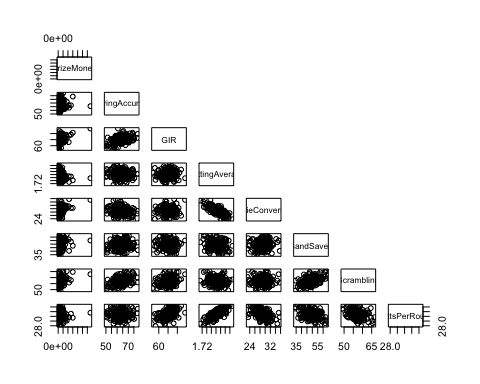
hwk5\_601

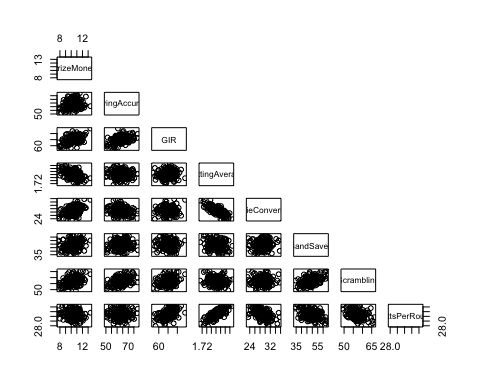
## 3

### a

d<-read.csv("pgatour2006.csv",header = T)  
dd<-d[-c(1,2,4,11)]  
  
pairs(dd,upper.panel = NULL)



ddd<-dd  
ddd$PrizeMoney<-log(dd$PrizeMoney)  
pairs(ddd, upper.panel = NULL)



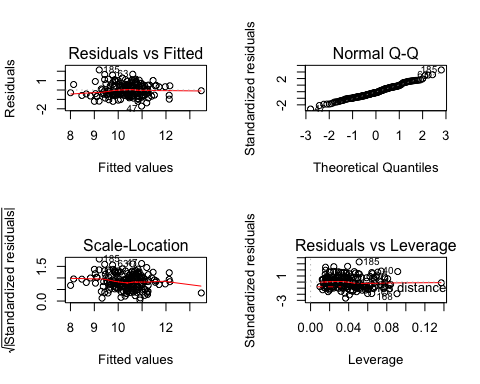
I think it is appropriate to transform Y using the log transformation, because the scatter plots between the log(y) and other independent variables show much clear relationship.

### b

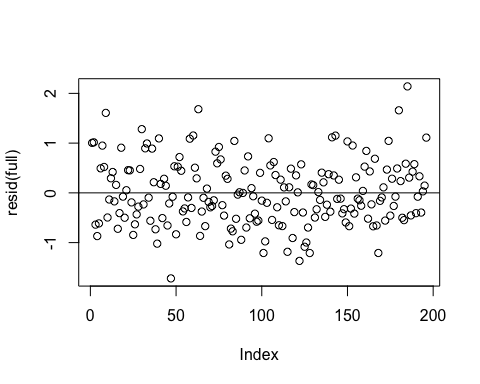
full<-lm(log(PrizeMoney)~.,data = dd)  
summary(full)

##   
## Call:  
## lm(formula = log(PrizeMoney) ~ ., data = dd)  
##   
## Residuals:  
## Min 1Q Median   
## -1.719485607964426 -0.486081222974908 -0.091716354686494   
## 3Q Max   
## 0.445610967666511 2.140134211915514   
##   
## Coefficients:  
## Estimate Std. Error t value  
## (Intercept) 0.1943002560626678 7.7771287796624202 0.02498  
## DrivingAccuracy -0.0035298595248293 0.0117727661520025 -0.29983  
## GIR 0.1993108708977513 0.0438168744060051 4.54872  
## PuttingAverage -0.4663038342259676 6.9056982855560767 -0.06752  
## BirdieConversion 0.1573408976045031 0.0403781881704152 3.89668  
## SandSaves 0.0151743845907787 0.0098616182439200 1.53873  
## Scrambling 0.0515136658168674 0.0317875604955650 1.62056  
## PuttsPerRound -0.3431314025101285 0.4735493257169385 -0.72459  
## Pr(>|t|)   
## (Intercept) 0.98009459   
## DrivingAccuracy 0.76463607   
## GIR 9.6584e-06 \*\*\*  
## PuttingAverage 0.94623592   
## BirdieConversion 0.00013551 \*\*\*  
## SandSaves 0.12555120   
## Scrambling 0.10678800   
## PuttsPerRound 0.46960144   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.050000000000000003 '.'  
## 0.10000000000000001 ' ' 1  
##   
## Residual standard error: 0.66390828855397999 on 188 degrees of freedom  
## Multiple R-squared: 0.55770869526747, Adjusted R-squared: 0.54124040200615   
## F-statistic: 33.86560382535 on 7 and 188 DF, p-value: < 2.22044604925e-16

par(mfrow=c(2,2))  
plot(full, ask = F)

