DATA 624 Fall 2017: Homework 2

Dan Smilowitz

December 12, 2017

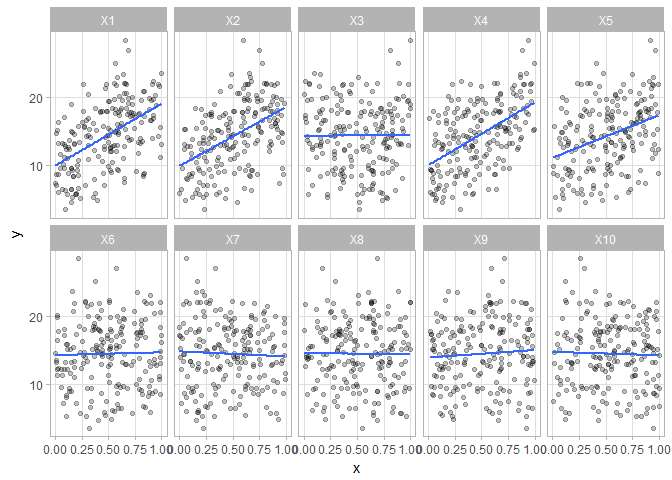
# KJ 7.2

Testing and training data is simulated as in the text:

library(mlbench)  
set.seed(200)  
# training data set  
fried\_train <- mlbench.friedman1(200, sd = 1)  
fried\_train$x <- data.frame(fried\_train$x)  
# test data set  
fried\_test <- mlbench.friedman1(5000, sd = 1)  
fried\_test$x <- data.frame(fried\_test$x)

The relationship between the predictors X1-X10 and the response y is investigated for the training data:

library(tidyverse)  
theme\_set(theme\_light())  
  
fried\_train$x %>%   
 # merge predictors & response into single df  
 mutate(y = fried\_train$y) %>%   
 # tidy data frame for easier manipulation & plotting  
 gather(var, x, -y) %>%   
 # factor x variables & change factors so X10 is last (rather than after X1)  
 mutate(var = forcats::fct\_relevel(factor(var), "X10", after = Inf)) %>%   
 # plot  
 ggplot(aes(x, y)) +  
 geom\_point(alpha = 0.25) +  
 stat\_smooth(method = "glm", se = FALSE) +  
 facet\_wrap(~ var, nrow = 2) +  
 theme(panel.grid.minor = element\_blank())



The variables X1, X2, x4, & X5 show positive correlations with the response y. The remaining variables appear to show near-zero correlation to the response.

## Model Training

Three models are tested: the k nearest neighbors model suggested by the test, as well as MARS and SVM models

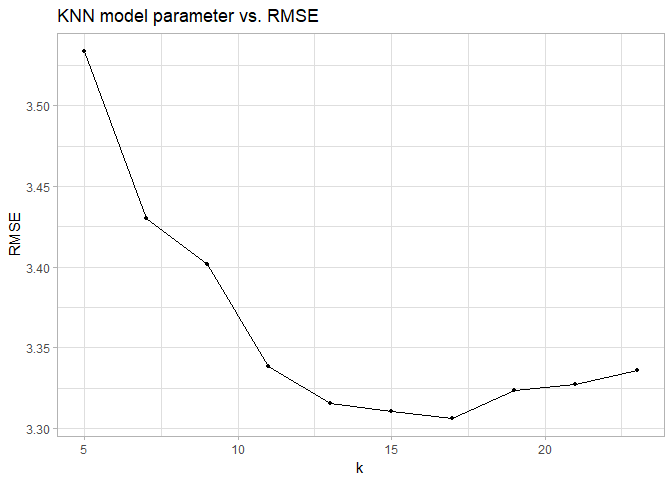
### K Nearest Neighbors Model

The k nearest neighbors model from the text is used as the first model, with predictors centered and scaled:

library(caret)  
set.seed(100) # for replicability  
fried\_knn <- train(fried\_train$x, fried\_train$y, method = "knn",  
 preProc = c("center", "scale"), tuneLength = 10)

The resampled RMSE of the model with various tuning parameters is shown below:

fried\_knn$results %>%   
 ggplot(aes(x = k, y = RMSE)) +  
 geom\_line() + geom\_point(size = 1) +  
 labs(title = "KNN model parameter vs. RMSE")



The optimal RMSE of this model occurs at , which yields an RMSE of 0.053.

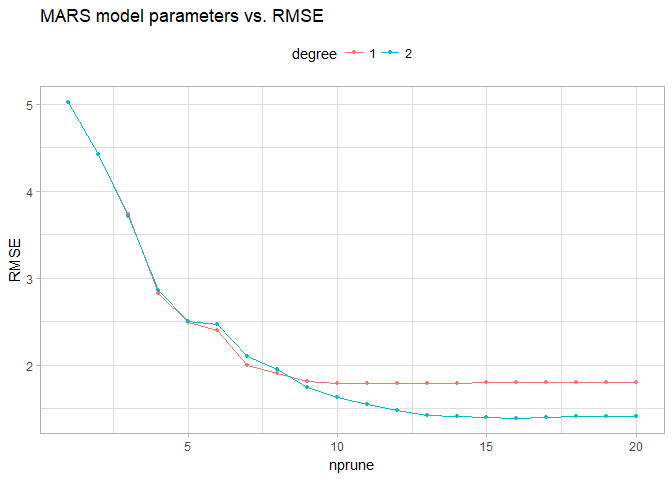
### MARS Model

A MARS model is also trained, using the same pre-processing as the KNN model above, for first- and second-degree products and pruning parameter from 1-20:

set.seed(100) # for replicability  
fried\_mars <- train(fried\_train$x, fried\_train$y, method = "earth",  
 preProc = c("center", "scale"),  
 tuneGrid = expand.grid(degree = 1:2, nprune = 1:20))

The tuning profile of this model is shown below:

fried\_mars$results %>%   
 ggplot(aes(x = nprune, y = RMSE, col = factor(degree))) +  
 geom\_line() + geom\_point(size = 1) +  
 labs(title = "MARS model parameters vs. RMSE", col = "degree") +  
 theme(legend.position = "top")



The optimal RMSE of 1.395 is achieved with a second-degree model with 16 parameters. The importance of the predictors is shown below:

|  |  |
| --- | --- |
|  | Overall |
| **X1** | 100 |
| **X4** | 85.12 |
| **X2** | 69.2 |
| **X5** | 49.23 |
| **X3** | 39.89 |
| **X6** | 0 |
| **X7** | 0 |
| **X8** | 0 |
| **X9** | 0 |
| **X10** | 0 |

It can be seen above that the five informative predictors X1-X5 are the only predictors selected by the model.

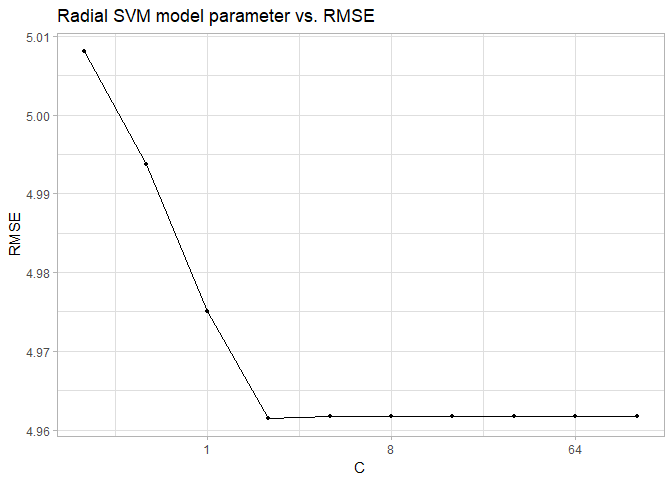
### SVM Model

An SVM model is fit using a radial kernel:

set.seed(100) # for replicability  
fried\_svm <- train(fried\_train$x, fried\_train$y, method = "svmRadial",  
 preProc = c("center", "scale"), tuneLength = 10)

The tuning profile of the model is shown below:

fried\_svm$results %>%   
 ggplot(aes(x = C, y = RMSE)) +  
 geom\_line() + geom\_point(size = 1) +  
 scale\_x\_continuous(trans = "log2") +  
 labs(title = "Radial SVM model parameter vs. RMSE")



The optimal RMSE of 4.961 achieved with a cost parameter and a shape parameter . The RMSE of the model levels out above .

## Model Performance & Selection

Each of the models is used to predict for the test set, and their performance is measured:

fried\_knn\_pred <- predict(fried\_knn, fried\_test$x)  
fried\_mars\_pred <- predict(fried\_mars, fried\_test$x)  
fried\_svm\_pred <- predict(fried\_svm, fried\_test$x)

fried\_knn\_perf <- defaultSummary(data.frame(obs = fried\_test$y, pred = fried\_knn\_pred))  
fried\_mars\_perf <- defaultSummary(data.frame(obs = fried\_test$y, pred = fried\_mars\_pred[, 1]))  
fried\_svm\_perf <- defaultSummary(data.frame(obs = fried\_test$y, pred = fried\_svm\_pred))  
  
library(pander)  
pander(data.frame(RMSE = c(fried\_knn\_perf["RMSE"], fried\_mars\_perf["RMSE"], fried\_svm\_perf["RMSE"]), row.names = c("KNN", "MARS", "SVM")))

|  |  |
| --- | --- |
|  | RMSE |
| **KNN** | 3.204 |
| **MARS** | 1.279 |
| **SVM** | 4.863 |

The most accurate model, based on both the resampled RMSE and performance against the test set, is the **MARS model**.

# KJ 7.5

data("ChemicalManufacturingProcess", package = "AppliedPredictiveModeling")  
# split into predictors & response  
chem\_pred <- ChemicalManufacturingProcess %>% select(-Yield)  
chem\_yield <- ChemicalManufacturingProcess %>% select(Yield)

As in problem 6.3, due to skewness of predictors, the data is preprocessed by performing k-nearest neighbor imputation, centering, scaling, with highly-correlated and near-zero variance predictors removed. The data is also split into training and test sets:

# set up pre-processing transformation  
chem\_preproc <- preProcess(chem\_pred, method = c("knnImpute", "center", "scale",  
 "nzv", "corr"))  
# apply pre-processing to data  
chem\_pred\_trans <- predict(chem\_preproc, chem\_pred)  
# get rows for training subsets  
set.seed(42)  
train\_rows <- createDataPartition(chem\_yield$Yield, p = 0.75, list = FALSE)  
# create training sets  
chem\_pred\_train <- chem\_pred\_trans[train\_rows, ]  
chem\_yield\_train <- chem\_yield[train\_rows, ]  
# creae test sets  
chem\_pred\_test <- chem\_pred\_trans[-train\_rows, ]  
chem\_yield\_test <- chem\_yield[-train\_rows, ]

## Model Training & Performance

For consistency, the same training controls are used as in Problem 6.3:

# use 15-fold cross-validation for training  
set.seed(100)  
chem\_ctrl <- trainControl(method = "cv", number = 15)

For completion, each of the four model types illustrated in Chapter 7 are trained.

