Statistical Learning with R: Using the API Dataset and a Simulated Dataset

**By Ira Sharenow**  
**February 28, 2014**

# File Name: Applied Statistics with R.html  
# Version: Version 1  
# Web Address: http://irasharenow.com/Projects.html  
# Statistical Analyst: Ira Sharenow  
# Associated Document:   
# API and AHS by Ira Sharenow.docx -- First paper on topic  
# Web Address of the Original Data Files from the California Department of Education (CDE):  
# http://www.cde.ca.gov/ta/ac/ap/   
# R Version: 3.0.2  
# RStudio Version: 0.97.551  
# Windows: Windows 7

**Introduction and Background Information**

This document has four parts. Page down several times to get past the preliminaries.

Part 0. Preliminaries  
Part 1. Continuous response variable  
Part 2. Categorical Analysis setting  
Appendix 1. Reading and cleaning data

**Part 0. Preliminaries**

I started out by reading in the Part 1 data frame, *APIBase2009H*, which contains data on 647 California high schools. In this part I also activated the libraries that contain the function I will need in the remainder of the document. There is also some background information.

**Part 1. Continuous response variable**

I tried to fit the data using a variety of techniques, starting with traditional regression techniques and then moving onto a number of more modern methods.

The response variable is API09.  
The predictors are AVGED09, MEALS09, and EL09

Each school receives an index score, API09, which is a summary measure for student performance on a number of tests.

The three predictors that I use measure several demographic variables: average parental education, percent of students on a subsidized meals program, and percent of students who were classified English Learners.

We will see that standard regression works quite well, but I think it was useful to look at the modern techniques as well.

**Part 2. Categorical analysis setting**

In this part I first divided a rectangular portion of the plane into two regions and then in the second section of this part I divided the plane into five regions and then gave each region a separate label (color). Then I tried several classification techniques. The performance of the various techniques on this dataset is quite varied. The standard linear measures do very poorly.

**Appendix 1. Reading and cleaning data**

In the appendix, I indicated how I read in the data from database dbf files and then cleaned the data in order to have a suitable data frame for my analysis.

In addition I showed how SQL can be used from within R in place of a number of the standard R techniques.

The attempt is to show how R can be used interactively to analyze data. This document only includes a few of the many techniques that I attempted. The idea is to give a flavor of what can be done.

The references provide far more code and analyses.

**Additional remarks**

The data is from the Academic Performance Index (API) section of the California Department of Education website.

<http://www.cde.ca.gov/ta/ac/ap/>

The main references for this paper are:

James, Witten, Hastie, Tibshirani. *An Introduction to Statistical Learning with Applications in R*  
Julian Faraway, *Linear Models with R*

For more background, please see my previous paper *API and AHS by Ira Sharenow.docx.*

All files related to this document can be found at my website: <http://irasharenow.com/Projects.html>

This paper was written using the R markdown language from within RStudio.

The main data frame for this analysis is *APIBase2009H*. It can be found in the file: APIDataFinal0209H.rda

**Background information on the Variables**

As a result of the No Child Left Behind legislation which was promoted by former President George W. Bush, California students take a large number of tests and each year each school receives a score called the Academic Performance Index (API). A score of 800 is considered satisfactory. The California Department of Education (CDE) compares schools based on demographic information. Their predictor, SCI, is itself a composite of more recognizable predictors. The three main underlying predictors are **parental education (AVGED)**, **percent of English Learners (EL)**, and **percent of students on a subsidized meals plan (MEALS)**. AVGED is by far the most important predictor.

In my previous paper I utilized data from 2002-2009. However, this time I only use 2009 data.

As a result of conversations with CDE statisticians, I focused on high school data and limited schools to those of at least modest size and in other ways. As a result I wound up with 647 schools and the three predictors. See the appendix for more details.

Finally in the comments after code that utilized a random number generator, I sometimes give a probability. That value may sometimes differ from the value given in the output.

## Table of Contents

### Part 1. Continuous Response

### Chapter 1. Basic Regression

1. Basic analysis for 1 predictor model
2. Basic analysis for 3 predictor model
3. Predictions for AHS
4. A brief examination of residuals
5. Exploratory plots

### Chapter 2. Model Selection for Regression

1. Method 1. Backward selection: delete variables with largest p value if p > 0.05
2. Method 2. Forward selection using Akaike Information Criterion AIC
3. Use the leaps function so that R can automatically make the decisions
4. Graphs

### Chapter 3. Resampling in a regression setting

1. Validation set approach to compare models
2. Leave one out cross-validation to compare models
3. Bootstrap to estimate the 90th percentile of the API09 variable

### Chapter 4. Advanced Regression Techniques

1. Ridge regression
2. Selecting ridge regression's best lambda via cross-validation
3. The lasso
4. Selecting lasso's best lambda via cross-validation
5. Principal components regression
6. Selecting the best pc regression using the validation set approach

### Chapter 5. Tree methods for Regression

1. Basic regression trees
2. Bagging
3. Random Forests
4. Boosting

### Chapter 6. Summary of continuous model results

### Part 2. Classification

### Chapter 7. Simulation of new dataset

1. Create a data frame with two categories
2. Graph the 2 category data
3. Create a data frame with five categories
4. Graph the 5 category data

### Chapter 8. Classification modeling for 2 categories

1. Logistic regression
2. Linear discriminant analysis (LDA)
3. Quadratic discriminant analysis (QDA)
4. Basic binary tree

### Chapter 9. Classification modeling for 5 categories

1. LDA
2. QDA
3. Classification tree
4. Bagging
5. Random forests
6. Support Vector Machines
7. KNN

### Chapter 10. Summary of Classification results

1. Tables/Confusion matrices

### Appendix 1. Reading, cleaning, and merging of data

1. Meaning of predictor variables
2. Reading data
3. Cleaning data
4. Merging data
5. SQL alternatives

**Preliminaries**

# Go to the directory where the data files are  
inputDir = "C:/Users/Ira/Documents/Statistics/My Website/Project 1 -- API/API 2014/Input"  
setwd(inputDir) # location of data files  
  
AHSCDS = "01611270130450" # Albany HS ID number  
outputFile = "API output, Version 1.xlsx" # Later some data frames will be sent to this file  
  
library(foreign) # needed for read.dbf which reads in the database files  
library(sqldf) # needed for SQL code

## Loading required package: DBI  
## Loading required package: gsubfn  
## Loading required package: proto  
## Loading required namespace: tcltk  
## Loading required package: chron  
## Loading required package: RSQLite  
## Loading required package: RSQLite.extfuns

library(car)   
library(class) # for K-nearest neighbors  
library(MASS)  
library(boot) # needed for bootstrap

##   
## Attaching package: 'boot'  
##   
## The following object is masked from 'package:car':  
##   
## logit

library(tree) # for decision trees  
library(randomForest) # for bagging and random forests

## randomForest 4.6-7  
## Type rfNews() to see new features/changes/bug fixes.

library(gbm) # for boosting

## Loading required package: survival  
## Loading required package: splines  
##   
## Attaching package: 'survival'  
##   
## The following object is masked from 'package:boot':  
##   
## aml  
##   
## Loading required package: lattice  
##   
## Attaching package: 'lattice'  
##   
## The following object is masked from 'package:boot':  
##   
## melanoma  
##   
## Loading required package: parallel  
## Loaded gbm 2.1

library (e1071) # for support vector machines  
library(glmnet) # for ridge regression and lasso

## Loading required package: Matrix  
## Loaded glmnet 1.9-5

library(leaps) # for best subset regression  
library(splines) # for splines  
library(pls) # for principal components regression

##   
## Attaching package: 'pls'  
##   
## The following object is masked from 'package:stats':  
##   
## loadings

# Load this file and then start the analysis  
# This file contains two data frames  
# One data frame has all of the data  
# The second data frame contains only 2009 data and is used for the analysis  
  
load(file = "APIDataFinal0209H.rda") # Contains two data frames  
head(APIBase2009H) # Preview the 2009 data. Analysis based on this data frame

## CDS SNAME DNAME CNAME API09  
## 1 01611190130229 Alameda High Alameda City Unified Alameda 812  
## 2 01611190132878 Encinal High Alameda City Unified Alameda 747  
## 3 01611270130450 Albany High Albany City Unified Alameda 806  
## 4 01611500132225 Castro Valley High Castro Valley Unified Alameda 806  
## 5 01611760130062 American High Fremont Unified Alameda 787  
## 6 01611760134270 Irvington High Fremont Unified Alameda 831  
## AVGED09 MEALS09 EL09 SCI09 NOT\_HSG09 HSG09 SOME\_COL09 COL\_GRAD09  
## 1 3.53 20 18 179.0 5 18 20 31  
## 2 2.92 48 24 162.3 13 26 28 24  
## 3 4.10 13 13 182.5 3 7 13 32  
## 4 3.64 14 4 179.1 2 12 28 35  
## 5 3.53 19 10 178.2 6 16 19 38  
## 6 3.79 13 10 182.8 4 12 17 36  
## GRAD\_SCH09  
## 1 25  
## 2 10  
## 3 45  
## 4 23  
## 5 21  
## 6 32

#######################   
# Part 1 #   
# Continuous response #   
#######################   
  
##################################   
#Chapter 1 #   
#Standard regression exploration #   
#with diagnostics #   
##################################

In the R code file I tested all 8 possible models and utilized an assortment of exploratory techniques. In this document I demonstrate a few of them.  
It appears as though the standard regression framework is quite adequate even though the variance is not constant.

1. Basic analysis for 1 predictor model
2. Basic analysis for 3 predictor model
3. Predictions for AHS
4. A brief examination of residuals
5. Exploratory plots

dim(APIBase2009H)

## [1] 647 14

names(APIBase2009H)

## [1] "CDS" "SNAME" "DNAME" "CNAME" "API09"   
## [6] "AVGED09" "MEALS09" "EL09" "SCI09" "NOT\_HSG09"   
## [11] "HSG09" "SOME\_COL09" "COL\_GRAD09" "GRAD\_SCH09"

# The best 1 predictor   
  
HS09.lm1AVGED = lm(API09 ~ AVGED09, data = APIBase2009H) # Average parental education  
HS09.lm1AVGED

##   
## Call:  
## lm(formula = API09 ~ AVGED09, data = APIBase2009H)  
##   
## Coefficients:  
## (Intercept) AVGED09   
## 464.0 98.7

summary(HS09.lm1AVGED)

##   
## Call:  
## lm(formula = API09 ~ AVGED09, data = APIBase2009H)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -148.7 -20.5 0.8 22.8 178.2   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 463.98 6.21 74.7 <2e-16 \*\*\*  
## AVGED09 98.66 2.13 46.3 <2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 37.5 on 645 degrees of freedom  
## Multiple R-squared: 0.769, Adjusted R-squared: 0.769   
## F-statistic: 2.15e+03 on 1 and 645 DF, p-value: <2e-16

lm1AVGEDRsq = summary(HS09.lm1AVGED)$adj.r.squared

# 3 predictors   
  
HS09.lm3AVGEDMealsEL = lm(API09 ~ AVGED09 + MEALS09 + EL09, data = APIBase2009H)  
HS09.lm3AVGEDMealsEL

##   
## Call:  
## lm(formula = API09 ~ AVGED09 + MEALS09 + EL09, data = APIBase2009H)  
##   
## Coefficients:  
## (Intercept) AVGED09 MEALS09 EL09   
## 510.604 87.055 -0.123 -0.578

summary(HS09.lm3AVGEDMealsEL)

##   
## Call:  
## lm(formula = API09 ~ AVGED09 + MEALS09 + EL09, data = APIBase2009H)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -132.46 -21.29 1.36 21.83 173.46   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 510.604 18.304 27.90 <2e-16 \*\*\*  
## AVGED09 87.055 4.704 18.51 <2e-16 \*\*\*  
## MEALS09 -0.123 0.130 -0.95 0.3426   
## EL09 -0.578 0.199 -2.91 0.0037 \*\*   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 37.2 on 643 degrees of freedom  
## Multiple R-squared: 0.773, Adjusted R-squared: 0.772   
## F-statistic: 731 on 3 and 643 DF, p-value: <2e-16

lm3AVGEDMealsELRsq = summary(HS09.lm3AVGEDMealsEL)$adj.r.squared

#######################  
# Predictions for AHS #  
#######################

attach(APIBase2009H)  
# 1 predictor models  
predict(HS09.lm1AVGED, newdata = data.frame(AVGED09 = AVGED09[CDS == AHSCDS],   
 MEALS09 = MEALS09[CDS == AHSCDS],  
 EL09 = EL09[CDS == AHSCDS]))

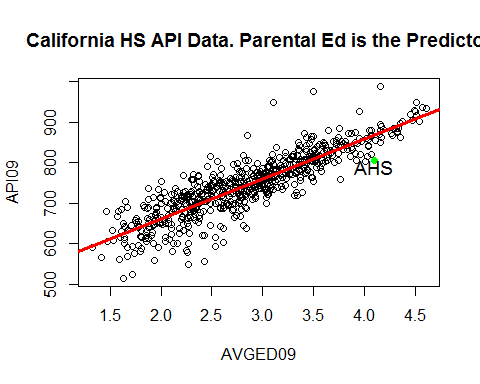
## 1   
## 868.5

# 3 predictor model  
predict(HS09.lm3AVGEDMealsEL, newdata = data.frame(AVGED09 = AVGED09[CDS == AHSCDS],   
 MEALS09 = MEALS09[CDS == AHSCDS],  
 EL09 = EL09[CDS == AHSCDS]))

## 1   
## 858.4

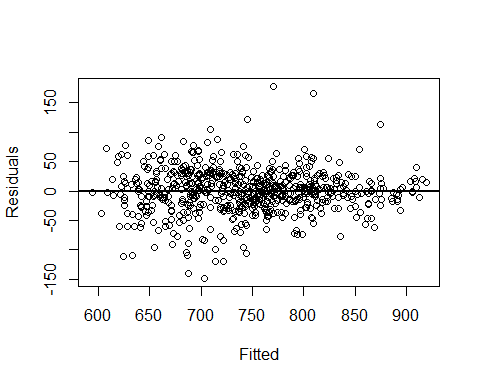
###############  
# Diagnostics #  
###############

plot(AVGED09, API09, main = "California HS API Data. Parental Ed is the Predictor")  
points(4.1, 806, col = "green", pch = 19)  
text(4.1, 790, labels = "AHS", col = 'black', cex = 1.2)  
abline(HS09.lm1AVGED, lwd = 3, col = "red")

