hw3

chris

2023-08-07

```r
library(tidyverse)
library(AppliedPredictiveModeling)
library(caret)

data(oil)
```

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.

(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?

Yes, imbalanced data should be handled using approaches such as cross-validation or resampling techniques such as stratified random sampling, up sampling, or down sampling.

```r
set.seed(123)
fattyAcids$oilType <- oilType
strat <- createDataPartition(fattyAcids$oilType,
 p = .59,
 list= FALSE,
 times=1)

strat <- as.data.frame(strat)

train <- fattyAcids[strat$Resample, ]
test <- fattyAcids[-strat$Resample,]
```

Linear discriminant analysis

```r
set.seed(123)

lda <- train(x = train[, 1:7], y = train[, 8],
             method = "lda",
             preProc = c('center','scale'),
             metric = "Accuracy",
             trControl = trainControl(summaryFunction = defaultSummary,
                                      method = "cv",
                                      classProbs = T,
                                      savePredictions = T))
lda
```

```
## Linear Discriminant Analysis
##
## 60 samples
##  7 predictor
##  7 classes: 'A', 'B', 'C', 'D', 'E', 'F', 'G'
##
## Pre-processing: centered (7), scaled (7)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 52, 53, 54, 55, 52, 55, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.98      0.9666667
```

```
confusionMatrix(data = predict(lda, test[,1:7]), reference = test[,8])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  A  B  C  D  E  F  G
##          A 13  0  0  0  0  0  0
##          B  2 10  0  0  0  0  0
##          C  0  0  1  0  0  0  0
##          D  0  0  0  2  0  0  0
##          E  0  0  0  0  4  0  0
##          F  0  0  0  0  0  4  0
##          G  0  0  0  0  0  0  0
##
## Overall Statistics
##
##                Accuracy : 0.9444
##                  95% CI : (0.8134, 0.9932)
##     No Information Rate : 0.4167
##     P-Value [Acc > NIR] : 2.641e-11
##
##                   Kappa : 0.9237
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity            0.8667   1.0000  1.00000  1.00000   1.0000   1.0000
## Specificity            1.0000   0.9231  1.00000  1.00000   1.0000   1.0000
## Pos Pred Value         1.0000   0.8333  1.00000  1.00000   1.0000   1.0000
## Neg Pred Value         0.9130   1.0000  1.00000  1.00000   1.0000   1.0000
## Prevalence             0.4167   0.2778  0.02778  0.05556   0.1111   0.1111
## Detection Rate         0.3611   0.2778  0.02778  0.05556   0.1111   0.1111
## Detection Prevalence   0.3611   0.3333  0.02778  0.05556   0.1111   0.1111
## Balanced Accuracy      0.9333   0.9615  1.00000  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
```

