Ch.9 Exercises: Support Vector Machines

```
library(e1071)
library(ISLR)
require(caTools)
```

```
## Loading required package: caTools
```

```
## Warning: package 'caTools' was built under R version 3.6.2
```

```
require(plotrix)
```

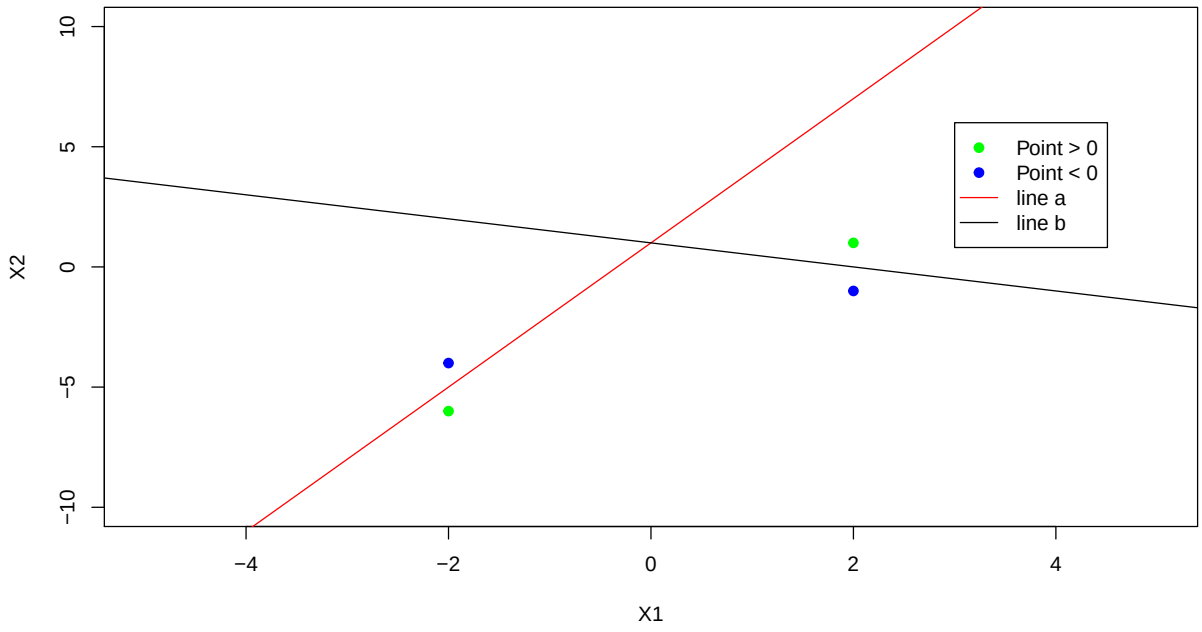
```
## Loading required package: plotrix
```

Conceptual

1.

(a) (b)

```
plot(0,type="n",xlab='X1', ylab='X2',ylim = c(-10,10),xlim = c(-5,5))
abline(1,3,col="red")      #Line (a): 1+3X1-X2=0
abline(1,-0.5,col="black") #Line (b): -2+X1+2X2=0

# Points where Line(a)>0 and Line(a)<0
points(-2,-4,col="blue",pch=19)
points(-2,-6,col="green",pch=19)

# Points where Line(b)>0 and Line(b)<0
points(2,1,col="green",pch=19)
points(2,-1,col="blue",pch=19)

legend(3,6,legend=c("Point > 0", "Point < 0", "line a", "line b"),
       col=c("green", "blue", "red", "black"), pch=c(19,19,NA,NA),lty=c(NA,NA,1,1))
```

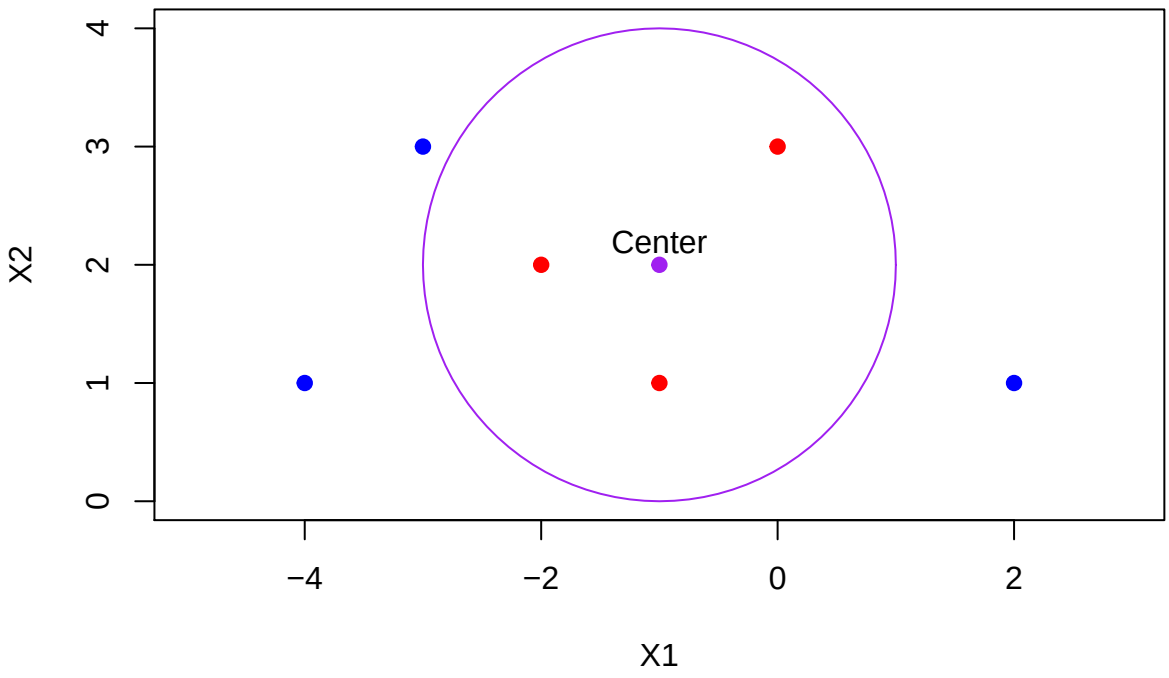


2.

- The equation is that of a circle in the format : $(x–h)^2 + (y–k)^2 = r^2$
- Where (h, k) is the center of the circle.
- So, we have : $(X1 - (-1))^2 + (-X2 + 2)^2 = 2^2$.
- So, the decision boundary is a circle with radius = 2 and center at (-1,2).

(a) (b)

```
# Drawing a circle on a plot with a radius of 2 and center at (-1,2).
plot(x=seq(-3,1), y=seq(0,4),type="n", asp = 1, xlab='X1', ylab='X2')
draw.circle(-1,2,2,border = 'purple')
points(-1,2, col='purple', pch=19)
text(-1,2.2,'Center')

# Points outside decision boundary.
points(c(-4,2,-3),c(1,1,3), col="blue",pch=19)

# Points inside decision boundary.
points(c(-1,-2,0),c(1,2,3), col="red",pch=19)
```



(c)

- (0,0) : Blue , (-1,1) : Red , (2,2) : Blue , (3,8) : Blue

(d)

Expanding out the decision boundary, we get:

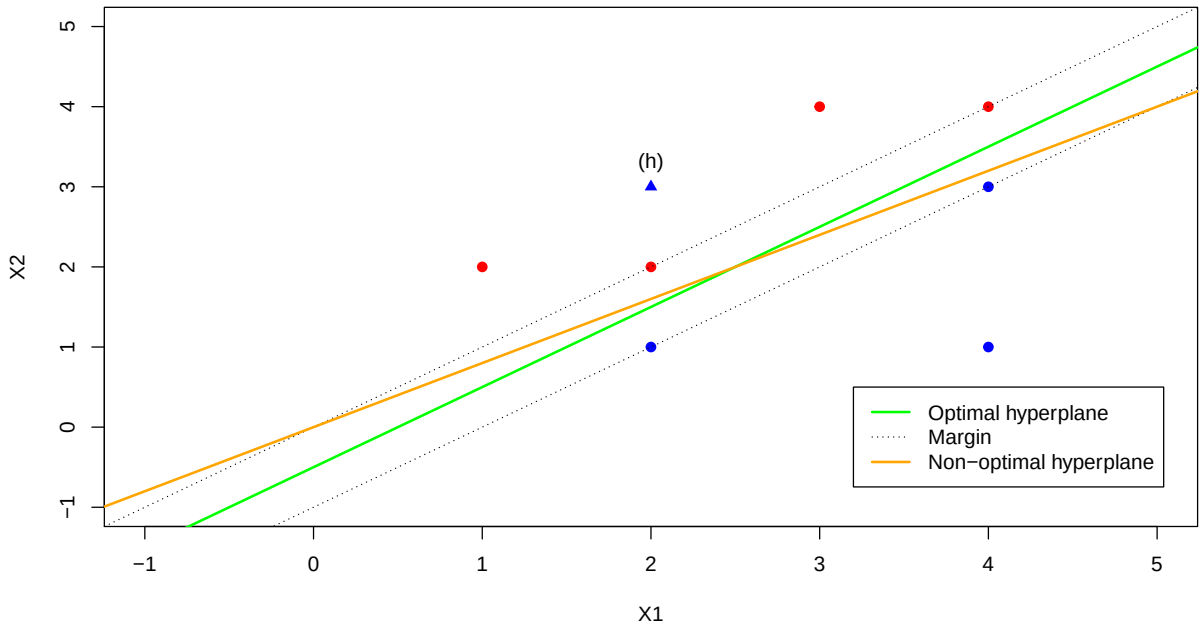
$$(1 + 2X_1 + X_1^2) + (4 - 4X_2 + X_2^2) = 4$$
$$1 + 2X_1 - 4X_2 + X_1^2 + X_2^2 = 0$$

The resulting equation shows that the circular boundary is linear in the feature space (X_1^2, X_2^2, X_1, X_2). In other words, the 2D equation of a circle when expanded to include X_1^2,X_2^2 becomes a linear equation.

3 (a)

```
plot(-1:5,-1:5,type="n",xlab='X1', ylab='X2')
points(c(3,2,4,1),c(4,2,4,2), col="red",pch=19)
points(c(2,4,4),c(1,3,1), col="blue",pch=19)
points(2,3,col="blue",pch=17)

abline(-0.5,1,col='green',lwd=2) #y intercept=-0.5 and gradient=1.
abline(-1, 1, col='black',lty='dotted')
abline(0, 1, col='black',lty='dotted')
abline(0,0.8, col="orange", lwd=2)
text(2,3.3,'(h)')
```

```
legend(3.2,0.5,legend=c("Optimal hyperplane", "Margin", "Non-optimal hyperplane"),
       col=c("green", "black", "orange"),lty=c(1,3,1), lwd=c(2,1,2))
```



(b)

Given equation of a line:

$$y = mx + c$$

And given X1, X2, m=1 and c=-0.5, we have:

$$X_2 - X_1 + 0.5 = 0$$

(c)

- Class Red if $X_2 - X_1 + 0.5 > 0$ and Blue otherwise. $\beta_0 = 0.5; \beta_1 = -1; \beta_2 = 1$

(d) (e) (g) (h)

- See above chart.

(f)

- Observation seven is not a support vector, and is also located far from the current support vectors. Moving it slightly towards the margin will not make it a support vector, and given that only the support vectors can affect the maximal margin classifier, it holds that observation seven won't affect the classifier.

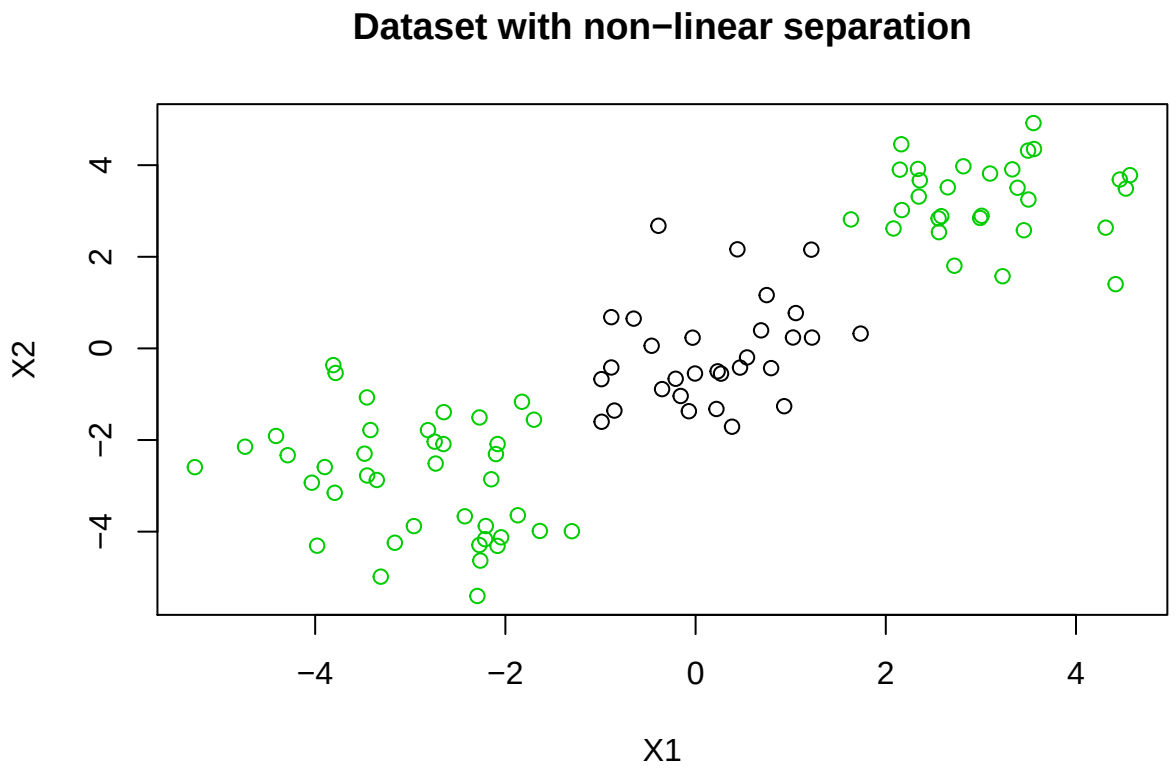
Applied

4.