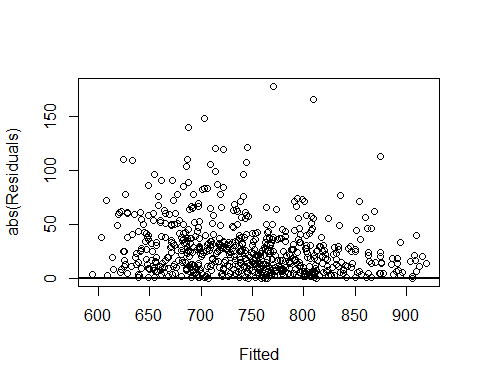


# The plot indicates that a linear model is appropriate

# Check for non-constant variance  
  
# Plot of residuals versus fitted  
plot(fitted(HS09.lm1AVGED), residuals(HS09.lm1AVGED), xlab = "Fitted", ylab = "Residuals")  
abline(h = 0, lwd = 2)



# Plot of abs(residuals) versus fitted  
plot(fitted(HS09.lm1AVGED), abs(residuals(HS09.lm1AVGED)), xlab = "Fitted", ylab = "abs(Residuals)")  
abline(h = 0, lwd = 2)



# A quick, rough check for non-constant variance  
# summary(lm(abs(residuals(HS09.lm1AVGED)) ~ fitted(HS09.lm1AVGED)))  
# Adjusted R-squared is only 0.04  
  
var.test(residuals(HS09.lm1AVGED)[AVGED09 > 3.75], residuals(HS09.lm1AVGED)[AVGED09 <= 3.75])

##   
## F test to compare two variances  
##   
## data: residuals(HS09.lm1AVGED)[AVGED09 > 3.75] and residuals(HS09.lm1AVGED)[AVGED09 <= 3.75]  
## F = 0.5774, num df = 64, denom df = 581, p-value = 0.007171  
## alternative hypothesis: true ratio of variances is not equal to 1  
## 95 percent confidence interval:  
## 0.4108 0.8571  
## sample estimates:  
## ratio of variances   
## 0.5774

# There may be an issue with non-constant variance

############################   
#Chapter 2 #   
#Standard regression model #   
#selection #   
############################

In Chapter 2, I attempt a number of model selection techniques  
1. Method 1. Backward selection: delete variables with largest p value if p > 0.05  
2. Method 2. Forward selection using Akaike Information Criterion AIC  
3. Use the leaps function so that R can automatically make the decisions  
4. Graphs

Method 1. Backward selection (delete variable with largest p value, if over .05)

g3 = lm(API09 ~ AVGED09 + MEALS09 + EL09, data = APIBase2009H)  
summary(g3)

##   
## Call:  
## lm(formula = API09 ~ AVGED09 + MEALS09 + EL09, data = APIBase2009H)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -132.46 -21.29 1.36 21.83 173.46   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 510.604 18.304 27.90 <2e-16 \*\*\*  
## AVGED09 87.055 4.704 18.51 <2e-16 \*\*\*  
## MEALS09 -0.123 0.130 -0.95 0.3426   
## EL09 -0.578 0.199 -2.91 0.0037 \*\*   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 37.2 on 643 degrees of freedom  
## Multiple R-squared: 0.773, Adjusted R-squared: 0.772   
## F-statistic: 731 on 3 and 643 DF, p-value: <2e-16

#The signs of the coefficients are correct  
#but the MEALS09 variable is not significant and is removed  
  
g2 = update(g3, . ~ . - MEALS09)  
summary(g2)

##   
## Call:  
## lm(formula = API09 ~ AVGED09 + EL09, data = APIBase2009H)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -131.70 -21.23 1.42 22.33 172.99   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 497.209 11.666 42.62 < 2e-16 \*\*\*  
## AVGED09 90.260 3.278 27.54 < 2e-16 \*\*\*  
## EL09 -0.635 0.189 -3.35 0.00084 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 37.2 on 644 degrees of freedom  
## Multiple R-squared: 0.773, Adjusted R-squared: 0.772   
## F-statistic: 1.1e+03 on 2 and 644 DF, p-value: <2e-16

#Since all predictors are now significant one could stop here  
#but it would be sensible to see what happens to such quantities  
#as adjusted R-squared and residual standard error before making a final decision  
  
g1 = update(g2, . ~ . - EL09)  
summary(g1)

##   
## Call:  
## lm(formula = API09 ~ AVGED09, data = APIBase2009H)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -148.7 -20.5 0.8 22.8 178.2   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 463.98 6.21 74.7 <2e-16 \*\*\*  
## AVGED09 98.66 2.13 46.3 <2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 37.5 on 645 degrees of freedom  
## Multiple R-squared: 0.769, Adjusted R-squared: 0.769   
## F-statistic: 2.15e+03 on 1 and 645 DF, p-value: <2e-16

#As noted above there is little change in adjusted R-squared and residual standard error  
#So the single variable, AVGED09 model is probably the best way to go

#Method 2. Forward selection using Akaike Information Criterion AIC (Also observe adjusted R-squared)  
gA3 = lm(API09 ~ 1, data = APIBase2009H) # No predictors  
  
step(gA3, scope=list(upper = ~ AVGED09 + MEALS09 + EL09), direction="forward") # All predictors

## Start: AIC=5638  
## API09 ~ 1  
##   
## Df Sum of Sq RSS AIC  
## + AVGED09 1 3017610 906894 4692  
## + MEALS09 1 2456331 1468172 5003  
## + EL09 1 1983639 1940865 5184  
## <none> 3924504 5638  
##   
## Step: AIC=4692  
## API09 ~ AVGED09  
##   
## Df Sum of Sq RSS AIC  
## + EL09 1 15571 891322 4683  
## + MEALS09 1 5097 901796 4690  
## <none> 906894 4692  
##   
## Step: AIC=4683  
## API09 ~ AVGED09 + EL09  
##   
## Df Sum of Sq RSS AIC  
## <none> 891322 4683  
## + MEALS09 1 1249 890074 4684

##   
## Call:  
## lm(formula = API09 ~ AVGED09 + EL09, data = APIBase2009H)  
##   
## Coefficients:  
## (Intercept) AVGED09 EL09   
## 497.209 90.260 -0.635

#We see that, as expected, the model stops with AVGED09 and EL09 in the model

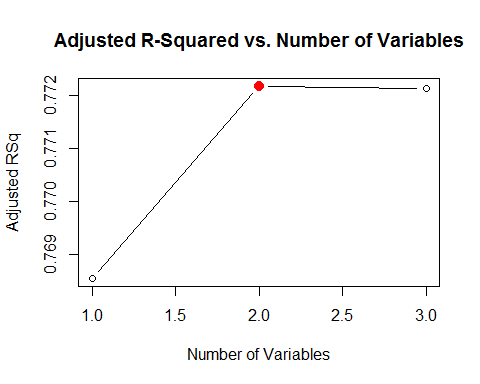
library(leaps)  
#Use the leaps library to automate some of the above strategies  
#Find the best model for 1, 2, and 3 predictors  
regfit.full = regsubsets(API09 ~ AVGED09 + MEALS09 + EL09, APIBase2009H)  
reg.summary = summary(regfit.full)  
  
reg.summary$adjr2 # adjusted R-Squared for each model. All of them are about 0.77.

## [1] 0.7686 0.7722 0.7721

#A plot of the adjusted R-Squared  
plot(reg.summary$adjr2, main = "Adjusted R-Squared vs. Number of Variables", xlab = "Number of Variables",  
 ylab = "Adjusted RSq", type ="b")  
  
#Plot the max point  
which.max(reg.summary$adjr2)

## [1] 2

points(2, reg.summary$adjr2[2], col = "red", cex =2, pch =20)



######################  
# Chapter 3 #  
# Resampling in a #  
# Regression setting #  
######################

In Chapter 2, I used all of the data to create models and assess them.  
A better way, one that can address overfitting, is to use resampling techniques.  
Resampling in the regression setting is the topic of this chapter.  
I compare the best 1 predictor model to the 3 predictor model.

1. Validation set approach to compare models
2. Leave one out cross-validation to compare models
3. Bootstrap to estimate the 90th percentile of the API09 variable

# Chapter 1 model. Repeated for convenience  
HS09.lm1AVGED = lm(API09 ~ AVGED09, data = APIBase2009H) # Average parental education  
HS09.lm3AVGEDMealsEL = lm(API09 ~ AVGED09 + MEALS09 + EL09, data = APIBase2009H)  
HS09.summary1 = summary(HS09.lm1AVGED)  
  
summary(HS09.summary1)

## Length Class Mode   
## call 3 -none- call   
## terms 3 terms call   
## residuals 647 -none- numeric  
## coefficients 8 -none- numeric  
## aliased 2 -none- logical  
## sigma 1 -none- numeric  
## df 3 -none- numeric  
## r.squared 1 -none- numeric  
## adj.r.squared 1 -none- numeric  
## fstatistic 3 -none- numeric  
## cov.unscaled 4 -none- numeric

dim(APIBase2009H)

## [1] 647 14

set.seed(2014)  
  
# Randomly divide data into a training set and a test set  
# Use the models created with the training set on the test set.  
# Compute the MSE for each model and compare  
train = sample(1:647, 327)  
test = (-train)  
  
lm.fit1 = lm(API09 ~ AVGED09, data = APIBase2009H, subset = train)  
summary(lm.fit1)

##   
## Call:  
## lm(formula = API09 ~ AVGED09, data = APIBase2009H, subset = train)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -145.64 -20.12 1.05 22.47 179.73   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 455.39 8.58 53.0 <2e-16 \*\*\*  
## AVGED09 100.93 2.94 34.4 <2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 36.1 on 325 degrees of freedom  
## Multiple R-squared: 0.784, Adjusted R-squared: 0.783   
## F-statistic: 1.18e+03 on 1 and 325 DF, p-value: <2e-16

mean((API09 - predict(lm.fit1, APIBase2009H))[- train ]^2)

## [1] 1526

lm.fit3 = lm(API09 ~ AVGED09 + MEALS09 + EL09, data = APIBase2009H, subset = train)  
summary(lm.fit3)

##   
## Call:  
## lm(formula = API09 ~ AVGED09 + MEALS09 + EL09, data = APIBase2009H,   
## subset = train)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -125.62 -21.27 1.14 19.88 173.76   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 492.13069 24.61358 19.99 <2e-16 \*\*\*  
## AVGED09 91.66159 6.35237 14.43 <2e-16 \*\*\*  
## MEALS09 0.00781 0.17621 0.04 0.9647   
## EL09 -0.73888 0.27162 -2.72 0.0069 \*\*   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 35.8 on 323 degrees of freedom  
## Multiple R-squared: 0.789, Adjusted R-squared: 0.787   
## F-statistic: 404 on 3 and 323 DF, p-value: <2e-16

mean((API09 - predict(lm.fit3, APIBase2009H))[- train ]^2)

## [1] 1514

mean((API09 - predict(HS09.lm1AVGED, APIBase2009H))^2)

## [1] 1402

# The validation approach does not appear to indicate there is much difference  
# between the 2 models

# Leave one out cross validation (loocv)  
# In this method each data point is left out and becomes the test set   
# with all others being used as the training set.  
# Since there is a lot of overlap in traing sets, there is the potential for  
# high variance  
  
# Use glm instead of lm  
glm.fit1 = glm(API09 ~ AVGED09, data = APIBase2009H)  
  
library(boot) # for cv.glm   
  
#This takes a little time to run  
cv.err1 = cv.glm(APIBase2009H, glm.fit1) # takes a bit of time on my computer  
cv.err1$delta

## [1] 1410 1410

glm.fit3 = glm(API09 ~ AVGED09 + MEALS09 + EL09, data = APIBase2009H)  
cv.err3 = cv.glm(APIBase2009H, glm.fit3)  
cv.err3$delta

## [1] 1397 1397

#Again the 3 predictor model only does slightly better than the 1 predictor model  
#Note that delta refers to estimates of the loocv MSE  
#MSE\_n = (1/n) \* sum(MSE\_i)  
#Since the code needs to be run n times, this can be quite time consuming

#Create a function that computes the 90th percentile (top 10%)  
set.seed(2014)  
q90 = function(x, index) {  
 return(quantile(APIBase2009H$API09[index], probs = 0.90))  
}  
q90(APIBase2009H$API09, sample(1:647, 647, replace = TRUE))

## 90%   
## 850

# Now the bootstrap  
boot(APIBase2009H$API09, q90, R = 1000)

##   
## ORDINARY NONPARAMETRIC BOOTSTRAP  
##   
##   
## Call:  
## boot(data = APIBase2009H$API09, statistic = q90, R = 1000)  
##   
##   
## Bootstrap Statistics :  
## original bias std. error  
## t1\* 845 -0.9156 5.276

# So the AHS API score of 806 is way below the top 10% score of about 845. AHS has a very high AVGED  
# This runs fast

#Now use the bootstrap to estimate the regression coefficients  
  
boot.fn = function(data, index) {  
 return(coef (lm(API09 ~ AVGED09, data = data, subset = index )))  
}  
  
boot.fn(APIBase2009H, 1:647) # AVGED coef = 98.66426

## (Intercept) AVGED09   
## 463.98 98.66

boot.fn(APIBase2009H, sample(1:647, 647)) # AVGED coef = 98.66426. Oops need to sample with replacement

## (Intercept) AVGED09   
## 463.98 98.66

boot.fn(APIBase2009H, sample(1:647, 647, replace = TRUE)) # AVGED coef = 102.8486. Different sample, different result

## (Intercept) AVGED09   
## 471.30 96.46

set.seed(2014)  
boot(APIBase2009H, boot.fn, 1000) # std error for AVGED09 = 2.092507

##   
## ORDINARY NONPARAMETRIC BOOTSTRAP  
##   
##   
## Call:  
## boot(data = APIBase2009H, statistic = boot.fn, R = 1000)  
##   
##   
## Bootstrap Statistics :  
## original bias std. error  
## t1\* 463.98 -0.04828 6.577  
## t2\* 98.66 -0.01160 2.093

#Now try with 3 predictors  
  
boot.fn3 = function(data, index) {  
 return(coef (lm(API09 ~ AVGED09 + MEALS09 + EL09, data = data, subset = index )))  
}  
  
boot.fn3(APIBase2009H, 1:647) # AVGED coef = 87.0550391

## (Intercept) AVGED09 MEALS09 EL09   
## 510.6042 87.0550 -0.1233 -0.5780

boot.fn3(APIBase2009H, sample(1:647, 647, replace = TRUE)) # AVGED coef = 84.6871307 Different sample, different # result

## (Intercept) AVGED09 MEALS09 EL09   
## 472.64136 96.51059 0.01955 -0.22065

set.seed(2014)  
boot(APIBase2009H, boot.fn3, 1000) # std error for AVGED09 = 5.1375519

##   
## ORDINARY NONPARAMETRIC BOOTSTRAP  
##   
##   
## Call:  
## boot(data = APIBase2009H, statistic = boot.fn3, R = 1000)  
##   
##   
## Bootstrap Statistics :  
## original bias std. error  
## t1\* 510.6042 -0.580062 20.1141  
## t2\* 87.0550 0.121090 5.1376  
## t3\* -0.1233 0.002133 0.1525  
## t4\* -0.5780 0.004476 0.2458

#This is a further indication that the 1 predictor model is better. The se just more than doubled  
  
summary(lm(API09 ~ AVGED09, data = APIBase2009H))$coef # se for AVGED09 2.129740

## Estimate Std. Error t value Pr(>|t|)  
## (Intercept) 463.98 6.209 74.73 7.988e-320  
## AVGED09 98.66 2.130 46.33 2.345e-207

summary(lm(API09 ~ AVGED09 + MEALS09 + EL09, data = APIBase2009H))$coef # se for AVGED09 4.7043129

## Estimate Std. Error t value Pr(>|t|)  
## (Intercept) 510.6042 18.3038 27.8961 7.757e-113  
## AVGED09 87.0550 4.7043 18.5054 1.295e-61  
## MEALS09 -0.1233 0.1298 -0.9497 3.426e-01  
## EL09 -0.5780 0.1986 -2.9101 3.738e-03

#Again more than double

######################  
#Chapter 4 #  
#Advanced Regression #  
#Techniques #  
######################

Regularization/shrinkage is very valuable when there are many predictors.  
I do not have many predictors, so I am just going to run some code without saying much about it.

