Week5, Assignment 1

Gabi Rivera

```
library(caret)
library(tidyverse)
library(gt)
library(AppliedPredictiveModeling)
library(corrplot)
library(ROSE)
library(pROC)
library(magrittr)

rseed <- 503
```

Problem 5.1 (60 points)

The hepatic injury data set was described in the introductory chapter and contains 281 unique compounds, each of which has been classified as causing no liver damage, mild damage, or severe damage. These compounds were analyzed with 184 biological screens (i.e., experiments) to assess each compound's effect on a particular biologically relevant target in the body. The larger the value of each of these predictors, the higher the activity of the compound. In addition to biological screens, 192 chemical fingerprint predictors were determined for these compounds. Each of these predictors represent a substructure (i.e., an atom or combination of atoms within the compound) and are either counts of the number of substructures or an indicator of presence or absence of the particular substructure. The objective of this data set is to build a predictive model for hepatic injury so that other compounds can be screened for the likelihood of causing hepatic injury.

Load the data

```
library(AppliedPredictiveModeling)
data(hepatic)
# ?hepatic
```

Apply the following pre-processing steps:

```
# Lump all compounds that cause injury into a "Yes" category:
outcome <- tibble(
    any_damage = factor(ifelse(injury == "None", "No", "Yes"),
                        levels = c("Yes", "No") ))

#print a table
outcome |> count(any_damage) |>
    gt() |>
    grand_summary_rows(columns = n,
                fns = list(total = ~sum(.)) )
```

	any_damage	n
	Yes	175
	No	106
total	—	281

The dataframes bio and chem contain the biological assay and chemical fingerprint predictors for the 281 compounds, while the factor variable injury contains the liver damage classification for each compound.

5.1.a (5 points)

Given the size of the dataset and the injury status distribution, describe if you would create a separate training and testing data set

Yes, creating a separate training and testing data set is always a good practice to evaluate the performance of the predictive model. I would split into 80-20% training and testing sets. Also, I will make sure that the injury status distribution is maintained or rebalanced.

5.1.b (5 points)

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

I would consider using precision, recall, and F1-score but for this case I would lean in to AUC-ROC measures. AUC-ROC is robust enough to class data sets that have imbalance and it provides a measure of how well a model discriminate between positive and negative classes.

5.1.c (20 points)

Perform appropriate pre-processing of data and build logistic regression, linear discriminant analysis, penalized logistic regression and nearest shrunken centroids models described in this chapter for the biological predictors and separately for the chemical fingerprint predictors.

Biological predictors

```
bio_df <- cbind(bio, outcome = outcome$any_damage)

# Remove low frequency predictors
filtered_bio <- bio_df[, -nearZeroVar(bio_df)]

# Remove highly correlated predictors
filtered_bio$outcome <- ifelse(filtered_bio$outcome == "No", 0, 1)
correlation_matrix <- cor(filtered_bio[, -1])
highly_correlated <- findCorrelation(correlation_matrix, cutoff = 0.75)
highly_correlated_names <- colnames(filtered_bio[, -1])[highly_correlated]
biodf_reduced <- filtered_bio[, -highly_correlated]
biodf_reduced$outcome <- ifelse(biodf_reduced$outcome == 0, "No", "Yes")

# Split Bio data set to training and test sets
set.seed(rseed)
train_index <- createDataPartition(y = biodf_reduced$outcome, p = 0.8, list = FALSE)
biodf_train <- biodf_reduced[train_index, ]
biodf_test <- biodf_reduced[-train_index, ]
biodf_train_bal <- ROSE(outcome ~ ., data = biodf_train)$data

# Build classification models
set.seed(rseed)
ctrl <- trainControl(method = "cv",
                     summaryFunction = twoClassSummary,
                     classProbs = TRUE,
                     savePredictions = TRUE)

# Create Logistic Regression
lrFit <- train(outcome ~ .,
               data = biodf_train_bal,
               method = "glm",
               metric = "ROC",
               trControl = ctrl)
```

```r
# Create Linear Discriminant Analysis
ldaFit <- train(outcome ~ .,
                data = biodf_train_bal,
                method = "lda",
                preProc = c("center", "scale"),
                metric = "ROC",
                trControl = ctrl)

# Create Penalized Logistic Regression
glmnGrid <- expand.grid(alpha = seq(0, 1, by = 0.1),
                        lambda = seq(.01, .2, length = 10))
glmnFit <- train(outcome ~ .,
                   data = biodf_train_bal,
                 method = "glmnet",
                 tuneGrid = glmnGrid,
                 preProc = c("center", "scale"),
                 metric = "ROC",
                 trControl = ctrl)
optimal_glmna <- glmnFit$bestTune$alpha
optimal_glmnl <- glmnFit$bestTune$lambda
glmnmodel <- train(outcome ~ .,
                   data = biodf_train_bal,
                   method = "glmnet",
                   preProc = c("center", "scale"),
                   metric = "ROC",
                   trControl = ctrl,
                   tuneGrid = expand.grid(alpha = optimal_glmna,
                                          lambda = optimal_glmnl))

# Create Nearest Shrunken Centroids
nscGrid <- expand.grid(threshold = seq(0, 25, length = 30))
nscFit <- train(outcome ~ .,
                data = biodf_train_bal,
                method = "pam",
                preProc = c("center", "scale"),
                tuneGrid = nscGrid,
                metric = "ROC",
                trControl = ctrl)
```

```
11111111111
```