Support Vector Classifier

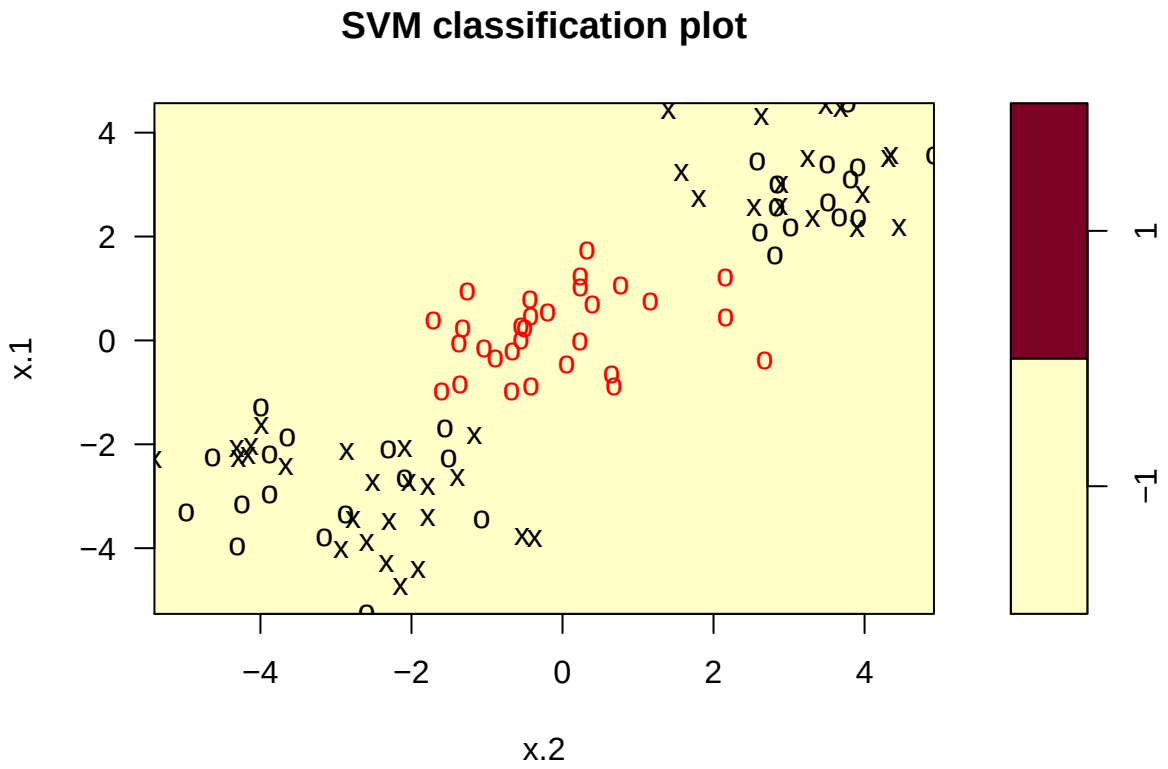
```
# Generating a dataset with  visible non-linear separation.
set.seed(3)

x=matrix(rnorm(100*2), ncol=2)
y=c(rep(-1,70), rep(1,30))
x[1:30,]=x[1:30,]+3.3
x[31:70,]=x[31:70,]-3
dat=data.frame(x=x, y=as.factor(y))


# Training and test sets.
sample.data = sample.split(dat$x.1, SplitRatio = 0.7)
train.set = subset(dat, sample.data==T)
test.set = subset(dat, sample.data==F)
```

```
plot(x,col=(2-y), xlab='X1', ylab='X2', main='Dataset with non-linear separation')
```




```
# Best model.
set.seed(3)
tune.out=tune(svm,y~.,data = train.set,kernel='linear',
```

```
              ranges=list(cost=c(0.001,0.01,0.1,1,5,10,100)))
bestmod=tune.out$best.model
plot(bestmod, dat)
```




```
# Predictions on training set.
ypred=predict(bestmod, train.set)
table(predict=ypred, truth=train.set$y)
```

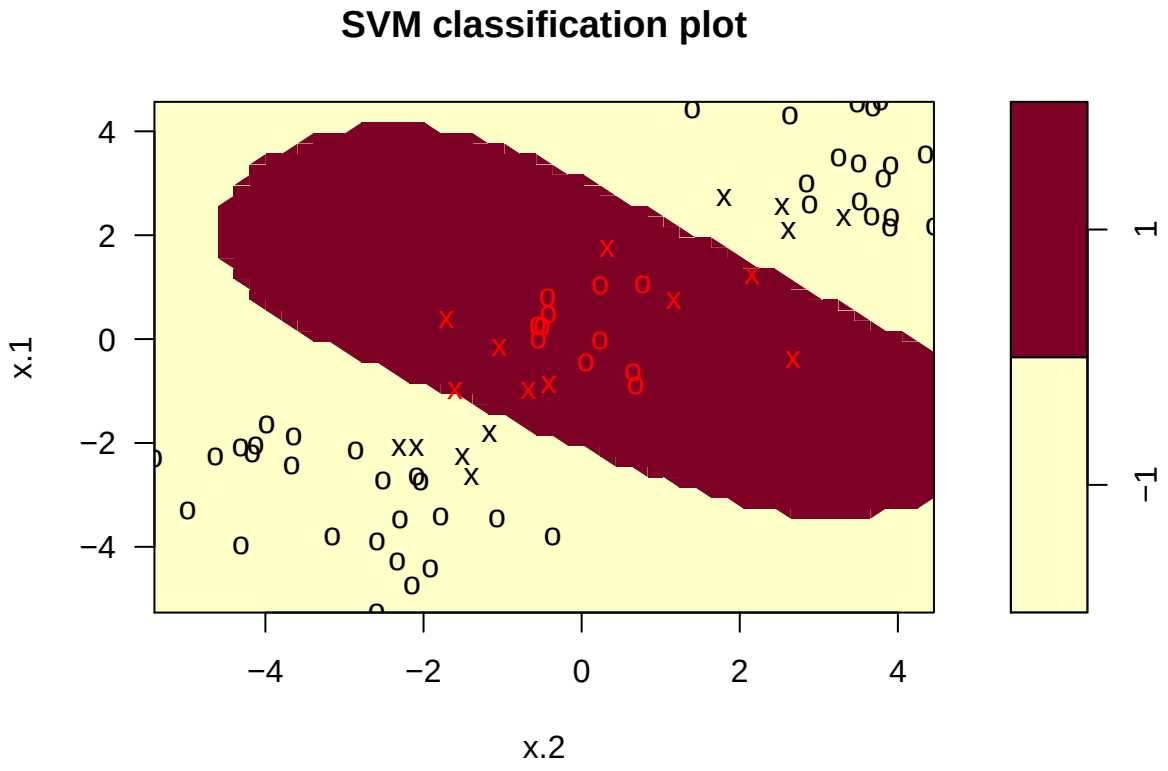
```
##        truth
## predict -1  1
##      -1 50 20
##      1   0  0
```

```
#Predictions on the test set.
ypred=predict(bestmod, test.set)
table(predict=ypred, truth=test.set$y)
```

```
##        truth
## predict -1  1
##      -1 20 10
##      1   0  0
```

SVM with a radial kernel

```
# Best model using cross validaitonon a set of values for cost and gamma.
set.seed(3)
tune.out=tune(svm, y~., data=train.set, kernel='radial',
              ranges=list(cost=c(0.1,1,10,100,1000),gamma=c(0.5,1,2,3,4)))
bestmod = tune.out$best.model
plot(bestmod, train.set)
```




```
# Predictions on training set.
ypred=predict(bestmod, train.set)
table(predict=ypred, truth=train.set$y)
```

```
##        truth
## predict -1  1
##      -1 50  0
##       1  0 20
```

```
#Predictions on the test set.
ypred=predict(bestmod, test.set)
table(predict=ypred, truth=test.set$y)
```

