8-7-21

Final Project Assignment

The Prediction of Classificational Boundary

Baruch College STA 3920-S2CA

Junlin Wu

Prof. Lawrence Tatum

Table of Contents

Part A Apply SVM on DPZ risk data.

For this final project, I want to apply the Support Vector Machines (SVM) method to the Domino's Pizza, Inc. (DPZ) stock daily range data with the classification of High risk/Low risk.

Section 1 Data Preparation

First, I downloaded the DPZ historical daily price data since 1-1-2006 to 8-6-2021 from Yahoo Finance website[1]. And I use Excel to create a new column called range, which is by subtracting the Low column from the High column. Then I create Lag1 and Lag2 two columns. The Lag1 column is the previous day range, the Lag2 column is the range of two days earlier. And I also create a column called HiLo, which means that day's range is higher than or lower than the median of all range values. If that day range is higher than the median of range, that day is High risk; otherwise, that day is Low risk. Shows on the right:

HiLo	Lag1	Lag2
Lo	0.25	0.680001
Lo	0.229999	0.25
Lo	0.440001	0.229999
Lo	0.43	0.440001
Hi	0.319999	0.43
Lo	1.83	0.319999
Lo	0.309999	1.83
Lo	0.15	0.309999
Lo	0.16	0.15
Lo	0.42	0.16

I save these three columns as a new csv file, call "DPZ.csv", then read it into R, name it as DPZ. And I check it with head, tail and dim function, commands show on the right. We can see that there 3924 rows and 3 columns in this DPZ data frame.

```
> DPZ=read.csv("DPZ.csv")
> head(DPZ,3)
  HiLo     Lag1     Lag2
1   Lo 0.250000 0.680001
2   Lo 0.229999 0.250000
3   Lo 0.440001 0.229999
> tail(DPZ,3)
     HiLo      Lag1     Lag2
3922   Hi 13.010009 11.29004
3923   Hi  6.500000 13.01001
3924   Hi  9.140015  6.50000
> dim(DPZ)
[1] 3924    3
```

Then, in order to remove the heteroscedasticity from the Lag1 and Lag2 variables, I will apply natural logarithm to transform these values. I name this after-transformation data frame as "lnDPZ.X". Shows on right:

```
> lnDPZ.X = log(DPZ[,2:3])
```

Next, I create a new data frame call "*dat*" with the lnDPZ.X and the first column of DPZ data frame. As shows on the right, there are also 3924 rows and 3 columns here.

Must be careful here, as I highlighted with a red box on the right, the class of the first column of DPZ is character. According to the book *An Introduction to Statistical Learning*, "Note that in order for the svm() function to perform classification (as opposed to SVM-based regression), we must encode the response as a

```
> dat = data.frame(x=lnDPZ.X, y=as.factor(DPZ[,1]))
> class(DPZ[,1])
[1] "character"
> head(dat,5)
      x.Lag1     x.Lag2  y
1 -1.3862944 -0.3856610 Lo
2 -1.4696803 -1.3862944 Lo
3 -0.8209783 -1.4696803 Lo
4 -0.8439701 -0.8209783 Lo
5 -1.1394374 -0.8439701 Hi
> dim(dat)
[1] 3924    3
```

[1]Source: https://finance.yahoo.com/quote/DPZ/history?period1=1136073600&period2=1628294400&interval=1d&filter=history&frequency=1d&includeAdjustedClose=true

factor variable."[2] So I apply as.factor to this column in order to encode it into factor and put it into the new data frame.

With the DPZ risk data frame, the "*dat*" prepared, I now decide to shuffle these 3924 rows, for 2000 rows as training datasets, and the rest 1924 as test datasets.

```
> train=sample(3924,2000)
> dat.train=dat[train,]
> dat.test=dat[-train,]
```

Section 2 Apply SVM (linear)

Then, I will try to run the *svm()* function with the *kernel="linear"*, which means the decision boundary between the two classes is linear. And I will try the parameter cost=10 first.

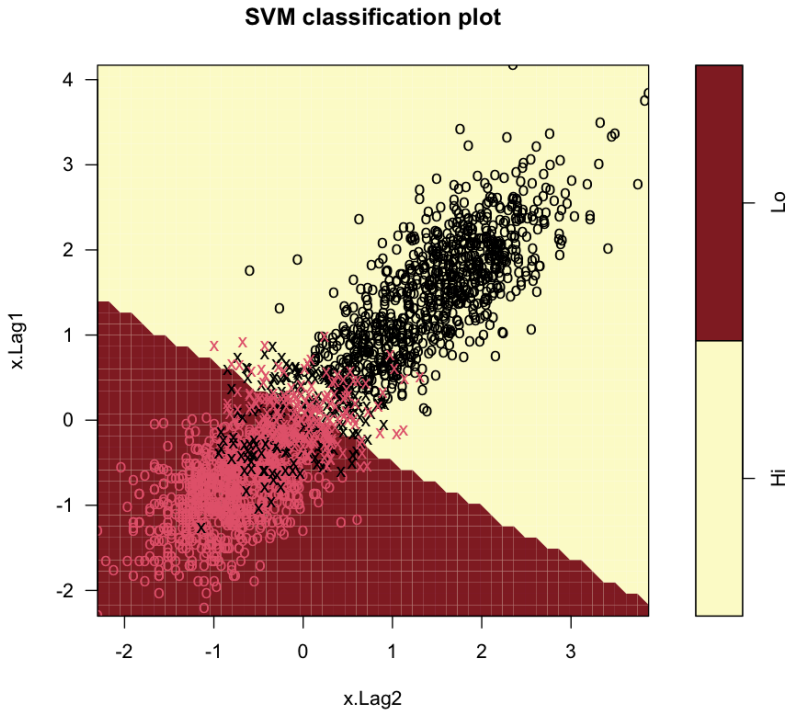
Unfortunately, we got an error message here, that is because I forgot to load *e1071* package before running the *svm()* function. The *svm()* function is included in the *e1071* package. We must load it first.

```
> svmfit=svm(y~., data=dat.train, kernel="linear", cost=10, scale=FALSE)
Error in svm(y ~ ., data = dat.train, kernel = "linear", cost = 10, scale = FALSE) :
  could not find function "svm"
```

I load the *e1071* package, then run the *svm()* function as previous again. Then I plot the support vector classifier obtained. Show as below:

We can see here the support vectors are plotted as crosses and the remaining observations are plotted as circles. The red crosses on the yellow region and the black crosses on the red region are misclassified observations.

```
> library(e1071)
> svmfit=svm(y~., data=dat.train, kernel="linear", cost=10, scale=FALSE)
> plot(svmfit, dat.train)
```



[2] Source: *An Introduction to Statistical Learning*,
by James, Witten, Hastie, and Tibshirani; 4th printing, Springer Press, page 359.

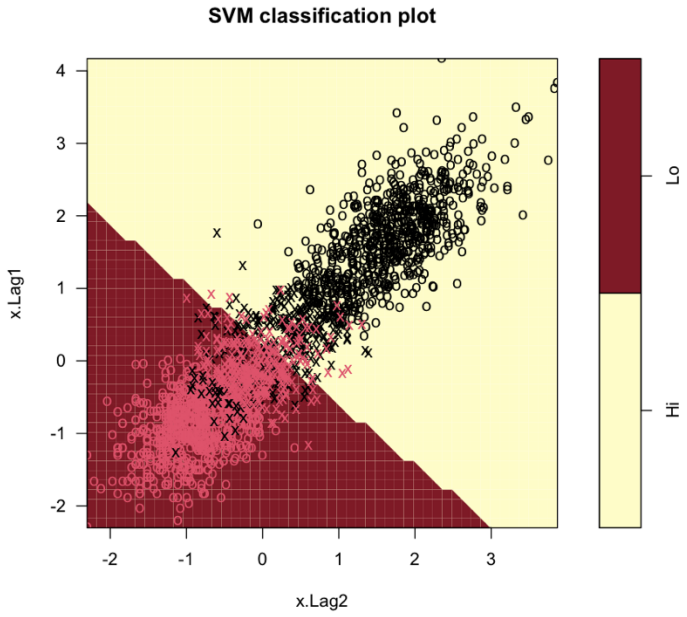
To get more information of the support vector classifier, I am using the *summary()* command on the right:

As I highlighted with a red box on the right. There are 436 support vectors. Half of them, 218 in one class and the other 218 in another class.

```
> summary(svmfit)

Call:
svm(formula = y ~ ., data = dat.train, kernel = "linear", cost = 10,
    scale = FALSE)


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

Number of Support Vectors:  436

 ( 218 218 )


Number of Classes:  2
```

Now I will try to use a smaller value of *cost* parameter, to see what will be changed.

```
> svmfit=svm(y~., data=dat.train, kernel="linear", cost=0.01, scale=FALSE)
> plot(svmfit, dat.train)
```

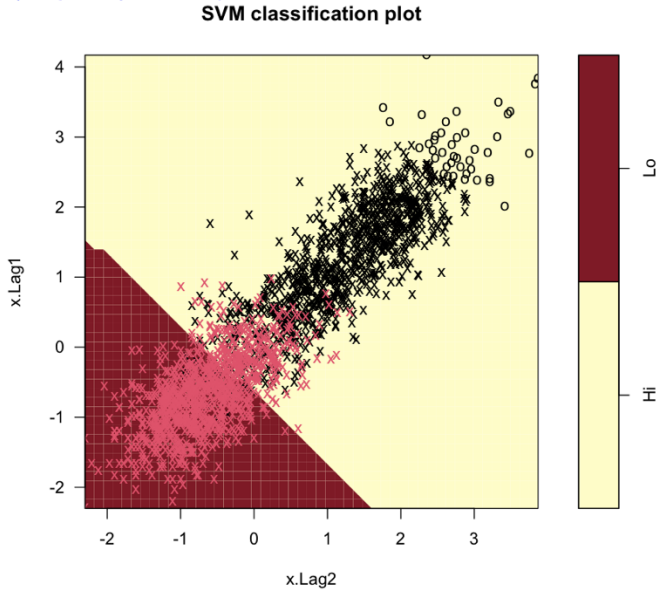


Commands are show on the right, this time a smaller value of *cost* parameter is being used. As I highlighted with a red box, with cost=0.01, there are 618 support vectors obtained. 309 in one class and the other 309 in another class.

