CIS3920

FINAL PROJECT

By

William Hall

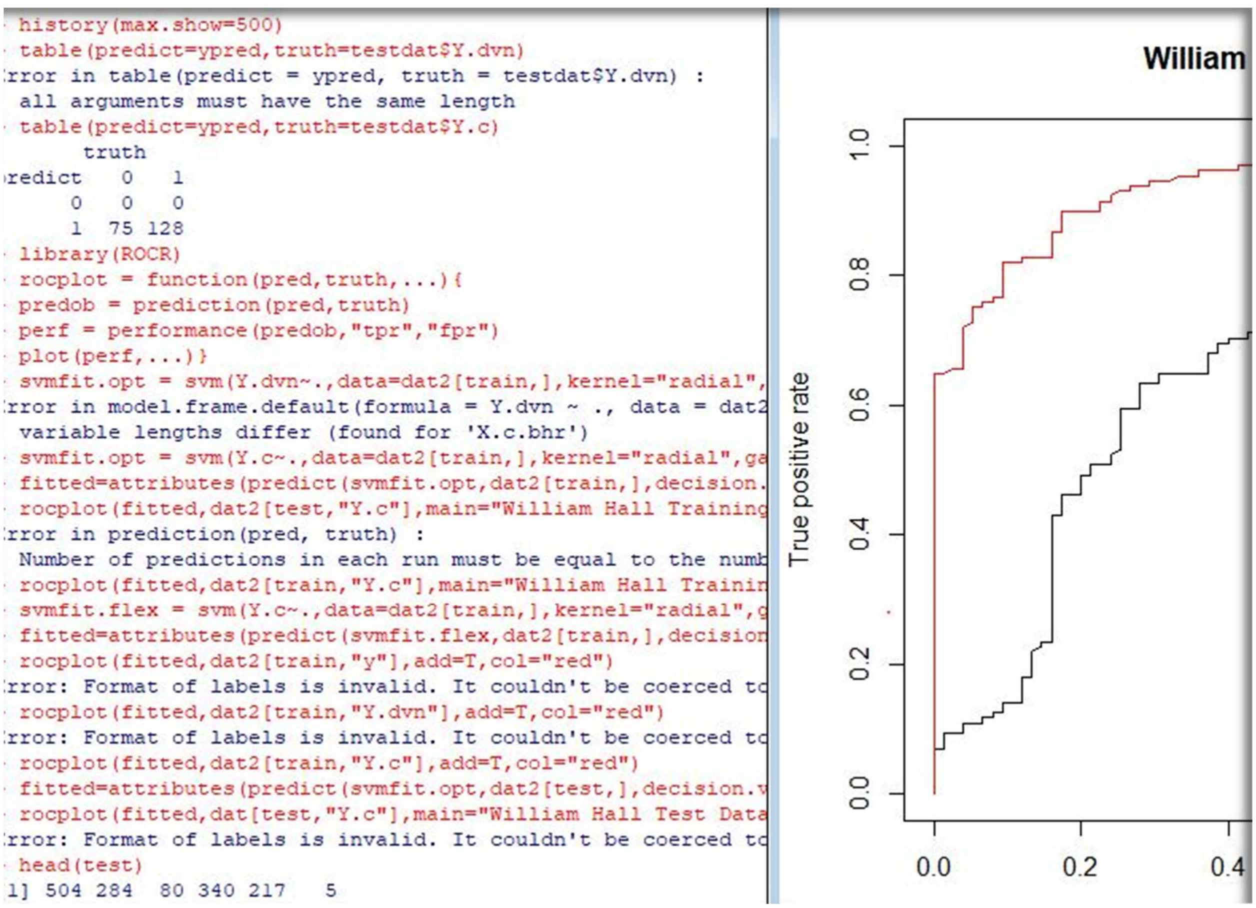


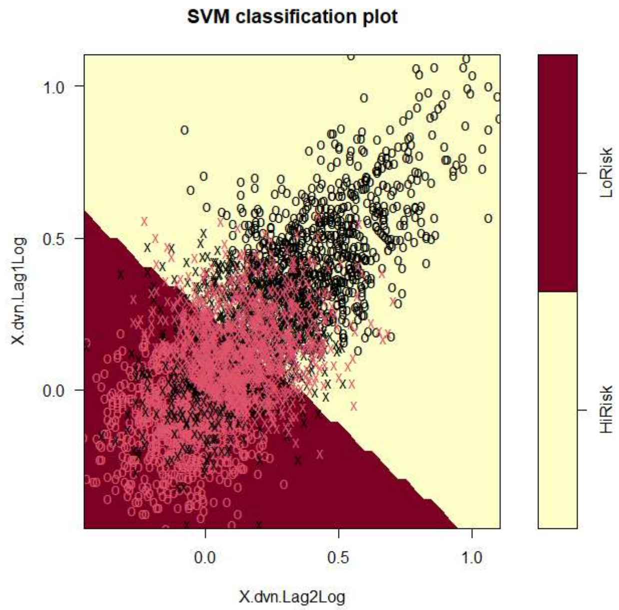
Table of Contents

DVN DATA SET

A. To start, I will assign the variables needed to perform SVM on my DVN data set. In this section, were using the *"linear"* kernel to perform analysis. Also, I will be setting my *cost=10*. When the *cost=* is a small value, the margins become wider making more support vectors. When it is a larger value, margins narrow making less support vectors.

```
> dat2 = data.frame(X.dvn=X.dvn,Y.dvn=as.factor(Y.dvn))
> svmfit2 = svm(Y.dvn~., data = dat2, kernel = "linear", cost =10, scale = FALSE)
> plot(svmfit2,dat2)
```

W



As you can see, there are many support vectors in between the margins with a cost= 10.

B. To find the best value for the *cost=* parameter, we use the *tune()* function to carry this out for us. *tune=* performs 10-fold cross-validation to find the best value for your parameters. In this example we use 0.001,0.01,0.1,1,5,10,100.

```
> tune.out = tune(svm,Y.dvn~.,data=dat2,kernel="linear", ranges=list(cost=c(0.001,0.01,0.1,1,5,10,100)))
> summary(tune.out)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
 0.01

- best performance: 0.2584879

- Detailed performance results:
   cost     error dispersion
1 1e-03 0.2614979 0.02714754
2 1e-02 0.2584879 0.02483171
3 1e-01 0.2595816 0.02384331
4 1e+00 0.2598548 0.02420557
5 5e+00 0.2595816 0.02425886
6 1e+01 0.2593083 0.02389565
7 1e+02 0.2593083 0.02389565
```

As the data shows, the best cost= value would be 0.01. This tells us that this is the lowest cross-validation error rate.