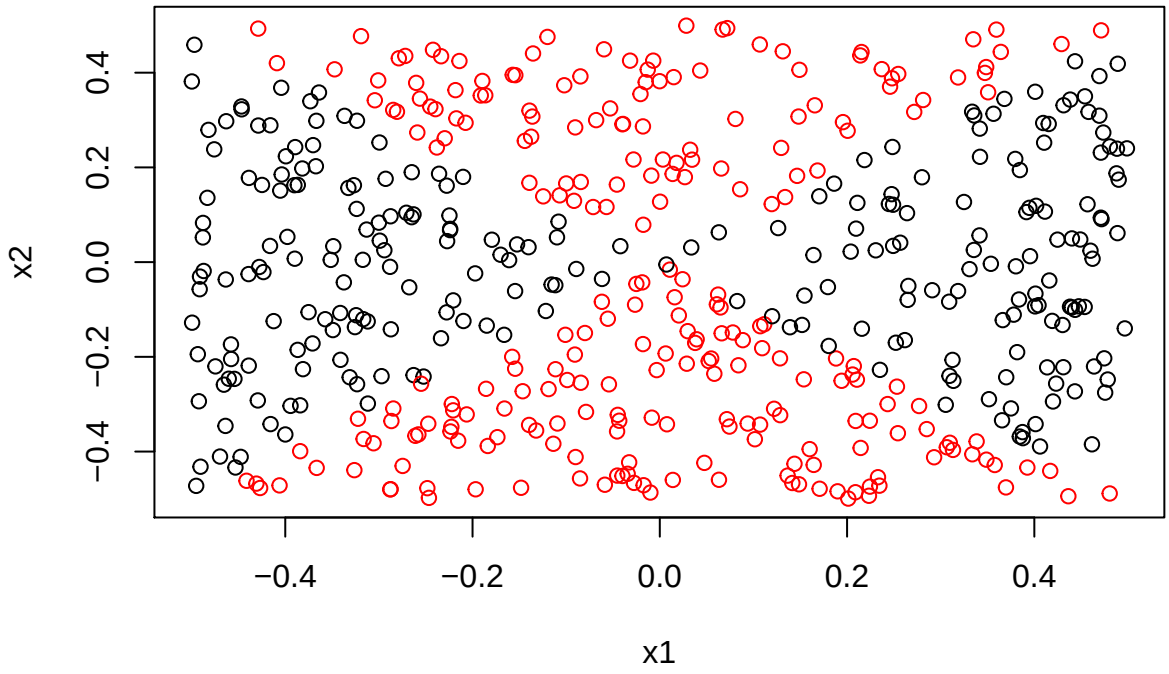
```
##        truth
## predict -1  1
##      -1 20  0
##       1  0 10
```

- The svm with a linear kernel classifies 20/70 training observations and 10/30 of the test observations wrongly.
- The svm with a radial kernel classifies all observations correctly.

5.

(a) (b)

```
set.seed(4)
x1 = runif(500)-0.50
x2 = runif(500)-0.50
y = 1*(x1^2-x2^2 > 0)
df=data.frame(x1=x1, x2=x2, y=as.factor(y))
plot(x1,x2,col = (2 - y))
```



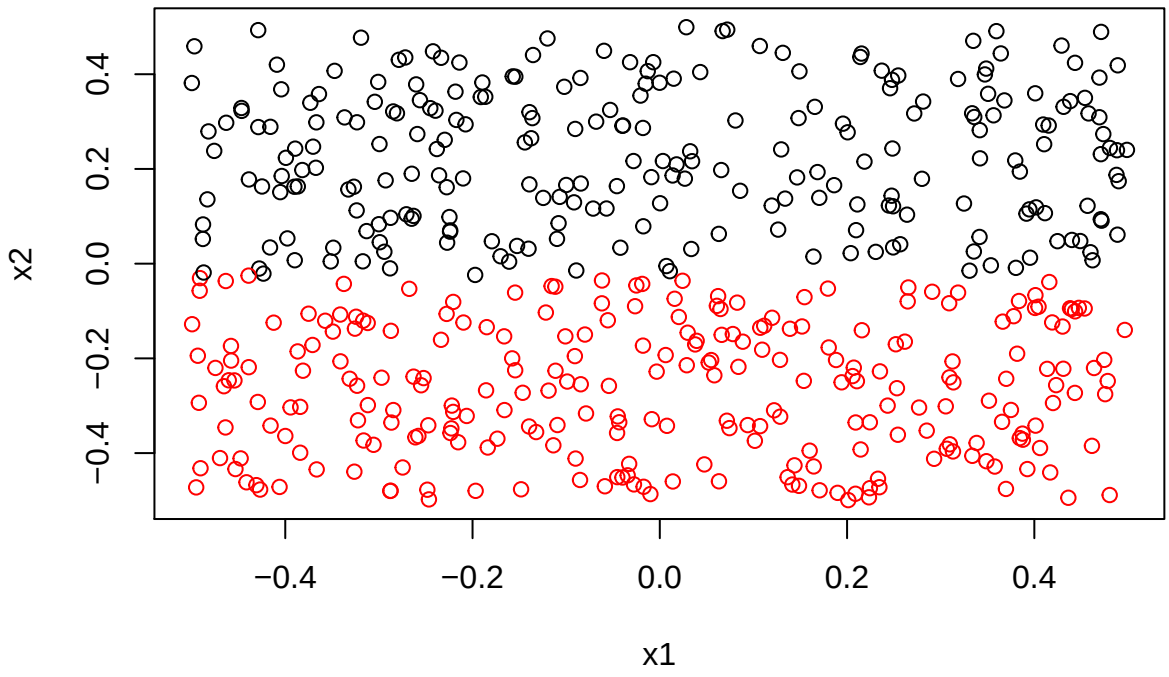
(c) (d)

Logistic Regression

```
glm.fit = glm(y~x1+x2, data=df, family = 'binomial')

# Predictions
glm.probs = predict(glm.fit, newdata=df, type = 'response')
glm.preds = rep(0,500)
glm.preds[glm.probs>0.50] = 1

table(preds=glm.preds, truth=df$y)
```

```
##       truth
## preds   0   1
##     0 144 113
##     1 106 137
```

```
# Plot using predicted class labels for observations.
plot(x1,x2,col=2-glm.preds)
```



- As expected, the decision boundary is linear.
- Error rate: 43.8%.
- The predicted probabilities, and hence the error rates are impacted by the set.seed value. In some cases all the observations are predicted to be the same class.

(e) (f)

Logistic regression using non-linear predictors

```
glm.fit = glm(y~I(x1^2)+I(x2^2), data = df, family = 'binomial')
```