Chemical predictors

```
chem_df <- cbind(chem, outcome = outcome$any_damage)

# Remove low frequency predictors
filtered_chem <- chem_df[, -nearZeroVar(chem_df)]

# Remove highly correlated predictors
filtered_chem$outcome <- ifelse(filtered_chem$outcome == "No", 0, 1)
correlation_matrix1 <- cor(filtered_chem[, -1])
highly_correlated1 <- findCorrelation(correlation_matrix1, cutoff = 0.75)
highly_correlated_names1 <- colnames(filtered_chem[, -1])[highly_correlated1]
chemdf_reduced <- filtered_chem[, -highly_correlated1]
chemdf_reduced$outcome <- ifelse(chemdf_reduced$outcome == 0, "No", "Yes")

# Split chem data set to training and test sets
set.seed(rseed)
train_index1 <- createDataPartition(y = chemdf_reduced$outcome, p = 0.8, list = FALSE)
chemdf_train <- chemdf_reduced[train_index1, ]
chemdf_test <- chemdf_reduced[-train_index1, ]
chemdf_train_bal <- ROSE(outcome ~ ., data = chemdf_train)$data


# Build classification models
set.seed(rseed)
ctrl1 <- trainControl(method = "cv", summaryFunction = twoClassSummary,
                      classProbs = TRUE, savePredictions = TRUE)

# Create Logistic Regression
clrFit <- train(outcome ~ .,
                data = chemdf_train_bal,
                method = "glm",
                metric = "ROC",
                trControl = ctrl1)

# Create Linear Discriminant Analysis
cldaFit <- train(outcome ~ .,
                 data = chemdf_train_bal,
                 method = "lda",
                 preProc = c("center", "scale"),
                 metric = "ROC",
                 trControl = ctrl1)
```

```
# Create Penalized Logistic Regression
cglmnGrid <- expand.grid(alpha = seq(0, 1, by = 0.1),
                         lambda = seq(.01, .2, length = 10))
cglmnFit <- train(outcome ~ .,
                  data = chemdf_train_bal,
                  method = "glmnet",
                  tuneGrid = cglmnGrid,
                  preProc = c("center", "scale"),
                  metric = "ROC",
                  trControl = ctrl1)
coptimal_glmna <- cglmnFit$bestTune$alpha
coptimal_glmnl <- cglmnFit$bestTune$lambda
cglmnmodel <- train(outcome ~ .,
                    data = chemdf_train_bal,
                    method = "glmnet",
                    preProc = c("center", "scale"),
                    metric = "ROC",
                    trControl = ctrl1,
                    tuneGrid = expand.grid(alpha = coptimal_glmna,
                                           lambda = coptimal_glmnl))

# Create Nearest Shrunken Centroids
cnscGrid <- expand.grid(threshold = seq(0, 25, length = 30))
cnscFit <- train(outcome ~ .,
                 data = chemdf_train_bal,
                 method = "pam",
                 preProc = c("center", "scale"),
                 tuneGrid = cnscGrid,
                 metric = "ROC",
                 trControl = ctrl1)
```

```
11111111111
```

```
# Bio Models Predictions and Confusion Matrix
testResults <- data.frame(obs = biodf_test$outcome,
                          LR = predict(lrFit, biodf_test[,1:81]))
testResults$LDA <- predict(ldaFit, biodf_test[,1:81])
testResults$GLMNet <- predict(glmnmodel, biodf_test[,1:81])
testResults$NSC <- predict(nscFit, biodf_test[,1:81])

testResults$obs <- factor(testResults$obs, levels = levels(testResults$LR))
confusionMatrix(testResults$LR, testResults$obs, positive = "Yes")
```

```
Confusion Matrix and Statistics

          Reference
Prediction No Yes
       No   5  19
       Yes 16  16

               Accuracy : 0.375
                 95% CI : (0.2492, 0.5145)
    No Information Rate : 0.625
    P-Value [Acc > NIR] : 1.0000

                  Kappa : -0.2963

 Mcnemar's Test P-Value : 0.7353

            Sensitivity : 0.4571
            Specificity : 0.2381
         Pos Pred Value : 0.5000
         Neg Pred Value : 0.2083
             Prevalence : 0.6250
         Detection Rate : 0.2857
   Detection Prevalence : 0.5714
      Balanced Accuracy : 0.3476

       'Positive' Class : Yes
```

```
confusionMatrix(testResults$LDA, testResults$obs, positive = "Yes")
```

```
Confusion Matrix and Statistics

          Reference
Prediction No Yes
       No   8  22
       Yes 13  13

               Accuracy : 0.375
                 95% CI : (0.2492, 0.5145)
    No Information Rate : 0.625
    P-Value [Acc > NIR] : 1.0000
```

```
                 Kappa : -0.2281

 Mcnemar's Test P-Value : 0.1763

            Sensitivity : 0.3714
            Specificity : 0.3810
         Pos Pred Value : 0.5000
         Neg Pred Value : 0.2667
             Prevalence : 0.6250
         Detection Rate : 0.2321
   Detection Prevalence : 0.4643
      Balanced Accuracy : 0.3762

       'Positive' Class : Yes
```

```
confusionMatrix(testResults$GLMNet, testResults$obs, positive = "Yes")
```

```
Confusion Matrix and Statistics

          Reference
Prediction No Yes
       No   7  19
       Yes 14  16

               Accuracy : 0.4107
                 95% CI : (0.281, 0.5502)
    No Information Rate : 0.625
    P-Value [Acc > NIR] : 0.9996

                  Kappa : -0.2

 Mcnemar's Test P-Value : 0.4862

            Sensitivity : 0.4571
            Specificity : 0.3333
         Pos Pred Value : 0.5333
         Neg Pred Value : 0.2692
             Prevalence : 0.6250
         Detection Rate : 0.2857
   Detection Prevalence : 0.5357
      Balanced Accuracy : 0.3952
```

```
       'Positive' Class : Yes
```

```r
confusionMatrix(testResults$NSC, testResults$obs, positive = "Yes")
```

```
Confusion Matrix and Statistics

          Reference
Prediction No Yes
       No   2   6
       Yes 19  29

               Accuracy : 0.5536
                 95% CI : (0.4147, 0.6866)
    No Information Rate : 0.625
    P-Value [Acc > NIR] : 0.8920

                  Kappa : -0.087

 Mcnemar's Test P-Value : 0.0164

            Sensitivity : 0.82857
            Specificity : 0.09524
         Pos Pred Value : 0.60417
         Neg Pred Value : 0.25000
             Prevalence : 0.62500
         Detection Rate : 0.51786
   Detection Prevalence : 0.85714
      Balanced Accuracy : 0.46190

       'Positive' Class : Yes
```

```r
# Chem Models Predictions and Confusion Matrix
testResults1 <- data.frame(obs = chemdf_test$outcome,
                           LR = predict(clrFit, chemdf_test[,1:73]))
testResults1$LDA <- predict(cldaFit, chemdf_test[,1:73])
testResults1$GLMNet <- predict(cglmnmodel, chemdf_test[,1:73])
testResults1$NSC <- predict(cnscFit, chemdf_test[,1:73])
```

```r
testResults1$obs <- factor(testResults1$obs, levels = levels(testResults1$LR))
confusionMatrix(testResults1$LR, testResults1$obs, positive = "Yes")
```

```
Confusion Matrix and Statistics

          Reference
Prediction No Yes
       No   8  17
       Yes 13  18

               Accuracy : 0.4643
                 95% CI : (0.3299, 0.6026)
    No Information Rate : 0.625
    P-Value [Acc > NIR] : 0.9951

                  Kappa : -0.1009

 Mcnemar's Test P-Value : 0.5839

            Sensitivity : 0.5143
            Specificity : 0.3810
         Pos Pred Value : 0.5806
         Neg Pred Value : 0.3200
             Prevalence : 0.6250
         Detection Rate : 0.3214
   Detection Prevalence : 0.5536
      Balanced Accuracy : 0.4476

       'Positive' Class : Yes
```

```r
confusionMatrix(testResults1$LDA, testResults1$obs, positive = "Yes")
```

```
Confusion Matrix and Statistics

          Reference
Prediction No Yes
       No   9  19
       Yes 12  16

               Accuracy : 0.4464
```

```
               95% CI : (0.3134, 0.5853)
    No Information Rate : 0.625
    P-Value [Acc > NIR] : 0.9978

                  Kappa : -0.1071

 Mcnemar's Test P-Value : 0.2812

            Sensitivity : 0.4571
            Specificity : 0.4286
         Pos Pred Value : 0.5714
         Neg Pred Value : 0.3214
             Prevalence : 0.6250
         Detection Rate : 0.2857
   Detection Prevalence : 0.5000
      Balanced Accuracy : 0.4429

       'Positive' Class : Yes
```

```r
confusionMatrix(testResults1$GLMNet, testResults1$obs, positive = "Yes")
```

```
Confusion Matrix and Statistics

          Reference
Prediction No Yes
       No  11  21
       Yes 10  14

               Accuracy : 0.4464
                 95% CI : (0.3134, 0.5853)
    No Information Rate : 0.625
    P-Value [Acc > NIR] : 0.99780

                  Kappa : -0.069

 Mcnemar's Test P-Value : 0.07249

            Sensitivity : 0.4000
            Specificity : 0.5238
         Pos Pred Value : 0.5833
         Neg Pred Value : 0.3438
```

