Project Four for Data Mining

Raymond Anthony Ford
raford2@miners.utep.edu

Due: 27 March 2018

Contents

1 Exploratory data analysis

We begin our sabermetrical journey by bringing the data in from the American Statistical Association's website.

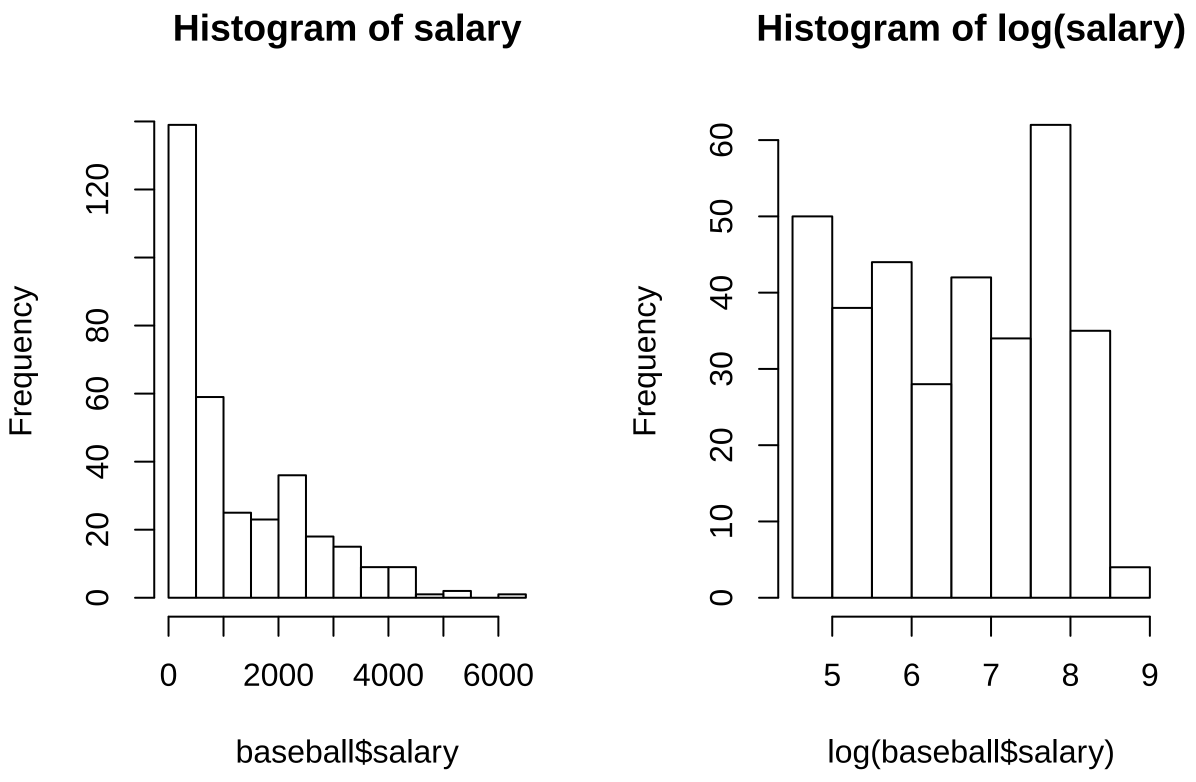

Figure 1: Histograms for salary and log(salary).

```
baseball <- read.table(file=
    "http://www.amstat.org/publications/jse/datasets/baseball.dat.txt",
    header = F, col.names=c("salary", "batting.avg", "OBP", "runs", "hits",
    "doubles", "triples", "homeruns", "RBI", "walks", "strike.outs",
    "stolen.bases", "errors", "free.agency.elig", "free.agent.91",
    "arb.elig", "arb.91", "name"))
```

Our goal is to predict a player's salary using linear regression, and we are instructed to plot histograms for both the salary and the logarithm of the salary.[1]

```
par(mfrow=c(1,2))
hist(baseball$salary, main="Histogram of salary")
hist(log(baseball$salary), main="Histogram of log(salary)")
```

We see in Figure 1 that the salary is heavily skewed using the raw data and the relationship appears to be exponential, but this skewness is greatly reduced when applying the logarithmic transformation. Since we will be using the logarithmic transformation for salaries for the rest of this project, we amend the data as follows.

```
baseball$salary <- log(baseball$salary)
names(baseball)[1] <- "log.salary"
```

[1]Unless otherwise specified, we will use the logarithm with the natural base e throughout this project.

Column No.	Variable	Type	No. Unique	No. Complete	Missing Proportion
1	log.salary	double	208	337	0
2	batting.avg	double	133	337	0
3	OBP	double	139	337	0
4	runs	integer	105	337	0
5	hits	integer	163	337	0
6	doubles	integer	44	337	0
7	triples	integer	14	337	0
8	home runs	integer	36	337	0
9	RBI	integer	106	337	0
10	walks	integer	90	337	0
11	strike.outs	integer	118	337	0
12	stolen.bases	integer	48	337	0
13	errors	integer	28	337	0
14	free.agency.elig	binary	2	337	0
15	free.agent.91	binary	2	337	0
16	arb.elig	binary	2	337	0
17	arb.91	binary	2	337	0
18	name	character	337	337	0

Table 1: Summary of the baseball data set.

We are also instructed to inspect the data set used for this project. We see in Table 1 that we have no missing data and the following counts for each predictor type: two continous (double) predictors, ten integer count predictors, four dichotomous (binary) predictors, 337 character predictors (name of the baseball player)[2], and our one target variable `log.salary` is of double type.[3]

2 Linear regression and variable selection methods

Next we will build a regression model to try to predict a baseball player's salary given his performance data.

2.1 Partitioning the data

We first start by partitioning the data randomly in two distinct sets with a ratio of approximately 2:1.

[2]We will not use this as a predictor *per se*, as our design matrix would be sparse and $p > n$, where p is the number of predictors and n is the number of players for which we have data on.

[3]The `R` code and output used to create this table can be found in the appendix of this project.

```r
set.seed(5474)
training.data.index <- sample(1:nrow(baseball), 0.667*nrow(baseball))
training.data <- baseball[training.data.index, ]
testing.data  <- baseball[-training.data.index, ]
```

We will use the training data to fit three different models, each created using a different variable selection method, and then use the testing data to arrive at a “best” model that will be used throughout the rest of this project.

2.2 Variable selection methods

We now deploy three different variable selection methods to our data: backward elimination, the least absolute shrinkage and selection operator (LASSO), and a Bayesian approach. Our hope is to find a “best model” and then see how well this “best model” performs given new data.