C. Now, we create the variable bestmod that will give us the best model for our data.

```
> bestmod=tune.out$best.model
> summary(bestmod)

Call:
best.tune(method = svm, train.x = Y.dvn ~ ., data = dat2, 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:  0.01

Number of Support Vectors:  2275

 ( 1137 1138 )


Number of Classes:  2

Levels:
 HiRisk LoRisk
```

D. Here we set up our train and test variables to make predictions for our data:

```
> TrainX.dvn = X.dvn[InSample,]
> TrainY.dvn = Y.dvn[InSample]
> TestX.dvn = X.dvn[OutSample,]
> TestY.dvn = Y.dvn[OutSample]
> TestX.dvn[TestY.dvn==1,]=TestX.dvn[TestY.dvn==1,]+1
> testdat = data.frame(x=TestX.dvn,y=as.factor(TestY.dvn))
> testdat = data.frame(X.dvn=TestX.dvn,Y.dvn=as.factor(TestY.dvn))
```

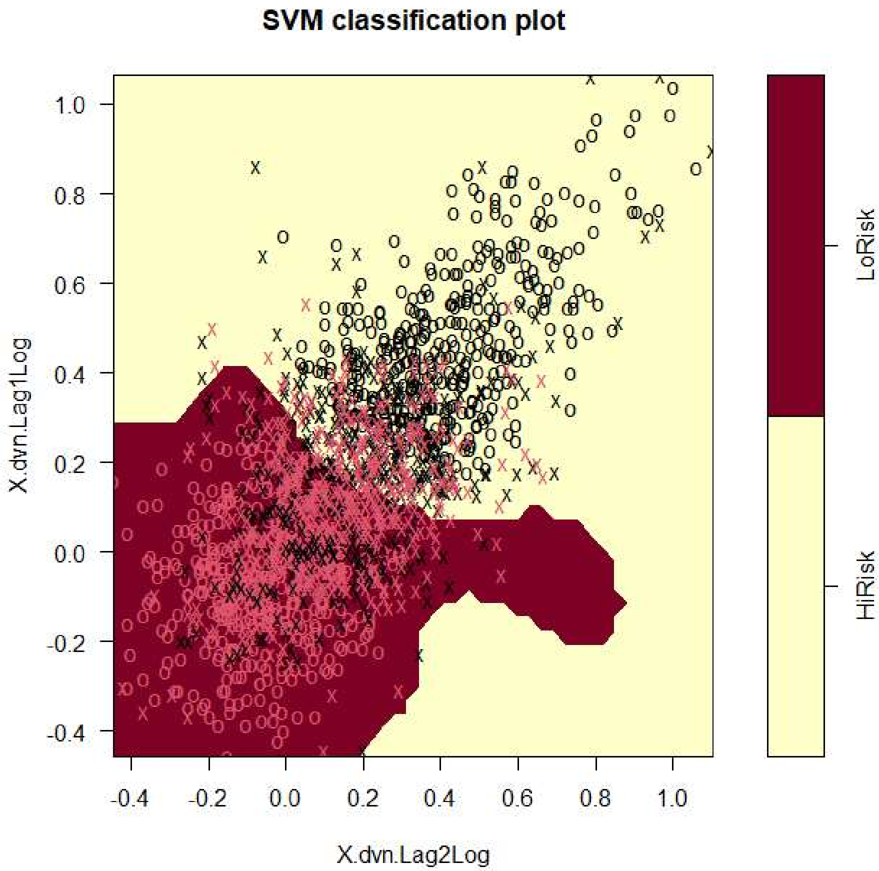
E. Now we do predictions based on our train and test variables using the *predict()* function. Then using the *table()* function to display its accuracy rate.

```
> ypred = predict(bestmod,testdat)
> table(predict=ypred,truth=testdat$Y.dvn)
        truth
predict  HiRisk LoRisk
  HiRisk    656    221
  LoRisk    282    697
```

According to what we have here, the rate of correct predictions is 0.746991

F. Next, we will be using the *"radial"* kernel for the rest of our observations. In this kernel we use the *gamma=* parameter to specify the value of γ (Gamma).

```
> svmfit2 = svm(Y.dvn~., data = dat2[train,], kernel = "radial", gamma = 1, cost =10)
> plot(svmfit2,dat2[train,])
```

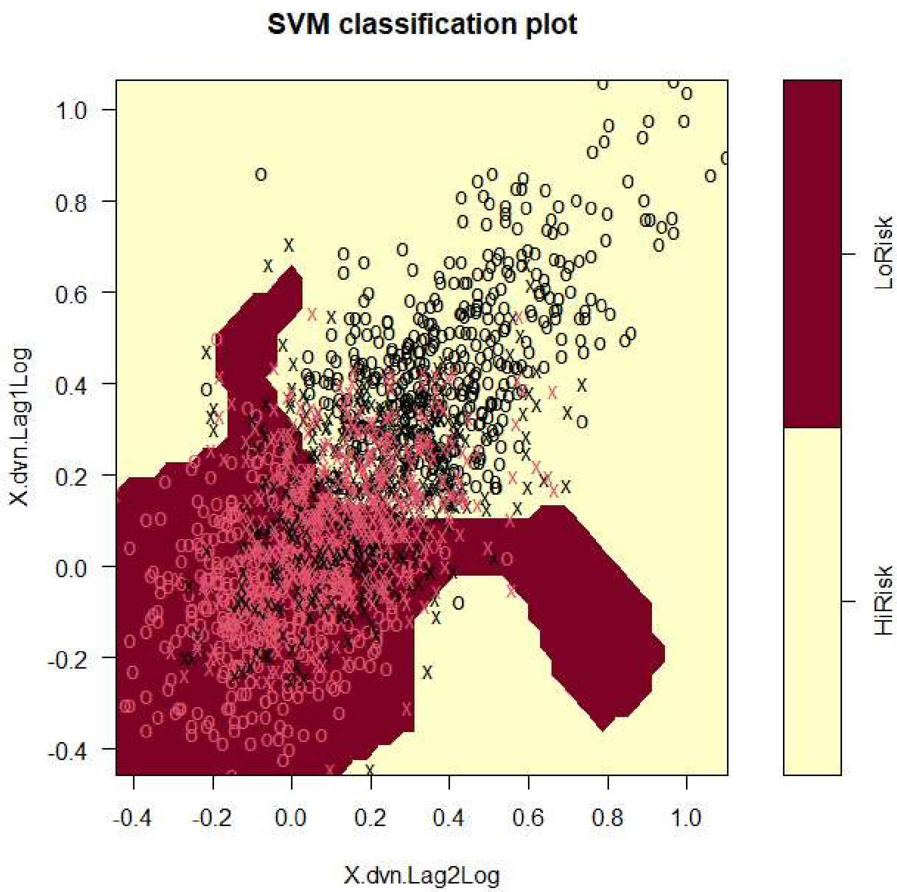


G. When using the summary on our new svmfit, we see that there are less support vectors in this plot. Quite possibly because [train,] is only a segment of the actual data set.

```
> summary(svmfit2)

Call:
svm(formula = Y.dvn ~ ., data = dat2[train, ], kernel = "radial", gamma = 1, cost = 10)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  10

Number of Support Vectors:  939

 ( 471 468 )


Number of Classes:  2

Levels:
 HiRisk LoRisk
```

H. When making the cost= 1000, it creates irregular shapes:

```
> svmfit2 = svm(Y.dvn~., data = dat2[train,], kernel = "radial", gamma = 1, cost =1000)
> plot(svmfit2,dat2[train,])
```



I. We will use *tune()* again for cross-validation to find the best *cost=* and *gamma=* value.
Cost= 0.001,0.01,0.1,1,5,10,100 and gamma= 0.5,1,2,3,4

```
> tune.out = tune(svm,Y.dvn~., data = dat2[train,], kernel = "radial", ranges=list(cost=c(0.001,0.01,
> summary(tune.out)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
   10     4

- best performance: 0.2455556

- Detailed performance results:
    cost gamma     error dispersion
1  1e-03   0.5 0.5127778 0.09590555
2  1e-02   0.5 0.2461111 0.03572203
3  1e-01   0.5 0.2511111 0.02878424
4  1e+00   0.5 0.2500000 0.02868877
```

This shows us the best *cost=* is 10, and the best *gamma=*4.