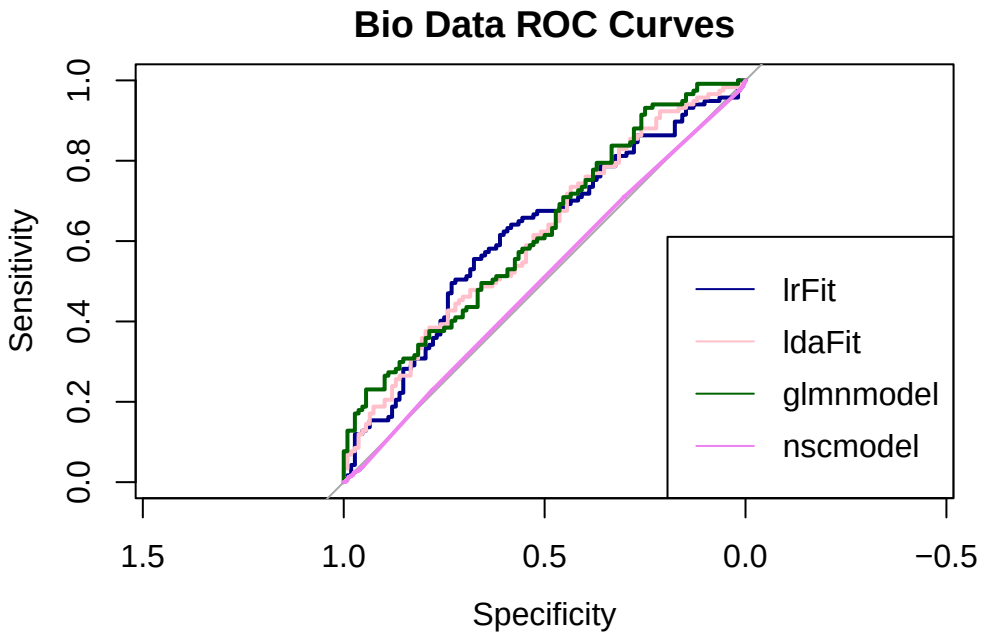
```
            Prevalence : 0.6250
        Detection Rate : 0.2500
  Detection Prevalence : 0.4286
     Balanced Accuracy : 0.4619

      'Positive' Class : Yes
```

```
confusionMatrix(testResults1$NSC, testResults1$obs, positive = "Yes")
```

```
Confusion Matrix and Statistics

          Reference
Prediction No Yes
       No   9  19
       Yes 12  16

               Accuracy : 0.4464
                 95% CI : (0.3134, 0.5853)
    No Information Rate : 0.625
    P-Value [Acc > NIR] : 0.9978

                  Kappa : -0.1071

 Mcnemar's Test P-Value : 0.2812

            Sensitivity : 0.4571
            Specificity : 0.4286
         Pos Pred Value : 0.5714
         Neg Pred Value : 0.3214
             Prevalence : 0.6250
         Detection Rate : 0.2857
   Detection Prevalence : 0.5000
      Balanced Accuracy : 0.4429

       'Positive' Class : Yes
```

