Heart Disease Classification: Model Fitting

Carlos Kelaidis

4/12/2021

Read in data set:

```
heart_dat <- read.csv("~/Documents/My Working Directory/Personal Projects/Heart Attack Prediciton & Ana
#View(heart_dat)
```

Load some libraries:

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.5          v purrr   0.3.4
## v tibble  3.1.6          v dplyr   1.0.7.9000
## v tidyr   1.1.4          v stringr 1.4.0
## v readr   1.3.1          v forcats 0.4.0

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(knitr)
library(ggpubr)
library(tibble)
```

We now have a set of candidate variables.

Those picked through observations: + **ExerciseAngina** + **Oldpeak** + **MaxHR** + **Age** + **Cholesterol** + **ChestPainType**

Then using the Best Susbet Selection method, we picked a 12 variable model including the following variables.

Important to note, that through the model fitting procedure, some factor variables with more than 2 levels, namely **ChestPainType**, are counted as more than one variable to account for the extra levels. Hence, the 12 variable model is actually a 9 variable model (without intercept), namely: + **Age** + **Sex** + **ChestPainType** + **Cholesterol** + **FastingBS** + **MaxHR** + **ExerciseAngina** + **Oldpeak** + **ST_Slope**

The 7 variable model included, so actually a 6 variable model (without intercept): + **Sex** + **ChestPainType** + **Cholesterol** + **FastingBS** + **ExerciseAngina** + **ST_Slope**

We will use the 10 (with intercept) variables picked by the BSS as they also include our previously noted variables, this is using all the variables available basically.

If we notice that some variables are insignificant going forward we can drop them.

Model Fitting and Selection

- Logistic regression
- LDA

- KNN
- Lasso
- SVM
- Classification Tree/Random forest

Logistic Regression

We begin by splitting the data into a train set and a test set:

```
#Split data into train and test set
set.seed(1)
test.sample<-sample(nrow(heart_dat), nrow(heart_dat)/3)#take a third of the data for a the test sample
heart.train<-heart_dat[-test.sample,]
heart.test<-heart_dat[test.sample,]
```

We fit a logistic regression model on all the variables:

```
#Fit on training set
log.fit.full<-glm(HeartDisease~., data=heart.train, family=binomial)
summary(log.fit.full)
```

```
##
## Call:
## glm(formula = HeartDisease ~ ., family = binomial, data = heart.train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5636  -0.3817   0.1722   0.4381   2.5911
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -1.158031   1.709467  -0.677 0.498138
## Age               0.020012   0.016248   1.232 0.218068
## SexM              1.397490   0.350643   3.986 6.73e-05 ***
## ChestPainTypeATA -1.973501   0.423387  -4.661 3.14e-06 ***
## ChestPainTypeNAP -1.825604   0.327484  -5.575 2.48e-08 ***
## ChestPainTypeTA  -1.448333   0.520454  -2.783 0.005389 **
## RestingBP         0.001252   0.006839   0.183 0.854738
## Cholesterol      -0.003774   0.001329  -2.839 0.004529 **
## FastingBS         1.193504   0.328815   3.630 0.000284 ***
## RestingECGNormal -0.022448   0.332244  -0.068 0.946133
## RestingECGST     -0.137850   0.424071  -0.325 0.745134
## MaxHR            -0.002369   0.006009  -0.394 0.693413
## ExerciseAnginaY   1.138517   0.305169   3.731 0.000191 ***
## Oldpeak           0.261969   0.143755   1.822 0.068406 .
## ST_SlopeFlat      1.188740   0.533557   2.228 0.025884 *
## ST_SlopeUp       -1.150411   0.560438  -2.053 0.040102 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 843.64  on 611  degrees of freedom
## Residual deviance: 403.57  on 596  degrees of freedom
## AIC: 435.57
##
```

```
## Number of Fisher Scoring iterations: 5
```

From the above summary it appears that a few variables are insignificant: + **Age** (don't know why since it clearly affected disease status based on the previous observations) + **RestingBP** + **RestingECG** + **MaxHR** + **Oldpeak**

Let's drop the above variables and refit:

```
log.fit.reduced<-glm(HeartDisease~Sex+ChestPainType+Cholesterol+
                       FastingBS+ExerciseAngina+
                       ST_Slope, data=heart.train, family=binomial)

summary(log.fit.reduced)
```

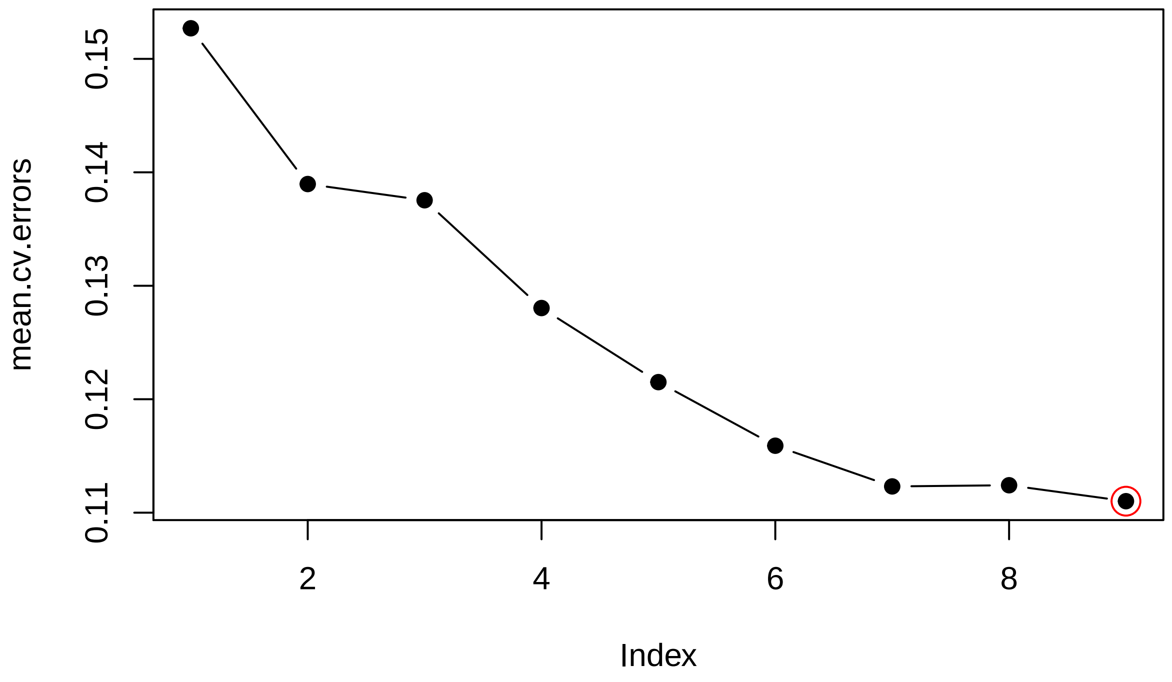
```
##
## Call:
## glm(formula = HeartDisease ~ Sex + ChestPainType + Cholesterol +
##     FastingBS + ExerciseAngina + ST_Slope, family = binomial,
##     data = heart.train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5481  -0.3821   0.1829   0.4361   2.6339
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)       0.372005   0.636667   0.584 0.559017
## SexM              1.351079   0.347129   3.892 9.94e-05 ***
## ChestPainTypeATA -2.085572   0.411416  -5.069 3.99e-07 ***
## ChestPainTypeNAP -1.824434   0.318403  -5.730 1.00e-08 ***
## ChestPainTypeTA  -1.362901   0.501833  -2.716 0.006611 **
## Cholesterol      -0.003820   0.001278  -2.989 0.002799 **
## FastingBS         1.229522   0.325157   3.781 0.000156 ***
## ExerciseAnginaY   1.376880   0.280955   4.901 9.55e-07 ***
## ST_SlopeFlat      0.759937   0.497115   1.529 0.126340
## ST_SlopeUp       -1.734498   0.497834  -3.484 0.000494 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 843.64  on 611  degrees of freedom
## Residual deviance: 410.43  on 602  degrees of freedom
## AIC: 430.43
##
## Number of Fisher Scoring iterations: 5
```

Variables are all significant, we are hence decided on using the following variables - **Sex, ChestPainType, Cholesterol, FastingBS, ExerciseAngina, ST__Slope**.

Let's fit a BSS model using the above variables and use CV to see what size model gets picked:

