitivity                NA
## Specificity                 1
## Pos Pred Value             NA
## Neg Pred Value             NA
## Prevalence                  0
## Detection Rate              0
## Detection Prevalence        0
## Balanced Accuracy          NA
```

```
#Confusion matrix of SVM
confusionMatrix(data = predict(SVMTune, test.FattyAcids), reference = test.OilType)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction A B C D E F G
##          A 5 0 0 0 0 0 0
##          B 2 5 0 0 0 0 0
##          C 0 0 0 0 0 0 0
##          D 0 0 0 1 0 0 0
##          E 0 0 0 0 2 0 0
##          F 0 0 0 0 0 2 0
##          G 0 0 0 0 0 0 0
## 
## Overall Statistics
## 
##                Accuracy : 0.8824          
##                  95% CI : (0.6356, 0.9854)
##     No Information Rate : 0.4118          
##     P-Value [Acc > NIR] : 8.516e-05       
## 
##                   Kappa : 0.8381          
## 
##  Mcnemar's Test P-Value : NA              
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity            0.7143   1.0000       NA  1.00000   1.0000   1.0000
## Specificity            1.0000   0.8333        1  1.00000   1.0000   1.0000
## Pos Pred Value         1.0000   0.7143       NA  1.00000   1.0000   1.0000
## Neg Pred Value         0.8333   1.0000       NA  1.00000   1.0000   1.0000
## Prevalence             0.4118   0.2941        0  0.05882   0.1176   0.1176
## Detection Rate         0.2941   0.2941        0  0.05882   0.1176   0.1176
## Detection Prevalence   0.2941   0.4118        0  0.05882   0.1176   0.1176
## Balanced Accuracy      0.8571   0.9167       NA  1.00000   1.0000   1.0000
##                      Class: G
## Sensitivity                NA
## Specificity                 1
## Pos Pred Value             NA
## Neg Pred Value             NA
## Prevalence                0
## Detection Rate            0
## Detection Prevalence      0
## Balanced Accuracy        NA
```

```
#Resamples of Training data

# Combining the models from Question 13
res13 = resamples(list(MDA=MDATune,NB=NBTune,NN=NNTune,FDA=FDATune))
dotplot(res13)
```



Q 13.2 b

b. Which oil type does the optimal model most accurately predict? Least accurately predict?

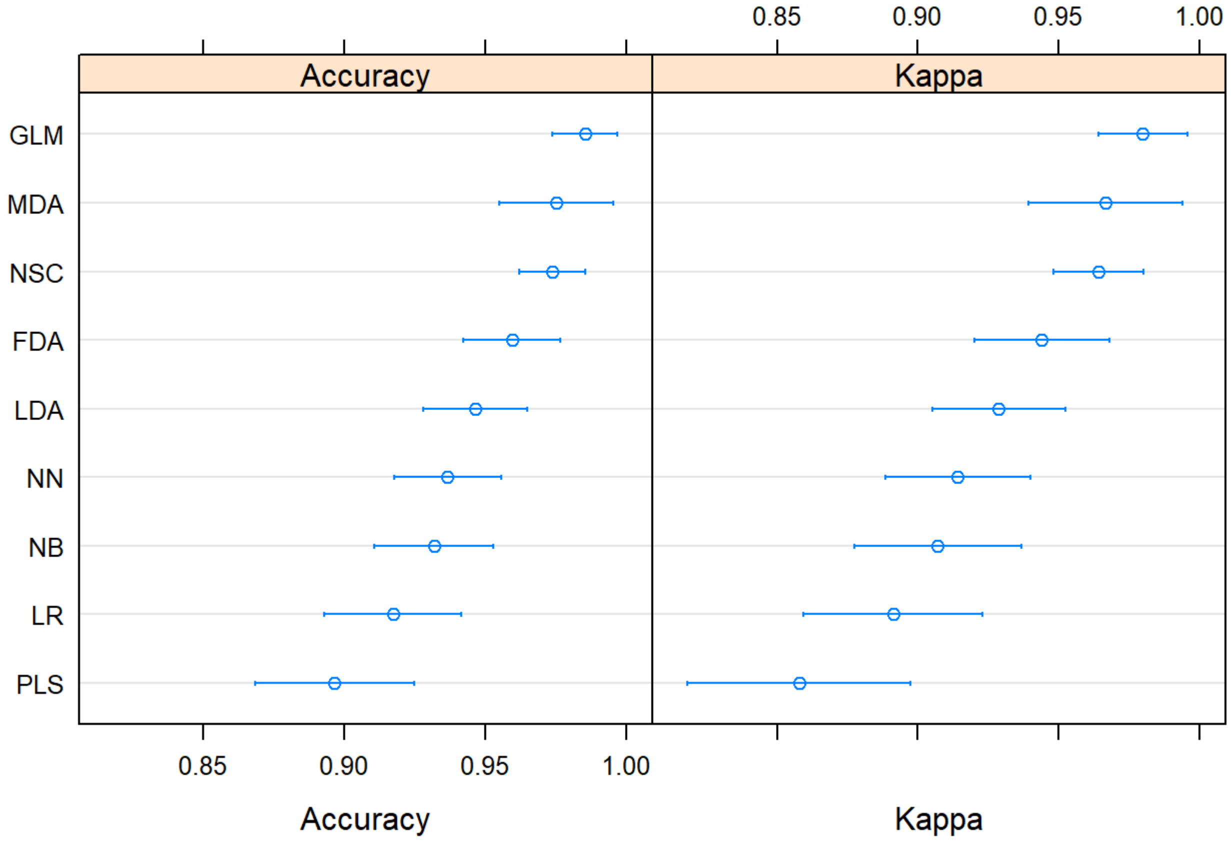
Answer (13.2 b)

Based on the output of the resamples function on the training data and using the classification statistic **“Accuracy”**, and also matching with the Confusion Matrix data for test data, we can say that **MDA (Mixed Discriminant Analysis)** is the best model as far as accuracy of prediction is concerned.

And from the data, we can also see that **NB (Naive Bayes)** is the model with the least accurately prediction.

(Overall)

```
# Combining the models from Question 12 & Question 13
res1213 = resamples(list(MDA=MDATune,NB=NBTune,NN=NNTune,FDA=FDATune,LR=lr.FattyAcids,LDA=lda.FattyAcids,PLS=pls.FattyAcids,
GLM=glmn.Tuned.LR.FattyAcids,NSC=nsc.Tuned.FattyAcids))
dotplot(res1213)
```



Confidence Level: 0.95

If we combine the outputs of Question 12 and Question 13, based on the output of the resamples function on the training data and using the classification statistic **“Accuracy”**, and also matching with the Confusion Matrix data for test data, we can say that **GLM (Penalized Model for Logistic Regression)** is the best model as far as accuracy of prediction is concerned.

Again if we combine the outputs of Question 12 and Question 13,from the data, we see that **PLS (Partial Least Squares Discriminant Analysis)** is the model with the least accurate prediction.