J. NOTE: Due to an overwhelming amount of errors and not fully grasping what [-train,] is for, I had to use [train,] to make it functional. Otherwise, I cannot continue to the ROC Curves portion.

```
> table(true=dat2[train,"Y.dvn"],pred=predict(tune.out$best.model,newx=dat2[train,]))
        pred
true     HiRisk LoRisk
  HiRisk    652    238
  LoRisk    172    738
```

It shows here that this prediction has a 0.77

K. Comparisons to other classifications:

a. KNN: 0.74

```
> knn.pred = knn(TrainX.dvn,TestX.dvn,TrainY.dvn,25)
> table(knn.pred,TestY.dvn)
          TestY.dvn
knn.pred   HighRisk LowRisk
  HighRisk      675     214
  LowRisk       273     693
```

b. NB: 0.79

```
> table(knn.pred,TestY.dvn)
        TestY.dvn
knn.pred HiRisk LoRisk
  HiRisk    244    157
  LoRisk    233   1222
```

c. NOTE: Unfortunately, I could not get Logistical Regression to work in LN8.

L. ROC Curves:

First, we load the ROCR function to our data. ROC curves compute false and true positive rates for a range of values through predictions. The Roc curves represent the training error rates of the predictions.

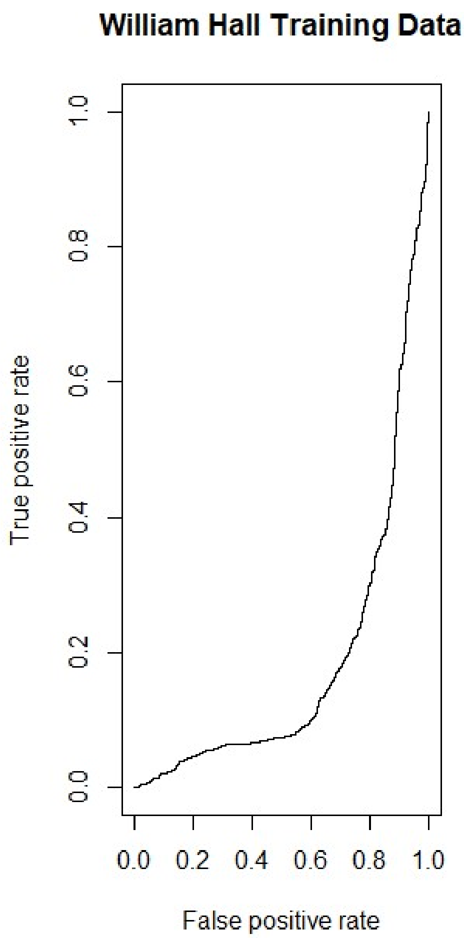
```
> library(ROCR)
> rocplot = function(pred,truth,...){
+ predob = prediction(pred,truth)
+ perf = performance(predob,"tpr","fpr")
+ plot(perf,...)}
```

Then add svmfit.opt and fitted variables:

```
> svmfit.opt = svm(Y.dvn~.,data=dat2[train,],kernel="radial",gamma=2,cost=1,decision.values=T)
> fitted=attributes(predict(svmfit.opt,dat2[train,],decision.values=TRUE))$decision.values
> par(mfrow=c(1,2))
```

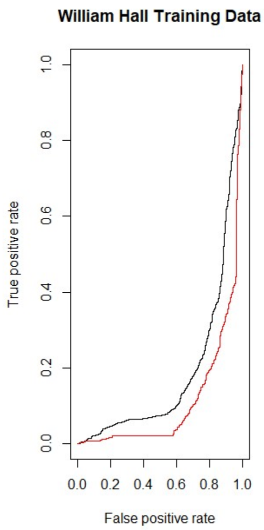
Now we make a rocplot from the training data provided in the variables:

```
> rocplot(fitted,dat2[train,"Y.dvn"],main="William Hall Training Data")
```



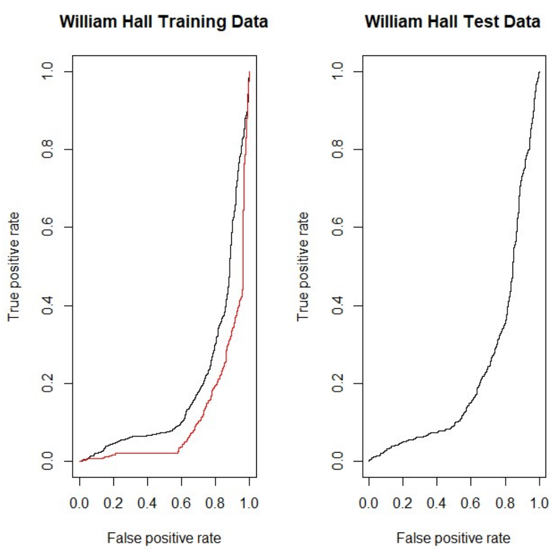
We create the svmfit.flex to show us the support vector classifier in a red line:

```
> svmfit.flex = svm(Y.dvn~.,data=dat2[train,],kernel="radial",gamma=50,cost=1,decision.values=T)
> fitted=attributes(predict(svmfit.flex,dat2[train,],decision.values=T))$decision.values
> rocplot(fitted,dat2[train,"Y.dvn"],add=T,col="red")
```



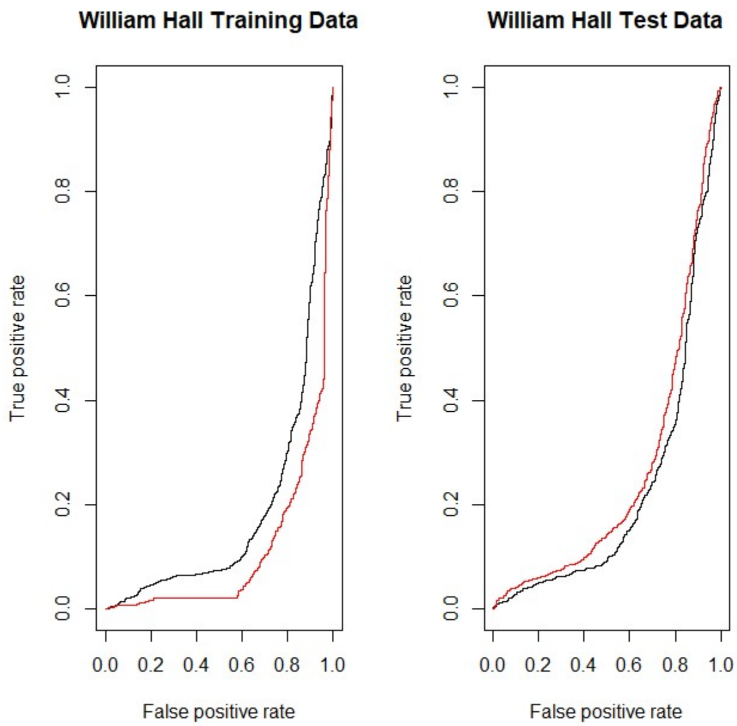
Now to recreate svmfit.opt and fitted variables for the test data to produce ROC curves:

```
> fitted=attributes(predict(svmfit.opt,dat2[test,],decision.values=T))$decision.values
> rocplot(fitted,dat2[test,"Y.dvn"],main="William Hall Test Data")
```