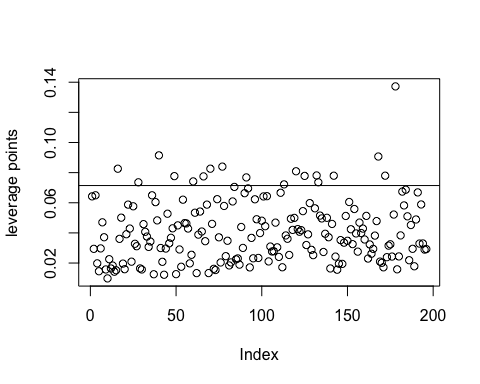


par(mfrow=c(1,1))  
  
plot(resid(full))  
abline(h=mean(resid(full)))



### c

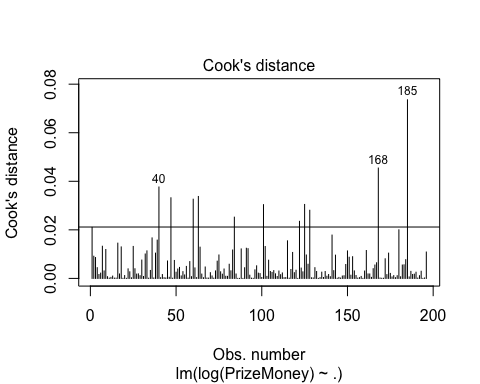
x<-model.matrix(full)  
lev<-hat(x)  
plot(lev,ylab = "leverage points")  
abline(h=(2\*7)/196)



which(lev>(14/196))#leverage points

## [1] 16 28 40 49 60 66 70 77 91 113 120 125 132 133 142 168 172  
## [18] 178

plot(full,which = 4)#influential points  
abline(h=4/(196-7))  
  
library(car)



outlierTest(full)#take 185th point as outliers

##   
## No Studentized residuals with Bonferonni p < 0.050000000000000003  
## Largest |rstudent|:  
## rstudent unadjusted p-value Bonferonni p  
## 185 3.4007416968548507 0.00082164 0.16103999999999999

### d

weakness:  
- not all the independent variables are significant.

### e

It is dangerous to remove all the insignificant variables at one time, because it may remove some potential significant variables. There are several reasons may cause a bunch of variables including important ones to be insignificant, say multi-colinearity.

## 6

You do need to generate 10,000 columns of X, but only a few s are non zero to generate Y. The idea is to regress Y on all X columns, but eventually you find those/some/none of non zero s. This problem is a low sample high dimension problem. Just clearly describe your data generating procedure, your data fitting procedure, and what you find.(Pro.Zhang)

### n=500

n=50 #sample size  
p=10000 # dimension  
mu<-sample(x = 1:p,size = p)# different generation dist's mean  
x<-sapply(mu, function(x) rnorm(n,mean = x,sd = 3))# x  
  
beta\_t<-rep(0,p)  
beta\_t[1:3]<- c(1,2,3)#true beta   
  
muu<-2+x%\*%beta\_t  
d<- as.data.frame(x)  
yy<-lapply(seq(100),function(x) return(muu+rnorm(n)))  
#y=2 +x1 +2\*x2 +3\*x3 +epsilon(~N(0,1))

Data generation is described above in the comment.

full\_fit<-function(y){  
 d<- cbind(y,d)   
 full<-lm(y~.,data = d[1:4])# true fitting  
 return(full$coefficients)  
}  
beta0123 <- sapply(yy, FUN = full\_fit);beta0123[,c(1,100)]

## [,1] [,2]  
## (Intercept) -640.1007982760287405 317.86694816718357970  
## V1 1.0559353813521584 0.99845396718643342  
## V2 2.0421097857029475 2.11372663814308970  
## V3 3.0528462714677955 2.92312364530748425

