# Statistical Learning HW 8 - SVMs

*Christian Conroy*

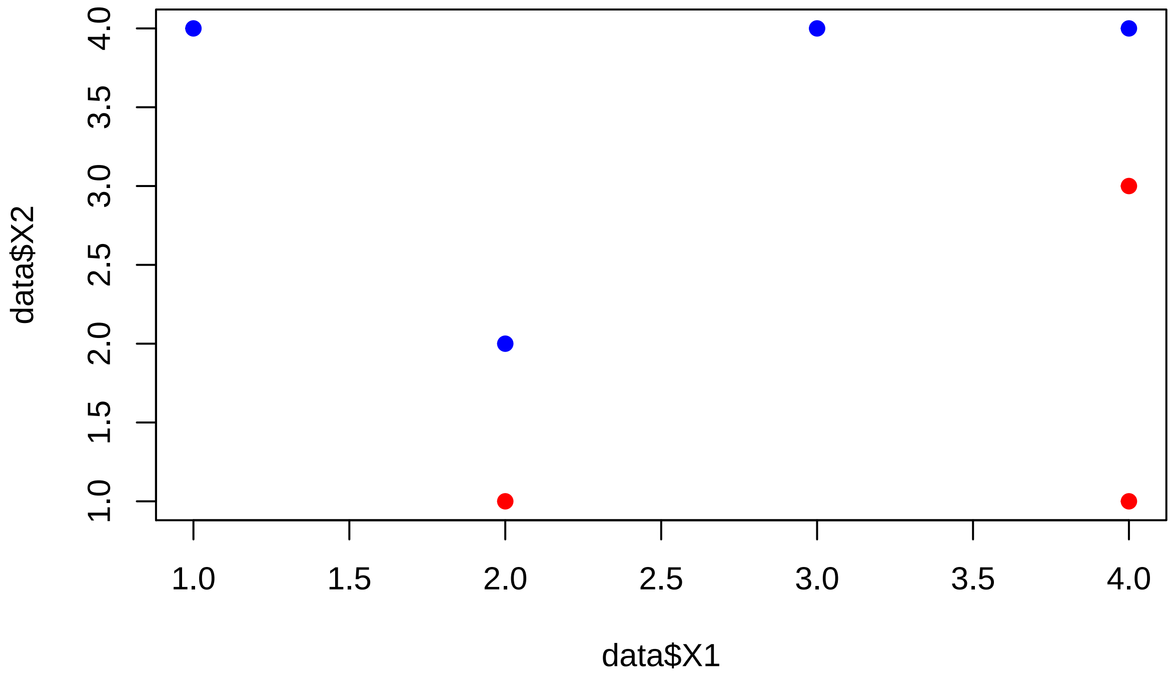
*April 25, 2019*

## Book 3 (5 points)

3. Here we explore the maximal margin classifier on a toy data set.

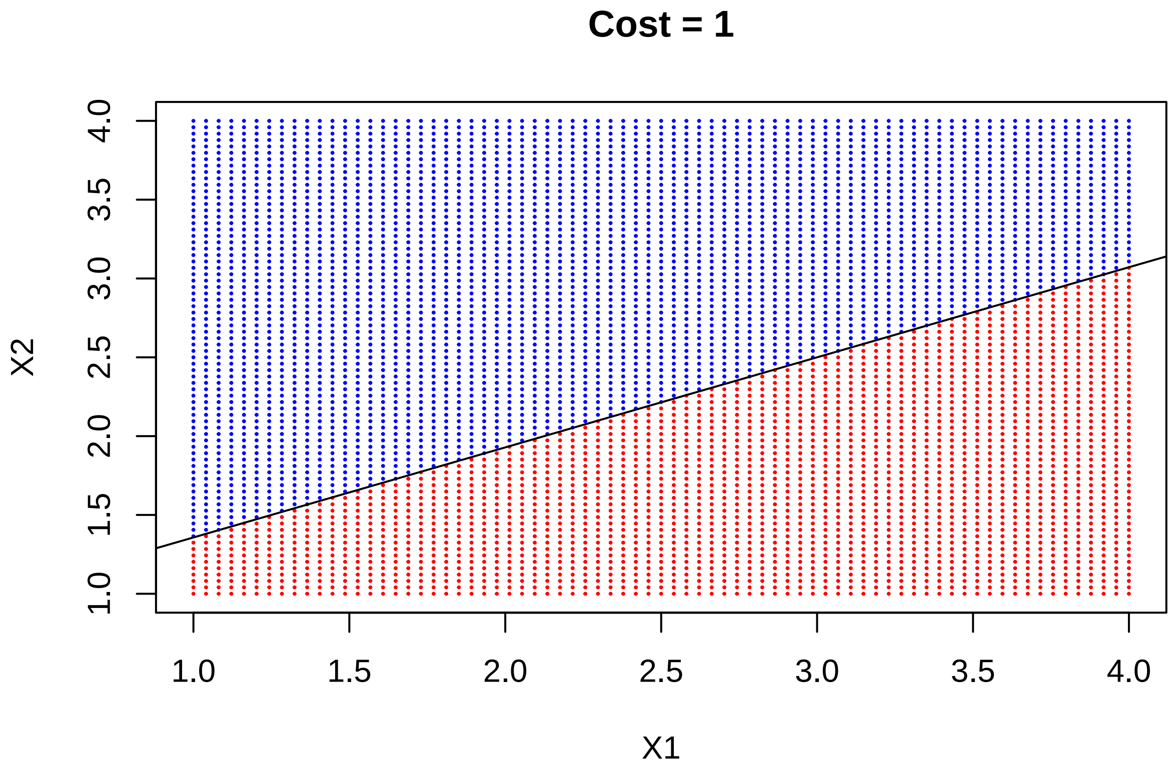
(a) We are given n = 7 observations in p = 2 dimensions. For each observation, there is an associated class label. Sketch the observations.

```
data <- data.frame(X1 = c(3, 2, 4, 1, 2, 4, 4),
                   X2 = c(4, 2, 4, 4, 1, 3, 1),
                   Y = c("Red", "Red", "Red", "Red", "Blue", "Blue", "Blue"))

plot(data$X1, data$X2, col = c("red", "blue")[data$Y], pch=19)
```



(b) Sketch the optimal separating hyperplane, and provide the equation for this hyperplane (of the form (9.1)).

```
# Make sure it outputs plot here insteaf of plot viewer!

# makegrid function
make.grid=function(x,n=75){
  grange=apply(x,2,range)
  x1=seq(from=grange[1,1],to=grange[2,1],length=n)
  x2=seq(from=grange[1,2],to=grange[2,2],length=n)
  expand.grid(X1=x1,X2=x2)
}

x <- as.matrix(data[,c(1:2)])
xgrid <- make.grid(x)
# make model with cost = 1
svm.model1 = svm(Y~.,data = data, kernel="linear",cost=1,scale=FALSE)
ygrid = predict(svm.model1,xgrid)
plot(xgrid,col=c("red","blue")[as.numeric(ygrid)],pch=20,cex=.2,
     main = "Cost = 1")
beta=drop(t(svm.model1$coefs)%*%x[svm.model1$index,])
beta0=svm.model1$rho
abline(beta0/beta[2],-beta[1]/beta[2])
```



```
svm.model1$coefs
```

```
##         [,1]
## [1,]  1.000
## [2,]  0.692
## [3,] -0.692
## [4,] -1.000
```

