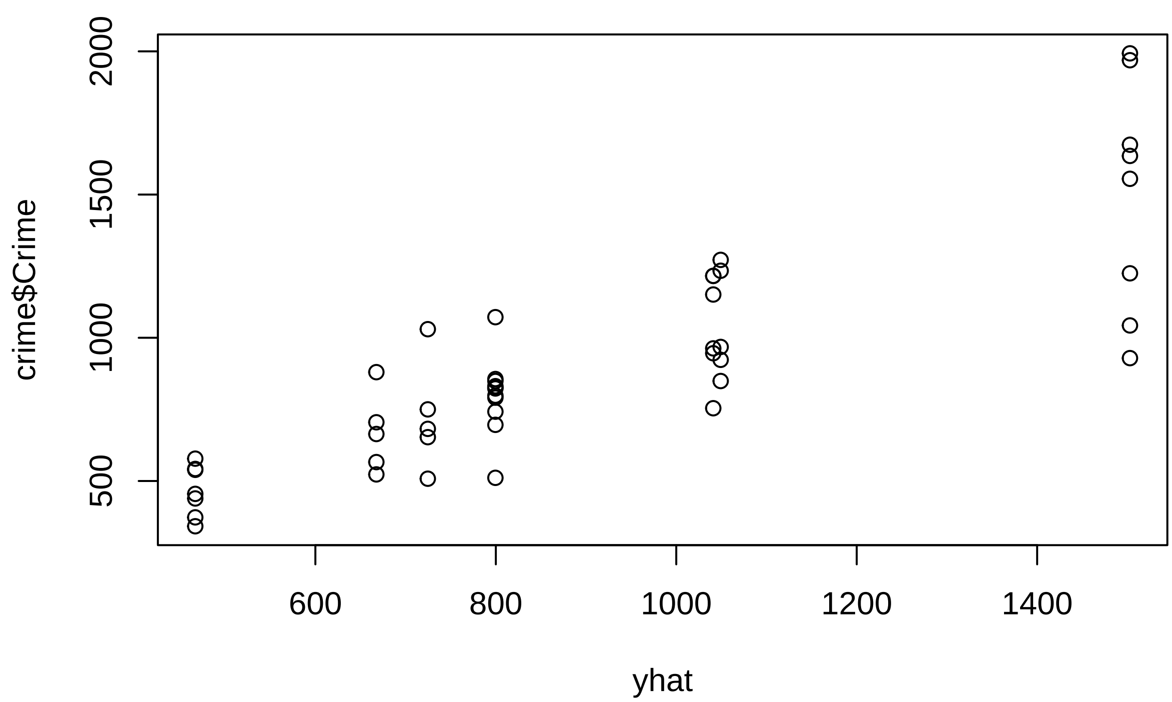
Hw7

```r
crime <- read.delim("http://www.statsci.org/data/general/uscrime.txt")

crimetree <- tree(Crime~., data = crime)
summary(crimetree)
```

```
##
## Regression tree:
## tree(formula = Crime ~ ., data = crime)
## Variables actually used in tree construction:
## [1] "Po1" "Pop" "LF"  "NW"
## Number of terminal nodes:  7
## Residual mean deviance:  47390 = 1896000 / 40
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -573.900  -98.300   -1.545    0.000  110.600  490.100
```

```r
yhat <- predict(crimetree)
plot(yhat,crime$Crime)
```



Looking at he predicted values from regression compared to the Crime response values from dataset.

```
prune.tree(crimetree)$size
```

```
## [1] 7 6 5 4 3 2 1
```

```
prune.tree(crimetree)$dev
```

```
## [1] 1895722 2013257 2276670 2632631 3364043 4383406 6880928
```

```
cv.tree(crimetree)$dev
```

```
## [1] 8003184 8000295 7842727 7948940 8104883 6951438 7575010
```

Doing Cross validation on tree model.

```
prunetree <- prune.tree(crimetree, best = 4)
yhat2 <- predict(prunetree)
SSres <- sum((yhat2-crime$Crime)^2)
SStot <- sum((crime$Crime - mean(crime$Crime))^2)
r2 <- 1-(SSres/SStot)
r2
```

```
## [1] 0.6174017
```

By pruning to 4 leaves we get a decent model with a 61% accuracy rate.

```
prunetree1 <- prune.tree(crimetree, best = 2)
yhat1 <- predict(prunetree1)
SSres <- sum((yhat1-crime$Crime)^2)
SStot <- sum((crime$Crime - mean(crime$Crime))^2)
r2 <- 1-(SSres/SStot)
r2
```

```
## [1] 0.3629629
```