(sample.mean <- rowMeans(beta0123)) #sample mean

## (Intercept) V1 V2   
## -23.6623571734490881 1.0001416845752715 1.9993869191499245   
## V3   
## 3.0033824953361079

(bias<-sample.mean-c(2,1,2,3))#bias

## (Intercept) V1 V2   
## -2.5662357173449088e+01 1.4168457527152789e-04 -6.1308085007549806e-04   
## V3   
## 3.3824953361079046e-03

(sample.var <- apply(beta0123, 1, var))#sample variance of true model's estimate

## (Intercept) V1 V2   
## 2.2920689675286494e+05 2.3620241948179056e-03 2.3068520859239943e-03   
## V3   
## 3.2456321290758152e-03

under\_fit<-function(y){  
 d<- cbind(y,d)   
 under<-lm(y~.,data = d[1:3])# underfitting  
 result<-summary(under)  
 return(result$coefficients[,1])  
}  
beta0123<-sapply(yy, FUN = under\_fit);beta0123[,c(1,100)]

## [,1] [,2]  
## (Intercept) 2.3479814532527209e+04 2.3412870284966943e+04  
## V1 8.4202059001546292e-01 7.9362891900715848e-01  
## V2 2.2941342362220145e+00 2.3550419763548986e+00

(sample.mean <- rowMeans(beta0123))

## (Intercept) V1 V2   
## 2.3705449773551034e+04 7.8969285005691514e-01 2.2473279406680962e+00

(bias<-sample.mean-c(2,1,2))#bias

## (Intercept) V1 V2   
## 23703.44977355103401351 -0.21030714994308486 0.24732794066809616

(sample.var <- apply(beta0123, 1, var))#variance of under-fitting model estimate

## (Intercept) V1 V2   
## 2.6459982083406689e+04 2.3720554511660751e-03 2.3079129753469092e-03

for under-fitting, its estimate is biased but with smaller variance.

new<-rt(n,df=1,ncp = 3) #overfitting  
d.new<-cbind(new,d)#new predictor  
over\_fit<-function(y){  
 d<- cbind(y,d.new)   
 under<-lm(y~.,data = d)  
 result<-summary(under)  
 return(result$coefficients[c(1,3,4,5,6),1])  
}  
beta0123<-sapply(yy, FUN = over\_fit);beta0123[,c(1,100)]

## [,1] [,2]  
## (Intercept) -642.44361259237450668 5373.69511333788614138  
## V1 1.08236110433716282 0.84917544962505476  
## V2 1.88511698707705855 1.99791337881887565  
## V3 2.65666621095188304 3.22324076679558225  
## V4 -0.30554702305115694 0.51812690190349087

(sample.mean <- rowMeans(beta0123)) #sample mean

## (Intercept) V1 V2   
## -5.9202973152282118e+02 9.8824351565447943e-01 1.9480885757139066e+00   
## V3 V4   
## 2.9348247795460916e+00 -1.9254767450908538e-04

(bias<-sample.mean-c(2,1,2,3,0))#bias

## (Intercept) V1 V2   
## -5.9402973152282118e+02 -1.1756484345520568e-02 -5.1911424286093411e-02   
## V3 V4   
## -6.5175220453908445e-02 -1.9254767450908538e-04

(sample.var <- apply(beta0123, 1, var))#sample variance of over-fitting model's estimate

## (Intercept) V1 V2   
## 3.5461230122821140e+08 7.0599348628453948e-02 6.0838362326663099e-02   
## V3 V4   
## 1.2424074533947811e-01 1.2596543389810286e-01

for over-fitting, its estimate is unbiased but with bigger variance.

## 8

job <- read.table("jobdata.txt", sep='',col.names=c("Y","X1","X2","X3","X4"))  
str(job)

## 'data.frame': 25 obs. of 5 variables:  
## $ Y : num 88 80 96 76 80 73 58 116 104 99 ...  
## $ X1: num 86 62 110 101 100 78 120 105 112 120 ...  
## $ X2: num 110 97 107 117 101 85 77 122 119 89 ...  
## $ X3: num 100 99 103 93 95 95 80 116 106 105 ...  
## $ X4: num 87 100 103 95 88 84 74 102 105 97 ...

f<-lm(Y~.,data = job)  
step(f,direction = "forward", trace = FALSE)