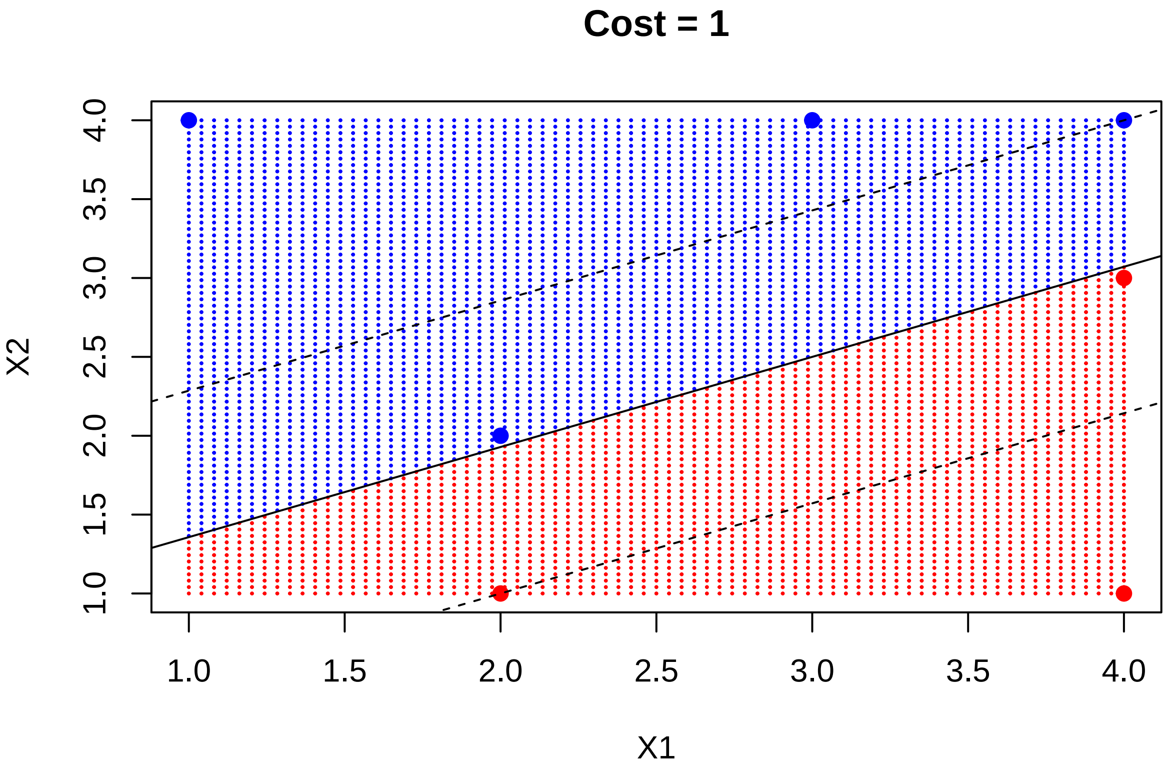
The hyperplane -0.615X1 + 1.077X2 + 0.846 = 0 is shown.

(c) Describe the classification rule for the maximal margin classifier. It should be something along the lines of "Classify to Red if $\beta_0 + \beta_1 X_1 + \beta_2 X_2 > 0$, and classify to Blue otherwise." Provide the values for $\beta_0$, $\beta_1$, and $\beta_2$.

The blue region is the set of points for which $-0.615X_1 + 1.077X_2 + 0.846 > 0$, and the red region is the set of points for which $-0.615X_1 + 1.077X_2 + 0.846 < 0$.

(d) On your sketch, indicate the margin for the maximal margin hyperplane.