Penalized multinomial logistic regression

```
set.seed(123)

log <- train(x = train[, 1:7], y = train[, 8],
             method = "multinom", preProc = c('center','scale'),
             metric = "Accuracy",
             trControl = trainControl(summaryFunction = defaultSummary,
                                      method = "cv",
                                      classProbs = T,
                                      savePredictions = T))
```

```
## # weights:  63 (48 variable)
## initial  value 101.187328
## iter  10 value 0.599206
## iter  20 value 0.037484
## iter  30 value 0.004450
## final  value 0.000059
## converged
## # weights:  63 (48 variable)
## initial  value 101.187328
## iter  10 value 12.224249
## iter  20 value 11.473406
## final  value 11.473403
## converged
## # weights:  63 (48 variable)
## initial  value 101.187328
## iter  10 value 0.629226
## iter  20 value 0.122900
## iter  30 value 0.101462
## iter  40 value 0.097294
## iter  50 value 0.095601
## iter  60 value 0.094506
## iter  70 value 0.094096
## iter  80 value 0.093682
## iter  90 value 0.093508
## iter 100 value 0.093462
## final  value 0.093462
## stopped after 100 iterations
## # weights:  63 (48 variable)
## initial  value 103.133238
## iter  10 value 0.690348
## iter  20 value 0.040551
## iter  30 value 0.005888
## iter  40 value 0.000548
## final  value 0.000086
## converged
## # weights:  63 (48 variable)
## initial  value 103.133238
## iter  10 value 12.249516
## iter  20 value 11.714671
## final  value 11.714668
## converged
## # weights:  63 (48 variable)
## initial  value 103.133238
## iter  10 value 0.719659
## iter  20 value 0.122539
## iter  30 value 0.099880
## iter  40 value 0.097096
## iter  50 value 0.094686
## iter  60 value 0.093683
## iter  70 value 0.093040
## iter  80 value 0.092855
## iter  90 value 0.092762
## iter 100 value 0.092715
## final  value 0.092715
## stopped after 100 iterations
## # weights:  63 (48 variable)
## initial  value 105.079148
## iter  10 value 0.402883
## iter  20 value 0.033452
## iter  30 value 0.003810
## iter  40 value 0.000834
## final  value 0.000090
## converged
## # weights:  63 (48 variable)
## initial  value 105.079148
## iter  10 value 12.389824
## iter  20 value 11.619711
## final  value 11.619704
## converged
## # weights:  63 (48 variable)
## initial  value 105.079148
## iter  10 value 0.438452
## iter  20 value 0.111779
## iter  30 value 0.100864
## iter  40 value 0.098092
## iter  50 value 0.096286
## iter  60 value 0.095455
## iter  70 value 0.094987
## iter  80 value 0.094702
## iter  90 value 0.094584
## iter 100 value 0.094499
## final  value 0.094499
## stopped after 100 iterations
## # weights:  63 (48 variable)
## initial  value 107.025058
## iter  10 value 0.619831
## iter  20 value 0.062320
## iter  30 value 0.006854
## iter  40 value 0.000946
## final  value 0.000058
## converged
## # weights:  63 (48 variable)
## initial  value 107.025058
## iter  10 value 13.243814
## iter  20 value 11.969719
## final  value 11.969648
## converged
## # weights:  63 (48 variable)
## initial  value 107.025058
## iter  10 value 0.652780
## iter  20 value 0.138696
## iter  30 value 0.112750
## iter  40 value 0.105837
## iter  50 value 0.103962
## iter  60 value 0.101638
## iter  70 value 0.100994
## iter  80 value 0.100672
## iter  90 value 0.100515
## iter 100 value 0.100410
## final  value 0.100410
## stopped after 100 iterations
## # weights:  63 (48 variable)
## initial  value 101.187328
## iter  10 value 0.621637
## iter  20 value 0.042523
## iter  30 value 0.007536
## iter  40 value 0.000448
## final  value 0.000070
## converged
## # weights:  63 (48 variable)
## initial  value 101.187328
## iter  10 value 11.796405
## iter  20 value 11.530351
## final  value 11.530349
## converged
## # weights:  63 (48 variable)
## initial  value 101.187328
## iter  10 value 0.652629
## iter  20 value 0.126397
## iter  30 value 0.103024
## iter  40 value 0.098913
## iter  50 value 0.097044
## iter  60 value 0.095737
## iter  70 value 0.095120
## iter  80 value 0.094772
## iter  90 value 0.094636
## iter 100 value 0.094546
## final  value 0.094546
## stopped after 100 iterations
## # weights:  63 (48 variable)
## initial  value 107.025058
## iter  10 value 0.720921
## iter  20 value 0.046004
## iter  30 value 0.002619
## iter  40 value 0.000651
## final  value 0.000070
## converged
## # weights:  63 (48 variable)
## initial  value 107.025058
## iter  10 value 13.210902
## iter  20 value 12.059714
## final  value 12.059709
## converged
## # weights:  63 (48 variable)
## initial  value 107.025058
## iter  10 value 0.752143
## iter  20 value 0.135221
## iter  30 value 0.109440
## iter  40 value 0.104074
## iter  50 value 0.102265
## iter  60 value 0.100403
## iter  70 value 0.099922
## iter  80 value 0.099536
## iter  90 value 0.099402
## iter 100 value 0.099342
## final  value 0.099342
## stopped after 100 iterations
## # weights:  63 (48 variable)
## initial  value 107.025058
## iter  10 value 0.708031
## iter  20 value 0.069137
## iter  30 