Next, we generate another support vector classifier line in red for our test data:

```
> fitted=attributes(predict(svmfit.flex,dat2[test,],decision.values=T))$decision.values
> rocplot(fitted,dat2[test,"Y.dvn"],add=T,col="red")
```

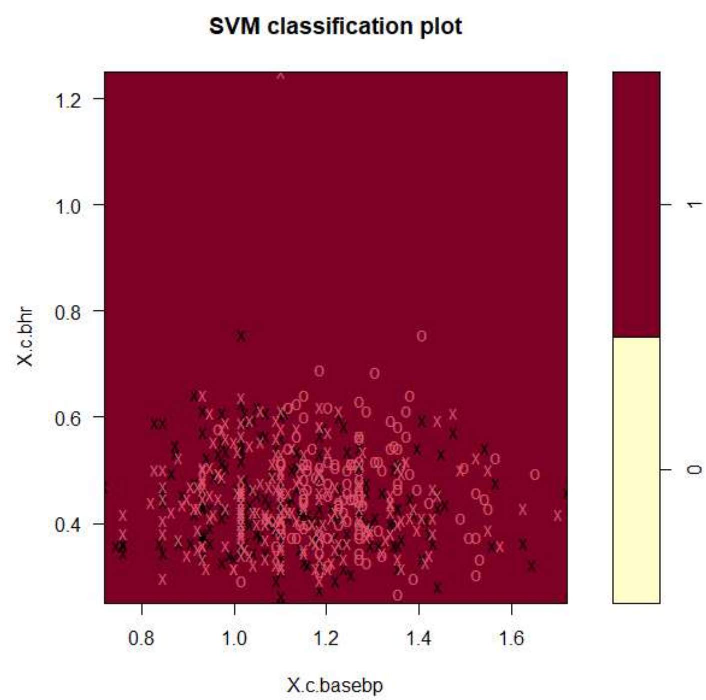


CARDIAC DATA SET

A. NOTE: I am running into a few issues with the Cardiac Data Set.

```
> C = read.csv("Cardiac.csv")
> CCV = cbind(C[,1:2],C[,24,drop=FALSE])
> head(CCV)
        bhr    basebp gender
1 0.5476190 0.8728814      0
2 0.3690476 1.1779661      0
3 0.3690476 1.1779661      0
4 0.5535714 1.0000000      1
5 0.5297619 0.8728814      0
6 0.3452381 0.8474576      0
> tail(CCV)
          bhr    basebp gender
553 0.4523810 1.4406780      1
554 0.4226190 1.2711864      0
555 0.4583333 1.1186441      0
556 0.4642857 0.8474576      0
557 0.3511905 1.1016949      1
558 0.5833333 1.1864407      1
> X.c = CCV[,1:2]
> Y.c = CCV[,3]
> dat2 = data.frame(X.c=X.c,Y.c=as.factor(Y.c))
> svmfit2 = svm(Y.c~., data = dat2, kernel = "linear", cost =10, scale = FALSE)
> plot(svmfit2, dat2)
```

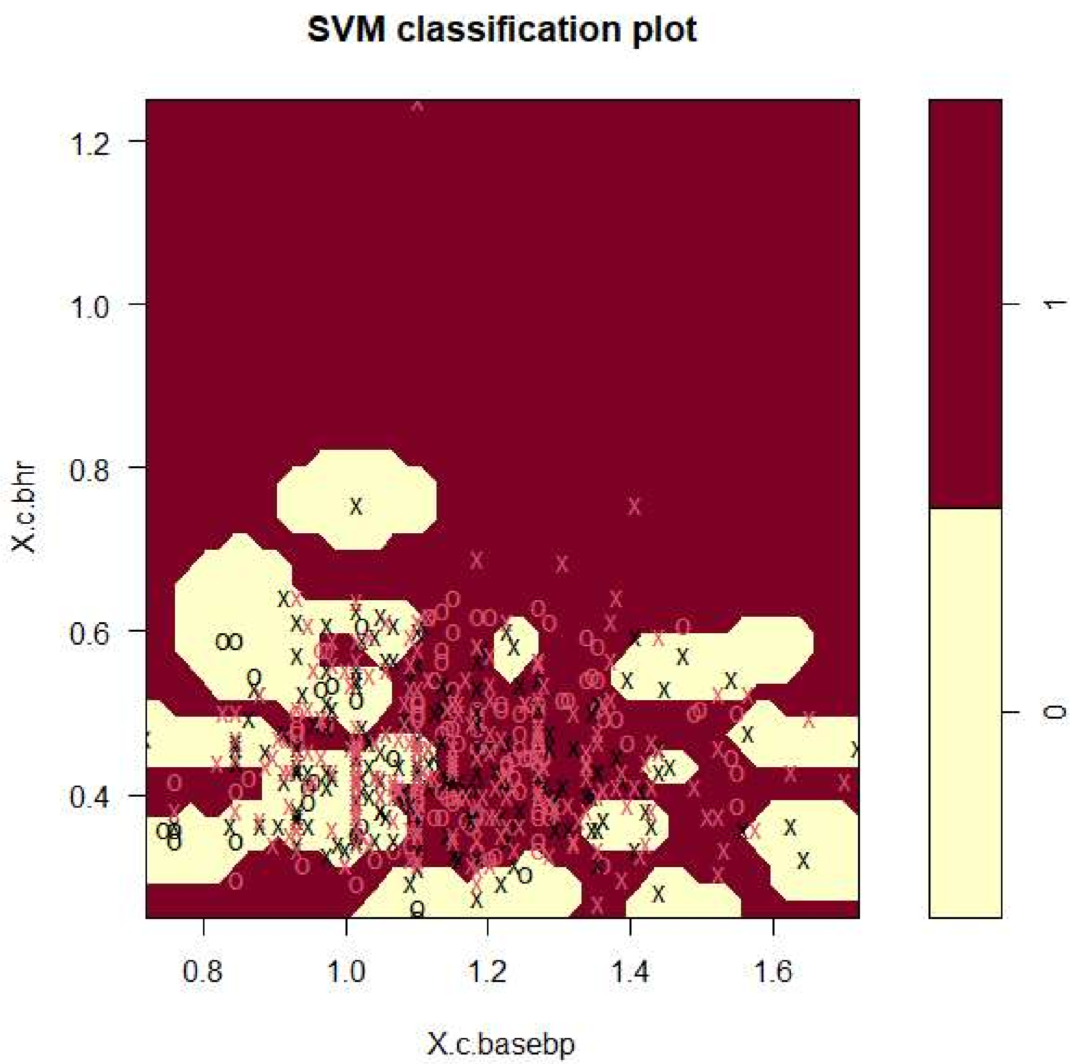
Entering this gives me a very strange plot:



I noticed that all of the data points are grouped together without the second classifier shown. Also, there's no seperation of the 2 classes by a hyperplane. I believe this data is non-linear.

