Module 4 exercises

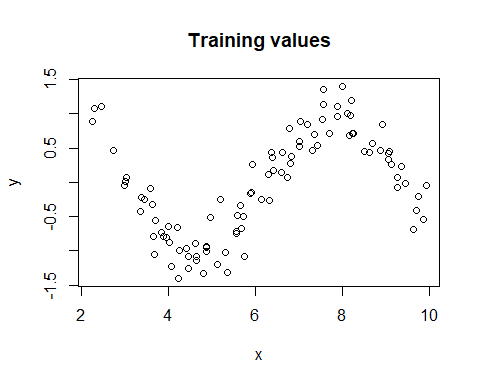
Filipp Krasovsky

6/7/2021

1. (15 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, 𝜺, and kernel parameters for a support vector machine model:

## Create a grid of x values to use for prediction > dataGrid <- data.frame(x = seq(2, 10, length = 100)).

#training data  
set.seed(100)  
x<-runif(100,min=2,max=10)  
y<-sin(x)+rnorm(length(x)) \* 0.25  
sinData<-data.frame(x=x,y=y)  
plot(x,y,main = "Training values")



#prediction data   
x\_test <- data.frame(x=seq(2,10,length=100))  
y\_test <- sin(x\_test)+rnorm(length(x\_test))\* 0.25

1. Fit different models using a radial basis function and different values of the cost (the C parameter) and 𝜺. Plot the fitted curve. For example: library(kernlab) rbfSVM <- ksvm(x = x, y = y, data = sinData, kernel =“rbfdot”, kpar = “automatic”, C = 1, epsilon = 0.1) modelPrediction <- predict(rbfSVM, newdata = dataGrid)

We start with a cost param of 1 and an epsilon of 0.1.

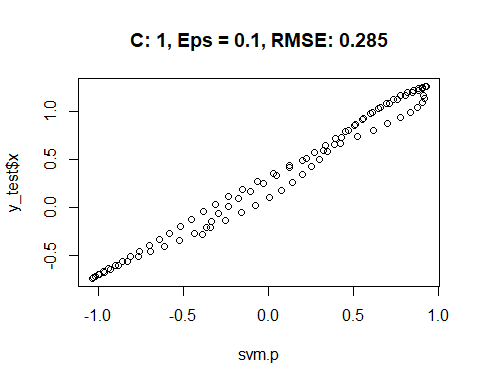
#use the ksvm model for svm with rbfdot as our kernel method.  
require(kernlab)

## Loading required package: kernlab

##   
## Attaching package: 'kernlab'

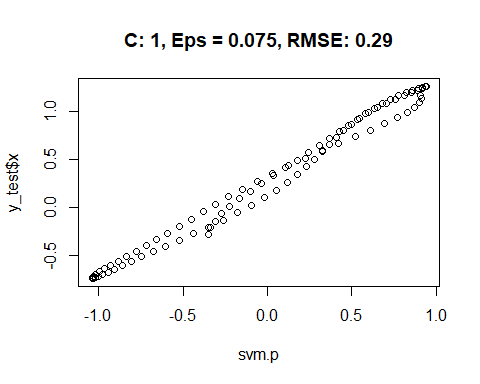
## The following object is masked from 'package:ggplot2':  
##   
## alpha

set.seed(100)  
svm <- ksvm(x=x,y=y,data=sinData,kernel="rbfdot",kpar="automatic",C=1,epsilon=0.1)  
svm.p<-predict(svm,newdata=x\_test)  
plot(x = svm.p, y = y\_test$x,main=paste("C: 1, Eps = 0.1, RMSE:",round(RMSE(svm.p,y\_test$x),3)))

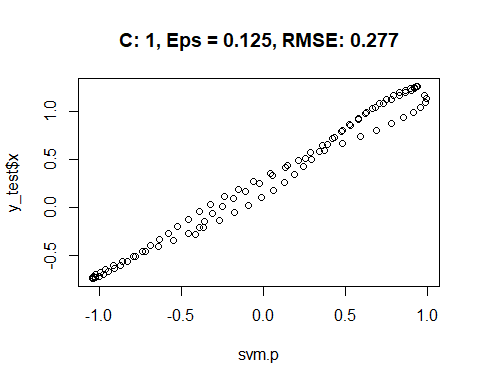


We move on to an epsilon of 0.075 and a cost param of 1:

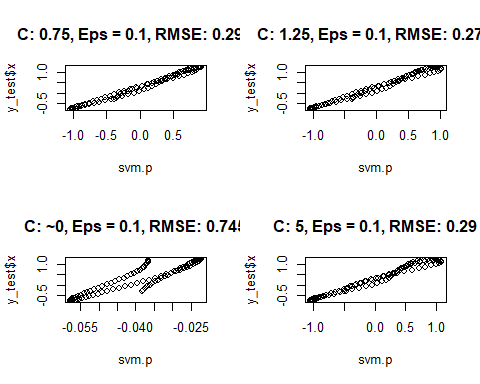
#use the ksvm model for svm with rbfdot as our kernel method.  
set.seed(100)  
svm <- ksvm(x=x,y=y,data=sinData,kernel="rbfdot",kpar="automatic",C=1,epsilon=0.075)  
svm.p<-predict(svm,newdata=x\_test)  
plot(x = svm.p, y = y\_test$x,main=paste("C: 1, Eps = 0.075, RMSE:",round(RMSE(svm.p,y\_test$x),3)))

 It appears decreasing out epsilon slightly drops our performance. We move on to an epsilon of 0.125 and a cost param of 1:

#use the ksvm model for svm with rbfdot as our kernel method.  
set.seed(100)  
svm <- ksvm(x=x,y=y,data=sinData,kernel="rbfdot",kpar="automatic",C=1,epsilon=0.125)  
svm.p<-predict(svm,newdata=x\_test)  
plot(x = svm.p, y = y\_test$x,main=paste("C: 1, Eps = 0.125, RMSE:",round(RMSE(svm.p,y\_test$x),3)))

 For boundary testing the cost parameter, we can scale C across 0.75 and 1.25 as an exercise, hosting epsilon constant at 0.1. As an extreme boundary, we can set cost to near zero and five as well.

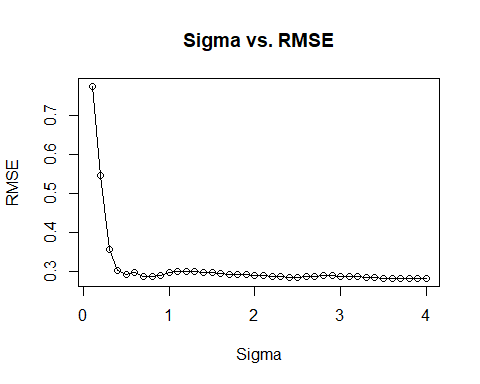
#use the ksvm model for svm with rbfdot as our kernel method.  
par(mfrow=c(2,2))  
set.seed(100)  
svm <- ksvm(x=x,y=y,data=sinData,kernel="rbfdot",kpar="automatic",C=0.75,epsilon=0.1)  
svm.p<-predict(svm,newdata=x\_test)  
plot(x = svm.p, y = y\_test$x,main=paste("C: 0.75, Eps = 0.1, RMSE:",round(RMSE(svm.p,y\_test$x),3)))  
  
set.seed(100)  
svm <- ksvm(x=x,y=y,data=sinData,kernel="rbfdot",kpar="automatic",C=1.25,epsilon=0.1)  
svm.p<-predict(svm,newdata=x\_test)  
plot(x = svm.p, y = y\_test$x,main=paste("C: 1.25, Eps = 0.1, RMSE:",round(RMSE(svm.p,y\_test$x),3)))  
  
set.seed(100)  
svm <- ksvm(x=x,y=y,data=sinData,kernel="rbfdot",kpar="automatic",C=0.001,epsilon=0.1)  
svm.p<-predict(svm,newdata=x\_test)  
plot(x = svm.p, y = y\_test$x,main=paste("C: ~0, Eps = 0.1, RMSE:",round(RMSE(svm.p,y\_test$x),3)))  
  
set.seed(100)  
svm <- ksvm(x=x,y=y,data=sinData,kernel="rbfdot",kpar="automatic",C=5,epsilon=0.1)  
svm.p<-predict(svm,newdata=x\_test)  
plot(x = svm.p, y = y\_test$x,main=paste("C: 5, Eps = 0.1, RMSE:",round(RMSE(svm.p,y\_test$x),3)))

 On face value, it appears that increasing the cost parameter is associated with an increase in RMSE, with diminishing marginal returns. Near zero cost parameters induce a much larger RMSE, while higher C-values decrease performance as well.

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

We know, as a rule, that higher values of sigma make the model more flexible because errors tend to get amplified, but lower values make the model “stiffen”, leading to underfitting.

sigma\_frame = seq(from = 0.1, to = 4, by = 0.1)  
results = data.frame(Sigma=character(),RMSE=double())  
  
for (i in sigma\_frame){  
 set.seed(100)  
 svm <- ksvm(x=x,y=y,data=sinData,kernel="rbfdot",kpar=list(sigma=i),C=1,epsilon=0.1)  
 svm.p<-predict(svm,newdata=x\_test)  
 this.result <- data.frame(i,RMSE(svm.p,y\_test$x))  
 results = rbind(results,this.result)  
}  
  
names(results) = c("Sigma","RMSE")  
  
plot(results,type="o",main="Sigma vs. RMSE")

 Overall, it seems that increasing sigma beyond 1 tends to produce an increase in error rates,with the optimal value lying between zero and one. Similarly, increasing the epsilon parameter increases the threshold at which data points become relevant to modeling given that SVM is epsilon-insensitive. Using an epsilon that is too high makes the model dependent on points with radically low fit, this failing to generalize to outside data.

1. (15 points) For the Tecator data described in chapter 6 of the textbook, build SVM, neural network, MARS, and KNN models. Since neural networks are especially sensitive to highly correlated predictors, does pre-processing using PCA help the model? For this problem, you will perform your analysis in R.

Because the textbook frames the data question as modeling the relationship to fat, we use column 2 of the endpoints set.

#load data - absorp is the predictor set and endpoints are the response set(s).  
library(caret)  
data(tecator)  
require(nnet)

## Loading required package: nnet

require(earth)

## Loading required package: earth

## Warning: package 'earth' was built under R version 4.0.5

## Loading required package: Formula

## Loading required package: plotmo

## Warning: package 'plotmo' was built under R version 4.0.5

## Loading required package: plotrix

## Loading required package: TeachingDemos

## Warning: package 'TeachingDemos' was built under R version 4.0.5

#without PCA   
x = as.data.frame(absorp)  
y = as.data.frame(endpoints[,2])  
x.p = preProcess(x,method = c("center","scale"))  
x = predict(x.p,x)  
  
names(y) = 'fat'  
  
set.seed(1)  
trainingRows <- createDataPartition(y$fat,p = 0.80, list=FALSE)  
  
trainx<- x[trainingRows,]  
trainy<- y[trainingRows,]  
testx <- x[-trainingRows,]  
testy <- y[-trainingRows,]  
  
  
  
#SVM  
set.seed(100)  
svmRtuned <- train((trainx),(trainy),method="svmRadial",preProc = c("center","scale"),tunelength=14,  
 trControl = trainControl(method="cv"))  
svm.p <- predict(svmRtuned$finalModel,testx)  
svm.rmse<-RMSE(svm.p,testy)  
print(paste("SVM",svm.rmse))

## [1] "SVM 6.55562134092721"

#NN  
set.seed(100)  
nnet <- avNNet(trainx,trainy,size=5,decay=0.01,repeats=5,linout=TRUE,trace=FALSE,maxit=500,maxNwts=5\*(ncol(trainx)+1)+5+1)

## Warning: executing %dopar% sequentially: no parallel backend registered

nnet.p<-predict(nnet,testx)  
nnet.rmse<-RMSE(nnet.p,testy)  
print(paste("nnet",nnet.rmse))

## [1] "nnet 0.37990906080454"

#MARS  
marsgrid <- expand.grid(.degree=1:2,.nprune=2:38)  
set.seed(100)  
mars.t <- train(trainx,trainy,method="earth",tuneGrid=marsgrid,trControl=trainControl(method="cv"))  
mars.p <- predict(mars.t,testx)  
mars.rmse<-RMSE(mars.p,testy)  
print(paste("MARS",mars.rmse))

## [1] "MARS 1.78792232873288"

#KNN - remove unbalanced first.  
knnDescr <- trainx[,-nearZeroVar(trainx)]  
knntune <- train(knnDescr,trainy,method="knn",tuneGrid=data.frame(.k=1:20),trControl=trainControl(method="cv"))

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

knn.p <-predict(knntune,testx)  
knn.rmse<-RMSE(knn.p,testy)  
print(paste("KNN",knn.rmse))

## [1] "KNN 12.4431471959476"

#NN with PCA   
x = as.data.frame(absorp)  
y = as.data.frame(endpoints[,2])  
x.p = preProcess(x,method = c("center","scale","pca"))  
x = predict(x.p,x)  
  
names(y) = 'fat'  
  
set.seed(1)  
trainingRows <- createDataPartition(y$fat,p = 0.80, list=FALSE)  
  
trainx<- x[trainingRows,]  
trainy<- y[trainingRows,]  
testx <- x[-trainingRows,]  
testy <- y[-trainingRows,]  
  
#NN  
set.seed(100)  
nnet <- avNNet(trainx,trainy,size=5,decay=0.01,repeats=5,linout=TRUE,trace=FALSE,maxit=500,maxNwts=5\*(ncol(trainx)+1)+5+1)  
nnet.p<-predict(nnet,testx)  
nnet.rmse<-RMSE(nnet.p,testy)  
print(paste("nnet w PCA",nnet.rmse))

## [1] "nnet w PCA 11.1405942861103"

PCA failed to improve our neural net.

1. The matrix processPredictors 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. This describes the data for a chemical manufacturing process. Use data imputation, data splitting, and pre-processing steps and train several nonlinear regression models.
2. Which predictors are most important in the optimal nonlinear regression model? Do either the biological or process variables dominate the list? How do the top ten important predictors compare to the top ten predictors from the optimal linear model?
3. Explore the relationships between the top predictors and the response for the predictors that are unique to the optimal nonlinear regression model. Do these plots reveal intuition about the biological or process predictors and their relationship with yield?

For this problem, you will perform your analysis in R.

#load data  
library(AppliedPredictiveModeling)

## Warning: package 'AppliedPredictiveModeling' was built under R version 4.0.5

data(ChemicalManufacturingProcess)  
  
#knn imputation  
require(Hmisc)

## Loading required package: Hmisc

## Warning: package 'Hmisc' was built under R version 4.0.5

## Loading required package: survival

##   
## Attaching package: 'survival'

## The following object is masked from 'package:caret':  
##   
## cluster

##   
## Attaching package: 'Hmisc'

## The following objects are masked from 'package:TeachingDemos':  
##   
## cnvrt.coords, subplot

## The following objects are masked from 'package:dplyr':  
##   
## src, summarize

## The following objects are masked from 'package:base':  
##   
## format.pval, units

x = (ChemicalManufacturingProcess)  
x = subset(x,select=-c(Yield))  
x = as.data.frame(x)  
x = Hmisc::impute(x)  
y <-ChemicalManufacturingProcess$Yield %>% as.data.frame   
  
#remove nzv   
nzv = nearZeroVar(x)  
x = x[-nzv]  
  
#center, scale   
x.cs <- preProcess(x,method=c("center","scale"))  
x = predict(x.cs,x)  
  
#partition  
set.seed(1)  
names(y)="yield"  
trainingRows <- createDataPartition(y$yield,p = 0.80, list=FALSE)  
  
trainx<- x[trainingRows,]  
trainy<- y[trainingRows,]  
testx <- x[-trainingRows,]  
testy <- y[-trainingRows,]

1. Which nonlinear regression model gives the optimal resampling and test set performance? For this instance, we will use svm, nnet, and mars.

#SVM  
set.seed(100)  
svmRtuned <- train((trainx),(trainy),method="svmRadial",preProc = c("center","scale"),tunelength=14,  
 trControl = trainControl(method="cv"))  
svm.p <- predict(svmRtuned$finalModel,testx)  
svm.rmse<-RMSE(svm.p,testy)  
  
#NN  
set.seed(100)  
nnet <- avNNet(trainx,trainy,size=5,decay=0.01,repeats=5,linout=TRUE,trace=FALSE,maxit=500,maxNwts=5\*(ncol(trainx)+1)+5+1)  
nnet.p<-predict(nnet,testx)  
nnet.rmse<-RMSE(nnet.p,testy)  
  
#MARS  
marsgrid <- expand.grid(.degree=1:2,.nprune=2:38)  
set.seed(100)  
mars.t <- train(trainx,trainy,method="earth",tuneGrid=marsgrid,trControl=trainControl(method="cv"))  
mars.p <- predict(mars.t,testx)  
mars.rmse<-RMSE(mars.p,testy)  
  
  
print(paste("SVM",svm.rmse))

## [1] "SVM 0.984948379124365"

print(paste("MARS",mars.rmse))

## [1] "MARS 1.04743569857994"

print(paste("nnet",nnet.rmse))

## [1] "nnet 0.996321573653602"

Our SVM model performed the best with a test RMSE of 0.984 and the following training RMSE of 1.21:

print(svmRtuned)

## Support Vector Machines with Radial Basis Function Kernel   
##   
## 144 samples  
## 56 predictor  
##   
## Pre-processing: centered (56), scaled (56)   
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 129, 130, 130, 130, 130, 130, ...   
## Resampling results across tuning parameters:  
##   
## C RMSE Rsquared MAE   
## 0.25 1.452999 0.4649044 1.1709934  
## 0.50 1.340040 0.5278748 1.0673606  
## 1.00 1.212608 0.6005815 0.9576347  
##   
## Tuning parameter 'sigma' was held constant at a value of 0.02358453  
## RMSE was used to select the optimal model using the smallest value.  
## The final values used for the model were sigma = 0.02358453 and C = 1.

1. Which predictors are most important in the optimal nonlinear regression model? Do either the biological or process variables dominate the list? How do the top ten important predictors compare to the top ten predictors from the optimal linear model?

imp = varImp(svmRtuned)$importance

The most important predictors are (in order): ManufacturingProcess32 ManufacturingProcess13 BiologicalMaterial06 BiologicalMaterial03 ManufacturingProcess17 BiologicalMaterial12 ManufacturingProcess09 BiologicalMaterial02 ManufacturingProcess06 ManufacturingProcess31

Manufacturing appears to dominate.

#construct an optimal linear model  
library(glmnet)

## Warning: package 'glmnet' was built under R version 4.0.5

## Loading required package: Matrix

## Loaded glmnet 4.1-1

require(caret)  
#perform k-fold cross-validation to find optimal lambda value  
cv\_model <- cv.glmnet(as.matrix(trainx), trainy, alpha = 0.5)  
  
#find optimal lambda value that minimizes test MSE  
best\_lambda <- cv\_model$lambda.min  
best\_model <- glmnet(as.matrix(trainx), trainy, alpha = 0.5, lambda = best\_lambda)  
lasso <- predict(best\_model, s = best\_lambda, newx = as.matrix(testx))  
lasso.rmse <- RMSE(lasso,testy)  
  
lasso.imp = varImp(best\_model,best\_model$lambda) %>% subset(Overall > 0)   
lasso.imp =   
print(head(lasso.imp,n=10))

## Overall  
## BiologicalMaterial03 0.01423861  
## BiologicalMaterial06 0.09731297  
## ManufacturingProcess09 0.36042005  
## ManufacturingProcess13 0.21886294  
## ManufacturingProcess15 0.01977422  
## ManufacturingProcess17 0.21249807  
## ManufacturingProcess32 0.81874937  
## ManufacturingProcess33 0.04338854  
## ManufacturingProcess36 0.01087368  
## ManufacturingProcess37 0.04999779

In the instance of our Lasso model, manufacturing also dominates but at a slightly higher rate.

1. Explore the relationships between the top predictors and the response for the predictors that are unique to the optimal nonlinear regression model. Do these plots reveal intuition about the biological or process predictors and their relationship with yield?

Unique predictors to the nonlinear model:

BiologicalMaterial12 ManufacturingProcess09 BiologicalMaterial02 ManufacturingProcess06 ManufacturingProcess31

unique.p = subset(trainx,select=c(BiologicalMaterial12,ManufacturingProcess09,BiologicalMaterial02,ManufacturingProcess06,ManufacturingProcess31))  
unique.p$yield = trainy  
  
cor(unique.p)

## BiologicalMaterial12 ManufacturingProcess09  
## BiologicalMaterial12 1.00000000 0.2211858  
## ManufacturingProcess09 0.22118585 1.0000000  
## BiologicalMaterial02 0.78948338 0.1677067  
## ManufacturingProcess06 0.14128234 0.2151233  
## ManufacturingProcess31 -0.04100431 -0.2522044  
## yield 0.35369877 0.4775427  
## BiologicalMaterial02 ManufacturingProcess06  
## BiologicalMaterial12 0.78948338 0.14128234  
## ManufacturingProcess09 0.16770673 0.21512334  
## BiologicalMaterial02 1.00000000 0.12964602  
## ManufacturingProcess06 0.12964602 1.00000000  
## ManufacturingProcess31 -0.09333995 -0.03050484  
## yield 0.44517169 0.14195375  
## ManufacturingProcess31 yield  
## BiologicalMaterial12 -0.041004311 0.353698768  
## ManufacturingProcess09 -0.252204422 0.477542656  
## BiologicalMaterial02 -0.093339953 0.445171693  
## ManufacturingProcess06 -0.030504840 0.141953750  
## ManufacturingProcess31 1.000000000 -0.006102568  
## yield -0.006102568 1.000000000

None of the unique predictors seem to suggest any strong visible correlation with the Yield variable separately, with the highest correlation only going into the 40th percentile.

1. (15 points) Recreate the simulated data as shown below:
2. Fit a random forest model to all of the predictors, then estimate the variable importance scores:

Did the random forest model significantly use the uninformative predictors (V6 – V10)?

For this problem, you will perform your analysis in R.

library(mlbench)

## Warning: package 'mlbench' was built under R version 4.0.5

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"  
  
library(randomForest)

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##   
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':  
##   
## combine

## The following object is masked from 'package:ggplot2':  
##   
## margin

library(caret)  
model1 <- randomForest(y ~ ., data = simulated, importance = TRUE,  
ntree = 1000)   
rfImp1 <- varImp(model1, scale = FALSE)  
print(rfImp1)

## Overall  
## V1 8.732235404  
## V2 6.415369387  
## V3 0.763591825  
## V4 7.615118809  
## V5 2.023524577  
## V6 0.165111172  
## V7 -0.005961659  
## V8 -0.166362581  
## V9 -0.095292651  
## V10 -0.074944788

The model did not significantly use V6-V10 compared to the other predictors.

1. Now add an additional predictor that is highly correlated with one of the informative predictors. For example: simulatedV1 + rnorm(200) \* .1 > cor(simulatedV1) Fit another random forest model to these data. Did the importance score for V1 change? What happens when you add another predictor that is also highly correlated with V1?

simulated$dup1 <- simulated$V1 + rnorm(200) \* 0.1  
#sanity check  
cor(simulated$V1,simulated$dup1)

## [1] 0.9460206

#refit  
model1 <- randomForest(y ~ ., data = simulated, importance = TRUE,  
ntree = 1000)   
rfImp1 <- varImp(model1, scale = FALSE)  
  
print(rfImp1)

## Overall  
## V1 5.69119973  
## V2 6.06896061  
## V3 0.62970218  
## V4 7.04752238  
## V5 1.87238438  
## V6 0.13569065  
## V7 -0.01345645  
## V8 -0.04370565  
## V9 0.00840438  
## V10 0.02894814  
## dup1 4.28331581

V1 dropped in importance while the dup1 variable took on a similar overall value. When we add another predictor, all three colinear predictors drop in value, with V1 having the highest magnitude and Dup1 havingthe second highest.

simulated$dup2 <- simulated$V1 + rnorm(200) \* 0.1  
#refit  
model1 <- randomForest(y ~ ., data = simulated, importance = TRUE,  
ntree = 1000)   
rfImp1 <- varImp(model1, scale = FALSE)  
  
print(rfImp1)

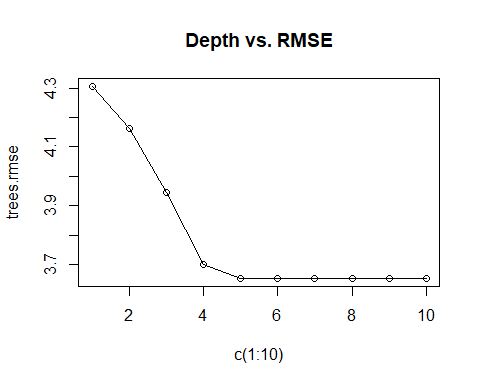
## Overall  
## V1 4.91687329  
## V2 6.52816504  
## V3 0.58711552  
## V4 7.04870917  
## V5 2.03115561  
## V6 0.14213148  
## V7 0.10991985  
## V8 -0.08405687  
## V9 -0.01075028  
## V10 0.09230576  
## dup1 3.80068234  
## dup2 1.87721959

1. (15 points) Use a simulation to show tree bias with different granularities. For this problem, you will perform your analysis in R.

require(rpart)

## Loading required package: rpart

library(mlbench)  
set.seed(200)  
simulated <- mlbench.friedman1(300, sd = 1)   
simulated <- cbind(simulated$x, simulated$y)  
simulated <- as.data.frame(simulated)  
colnames(simulated)[ncol(simulated)] <- "y"  
  
trainingRows <- createDataPartition(simulated$y,p = 0.80, list=FALSE)  
train <- simulated[trainingRows,]  
test <- simulated[-trainingRows,]  
  
trees.rmse = c()  
  
for (i in c(1:10)){  
 rpartTree <- rpart(y ~ ., data = as.data.frame(train),maxdepth=i)  
 tree.p <- predict(rpartTree,subset(test,select=-c(y)))  
 trees.rmse <- cbind(trees.rmse,RMSE(tree.p,test$y))  
}  
  
plot(x=c(1:10),y=trees.rmse,main="Depth vs. RMSE",type="o")

 6. (15 points) In stochastic gradient boosting, the bagging fraction and learning rate will govern the construction of the trees as they are guided by the gradient. Although the optimal values of these parameters should be obtained through the tuning process, it is helpful to understand how the magnitudes of these parameters affect magnitudes of variable importance. Figure 8.24 below 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:

1. 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?

A higher learning rate means that the step size during our optimization process is larger - generally, a larger learning rate means that fewer iterations of tuning are required to construct a model. This, combined with the fact that 90% of the training set is used in each bagging iteration, means that the model on the right is likely to pick up on the same signal across samples and terminate faster, leading it to favor a handful of predictors. The converse is true for the model on the left, which takes longer to fully propagate due to its learning rate, and because of the small bagging fraction, is likely to capture a variety of different signals.

1. Which model do you think would be more predictive of other samples? The model on the left exhibits less bias the first few predictors, and is less likely to underfit unknown data.
2. How would increasing interaction depth affect the slope of predictor importance for either model in Fig. 8.24? A smaller learning rate requires more iterations to properly develop a model - therefore, it’s likely that the slopes of certain predictors on the left are going to increase in magnitude at a greater rate than of those on the right.