value 0.007287
## iter  40 value 0.000876
## iter  50 value 0.000279
## iter  60 value 0.000201
## final  value 0.000092
## converged
## # weights:  63 (48 variable)
## initial  value 107.025058
## iter  10 value 12.709263
## iter  20 value 11.497577
## final  value 11.497223
## converged
## # weights:  63 (48 variable)
## initial  value 107.025058
## iter  10 value 0.738300
## iter  20 value 0.144565
## iter  30 value 0.105543
## iter  40 value 0.101126
## iter  50 value 0.098854
## iter  60 value 0.096765
## iter  70 value 0.096060
## iter  80 value 0.095586
## iter  90 value 0.095426
## iter 100 value 0.095265
## final  value 0.095265
## stopped after 100 iterations
## # weights:  63 (48 variable)
## initial  value 107.025058
## iter  10 value 0.271695
## iter  20 value 0.031816
## iter  30 value 0.007513
## iter  40 value 0.002450
## iter  50 value 0.000302
## final  value 0.000082
## converged
## # weights:  63 (48 variable)
## initial  value 107.025058
## iter  10 value 11.827684
## iter  20 value 10.954272
## final  value 10.954268
## converged
## # weights:  63 (48 variable)
## initial  value 107.025058
## iter  10 value 0.303030
## iter  20 value 0.086578
## iter  30 value 0.074040
## iter  40 value 0.069612
## iter  50 value 0.069018
## iter  60 value 0.068583
## iter  70 value 0.068365
## iter  80 value 0.068283
## iter  90 value 0.068242
## iter 100 value 0.068224
## final  value 0.068224
## stopped after 100 iterations
## # weights:  63 (48 variable)
## initial  value 107.025058
## iter  10 value 0.178304
## iter  20 value 0.021593
## iter  30 value 0.010127
## iter  40 value 0.005794
## iter  50 value 0.002059
## iter  60 value 0.000448
## final  value 0.000075
## converged
## # weights:  63 (48 variable)
## initial  value 107.025058
## iter  10 value 13.043957
## iter  20 value 12.083796
## final  value 12.083783
## converged
## # weights:  63 (48 variable)
## initial  value 107.025058
## iter  10 value 0.354878
## iter  20 value 0.201882
## iter  30 value 0.182415
## iter  40 value 0.154643
## iter  50 value 0.113968
## iter  60 value 0.106875
## iter  70 value 0.102535
## iter  80 value 0.100332
## iter  90 value 0.099402
## iter 100 value 0.098828
## final  value 0.098828
## stopped after 100 iterations
## # weights:  63 (48 variable)
## initial  value 105.079148
## iter  10 value 0.524495
## iter  20 value 0.041461
## iter  30 value 0.002206
## iter  40 value 0.000473
## final  value 0.000054
## converged
## # weights:  63 (48 variable)
## initial  value 105.079148
## iter  10 value 12.445010
## iter  20 value 11.802055
## final  value 11.802021
## converged
## # weights:  63 (48 variable)
## initial  value 105.079148
## iter  10 value 0.557829
## iter  20 value 0.125999
## iter  30 value 0.106508
## iter  40 value 0.099708
## iter  50 value 0.097845
## iter  60 value 0.096411
## iter  70 value 0.095651
## iter  80 value 0.095296
## iter  90 value 0.095204
## iter 100 value 0.095098
## final  value 0.095098
## stopped after 100 iterations
## # weights:  63 (48 variable)
## initial  value 116.754609
## iter  10 value 0.560776
## iter  20 value 0.063385
## iter  30 value 0.007299
## iter  40 value 0.004347
## final  value 0.000077
## converged
```

```
log
```

```
## Penalized Multinomial Regression
##
## 60 samples
##  7 predictor
##  7 classes: 'A', 'B', 'C', 'D', 'E', 'F', 'G'
##
## Pre-processing: centered (7), scaled (7)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 52, 53, 54, 55, 52, 55, ...
## Resampling results across tuning parameters:
```

```
##
##   decay  Accuracy   Kappa
##   0e+00  0.9708333  0.9623932
##   1e-04  0.9433333  0.9150327
##   1e-01  0.9433333  0.9181287
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was decay = 0.
```

```
confusionMatrix(data = predict(log, test[,1:7]), reference = test[,8])
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   A  B  C  D  E  F  G
##          A 13  0  0  0  0  0  0
##          B  2 10  0  0  0  0  0
##          C  0  0  1  0  0  0  0
##          D  0  0  0  2  0  0  0
##          E  0  0  0  0  4  0  0
##          F  0  0  0  0  0  4  0
##          G  0  0  0  0  0  0  0
##
## Overall Statistics
##
##                Accuracy : 0.9444
##                  95% CI : (0.8134, 0.9932)
##     No Information Rate : 0.4167
##     P-Value [Acc > NIR] : 2.641e-11
##
##                   Kappa : 0.9237
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity            0.8667   1.0000  1.00000  1.00000   1.0000   1.0000
## Specificity            1.0000   0.9231  1.00000  1.00000   1.0000   1.0000
## Pos Pred Value         1.0000   0.8333  1.00000  1.00000   1.0000   1.0000
## Neg Pred Value         0.9130   1.0000  1.00000  1.00000   1.0000   1.0000
## Prevalence             0.4167   0.2778  0.02778  0.05556   0.1111   0.1111
## Detection Rate         0.3611   0.2778  0.02778  0.05556   0.1111   0.1111
## Detection Prevalence   0.3611   0.3333  0.02778  0.05556   0.1111   0.1111
## Balanced Accuracy      0.9333   0.9615  1.00000  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
```

