Week4, Assignment 1

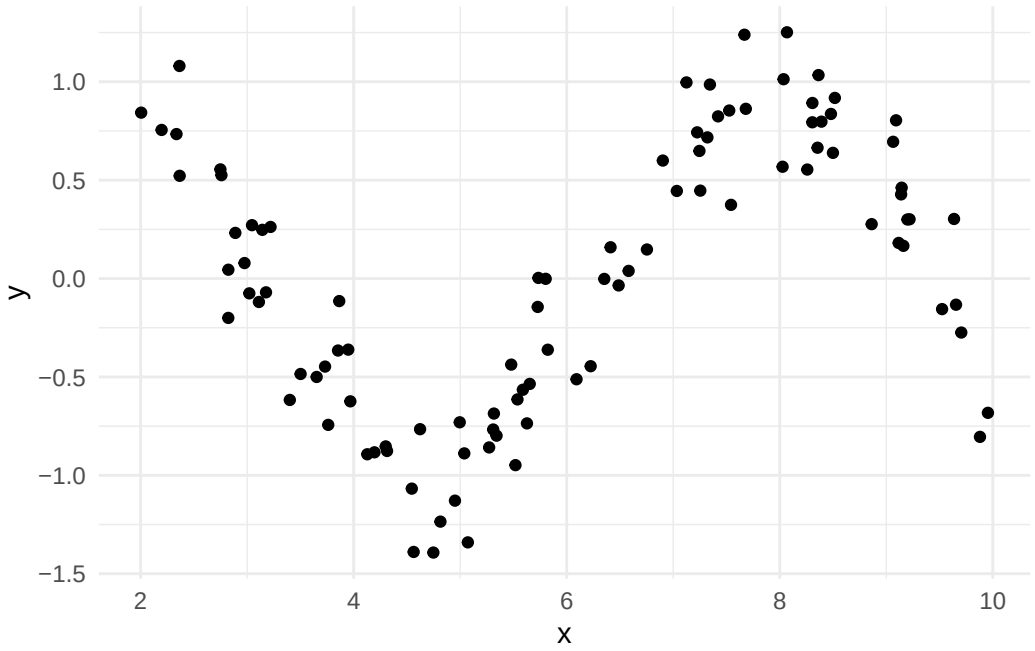
Gabi Rivera

```
library(caret)
library(kernlab)
library(tidyverse)
library(ggplot2)
library(naniar)
library(mice)
library(gt)
library(randomForest)
# ggplot
theme_set(theme_minimal())
seed <- 123
```

Problem 4.1 (20 points)

Simulate a single predictor and a nonlinear relationship, such as a sin wave shown in Fig. 7.7 of textbook, and investigate the relationship between the cost, , and kernel parameters for a support vector machine model

```
set.seed(seed)
sinData <- tibble(
    x = runif(100, min = 2, max = 10),
    y = sin(x) + rnorm(length(x)) * .25)
sinData |>
    ggplot(aes(x,y)) +
    geom_point()
```



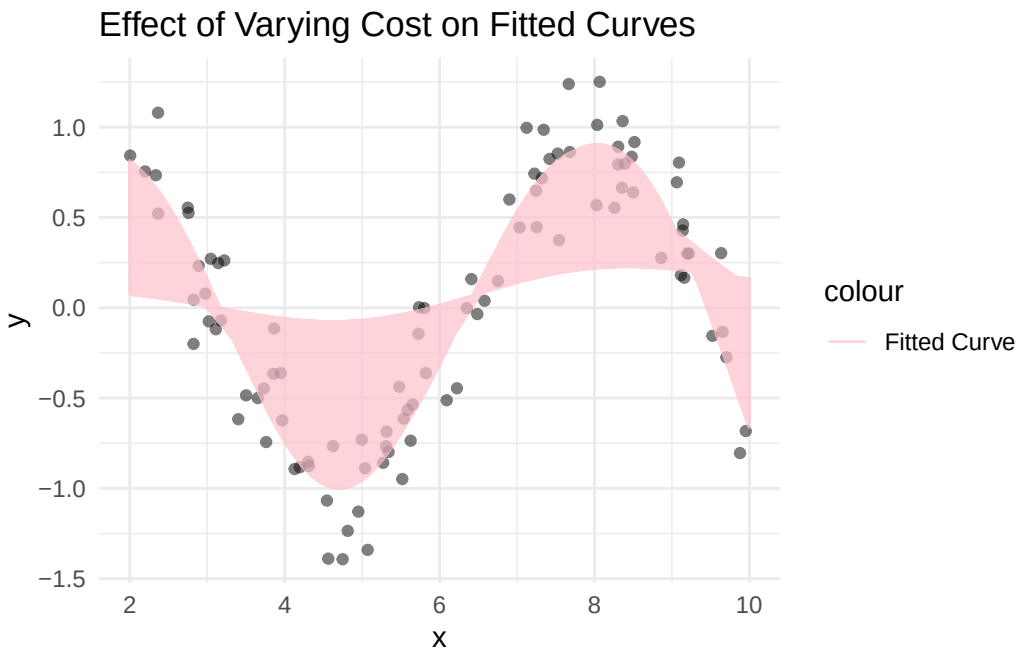

```
## Create a grid of x values to use for prediction
dataGrid <- tibble(x = seq(2, 10, length = 1000))
```

