PubH 6886 Fall 2025 Assignment Three

AUTHOR
Annie Allred

```r
library(tidyverse)
library(knitr)
library(leaps)
library(glmnet)
library(caret)
library(pls)
library(plotmo)
library(gridExtra)
```

Question One

(106 pts.) The data set `tecator` from the `caret` package contains data from 215 samples of meat that were analyzed for water, fat, and protein content. Each sample was also analyzed on a Tecator Infratec Food and Feed Analyzer working in the wavelength range 850 - 1050 nm by the Near Infrared Transmission (NIT) principle. The Tecator Analyzer provides absorbency measures at 100 different wavelengths in the 850 - 1050 nm range. For the purposes of this assignment, we will only use the first 130 observations from this data set. Use the following code to load the data, create the predictor matrix, and create the response vector.

```r
library(caret)
# load dataset
data(tecator)
# create predictor matrix
absorp_X <- absorp[1:130,]
colnames(absorp_X) <- paste0("wl", 1:100) # name variables wl1, ..., wl100
# create response vector
protein <- endpoints[1:130,3]
```

For each part below, if you use the `train()` function to perform cross-validation, set the seed to 1234 (`set.seed(1234)`) prior to running the function.

(a)

(6 pts.) Suppose that we want to fit a linear regression model that uses the absorbency measures to predict protein content. Briefly explain why the least squares estimates for a linear regression model based on the absorbency measures may not provide the best predictive model with respect to prediction error.

- The reason that the least squares estimates for a linear regression model may not provide the best predictive model with respect to prediction error is due to the sample size of the study, the number of variables, and their relationship. Sample size, n is only slightly larger than the number of variables, p. Least squares estimates works best when $n >> p$.

(b)

(6 pts.) Regardless of what was stated in part (a) fit a linear regression model via least squares estimation with the absorbency measures as the predictors and protein content as the response. Report the 10-fold CV RMSE and R^2 values.

- For the linear regression model using least squares estimation, the 10-fold CV RMSE and R^2 values are 2.572 and 0.705 respectively.

```r
# setting the seed
set.seed(1234)
# using train() to find the CV RMSE and R^2 values
(q1_lm <- train(x = absorp_X, y = protein, method = "lm",
                trControl = trainControl(method = "cv", number = 10,
                                         savePredictions = TRUE)))
```

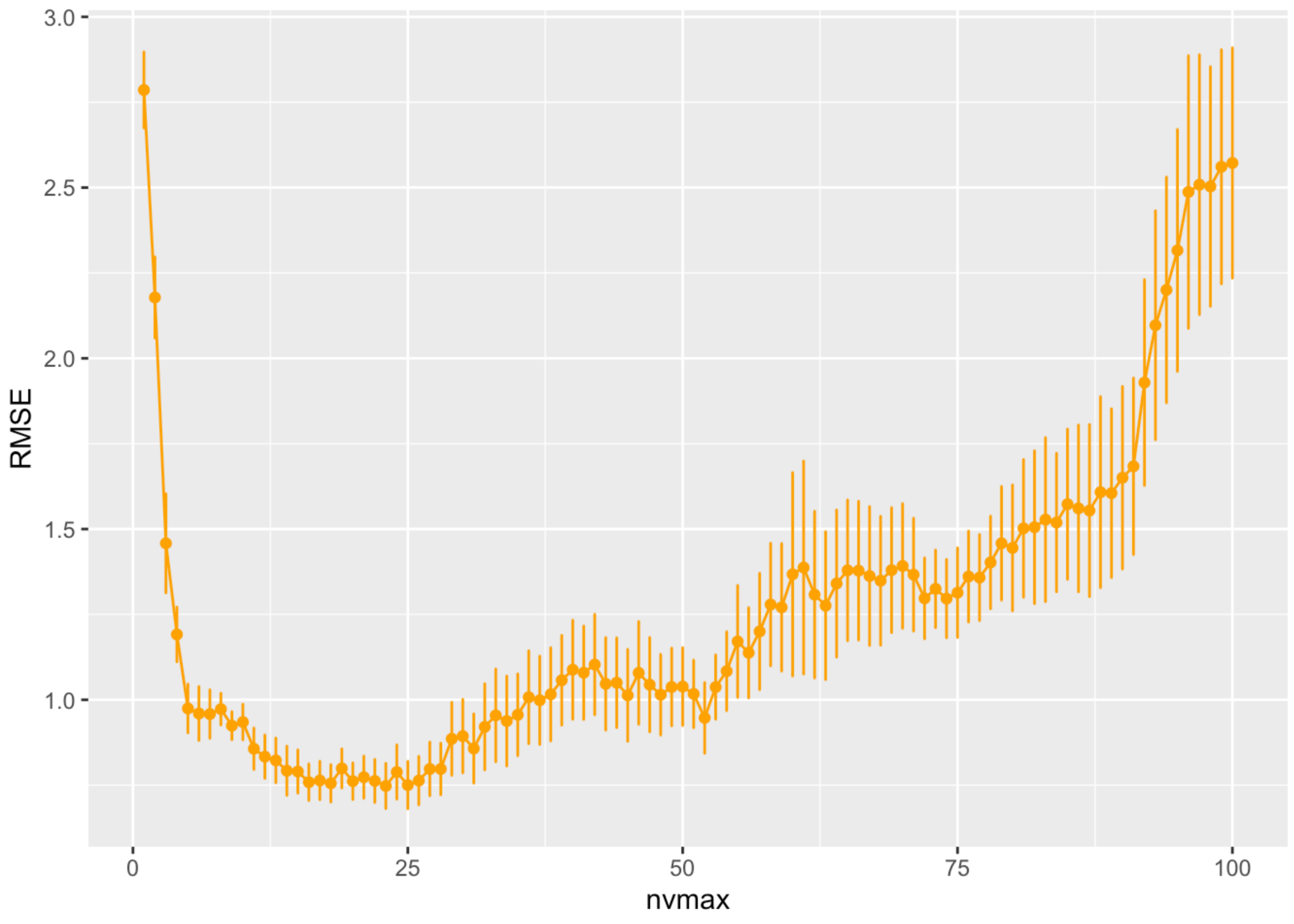
```
Linear Regression

130 samples
100 predictors

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 117, 116, 116, 118, 116, 118, ...
Resampling results:

  RMSE      Rsquared   MAE
  2.572102  0.7048167  1.54235

Tuning parameter 'intercept' was held constant at a value of TRUE
```

(c)

(12 pts.) Conduct forward stepwise selection for constructing a linear regression model to predict protein content using the absorbency measures. Use 10-fold CV RMSE to select your final model. Which absorbency measures are included in your final model? Report the 10-fold CV RMSE and R^2 values for the final model. DO NOT print out a table with 100 rows showing the results from the `train()` function. Instead, make a scatterplot of tuning parameter vs. CV RMSE that indicates the optimal tuning parameter value.