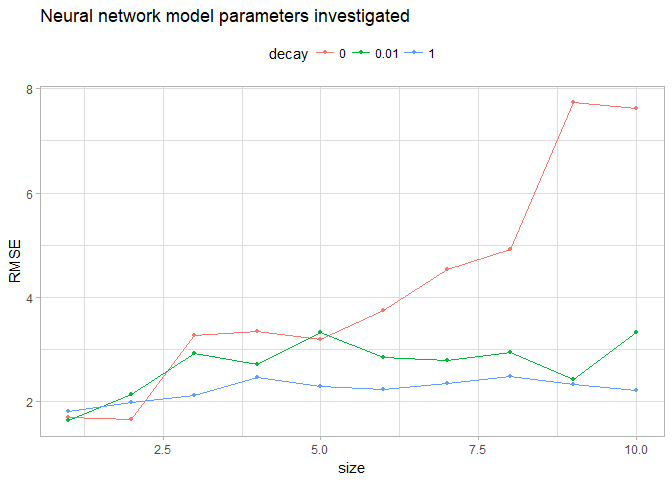
### Neural Network Model

A neural network model is fit to the data with 1-10 hidden layers and possible decays of 0, 0.01, or 1:

set.seed(100)  
chem\_nn <- train(chem\_pred\_train, chem\_yield\_train,  
 method = "nnet", trControl = chem\_ctrl,  
 linout = TRUE, trace = FALSE, maxit = 500,  
 tuneGrid = expand.grid(decay = c(0, 0.01, 1), size = 1:10))

The RMSE of the 30 models tested are presented below.

chem\_nn$results %>%  
 ggplot(aes(x = size, y = RMSE, col = factor(decay))) +  
 geom\_line() + geom\_point(size = 1) +  
 labs(title = "Neural network model parameters investigated", col = "decay") +  
 theme(legend.position = "top")



The plot shows an initial low RMSE value for , which continues to rise as the number of hidden layers increases. Meanwhile remains fairly constant, while appears to have an optimal value at a size of 7. If there were a desire to expand the model to include more hidden layers, these other values of should be considered.

The optimal resampled RMSE is 1.632, which occurs with 2 hidden layers and no decay (i.e. ). The model is also tested against the test set of yield values:

chem\_nn\_pred <- predict(chem\_nn, chem\_pred\_test)  
chem\_nn\_perf <- defaultSummary(data.frame(obs = chem\_yield\_test, pred = chem\_nn\_pred[, 1]))

The RMSE of predictions against the test set of these values is 1.604.

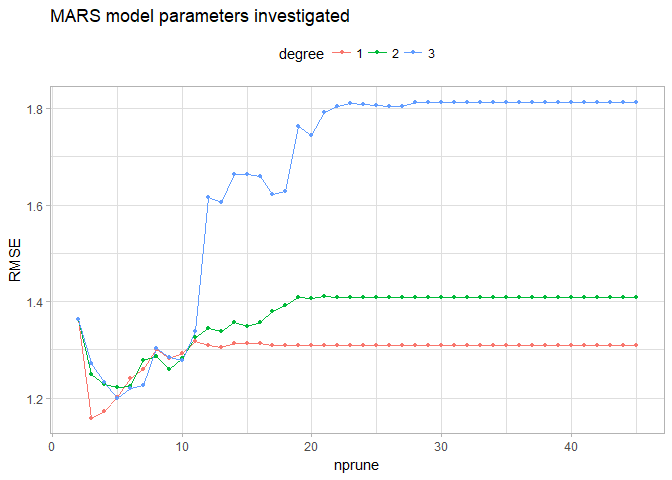
### MARS Model

A multivariate adaptive regression splines model is also fit to the data. Terms of degree 1-3 are investigated with pruning parameters ranging from 2-45 (one fewer than the number of predictors):

set.seed(100)  
chem\_mars <- train(chem\_pred\_train, chem\_yield\_train,  
 method = "earth", trControl = chem\_ctrl,  
 tuneGrid = expand.grid(degree = 1:3, nprune = 2:45))

The candidate models investigated are shown below:

chem\_mars$results %>%  
 ggplot(aes(x = nprune, y = RMSE, col = factor(degree))) +  
 geom\_line() + geom\_point(size = 1) +  
 labs(title = "MARS model parameters investigated", col = "degree") +  
 theme(legend.position = "top")



The optimal resampled RMSE is 1.158, which occurs with a first-degree equation with 2 terms.

The four lowest RMSE values returned were for first-degree fits with 1-4 terms. The four next-lowest RMSEs were returned by the second- and third-degree models with 4-5 terms. If an expanded model is important enough to sacrifice some accuracy, these models could be selected.

Since the purpose of this investigation is to attain the highest-accuracy model, the optimal model identified above is used. This model is used to predict the withheld values and compared for accuracy against the test set of outcomes:

chem\_mars\_pred <- predict(chem\_mars, chem\_pred\_test)  
chem\_mars\_perf <- defaultSummary(data.frame(obs = chem\_yield\_test, pred = chem\_mars\_pred[, 1]))

The test performance of this model yields an RMSE of 1.162 -- this represents an improvement over the performance of the neural network model.

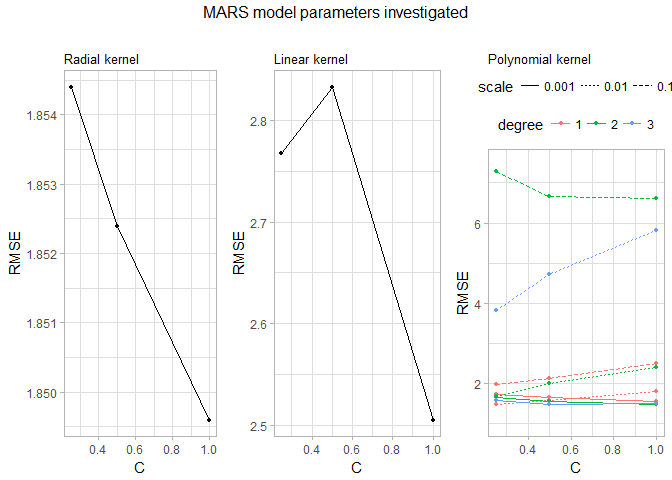
### Support Vector Machine Models

Three support vector machine models are evaluated -- one for each kernel covered in the text:

# radial kernel with analytically calculated sigma  
set.seed(100)  
chem\_svm\_r <- train(chem\_pred\_train, chem\_yield\_train,  
 method = "svmRadial", trControl = chem\_ctrl)  
# polynomial kernel with analytically calculated sigma  
set.seed(100)  
chem\_svm\_p <- train(chem\_pred\_train, chem\_yield\_train,  
 method = "svmPoly", trControl = chem\_ctrl)  
# linear kernel with analytically calculated sigma  
set.seed(100)  
chem\_svm\_l <- train(chem\_pred\_train, chem\_yield\_train,  
 method = "svmLinear", trControl = chem\_ctrl,  
 tuneGrid = data.frame(C = c(0.25, 0.5, 1)))

The candidate models for each of these kernels is investigated below:

gridExtra::grid.arrange(top = "MARS model parameters investigated\n",   
 chem\_svm\_r$results %>%   
 ggplot(aes(x = C, y = RMSE)) + geom\_line() + geom\_point(size = 1) +  
 labs(subtitle = "Radial kernel"),  
 chem\_svm\_l$results %>%   
 ggplot(aes(x = C, y = RMSE)) + geom\_line() + geom\_point(size = 1) +  
 labs(subtitle = "Linear kernel"),  
 chem\_svm\_p$results %>%   
 ggplot(aes(x = C, y = RMSE, col = factor(degree), lty = factor(scale))) +  
 geom\_line() + geom\_point(size = 1) +  
 labs(subtitle = "Polynomial kernel", col = "degree", lty = "scale") +  
 ylim(1, 7.5) + theme(legend.position = "top", legend.box = "vertical",  
 legend.margin = margin(0, 0, 0, 0),  
 legend.box.margin = margin(0, 0, 0, 0)),  
 nrow = 1  
)



For the radial and linear kernels, it is clear that the radial and linear kernels have clear minima at . For the polynomial fit, there was a great deal more variation -- this may be due to the presence of two additional tuning parameters. The line for the third-degree polynomial with scale = 0.1 was so high (> 40) that it was removed from the graph to keep the scales of the other values readable. The minimum values for the three models occurred with the following parameters:

data.frame(C = c(chem\_svm\_r$results %>% slice(which.min(RMSE)) %>% pull(C),  
 chem\_svm\_l$results %>% slice(which.min(RMSE)) %>% pull(C),  
 chem\_svm\_p$results %>% slice(which.min(RMSE)) %>% pull(C)),  
 degree = c(NA, NA, chem\_svm\_p$results %>% slice(which.min(RMSE)) %>% pull(degree)),  
 scale = c(NA, NA, chem\_svm\_p$results %>% slice(which.min(RMSE)) %>% pull(scale)),  
 RMSE = c(min(chem\_svm\_r$results$RMSE),  
 min(chem\_svm\_l$results$RMSE),  
 min(chem\_svm\_p$results$RMSE)),  
 row.names = c("Radial", "Linear", "Polynomial")) %>%   
 pander()

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | C | degree | scale | RMSE |
| **Radial** | 1 | NA | NA | 1.85 |
| **Linear** | 1 | NA | NA | 2.505 |
| **Polynomial** | 1 | 2 | 0.001 | 1.479 |

The polynomial SVM model has the best resampled RMSE, and it falls between the resampled RMSE performance of the neural network and MARS models. Each of the SVM models is used to predict the test set, and their performance presented below:

# radial kernel  
chem\_svm\_r\_pred <- predict(chem\_svm\_r, chem\_pred\_test)  
chem\_svm\_r\_perf <- defaultSummary(data.frame(obs = chem\_yield\_test, pred = chem\_svm\_r\_pred))  
# linear kernel  
chem\_svm\_l\_pred <- predict(chem\_svm\_l, chem\_pred\_test)  
chem\_svm\_l\_perf <- defaultSummary(data.frame(obs = chem\_yield\_test, pred = chem\_svm\_l\_pred))  
# polynomial kernel  
chem\_svm\_p\_pred <- predict(chem\_svm\_p, chem\_pred\_test)  
chem\_svm\_p\_perf <- defaultSummary(data.frame(obs = chem\_yield\_test, pred = chem\_svm\_p\_pred))  
  
pander(data.frame(RMSE = c(chem\_svm\_r\_perf["RMSE"], chem\_svm\_l\_perf["RMSE"], chem\_svm\_p\_perf["RMSE"]),  
 row.names = c("Radial", "Linear", "Polynomial")))

|  |  |
| --- | --- |
|  | RMSE |
| **Radial** | 1.774 |
| **Linear** | 1.191 |
| **Polynomial** | 1.261 |

Now the linear SVM model shows a better performance than the polynomial model; both models shows performance against the test set between the neural network and MARS models.

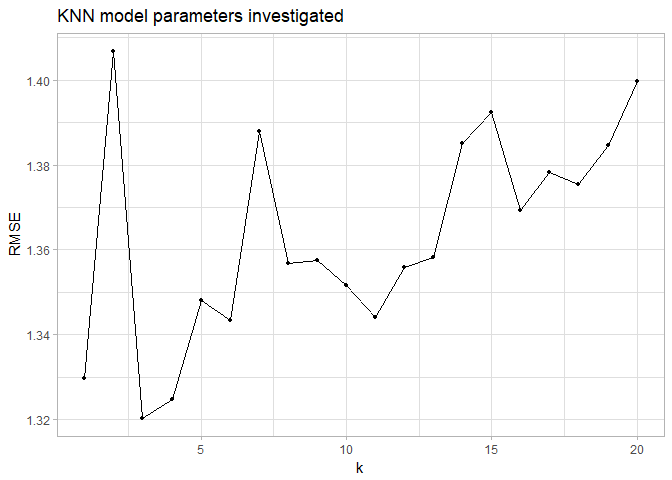
### K Nearest Neighbors Model

Finally, a k-nearest neighbor model is fit using 1-20 nearest neighbors:

set.seed(100)  
chem\_knn <- train(chem\_pred\_train, chem\_yield\_train,  
 method = "knn", trControl = chem\_ctrl,  
 tuneGrid = data.frame(k = 1:20))

The relationship between parameter and RMSE are explored below:

chem\_knn$results %>%   
 ggplot(aes(x = k, y = RMSE)) +  
 geom\_line() + geom\_point(size = 1) +  
 labs(title = "KNN model parameters investigated")



The minimum RMSE occurs at . The corresponding RMSE is 1.32. The model is also used to predict the test set:

chem\_knn\_pred <- predict(chem\_knn, chem\_pred\_test)  
chem\_knn\_perf <- defaultSummary(data.frame(obs = chem\_yield\_test, pred = chem\_knn\_pred))

This model yields a test performance of 1.16.