1. Ridge regression
2. The lasso
3. Principal components regression

#First recall the standard method  
lm(API09 ~ AVGED09 + MEALS09 + EL09, data = APIBase2009H)

##   
## Call:  
## lm(formula = API09 ~ AVGED09 + MEALS09 + EL09, data = APIBase2009H)  
##   
## Coefficients:  
## (Intercept) AVGED09 MEALS09 EL09   
## 510.604 87.055 -0.123 -0.578

x = model.matrix(API09 ~ AVGED09 + MEALS09 + EL09, APIBase2009H)   
y = APIBase2009H$API09  
  
#ridge regress ion (alpha = 0)  
#alpha = 1 is lasso  
ridge.mod = glmnet(x, y, alpha = 0)  
dim(coef(ridge.mod))

## [1] 5 100

ridge.mod$lambda[50]

## [1] 715.5

nrow(x)

## [1] 647

grid = 10^seq(10, -2, length = 100)  
ridge.mod = glmnet(x, y, alpha = 0, lambda = grid)  
  
dim(coef(ridge.mod)) # 5 100

## [1] 5 100

ridge.mod$lambda[50] # lambda = 11497.57

## [1] 11498

coef(ridge.mod)[,50]

## (Intercept) (Intercept) AVGED09 MEALS09 EL09   
## 742.67778 0.00000 0.65758 -0.01629 -0.03071

ridge.mod$lambda[100] # lambda = 0.01 very little shrinkage

## [1] 0.01

coef(ridge.mod)[,100]

## (Intercept) (Intercept) AVGED09 MEALS09 EL09   
## 510.9451 0.0000 86.9655 -0.1250 -0.5792

sqrt(sum(coef(ridge.mod)[-1, 50]^2)) # 0.6585026 l2 norm

## [1] 0.6585

#validation  
set.seed (2014)  
train = sample (1:nrow(x), 327)  
test = (-train )  
y.test = y[test]  
  
ridge.mod = glmnet(x[train , ], y[train], alpha = 0, lambda = grid,  
 thresh = 1e-12)  
ridge.pred = predict(ridge.mod, s = 4, newx = x[test ,]) # predict with lambda = 4  
mean((ridge.pred - y.test)^2) # MSE 1506.487

## [1] 1506

#Compare to the null which just predicts the average for every observation  
mean((mean(y[train]) - y.test)^2) # MSE 6140.538

## [1] 6141

#A large value of lambda will shrink the regression coefficients towards 0 and   
#produce a similar result  
  
ridge.pred = predict(ridge.mod , s = 1e10, newx = x[test ,])  
mean((ridge.pred - y.test)^2) # MSE 6140.538

## [1] 6141

ridge.pred = predict(ridge.mod, s = 0, newx =x[test ,], exact =T) # lambda = 0 is the same as standard lm  
mean((ridge.pred - y.test)^2) # 1513.96

## [1] 1514

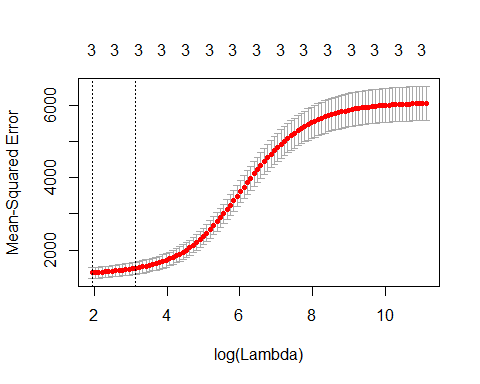
lm(y ~ x, subset = train)

##   
## Call:  
## lm(formula = y ~ x, subset = train)  
##   
## Coefficients:  
## (Intercept) x(Intercept) xAVGED09 xMEALS09 xEL09   
## 492.13069 NA 91.66159 0.00781 -0.73888

predict(ridge.mod , s = 0, exact = T, type = "coefficients")[1:5,]

## (Intercept) (Intercept) AVGED09 MEALS09 EL09   
## 492.131326 0.000000 91.661421 0.007805 -0.738885

#Now the cross-validation  
set.seed(2014)  
cv.out = cv.glmnet(x[train ,], y[train], alpha = 0)  
plot(cv.out)



bestlam = cv.out$lambda.min  
bestlam # 6.856295

## [1] 6.856

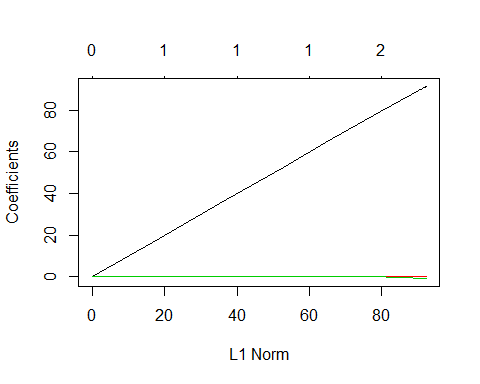
ridge.pred = predict(ridge.mod, s= bestlam, newx = x[test,])  
mean((ridge.pred - y.test)^2) # 1518.468

## [1] 1518

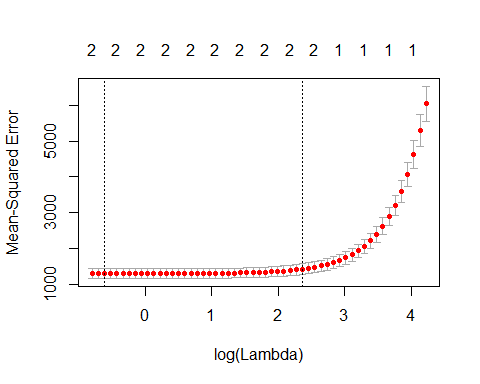
out = glmnet(x,y,alpha =0)  
predict(out, type = "coefficients", s = bestlam)[1:5 ,]

## (Intercept) (Intercept) AVGED09 MEALS09 EL09   
## 591.3733 0.0000 65.5546 -0.5057 -0.8379

#Now repeat the procedure using the lasso, alpha = 1  
lasso.mod = glmnet(x[train ,], y[train ], alpha = 1, lambda = grid)  
plot(lasso.mod)



set.seed(2014)  
cv.out = cv.glmnet(x[train ,], y[train ], alpha = 1)  
plot(cv.out)



bestlam = cv.out$lambda.min # 0.49508  
lasso.pred = predict(lasso.mod , s = bestlam, newx =x[test ,])  
mean((lasso.pred -y.test)^2) # 1509.479

## [1] 1509

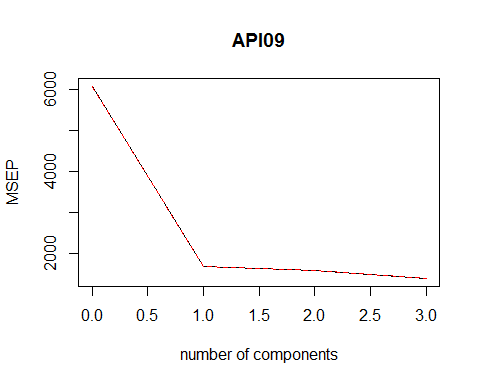
#Not much different than ridge regression. As indicated, this problem is  
#less than an ideal candidate for shrinkage methods  
  
out = glmnet(x,y, alpha = 1, lambda = grid)  
lasso.coef = predict(out, type = "coefficients", s = bestlam)[1:5,]  
lasso.coef

## (Intercept) (Intercept) AVGED09 MEALS09 EL09   
## 510.5941 0.0000 86.8280 -0.1161 -0.5542

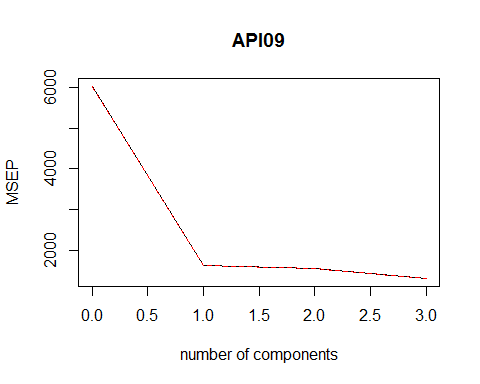
#Principal components regression  
  
library(pls) # for principal components regression  
set.seed(2014)  
pcr.fit = pcr(API09 ~ AVGED09 + MEALS09 + EL09, data = APIBase2009H, scale = TRUE,  
 validation = "CV")  
  
summary(pcr.fit)

## Data: X dimension: 647 3   
## Y dimension: 647 1  
## Fit method: svdpc  
## Number of components considered: 3  
##   
## VALIDATION: RMSEP  
## Cross-validated using 10 random segments.  
## (Intercept) 1 comps 2 comps 3 comps  
## CV 78 41.03 39.83 37.38  
## adjCV 78 41.02 39.82 37.37  
##   
## TRAINING: % variance explained  
## 1 comps 2 comps 3 comps  
## X 87.02 96.10 100.00  
## API09 72.47 74.19 77.32

validationplot(pcr.fit, val.type = "MSEP")



#Now perform cross-validation  
set.seed(2014)  
pcr.fit = pcr(API09 ~ AVGED09 + MEALS09 + EL09, data = APIBase2009H, subset = train, scale = TRUE,  
 validation = "CV")  
validationplot(pcr.fit, val.type = "MSEP")



pcr.pred = predict(pcr.fit , x[test, 2:4], ncomp = 3)  
mean((pcr.pred - y.test)^2) # 1513.961

## [1] 1514

pcr.pred = predict(pcr.fit , x[test, 2:4], ncomp = 2)  
mean((pcr.pred - y.test)^2) # 1649.928

## [1] 1650

pcr.pred = predict(pcr.fit , x[test, 2:4], ncomp = 1)  
mean((pcr.pred - y.test)^2) # 1775.506

## [1] 1776

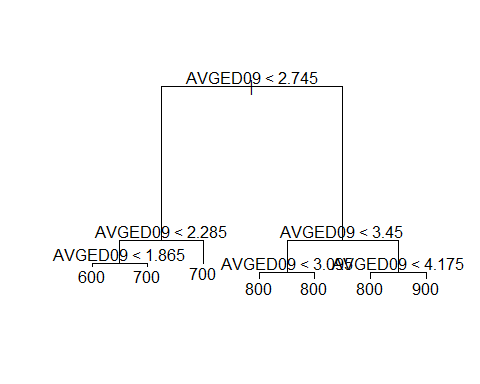
######################   
#Chapter 5 #   
#Tree Methods #   
#Continuous response #   
######################

In this chapter, I discuss a number of tree methods  
1. Basic regression tree. First fit data to training set  
2. Bagging  
3. Random forests  
4. Boosting

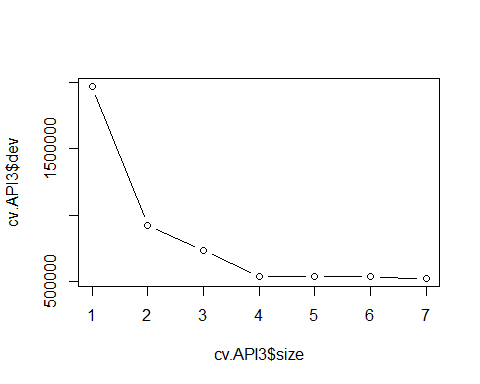
#Basic regression tree. First fit data to training set  
set.seed(2014)  
train = sample (1:nrow(APIBase2009H), 323)  
tree.API3 = tree(API09 ~ AVGED09 + MEALS09 + EL09, APIBase2009H, subset = train)  
summary(tree.API3)

##   
## Regression tree:  
## tree(formula = API09 ~ AVGED09 + MEALS09 + EL09, data = APIBase2009H,   
## subset = train)  
## Variables actually used in tree construction:  
## [1] "AVGED09"  
## Number of terminal nodes: 7   
## Residual mean deviance: 1270 = 401000 / 316   
## Distribution of residuals:  
## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## -157.00 -21.90 0.12 0.00 23.00 157.00

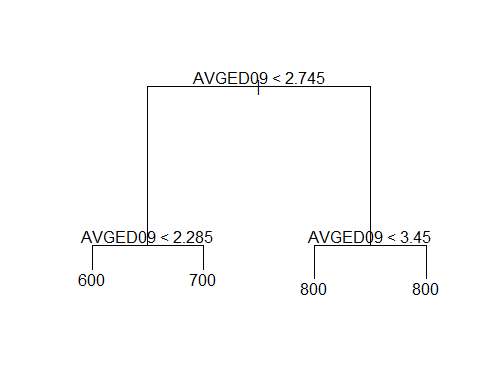
#We see that only AVGED09 was used. There were 7 terminal nodes  
#residual mean deviance is 1269  
  
plot(tree.API3)  
text(tree.API3, pretty = 0)



