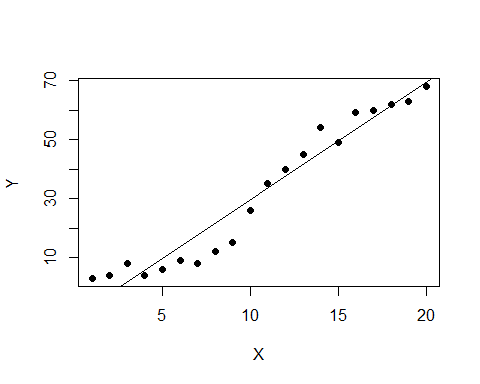
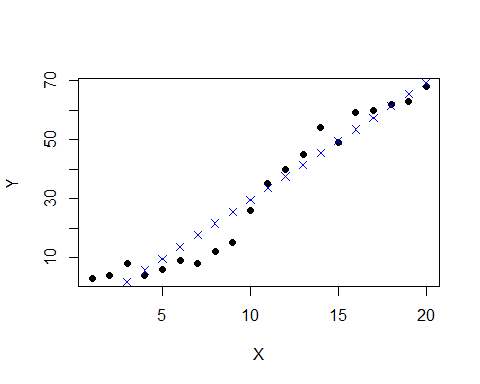
regressionTests.r

## Originally from http://www.svm-tutorial.com/2014/10/support-vector-regression-r/  
## Neural Network added by A. Breitzman   
  
setwd("C:/Users/Tony/Dropbox/Rowan/DM2/Lecture2")  
  
data <- read.csv('regression.csv', sep=",", header = TRUE)  
  
# Plot the data  
plot(data, pch=16)  
  
# Create a linear regression model  
model <- lm(Y ~ X, data)  
  
# Add the fitted line  
abline(model)



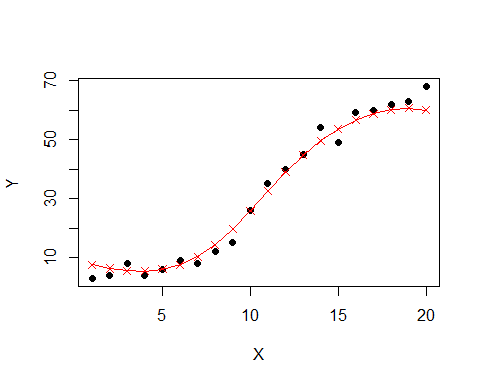
# Look at this again as a prediction  
plot(data, pch=16)  
model <- lm(Y ~ X , data)  
  
# make a prediction for each X  
predictedY <- predict(model, data)  
  
# display the predictions  
points(data$X, predictedY, col = "blue", pch=4)



# Compute Mean Squared Error  
rmse <- function(error){  
 sqrt(mean(error^2))  
}  
  
error <- model$residuals # same as data$Y - predictedY  
rmse(error)

## [1] 5.703778

## 5.703778 is our baseline  
## See if we can do a little better with an SVM  
  
library(e1071)  
model <- svm(Y ~ X , data)  
  
predictedY <- predict(model, data)  
plot(data, pch=16)  
points(data$X, predictedY, col = "red", pch=4)  
lines(data$X, predictedY, col = "red", pch=4)



error <- data$Y - predictedY  
svrPredictionRMSE <- rmse(error) # 3.157061  
  
## Better  
## Can we do better by tuning  
  
## We'll do a grid search  
## This will take a while because we're actually running 88 models  
# perform a grid search  
tuneResult <- tune(svm, Y ~ X, data = data,  
 ranges = list(epsilon = seq(0,1,0.1), cost = 2^(2:9))  
)  
print(tuneResult)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## epsilon cost  
## 0 128  
##   
## - best performance: 7.026237

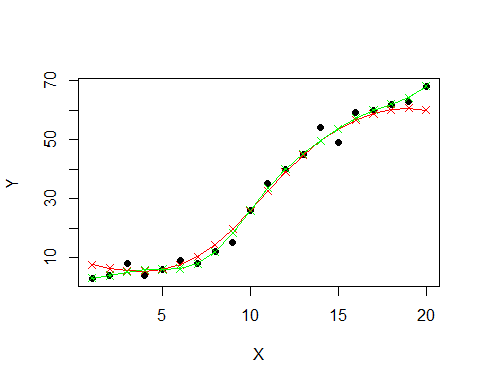
# best performance: MSE = 8.48; epsilon=0, cost=4  
  
## Do another grid search with epsilon close to 0  
tuneResult <- tune(svm, Y ~ X, data = data,  
 ranges = list(epsilon = seq(0,0.2,0.01), cost = 2^(2:9))  
)   
  
print(tuneResult)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## epsilon cost  
## 0 256  
##   
## - best performance: 7.460289

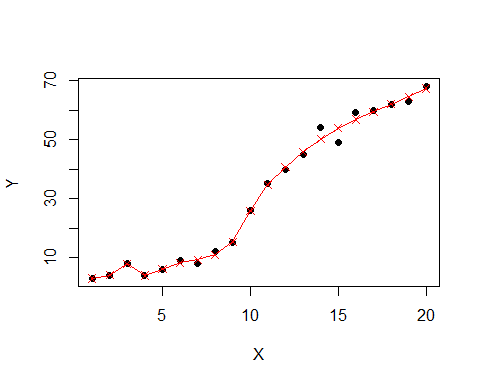
## epsilon=.09 cost=128  
tunedModel <- tuneResult$best.model  
tunedModelY <- predict(tunedModel, data)   
  
error <- data$Y - tunedModelY   
  
# this value can be different on your computer  
# because the tune method randomly shuffles the data  
rmse(error)

## [1] 1.967058

plot(data, pch=16)  
points(data$X, predictedY, col = "red", pch=4)  
lines(data$X, predictedY, col = "red", pch=4)  
points(data$X, tunedModelY, col = "green", pch=4)  
lines(data$X, tunedModelY, col = "green", pch=4)



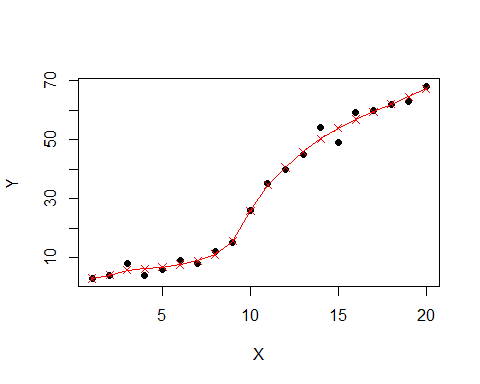
library("neuralnet")  
nnet<-neuralnet(Y~X,data, hidden=20, threshold=0.1)  
## Ran for a while  
  
plot(data, pch=16)  
  
  
results<-compute(nnet,data$X)  
points(data$X, results$net.result, col = "red", pch=4)  
lines(data$X, results$net.result, col = "red", pch=4)



## Pretty good from 10 up, but probably overfit below 10  
  
  
error <- data$Y - results$net.result   
rmse(error)

## [1] 1.585371686

## 1.59 but overfit. Let's do again with 10 hidden nodes  
nnet<-neuralnet(Y~X,data, hidden=10, threshold=0.5)  
plot(data, pch=16)  
  
  
results<-compute(nnet,data$X)  
points(data$X, results$net.result, col = "red", pch=4)  
lines(data$X, results$net.result, col = "red", pch=4)



## That one looks better without an overfit  
error <- data$Y - results$net.result   
rmse(error)

## [1] 1.743550757

## 1.74

## Somehow I still feel more confident with the SVM