### Model Selection

A function is created and utilized to compare the resampled and test RMSEs across the six models discussed above:

RMSEs <- function (model\_set) {  
 mdl\_names <- character()  
 resampled <- numeric()  
 test <- numeric()  
 for (mdl in model\_set) {  
 mdl\_names <- c(mdl\_names, mdl)  
 resampled <- c(resampled, min(get(paste0("chem\_", mdl))$results$RMSE))  
 test <- c(test, get(paste0("chem\_", mdl, "\_perf"))["RMSE"])  
 }  
 pander(data.frame(`Resampled RMSE` = resampled, `Test RMSE` = test,  
 row.names = mdl\_names, check.names = FALSE), digits = 4)  
}  
  
RMSEs(c("nn", "mars", "svm\_r", "svm\_l", "svm\_p", "knn"))

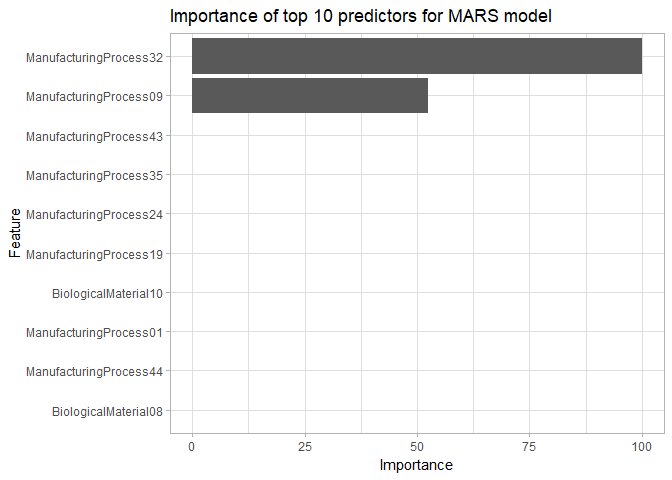
|  |  |  |
| --- | --- | --- |
|  | Resampled RMSE | Test RMSE |
| **nn** | 1.632 | 1.604 |
| **mars** | 1.158 | 1.162 |
| **svm\_r** | 1.85 | 1.774 |
| **svm\_l** | 2.505 | 1.191 |
| **svm\_p** | 1.479 | 1.261 |
| **knn** | 1.32 | 1.16 |

The model with the best resampled RMSE is the MARS model; the model with the best test RMSE is the KNN model. The difference between the two models' resampled RMSE (0.162) is far more significant than the difference in test set RMSE (0.002); thus the **MARS model** is selected and used going forwards.

## Predictor Importance & Relationships

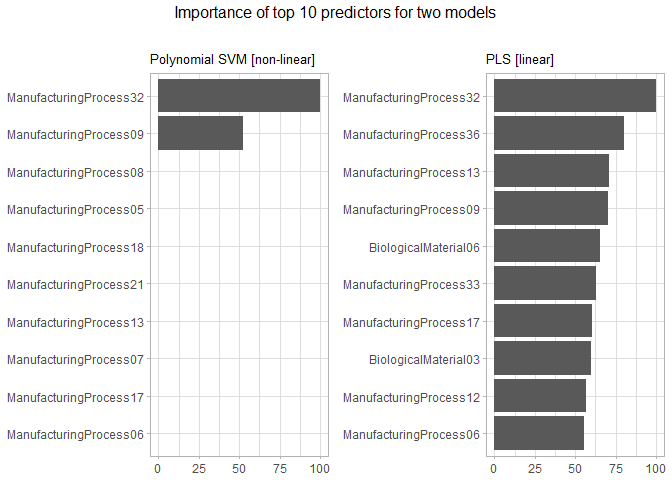
The importance of each predictor in the polynomial SVM model is calculated, and the top 10 predictors are plotted below:

# get predictor importance  
mars\_imp <- varImp(chem\_mars)  
# plot importance  
ggplot(mars\_imp, top = 10) + ggtitle("Importance of top 10 predictors for MARS model")



It is clear that there are only two predictors of any importance in this model -- *ManufacturingProcess32* and *ManufacturingProcess09*. Of the two, *ManufacturingProcess32* is roughly twice as important as *ManufacturingProcess09*. Obviously, 100% of these predictors are manufacturing-related. This is contrary to the finding from the linear model, which found 20% of the top 10 predictors to be biological. For further investigation, variable importance is compared with the linear PLS model fit in problem 6.3:

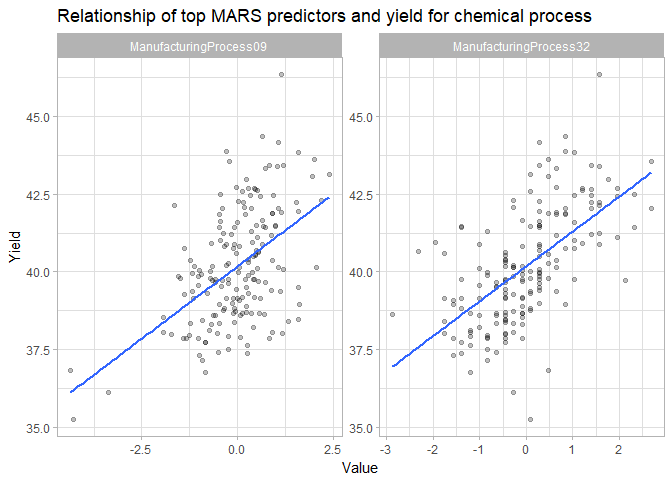
# retrain linear model used  
set.seed(100)  
chem\_pls <- train(x = chem\_pred\_train, y = chem\_yield\_train,  
 method = "pls", tuneLength = 20, trControl = chem\_ctrl)  
# get predictor importance for linear model  
pls\_imp <- varImp(chem\_pls)  
# plot importance for both models next to each other  
gridExtra::grid.arrange(top = "Importance of top 10 predictors for two models\n",  
 ggplot(mars\_imp, top = 10) +  
 labs(subtitle = "Polynomial SVM [non-linear]", x = NULL, y = NULL),  
 ggplot(pls\_imp, top = 10) +  
 labs(subtitle = "PLS [linear]", x = NULL, y = NULL),  
 nrow = 1)



For both models, *ManufacturingProcess32* is the most important predictor. Both models also contain *ManufacturingProcess09* -- it is the second-most important variable in the MARS model and the fourth-most important in the PLS model.

For the MARS model, the relationship of the top 2 predictors is shown below:

# get top 2 predictors  
mars\_top <- rownames(mars\_imp$importance)[order(mars\_imp$importance$Overall, decreasing = TRUE)][1:2]  
# separate top predictors & yield into data frame  
mars\_rel <- as.data.frame(cbind(chem\_pred\_trans, chem\_yield)[, c(mars\_top, "Yield")])  
# plot to investigate relationships  
mars\_rel %>%   
 # convert to tidy format  
 gather(Predictor, Value, -Yield) %>%   
 # plot  
 ggplot(aes(x = Value, y = Yield)) +  
 geom\_point(alpha = 0.25) +  
 stat\_smooth(se = FALSE, method = "glm") +  
 facet\_wrap(~ Predictor, nrow = 1, scales = "free") +  
 labs(title = "Relationship of top MARS predictors and yield for chemical process")



Both of the predictors have positive correlations with yield, and both appear to be fairly strong.

# KJ 8.1

Reusing the simulated data from exercise 7.2:

# convert to single df  
fried\_train$x$y <- fried\_train$y  
fried\_train <- fried\_train$x

## Traditional Forest

### Original Predictors

A random forest model is fit using all 10 predictors using 1000 trees:

library(randomForest)  
# fit rf model to data  
set.seed(100)  
fried\_rf\_orig <- randomForest(y ~ ., data = fried\_train, importance = TRUE, ntree = 1000)  
# view importance  
fried\_imp\_orig <- varImp(fried\_rf\_orig, scale = FALSE)  
pander(fried\_imp\_orig)

|  |  |
| --- | --- |
|  | Overall |
| **X1** | 8.813 |
| **X2** | 6.488 |
| **X3** | 0.698 |
| **X4** | 7.82 |
| **X5** | 2.291 |
| **X6** | 0.115 |
| **X7** | 0.07132 |
| **X8** | -0.1007 |
| **X9** | -0.01398 |
| **X10** | 0.061 |

All ten variables have non-zero importance, but the variables X1-X5 have importances far higher than the uninformative predictors. It should be noted that X3 has an importance closer to the uninformative predictors than the other informative predictors; this is likely do to the low correlation shown in Problem 7.2.

### Correlated Predictor

A predictor correlated to X1 (scaled by a factor of 1.1 with random noise added) is created as X11. A model is then fit to this data:

# create correlated variable  
set.seed(100)  
fried\_train$X11 <- fried\_train$X1 + rnorm(200, sd = 0.1)  
# fit rf model to new full data  
set.seed(100)  
fried\_rf\_corr <- randomForest(y ~ ., data = fried\_train, importance = TRUE, ntree = 1000)  
# view importance  
fried\_imp\_corr <- varImp(fried\_rf\_corr, scale = FALSE)  
pander(fried\_imp\_corr)

|  |  |
| --- | --- |
|  | Overall |
| **X1** | 6.447 |
| **X2** | 5.982 |
| **X3** | 0.5694 |
| **X4** | 7.204 |
| **X5** | 2.22 |
| **X6** | 0.223 |
| **X7** | 0.112 |
| **X8** | -0.08354 |
| **X9** | -0.03611 |
| **X10** | -0.03111 |
| **X11** | 4.063 |

As expected, the importance score for X1 decreased -- the value decreased from a value of 8.81 to 6.45.

A second correlated variable is added and another random forest model fit:

# create correlated variable  
set.seed(200)  
fried\_train$X12 <- fried\_train$X1 + rnorm(200, sd = 0.2)  
# fit rf model to new full data  
set.seed(200)  
fried\_rf\_corr2 <- randomForest(y ~ ., data = fried\_train, importance = TRUE, ntree = 1000)  
# view importance  
fried\_imp\_corr2 <- varImp(fried\_rf\_corr2, scale = FALSE)  
pander(fried\_imp\_corr2)

|  |  |
| --- | --- |
|  | Overall |
| **X1** | 6.224 |
| **X2** | 6.489 |
| **X3** | 0.4682 |
| **X4** | 7.704 |
| **X5** | 2.311 |
| **X6** | 0.2155 |
| **X7** | 0.02916 |
| **X8** | -0.06198 |
| **X9** | 0.02058 |
| **X10** | -0.03203 |
| **X11** | 3.354 |
| **X12** | 1.014 |

The importance of X1 is further decreased -- now to 6.22. The importance of the other correlated variable X11 also decreased from 4.06 to 3.35.