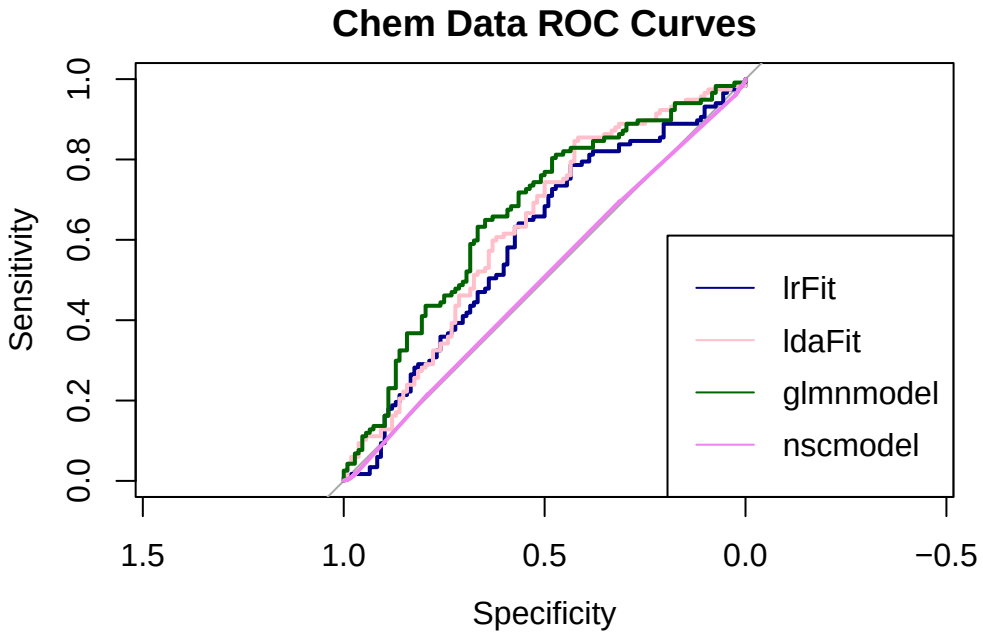
```
# Bio ROC Plots
roc_lrFit <- roc(response = lrFit$pred$obs,
                 predictor = lrFit$pred$Yes,
                 levels = rev(levels(lrFit$pred$obs)))
```

```
roc_ldaFit <- roc(response = ldaFit$pred$obs,
                  predictor = ldaFit$pred$Yes,
                  levels = rev(levels(ldaFit$pred$obs)))
roc_glmnmodel <- roc(response = glmnmodel$pred$obs,
                     predictor = glmnmodel$pred$Yes,
                     levels = rev(levels(glmnmodel$pred$obs)))
roc_nscmodel <- roc(response = nscFit$pred$obs,
                    predictor = nscFit$pred$Yes,
                    levels = rev(levels(nscFit$pred$obs)))

plot(roc_lrFit, col = "darkblue", main = "Bio Data ROC Curves")
plot(roc_ldaFit, col = "pink", add = TRUE)
plot(roc_glmnmodel, col = "darkgreen", add = TRUE)
plot(roc_nscmodel, col = "violet", add = TRUE)
legend("bottomright", legend = c("lrFit", "ldaFit", "glmnmodel", "nscmodel"),
       col = c("darkblue", "pink", "darkgreen", "violet"), lty = 1)
```




```
# Chem ROC Plots
roc_clrFit <- roc(response = clrFit$pred$obs,
                  predictor = clrFit$pred$Yes,
                  levels = rev(levels(clrFit$pred$obs)))
roc_cldaFit <- roc(response = cldaFit$pred$obs,
                   predictor = cldaFit$pred$Yes,
```

```
                    levels = rev(levels(cldaFit$pred$obs)))
roc_cglmnmodel <- roc(response = cglmnmodel$pred$obs,
                      predictor = cglmnmodel$pred$Yes,
                      levels = rev(levels(cglmnmodel$pred$obs)))
roc_cnscmodel <- roc(response = cnscFit$pred$obs,
                     predictor = cnscFit$pred$Yes,
                     levels = rev(levels(cnscFit$pred$obs)))

plot(roc_clrFit, col = "darkblue", main = "Chem Data ROC Curves")
plot(roc_cldaFit, col = "pink", add = TRUE)
plot(roc_cglmnmodel, col = "darkgreen", add = TRUE)
plot(roc_cnscmodel, col = "violet", add = TRUE)
legend("bottomright", legend = c("lrFit", "ldaFit", "glmnmodel", "nscmodel"),
       col = c("darkblue", "pink", "darkgreen", "violet"), lty = 1)
```




```
# Calculate AUC of each model
calc_auc <- function(obs, LR, LDA, GLMNet, NSC) {
  roc_lr <- roc(obs, as.numeric(LR))
  roc_lda <- roc(obs, as.numeric(LDA))
  roc_glmnet <- roc(obs, as.numeric(GLMNet))
  roc_nsc <- roc(obs, as.numeric(NSC))

  auc_lr <- auc(roc_lr)
```

```
  auc_lda <- auc(roc_lda)
  auc_glmnet <- auc(roc_glmnet)
  auc_nsc <- auc(roc_nsc)

  return(c(auc_lr, auc_lda, auc_glmnet, auc_nsc))}

bio_models <- c("LR", "LDA", "GLMNet", "NSC")
bio_aucs <- calc_auc(testResults$obs, testResults$LR, testResults$LDA,
                     testResults$GLMNet, testResults$NSC)

chem_models <- c("LR", "LDA", "GLMNet", "NSC")
chem_aucs <- calc_auc(testResults1$obs, testResults1$LR, testResults1$LDA,
                      testResults1$GLMNet, testResults1$NSC)

# Table AUC-ROC Comparison
summary_table <- data.frame(
  Model = c(rep(bio_models, each = 1), rep(chem_models, each = 1)),
  Dataset = c(rep("Biological", times = length(bio_models)),
              rep("Chemical", times = length(chem_models))),
  AUC = c(bio_aucs, chem_aucs))

summary_table |> gt() |>
  tab_header(title = "Summary of AUC Values for Diff. Models and Datasets using Test Data")
  fmt_number(columns = vars(AUC), decimals = 3)
```

Summary of AUC Values for Diff. Models and Datasets using Test Data

Model	Dataset	AUC
LR	Biological	0.652
LDA	Biological	0.624
GLMNet	Biological	0.605
NSC	Biological	0.462
LR	Chemical	0.448
LDA	Chemical	0.557
GLMNet	Chemical	0.462
NSC	Chemical	0.557