4.1.a (12 points)

Fit different models using a radial basis function and different values of the cost (the C parameter) and assigning sigma a constant value of 1. To study the impact of each parameter, keep all other parameters constant and vary only one parameter at a time. Plot the fitted curves and discuss what happens when cost and epsilon parameters are varied in terms of overfitting/underfitting.

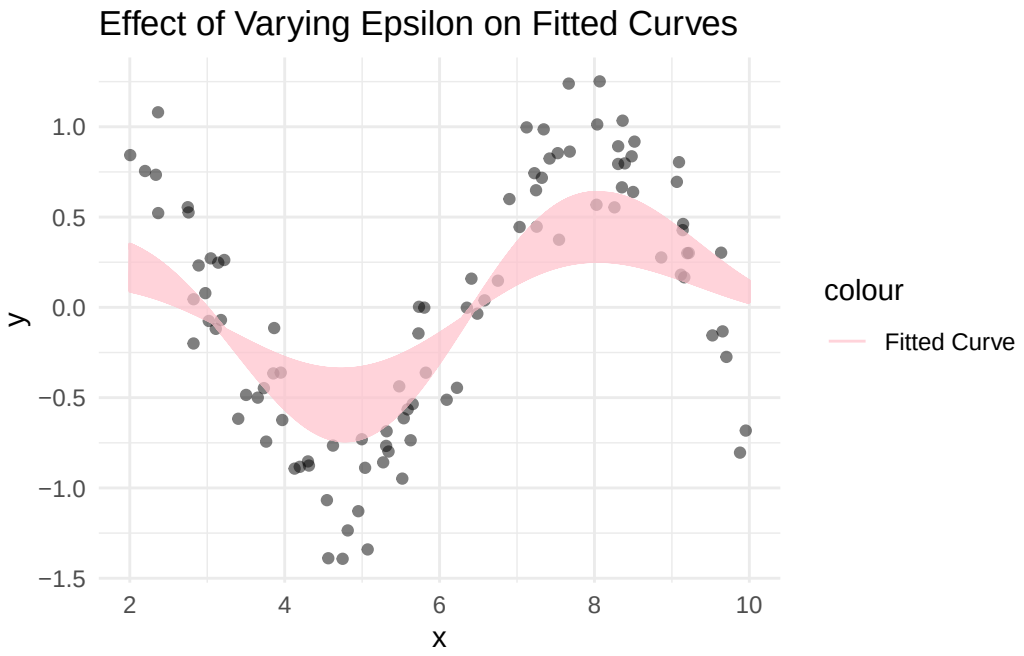
```
# Fit different models: Radial basis function
SVM_explore <- function(cost, epsilon,
                        data_grid = dataGrid, sin_data = sinData) {
  model <- ksvm(y ~ ., data = sin_data, type = "eps-svr", kernel = "rbfdot",
                C = cost, epsilon = epsilon, kpar = list(sigma = 1))
  predictions <- predict(model, data_grid)

  plot_data <- tibble(
    x = data_grid$x,
    y = predictions)
  return(plot_data)}
```

```r
cost_varies <- 10^seq(-2, 1, length = 4)
eps01_fixed <- rep(0.1, 4)
cost_curves <- map2_dfr(cost_varies,
                        eps01_fixed,
                        ~ SVM_explore(.x, .y))

# Plot curves for varying cost
cost_curves |>
  ggplot(aes(x, y)) +
  geom_point(data = sinData, aes(x, y), alpha = 0.5) +
  geom_line(aes(color = "Fitted Curve"), alpha = 0.7) +
  scale_color_manual(values = c("Fitted Curve" = "pink", "Actual Data" = "darkgray")) +
  labs(title = "Effect of Varying Cost on Fitted Curves")
```




```r
eps_varies <- c(0.01, 0.1, 0.5, 1.0)
cost_fixed <- rep(0.1, 4)
eps_curves <- map2_dfr(cost_fixed,
                        eps_varies,
                        ~ SVM_explore(.x, .y))

# Plot curves for varying epsilon
eps_curves |>
  ggplot(aes(x, y)) +
```

```
  geom_point(data = sinData, aes(x, y), alpha = 0.5) +
  geom_line(aes(color = "Fitted Curve"), alpha = 0.7) +
  scale_color_manual(values = c("Fitted Curve" = "pink", "Actual Data" = "darkgray")) +
  labs(title = "Effect of Varying Epsilon on Fitted Curves")
```



