Regression Subset Selection in R

For linear regression: lm() from stats package (built-in) For logistic regression: glm() from stats package

For lasso and ridge regression: glmnet() from glmnet package

Load packages

```
library(glmnet)
library(QuantPsyc) # for standardized regression coefficients
library(subselect)
#library(lars)
library(tabuSearch)
```

Additionally

```
library(lavaan) # for HolzingerSwineford1939 dataset
HS <- HolzingerSwineford1939

library(elasticnet) # regularized PCA
#library(fanc) # regularized FA
#library(FAiR) # semi-exploratory factor analysis; only works on Windows
library(GA) # genetic algorithm for subset selection
```

http://www.jstatsoft.org/v53/i04/paper

You can also embed plots, for example:

```
data(diabetes)
X <- diabetes$x
Y <- diabetes$y
```

```
lm(Y ~ X)
# note: equivalent to
lm(y ~ x,data=diabetes)
# which is equivalent to
lm(diabetes$y ~ diabetes$x)
```

Note, in this dataset, x is essentially a matrix within a dataframe. This is a little unusual, where the typical format would be:

```
diabetes2 <- data.frame(cbind(Y,X))
head(diabetes2)
```

```
##     Y          age         sex         bmi          map           tc
## 1 151  0.038075906  0.05068012  0.06169621  0.021872355 -0.044223498
## 2  75 -0.001882017 -0.04464164 -0.05147406 -0.026327835 -0.008448724
## 3 141  0.085298906  0.05068012  0.04445121 -0.005670611 -0.045599451
## 4 206 -0.089062939 -0.04464164 -0.01159501 -0.036656447  0.012190569
## 5 135  0.005383060 -0.04464164 -0.03638469  0.021872355  0.003934852
## 6  97 -0.092695478 -0.04464164 -0.04069594 -0.019442093 -0.068990650
```

```
##           ldl          hdl          tch          ltg          glu
## 1 -0.03482076 -0.043400846 -0.002592262  0.019908421 -0.017646125
## 2 -0.01916334  0.074411564 -0.039493383 -0.068329744 -0.092204050
## 3 -0.03419447 -0.032355932 -0.002592262  0.002863771 -0.025930339
## 4  0.02499059 -0.036037570  0.034308859  0.022692023 -0.009361911
## 5  0.01559614  0.008142084 -0.002592262 -0.031991445 -0.046640874
## 6 -0.07928784  0.041276824 -0.076394504 -0.041180385 -0.096346157
```

```
lm(Y ~ ., data=diabetes2)
```

```
##
## Call:
## lm(formula = Y ~ ., data = diabetes2)
##
## Coefficients:
## (Intercept)          age          sex          bmi          map
##      152.13       -10.01      -239.82       519.84       324.39
##          tc          ldl          hdl          tch          ltg
##     -792.18       476.75       101.04       177.06       751.28
##         glu
##       67.63
```

Using the "." means we want to use all variables that aren't the outcome as predictors
Fun Fact: Can do the same thing with a SEM package

```
library(lavaan)
lm.mod <-'
Y ~ age + sex + bmi + map + tc + ldl + hdl + tch + ltg + glu
Y ~ 1 # intercept
'
lm.sem <- sem(lm.mod,diabetes2)
#summary(lm.sem)
#parameterEstimates(lm.sem)
coef(lm.sem)
```

```
##    Y~age    Y~sex    Y~bmi    Y~map     Y~tc    Y~ldl    Y~hdl    Y~tch
##  -10.012 -239.819  519.840  324.390 -792.184  476.746  101.045  177.064
##    Y~ltg    Y~glu      Y~1     Y~~Y
##  751.279   67.625  152.133 2859.690
```

Exact same answer.

```
lm.out <- lm(y ~ x,data=diabetes)
#summary(lm.out)

# check assumptions
#plot(lm.out)

# get standardized coefficients
lm.beta(lm.out) # from QuantPsyc
```

```
##         xage         xsex         xbmi         xmap          xtc
## -0.006178065 -0.147981279  0.320769113  0.200166344 -0.488820243
##         xldl         xhdl         xtch         xltg         xglu
##  0.294177828  0.062349936  0.109258122  0.463579756  0.041728501
```

So we are doing pretty good, R^2 of 0.51, with only four significant predictors. So the question we are going to answer today is whether we can get rid of a few predictors and still do a good job of predicting the outcome. Now, there are two + reasons to do this:

1. In future studies, maybe time is of the essence, or each additional question costs a certain amount of money. By reducing the number of questions we have to ask participants, both money and time can be saved. The question is what is the tradeoff, can we reduce the number of scales/items/questionnaires, and still answer the questions we want?
2. Remember when using R^2 as a criterion, by using more variables as predictors, these can only improve are within sample predictive power. But when we become concerned with generalizability, then in some cases, a reduced number of predictors, only important ones, can generalize better than a larger set of X's. This was somewhat demonstrated in the "preprocessing" lab.

Let's try #2 on the diabetes dataset

```
ids <- sample(1:nrow(diabetes2), .5*nrow(diabetes2),replace=FALSE)
diab.train <- diabetes2[ids,]
diab.test <- diabetes2[-ids,]

lm.trainFull <- lm(Y ~ ., data= diab.train)
summary(lm.trainFull)$r.squared
```

```
## [1] 0.5030062
```

```
lm.trainSub <- lm(Y ~ sex + bmi + map + ltg, data= diab.train)
summary(lm.trainSub)$r.squared
```

```
## [1] 0.4580293
```

```
pred.full <- predict(lm.trainFull,diab.test)
pred.sub <- predict(lm.trainSub,diab.test)

cor(pred.full,diab.test$Y)**2
```

```
## [1] 0.510087
```

```
cor(pred.sub,diab.test$Y)**2
```

```
## [1] 0.508507
```

Not in this case, but let's try some different methods specifically designed for subset selection.

Subset Selection

First off, why don't we just try out all combination of predictors – entering them all separately into lm()? Problems:

1. How do we choose. R^2 can only go up with added predictors (RSS can only go down).
2. This is usually computationally infeasible, as there are 2^p possible models, where p is the number of predictors. In our case $2^{10} = 1024$, which is a lot but not too many.