We see the model with 2 leaves has the accuracy way worse and this is so because the regression tree has to few leaves so there is underfitting. The model with 4 leaves has much higher accuracy.

```
prunetree <- prune.tree(crimetree, best = 5)
yhat2 <- predict(prunetree)
SSres <- sum((yhat2-crime$Crime)^2)
SStot <- sum((crime$Crime - mean(crime$Crime))^2)
r2 <- 1-(SSres/SStot)
r2
```

```
## [1] 0.6691333
```

When we have 5 leaves we see the accuracy at 66% which shows it has the highest when compared to the other models.

```r
rf <- randomForest(Crime~., data = crime)
print(rf)
```

```
##
## Call:
##  randomForest(formula = Crime ~ ., data = crime)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 5
##
##           Mean of squared residuals: 85004.63
##                     % Var explained: 41.94
```

```r
importance(rf) #Looking at the importance of each predictor
```

```
##        IncNodePurity
## M          211092.90
## So          20755.88
## Ed         223157.34
## Po1       1294407.42
## Po2       1112767.88
## LF         302487.09
## M.F        243179.14
## Pop        345551.53
## NW         510115.94
## U1         118809.05
## U2         169611.94
## Wealth     589715.72
## Ineq       225217.40
## Prob       824187.34
## Time       204999.54
```

```r
fit4 <- randomForest(Crime~., data = crime, mtry = 4, importance = TRUE)
fit4
```

```
##
## Call:
##  randomForest(formula = Crime ~ ., data = crime, mtry = 4, importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 85081.14
##                     % Var explained: 41.89
```

```r
yhatrf <- predict(fit4)
SS <- sum((yhatrf - crime$Crime)^2)
SStot <- sum((crime$Crime - mean(crime$Crime))^2)

r2 <- 1 - (SS/SStot)
r2
```

```
## [1] 0.4188554
```

```r
fit2 <- randomForest(Crime~., data = crime, mtry = 2, importance = TRUE)
yhatrf <- predict(fit2)
SS <- sum((yhatrf - crime$Crime)^2)
SStot <- sum((crime$Crime - mean(crime$Crime))^2)

r2 <- 1 - (SS/SStot)
r2
```

```
## [1] 0.4001285
```

We can see with 2 predictors the Random Forest model has a higher accuracy compared to the model with 4 predictors. 4 may be too many predictors and can cause some overfitting.

10.2 Medical researches want to know how exercise and weight impact prob of heart attack. Logistic regression can be performed to understand relationship. Binary response values would be patient has heart attack and doesn't have heart attack. Predictors can include exercise and weight.