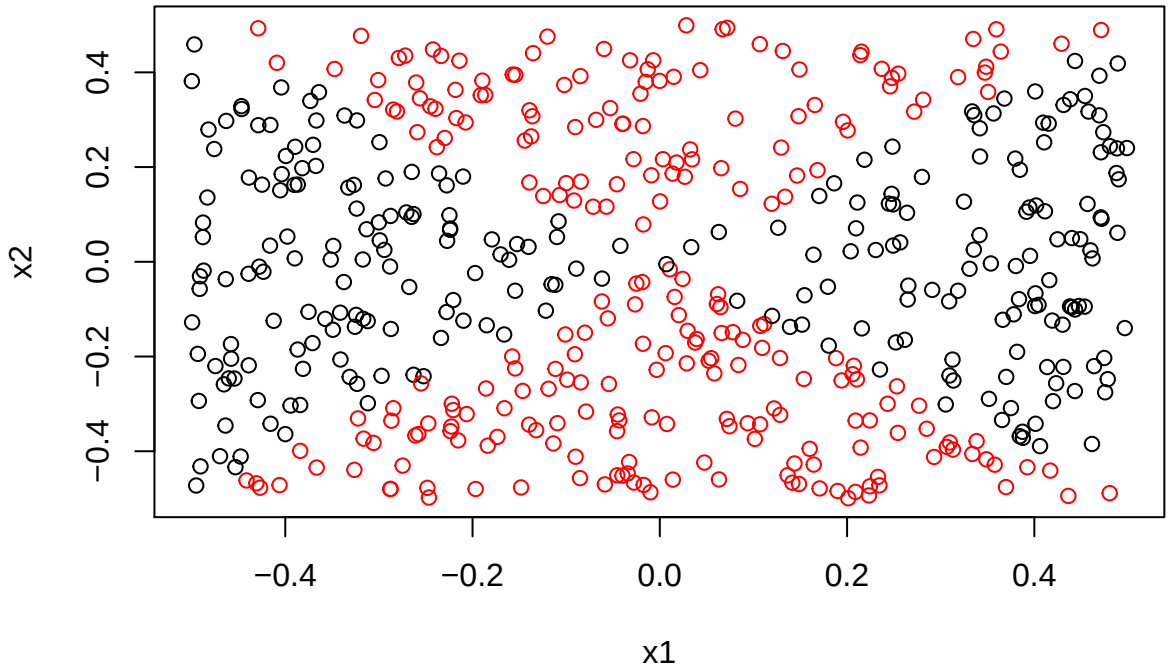
```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
glm.probs = predict(glm.fit, newdata = df, type = 'response')
glm.preds = rep(0,500)
glm.preds[glm.probs>0.5] = 1
table(preds=glm.preds, truth=df$y)
```

```
##       truth
## preds   0   1
##     0 250   0
##     1   0 250
```

```
plot(x1,x2,col=2-glm.preds)
```



- Quadratic transformations of X1 and X2 result in perfect separation.
- All observations are correctly classified.
- Error rate : 0%.

(g)

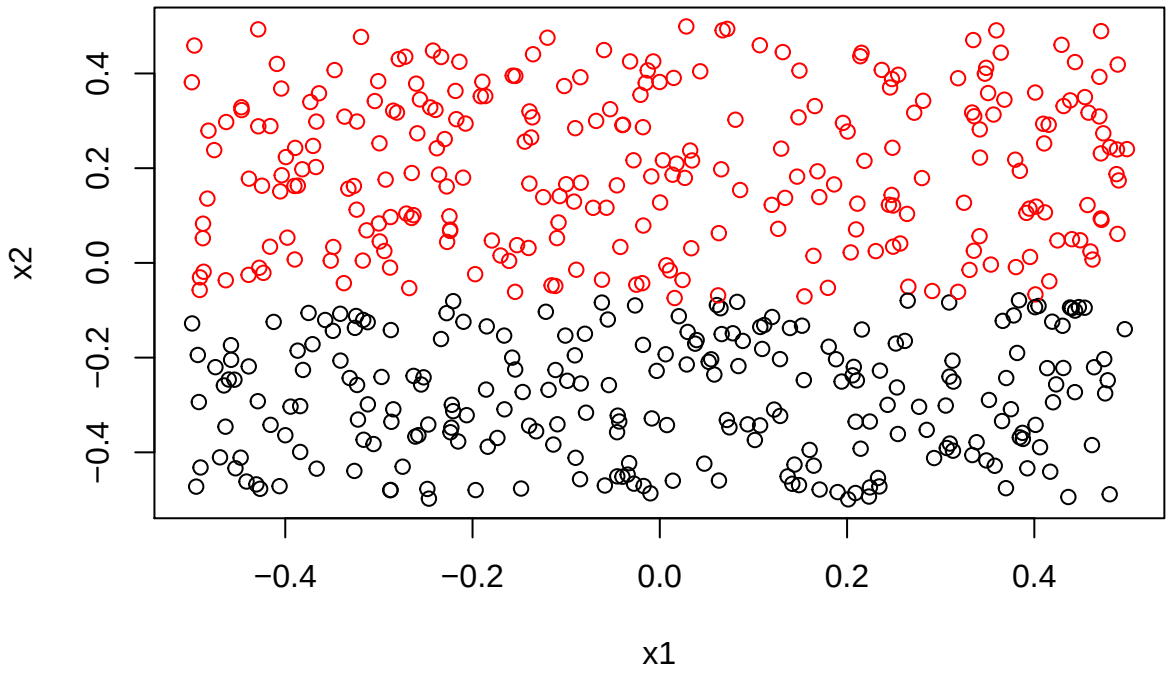
Support Vector Classifier

```
#Best model
tune.out=tune(svm,y~.,data = df,kernel='linear',
              ranges=list(cost=c(0.001,0.01,0.1,1,5,10,100)))
bestmod=tune.out$best.model
```

```r
#Predictions
ypred=predict(bestmod, newdata=df, type='response')
table(predict=ypred, truth=df$y)
```

```
##         truth
## predict   0   1
##       0 139  96
##       1 111 154
```

```r
plot(x1,x2,col=ypred)
```



- Error rate: 41.4%.

(h)

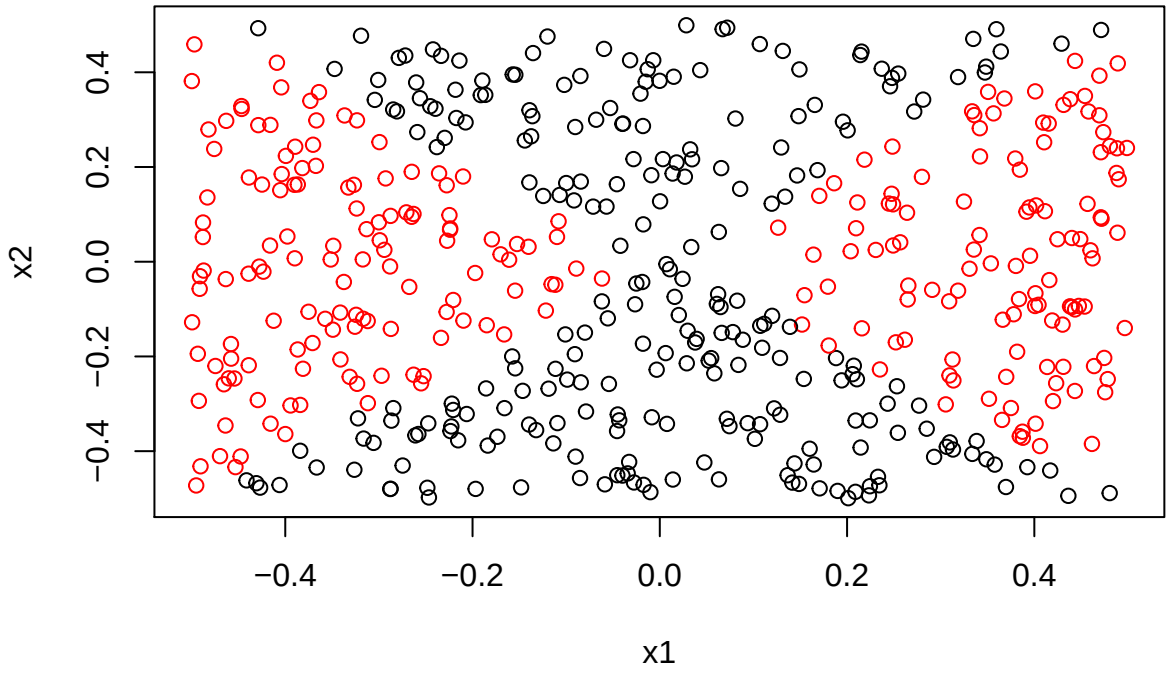
SVM with a non-linear kernel

```r
tune.out=tune(svm, y~., data=df, kernel='radial',
              ranges=list(cost=c(0.1,1,10,100,1000),gamma=c(0.5,1,2,3,4)))
bestmod=tune.out$best.model
```

```r
#Predictions
ypred=predict(bestmod, newdata=df, type='response')
table(predict=ypred, truth=df$y)
```

```
##         truth
## predict   0   1
##       0 247   7
##       1   3 243
```

```
plot(x1,x2,col=ypred)
```



- Error rate: 3.6%. **i**
- The logistic regression models using linear terms and non-linear terms closely match the SVM with linear and radial kernels respectively.
- As expected, logistic regression with non-linear terms and SVM with a radial kernel outperform the models using linear terms. These models achieve near perfect accuracy in predicting the class of the training observations.
- The results confirm that Logistic Regression and SVM are similar methods.