Which model has the best predictive ability for the biological predictors, and what is the optimal performance? Which model has the best predictive ability for the chemical predictors, and what is the optimal performance? Based on

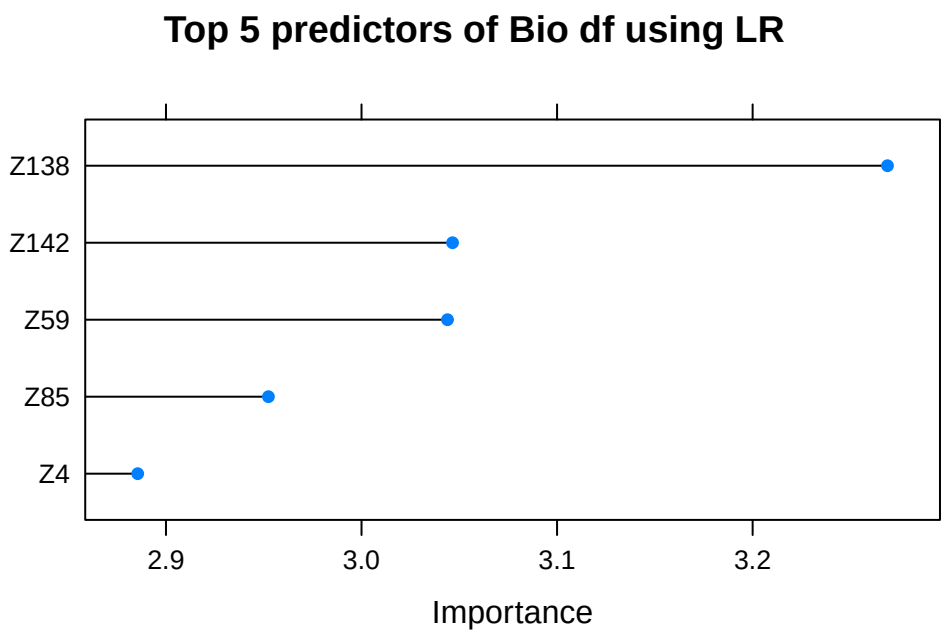
these results, which set of predictors (biological or chemical) contains the most information about hepatic toxicity?

For biological predictors, Logistic regression model has the best predictive ability at ~0.65 AUC score. While for chemical predictors, Nearest Shrunken Centroids and Linear discriminant analysis tied at ~0.56 AUC score. All in all though, the AUC scores are showing poor predictive ability at <.7. Z138 contains the most information about hepatic toxicity for biological predictors and X171 for chemical predictors (See 5.1.d).

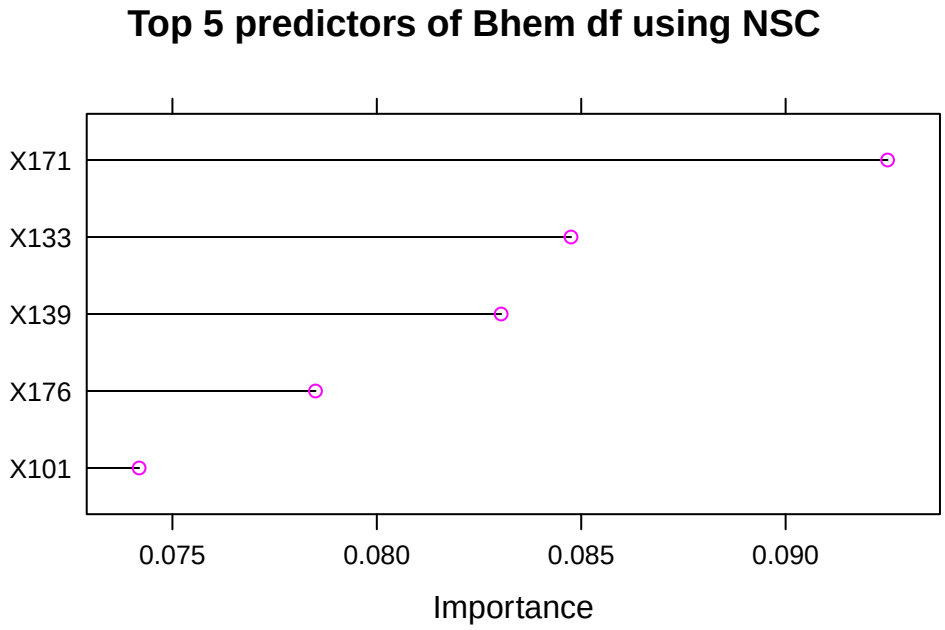
5.1.d (5 points)

For the optimal models for both the biological and chemical predictors, what are the top five important predictors?

```
# Top 5 predictors of bio df using LR
bioImp <- varImp(lrFit, scale = FALSE)
plot(bioImp, top = 5, main = "Top 5 predictors of Bio df using LR")
```




```
# Top 5 predictors of chem df using NSC
chemImp1 <- varImp(cnscFit, scale = FALSE)
plot(chemImp1, top = 5, main = "Top 5 predictors of Bhem df using NSC")
```




```
#Cant run LDA model through varImp or any alternatives
```

The top 5 bio important predictors are Z138, Z142, Z59, Z85, and Z4. The top 5 chem important predictors are X171, X133 X139, X176, and X101.