```
predict.regsubsets<-function(object, newdata, id,...){
  form<-as.formula(object$call[[2]])
  mat<-model.matrix(form, newdata)
  coefs<-coef(object, id=id)
  xvars<-names(coefs)
  mat[,xvars]%*%coefs
}
```

```
library(leaps)
#10-fold CV:
bss.fit2<-regsubsets(HeartDisease~Sex+ChestPainType+Cholesterol+
                       FastingBS+ExerciseAngina+
                       ST_Slope, data=heart_dat, nvmax=15)
k<-10
set.seed(1)
folds<-sample(rep(1:k, length=nrow(heart_dat)))
cv.errors<-matrix(NA, k, 9, dimnames = list(NULL, paste(1:9)))

for(i in 1:k){
  bss.fit<-regsubsets(HeartDisease~Sex+ChestPainType+Cholesterol+
                       FastingBS+ExerciseAngina+
                       ST_Slope, data=heart_dat[folds!=i,], nvmax=15)
  for(j in 1:9){
    preds<-predict.regsubsets(bss.fit, heart_dat[folds==i,], id=j)
    cv.errors[i,j]<-mean((heart_dat$HeartDisease[folds==i]-preds)^2)
  }
}
mean.cv.errors<-apply(cv.errors, 2, mean)
plot(mean.cv.errors, pch=19, type="b")+
  points(which.min(mean.cv.errors), mean.cv.errors[which.min(mean.cv.errors)],
         col=2, cex=2)
```



```
## integer(0)
```

As expected, the best performing model is the one using all the variables, let's compare the one of size 7 though:

```
mean.cv.errors[9]
```

```
##         9
## 0.1110129
```

```
mean.cv.errors[7]
```

```
##         7
## 0.1123142
```

So the CV error rate is about the same between the 9 variable model and the 7 variable one.

```
coef(bss.fit2, 9)
```

```
##      (Intercept)             SexM ChestPainTypeATA ChestPainTypeNAP
##     0.6098170952     0.1599721369    -0.2746092355    -0.2409557326
##   ChestPainTypeTA      Cholesterol       FastingBS  ExerciseAnginaY
##    -0.1838422717    -0.0004693035     0.1379010961     0.1714461086
##      ST_SlopeFlat       ST_SlopeUp
##     0.1094851551    -0.3056893520
```

```
coef(bss.fit2, 7)
```

```
##      (Intercept)             SexM ChestPainTypeATA ChestPainTypeNAP
##     0.6831816536     0.1597063670    -0.2492698622    -0.2183257397
##      Cholesterol        FastingBS  ExerciseAnginaY       ST_SlopeUp
##    -0.0004747709     0.1332335080     0.1889644545    -0.4037395302
```

These models use the same variables, the only difference is that the 9 variable model uses more dummies than the 7 variable one. So nothing new to see with regards to variables and we stick with the variables picked for our reduced logistic fit.

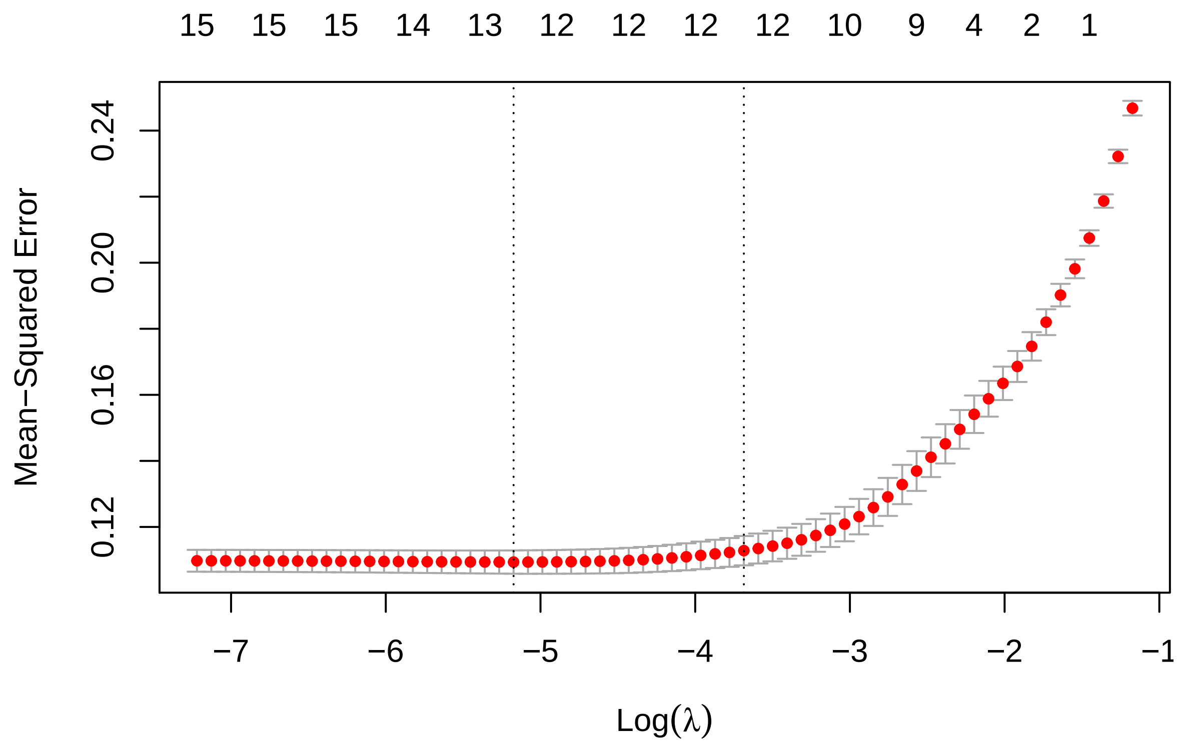
Lasso

One final attempt for the variables. We fit a Lasso model and let it perform variable selection for us. Let's see how it does:

```
library(glmnet)
```

```
## Loading required package: Matrix

##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack

## Loaded glmnet 4.1-1
```

```
#matrix of predictors
X<-model.matrix(HeartDisease~., data=heart_dat)
#response vector
Y<-heart_dat$HeartDisease

#Use CV to pick tuning parameter
set.seed(1)
cv.heart_dat<-cv.glmnet(X,Y, alpha=1)
plot(cv.heart_dat)
```



```
print(cv.heart_dat)
```

```
##
## Call:  cv.glmnet(x = X, y = Y, alpha = 1)
##
## Measure: Mean-Squared Error
##
##       Lambda Index Measure       SE Nonzero
## min 0.005663    44  0.1094 0.003527      12
## 1se 0.025089    28  0.1128 0.004439      12
```

So we see a Lasso model performs no variable selection, it keeps all variables, we can/will further examine the lasso model, but was mainly curious to see whether it would perform variable selection, and if yes on which variables. However, no variable selection was done.

Logistic regression does not make the same normality assumptions as linear regression, so no need to check for heteroscedasticity or normality. We do however check for mulitcollinearity:

Let's test for potential **multicollinearity** in our variables:

