Data 624 - HW7 (Fall 2024)

Khyati Naik

6.2.

Developing a model to predict permeability (see Sect. 1.4) could save significant resources for a pharmaceutical company, while at the same time more rapidly identifying molecules that have a sufficient permeability to become a drug:

(a) Start R and use these commands to load the data: The matrix fingerprints contains the 1,107 binary molecular predictors for the 165 compounds, while permeability contains permeability response.

```
library(AppliedPredictiveModeling)
library(caret)
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
library(caTools)
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v lubridate 1.9.2     v tibble    3.2.1
## v purrr     1.0.2     v tidyr     1.3.0

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x purrr::lift()   masks caret::lift()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(pls)
```

```
##
## Attaching package: 'pls'
##
## The following object is masked from 'package:corrplot':
##
##     corrplot
##
## The following object is masked from 'package:caret':
##
##     R2
##
## The following object is masked from 'package:stats':
##
##     loadings
```

```
library(arm)
```

```
## Loading required package: MASS
##
## Attaching package: 'MASS'
##
## The following object is masked from 'package:dplyr':
##
##     select
##
## Loading required package: Matrix
##
## Attaching package: 'Matrix'
##
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
##
## Loading required package: lme4
##
## arm (Version 1.14-4, built: 2024-4-1)
##
## Working directory is K:/khyati/cuny/624/hw7
##
##
## Attaching package: 'arm'
##
## The following objects are masked from 'package:pls':
##
##     coefplot, corrplot
##
## The following object is masked from 'package:corrplot':
##
##     corrplot
```

```
library(lars)
```

```
## Loaded lars 1.3
```

```
library(elasticnet)
library(RANN)
```

```
data(permeability)
```

(b) The fingerprint predictors indicate the presence or absence of substructures of a molecule and are often sparse meaning that relatively few of the molecules contain each substructure. Filter out the predictors that have low frequencies using the nearZeroVar function from the caret package. How many predictors are left for modeling?

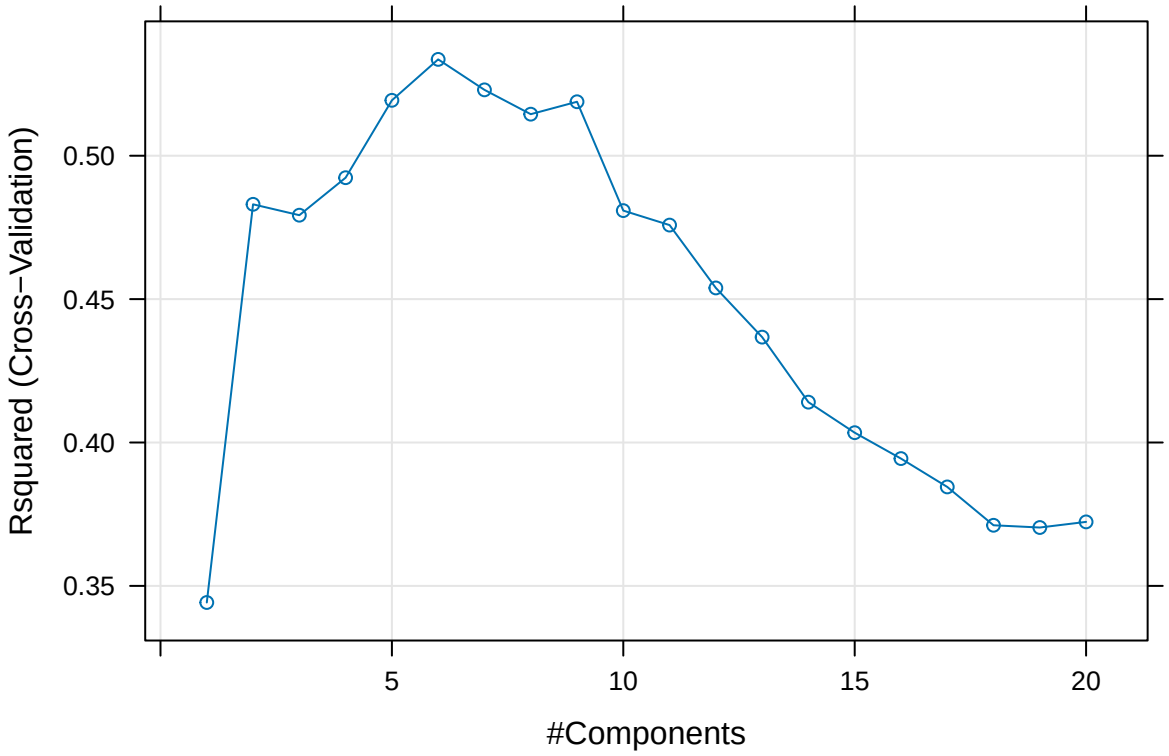
```
### Step b: Filter Near Zero Variance Predictors
nzv_indices <- nearZeroVar(fingerprints)  # Identify predictors with near-zero variance
filtered_fingerprints <- fingerprints[, -nzv_indices]  # Remove those predictors
remaining_predictors_count <- ncol(filtered_fingerprints)  # Count remaining predictors
remaining_predictors_count
```

```
## [1] 388
```