## Conditional Information Tree Models

A random forest model is fit using conditional inference trees, first using the original predictors and then including the correlated predictor:

library(party)  
# fit model with original predictors  
set.seed(100)  
fried\_cond\_orig <- cforest(y ~ ., data = fried\_train[, 1:11],  
 controls = cforest\_control(ntree = 1000))  
# fit model with correlated predictor  
set.seed(100)  
fried\_cond\_corr <- cforest(y ~ ., data = fried\_train[, 1:12],  
 controls = cforest\_control(ntree = 1000))  
# view importance  
fried\_imp\_cond\_orig <- varImp(fried\_cond\_orig)  
fried\_imp\_cond\_corr <- varImp(fried\_cond\_corr)  
pander(data.frame(Original = c(fried\_imp\_cond\_orig[, 1], NA),  
 Correlated = fried\_imp\_cond\_corr[, 1],  
 row.names = paste0("X", 1:11)))

|  |  |  |
| --- | --- | --- |
|  | Original | Correlated |
| **X1** | 9.02 | 6.614 |
| **X2** | 7.01 | 6.557 |
| **X3** | 0.06789 | 0.07207 |
| **X4** | 8.658 | 8.199 |
| **X5** | 2.129 | 2.129 |
| **X6** | -0.02431 | 0.0358 |
| **X7** | 0.02712 | 0.08044 |
| **X8** | -0.07405 | -0.0551 |
| **X9** | -0.07777 | -0.01891 |
| **X10** | -0.0301 | -0.0006652 |
| **X11** | NA | 3.105 |

Similar behavior is observed as for the traditional model:

* Of the original predictors, X1-X5 have the highest importance
* The importance of predictors X1 decreases when the correlated predictor X11 is added

## Additional Models

The process of fitting a model, both with the original and correlated predictors, and investigating the predictor importance, is repeated for additional tree models.

### Bagged Trees

library(ipred)  
# fit model with original predictors  
set.seed(100)  
fried\_bag\_orig <- bagging(y ~ ., data = fried\_train[, 1:11], nbag = 50)  
# fit model with correlated predictor  
set.seed(100)  
fried\_bag\_corr <- bagging(y ~ ., data = fried\_train[, 1:12], nbag = 50)  
# view importance  
fried\_imp\_bag\_orig <- varImp(fried\_bag\_orig)  
fried\_imp\_bag\_corr <- varImp(fried\_bag\_corr)  
pander(data.frame(Original = c(fried\_imp\_bag\_orig[, 1], NA),  
 Correlated = fried\_imp\_bag\_corr[, 1],  
 row.names = paste0("X", 1:11)))

|  |  |  |
| --- | --- | --- |
|  | Original | Correlated |
| **X1** | 1.752 | 1.691 |
| **X2** | 0.804 | 0.7728 |
| **X3** | 2.234 | 1.483 |
| **X4** | 1.26 | 2.218 |
| **X5** | 2.689 | 1.108 |
| **X6** | 2.467 | 2.648 |
| **X7** | 0.9852 | 2.436 |
| **X8** | 1.081 | 0.9321 |
| **X9** | 0.5365 | 1.009 |
| **X10** | 0.6874 | 0.4758 |
| **X11** | NA | 0.623 |

This model shows weakly similar behavior with noticeable differences: the variable X6 is significant; the variables X7 and X8 have stronger significance relative to the previous models; and the variable X2 is very insignificant. The expected reduction in X1's importance with the inclusion of X11 is still shown, however.

### Cubist

library(Cubist)  
# fit model with original predictors  
set.seed(100)  
fried\_cube\_orig <- cubist(fried\_train[, 1:10], fried\_train$y, committees = 100)  
# fit model with correlated predictor  
set.seed(100)  
fried\_cube\_corr <- cubist(fried\_train[, c(1:10, 12)], fried\_train$y, committees = 100)  
# view importance  
fried\_imp\_cube\_orig <- varImp(fried\_cube\_orig)  
fried\_imp\_cube\_corr <- varImp(fried\_cube\_corr)  
pander(data.frame(Original = c(fried\_imp\_cube\_orig[, 1], NA),  
 Correlated = fried\_imp\_cube\_corr[, 1],  
 row.names = paste0("X", 1:11)))

|  |  |  |
| --- | --- | --- |
|  | Original | Correlated |
| **X1** | 71.5 | 69 |
| **X2** | 47 | 44 |
| **X3** | 58.5 | 62 |
| **X4** | 48 | 3 |
| **X5** | 33 | 47 |
| **X6** | 13 | 27.5 |
| **X7** | 0 | 11 |
| **X8** | 0 | 3.5 |
| **X9** | 0 | 0 |
| **X10** | 0 | 0 |
| **X11** | NA | 0 |

Similar behavior is seen as in the CART model, with the exception of the importance of X6.

### Boosted Trees

library(gbm)  
# fit model with original predictors  
set.seed(100)  
fried\_boost\_orig <- gbm(y ~ ., data = fried\_train[1:11], distribution = "gaussian")  
# fit model with correlated predictor  
set.seed(100)  
fried\_boost\_corr <- gbm(y ~ ., data = fried\_train[1:12], distribution = "gaussian")  
# view importance  
fried\_imp\_boost\_orig <- summary.gbm(fried\_boost\_orig, plotit = FALSE)  
fried\_imp\_boost\_corr <- summary.gbm(fried\_boost\_corr, plotit = FALSE)  
pander(right\_join(fried\_imp\_boost\_orig, fried\_imp\_boost\_corr, by = "var") %>%   
 column\_to\_rownames("var") %>%  
 rename("Original" = "rel.inf.x", "Correlated" = "rel.inf.y"))

|  |  |  |
| --- | --- | --- |
|  | Original | Correlated |
| **X1** | 57.89 | 54.77 |
| **X2** | 22.62 | 21.87 |
| **X4** | 19.49 | 18.41 |
| **X11** | NA | 4.955 |
| **X3** | 0 | 0 |
| **X5** | 0 | 0 |
| **X6** | 0 | 0 |
| **X7** | 0 | 0 |
| **X8** | 0 | 0 |
| **X9** | 0 | 0 |
| **X10** | 0 | 0 |

The boosted tree model exhibits the observed behavior most strongly -- only 3 of the 5 informative predictors have a non-zero importance, and there is again a reduction in the importance of X1 with the addition of X11.

# KJ 8.2

Trees suffer from selection bias --- per the text (citing Loh & Shih [1997]),

The danger occurs when a data set consists of a mix of informative and noise variables, and the noise variables have many more splits than the informative variables. Then there is a high probability that the noise variables will be chosen to split the top nodes of the tree. Pruning will produce either a tree with misleading structure or no tree at all.

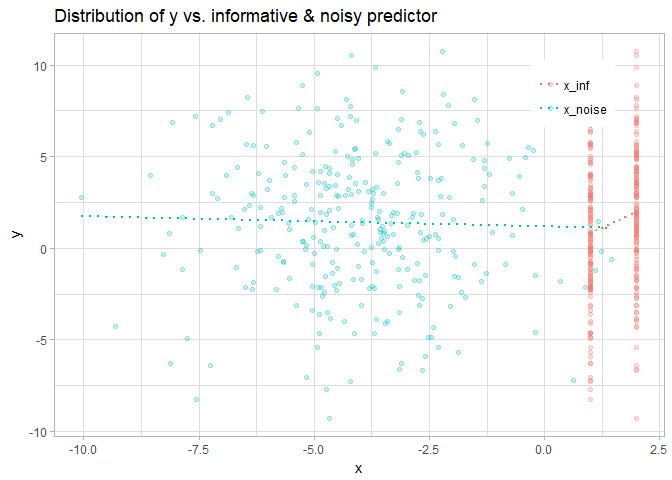
## Data Creation

To simulate this bias, a simple dataset is created with three columns:

* *x\_noise*, a noise predictor of randomly generated values
* *x\_inf*, an informative predictor with two discrete values
* *y*, the response, highly correlated with *x\_inf*

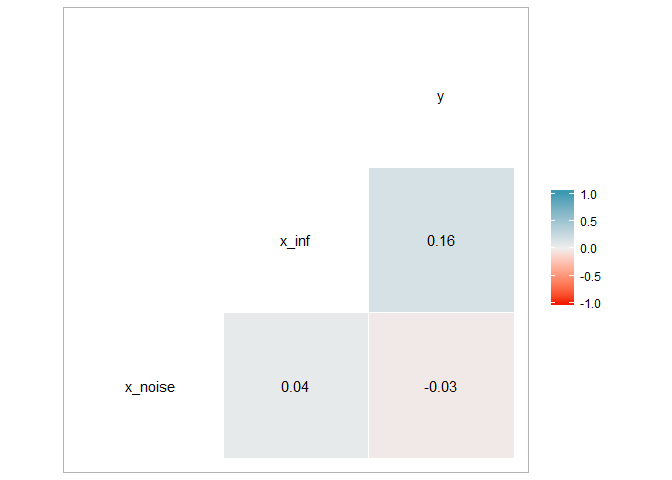
An example of this data is generated, and a plot of the data is investigated:

# generate example data  
set.seed(100)  
x\_noise <- rnorm(300, mean = -4, sd = 2)  
x\_inf <- rep(1:2, each = 150)  
set.seed(101)  
y <- x\_inf + rnorm(300, sd = 4)  
# create df of generated data  
bias\_ex <- data\_frame(x\_noise, x\_inf, y)  
  
# tidy & plot data  
bias\_ex %>%   
 gather(xvar, xval, -y) %>%   
 ggplot(aes(xval, y, col = xvar)) +  
 geom\_point(alpha = 0.25) +  
 stat\_smooth(method = "lm", se = FALSE, lty = 3) +  
 labs(title = "Distribution of y vs. informative & noisy predictor",  
 x = "x", color = NULL) +  
 theme(legend.position = c(0.85, 0.85))



The plot shows a slightly-negative, near-zero correlation between *x\_noise* and *y*, with a stronger, positive correlation between *x\_inf* and *y*. This is further investigated with a correlation plot:

GGally::ggcorr(bias\_ex, label = TRUE, label\_round = 2, low = "#F21A00", high = "#3B9AB2") +  
 labs("Correlation of predictors (informative & noisy) and response")



This plot confirms the above observations: the correlation between *x\_noise* and *y* is -0.03; the correlation between *x\_inf* and *y* is 0.16.

## Simulation

Following data generation, a tree model with a maximum depth of 1 is fit. The variable used to split the tree is extracted from the model's frame. This process (including data generation) is repeated 100 times to return simulated results showing the frequency of each predictor being selected to split the tree:

# initiate vector to track selected predictor  
bias\_pred <- character()  
# simulate fitted rpart models  
library(rpart)  
for (i in 1:100) {  
 # generate data  
 x\_inf <- rep(1:2, each = 150)  
 set.seed(100 + i) # different predictors per iteration  
 x\_noise <- rnorm(300, mean = -4, sd = 2)  
 set.seed(100 \* i)  
 y = x\_inf + rnorm(300, sd = 4)  
 bias\_sim <- data\_frame(x\_noise, x\_inf, y)  
 # train tree  
 bias\_tree <- rpart(y ~ x\_inf + x\_noise, bias\_sim,  
 control = rpart.control(maxdepth = 1))  
 # get variable used for tree split & add to vector  
 bias\_pred <- c(bias\_pred, as.character(bias\_tree$frame$var[1]))  
}  
pander(table(bias\_pred))

|  |  |  |
| --- | --- | --- |
|  | x\_inf | x\_noise |
| 7 | 46 | 47 |

The table above shows that the two variables are selected nearly as frequently, with the noise variable *x\_noise* having selected slightly more frequently (the count indicated without a variable indicates neither predictor being selected for a split in the tree i.e. the tree does not split). As shown in the section above, the response *y* is defined from the predictor *x\_inf*, but the larger spread of *x\_noise* creates more possible splits in the variable on which to define a tree. The selection of the granular noise variable over the less granular informative variable illustrates the bias of tree models based on granularity.