B. When data is non-linear, different kernels must be used to try to seperate the 2 classes. The first non-linear kernel I will try is the *"radial"* kernel:

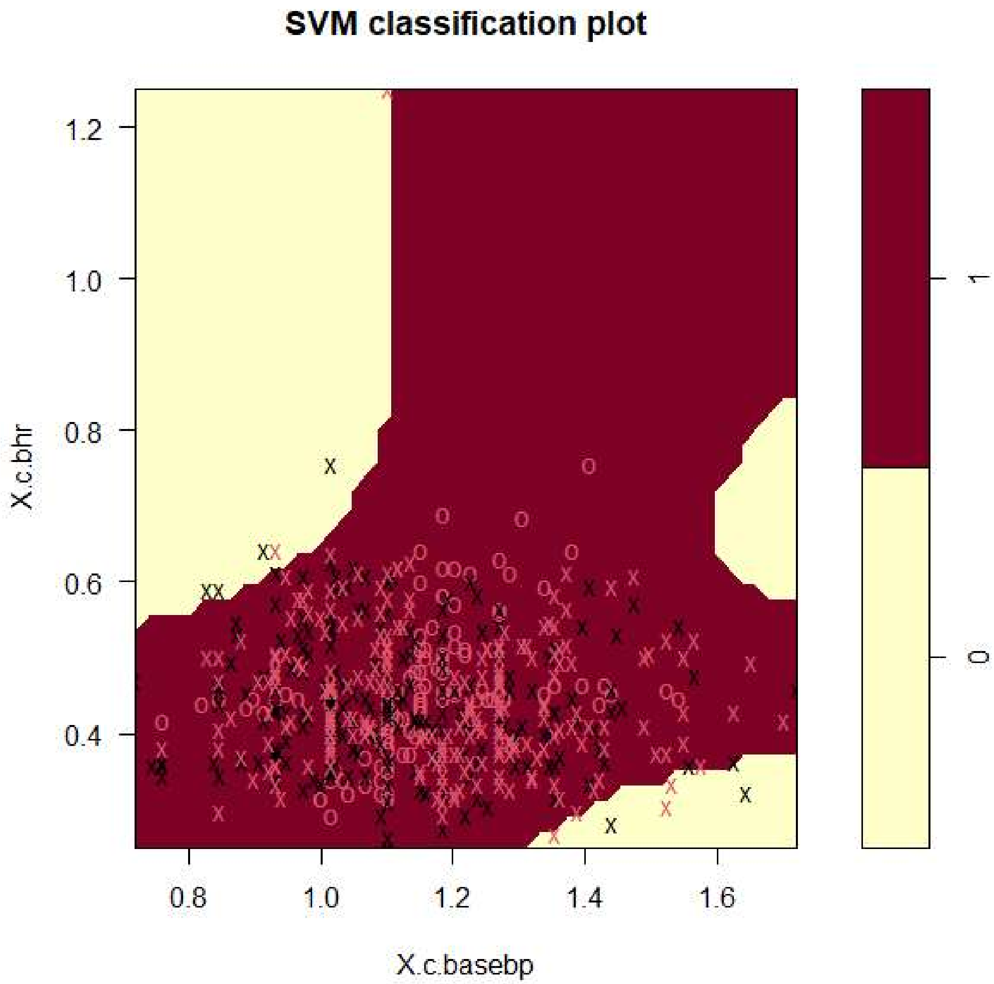
```
> svmfit2 = svm(Y.c~., data = dat2, kernel = "radial", cost =10, gamma=4)
> plot(svmfit2, dat2)
```



In my opinion, this doesn't appear to be correct. The 0 classifier zones are splotches all over the plot. Also, the support vectors are mixed with the regular data points showing that the margins and support vector classifiers aren't separating them correctly.

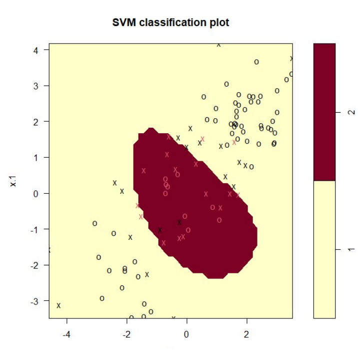
C. Next, I'm going to try to use polynomial kernel to see if it works better than the previous kernels:

```
> svmfit2 = svm(Y.c~., data = dat2, kernel = "polynomial", cost =100, degree=4)
> plot(svmfit2, dat2)
```



I don't have an idea what is happening in this data set. The 0 classification area is separated in 3 parts of this plot.

D. When visualizing a non linear plot, I envisioned a plot like this one below.



E. Going back to the linear kernel, I want a summary of how many support vectors are in my plot.

```
> svmfit2 = svm(Y.c~., data = dat2, kernel = "linear", cost =10, scale = FALSE)
> summary(svmfit)

Call:
svm(formula = y ~ ., data = dat, kernel = "radial", cost = 10, gamma = 1)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  10

Number of Support Vectors:  104

 ( 40 32 32 )


Number of Classes:  3

Levels:
 0 1 2
```

F. Next, I want to find out what is the best cost for this data using *tune().*
(0.001,0.01,0.1,1,5,10,100)

```
> tune.out = tune(svm,Y.c~.,data=dat2,kernel="linear", ranges=list(cost=c(
> summary(tune.out)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
  cost
 0.001

- best performance: 0.3942532

- Detailed performance results:
   cost     error dispersion
1 1e-03 0.3942532 0.08279611
2 1e-02 0.3942532 0.08279611
3 1e-01 0.3942532 0.08279611
4 1e+00 0.3942532 0.08279611
5 5e+00 0.3942532 0.08279611
6 1e+01 0.3942532 0.08279611
7 1e+02 0.3942532 0.08279611
```

According to this output, 0.001 is the best *cost=* value.

```
> bestmod=tune.out$best.model
> summary(bestmod)

Call:
best.tune(method = svm, train.x = Y.c ~ ., data =
    0.01, 0.1, 1, 5, 10, 100)), kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.001

Number of Support Vectors:  441

 ( 220 221 )


Number of Classes:  2

Levels:
 0 1
```

G. It is time to set up my train and test variables to make my predictions.

```
> TrainX.c = X.c[InSample,]
> TrainY.c = Y.c[InSample]
> TestX.c = X.c[OutSample,]
> TestY.c = Y.c[OutSample]
> TestX.c[TestY.c==1,]=TestX.c[TestY.c==1,]+1
> testdat = data.frame(X.c=TestX.c,Y.c=as.factor(TestY.c))
> ypred = predict(bestmod,testdat)
> table(predict=ypred,truth=testdat$Y.c)
       truth
predict   0   1
      0   0   0
      1  75 128
```