```
library(car)
```

```
## Loading required package: carData
```

```
##
## Attaching package: 'car'
```

```
## The following object is masked from 'package:dplyr':
##
##     recode
```

```
## The following object is masked from 'package:purrr':
##
##     some
```

```
vif(log.fit.reduced)
```

```
##                    GVIF Df GVIF^(1/(2*Df))
## Sex            1.049867  1        1.024630
## ChestPainType  1.183293  3        1.028447
## Cholesterol    1.091503  1        1.044750
## FastingBS      1.075821  1        1.037218
## ExerciseAngina 1.124391  1        1.060373
## ST_Slope       1.167137  2        1.039395
```

All the VIFs < 2.5 (For logistic regression), indicating no issues regarding multicollinearity.

Evaluating the accuracy of the fit

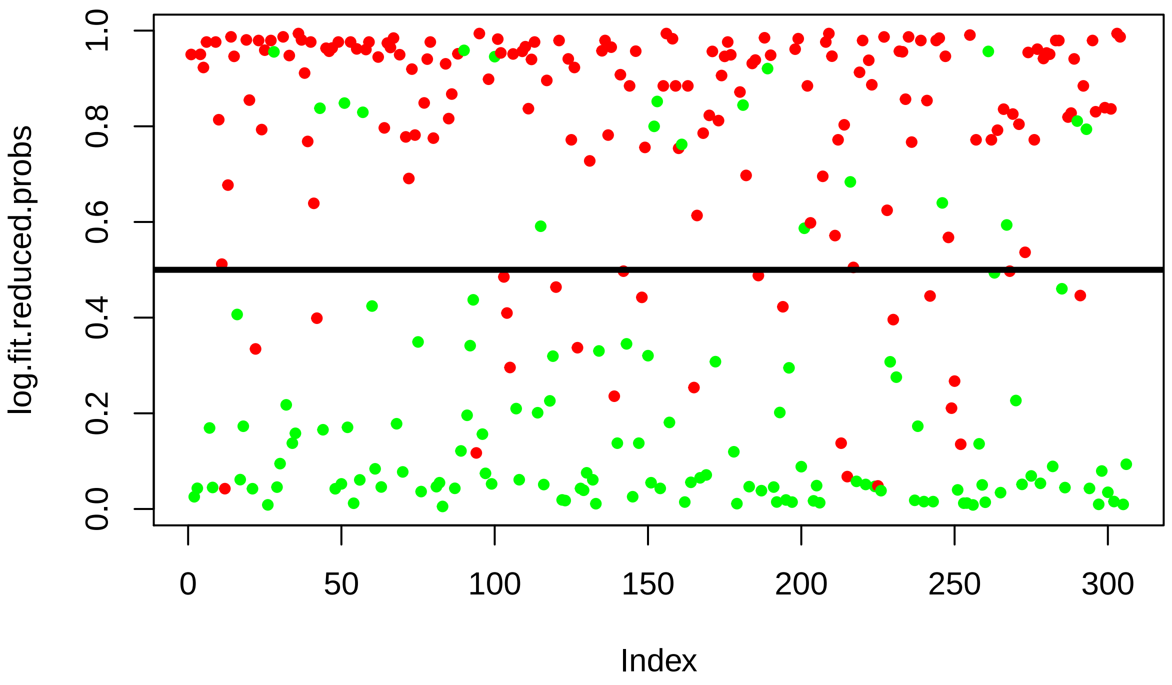
We can now evaluate the accuracy of logistic regression fit using the test MSE:

```
#Probabilities
log.fit.reduced.probs<-predict(log.fit.reduced, heart.test,type="response")
length(log.fit.reduced.probs)#306
```

```
## [1] 306
```

Let's create a plot and observe how the 0.5 threshold for our posterior probabilities separates the observations:

```
plot(log.fit.reduced.probs, col=ifelse(heart.test$HeartDisease==1,
                                       "red","green"), pch=20)+
  abline(h=.5, lwd=3)
```



```
## integer(0)
```

The observations are separated quite nicely, the 0.5 threshhold seems to be appropriate.

```
log.fit.reduced.preds<-rep("0", length(log.fit.reduced.probs))
log.fit.reduced.preds[log.fit.reduced.probs>0.5]<-"1"
#The confusion matrix
table(log.fit.reduced.preds, heart.test$HeartDisease)
```

```
##
## log.fit.reduced.preds   0   1
##                     0 112  25
##                     1  19 150
```

The classification error rate:

```
#Test MSE
log.fit.reduced.MSE<-1-mean(log.fit.reduced.preds==heart.test$HeartDisease)
log.fit.reduced.MSE
```

```
## [1] 0.1437908
```

```
#Storing the test MSE
testMSEs<-rbind(c("Logistic Regression"),
                c(round(log.fit.reduced.MSE,3)))
```

Hence, our logistic model fit above will misslcassify patients as having heart disease 14.4% of the time, which is pretty good.

Since Logistic Regression is a more linear approach, let's try one that's a bit less parametric, namely **KNN**.

The predictors used in the logistic regression are Sex, ChestPainType, Cholesterol, FastingBS, ExerciseAngina and ST_Slope.

KNN with K = 5

```
#Cannot run knn with categorical predictors, so need to convert
#all the predictors to numerical data
#Need to convert Sex, ChestPainType, ExerciseAngina and ST_Slope
#View(heart.train)

#Converting Sex
train.num_Sex<-factor(NA, levels=c("1","0"))
train.num_Sex[heart.train$Sex=="M"]<-"1"
train.num_Sex[heart.train$Sex=="F"]<-"0"
train.num_Sex<-as.numeric(as.character(train.num_Sex))#0 is female, 1 is male

#Converting ChestPainType
summary(heart.train$ChestPainType)
```

```
## ASY ATA NAP  TA
## 337 106 137  32
```

```
train.num_ChestPainType<-factor(NA, levels=c("1","2","3","4"))
train.num_ChestPainType[heart.train$ChestPainType=="ASY"]<-"1"
train.num_ChestPainType[heart.train$ChestPainType=="ATA"]<-"2"
train.num_ChestPainType[heart.train$ChestPainType=="NAP"]<-"3"
train.num_ChestPainType[heart.train$ChestPainType=="TA"]<-"4"
train.num_ChestPainType<-as.numeric(as.character(train.num_ChestPainType))

#Converting ExerciseAngina
summary(heart.train$ExerciseAngina)
```

```
##   N   Y
## 364 248
```

```r
train.num_ExerciseAngina<-factor(NA, levels=c("1","0"))
train.num_ExerciseAngina[heart.train$ExerciseAngina=="Y"]<-"1"
train.num_ExerciseAngina[heart.train$ExerciseAngina=="N"]<-"0"
train.num_ExerciseAngina<-as.numeric(as.character(train.num_ExerciseAngina))

#Converting ST_Slope
summary(heart.train$ST_Slope)
```

```
## Down Flat   Up
##   42  292  278
```

```r
train.num_STslope<-factor(NA, levels=c("1","0","2"))
train.num_STslope[heart.train$ST_Slope=="Down"]<-"1"
train.num_STslope[heart.train$ST_Slope=="Flat"]<-"0"
train.num_STslope[heart.train$ST_Slope=="Up"]<-"2"
train.num_STslope<-as.numeric(as.character(train.num_STslope))