Initially, there were 1107 predictors; after filtering, 388 predictors remain.

(c) Split the data into a training and a test set, pre-process the data, and tune a PLS model. How many latent variables are optimal and what is the corresponding resampled estimate of R2?

```
### Step c: Data Splitting
set.seed(123)  # Ensure reproducibility
partition_index <- createDataPartition(permeability, p = .8, list = FALSE)  # Create an index for 80% t

# Split the data into training and testing sets
train_fingerprints <- filtered_fingerprints[partition_index, ]  # Training set for fingerprints
test_fingerprints <- filtered_fingerprints[-partition_index, ]  # Testing set for fingerprints
train_permeability <- permeability[partition_index]  # Training set for permeability
test_permeability <- permeability[-partition_index]  # Testing set for permeability

# Set up 10-fold cross-validation
cv_control <- trainControl(method = "cv", number = 10)

# Train a Partial Least Squares (PLS) model
pls_model <- train(train_fingerprints, train_permeability, method = "pls",
                   metric = "Rsquared", tuneLength = 20, trControl = cv_control,
                   preProc = c("center", "scale"))

# Plot the results of the PLS model
plot(pls_model)
```




```
pls_model  # Display the PLS model results
```

```
## Partial Least Squares
##
## 133 samples
## 388 predictors
##
## Pre-processing: centered (388), scaled (388)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 121, 121, 118, 119, 119, 119, ...
## Resampling results across tuning parameters:
##
##   ncomp  RMSE      Rsquared   MAE
##    1     13.31894  0.3442124  10.254018
##    2     11.78898  0.4830504   8.534741
##    3     11.98818  0.4792649   9.219285
##    4     12.04349  0.4923322   9.448926
##    5     11.79823  0.5193195   9.049121
##    6     11.53275  0.5335956   8.658301
##    7     11.64053  0.5229621   8.878265
##    8     11.86459  0.5144801   9.265252
##    9     11.98385  0.5188205   9.218594
##   10     12.55634  0.4808614   9.610747
##   11     12.69674  0.4758068   9.702325
##   12     13.01534  0.4538906   9.956623
```

```
##   13     13.12637  0.4367362   9.878017
##   14     13.44865  0.4140715  10.065088
##   15     13.60135  0.4034269  10.188150
##   16     13.79361  0.3943904  10.247160
##   17     14.00756  0.3845119  10.412776
##   18     14.18113  0.3711378  10.587027
##   19     14.25674  0.3703610  10.575726
##   20     14.33121  0.3723176  10.679764
##
## Rsquared was used to select the optimal model using the largest value.
## The final value used for the model was ncomp = 6.
```

Based on the graph and results, the optimal number of components is determined to be 6, yielding an R-Squared value of 0.5335.

(d) Predict the response for the test set. What is the test set estimate of R2?