Varying cost looks like it produces high fits for high C values and under fits for lower C values.

Looks like it's the opposite for varying epsilon, high epsilon causes a larger deviation from the regression line and lower epsilon have lower bias. But the fit at lower bias is not as over fitted as high cost.

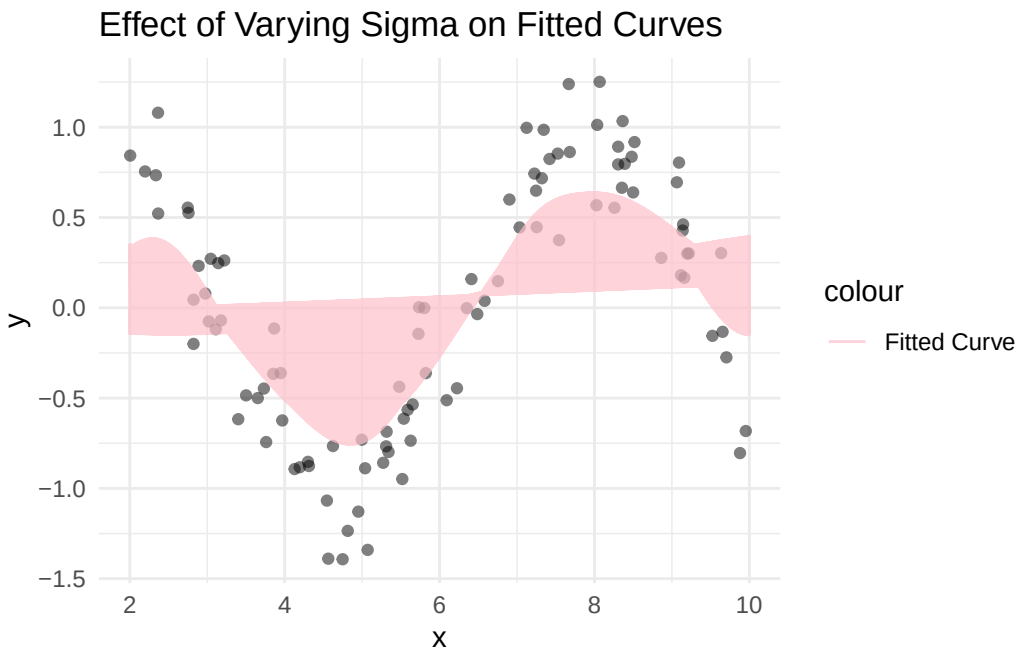
4.1.b (8 points)

The parameter can be adjusted using the kpar argument, such as kpar = list(sigma = 1). Try different values of to understand how this parameter changes the model fit. How do the cost, , and values affect the model?