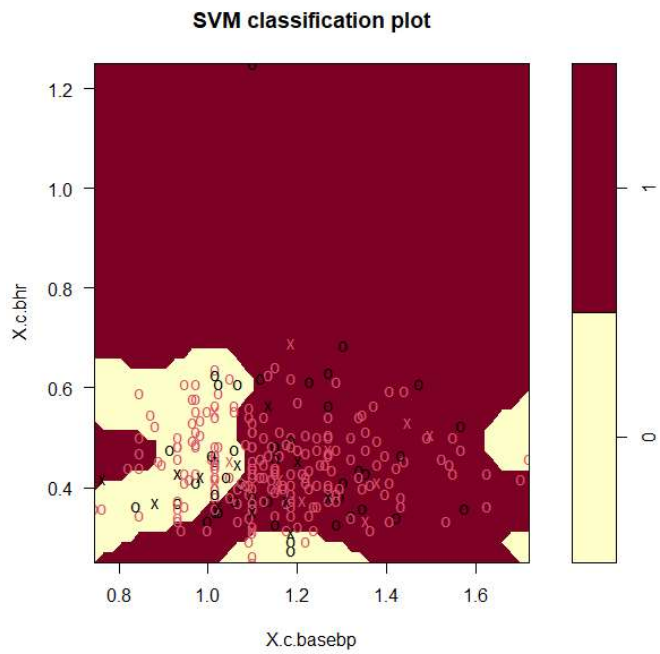
According to the results, truth has an accuracy of 0.63. Looking at how 1 had all predictions going there, I had to do a little investigating:

```
> head(Y.c)
[1] 0 0 0 1 0 0
> tail(Y.c)
[1] 1 0 0 0 1 1
```

By looking at the results, I feared that my Y variable must have all 1's. Checking the head and tail shows otherwise.

H. Now I will work with the radial kernel and see if I could obtain better results:

```
> svmfit2 = svm(Y.c~., data = dat2[train,], kernel = "radial", gamma = 1, cost $
> plot(svmfit2,dat2[train,])
```



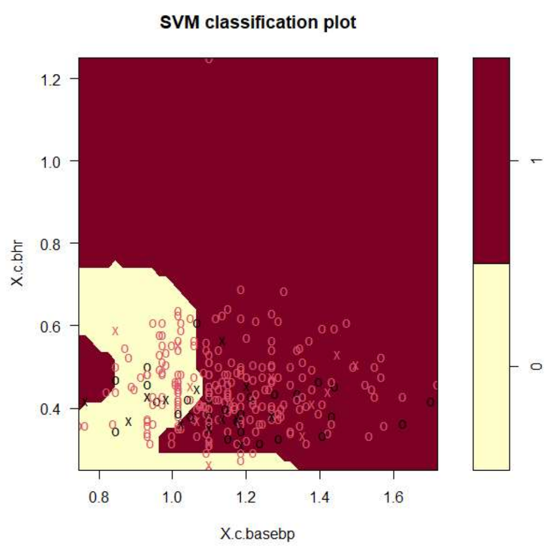
This plot is from the train data. 0 doesn't appear to have a solid classification area still. Creating a summary of svmfit may answer more questions I may have.

```
> summary(svmfit2)

Call:
svm(formula = Y.c ~ ., data = dat2[train, ], kernel = "radial", gamma = 1,
    cost = 10)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  10

Number of Support Vectors:  205

 ( 108 97 )


Number of Classes:  2

Levels:
 0 1
```

I. Perhaps using the *tune()* function will give me more information about the best cost=, and gamma= values to use: (cost=c(0.001,0.01,0.1,1,5,10,100),gamma=c(0.5,1,2,3,4)

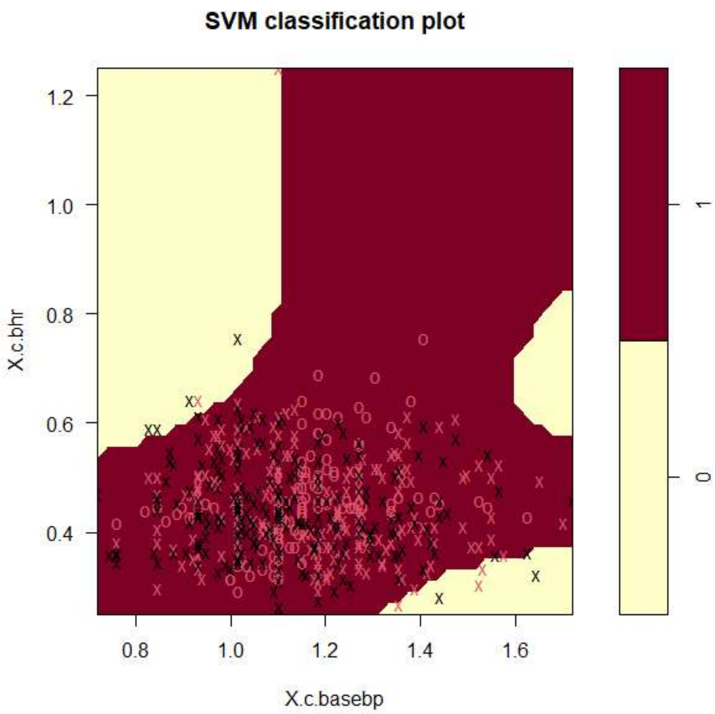
```
> tune.out = tune(svm,Y.c~., data = dat2[train,], kernel = "radial", ranges=
> summary(tune.out)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
    5   0.5

- best performance: 0.3539724

- Detailed performance results:
    cost gamma     error dispersion
1  1e-03   0.5 0.4134901 0.15808822
2  1e-02   0.5 0.4134901 0.15808822
3  1e-01   0.5 0.4134901 0.15808822
4  1e+00   0.5 0.3820683 0.12024768
```

The summary displays the best *cost*=5 and the best *gamma*=0.5….
Using those parameters shows slight improvement to the 0 classification area.



J. Finally, I will try the polynomial kernel to work for this data set with the train data:

```
> svmfit2 = svm(Y.c~., data = dat2[train,], kernel = "polynomial", gamma = 1, cost =1,degree =4)
> plot(svmfit2,dat2[train,])
```

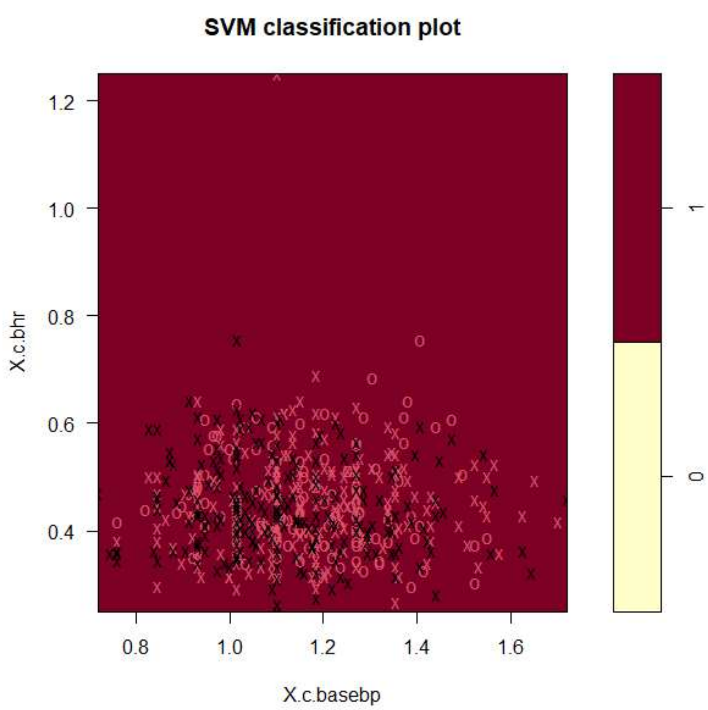



```
> summary(svmfit2)

Call:
svm(formula = Y.c ~ ., data = dat2[train, ], kernel = "polynomial", gamma = 1, cost = 1,
    degree = 4)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  1
     degree:  4
     coef.0:  0

Number of Support Vectors:  442

 ( 222 220 )


Number of Classes:  2

Levels:
 0 1
```