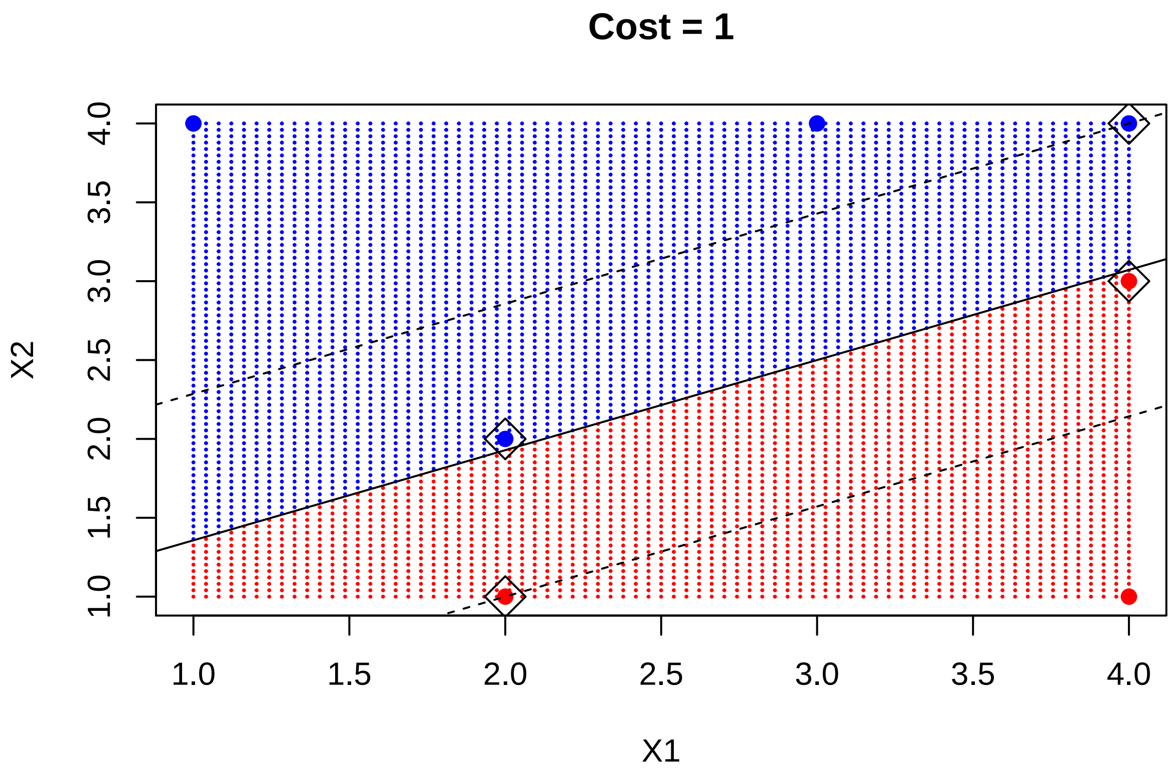
```
plot(xgrid,col=c("red","blue")[as.numeric(ygrid)],pch=20,cex=.2,
     main = "Cost = 1")
points(x,col=c("red", "blue")[data$Y],pch=19)
beta=drop(t(svm.model1$coefs)%*%x[svm.model1$index,])
beta0=svm.model1$rho
abline(beta0/beta[2],-beta[1]/beta[2])
abline((beta0-1)/beta[2],-beta[1]/beta[2],lty=2)
abline((beta0+1)/beta[2],-beta[1]/beta[2],lty=2)
```



(e) Indicate the support vectors for the maximal margin classifier.

```
plot(xgrid,col=c("red","blue")[as.numeric(ygrid)],pch=20,cex=.2,
     main = "Cost = 1")
points(x,col=c("red", "blue")[data$Y],pch=19)
points(x[svm.model1$index,],pch=5,cex=2)
beta=drop(t(svm.model1$coefs)%*%x[svm.model1$index,])
beta0=svm.model1$rho
abline(beta0/beta[2],-beta[1]/beta[2])
abline((beta0-1)/beta[2],-beta[1]/beta[2],lty=2)
```

```r
abline((beta0+1)/beta[2],-beta[1]/beta[2],lty=2)
```