Stepwise Selection

Forward

Efficient, but not guaranteed to find best overall model.

```
library(MASS)
set.seed(1034)
lmNULL <- lm(Y ~ 1, data=diab.train)
lmFULL <- lm(Y ~ ., data=diab.train)

stepFor <- stepAIC(lmNULL,scope=list(lower=lmNULL,upper=lmFULL),
                   direction="forward")
```

```
stepFor$anova
```

```
## Stepwise Model Path
## Analysis of Deviance Table
##
## Initial Model:
## Y ~ 1
##
## Final Model:
## Y ~ bmi + ltg + map + hdl + sex + glu
##
##
##     Step Df   Deviance Resid. Df Resid. Dev      AIC
## 1                           220  1300670.2 1920.330
## 2 + bmi   1 394504.687       219   906165.5 1842.458
## 3 + ltg   1 143555.736       218   762609.7 1806.341
## 4 + map   1  50917.121       217   711692.6 1793.070
## 5 + hdl   1  16872.125       216   694820.5 1789.767
## 6 + sex   1  26763.827       215   668056.7 1783.086
## 7 + glu   1   7194.799       214   660861.9 1782.693
```

```
pred.for<- predict(stepFor,diab.test)

cor(pred.for,diab.test$Y)**2
```

```
## [1] 0.5192895
```

AIC (Akaike Information Criterion) induces a penalty for complexity – meaning that it will try and choose a model that balances predictive accuracy with simplicity (less predictors).

Backward

```
library(MASS)
stepBack <- stepAIC(lmFULL,direction="backward")
```

```
stepBack$anova
```

```
## Stepwise Model Path
## Analysis of Deviance Table
##
## Initial Model:
## Y ~ age + sex + bmi + map + tc + ldl + hdl + tch + ltg + glu
##
## Final Model:
## Y ~ sex + bmi + map + tc + ldl + ltg + glu
##
##
##    Step Df    Deviance Resid. Df Resid. Dev      AIC
## 1                           210   646425.0 1785.812
## 2 - tch  1    5.878708       211   646430.8 1783.814
## 3 - age  1   65.659495       212   646496.5 1781.836
## 4 - hdl  1 1687.132174       213   648183.6 1780.412
```

```
pred.back<- predict(stepBack,diab.test)

cor(pred.back,diab.test$Y)**2
```

```
## [1] 0.5171596
```

Ridge and Lasso Regression

- Including a penalty on the β parameters, and by varying the penalty we can shrink some of the $\beta's$ to zero, doing a form of "automatic" subset selection.

Althought there a number of packages to do this, maybe the best is *glmnet*

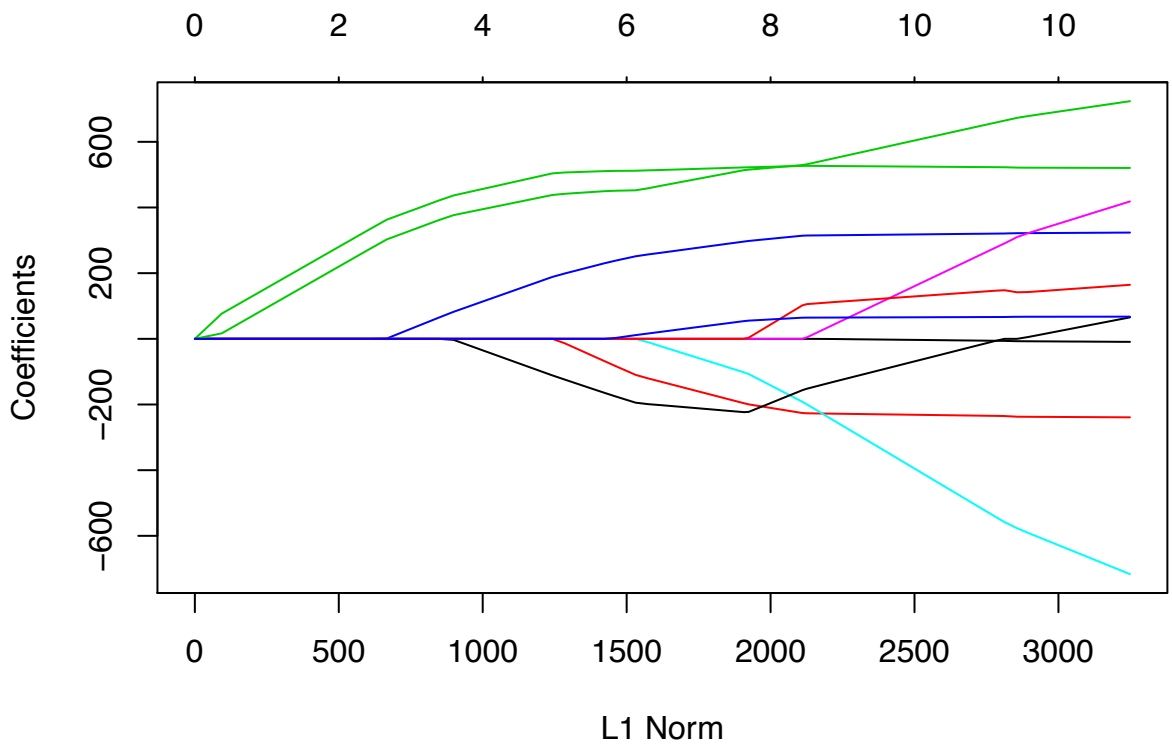
Note, for glmnet, your data has to be set up in two separate matrices. Doing this can be accomplished by:

```
YY <- as.matrix(diabetes2$Y)
XX <- as.matrix(diabetes2[,2:11])
# or
XX <- as.matrix(diabetes2[,c("age","sex","bmi","map","tc","ldl","hdl","tch","ltg","glu")])
```

Two things to note: 1. Because we are doing regression with a continuous outcome, we specify the family(distribution) as "gaussian" 2. Shrinkage in lasso and ridge is sensitive to the scale of the variables, therefore, it is best to standardize the predictors before entering. glmnet does this by default (look at ?glmnet).