5.1.e (20 points)

Combined predictors

Now combine the biological and chemical fingerprint predictors into one predictor set. Retrain the same set of predictive models you built from part (c).

```
hepatic_df <- cbind(bio, chem, outcome = outcome$any_damage)

# Remove low frequency predictors
filt_hepatic <- hepatic_df[, -nearZeroVar(hepatic_df)]

# Remove highly correlated predictors
filt_hepatic$outcome <- ifelse(filt_hepatic$outcome == "No", 0, 1)
correlation_matrix2 <- cor(filt_hepatic[, -1])
highly_correlated2 <- findCorrelation(correlation_matrix2, cutoff = 0.70)
highly_correlated_names2 <- colnames(filt_hepatic[, -1])[highly_correlated2]
hepatic_reduced <- filt_hepatic[, -highly_correlated2]
hepatic_reduced$outcome <- ifelse(hepatic_reduced$outcome == 0, "No", "Yes")
```

```r
# Split Bio data set to training and test sets
set.seed(rseed)
train_index2 <- createDataPartition(y = hepatic_reduced$outcome, p = 0.8, list = FALSE)
hepatic_train <- hepatic_reduced[train_index2, ]
hepatic_test <- hepatic_reduced[-train_index2, ]
hepatic_train_bal <- ROSE(outcome ~ ., data = hepatic_train)$data

# Build classification models
set.seed(rseed)
ctrl2 <- trainControl(method = "cv",
                      summaryFunction = twoClassSummary,
                      classProbs = TRUE,
                      savePredictions = TRUE)

# Create Logistic Regression
lrFit_h <- train(outcome ~ .,
                 data = hepatic_train_bal,
                 method = "glm",
                 metric = "ROC",
                 trControl = ctrl2)

# Create Linear Discriminant Analysis
ldaFit_h <- train(outcome ~ .,
                  data = hepatic_train_bal,
                  method = "lda",
                  preProc = c("center", "scale"),
                  metric = "ROC",
                  trControl = ctrl2)

# Create Penalized Logistic Regression
glmnGrid_h <- expand.grid(alpha = seq(0, 1, by = 0.1),
                          lambda = seq(.01, .2, length = 10))
glmnFit_h <- train(outcome ~ .,
                     data = hepatic_train_bal,
                   method = "glmnet",
                   tuneGrid = glmnGrid_h,
                   preProc = c("center", "scale"),
                   metric = "ROC",
                   trControl = ctrl2)
optimal_glmnah <- glmnFit_h$bestTune$alpha
optimal_glmnlh <- glmnFit_h$bestTune$lambda
glmnmodel_h <- train(outcome ~ .,
```

```
                    data = hepatic_train_bal,
                    method = "glmnet",
                    preProc = c("center", "scale"),
                    metric = "ROC",
                    trControl = ctrl2,
                    tuneGrid = expand.grid(alpha = optimal_glmnah,
                                           lambda = optimal_glmnlh))

# Create Nearest Shrunken Centroids
nscGridh <- expand.grid(threshold = seq(0, 25, length = 30))
nscFit_h <- train(outcome ~ .,
                  data = hepatic_train_bal,
                  method = "pam",
                  preProc = c("center", "scale"),
                  tuneGrid = nscGridh,
                  metric = "ROC",
                  trControl = ctrl2)
```

```
11111111111
```

Which model yields the best predictive performance?

Logistic regression model performed the best in this case.

```
# Run test through models
testResults2 <- data.frame(obs = hepatic_test$outcome,
                           LR = predict(lrFit_h, hepatic_test[,1:143]))
testResults2$LDA <- predict(ldaFit_h, hepatic_test[,1:143])
testResults2$GLMNet <- predict(glmnmodel_h, hepatic_test[,1:143])
testResults2$NSC <- predict(nscFit_h, hepatic_test[,1:143])

# Calculate AUC of each model
calc_auc <- function(obs, LR, LDA, GLMNet, NSC) {
  roc_lr <- roc(obs, as.numeric(LR))
  roc_lda <- roc(obs, as.numeric(LDA))
  roc_glmnet <- roc(obs, as.numeric(GLMNet))
  roc_nsc <- roc(obs, as.numeric(NSC))

  auc_lr <- auc(roc_lr)
  auc_lda <- auc(roc_lda)
  auc_glmnet <- auc(roc_glmnet)
  auc_nsc <- auc(roc_nsc)
```

```
  return(c(auc_lr, auc_lda, auc_glmnet, auc_nsc))}

hepatic_models <- c("LR", "LDA", "GLMNet", "NSC")
hepatic_aucs <- calc_auc(testResults2$obs, testResults2$LR, testResults2$LDA,
                         testResults2$GLMNet,testResults2$NSC)

# Table AUC-ROC Comparison
summary_table2 <- data.frame(
  Model = c(rep(hepatic_models, each = 1)),
  Dataset = c(rep("Hepatic", times = length(hepatic_models))),
  AUC = c(hepatic_aucs))

summary_table2 |> gt() |>
  tab_header(title = "Summary of AUC Values Using Test Data") |>
  fmt_number(columns = vars(AUC), decimals = 3)
```

Summary of AUC Values Using Test Data

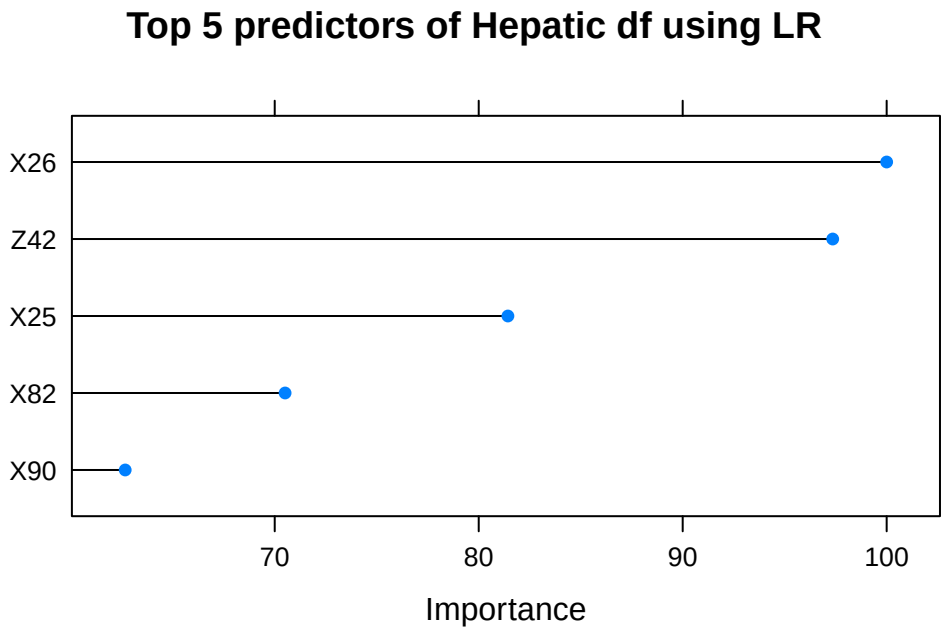
Model	Dataset	AUC
LR	Hepatic	0.643
LDA	Hepatic	0.519
GLMNet	Hepatic	0.486
NSC	Hepatic	0.581

Is the model performance better than either of the best models from part (c)?