- According to the 10-fold CV model using forward stepwise selection, the linear regression model with the best RMSE is the model with 23 variables. The RMSE $= 0.748$ and $R^2 = 0.941$. The absorbency measures that are included in the model are wl1, wl3, wl13, wl16, wl18, wl19, wl23, wl26, wl28, wl29, wl30, wl31, wl34, wl36, wl38, wl39, wl41, wl50, wl52, wl80, wl92, wl97, and wl100.

```r
# setting the seed
set.seed(1234)
# the 10-fold CV model for forward stepwise selection
q1_fwd <- train(x = absorp_X, y = protein, method = "leapForward",
                trControl = trainControl(method = "cv", number = 10),
                tuneGrid = data.frame(nvmax = 1:100))
# plot of the number of parameters against the RMSE
q1_fwd$results %>%
  ggplot(aes(x = nvmax, y = RMSE)) +
  geom_point(col = "orange") +
  geom_line(col = "orange") +
  geom_errorbar(aes(ymin = RMSE - RMSESD/sqrt(10),
                    ymax = RMSE + RMSESD/sqrt(10)),
                width = 0.2, position = position_dodge(0.9),
                col = "orange")
```



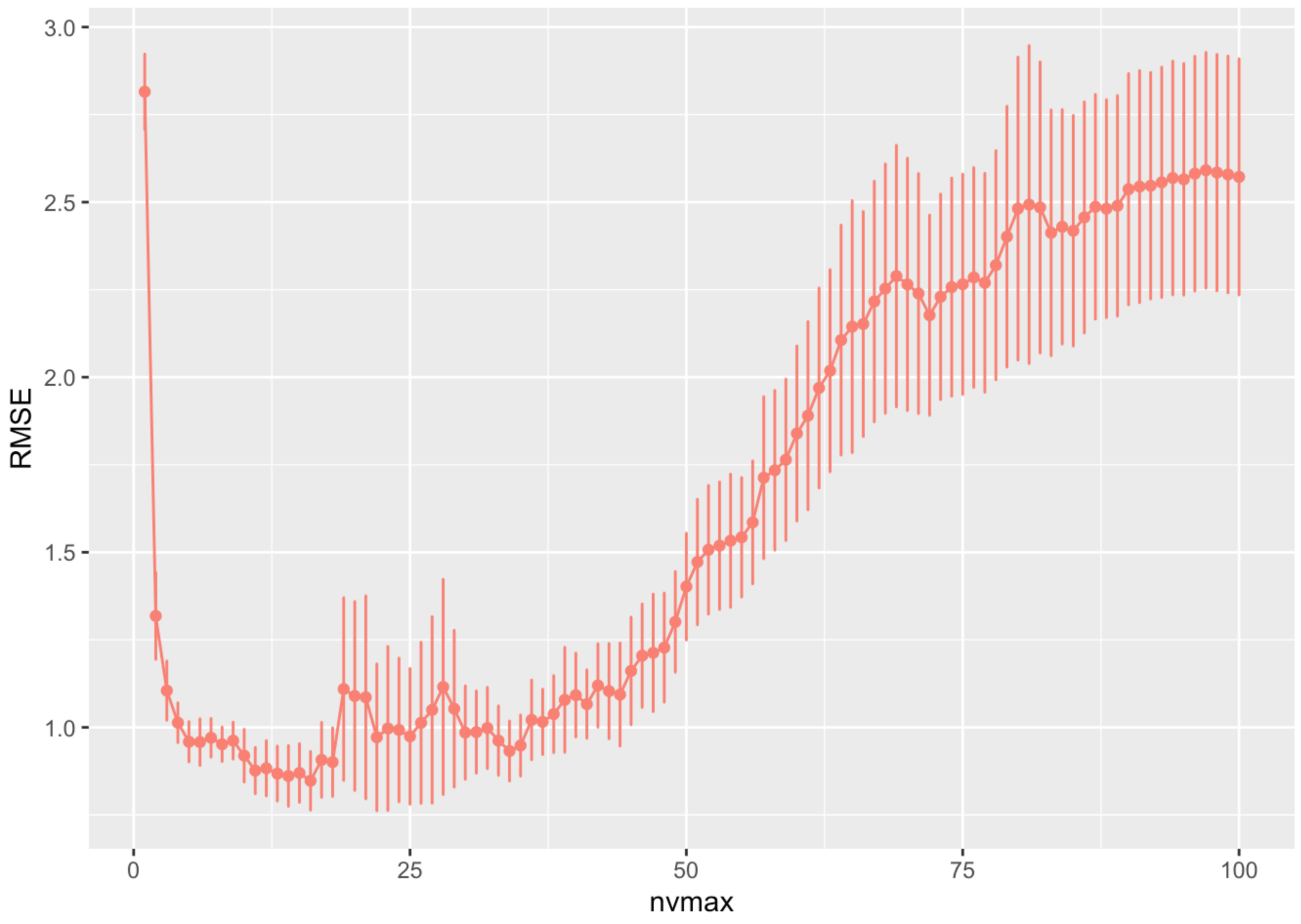
(d)

(14 pts.) Conduct backward stepwise selection for constructing a linear regression model to predict protein content using the absorbency measures. Use 10-fold CV RMSE to select your final model. Which absorbency measures are included in your final model? Report the 10-fold CV RMSE and R^2 values for the final model. DO NOT print out a table with 100 rows showing the results from the `train()` function. Instead, make a scatterplot of tuning parameter

vs. CV RMSE that indicates the optimal tuning parameter value. Is there any overlap in the set of variables selected using forward stepwise selection and those selected by backward stepwise selection? If so, what are the variables.

- According to the 10-fold CV model using backward stepwise selection, the linear regression model with the best RMSE is the model with 16 variables. The $\text{RMSE} = 0.847$ and $R^2 = 0.923$. The absorbency measures that are included in the model are wl1, wl3, wl17, wl18, wl32, wl37, wl40, wl44, wl48, wl57, wl68, wl72, wl75, wl87, and wl91. There are three variables that are selected to be in both the forward and backward models, which are wl1, wl3, and wl18.

```r
# setting the seed
set.seed(1234)
# the 10-fold CV model for forward stepwise selection
q1_bwd <- train(x = absorp_X, y = protein, method = "leapBackward",
                trControl = trainControl(method = "cv", number = 10),
                tuneGrid = data.frame(nvmax = 1:100))
# plot of the number of parameters against the RMSE
q1_bwd$results %>%
  ggplot(aes(x = nvmax, y = RMSE)) +
  geom_point(col = "salmon") +
  geom_line(col = "salmon") +
  geom_errorbar(aes(ymin = RMSE - RMSESD/sqrt(10),
                    ymax = RMSE + RMSESD/sqrt(10)),
                width = 0.2, position = position_dodge(0.9),
                col = "salmon")
```



```
#tail(summary(q1_bwd)$outmat)
#apply(summary(q1_bwd)$outmat, 2, function(x){1*I(sum(x != "")>0)})
```