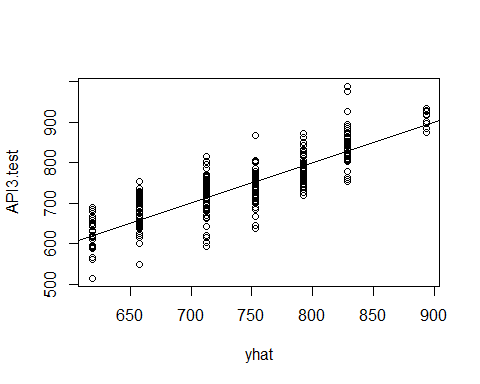
cv.API3 = cv.tree(tree.API3)  
plot(cv.API3$size, cv.API3$dev, type= 'b')



prune.API3 = prune.tree(tree.API3, best = 4)  
plot(prune.API3)  
text(prune.API3, pretty = 0)



# Pruning can help with overfitting and with interpretation  
#Use the unpruned tree on the test set  
  
yhat = predict(tree.API3, newdata = APIBase2009H[-train, ])  
API3.test = APIBase2009H[-train, "API09"]  
plot(yhat, API3.test)  
abline(0, 1)



mean((yhat - API3.test) ^2) # 1758.778

## [1] 1759

sqrt(1758.778) # 41.93779

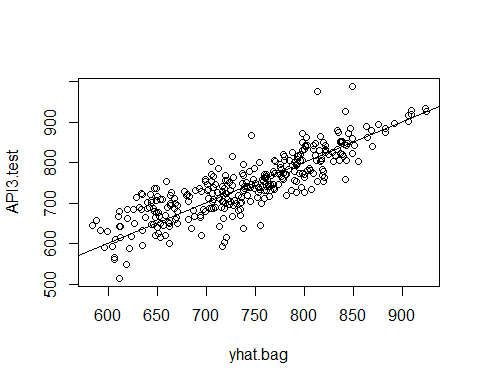
## [1] 41.94

#Recall the standard regression results  
#HS09.lm3AVGEDMealsEL = lm(API09 ~ AVGED09 + MEALS09 + EL09, data = APIBase2009H)  
#Residual standard error: 37.21  
  
#So the tree did somewhat worse

#Bagging  
#Bagging and random forests build multiple trees and then takes an average.  
#Random forests randomly choose a subset of predictors. This allows for more variation in the trees.  
  
library(randomForest)  
set.seed(2014)  
bag.API = randomForest(API09 ~ AVGED09 + MEALS09 + EL09, data = APIBase2009H, subset = train ,  
 mtry = 3, importance = TRUE)  
bag.API

##   
## Call:  
## randomForest(formula = API09 ~ AVGED09 + MEALS09 + EL09, data = APIBase2009H, mtry = 3, importance = TRUE, subset = train)   
## Type of random forest: regression  
## Number of trees: 500  
## No. of variables tried at each split: 3  
##   
## Mean of squared residuals: 1522  
## % Var explained: 74.82

#test the results  
yhat.bag = predict (bag.API, newdata = APIBase2009H[-train, ])  
plot(yhat.bag, API3.test)  
abline (0 ,1)



mean((yhat.bag - API3.test)^2) # 1711.402

## [1] 1709

sqrt(1711.402) # 41.36909

## [1] 41.37

#A small improvement over the basic tree method

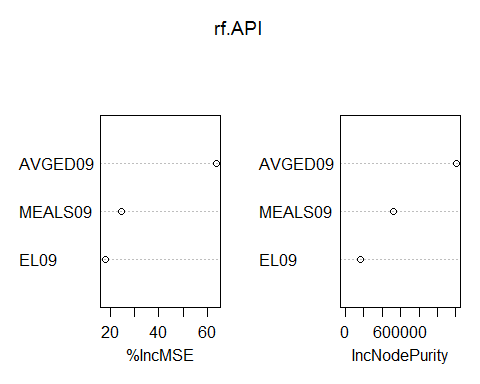
#Random Forests  
#Same idea as bagging but now mtry will be 2 instead of 3  
#So not all predictors will be tried at each iteration  
  
set.seed (2014)  
rf.API = randomForest(API09 ~ AVGED09 + MEALS09 + EL09, data = APIBase2009H, subset = train,  
 mtry = 2, importance = TRUE)  
yhat.rf = predict(rf.API, newdata = APIBase2009H[-train, ])  
mean((yhat.rf - API3.test)^2) # 1659.059

## [1] 1659

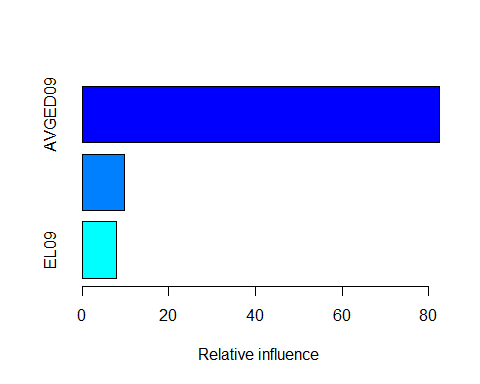
#An improvement over bagging  
  
importance (rf.API) # AVGED09 is by far the most important

## %IncMSE IncNodePurity  
## AVGED09 63.67 1205176  
## MEALS09 24.49 526249  
## EL09 17.99 164829

varImpPlot(rf.API)

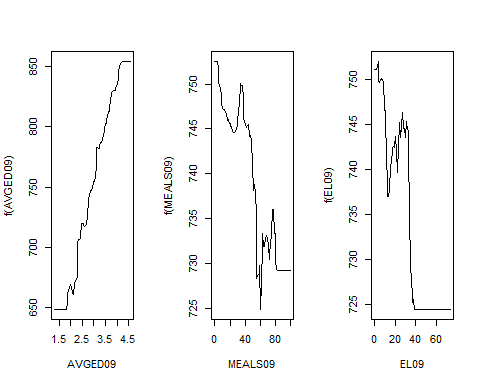


#Boosting  
  
library(gbm)  
  
set.seed(2014)  
boost.API = gbm (API09 ~ AVGED09 + MEALS09 + EL09, data = APIBase2009H[train, ],   
 distribution = "gaussian", n.trees = 5000, interaction.depth = 4)  
  
summary(boost.API)



## var rel.inf  
## AVGED09 AVGED09 82.561  
## MEALS09 MEALS09 9.641  
## EL09 EL09 7.798

# AVGED09 is by far the most important predictor  
par(mfrow = c(1, 3))  
plot(boost.API, i = "AVGED09")  
plot(boost.API, i= "MEALS09")  
plot(boost.API, i= "EL09")



par(mfrow = c(1, 1))  
  
yhat.boost = predict(boost.API, newdata = APIBase2009H[-train, ],  
 n.trees =5000)  
mean((yhat.boost - API3.test )^2) # 1607.268

## [1] 1607

sqrt(1607.268) # 40.09075

## [1] 40.09

#Another reduction in MSE

########################  
#Chapter 6 #  
#Summary of continuous #  
#model results #  
########################  
  
Summary of Results  
647 observations  
Trained and tested on all data  
Not all results displayed here were computed in this document.  
  
Method Object Residual SE Adjusted R-squared  
Regression 0 predictors HS09.lm0 77.94  
Regression 1 predictor HS09.lm1AVGED 37.5 0.7686 (AVGED09)  
Regression 1 predictor HS09.lm1Meals 47.71 0.6253 (MEALS09)  
Regression 1 predictor HS09.lm1EL 54.86 0.5047 (EL09)  
Regression 2 predictors HS09.lm2AVGEDMeals 37.42 0.7695 (AVGED09, MEALS09)  
Regression 2 predictors HS09.lm2AVGEDEL 37.2 0.7722 (AVGED09, EL09)  
Regression 2 predictors HS09.lm2ELMeals 46.02 0.6513 (MEALS09, EL09)  
Regression 3 predictors HS09.lm3AVGEDMealsEL 37.21 0.7721 (AVGED09, MEALS09, EL09)  
Regression 1 predictor gl 34.68 0.7862 (AVGED09 -- delete high Cook)  
Regression 2 predictors lm.interact 37.19 0.7724 (AVGED09, EL09 -- interaction)  
Regression 2 predictors lm.fitPoly2 37.53 0.7682 (AVGED09, AVGED09^2 )  
Regression 3 predictors lm.fitPoly3 37.55 0.7679 (AVGED09, AVGED09^2, AVGED09^2 )  
###  
Summary of Results  
320 observations  
Test data only  
  
Method Object Residual SE Adjusted R-squared  
See chapters 3, 4, 5  
  
#######  
#######  
  
Predictions for AHS  
Method Object Prediction  
  
Regression 0 predictors HS09.lm0 743  
Regression 1 predictor HS09.lm1AVGED 869 (AVGED09)  
Regression 1 predictor HS09.lm1Meals 814 (MEALS09)  
Regression 1 predictor HS09.lm1EL 752 (EL09)  
Regression 2 predictors HS09.lm2AVGEDMeals 866 (AVGED09, MEALS09)  
Regression 2 predictors HS09.lm2AVGEDEL 860 (AVGED09, EL09)  
Regression 2 predictors HS09.lm2ELMeals 799 (MEALS09, EL09)  
Regression 3 predictors HS09.lm3AVGEDMealsEL 858 (AVGED09, MEALS09, EL09)  
Regression 1 predictor HSSCI09.lm 859 (SCI)  
Regression 4 predictors HS09.lmDetail2 869 (HSG09, SOME\_COL09, COL\_GRAD09, GRAD\_SCH09)  
Regression 1 predictor gl 866 (AVGED09 -- delete high Cook)  
Regression 2 predictors lm.interact 856 (AVGED09, EL09 -- interaction)  
Regression 2 predictors lm.fitPoly2 869 (AVGED09, AVGED09^2 )  
Regression 3 predictors lm.fitPoly3 869 (AVGED09, AVGED09^2, AVGED09^2 )

####################################  
####################################  
####################################  
####################################

#########################  
#Part 2 #  
#Classification Methods #  
#########################

\*Part 2 Analysis of categorical data\*\*

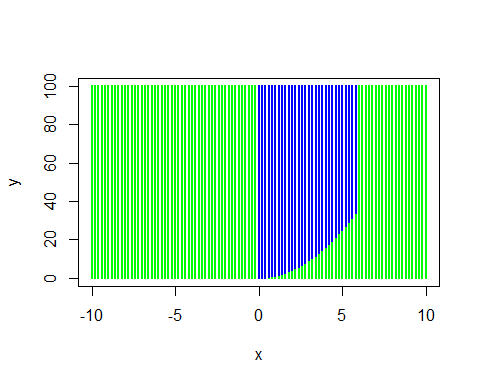
After doing the continuous analysis, I concluded that the data was too nice, so I created two simulation sets, one with binary data and the other with five categories. For each set of data I do a basic preliminary analysis, mainly to get the techniques going. Then I use resampling to measure the quality of the data. I need a training set and a test set to protect against over-fitting. Sometimes a technique can too closely approximate the idiosyncrasies of the given data and so work poorly when used to predict new data.

**Chapter 7. Data creation**

In this chapter I will create via simulation a data frame and then plot it. I will have two categories that I will then use in the next chapter. I will then sub-divide the data into five categories and expand the number of statistical learning categories that are attempted to analyze the data.

1. Create a data frame with two categories
2. Graph the 2 category data
3. Create a data frame with two categories
4. Graph the 5 category data ```

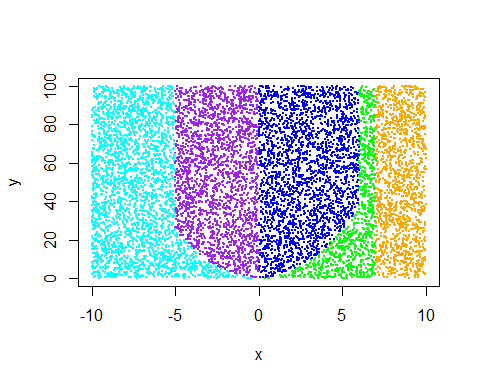
# First create a data frame   
# Color the input region   
x = seq(from = -10, to = 10, by = 0.2)  
y = seq(from = 0, to = 100, by = 0.2)  
df = expand.grid(x,y)  
names(df) = c('x', 'y')  
  
#Two colors  
with(df,plot(x, y,   
 col = ifelse(x >= 0 & x < 6 & y > x^2, "blue",   
 ifelse(x >= 0 & x <= 10 & y >= 0 & y <= 100, "green", "green")), pch = 19, cex = 0.2))



# By assigning colors/labels, create the data frame that will be used in the two category example  
  
labelsFunc2 = function(x,y) {ifelse(x >= 0 & x < 6 & y > x^2, "blue",   
 ifelse(x >= 0 & x <= 10 & y >= 0 & y <= 100, "green", "green"))   
}  
  
set.seed(2014)  
x = runif(10000, min = -10, max = 10)  
y = runif(10000, min = 0, max = 100)  
colors2 = labelsFunc2(x,y)  
df2 = data.frame(x = x, y = y, colors2 = colors2)  
head(df2)

## x y colors2  
## 1 -4.2839 78.86 green  
## 2 -6.6218 63.65 green  
## 3 2.5182 99.69 blue  
## 4 -3.8063 76.14 green  
## 5 0.9969 45.65 blue  
## 6 -8.3034 99.54 green

# 5 colors  
with(df2,plot(x, y,   
 col = ifelse(x >= 0 & x < 6 & y > x^2, "blue",   
 ifelse(x >= 0 & x < 7, "green",  
 ifelse(x >= 7 & x <= 10, "orange",   
 ifelse(x >= -5 & y > x^2, "purple", "cyan")))), pch = 19, cex = 0.2))



labelsFunc5 = function(x,y) {ifelse(x >= 0 & x < 6 & y > x^2, "blue",   
 ifelse(x >= 0 & x < 7, "green",   
 ifelse(x >= 7 & x <= 10, "orange",   
 ifelse(x < 0 & x >= -5 & y > x^2, "purple","cyan"))))}  
  
colors5 = labelsFunc5(x,y)  
df5 = data.frame(x = x, y = y, colors5 = colors5)   
head(df5, n = 12)

## x y colors5  
## 1 -4.2839 78.861 purple  
## 2 -6.6218 63.648 cyan  
## 3 2.5182 99.686 blue  
## 4 -3.8063 76.137 purple  
## 5 0.9969 45.653 blue  
## 6 -8.3034 99.541 cyan  
## 7 8.2402 37.460 orange  
## 8 2.0208 68.220 blue  
## 9 -8.0216 74.309 cyan  
## 10 -6.8828 6.329 cyan  
## 11 2.5297 69.326 blue  
## 12 -8.9470 93.114 cyan

**Chapter 8. Two category case**

In this chapter I will first try out some techniques on all of the two category data. Then I will divide the data into a training set and a test set and use resampling to further analyze the data. The decision boundaries are not linear and within each category the variables are not even close to being normal, so I expect the first 3 methods to do poorly (but QDA has more flexibility so will do better than the first two methods). I expect method 4 to win.