Yes, the combined data's Logistic regression model performed slightly better than chem data's NSC/LDA based on AUC scores. But it scored slightly below bio data's LR.

What are the top five important predictors for the optimal model?

```
# Top 5 predictors of hepatic df using LR
hepImp <- varImp(lrFit_h)
plot(hepImp, top = 5, main = "Top 5 predictors of Hepatic df using LR")
```



How do these compare with the optimal predictors from each individual predictor set?

None of these top 5 predictors for the combined data set matches with either bio and chem data set's top 5 predictors.

5.1.f (5 points)

Which model (either model of individual biology or chemical fingerprints or the combined predictor model), if any, would you recommend using to predict compounds' hepatic toxicity? Explain.

I wouldn't recommend any of the model based on the AUC scores. They are all under 0.7 and suggest that none of the models are capable of differentiating positive from negative classes. But if I have to chose I would recommend the Logistic regression because of it's consistent higher AUC score between bio data and combined data evaluation.

Problem 5.2 (30 points)

Brodnjak-Vonina et al. (2005) develop a methodology for food laboratories to determine the type of oil from a sample. In their procedure, they used a gas chromatograph (an instrument that separates chemicals in a sample) to measure seven different fatty acids in an oil. These measurements would then be used to predict the type of oil in a food sample. To create their model, they used 96 samples of seven types of oils. These data can be found in the caret

package using data(oil). The oil types are contained in a factor variable called oilType. The types are pumpkin (coded as A), sunflower (B), peanut (C), olive (D), soybean (E), rapeseed (F), and corn (G). 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.

```
data(oil)
table(oilType)
```

```
oilType
 A  B  C  D  E  F  G
37 26  3  7 11 10  2
```

5.2.a (5 points)

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

Yes, splitting into training and test set to have a way to evaluate the data and helps address issues like over-fitting and improve model through tuning. It is also good to perform train set rebalancing technique to address class imbalance in the outcome distribution. It helps the minority class gain traction and prevent the model from being bias towards the majority class.

5.2.b (5 points)

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

I think balanced accuracy and AUC-ROC would be able to capture the model's ability to discriminate against the 7 different oil types. Balanced accuracy accounts for unequal class distribution by looking at the average accuracy of each class weighted by the true instances. AUC-ROC is a lot more comprehensive by considering sensitivity and specificity. It evaluates the overall performance of the models at all possible classification thresholds.

5.2.c (20 points)

Build linear discriminant analysis, penalized multinomial regression, and nearest shrunken centroids models to this data; predict the fitted models on the training data set and evaluate (show the confusion matrix for each model output) ...

(Warnings suppressed - these warnings related to the sparsity of the data for some oil types.)

```r
# Split data to train and test sets
oilType <- as.data.frame(oilType)
oil_df <- cbind(fattyAcids, outcome = oilType$oilType)

set.seed(rseed)
train_index3 <- createDataPartition(y = oil_df$outcome, p = 0.8, list = FALSE)
oil_train <- oil_df[train_index3, ]
oil_test <- oil_df[-train_index3, ]
```

LDA

```r
set.seed(rseed)
ctrl3 <- trainControl(method = "cv",
                      summaryFunction = defaultSummary,
                      classProbs = TRUE,
                      savePredictions = TRUE)

# Create Linear Discriminant Analysis
ldaFit_o <- train(outcome ~ .,
                  data = oil_train,
                  method = "lda",
                  preProc = c("center", "scale"),
                  metric = "Accuracy",
                  trControl = ctrl3)
lda_train_predictions <- predict(ldaFit_o, newdata = oil_train)

ldaFit_cm <- confusionMatrix(lda_train_predictions, reference = oil_train$outcome)
ldaFit_cm
```

```
Confusion Matrix and Statistics

          Reference
Prediction  A  B  C  D  E  F  G
         A 28  0  0  0  0  0  0
         B  2 21  0  0  0  0  0
         C  0  0  3  0  0  0  0
         D  0  0  0  6  0  0  0
         E  0  0  0  0  9  0  0
         F  0  0  0  0  0  8  0
         G  0  0  0  0  0  0  2
```

```
Overall Statistics

               Accuracy : 0.9747
                 95% CI : (0.9115, 0.9969)
    No Information Rate : 0.3797
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9666

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: A Class: B Class: C Class: D Class: E Class: F
Sensitivity            0.9333   1.0000  1.00000  1.00000   1.0000   1.0000
Specificity            1.0000   0.9655  1.00000  1.00000   1.0000   1.0000
Pos Pred Value         1.0000   0.9130  1.00000  1.00000   1.0000   1.0000
Neg Pred Value         0.9608   1.0000  1.00000  1.00000   1.0000   1.0000
Prevalence             0.3797   0.2658  0.03797  0.07595   0.1139   0.1013
Detection Rate         0.3544   0.2658  0.03797  0.07595   0.1139   0.1013
Detection Prevalence   0.3544   0.2911  0.03797  0.07595   0.1139   0.1013
Balanced Accuracy      0.9667   0.9828  1.00000  1.00000   1.0000   1.0000
                     Class: G
Sensitivity           1.00000
Specificity           1.00000
Pos Pred Value        1.00000
Neg Pred Value        1.00000
Prevalence            0.02532
Detection Rate        0.02532
Detection Prevalence  0.02532
Balanced Accuracy     1.00000
```