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

Accuracy and Kappa since they were similar in training and testing data

(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?

Comparing LDA to penalized multinomial logistic regression, both methods perform equally well on the data.

12.3. The web site17 for the MLC++ software package contains a number of machine learning data sets. The "churn" data set was developed to predict telecom customer churn based on information about their account. The data files state that the data are "artificial based on claims similar to real world." The data consist of 19 predictors related to the customer account, such as the number of customer service calls, the area code, and the number of minutes. The outcome is whether the customer churned

The data are contained in the C50 package and can be loaded using:

```
# library(C50)
# data(churn)
# ## Two objects are loaded: churnTrain and churnTest
# str(churnTrain)
# table(churnTrain$Class)
#
# Error in str(churnTrain) : object 'churnTrain' not found
```

Data won't load

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

(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?

Nonlinear discriminant analysis

```
set.seed(123)
nda <- train(train[,1:7], train[,8],
 method = "mda",
 metric = "Accuracy",
 preProc = c('center','scale'),
 tuneGrid = expand.grid(.subclasses = 1:8),
 trControl = trainControl(summaryFunction = defaultSummary,
                                       method = "cv",
                                       classProbs = T,
                                       savePredictions = T))
nda
```

```
## Mixture Discriminant Analysis
##
## 60 samples
##  7 predictor
##  7 classes: 'A', 'B', 'C', 'D', 'E', 'F', 'G'
##
## Pre-processing: centered (7), scaled (7)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 52, 53, 54, 55, 52, 55, ...
## Resampling results across tuning parameters:
##
##   subclasses  Accuracy   Kappa
##   1           0.9800000  0.9666667
##   2           0.9600000  0.9372549
##   3           0.9600000  0.9372549
##   4           0.9433333  0.9150327
##   5           0.9600000  0.9372549
##   6           0.9600000  0.9372549
##   7           0.9600000  0.9372549
##   8                 NaN        NaN
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was subclasses = 1.
```

```
confusionMatrix(data = predict(nda, test[,1:7]), reference = test[,8])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  A  B  C  D  E  F  G
##          A 13  0  0  0  0  0  0
##          B  2 10  0  0  0  0  0
##          C  0  0  1  0  0  0  0
##          D  0  0  0  2  0  0  0
##          E  0  0  0  0  4  0  0
##          F  0  0  0  0  0  4  0
##          G  0  0  0  0  0  0  0
##
## Overall Statistics
##
##                Accuracy : 0.9444
##                  95% CI : (0.8134, 0.9932)
##     No Information Rate : 0.4167
##     P-Value [Acc > NIR] : 2.641e-11
##
##                   Kappa : 0.9237
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity            0.8667   1.0000  1.00000  1.00000   1.0000   1.0000
## Specificity            1.0000   0.9231  1.00000  1.00000   1.0000   1.0000
## Pos Pred Value         1.0000   0.8333  1.00000  1.00000   1.0000   1.0000
## Neg Pred Value         0.9130   1.0000  1.00000  1.00000   1.0000   1.0000
## Prevalence             0.4167   0.2778  0.02778  0.05556   0.1111   0.1111
## Detection Rate         0.3611   0.2778  0.02778  0.05556   0.1111   0.1111
## Detection Prevalence   0.3611   0.3333  0.02778  0.05556   0.1111   0.1111
## Balanced Accuracy      0.9333   0.9615  1.00000  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
```