#Now let's add these numeric variables to a matrix
#cbind(train.num_Sex,train.num_ChestPainType,train.num_ExerciseAngina,train.num_STslope, heart.train[,c

#Need to do the same for test set
#Converting Sex
test.num_Sex<-factor(NA, levels=c("1","0"))
test.num_Sex[heart.test$Sex=="M"]<-"1"
test.num_Sex[heart.test$Sex=="F"]<-"0"
test.num_Sex<-as.numeric(as.character(test.num_Sex))#0 is female, 1 is male

#Converting ChestPainType
test.num_ChestPainType<-factor(NA, levels=c("1","2","3","4"))
test.num_ChestPainType[heart.test$ChestPainType=="ASY"]<-"1"
test.num_ChestPainType[heart.test$ChestPainType=="ATA"]<-"2"
test.num_ChestPainType[heart.test$ChestPainType=="NAP"]<-"3"
test.num_ChestPainType[heart.test$ChestPainType=="TA"]<-"4"
test.num_ChestPainType<-as.numeric(as.character(test.num_ChestPainType))

#Converting ExerciseAngina
test.num_ExerciseAngina<-factor(NA, levels=c("1","0"))
test.num_ExerciseAngina[heart.test$ExerciseAngina=="Y"]<-"1"
test.num_ExerciseAngina[heart.test$ExerciseAngina=="N"]<-"0"
test.num_ExerciseAngina<-as.numeric(as.character(test.num_ExerciseAngina))

#Converting ST_Slope
test.num_STslope<-factor(NA, levels=c("1","0","2"))
test.num_STslope[heart.test$ST_Slope=="Down"]<-"1"
test.num_STslope[heart.test$ST_Slope=="Flat"]<-"0"
test.num_STslope[heart.test$ST_Slope=="Up"]<-"2"
test.num_STslope<-as.numeric(as.character(test.num_STslope))

#cbind(test.num_Sex,test.num_ChestPainType,test.num_ExerciseAngina,test.num_STslope, heart.test[,c(5,6)

#Before we run KNN need to set a few variables
train.X<-cbind(train.num_Sex,train.num_ChestPainType,train.num_ExerciseAngina,
               train.num_STslope, heart.train[,c(5,6)])
test.X<-cbind(test.num_Sex,test.num_ChestPainType,test.num_ExerciseAngina,
              test.num_STslope, heart.test[,c(5,6)])
train.Y<-heart.train$HeartDisease
```

Now that we have converted everything to numeric variables, we can perform KNN:

```
library(class)
set.seed(1)
knn.pred<-knn(train.X, test.X, train.Y, k=5)
table(knn.pred, heart.test$HeartDisease)
```

```
##
## knn.pred   0   1
##        0  97  33
##        1  34 142
```

```
#The test MSE
knn.testMSE<-1-mean(knn.pred==heart.test$HeartDisease)
knn.testMSE
```

```
## [1] 0.2189542
```

```
testMSEs<-rbind(c("Logistic Regression", "KNN"),
                c(round(log.fit.reduced.MSE,3), round(knn.testMSE,3)))
```

Hence, our KNN (with K = 5) fit will misslcassify patients as having heart disease 21.9% of the time. It is hence outperformed by the logistic regression model.

We have now fit a KNN & a logistic regression, since the logistic regression outperformed the KNN model, the decision-boundary could be twoards the more linear side. Hence, next we fit a linear SVM and perhaps also a kernel SVM. Also, LDA sometimes outperforms logistic regression, so we fit a LDA model as well.

LDA

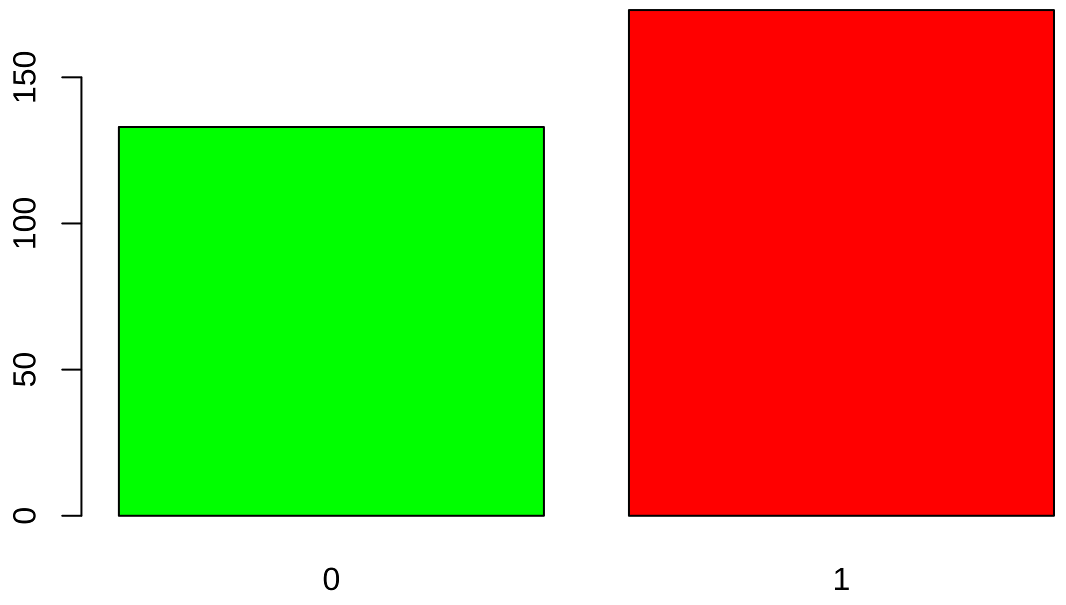
```
library(MASS)
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

```
lda.fit<-lda(HeartDisease~Sex+ChestPainType+Cholesterol+
                       FastingBS+ExerciseAngina+
                       ST_Slope, data=heart.train)
lda.fit
```

```
## Call:
## lda(HeartDisease ~ Sex + ChestPainType + Cholesterol + FastingBS +
##     ExerciseAngina + ST_Slope, data = heart.train)
##
## Prior probabilities of groups:
##         0         1
## 0.4558824 0.5441176
##
## Group means:
##        SexM ChestPainTypeATA ChestPainTypeNAP ChestPainTypeTA Cholesterol
## 0 0.6738351       0.32974910        0.3333333      0.07168459    223.9247
## 1 0.9099099       0.04204204        0.1321321      0.03603604    177.2853
##   FastingBS ExerciseAnginaY ST_SlopeFlat ST_SlopeUp
## 0 0.1075269       0.1254480    0.1827957  0.7813620
## 1 0.3363363       0.6396396    0.7237237  0.1801802
##
## Coefficients of linear discriminants:
##                          LD1
## SexM               0.587168536
## ChestPainTypeATA -1.130963259
## ChestPainTypeNAP -1.025739185
## ChestPainTypeTA  -0.801288121
## Cholesterol      -0.001930109
## FastingBS          0.604040849
## ExerciseAnginaY   0.856645199
## ST_SlopeFlat       0.378939086
## ST_SlopeUp       -1.161916307
```

```r
#Confusion matrix
lda.preds<-predict(lda.fit, heart.test, type="response")
plot(lda.preds$class, col=ifelse(heart.test$HeartDisease==1, "green","red"))
```



We can see, LDA classifies observation fairly evenly, there should not be a too high amount of false positives nor false negatives. LDA model should be fairly accurate.

```r
table(lda.preds$class, heart.test$HeartDisease)
```

```
##
##        0   1
##   0 111  22
##   1  20 153
```

```r
#Test MSE
lda_testMSE<-1-mean(lda.preds$class==heart.test$HeartDisease)
lda_testMSE
```

```
## [1] 0.1372549
```

```
testMSEs<-rbind(c("Logistic Regression", "KNN", "LDA"),
                c(round(log.fit.reduced.MSE,3), round(knn.testMSE,3),
                  round(lda_testMSE, 3)))
```

Hence, our LDA fit will misslcassify patients as having heart disease 13.7% of the time, outperforming both KNN and Logistic Regression.