# KJ 8.3

The right model (with both parameters set to 0.9) is focused on only the first few predictors for reasons related to each of the tuning parameters: the higher learning rate yields a "greedier" model (likely to identify fewer predictors); the higher bagging fraction means that a larger portion of the data is used in model building (yielding more concentrated importance on few variables).

The concentration of the model importance on fewer predictors in the right model will likely cause lower performance, as it does provide importance across the full range of predictors; thus the left model (with both parameters set to 0.1) will likely have better predictive performance.

Increased interaction depth (i.e. tree depth) is likely to spread importance more evenly across predictors, similar to the behavior of an increased bagging fraction. This would yield a decreased slope in predictor importance for both the left and right models.

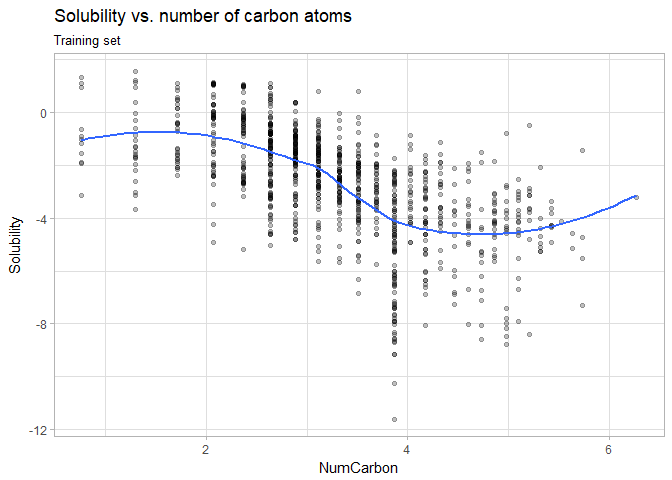
# KJ 8.4

The predictor *NumCarbon* (the number of carbon atoms) is used to train each of the suggested models:

data("solubility", package = "AppliedPredictiveModeling")  
# separate number of carbon atoms  
sol\_carbon\_train <- solTrainXtrans %>% select(NumCarbon)  
sol\_carbon\_test <- solTestXtrans %>% select(NumCarbon)

The relationship between the predictor and the response (solubility) is shown below for the training set, along with a Loess fit to the data:

sol\_carbon\_train %>%   
 mutate(Solubility = solTrainY) %>%   
 ggplot(aes(NumCarbon, Solubility)) +  
 geom\_point(alpha = 0.25) +  
 stat\_smooth(method = "loess", se = FALSE) +  
 ggtitle("Solubility vs. number of carbon atoms", subtitle = "Training set")



From this plot, it is clear that the number of carbon atoms can only take certain discrete values, and there is a generally negative relationship between the number of carbon atoms and the chemical solubility.

## Simple Regression Tree

A simple tree is trained using maximum depth (with the rpart2 method):

# train model  
set.seed(100)  
sol\_tree <- train(sol\_carbon\_train, solTrainY,  
 method = "rpart2")  
# predict solubility & get results  
sol\_tree\_pred <- predict(sol\_tree, sol\_carbon\_test)  
sol\_tree\_perf <- defaultSummary(data.frame(obs = solTestY, pred = sol\_tree\_pred))

The optimal simple tree model has a max depth of 3 and yields a resampled RMSE of 1.52 and a test set RMSE of 1.503.

## Random Forest Model

A random forest model is also fit to the data:

# train model  
set.seed(100) # for replicability  
sol\_rf <- train(sol\_carbon\_train, solTrainY,  
 method = "rf")  
# predict solubility & get results  
sol\_rf\_pred <- predict(sol\_rf, sol\_carbon\_test)  
sol\_rf\_perf <- defaultSummary(data.frame(obs = solTestY, pred = sol\_rf\_pred))

The random forest model yields a resampled RMSE of 1.524 and a test set RMSE of 1.506.

## Cubist Models

Four Cubist models are fit:

* single committee; no neighbors
* multiple committees; no neighbors
* single committees; neighbors
* multiple committees; neighbors

# train models  
## single committee; no neighbors  
set.seed(100)  
sol\_cube <- train(sol\_carbon\_train, solTrainY,  
 method = "cubist", verbose = FALSE,  
 tuneGrid = data.frame(committees = 1, neighbors = 0))  
## multiple committees; no neighbors  
set.seed(100)  
sol\_cube\_c <- train(sol\_carbon\_train, solTrainY,  
 method = "cubist", verbose = FALSE,  
 tuneGrid = expand.grid(committees = c(50, 75, 100), neighbors = 0))  
## single committee; neighbors  
set.seed(100)  
sol\_cube\_n <- train(sol\_carbon\_train, solTrainY,  
 method = "cubist", verbose = FALSE,  
 tuneGrid = data.frame(committees = 1, neighbors = c(1, 5, 9)))  
## multiple committees; neighbors  
set.seed(100)  
sol\_cube\_cn <- train(sol\_carbon\_train, solTrainY,  
 method = "cubist", verbose = FALSE,  
 tuneGrid = expand.grid(committees = c(50, 75, 100), neighbors = c(1, 5, 9)))  
  
# get predictions & performance  
sol\_cube\_pred <- predict(sol\_cube, sol\_carbon\_test)  
sol\_cube\_perf <- defaultSummary(data.frame(obs = solTestY, pred = sol\_cube\_pred))  
sol\_cube\_c\_pred <- predict(sol\_cube\_c, sol\_carbon\_test)  
sol\_cube\_c\_perf <- defaultSummary(data.frame(obs = solTestY, pred = sol\_cube\_c\_pred))  
sol\_cube\_n\_pred <- predict(sol\_cube\_n, sol\_carbon\_test)  
sol\_cube\_n\_perf <- defaultSummary(data.frame(obs = solTestY, pred = sol\_cube\_n\_pred))  
sol\_cube\_cn\_pred <- predict(sol\_cube\_cn, sol\_carbon\_test)  
sol\_cube\_cn\_perf <- defaultSummary(data.frame(obs = solTestY, pred = sol\_cube\_cn\_pred))

The table below shows the optimal parameters, as well as the resampled & test RMSE, for all four Cubist models:

data\_frame(  
 model = c("Base", "Committees", "Neighbors", "Committees & Neighbors"),  
 Committees = c(sol\_cube$results %>% slice(which.min(RMSE)) %>% pull(committees),  
 sol\_cube\_c$results %>% slice(which.min(RMSE)) %>% pull(committees),  
 sol\_cube\_n$results %>% slice(which.min(RMSE)) %>% pull(committees),  
 sol\_cube\_cn$results %>% slice(which.min(RMSE)) %>% pull(committees)),  
 Neighbors = c(sol\_cube$results %>% slice(which.min(RMSE)) %>% pull(neighbors),  
 sol\_cube\_c$results %>% slice(which.min(RMSE)) %>% pull(neighbors),  
 sol\_cube\_n$results %>% slice(which.min(RMSE)) %>% pull(neighbors),  
 sol\_cube\_cn$results %>% slice(which.min(RMSE)) %>% pull(neighbors)),  
 Resampled = c(min(sol\_cube$results$RMSE), min(sol\_cube\_c$results$RMSE),  
 min(sol\_cube\_n$results$RMSE), min(sol\_cube\_cn$results$RMSE)),  
 Test = c(sol\_cube\_perf[["RMSE"]], sol\_cube\_c\_perf[["RMSE"]],  
 sol\_cube\_n\_perf[["RMSE"]], sol\_cube\_cn\_perf[["RMSE"]])  
) %>% pander()

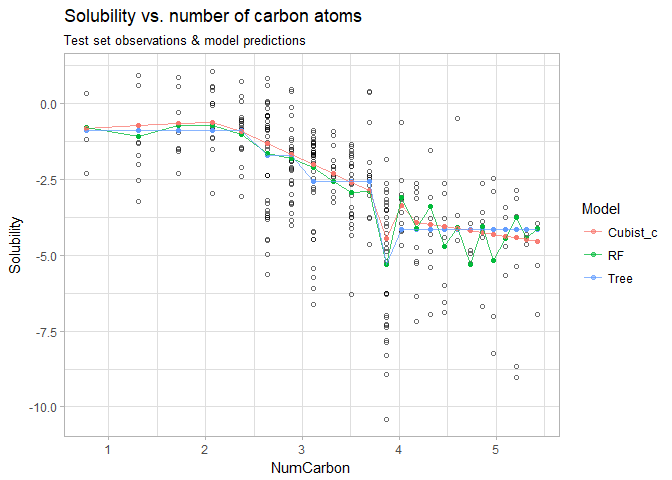
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| model | Committees | Neighbors | Resampled | Test |
| Base | 1 | 0 | 1.547 | 1.566 |
| Committees | 100 | 0 | 1.55 | 1.534 |
| Neighbors | 1 | 9 | 1.628 | 1.706 |
| Committees & Neighbors | 75 | 9 | 1.629 | 1.704 |

The best test set RMSE, and second-best resampled RMSE (by 0.003) performance among Cubist models fit is for the model with 100 committees and no neighbors.

## Model Comparison

The predictor and response are plotted below for the test set, with the predictions for each of the three models (simple tree, random forest, and the best Cubist model) overlaid:

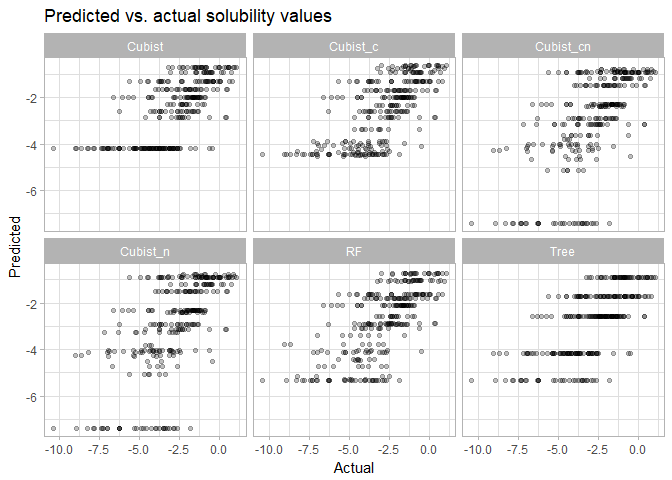
sol\_carbon\_test %>%   
 mutate(Solubility = solTestY,  
 Tree = sol\_tree\_pred,  
 RF = sol\_rf\_pred,  
 Cubist\_c = sol\_cube\_c\_pred) %>%   
 gather(Model, Prediction, -(NumCarbon:Solubility)) %>%   
 ggplot(aes(x = NumCarbon)) +  
 geom\_point(aes(y = Solubility), alpha = 0.25, color = "black", shape = 1) +  
 geom\_point(aes(y = Prediction, col = Model), alpha = 0.55) +  
 geom\_line(aes(y = Prediction, col = Model), alpha = 0.75) +  
 ggtitle("Solubility vs. number of carbon atoms",  
 subtitle = "Test set observations & model predictions")



The models appear to match one another reasonably well for ; however above this point, the model differences become more apparent -- the Cubist model appears least influenced by the low points just below 4 on the x axis. Above , both the Cubist and tree models show fairly steady behavior, while the random forest model shows a great deal of variation from the data, suggesting that it may be more biased.

To get a more direct comparison of all models' predictions vs. the actual test observations, an additional plot is created:

data\_frame(Actual = solTestY,  
 Tree = sol\_tree\_pred,  
 RF = sol\_rf\_pred,  
 Cubist = sol\_cube\_pred,  
 Cubist\_c = sol\_cube\_c\_pred,  
 Cubist\_n = sol\_cube\_n\_pred,  
 Cubist\_cn = sol\_cube\_cn\_pred) %>%   
 gather(Model, Predicted, -Actual) %>%   
 ggplot(aes(x = Actual, y = Predicted)) +  
 geom\_point(alpha = 0.25) +  
 facet\_wrap(~ Model) +  
 ggtitle("Predicted vs. actual solubility values")



In this plot, the nature of the tree plot (6 outcomes due to a tree depth of 3) is quite clear. The effect of altering the tuning parameters of the Cubist models is also apparent. The introduction of committees creates more continuous predictions, while the introduction of neighbors creates a larger spread in the data -- models without neighbors have a lower bound around -4.5, while those with neighbors have a lower bound around -7.

# KJ 8.7

The pre-processed/imputed/split data from Problem 7.5 (which was prepared in the same way as in Problem 6.3) is reused for this exercise.

## Model Training & Performance

For consistency, the same training controls chem\_ctrl are used as in Problems 6.3 & 7.5. For completion, each of the six model types illustrated in Chapter 8 are trained.

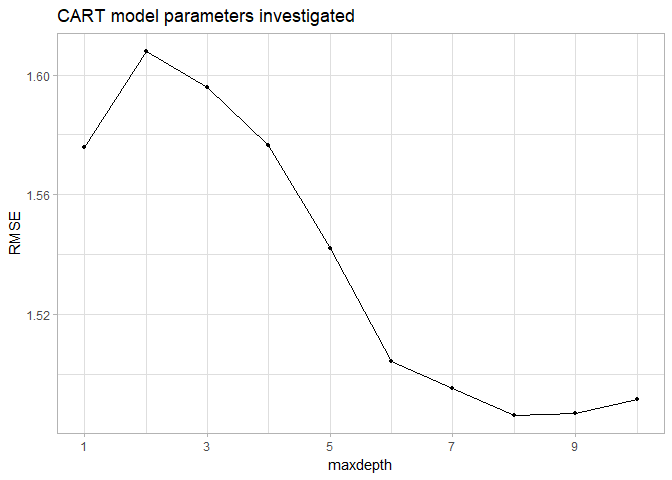
### Single Tree Model

A single tree model of maximum depth is trained using a maximum depth of 1-10:

set.seed(100)  
chem\_tree <- train(chem\_pred\_train, chem\_yield\_train,  
 method = "rpart2", trControl = chem\_ctrl,  
 tuneGrid = expand.grid(maxdepth = 1:10))

The performance profile of the trained models is presented below:

chem\_tree$results %>%  
 ggplot(aes(x = maxdepth, y = RMSE)) +  
 geom\_line() + geom\_point(size = 1) +  
 scale\_x\_continuous(breaks = chem\_tree$results$maxdepth[seq(1, 10, 2)]) +  
 labs(title = "CART model parameters investigated")



The optimal resampled RMSE is 1.486, which occurs achieved with a parameter maxdepth = 8. The model is also used to predict the test set:

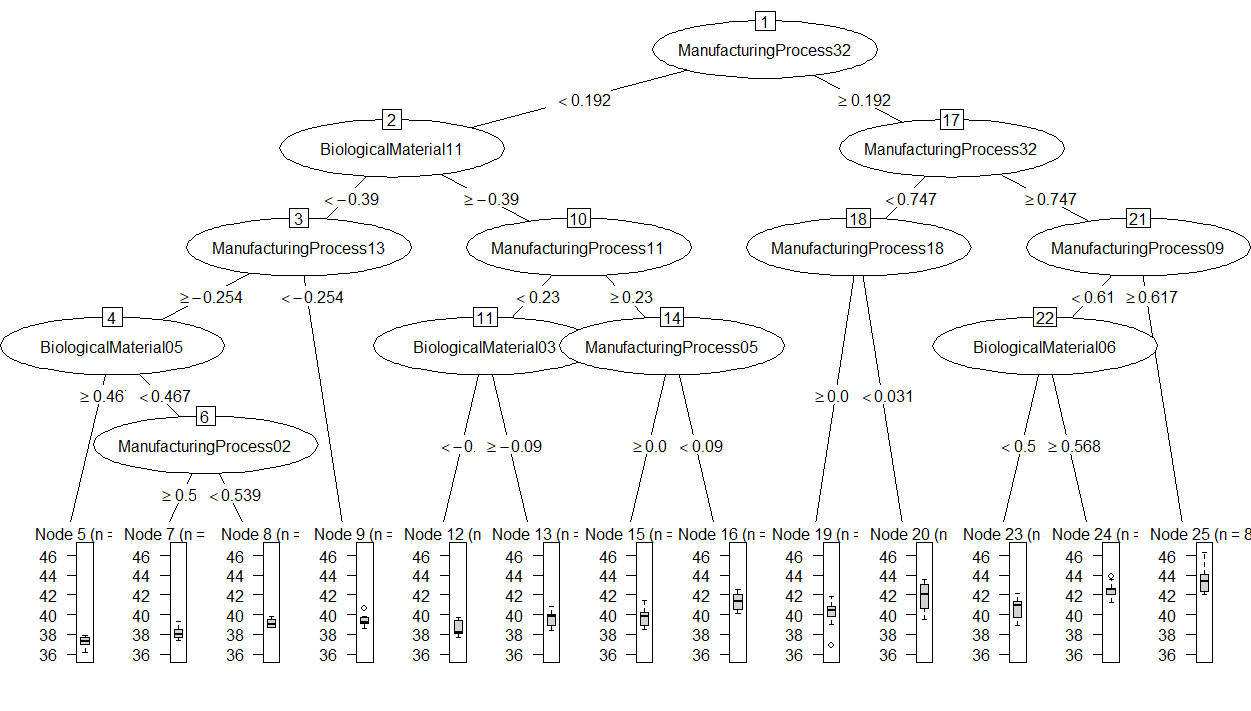
chem\_tree\_pred <- predict(chem\_tree, chem\_pred\_test)  
chem\_tree\_perf <- defaultSummary(data.frame(obs = chem\_yield\_test, pred = chem\_tree\_pred))

This model yields a test performance of 1.309.

#### Tree Visualization

The optimal single tree is visualized below, along with distribution of the response variable *Yield* in each terminal node:

plot(partykit::as.party(chem\_tree$finalModel))



The visualization makes it clear that manufacturing predictors are used more commonly than biological predictors. Predictors *ManufacturingProcess02* and *ManufacturingProcess05* have a negative relationship with *Yield*, while *BiologicalMaterial03* and *BiologicalMaterial06* have positive relationships with *Yield*. The visualization suggests other relationships, but later splits in the tree may make these conclusions unsafe due to collinearity.

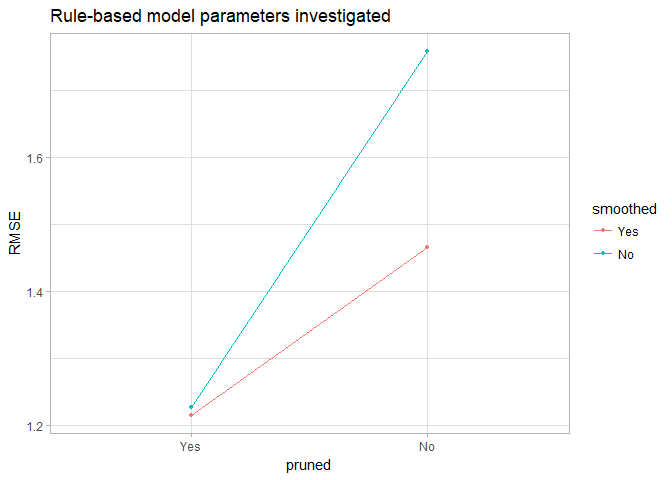
### Rule-Based Model

Next, a rule-based model is trained across the default tuning parameters:

set.seed(100)  
chem\_rule <- train(chem\_pred\_train, chem\_yield\_train,  
 method = "M5Rules", trControl = chem\_ctrl)

The performance across the parameters is shown below:

chem\_rule$results %>%  
 ggplot(aes(x = pruned, y = RMSE, col = smoothed, group = smoothed)) +  
 geom\_line() + geom\_point(size = 1) +  
 labs(title = "Rule-based model parameters investigated")



The optimal RMSE is 1.215 (better than the tree-based model), occurring with both pruning and smoothing. The model is used to predict the test set:

chem\_rule\_pred <- predict(chem\_rule, chem\_pred\_test)  
chem\_rule\_perf <- defaultSummary(data.frame(obs = chem\_yield\_test, pred = chem\_rule\_pred))

The rule-based model yields test performance of 1.186, also better than the tree-based model.

### Bagged Tree Model

A bagged (bootstrapped aggregated) model is also fit to the data, which is not tuned along any parameters:

set.seed(100)  
chem\_bag <- train(chem\_pred\_train, chem\_yield\_train,  
 method = "treebag", trControl = chem\_ctrl)

The bagged model returns a resampled RMSE of 1.265. The model is used to predict the training set:

chem\_bag\_pred <- predict(chem\_bag, chem\_pred\_test)  
chem\_bag\_perf <- defaultSummary(data.frame(obs = chem\_yield\_test, pred = chem\_bag\_pred))

The training RMSE for the bagged model is 1.157, the best test performance of the three tree-based models.

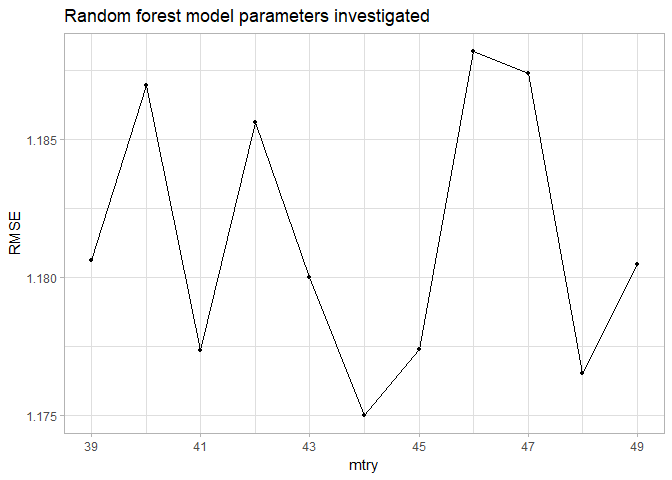
### Random Forest Model

Next, a random forest model is trained across 11 values the number of variables to use mtry around the default of 1/3 the number of observations; using 1000 trees for each model, and returning variable importance:

set.seed(100)  
chem\_rf <- train(chem\_pred\_train, chem\_yield\_train,  
 method = "rf", trControl = chem\_ctrl,  
 tuneGrid = data.frame(mtry = nrow(chem\_pred\_train) / 3 + -5:5),  
 ntrees = 1000, importance = TRUE)

The resampled RMSE of the model is investigated across the tuning parameter:

chem\_rf$results %>%  
 ggplot(aes(x = mtry, y = RMSE)) +  
 geom\_line() + geom\_point(size = 1) +  
 scale\_x\_continuous(breaks = chem\_rf$results$mtry[seq(1, 11, 2)]) +  
 labs(title = "Random forest model parameters investigated")



The optimal RMSE of 1.175 occurs at mtry = 44, equivalent to the default value mtry = nrow(x) / 3. The performance profile shows an interesting pattern, with oscillations of increasing & decreasing values as mtry increases. The test set is predicted with the random forest model:

chem\_rf\_pred <- predict(chem\_rf, chem\_pred\_test)  
chem\_rf\_perf <- defaultSummary(data.frame(obs = chem\_yield\_test, pred = chem\_rf\_pred))

The test performance of this model is 1.09. This model shows the best resampled & test RMSE of the tree-based models investigated.