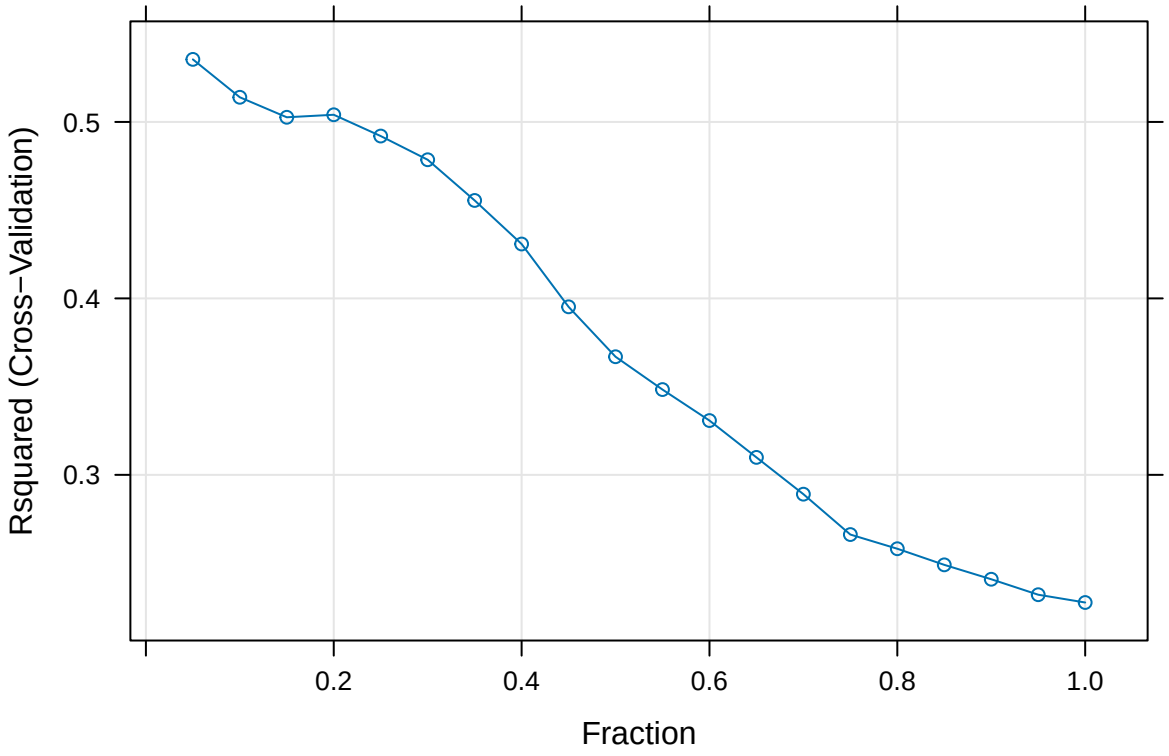
```
### Step d: Make Predictions
predicted_values <- predict(pls_model, test_fingerprints)  # Generate predictions on the test set

# Combine observed and predicted values in a new data frame with correct column names
results_comparison <- data.frame(obs = test_permeability, pred = predicted_values)

# Calculate prediction accuracy metrics using the default summary function
prediction_accuracy <- defaultSummary(results_comparison)

# Display the prediction accuracy metrics
prediction_accuracy
```

```
##       RMSE   Rsquared        MAE
## 12.3486900  0.3244542  8.2881075
```

The R-Squared value for the test set is found to be 0.3244.

(e) Try building other models discussed in this chapter. Do any have better predictive performance?

```
### Step e: Explore Alternative Models
# Train a Least Angle Regression (LARS) model
set.seed(123)
lars_model <- train(train_fingerprints, train_permeability, method = "lars", metric = "Rsquared",
                    tuneLength = 20, trControl = cv_control, preProc = c("center", "scale"))
plot(lars_model)  # Visualize LARS model results
```