SVMs

Let's begin by fitting a linear SVM (SVC) and use CV to pick the tuning parameters:

```
#For SVMs need a factor as my response
library(e1071)
set.seed(1)
svc.tune.out<-tune(svm, as.factor(HeartDisease)~Sex+ChestPainType+Cholesterol+
                       FastingBS+ExerciseAngina+
                       ST_Slope,
                   data=heart.train, kernel="linear",
                   ranges = list(cost=c(0.001, 0.01, 0.1, 1, 5, 10, 100)))
summary(svc.tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.1404283
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-03 0.4558435 0.03650012
## 2 1e-02 0.1535167 0.05374642
## 3 1e-01 0.1437070 0.04958856
## 4 1e+00 0.1404283 0.05252013
## 5 5e+00 0.1469064 0.05496121
## 6 1e+01 0.1485457 0.05468272
## 7 1e+02 0.1485457 0.05468272
```

The lowest CV error (0.1404283) occurs with **cost** equal to 1.

```
svc.bestfit<-svc.tune.out$best.model
summary(svc.bestfit)
```

```
##
## Call:
## best.tune(method = svm, train.x = as.factor(HeartDisease) ~ Sex +
##     ChestPainType + Cholesterol + FastingBS + ExerciseAngina + ST_Slope,
##     data = heart.train, ranges = list(cost = c(0.001, 0.01, 0.1,
##         1, 5, 10, 100)), kernel = "linear")
##
##
## Parameters:
```

```
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  215
##
##  ( 108 107 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

The confusion matrix and classification error rate:

```r
svc.bestfit.preds<-predict(svc.bestfit, heart.test)
#Confusion matrix
table(predict=svc.bestfit.preds, truth=heart.test$HeartDisease)
```

```
##        truth
## predict   0   1
##       0 110  25
##       1  21 150
```

```r
#Test MSE
svc_testMSE<-1-(110+150)/nrow(heart.test)
svc_testMSE
```

```
## [1] 0.1503268
```

```r
testMSEs<-rbind(c("Logistic Regression", "KNN", "LDA", "SVC"),
                c(round(log.fit.reduced.MSE,3), round(knn.testMSE,3),
                  round(lda_testMSE, 3), round(svc_testMSE, 3)))
```

Hence, our SVC will missclcassify patients as having heart disease 15.0% of the time, outperforming both KNN but not Logistic Regression and LDA.

Let's observe our models through some **ROC** plots: Need to create a ROC plot function first:

```r
library(ROCR)
roc_plot<-function(pred, truth, ...){
  predob=prediction(pred, truth)
  perf=performance(predob, "tpr", "fpr")
  plot(perf, ...)
}
```

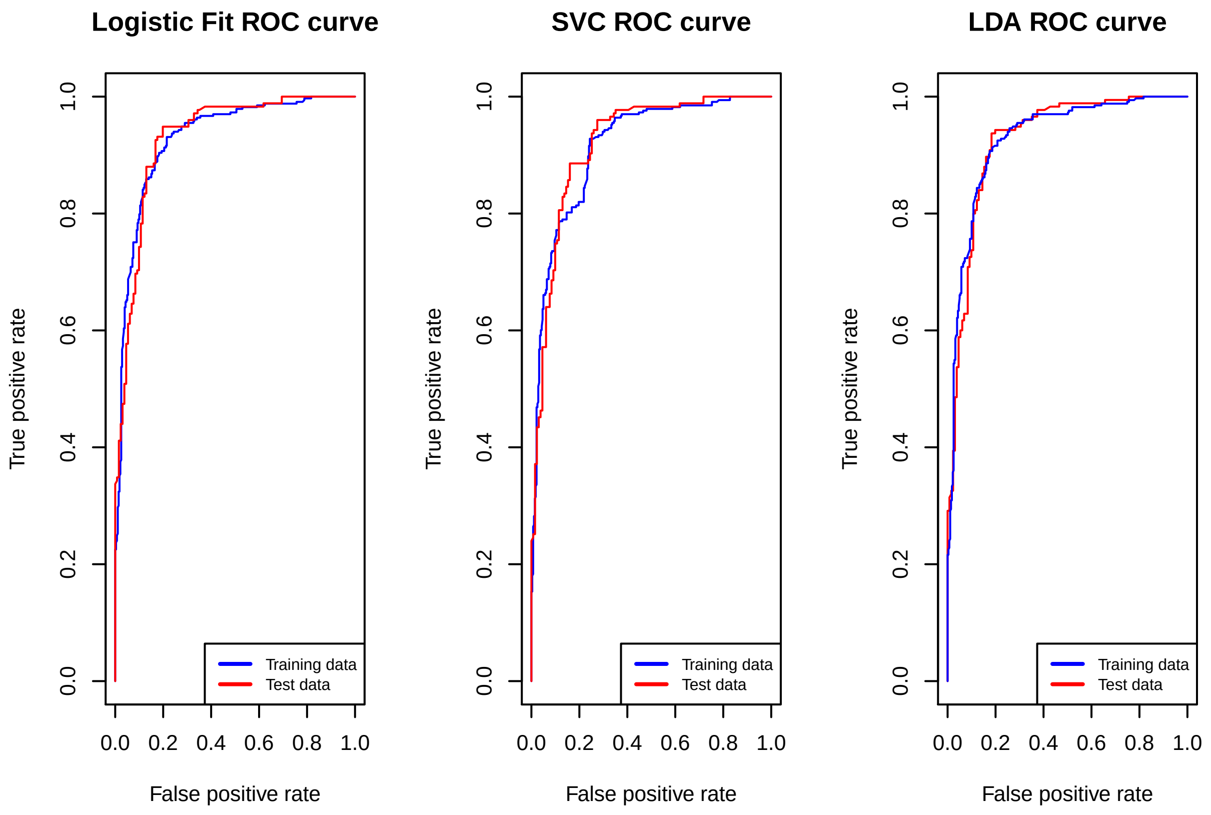
Some variable manipualtions:

```r
#Logisitic fit on test data
log.test.fit<-glm(HeartDisease~Sex+ChestPainType+Cholesterol+
                    FastingBS+ExerciseAngina+ST_Slope,
                  data=heart.test, family=binomial)
#SVC with cost = 1
svc.fit.opt<-svm(HeartDisease~Sex+ChestPainType+Cholesterol+
                    FastingBS+ExerciseAngina+ST_Slope,
                 data=heart_dat, kernel="linear", cost=1, decision.values=T)
#SVC fitted on training
svc.train.fitted<-attributes(predict(svc.fit.opt, heart.train, decision.values = T))$decision.values
```

```r
#SVC fitted on test
svc.test.fitted<-attributes(predict(svc.fit.opt, heart.test, decision.values = T))$decision.values

#LDA on train
lda.train.preds<-predict(lda.fit, heart.train, type="response")
lda.train.pr<-prediction(lda.train.preds$posterior[,2],
                         heart.train$HeartDisease)
lda.train.perf<-performance(lda.train.pr, "tpr", "fpr")
#LDA on test
lda.test.pr<-prediction(lda.preds$posterior[,2], heart.test$HeartDisease)
lda.test.perf<-performance(lda.test.pr, "tpr", "fpr")
```

Now the ROC plots:

```r
par(mfrow=c(1,3))
roc_plot(log.fit.reduced$fitted.values, heart.train$HeartDisease, col="blue",main="Logistic Fit ROC cur
roc_plot(log.test.fit$fitted.values, heart.test$HeartDisease, add=T,col=2)
legend("bottomright", legend=c("Training data", "Test data"),
       col=c("blue",2), lty=1, lwd=2, cex=.8)
roc_plot(svc.train.fitted, heart.train$HeartDisease, col="blue",main="SVC ROC curve")
roc_plot(svc.test.fitted, heart.test$HeartDisease, add=T,col=2)
legend("bottomright", legend=c("Training data", "Test data"),
       col=c("blue",2), lty=1, lwd=2, cex=.8)
#LDA
plot(lda.test.perf, col="red",main="LDA ROC curve")
plot(lda.train.perf, col="blue", add=T)
legend("bottomright", legend=c("Training data", "Test data"),
       col=c("blue",2), lty=1, lwd=2, cex=.8)
```



We can see all three methods perform pretty accurately on the test data.

Lastly, let's fit a tree-based method.

Random forest

We fit a random forest on all the predictors:

```r
library(randomForest)
```

```
## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':
##
##     combine

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
set.seed(1)
#mtry = 3 because sqrt(11) = 3.3 (11 predictors)
rf.class.tree<-randomForest(as.factor(HeartDisease)~., data=heart.train,
                            mtry=3,
                            importance = T)
rf.class.tree
```

```
##
## Call:
##  randomForest(formula = as.factor(HeartDisease) ~ ., data = heart.train,      mtry = 3, importance =
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 13.89%
## Confusion matrix:
##     0   1 class.error
## 0 227  52   0.1863799
## 1  33 300   0.0990991
```

The classification error rate of our random forest:

```r
rf.preds<-predict(rf.class.tree, newdata=heart.test, type="class")
#Confusion matrix
table(rf.preds, as.factor(heart.test$HeartDisease))
```

```
##
## rf.preds   0   1
##        0 112  18
##        1  19 157
```

```r
#Test MSE
rf_testMSE<-1-(112+157)/nrow(heart.test)
rf_testMSE
```

```
## [1] 0.120915
```

```
testMSEs<-rbind(c("Logistic Regression", "KNN", "LDA", "SVC",
                  "Random Forest"),
                c(round(log.fit.reduced.MSE,3), round(knn.testMSE,3),
                  round(lda_testMSE, 3), round(svc_testMSE, 3),
                  round(rf_testMSE, 3)))
```

Hence, our Random Forest will misslcassify patients as having heart disease 12.1% of the time, outperforming all other models.

Here are all our models' test MSEs:

```
testMSEs
```

```
##      [,1]                  [,2]    [,3]    [,4]   [,5]
## [1,] "Logistic Regression" "KNN"   "LDA"   "SVC"  "Random Forest"
## [2,] "0.144"               "0.219" "0.137" "0.15" "0.121"
```

They are all pretty close, by our top three is the **Random Forest**, followed by **LDA**, followed by **Logistic Regression**.

Important to note, is that the Random Forest uses all the predictors, let's fit a random forest with the predicotrs that appeared significant for the logistic regression model & LDA, see if this new random forest has an increased accuracy:

```
set.seed(1)
#mtry = 3 because sqrt(11) = 3.3 (11 predictors)
rf.class.tree2<-randomForest(as.factor(HeartDisease)~Sex+ChestPainType+
                                Cholesterol+FastingBS + ExerciseAngina+
                                ST_Slope,
                              data=heart.train,
                             mtry=3,
                             importance = T)
rf.preds2<-predict(rf.class.tree2, newdata=heart.test, type="class")
#Confusion matrix
table(rf.preds2, as.factor(heart.test$HeartDisease))
```

```
##
## rf.preds2   0   1
##         0 110  22
##         1  21 153
```

```
#Test MSE
1-(110+153)/nrow(heart.test)
```

```
## [1] 0.1405229
```

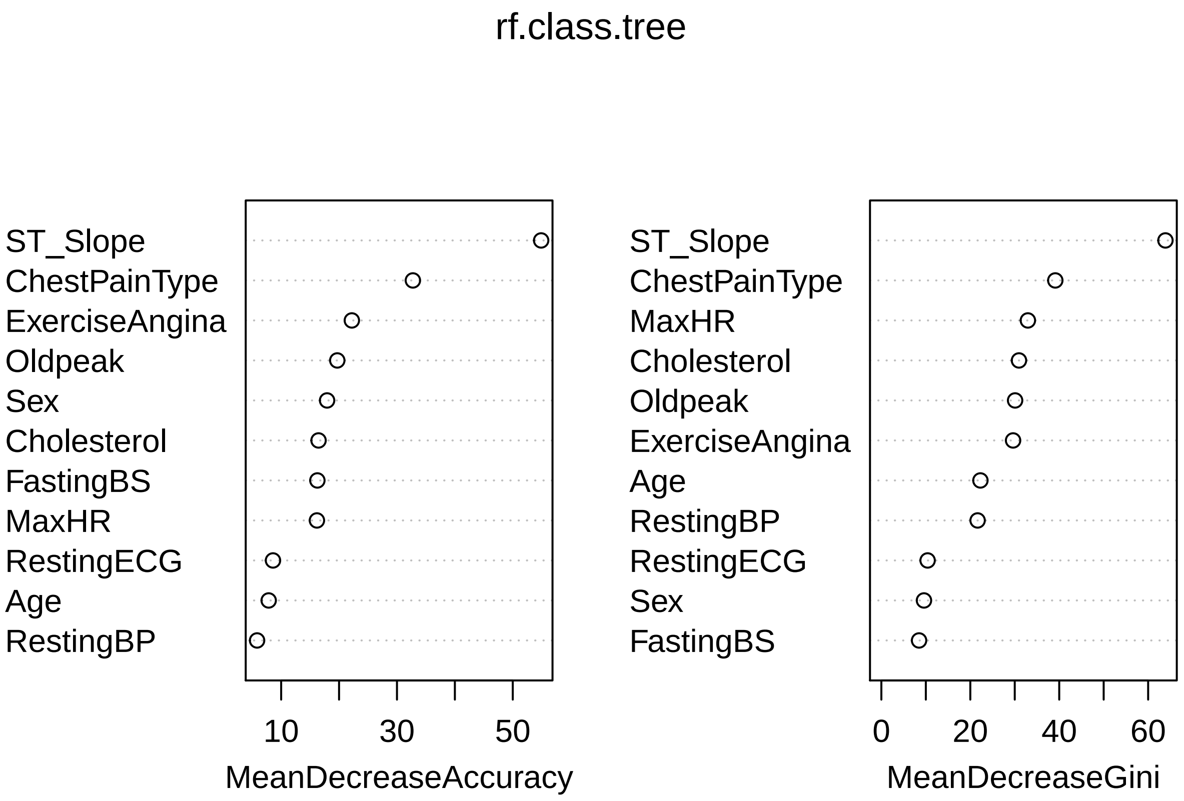
No, so the test MSE of the random forest fit with just the variables present in the logistic regression model and LDA, is of 0.141. Which is actually worse than LDA, but still, although even barely, better than logistic regression.

Let's take a closer look at our optimal random forest (the 1st one fit):

