Logistic Regression Models

Daniel Gardner

2022-12-13

```r
#Loading in packages
set.seed(1234)
suppressPackageStartupMessages(library(tidyverse))
suppressPackageStartupMessages(library(caret))
suppressPackageStartupMessages(library(verification))
library(repr)
library(tidyverse)
library(tidymodels)
library(pROC)
```

In this section we create a baseline logistic regression model using the refined data, and then aim to reduce the affect of irrelevant variables to try and create new, improved regression models using LASSO and ridge regression.

We first import the normalised Gini function as this will be used later in the project for model validation and choosing parameters. This is a function designed to replicate the code used upon submission of the actual Kaggle competition.

```r
normalizedGini <- function(aa, pp) {
    Gini <- function(a, p) {
        if (length(a) !=  length(p)) stop("Actual and Predicted need to be equal lengths!")
        temp.df <- data.frame(actual = a, pred = p, range=c(1:length(a)))
        temp.df <- temp.df[order(-temp.df$pred, temp.df$range),]
        population.delta <- 1 / length(a)
        total.losses <- sum(a)
        null.losses <- rep(population.delta, length(a)) # Hopefully is similar to accumulatedPopulation
        accum.losses <- temp.df$actual / total.losses # Hopefully is similar to accumulatedLossPercenta
        gini.sum <- cumsum(accum.losses - null.losses) # Not sure if this is having the same effect or
        sum(gini.sum) / length(a)
    }
    Gini(aa,pp) / Gini(aa,aa)
}
```

We begin by loading in our training data, and then splitting once again into a training and validation ('testing') dataset which we will use later for comparing different regression models.

```r
#LOADING IN DATA
train<-read.csv("Dataset/LogisticTrain.csv")
#Delete X and id column
train<-train[,-c(1,2)]
#Then split into training set (70%) and validation set (30%)
train_index <- sample(c (TRUE, FALSE), nrow (train), replace=TRUE, prob=c (0.7,0.3))
```

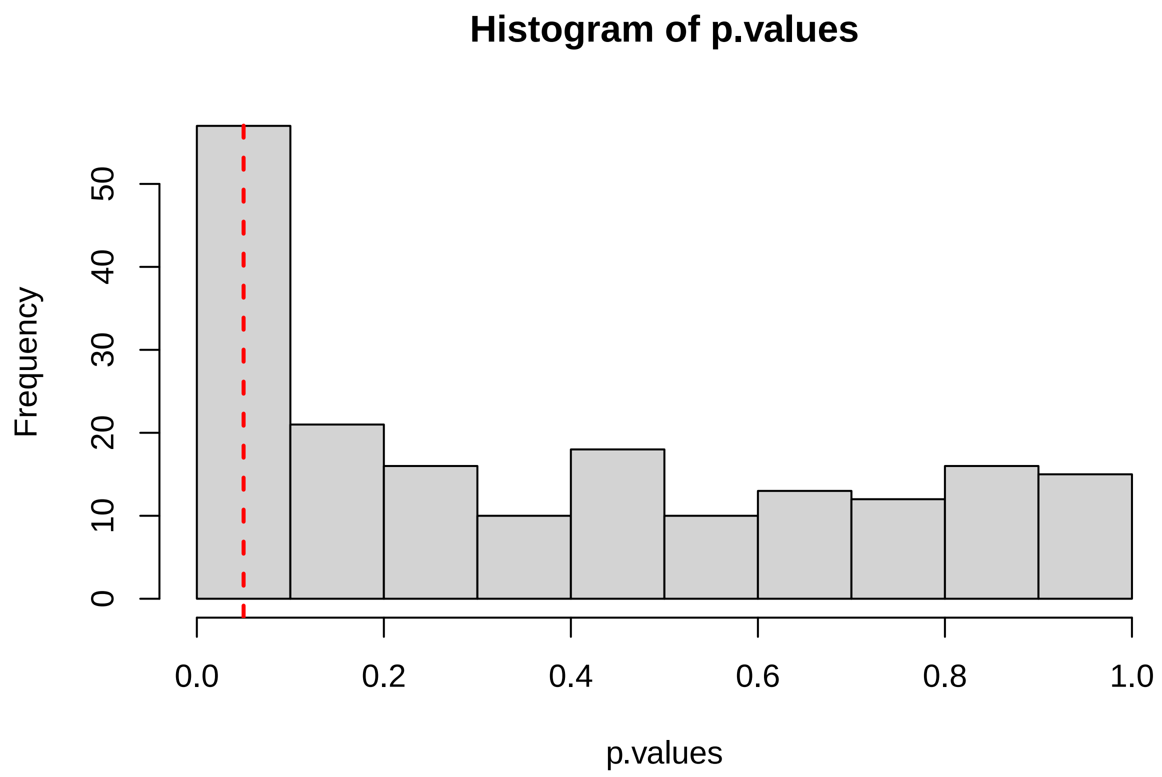
```r
# split the data according to the train index
training <- as.data.frame(train[train_index, ])
testing <- as.data.frame(train[!train_index, ])
```