10.3

```r
german <- read.table("german.txt", sep = " ")
head(german)
```

```
##    V1 V2  V3  V4   V5  V6  V7 V8  V9  V10 V11  V12 V13  V14  V15 V16  V17 V18
## 1 A11  6 A34 A43 1169 A65 A75  4 A93 A101   4 A121  67 A143 A152   2 A173   1
## 2 A12 48 A32 A43 5951 A61 A73  2 A92 A101   2 A121  22 A143 A152   1 A173   1
## 3 A14 12 A34 A46 2096 A61 A74  2 A93 A101   3 A121  49 A143 A152   1 A172   2
## 4 A11 42 A32 A42 7882 A61 A74  2 A93 A103   4 A122  45 A143 A153   1 A173   2
## 5 A11 24 A33 A40 4870 A61 A73  3 A93 A101   4 A124  53 A143 A153   2 A173   2
## 6 A14 36 A32 A46 9055 A65 A73  2 A93 A101   4 A124  35 A143 A153   1 A172   2
##    V19  V20 V21
## 1 A192 A201   1
## 2 A191 A201   2
## 3 A191 A201   1
## 4 A191 A201   1
## 5 A191 A201   2
## 6 A192 A201   1
```

```r
german$V21[german$V21==1]<-0 #making response variables binary, 0 and 1
german$V21[german$V21==2]<-1

#Creating train and test set
german_train <- german[1:800,]
german_test <- german[801:1000,]

#creating logistic regression model
germanmodel = glm(V21~.,
                  family=binomial(link = "logit"),
                  data = german_train)


summary(germanmodel)
```

```
##
## Call:
## glm(formula = V21 ~ ., family = binomial(link = "logit"), data = german_train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7373  -0.6979  -0.3604   0.6663   2.5591
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  3.978e-01  1.249e+00   0.318 0.750139
## V1A12       -2.701e-01  2.437e-01  -1.108 0.267784
## V1A13       -9.306e-01  4.018e-01  -2.316 0.020567 *
## V1A14       -1.737e+00  2.686e-01  -6.465 1.02e-10 ***
## V2           2.893e-02  1.042e-02   2.777 0.005479 **
## V3A31        2.255e-01  6.289e-01   0.359 0.719936
## V3A32       -7.639e-01  4.753e-01  -1.607 0.108022
## V3A33       -9.172e-01  5.233e-01  -1.753 0.079627 .
## V3A34       -1.487e+00  4.907e-01  -3.031 0.002440 **
## V4A41       -1.832e+00  4.425e-01  -4.141 3.46e-05 ***
## V4A410      -1.413e+00  8.263e-01  -1.710 0.087326 .
## V4A42       -9.368e-01  2.990e-01  -3.134 0.001727 **
## V4A43       -9.044e-01  2.799e-01  -3.230 0.001236 **
## V4A44       -8.312e-01  8.946e-01  -0.929 0.352807
## V4A45       -3.222e-01  6.092e-01  -0.529 0.596843
## V4A46        1.688e-02  4.255e-01   0.040 0.968354
## V4A48       -2.213e+00  1.219e+00  -1.816 0.069365 .
## V4A49       -8.368e-01  3.850e-01  -2.173 0.029760 *
## V5           1.138e-04  5.166e-05   2.202 0.027682 *
## V6A62       -3.991e-01  3.182e-01  -1.254 0.209771
## V6A63       -4.615e-01  4.762e-01  -0.969 0.332404
## V6A64       -1.222e+00  5.473e-01  -2.232 0.025592 *
## V6A65       -7.093e-01  2.929e-01  -2.421 0.015462 *
## V7A72       -2.017e-01  4.948e-01  -0.408 0.683485
## V7A73       -3.028e-01  4.706e-01  -0.643 0.519975
## V7A74       -1.105e+00  5.113e-01  -2.162 0.030623 *
## V7A75       -4.092e-01  4.712e-01  -0.869 0.385102
## V8           3.602e-01  9.933e-02   3.626 0.000287 ***
## V9A92       -4.434e-01  4.300e-01  -1.031 0.302374
## V9A93       -1.230e+00  4.245e-01  -2.897 0.003769 **
## V9A94       -4.630e-01  5.119e-01  -0.905 0.365705
## V10A102      7.521e-01  4.771e-01   1.576 0.114917
## V10A103     -9.329e-01  4.830e-01  -1.931 0.053423 .
## V11          3.282e-03  9.850e-02   0.033 0.973420
## V12A122      4.101e-01  2.897e-01   1.415 0.156969
## V12A123      1.536e-01  2.649e-01   0.580 0.562115
## V12A124      7.122e-01  4.714e-01   1.511 0.130827
## V13         -1.868e-02  1.055e-02  -1.770 0.076682 .
## V14A142     -1.442e-02  4.733e-01  -0.030 0.975695
## V14A143     -4.354e-01  2.724e-01  -1.599 0.109919
## V15A152     -3.967e-01  2.739e-01  -1.448 0.147576
## V15A153     -5.576e-01  5.303e-01  -1.051 0.293071
## V16          3.297e-01  2.124e-01   1.552 0.120602
## V17A172      5.151e-01  7.807e-01   0.660 0.509351
```

```
## V17A173      5.655e-01  7.507e-01   0.753 0.451267
## V17A174      8.202e-01  7.597e-01   1.080 0.280307
## V18          5.065e-01  2.854e-01   1.775 0.075972 .
## V19A192     -3.739e-01  2.323e-01  -1.610 0.107489
## V20A202     -1.498e+00  8.079e-01  -1.854 0.063779 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 975.68  on 799  degrees of freedom
## Residual deviance: 705.07  on 751  degrees of freedom
## AIC: 803.07
##
## Number of Fisher Scoring iterations: 5
```

```
yhat <- predict(germanmodel,german_test, type = "response")

yhat
```

```