The basic binary tree is a greedy algorithm that is a top down approach that does recursive binary splitting. Look at the graphs below to obtain a pictorial representation.

1. Logistic regression
2. Linear discriminant analysis (LDA)
3. Quadratic discriminant analysis (QDA)
4. Basic binary tree

**Logistic regression 2 categories**

glm.logistic = glm(colors2 ~ x + y, data = df2, family = binomial)  
summary(glm.logistic)

##   
## Call:  
## glm(formula = colors2 ~ x + y, family = binomial, data = df2)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -1.880 -1.262 0.543 0.772 1.397   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) 1.598746 0.051238 31.2 <2e-16 \*\*\*  
## x -0.127247 0.004469 -28.5 <2e-16 \*\*\*  
## y -0.008644 0.000831 -10.4 <2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 11476 on 9999 degrees of freedom  
## Residual deviance: 10465 on 9997 degrees of freedom  
## AIC: 10471  
##   
## Number of Fisher Scoring iterations: 4

# Check the R coding  
contrasts(df2$colors2) # blue is coded as 0 and green is coded as 1

## green  
## blue 0  
## green 1

# Assign the probabilities  
glm.probs = predict (glm.logistic, type = "response")  
glm.probs[1:10]

## 1 2 3 4 5 6 7 8 9 10   
## 0.8119 0.8689 0.6027 0.8061 0.7460 0.8575 0.5564 0.6796 0.8784 0.9183

# Translate the vector of probabilities to a vector of 'S' and 'U'  
numRows = length(df2$colors2)  
glm.pred = rep ("blue", numRows)  
glm.pred[glm.probs > .5]= "green"  
  
#Create a confusion matrix to see how well the logistic regression model did  
table(glm.pred, df2$colors2)

##   
## glm.pred blue green  
## blue 1 619  
## green 2606 6774

mean(glm.pred != df2$colors2) # 0.3225

## [1] 0.3225

**LDA 2 categories**

library(MASS)  
lda.fit2 = lda(colors2 ~ x + y, data = df2)  
lda.fit2

## Call:  
## lda(colors2 ~ x + y, data = df2)  
##   
## Prior probabilities of groups:  
## blue green   
## 0.2607 0.7393   
##   
## Group means:  
## x y  
## blue 2.754 54.95  
## green -1.120 48.40  
##   
## Coefficients of linear discriminants:  
## LD1  
## x -0.17288  
## y -0.01149

lda.pred2 = predict(lda.fit2)  
lda.class2 = lda.pred2$class  
table(lda.class2, df2$colors2)

##   
## lda.class2 blue green  
## blue 1 656  
## green 2606 6737

mean(lda.class2 != df2$colors2) # 0.3262

## [1] 0.3262

**QDA 2 categories**

qda.fit2 = qda(colors2 ~ x + y, data = df2)  
qda.fit2

## Call:  
## qda(colors2 ~ x + y, data = df2)  
##   
## Prior probabilities of groups:  
## blue green   
## 0.2607 0.7393   
##   
## Group means:  
## x y  
## blue 2.754 54.95  
## green -1.120 48.40

qda.pred2 = predict(qda.fit2)  
qda.class2 = qda.pred2$class  
table(qda.class2, df2$colors2)

##   
## qda.class2 blue green  
## blue 1519 0  
## green 1088 7393

mean(qda.class2 != df2$colors2) # 0.1088

## [1] 0.1088

library(tree)  
tree.colors2 = tree(colors2 ~ x + y, df2)  
tree.summary2 = summary(tree.colors2)  
tree.summary2

##   
## Classification tree:  
## tree(formula = colors2 ~ x + y, data = df2)  
## Number of terminal nodes: 6   
## Residual mean deviance: 0.0641 = 641 / 9990   
## Misclassification error rate: 0.0126 = 126 / 10000

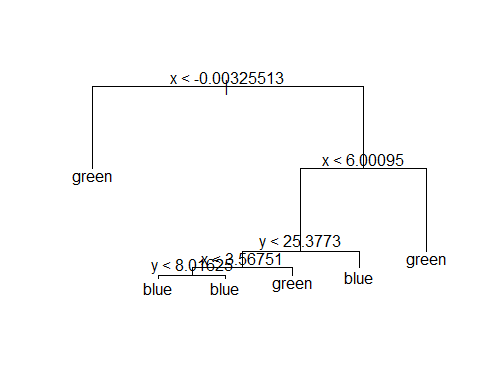
tree.summary2$misclass/length(colors2) # 0.0126

## [1] 0.0126 1.0000

names(tree.summary2)

## [1] "call" "type" "size" "df" "dev" "misclass"

plot(tree.colors2)  
text(tree.colors2, pretty = 0)



tree.colors2

## node), split, n, deviance, yval, (yprob)  
## \* denotes terminal node  
##   
## 1) root 10000 10000 green ( 0.261 0.739 )   
## 2) x < -0.00325513 5064 0 green ( 0.000 1.000 ) \*  
## 3) x > -0.00325513 4936 7000 blue ( 0.528 0.472 )   
## 6) x < 6.00095 2964 2000 blue ( 0.880 0.120 )   
## 12) y < 25.3773 781 1000 blue ( 0.561 0.439 )   
## 24) x < 3.56751 474 400 blue ( 0.844 0.156 )   
## 48) y < 8.01625 150 200 blue ( 0.527 0.473 ) \*  
## 49) y > 8.01625 324 30 blue ( 0.991 0.009 ) \*  
## 25) x > 3.56751 307 200 green ( 0.124 0.876 ) \*  
## 13) y > 25.3773 2183 200 blue ( 0.994 0.006 ) \*  
## 7) x > 6.00095 1972 0 green ( 0.000 1.000 ) \*

#Now use the validation set approach to test the methodologies  
set.seed(2014)  
  
train = sample(x = 1:numRows, size = numRows/2, replace = FALSE) # Index for training  
  
train.df2 = df2[train,]  
test.df2 = df2[-train,]  
head(train.df2)

## x y colors2  
## 2859 -6.4062 54.16 green  
## 1689 3.0838 34.02 blue  
## 6258 -6.1622 96.66 green  
## 3096 1.5951 32.55 blue  
## 5497 0.9535 94.96 blue  
## 848 9.7146 34.37 green

dim(train.df2)

## [1] 5000 3

dim(test.df2)

## [1] 5000 3

**Validation approach for logistic regression**

glm.fitLogisTrain2 = glm(colors2 ~ x+y, data = train.df2, family = binomial)  
glm.probsLogisTrain2 = predict.glm(glm.fitLogisTrain2, newdata = test.df2, type = "response")  
  
glm.predLogis2 = rep ("blue", numRows/2)  
glm.predLogis2[glm.probsLogisTrain2 > .5]= "green"  
table(glm.predLogis2, test.df2$colors2)

##   
## glm.predLogis2 blue green  
## blue 0 316  
## green 1290 3394

mean(glm.predLogis2 != test.df2$colors2) # perc errors 0.3234

## [1] 0.3212

percGreen = (282 + 3383)/(282 + 3383 + 1335)  
1 - percGreen # 0.267

## [1] 0.267

#The percent correct is 92.7%  
# Simply predicting green yields an error rate of 26.7%, so logistic regression did not fair very well.

**Validation approach for LDA**

lda.fit2 = lda(colors2 ~ x+y, data = train.df2)  
lda.fit2

## Call:  
## lda(colors2 ~ x + y, data = train.df2)  
##   
## Prior probabilities of groups:  
## blue green   
## 0.2634 0.7366   
##   
## Group means:  
## x y  
## blue 2.755 55.04  
## green -1.153 49.17  
##   
## Coefficients of linear discriminants:  
## LD1  
## x -0.17452  
## y -0.01081

lda.pred2 = predict(lda.fit2, newdata = test.df2)  
  
lda.class2 = lda.pred2$class  
table(lda.class2, test.df2$colors2)

##   
## lda.class2 blue green  
## blue 0 344  
## green 1290 3366

mean(lda.class2 != test.df2$colors2) # perc errors 0.3296

## [1] 0.3268

**Validation approach for QDA**

qda.fit2 = qda(colors2 ~ x+y, data = train.df2)  
qda.fit2

## Call:  
## qda(colors2 ~ x + y, data = train.df2)  
##   
## Prior probabilities of groups:  
## blue green   
## 0.2634 0.7366   
##   
## Group means:  
## x y  
## blue 2.755 55.04  
## green -1.153 49.17

qda.class2 = predict(qda.fit2, newdata = test.df2)$class  
table(qda.class2, test.df2$colors2)

##   
## qda.class2 blue green  
## blue 801 0  
## green 489 3710

mean(qda.class2 != test.df2$colors2) # perc errors 0.1164

## [1] 0.0978

**Validation approach for classification tree**

df2.test= df2[-train, ]  
tree.df2 = tree(colors2 ~ x+y, data = train.df2, subset = train)  
tree.pred2 = predict(tree.df2, df2.test, type = "class")  
  
table(tree.pred2, df2.test$colors2)

##   
## tree.pred2 blue green  
## blue 1250 17  
## green 40 3693

mean(tree.pred2 != test.df2$colors2) # perc errors 0.0162

## [1] 0.0114

# The linear methods used to analyze non-linear data did very poorly. QDA did not do as badly but it did not do very since the assumptions underlying QDA were not met. So the basic tree won in the two smaple case.

**Chapter 9. Five category case**

In this chapter I explore more techniques.

Bagging repeatedly creates binary trees via bootstrap. Typically about 2/3 of the data points will be chosen on each iteration, so the remaining 1/3 can be used as a test set and can be used for error estimates.

Random forest is quite similar to bagging but only takes a random sample of possible variables that can produce the next split.

SVM Various kernels explored.

KNN looks at nearby points

1. LDA
2. QDA
3. Classification tree
4. Bagging
5. Random forests
6. Support Vector Machines
7. KNN

**LDA 5 categries**

lda.fit5 = lda(colors5 ~ x + y, data = df5)  
lda.fit5

## Call:  
## lda(colors5 ~ x + y, data = df5)  
##   
## Prior probabilities of groups:  
## blue cyan green orange purple   
## 0.2607 0.2762 0.0895 0.1434 0.2302   
##   
## Group means:  
## x y  
## blue 2.754 54.95  
## cyan -7.245 47.01  
## green 5.679 34.65  
## orange 8.490 50.09  
## purple -2.402 54.37  
##   
## Coefficients of linear discriminants:  
## LD1 LD2  
## x -6.557e-01 0.0006304  
## y -6.153e-05 0.0348690  
##   
## Proportion of trace:  
## LD1 LD2   
## 0.9969 0.0031

lda.pred5 = predict(lda.fit5)  
lda.class5 = lda.pred5$class  
table(lda.class5, df5$colors5)

##   
## lda.class5 blue cyan green orange purple  
## blue 2356 0 144 0 0  
## cyan 0 2594 0 0 89  
## green 198 0 514 19 0  
## orange 2 0 237 1415 0  
## purple 51 168 0 0 2213

mean(lda.class5 != df5$colors5) # 0.0908

## [1] 0.0908

**QDA 5 categories**

qda.fit5 = qda(colors5 ~ x + y, data = df5)  
qda.fit5

## Call:  
## qda(colors5 ~ x + y, data = df5)  
##   
## Prior probabilities of groups:  
## blue cyan green orange purple   
## 0.2607 0.2762 0.0895 0.1434 0.2302   
##   
## Group means:  
## x y  
## blue 2.754 54.95  
## cyan -7.245 47.01  
## green 5.679 34.65  
## orange 8.490 50.09  
## purple -2.402 54.37

qda.pred5 = predict(qda.fit5)  
qda.class5 = qda.pred5$class  
table(qda.class5, df5$colors5)

##   
## qda.class5 blue cyan green orange purple  
## blue 2412 0 113 0 14  
## cyan 0 2647 0 0 101  
## green 182 0 777 27 0  
## orange 0 0 5 1407 0  
## purple 13 115 0 0 2187

mean(qda.class5 != df5$colors5) # 0.057

## [1] 0.057

**Basic classification tree 5 categories**

tree.colors5 = tree(colors5 ~ x + y, df5)  
tree.summary5 = summary(tree.colors5)  
tree.summary5

##   
## Classification tree:  
## tree(formula = colors5 ~ x + y, data = df5)  
## Number of terminal nodes: 8   
## Residual mean deviance: 0.159 = 1590 / 9990   
## Misclassification error rate: 0.0343 = 343 / 10000

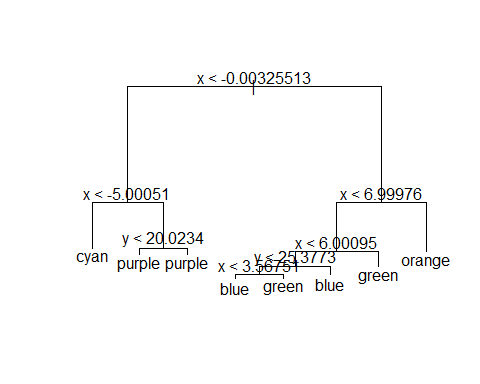
tree.summary5$misclass/length(colors5) # 0.0343

## [1] 0.0343 1.0000

names(tree.summary5)