2.2.1 Backward elimination

Our first venture into variable selection is to employ backward elimination. We start by first fitting the full model containing all of the predictors.

```r
baseball.back <- lm(log.salary ~. -name, data=training.data)
```

With this full model, we then proceed by deleting one predictor-at-a-time until an optimal (minimum) AIC is computed. The model with this minimum will be considered the best model from the backward elimination.[4]

```r
baseball.backward <- step(baseball.back, direction="backward", trace=0)
summary(baseball.backward)
```

```
##
## Call:
## lm(formula = log.salary ~ hits + RBI + walks + strike.outs +
##     errors + free.agency.elig + free.agent.91 + arb.elig, data = training.data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.39429 -0.28213 -0.00838  0.32632  1.35469
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       4.881651   0.078389  62.275  < 2e-16 ***
## hits              0.005340   0.001505   3.548 0.000476 ***
```

[4]The output from this method can be found in the appendix of this project.

```
## RBI               0.011930   0.002688   4.439 1.45e-05 ***
## walks             0.005333   0.002423   2.201 0.028816 *
## strike.outs      -0.004800   0.001710  -2.807 0.005457 **
## errors           -0.012136   0.006776  -1.791 0.074664 .
## free.agency.elig  1.422893   0.108859  13.071  < 2e-16 ***
## free.agent.91    -0.187181   0.130729  -1.432 0.153646
## arb.elig          1.182013   0.106144  11.136  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5361 on 215 degrees of freedom
## Multiple R-squared:  0.8084, Adjusted R-squared:  0.8012
## F-statistic: 113.4 on 8 and 215 DF,  p-value: < 2.2e-16
```

From the `summary()` output, we see that the best model discovered via backward elimination contains the following predictors: `hits, RBI, walks, strike.outs, errors, free.agency.elig, free.agent.91, arb.elig`. That is, from 17 possible predictors, backward elimination selected 8 of them to be included in the model.

Moreover, we can view some of the selected predictors and their coefficients for a sanity check. As expected, `strike.outs` and `errors` negatively impact a player's salary, while `RBI` and `hits` have a positive impact.

2.2.2 Least absolute shrinkage and selection operator (LASSO)

Our next model selection method is the LASSO. LASSO acts as variable selection method as some of the coefficients will shrink towards zero. In addition to the explanation given in Hastie, Tibshirani, and Friedman (2009), another way to view LASSO is in the Bayesian context where we assume that the β's in the model have a prior distribution of the double exponentional.

While normally one would want to standardize the data prior to fitting a LASSO model, we do not need to do that in the following code, as the functions available in the `ncvreg` package do this for us.

```r
library(ncvreg)
set.seed(5474)
baseball.L1 <- cv.ncvreg(X=training.data[,-c(1,18)], y=training.data[,1],
                         nfolds=10, family="gaussian", penalty="lasso",
                          lambda.min=.005, nlambda=100, eps=.001,
                          max.iter=1000)
beta.hat <- coef(baseball.L1)
cutoff <- 0
terms <- names(beta.hat)[abs(beta.hat) > cutoff]
formula.LASSO <- as.formula(paste(c("log.salary ~ ", terms[-1]),
                                  collapse= "+"))
```

```r
baseball.L1 <- lm(formula.LASSO, data=training.data[,-18])
summary(baseball.L1)
```

```
##
## Call:
## lm(formula = formula.LASSO, data = training.data[, -18])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.39123 -0.25951 -0.02732  0.37337  1.31416
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       4.895834   0.082516  59.332  < 2e-16 ***
## runs             -0.001126   0.005099  -0.221  0.82546
## hits              0.005736   0.002749   2.086  0.03816 *
## doubles           0.004222   0.008201   0.515  0.60722
## triples          -0.010688   0.021117  -0.506  0.61330
## homeruns          0.009205   0.012702   0.725  0.46946
## RBI               0.008809   0.005097   1.728  0.08542 .
## walks             0.005499   0.003097   1.776  0.07724 .
## strike.outs      -0.005443   0.001966  -2.769  0.00614 **
## stolen.bases      0.004011   0.004503   0.891  0.37412
## errors           -0.011237   0.006931  -1.621  0.10645
## free.agency.elig  1.418450   0.111973  12.668  < 2e-16 ***
## free.agent.91    -0.168842   0.132833  -1.271  0.20511
## arb.elig          1.221698   0.111715  10.936  < 2e-16 ***
## arb.91           -0.326232   0.239462  -1.362  0.17455
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5393 on 209 degrees of freedom
## Multiple R-squared:  0.8115, Adjusted R-squared:  0.7989
## F-statistic: 64.27 on 14 and 209 DF,  p-value: < 2.2e-16
```

From the `summary()` output, we see that the best model discovered via LASSO contains the following predictors: `runs`, `hits`, `doubles`, `triples`, `homeruns`, `RBI`, `walks`, `strike.outs`, `stolen.bases`, `errors`, `free.agency.elig`, `free.agent.91`, `arb.elig`, and `arb.91`. The sign of some of the coefficients is a little perplexing. For example, `runs` and `triples` have negative coefficients, but we would expect them to be positive as these help teams win games and we would think that they should have a positive impact on a player's salary.

2.2.3 Bayesian

Our final variable selection method is a Bayesian one. We will use model selection method using Zellner's g as outlined in Marin and Robert (2014). Our goal is to compare this method with the other methods discussed in the class and see how they compare.

```r
suppressMessages(library(bayess, quietly=TRUE))
set.seed(5474)
baseball.bayes.var.sel <- ModChoBayesReg(y=training.data[, 1],
                                         X=training.data[, -c(1, 18)],
                                         g=length(training.data[, 1]),
                                         prt=FALSE)
```

We need to find our top model (the one with highest posterior probability), and this is accomplished with the following code.

```r
top.model <- baseball.bayes.var.sel$top10models[1]
top.model
```

```
## [1] "3 8 10 13 15"
```

We're only given the number (column) for each variable in the data matrix and not it's name, so we use the code below to find out the names of the variables selected.

```r
which.variables <- as.numeric(strsplit(top.model, " ")[[1]])
vars.selected <- NULL
row.selected <- NULL
for (k in which.variables) {
    selected <- k
    actual.name <- names(training.data)[k]
    row.selected <- cbind(selected, actual.name)
    vars.selected <- rbind(vars.selected, row.selected)
}
vars.selected
```

```
##      selected actual.name
## [1,] "3"      "OBP"
## [2,] "8"      "homeruns"
## [3,] "10"     "walks"
## [4,] "13"     "errors"
## [5,] "15"     "free.agent.91"
```