##   
## Call:  
## lm(formula = Y ~ X1 + X2 + X3 + X4, data = job)  
##   
## Coefficients:  
## (Intercept) X1 X2   
## -124.381820576101745 0.295725374481953 0.048287717548395   
## X3 X4   
## 1.306011003903357 0.519819090192481

step(f,direction = "backward", trace = FALSE)

##   
## Call:  
## lm(formula = Y ~ X1 + X3 + X4, data = job)  
##   
## Coefficients:  
## (Intercept) X1 X3   
## -124.20001657151947 0.29632599067880 1.35696754244408   
## X4   
## 0.51742114381702

step(f,direction = "both", trace = FALSE)

##   
## Call:  
## lm(formula = Y ~ X1 + X3 + X4, data = job)  
##   
## Coefficients:  
## (Intercept) X1 X3   
## -124.20001657151947 0.29632599067880 1.35696754244408   
## X4   
## 0.51742114381702

I personally tend to choose the model selected by the stepwise selection. Because the method of stepwise will test more combinations of the variables, while the forward selection or backward selection will all omit more potential combination of the independent variables.

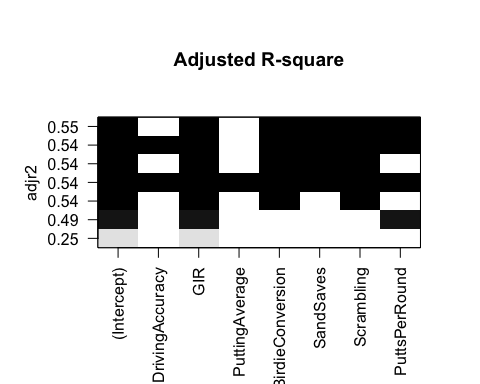
## 9

### a

d<-read.csv("pgatour2006.csv",header = T);head(d)

## Name TigerWoods PrizeMoney AveDrivingDistance  
## 1 Aaron Baddeley 0 60661 288.30000000000001  
## 2 Adam Scott 0 262045 301.10000000000002  
## 3 Alex Aragon 0 3635 302.60000000000002  
## 4 Alex Cejka 0 17516 288.80000000000001  
## 5 Arjun Atwal 0 16683 287.69999999999999  
## 6 Arron Oberholser 0 107294 285.00000000000000  
## DrivingAccuracy GIR PuttingAverage  
## 1 60.729999999999997 58.259999999999998 1.7450000000000001  
## 2 62.000000000000000 69.120000000000005 1.7669999999999999  
## 3 51.119999999999997 59.109999999999999 1.7869999999999999  
## 4 66.400000000000006 67.700000000000003 1.7769999999999999  
## 5 63.240000000000002 64.040000000000006 1.7609999999999999  
## 6 62.530000000000001 69.269999999999996 1.7749999999999999  
## BirdieConversion SandSaves Scrambling  
## 1 31.359999999999999 54.799999999999997 59.369999999999997  
## 2 30.390000000000001 53.609999999999999 57.939999999999998  
## 3 29.890000000000001 37.930000000000000 50.780000000000001  
## 4 29.329999999999998 45.130000000000003 54.820000000000000  
## 5 29.320000000000000 52.439999999999998 57.070000000000000  
## 6 29.199999999999999 47.200000000000003 57.670000000000002  
## BounceBack PuttsPerRound  
## 1 19.300000000000001 27.960000000000001  
## 2 19.350000000000001 29.280000000000001  
## 3 16.800000000000001 29.199999999999999  
## 4 17.050000000000001 29.460000000000001  
## 5 18.210000000000001 28.930000000000000  
## 6 20.000000000000000 29.559999999999999

dd<-d[-c(1,2,4,11)]  
  
library(leaps)  
opt<-regsubsets(log(PrizeMoney)~.,data = dd,  
 nbest = 1, # 1 best model for each number of predictors  
 nvmax = NULL, # NULL for no limit on number of variables  
 force.in = NULL, force.out = NULL,  
 method = "exhaustive")  
par(mfrow=c(1,1))  
plot(opt, scale = "adjr2", main = "Adjusted R-square")



according to the plot, the optimal model by adjusted R-square should be

library(bestglm)  
library(dplyr)

##   
## Attaching package: 'dplyr'  
##   
## The following objects are masked from 'package:stats':  
##   
## filter, lag  
##   
## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

dd<-mutate(y=log(PrizeMoney),.data = dd)  
dd<- select(dd,-PrizeMoney)  
  
aic<-bestglm(Xy = dd,  
 family = gaussian,  
 IC = "AIC",   
 method = "exhaustive")   
aic$BestModel

##   
## Call:  
## lm(formula = y ~ ., data = data.frame(Xy[, c(bestset[-1], FALSE),   
## drop = FALSE], y = y))  
##   
## Coefficients:  
## (Intercept) GIR BirdieConversion   
## -0.583180705459968 0.197022121114633 0.162752405776287   
## SandSaves Scrambling PuttsPerRound   
## 0.015524101092826 0.049634716223792 -0.349738493026655

according to the optimal model by AIC, it is the same as produced by the adjusted R-square.

bic<-bestglm(Xy = dd,  
 family = gaussian,  
 IC = "BIC",   
 method = "exhaustive")   
bic$BestModel

##   
## Call:  
## lm(formula = y ~ ., data = data.frame(Xy[, c(bestset[-1], FALSE),   
## drop = FALSE], y = y))  
##   
## Coefficients:  
## (Intercept) GIR BirdieConversion   
## -11.083136410894085 0.156581267911773 0.206245581531053   
## Scrambling   
## 0.091779790709023

according to the optimal model by BIC, it is

### b

aic<-bestglm(Xy = dd,  
 family = gaussian,  
 IC = "AIC",   
 method = "backward")   
aic$BestModel

##   
## Call:  
## lm(formula = y ~ ., data = data.frame(Xy[, c(bestset[-1], FALSE),   
## drop = FALSE], y = y))  
##   
## Coefficients:  
## (Intercept) GIR BirdieConversion   
## -0.583180705459968 0.197022121114633 0.162752405776287   
## SandSaves Scrambling PuttsPerRound   
## 0.015524101092826 0.049634716223792 -0.349738493026655

bic<-bestglm(Xy = dd,  
 family = gaussian,  
 IC = "BIC",   
 method = "backward")   
bic$BestModel

##   
## Call:  
## lm(formula = y ~ ., data = data.frame(Xy[, c(bestset[-1], FALSE),   
## drop = FALSE], y = y))  
##   
## Coefficients:  
## (Intercept) GIR BirdieConversion   
## -11.083136410894085 0.156581267911773 0.206245581531053   
## Scrambling   
## 0.091779790709023

### c

aic<-bestglm(Xy = dd,  
 family = gaussian,  
 IC = "AIC",   
 method = "forward")   
aic$BestModel

##   
## Call:  
## lm(formula = y ~ ., data = data.frame(Xy[, c(bestset[-1], FALSE),   
## drop = FALSE], y = y))  
##   
## Coefficients:  
## (Intercept) GIR BirdieConversion   
## -0.583180705459968 0.197022121114633 0.162752405776287   
## SandSaves Scrambling PuttsPerRound   
## 0.015524101092826 0.049634716223792 -0.349738493026655

bic<-bestglm(Xy = dd,  
 family = gaussian,  
 IC = "BIC",   
 method = "forward")   
bic$BestModel

##   
## Call:  
## lm(formula = y ~ ., data = data.frame(Xy[, c(bestset[-1], FALSE),   
## drop = FALSE], y = y))  
##   
## Coefficients:  
## (Intercept) GIR BirdieConversion   
## 0.393202628851585 0.193518307354211 0.165893663676124   
## Scrambling PuttsPerRound   
## 0.062820010510926 -0.378397142876483

In my point of view, it is a coincidence that a and b are the same, because they could be different. I don't know how to explain carefully, but it is easy to think simply that if the optimal model from all possible options is within the step selecion process, the result should be the same like a and b. Otherwise the results are different like a and c.

### e

since the size of sample is over 100, so I think we should rely on the unbiased criteria **BIC**. Besides, since we can afford the cost of examing all possible models, we don't need step selection. As a result, I prefer the model

.

The choice above is only based on the statistics, more careful consideration may be taken into account by some experts in this game(Frankly,I know nothing about it).

### f

since the response variable is the log function of PrizeMoney, we can interpret the model like when GIR increases by one unit the PrizeMoney will increase by 17% in average. Likewise, when BirdieConversion or Scrambling increases by one unit the PrizeMoney will increase by 21% or 9.25% in average