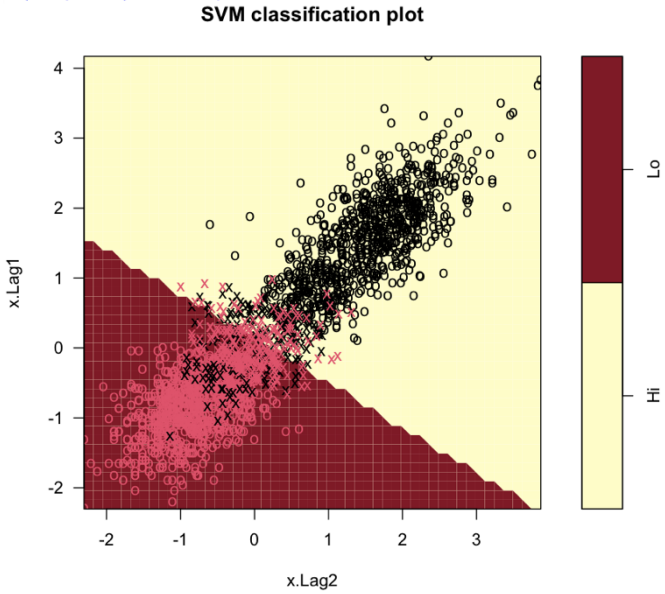
```
> summary(svmfit)

Call:
svm(formula = y ~ ., data = dat.train, kernel = "linear", cost = 0.01,
    scale = FALSE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.01

Number of Support Vectors:  618

 ( 309 309 )


Number of Classes:  2

Levels:
 Hi Lo
```

Now, I will try to use cost = 0.0001, 0.1, 10, 100 for the *svm()* function, and then plot each of them to see what is the different between the cost parameter and the classifier.

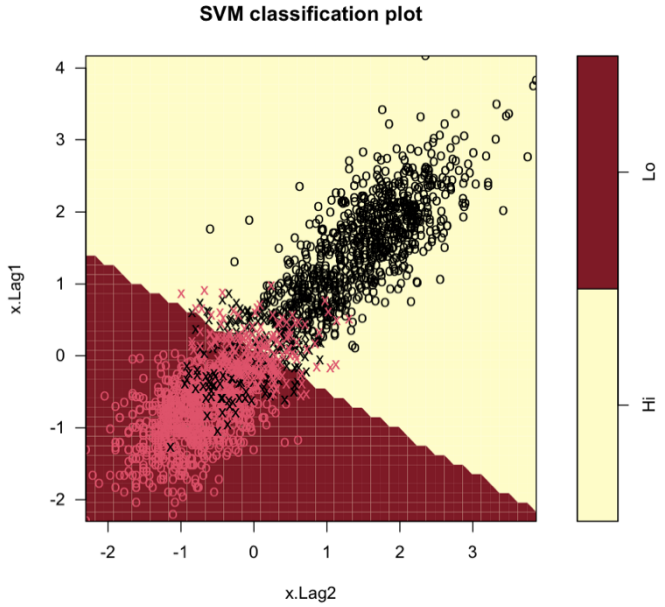
```
> svmfit = svm(y~., data=dat.train, kernel="linear", cost=1e-4, scale=FALSE)
> plot(svmfit, dat.train)
```




```
> svmfit = svm(y~., data=dat.train, kernel="linear", cost=0.1, scale=FALSE)
> plot(svmfit, dat.train)
```




```
> svmfit = svm(y~., data=dat.train, kernel="linear", cost=10, scale=FALSE)
> plot(svmfit, dat.train)
```




```
> svmfit = svm(y~., data=dat.train, kernel="linear", cost=100, scale=FALSE)
> plot(svmfit, dat.train)
```

A2. From the above four plots, we can see that, on the top left, when cost=0.0001, the classifier is the smoothest, it is a line between two classification regions. As the cost parameter increases, from top left to bottom right, the classifier becomes less smooth, the classification is more like wiggy. And the classifiers are rotated anti-clockwise as the cost parameter increases. That is because when cost is a smaller value, the more support vectors are being used. As we can see on the top left, cost=0.0001, more red crosses in the red region compared to other three plots.

So on for the black crosses on the yellow region. More support vectors being used means the margin is wider, so the classifier will be more like a smooth line.

To sum up, by comparing the above examples and information. I know that the *cost* parameter controls how many support vectors will be used. So that with a smaller value of *cost* parameter being used, we will obtain a larger number of support vectors and a smoother classifier, that is because the margin is become wider.

Then in order to find a better value of *cost* to build a predict model, I will use the *tune()* function, which is a built-in function of the *e1071* package. By default, *tune()* function performs ten-fold cross validation on a set of models and collect their performances. I will try to use the same x and y variables, same data frame, same linear kernel as above, but will validate a range of *cost* value includes (0.0001, 0.001, 0.01, 0.1, 1, 10, 100). Commands show on the right:

```
> tune.out=tune(svm, y~., data=dat.train, kernel="linear",
ranges=list(cost=c(1e-4, 1e-3, 0.01, 0.1, 1, 10, 100)))
> summary(tune.out)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
 0.01

- best performance: 0.087

- Detailed performance results:
   cost  error dispersion
1 1e-04 0.4875 0.03343734
2 1e-03 0.0995 0.01499074
3 1e-02 0.0870 0.02030326
4 1e-01 0.0895 0.01817355
5 1e+00 0.0895 0.01723208
6 1e+01 0.0905 0.01786524
7 1e+02 0.0905 0.01786524
```

We can see here, of these 7 cost parameter values, the best one is cost=0.01, with the lowest cross-validation error of 0.0870.

Since we obtained the best cost parameter, cost=0.01, I will access and store this best model obtained as *bestmod*. Shows on the right:

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

Call:
best.tune(method = svm, train.x = y ~ ., data = dat.train, ranges = list(cost =
c(1e-04,
    0.001, 0.01, 0.1, 1, 10, 100)), kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.01

Number of Support Vectors:  666

 ( 333 333 )


Number of Classes:  2

Levels:
 Hi Lo
```

Then I will use this *bestmod,* the best model obtained through cross-validation and *predict()* function to predict y classification from *dat.test* data frame. As shows on the right, I then create a table of predicted classification of test observations and the true classifications. I compute out the correct prediction rate is about 91.06%, 1752 of test observations are correctly classified.

```
> ypred=predict(bestmod, dat.test)
> tab = table(pred=ypred, true=dat.test$y)
> tab
    true
pred  Hi  Lo
  Hi 827  57
  Lo 115 925
> (tab[1,1]+tab[2,2])/sum(tab)
[1] 0.9106029
```

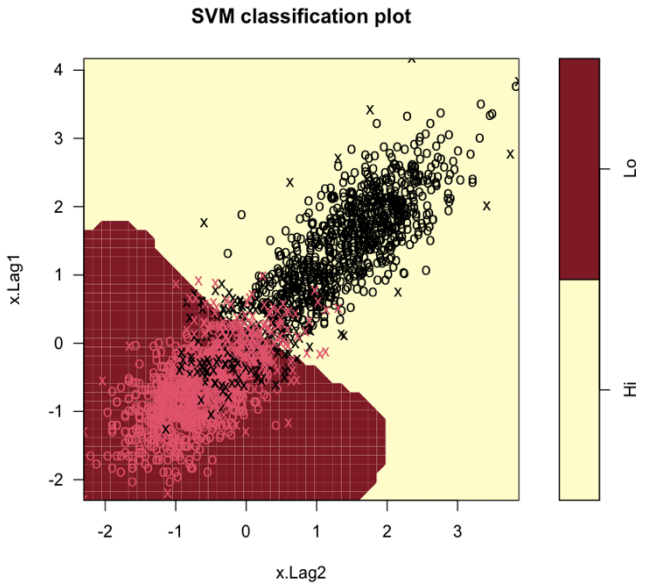
Now I try to use cost=0.0001 to get the correct prediction rate. We can see on the right, with cost=0.0001, the correct prediction rate falls down to 84.98%. In this case, 117 more observations are being misclassified compared to the best model above with cost=0.01.

```
> svmfit = svm(y~., data=dat.train, kernel="linear", cost=1e-4, scale=FALSE)
> ypred=predict(svmfit, dat.test)
> tab = table(pred=ypred, true=dat.test$y)
> tab
    true
pred  Hi  Lo
  Hi 908 255
  Lo  34 727
> (tab[1,1]+tab[2,2])/sum(tab)
[1] 0.8497921
```

Section 3 Apply SVM (radial)

From the above cases, I was assumed the classifier is linear, but what if it is not in linear? So, I will try to apply *radial* to the *kernel* argument. Show on the right:

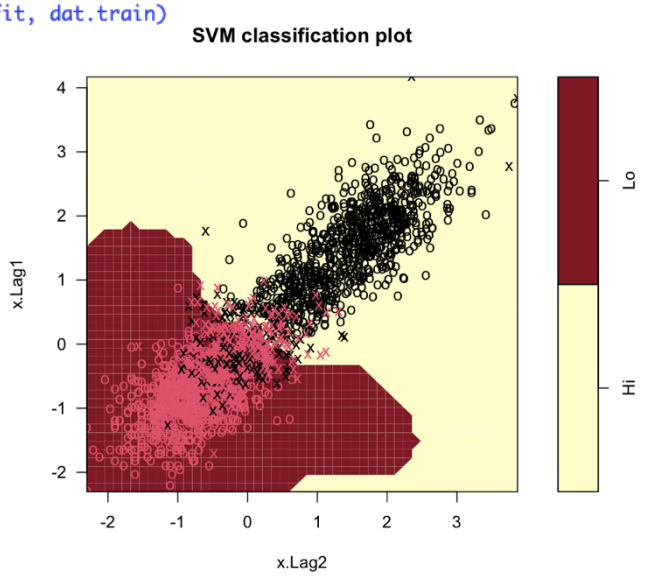
```
> svmfit = svm(y~., data=dat.train, kernel="radial", cost=1, gamma=1)
> plot(svmfit, dat.train)
```



We can see here we have a non-linear boundary. And we still have numbers of errors in this *svmfit* model, which shows on the plot are the black crosses in red region and red crosses in yellow region. We can try to increase the cost parameter to reduce these errors.

Shows on the right here. Yes, as *cost* parameter increases to 100, both black crosses in red region and red crosses in yellow region are reduced which means the errors are reduced, but when we compared these two boundaries of two plots, the one with *cost=100* is a more irregular boundary. It seems that with cost increase, the errors will reduce but at risk of overfitting the data.