K. Now I will try to use the *tune()* function to find the best parameters for *cost=,degree=*: (cost=c(0.001,0.01,0.1,1,5,10),degree=c(0.5,1,2,3,4)

```
> tune.out = tune(svm,Y.c~., data = dat2[train,], kernel = "polynomial", ranges=list
> summary(tune.out)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
  cost degree
 0.001    0.5

- best performance: 0.3941883

- Detailed performance results:
    cost degree     error dispersion
1  1e-03    0.5 0.3941883 0.05011499
2  1e-02    0.5 0.3941883 0.05011499
3  1e-01    0.5 0.3941883 0.05011499
4  1e+00    0.5 0.3941883 0.05011499
5  5e+00    0.5 0.3941883 0.05011499
```

According to the results, *tune()* has determined that a cost= of 0.001 and degree= of 0.5 are the best parameters for this data set.



I'm not sure if this correct. The 0 classification is completely gone. This is the summary of svmfit with the best parameters

```
> summary(svmfit2)

Call:
svm(formula = Y.c ~ ., data = dat2[train, ], kernel = "polynomial", cost = 0.001,
    degree = 0.5)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  0.001
     degree:  0.5
     coef.0:  0

Number of Support Vectors:  440

 ( 220 220 )


Number of Classes:  2

Levels:
 0 1
```

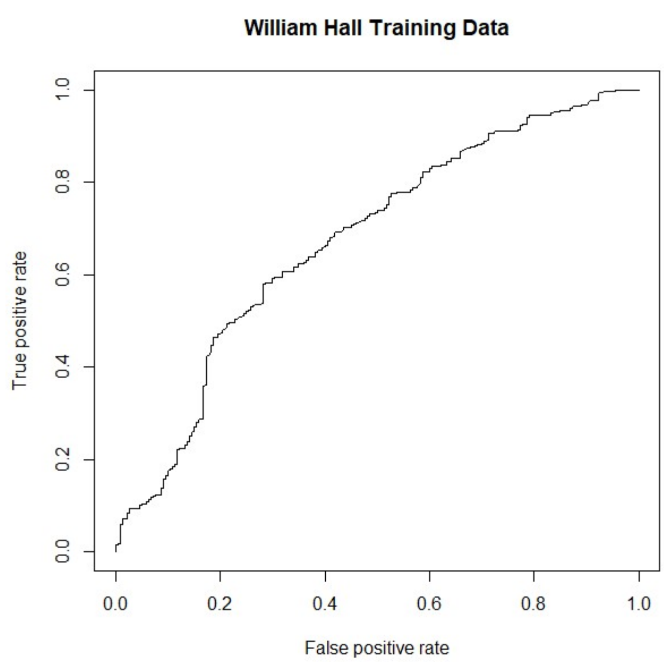
L. ROC Curves:

First, we load the ROCR function

```
> library(ROCR)
> rocplot = function(pred,truth,...){
+ predob = prediction(pred,truth)
+ perf = performance(predob,"tpr","fpr")
+ plot(perf,...)}
```

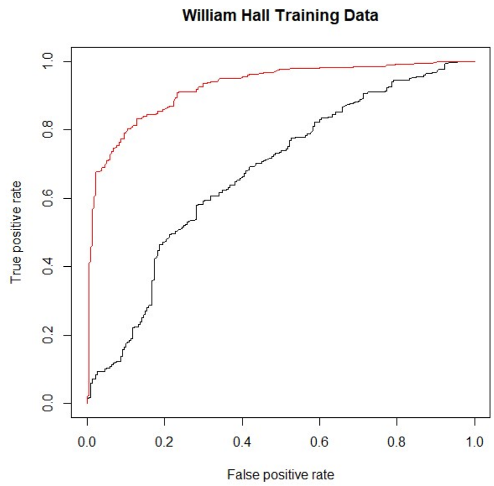
M. I set up svmfit.opt and fitted variables to use rocplot:

```
> svmfit.opt = svm(Y.c~.,data=dat2[train,],kernel="radial",gamma=2,cost=1,decision.values=T)
> fitted=attributes(predict(svmfit.opt,dat2[train,],decision.values=TRUE))$decision.values
> rocplot(fitted,dat2[train,"Y.c"],main="William Hall Training Data")
```



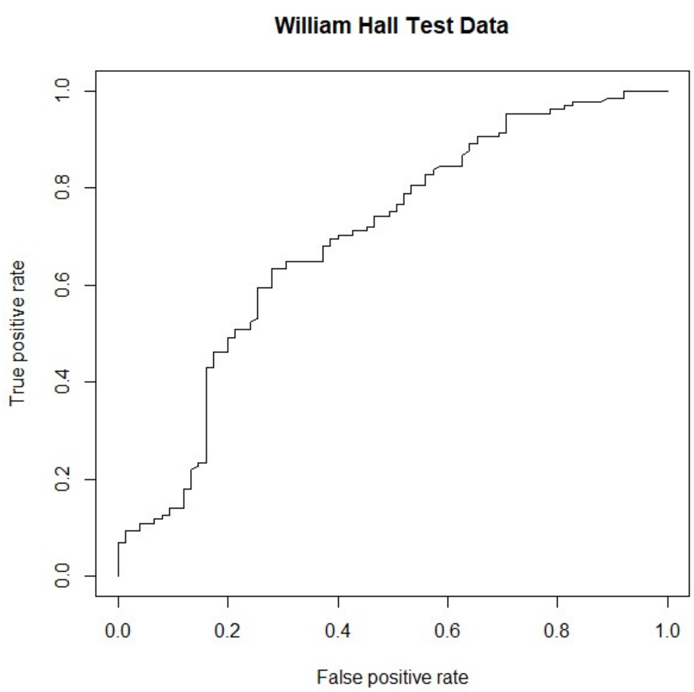
N. Now to draw the red line:

```
> svmfit.flex = svm(Y.c~.,data=dat2[train,],kernel="radial",gamma=50,cost=1,decision.values=T)
> fitted=attributes(predict(svmfit.flex,dat2[train,],decision.values=T))$decision.values
> rocplot(fitted,dat2[train,"Y.c"],add=T,col="red")
```