PLR

```
# Create Penalized Logistic Regression
glmnGrid_o <- expand.grid(alpha = seq(0, 1, by = 0.1),
                          lambda = seq(.01, .2, length = 10))
glmnFit_o <- train(outcome ~ .,
                   data = oil_train,
                   method = "glmnet",
                   tuneGrid = glmnGrid_o,
                   preProc = c("center", "scale"),
```

```
                   metric = "Accuracy",
                   trControl = ctrl3)
optimal_glmnao <- glmnFit_o$bestTune$alpha
optimal_glmnlo <- glmnFit_o$bestTune$lambda
glmnmodel_o <- train(outcome ~ .,
                   data = oil_train,
                   method = "glmnet",
                   preProc = c("center", "scale"),
                   metric = "Accuracy",
                   trControl = ctrl3,
                   tuneGrid = expand.grid(alpha = optimal_glmnao,
                                          lambda = optimal_glmnlo))
glmn_train_predictions <- predict(glmnmodel_o, newdata = oil_train)

glmn_cm <- confusionMatrix(glmn_train_predictions, reference = oil_train$outcome)
glmn_cm
```

```
Confusion Matrix and Statistics

          Reference
Prediction  A  B  C  D  E  F  G
         A 28  0  0  0  0  0  0
         B  2 21  0  0  0  0  2
         C  0  0  3  0  0  0  0
         D  0  0  0  6  0  0  0
         E  0  0  0  0  9  0  0
         F  0  0  0  0  0  8  0
         G  0  0  0  0  0  0  0

Overall Statistics

               Accuracy : 0.9494
                 95% CI : (0.8754, 0.986)
    No Information Rate : 0.3797
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9326

 Mcnemar's Test P-Value : NA

Statistics by Class:
```

```
                     Class: A Class: B Class: C Class: D Class: E Class: F
Sensitivity            0.9333   1.0000  1.00000  1.00000   1.0000   1.0000
Specificity            1.0000   0.9310  1.00000  1.00000   1.0000   1.0000
Pos Pred Value         1.0000   0.8400  1.00000  1.00000   1.0000   1.0000
Neg Pred Value         0.9608   1.0000  1.00000  1.00000   1.0000   1.0000
Prevalence             0.3797   0.2658  0.03797  0.07595   0.1139   0.1013
Detection Rate         0.3544   0.2658  0.03797  0.07595   0.1139   0.1013
Detection Prevalence   0.3544   0.3165  0.03797  0.07595   0.1139   0.1013
Balanced Accuracy      0.9667   0.9655  1.00000  1.00000   1.0000   1.0000
                     Class: G
Sensitivity           0.00000
Specificity           1.00000
Pos Pred Value            NaN
Neg Pred Value        0.97468
Prevalence            0.02532
Detection Rate        0.00000
Detection Prevalence  0.00000
Balanced Accuracy     0.50000
```

NSC

```r
# Create Nearest Shrunken Centroids
nscGrido <- expand.grid(threshold = seq(0, 25, length = 30))
nscFit_o <- train(outcome ~ .,
                  data = oil_train,
                  method = "pam",
                  preProc = c("center", "scale"),
                  tuneGrid = nscGrido,
                  metric = "Accuracy",
                  trControl = ctrl3)
```

```
1Warning: a class contains only 1 sample111Warning: a class contains only 1 sample1111111
```

```r
nsc_train_predictions <- predict(nscFit_o, newdata = oil_train)

nsc_cm <- confusionMatrix(nsc_train_predictions, reference = oil_train$outcome)
nsc_cm
```

```
Confusion Matrix and Statistics

          Reference
```

```
Prediction  A  B  C  D  E  F  G
         A 28  0  0  0  0  0  0
         B  2 21  0  0  0  0  0
         C  0  0  3  0  0  0  0
         D  0  0  0  6  0  0  0
         E  0  0  0  0  9  0  0
         F  0  0  0  0  0  8  0
         G  0  0  0  0  0  0  2

Overall Statistics

               Accuracy : 0.9747
                 95% CI : (0.9115, 0.9969)
    No Information Rate : 0.3797
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9666

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: A Class: B Class: C Class: D Class: E Class: F
Sensitivity            0.9333   1.0000  1.00000  1.00000   1.0000   1.0000
Specificity            1.0000   0.9655  1.00000  1.00000   1.0000   1.0000
Pos Pred Value         1.0000   0.9130  1.00000  1.00000   1.0000   1.0000
Neg Pred Value         0.9608   1.0000  1.00000  1.00000   1.0000   1.0000
Prevalence             0.3797   0.2658  0.03797  0.07595   0.1139   0.1013
Detection Rate         0.3544   0.2658  0.03797  0.07595   0.1139   0.1013
Detection Prevalence   0.3544   0.2911  0.03797  0.07595   0.1139   0.1013
Balanced Accuracy      0.9667   0.9828  1.00000  1.00000   1.0000   1.0000
                     Class: G
Sensitivity           1.00000
Specificity           1.00000
Pos Pred Value        1.00000
Neg Pred Value        1.00000
Prevalence            0.02532
Detection Rate        0.02532
Detection Prevalence  0.02532
Balanced Accuracy     1.00000
```

Accuracies Summary

```r
# summarize the accuracies for all models in a table
model_accuracies <- data.frame(
  Model = c("LDA", "GLMNet", "NSC"),
  Accuracy = c(round(ldaFit_cm$overall["Accuracy"],4),
               round(glmn_cm$overall["Accuracy"],4),
               round(nsc_cm$overall["Accuracy"],4)))

model_accuracies |> gt() |> tab_header(title = "Model Accuracies")
```

Model Accuracies

Model	Accuracy
LDA	0.9747
GLMNet	0.9494
NSC	0.9747

which model performs best on these data?

Based on the accuracy results, LDA and NSC both have the same Accuracy scores at 97%.

Which oil type does the optimal model most accurately predict?

Looking at statistics by class, all classes were accurately classified as the lowest was scored 97% balanced accuracy for Palmitic. Oleic, linoleic, linolenic, Eicosenoic, and Eicosanoic were classified at 100% balanced accuracy.

Which oil type does the optimal model least accurately predict?

Palmitic was the oil type that the models predicted least accurately using the train data at 97% balanced accuracy which is still a great score to have.