```
> svmfit = svm(y~., data=dat.train, kernel="radial", cost=100, gamma=1)
> plot(svmfit, dat.train)
```



In order to find out the best choice of *gamma* and *cost* parameter, I will use *tune()* function to perform cross-validation again. Show on the right:

I set that the *cost* values of (0.0001, 0.001, 0.01, 0.1, 1, 10) and *gamma* values of (0.001, 0.01, 0.1, 1, 2, 10).

There is total 36 pairs of choice here, and the best choice is, as I have highlighted with red boxes, *cost=10* and *gamma=0.001*, it performs the less cross-validation errors of 0.086.

```
> tune.out=tune(svm, y~., data=dat.train, kernel="radial",
ranges=list(cost=c(1e-4, 0.001, 0.01, 0.1, 1, 10), gamma=c(0.001, 0.01, 0.1, 1, 2,
10)))
> summary(tune.out)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
   10 0.001

- best performance: 0.086

- Detailed performance results:
    cost gamma  error dispersion
1  1e-04 1e-03 0.4895 0.03077246
2  1e-03 1e-03 0.4895 0.03077246
3  1e-02 1e-03 0.4895 0.03077246
4  1e-01 1e-03 0.1005 0.02692067
5  1e+00 1e-03 0.0915 0.01313393
6  1e+01 1e-03 0.0860 0.01449138
7  1e-04 1e-02 0.4895 0.03077246
8  1e-03 1e-02 0.4895 0.03077246
9  1e-02 1e-02 0.0990 0.01390444
10 1e-01 1e-02 0.0935 0.01434689
11 1e+00 1e-02 0.0865 0.01375379
12 1e+01 1e-02 0.0895 0.01257201
13 1e-04 1e-01 0.4895 0.03077246
14 1e-03 1e-01 0.4895 0.03077246
15 1e-02 1e-01 0.0925 0.01379412
16 1e-01 1e-01 0.0870 0.01295291
17 1e+00 1e-01 0.0900 0.01452966
18 1e+01 1e-01 0.0885 0.01510151
19 1e-04 1e+00 0.4895 0.03077246
20 1e-03 1e+00 0.4895 0.03077246
```

Since we have found out the best choice of cost and gamma values. I will access and store this best model obtained as *bestmod.radial*, then use *predict()* function on the test data from *dat.test* data frame to get the predict y classification. As shows on the right, I create a table of predicted classification of test observations and the true classifications. I compute out the correct prediction rate is about 91.27%, 1756 of test observations are correctly classified. Compared to the previous best model using *kernel=linear*, *cost=0.01*, which correct prediction rate is about 91.06%, 1752 of test observations are correctly classified.

```
> bestmod.radial=tune.out$best.model
> tab = table(pred=predict(bestmod.radial, dat.test), true=dat.test$y)
> tab
    true
pred  Hi  Lo
  Hi 832  58
  Lo 110 924
> (tab[1,1]+tab[2,2])/sum(tab)
[1] 0.9126819
```

That is, this time we use the *kernel=radial*, we have about 0.21% higher on correct prediction rate, and 4 more observations being correctly classified.

Section 4 SVM Classification Space Plot

A3. Then, I will create a classification space plot with this best model that *kernel=radial, cost=10, gamma=0.001,* to the *dat.test* data frame.

First is to create a data frame with *expand.grid* function on the *x.Lag1* and *x.Lag2* of the *dat.test* data frame. And apply the *predict* function using *bestmod.radial* to get the prediction of *xgrid*. Commands show as above.

```
> xgrid = expand.grid(x.Lag1=dat.test[,1], x.Lag2=dat.test[,2])
> ygrid = predict(bestmod.radial, xgrid)
```

```
> plot(xgrid, col= as.numeric(ygrid)+1, pch=20, cex=0.2, main="SVM Classification
Space Plot")+points(dat.test[,1:2], col=as.numeric(dat.test$y)+1, pch=4)
```

As show on the right, I plot out the SVM Classification Space Plot. We can see here, even though the *kernel* argument is radial in our model, the boundary here is more like a smooth line, that is because the *gamma* parameter is now much smaller, it is 0.001.

As the boundary, the classifier cuts the plot into two color regions, the red region means any point located here are predicted as High risk, the green region means any points located here are predicted as Low risk.

Both red and green crosses are the actual data points in our *dat.test* data frame. We can also see

that there are misclassified observations here, they are the red crosses in green region and the green crosses in red region.

Section 5 SVM ROC curves

In this part, I will try to use *ROCR* package to produce ROC curves for our case here with two model, one in *kernel=linear* another in *kernel=radial*.

After checking, I find out that for the ROC function, the input values, the predicted classification, and actual classification values need to be numeric format. But in our case here, they are in factor format, "Hi" or "Lo". So I use *as.numeric()* function to encode them into numeric format, and create a new data frames call *num.dat.train* shows on the right.

And do the same to *dat.test* and create *num.dat.test*.

```
> class(dat.train[,3])
[1] "factor"
> num.dat.train = dat.train
> num.dat.train[,3] = as.numeric(num.dat.train[,3])
> class(num.dat.train[,3])
[1] "numeric"
> head(num.dat.train)
        x.Lag1      x.Lag2 y
1590  0.1043609  0.04878921 1
462  -0.5978370 -0.65392647 2
1099 -0.3710637 -0.41551544 2
1674 -0.1743522  0.23901690 2
2281  0.4824231 -0.06188072 1
3024  1.6213689  1.44456445 1
> num.dat.test = dat.test
> num.dat.test[,3] = as.numeric(num.dat.test[,3])
```

With the previous cross-validations, we already have best parameters for both linear and radial, which is *cost=0.01* for linear and *cost=10, gamma=0.01* for radial. So I use *svm()* function and apply these parameters to create two models, "*svmfit.l*" for linear, "*svmfit.r*" for radial. Show on the above right.

```
> svmfit.l = svm(y~., data=num.dat.train, kernel="linear", cost=0.01,
decision.values=TRUE)
> svmfit.r = svm(y~., data=num.dat.train, kernel="radial", cost=10, gamma=0.01,
decision.values=TRUE)
```

Load ROCR package and create a function call *rocplot* according to the commands in the ISLR.[3] Show on the right:

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

Then I ues the *svmfit.l* and *predict()* function to get predicted classification for *num.dat.test*, and put actual classification values into *rocplot()* function to create a ROC curve. But I got an error here.

```
> rocplot(predict(svmfit.l, num.dat.test[,1:2], decision.values=TRUE),
num.dat.test[,3], main="Training Data")
Error: Format of predictions is invalid. It couldn't be coerced to a list.
```

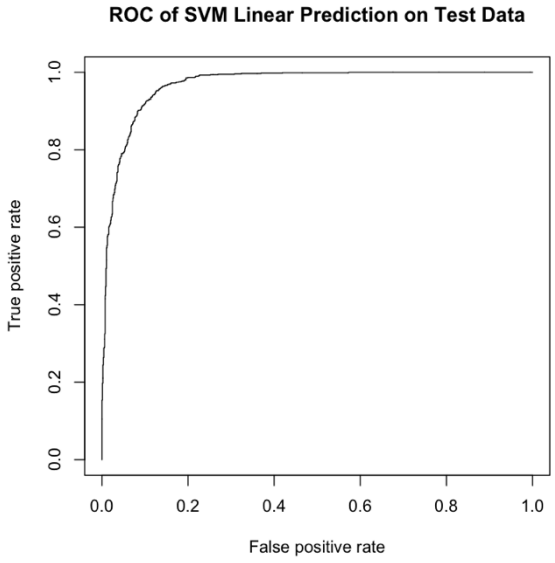
[3] Source: *An Introduction to Statistical Learning*,
by James, Witten, Hastie, and Tibshirani; 4th printing, Springer Press, page 365.

After checking, I add *as.numeric* into the rocplot() function to encode the inputs to numeric format as I highlighted with red box shows on the right.

```
> fix(rocplot)
> rocplot
function (pred, truth, ...) {
    predob = prediction (as.numeric(pred), as.numeric(truth))
    perf = performance(predob, "tpr", "fpr")
    plot(perf, ...)
}
```

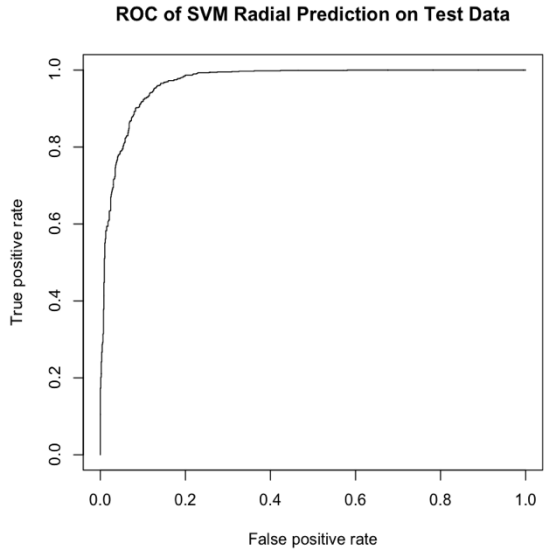
Run it again, as shows on the right. Finally, I got the ROC curve for the predicted classification on *num.dat.test* with SVM linear model.

```
> rocplot(predict(svmfit.l, num.dat.test[,1:2], decision.values=TRUE),
num.dat.test[,3], main="ROC of SVM Linear Prediction on Test Data")
```



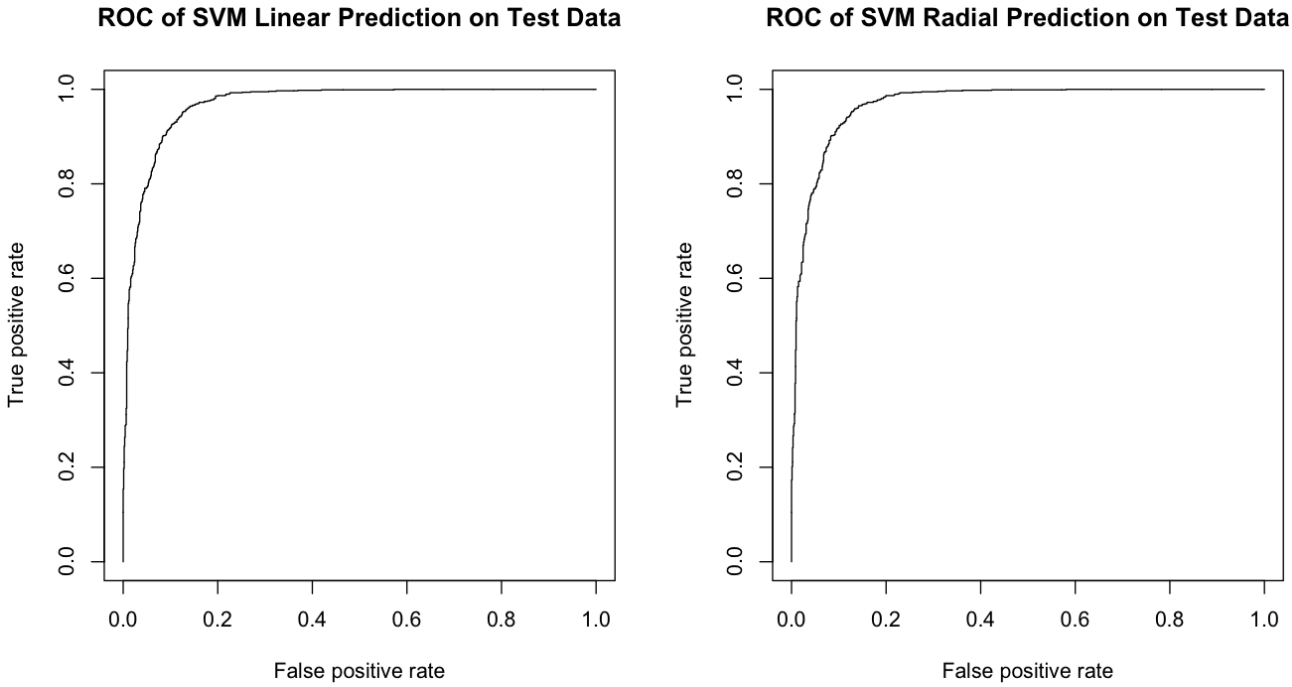
And I do the same with SVM radial model.

```
> rocplot(predict(svmfit.r, num.dat.test[,1:2], decision.values=TRUE),
num.dat.test[,3], main="ROC of SVM Radial Prediction on Test Data")
```



For a better comparation, I put these two plots together. Show as below:

```
> par(mfrow=c(1,2))
> rocplot(predict(svmfit.l, num.dat.test[,1:2], decision.values=TRUE),
num.dat.test[,3], main="ROC of SVM Linear Prediction on Test Data")
> rocplot(predict(svmfit.r, num.dat.test[,1:2], decision.values=TRUE),
num.dat.test[,3], main="ROC of SVM Radial Prediction on Test Data")
```



These two curves are mostly the same, that is because as we have known previously, the radial model only have about 0.21% higher on correct prediction rate than linear model. Which is only 4 more observations being correctly classified among total 3924 observations.

Part B Cardiac Dataset Practice

In this part, I will do a comparative study of the performance of knn, naive Bayes, logistic regression (with bestglm) and SVM on the given Cardiac Data.

Section 1 Data Preparation

I download the given "CIS-STA 3920 LN8.B Cardiac.csv" from BlackBoard and load it into R, name as "*cardiac*". Then I choose the 1st to the 10th columns as the x variables, name as *card.x*, shows on the right:

```
> cardiac = read.csv("CIS-STA 3920 LN8.B Cardiac.csv")
> card.x = cardiac[,1:10]
```

On the right, I apply a logistic check to the 13th column, *"age"* which is patient's age, to see if each value from this column is bigger than the median of all age values. If yes, then it will be classified as 1, if not, classified as 0. I name this vector as *card.y*.

```
> card.y = cardiac$age > median(cardiac$age)
> head(card.y)
[1]  TRUE  TRUE  TRUE FALSE FALSE  TRUE
> card.y = as.numeric(card.y)
> head(card.y)
[1] 1 1 1 0 0 1
```

Then, I combine the *card.x* and *card.y* into a new data frame, name "*card*", and name the *card.y* column as *y*. There 558 rows and 11 columns in this *card* data frame. Shows on the right:

```
> card = cbind(card.x, y=card.y)
> head(card)
       bhr    basebp    basedp      pkhr       sbp        dp     dose     maxhr
1 0.5476190 0.8728814 0.4249327 0.7215190 0.3197026 0.2450142 1.333333 0.7042254
2 0.3690476 1.1779661 0.3864574 0.7594937 0.5873606 0.4738342 1.333333 0.8450704
3 0.3690476 1.1779661 0.3864574 0.7594937 0.5836431 0.4708352 1.333333 0.8450704
4 0.5535714 1.0000000 0.4921076 0.7468354 0.3903346 0.3096416 1.000000 0.8309859
5 0.5297619 0.8728814 0.4110762 0.8164557 0.6431227 0.5577298 1.333333 0.9084507
6 0.3452381 0.8474576 0.2600897 0.7784810 0.5204461 0.4303494 1.333333 0.8661972
  X.mphr.b.       mbp y
1 0.7789474 0.5377778 1
2 0.8631579 0.7022222 1
3 0.8631579 0.6977778 1
4 0.7578947 0.4666667 0
5 0.7263158 0.7822222 0
6 0.8736842 0.6222222 1
> dim(card)
[1] 558  11
```

Section 2 Apply Logistic Regression (with bestglm)

With above data prepared, I will run the bestglm() function for logistic regression on *card* data frame. First, I will try the cross-validation (CV) method. Shows on the right:

```
> library(bestglm)
Loading required package: leaps
> CV.out = bestglm(card, IC="CV", family=binomial)
Morgan-Tatar search since family is non-gaussian.
There were 50 or more warnings (use warnings() to see the first 50)
> CV.out
CVd(d = 454, REP = 1000)
BICq equivalent for q in (0, 0.690144985877429)
Best Model:
              Estimate Std. Error   z value     Pr(>|z|)
(Intercept)   -2.623215   2.057765 -1.274788 2.023842e-01
maxhr       -651.069729 139.610126 -4.663485 3.108989e-06
X.mphr.b.    658.969556 141.333373  4.662519 3.123621e-06
```

Then try the BICq method, shows on the right. We got the same model from these two summary informations.

```
> bicq.out = bestglm(card, IC="BICq", family=binomial)
Morgan-Tatar search since family is non-gaussian.
There were 50 or more warnings (use warnings() to see the first 50)
> bicq.out
BICq(q = 0.25)
BICq equivalent for q in (0, 0.690144985877429)
Best Model:
              Estimate Std. Error   z value     Pr(>|z|)
(Intercept)   -2.623215   2.057765 -1.274788 2.023842e-01
maxhr       -651.069729 139.610126 -4.663485 3.108989e-06
X.mphr.b.    658.969556 141.333373  4.662519 3.123621e-06
```

From the above inforamtiom, on the following pages, I will choose the *maxhr* and *X.mphr.b.* as two x-variables to predict *y*. Check back to the explaination of Cardiac Dataset, I know that *maxhr* means patient's maximum heart rate, *X.mphr.b.* menas % of maximun predicted heart rate achieved by that patient. And the *y* variable we want to predict is to see weather this patient's age is above the median of all patient's ages. If above, *y=1*; if equal or below, *y=0*.

For a better prediction and modeling, I will apply natural log on the *maxhr* and *X.mphr.b.* variables using *log()* function, then combine with y, as a new data frame call *lncard*. Show on the right:

```
> lncard = log(card[,8:9])
> lncard = cbind(lncard, y=card$y)
> head(lncard)
       maxhr  X.mphr.b. y
1 -0.35065687 -0.2498118 1
2 -0.16833531 -0.1471576 1
3 -0.16833531 -0.1471576 1
4 -0.18514243 -0.2772108 0
5 -0.09601465 -0.3197704 0
6 -0.14364270 -0.1350363 1
```

Using the *glm()* function on *lncard* to create a model call "*glm.fit*". Show on the right:

```
> glm.fit=glm(y~., data=lncard, family=binomial)
Warning message:
glm.fit: fitted probabilities numerically 0 or 1 occurred
> glm.fit

Call:  glm(formula = y ~ ., family = binomial, data = lncard)

Coefficients:
(Intercept)        maxhr    X.mphr.b.
       3.69      -467.38       469.32

Degrees of Freedom: 557 Total (i.e. Null);  555 Residual
Null Deviance:      770.7
Residual Deviance: 51.93   AIC: 57.93
```

On the right, then use the *predict()* function to get the estimated probabilities.

```
> glm.prob = predict(glm.fit, type="response")
> head(glm.prob)
           1            2            3            4            5            6
1.000000e+00 9.999983e-01 9.999983e-01 2.220446e-16 2.220446e-16 9.994184e-01
```

Then I need to covert these estimated probabilities into fitted logistic values, either 1 or 0. So I round these estimated probabilities up to 1 or down to 0 which is actual using for *y* in the *card* data frame.

```
> glm.fore = round(glm.prob)
> head(glm.fore)
1 2 3 4 5 6
1 1 1 0 0 1
```

Next, I create a table call *glm.tab* to put the forecasted values and actual values together and computed out the correct forecast rate. Show on the right:

We can see in our case here, the *glm.fit* model has a correct forecast rate of 97.49%, which means based on *maxhr* and *X.mphr.b* information, we have 97.49% correct rate to predict weather that patient's age is above the median of all patient's ages.

```
> glm.tab = table(glm.fore, lncard$y)
> glm.tab

glm.fore   0   1
       0 293   8
       1   6 251
> (glm.tab[1,1]+glm.tab[2,2])/sum(glm.tab)
[1] 0.9749104
```

Then, I will try to create a classification space plot for that glm fitted model. First is to create a data frame with *expand.grid* function on the

```
> xgrid = expand.grid(maxhr=lncard$maxhr, X.mphr.b.=lncard$X.mphr.b.)
> ygrid = predict(glm.fit, xgrid, type="response")
```

maxhr and *X.mphr.b.* of the *lncard* data frame. And apply the *predict* function using *glm.fit* to get the probablities of *xgrid*, name as *ygrid*. Commands show as above.

```
> plot(xgrid, col=round(ygrid)+2, pch=20, cex=0.5, main="glm Classification Space
Plot")+points(lncard[,1:2], col=lncard$y+2, pch=16)
```

Show on the right, I plot out the glm Classification Space Plot. We can see there are two regions in the plot, the red region means any point located here are predicted as 0, which mean that patient's age predicted equal or below the median of all patient ages. The green region means any points located here are predicted as 1, which mean that patient's age predicted above the median of all patient ages.

And I also put the actual data into the plot, we can see the actual data are mostly correct classified, not much of them are located in wrong color region. That means our glm fitted model have a high correct predict rate.

Section 3 Apply Knn

I will apply knn method using the the same *maxhr* and *X.mphr.b.* as two x-variables to predict *y*, using the *kcvSearch()* function to find out the best *k* value first.

First, I decided the train-test split to be 400-158. That is, 400 shuffled rows for training, and the rest 158 rows for testing. The 400 shuffled after-natural-log *maxhr* and *X.mphr.b.* are as *train.x* variable pairs, their correspond classification y as *trian.y*. So on for the rest 158 pairs as *test.x,* and *test.y*.

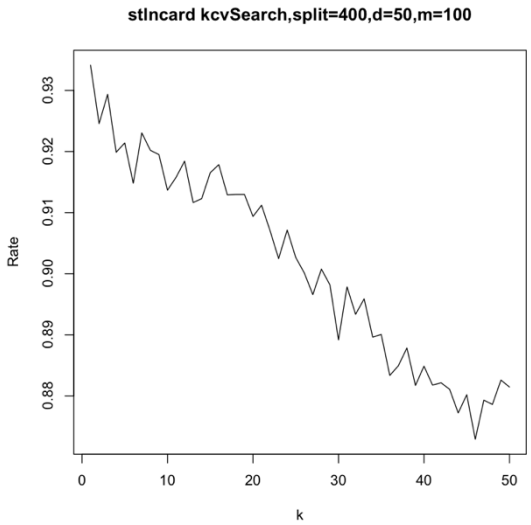
Then, for a better performance of knn, I need to standardize the *maxhr* and *X.mphr* two variables using *apply()* function, and create a new data frame call *"stlncard"*, shows on the right:

```
> stlncard = lncard
> stlncard[,1:2] = apply(stlncard[,1:2],2,scale)
> head(stlncard)
       maxhr  X.mphr.b. y
1 -0.82509095 -0.2022639 1
2  0.11981281  0.3096142 1
3  0.11981281  0.3096142 1
4  0.03270786 -0.3388871 0
5  0.49462348 -0.5511078 0
6  0.24778529  0.3700566 1
```

Then load the class package, run the *kcvSearch()* function, using *maxhr* and *X.mphr.b.* of *stlncard* data frame as x, and *y* of *stlncard* data frame as y. Split= 400, d=50, m=100. Plot it out. Show on the right:

```
> library(class)
> cardKcvSearch = kcvSearch(stlncard[,1:2], as.factor(stlncard$y), 400, 50, 100)
> plot(cardKcvSearch, type="l", xlab="k", ylab="Rate", main="stlncard
kcvSearch,split=400,d=50,m=100")
```

We can see from the plot on right, when using knn method with *maxhr* and *X.mphr.b.* as x-variables to predict *y*, when k=1, we have the highest correct forecast rate about 93.4%.



Then I will create a knn classification space plot using *NewProbeKnnBig()*[4] function which from Lecture Note 5 created by Prof. Tatum with k=1, train-test split as 400-158. I use *sample()* to shuffle and the 400-158 train-test split. The 400 shuffled *maxhr* and *X.mphr.* as *train.x* variable pairs, their correspond classification y as *trian.y*. So on for the rest 158 pairs as *test.x,* and *test.y*. Show on the right:

```
> train = sample(558,400)
> stlncard.train.x = stlncard[train, 1:2]
> stlncard.train.y = stlncard[train, 3]
> stlncard.test.x = stlncard[-train, 1:2]
> stlncard.test.y = stlncard[-train, 3]
```

[4] Source: CIS-STA 3920 LN5.A Classification Space for KNN 7-20-21 by Prof. L. Tatum, page 14.

Then, like previous, use the *expand.grid* on *stlncard.test.x* to create *xgrid*. Make sure the two variables' names keep the same as them in the *stlncard.train.x*.

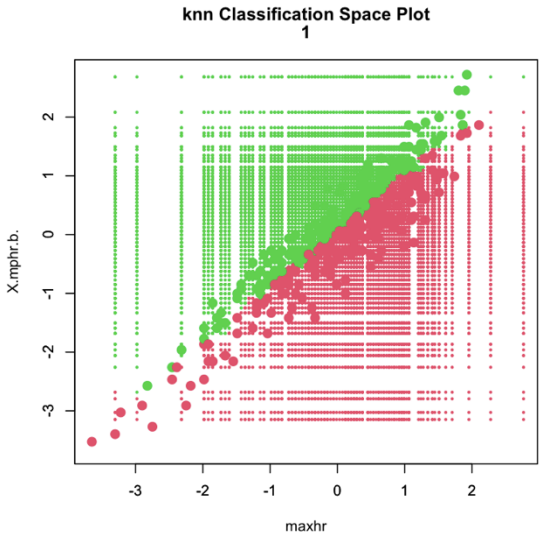
```
> xgrid = expand.grid(maxhr=stlncard.test.x$maxhr,
X.mphr.b.=stlncard.test.x$X.mphr.b.)
```

And I edit some codes for the plot appearance in the *NewProbeKnnBig()* function, shows on the right:

```
> NewProbeKnnBig
function (TrialX, ProbeX, TrialY, k)
{
library(class)

ProbeYhat = knn(TrialX,ProbeX,TrialY,k)

MinX = min(TrialX)
MaxX = max(TrialX)

ProbeColor = c(cbind(ProbeYhat)+1)

plot(ProbeX[,1],ProbeX[,2],
main=c("knn Classification Space Plot", k),
xlab="maxhr",ylab="X.mphr.b.",
cex=0.5, pch=20,
col=ProbeColor,
xlim=c(MinX,MaxX),
ylim=c(MinX,MaxX))

par(new=TRUE)

plot(TrialX[,1],TrialX[,2],
ann=FALSE,
pch=20,
col=c(cbind(TrialY)+2),
xlim=c(MinX,MaxX),
ylim=c(MinX,MaxX),
cex=2)

return()
}
```

Put defined input into *NewProbeKnnBig()* function with k=1. Got the knn Classification Space Plot, show on the right:

```
> NewProbeKnnBig(stlncard.train.x, xgrid, stlncard.train.y, k=1)
```



We can the knn Classification Space Plot is similar to the glm Classification Space Plot we got previously. The actual *stlncard.test.y* data are mostly correct classified, not much of them are located in wrong color region. That means our knn when k =1 model has a high correct predict rate.

Section 4 Apply Naive Bayes

In this section, I will apply Naive Bayes method using the same after-natural-log *maxhr* and *X.mphr.b.* as two x-variables to predict *y* in the *lncard* data frame.

First, using the same vector *train* to get the same 400-158 train-test split. The 400 shuffled *maxhr* and *X.mphr*. form *lncard* data frame, are as *lncard.train.x* variable pairs, their correspond classification y as *lncard.train.y*. So on for the rest 158 pairs as *lncard.test.x* and *lncard.test.y*. Show on right:

```
> lncard.train.x = lncard[train, 1:2]
> lncard.train.y = lncard[train, 3]
> lncard.test.x = lncard[-train, 1:2]
> lncard.test.y = lncard[-train, 3]
```

Load the required *caret, klaR* and *e1071* packages.

```
> library(caret)
Loading required package: lattice
Loading required package: ggplot2
Warning message:
In (function (kind = NULL, normal.kind = NULL, sample.kind = NULL)  :
  non-uniform 'Rounding' sampler used
> library(klaR)
Loading required package: MASS
> library(e1071)
```

As on the right, I put the *lncard.train.x* and *lncard.train.y* into the *train()* function, using the Naive Bayes program refer as "nb". And the cross-validation program refers as "cv" here, giving a parameter value of 20, means break the input data, the 400 rows, into 20 sections, so 20 rows are assigned to each section. We will get the fitted NB model name as *"nb.fit"*. Then I use this fitted model with *lncard.test.x* as input to get the ŷ classification, and these ŷ classification together with the actual values, *lncard.test.y,* into a new table call *"nb.tab"*. Then computed out the correct forecast rate.

```
> nb.fit = train(lncard.train.x, as.factor(lncard.train.y), 'nb',
trControl=trainControl(method='cv', number=20))
Warning message:
In (function (kind = NULL, normal.kind = NULL, sample.kind = NULL)  :
  non-uniform 'Rounding' sampler used
> nb.tab = table(predict(nb.fit$finalModel, lncard.test.x)$class, lncard.test.y)
> nb.tab
   lncard.test.y
     0  1
  0 49 36
  1 31 42
> (nb.tab[1,1]+nb.tab[2,2])/sum(nb.tab)
[1] 0.5759494
```

We can see in our case here, the *nb.fit* model has a correct forecast rate of about 57.59%.

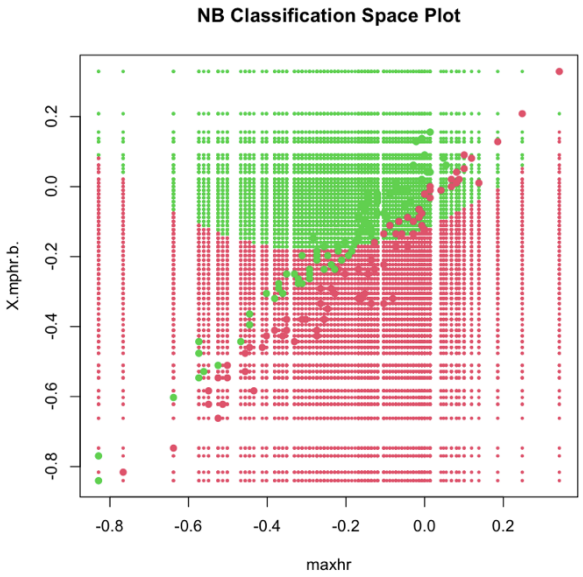
Then I will also create a NB classification space plot. First, like previous, use the *expand.grid* on *lncard.test.x* to create *xgrid*. Make sure the two variables' names keep the same as them in the *lncard.train.x*. And apply the *predict* function using *nb.fit* to get the prediction of *xgrid*, name as *ygrid*. Show as above.

```
> xgrid = expand.grid(maxhr=lncard.test.x$maxhr,
X.mphr.b.=lncard.test.x$X.mphr.b.)
> ygrid = predict(nb.fit$finalModel, xgrid)$class
```

Show on the right, I plot out the NB Classification Space Plot. We can see there are two regions in the plot, the red region means any point located here are predicted as 0, which mean that patient's age predicted equal or below the median of all patient ages. The green region means any points located here are predicted as 1, which mean that patient's age predicted above the median of all patient ages.

And I also put the actual data into the plot, we can see a lot of actual data are being misclassified, not much of them are located in their match color region. That means our NB fitted model does not perform well in our case here.

```
> plot(xgrid, col=as.numeric(ygrid)+1, pch=20, cex=0.5, main="NB Classification
Space Plot")+points(lncard.test.x, col=lncard.test.y+2, pch=16)
```



Section 5 Apply SVM

In this section, I will apply SVM method using the after-natural-log *maxhr* and *X.mphr.b.* as two x-variables to predict *y* in the *lncard* data frame.

First, using the same 400-158 train-test split data frames that we used previously, to combine them into train and test data frame, and encode the *y* column for each one as factor. We got *lncard.train* and *lncard.test*. Show on the right:

```
> lncard.train = cbind(lncard.train.x, y=as.factor(lncard.train.y))
> lncard.test = cbind(lncard.test.x, y=as.factor(lncard.test.y))
```

```
> head(lncard.train)
          maxhr  X.mphr.b. y
333 -0.07302513 -0.1112256 0
307 -0.21962861 -0.2363888 0
558  0.09402895  0.0000000 0
291 -0.08062973 -0.2231436 0
147 -0.47849024 -0.4934339 0
373 -0.38111608 -0.3642221 1
> head(lncard.test)
          maxhr   X.mphr.b. y
1  -0.35065687 -0.24981180 1
3  -0.16833531 -0.14715764 1
8   0.01398624  0.15565331 1
11  0.18583650  0.12825434 0
16 -0.04317217 -0.08796877 0
22 -0.15180601 -0.06524052 1
```

Then, load the *e1071* package. Use *tune()* function to performs ten-fold cross validation on a set of models and collect their performances. I use *card.train* as the input data frame, *kernel=linear* to validate a range of *cost* value includes (0.0001, 0.001, 0.01, 0.1, 1, 10). Commands show on the right:

We can see here, as I have highlighted with a red box, the cost=10 is the best choice for SVM linear model.

```
> library(e1071)
> tune.out =tune(svm, y~., data=lncard.train, kernel="linear",
ranges=list(cost=c(1e-4,1e-3, 0.01, 0.1, 1, 10)))
> tune.out

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
cost
  10

- best performance: 0.03
```

Then I access and store this fitted model obtained as *svm.fit.line*, then use *predict()* function on the test data from *lncard.test* data frame to get the predict *y* classification. As shows on the right, I create a table of predicted classification of test observations and the true classifications.

```
> svm.fit.line=tune.out$best.model
> svm.tab.line = table(predict(svm.fit.line, lncard.test), lncard.test$y)
> svm.tab.line

    0  1
  0 80  3
  1  0 75
> (svm.tab.line[1,1]+svm.tab.line[2,2])/sum(svm.tab.line)
[1] 0.9810127
```

We can see here *kernel=linear* the correct prediction rate is about 98.1%.

Next, I use *tune()* function again to performs the ten-fold cross validation on a set of models and collect their performances. The same *lncard.train* as the input data frame, but *kernel=radial*, to validate a range of *cost* value includes (0.0001, 0.001, 0.01, 0.1, 1, 10) and *gamma* value includes (0.001, 0.01, 0.1, 1, 2, 10). Commands show on the above right. We can see as I have highlighted with a red box, the cost=10 and gamma=1 is the best choice pair for SVM radial model.

```
> tune.out =tune(svm, y~., data=lncard.train, kernel="radial",
ranges=list(cost=c(1e-4,1e-3, 0.01, 0.1, 1, 10), gamma=c(1e-3, 0.01, 0.1, 1, 2,
10)))
> tune.out

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
   10     1

- best performance: 0.045
```

Then I access and store this fitted model obtained as *svm.fit.radi*, then use *predict()* function on the test data from *lncard.test* data frame to get the predict *y* classification. As shows on the right, I create a table of predicted classification of test observations and the true classifications.

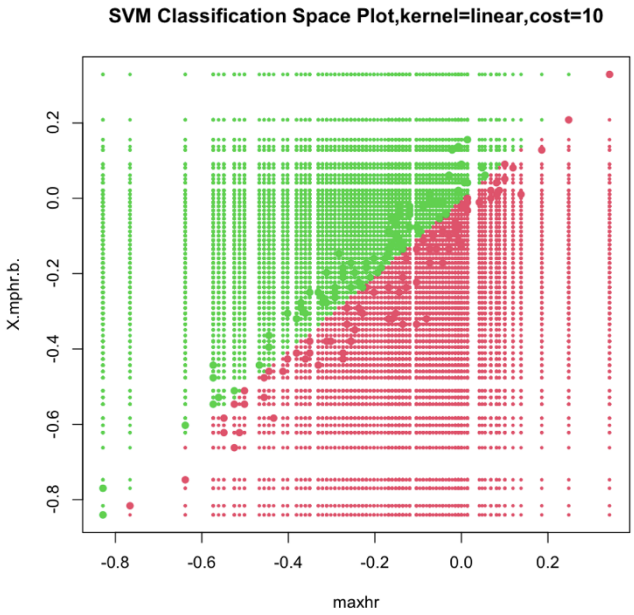
```
> svm.fit.radi=tune.out$best.model
> svm.tab.radi = table(predict(svm.fit.radi, lncard.test), lncard.test$y)
> svm.tab.radi

    0  1
  0 79  3
  1  1 75
> (svm.tab.radi[1,1]+svm.tab.radi[2,2])/sum(svm.tab.radi)
[1] 0.9746835
```

We can see here with *kernel=radial* the correct prediction rate is about 97.47%.

As the above information, we know that using SVM method, *kernel=linear,cost=10* has a higher correct prediction rate is of 98.10%. I will also create a SVM classification space plot using this fitted model.

First, like previous, to create a data frame with *expand.grid()* function on *maxhr* and *X.mphr.b.* two variables as *xgrid*. And apply the *predict* function using *svm.fit.line* to get the prediction of *xgrid*, as ygrid. Commands show on the right:

```
> xgrid = expand.grid(maxhr=lncard.test$maxhr, X.mphr.b.=lncard.test$X.mphr.b.)
> ygrid = predict(svm.fit.line, xgrid)

> plot(xgrid, col=as.numeric(ygrid)+1, pch=20, cex=0.5, main="SVM Classification
Space Plot,kernel=linear,cost=10")+points(lncard.test,
col=as.numeric(lncard.test$y)+1, pch=16)
```



Show on the right, I plot out the SVM Classification Space Plot. We can see there are two regions in the plot, the red region means any point located here are predicted as 0, which mean that patient's age predicted equal or below the median of all patient ages. The green region means any points located here are predicted as 1, which mean that patient's age predicted above the median of all patient ages.

And I also put the actual data into the plot, we can see the actual data are mostly being correct classified, not much of them are located in wrong color region. That means our SVM fitted model have a high correct predict rate.

Section 6 Sum up

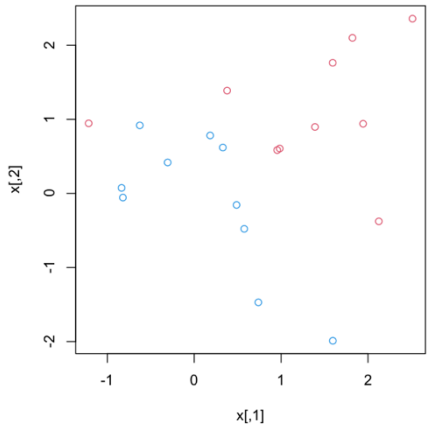
Form the above information, after studying with glm, knn, Naive Bayes and SVM four methods on our chosen Cardiac Data, the SVM (kernel=linear, cost=10) has the best performance as it has a high correct predict rate of 98.1%. The second is glm with correct predict rate of 97.49%, knn has a correct predict rate of 93.4%. While the Naive Bayes has the worst performance among these four methods with a correct prediction rate of only 57.59%.

In our case here, the method which has a high correct prediction rate means based on the *maxhr* and *X.mphr.b* information, we have high correct rate to predict weather that patient's age is above the median of all patient's ages or not.

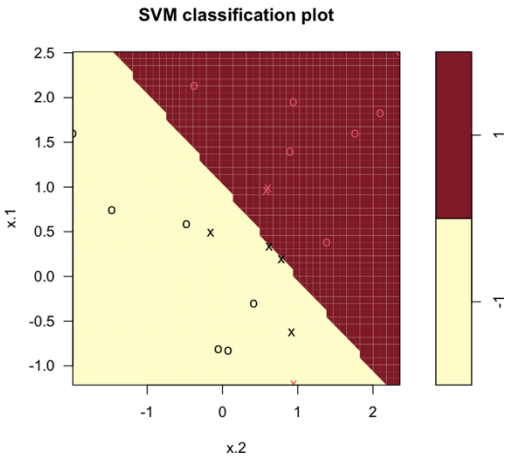
Appendix: Lab Example of SVM

9.6.1 Support Vector Classifier

