We demonstrated a positive results bias in optimism corrected bootstrapping by simply adding random features to our labels. This problem is due to an ‘information leak’ in the algorithm, meaning the training and test datasets are not kept seperate when estimating the optimism. Due to this, the optimism, under some conditions, can be very under estimated. Let’s analyse the code, it is pretty straightforward to understand then we can see where the problem originates.

1. Fit a model M to entire data S and estimate predictive ability C. **## this part is our overfitted estimation of performance (can be AUC, accuracy, etc)**
2. Iterate from b=1…B: **## now we are doing resampling of the data to estimate the error**
   1. Take a resample from the original data, S\*
   2. Fit the bootstrap model M\* to S\* and get predictive ability, C\_boot **## this will clearly give us another overfitted model performance, which is fine**
   3. Use the bootstrap model M\* to get predictive ability on S, C\_orig **## we are using the same data (samples) used to train the model to test it, therefore it is not surprising that we have values with better performance than expected. C\_orig values will be too high.**
3. Optimism O is calculated as mean(C\_boot – C\_orig)
4. Calculate optimism corrected performance as C-O.

One way of correcting for this would be changing **step 3 of the bootstrap**, instead of testing on the original data, to test on the left out (unseen) data in the bootstrap. This way the training and test data is kept entirely seperate in terms of samples, thus eliminating our bias towards inflated model performance on null datasets with high features. There is probably no point of doing anyway this when we have methods such as LOOCV and K fold cross validation.

As p (features) >> N (samples) we are going to get better and better ability to get good model performance using the bootstrapped data on the original data. Why? Because the original data contains the same samples as the bootstrap and as we get more features, greater the chance we are going to get some randomly correlating with our response variable. When we test the bootstrap on the original data (plus more samples) it retains some of this random ability to predict the real labels. This is a typical overfitting problem when we have higher numbers of features, and the procedure is faulty.

Let’s take another experimental look at the problem, this code can be directly copied and pasted into R for repeating the analyses and plots. We have two implementations of the method, the first by me for glmnet (lasso logisitic regression). Feel free to try different machine learning algorithms and play with the parameters.

library(glmnet)

library(pROC)

library(caret)

library(ggplot)

library(kimisc)

### TEST 1: bootstrap optimism with glmnet

cc <- c()

for (zz in seq(2,100,1)){

print(zz)

## set up test data

test <- matrix(rnorm(100\*zz, mean = 0, sd = 1),

nrow = 100, ncol = zz, byrow = TRUE)

labelsa <- as.factor(c(rep('A',50),rep('B',50)))

colnames(test) <- paste('Y',seq(1,zz),sep='')

row.names(test) <- paste('Z',seq(1,100),sep='')

### my own implementation of optimism corrected bootstrapping

## 1. fit model to entire test data (i.e. overfit it)

orig <- glmnet(test,y=labelsa,alpha=1,family = "binomial")

preds <- predict(orig,newx=test,type='response',s=0.01)

auc <- roc(labelsa,as.vector(preds))

original\_auc <- as.numeric(auc$auc)

## 2. take resample of data and try to estimate optimism

test2 <- cbind(test,labelsa)

B <- 50

results <- matrix(ncol=2,nrow=B)

for (b in seq(1,B)){

## get the bootstrapped resampled data

boot <- test2[sample(row.names(test2),50),]

labelsb <- boot[,ncol(boot)]

boot <- boot[,-ncol(boot)]

## use the bootstrapped model to predict its own labels

bootf <- glmnet(boot,y=labelsb,alpha=1,family = "binomial")

preds <- predict(bootf,newx=boot,type='response',s=0.01)

auc <- roc(labelsb,as.vector(preds))

boot\_auc <- as.numeric(auc$auc)

## use bootstrap model to predict original labels

preds <- predict(bootf,newx=test,type='response',s=0.01)

auc <- roc(labelsa,as.vector(preds))

boot\_original\_auc <- as.numeric(auc$auc)

## add the data to results

results[b,1] <- boot\_auc

results[b,2] <- boot\_original\_auc

}

## calculate the optimism

O <- mean(results[,1]-results[,2])

## calculate optimism corrected measure of prediction (AUC)

corrected <- original\_auc-O

##

cc <- c(cc,corrected)

print(cc)

}

## print it

df <- data.frame(p=seq(2,100,1),optimism\_corrected\_boot\_AUC=cc)

p1 <- ggplot(df, aes(x=p, y=optimism\_corrected\_boot\_AUC)) +

geom\_line() + ggtitle('glmnet - random data only gives positive result with optimism corrected bootstrap')

print(p1)