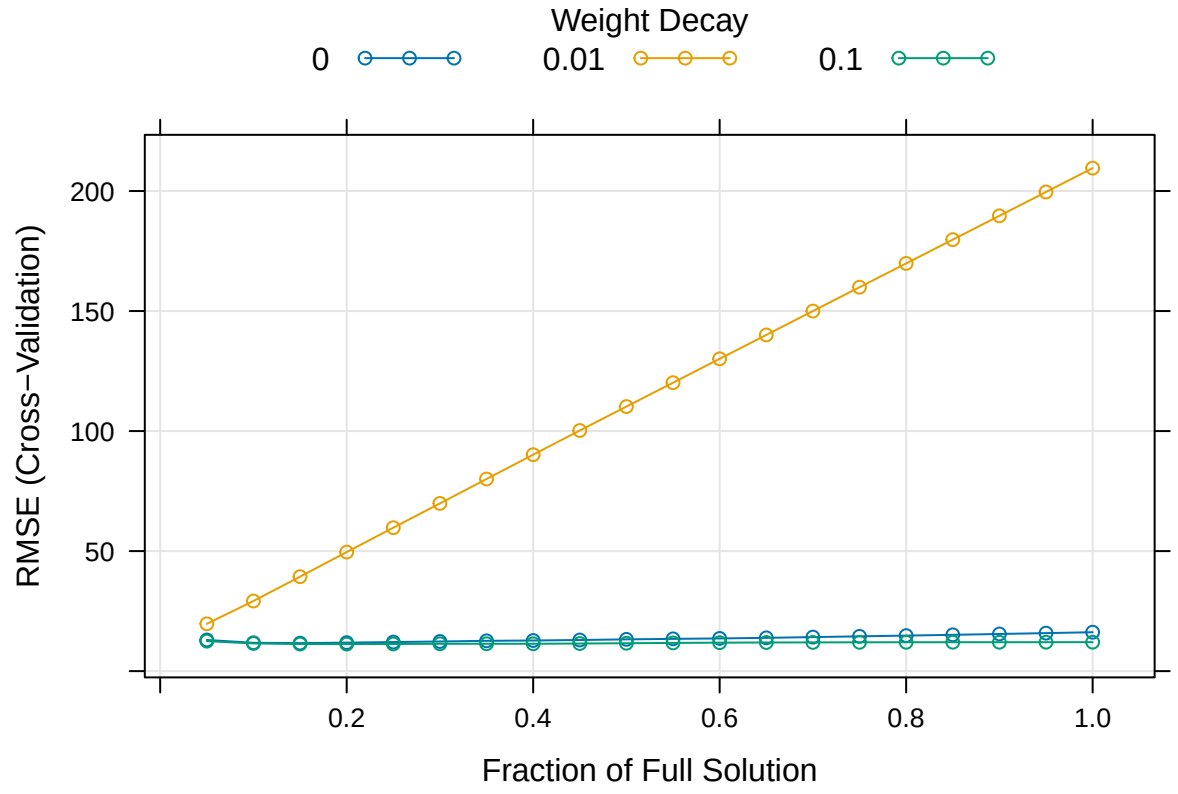



```
lars_model  # Display LARS model results
```

```
## Least Angle Regression
##
## 133 samples
## 388 predictors
##
## Pre-processing: centered (388), scaled (388)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 120, 119, 118, 120, 121, 119, ...
## Resampling results across tuning parameters:
##
##   fraction  RMSE      Rsquared   MAE
##   0.05      11.50137  0.5354619   8.621150
##   0.10      11.80187  0.5139834   8.889320
##   0.15      12.15728  0.5026509   9.179633
##   0.20      12.33026  0.5040708   9.133236
##   0.25      12.77929  0.4920045   9.422442
##   0.30      13.24694  0.4785748   9.649433
##   0.35      13.87708  0.4555026   9.958535
##   0.40      14.53772  0.4308134  10.225755
##   0.45      15.59410  0.3952284  10.941873
##   0.50      16.82173  0.3669388  11.881953
##   0.55      17.90121  0.3483293  12.631532
##   0.60      19.13031  0.3307779  13.468422
```

```
##   0.65      20.32695  0.3099028  14.137917
##   0.70      21.59298  0.2890390  14.904013
##   0.75      22.96601  0.2661699  15.633892
##   0.80      23.97961  0.2581207  16.108147
##   0.85      24.98966  0.2489990  16.596507
##   0.90      26.05609  0.2408344  17.148103
##   0.95      27.10684  0.2320677  17.734636
##   1.00      28.08270  0.2276614  18.268258
##
## Rsquared was used to select the optimal model using the largest value.
## The final value used for the model was fraction = 0.05.
```

```
# Set up for Elastic Net model training
set.seed(123)
enet_parameter_grid <- expand.grid(.lambda = c(0, 0.01, .1), .fraction = seq(.05, 1, length = 20))  # D
enet_model <- train(train_fingerprints, train_permeability, method = "enet",
                    tuneGrid = enet_parameter_grid, trControl = cv_control, preProc = c("center", "scal
plot(enet_model)  # Visualize Elastic Net model results
```




```
enet_model  # Display Elastic Net model results
```

```
## Elasticnet
##
## 133 samples
## 388 predictors
```

```
##
## Pre-processing: centered (388), scaled (388)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 120, 119, 118, 120, 121, 119, ...
## Resampling results across tuning parameters:
##
##   lambda  fraction  RMSE       Rsquared   MAE
##   0.00    0.05       12.97787  0.4301959    9.877733
##   0.00    0.10       11.80785  0.4931852    9.107986
##   0.00    0.15       11.66393  0.4930871    8.897928
##   0.00    0.20       11.86820  0.4793179    9.030145
##   0.00    0.25       12.10947  0.4728912    9.253825
##   0.00    0.30       12.34485  0.4658690    9.393513
##   0.00    0.35       12.61952  0.4590835    9.612879
##   0.00    0.40       12.75663  0.4601371    9.686790
##   0.00    0.45       12.96171  0.4539654    9.774599
##   0.00    0.50       13.22572  0.4468568    9.900789
##   0.00    0.55       13.44341  0.4427435    9.994948
##   