```
> set.seed(1)
> x=matrix(rnorm(20*2), ncol=2)
> y=c(rep(-1,10), rep(1,10))
> x[y==1,]=x[y==1,]+1
> plot(x,col=(3-y))
```



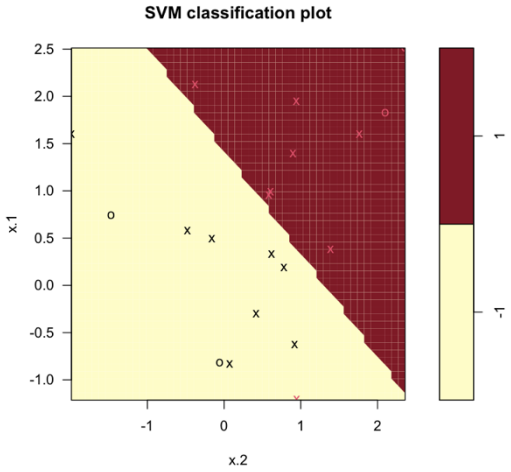

```
> dat=data.frame(x=x, y=as.factor(y))
> library(e1071)
> svmfit=svm(y~., data=dat, kernel = "linear",
cost=10, scale=FALSE)
> plot(svmfit, dat)
```




```
> svmfit$index
[1]  1  2  5  7 14 16 17
> summary(svmfit)

Call:
svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10, scale = FALSE)


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

Number of Support Vectors:  7

 ( 4 3 )

Number of Classes:  2

Levels:
```

-1 1

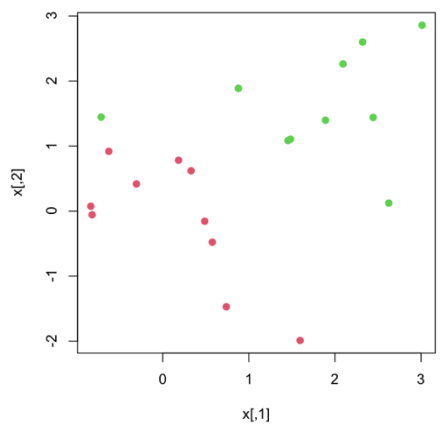
```
> svmfit=svm(y~., data=dat,
kernel="linear", cost=0.1, scale=FALSE)
> plot(svmfit, dat)
```




```
> svmfit$index
 [1]  1  2  3  4  5  7  9 10 12 13 14 15 16 17 18 20

> set.seed(1)
> tune.out=tune(svm,y~.,data=dat, kernel="linear", range=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
  0.1

- best performance: 0.05

- Detailed performance results:
    cost error dispersion
1   0.001  0.55  0.4377975
2   0.010  0.55  0.4377975
3   0.100  0.05  0.1581139
4   1.500  0.15  0.2415229
5  10.000  0.15  0.2415229
6 100.000  0.15  0.2415229

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

Call:
best.tune(method = svm, train.x = y ~ ., data = dat, ranges = list(cost = c(0.001,
```

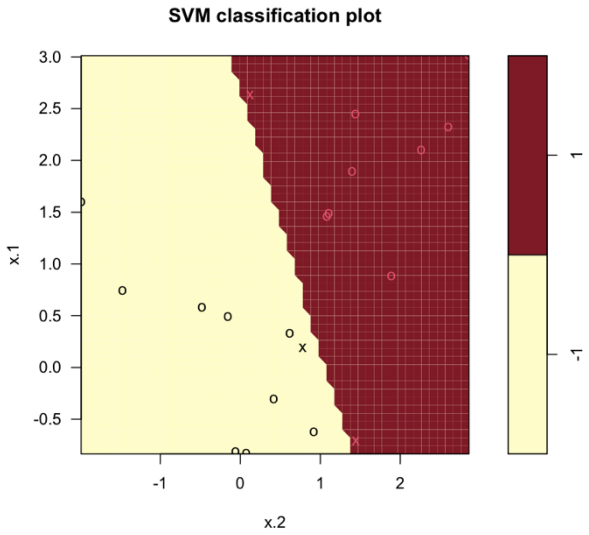
```
    0.01, 0.1, 1.5, 10, 100)), kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.1

Number of Support Vectors:  16

 ( 8 8 )


Number of Classes:  2

Levels:
 -1 1

> xtest=matrix(rnorm(20*2), ncol=2)
> ytest=sample(c(-1,1), 20, rep=TRUE)
> xtest[ytest==1,]=xtest[ytest==1,] + 1
> testdat=data.frame(x=xtest, y=as.factor(ytest))
> ypred=predict(bestmod,testdat)
> table(predict=ypred, truth=testdat$y)
       truth
predict -1 1
     -1  9 1
     1   2 8

> svmfit=svm(y~., data=dat, kernel="linear", cost=0.01, scale=FALSE)
> ypred=predict(svmfit,testdat)
> table(predict=ypred, truth=testdat$y)
       truth
predict -1  1
     -1 11  6
     1   0  3
```

```
> x[y==1,]=x[y==1] + 0.5
> plot(x, col=(y+5)/2, pch=19)
```




```
> dat= data.frame(x=x,y=as.factor(y))
> svmfit=svm(y~., data=dat, kernel="linear", cost=1e5)
> summary(svmfit)

Call:
svm(formula = y ~ ., data = dat, kernel = "linear", cost = 1e+05)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1e+05

Number of Support Vectors:  3

 ( 1 2 )


Number of Classes:  2

Levels:
 -1 1
```

> plot(svmfit, dat)



> svmfit=svm(y~., data=dat, kernel="linear", cost=1)
> summary(svmfit)

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

Parameters:
SVM-Type: C-classification
SVM-Kernel: linear
cost: 1

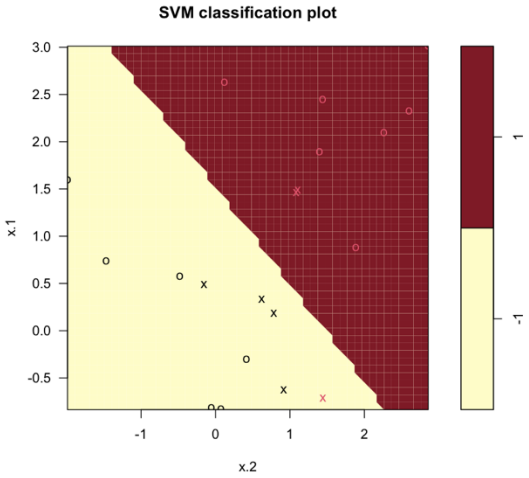
Number of Support Vectors: 7

(4 3)

Number of Classes: 2

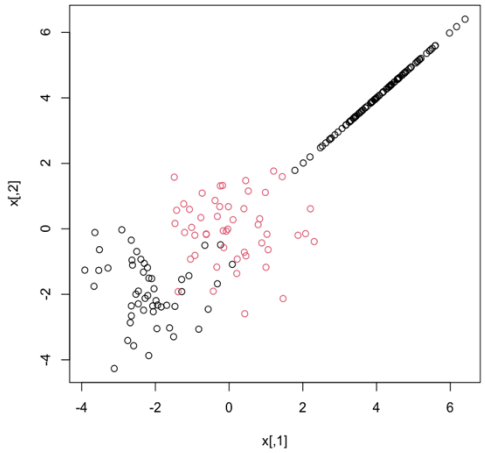
Levels:
-1 1

> plot(svmfit, dat)

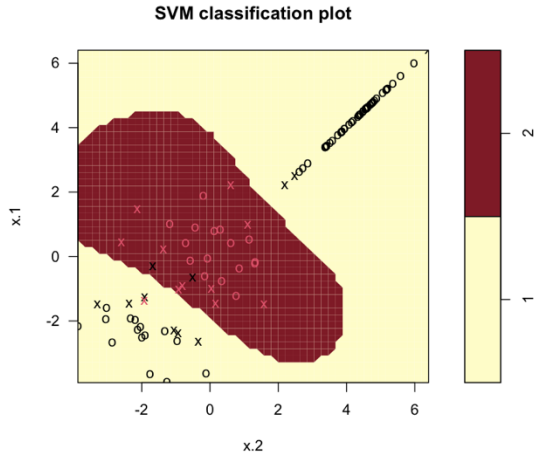


9.6.2 Support Vector Machine

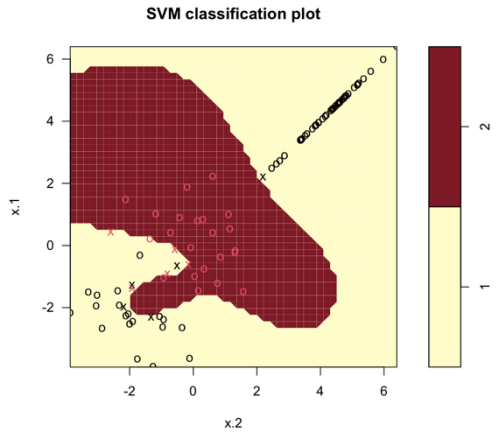
```
> set.seed(1)
> x=matrix(rnorm(200*2), ncol=2)
> x[1:100,]=x[1:100,] +2
> x[101:150,]=x[101:150,] -2
> y=c(rep(1,150),rep(2,50))
> dat= data.frame(x=x,y=as.factor(y))
> plot(x, col=y)
```




```
> train=sample(200,100)
> svmfit=svm(y~., data=dat[train,],
kernel="radial", gamma=1, cost=1)
> plot(svmfit, dat[train,])
```



