Marcus Crowder
LN-8.2 and 8.3
CIS 3920 Data-Mining

8.2

1. Apply CV to your current data set. Compare the selection to what you got from BICq.

For today's exercise the goal was to test out Cross Validation which is a part of the best glm Data.

I applied the data and some unusual results. For the CV the X-variables chosen were R(runs), ER(earned run), and WHIP, which is different from the choosen variables from the bicq algorithm that used just R and Whip. In addition the CV algorithm produced extremely high P-Values(basically unusable)

```
> head(Baseball.new)
    R  HR RBI   SO    BA   OBP   SLG OPS.  ERA  WHIP  ER   E Season
1 685 130 647 1142 0.259 0.323 0.391   96 3.92 1.301 651  75      1
2 688 181 656 1384 0.249 0.321 0.402   99 3.18 1.196 512  85      1
3 745 212 719 1125 0.260 0.313 0.431  101 4.20 1.315 678  54      1
4 853 178 819 1308 0.277 0.349 0.446  116 3.79 1.300 613  80      1
5 602 172 576 1230 0.238 0.300 0.392   89 4.00 1.293 643 100      0
6 598 148 574 1207 0.249 0.302 0.378   84 3.98 1.329 643 121      0
> CV.out = bestglm(Baseball.new, IC="CV", family=binomial)
Error in bestglm(Baseball.new, IC = "CV", family = binomial) :
  could not find function "bestglm"
> library(bestglm)
Loading required package: leaps
> CV.out = bestglm(Baseball.new, IC="CV", family=binomial)
Morgan-Tatar search since family is non-gaussian.
There were 50 or more warnings (use warnings() to see the first 50)
> CV.out
CVd(d = 18, REP = 1000)
BICq equivalent for q in (0.304836063521138, 0.845612911171659)
Best Model:
               Estimate   Std. Error       z value  Pr(>|z|)
(Intercept) 199.7898571 704923.6530  0.0002834206 0.9997739
R             0.7362572    849.4482  0.0008667477 0.9993084
ER           -0.8781377    995.2576 -0.0008823220 0.9992960
E            -1.4466129   1983.4014 -0.0007293596 0.9994181
> best.out = bestglm(Baseball, IC="BICq", family=binomial, TopModels=1)
Morgan-Tatar search since family is non-gaussian.
There were 50 or more warnings (use warnings() to see the first 50)
> best.out
BICq(q = 0.25)
BICq equivalent for q in (0.00226905182845594, 0.304836063521138)
Best Model:
                Estimate   Std. Error   z value  Pr(>|z|)
(Intercept)  164.7210503 140.02282905  1.176387 0.2394402
R              0.1122553   0.09379856  1.196770 0.2313961
WHIP        -183.0948870 149.86962224 -1.221694 0.2218232
```

Confused I decided to test the data based on the confusion matrix used in a previous learning exercise. The GLM and confusion matrix predicted 29/30 correct answers using the same xvariables which is pretty great but just enhanced y confusion towards the cross validation algorithm.

```
> glm.fit=glm(Season~R+WHIP,data=Baseball.new,family=binomial)
Warning message:
glm.fit: fitted probabilities numerically 0 or 1 occurred
> glm.fit

Call:  glm(formula = Season ~ R + WHIP, family = binomial, data = Baseball.new)

Coefficients:
(Intercept)            R         WHIP
   164.7211       0.1123     -183.0949

Degrees of Freedom: 29 Total (i.e. Null);  27 Residual
Null Deviance:        41.46
Residual Deviance: 5.05    AIC: 11.05
> glm.prob = predict(glm.fit,type="response")
> glm.forecast = Baseball$Season
> gml.forecast[glm.prob>0.5] = "Good"
Error in gml.forecast[glm.prob > 0.5] = "Good" :
  object 'gml.forecast' not found
> glb.forecast[glm.prob>0.5] = "Good"
Error in glb.forecast[glm.prob > 0.5] = "Good" :
  object 'glb.forecast' not found
> glm.forecast[glm.prob>0.5] = "Good"
> glm.forecast[glm.prob<0.5] = "Bad"
> summary(gml.forecast)
Error in summary(gml.forecast) : object 'gml.forecast' not found
> summary(glm.forecast)
 Bad Good
  15   15
> confusion.matrix = table(gml.forecast,Baseball$Season)
Error in table(gml.forecast, Baseball$Season) :
  object 'gml.forecast' not found
> confusion.matrix = table(glm.forecast,Baseball$Season)
> confusion.matrix

glm.forecast Bad Good
        Bad   14    1
        Good   0   15
>
```

My question for CV is simple why does CV have a higher p-value for the sample data that works with bestglm algorithm. In addition to choice of different x-variables. I was extremely confused by this but I'm guessing CV has higher standards for testing if an x-variable has any affect amd the tests done by CV are not just meant to fit the sample data.

8.3

Walking through 8.3 I encountered a problem.

The Dcardiac.LGT data that was provided in the professor's R data was lacking a column name of hardness so when I tried to use the glm.fit=glm(hardness~.data=Dcardiac.LGT, family=binomial) I received an error stating that hardness could not be found. So I had to search online on how to swap the names of columns. What I encountered was the column for hardness was name x[, 19, drop = TRUE] meaning an error occurred where the column name received the r commands used to produce it. Which happens in coding more often that one would think.

```
> Dcard = Dcardiac.LGT
> head(Dcard)
        bhr    basebp x[, 19, drop = TRUE]
1 0.5476190 0.8728814                    1
2 0.3690476 1.1779661                    0
3 0.3690476 1.1779661                    0
4 0.5535714 1.0000000                    1
5 0.5297619 0.8728814                    1
6 0.3452381 0.8474576                    1
> head(Dcard[,3])
[1] 1 0 0 1 1 1
> colnames(Dcard)[3]<-"hardness"
```

To fix this problem I used the colnames function

Colnames(Dcard)[3] <-"hardness"

I also renamed the Dcardiac.LGT dataset into Dcard to not alter it. Finally the Dcard data worked and found hardness. Additionally I was curious towards the outlier in the Cardiac data set at position 412.

```
> colnames(Dcard)[3]<-"hardness"
> glm.fit=glm(hardness~.,data=Dcardiac.LGT, family=binomial)
Error in eval(predvars, data, env) : object 'hardness' not found
> glm.fit=glm(hardness~.,data=Dcard, family=binomial)
> glm.probs=predict(glm.fit,type="response")
> glm.probs[1:5]
        1         2         3         4         5
0.8505819 0.8451741 0.8451741 0.8416282 0.8520252
> which.max(Cardiac[,1])
[1] 412
```

So I used the slicing operator

> Cardiac.Scrub[412,1]

and found the value to be 1.25

I tried to find an outlier among my data set by plotting the values vs the Season Statistics for Baseball.

```
> Baseball <- read.csv('~/Downloads/Baseball2013.csv', header = TRUE)
> head(Baseball)
    R  HR RBI   SO    BA   OBP   SLG OPS.  ERA  WHIP  ER   E Season
1 685 130 647 1142 0.259 0.323 0.391   96 3.92 1.301 651  75   Good
2 688 181 656 1384 0.249 0.321 0.402   99 3.18 1.196 512  85   Good
3 745 212 719 1125 0.260 0.313 0.431  101 4.20 1.315 678  54   Good
4 853 178 819 1308 0.277 0.349 0.446  116 3.79 1.300 613  80   Good
5 602 172 576 1230 0.238 0.300 0.392   89 4.00 1.293 643 100    Bad
6 598 148 574 1207 0.249 0.302 0.378   84 3.98 1.329 643 121    Bad
> y=cbind(Baseball[,1,drop=FALSE],Baseball[,10,drop=False],Baseball[,13,drop=False])
Error in `[.data.frame`(Baseball, , 10, drop = False) :
  object 'False' not found
> y=cbind(Baseball[,1,drop=FALSE],Baseball[,10,drop=FALSE],Baseball[,13,drop=FALSE])
> head(y)
    R  WHIP Season
1 685 1.301   Good
2 688 1.196   Good
3 745 1.315   Good
4 853 1.300   Good
5 602 1.293    Bad
6 598 1.329    Bad
> y$Season = c(y$Season)
> head(y)
    R  WHIP Season
1 685 1.301      2
2 688 1.196      2
3 745 1.315      2
4 853 1.300      2
```

I brought in my baseball data set then decided to use the Runs and WHIP x variables to test for outliers.But first I had to replace the Season stats for Good to equal 1 and Bad season to equal 0

```
> y$Season = c(y$Season)
> head(y)
    R  WHIP Season
1 685 1.301      2
2 688 1.196      2
3 745 1.315      2
4 853 1.300      2
5 602 1.293      1
6 598 1.329      1
> y[y[,3]==1,3] =0
> head(y)
    R  WHIP Season
1 685 1.301      2
2 688 1.196      2
3 745 1.315      2
4 853 1.300      2
5 602 1.293      0
6 598 1.329      0
> y[y[,3]==2,3] =1
> head(y)
    R  WHIP Season
1 685 1.301      1
2 688 1.196      1
3 745 1.315      1
4 853 1.300      1
5 602 1.293      0
6 598 1.329      0
```

Again I used the predict function with the best glm algorithm to predict 29/30 seasons correctly. Data does not seem to contain any outliers just based on this but that's naïve maybe I could do 30/30 so I went digging.

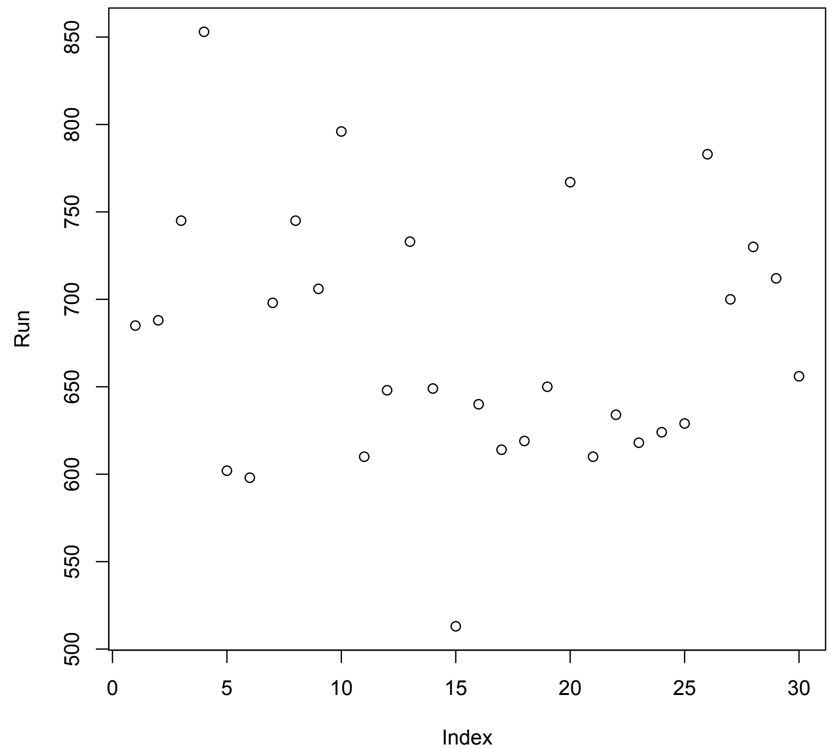
```
> glm.fit=glm(Season~.,data=y,family=binomial)
Warning message:
glm.fit: fitted probabilities numerically 0 or 1 occurred
> glm.probs=predict(glm.fit,type="response")
> table(glm.forecast,Baseball$Season)
Error in table(glm.forecast, Baseball$Season) :
  all arguments must have the same length
> head(glm.probs)
           1            2            3            4            5            6
9.680000e-01 1.000000e+00 9.994905e-01 1.000000e+00 1.162462e-02 1.029981e-05
> glm.forecast=y$Season
> glm.forecast[glm.probs>0.5]=1
> glm.forecast[glm.probs<0.5]=0
> table(glm.forecast,y$Baseball)
Error in table(glm.forecast, y$Baseball) :
  all arguments must have the same length
> table(glm.forecast,y$Season)

glm.forecast  0  1
           0 14  1
           1  0 15
```

So I decided to look for outliers with the plot function. I plotted both runs and WHIP by themselves to see which one had the biggest deviation from other points (this was a naïve apparaoach but there was no other way to classify the points as a linear model with the classifications of 1 and 0 it may have been easier to detect an outlier if it was based on Wins instead of good or bad season but these are classification problems)

So I decided to plot Run with the function and noticed a extremely high and low point with the low point drawing me to pay more attention to it because it was the furthest from any other point. So I used the which.min function to find out where it was.

> plot(y[,1], ylab="Run")
> which.min(y)
Error in which.min(y) : (list) object cannot be coerced to type 'double'
> which.min(y[,1])
[1] 15



I ended up removing the point from my copy baseball stats held in Y. To get rid of it I used y[-15,-1]. Which copied all he data To x except for the data in the 15th row of the 1st column The Run Column and the lowest run point (potential outlier) Then as you can see position 15 on my new X dataframe has been removed.

```
> x = y[-15,-1]
> head(x)
   WHIP Season
1 1.301      1
2 1.196      1
3 1.315      1
4 1.300      1
5 1.293      0
6 1.329      0
> x
    WHIP Season
1  1.301      1
2  1.196      1
3  1.315      1
4  1.300      1
5  1.293      0
6  1.329      0
7  1.173      1
8  1.327      1
9  1.436      0
10 1.252      1
11 1.490      0
12 1.267      1
13 1.378      0
14 1.228      1
16 1.294      0
17 1.413      0
18 1.287      0
19 1.305      1
20 1.217      1
21 1.372      0
22 1.233      1
23 1.328      0
24 1.328      0
25 1.313      0
26 1.245      1
27 1.227      1
28 1.277      1
29 1.344      0
30 1.226      1
```

Finally I used the glm function to predict Season against R and Whip without the potential outlier this time.

```
> glm.fit=glm(Season~.,data=x,family=binomial)
> glm.probs=predict(glm.fit,type="response")
> glm.probs
           1            2            3            4            5            6            7            8            9           10           11           12 
5.535974e-01 9.971834e-01 3.684955e-01 5.668635e-01 6.561113e-01 2.154182e-01 9.991821e-01 2.341776e-01 8.628000e-04 9.455169e-01 4.714023e-05 8.855528e-01 
          13           14           16           17           18           19           20           21           22           23           24           25 
1.924268e-02 9.844225e-01 6.438612e-01 2.970867e-03 7.249419e-01 4.999546e-01 9.913248e-01 2.638816e-02 9.797071e-01 2.246589e-01 2.246589e-01 3.938929e-01 
          26           27           28           29           30 
9.619766e-01 9.852271e-01 8.187058e-01 1.090665e-01 9.859907e-01 
> glm.forecast=x$Season
> glm.forecast[glm.probs>0.5]=1
> glm.forecast[glm.probs<0.5]=0
> table(glm.forecast,x$Season)

glm.forecast  0  1
           0 10  3
           1  3 13
>
```

By creating glm.probs from the predict function. It turned out that without the potential outlier the prediction rate went down. The success rate of the model went to 23/29 which is not as good as the previous 29/30. With the substantially lower prediction rate I realized that The point was not an outlier and actually helped the predictions. But it was worth learning how to removed points from the dataframe.