We see that this Bayesian method selected `OBP`, `homeruns`, `walks`, `errors`, and `free.agent.91`. We now fit this model, and output it's summary.

```r
baseball.bayes <- lm(log.salary ~ OBP + homeruns + walks + errors +
                     free.agent.91, data=training.data[,-c(18)])
summary(baseball.bayes)
```

```
##
## Call:
## lm(formula = log.salary ~ OBP + homeruns + walks + errors + free.agent.91,
##     data = training.data[, -c(18)])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.56711 -0.63573 -0.00436  0.66089  1.62928
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    5.214372   0.483261  10.790  < 2e-16 ***
## OBP            0.392606   1.602097   0.245   0.8066
## homeruns       0.038451   0.007737   4.970 1.36e-06 ***
## walks          0.021398   0.003546   6.035 6.77e-09 ***
## errors        -0.002801   0.009442  -0.297   0.7670
## free.agent.91  0.453022   0.183442   2.470   0.0143 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8606 on 218 degrees of freedom
## Multiple R-squared:  0.4992, Adjusted R-squared:  0.4877
## F-statistic: 43.46 on 5 and 218 DF,  p-value: < 2.2e-16
```

The variables selected with method all make sense intuitively, but it should be noted that the adjusted R^2 for this model is horrible when compared with the other two methods. Having only 49% of the variation explained makes this model a poor choice, but we will compare it's performance to the other models in the next subsection.

2.3 Which model and why?

We now need to select a final model. To do this we compare the performance of the three above models using SSPE, and select the one having the lowest SSPE.

```r
predict.backward <- predict(baseball.backward, testing.data[, -c(1, 18)])
sspe.backward <- sum((testing.data[,1] - predict.backward) **2)

predict.L1 <- predict(baseball.L1, testing.data[, -c(1, 18)])
sspe.L1 <- sum((testing.data[,1] - predict.L1) **2)

predict.bayes <- predict(baseball.bayes, testing.data[, -c(1, 18)])
sspe.bayes <- sum((testing.data[,1] - predict.bayes) **2)

SSPE <- matrix(c("sspe.backward", sspe.backward, "sspe.L1", sspe.L1,
                 "sspe.bayes", sspe.bayes), nrow=3, byrow=TRUE)
```

```
SSPE
```

```
##      [,1]             [,2]
## [1,] "sspe.backward" "35.3273156056771"
## [2,] "sspe.L1"       "36.0983297607705"
## [3,] "sspe.bayes"    "121.970225261275"
```

We see that we should use the model fitted with backwards elimination.

2.4 The final model

We now fit our final model by using the predictors selected with backward elimination, and use the entire baseball data set to fit the model.

```
fit.final <- lm(log.salary ~ hits + RBI + walks + strike.outs + errors +
                free.agency.elig + free.agent.91 + arb.elig,
                data=baseball[,-18])
summary(fit.final)
```

```
##
## Call:
## lm(formula = log.salary ~ hits + RBI + walks + strike.outs +
##     errors + free.agency.elig + free.agent.91 + arb.elig, data = baseball[,
##     -18])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.43061 -0.29106 -0.03605  0.32692  1.50084
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       4.945703   0.067270  73.521  < 2e-16 ***
## hits              0.004749   0.001215   3.908 0.000113 ***
## RBI               0.010536   0.002223   4.740 3.19e-06 ***
## walks             0.003423   0.001911   1.791 0.074247 .
## strike.outs      -0.004311   0.001374  -3.138 0.001856 **
## errors           -0.007602   0.005679  -1.339 0.181646
## free.agency.elig  1.606375   0.081298  19.759  < 2e-16 ***
## free.agent.91    -0.270868   0.104750  -2.586 0.010145 *
## arb.elig          1.315687   0.086930  15.135  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5359 on 328 degrees of freedom
## Multiple R-squared:  0.7975, Adjusted R-squared:  0.7925
## F-statistic: 161.4 on 8 and 328 DF,  p-value: < 2.2e-16
```

3 Regression diagnostics

We will now perform some regression diagnostics.

3.1 Normality

We begin our regression diagnostics by checking that the normality assumption has not been violated. We start by making a histogram and Q-Q plot.

```
library(car)
final.jack <- rstudent(fit.final)
par(mfrow=c(1,2))
hist(final.jack, main="Studentized residuals", xlab="Value", prob=TRUE)
curve(dnorm(x, mean=mean(final.jack), sd=sd(final.jack)), add=TRUE,
      col="orange", lwd=2)
invisible(qqPlot(fit.final, main="Q-Q plot", col="mediumblue",
                 col.lines="orange", pch="*", grid=FALSE,
                 labels=baseball$name, id.n=5))
```

In the histogram in Figure 2 there appears to be some outliers, and the Q-Q plot confirms this. So, we perform a Shapiro-Wilk test for normality.

```
shapiro.test(final.jack)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  final.jack
## W = 0.9505, p-value = 3.227e-09
```

We see that we have such a small p-value and thus conclude that the studentized jackknife residuals are not normally distributed for any reasonable level of significance.

3.2 Homoscedasticity

We next check whether or not we violated our homoscedasticity assumption. We begin by plotting the absolute jackknife residuals against the fitted values.

```
invisible(spreadLevelPlot(fit.final, main="Spread level plot", col="mediumblue",
                          col.lines="orange", col.smoother="red", pch="*",
                          grid=FALSE))
```

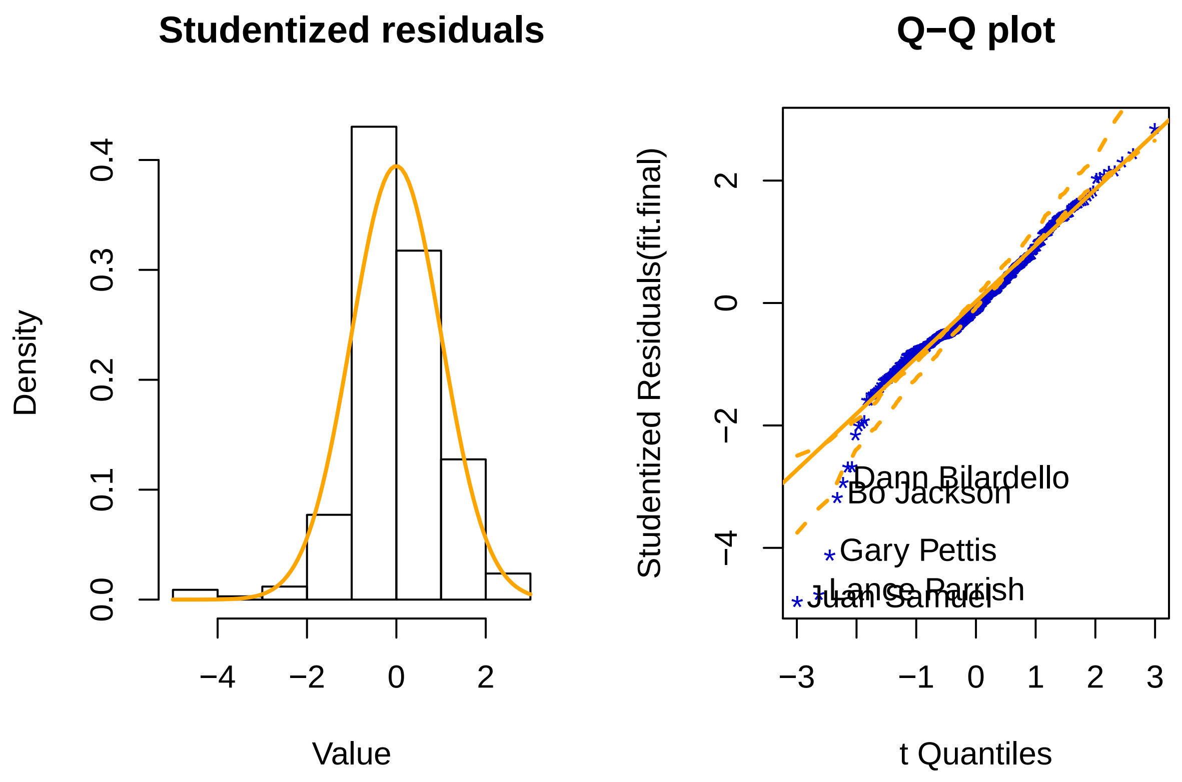

Figure 2: Histogram of studentized jackknife residuals and Q-Q plot.

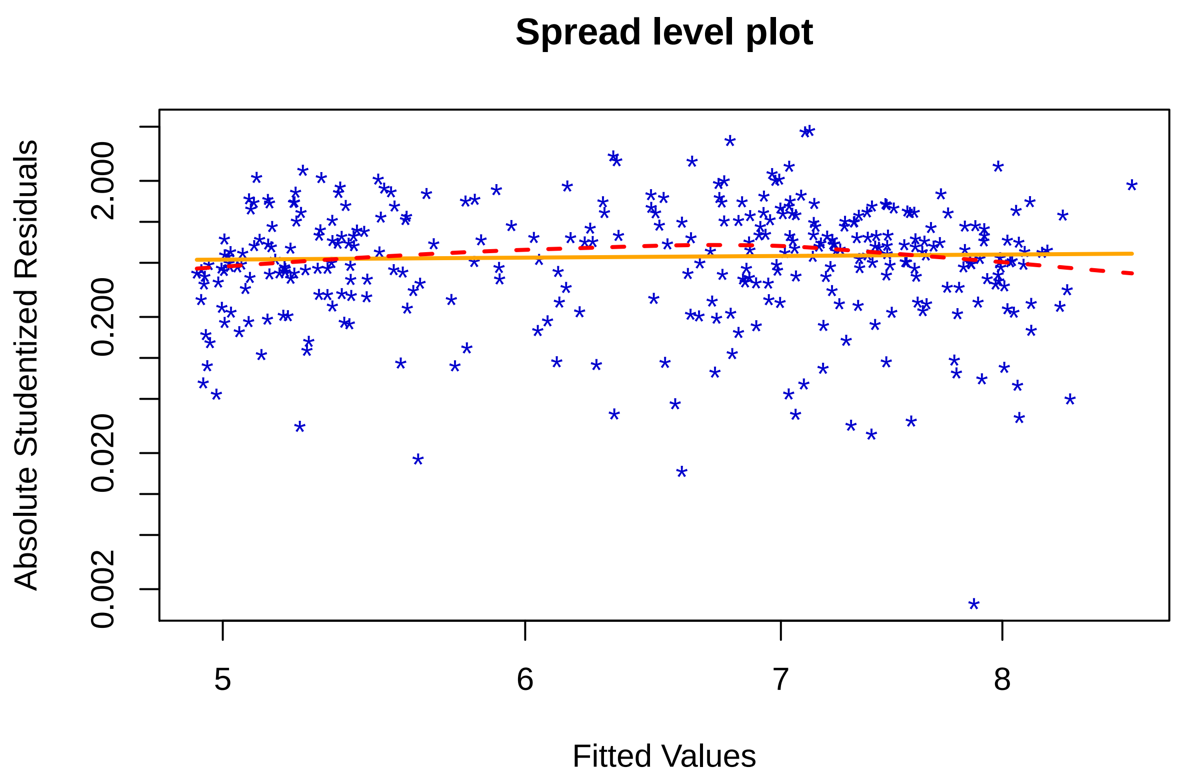

Figure 3: Spread level plot to check homoscedasticity assumption.

In Figure 3 we do not notice any issues, but just to be safe we perform a Breusch-Pagan test to check for non-constant error variance.

```r
ncvTest(fit.final)
```

```
## Non-constant Variance Score Test
## Variance formula: ~ fitted.values
## Chisquare = 0.2951675    Df = 1     p = 0.586928
```

We have a fairly large *p*-value which suggests that we have not violated the homoscedasticity assumption.

3.3 Independence

We next need to test for autocorrelated errors, so we perform a Durbin-Watson test.

```r
durbinWatsonTest(fit.final)
```

```
##  lag Autocorrelation D-W Statistic p-value
##    1       0.1026467      1.793532   0.058
##  Alternative hypothesis: rho != 0
```

From the output we see that there is no evidence to suggest autocorrelated errors when using a level of significance of 0.05 as our *p*-value is 0.058.

3.4 Linearity

Next, we need to check linearity for each of the continuous predictors in our model.

```r
leveragePlots(fit.final, col="mediumblue", col.lines="orange", grid=FALSE,
              pch=".", main="Leverage plots")
```

In Figure 4 we see that the relationship is not strictly linear, but the linear lines fit the data in an okay fashion. There are some points that are likely influencing the slope of the line. In particular, there are a lot of points blow the line in RBI|others, hits|others, and walks|others.

3.5 Outlier detection

While we're instructed to construct a bubble plot to identify outliers, we prefer to use the index plot of diagnostic statistics from Fox and Weisberg (2013), as they are more informative.

```r
influenceIndexPlot(fit.final, col="orange", labels=baseball$name, id.n=5,
                   id.col="mediumblue", grid=FALSE, pch="*")
```

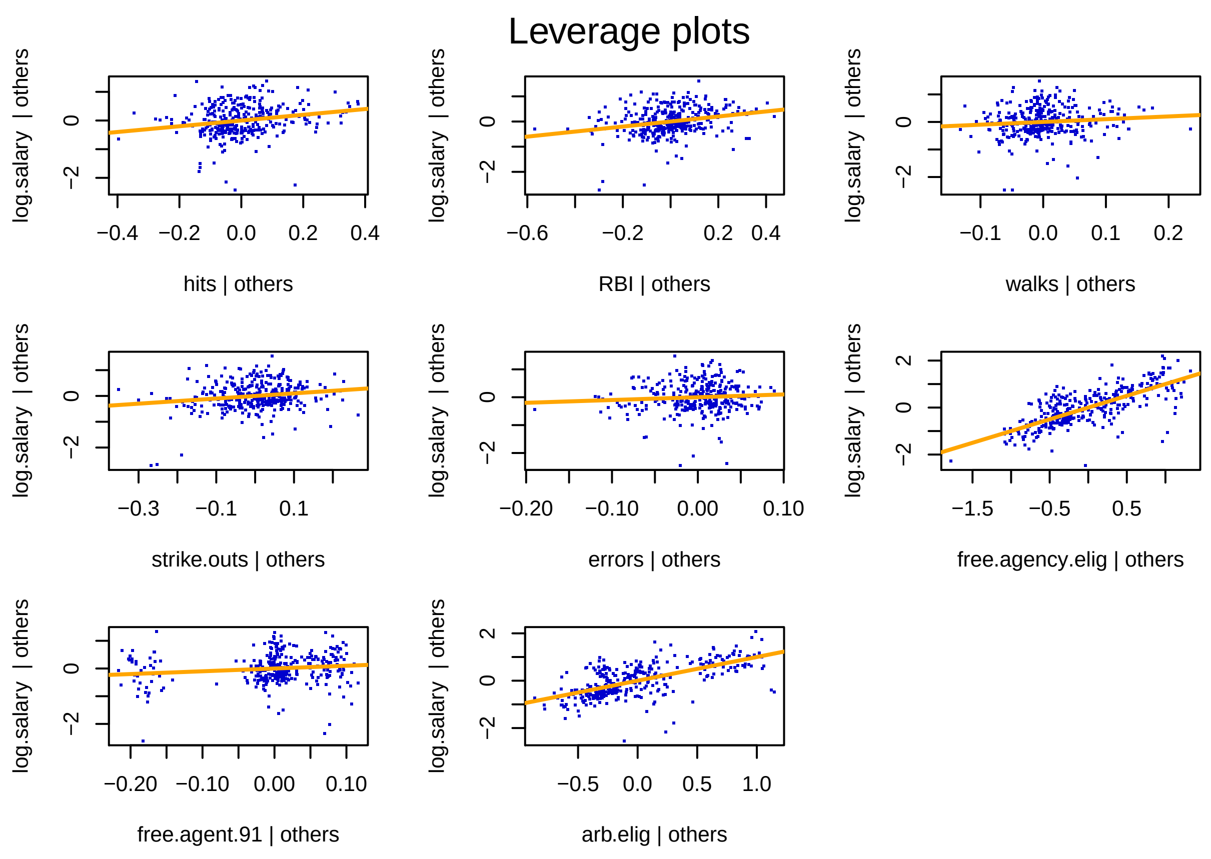

Figure 4: Leverage plots for our final regression model.

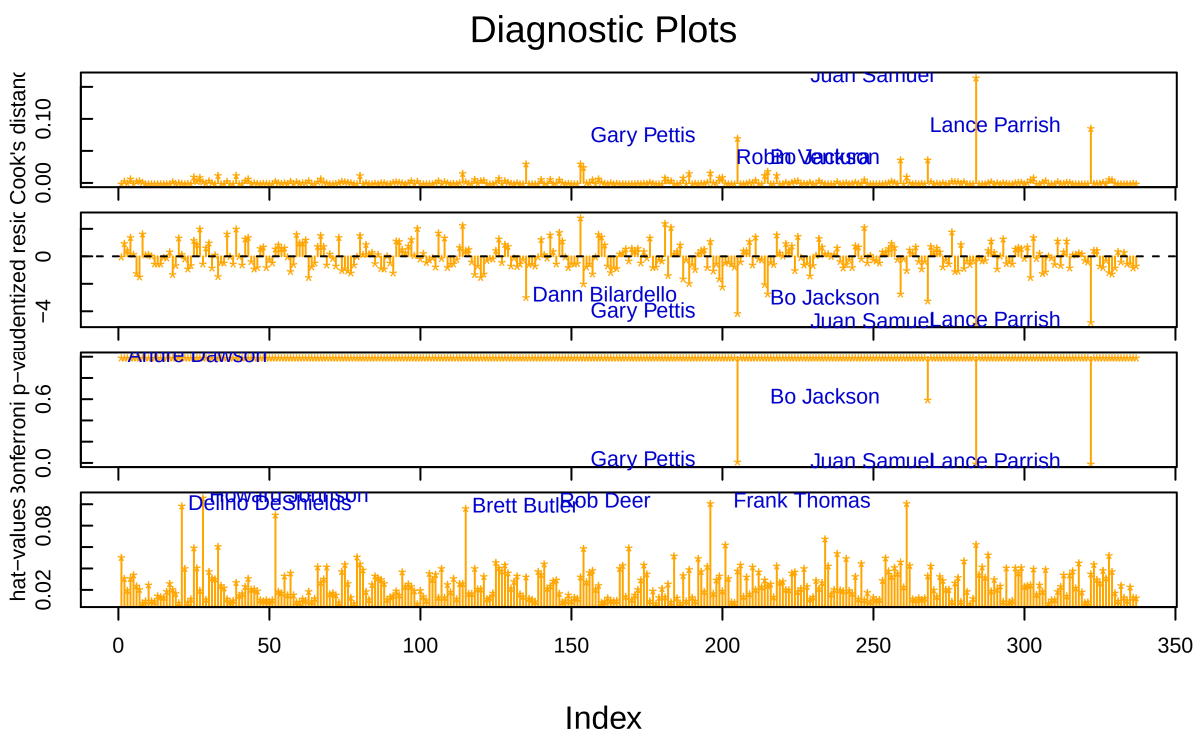

Figure 5: Diagnostic index plot for outlier detection.

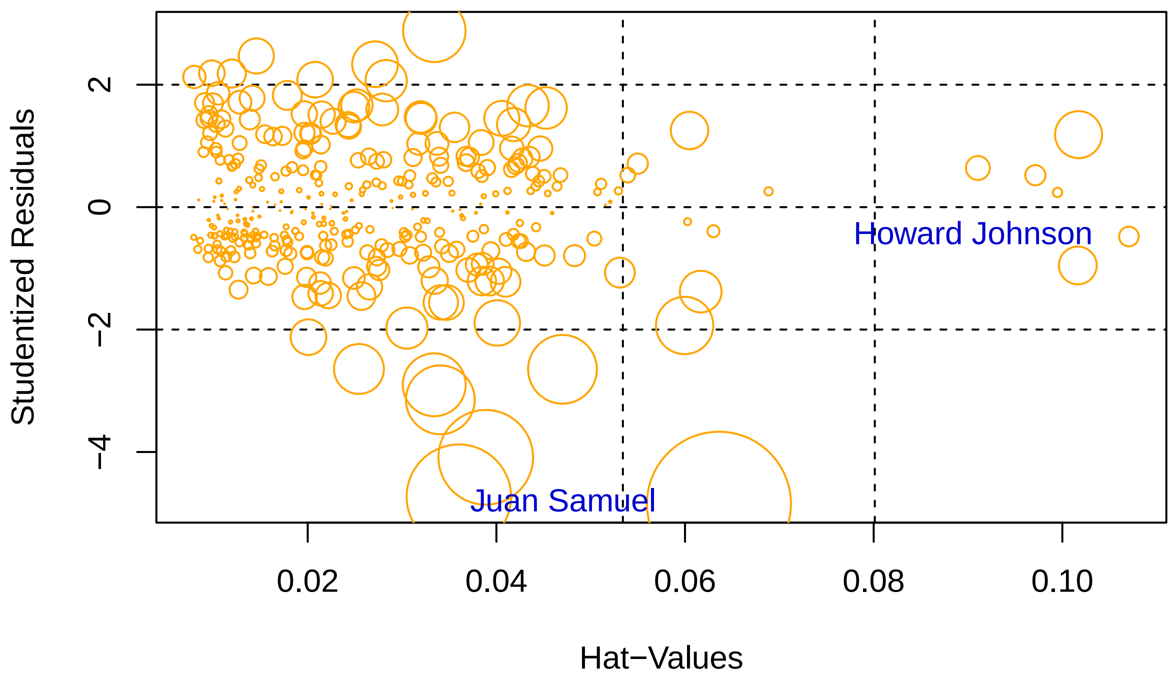

Figure 6: Bubble plot for outlier detection.

In Figure 5 we see that for the most part, the outliers are identical for Cook's distance, studentized residuals, and Bonferonni's test for outlier detection. However, the outliers from these methods differ from those identified using the hat values.

To compare these findings with the required bubble plot, we now create a bubble plot.

```
invisible(influencePlot(fit.final, col="orange", id.col="mediumblue",
                        labels=baseball$name))
```

Interestingly, only Juan Samuel from the three diagnostic measures previously discussed is the only one identified, and only Howard Johnson from the fourth (hat values), is identified as being outliers using the bubble plot.

3.6 Multicollinearity

Finally we seek to determine whether or not there exists multicollinearity with our predictors. We use both `kappa()` and `vif()` to do so.

```
kappa(lm(log.salary ~ hits + RBI + walks + strike.outs + errors +
                free.agency.elig + free.agent.91 + arb.elig - 1,
                data=baseball[,-18]))
```

```
## [1] 415.5786
```

```
vif(fit.final)
```

```
##             hits              RBI            walks      strike.outs
##         4.652468         5.050359         2.637470         2.527120
##           errors free.agency.elig    free.agent.91         arb.elig
##         1.325905         1.857650         1.317624         1.380452
```

Here we notice some conflicting results. Using `kappa()` with a cutoff of 100, we must conclude that multicollinearity is present, but using `vif()` with a cutoff of 10 we conclude that multicollinearity is not an issue. This is an interesting finding, as these rules-of-thumb disagree with each other.

4 Model deployment

Lastly, we will see how well our final model performs when given new data.

We first start by bringing the new data into `R` with the following code.

```r
test <- read.csv("~/Dropbox/Spring2018/STAT5474/bb92-test.csv", header=TRUE,
                 sep=",")
```

Next we construct prediction intervals for these new observations and then plot them with the following code.

```r
pred <- predict(fit.final, test, interval="prediction")
dat.plot <- data.frame(player=1:20, exp(pred))
library(ggplot2)
library(gridExtra)
plot0 <- ggplot(dat.plot, aes(x=player, y=fit)) + xlab("Player") +
         geom_errorbar(aes(ymin=lwr, ymax=upr)) + geom_point() +
         ylab("Salary") + ggtitle(label= "Salary (USD)") + theme_bw()
dat.plot1 <- data.frame(player=1:20, pred)
plot1 <- ggplot(dat.plot1, aes(x=player, y=fit)) + xlab("Player") +
         geom_errorbar(aes(ymin=lwr, ymax=upr)) + geom_point() +
         ylab("log.salary") + ggtitle(label="Log salary (USD)") + theme_bw()
grid.arrange(plot1, plot0, ncol=2)
```

In Figure 7 we see both the `log.salary` and `salary` prediction intervals for the players when fed into our final model. While there is considerable overlap between the intervals in the `log.salary` plot, this overlap seemingly disappears when we remove the logarithmic transformation as seen in the right plot, and we note that the symmetry of the error bars disappears.

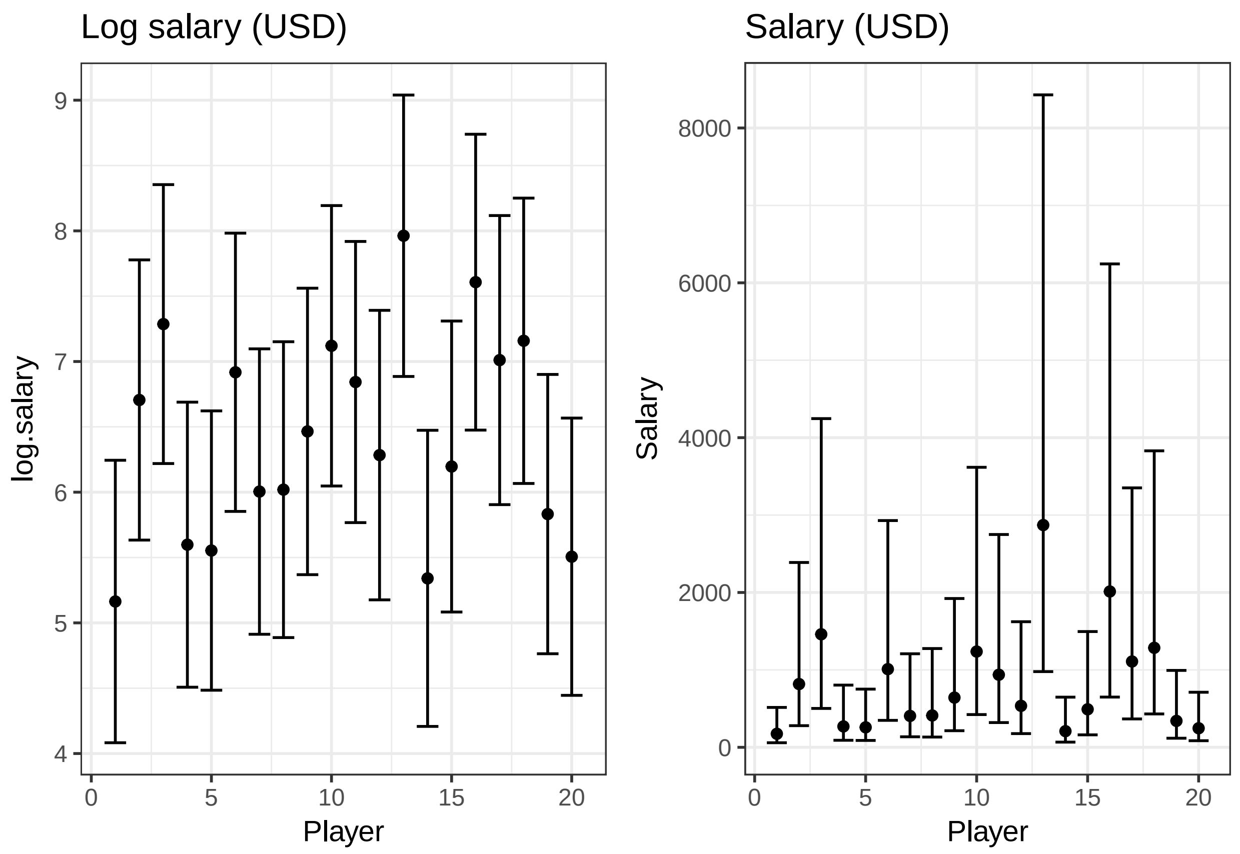

Figure 7: Log(salary) and salary for the players in our validation data set.

5 References

[0] Fox, J., and Weisberg, S. (2013), *An R companion to applied regression.* Sage: Thousand Oaks, California.

[1] Marin, J.M, and Robert, C.P. (2014), *Bayesian essentials with R.* Springer: New York, New York.

6 Appendix

6.1 R code and output used to inspect the data

```
n <- nrow(baseball)
out <- NULL
for (k in 1:ncol(baseball)) {
    vname <- colnames(baseball)[k]
    x <- as.vector(baseball[,k])
    n1 <- sum(is.na(x), na.rm=TRUE)
    n2 <- sum(x=="NA", na.rm=TRUE)
    n3 <- sum(x=='', na.rm=TRUE)
    nmiss <- n1 + n2 + n3
    ncomplete <- n - nmiss
    var.type <- typeof(x)
    if (var.type == "integer") {
        if (length(unique(x)) == 2) {
            out <- rbind(out, c(col.number=k, vname=vname, mode="binary",
                                n.levels=length(unique(x)), ncomplete=ncomplete,
                                miss.prop=nmiss/n))
        } else {
            out <- rbind(out, c(col.number=k, vname=vname, mode=typeof(x),
                                n.levels=length(unique(x)), ncomplete=ncomplete,
                                miss.prop=nmiss/n))
        }
    } else {
        out <- rbind(out, c(col.number=k, vname=vname, mode=typeof(x),
                            n.levels=length(unique(x)), ncomplete=ncomplete,
                            miss.prop=nmiss/n))
    }
}
out <- as.data.frame(out)
row.names(out) <- NULL
out
```

```
##   col.number        vname    mode n.levels ncomplete miss.prop
## 1          1   log.salary  double      208       337         0
## 2          2  batting.avg  double      133       337         0
## 3          3          OBP  double      139       337         0
## 4          4         runs integer      105       337         0
## 5          5         hits integer      163       337         0
## 6          6      doubles integer       44       337         0
## 7          7      triples integer       14       337         0
## 8          8     homeruns integer       36       337         0
## 9           9              RBI   integer      106        337           0
## 10         10            walks   integer       90        337           0
## 11         11      strike.outs   integer      118        337           0
## 12         12     stolen.bases   integer       48        337           0
## 13         13           errors   integer       28        337           0
## 14         14 free.agency.elig    binary        2        337           0
## 15         15     free.agent.91    binary        2        337           0
## 16         16         arb.elig    binary        2        337           0
## 17         17           arb.91    binary        2        337           0
## 18         18             name character      337        337           0
```

6.2 Output backward selection

```
baseball.back <- lm(log.salary ~. -name, data=baseball)
baseball.backward <- step(baseball.back, direction="backward")
```

```
## Start:  AIC=-402.87
## log.salary ~ (batting.avg + OBP + runs + hits + doubles + triples +
##     homeruns + RBI + walks + strike.outs + stolen.bases + errors +
##     free.agency.elig + free.agent.91 + arb.elig + arb.91 + name) -
##     name
##
##                    Df Sum of Sq     RSS     AIC
## - runs              1     0.018  92.196 -404.81
## - doubles           1     0.037  92.215 -404.74
## - arb.91            1     0.047  92.225 -404.70
## - batting.avg       1     0.053  92.231 -404.68
## - homeruns          1     0.056  92.234 -404.67
## - triples           1     0.292  92.470 -403.81
## - OBP               1     0.412  92.590 -403.37
## - stolen.bases      1     0.478  92.656 -403.13
## - errors            1     0.539  92.717 -402.91
## <none>                           92.178 -402.87
## - walks             1     0.649  92.827 -402.51
## - hits              1     1.074  93.252 -400.97
## - free.agent.91     1     1.715  93.893 -398.66
## - RBI               1     1.852  94.029 -398.17
## - strike.outs       1     3.116  95.293 -393.67
## - arb.elig          1    59.869 152.047 -236.22
## - free.agency.elig  1   103.376 195.554 -151.41
##
## Step:  AIC=-404.81
## log.salary ~ batting.avg + OBP + hits + doubles + triples + homeruns +
##     RBI + walks + strike.outs + stolen.bases + errors + free.agency.elig +
##     free.agent.91 + arb.elig + arb.91
##
##                     Df Sum of Sq     RSS     AIC
## - doubles            1     0.030  92.226 -406.70
## - arb.91             1     0.042  92.238 -406.66
## - batting.avg        1     0.052  92.247 -406.62
## - homeruns           1     0.107  92.303 -406.42
## - triples            1     0.276  92.471 -405.80
## - OBP                1     0.408  92.604 -405.32
## - errors             1     0.534  92.730 -404.86
## <none>                            92.196 -404.81
## - stolen.bases       1     0.708  92.904 -404.23
## - walks              1     0.909  93.104 -403.50
## - hits               1     1.622  93.818 -400.93
## - free.agent.91      1     1.719  93.914 -400.58
## - RBI                1     1.849  94.045 -400.12
## - strike.outs        1     3.328  95.524 -394.86
## - arb.elig           1    59.892 152.088 -238.13
## - free.agency.elig   1   103.421 195.617 -153.30
##
## Step:  AIC=-406.7
## log.salary ~ batting.avg + OBP + hits + triples + homeruns +
##     RBI + walks + strike.outs + stolen.bases + errors + free.agency.elig +
##     free.agent.91 + arb.elig + arb.91
##
##                     Df Sum of Sq     RSS     AIC
## - arb.91             1     0.044  92.270 -408.54
## - batting.avg        1     0.054  92.280 -408.50
## - homeruns           1     0.104  92.330 -408.32
## - triples            1     0.263  92.489 -407.74
## - OBP                1     0.416  92.642 -407.18
## - errors             1     0.520  92.746 -406.80
## <none>                            92.226 -406.70
## - stolen.bases       1     0.743  92.969 -405.99
## - walks              1     0.934  93.160 -405.30
## - free.agent.91      1     1.704  93.930 -402.53
## - RBI                1     1.819  94.045 -402.12
## - hits               1     1.881  94.107 -401.89
## - strike.outs        1     3.316  95.541 -396.80
## - arb.elig           1    59.869 152.095 -240.11
## - free.agency.elig   1   104.197 196.423 -153.92
##
## Step:  AIC=-408.54
## log.salary ~ batting.avg + OBP + hits + triples + homeruns +
##     RBI + walks + strike.outs + stolen.bases + errors + free.agency.elig +
##     free.agent.91 + arb.elig
##
##                    Df Sum of Sq     RSS     AIC
## - batting.avg       1     0.058  92.329 -410.32
## - homeruns          1     0.105  92.375 -410.15
## - triples           1     0.262  92.532 -409.58
## - OBP               1     0.431  92.701 -408.97
## - errors            1     0.515  92.786 -408.66
## <none>                           92.270 -408.54
## - stolen.bases      1     0.760  93.030 -407.77
## - walks             1     0.959  93.230 -407.05
## - free.agent.91     1     1.708  93.978 -404.36
## - RBI               1     1.831  94.101 -403.92
## - hits              1     1.844  94.115 -403.87
## - strike.outs       1     3.324  95.595 -398.61
## - arb.elig          1    63.957 156.227 -233.08
## - free.agency.elig  1   104.483 196.753 -155.35
##
## Step:  AIC=-410.32
## log.salary ~ OBP + hits + triples + homeruns + RBI + walks +
##     strike.outs + stolen.bases + errors + free.agency.elig +
##     free.agent.91 + arb.elig
##
##                    Df Sum of Sq     RSS     AIC
## - homeruns          1     0.105  92.434 -411.94
## - triples           1     0.278  92.607 -411.31
## - errors            1     0.534  92.863 -410.38
## <none>                           92.329 -410.32
## - stolen.bases      1     0.754  93.082 -409.58
## - OBP               1     0.942  93.270 -408.90
## - walks             1     1.283  93.612 -407.67
## - free.agent.91     1     1.690  94.018 -406.21
## - RBI               1     1.857  94.186 -405.61
## - hits              1     3.031  95.360 -401.44
## - strike.outs       1     3.495  95.823 -399.80
## - arb.elig          1    64.034 156.362 -234.78
## - free.agency.elig  1   105.782 198.110 -155.03
##
## Step:  AIC=-411.94
## log.salary ~ OBP + hits + triples + RBI + walks + strike.outs +
##     stolen.bases + errors + free.agency.elig + free.agent.91 +
##     arb.elig
##
##                    Df Sum of Sq     RSS     AIC
## - triples           1     0.358  92.792 -412.64
## <none>                           92.434 -411.94
## - errors            1     0.590  93.024 -411.80
## - stolen.bases      1     0.770  93.204 -411.14
## - OBP               1     0.884  93.318 -410.73
## - walks             1     1.235  93.669 -409.47
## - free.agent.91     1     1.844  94.278 -407.28
## - hits              1     3.076  95.510 -402.91
## - strike.outs       1     3.604  96.038 -401.05
## - RBI               1     6.132  98.566 -392.29
## - arb.elig          1    64.302 156.736 -235.98
## - free.agency.elig  1   109.342 201.776 -150.86
##
## Step:  AIC=-412.64
## log.salary ~ OBP + hits + RBI + walks + strike.outs + stolen.bases +
##     errors + free.agency.elig + free.agent.91 + arb.elig
##
##                    Df Sum of Sq     RSS     AIC
## - errors            1     0.501  93.293 -412.82
## - stolen.bases      1     0.527  93.319 -412.73
## <none>                           92.792 -412.64
## - OBP               1     0.884  93.676 -411.44
## - walks             1     1.430  94.222 -409.48
## - free.agent.91     1     1.924  94.716 -407.72
## - hits              1     2.732  95.523 -404.86
## - strike.outs       1     3.937  96.729 -400.63
## - RBI               1     6.942  99.734 -390.32
## - arb.elig          1    64.080 156.872 -237.69
## - free.agency.elig  1   110.541 203.332 -150.27
##
## Step:  AIC=-412.82
## log.salary ~ OBP + hits + RBI + walks + strike.outs + stolen.bases +
##     free.agency.elig + free.agent.91 + arb.elig
##
##                    Df Sum of Sq     RSS     AIC
## <none>                           93.293 -412.82
## - stolen.bases      1     0.611  93.904 -412.62
## - OBP               1     0.820  94.112 -411.88
## - walks             1     1.443  94.736 -409.65
## - hits              1     2.263  95.556 -406.75
## - free.agent.91     1     2.271  95.564 -406.72
## - strike.outs       1     4.418  97.710 -399.23
## - RBI               1     7.800 101.092 -387.76
## - arb.elig          1    63.580 156.872 -239.69
## - free.agency.elig  1   114.716 208.009 -144.60
```

6.3 Bayes output

```r
library(bayess)
set.seed(5474)
ModChoBayesReg(y=training.data[, 1], X=training.data[, -c(1, 18)],
               g=length(training.data[, 1]))
```

```
##
## Number of variables greather than 15
## Model posterior probabilities are estimated by using an MCMC algorithm
##
##           Top10Models PostProb
## 1       3 8 10 13 15   0.0552
## 2     4 8 9 10 13 15   0.0496
## 3       4 8 10 13 15   0.0457
## 4       4 8 12 13 15   0.0426
## 5    4 8 10 12 13 15   0.0350
## 6  4 8 9 10 12 13 15   0.0282
## 7          4 8 13 15   0.0220
## 8    4 8 10 13 14 15   0.0182
## 9  4 8 9 10 13 14 15   0.0150
## 10   3 8 10 13 14 15   0.0138
```

```
## $top10models
##  [1] "3 8 10 13 15"       "4 8 9 10 13 15"    "4 8 10 13 15"
##  [4] "4 8 12 13 15"       "4 8 10 12 13 15"   "4 8 9 10 12 13 15"
##  [7] "4 8 13 15"          "4 8 10 13 14 15"   "4 8 9 10 13 14 15"
## [10] "3 8 10 13 14 15"
##
## $postprobtop10
##  [1] 0.0552500 0.0495750 0.0457000 0.0426000 0.0350125 0.0281625 0.0219750
##  [8] 0.0181875 0.0149750 0.0138125
```