Hint: you'll need to modify the `SVM_explore` function above

```
SVM_explore3 <- function(cost = 0.1, epsilon = 0.1, sigma = 1,
                         data_grid = dataGrid, sin_data = sinData) {
  model <- ksvm(y ~ ., data = sin_data, type = "eps-svr", kernel = "rbfdot",
                C = cost, epsilon = epsilon, kpar = list(sigma = sigma))
  predictions <- predict(model, data_grid)
```

```r
  plot_data <- tibble(
    x = data_grid$x,
    y = predictions)
  return(plot_data)}

sigma_varies <- 10^seq(-2, 1, length = 4)
sigma_curves <- map_dfr(sigma_varies,
                        ~ SVM_explore3(sigma = .x))

# Plot curves for varying sigma
ggplot(sigma_curves, aes(x, y)) +
  geom_point(data = sinData, aes(x, y), alpha = 0.5) +
  geom_line(aes(color = "Fitted Curve"), alpha = 0.7) +
  scale_color_manual(values = c("Fitted Curve" = "pink", "Actual Data" = "darkgray")) +
  labs(title = "Effect of Varying Sigma on Fitted Curves")
```



Varying sigma is similar to the pattern of varying cost, higher sigma looks like more fitted decision boundaries and lower sigma have the opposite with higher bias.

Problem 4.2 (30 points)

Returning to the chemical manufacturing process data (from Module 3 Assignment Problem 3.3):

```
library(AppliedPredictiveModeling)
data(ChemicalManufacturingProcess)
```

Split the data into a training (80%) and a test set (20%). A small percentage of cells in the predictor set contain missing values. Use an imputation function to fill in these missing values in both training and test data sets, also perform centering and scaling. Build and tune nonlinear regression models Neural network, MARS, SVM and KNN.

```
set.seed(seed)
chem_predictors <- ChemicalManufacturingProcess[ ,-1]
chem_yield <- ChemicalManufacturingProcess[ ,1]

# Split the data into training and test sets
chem_index <- createDataPartition(chem_yield, p = 0.8, list = FALSE)
chem_train <- chem_predictors[chem_index, ]
chem_test <- chem_predictors[-chem_index, ]

yield_train <- chem_yield[chem_index ]
yield_test <- chem_yield[-chem_index ]

# Imputation and scaling using preProcess
chem_prep <- preProcess(chem_train, method = c("center", "scale", "medianImpute"))

# Apply the transformation to the training data
chem_train_transformed <- predict(chem_prep, chem_train)
chem_test_transformed <- predict(chem_prep, chem_test)
```

Model fits

```
set.seed(123)
indx <- createFolds(yield_train, returnTrain = TRUE)
ctrl <- trainControl(method = "cv", index = indx)

# Build neural network Model: Optimized
nnetGrid <- expand.grid(decay = c(0, 0.01, 0.1), size = c(3, 7, 11, 13))
nnetTune <- train(x = chem_train_transformed, y = yield_train,
                  method = "nnet",
                  tuneGrid = nnetGrid,
                  trControl = ctrl,
```

```
                  linout = TRUE,
                  trace = FALSE,
                  MaxNWts = 13 * (ncol(chem_train_transformed) + 1) + 13 + 1,
                  maxit = 1000)
#plot(nnetTune)
opt_nnetd <- nnetTune$bestTune$decay
opt_nnets <- nnetTune$bestTune$size
nnet_model <- train(x = chem_train_transformed,y = yield_train,
                    method = "nnet",
                    tuneGrid = expand.grid(decay = opt_nnetd,
                                           size = opt_nnets),
                    trControl = ctrl,
                    linout = TRUE,
                    trace = FALSE,
                    MaxNWts = 13 * (ncol(chem_train_transformed) + 1) + 13 + 1,
                    maxit = 1000)

# Build MARS Model: Optimized
marsGrid <- expand.grid(degree = 1, nprune = 2:20)
marsTune <- train(x = chem_train_transformed, y = yield_train,
                  method = "earth",
                  tuneGrid = marsGrid,
                  trControl = ctrl)
#plot(marsTune)
opt_marsd <- marsTune$bestTune$degree
opt_marsn <- marsTune$bestTune$nprune
mars_model <- train(x = chem_train_transformed, y = yield_train,
                  method = "earth",
                  tuneGrid = expand.grid(degree = opt_marsd,
                                         nprune = opt_marsn),
                  trControl = ctrl)

# Build SVM Radial Model: Optimzied
svmRTune <- train(x = chem_train_transformed, y = yield_train,
                  method = "svmRadial",
                  tuneLength = 14,
                  trControl = ctrl)
#plot(svmRTune, scales = list(x = list(log = 2)))
opt_svmRs <- svmRTune$bestTune$sigma
opt_svmRC <- svmRTune$bestTune$C
svmR_model <- train(x = chem_train_transformed, y = yield_train,
                  method = "svmRadial",
```

```
                  tuneGrid = expand.grid(sigma = opt_svmRs,
                                         C = opt_svmRC),
                  trControl = ctrl)

# Build KNN Model: Optimized
knnDescr <- chem_train_transformed[, -nearZeroVar(chem_train_transformed)]
knnTune <- train(x = knnDescr, y = yield_train,
                 method = "knn",
                 tuneGrid = data.frame(k = 1:20),
                 trControl = ctrl)
#plot(knnTune)
opt_knn <- knnTune$bestTune$k
knn_model <- train(x = knnDescr, y = yield_train,
                 method = "knn",
                 tuneGrid = expand.grid(k = opt_knn),
                 trControl = ctrl)
```

4.2.a (20 point)

Which nonlinear regression model gives the optimal resampling and test set performance (use RMSE metric to make this determination)?

```
# Nonlinear regression optimal model's RMSE score
nnet_optimal_rmse <- nnet_model$results$RMSE
mars_optimal_rmse <- mars_model$results$RMSE
svmR_optimal_rmse <- svmR_model$results$RMSE
knn_optimal_rmse <- knn_model$results$RMSE

optimal_RMSE <- c(nnet_optimal_rmse, mars_optimal_rmse,svmR_optimal_rmse, knn_optimal_rmse)

# Use optimized nonlinear regression models to predict test dataset's outcome
predictions <- list(nnet = predict(nnet_model, newdata = chem_test_transformed),
                    mars = predict(mars_model, newdata = chem_test_transformed),
                    svmR = predict(svmR_model, newdata = chem_test_transformed),
                    knn = predict(knn_model, newdata = chem_test_transformed))

# Calculate RMSE on test data set
calc_rmse <- function(actual, predicted) {
sqrt(mean((actual - predicted)^2))}
rmse_scores <- sapply(predictions, function(pred)
calc_rmse(yield_test, pred))
```

```
# Table RMSE results
model_names <- c("NNET", "MARS", "SVM Radial", "KNN")
optimal_RMSE_df <- data.frame(Model = model_names,opt_Test_RMSE = optimal_RMSE,
                              Test_RMSE = rmse_scores)
optimal_RMSE_df |> gt() |>
  tab_header(title = "RMSE Score for Each Model")
```

RMSE Score for Each Model

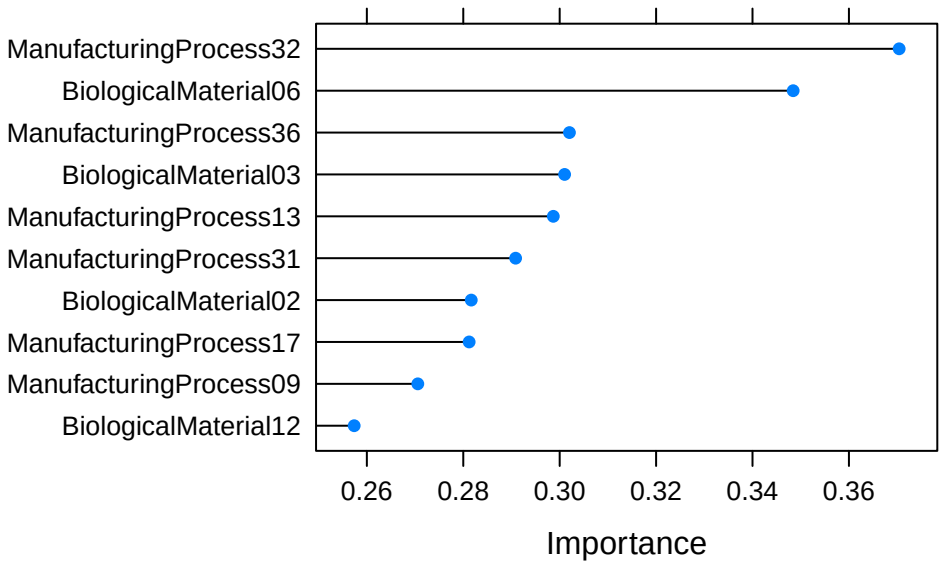
Model	opt_Test_RMSE	Test_RMSE
NNET	2.765662	1.600317
MARS	1.202007	1.302778
SVM Radial	1.088563	1.224386
KNN	1.199076	1.409359

SVM Radial model is the nonlinear regression that performed the best having the lowest RMSE for both the optimal resampling and test sets.

4.2.b (5 point)

What are the ten most important predictors in the optimal nonlinear regression model? Do either the biological or process variables dominate the list?

```
# Ten most important predictors in the SVM Radial
svmRImp <- varImp(svmRTune, scale = FALSE)
plot(svmRImp, top = 10)
```



The ten most important predictors are listed on the plot above and ManufacturingProcess dominates at 60% .

4.2.c (5 point)

How do the important predictors compare to the important predictors from the optimal linear model you built in module 3 assignment? (complete the table below)

Predictor	SVM Rank (1-10)	Lasso Rank (1-5)
ManufacturingProcess32	1	1
BiologicalMaterial06	2	4
ManufacturingProcess36	3	Empty
BiologicalMaterial03	4	Empty
ManufacturingProcess13	5	Empty
ManufacturingProcess31	6	Empty
BiologicalMaterial02	7	Empty
ManufacturingProcess17	8	3
ManufacturingProcess09	9	2
BiologicalMaterial12	10	Empty
ManufacturingProcess06	Empty	5

Problem 4.3 (30 points)

Recreate the simulated data as shown below:

```
library(mlbench)
set.seed(200)
simulated <- mlbench.friedman1(200, sd = 1)
simulated <- cbind(simulated$x, simulated$y)
simulated <- as.data.frame(simulated)
colnames(simulated)[ncol(simulated)] <- "y"
```

In this data set V6 to V10 are columns that are random noise and have no relation to the response.

4.3.a (10 points)

Fit a random forest model to all of the predictors, then estimate the variable importance scores

```
set.seed(seed)

# Fit random forest model
model1 <- randomForest(y ~ ., data = simulated, importance = TRUE, ntree = 1000)

# Table variable importance values
importance1 <- round(varImp(model1), 4)
importance1 <- cbind(Predictor = rownames(importance1), importance1)
importance1 |> gt() |>
  tab_header(title = "Variable Importance Scores")
```

Variable Importance Scores

Predictor	Overall
V1	55.9234
V2	45.4511
V3	10.5825
V4	55.8112
V5	22.9747
V6	1.8927
V7	1.7614
V8	-4.1274
V9	-1.9323
V10	-0.1111

Did the random forest model significantly use the uninformative predictors (V6 – V10)? What do you think the impact of having uninformative predictors will be on random forest model performance?

It seems that random forest model used the uninformative predictors (V6-V10) the least. They are all at the bottom rank of importance score. Basing from this data, it looks like random forest is a bit more robust in handling uninformative predictors. It might not be entirely significant if there is no choice.

4.3.b (10 points)

Now add an additional predictor that is highly correlated with one of the informative predictors, V1.

```
simulated$duplicate1 <- simulated$V1 + rnorm(200) * .1
cor(simulated$duplicate1, simulated$V1)
```

```
[1] 0.9434714
```

Fit another random forest model to these data. Did the importance score for V1 change?

```
# Fit another random forest model
model2 <- randomForest(y ~ ., data = simulated, importance = TRUE, ntree = 1000)

# Table variable importance values
importance2 <- round(varImp(model2), 4)
importance2 <- cbind(Predictor = rownames(importance2), importance2)
importance2 |> head(1) |> gt() |>
  tab_header(title = "Variable Importance Scores")
```

Variable Importance Scores

Predictor	Overall
V1	32.0422

Yes, the importance score for V1 changed from 55.9 down to 33.98.

What happens if you add one more predictor to the data set that is also highly correlated with V1 (after this you should have 12 predictors, 10 original and two added correlated predictors) and fit another random forest model again to this data. Did the importance score for V1 change?

```r
# Add another V1 highly correlated predictor
simulated$duplicate2 <- simulated$V2 + rnorm(200) * .01
cor(simulated$duplicate2, simulated$V2)
```

```
[1] 0.9992561
```

```r
# Fit another random forest model
model3 <- randomForest(y ~ ., data = simulated, importance = TRUE, ntree = 1000)

# Table variable importance values
importance3 <- round(varImp(model3), 4)
importance3 <- cbind(Predictor = rownames(importance3), importance3)
importance3 |> gt() |>
  tab_header(title = "Variable Importance Scores")
```

Variable Importance Scores

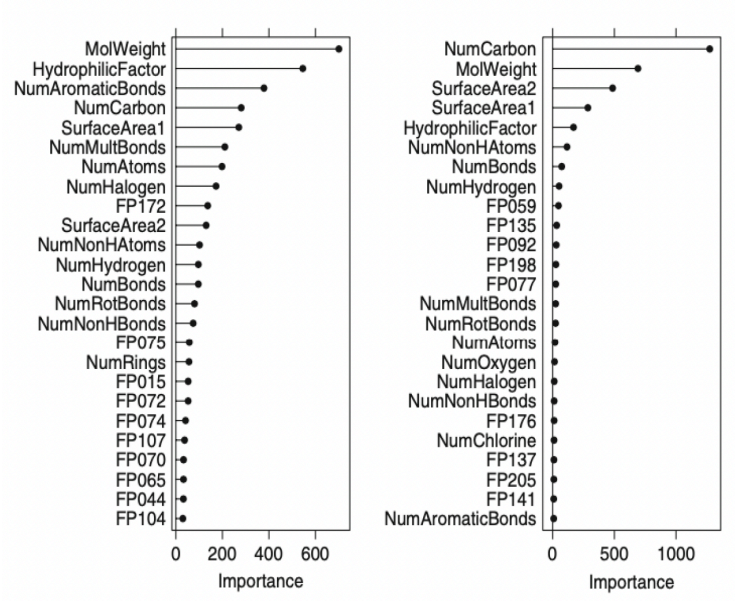
Predictor	Overall
V1	35.3320
V2	28.3645
V3	10.9994
V4	56.9712
V5	24.2447
V6	2.8329
V7	0.3768
V8	-3.1496
V9	-3.3485
V10	0.4034
duplicate1	25.9893
duplicate2	31.1724

The importance score of V1 changed but this time from 55.9 original data to 37.0 which is a new score that's higher than duplicate 1 at 33.98.

Problem 4.4 (20 points)

Figure 8.24 provides the variable importance plots for boosting using two extreme values for the bagging fraction (0.1 and 0.9) and the learning rate (0.1 and 0.9) for the solubility data.

The left-hand plot has both parameters set to 0.1, and the right-hand plot has both set to 0.9:



4.3.a (7 points)

Why does the model on the right focus its importance on just the first few of predictors, whereas the model on the left spreads importance across more predictors?

The combination of high bagging fraction and high learning rate results in a more aggressive fit where the first few predictors gets prioritized. The use of 90% subsample reduces the diversity during training as most tree see similar patterned subsets. Higher learning rate leads to larger influence of newly added tree in the ensemble. Combination of high bagging fraction and high learning rate increases the risk of overfitting and the result concentrates importance in a smaller subset of predictors that provide the highest predictive power. While low bagging fraction and low learning rate spreads importance across more predictors by having a more conservative fitting process. Low bagging increases the diversity among the trees and enable new trees added to have smaller contribution. All these allow more features to have influence to the final model.

4.3.b (7 points)

Which model do you think would be more predictive of other samples?

The left model might be more predictive of other samples because it spreads importance across more predictors. It has the potential to capture a boarder range of patterns and can generalize better.

4.3.c (6 points)

How would increasing interaction depth affect the slope of predictor importance for either model in the figure below?

Interaction depth refers to the maximum number of splits or interactions between predictors. Increasing depth for high bagging and high learning rate may lead to increased or steeper slope, inflating the importance of predictors due to complexed trees. Same goes for low bagging and low learning rate but the complexity might be less pronounced since the model is more conservative and less likeliness to over fit.