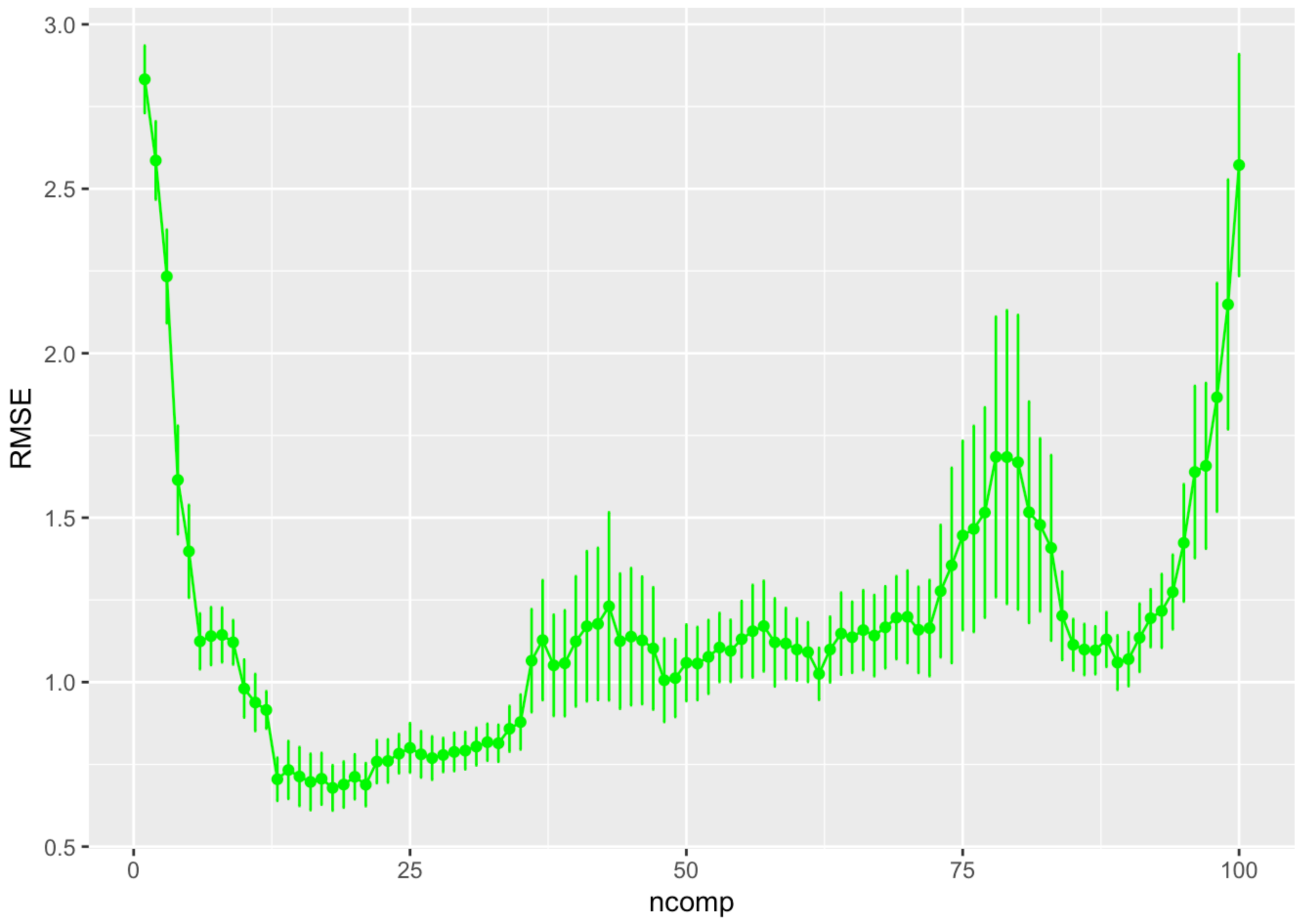
(e)

(12 pts.) Fit a principal components linear regression model using absorbency measures as the predictors and protein content as the response. Since all of the predictors are measured on the same scale, you do not need to scale them before constructing the principal components. Use 10-fold CV RMSE to select your final model. Report the 10-fold CV RMSE and R^2 values for the final model. DO NOT print out a table with 100 rows showing the results from the `train()` function. Instead, make a scatterplot of tuning parameter vs. CV RMSE that indicates the optimal tuning parameter value.

- According to the 10-fold CV model using principal components linear regression, the linear regression model with the best RMSE has 18 principal components. RMSE $= 0.678$ and $R^2 = 0.951$.

```
# setting the seed
set.seed(1234)
# setting the tuning grid
q1_tg <- data.frame(ncomp = 1:100)
# the 10-fold CV model for principal components linear regression
q1_pcr <- train(x = absorp_X, y = protein,
                method = "pcr",
```

```r
                 trControl = trainControl(method = "cv", number = 10),
                 tuneGrid = q1_tg)
# plot of the number of principal components against the RMSE
q1_pcr$results %>%
  ggplot(aes(x = ncomp, y = RMSE)) +
  geom_point(col = "green") +
  geom_line(col = "green") +
  geom_errorbar(aes(ymin = RMSE - RMSESD/sqrt(10),
                    ymax = RMSE + RMSESD/sqrt(10)),
                width = 0.2, position = position_dodge(0.9),
                col = "green")
```

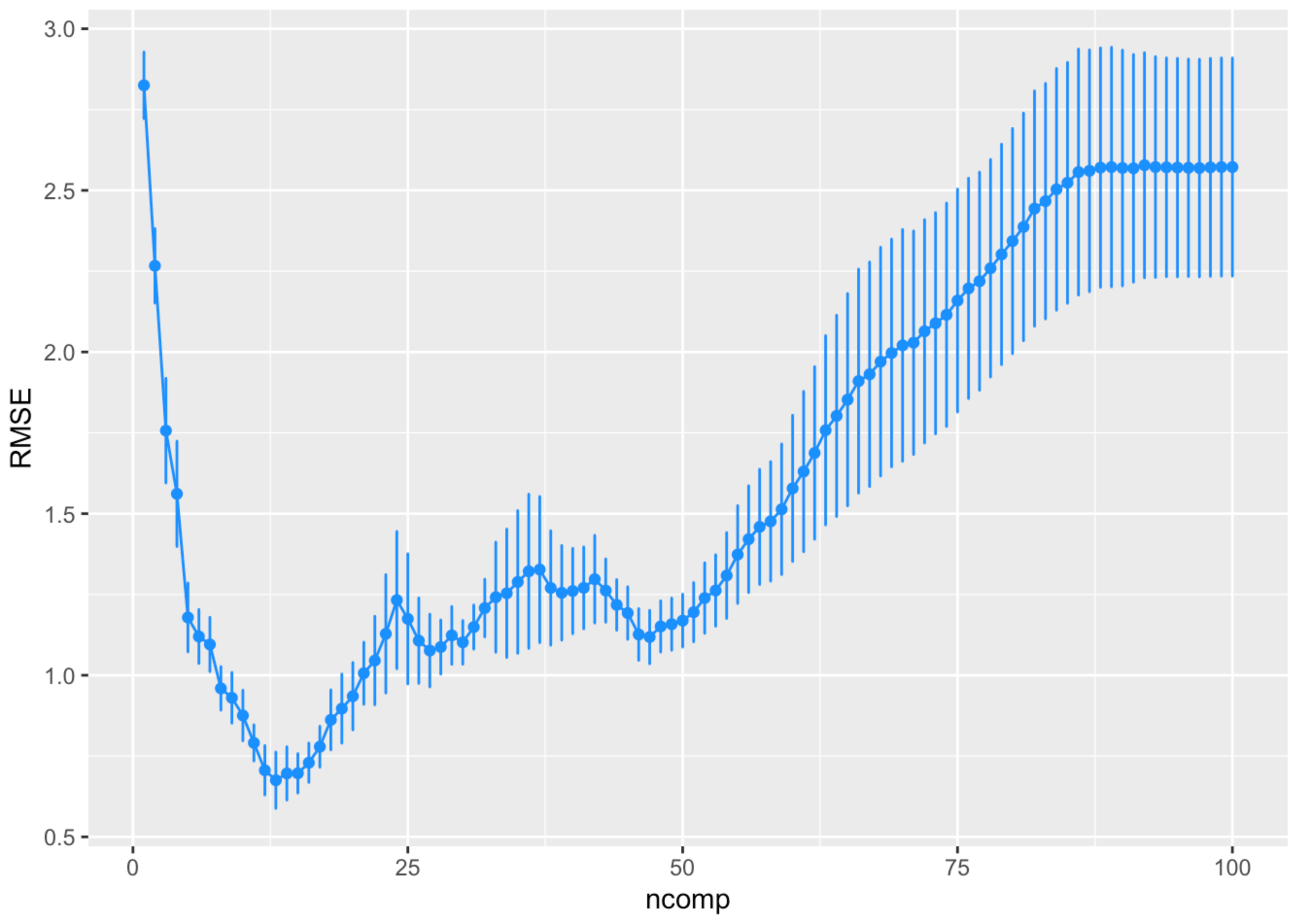


(f)

(12 pts.) Fit a partial least squares regression model using absorbency measures as the predictors and protein content as the response. Since all of the predictors are measured on the same scale, you do not need to scale them. Use 10-fold CV RMSE to select your final model. Report the 10-fold CV RMSE and R^2 values for the final model. DO NOT print out a table with 100 rows showing the results from the `train()` function. Instead, make a scatterplot of tuning parameter vs. CV RMSE that indicates the optimal tuning parameter value.

- According to the 10-fold CV model using partial least squares regression, the linear regression model with the best RMSE has 13 principal components. $\text{RMSE} = 0.675$ and $R^2 = 0.950$.

```
# setting the seed
set.seed(1234)
# the 10-fold CV model for principal components linear regression
q1_pls <- train(x = absorp_X, y = protein,
                method = "pls",
                trControl = trainControl(method = "cv", number = 10),
                tuneGrid = q1_tg)
# plot of the number of principal components against the RMSE
q1_pls$results %>%
  ggplot(aes(x = ncomp, y = RMSE)) +
  geom_point(col = "dodgerblue") +
  geom_line(col = "dodgerblue") +
  geom_errorbar(aes(ymin = RMSE - RMSESD/sqrt(10),
                    ymax = RMSE + RMSESD/sqrt(10)),
                width = 0.2, position = position_dodge(0.9),
                col = "dodgerblue")
```



(g)

(14 pts.) Fit a linear regression model with a ridge penalty using absorbency measures as the predictors and protein content as the response. Since all of the predictors are measured on the same scale, you do not need to scale them. Use the `glmnet()` function to obtain a grid of tuning parameter values to assess. Use 10-fold CV RMSE to select your

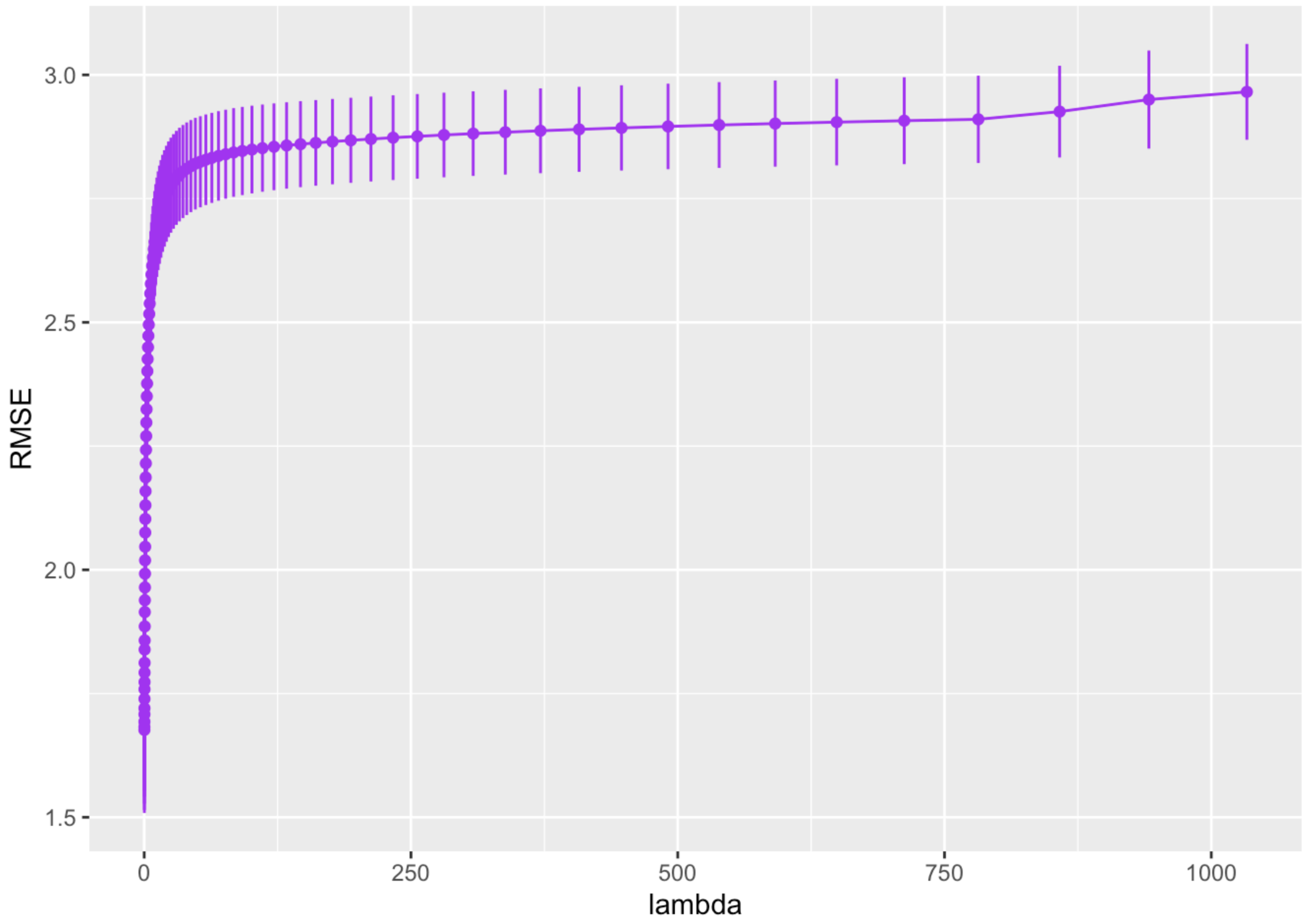
final model. Report the 10-fold CV RMSE and R^2 values for the final model. DO NOT print out a table with 100 rows showing the results from the `train()` function. Instead, make a scatterplot of tuning parameter vs. CV RMSE that indicates the optimal tuning parameter value.

- According to the 10-fold CV model using a linear regression model with a ridge penalty, the model with the best RMSE uses a tuning parameter of $\lambda = 0.1645$. The RMSE and R^2 values are 1.677 and 0.692 respectively. In this case, it didn't matter which function was used, both `cv.glmnet()` and `train()` demonstrated that the best model in this case has a lambda of 0.1645

```r
# setting the seed
set.seed(1234)
# using glmnet() to obtain a grid of tuning (lambda) parameters
q1_ridge <- glmnet(x = absorp_X, y = protein, alpha = 0)
# the 10-fold cv using cv.glmnet()
q1_cvRidge <- cv.glmnet(x = absorp_X, y = protein, alpha = 0, nfolds = 10)
# uses the lambdas from q1_cvRidge to create a tuning grid for train()
q1_tgRidge <- data.frame(alpha = 0, lambda = q1_ridge$lambda)
# the 10-fold cv using train()
q1_tcvRidge <- train(x = absorp_X, y = protein,
                     method = "glmnet", tuneGrid = q1_tgRidge,
                     trControl = trainControl(method = "cv", number = 10,
                                              selectionFunction = "best"))
```

```
Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo,
: There were missing values in resampled performance measures.
```

```r
q1_tcvRidge$results %>%
  ggplot(aes(x = lambda, y = RMSE)) +
  geom_point(col = "purple") +
  geom_line(col = "purple") +
  geom_errorbar(aes(ymin = RMSE - RMSESD/sqrt(10),
                    ymax = RMSE + RMSESD/sqrt(10)),
                width = 0.01, position = position_dodge(0.9),
                col = "purple")
```



(h)

(14 pts.) Fit a linear regression model with a LASSO penalty using absorbency measures as the predictors and protein content as the response. Since all of the predictors are measured on the same scale, you do not need to scale them. Use the `glmnet()` function to obtain a grid of tuning parameter values to assess. Use 10-fold CV RMSE to select your final model. Report the 10-fold CV RMSE and R^2 values for the final model. DO NOT print out a table with 100 rows showing the results from the `train()` function. Instead, make a scatterplot of tuning parameter vs. CV RMSE that indicates the optimal tuning parameter value.

- According to the 10-fold CV model using a linear regression model with a LASSO penalty, the model with the best RMSE uses a tuning parameter of $\lambda = 0.000964$, when using the `train()` function. The RMSE and R^2 values are 1.088 and 0.883 respectively. In this case, it didn't matter which function was used, both `cv.glmnet()` and `train()` demonstrated that the best model in this case has a lambda of 0.000964

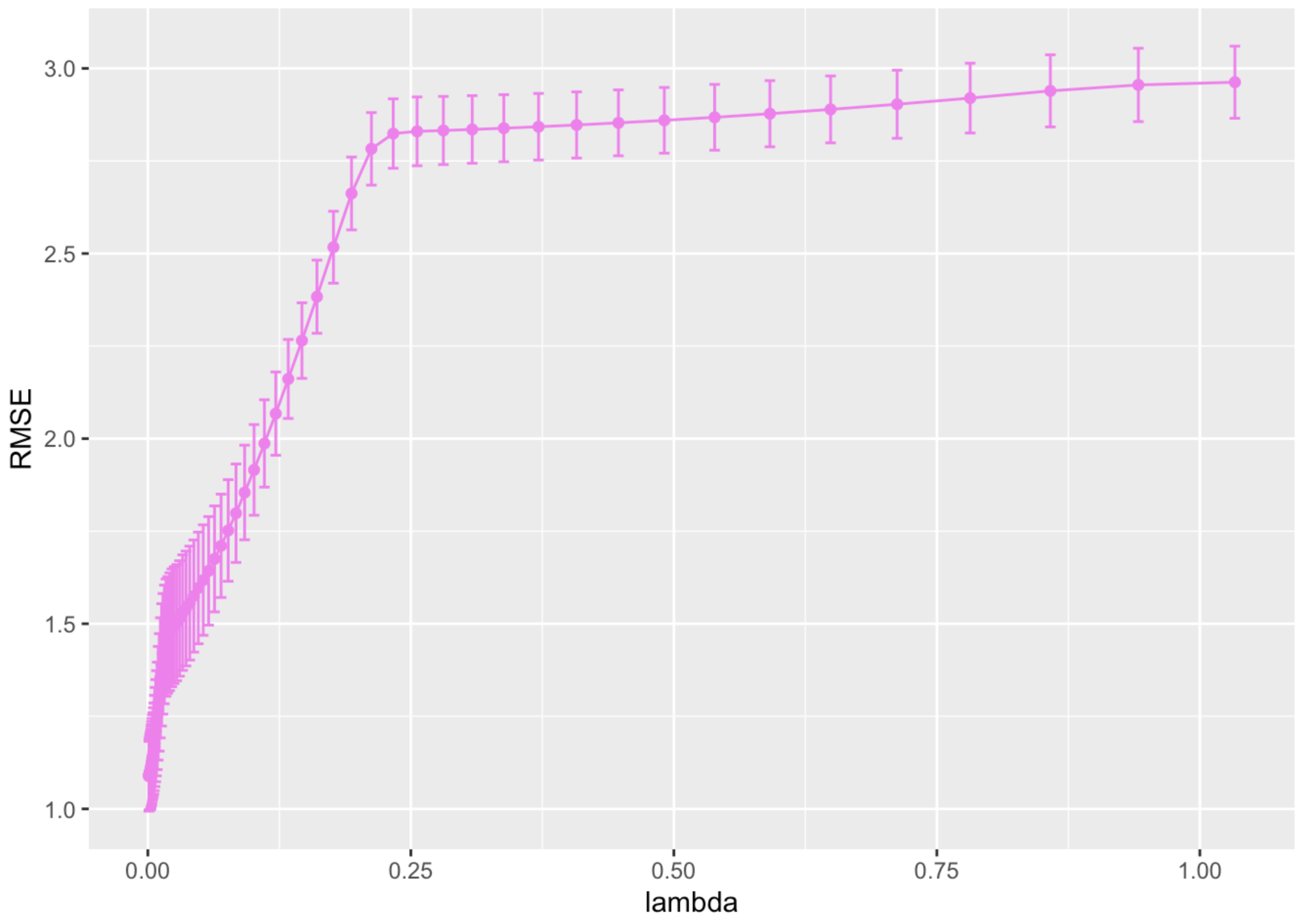
```
# setting the seed
set.seed(1234)
# using glmnet() to obtain a grid of tuning (lambda) parameters
q1_lasso <- glmnet(x = absorp_X, y = protein, alpha = 1)
# plot_glmnet(q1_ridge, xvar = "lambda")
# the 10-fold cv using cv.glmnet()
q1_cvLasso <- cv.glmnet(x = absorp_X, y = protein, alpha = 1, nfolds = 10)
# uses the lambdas from q1_cvLasso to create a tuning grid for train()
```

```r
q1_tgLasso <- data.frame(alpha = 1, lambda = q1_lasso$lambda)
# the 10-fold cv using train()
q1_tcvLasso <- train(x = absorp_X, y = protein,
                     method = "glmnet", tuneGrid = q1_tgLasso,
                     trControl = trainControl(method = "cv", number = 10,
                                              selectionFunction = "best"))
```

```
Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo,
: There were missing values in resampled performance measures.
```

```r
q1_tcvLasso$results %>%
  ggplot(aes(x = lambda, y = RMSE)) +
  geom_point(col = "violet") +
  geom_line(col = "violet") +
  geom_errorbar(aes(ymin = RMSE - RMSESD/sqrt(10),
                    ymax = RMSE + RMSESD/sqrt(10)),
                width = 0.01, position = position_dodge(0.9),
                col = "violet")
```

```
Warning: `position_dodge()` requires non-overlapping x intervals.
```



(i)

(6 pts.) Based on the plots of tuning parameter vs. CV RMSE for the ridge and LASSO regression models, why might you be uncertain if you have selected an optimal value for the tuning parameter in each case? What would be your next step to ensure that you can find the optimal parameter? (You do not have to carry this out in R, just explain how you would do it.)

- Using the plots of tuning parameter vs. CV RMSE for the ridge and LASSO regression models, it can be difficult to determine whether or not we actually picked the correct model because of how compressed the λ points are when λ is small. One step that can be taken is to zoom in on the on the portion of the graph that is compressed. This allows the λ's to be more easily differentiated from each other. Another step is to expand the tuning grid of λ 's towards zero. For both models, the λ chosen is the smallest one in the tuning grid. There might be a smaller value of λ that performs better than the one chosen.