6.

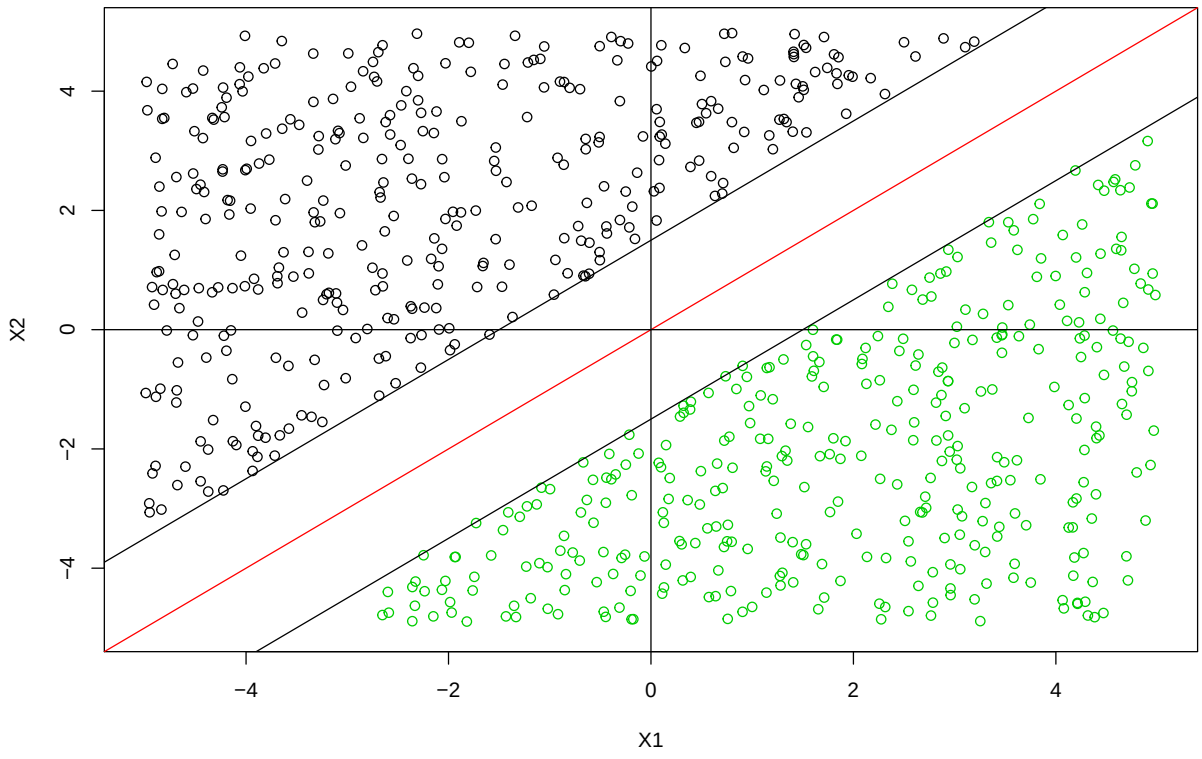
(a)

```
set.seed(111)
x1 = runif(1000,-5,5)
x2 = runif(1000,-5,5)
x = cbind(x1,x2)
```

```
y = rep(NA,1000)

# Classify points above abline(1.5,1) as 1 and below abline(-1.5,1) as -1, and the rest as 0.
#Removing points classed as 0 will created a more widely separated dataset.
# Actual decision boundary is a line where y=x, which is abline(0,1).

for (i in 1:1000)
  if (x[i,2]-x[i,1] > 1.5) y[i]=1 else if (x[i,2]-x[i,1] < -1.5) y[i]=-1 else y[i]=0
```

```
# Combine x an y  and remove all rows with y=0.
x = cbind(x,y)
x = x[x[,3]!=0,]
```

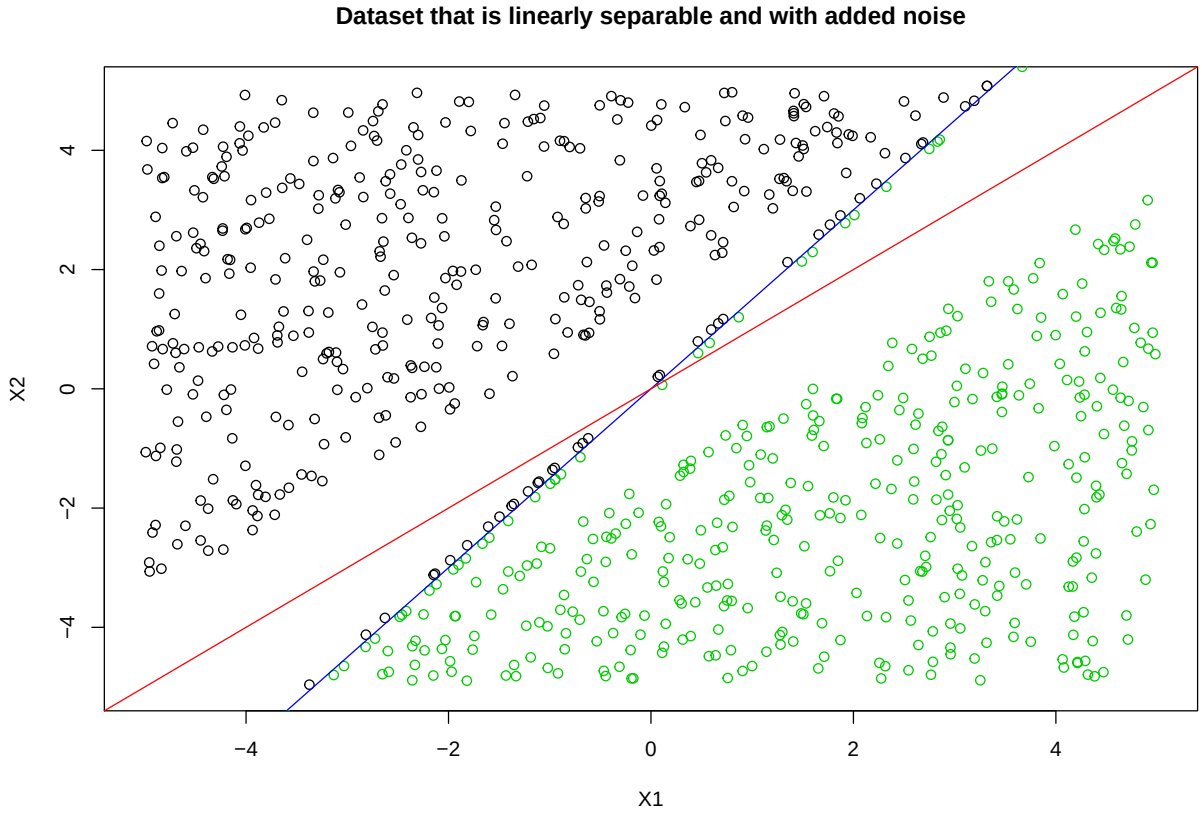
```
plot(x[,1],x[,2],col=2-x[,3], xlab="X1", ylab="X2",xlim = c(-5,5), ylim = c(-5,5))
abline(0,1, col="red")
abline(1.5,1)
abline(-1.5,1)
abline(h=0,v=0)
```




```
#Generate random points to be used as noise along line y=1.5x(+-0.1).
x.noise = matrix(NA,100,3)

x.noise[,1] = runif(100,-5,5)
x.noise[1:50,2] = (1.5*x.noise[1:50,1])-0.1      #Y co-ordinate values for first 50 points
```

```r
x.noise[51:100,2] = (1.5*x.noise[51:100,1])+0.1

x.noise[,3] = c(rep(-1,50), rep(1,50)) # class values for all noise observations
```

```r
plot(x[,1],x[,2],col=2-x[,3], xlab='X1', ylab='X2',
     ylim = c(-5,5),xlim = c(-5,5),
     main="Dataset that is linearly separable and with added noise")

par(new = TRUE)
plot(x.noise[,1],x.noise[,2],col=2-x.noise[,3], axes=F,
     xlab="", ylab="", ylim = c(-5,5), xlim = c(-5,5))

#Noise
abline(0,1.5,col="blue")

#Actual decision boundary
abline(0,1,col="red")
```



(b)

```r
x = rbind(x,x.noise)
train.dat = data.frame(x1=x[,1],x2=x[,2], y=as.factor(x[,3]))
```

```r
#Linear SVM models with various values of cost.
tune.out=tune(svm,y~.,data=train.dat,kernel='linear',
              ranges=list(cost=c(0.001,0.01,0.1,1,5,10,100,1000)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   100
##
## - best performance: 0
##
## - Detailed performance results:
##    cost      error dispersion
## 1 1e-03 0.05351161 0.02002264
## 2 1e-02 0.05352630 0.01834727
## 3 1e-01 0.05352630 0.01834727
## 4 1e+00 0.05352630 0.01834727
## 5 5e+00 0.04745813 0.02110627
## 6 1e+01 0.04138995 0.02317355
## 7 1e+02 0.00000000 0.00000000
## 8 1e+03 0.00000000 0.00000000
```

```r
# Training set misclassification
tune.out$performances$cost
```

```
## [1] 1e-03 1e-02 1e-01 1e+00 5e+00 1e+01 1e+02 1e+03
```

```r
tune.out$performances$error*822
```

```
## [1] 43.98654 43.99862 43.99862 43.99862 39.01058 34.02254  0.00000  0.00000
```

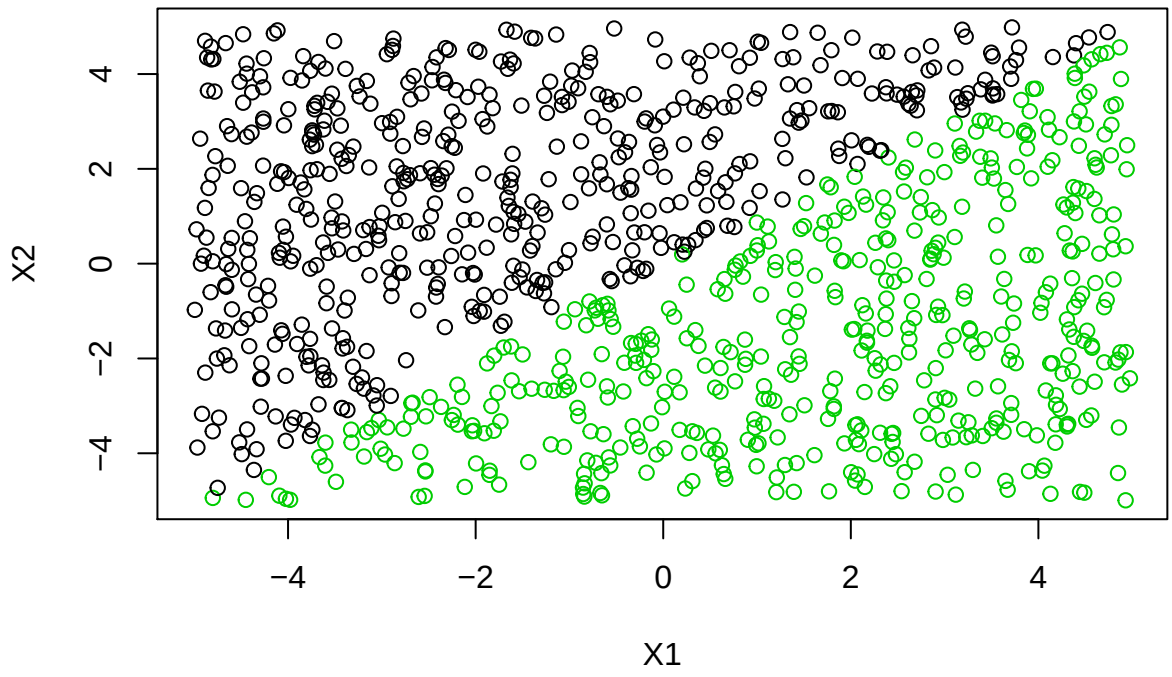
- Fewer training observations are misclassified as cost increases.
- The CV error is directly related to the number of training errors. A lower CV means fewer training errors.

(c)

```r
# New test set
set.seed(1221)
test.x1 = runif(1000,-5,5)
test.x2 = runif(1000,-5,5)
test.y = rep(NA,1000)

# Actual decision boundary of the train set is y=x,
# so points above line are classed as 1 and points below -1.
```

```r
for (i in 1:1000)
  if (test.x1[i]-test.x2[i] < 0) test.y[i]=1 else if (test.x1[i]-test.x2[i] > 0) test.y[i]=-1

# Test dataframe
test.dat = data.frame(x1=test.x1,x2=test.x2,y=as.factor(test.y))
```

```r
plot(test.dat$x1,test.dat$x2,col=2-test.y, xlab="X1", ylab="X2")
```




```r
# Performance of model with cost of 0.1 on test set.
svmfit = svm(y~., data = train.dat, kernel = 'linear', cost = 0.1)
ypred = predict(svmfit, newdata = test.dat, type = 'response')
table(predict=ypred, truth=test.dat$y)
```

```
##         truth
## predict  -1   1
##      -1 483  10
##       1   4 503
```

```r
svmfit2 = svm(y~., data = train.dat, kernel = 'linear', cost = 10)
ypred = predict(svmfit2, newdata = test.dat, type = 'response')
table(predict=ypred, truth=test.dat$y)
```

```
##         truth
```

```
## predict  -1   1
##      -1 446  48
##      1   41 465
```

```
svmfit3 = svm(y~., data = train.dat, kernel = 'linear', cost = 100)
ypred = predict(svmfit3, newdata = test.dat, type = 'response')
table(predict=ypred, truth=test.dat$y)
```

```
##        truth
## predict  -1   1
##      -1 444  51
##      1   43 462
```

- As can be seen from the truth tables above, the model with a cost of 0.1 performs much better than models with higher cost on the test set.
- On the training set the models with a cost above 100 made no misclassification errors.