Most work in refining the data has already been done, however to make our regression work well down the line, we want our matrix of covariates to be fully invertible. To achieve this we find all linear combinations in the variables and remove these columns.

```r
#REMOVING LINEAR COMBINATIONS
# find any linear combos in features
lin_comb <- findLinearCombos(training)
# take set difference of feature names and linear combos
d <- setdiff(seq(1:ncol(training)), lin_comb$remove)
# remove linear combo columns
removed_columns<-names(training)[-d]
training <- training[, d]
training<-as.data.frame(training)
```

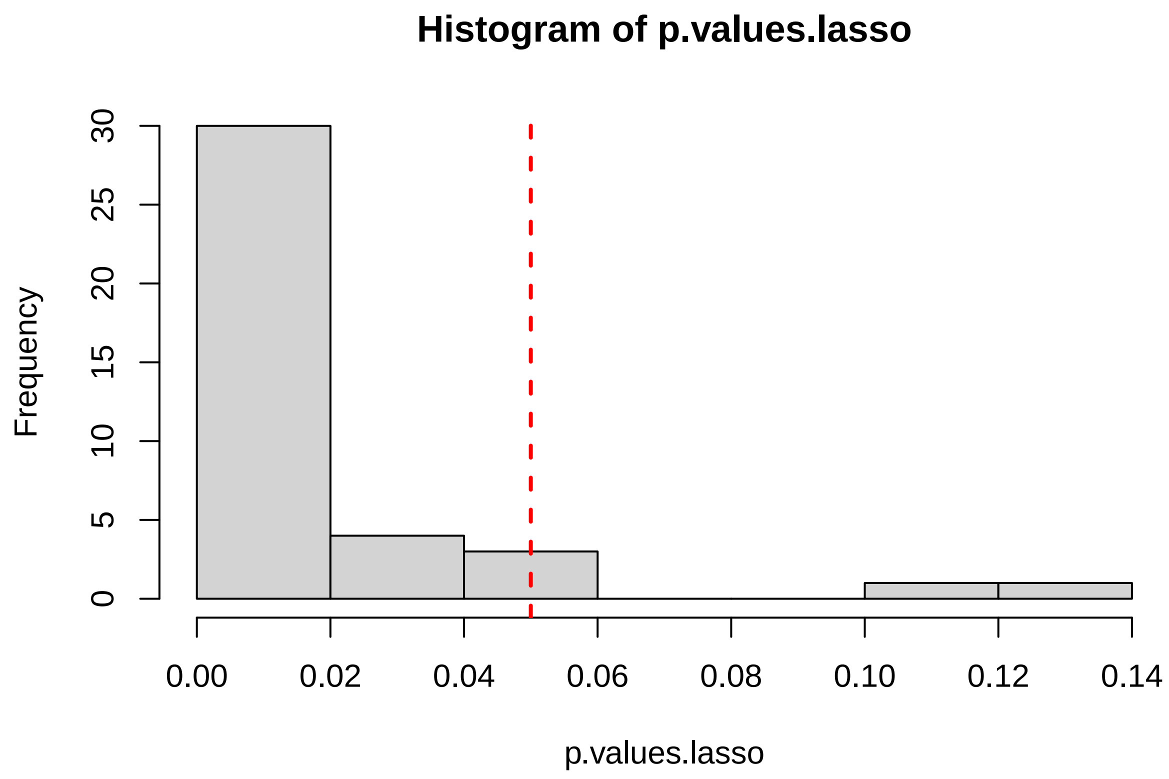
For our first baseline model we use the most basic implementation of logistic regression as done here.

```r
#Building baseline logistic regression model
logmod <- glm(target ~ . , data = training, family = binomial(link = 'logit'))
#Building a list of p-values for each co-efficient
p.values<-summary(logmod)$coefficients[,4]
#Histogram of p-values with a red dotted line at p=0.05
hist(p.values)
abline(v=0.05,col='red',lwd=2,lty=2)
```



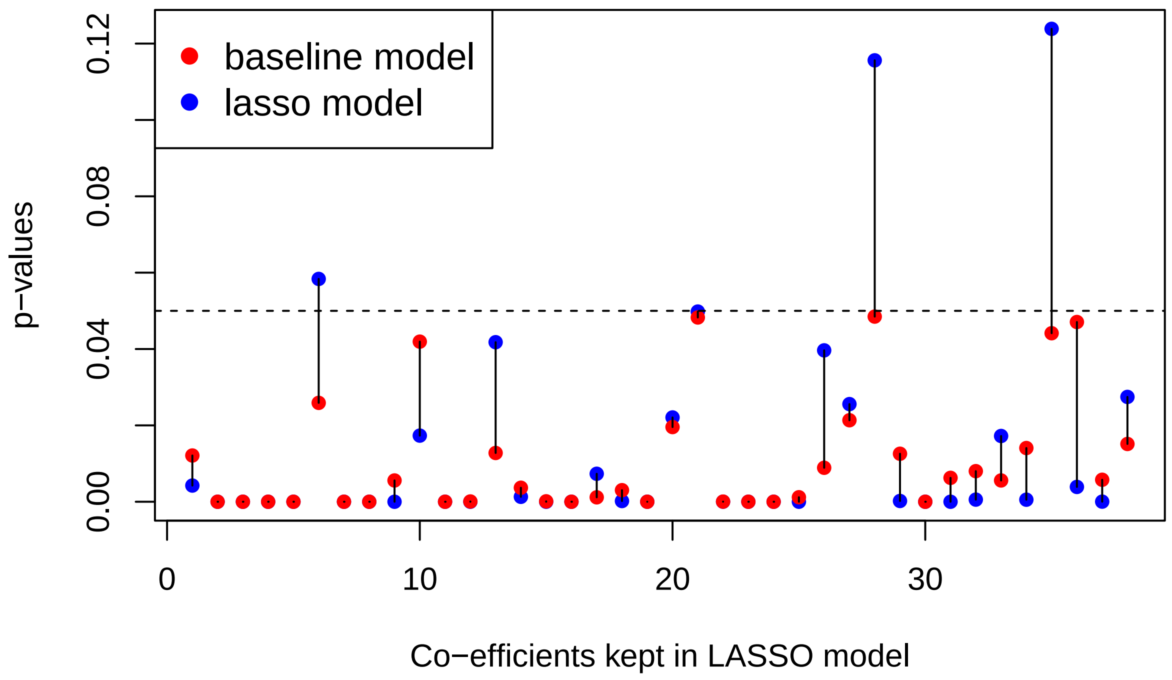
One immediate problem here is that despite our many encoded variables, not very many seem to actually contribute to our prediction. A lot of the p-values are way above the standard significance level of 0.05 as shown here by the red dotted line. This leads us to consider a LASSO dimensionality reduction: removing all variables which do not contribute enough (say a p-value of >0.05) and re-running the logistic model.

```
#BUILDING LASSO MODEL
#First we group all variables with p-values < 0.05
formula.names<-logmod %>% tidy() %>% filter(p.value < 0.05) %>% pull(term)
#Remove intercept
formula.names<-formula.names[-1]
#Then create a formula containing only these variables
model.formula<-paste("target ~ ",formula.names[1])
for (i in 2:(length(formula.names))){
  model.formula<-paste(model.formula," + ",formula.names[i])
}
logmod.lasso <- glm(model.formula, data = training, family = binomial(link = 'logit'))
p.values.lasso<-summary(logmod.lasso)$coefficients[,4]
hist(p.values.lasso)
abline(v=0.05,col='red',lwd=2,lty=2)
```



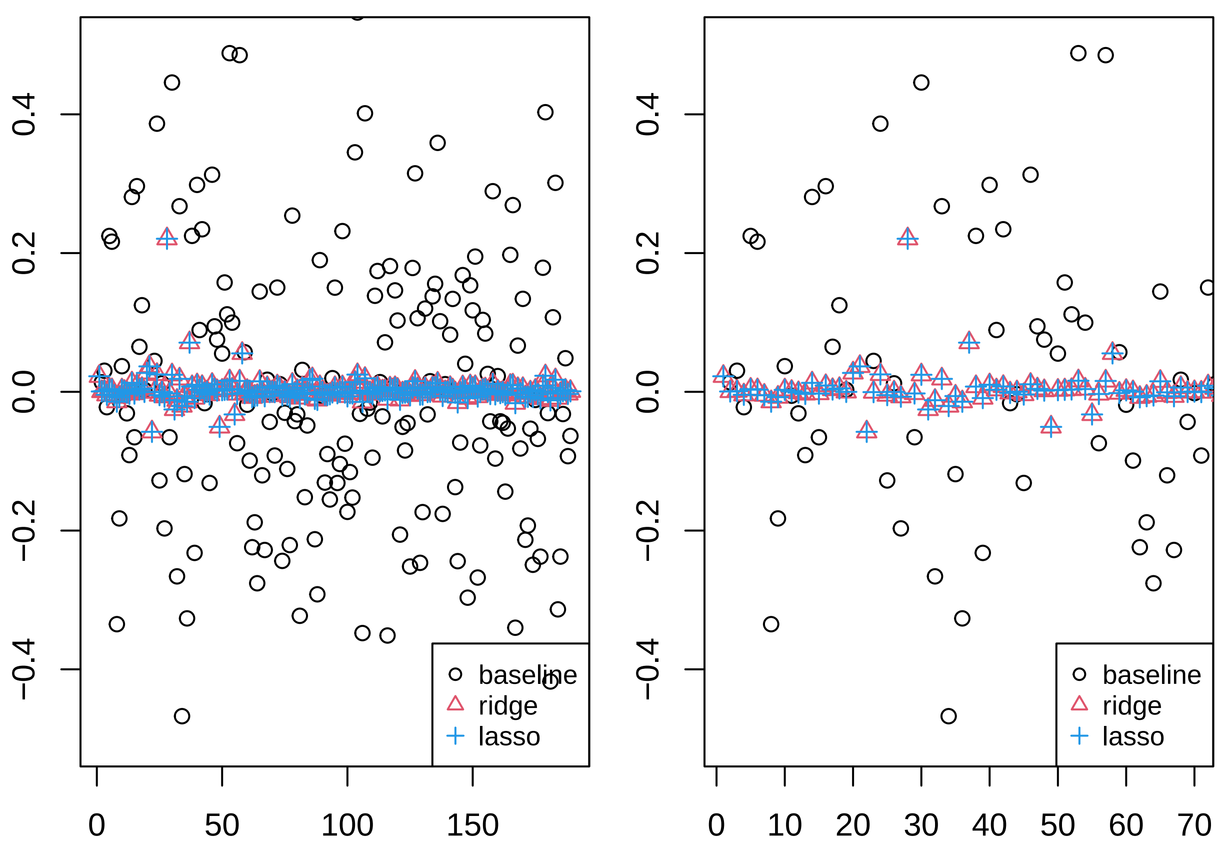
Immediately we can see that not now the vast majority of variables are significant, leading to a potentially more accurate model with less features and thus less computation needed. However one problem is that by removing a lot of the data completely, we now generate different parameter estimates due to the affect of the irrelevant data being gone. This leads to some instances where variables that were considered significant enough to be kept in the LASSO model are now revealed to have a much higher p-value, now that they exist only amongst other 'significant' variables. This can be seen in the plot below.

```r
#PLOTTING P-VALUES
#Combine p-values for only covariates used in the LASSO model
Ps<-data.frame(unlassoed=p.values[formula.names],lassoed=p.values.lasso[formula.names])
test<-as.matrix(Ps)
n<-dim(Ps)[1]
plot(seq(1,n),Ps[,2],col='blue',pch=16,xlab='Co-efficients kept in LASSO model',ylab='p-values')
points(seq(1,n),Ps[,1],col='red',pch=16)
#Adding lines between points to show the change
for (i in 1:n){
  segments(x0=i,y0=Ps[i,1],x1=i,y1=Ps[i,2])
}
#Adding 0.05 significance level line
abline(h=0.05,col='black',lty=2)
legend("topleft",pch=c(16,16),col=c('red','blue'),cex=1.2,legend=c("baseline model","lasso model"))
```



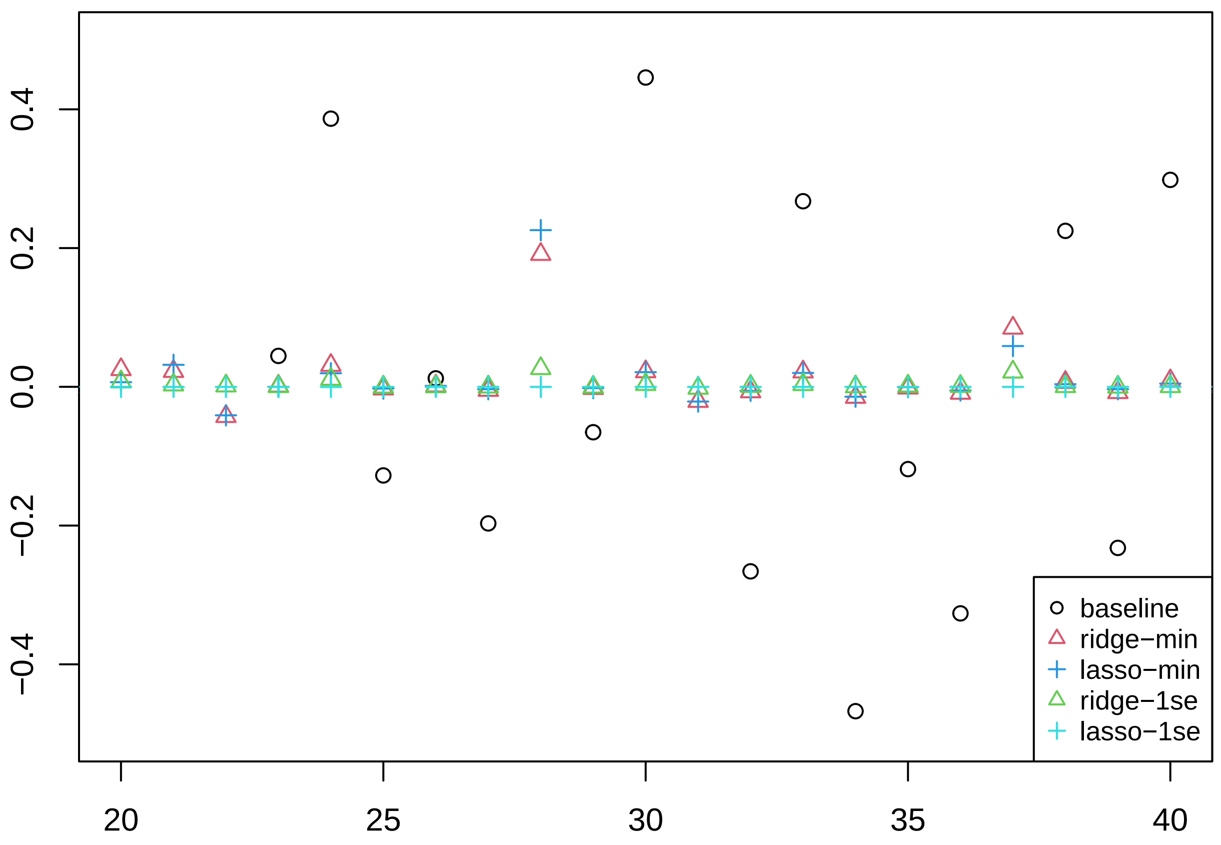
We can improve on this however by using pre-packaged methods of LASSO and ridge regression using GLMNET, which will allow us to calibrate how harsh to be when reducing the impact of certain variables, instead of making such a binary decision as setting a co-efficient to exactly 0 for a small p-value.

```
#Building LASSO and ridge models using the glmnet package
library(glmnet)
#Convert our training data into two response and covariate matrices y and X
X<-as.matrix(training[,-1])
y<-as.matrix(training[,1])
#Building standard ridge and LASSO fits with very small penalty parameter lambda
ridge.fit<-glmnet(X,y,lambda=1e-7,standardise=TRUE,alpha=0)
lasso.fit<-glmnet(X,y,lambda=1e-7,standardise=TRUE,alpha=1)
#Visualising the change in coefficients
plotter<-function(a,b){
  par(mar=c(2.5,2.5,.5,.5))
    plot(coef(logmod),ylim=c(-0.5,0.5),xlim=c(a,b),ylab="Co-efficient",xlab=paste("Covariates between",
    points(coef(ridge.fit),pch=2,col=2)
    points(coef(lasso.fit),pch=3,col=4)
    legend("bottomright",pch=c(1,2,3),col=c(1,2,4),cex=.8,legend=c("baseline","ridge","lasso"))
}
#Plotting the full plot vs a zoomed in version.
par(mfrow=c(1,2))
plotter(1,length(coef(logmod)))
plotter(1,70)
```