```
importance(rf.class.tree)
```

```
##                       0         1 MeanDecreaseAccuracy MeanDecreaseGini
## Age            7.606081  3.493711             7.842543        22.274020
## Sex           12.413726 13.912112            17.929266         9.583371
## ChestPainType 28.292830 22.116318            32.748089        39.118943
## RestingBP      2.268328  5.654022             5.841480        21.646204
## Cholesterol    14.756852  8.823478           16.455033        30.947340
## FastingBS       9.692856 14.303261           16.253232         8.477526
## RestingECG      5.185747  6.881668            8.587140        10.404412
## MaxHR           3.361769 16.723785           16.174928        32.952375
## ExerciseAngina 16.127171 16.897771           22.208088        29.621633
## Oldpeak        19.065441  9.080074           19.691733        30.075289
## ST_Slope       49.352894 34.869290           54.901592        63.888538
```

```
varImpPlot(rf.class.tree)
```



Based on the random forest, it seems the most important variables in classifying patients as having heart disease or not are **ST_Slope** and **ChestPainType**.

Let's take another look at our top 3 models:

```
log.fit.reduced
```

```
##
## Call:  glm(formula = HeartDisease ~ Sex + ChestPainType + Cholesterol +
##     FastingBS + ExerciseAngina + ST_Slope, family = binomial,
##     data = heart.train)
##
## Coefficients:
##      (Intercept)              SexM  ChestPainTypeATA  ChestPainTypeNAP
##          0.37201           1.35108          -2.08557          -1.82443
##   ChestPainTypeTA       Cholesterol         FastingBS   ExerciseAnginaY
##         -1.36290          -0.00382           1.22952           1.37688
##     ST_SlopeFlat        ST_SlopeUp
##          0.75994          -1.73450
##
## Degrees of Freedom: 611 Total (i.e. Null);  602 Residual
## Null Deviance:       843.6
## Residual Deviance: 410.4     AIC: 430.4
```

```
lda.fit
```

```
## Call:
## lda(HeartDisease ~ Sex + ChestPainType + Cholesterol + FastingBS +
##     ExerciseAngina + ST_Slope, data = heart.train)
##
## Prior probabilities of groups:
##         0         1
## 0.4558824 0.5441176
##
## Group means:
##        SexM ChestPainTypeATA ChestPainTypeNAP ChestPainTypeTA Cholesterol
## 0 0.6738351       0.32974910        0.3333333      0.07168459    223.9247
## 1 0.9099099       0.04204204        0.1321321      0.03603604    177.2853
##   FastingBS ExerciseAnginaY ST_SlopeFlat ST_SlopeUp
## 0 0.1075269       0.1254480    0.1827957  0.7813620
## 1 0.3363363       0.6396396    0.7237237  0.1801802
##
## Coefficients of linear discriminants:
##                          LD1
## SexM              0.587168536
## ChestPainTypeATA -1.130963259
## ChestPainTypeNAP -1.025739185
## ChestPainTypeTA  -0.801288121
## Cholesterol      -0.001930109
## FastingBS         0.604040849
## ExerciseAnginaY   0.856645199
## ST_SlopeFlat      0.378939086
## ST_SlopeUp       -1.161916307
```

```
rf.class.tree
```

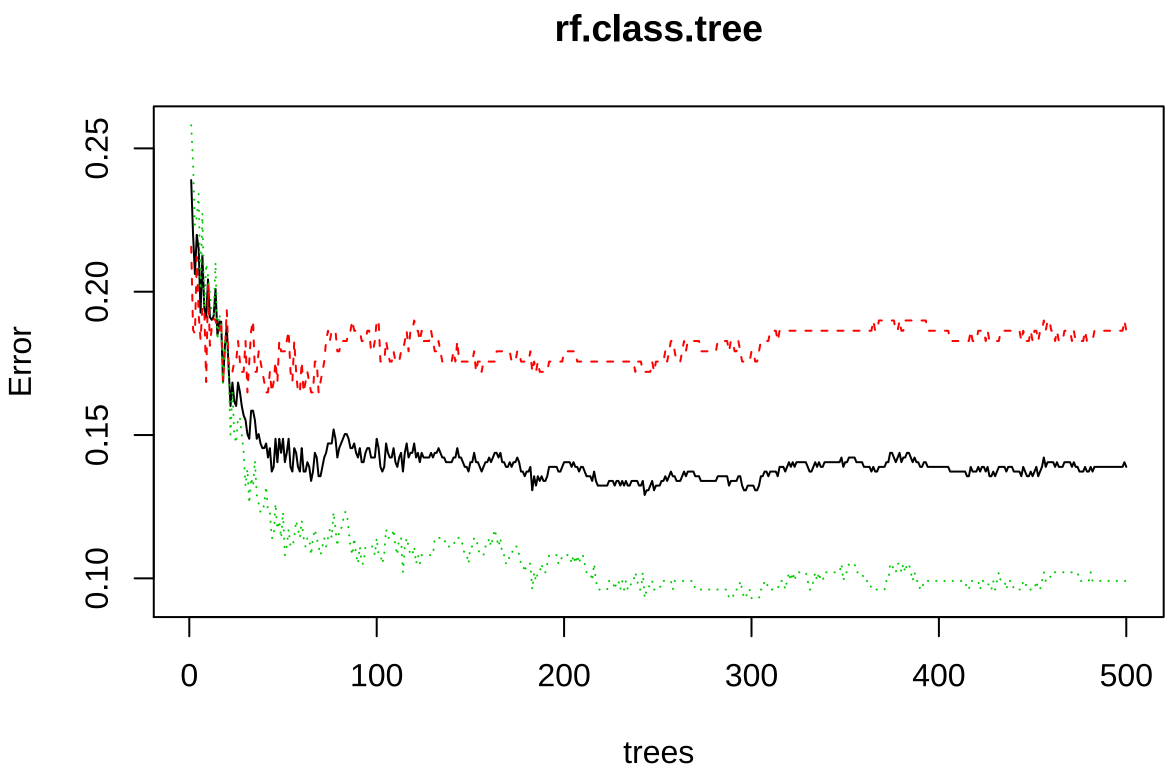
```
##
## Call:
##  randomForest(formula = as.factor(HeartDisease) ~ ., data = heart.train,      mtry = 3, importance =
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 13.89%
## Confusion matrix:
##     0   1 class.error
## 0 227  52   0.1863799
## 1  33 300   0.0990991
```

Our final **preferred** model is the random forest, as it is the most accurate fit, and also the most interpretable (sorta, need to understand more myself).

```
rf.class.tree
```

```
##
## Call:
##  randomForest(formula = as.factor(HeartDisease) ~ ., data = heart.train,      mtry = 3, importance =
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 13.89%
## Confusion matrix:
##     0   1 class.error
## 0 227  52   0.1863799
## 1  33 300   0.0990991
```

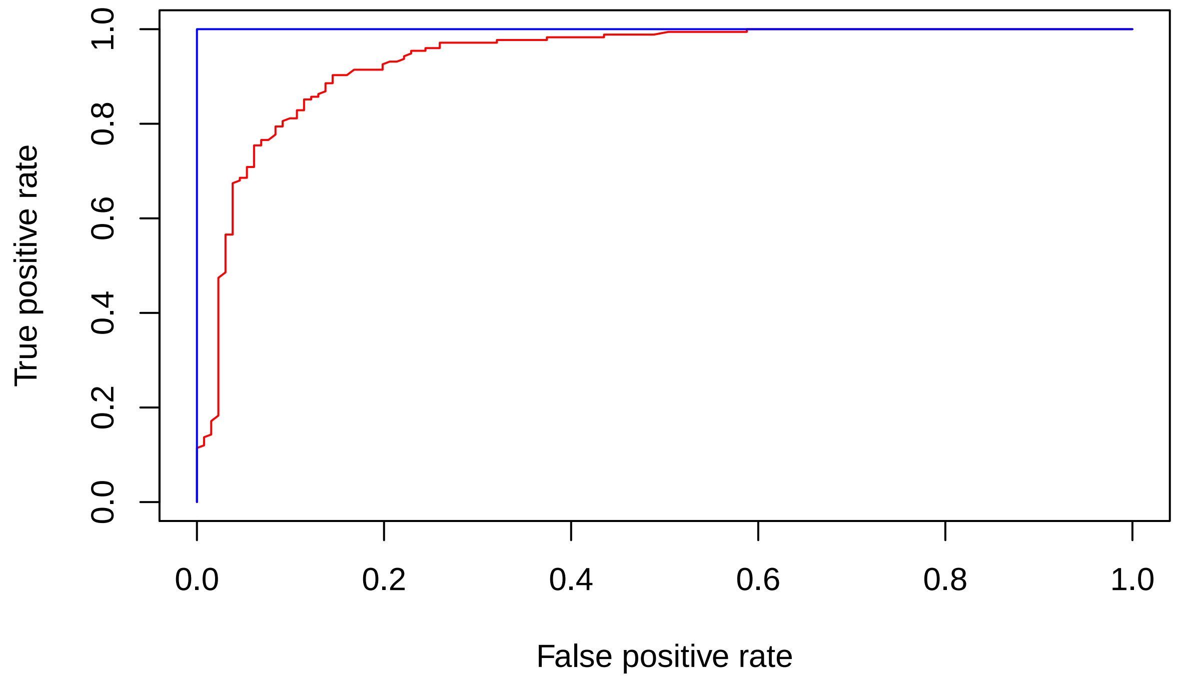
```
plot(rf.class.tree)
```



We see as the number of trees increase, the error rate stabilizes. Our Random Forest chose 500 trees.

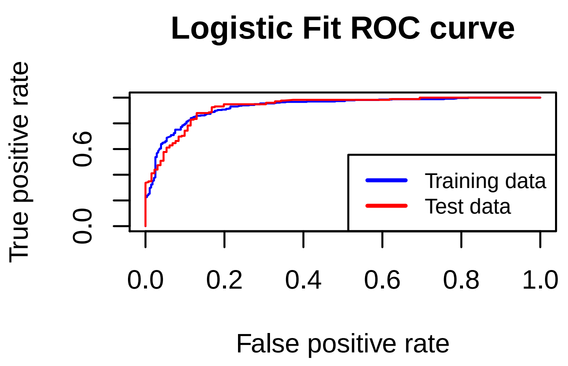
ROC for random forest:

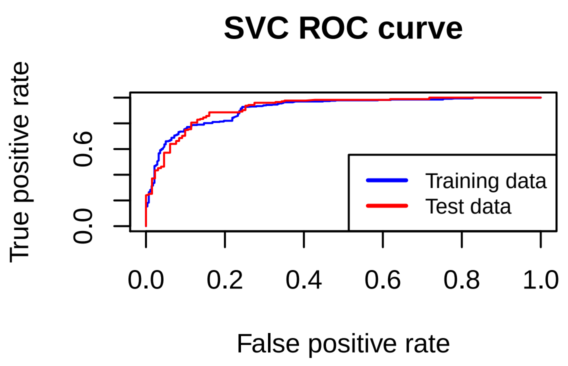
```
#RF on train
rf.train.pr<-predict(rf.class.tree, heart.train, type = "prob")[,2]
rf.train.prediction<-prediction(rf.train.pr, heart.train$HeartDisease)
rf.train.perf<-performance(rf.train.prediction, "tpr", "fpr")
#RF on test
rf.test.pr<-predict(rf.class.tree, heart.test, type = "prob")[,2]
rf.test.prediction<-prediction(rf.test.pr, heart.test$HeartDisease)
rf.test.perf<-performance(rf.test.prediction, "tpr", "fpr")

#The plot
plot(rf.test.perf, col="red")
plot(rf.train.perf, add=T, col="blue")
```

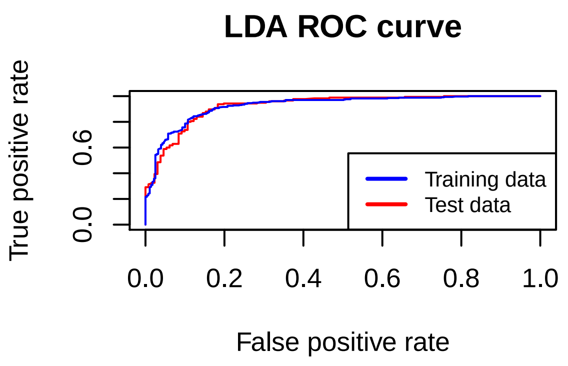


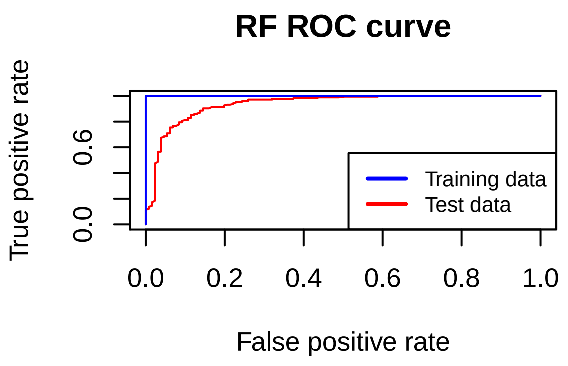
So we can see the 100% accuracy the random forest achieves on the training data, but also the high accuracy it achieves on the test data, which is what interests us.

```r
par(mfrow=c(2,2))
#Log fit
roc_plot(log.fit.reduced$fitted.values, heart.train$HeartDisease, col="blue",main="Logistic Fit ROC cur
roc_plot(log.test.fit$fitted.values, heart.test$HeartDisease, add=T,col=2)
legend("bottomright", legend=c("Training data", "Test data"),
       col=c("blue",2), lty=1, lwd=2, cex=.8)
#SVC
roc_plot(svc.train.fitted, heart.train$HeartDisease, col="blue",main="SVC ROC curve")
roc_plot(svc.test.fitted, heart.test$HeartDisease, add=T,col=2)
legend("bottomright", legend=c("Training data", "Test data"),
       col=c("blue",2), lty=1, lwd=2, cex=.8)
#LDA
plot(lda.test.perf, col="red",main="LDA ROC curve")
plot(lda.train.perf, col="blue", add=T)
legend("bottomright", legend=c("Training data", "Test data"),
       col=c("blue",2), lty=1, lwd=2, cex=.8)
#RF
plot(rf.test.perf, col="red",main="RF ROC curve")
plot(rf.train.perf, add=T, col="blue")
legend("bottomright", legend=c("Training data", "Test data"),
       col=c("blue",2), lty=1, lwd=2, cex=.8)
```









So in comparison, we see the all four methods have similar accuracy on the test data, looking at it in more detail we saw the Random Forest performed the best, followed by LDA, then Logistic Regression and then SVC. KNN was not in its ballpark in this study.

```
testMSEs
```

```
##      [,1]                  [,2]    [,3]    [,4]   [,5]
## [1,] "Logistic Regression" "KNN"   "LDA"   "SVC"  "Random Forest"
## [2,] "0.144"               "0.219" "0.137" "0.15" "0.121"
```

Perhaps if we had more observations, KNN could perform better due to is non-parametric nature, however it is not the case.

Do a few predictions perhaps? also add the ROC curves!!!