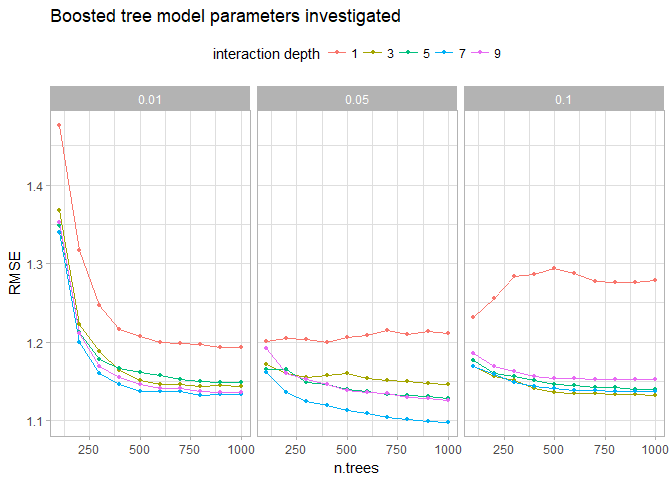
### Boosted Tree Model

A gradient-boosted model is fitted, training across three shrinkage values, five interaction depths, and ten number of tree values, keeping the minimum number of observations per node set to the default of 10:

set.seed(100)  
chem\_boost <- train(chem\_pred\_train, chem\_yield\_train,  
 method = "gbm", trControl = chem\_ctrl,  
 tuneGrid = expand.grid(shrinkage = c(0.01, 0.05, 0.1),  
 interaction.depth = seq(1, 9, 2),  
 n.trees = seq(100, 1000, 100),  
 n.minobsinnode = 10),  
 verbose = FALSE)

The performance of the model across these 3 tuning parameters is shown below:

chem\_boost$results %>%  
 ggplot(aes(x = n.trees, y = RMSE, col = factor(interaction.depth))) +  
 geom\_line() + geom\_point(size = 1) +  
 facet\_wrap(~ shrinkage, nrow = 1) +  
 labs(title = "Boosted tree model parameters investigated", col = "interaction depth") +  
 theme(legend.position = "top")



The optimal model, with n.trees = 1000, interaction.depth = 7, and shrinkage = 0.05, yields an optimal resampled RMSE of 1.098 -- this is the lowest resampled RMSE observed. The boosted model is used to predict the test set:

chem\_boost\_pred <- predict(chem\_boost, chem\_pred\_test)  
chem\_boost\_perf <- defaultSummary(data.frame(obs = chem\_yield\_test, pred = chem\_boost\_pred))

The RMSE of this model against the test set is 1.057 -- this is also the lowest observed.

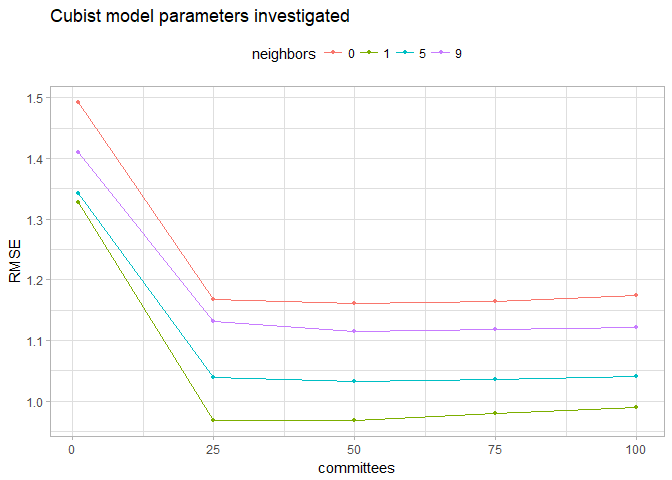
### Cubist Model

Finally, a Cubist model is trained across four values of the tuning parameter neighbors and five values of the tuning parameter committees:

set.seed(100)  
chem\_cube <- train(chem\_pred\_train, chem\_yield\_train,  
 method = "cubist", trControl = chem\_ctrl,  
 tuneGrid = expand.grid(neighbors = c(0, 1, 5, 9),  
 committees = c(1, 25, 50, 75, 100)))

The performance profile of this model is presented across the two tuning parameters below:

chem\_cube$results %>%  
 ggplot(aes(x = committees, y = RMSE, col = factor(neighbors))) +  
 geom\_line() + geom\_point(size = 1) +  
 labs(title = "Cubist model parameters investigated", col = "neighbors") +  
 theme(legend.position = "top")



The optimal resampled RMSE for the Cubist model is 0.968, achieved with 1 neighbor and 50 committees. It should be noted that the difference in performance between 25 committees and 50 committees is rather minor (). If this model were used to predict larger test sets, it may be desirable to make a slight sacrifice in accuracy for improved performance by using the simpler model; however, for this exercise, the model with 50 committees is used for improved accuracy.

The model is used to predict the training set:

chem\_cube\_pred <- predict(chem\_cube, chem\_pred\_test)  
chem\_cube\_perf <- defaultSummary(data.frame(obs = chem\_yield\_test, pred = chem\_cube\_pred))

The test performance of this model is 1.004.

### Model Selection

The resampled and test RMSE of the six tree-based models is presented below:

RMSEs(c("tree", "rule", "bag", "rf", "boost", "cube"))

|  |  |  |
| --- | --- | --- |
|  | Resampled RMSE | Test RMSE |
| **tree** | 1.486 | 1.309 |
| **rule** | 1.215 | 1.186 |
| **bag** | 1.265 | 1.157 |
| **rf** | 1.175 | 1.09 |
| **boost** | 1.098 | 1.057 |
| **cube** | 0.9677 | 1.005 |

The table above shows that the **Cubist model** is clearly the most performant, as measured both by resampled and test set RMSE. This model is selected as the optimal tree-based model and used going forward.

The performance of the best linear model (partial least squares), best nonlinear model (MARS), and best tree-based model (Cubist) are compared below:

# recalculate PLS predictions & performance  
chem\_pls\_pred <- predict(chem\_pls, chem\_pred\_test)  
chem\_pls\_perf <- defaultSummary(data.frame(obs = chem\_yield\_test, pred = chem\_pls\_pred))  
# return RMSEs for three models  
RMSEs(c("pls", "mars", "cube"))

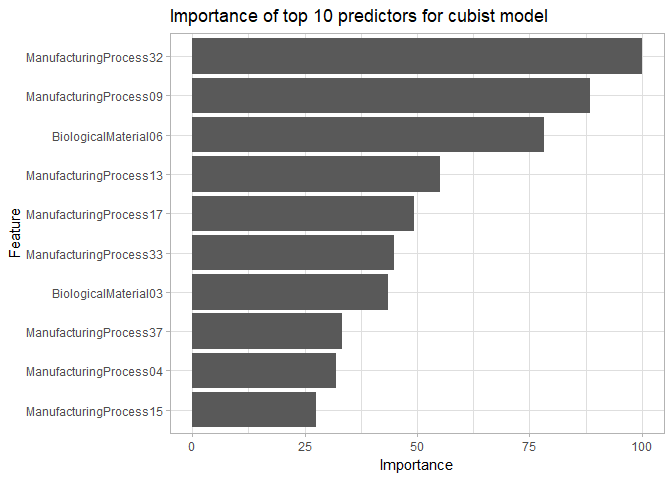
|  |  |  |
| --- | --- | --- |
|  | Resampled RMSE | Test RMSE |
| **pls** | 1.186 | 1.183 |
| **mars** | 1.158 | 1.162 |
| **cube** | 0.9677 | 1.005 |

Clearly, the Cubist model is the best-performing model of the eleven total models fit to the chemical manufacturing data.

## Predictor Importance & Relationships

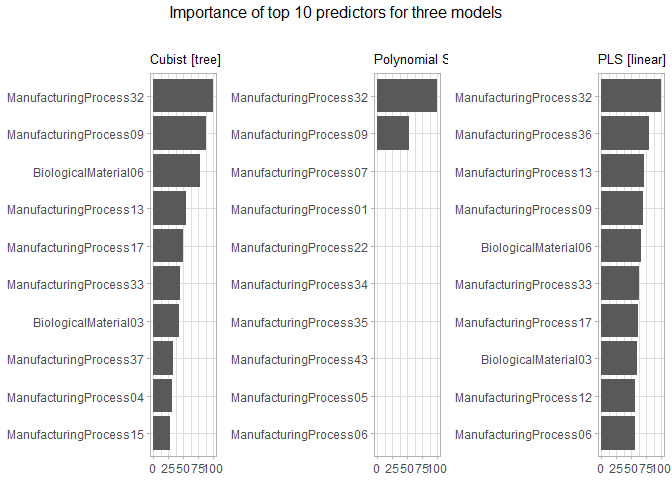
The importance of each predictor in the Cubist model is calculated, and the top 10 predictors are plotted below:

# get importance  
cube\_imp <- varImp(chem\_cube)  
# plot  
ggplot(cube\_imp, top = 10) + ggtitle("Importance of top 10 predictors for cubist model")



It is clear that *ManufacturingProcess32* is the most important predictor in this model. As with the linear model fit in problem 6.3 and the nonlinear model fit in problem 7.5, two of the 10 top predictors are biological; this ratio nearly matches the 21% share of total predictors available that are biological. For further investigation, variable importance is compared with the linear and nonlinear models:

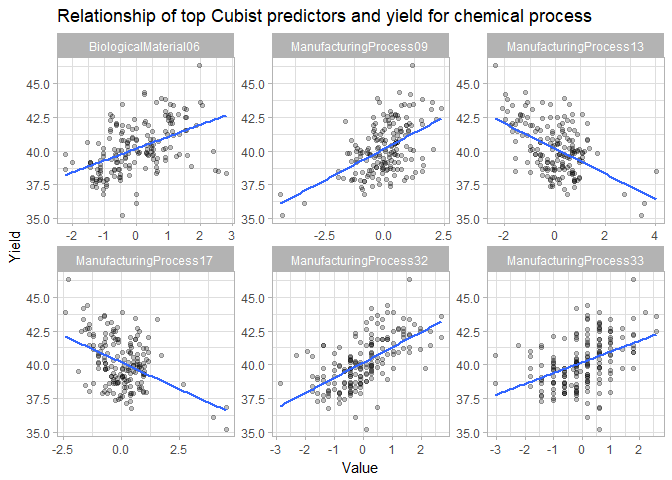
gridExtra::grid.arrange(top = "Importance of top 10 predictors for three models\n",  
 ggplot(cube\_imp, top = 10) +  
 labs(subtitle = "Cubist [tree]", x = NULL, y = NULL),  
 ggplot(mars\_imp, top = 10) +  
 labs(subtitle = "Polynomial SVM [non-linear]", x = NULL, y = NULL),  
 ggplot(pls\_imp, top = 10) +  
 labs(subtitle = "PLS [linear]", x = NULL, y = NULL),  
 nrow = 1)



For all three models, *ManufacturingProcess32* is the most important predictor. The Cubist model shares the top 2 predictors with both the MARS model, and shares 7 of 10 predictors with the PLS model.