NDA: Accuracy 0.944, Kappa = 0.924

```
nnetGrid <- expand.grid(.size = 1:10, .decay = c(0, .1, 1, 2))
maxSize <- max(nnetGrid$.size)
numWts <- 1*(maxSize * (length(train[,1:7]) + 1) + maxSize + 1)

set.seed(123)
nnet <- train(train[,1:7], train[,8],
 method = "nnet",
 preProc = c('center','scale'),
 metric = "Accuracy",
 tuneGrid = nnetGrid,
 trace = F,
 maxit = 100,
 MaxNWts = numWts,
 trControl = trainControl(summaryFunction = defaultSummary,
                                     method = "cv",
                                     classProbs = T,
                                     savePredictions = T))
nnet
```

```
## Neural Network
##
## 60 samples
##  7 predictor
##  7 classes: 'A', 'B', 'C', 'D', 'E', 'F', 'G'
##
## Pre-processing: centered (7), scaled (7)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 52, 53, 54, 55, 52, 55, ...
## Resampling results across tuning parameters:
##
##   size  decay  Accuracy   Kappa
##    1    0.0    0.6571429  0.50535563
##    1    0.1    0.6063095  0.41904680
##    1    1.0    0.4194048  0.09604432
##    1    2.0    0.3777381  0.00000000
##    2    0.0    0.8364286  0.77036180
##    2    0.1    0.8423810  0.78255254
##    2    1.0    0.6463095  0.48480822
##    2    2.0    0.6120238  0.42988175
##    3    0.0    0.8582143  0.80395816
##    3    0.1    0.9383333  0.91891498
##    3    1.0    0.6629762  0.51344070
##    3    2.0    0.6263095  0.45539645
##    4    0.0    0.9100000  0.86887883
##    4    0.1    0.9383333  0.91189744
##    4    1.0    0.7789286  0.68909116
##    4    2.0    0.6463095  0.48807619
##    5    0.0    0.9383333  0.91839043
##    5    0.1    0.9800000  0.96666667
##    5    1.0    0.8355952  0.76996491
##    5    2.0    0.6263095  0.45539645
##    6    0.0          NaN         NaN
##    6    0.1          NaN         NaN
##    6    1.0          NaN         NaN
##    6    2.0          NaN         NaN
##    7    0.0          NaN         NaN
##    7    0.1          NaN         NaN
##    7    1.0          NaN         NaN
##    7    2.0          NaN         NaN
##    8    0.0          NaN         NaN
##    8    0.1          NaN         NaN
##    8    1.0          NaN         NaN
##    8    2.0          NaN         NaN
##    9    0.0          NaN         NaN
##    9    0.1          NaN         NaN
##    9    1.0          NaN         NaN
##    9    2.0          NaN         NaN
##   10    0.0          NaN         NaN
##   10    0.1          NaN         NaN
##   10    1.0          NaN         NaN
##   10    2.0          NaN         NaN
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were size = 5 and decay = 0.1.
```

```
confusionMatrix(data = predict(nnet, test[,1:7]), reference = test[,8])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  A  B  C  D  E  F  G
##          A 13  0  0  0  0  0  0
##          B  2 10  0  0  0  0  0
##          C  0  0  1  0  0  0  0
##          D  0  0  0  2  0  0  0
##          E  0  0  0  0  4  0  0
##          F  0  0  0  0  0  4  0
##          G  0  0  0  0  0  0  0
##
## Overall Statistics
##
##                Accuracy : 0.9444
##                  95% CI : (0.8134, 0.9932)
##     No Information Rate : 0.4167
##     P-Value [Acc > NIR] : 2.641e-11
##
##                   Kappa : 0.9237
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity            0.8667   1.0000  1.00000  1.00000   1.0000   1.0000
## Specificity            1.0000   0.9231  1.00000  1.00000   1.0000   1.0000
## Pos Pred Value         1.0000   0.8333  1.00000  1.00000   1.0000   1.0000
## Neg Pred Value         0.9130   1.0000  1.00000  1.00000   1.0000   1.0000
## Prevalence             0.4167   0.2778  0.02778  0.05556   0.1111   0.1111
## Detection Rate         0.3611   0.2778  0.02778  0.05556   0.1111   0.1111
## Detection Prevalence   0.3611   0.3333  0.02778  0.05556   0.1111   0.1111
## Balanced Accuracy      0.9333   0.9615  1.00000  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
```