## [1] "call" "type" "size" "df" "dev" "misclass"

plot(tree.colors5)  
text(tree.colors5, pretty = 0)



tree.colors5

## node), split, n, deviance, yval, (yprob)  
## \* denotes terminal node  
##   
## 1) root 10000 30000 cyan ( 0.261 0.276 0.089 0.143 0.230 )   
## 2) x < -0.00325513 5064 7000 cyan ( 0.000 0.545 0.000 0.000 0.455 )   
## 4) x < -5.00051 2545 0 cyan ( 0.000 1.000 0.000 0.000 0.000 ) \*  
## 5) x > -5.00051 2519 1000 purple ( 0.000 0.086 0.000 0.000 0.914 )   
## 10) y < 20.0234 524 700 purple ( 0.000 0.405 0.000 0.000 0.595 ) \*  
## 11) y > 20.0234 1995 70 purple ( 0.000 0.003 0.000 0.000 0.997 ) \*  
## 3) x > -0.00325513 4936 10000 blue ( 0.528 0.000 0.181 0.291 0.000 )   
## 6) x < 6.99976 3502 4000 blue ( 0.744 0.000 0.256 0.000 0.000 )   
## 12) x < 6.00095 2964 2000 blue ( 0.880 0.000 0.120 0.000 0.000 )   
## 24) y < 25.3773 781 1000 blue ( 0.561 0.000 0.439 0.000 0.000 )   
## 48) x < 3.56751 474 400 blue ( 0.844 0.000 0.156 0.000 0.000 ) \*  
## 49) x > 3.56751 307 200 green ( 0.124 0.000 0.876 0.000 0.000 ) \*  
## 25) y > 25.3773 2183 200 blue ( 0.994 0.000 0.006 0.000 0.000 ) \*  
## 13) x > 6.00095 538 0 green ( 0.000 0.000 1.000 0.000 0.000 ) \*  
## 7) x > 6.99976 1434 0 orange ( 0.000 0.000 0.000 1.000 0.000 ) \*

**Now use the validation set approach to test the methodologies**

train.df5 = df5[train,]  
test.df5 = df5[-train,]  
head(train.df5, n = 12)

## x y colors5  
## 2859 -6.4062 54.157 cyan  
## 1689 3.0838 34.017 blue  
## 6258 -6.1622 96.665 cyan  
## 3096 1.5951 32.548 blue  
## 5497 0.9535 94.962 blue  
## 848 9.7146 34.372 orange  
## 9115 -9.2564 34.886 cyan  
## 6007 -6.3933 82.895 cyan  
## 989 -2.0864 85.547 purple  
## 1558 -1.2485 9.259 purple  
## 6259 -7.2311 4.526 cyan  
## 526 -6.1972 12.184 cyan

dim(train.df5)

## [1] 5000 3

dim(test.df5)

## [1] 5000 3

**Validation approach for LDA**

lda.fit5 = lda(colors5 ~ x+y, data = train.df5)  
lda.fit5

## Call:  
## lda(colors5 ~ x + y, data = train.df5)  
##   
## Prior probabilities of groups:  
## blue cyan green orange purple   
## 0.2634 0.2784 0.0886 0.1424 0.2272   
##   
## Group means:  
## x y  
## blue 2.755 55.04  
## cyan -7.261 48.51  
## green 5.653 34.29  
## orange 8.520 50.61  
## purple -2.385 54.87  
##   
## Coefficients of linear discriminants:  
## LD1 LD2  
## x -0.6514575 0.003359  
## y -0.0001298 0.034739  
##   
## Proportion of trace:  
## LD1 LD2   
## 0.9969 0.0031

lda.pred5 = predict(lda.fit5, newdata = test.df5)  
  
lda.class5 = lda.pred5$class  
table(lda.class5, test.df5$colors5)

##   
## lda.class5 blue cyan green orange purple  
## blue 1192 0 77 0 0  
## cyan 0 1286 0 0 51  
## green 78 0 253 13 0  
## orange 0 0 122 709 0  
## purple 20 84 0 0 1115

mean(lda.class5 != test.df5$colors5) # perc errors 0.097

## [1] 0.089

**Validation approach for QDA**

qda.fit5 = qda(colors5 ~ x+y, data = train.df5)  
qda.fit5

## Call:  
## qda(colors5 ~ x + y, data = train.df5)  
##   
## Prior probabilities of groups:  
## blue cyan green orange purple   
## 0.2634 0.2784 0.0886 0.1424 0.2272   
##   
## Group means:  
## x y  
## blue 2.755 55.04  
## cyan -7.261 48.51  
## green 5.653 34.29  
## orange 8.520 50.61  
## purple -2.385 54.87

qda.class5 = predict(qda.fit5, newdata = test.df5)$class  
table(qda.class5, test.df5$colors5)

##   
## qda.class5 blue cyan green orange purple  
## blue 1208 0 60 0 2  
## cyan 0 1309 0 0 50  
## green 79 0 388 20 0  
## orange 0 0 4 702 0  
## purple 3 61 0 0 1114

mean(qda.class5 != test.df5$colors5) # perc errors 0.0606

## [1] 0.0558

**Validation approach for classification tree**

df5.test= df5[-train, ]  
tree.df5 = tree(colors5 ~ x+y, data = train.df5, subset = train)  
tree.pred5 = predict(tree.df5, df5.test, type = "class")  
  
table(tree.pred5, df5.test$colors5)

##   
## tree.pred5 blue cyan green orange purple  
## blue 1287 0 80 0 0  
## cyan 0 1269 0 0 7  
## green 3 0 371 0 0  
## orange 0 0 1 722 0  
## purple 0 101 0 0 1159

mean(tree.pred5 != test.df5$colors5) # perc errors 0.0276

## [1] 0.0384

**Bagging**

library(randomForest)  
set.seed (2014)  
bag.df5 = randomForest(colors5 ~ x + y, data = df5, subset = train, mtry = 2, importance = TRUE)  
bag.df5

##   
## Call:  
## randomForest(formula = colors5 ~ x + y, data = df5, mtry = 2, importance = TRUE, subset = train)   
## Type of random forest: classification  
## Number of trees: 500  
## No. of variables tried at each split: 2  
##   
## OOB estimate of error rate: 0.58%  
## Confusion matrix:  
## blue cyan green orange purple class.error  
## blue 1311 0 6 0 0 0.004556  
## cyan 0 1384 0 0 8 0.005747  
## green 9 0 434 0 0 0.020316  
## orange 0 0 0 712 0 0.000000  
## purple 0 6 0 0 1130 0.005282

names(bag.df5)

## [1] "call" "type" "predicted"   
## [4] "err.rate" "confusion" "votes"   
## [7] "oob.times" "classes" "importance"   
## [10] "importanceSD" "localImportance" "proximity"   
## [13] "ntree" "mtry" "forest"   
## [16] "y" "test" "inbag"   
## [19] "terms"

errRate = 1 - (1266 + 1419 + 432 + 712 + 1142)/5000   
errRate # 0.0058

## [1] 0.0058

# Now on the test sets  
yhat.bag = predict(bag.df5, newdata = df5[-train, ])  
mean(yhat.bag != df5.test$colors5) # 0.0074

## [1] 0.0036

**Random forests (change mtry to 1)**

set.seed(2014)  
rf.df5 = randomForest(colors5 ~ x + y, data = df5, subset = train, mtry = 1, importance = TRUE)  
rf.df5

##   
## Call:  
## randomForest(formula = colors5 ~ x + y, data = df5, mtry = 1, importance = TRUE, subset = train)   
## Type of random forest: classification  
## Number of trees: 500  
## No. of variables tried at each split: 1  
##   
## OOB estimate of error rate: 0.42%  
## Confusion matrix:  
## blue cyan green orange purple class.error  
## blue 1312 0 5 0 0 0.003797  
## cyan 0 1388 0 0 4 0.002874  
## green 6 0 437 0 0 0.013544  
## orange 0 0 1 711 0 0.001404  
## purple 0 5 0 0 1131 0.004401

yhat.rf = predict(rf.df5 , newdata = df5[- train, ]) #   
mean(yhat.rf != df5.test$colors5) # 0.0064, a slight improvement over bagging

## [1] 0.004

**Support Vector Machines (SVM)**

library(e1071)  
  
# Linear kernel   
dat = df5[train, ]  
out = svm(colors5 ~ x + y, data = dat, kernel = "linear", cost = 10)  
summary(out)

##   
## Call:  
## svm(formula = colors5 ~ x + y, data = dat, kernel = "linear",   
## cost = 10)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: linear   
## cost: 10   
## gamma: 0.5   
##   
## Number of Support Vectors: 985  
##   
## ( 181 262 58 241 243 )  
##   
##   
## Number of Classes: 5   
##   
## Levels:   
## blue cyan green orange purple

head(dat)

## x y colors5  
## 2859 -6.4062 54.16 cyan  
## 1689 3.0838 34.02 blue  
## 6258 -6.1622 96.66 cyan  
## 3096 1.5951 32.55 blue  
## 5497 0.9535 94.96 blue  
## 848 9.7146 34.37 orange

table(out$fitted , dat$colors5) # 308 training errors

##   
## blue cyan green orange purple  
## blue 1234 0 92 0 6  
## cyan 0 1319 0 0 58  
## green 76 0 351 5 0  
## orange 0 0 0 707 0  
## purple 7 73 0 0 1072

mean(out$fitted != dat$colors5) # 0.0616

## [1] 0.0634

# Now on the test set  
dat.test = df5[-train, ]  
pred.test = predict(out, newdata = dat.test)  
table(pred.test, dat.test$colors5) # 305 test errors

##   
## pred.test blue cyan green orange purple  
## blue 1221 0 95 0 5  
## cyan 0 1297 0 0 53  
## green 64 0 356 8 0  
## orange 0 0 1 714 0  
## purple 5 73 0 0 1108

mean(pred.test != dat.test$colors5) # 0.061

## [1] 0.0608

tune.out = tune(svm, colors5 ~ ., data = dat, kernel = "linear",  
 ranges = list( cost=c(0.001, 0.01, 0.1, 1, 5 , 10 , 100)))  
summary(tune.out)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost  
## 100  
##   
## - best performance: 0.06158   
##   
## - Detailed performance results:  
## cost error dispersion  
## 1 1e-03 0.44994 0.02665  
## 2 1e-02 0.11979 0.01386  
## 3 1e-01 0.06498 0.01633  
## 4 1e+00 0.06538 0.01286  
## 5 5e+00 0.06418 0.01320  
## 6 1e+01 0.06378 0.01323  
## 7 1e+02 0.06158 0.01281

# examine the best model  
bestmod = tune.out$best.model  
summary(bestmod)

##   
## Call:  
## best.tune(method = svm, train.x = colors5 ~ ., data = dat, ranges = list(cost = c(0.001,   
## 0.01, 0.1, 1, 5, 10, 100)), kernel = "linear")  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: linear   
## cost: 100   
## gamma: 0.5   
##   
## Number of Support Vectors: 856  
##   
## ( 179 225 27 206 219 )  
##   
##   
## Number of Classes: 5   
##   
## Levels:   
## blue cyan green orange purple

# Test the model using the best tuning parameter   
ypred = predict(bestmod, dat.test)  
table(predict = ypred, truth = dat.test$colors5)

## truth  
## predict blue cyan green orange purple  
## blue 1224 0 97 0 2  
## cyan 0 1296 0 0 51  
## green 64 0 355 7 0  
## orange 0 0 0 715 0  
## purple 2 74 0 0 1113

err.Rate = 302/5000  
err.Rate # 0.0604

## [1] 0.0604

# Polynomial kernel   
dat = df5[train, ]  
outP = svm(colors5 ~ x + y, data = dat, kernel = "polynomial", degree = 2, cost = 10)  
summary(outP)

##   
## Call:  
## svm(formula = colors5 ~ x + y, data = dat, kernel = "polynomial",   
## degree = 2, cost = 10)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: polynomial   
## cost: 10   
## degree: 2   
## gamma: 0.5   
## coef.0: 0   
##   
## Number of Support Vectors: 4653  
##   
## ( 1168 1281 691 1070 443 )  
##   
##   
## Number of Classes: 5   
##   
## Levels:   
## blue cyan green orange purple

table(outP$fitted , dat$colors5)

##   
## blue cyan green orange purple  
## blue 414 177 0 0 239  
## cyan 134 908 395 378 35  
## green 0 0 0 0 0  
## orange 0 303 0 334 0  
## purple 769 4 48 0 862

mean(outP$fitted != dat$colors5)

## [1] 0.4964

# Now on the test set  
dat.test = df5[-train, ]  
pred.testP = predict(outP, newdata = dat.test)  
table(pred.testP, dat.test$colors5) # 305 test errors

##   
## pred.testP blue cyan green orange purple  
## blue 397 187 0 0 269  
## cyan 119 884 408 409 30  
## green 0 0 0 0 0  
## orange 0 297 0 313 0  
## purple 774 2 44 0 867

mean(pred.testP != dat.test$colors5) # 0.061

## [1] 0.5078

tune.outP = tune(svm, colors5 ~ ., data = dat, kernel = "polynomial", degree = 2,  
 ranges = list( cost=c(0.001, 0.01, 0.1, 1, 5 , 10 , 100)))  
summary(tune.outP)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost  
## 10  
##   
## - best performance: 0.4933   
##   
## - Detailed performance results:  
## cost error dispersion  
## 1 1e-03 0.6765 0.06909  
## 2 1e-02 0.5270 0.01823  
## 3 1e-01 0.4989 0.02366  
## 4 1e+00 0.4949 0.02366  
## 5 5e+00 0.4937 0.02542  
## 6 1e+01 0.4933 0.02508  
## 7 1e+02 0.4933 0.02483

# examine the best model  
bestmodP = tune.outP$best.model  
summary(bestmodP)

##   
## Call:  
## best.tune(method = svm, train.x = colors5 ~ ., data = dat, ranges = list(cost = c(0.001,   
## 0.01, 0.1, 1, 5, 10, 100)), kernel = "polynomial", degree = 2)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: polynomial   
## cost: 10   
## degree: 2   
## gamma: 0.5   
## coef.0: 0   
##   
## Number of Support Vectors: 4653  
##   
## ( 1168 1281 691 1070 443 )  
##   
##   
## Number of Classes: 5   
##   
## Levels:   
## blue cyan green orange purple

# Test the model using the best tuning parameter   
ypredP = predict(bestmodP, dat.test)  
table(predict = ypredP, truth = dat.test$colors5)

## truth  
## predict blue cyan green orange purple  
## blue 397 187 0 0 269  
## cyan 119 884 408 409 30  
## green 0 0 0 0 0  
## orange 0 297 0 313 0  
## purple 774 2 44 0 867

# This did very poorly

# radial kernel   
dat = df5[train, ]  
outR = svm(colors5 ~ x + y, data = dat, kernel = "radial", gamma = 1, cost = 10)  
summary(outR)

##   
## Call:  
## svm(formula = colors5 ~ x + y, data = dat, kernel = "radial",   
## gamma = 1, cost = 10)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: radial   
## cost: 10   
## gamma: 1   
##   
## Number of Support Vectors: 556  
##   
## ( 80 138 67 131 140 )  
##   
##   
## Number of Classes: 5   
##   
## Levels:   
## blue cyan green orange purple

table(outR$fitted , dat$colors5)

##   
## blue cyan green orange purple  
## blue 1310 0 7 0 6  
## cyan 0 1383 0 0 2  
## green 2 0 431 8 0  
## orange 0 0 5 704 0  
## purple 5 9 0 0 1128

mean(outR$fitted != dat$colors5) # 0.0088

## [1] 0.0088

# Now on the test set  
dat.test = df5[-train, ]  
pred.testR = predict(outR, newdata = dat.test)  
table(pred.testR, dat.test$colors5) # 55 test errors

##   
## pred.testR blue cyan green orange purple  
## blue 1284 0 9 0 3  
## cyan 0 1361 0 0 7  
## green 2 0 437 15 0  
## orange 0 0 6 707 0  
## purple 4 9 0 0 1156

mean(pred.testR != dat.test$colors5) # 0.011

## [1] 0.011

tune.outR = tune(svm, colors5 ~ ., data = dat, kernel = "radial", gamma = 1,  
 ranges = list( cost=c(0.001, 0.01, 0.1, 1, 5 , 10 , 100)))  
summary(tune.outR)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost  
## 100  
##   
## - best performance: 0.008807   
##   
## - Detailed performance results:  
## cost error dispersion  
## 1 1e-03 0.692598 0.020054  
## 2 1e-02 0.100834 0.017639  
## 3 1e-01 0.033999 0.008356  
## 4 1e+00 0.015599 0.005558  
## 5 5e+00 0.011998 0.004413  
## 6 1e+01 0.011801 0.004842  
## 7 1e+02 0.008807 0.004352

# examine the best model  
bestmodR = tune.outR$best.model  
summary(bestmodR)

##   
## Call:  
## best.tune(method = svm, train.x = colors5 ~ ., data = dat, ranges = list(cost = c(0.001,   
## 0.01, 0.1, 1, 5, 10, 100)), kernel = "radial", gamma = 1)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: radial   
## cost: 100   
## gamma: 1   
##   
## Number of Support Vectors: 286  
##   
## ( 43 70 37 68 68 )  
##   
##   
## Number of Classes: 5   
##   
## Levels:   
## blue cyan green orange purple

# Test the model using the best tuning parameter   
ypredR = predict(bestmodR, dat.test)  
table(predict = ypredR, truth = dat.test$colors5) # only 34 errors

## truth  
## predict blue cyan green orange purple  
## blue 1286 0 5 0 1  
## cyan 0 1367 0 0 9  
## green 2 0 447 12 0  
## orange 0 0 0 710 0  
## purple 2 3 0 0 1156

err.RateR = 34/5000 # 0.0068  
# This did quite well

**KNN**

library(class)  
train.X = df5[train, 1:2]  
test.X = df5[-train, 1:2]  
train.colors5 = df5[train, 3]  
  
# At first I fail to standardize the variables and this leads to terrible predictive accuracy  
set.seed(2014)  
knn.pred = knn(train.X, test.X, train.colors5, k = 1)  
table(knn.pred, train.colors5)

## train.colors5  
## knn.pred blue cyan green orange purple  
## blue 340 369 115 206 274  
## cyan 362 403 115 186 306  
## green 117 122 49 51 117  
## orange 187 175 69 117 158  
## purple 311 323 95 152 281

#   
#   
knn.pred = knn(train.X, test.X, train.colors5 ,k=3)  
table(knn.pred , train.colors5)

## train.colors5  
## knn.pred blue cyan green orange purple  
## blue 349 360 115 200 268  
## cyan 364 393 114 184 307  
## green 118 126 47 54 115  
## orange 183 176 71 118 160  
## purple 303 337 96 156 286

knn.pred = knn(train.X, test.X, train.colors5 ,k=5)  
table(knn.pred , train.colors5)

## train.colors5  
## knn.pred blue cyan green orange purple  
## blue 348 360 115 205 270  
## cyan 362 397 114 183 307  
## green 118 119 45 50 115  
## orange 184 182 71 121 161  
## purple 305 334 98 153 283

mean(knn.pred != train.colors5)

## [1] 0.7612

standardized.X = scale (df5[ , 1:2])  
var(df5[ , 1])

## [1] 33.34

var(df5[ , 2])

## [1] 856.5

var(standardized.X[ , 1])

## [1] 1

var( standardized.X[ , 2])

## [1] 1

train.X = standardized.X[train, ]  
test.X= standardized.X[-train, ]  
  
train.Y = df5[train, 3]  
test.Y = df5[-train, 3]  
  
set.seed (2014)  
knn.pred = knn(train.X, test.X, train.Y, k = 1)  
mean(test.Y != knn.pred) # 0.0156 This is much better!

## [1] 0.0156

set.seed (2014)  
knn.pred = knn(train.X, test.X, train.Y, k = 3)  
mean(test.Y != knn.pred)

## [1] 0.0162

set.seed (2014)  
knn.pred = knn(train.X, test.X, train.Y, k = 5)  
mean(test.Y != knn.pred)

## [1] 0.0196

**Chapter 10. Summary of categorical results**  
All the tables in one place  
The summaries are on test data  
i. Two categories  
ii. Five categories

**Two categories**

**Logistic regression**

table(glm.predLogis2, test.df2$colors2)

##   
## glm.predLogis2 blue green  
## blue 0 316  
## green 1290 3394

mean(glm.predLogis2 != test.df2$colors2)

## [1] 0.3212

**LDA**

table(lda.class2, test.df2$colors2)

##   
## lda.class2 blue green  
## blue 0 344  
## green 1290 3366

mean(lda.class2 != test.df2$colors2) # perc errors 0.3296

## [1] 0.3268

**QDA**

table(qda.class2, test.df2$colors2)

##   
## qda.class2 blue green  
## blue 801 0  
## green 489 3710

mean(qda.class2 != test.df2$colors2) # perc errors 0.1164

## [1] 0.0978

**Classification tree**

table(tree.pred2, df2.test$colors2)

##   
## tree.pred2 blue green  
## blue 1250 17  
## green 40 3693

mean(tree.pred2 != test.df2$colors2)

## [1] 0.0114

**Summary**  
Techniques and error rates

|  |  |  |
| --- | --- | --- |
| Technique | All Data | Validation Set |
| Logistic Regression | 0.3225 | 0.3234 |
| LDA | 0.3262 | 0.3296 |
| QDA | 0.1088 | 0.1164 |
| Tree | 0.0126 | 0.0162 |

Going into the 2 category analysis, I knew that logistic regression and LDA were not appropriate. I also knew that QDA was not appropriate but more flexible, so it did better but not good enough. The basic tree did far better.

**Five categories**

**LDA**

table(lda.class5, test.df5$colors5)

##   
## lda.class5 blue cyan green orange purple  
## blue 1192 0 77 0 0  
## cyan 0 1286 0 0 51  
## green 78 0 253 13 0  
## orange 0 0 122 709 0  
## purple 20 84 0 0 1115

mean(lda.class5 != test.df5$colors5) # perc errors 0.097

## [1] 0.089

**QDA**

table(qda.class5, test.df5$colors5)

##   
## qda.class5 blue cyan green orange purple  
## blue 1208 0 60 0 2  
## cyan 0 1309 0 0 50  
## green 79 0 388 20 0  
## orange 0 0 4 702 0  
## purple 3 61 0 0 1114

mean(qda.class5 != test.df5$colors5) # perc errors 0.0606

## [1] 0.0558

**Basic classification tree 5 categories**

table(tree.pred5, df5.test$colors5)

##   
## tree.pred5 blue cyan green orange purple  
## blue 1287 0 80 0 0  
## cyan 0 1269 0 0 7  
## green 3 0 371 0 0  
## orange 0 0 1 722 0  
## purple 0 101 0 0 1159

mean(tree.pred5 != test.df5$colors5) # perc errors 0.0276

## [1] 0.0384

**Bagging**

yhat.bag = predict(bag.df5, newdata = df5[-train, ])  
mean(yhat.bag != df5.test$colors5) # 0.0074

## [1] 0.0036

**Random forests (change mtry to 1)**

yhat.rf = predict(rf.df5 , newdata = df5[- train, ]) #   
mean(yhat.rf != df5.test$colors5) # 0.0064, a slight improvement over bagging

## [1] 0.004

**Support Vector Machines (SVM, linear)**

table(predict = ypred, truth = dat.test$colors5)

## truth  
## predict blue cyan green orange purple  
## blue 1224 0 97 0 2  
## cyan 0 1296 0 0 51  
## green 64 0 355 7 0  
## orange 0 0 0 715 0  
## purple 2 74 0 0 1113

err.Rate = 302/5000  
err.Rate # 0.0604

## [1] 0.0604

**Support Vector Machines (SVM, radial)**  
table(predict = ypredR, truth = dat.test$colors5) # only 34 errors err.RateR = 34/5000 # 0.0068 ```

**KNN**

set.seed (2014)  
knn.pred1 = knn(train.X, test.X, train.Y, k = 1)  
mean(test.Y != knn.pred1) # 0.0156

## [1] 0.0156

**Five categories with ranking** Technique | Validation Set  
------- | -------- |  
Bagging | 0.004 RF | 0.0044 SVM (radial) | 0.0068 KNN | 0.0176 Tree | 0.0392 QDA | 0.0594 SVM (linear) | 0.0604 LDA | 0.1004

###########################   
# Appendix 1 #   
# Reading, cleaning and #   
# merging of data #   
###########################   
  
####################################################   
# The predictor variables and the outcome variable #   
####################################################   
# http://www.cde.ca.gov/ta/ac/ap/reclayout02b.asp   
# The DBF column headers change slightly from year to year   
# CDS: County/District/School code   
# STYPE: (E=Elem, M=Middle, H=High)   
# SNAME: School Name   
# DNAME: District Name   
# CNAME: County Name   
# FLAG: Codes for irregularities (all 2002 flagged schools  
# were eliminated from the analysis  
# VALID: Number of Students Included in the 2002 API   
  
# API02: Tested 2002 API (Base) 200 <= API02 <= 1000   
# API02 through API09 are the outcome variables  
  
# AVG\_ED: Average Parent Education Level (most important  
# predictor  
# AVG\_ED based on:  
# 1 = Not a high school graduate   
# 2 = High school graduate  
# 3 = Some college  
# 4 = College graduate  
# 5 = Graduate school   
  
# NOT\_HSG: Percent of parents with less than HS education  
# HSG: Percent of parents who are HS grad  
# SOME\_COL: Percent of parents who have some college education  
# COL\_GRAD: Percent of parents who have a college degree  
# GRAD\_SCH: Percent of parents who have a graduate degree  
  
#  
# MEALS: Percentage of Students Tested that are Participants  
# in the Free or Reduced Price Lunch Program. Second predictor   
# EL: Percent English Learners. Third predictor.  
# SCI: School Characteristic Index (CDE's composite predictor)  
# CDE combines the predictors I am using and others into the single predictor, SCI  
  
  
  
\*\*2002\*\*

AHSCDS = "01611270130450" # Albany HS ID number  
outputFile = "API output, Version 1.xlsx" # Later some data frames will be sent to this file  
  
library(foreign) # needed for read.dbf which reads in the database files  
library(sqldf) # needed for SQL code  
setwd("C:/Users/Ira/Documents/Statistics/My Website/Project 1 -- API/API Linear Models 2011/Database files")  
APIBase2002 = read.dbf("api02bdb.dbf",as.is = TRUE)  
dim(APIBase2002) # 8733 82

## [1] 8733 82

names(APIBase2002)

## [1] "CDS" "STYPE" "SMALL" "SNAME" "DNAME"   
## [6] "CNAME" "FLAG" "VALID" "API02" "ST\_RANK"   
## [11] "SIM\_RANK" "GR\_TARG" "API\_TARG" "AA\_NUM" "AA\_SIG"   
## [16] "AA\_API" "AA\_GT" "AA\_TARG" "AI\_NUM" "AI\_SIG"   
## [21] "AI\_API" "AI\_GT" "AI\_TARG" "AS\_NUM" "AS\_SIG"   
## [26] "AS\_API" "AS\_GT" "AS\_TARG" "FI\_NUM" "FI\_SIG"   
## [31] "FI\_API" "FI\_GT" "FI\_TARG" "HI\_NUM" "HI\_SIG"   
## [36] "HI\_API" "HI\_GT" "HI\_TARG" "PI\_NUM" "PI\_SIG"   
## [41] "PI\_API" "PI\_GT" "PI\_TARG" "WH\_NUM" "WH\_SIG"   
## [46] "WH\_API" "WH\_GT" "WH\_TARG" "SD\_NUM" "SD\_SIG"   
## [51] "SD\_API" "SD\_GT" "SD\_TARG" "PCT\_AA" "PCT\_AI"   
## [56] "PCT\_AS" "PCT\_FI" "PCT\_HI" "PCT\_PI" "PCT\_WH"   
## [61] "MEALS" "EL" "YR\_RND" "SMOB" "DMOB"   
## [66] "ASC\_K3" "ASC\_46" "ASC\_CORE" "PCT\_RESP" "NOT\_HSG"   
## [71] "HSG" "SOME\_COL" "COL\_GRAD" "GRAD\_SCH" "AVG\_ED"   
## [76] "FULL" "EMER" "ENROLL" "IEP" "PARENT\_OPT"  
## [81] "TESTED" "SCI"

# Select the variables of interest for 2002  
APIBase2002 = subset(APIBase2002, select = c(CDS,STYPE,SNAME,DNAME,CNAME,FLAG,VALID,API02,  
 AVG\_ED,MEALS,EL,SCI,NOT\_HSG,HSG,SOME\_COL,COL\_GRAD,GRAD\_SCH))   
APIBase2002[APIBase2002$CDS == AHSCDS,] # Examine AHS data

## CDS STYPE SNAME DNAME CNAME FLAG VALID  
## 27 01611270130450 H Albany High Albany City Unified Alameda <NA> 576  
## API02 AVG\_ED MEALS EL SCI NOT\_HSG HSG SOME\_COL COL\_GRAD GRAD\_SCH  
## 27 817 4.13 9 8 171.504039 3 7 11 33 46

# The mssing values were removed in code that is not shown here.  
  
# Convert data to numeric data. Will get rid of the missing data at the end  
APIBase2002$VALID02 = as.numeric(APIBase2002$VALID)  
APIBase2002$API02 = as.numeric(APIBase2002$API02)  
APIBase2002$AVGED02 = as.numeric(APIBase2002$AVG\_ED)

## Warning: NAs introduced by coercion

APIBase2002$MEALS02 = as.numeric(APIBase2002$MEALS)  
APIBase2002$EL02 = as.numeric(APIBase2002$EL)  
APIBase2002$SCI02 = as.numeric(APIBase2002$SCI)  
APIBase2002$NOT\_HSG02 = as.numeric(APIBase2002$NOT\_HSG)

## Warning: NAs introduced by coercion

APIBase2002$HSG02 = as.numeric(APIBase2002$HSG)

## Warning: NAs introduced by coercion

APIBase2002$SOME\_COL02 = as.numeric(APIBase2002$SOME\_COL)

## Warning: NAs introduced by coercion

APIBase2002$COL\_GRAD02 = as.numeric(APIBase2002$COL\_GRAD)

## Warning: NAs introduced by coercion

APIBase2002$GRAD\_SCH02 = as.numeric(APIBase2002$GRAD\_SCH)

## Warning: NAs introduced by coercion

# Select a subset of high schools  
APIBase2002H = subset(APIBase2002, STYPE == 'H' & VALID02 >= 300 & is.na(FLAG),  
 select = c(CDS,SNAME,DNAME,CNAME,API02,AVGED02,MEALS02,EL02, SCI02,NOT\_HSG02,  
 HSG02,SOME\_COL02,COL\_GRAD02,GRAD\_SCH02))   
  
# Take a quick look at the tables  
length(APIBase2002H$API02) # 788

## [1] 788

head(APIBase2002H)

## CDS SNAME DNAME CNAME API02  
## 6 01611190130229 Alameda High Alameda City Unified Alameda 733  
## 9 01611190132878 Encinal High Alameda City Unified Alameda 606  
## 27 01611270130450 Albany High Albany City Unified Alameda 817  
## 48 01611500132225 Castro Valley High Castro Valley Unified Alameda 737  
## 64 01611760130062 American High Fremont Unified Alameda 681  
## 68 01611760134270 Irvington High Fremont Unified Alameda 732  
## AVGED02 MEALS02 EL02 SCI02 NOT\_HSG02 HSG02 SOME\_COL02 COL\_GRAD02  
## 6 3.55 13 14 169.3 6 14 21 36  
## 9 2.94 34 22 152.2 11 25 31 28  
## 27 4.13 9 8 171.5 3 7 11 33  
## 48 3.57 7 5 167.4 3 14 25 38  
## 64 3.29 15 12 162.7 8 19 24 36  
## 68 3.44 10 8 166.6 6 16 25 35  
## GRAD\_SCH02  
## 6 23  
## 9 6  
## 27 46  
## 48 20  
## 64 14  
## 68 19

``` ##################################### # Perform some of the manipulations # # using SQL # #####################################

This is just an indication of what can be done with SQL It is not used for the subsequent analysis

**SQL** '''

q1 = "SELECT CDS,STYPE,SNAME,DNAME,CNAME,FLAG,VALID, API02, AVGED02,   
Meals02, EL02, SCI02  
FROM APIBase2002  
"   
query1 = sqldf(q1)

## Loading required package: tcltk

head(query1, n = 15)

## CDS STYPE SNAME  
## 1 01100170130401 H Juvenile Hall/Court  
## 2 01100170130419 H County Community  
## 3 01100170130427 H Alternative/Opportunity  
## 4 01316090131755 H Calif School for the Blind  
## 5 01316170131763 H Calif School for the Deaf  
## 6 01611190130229 H Alameda High  
## 7 01611190130609 H Anderson Community Learning  
## 8 01611190130625 H Bay Area School  
## 9 01611190132878 H Encinal High  
## 10 01611190134304 H Island High (Cont.)  
## 11 01611196000004 M Chipman Middle  
## 12 01611196090005 E Lum (Donald D.) Elementary  
## 13 01611196090013 E Edison Elementary  
## 14 01611196090021 E Otis (Frank) Elementary  
## 15 01611196090039 E Franklin Elementary  
## DNAME CNAME FLAG VALID API02 AVGED02 MEALS02  
## 1 Alameda Co. Office of Educatio Alameda 4 <NA> NA NA NA  
## 2 Alameda Co. Office of Educatio Alameda 4 <NA> NA NA NA  
## 3 Alameda Co. Office of Educatio Alameda 4 <NA> NA NA NA  
## 4 Calif. School for the Blind Alameda 10 <NA> NA NA NA  
## 5 Calif. School for the Deaf Alameda <NA> 78 558 NA 0  
## 6 Alameda City Unified Alameda <NA> 1166 733 3.55 13  
## 7 Alameda City Unified Alameda <NA> 120 757 3.90 12  
## 8 Alameda City Unified Alameda <NA> 31 494 3.58 19  
## 9 Alameda City Unified Alameda <NA> 792 606 2.94 34  
## 10 Alameda City Unified Alameda 4 <NA> NA NA NA  
## 11 Alameda City Unified Alameda <NA> 532 634 2.92 55  
## 12 Alameda City Unified Alameda <NA> 313 781 3.36 32  
## 13 Alameda City Unified Alameda <NA> 233 850 3.84 14  
## 14 Alameda City Unified Alameda <NA> 258 789 3.47 24  
## 15 Alameda City Unified Alameda <NA> 143 812 3.60 27  
## EL02 SCI02  
## 1 NA NA  
## 2 NA NA  
## 3 NA NA  
## 4 NA NA  
## 5 100 NA  
## 6 14 169.3  
## 7 1 166.3  
## 8 0 NA  
## 9 22 152.2  
## 10 NA NA  
## 11 24 154.5  
## 12 27 171.0  
## 13 10 180.9  
## 14 25 175.3  
## 15 12 173.8

# R code  
# APIBase2002[APIBase2002$CDS == AHSCDS,] # Examine AHS data  
  
q2 = "SELECT CDS,STYPE,SNAME,DNAME,CNAME,FLAG,VALID,API02,AVGED02,MEALS02,EL02, SCI02  
FROM query1  
WHERE CDS = '01611270130450'  
"  
query2 = sqldf(q2)  
head(query2)

## CDS STYPE SNAME DNAME CNAME FLAG VALID  
## 1 01611270130450 H Albany High Albany City Unified Alameda <NA> 576  
## API02 AVGED02 MEALS02 EL02 SCI02  
## 1 817 4.13 9 8 171.5

# R code  
# Select on high schools that meet specified criteria   
APIBase2002H = subset(APIBase2002, STYPE == 'H' & VALID >= 300 & is.na(FLAG),  
 select = c(CDS,SNAME,DNAME,CNAME,FLAG, API02,AVG\_ED,MEALS,EL, SCI))   
head(APIBase2002H, n = 15)

## CDS SNAME  
## 5 01316170131763 Calif School for the Deaf  
## 8 01611190130625 Bay Area School  
## 9 01611190132878 Encinal High  
## 27 01611270130450 Albany High  
## 69 01611760134452 Kennedy (John F.) High  
## 139 01612000130492 Phoenix High (Cont.)  
## 140 01612000132670 Del Valle Continuation High  
## 162 01612340135426 Bridgepoint High (Cont.)  
## 184 01612590130146 Far West (Cont.)  
## 190 01612590130591 University Preparatory  
## 191 01612590132092 Castlemont Senior High  
## 194 01612590134791 McClymonds Senior High  
## 284 01612750136515 Piedmont High  
## 289 01612910134528 Lincoln High (Cont.)  
## 303 01613090137810 San Lorenzo High  
## DNAME CNAME FLAG API02 AVG\_ED MEALS EL  
## 5 Calif. School for the Deaf Alameda <NA> 558 N/A 0 100  
## 8 Alameda City Unified Alameda <NA> 494 3.58 19 0  
## 9 Alameda City Unified Alameda <NA> 606 2.94 34 22  
## 27 Albany City Unified Alameda <NA> 817 4.13 9 8  
## 69 Fremont Unified Alameda <NA> 648 2.97 20 15  
## 139 Livermore Valley Joint Unified Alameda <NA> 394 3.02 5 7  
## 140 Livermore Valley Joint Unified Alameda <NA> 458 2.87 10 7  
## 162 Newark Unified Alameda <NA> 419 2.49 1 0  
## 184 Oakland Unified Alameda <NA> 542 3.13 33 16  
## 190 Oakland Unified Alameda <NA> 620 3.15 27 0  
## 191 Oakland Unified Alameda <NA> 417 2.20 51 34  
## 194 Oakland Unified Alameda <NA> 437 2.47 58 15  
## 284 Piedmont City Unified Alameda <NA> 902 4.62 0 1  
## 289 San Leandro Unified Alameda <NA> 360 2.44 11 24  
## 303 San Lorenzo Unified Alameda <NA> 560 2.64 27 19  
## SCI  
## 5 <NA>  
## 8 <NA>  
## 9 152.229525  
## 27 171.504039  
## 69 157.520208  
## 139 <NA>  
## 140 <NA>  
## 162 <NA>  
## 184 <NA>  
## 190 <NA>  
## 191 135.260334  
## 194 131.594474  
## 284 177.327319  
## 289 <NA>  
## 303 148.173996

q3 = "SELECT CDS,SNAME,DNAME,CNAME, API02,AVGED02,MEALS02,EL02, SCI02  
FROM query1  
WHERE STYPE == 'H'   
AND VALID >= 300  
AND FLAG IS null  
;"  
  
  
query3 = sqldf(q3)  
head(query3, n= 15)

## CDS SNAME  
## 1 01316170131763 Calif School for the Deaf  
## 2 01611190130625 Bay Area School  
## 3 01611190132878 Encinal High  
## 4 01611270130450 Albany High  
## 5 01611760134452 Kennedy (John F.) High  
## 6 01612000130492 Phoenix High (Cont.)  
## 7 01612000132670 Del Valle Continuation High  
## 8 01612340135426 Bridgepoint High (Cont.)  
## 9 01612590130146 Far West (Cont.)  
## 10 01612590130591 University Preparatory  
## 11 01612590132092 Castlemont Senior High  
## 12 01612590134791 McClymonds Senior High  
## 13 01612750136515 Piedmont High  
## 14 01612910134528 Lincoln High (Cont.)  
## 15 01613090137810 San Lorenzo High  
## DNAME CNAME API02 AVGED02 MEALS02 EL02 SCI02  
## 1 Calif. School for the Deaf Alameda 558 NA 0 100 NA  
## 2 Alameda City Unified Alameda 494 3.58 19 0 NA  
## 3 Alameda City Unified Alameda 606 2.94 34 22 152.2  
## 4 Albany City Unified Alameda 817 4.13 9 8 171.5  
## 5 Fremont Unified Alameda 648 2.97 20 15 157.5  
## 6 Livermore Valley Joint Unified Alameda 394 3.02 5 7 NA  
## 7 Livermore Valley Joint Unified Alameda 458 2.87 10 7 NA  
## 8 Newark Unified Alameda 419 2.49 1 0 NA  
## 9 Oakland Unified Alameda 542 3.13 33 16 NA  
## 10 Oakland Unified Alameda 620 3.15 27 0 NA  
## 11 Oakland Unified Alameda 417 2.20 51 34 135.3  
## 12 Oakland Unified Alameda 437 2.47 58 15 131.6  
## 13 Piedmont City Unified Alameda 902 4.62 0 1 177.3  
## 14 San Leandro Unified Alameda 360 2.44 11 24 NA  
## 15 San Lorenzo Unified Alameda 560 2.64 27 19 148.2

q3a = "SELECT \*  
FROM query3 LIMIT 15;  
"  
sqldf(q3a)

## CDS SNAME  
## 1 01316170131763 Calif School for the Deaf  
## 2 01611190130625 Bay Area School  
## 3 01611190132878 Encinal High  
## 4 01611270130450 Albany High  
## 5 01611760134452 Kennedy (John F.) High  
## 6 01612000130492 Phoenix High (Cont.)  
## 7 01612000132670 Del Valle Continuation High  
## 8 01612340135426 Bridgepoint High (Cont.)  
## 9 01612590130146 Far West (Cont.)  
## 10 01612590130591 University Preparatory  
## 11 01612590132092 Castlemont Senior High  
## 12 01612590134791 McClymonds Senior High  
## 13 01612750136515 Piedmont High  
## 14 01612910134528 Lincoln High (Cont.)  
## 15 01613090137810 San Lorenzo High  
## DNAME CNAME API02 AVGED02 MEALS02 EL02 SCI02  
## 1 Calif. School for the Deaf Alameda 558 NA 0 100 NA  
## 2 Alameda City Unified Alameda 494 3.58 19 0 NA  
## 3 Alameda City Unified Alameda 606 2.94 34 22 152.2  
## 4 Albany City Unified Alameda 817 4.13 9 8 171.5  
## 5 Fremont Unified Alameda 648 2.97 20 15 157.5  
## 6 Livermore Valley Joint Unified Alameda 394 3.02 5 7 NA  
## 7 Livermore Valley Joint Unified Alameda 458 2.87 10 7 NA  
## 8 Newark Unified Alameda 419 2.49 1 0 NA  
## 9 Oakland Unified Alameda 542 3.13 33 16 NA  
## 10 Oakland Unified Alameda 620 3.15 27 0 NA  
## 11 Oakland Unified Alameda 417 2.20 51 34 135.3  
## 12 Oakland Unified Alameda 437 2.47 58 15 131.6  
## 13 Piedmont City Unified Alameda 902 4.62 0 1 177.3  
## 14 San Leandro Unified Alameda 360 2.44 11 24 NA  
## 15 San Lorenzo Unified Alameda 560 2.64 27 19 148.2

q4 = "SELECT COUNT(\*) As NumSchools  
FROM query3   
"  
  
query4 = sqldf(q4)   
query4 # Same as standard R result

## NumSchools  
## 1 536