Lasso

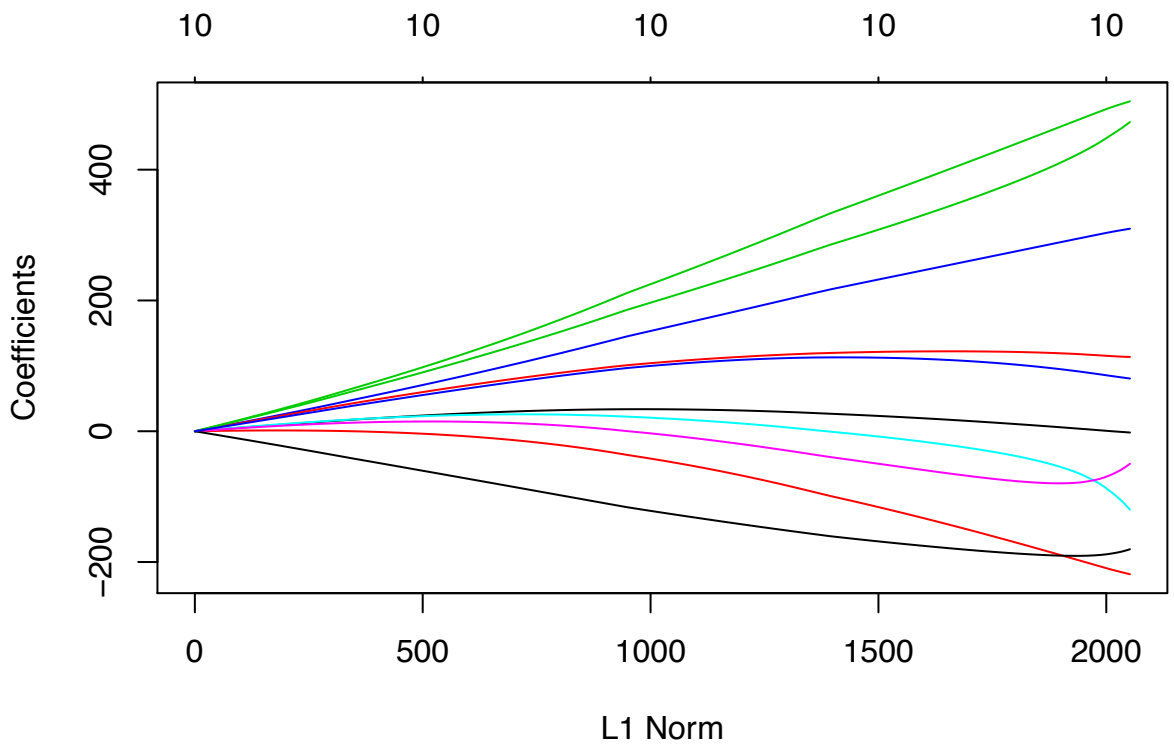
```
?glmnet
lasso.out <- glmnet(XX,YY,family="gaussian",alpha=1)
plot(lasso.out)
```




```
#gaussian for continuous outcomes, "binomial" for categorical
# alpha=1 is lasso, alpha=0 is ridge
```

Ridge

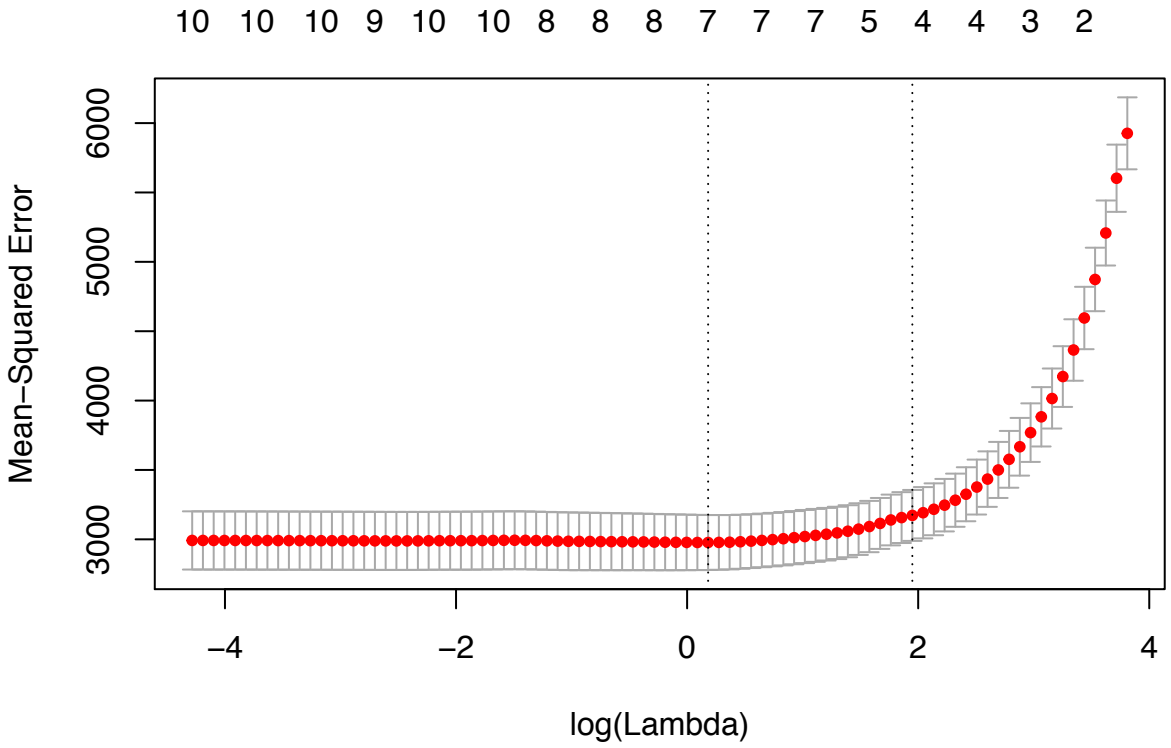
```
ridge.out <- glmnet(XX,YY,family="gaussian",alpha=0)
#plot(ridge.out,type.coef="2norm")
plot(ridge.out)
```



Since ridge regression does not shrink the β coefficients to 0 with increase penalization, it does not do an "automatic" form of subset selection

The problem now becomes, which value of λ (amount of shrinkage) do we choose? Using cross-validation is one of the better ways, and is implemented the glmnet package

```
cv.lasso <- cv.glmnet(XX,YY,family="gaussian",alpha=1)
plot(cv.lasso)
```



Two-strategies for selecting λ: either pick the lowest CV error, or the best solution within 1 standard error. I don't think that there is a clear best choice. The one advantage of using the 1SE rule is that you need fewer predictors.In our example 4 instead of 7.

```
#str(cv.lasso)
(lmin <- cv.lasso$lambda.min)
```

```
## [1] 1.19949
```

```
(lminSE <- cv.lasso$lambda.1se)
```

```
## [1] 7.025438
```

```
lasso.out2 = glmnet(XX,YY,family="gaussian",alpha=1,lambda=lminSE)
lasso.out2
```

```
##
## Call:  glmnet(x = XX, y = YY, family = "gaussian", alpha = 1, lambda = lminSE)
##
##      Df   %Dev Lambda
## [1,]  4 0.4753  7.025
```

Note that Df correspond to the number of non-zero $\beta's$
So how are we doing?

```
# ?predict.glmnet
pred.1se <- predict(lasso.out2,XX)
cor(pred.1se,YY)**2
```

```
##         [,1]
## s0 0.4891983
```

So with only 4 predictors entered into the model, we only lose 2-3% of our predicted variance(R^2).
With the lasso, there are two recommended strategies for using the results. 1. Taking the predictors with non-zero $\beta's$, and just using that subset in linear regression. 2. Or, bypass this all together and use least angle regression.

"One approach for reducing this bias is to run the lasso to identify the set of non-zero coefficients, and then fit an un-restricted linear model to the selected set of features." p. 91 Hastie et al., 2009

In our case, we will take the predictors with non-zero $\beta's$ and use them with lm() to get our final model. This will probably be our most realistic estimate of R^2 when caring about generalization, as we are using the test dataset to derive the estimate.

```
coef(lasso.out2)
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                     s0
## (Intercept)  152.1335
## age            .
## sex            .
## bmi          498.9887
## map          180.6668
## tc             .
## ldl            .
## hdl         -103.2937
## tch            .
## ltg          433.5503
## glu            .
```

```
lm.lasso <- lm(Y ~ bmi + map + hdl + ltg,diab.train)
lmLas.pred <- predict(lm.lasso,diab.test)

cor(lmLas.pred,diab.test$Y)**2
```

```
## [1] 0.5102243
```

Try Elastic Net – from caret

This optimizes both the λ and mixing percentage

```
library(caret)
XX <- as.matrix(diab.train[,-1])
YY <- diab.train$Y # important to change class of variable
enet.out <- train(XX,YY,method="glmnet",tuneLength=8)

row.result <- best(enet.out$results,"Rsquared",maximize=T)
enet.out$results[row.result,]
```

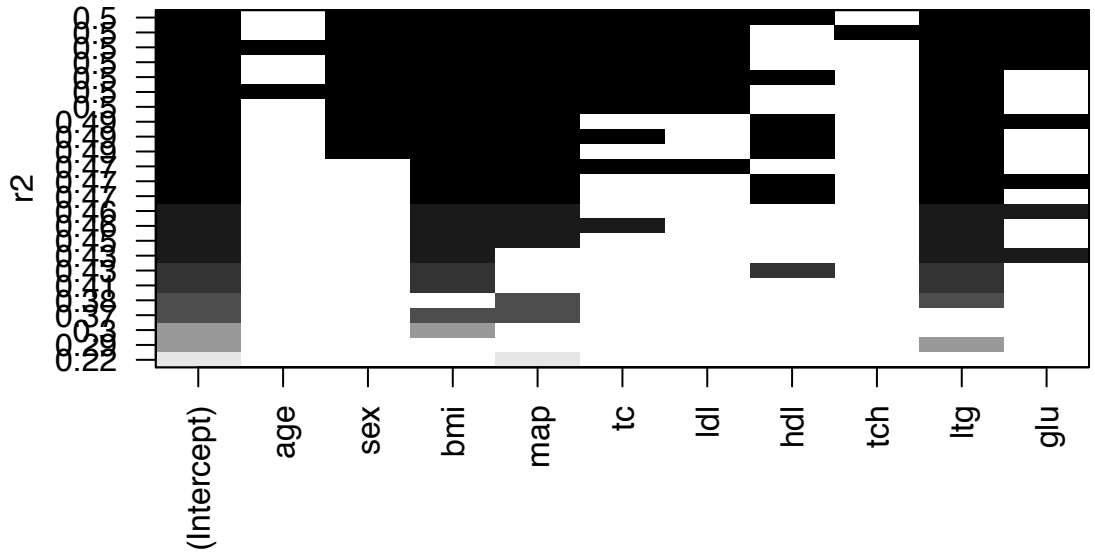
Stochastic Search

Use 3 packages: "tabuSearch" "leaps" "GA" (genetic algorithm, note there is also "genalg")

```
library(leaps)
```

```
##
## Attaching package: 'leaps'
##
## The following object is masked from 'package:subselect':
##
##     leaps
```

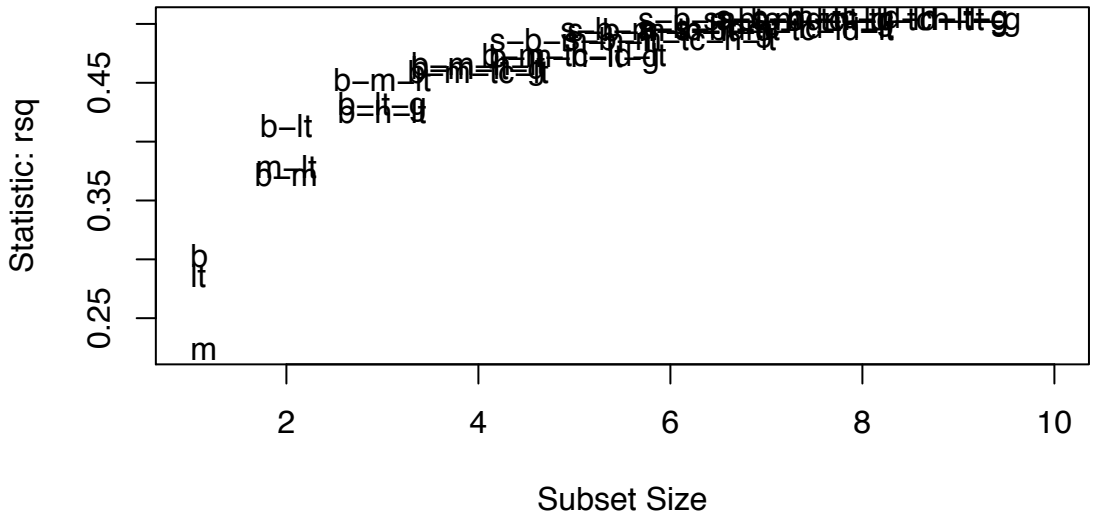
```
# leaps and bounds
leaps <- regsubsets(Y ~.,,data=diab.train,nbest=3)
#summary(leaps)
# plot a table of models showing variables in each model.
# models are ordered by the selection statistic.
plot(leaps,scale="r2")
```




```
# plot statistic by subset size
library(car)
```

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

```
subsets(leaps, statistic="rsq",legend=F)
```




```
##     Abbreviation
## age            a
## sex            s
## bmi            b
## map            m
## tc            tc
## ldl           ld
## hdl            h
## tch          tch
## ltg           lt
## glu            g
```

```
# have to press ESC to exit
```

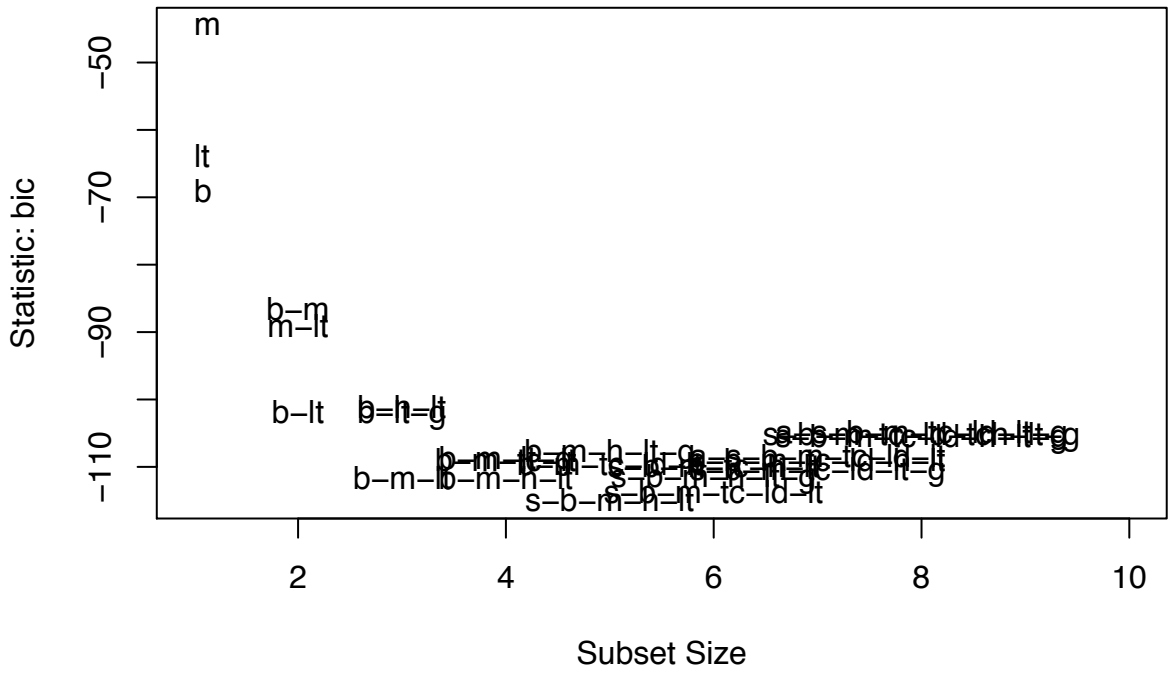
Based on this, looks like a subset size of 5 might be best, as we get pretty close to an R^2 of 0.5

Maybe R^2 isn't the best, as this doesn't penalize for complexity and will probably not generalize well. In regsubsets() we can use BIC which may help.

```
summary(leaps)$bic
```

```
##  [1]  -69.07594  -63.82772  -44.68232 -101.79487  -89.03482  -86.47742
##  [7] -111.66788 -102.34958 -101.00919 -111.57207 -109.11731 -108.98023
## [13] -114.85491 -109.53601 -108.13102 -113.72715 -111.84977 -109.87648
## [19] -110.73256 -108.88665 -108.35441 -105.91037 -105.50404 -105.34391
```

```
subsets(leaps, statistic="bic",legend=F)
```




```
##     Abbreviation
## age            a
## sex            s
## bmi            b
## map            m
## tc            tc
## ldl           ld
## hdl            h
## tch          tch
## ltg           lt
## glu            g
```

Looks like we get a similar answer, but seem to also have a "clearer" best model The

Genetic Algorithm

An example using the "GA" package to do this: http://www.jstatsoft.org/v53/i04/paper

But there is an easier way:

```
library(glmulti)
```

```
## Loading required package: rJava
```

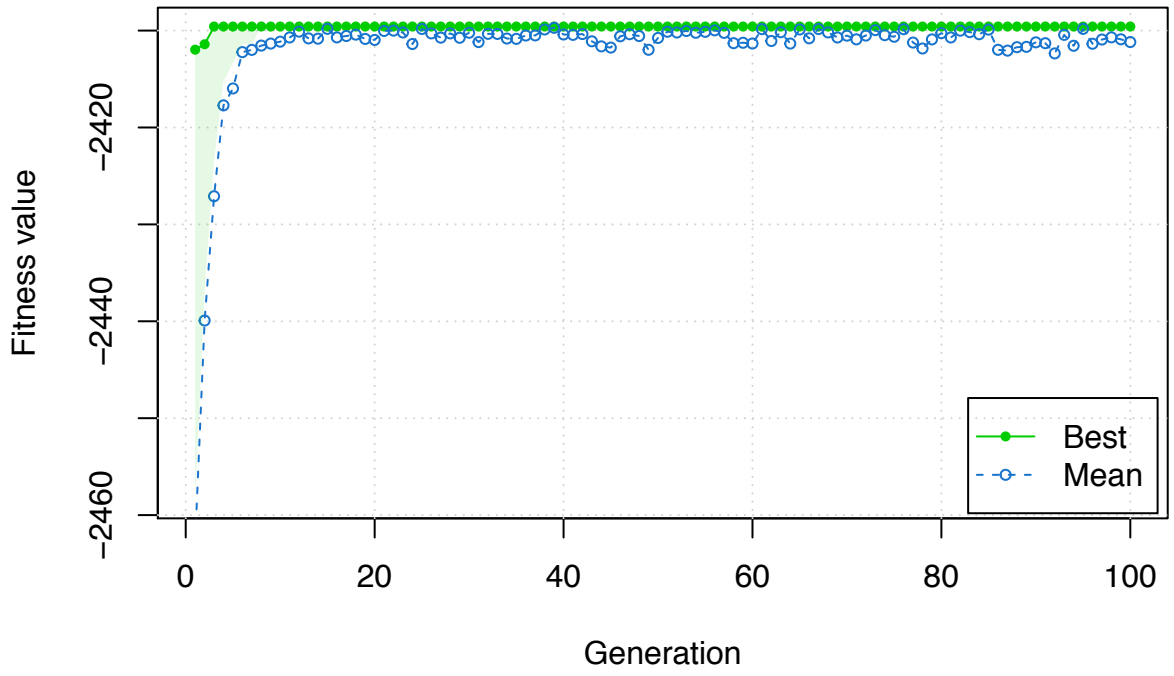
Using "GA"

```
mod <- lm(Y ~ ., diab.train)

x <- model.matrix(mod)[, -1]
y <- model.response(model.frame(mod))
```

```
fitness <- function(string) {
 inc <- which(string == 1)
 X <- cbind(1, x[,inc])
 mod <- lm.fit(X, y)
 class(mod) <- "lm"
 -AIC(mod)
}

GA <- ga("binary", fitness = fitness, nBits = ncol(x),
         names = colnames(x),monitor=F)
plot(GA)
```




```
summary(GA)
```

```
## +-----------------------------------+
## |         Genetic Algorithm         |
## +-----------------------------------+
##
## GA settings:
## Type                  =  binary
## Population size       =  50
## Number of generations =  100
## Elitism               =  2
## Crossover probability =  0.8
## Mutation probability  =  0.1
##
## GA results:
## Iterations             = 100
## Fitness function value = -2409.583
## Solution               =
##      age sex bmi map tc ldl hdl tch ltg glu
```

```
## [1,]   0   1   1   1  1   1   0   0   1   1
```

```
summary(GA)$solution
```

```
##      age sex bmi map tc ldl hdl tch ltg glu
## [1,]   0   1   1   1  1   1   0   0   1   1
```

Tabu Search

An example: http://www.r-bloggers.com/finding-the-best-subset-of-a-gam-using-tabu-search-and-visualizing-it-in-r/