##         801         802         803         804         805         806
## 0.236906136 0.139383734 0.358926129 0.022590994 0.432251065 0.504898629
##         807         808         809         810         811         812
## 0.088000916 0.021306066 0.583870811 0.747847466 0.267757849 0.091849548
##         813         814         815         816         817         818
## 0.192269526 0.453664774 0.795236975 0.809340017 0.026914462 0.019175717
##         819         820         821         822         823         824
## 0.938390487 0.647357108 0.216193713 0.285379382 0.572189859 0.271130490
##         825         826         827         828         829         830
## 0.161527886 0.598038134 0.616286850 0.336548410 0.227884212 0.569096138
##         831         832         833         834         835         836
## 0.090175170 0.804507533 0.901098391 0.177743783 0.203400485 0.799172865
##         837         838         839         840         841         842
## 0.096528686 0.105465313 0.089689617 0.202872785 0.469538312 0.055263433
##         843         844         845         846         847         848
## 0.324216632 0.108243617 0.271802091 0.041812278 0.105096840 0.392434659
##         849         850         851         852         853         854
## 0.067308505 0.240082680 0.382129853 0.004856498 0.058155846 0.850162583
##         855         856         857         858         859         860
## 0.367121263 0.218470990 0.007584698 0.056801250 0.715666204 0.011616026
##         861         862         863         864         865         866
## 0.014546491 0.139724499 0.512977246 0.053756599 0.081656396 0.110961798
##         867         868         869         870         871         872
## 0.723270786 0.034689442 0.154722431 0.656477009 0.067671457 0.019639384
##         873         874         875         876         877         878
## 0.089077924 0.100682778 0.572771449 0.249846644 0.837453365 0.175379454
##         879         880         881         882         883         884
## 0.552061103 0.022474003 0.024071507 0.068265275 0.286812549 0.057264818
##         885         886         887         888         889         890
## 0.291907310 0.708205615 0.159045272 0.692645355 0.285240029 0.108253821
##         891         892         893         894         895         896
## 0.660354170 0.030660053 0.252200651 0.131762568 0.030060544 0.045916786
##         897         898         899         900         901         902
## 0.764274161 0.002260822 0.029118832 0.396208430 0.177754416 0.089072390
```

```
##         903         904         905         906         907         908 
## 0.033130736 0.069265758 0.070153484 0.349083915 0.089280836 0.578100771 
##         909         910         911         912         913         914 
## 0.042113526 0.231742665 0.345169829 0.417336266 0.288086941 0.018535757 
##         915         916         917         918         919         920 
## 0.764703977 0.735049361 0.025182424 0.637366984 0.496835774 0.527192120 
##         921         922         923         924         925         926 
## 0.223426837 0.146645464 0.636265744 0.412377649 0.845297386 0.789307659 
##         927         928         929         930         931         932 
## 0.600291841 0.747589384 0.027781668 0.610062339 0.340354659 0.413432099 
##         933         934         935         936         937         938 
## 0.073366311 0.035127670 0.662365992 0.676837015 0.179619873 0.523410323 
##         939         940         941         942         943         944 
## 0.889895390 0.020700237 0.096338400 0.041731089 0.025449900 0.037123674 
##         945         946         947         948         949         950 
## 0.431405415 0.901710271 0.734155496 0.188055727 0.455007219 0.061141123 
##         951         952         953         954         955         956 
## 0.392351006 0.189781260 0.326839022 0.668653006 0.508768251 0.221312879 
##         957         958         959         960         961         962 
## 0.145521309 0.091935096 0.571133671 0.365144581 0.059047074 0.642797055 
##         963         964         965         966         967         968 
## 0.157260184 0.069603946 0.550859613 0.437448393 0.286301992 0.241348376 
##         969         970         971         972         973         974 
## 0.113026510 0.293030308 0.166483186 0.250848967 0.954185672 0.949609436 
##         975         976         977         978         979         980 
## 0.108473034 0.133194265 0.054357256 0.230486947 0.304280382 0.802557955 
##         981         982         983         984         985         986 
## 0.117307671 0.420017656 0.349265835 0.493519401 0.023038231 0.625779121 
##         987         988         989         990         991         992 
## 0.855174343 0.038954890 0.480477108 0.231387578 0.083233821 0.267039238 
##         993         994         995         996         997         998 
## 0.248963549 0.670370517 0.048986213 0.046981948 0.592060065 0.060649024 
##         999        1000 
## 0.617781397 0.156366065
```

```r
#Looking at threshold prob. values. 0.5 is threshold value
thresh <- 0.5
yhat_threshold <- as.integer(yhat > thresh)
conf_matrix <- as.matrix(table(yhat_threshold,german_test$V21))
conf_matrix
```

```
## 
## yhat_threshold   0   1
##              0 115  29
##              1  24  32
```