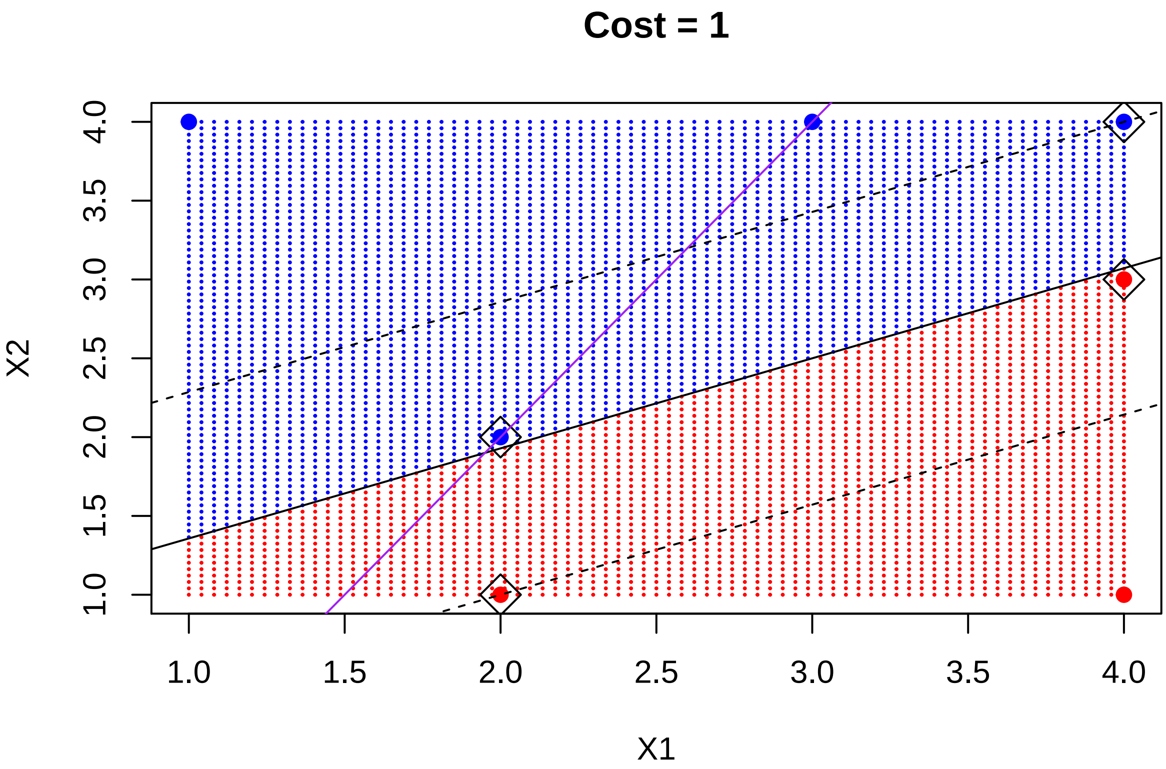


(f) Argue that a slight movement of the seventh observation would not affect the maximal margin hyperplane.

The maximal margin margin hyperplane is the separating hyperplane for which the margin is largest-that is, it is the hyperplane that has the farthest minimum distance to the training observations. So long as he seventh observation does not move closer to the boundary in a way that would change that farthest mi imum distance or move to a different side of the boundary, then moving the seventh observation will not affect the maximal margin hyperplane.

(g) Sketch a hyperplane that is not the optimal separating hyperplane, and provide the equation for this hyperplane.

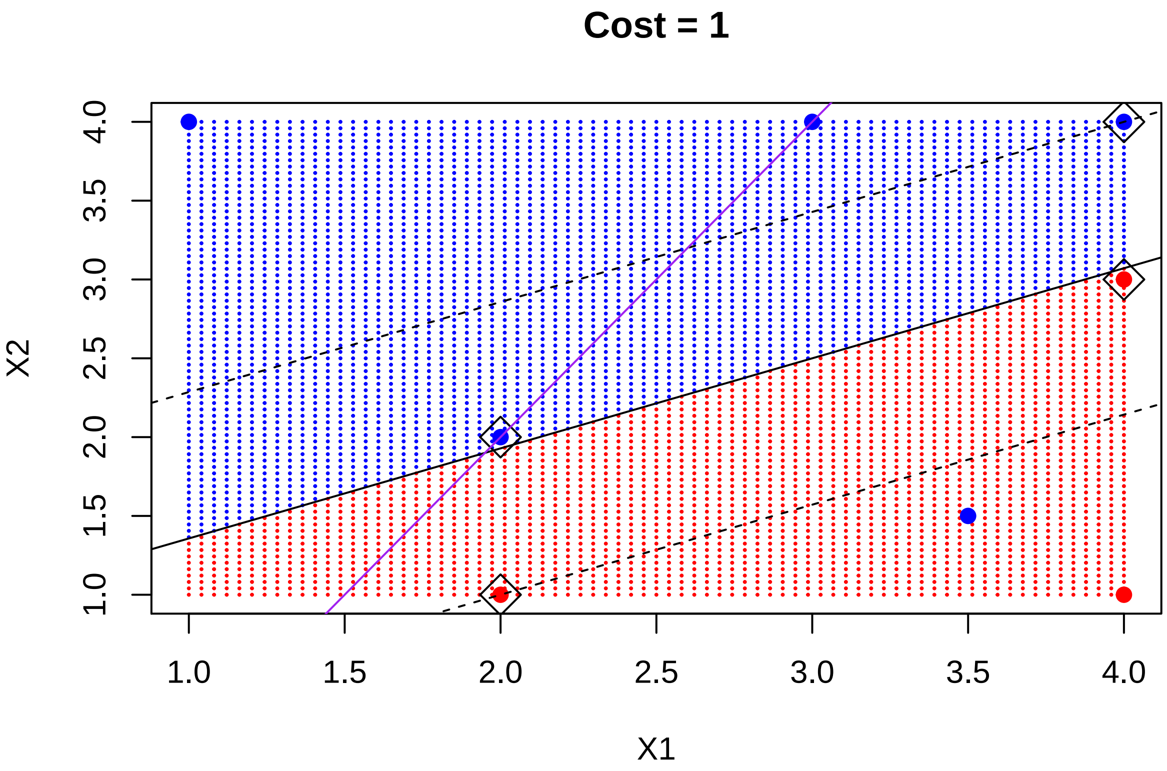
```r
plot(xgrid,col=c("red","blue")[as.numeric(ygrid)],pch=20,cex=.2,
     main = "Cost = 1")
points(x,col=c("red", "blue")[data$Y],pch=19)
points(x[svm.model1$index,],pch=5,cex=2)
beta=drop(t(svm.model1$coefs)%*%x[svm.model1$index,])
beta0=svm.model1$rho
abline(beta0/beta[2],-beta[1]/beta[2])
abline((beta0-1)/beta[2],-beta[1]/beta[2],lty=2)
abline((beta0+1)/beta[2],-beta[1]/beta[2],lty=2)
abline(-2, 2, col = "purple")
```



The hyperplane $2X_1$ + $X_2$ - 2 = 0 is shown.

(h) Draw an additional observation on the plot so that the two classes are no longer separable by a hyperplane.

```
plot(xgrid,col=c("red","blue")[as.numeric(ygrid)],pch=20,cex=.2,
     main = "Cost = 1")
points(x,col=c("red", "blue")[data$Y],pch=19)
points(x[svm.model1$index,],pch=5,cex=2)
points(3.5, 1.5, pch = 19, col = "blue")
beta=drop(t(svm.model1$coefs)%*%x[svm.model1$index,])
beta0=svm.model1$rho
abline(beta0/beta[2],-beta[1]/beta[2])
abline((beta0-1)/beta[2],-beta[1]/beta[2],lty=2)
abline((beta0+1)/beta[2],-beta[1]/beta[2],lty=2)
abline(-2, 2, col = "purple")
```



## Book 7 (5 points)

7. In this problem, you will use support vector approaches in order to predict whether a given car gets high or low gas mileage based on the Auto data set.

```r
data('Auto')
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
```

(a) Create a binary variable that takes on a 1 for cars with gas mileage above the median, and a 0 for cars with gas mileage below the median.

```
Auto$abovemedmpg <- as.factor(ifelse((Auto$mpg > median(Auto$mpg)), 1, 0))
Autoset <- Auto[,c(2:8, 10)]
```

(b) Fit a support vector classifier to the data with various values of cost, in order to predict whether a car gets high or low gas mileage. Report the cross-validation errors associated with different values of this parameter. Comment on your results.

```
# Create Train and Test
set.seed(1)
train <- sample(nrow(Autoset),(nrow(Autoset) * .70), replace = FALSE)
Auto_train <- Autoset[train,]
Auto_test <- Autoset[-train,]
```

```
# SVM - Use Tune to run with different values of costs
  # 10-fold cv sampling method built into tune
tune_out_linear1 <- tune(svm, abovemedmpg ~ ., data = Auto_train, kernel = "linear", ranges = list(cost

tune_out_linear1$performances
```

```
##    cost error dispersion
## 1 1e-03 0.168     0.0616
## 2 1e-02 0.109     0.0512
## 3 1e-01 0.113     0.0721
## 4 1e+00 0.127     0.0753
## 5 5e+00 0.113     0.0642
## 6 1e+01 0.106     0.0599
```

```
best_auto_linear1 <- tune_out_linear1$best.model

summary(best_auto_linear1)
```

```
##
## Call:
## best.tune(method = svm, train.x = abovemedmpg ~ ., data = Auto_train,
##     ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##       gamma:  0.143
##
## Number of Support Vectors:  68
##
##  ( 34 34 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

The best performance is for a cost of 10 with an error of 0.106.

(c) Now repeat (b), this time using SVMs with radial and polynomial basis kernels, with different values of gamma and degree and cost. Comment on your results.