(j)

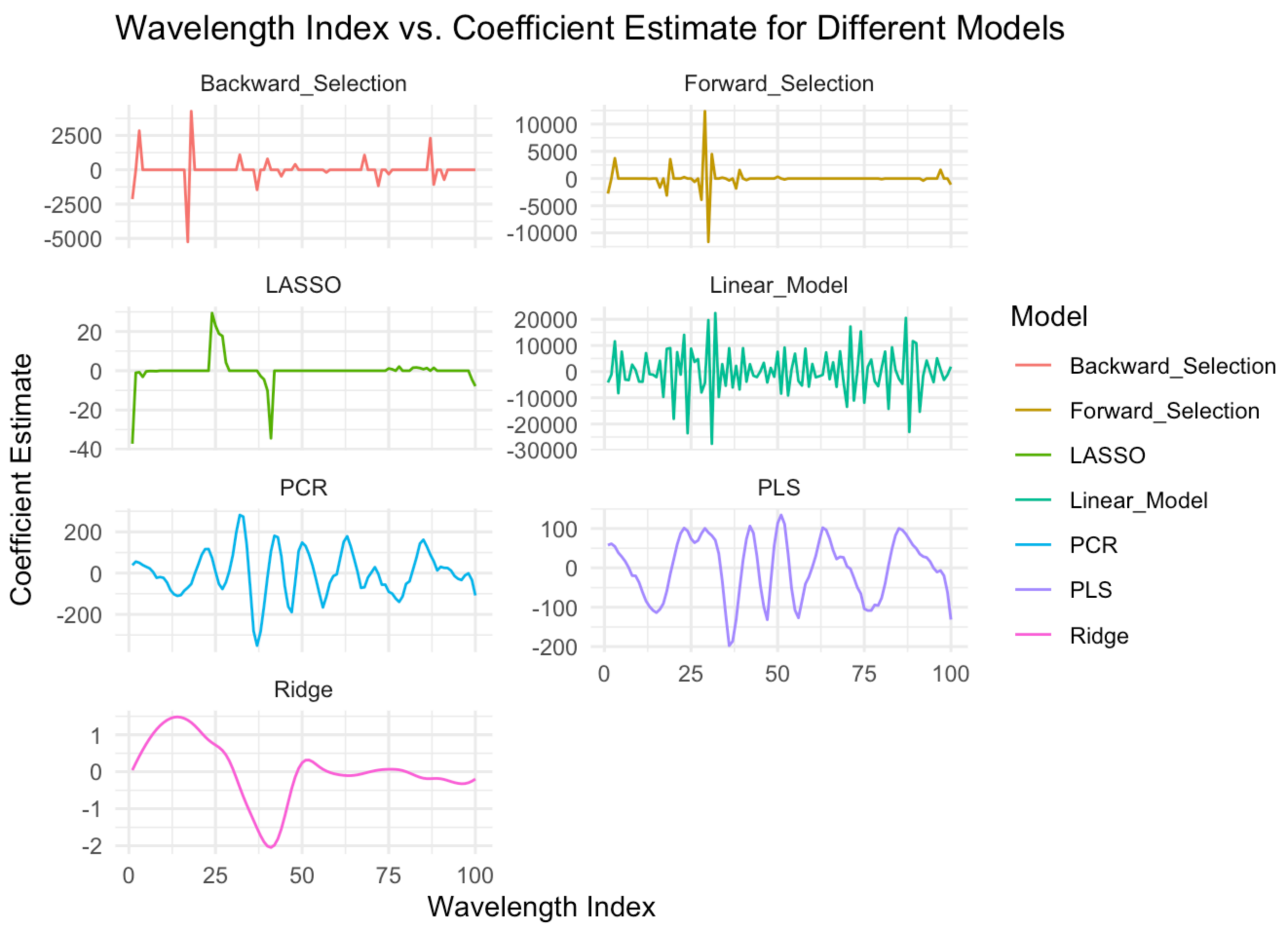
(10 pts.) In the parts above you have fit the following models: (1) linear model based on least squares estimation, (2) linear model based on forward stepwise selection, (3) linear model based on backward stepwise selection, (4) principal components linear model, (5) partial least squares model, (6) linear ridge regression model, and (7) linear LASSO regression model. Based on 10-fold CV RMSE, which model appears to be the best predictive model? Which is best if we consider 10-fold CV R^2? For comparison, make a plot of Wavelength Index vs. coefficient estimate for each of the 7 final models. Make 7 separate plots but put them in the same plotting region (i.e., there should be 7 separate plot/panels but in one figure).

- For models based on the 10-fold RMSE, the model that performs the best is the partial least squares regression model. It has the smallest RMSE value of 0.675

- For models based on the 10-fold CV R^2, the model that appears to be the best predictive model is the principal components linear regression model. It has the largest R^2 value of 0.951

```r
wavelength_ID <- 1:100
wl <- paste0("wl", 1:100)

# linear model based on least squares estimation
coef_lm <- q1_lm$finalModel$coefficients[-1]

# linear model based on forward stepwise selection
coef_fwd <- setNames(rep(0, length(wl)), wl)
c_fwd <- coef(q1_fwd$finalModel, id = q1_fwd$bestTune$nvmax)[-1]
coef_fwd[names(c_fwd)] <- c_fwd

# linear model based on backward stepwise selection
coef_bwd <- setNames(rep(0, length(wl)), wl)
c_bwd <- coef(q1_bwd$finalModel, id = q1_bwd$bestTune$nvmax)[-1]
coef_bwd[names(c_bwd)] <- c_bwd

# principal components linear model
coef_pcr <- coef(q1_pcr$finalModel, ncomp = q1_pcr$bestTune$ncomp)

# partial least squares model
```

```r
coef_pls <- coef(q1_pls$finalModel, ncomp = q1_pls$bestTune$ncomp)

# linear regression model with ridge penalty
coef_ridge <- coef(q1_ridge, s = q1_tcvRidge$bestTune$lambda)[-1]

#  linear regression model with LASSO penalty
coef_lasso <- coef(q1_lasso, s = q1_tcvLasso$bestTune$lambda)[-1]

plot_data <- data.frame(Wavelength_Index = wavelength_ID,
                            Linear_Model = coef_lm,
                            Forward_Selection = coef_fwd,
                            Backward_Selection = coef_bwd,
                            PCR = as.numeric(coef_pcr),
                            PLS = as.numeric(coef_pls),
                            Ridge = coef_ridge,
                            LASSO = coef_lasso)
plot_data <- plot_data %>%
  gather(key = "Model", value = "Coefficient", -Wavelength_Index)

ggplot(aes(x=Wavelength_Index, y=Coefficient, color = Model), data=plot_data) +
  geom_line() +
  facet_wrap(~ Model, ncol=2, scales = "free_y") +
  labs(title="Wavelength Index vs. Coefficient Estimate for Different Models",
       x="Wavelength Index",
       y="Coefficient Estimate") +
  theme_minimal()
```



Question Two

(44 pts.) The Core Temperature Dataset contains information on 7908 participants from a retrospective cohort study of intraoperative core temperature during colorectal surgery and its association with risk of surgical site infection. Detailed information about the data set and the data dictionary can be found at https://www.causeweb.org/tshs/core-temperature/. The data dictionary is also available on Blackboard (Core_Temperature_Data_Dictionary.pdf) along with the data set (Core_Temperature.csv). Suppose that you want to build a classifier to predict whether a patient develops any infection (`AnyInfection`) based on the following list of potential predictors: `Age, FEMALE, BMI, CharlsonScore, CHF, VALVE, DM, RENLFAIL, LIVER, METS, TUMOR, COAG, OBESE, WGHTLOSS, LYTES, BLDLOSS, ANEMDEF, DRUG, SteroidHx, ImmunosuppressantHx, SurgDuration, Open, TWATemp, LastReadingTemp, EndCaseTemp`

```r
# reads in data
q2Data <- read.csv("Core_Temperature.csv")
# the data with only the selected 25 predictors and outcome, factors outcome
coreTemp <- q2Data %>%
  select(c(Age, FEMALE, BMI, CharlsonScore, CHF, VALVE, DM, RENLFAIL, LIVER,
           METS, TUMOR, COAG, OBESE, WGHTLOSS, LYTES, BLDLOSS, ANEMDEF, DRUG,
           SteroidHx, ImmunosuppressantHx, SurgDuration, Open, TWATemp,
           LastReadingTemp, EndCaseTemp, AnyInfection)) %>%
  mutate(AnyInfection = factor(AnyInfection, levels = c(0, 1),
                               labels = c("No", "Yes")))
```

(a)

(10 pts.) Fit a logistic model via maximum likelihood estimation that uses all of the predictors listed above. Use 10-fold CV to select an optimal threshold for the classifier based on the best combination of sensitivity and specificity (i.e., the threshold yielding the largest sum of these values). Use possible threshold values of 0.10 to 0.90 in increments of 0.10. Report the threshold, sensitivity, specificity, and Cohen's κ values for the selected classifier.

- The optimal threshold for the classifier based on the best combination of sensitivity and specificity is a threshold of 0.9: Threshold = 0.9, Sensitivity = 0.372, Specificity = \$0.812 \$, and Cohen's $\kappa = 0.0738$

```
# setting the seed
set.seed(1234)
# the 10-fold CV of the logistic model
q2_glm <- train(x = coreTemp[,1:25],
                  y = coreTemp$AnyInfection,
                  method = "glm", family = binomial(link = "logit"),
                  trControl = trainControl(method = "cv", number = 10,
                                              classProbs = TRUE,
                                              savePredictions = TRUE))

# the sequence of threshold values
prob_thresh <- seq(0.1, 0.9, by = 0.1)
# use thresholder() function to obtain accuracy measures at each threshold,
#   using the q2_glm train object from above
(q2_glm_thresh <- thresholder(q2_glm,
                               threshold = prob_thresh,
                               final = TRUE,
                               statistics = c("Sensitivity", "Specificity",
                                              "Accuracy", "Kappa")))
```

```
  parameter prob_threshold Sensitivity Specificity  Accuracy         Kappa
1      none            0.1   1.0000000 0.000000000 0.8573599  0.0000000000
2      none            0.2   1.0000000 0.000000000 0.8573599  0.0000000000
3      none            0.3   0.9997050 0.000000000 0.8571069 -0.0005028117
4      none            0.4   0.9986726 0.001769912 0.8564747  0.0007563745
5      none            0.5   0.9961652 0.010619469 0.8555895  0.0112373886
6      none            0.6   0.9895280 0.035477244 0.8534409  0.0401058456
7      none            0.7   0.9625369 0.101082491 0.8396582  0.0889270589
8      none            0.8   0.8538348 0.327164981 0.7787083  0.1668929873
9      none            0.9   0.3718289 0.812104930 0.4346256  0.0737572249
```

```
best_glm <- numeric(9)
for (i in 1:9){
  best_glm[i] <- q2_glm_thresh$Sensitivity[i] + q2_glm_thresh$Specificity[i]
}
```

```
# finds the best model wrt to a combination of sensitivity and specificity
max(best_glm)
```

```
[1] 1.183934
```

```
(best_glm)
```

```
[1] 1.000000 1.000000 0.999705 1.000442 1.006785 1.025005 1.063619 1.181000
[9] 1.183934
```

(b)

(12 pts.) Fit a logistic model with a ridge penalty. Use 10-fold CV to select both an optimal tuning parameter and threshold for the classifier based on the best combination of sensitivity and specificity (i.e., the threshold yielding the largest sum of these values). Use possible threshold values of 0.10 to 0.90 in increments of 0.10. Report the threshold, sensitivity, specificity, and Cohen's κ values for the selected classifier.

- The optimal threshold for the classifier based on the best combination of sensitivity and specificity is a threshold of 0.8. The model chosen has $\lambda = 0.00739$, Threshold = 0.8, Sensitivity = 0.861, Specificity = 0.317, and Cohen's $\kappa = 0.167$

```
# setting the seed
set.seed(1234)
# need the predictors to be in a matrix
coreTempMat <- as.matrix(coreTemp[,1:25])
anyInfec <- coreTemp$AnyInfection
# using glmnet() to obtain a grid of tuning (lambda) parameters
q2_ridge <- glmnet(x = coreTempMat, y = anyInfec, alpha = 0,
                   family = "binomial")
# uses the lambdas from q2_ridge to create a tuning grid for train()
q2_tgRidge <- data.frame(alpha = 0, lambda = q2_ridge$lambda)
# the 10-fold cv using train()
q2_tcvRidge <- train(x = coreTemp, y = anyInfec,
                     method = "glmnet", family = "binomial",
                     tuneGrid = q2_tgRidge,
                     trControl = trainControl(method = "cv", number = 10,
                                              classProbs = TRUE,
                                              selectionFunction = "best",
                                              summaryFunction = twoClassSummary,
                                              savePredictions = TRUE))
```

```
Warning in train.default(x = coreTemp, y = anyInfec, method = "glmnet", : The
metric "Accuracy" was not in the result set. ROC will be used instead.

Warning in storage.mode(x) <- "double": NAs introduced by coercion

Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
```

```
    Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
    Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
    Warning in storage.mode(x) <- "double": NAs introduced by coercion

    Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
    Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
    Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
    Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

    Warning in storage.mode(x) <- "double": NAs introduced by coercion

    Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
    Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
    Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
    Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

    Warning in storage.mode(x) <- "double": NAs introduced by coercion

    Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
    Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
    Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
    Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

    Warning in storage.mode(x) <- "double": NAs introduced by coercion

    Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
    Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
    Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
    Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

    Warning in storage.mode(x) <- "double": NAs introduced by coercion

    Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
    Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
    Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
    Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

    Warning in storage.mode(x) <- "double": NAs introduced by coercion

    Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
    Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
    Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
    Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

    Warning in storage.mode(x) <- "double": NAs introduced by coercion

    Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
    Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
    Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
    Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
```

```
Warning in storage.mode(x) <- "double": NAs introduced by coercion

Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

Warning in storage.mode(x) <- "double": NAs introduced by coercion

Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

Warning in storage.mode(x) <- "double": NAs introduced by coercion
```

```r
# use thresholder() function to obtain accuracy measures at each threshold
(q2_ridge_thresh <- thresholder(q2_tcvRidge,
                               threshold = prob_thresh,
                               final = TRUE,
                               statistics = c("Sensitivity", "Specificity",
                                              "Kappa")))
```

```
  alpha      lambda prob_threshold Sensitivity Specificity         Kappa
1     0 0.007393328            0.1   1.0000000 0.000000000  0.0000000000
2     0 0.007393328            0.2   1.0000000 0.000000000  0.0000000000
3     0 0.007393328            0.3   0.9998525 0.000000000 -0.0002515554
4     0 0.007393328            0.4   0.9988201 0.001769912  0.0009983941
5     0 0.007393328            0.5   0.9973451 0.008849558  0.0103373221
6     0 0.007393328            0.6   0.9911504 0.027481037  0.0302064086
7     0 0.007393328            0.7   0.9660767 0.087776549  0.0768165236
8     0 0.007393328            0.8   0.8606195 0.316529709  0.1668002033
9     0 0.007393328            0.9   0.3528024 0.823625158  0.0692600582
```

```r
best_ridge <- numeric(9)
for (i in 1:9){
  best_ridge[i] <- q2_ridge_thresh$Sensitivity[i] + q2_ridge_thresh$Specificity[i]
}
# finds the best model wrt to a combination of sensitivity and specificity
max(best_ridge)
```

```
[1] 1.177149
```

```r
(best_ridge)
```

```
[1] 1.0000000 1.0000000 0.9998525 1.0005900 1.0061947 1.0186315 1.0538532
[8] 1.1771492 1.1764275
```

(c)

(12 pts) Fit a logistic model with a LASSO penalty. Use 10-fold CV to select both an optimal tuning parameter and threshold for the classifier based on the best combination of sensitivity and specificity (i.e., the threshold yielding the largest sum of these values). Use possible threshold values of 0.10 to 0.90 in increments of 0.10. Report the threshold, sensitivity, specificity, and Cohen's κ values for the selected classifier.