The relationships between the top 6 predictors and the response *Yield* are shown below:

# get top 6 predictors  
cube\_top <- rownames(cube\_imp$importance)[order(cube\_imp$importance$Overall, decreasing = TRUE)][1:6]  
# separate top predictors & yield into data frame  
cube\_rel <- as.data.frame(cbind(chem\_pred\_trans, chem\_yield)[, c(cube\_top, "Yield")])  
# plot to investigate relationships  
cube\_rel %>%   
 # convert to tidy format  
 gather(Predictor, Value, -Yield) %>%   
 # plot  
 ggplot(aes(x = Value, y = Yield)) +  
 geom\_point(alpha = 0.25) +  
 stat\_smooth(se = FALSE, method = "glm") +  
 facet\_wrap(~ Predictor, nrow = 2, scales = "free") +  
 labs(title = "Relationship of top Cubist predictors and yield for chemical process")



Four of the six predictors have positive correlations with yield, including the most important predictor *ManufacturingProcess32* and the sole biological variable *BiologicalMaterial06*.

# Market Basket Analysis

library(arules)  
grocery <- read.transactions("data/GroceryDataSet.csv", sep = ",", format = "basket")  
summary(grocery)

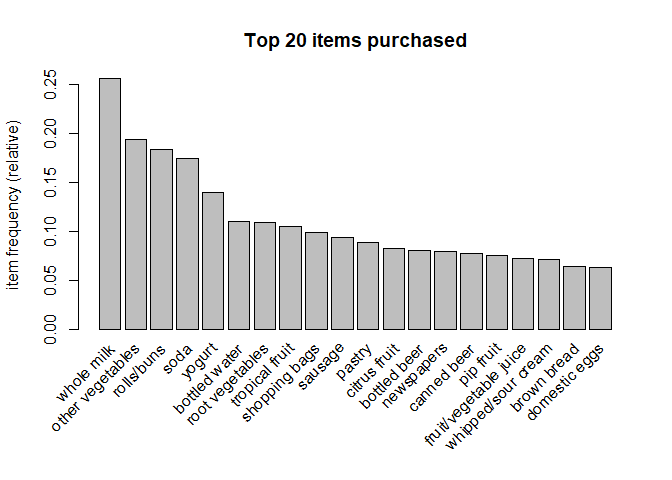
transactions as itemMatrix in sparse format with  
 9835 rows (elements/itemsets/transactions) and  
 169 columns (items) and a density of 0.02609146   
  
most frequent items:  
 whole milk other vegetables rolls/buns soda   
 2513 1903 1809 1715   
 yogurt (Other)   
 1372 34055   
  
element (itemset/transaction) length distribution:  
sizes  
 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15   
2159 1643 1299 1005 855 645 545 438 350 246 182 117 78 77 55   
 16 17 18 19 20 21 22 23 24 26 27 28 29 32   
 46 29 14 14 9 11 4 6 1 1 1 1 3 1   
  
 Min. 1st Qu. Median Mean 3rd Qu. Max.   
 1.000 2.000 3.000 4.409 6.000 32.000   
  
includes extended item information - examples:  
 labels  
1 abrasive cleaner  
2 artif. sweetener  
3 baby cosmetics

The data contain information on 9835 transactions across 169 possible items.

## Data Investigation

The top items by frequency are investigated:

itemFrequencyPlot(grocery, topN = 20, main = "\nTop 20 items purchased\n")



It can be seen that *whole milk* is the single most common purchased item, followed by *other vegetables*, *rolls/buns*, *soda*, and *yogurt*, after which frequencies slowly decline. The probability of these top five items occurring together is presented in the table below:

crossTable(grocery, measure = "support", sort = TRUE)[1:5, 1:5] %>%  
 pander(split.table = Inf, round = 3)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | whole milk | other vegetables | rolls/buns | soda | yogurt |
| **whole milk** | 0.256 | 0.075 | 0.057 | 0.04 | 0.056 |
| **other vegetables** | 0.075 | 0.193 | 0.043 | 0.033 | 0.043 |
| **rolls/buns** | 0.057 | 0.043 | 0.184 | 0.038 | 0.034 |
| **soda** | 0.04 | 0.033 | 0.038 | 0.174 | 0.027 |
| **yogurt** | 0.056 | 0.043 | 0.034 | 0.027 | 0.14 |

The table shows that the most commonly purchased item with *whole milk* is *other vegetables*, the second most commonly purchased item. In fact, for each item in the top 5, the most commonly purchased other item is *whole milk*.

## Association Rules

The arules package is used to mine the data for association rules. In order to do so, two parameters are set:

* The minimum support (i.e. the percent of transactions containing a given antecedent item & consequent item) required when identifying frequent item sets. The default setting of 0.1 will not yield many sets -- this is verified by running the command length(which(crossTable(grocery, measure = "support") > 0.1)). This value is set to support = 0.001.
* The minimum confidence (i.e. the percentage of transactions with a given antecendet item that also contain a given consequent item). Given the large number of items, this value is lowered from the default value of 0.8 to a value of confidence = 0.25.
* The minimum length of rules. To avoid single-item rules, this is set to minlen = 2

The rules are mined, using the verbose = FALSE argument to suppress output during mining:

grocery\_rules <- apriori(grocery, control = list(verbose = FALSE),  
 parameter = list(support = 0.001, confidence = 0.25, minlen = 2))

A summary of the rules reveals that 17,392 rules were generated, ranging in length from 2 to 6, with a median length of 4:

summary(grocery\_rules)

set of 17391 rules  
  
rule length distribution (lhs + rhs):sizes  
 2 3 4 5 6   
 367 6906 8371 1687 60   
  
 Min. 1st Qu. Median Mean 3rd Qu. Max.   
 2.000 3.000 4.000 3.665 4.000 6.000   
  
summary of quality measures:  
 support confidence lift count   
 Min. :0.001017 Min. :0.2500 Min. : 0.9784 Min. : 10.00   
 1st Qu.:0.001118 1st Qu.:0.3125 1st Qu.: 2.1557 1st Qu.: 11.00   
 Median :0.001322 Median :0.4016 Median : 2.7955 Median : 13.00   
 Mean :0.001914 Mean :0.4394 Mean : 3.0837 Mean : 18.83   
 3rd Qu.:0.001932 3rd Qu.:0.5397 3rd Qu.: 3.6654 3rd Qu.: 19.00   
 Max. :0.074835 Max. :1.0000 Max. :35.7158 Max. :736.00   
  
mining info:  
 data ntransactions support confidence  
 grocery 9835 0.001 0.25

Lift provides the ratio of confidence to the unconditional probability of consequents, indicating how informative the antecedents (LHS) of a rule are in predicting the consequent (RHS). The top rules ordered by lift are presented below:

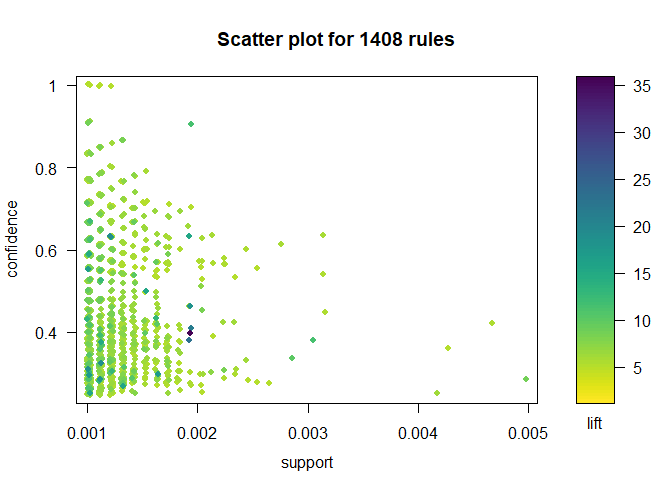
grocery\_rules\_top <- sort(grocery\_rules, by = "lift", decreasing = TRUE)[1:10]  
inspect(grocery\_rules\_top)

lhs rhs support   
[1] {bottled beer,red/blush wine} => {liquor} 0.001931876  
[2] {ham,white bread} => {processed cheese} 0.001931876  
[3] {bottled beer,liquor} => {red/blush wine} 0.001931876  
[4] {Instant food products,soda} => {hamburger meat} 0.001220132  
[5] {curd,sugar} => {flour} 0.001118454  
[6] {baking powder,sugar} => {flour} 0.001016777  
[7] {processed cheese,white bread} => {ham} 0.001931876  
[8] {fruit/vegetable juice,ham} => {processed cheese} 0.001118454  
[9] {margarine,sugar} => {flour} 0.001626843  
[10] {root vegetables,sugar,whole milk} => {flour} 0.001016777  
 confidence lift count  
[1] 0.3958333 35.71579 19   
[2] 0.3800000 22.92822 19   
[3] 0.4130435 21.49356 19   
[4] 0.6315789 18.99565 12   
[5] 0.3235294 18.60767 11   
[6] 0.3125000 17.97332 10   
[7] 0.4634146 17.80345 19   
[8] 0.2894737 17.46610 11   
[9] 0.2962963 17.04137 16   
[10] 0.2941176 16.91606 10

The top rule, with a lift of 35.72, is for a purchase of *liquor* after purchasing *bottled beer* & *red/blush wine*. This pattern appeared 19 times in the sample data. A different order of this combination, with *red/blush wine* as the right hand side; this rule has a lift of 21.49.

The support, confidence, and lift of all identified rules with a lift greater than 5 are presented below:

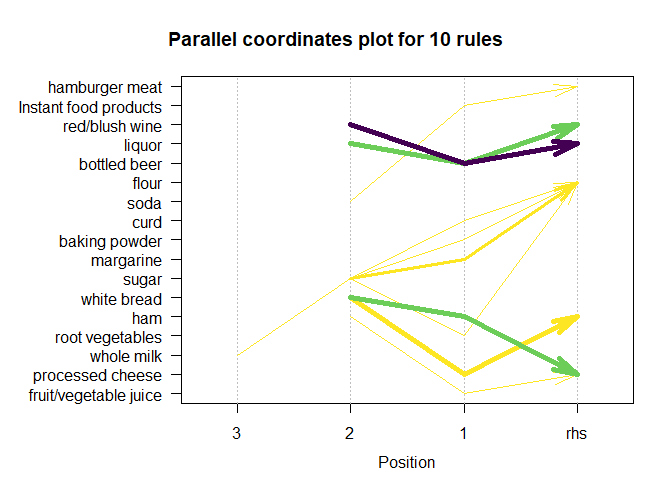
library(arulesViz)  
library(viridisLite)  
plot(subset(grocery\_rules, lift > 5), control = list(col = viridis(100)))



It can be seen that most of these 1408 rules, including those identified above as top rules, show support less than 0.002 and confidence less than 0.6; this indicates that the modification of the default parameters in the call to apriori helped uncover far more informative rules.

Finally, the parallel coordinates of the top ten rules is show below, with the color indicating the lift of each rule and the width indicating the count:

plot(grocery\_rules\_top, method = "paracoord", control = list(col = viridis(10), reorder = TRUE))



As discussed above, the two rules with *beer* in the second position exchanging *liquor* or *red/blush wine* as the first position or RHS have the highest frequencies and lifts.