Similar to the by-hand method above, the vast majority of co-efficients are pushed down to 0 or near enough, with only a few kept as signficant. Also at this very low level of the penalty parameter lambda=1e-7, the estimates for LASSO regression and ridge regression look very similar. To choose the best parameter of lambda, we use cross-validation on our training data. The cv.glmnet function produces two possible options: the minimum cross validated error (lambda.min), and the largest lambda such that the cross-validated error is within one standard error of the minimum (lambda.1se).

```
#CROSS-VALIDATION MODELS
#We use the default 10-fold cross validation method in glmnet to find the an optimal value of lambda
cv.ridge.fit<-cv.glmnet(X,y,standardise=TRUE,alpha=0)
cv.lasso.fit<-cv.glmnet(X,y,standardise=TRUE,alpha=1)
#We then use the optimised lambda.min and lambda.1se to get two optimised models each for regression an
cv.lasso.fit.min<-glmnet(X,y,lambda=cv.lasso.fit$lambda.min,standardise=TRUE,alpha=1)
cv.lasso.fit.1se<-glmnet(X,y,lambda=cv.lasso.fit$lambda.1se,standardise=TRUE,alpha=1)
cv.ridge.fit.min<-glmnet(X,y,lambda=cv.ridge.fit$lambda.min,standardise=TRUE,alpha=0)
cv.ridge.fit.1se<-glmnet(X,y,lambda=cv.ridge.fit$lambda.1se,standardise=TRUE,alpha=0)
#Again we can visualise the data here in a zoomed-in snapshot
par(mar=c(2.5,2.5,.5,.5))
plot(coef(logmod),ylab='',ylim=c(-0.5,0.5),xlim=c(20,40))
points(coef(cv.ridge.fit.min),pch=2,col=2)
points(coef(cv.lasso.fit.min),pch=3,col=4)
points(coef(cv.ridge.fit.1se),pch=2,col=3)
points(coef(cv.lasso.fit.1se),pch=3,col=5)
legend("bottomright",pch=c(1,2,3,2,3),col=c(1,2,4,3,5),cex=.8,legend=c("baseline","ridge-min","lasso-mi
```



As we can see from this snapshot of the new co-efficients, the 1se models are overly harsh on their penalties, setting pretty much every co-efficient to 0, whereas the models using the minimum lambda and much more fair. We would expect this to negatively affect prediction, but we will include them anyway in our final analysis for comparison.

Now finally we can compare all 5 potenital improvements to our baseline regression, using the NormalisedGini score as defined at the beginning, and the area under the ROC curve.

```
#VALIDATION
#Saving the actual target values
actual<-testing$target
# remove linear combo columns
testing <- testing[, d]
testing<-as.data.frame(testing)
#Removing the target column
testing<-testing[-1]
#Getting predicted values for p_i via the default predict function in R
baseline.preds<-predict(logmod,newdata=testing,type='response')
logmod.lasso.preds<-predict(logmod.lasso,newdata=testing,type='response')
#Getting predicted values for p_i via glmnet's predict function - this requires the testing data becomi
lasso.min.preds<-predict.glmnet(cv.lasso.fit.min,newx=as.matrix(testing))
lasso.1se.preds<-predict.glmnet(cv.lasso.fit.1se,newx=as.matrix(testing))
ridge.min.preds<-predict.glmnet(cv.ridge.fit.min,newx=as.matrix(testing))
ridge.1se.preds<-predict.glmnet(cv.ridge.fit.1se,newx=as.matrix(testing))
#Combining all our predicted data into one data frame
preds<-data.frame(
  'baseline'=baseline.preds,
  'logmod.lasso'=logmod.lasso.preds,
  'lasso.min'=lasso.min.preds[,1],
  'lasso.1se'=lasso.1se.preds[,1],
  'ridge.min'=ridge.min.preds[,1],
  'ridge.1se'=ridge.1se.preds[,1],
  stringsAsFactors = FALSE
)
#Creating a table showing the ROC area and NormalisedGini score for each model
validation<-matrix(data=NA,nrow=6,ncol=3)
for (i in 1:6){
  validation[i,1]<-names(preds)[i]
  validation[i,2]<-roc.area(actual,preds[,i])$A
  validation[i,3]<-normalizedGini(actual,preds[,i])
}
validation
```

```
##      [,1]           [,2]                [,3]
## [1,] "baseline"     "0.624633815753172" "0.249267631506343"
## [2,] "logmod.lasso" "0.623456442356521" "0.246912884713042"
## [3,] "lasso.min"    "0.625324233218155" "0.250648466436309"
## [4,] "lasso.1se"    "0.5"               "-0.00551882221085229"
## [5,] "ridge.min"    "0.62472661105904"  "0.24945322211808"
## [6,] "ridge.1se"    "0.620243829203752" "0.240487658407504"
```

Here we can see that using the penalty parameter of lambda.min, we gain a slight improvement in prediction over the baseline logistic regression model. The scores achieved by ridge.min and lasso.min are near identical, and either could be chosen really, although as lasso.min holds a slight advantage we will choose this as our most optimised regression model. Also we can see that the models using lambda.1se perform very badly as expected, performing even worse than the default baseline model. This confirms what we suspected before about the penalties they assign to the co-efficients being to too extreme. As well as this, whilst our original LASSO model performs worse than the baseline, the difference is not massive, so it could be argued that it could be worth taking a reduction in accuracy to reduce the dimensionality of the model and the time it takes to run.

```
#PREDICTION
#Loading in actual test data
test<-read.csv('Dataset/LogisticTest.csv')
#Save id column
ids<-test['id']
test<-test[-c(1,2)]
#Remove linear combinations
test <- test[, d]
test<-as.data.frame(test)
#Removing the target column
test<-test[-1]
baseline.preds.test<-predict(logmod,newdata=test,type='response')
table1<-data.frame('id'=ids,'predictions'=baseline.preds.test)
write.csv(table1,'Results/baseline_preds.csv')
lasso.min.preds.test<-predict.glmnet(cv.lasso.fit.min,newx=as.matrix(test))
table2<-data.frame('id'=ids,'predictions'=lasso.min.preds.test)
write.csv(table2,'Results/lasso_preds.csv')
```

#References

Code for priming the data and AUC scores was taken from https://www.kaggle.com/code/captcalculator/logistic-regression-and-roc-curve-primer

Code for by-hand LASSO model was taken from https://www.kaggle.com/code/sukhyun5/predict-with-logistic-regression-and-lasso

Code for Cross-Validation models was taken from 'Linear and Generalised Linear Models (MATH30013)' by Haeran Cho, mainly from Section 7: 'High-dimensional linear regression'.

Code for Gini-coefficient was taken from https://www.kaggle.com/c/ClaimPredictionChallenge/discussion/703