- The optimal threshold for the classifier based on the best combination of sensitivity and specificity is a threshold of 0.9. The model chosen has $\lambda = 0.00123$, Threshold = 0.9, Sensitivity = 0.354, Specificity = 0.824, and Cohen's $\kappa = 0.070$

```r
# setting the seed
set.seed(1234)
# using glmnet() to obtain a grid of tuning (lambda) parameters
q2_lasso <- glmnet(x = coreTempMat, y = anyInfec, alpha = 1,
                   family = "binomial")
# uses the lambdas from q2_ridge to create a tuning grid for train()
q2_tgLasso <- data.frame(alpha = 1, lambda = q2_lasso$lambda)
# the 10-fold cv using train()
q2_tcvLasso <- train(x = coreTemp, y = anyInfec,
                     method = "glmnet", family = "binomial",
                     tuneGrid = q2_tgLasso,
                     trControl = trainControl(method = "cv", number = 10,
                                              classProbs = TRUE,
                                              selectionFunction = "best",
                                              summaryFunction = twoClassSummary,
                                              savePredictions = TRUE))
```

```
Warning in train.default(x = coreTemp, y = anyInfec, method = "glmnet", : The
metric "Accuracy" was not in the result set. ROC will be used instead.

Warning in storage.mode(x) <- "double": NAs introduced by coercion

Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

Warning in storage.mode(x) <- "double": NAs introduced by coercion

Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

Warning in storage.mode(x) <- "double": NAs introduced by coercion

Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in storage.mode(x) <- "double": NAs introduced by coercion

Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

Warning in storage.mode(x) <- "double": NAs introduced by coercion

Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

Warning in storage.mode(x) <- "double": NAs introduced by coercion

Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

Warning in storage.mode(x) <- "double": NAs introduced by coercion

Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

Warning in storage.mode(x) <- "double": NAs introduced by coercion

Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

Warning in storage.mode(x) <- "double": NAs introduced by coercion

Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

Warning in storage.mode(x) <- "double": NAs introduced by coercion

Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

Warning in storage.mode(x) <- "double": NAs introduced by coercion
```

```r
set.seed(1234)
# use thresholder() function to obtain accuracy measures at each threshold
(q2_lasso_thresh <- thresholder(q2_tcvLasso,
                               threshold = prob_thresh,
                               final = TRUE,
                               statistics = c("Sensitivity", "Specificity",
                                              "Kappa")))
```

```
  alpha      lambda prob_threshold Sensitivity Specificity         Kappa
1     1 0.001233281            0.1   1.0000000 0.000000000  0.0000000000
2     1 0.001233281            0.2   1.0000000 0.000000000  0.0000000000
3     1 0.001233281            0.3   0.9998525 0.000000000 -0.0002515554
4     1 0.001233281            0.4   0.9988201 0.002654867  0.0024980898
5     1 0.001233281            0.5   0.9973451 0.008849558  0.0103538962
6     1 0.001233281            0.6   0.9908555 0.029258850  0.0325630646
7     1 0.001233281            0.7   0.9663717 0.090431416  0.0810420791
8     1 0.001233281            0.8   0.8610619 0.315628951  0.1666511316
9     1 0.001233281            0.9   0.3544248 0.823656764  0.0700055375
```

```r
best_lasso <- numeric(9)
for (i in 1:9){
  best_lasso[i] <- q2_lasso_thresh$Sensitivity[i] + q2_lasso_thresh$Specificity[i]
}
# finds the best model wrt to a combination of sensitivity and specificity
max(best_lasso)
```

```
[1] 1.178082
```

```r
(best_lasso)
```

```
[1] 1.0000000 1.0000000 0.9998525 1.0014749 1.0061947 1.0201143 1.0568031
[8] 1.1766909 1.1780815
```

(d)

(10 pts) Of the three final classifiers that you selected in parts (a) - (c) select the one that performs best with respect to the sum of the sensitivity and specificity and provide the relevant information that would be required for another person to use the selected classifier to predict the class for a new observation.

- The classifier that performs best with respect to the sum of the sensitivity and specificity is the logistic regression classification model fitted via maximum likelihood estimation with a threshold of 0.9. The coefficients of the logistic regression classification model are printed in the following code block.

```r
summary(q2_glm)
```

```

Call:
NULL
```

```
Coefficients:
                      Estimate Std. Error z value Pr(>|z|)    
(Intercept)         -4.5231059  3.2957453  -1.372  0.16994    
Age                 -0.0039223  0.0023354  -1.680  0.09305 .  
FEMALE              -0.0129289  0.0673842  -0.192  0.84785    
BMI                  0.0332369  0.0066692   4.984 6.24e-07 ***
CharlsonScore        0.0904930  0.0465490   1.944  0.05189 .  
CHF                  0.0711794  0.1895383   0.376  0.70726    
VALVE               -0.0951073  0.1523254  -0.624  0.53239    
DM                  -0.0503185  0.1212615  -0.415  0.67817    
RENLFAIL             0.0084440  0.1877765   0.045  0.96413    
LIVER               -0.3433301  0.2147002  -1.599  0.10980    
METS                -0.8174095  0.3096360  -2.640  0.00829 ** 
TUMOR               -0.2234694  0.1306126  -1.711  0.08709 .  
COAG                 0.2342107  0.1170302   2.001  0.04536 *  
OBESE               -0.2522048  0.1097116  -2.299  0.02152 *  
WGHTLOSS             0.6156853  0.0793206   7.762 8.36e-15 ***
LYTES                0.2411846  0.0756377   3.189  0.00143 ** 
BLDLOSS              0.0201048  0.2078181   0.097  0.92293    
ANEMDEF             -0.0897703  0.0809061  -1.110  0.26719    
DRUG                 0.2746696  0.2041438   1.345  0.17847    
SteroidHx           -0.0036260  0.0725142  -0.050  0.96012    
ImmunosuppressantHx  0.0162036  0.1880812   0.086  0.93135    
SurgDuration         0.0034523  0.0003052  11.311  < 2e-16 ***
Open                 0.1267133  0.0916697   1.382  0.16689    
TWATemp              0.1657218  0.0697484   2.376  0.01750 *  
LastReadingTemp     -0.0245768  0.0172255  -1.427  0.15365    
EndCaseTemp         -0.1152344  0.0854147  -1.349  0.17730    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 6480.3  on 7907  degrees of freedom
Residual deviance: 6140.3  on 7882  degrees of freedom
AIC: 6192.3

Number of Fisher Scoring iterations: 5
```