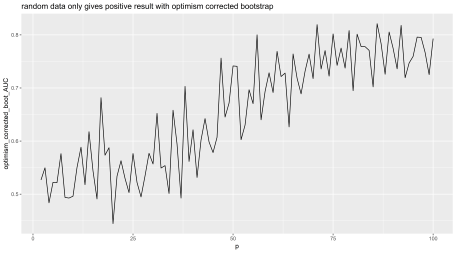
png('glmnet\_test\_upto100.png', height = 15, width = 27, units = 'cm',

res = 900, type = 'cairo')

print(p1)

dev.off()

So here are the results, as number of noise only features on the x axis increase our 'corrected' estimate of AUC (on y axis) also increases when we start getting enough to allow that noise to randomly predict the labels. So this shows the problem starts about 40-50 features, then gets worse until about 75+. This is with the 'glmnet' function.



Let’s look at the method using glm, we find the same trend, different implementation.

## TEST2

auc.adjust <- function(data, fit, B){

fit.model <- fit

data$pred.prob <- fitted(fit.model)

# get overfitted AUC

auc.app <- roc(data[,1], data$pred.prob, data=data)$auc # require 'pROC'

auc.boot <- vector (mode = "numeric", length = B)

auc.orig <- vector (mode = "numeric", length = B)

o <- vector (mode = "numeric", length = B)

for(i in 1:B){

boot.sample <- sample.rows(data, nrow(data), replace=TRUE) # require 'kimisc'

fit.boot <- glm(formula(fit.model), data = boot.sample, family = "binomial")

boot.sample$pred.prob <- fitted(fit.boot)

# get bootstrapped AUC

auc.boot[i] <- roc(boot.sample[,1], boot.sample$pred.prob, data=boot.sample)$auc

# get original data boot AUC

data$pred.prob.back <- predict.glm(fit.boot, newdata=data, type="response")

auc.orig[i] <- roc(data[,1], data$pred.prob.back, data=data)$auc

# calculated optimism corrected version

o[i] <- auc.boot[i] - auc.orig[i]

}

auc.adj <- auc.app - (sum(o)/B)

return(auc.adj)

}

cc <- c()

for (zz in seq(2,100,1)){

print(zz)

## set up test data

test <- matrix(rnorm(100\*zz, mean = 0, sd = 1),

nrow = 100, ncol = zz, byrow = TRUE)

labelsa <- as.factor(c(rep('A',50),rep('B',50)))

colnames(test) <- paste('Y',seq(1,zz),sep='')

row.names(test) <- paste('Z',seq(1,100),sep='')

test2 <- data.frame(cbind(labelsa,test))

test2$labelsa <- as.factor(test2$labelsa)

## 1. make overfitted model

model <- glm(labelsa ~., data = test2, family = "binomial")

## 2. estimate optimism and correct

d <- auc.adjust(test2, model, B=200)

cc <- c(cc,d)

}

## print it

df <- data.frame(p=seq(2,100,1),optimism\_corrected\_boot\_AUC=cc)

p2 <- ggplot(df, aes(x=p, y=optimism\_corrected\_boot\_AUC)) +

geom\_line() + ggtitle('glm - random data only gives positive result with optimism corrected bootstrap')

print(p2)

png('glmt\_test\_upto100.png', height = 15, width = 27, units = 'cm',

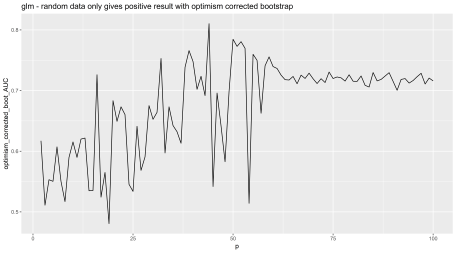
res = 900, type = 'cairo')

print(p2)

dev.off()

So there we have it, the method has a problem with it and should not be used with greater than about 40 features. This method unfortunately is currently being used for datasets with higher than this number of dimensions (40+) because people think it is a published method it is safe, unfortunately this is not how the world works R readers. Remember, the system is corrupt, science and statistics is full of lies, and if in doubt, do your own tests on positive and negative controls.

This is with the 'glm' function. Random features added on x axis, the corrected AUC on y axis.



What if you don’t believe it? I mean this is a text book method. Well R readers if that is so I suggest code it your self and try the code here, run the experiments on null (random data only) datasets with increasing features.

This is the last part in the series on debunking the optimism corrected bootstrap method. I consider it, debunked.