(d)

- The higher cost models have a very small margin, and so there will be fewer support vectors on the margin or violating the margin. This allows the model to classify most of the training observations correctly. These models perform very well on the training set and poorly on the test set.
- The lower cost models will have a wider margin, and so there will be more support vectors on the margin or violating the margin. This means the model classifies more of the training observations incorrectly. These models perform poorly on the training set and much better on the test set.
- From these results, we can surmise that the higher cost models are over fitting the training data.

7.

(a)

```
head(Auto)
```

```
##   mpg cylinders displacement horsepower weight acceleration year origin
## 1  18         8          307        130   3504         12.0   70      1
## 2  15         8          350        165   3693         11.5   70      1
## 3  18         8          318        150   3436         11.0   70      1
## 4  16         8          304        150   3433         12.0   70      1
## 5  17         8          302        140   3449         10.5   70      1
## 6  15         8          429        198   4341         10.0   70      1
##                       name
## 1 chevrolet chevelle malibu
## 2         buick skylark 320
## 3        plymouth satellite
## 4             amc rebel sst
## 5               ford torino
## 6          ford galaxie 500
```

```r
set.seed(222)
auto.length = length(Auto$mpg)
mpg.median = median(Auto$mpg)
mpg01 = rep(NA,auto.length)

# Class 1 if car's mpg is above median and 0 if below. Results stored in mpg01 variable.
for (i in 1:auto.length) if (Auto$mpg[i] > mpg.median) mpg01[i]=1 else mpg01[i]=0

# Dataframe
auto.df = Auto
auto.df$mpg01 = as.factor(mpg01)
```

(b)

```r
# Using a linear SVM to predict mpg01.
linear.tune=tune(svm,mpg01~.,data=auto.df,kernel='linear',
                 ranges=list(cost=c(0.001,0.01,0.1,1,5,10,100,1000)))
summary(linear.tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.01275641
##
## - Detailed performance results:
##    cost      error dispersion
## 1 1e-03 0.09955128 0.04760888
## 2 1e-02 0.07666667 0.04375200
## 3 1e-01 0.04596154 0.02359743
## 4 1e+00 0.01275641 0.01808165
## 5 5e+00 0.01532051 0.01318724
## 6 1e+01 0.01788462 0.01234314
## 7 1e+02 0.03057692 0.01606420
## 8 1e+03 0.03057692 0.01606420
```

- The training CV error decreases as the cost increases, with a minimum at cost=1, thereafter it starts increasing.

SVMs with Radial and Polynomial Kernels

```r
set.seed(222)
# Radial kernel with various values of gamma and cost.
radial.tune=tune(svm, mpg01~., data=auto.df, kernel='radial',
                 ranges=list(cost=c(0.1,1,10,100,1000),gamma=c(0.5,1,2,3,4)))

radial.tune$best.parameters
```

```
##   cost gamma
## 2    1   0.5
```

```
radial.tune$best.performance
```

```
## [1] 0.04602564
```

- The training CV error is lowest for a radial model with cost=1 and gamma=0.5, but the value is around 4x higher than for the linear model.

```
# Polynomial kernel with various degrees
set.seed(222)
poly.tune = tune(svm, mpg01~., data=auto.df, kernel='polynomial',
                 ranges=list(cost=c(0.1,1,10,100,1000), degree=c(1,2,3,4,5)))

poly.tune$best.parameters
```

```
##   cost degree
## 5 1000      1
```

```
poly.tune$best.performance
```

```
## [1] 0.01019231
```

- The best polynomial model is with degree=1 and cost=1000.
- The lowest training CV errors are given by the linear SVM and polynomial with degree=1, and this suggest the true decision boundary is linear.
- We would have to test these models on a test set to properly ascertain which of the models is the best.

(d) Incomplete

8.

(a) (b)

```
set.seed(131)

# Training and test sets.
sample.data = sample.split(OJ$Purchase, SplitRatio = 800/length(OJ$Purchase))
train.set = subset(OJ, sample.data==T)
test.set = subset(OJ, sample.data==F)
```

```
svmfit = svm(Purchase~., data = train.set, kernel = "linear", cost=0.01)
summary(svmfit)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train.set, kernel = "linear",
##     cost = 0.01)
##
```

```
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  438
##
##  ( 220 218 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

- There are a large number of support vectors, so the margin is very wide.

(c)

```
# Predictions on training set
svm.pred = predict(svmfit, train.set)
table(predict=svm.pred, truth=train.set$Purchase)
```

```
##        truth
## predict  CH  MM
##      CH 435  76
##      MM  53 236
```

```
#Predictions on test set
svm.pred = predict(svmfit, test.set)
table(predict=svm.pred, truth=test.set$Purchase)
```

```
##        truth
## predict  CH  MM
##      CH 151  36
##      MM  14  69
```

- Training set error rate: $129/800 = 0.16$
- Test set error rate: $50/270 = 0.185$

(d)

```
# Using cross validation to select optimal cost
set.seed(131)
tune.out = tune(svm, Purchase~., data = train.set, kernel = "linear",
                ranges=list(cost=c(0.01,0.1,0.5,1,10)))
```

(e)

```r
# Training error
svm.pred = predict(tune.out$best.model, train.set)
table(predict=svm.pred, truth=train.set$Purchase)
```

```
##        truth
## predict  CH  MM
##      CH 432  72
##      MM  56 240
```

```r
# Test error
svm.pred = predict(tune.out$best.model, test.set)
table(predict=svm.pred, truth=test.set$Purchase)
```

```
##        truth
## predict  CH  MM
##      CH 151  34
##      MM  14  71
```

- Training error rate: $128/800 = 0.16$
- Test error rate: $48/270 = 0.178$
- Using the optimal value of cost improves the test error rate slightly.

(f)

Repeating (b) to (e) using a SVM with a radial kernel

```r
set.seed(131)
tune.out = tune(svm, Purchase~., data = train.set, kernel = "radial",
                ranges=list(cost=c(0.01,0.1,0.5,1,10)))
```

```r
# Training error
svm.pred = predict(tune.out$best.model, train.set)
table(predict=svm.pred, truth=train.set$Purchase)
```

```
##        truth
## predict  CH  MM
##      CH 450  81
##      MM  38 231
```

```r
# Test error
svm.pred = predict(tune.out$best.model, test.set)
table(predict=svm.pred, truth=test.set$Purchase)
```

```
##        truth
## predict  CH  MM
##      CH 150  37
##      MM  15  68
```

- Training error rate: $119/800 = 0.149$

- Test error rate: $52/270 = 0.193$

Repeating (b) to (e) using a SVM with a polynomial kernel

```
set.seed(131)
tune.out = tune(svm, Purchase~., data = train.set, kernel = "polynomial",
                ranges=list(cost=c(0.01,0.1,0.5,1,10)), degree=2)
```

```
# Training error
svm.pred = predict(tune.out$best.model, train.set)
table(predict=svm.pred, truth=train.set$Purchase)
```

```
##        truth
## predict  CH  MM
##      CH 455  76
##      MM  33 236
```

```
# Test error
svm.pred = predict(tune.out$best.model, test.set)
table(predict=svm.pred, truth=test.set$Purchase)
```

```
##        truth
## predict  CH  MM
##      CH 148  41
##      MM  17  64
```

- Training error rate: $109/800 = 0.136$
- Test error rate: $58/270 = 0.215$

(h)

- The optimal radial and polynomial models both have a lower training and higher test error rate than the linear SVM. This suggests that both models are over fitting the training set when compared to the linear SVM.
- The linear SVM with optimal cost has a test error rate that is slightly above its training error rate. The small increase is normal behaviour, and the fact that it is still below the radial and polynomial error rates strongly supports the linear SVM being the best model.