0.00    0.60       13.62322  0.4378770   10.071188
##   0.00    0.65       13.86382  0.4299241   10.153834
##   0.00    0.70       14.18080  0.4200728   10.374171
##   0.00    0.75       14.48298  0.4084205   10.465235
##   0.00    0.80       14.79949  0.3962019   10.559980
##   0.00    0.85       15.11740  0.3847540   10.634954
##   0.00    0.90       15.47863  0.3758087   10.864504
##   0.00    0.95       15.82572  0.3675399   11.039900
##   0.00    1.00       16.20635  0.3601753   11.220501
##   0.01    0.05       19.70369  0.5419408   13.658257
##   0.01    0.10       29.20414  0.5517070   18.690554
##   0.01    0.15       39.34872  0.5464403   24.503744
##   0.01    0.20       49.58867  0.5300114   30.253422
##   0.01    0.25       59.75378  0.5135458   35.825185
##   0.01    0.30       69.86658  0.5025866   41.376372
##   0.01    0.35       80.04013  0.4907761   46.959417
##   0.01    0.40       90.16964  0.4820232   52.431432
##   0.01    0.45      100.22907  0.4769870   57.896985
##   0.01    0.50      110.20374  0.4733163   63.339107
##   0.01    0.55      120.15283  0.4689652   68.748714
##   0.01    0.60      130.10820  0.4634918   74.138916
##   0.01    0.65      140.04834  0.4574078   79.532014
##   0.01    0.70      149.98967  0.4508075   84.943512
##   0.01    0.75      159.92961  0.4435334   90.354687
##   0.01    0.80      169.84077  0.4381988   95.756154
##   0.01    0.85      179.74379  0.4319106  101.148939
##   0.01    0.90      189.66445  0.4256649  106.576223
##   0.01    0.95      199.59443  0.4201282  112.076637
##   0.01    1.00      209.53137  0.4142206  117.588754
##   0.10    0.05       12.48366  0.5107258    9.539035
##   0.10    0.10       11.53534  0.5261893    8.482600
##   0.10    0.15       11.27266  0.5429020    8.204349
##   0.10    0.20       11.27554  0.5488762    8.346509
##   0.10    0.25       11.30648  0.5527622    8.491996
##   0.10    0.30       11.39070  0.5510568    8.624229
##   0.10    0.35       11.39403  0.5533536    8.686883
```

```
##   0.10    0.40       11.39420  0.5565505    8.705934
##   0.10    0.45       11.50017  0.5532039    8.805005
##   0.10    0.50       11.62477  0.5502964    8.891474
##   0.10    0.55       11.75005  0.5467109    8.971633
##   0.10    0.60       11.85638  0.5433706    9.039529
##   0.10    0.65       11.92754  0.5414808    9.064147
##   0.10    0.70       11.97002  0.5408816    9.061763
##   0.10    0.75       12.00539  0.5408888    9.064088
##   0.10    0.80       12.02698  0.5416069    9.059826
##   0.10    0.85       12.03704  0.5428213    9.068134
##   0.10    0.90       12.04706  0.5438642    9.086377
##   0.10    0.95       12.05637  0.5446054    9.098177
##   0.10    1.00       12.06264  0.5453869    9.100052
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were fraction = 0.15 and lambda = 0.1.
```

Upon reviewing the LARS and Elastic Net models, the optimal settings for LARS yield a fraction of 0.05 with an R-Squared value of 0.5354, while the Elastic Net model shows an optimal lambda of 0.1 and fraction of 0.15 with an R-Squared value of 0.5429. Both alternative models underperformed relative to the PLS model.

(f) Would you recommend any of your models to replace the permeability laboratory experiment?

I do not recommend using any of the models tested to replace the permeability laboratory experiment. Instead, I plan to investigate *XGBoost* or *Support Vector Machines (SVM)* to see if they can achieve a higher R-Squared and lower RMSE and MAE. I believe these methods can more effectively manage the larger number of components or predictors compared to those previously used. In particular, SVM may be more beneficial when the number of predictors exceeds the number of samples, making it a potentially more robust option.

6.3

A chemical manufacturing process for a pharmaceutical product was discussed in Sect. 1.4. In this problem, the objective is to understand the relationship between biological measurements of the raw materials (predictors), measurements of the manufacturing process (predictors), and the response of product yield. Biological predictors cannot be changed but can be used to assess the quality of the raw material before processing. On the other hand, manufacturing process predictors can be changed in the manufacturing process. Improving product yield by 1 % will boost revenue by approximately one hundred thousand dollars per batch:

(a) Start R and use these commands to load the data: The matrix processPredictors contains the 57 predictors: Predictors contains the 57 predictors (12 describing the input biological material and 45 describing the process predictors) for the 176 manufacturing runs. yield contains the percent yield for each run.

```
data(ChemicalManufacturingProcess)
```

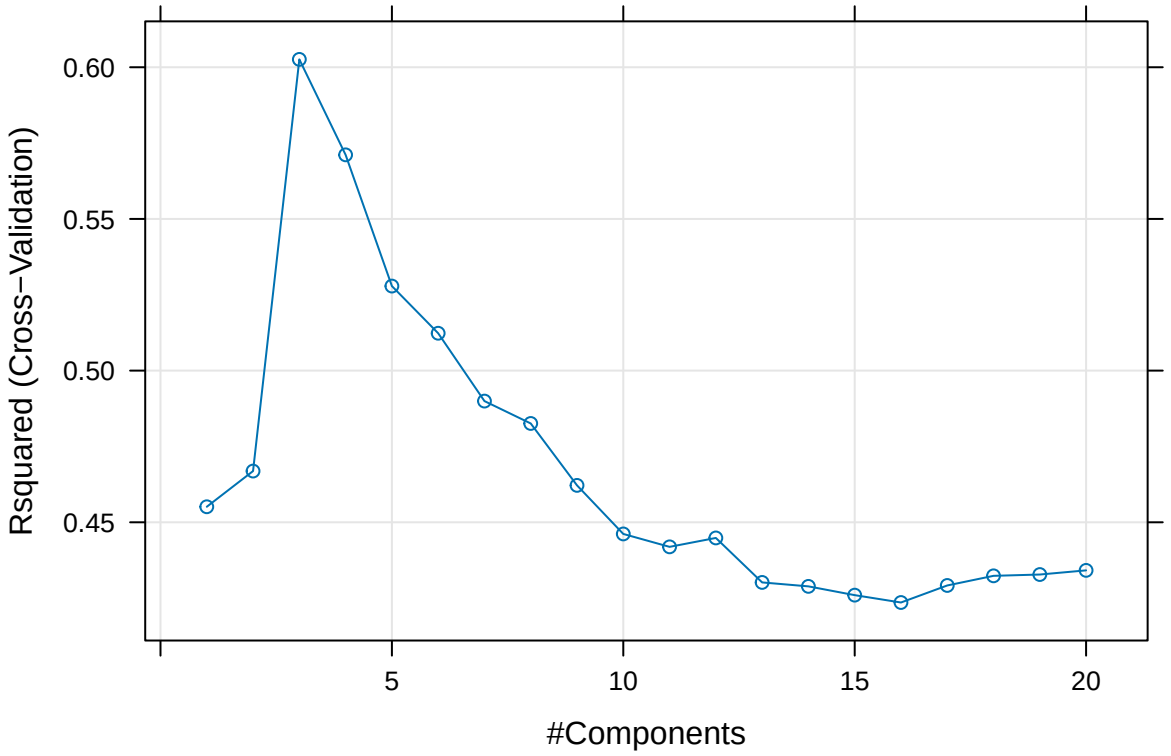
(b) A small percentage of cells in the predictor set contain missing values. Use an imputation function to fill in these missing values (e.g., see Sect. 3.8).

```
# Check for missing values in the dataset
missing_values_count <- sum(is.na(ChemicalManufacturingProcess))

# Impute missing values using K-Nearest Neighbors (KNN)
imputer <- preProcess(ChemicalManufacturingProcess, method = "knnImpute")
imputed_data <- predict(imputer, ChemicalManufacturingProcess)

# Check if any missing values remain after imputation
remaining_missing_count <- sum(is.na(imputed_data))
```

KNN imputation transformed 106 missing values into their imputed counterparts. KNN was chosen due to the biological nature of the dataset, which often displays easier density functions.Biological data usually reveals patterns where similar observations yield similar outcomes. Instead of discarding the missing data, I chose to retain it to avoid losing potential relationships within the dataset, especially given the small sample size.

(c) Split the data into a training and a test set, pre-process the data, and tune a model of your choice from this chapter. What is the optimal value of the performance metric?

```
### Step c: Filter Variables and Split Data
imputed_data <- imputed_data[, -nearZeroVar(imputed_data)]  # Remove near-zero variance predictors
features <- dplyr::select(imputed_data, -Yield)  # Select predictors excluding the target variable
target <- imputed_data$Yield  # Define the target variable

set.seed(123)  # For reproducibility
train_index <- createDataPartition(target, p = .8, list = FALSE)  # Create a training set index

# Split the dataset into training and testing sets
training_features <- features[train_index, ] %>% as.matrix()  # Training features
testing_features <- features[-train_index, ] %>% as.matrix()   # Testing features
training_target <- target[train_index]  # Training target
testing_target <- target[-train_index]   # Testing target

# Set up cross-validation control with 10 folds
control_settings <- trainControl(method = "cv", number = 10)
# Train a Partial Least Squares (PLS) regression model
pls_model <- train(x = training_features, y = training_target, method = "pls",
                   metric = "Rsquared", tuneLength = 20, trControl = control_settings,
                   preProc = c("center", "scale"))
plot(pls_model)  # Plot model performance
```




```
# Display the model results
pls_model$results
```

```
##    ncomp      RMSE  Rsquared       MAE    RMSESD RsquaredSD     MAESD
## 1      1 0.7754057 0.4551107 0.6289419 0.2064820  0.2080301 0.1677273
## 2      2 1.0635795 0.4668878 0.6697724 0.8321828  0.2676093 0.2979132
## 3      3 0.6606301 0.6025962 0.5369693 0.1960452  0.1861347 0.1430776
## 4      4 0.8085067 0.5711244 0.5714930 0.5623388  0.2232811 0.1673670
## 5      5 1.0981659 0.5278682 0.6569408 1.2414162  0.2592831 0.3470588
## 6      6 1.1435873 0.5123137 0.6780413 1.3033184  0.2693839 0.3669206
## 7      7 1.3703141 0.4899506 0.7582540 1.8495620  0.2793116 0.5077298
## 8      8 1.5909598 0.4825770 0.8308428 2.4304549  0.2850208 0.6751223
## 9      9 1.8112121 0.4621626 0.8994732 2.9010201  0.2898199 0.8044436
## 10    10 2.1241168 0.4461615 0.9920989 3.6488540  0.3000363 1.0088230
## 11    11 2.4415219 0.4419015 1.0860689 4.5161603  0.2921561 1.2512727
## 12    12 2.5920306 0.4448361 1.1286992 4.8529929  0.2834402 1.3412468
## 13    13 2.7741817 0.4301744 1.1886044 5.0474725  0.2986369 1.3984718
## 14    14 2.8885256 0.4288722 1.2222728 5.1932459  0.3045492 1.4432394
## 15    15 2.9286665 0.4259796 1.2337177 5.3155858  0.3036742 1.4752137
## 16    16 2.9819282 0.4235582 1.2454071 5.3986261  0.3028103 1.4904750
## 17    17 2.9768463 0.4291762 1.2388964 5.3700104  0.3040684 1.4836888
## 18    18 3.0097692 0.4323490 1.2477385 5.4427249  0.3042285 1.5061909
## 19    19 3.0756863 0.4327794 1.2680481 5.5807034  0.3048951 1.5488670
## 20    20 3.1017228 0.4341464 1.2705427 5.6564787  0.3057286 1.5581539
```

The optimal number of components for PLS regression was found to be 3, achieving an R-Squared value of

0.6025.

(d) Predict the response for the test set. What is the value of the performance metric and how does this compare with the resampled performance metric on the training set?

```
### Step d: Make Predictions
predicted_yield <- predict(pls_model, testing_features)  # Generate predictions on the test set
results_summary <- data.frame(obs = testing_target, pred = predicted_yield)  # Combine observed and pre

# Calculate prediction accuracy metrics
prediction_metrics <- defaultSummary(results_summary)
```

We chose PLS regression due to its superior performance in a previous analysis, but the lower R-Squared value on the resampled data compared to the training set indicates that the model may require further tuning.

(e) Which predictors are most important in the model you have trained? Do either the biological or process predictors dominate the list?

```
### Step e: Identify Top Predictors
# Assess variable importance from the model
importance_plot <- varImp(pls_model, scale = FALSE)

importance_data <- data.frame(
  Predictor = rownames(importance_plot$importance),
  Importance_Score = importance_plot$importance$Overall
)

# Sort the importance scores in descending order
sorted_importance <- importance_data[order(-importance_data$Importance_Score), ]

# Extract the top 10 predictors
top_predictors <- head(sorted_importance$Predictor, 10)
top_predictors
```

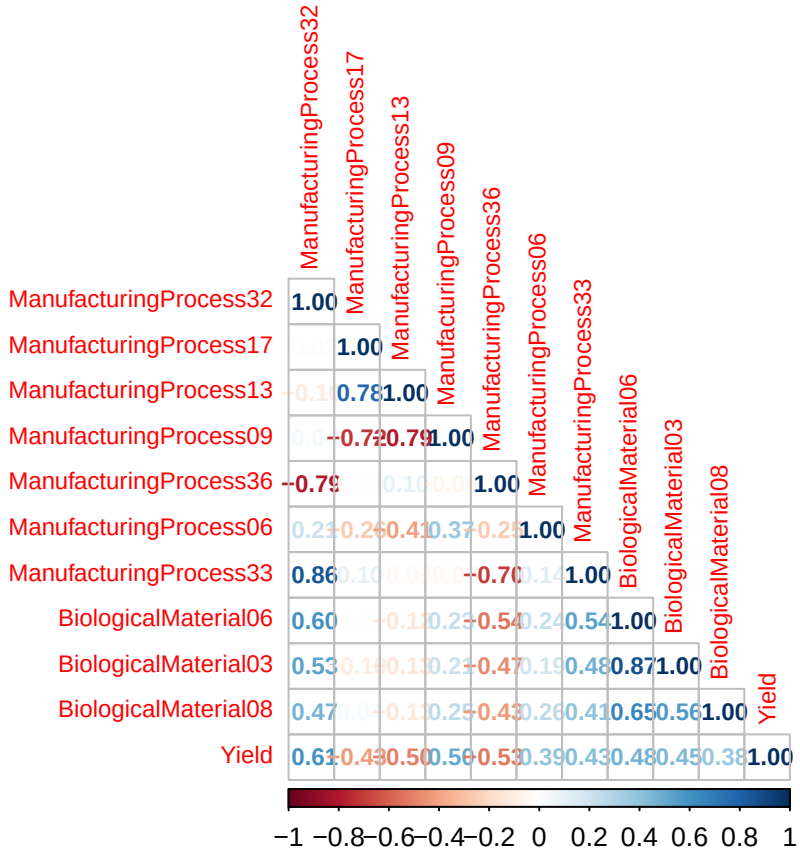
```
##  [1] "ManufacturingProcess32" "ManufacturingProcess17" "ManufacturingProcess13"
##  [4] "ManufacturingProcess09" "ManufacturingProcess36" "ManufacturingProcess06"
##  [7] "ManufacturingProcess33" "BiologicalMaterial06"   "BiologicalMaterial03"
## [10] "BiologicalMaterial08"
```

```
# Create a new dataframe with the top predictors and yield
top_predictors_df <- dplyr::select(features, all_of(top_predictors))
top_predictors_df$Yield <- target
```

In the top 10 predictors, manufacturing process variables dominate, with 7 out of 10 being process-related and only 3 from biological materials.

(f) Explore the relationships between each of the top predictors and the response. How could this information be helpful in improving yield in future runs of the manufacturing process?

```
### Step f: Correlation Analysis
# Calculate correlation between numeric variables in the top predictors
correlation_data <- cor(dplyr::select_if(top_predictors_df, is.numeric), use = "complete.obs")
# Create a correlation plot
corrplot::corrplot(correlation_data, method = 'number', type = 'lower', number.cex = 0.75, tl.cex= 0.75)
```



The correlation analysis reveals interesting insights: Three negative correlations with yield were found among manufacturing processes, with Manufacturing Process 36 showing the most significant negative impact. In contrast, Manufacturing Process 32 exhibited the highest positive correlation at 0.61. All biological material predictors positively correlated with yield, indicating their relevance to yield outcomes. The correlation among predictors is noteworthy, with all biological materials showing positive relationships with each other. The lowest negative correlation between predictors was -0.79, observed between Manufacturing Process 36 and Manufacturing Process 32, as well as between Manufacturing Process 13 and Manufacturing Process 9.