NNet equal to NDA

```
set.seed(123)



knn <- train(train[,1:7], train[,8],
 method = "knn",
 metric = "Accuracy",
 preProc = c('center','scale'),
 tuneGrid = data.frame(.k = c(4*(0:5)+1, 20*(1:5)+1, 50*(2:9)+1)),
 trControl = trainControl(summaryFunction = defaultSummary,
                                     method = "cv",
                                     classProbs = T,
                                     savePredictions = T))
knn
```

```
## k-Nearest Neighbors
##
## 60 samples
##  7 predictor
##  7 classes: 'A', 'B', 'C', 'D', 'E', 'F', 'G'
##
## Pre-processing: centered (7), scaled (7)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 52, 53, 54, 55, 52, 55, ...
## Resampling results across tuning parameters:
##
##   k    Accuracy   Kappa
##     1  0.9433333  0.91503268
##     5  0.8983333  0.85852821
##     9  0.9183333  0.88827245
##    13  0.7664286  0.67637671
##    17  0.6972619  0.56251287
##    21  0.6463095  0.49752418
##    41  0.3777381  0.01893902
##    61  0.3777381  0.00000000
##    81  0.3777381  0.00000000
##   101  0.3777381  0.00000000
##   151  0.3777381  0.00000000
##   201  0.3777381  0.00000000
##   251  0.3777381  0.00000000
##   301  0.3777381  0.00000000
##   351  0.3777381  0.00000000
##   401  0.3777381  0.00000000
##   451  0.3777381  0.00000000
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 1.
```

```
confusionMatrix(data = predict(knn, test[,1:7]), reference = test[,8])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  A  B  C  D  E  F  G
##          A 13  0  0  0  0  0  0
##          B  2 10  0  0  0  0  0
##          C  0  0  1  0  0  0  0
##          D  0  0  0  2  0  0  0
##          E  0  0  0  0  4  0  0
##          F  0  0  0  0  0  4  0
##          G  0  0  0  0  0  0  0
##
## Overall Statistics
##
##                Accuracy : 0.9444
##                  95% CI : (0.8134, 0.9932)
##     No Information Rate : 0.4167
##     P-Value [Acc > NIR] : 2.641e-11
##
##                   Kappa : 0.9237
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity            0.8667   1.0000  1.00000  1.00000   1.0000   1.0000
## Specificity            1.0000   0.9231  1.00000  1.00000   1.0000   1.0000
## Pos Pred Value         1.0000   0.8333  1.00000  1.00000   1.0000   1.0000
## Neg Pred Value         0.9130   1.0000  1.00000  1.00000   1.0000   1.0000
## Prevalence             0.4167   0.2778  0.02778  0.05556   0.1111   0.1111
## Detection Rate         0.3611   0.2778  0.02778  0.05556   0.1111   0.1111
## Detection Prevalence   0.3611   0.3333  0.02778  0.05556   0.1111   0.1111
## Balanced Accuracy      0.9333   0.9615  1.00000  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
```

Accuracy = 0.9444 Kappa = 0.924

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

Nonlinear discriminant analysis, Neural networks, and K-Nearest Neighbors have all equal performance in predicting oil. Perhaps I messed up somewhere