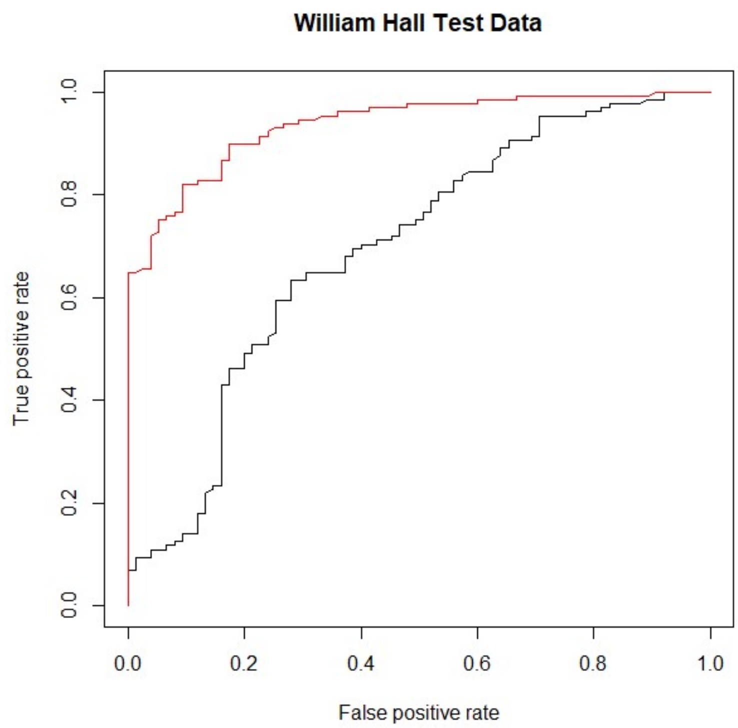


O. Now for the test data:

```
> fitted=attributes(predict(svmfit.opt,dat2[test,],decision.values=T))$decision.values
> rocplot(fitted,dat2[test,"Y.c"],main="William Hall Test Data")
```



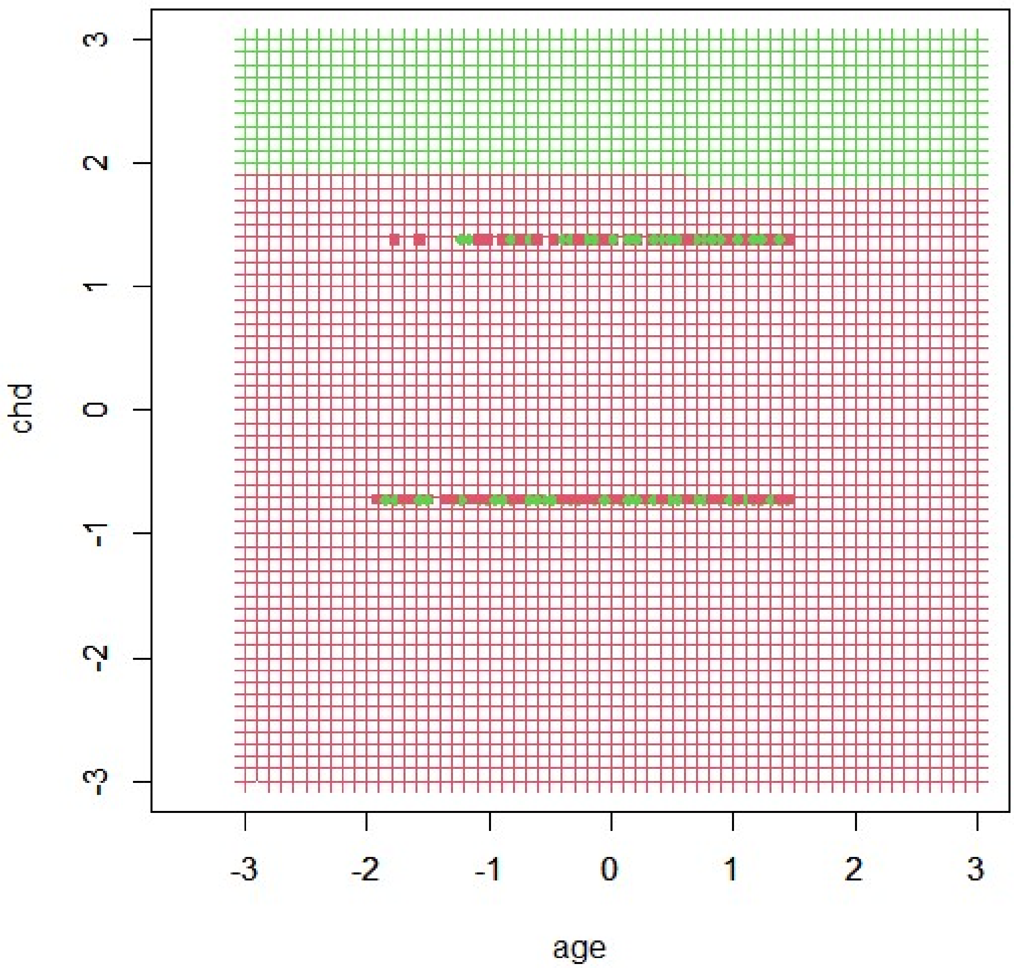
And now for the red line:



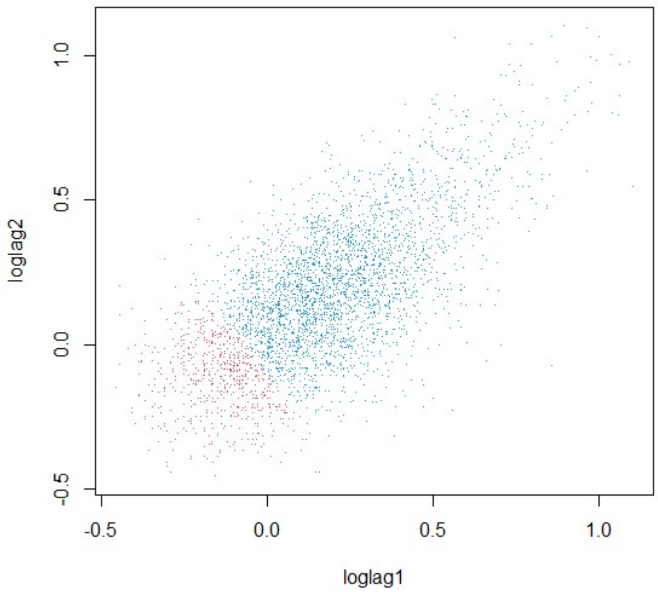
In both cases, the support vector classifier line (red) is well above the LDA line showing better performance than my stock data.

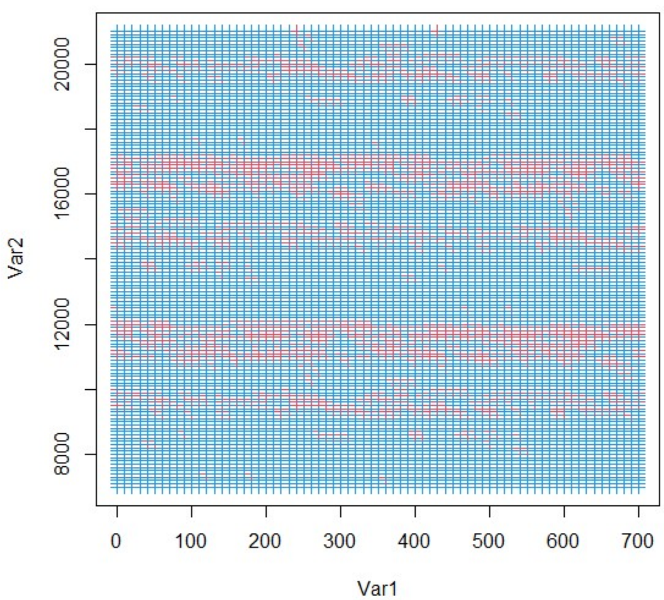
COMPARISONS FROM PREVIOUS EXERCISES

LN8: Linear Regression



LN7:


loglag2
loglag1
-0.5
0.0
0.5
1.0


Var2
Var1
8000
12000
16000
20000
0
100
200
300
400
500
600
700

LN5: