ECS 132 Final Project

Aaron Okano, Anatoly Torchinsky, Justin Maple, Samuel Huang

December 10, 2012

1 Forest Fire

In [Cortez and Morais, 2007], the output 'area' was first transformed with a ln(x+1) function. Then, several Data Mining methods were applied to produce the data set of interest. Our goal with this data is to predict the fire size given the data set.

1.1 Which data matters?

Initially, doing variations and correlations between predictor variables and the response variable resulted in abnormally low values. This led us to consider what the forest looked like. We did this by creating a `frequencyGrid` and `areaGrid` that corresponds to each of X,Y coordinates that exist. The data shown was below:

```
> frequencyGrid(data)
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
 [1,]    0    0    0    0    0    0    0    0    0
 [2,]   19   25    0    0    0    0    0    0    0
 [3,]   10    1    1   22    0   25    2    3    0
 [4,]   15   27   43   36   23    9   45    1    4
 [5,]    4   20    7   25    3   49   11    4    2
 [6,]    0    0    4    8    4    3    2   52    1
 [7,]    0    0    0    0    0    0    0    0    0
 [8,]    0    0    0    0    0    0    0    1    0
 [9,]    0    0    0    0    0    0    0    0    6

> areaGrid(data)
          [,1]    [,2]      [,3]      [,4]     [,5]      [,6]      [,7]      [,8]     [,9]
 [1,]      NaN     NaN       NaN       NaN      NaN       NaN       NaN       NaN      NaN
 [2,] 11.57579 18.5060       NaN       NaN      NaN       NaN       NaN       NaN      NaN
 [3,] 15.71400  0.0000 6.5800000  7.858182      NaN  7.711200 13.675000   8.77000      NaN
 [4,] 10.01867  5.3100 2.9383721 11.039722 3.206522 16.052222 10.541556  12.18000  46.4025
 [5,] 28.86750  4.6315 0.3114286 11.480400 0.000000 28.245918  7.035455   0.73250   4.0800
 [6,]      NaN     NaN 0.0000000 10.966250 4.405000  2.863333 43.225000  24.33269  42.8700
 [7,]      NaN     NaN       NaN       NaN      NaN       NaN       NaN       NaN      NaN
 [8,]      NaN     NaN       NaN       NaN      NaN       NaN       NaN 185.76000      NaN
 [9,]      NaN     NaN       NaN       NaN      NaN       NaN       NaN       NaN   0.7450
```

`frequencyGrid` and `areaGrid` reveal several trends about the forest:

1. $\approx 44\%$ of the forest locations never had a fire
2. Locations with many fires usually have small mean areas
3. Locations with few fires usually have large mean areas

These trends suggested that certain parts of the data set were irrelevant to our prediction of fire area. In order to predict fire size in the places that actually had fires, we decided to ignore data that had particular X,Y coordinates with a mean area of NaN or frequency of 0.

1.2 Predictor Selection 1

Backwards step-wise approach

1.3 Predictor Selection 2

1.4 Final predictor set

After testing various predictor selection methods, we came up with the following predictor set:

- FFMC
- ISI
- DMC
- Month
- DC
- DC:ISI
- DMC:FFMC

1.5 Cross validation

to be filled...

2 Parkinson's disease

The dataset was created by Max Little of the University of Oxford, in collaboration with the National Centre for Voice and Speech, Denver, Colorado, who recorded the speech signals. The main aim of the data is to discriminate healthy people from those with PD, according to "status" column which is set to 0 for healthy and 1 for PD. Our goal with this data is to try to monitor patients with Parkinson's disease remotely, by simply analyzing their voices on the phone.

2.1 Predictor Selection 1

We used the following:

$$m_{Y;X}(t) = P(Y = 1|X = t) = \frac{1}{1 + e^{-(\beta_0+\beta_1 t_1+\ldots+\beta_r t_r)}} \tag{1}$$

predictor	class 0	class 1	Comments
spread1	0.4	0.9	Good!
PPE	0.4	0.9	Good!

2.2 Predictor Selection 2

2.3 Final predictor set

After testing various predictor selection methods, we came up with the following predictor set:

- spread1
- PPE

2.4 Cross validation

to be filled...

3 Beyond ECS 132

Professor Devanbu's paper, "Clones: What is that smell?" assesses the validity of the "stink" that surrounds clones. This reputation stems from the long standing belief that clone's require more project maintenance, as well as their tendency to create code bloat. One of the major problems that people face with software life cycles is maintenance costs, which can require around 80% of the total cost. As a result, Devanbu and others have invested time in research to minimize maintenance costs. One of the simplest ways to lower these costs involves reducing defects in code. Three tests were structured to analyze the relationship between clones and bugs. The first test determines the bug rate of cloned code; the second test compares the bug rate in cloned code to that of regular code; and the last test checks if prolific clone groups are buggier than the non-prolific clone groups. The results and conclusions of each test were formulated using statistical computation and analysis.

The first test was an attempt to discover to what extent cloned code contributed to bugs. The graph plots the cumulative bug convergence on the Y-axis against the clone ratio on the X-axis. According the vertical line, which represents the average clone ratio across all snapshots, both of the projects had that about 80% of bugs have a lower clone ratio than the overall project clone ratio. The test shows that in each case the bugs contained little cloned code.

The second test finds whether clones occur more often in buggy code than elsewhere. A box and whisker plot simply illustrates for all four projects that buggy code had a lower clone ratio. This provides strong evidence that clones are not a large contributor of bugs. To further support this claim, a second table displays the adjusted p values in which a Wilcoxon paired test is used as the null hypothesis and the alternative hypothesis is set to "snapshot clone ratio > bug clone ratio". Since the p values shown are between 0.01 and 0.05, this provides moderate evidence against the null hypothesis in favor of the alternative hypothesis. In each of the four projects, clones were not a major source of bugs. This suggests that clones are less buggy than regular code.

The third test assesses whether prolific clone groups are buggier than non-prolific clone groups. This was accomplished by finding the defect density in prolific clone groups and comparing it to those found in non-prolific clone groups. Figure 2 utilizes the resulting data in a box and whisker plot. The graph shows that in each case the bug density in prolific clone groups is lower than that of the non-prolific clone group. In addition, table three utilizes a Wilcoxon test with the alternative hypothesis set to "defect density in non-prolific groups > defect density of prolific group" provides p values that are below 0.5. These p values reject the null hypothesis in favor of the alternative, providing evidence in that more prolific clone groups are less buggy than non-prolific clone groups.

In each of the three tests, the clones were falsely attributed to a bad trait. Since the four projects used were medium to large open source projects, this data should be applicable in most situations. One area of contention over these results is how cloned code is identified. To address this issue, the tests were applied to two separate data sets. The first data set uses a conservative clone detector, while the second employs a more liberal one. Both detectors require 50 tokens in length to consider a code segment to be cloned, but the liberal detector allows for 1% less similarity than that of the conservative detector. Since analyzing both data sets led to the same conclusion, the difference in detection methods is negligible. Ultimately, the evidence from each test shows that clones may have unfairly garnered a bad reputation.

A Code

A.1 Problem 1

Listing 1: Analysis for Forest Fire Data Set

```
# Construct matrix with mean area for each coordinate
areaGrid <- function( data ) {
  # For a general dataset of the same format, since I'm in the mood
  origin <- min( data$X,data$Y )
  maximum <- max( data$X,data$Y )
  # Generate all possible coordinates. Use various transformations to create a
  # list of the desired form
  coords <- as.list(as.data.frame(t(expand.grid( origin:maximum, origin:maximum ))))
  # Create map with mean areas
  t(matrix( sapply( coords,
    function(x) mean( data[ which( data$X == x[1] & data$Y == x[2] ), ]$area ) ),
          maximum, maximum ))
}

# Modification to show mean temperature per location
tempGrid <- function( data ) {
  origin <- min( data$X,data$Y )
  maximum <- max( data$X,data$Y )
  coords <- as.list(as.data.frame(t(expand.grid( origin:maximum, origin:maximum ))))
  # Create map with mean temperature
  t(matrix( sapply( coords,
    function(x) mean( data[ which( data$X == x[1] & data$Y == x[2] ), ]$temp ) ),
          maximum, maximum ))
}

# Modification of above to show number of fires per location
frequencyGrid <- function( data ) {
  origin <- min( data$X,data$Y )
  maximum <- max( data$X,data$Y )
  coords <- as.list(as.data.frame(t(expand.grid( origin:maximum, origin:maximum ))))
  # Create map with fire frequencies
  t(matrix( sapply( coords,
    function(x) length( data[ which( data$X == x[1] & data$Y == x[2] ), ]$area ) ),
          maximum, maximum ))
}

# Find the mean area for each unique entry in dataVector, e.g. the mean area
# for each value of DC would be found with meanArea( data, data$DC ).
# Returns a matrix where the first column is the value in the vector and the
# second is the mean area for that value.
meanArea <- function( data, dataVector ) {
  uniques <- unique( dataVector )
  cbind( uniques, lapply( uniques, function(x)
                    mean( data[dataVector == x,]$area ) ) )
}

genCond <- function( v ) {
  sorted <- sort( v )
  s <- split( sorted, ceiling( seq( length(v) ) / (length(v)/3) ) )
```

```
  cbind( v >= head(s$'1', n=1) & v <= tail(s$'1', n=1),
         v >= head(s$'2', n=1) & v <= tail(s$'2', n=1),
         v >= head(s$'3', n=1) & v <= tail(s$'3', n=1))
}

# Mean area based on conditions
meanConds <- function(data, conditions, v) {
  mean( data[ !apply( conditions, 1, function(x)
                    any(!x[cbind(v,1:length(x[1,]))])), ]$area )
}

# Makes condition arrays using list of variables to form conditions on
makeConds <- function( variables ) {
  m <- lapply( variables, genCond )
  array( unlist( m ), dim = c(dim(m[[1]]),length(m)))
}

#conds <- makeConds( data[,5:11] )
#perms <- data.matrix( expand.grid( rep( list(1:4), 7 ) ) )
#means <- apply( perms, 1, function(x) meanConds( data, conds, x ) )
#table <- cbind( perms, means )
#table <- table[complete.cases(table),]
#table <- table[order(table[,8]),]
#mapply( function(x,y,n) mean( table[ table[,8] > x & table[,8] <= y, ][,n] ),
#        0, 4.4, 1:7 )


#model <- lm( log( area + 1 ) ~ month + temp + RH + DC + month:RH, data=ordata)


#> summary( lm( area ~ FFMC + ISI + ISI:FFMC + DC + DC:ISI + month, data=ordata
#                 > ) )
#
#Call:
#lm(formula = area ~ FFMC + ISI + ISI:FFMC + DC + DC:ISI + month,
#        data = ordata)
#
#Residuals:
#    Min      1Q  Median      3Q     Max
#-3.353  -1.556  -0.476   1.118   4.743
#
#Coefficients:
#                Estimate Std. Error t value Pr(>|t|)
#(Intercept) 18.7922107  8.5250001   2.204  0.02909 *
#FFMC        -0.2123159  0.1031263  -2.059  0.04132 *
#ISI         -2.1986055  1.3395298  -1.641  0.10291
#DC           0.0103264  0.0031436   3.285  0.00128 **
#month        2.6778341  0.8689659   3.082  0.00247 **
#FFMC:ISI     0.0258508  0.0146401   1.766  0.07956 .
#ISI:DC      -0.0004180  0.0002632  -1.588  0.11442
#---
#Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
#Residual standard error: 2.007 on 144 degrees of freedom
```

```
#Multiple R-squared: 0.1142, Adjusted R-squared: 0.0773
#F-statistic: 3.094 on 6 and 144 DF, p-value: 0.007022

##### Better Result #########

#> g <- lm(formula = area ~  FFMC + ISI +  DC + DC:ISI + month + DMC + DMC:FFMC , data=ordata,
#> summary(g)

#Call:
#lm(formula = area ~ FFMC + ISI + DC + DC:ISI + month + DMC +
#     DMC:FFMC, data = ordata)

#Residuals:
#     Min       1Q   Median       3Q      Max
#-3.3620 -1.4821 -0.3396  1.0433  4.8055

#Coefficients:
#               Estimate Std. Error t value Pr(>|t|)
#(Intercept) 23.1151201  9.0213988   2.562 0.011419 *
#FFMC        -0.2744283  0.1118159  -2.454 0.015301 *
#ISI          0.3185786  0.1871206   1.703 0.090799 .
#DC           0.0122545  0.0035439   3.458 0.000715 ***
#month        2.5926093  0.8779764   2.953 0.003673 **
#DMC         -0.2495317  0.1012981  -2.463 0.014935 *
#ISI:DC      -0.0006686  0.0002937  -2.277 0.024264 *
#FFMC:DMC     0.0027410  0.0011059   2.479 0.014335 *
#---
#Signif. codes:  0    ***    0.001    **    0.01    *    0.05    .    0.1         1

#Residual standard error: 1.999 on 145 degrees of freedom
#Multiple R-squared: 0.1379, Adjusted R-squared: 0.09632
#F-statistic: 3.314 on 7 and 145 DF, p-value: 0.002658


#> predictorMat <-cbind(ordata$FFMC, ordata$ISI, ordata$DC, ordata$month, ordata$DMC, ordata$
data <- read.csv('forestfires.csv',head=TRUE)
data$month <- factor(data$month,
     levels=c('jan','feb','mar','apr','may','jun',
              'jul','aug','sep','oct','nov','dec'))
data$day <-
  factor(data$day,levels=c('mon','tue','wed','thu','fri','sat','sun'))
data$month <- sin( as.integer( data$month ) * pi / 6 )
data$day <- sin( as.integer( data$day ) * 2 * pi / 7 )
ordata <- data[order(data$area),][248:400,]
ordata2 <- data[order(data$area),][248:448,]
ordata2$area <- log( ordata2$area + 1 )
```

A.2 Problem 2

Listing 2: Analysis for Parkinson's Data Set

```
parkinson <- read.csv('parkinsons.data', header=TRUE)

```

```
# Generalized logit function. Inputs are the input variables (the t's in the
# book) and the coefficients (the betas in the book). Both inputs are in vector
# form. Naturally, the vector t should have one less element than the vector b.
logit <- function(t,b) {1/(1+exp(-(b %*% c(1,t)))) }
crossvalglm <- function( response, predictor, predictor2, predictor3 ) {
  v <- sample( 1:(length(response) - 1), (length( response ) - 1) * 0.5 )
  notv <- setdiff( 1:(length(response) - 1), v )
  model <- glm( response[v] ~ predictor[v] + predictor2[v] + predictor3[v],
                family = binomial )
  cor( response[notv], mapply( logit3, model$coefficients[1],
                               model$coefficients[2],
                               model$coefficients[3],
                               model$coefficients[4],
                               predictor[notv],
                               predictor2[notv],
                               predictor3[notv])
  )^2
}

crossvalglm2 <- function( response, predictor, predictor2 ) {
  v <- sample( 1:(length(response) - 1), (length( response ) - 1) * 0.5 )
  notv <- setdiff( 1:(length(response) - 1), v )
  model <- glm( response[v] ~ predictor[v] + predictor2[v], family = binomial
                )
  cor( response[notv], mapply( logit2, model$coefficients[1],
                               model$coefficients[2],
                               model$coefficients[3],
                               predictor[notv],
                               predictor2[notv])
  )^2
}

crossvalglm3 <- function( response, predictor ) {
  v <- sample( 1:(length(response) - 1), (length( response ) - 1) * 0.5 )
  notv <- setdiff( 1:(length(response) - 1), v )
  model <- glm( response[v] ~ predictor[v,], family = binomial )
  cor( response[notv], mapply( logit, model$coefficients[1],
                               model$coefficients[2],
                               predictor[notv]
                               ) )^2
}

# Generalized glm cross validation. Takes as input a data frame where the first
# column is status and the remaining columns are predictors. Returns proportion
# of successful predictions.
# Call the function with cvglmprop( parkinson[,c("status","var1","var2",...)])
# If you want to add interaction terms, perform the multiplication beforehand:
# temp <- parkinson[,c("status","var1","var2")]
# temp$v1v2 <- temp$var1 * temp$var2
# cvglmprop( temp )
cvglmprop <- function( variables ) {
  v <- sample( 1:(nrow(variables) - 1), (nrow( variables ) - 1) * 0.5 )
  notv <- setdiff( 1:(nrow(variables) - 1), v )
  model <- glm( status ~ ., data = variables[v,], family = binomial )
```

```
  pred <- apply( as.matrix( variables[notv,-1] ), 1, logit,
                 model$coefficients )
  pred <- pred > 0.5
  mean( pred == variables[notv,1] )
}
```

B Who did what

- Aaron - implemented R functions for Problem 1 and 2 and did initial analysis
- Anatoly - assisted Aaron in development of Problem 1 and 2 code; used code to generate plots
- Justin - used Devanbu's article to finish Problem 3
- Samuel - wrote R to do initial analysis, copy-edited Problem 3 and did write-up for Problem 1 and 2