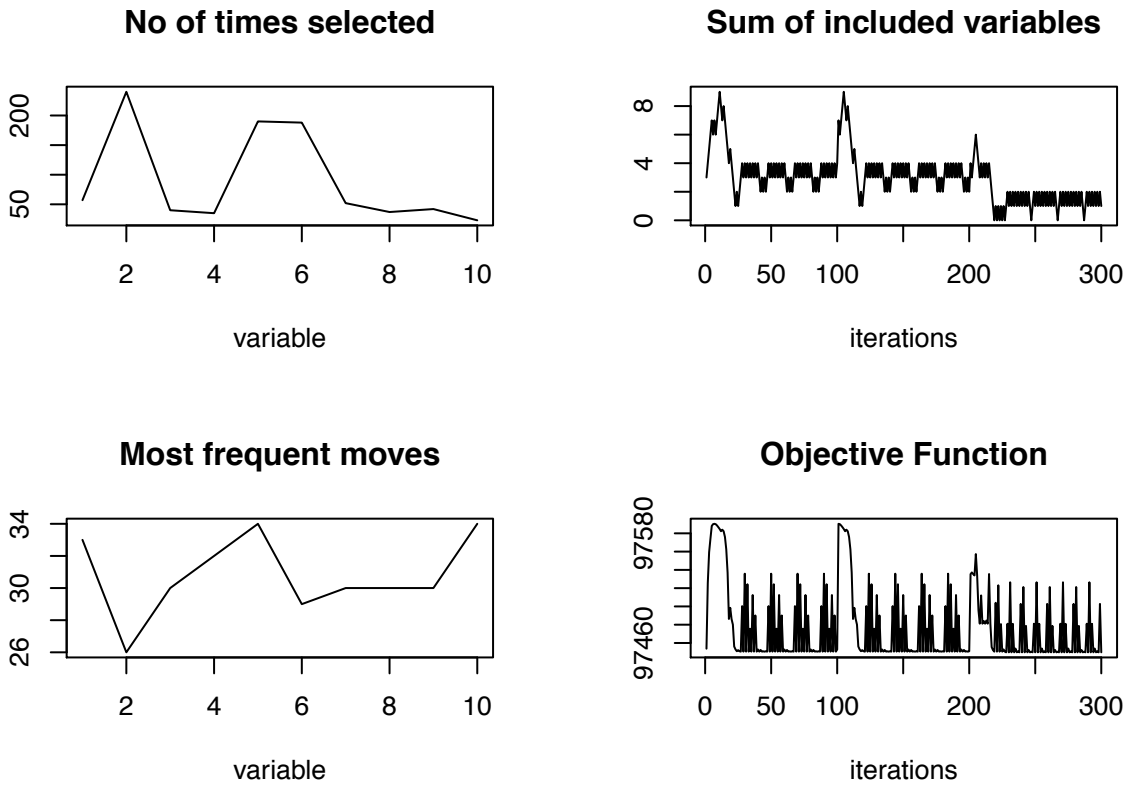
```
library(tabuSearch)

mod <- lm(Y ~ ., diab.train)

x <- model.matrix(mod)[, -1]
y <- model.response(model.frame(mod))

fitness2 <- function(string) {
 inc <- which(string == 1)
 X <- cbind(1, x[,inc])
 mod <- lm.fit(X, y)
 class(mod) <- "lm"
 -AIC(mod) + 100000 # won't take negative
}

result <- tabuSearch(size = 10, iters = 100,objFunc = fitness2)
plot(result) #fit margins too large
```



```
summary(result,verbose=T) # 6 predictors
```

```
## Tabu Settings
##   Type                                      = binary configuration
##   No of algorithm repeats                   = 1
##   No of iterations at each prelim search    = 100
##   Total no of iterations                    = 300
##   No of unique best configurations          = 55
##   Tabu list size                            = 9
##   Configuration length                      = 10
##   No of neighbours visited at each iteration = 10
## Results:
##   Highest value of objective fn    = 97590.41675
##   Occurs # of times                = 2
##   Optimum number of variables      = c(7, 7)
## Optimum configuration:
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    0    1    1    1    1    1    0    0    1     1
## [2,]    0    1    1    1    1    1    0    0    1     1
```

glmulti

Looks for interactions as well

```
library(glmulti)
# method = "g" for genetic algorithm
# default fitfunction = "glm"
multi.out = glmulti(Y ~., data=diab.train,method="g",plotty=F,
                    report=F,fitfunction="lm",crit="aic")
```

```
## TASK: Genetic algorithm in the candidate set.
## Initialization...
## Algorithm started...
## Improvements in best and average IC have bebingo en below the specified goals.
## Algorithm is declared to have converged.
## Completed.
```

```
#summary(multi.out)
summary(multi.out)$bestmodel
```

```
## [1] "Y ~ 1 + sex + bmi + map + tc + ldl + ltg + glu + bmi:age + map:bmi + "
## [2] "    tc:age + tc:sex + ldl:sex + hdl:bmi + tch:tc + tch:ldl + "
## [3] "    ltg:ldl + glu:ldl + glu:hdl + glu:tch"
```

Now we can take the output and test it out in lm()

```
eq = summary(multi.out)$bestmodel
lm.multi = lm(eq,data=diab.train)
summary(lm.multi)
```

```
##
## Call:
## lm(formula = eq, data = diab.train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -120.674  -33.210   -2.658   33.217  164.897
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)    150.061      4.617  32.504  < 2e-16 ***
## sex           -238.517     88.912  -2.683 0.007913 **
## bmi            376.726    101.401   3.715 0.000263 ***
## map            419.511     88.825   4.723 4.36e-06 ***
## tc            -795.669    251.984  -3.158 0.001836 **
## ldl            640.827    241.715   2.651 0.008661 **
## ltg            766.361    119.785   6.398 1.09e-09 ***
## glu            169.310     90.562   1.870 0.063001 .
## bmi:age       3466.129   1987.275   1.744 0.082660 .
## bmi:map       5201.707   1698.231   3.063 0.002491 **
## tc:age       -3239.013   1760.440  -1.840 0.067259 .
## sex:tc        9457.575   4462.404   2.119 0.035285 *
## sex:ldl      -7335.897   4734.103  -1.550 0.122814
## bmi:hdl       3153.680   2036.078   1.549 0.122979
## tc:tch      -14489.322   4342.291  -3.337 0.001010 **
## ldl:tch       8392.404   3740.014   2.244 0.025926 *
## ldl:ltg       5924.064   2178.279   2.720 0.007108 **
## ldl:glu      -6070.442   2964.496  -2.048 0.041889 *
## glu:hdl       8444.079   3836.584   2.201 0.028879 *
## glu:tch      12578.225   4238.164   2.968 0.003363 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 51.81 on 201 degrees of freedom
## Multiple R-squared:  0.5852, Adjusted R-squared:  0.546
## F-statistic: 14.92 on 19 and 201 DF,  p-value: < 2.2e-16
```

The addition of interactions increases our R^2 even though our criterion for glmulti was AIC.

simulated annealing

http://topepo.github.io/caret/SA.html

part of this, other options including GA available: http://topepo.github.io/caret/featureselection.html

To do, have to install most current version of caret from github. CRAN version doesn't include functions

The great thing about this function is that we can use it for all of the methods in caret (100+).

```
library(devtools)
devtools::install_github("cran/caret")
library(caret)
```

```
ctrl <- safsControl(functions = caretSA)
obj <- safs(x = diab.train[,-1],
            y = diab.train$Y,
            iters = 50,
            safsControl = ctrl,
            method = "lm")
#quartz()
plot(obj) + theme_bw()
# should increase the iterations
```

Use with genetic algorithm: http://topepo.github.io/caret/GA.html

Pretty slow

```
ctrl <- gafsControl(functions = caretGA)
obj <- gafs(x = diab.train[,-1],
            y = diab.train$Y,
            iters = 50,
            gafsControl = ctrl,
            method = "lm")
```

Classification

All of the methods used previously will also work in the classification context in using forms of logistic regression.

Logistic Regression

```
Ybin <- ifelse(diab.train$Y > mean(diab.train$Y),1,0)
diab.train2 <- diab.train
diab.train2$Y <- Ybin

# logistic
glm.out <- glm(Y ~ ., diab.train2,family="binomial")
#summary(glm.out)
```

So how well did we do? I like using receive operating characteristic (ROC) curves and the area under the curve (AUC) to evaluate results in classification.

For binary outcomes, my favorite way to assess how well I did is using the confusionMatrix() function from caret

sensitivity = true positive – with event AND predicted to have event / having the event

people who respondend AND people predicted to respond / people who respondend

specificity = samples w/o event AND predicted as non-event / samples w/o event

actual non-response AND predicted non-response/ actual non-response

false-positive = 1-specificity

```r
library(pROC)
library(caret)

glm.probs=predict(glm.out,type="response")
glm.pred=ifelse(glm.probs>0.5,1,0)


table(diab.train2$Y,glm.pred)
```

```
##    glm.pred
##      0  1
##   0 99 24
##   1 27 71
```

```r
confusionMatrix(diab.train2$Y,glm.pred,positive="1") # from caret package
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 99 24
##          1 27 71
##
##                Accuracy : 0.7692
##                  95% CI : (0.708, 0.8231)
##     No Information Rate : 0.5701
##     P-Value [Acc > NIR] : 4.809e-10
##
##                   Kappa : 0.531
##  Mcnemar's Test P-Value : 0.7794
##
##             Sensitivity : 0.7474
##             Specificity : 0.7857
##          Pos Pred Value : 0.7245
##          Neg Pred Value : 0.8049
##              Prevalence : 0.4299
##          Detection Rate : 0.3213
##    Detection Prevalence : 0.4434
##       Balanced Accuracy : 0.7665
##
##        'Positive' Class : 1
##
```

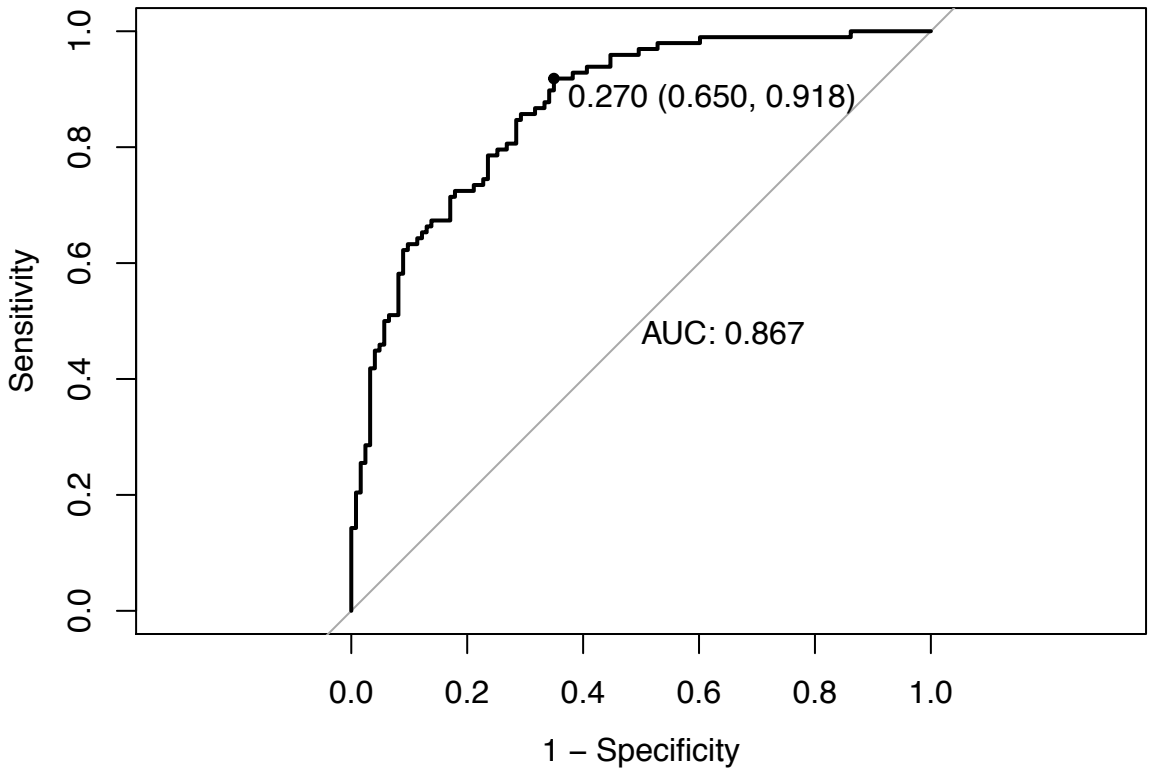
```r
rocCurve <- roc(diab.train2$Y,glm.probs)
auc(rocCurve)
```

```
## Area under the curve: 0.8672
```

```r
ci.roc(rocCurve)
```

```
## 95% CI: 0.821-0.9134 (DeLong)
```

```
plot(rocCurve, legacy.axes = TRUE,print.thres=T,print.auc=T)
```

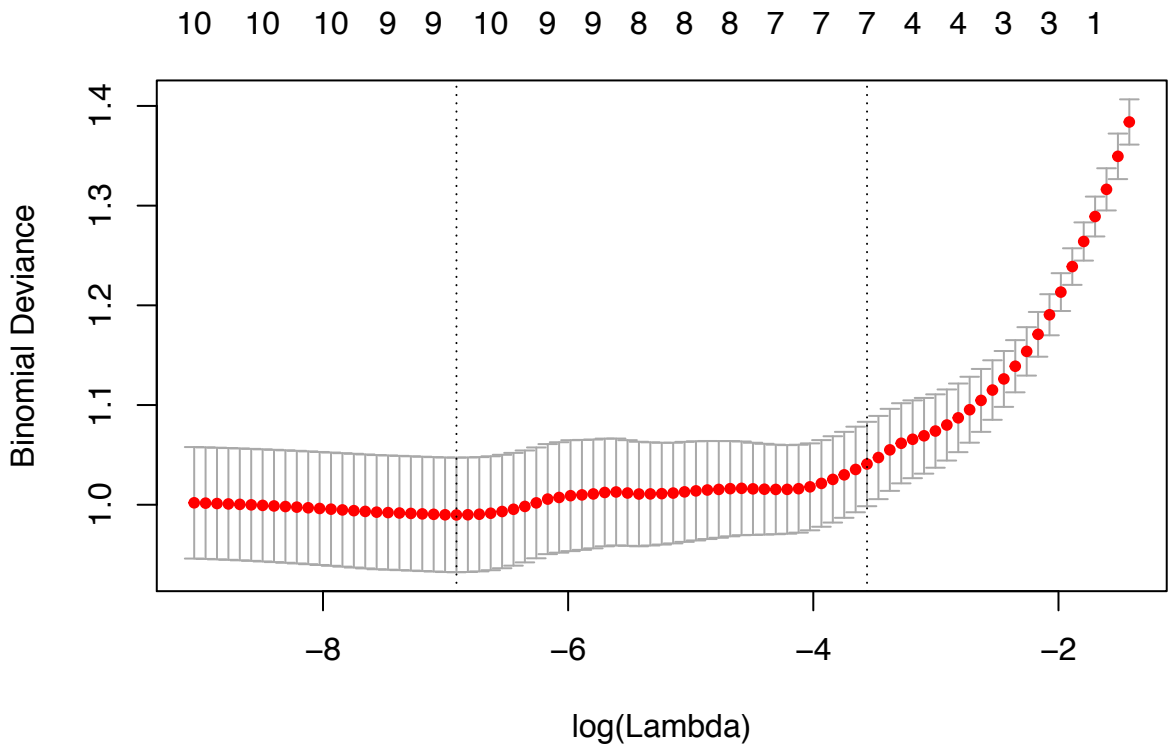



```
##
## Call:
## roc.default(response = diab.train2$Y, predictor = glm.probs)
##
## Data: glm.probs in 123 controls (diab.train2$Y 0) < 98 cases (diab.train2$Y 1).
## Area under the curve: 0.8672
```

Try also with elastic net

Regularization in caret package: http://topepo.github.io/caret/L1_Regularization.html

```
XX <- as.matrix(diab.train2[,-1])
YY <- diab.train2$Y
lasLog <- cv.glmnet(XX,YY,family="binomial")
plot(lasLog)
```




```
pred.lasLog <- predict(lasLog, XX, s="lambda.1se",type="response")
```

How did we do?

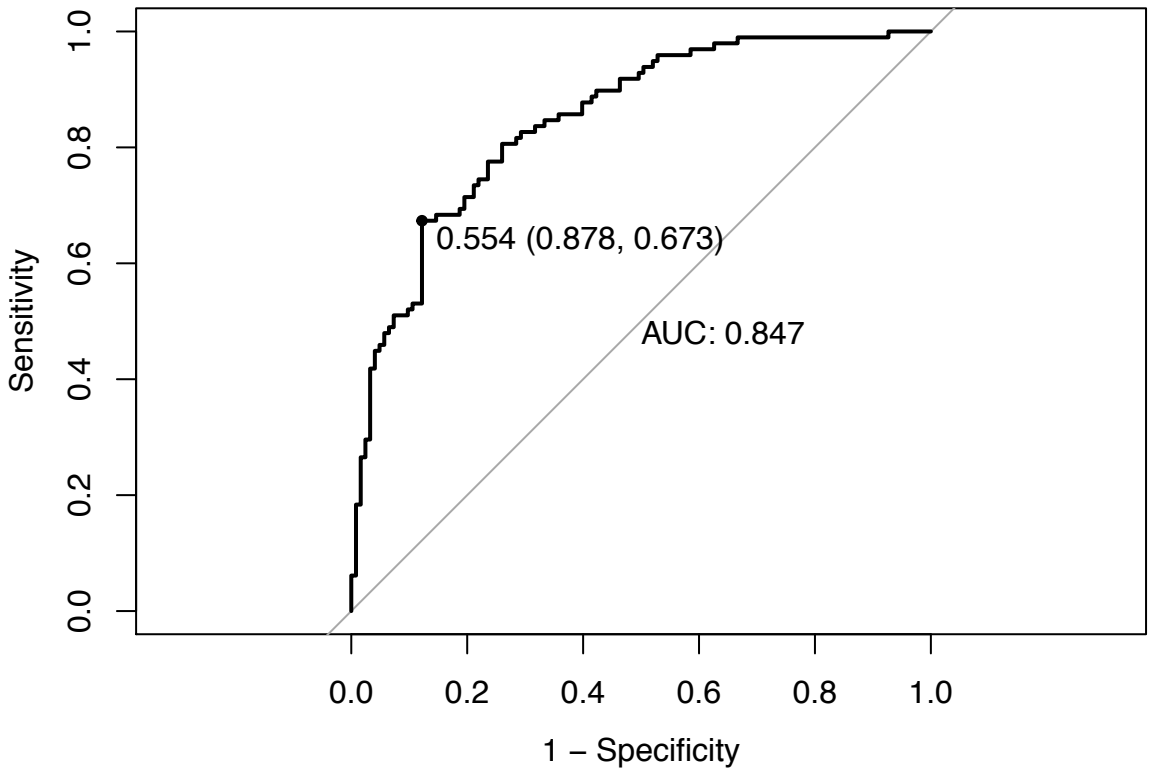
```
rocCurve2 <- roc(diab.train2$Y,pred.lasLog)
auc(rocCurve2)
```

```
## Area under the curve: 0.8474
```

```
ci.roc(rocCurve2)
```

```
## 95% CI: 0.7972-0.8977 (DeLong)
```

```
plot(rocCurve2, legacy.axes = TRUE,print.thres=T,print.auc=T)
```




```
##
## Call:
## roc.default(response = diab.train2$Y, predictor = pred.lasLog)
##
## Data: pred.lasLog in 123 controls (diab.train2$Y 0) < 98 cases (diab.train2$Y 1).
## Area under the curve: 0.8474
```

We got rid of 6 predictors and only lost 0.01 in the AUC