```
> summary(svmfit)

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


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1

Number of Support Vectors:  22

 ( 11 11 )


Number of Classes:  2

Levels:
 1 2

> svmfit=svm(y~., data=dat[train,], kernel="radial", gamma=1, cost=1e5)
> plot(svmfit, dat[train,])
```



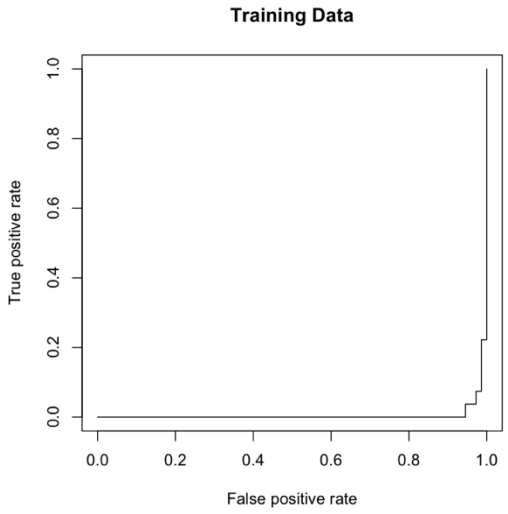

```
> set.seed(1)
> tune.out=tune(svm, y~., data=dat[train,], kernel="radial",
ranges=list(cost=c(0.1,1,10,100,1000), gamma=c(0.5,1,2,3,4)))
> summary(tune.out)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
```

```
   1   0.5

- best performance: 0.03

- Detailed performance results:
   cost gamma error dispersion
1  1e-01   0.5  0.20 0.19436506
2  1e+00   0.5  0.03 0.04830459
3  1e+01   0.5  0.03 0.04830459
4  1e+02   0.5  0.03 0.04830459
5  1e+03   0.5  0.05 0.07071068
6  1e-01   1.0  0.07 0.06749486
7  1e+00   1.0  0.03 0.04830459
8  1e+01   1.0  0.03 0.04830459
9  1e+02   1.0  0.04 0.05163978
10 1e+03   1.0  0.06 0.06992059
11 1e-01   2.0  0.09 0.05676462
12 1e+00   2.0  0.03 0.04830459
13 1e+01   2.0  0.03 0.04830459
14 1e+02   2.0  0.05 0.07071068
15 1e+03   2.0  0.07 0.04830459
16 1e-01   3.0  0.12 0.07888106
17 1e+00   3.0  0.03 0.04830459
18 1e+01   3.0  0.03 0.04830459
19 1e+02   3.0  0.05 0.07071068
20 1e+03   3.0  0.07 0.06749486
21 1e-01   4.0  0.16 0.11737878
22 1e+00   4.0  0.03 0.04830459
23 1e+01   4.0  0.04 0.05163978
24 1e+02   4.0  0.06 0.05163978
25 1e+03   4.0  0.09 0.07378648

> table(true=dat[-train,"y"], pred=predict(tune.out$best.model,newx=dat[-train,]))
   pred
true  1  2
   1 55 22
   2 17  6
```

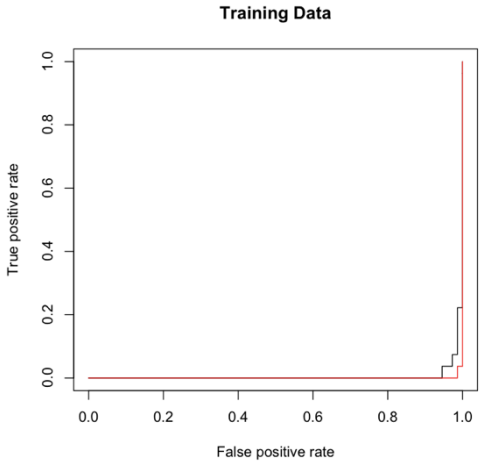
9.6.3 ROC Curves

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

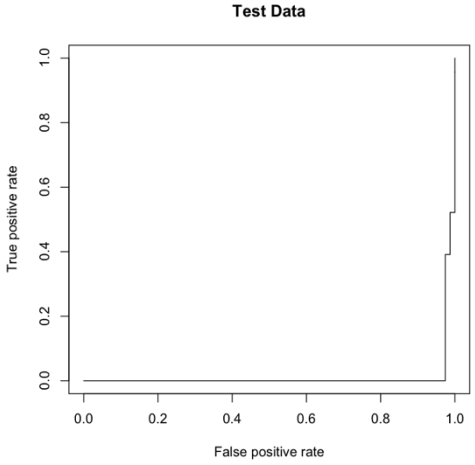
```
        plot(perf,...)}
> svmfit.opt=svm(y~., data=dat[train,], kernel="radial", gamma=2, cost=1, decision.values=T)
> fitted=attributes(predict(svmfit.opt,dat[train,],decision.values=TRUE))$decision.values
> par(mfrow=c(1,2))
> rocplot(fitted,dat[train,"y"],main="Training Data")
```



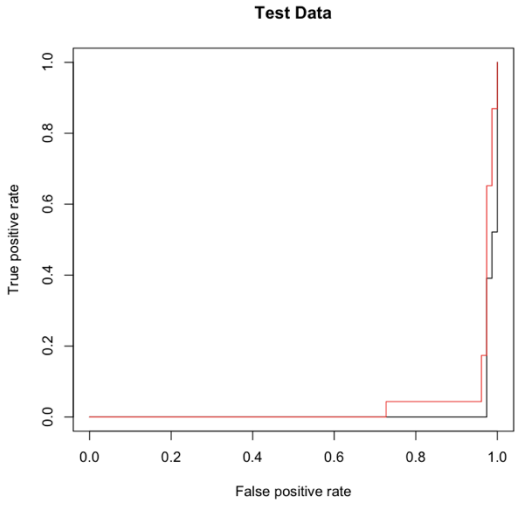

```
> svmfit.flex=svm(y~., data=dat[train,],
kernel="radial", gamma=50, cost=1, decision.values=T)
> fitted1=attributes(predict(svmfit.flex,dat[train,],decision.values=T))$decision.values
> rocplot(fitted,dat[train,"y"],main="Training
Data")+rocplot(fitted1,dat[train,"y"],add=T,col="red")
```




```
> fitted=attributes(predict(svmfit.opt,dat[-
train,],decision.values=T))$decision.values
> rocplot(fitted,dat[-train,"y"],main="Test Data")
```

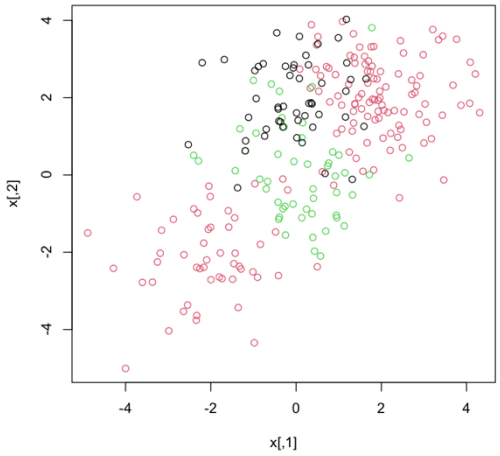


```
> fitted1=attributes(predict(svmfit.flex,dat[-train,],decision.values=T))$decision.values
> rocplot(fitted,dat[-train,"y"],main="Test Data")+rocplot(fitted1,dat[-
train,"y"],add=T,col="red")
```

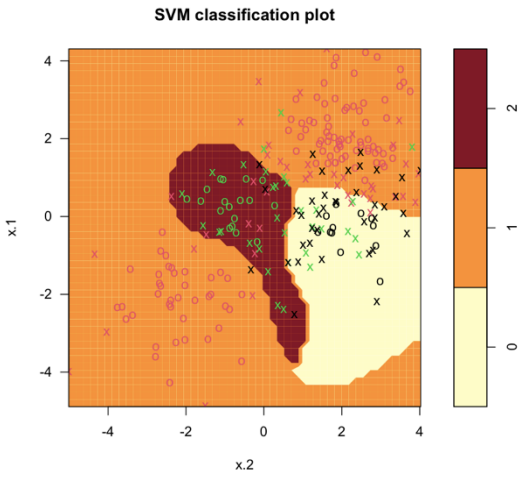


9.6.4 SVM with Multiple Classes

```
> set.seed(1)
> x=rbind(x, matrix(rnorm(50*2),ncol=2))
> y=c(y, rep(0,50))
> x[y==0,2]=x[y==0,2]+2
> dat=data.frame(x=x, y=as.factor(y))
> par(mfrow=c(1,1))
> plot(x,col=(y+1))
```




```
> svmfit=svm(y~., data=dat, kernel="radial", cost=10,
gamma=1)
> plot(svmfit, dat)
```



9.6.5 Application to Gene Expression Data

```
> library(ISLR)
> names(Khan)
[1] "xtrain" "xtest"  "ytrain" "ytest"
> dim(Khan$xtrain)
[1]   63 2308
> dim(Khan$xtest)
[1]   20 2308
> length(Khan$ytrain)
[1] 63
> length(Khan$ytest)
[1] 20
> table(Khan$ytrain)

 1  2  3  4
 8 23 12 20
> table(Khan$ytest)

1 2 3 4
3 6 6 5
> dat=data.frame(x=Khan$xtrain, y=as.factor(Khan$ytrain))
> out=svm(y~., data=dat, kernel="linear", cost=10)
> summary(out)

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


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

Number of Support Vectors:  58

 ( 20 20 11 7 )


Number of Classes:  4

Levels:
 1 2 3 4
```

```
> table(out$fitted, dat$y)

   1  2  3  4
1  8  0  0  0
2  0 23  0  0
3  0  0 12  0
4  0  0  0 20

> dat.te=data.frame(x=Khan$xtest, y=as.factor(Khan$ytest))
> pred.te=predict(out, newdata=dat.te)
> table(pred.te, dat.te$y)

pred.te 1 2 3 4
     1 3 0 0 0
     2 0 6 2 0
     3 0 0 4 0
     4 0 0 0 5
```