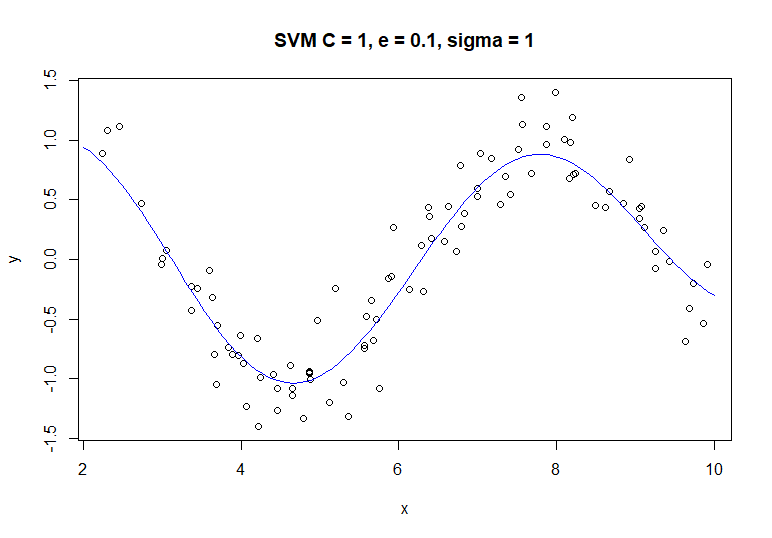
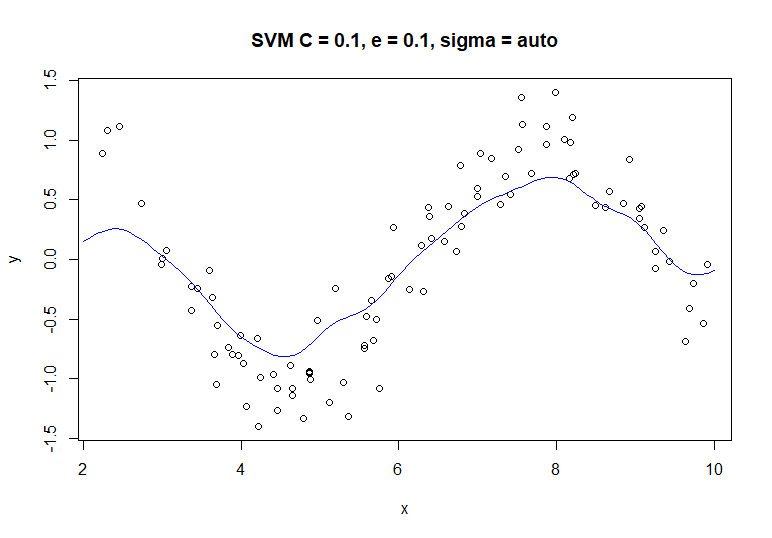
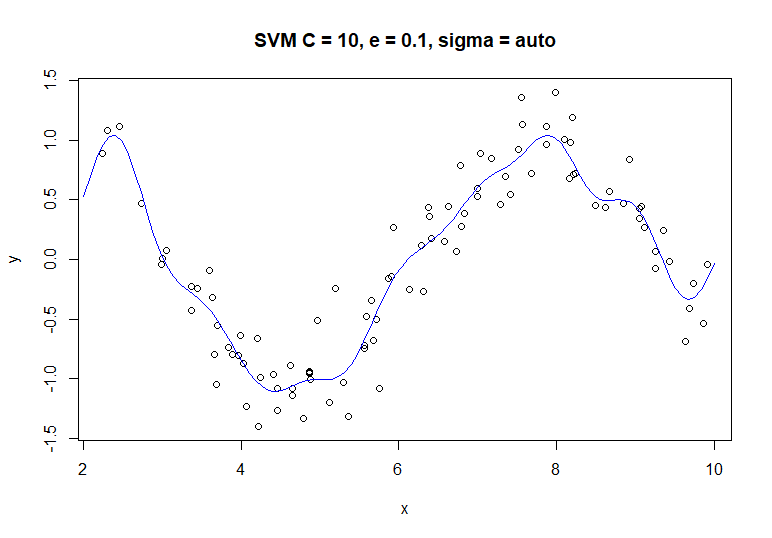
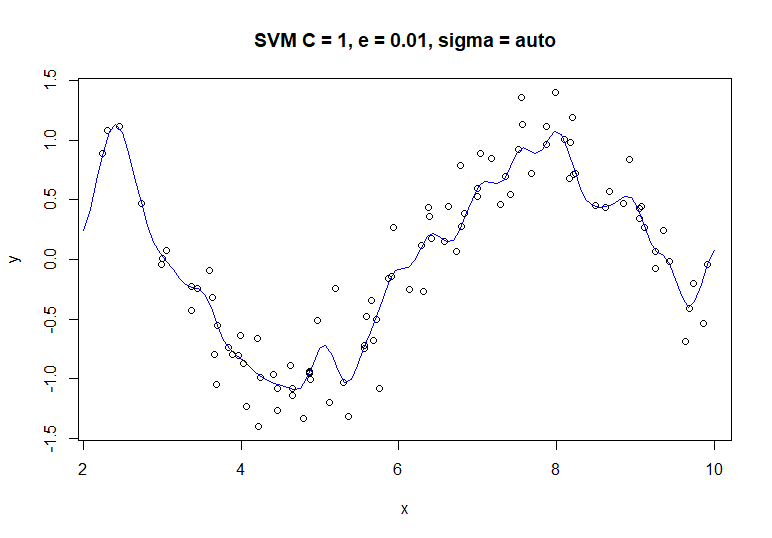
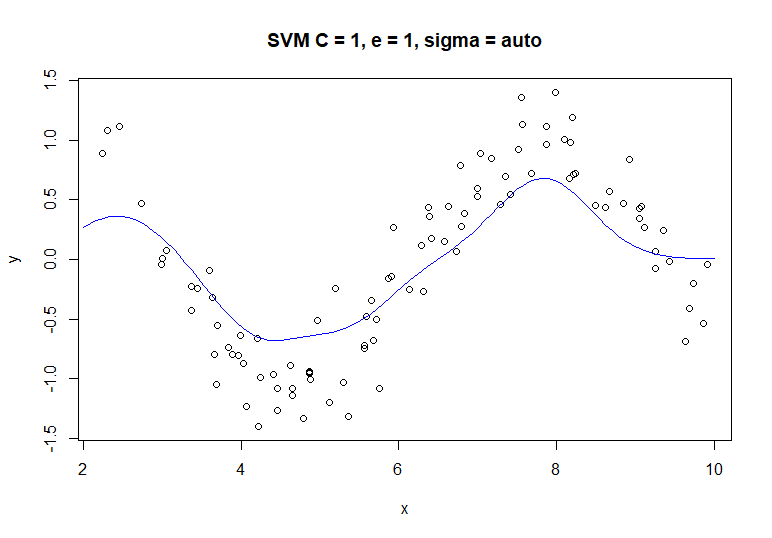
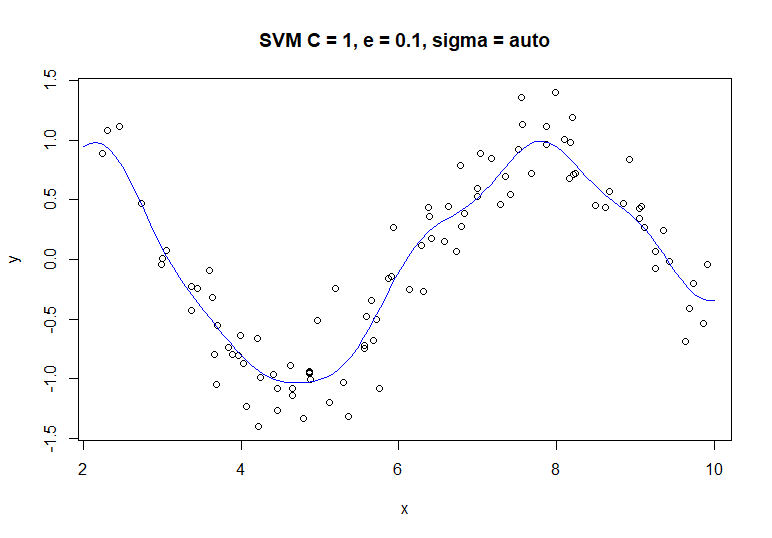
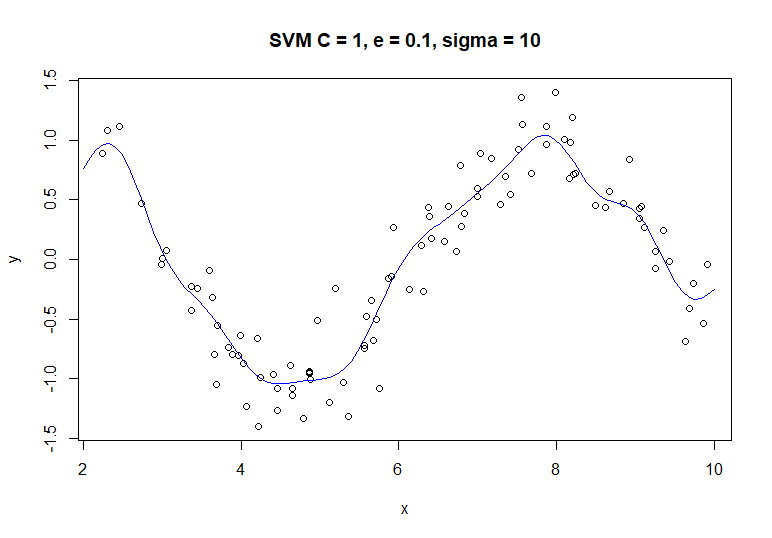
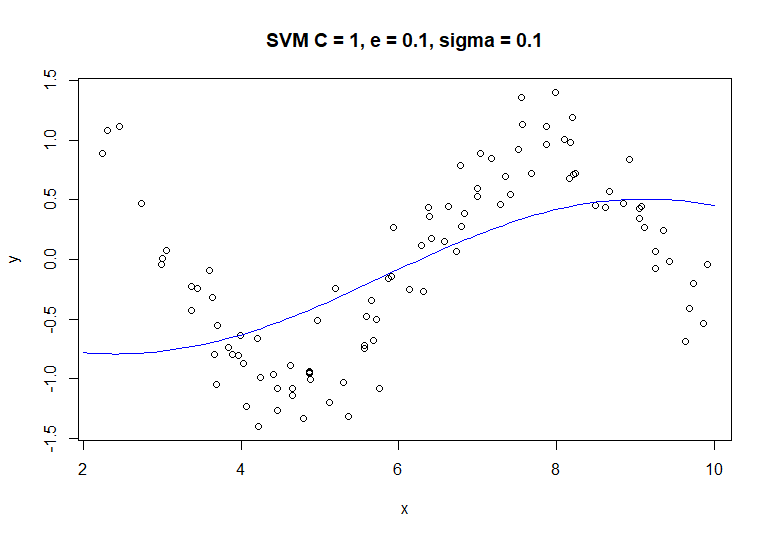
Deanna Springgay

Ch 7 Homework

7.1





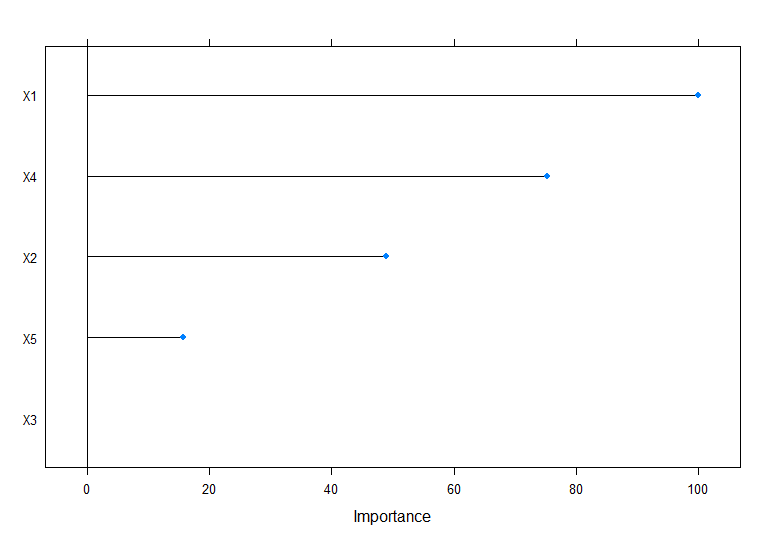
Based on the previous plots, the cost parameter effects penalties on large residuals: a higher cost value leads to less penalties, higher complexity, and likely to over-fit. ε is the margin of tolerance for penalties, a higher ε leads to a smaller margin, though has less of an effect on penalties than the cost parameter. σ affects the variance and bias in the model: a larger σ leads to less variance and more bias, which can lead to overfitting.

7.2

MARS appears to give a better performance than KNN as it has a lower RMSE value on the testing data as shown in the table below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **Best Tuning Parameter** | **Training** | | **Testing** | |
| **RMSE** | **R²** | **RMSE** | **R²** |
| KNN | k = 17 | 3.349428 | 0.5452823 | 3.2040595 | 0.6819919 |
| MARS | nprune = 13  degree = 2 | 1.261130 | 0.9357469 | 1.2803060 | 0.9335241 |

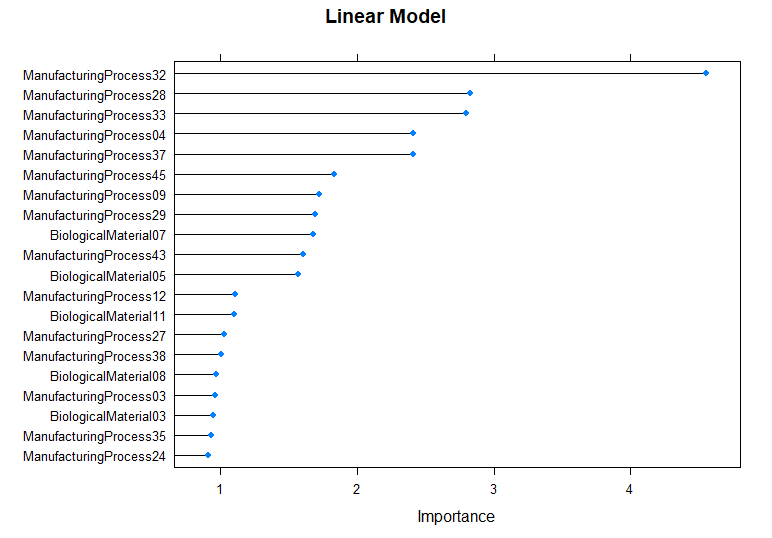
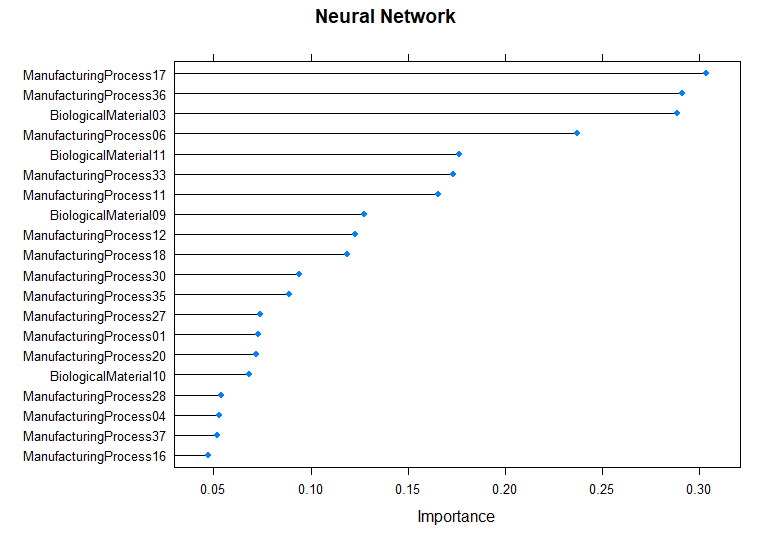
MARS does not select all the informative predictors, seeing as X3 has an importance of zero as shown by the following plot:



7.5

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **Best Tuning Parameter** | **Training** | | **Testing** | |
| **RMSE** | **R²** | **RMSE** | **R²** |
| Neural Network | size = 8  decay = 0.1  bag = FALSE | 0.7520576 | 0.4805693 | 0.24868980 | 0.93915322 |
| MARS | nprune = 7  degree = 1 | 0.6600063 | 0.5686642 | 0.6296545 | 0.6023783 |
| SVM | sigma = 0.01906204  C = 4 | 0.6495206 | 0.6135844 | 0.3166915 | 0.9035918 |
| KNN | k = 4 | 0.6735640 | 0.5716295 | 0.5544946 | 0.7096414 |

The neural network is the best model out of the group, and the top predictors are a mix of manufacturing and biological processes. The linear model has primarily manufacturing processes as the most important predictors, so there is some difference between the two models.



R Code

library(AppliedPredictiveModeling)

library(caret)

library(earth)

library(kernlab)

library(mlbench)

library(nnet)

library(tidyverse)

####7.1####

set.seed(100)

x <- runif(100, min = 2, max = 10)

y <- sin(x) + rnorm(length(x)) \* .25

sinData <- data.frame(x = x, y = y)

plot(x, y)

## Create a grid of x values to use for prediction

dataGrid <- data.frame(x = seq(2, 10, length = 100))

####7.1a####

#example

rbfSVM <- ksvm(x = x, y = y, data = sinData, kernel ="rbfdot", kpar = "automatic", C = 1, epsilon = 0.1)

modelPrediction <- predict(rbfSVM, newdata = dataGrid)

plot(x, y, main = "SVM C = 1, e = 0.1, sigma = auto")

points(x = dataGrid$x, y = modelPrediction[,1], type = "l", col = "blue")

#constant C, change epsilon

rbfSVM <- ksvm(x = x, y = y, data = sinData, kernel ="rbfdot", kpar = "automatic", C = 1, epsilon = 1)

modelPrediction <- predict(rbfSVM, newdata = dataGrid)

plot(x, y, main = "SVM C = 1, e = 1, sigma = auto")

points(x = dataGrid$x, y = modelPrediction[,1], type = "l", col = "blue")

rbfSVM <- ksvm(x = x, y = y, data = sinData, kernel ="rbfdot", kpar = "automatic", C = 1, epsilon = 0.01)

modelPrediction <- predict(rbfSVM, newdata = dataGrid)

plot(x, y, main = "SVM C = 1, e = 0.01, sigma = auto")

points(x = dataGrid$x, y = modelPrediction[,1], type = "l", col = "blue")

#change C, constant epsilon

rbfSVM <- ksvm(x = x, y = y, data = sinData, kernel ="rbfdot", kpar = "automatic", C = 10, epsilon = 0.1)

modelPrediction <- predict(rbfSVM, newdata = dataGrid)

plot(x, y, main = "SVM C = 10, e = 0.1, sigma = auto")

points(x = dataGrid$x, y = modelPrediction[,1], type = "l", col = "blue")

rbfSVM <- ksvm(x = x, y = y, data = sinData, kernel ="rbfdot", kpar = "automatic", C = 0.1, epsilon = 0.1)

modelPrediction <- predict(rbfSVM, newdata = dataGrid)

plot(x, y, main = "SVM C = 0.1, e = 0.1, sigma = auto")

points(x = dataGrid$x, y = modelPrediction[,1], type = "l", col = "blue")

####7.1b####

rbfSVM <- ksvm(x = x, y = y, data = sinData, kernel ="rbfdot", kpar = list(sigma = 1), C = 1, epsilon = 0.1)

modelPrediction <- predict(rbfSVM, newdata = dataGrid)

plot(x, y, main = "SVM C = 1, e = 0.1, sigma = 1")

points(x = dataGrid$x, y = modelPrediction[,1], type = "l", col = "blue")

rbfSVM <- ksvm(x = x, y = y, data = sinData, kernel ="rbfdot", kpar = list(sigma = 0.1), C = 1, epsilon = 0.1)

modelPrediction <- predict(rbfSVM, newdata = dataGrid)

plot(x, y, main = "SVM C = 1, e = 0.1, sigma = 0.1")

points(x = dataGrid$x, y = modelPrediction[,1], type = "l", col = "blue")

rbfSVM <- ksvm(x = x, y = y, data = sinData, kernel ="rbfdot", kpar = list(sigma = 10), C = 1, epsilon = 0.1)

modelPrediction <- predict(rbfSVM, newdata = dataGrid)

plot(x, y, main = "SVM C = 1, e = 0.1, sigma = 10")

points(x = dataGrid$x, y = modelPrediction[,1], type = "l", col = "blue")

####7.2####

set.seed(200)

trainingData <- mlbench.friedman1(200, sd = 1)

trainingData$x <- data.frame(trainingData$x)

featurePlot(trainingData$x, trainingData$y)

testData <- mlbench.friedman1(5000, sd = 1)

testData$x <- data.frame(testData$x)

#example

knnModel <- train(x = trainingData$x, y = trainingData$y, method = "knn", preProc = c("center", "scale"), tuneLength = 10)

knnModel

knnPred <- predict(knnModel, newdata = testData$x)

postResample(pred = knnPred, obs = testData$y)

marsGrid <- expand.grid(.degree = 1:2, .nprune = 2:50)

marsModel <- train(x = trainingData$x, y = trainingData$y,

method = "earth", tuneGrid = marsGrid, trControl = trainControl(method = "cv"))

marsModel

marsPred <- predict(marsModel, newdata = testData$x)

postResample(pred = marsPred, obs = testData$y)

plot(varImp(marsModel), top = 5)

####7.5####

set.seed(80)

data(ChemicalManufacturingProcess)

?ChemicalManufacturingProcess

#yield is outcome

#imputing missing values

imputedManufacturing <- predict(preProcess(ChemicalManufacturingProcess, "knnImpute"), ChemicalManufacturingProcess)

#splitting data

trainingRows <- createDataPartition(imputedManufacturing[,1], p=0.80, list=FALSE)

training <- imputedManufacturing[trainingRows,]

testing <- imputedManufacturing[-trainingRows,]

ctrl <- trainControl(method = "cv", number = 10)

xTest <- imputedManufacturing[,2:58]

#Neural Network

tooHigh <- findCorrelation(cor(training), cutoff = .75)

trainXnnet <- training[, -tooHigh]

testXnnet <- testing[, -tooHigh]

zeroVar <- nearZeroVar(trainXnnet)

trainXnnet <- trainXnnet[, -zeroVar]

testXnnet <- testXnnet[, -zeroVar]

nnetGrid <- expand.grid(.decay = c(0, 0.01, .1), .size = c(1:10), .bag = FALSE)

nnFit <- train(Yield ~ ., data = trainXnnet,

method = "avNNet",

tuneGrid = nnetGrid,

trControl = ctrl,

preProc = c("center", "scale"),

linout = TRUE,

trace = FALSE,

MaxNWts = 10 \* (ncol(trainXnnet) + 1) + 10 + 1,

maxit = 500)

nnFit

predicted <- predict(nnFit, xTest)

nnValues <- data.frame(obs = imputedManufacturing[,1], pred = predicted)

defaultSummary(nnValues)

#MARS

marsGrid <- expand.grid(.degree = 1:2, .nprune = 2:50)

marsFit <- train(Yield ~ ., data = trainXnnet,

method = "earth",

tuneGrid = marsGrid,

trControl = trainControl(method = "cv"))

marsFit

predicted <- predict(marsFit, xTest)

marsValues <- data.frame(obs = imputedManufacturing[,1], pred = predicted)

colnames(marsValues) <- c("obs", "pred")

defaultSummary(marsValues) #not working

#SVM

svmFit <- train(Yield ~ ., data = trainXnnet,

method = "svmRadial",

preProc = c("center", "scale"),

tuneLength = 14,

trControl = trainControl(method = "cv"))

svmFit

predicted <- predict(svmFit, xTest)

svmValues <- data.frame(obs = imputedManufacturing[,1], pred = predicted)

defaultSummary(svmValues)

#KNN

knnDescr <- training[, -nearZeroVar(training)]

knnFit <- train(Yield ~ ., data = knnDescr,

method = "knn",

preProc = c("center", "scale"),

tuneGrid = data.frame(.k = 1:20),

trControl = trainControl(method = "cv"))

knnFit

predicted <- predict(knnFit, xTest)

knnValues <- data.frame(obs = imputedManufacturing[,1], pred = predicted)

defaultSummary(knnValues)

#best model lowest RMSE: nn

nnImp <- varImp(nnFit, scale = FALSE)

nnImp

plot(nnImp, top = 20, main = "Neural Network")