```
# Radial Kernel

tune_out_linear2 <- tune(svm, abovemedmpg ~ ., data = Auto_train, kernel = "radial", ranges = list(cost

tune_out_linear2$performances
```

```
##     cost gamma degree error dispersion
## 1  0.001  0.01      1 0.556     0.1315
## 2  0.010  0.01      1 0.556     0.1315
## 3  0.100  0.01      1 0.139     0.0494
## 4  0.001  0.10      1 0.541     0.1671
## 5  0.010  0.10      1 0.483     0.1792
## 6  0.100  0.10      1 0.103     0.0693
## 7  0.001  0.50      1 0.534     0.1864
## 8  0.010  0.50      1 0.534     0.1864
## 9  0.100  0.50      1 0.110     0.0699
## 10 0.001  0.01      2 0.556     0.1315
## 11 0.010  0.01      2 0.556     0.1315
## 12 0.100  0.01      2 0.139     0.0494
## 13 0.001  0.10      2 0.541     0.1671
## 14 0.010  0.10      2 0.483     0.1792
## 15 0.100  0.10      2 0.103     0.0693
## 16 0.001  0.50      2 0.534     0.1864
## 17 0.010  0.50      2 0.534     0.1864
## 18 0.100  0.50      2 0.110     0.0699
## 19 0.001  0.01      3 0.556     0.1315
## 20 0.010  0.01      3 0.556     0.1315
## 21 0.100  0.01      3 0.139     0.0494
## 22 0.001  0.10      3 0.541     0.1671
## 23 0.010  0.10      3 0.483     0.1792
## 24 0.100  0.10      3 0.103     0.0693
## 25 0.001  0.50      3 0.534     0.1864
## 26 0.010  0.50      3 0.534     0.1864
## 27 0.100  0.50      3 0.110     0.0699
```

```
best_auto_linear2 <- tune_out_linear2$best.model

summary(best_auto_linear2)
```

```
##
## Call:
## best.tune(method = svm, train.x = abovemedmpg ~ ., data = Auto_train,
##     ranges = list(cost = c(0.001, 0.01, 0.1), gamma = c(0.01,
##         0.1, 0.5), degree = c(1, 2, 3)), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.1
##       gamma:  0.1
##
## Number of Support Vectors:  148
##
```

```
##  ( 74 74 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

The best model uses a cost of 0.1 and a gamma of 0.1 and 171 support vectors. It does not appear as if increasing degrees have any impact on the error.

```
# Polynomial Kernel
tune_out_linear3 <- tune(svm, abovemedmpg ~ ., data = Auto_train, kernel = "polynomial", ranges = list(

tune_out_linear3$performances
```

```
##     cost gamma degree error dispersion
## 1  0.001  0.01      1 0.522     0.0884
## 2  0.010  0.01      1 0.522     0.0884
## 3  0.100  0.01      1 0.175     0.0820
## 4  0.001  0.10      1 0.522     0.0884
## 5  0.010  0.10      1 0.175     0.0820
## 6  0.100  0.10      1 0.110     0.0574
## 7  0.001  0.50      1 0.208     0.1527
## 8  0.010  0.50      1 0.128     0.0569
## 9  0.100  0.50      1 0.113     0.0507
## 10 0.001  0.01      2 0.540     0.0550
## 11 0.010  0.01      2 0.540     0.0550
## 12 0.100  0.01      2 0.540     0.0550
## 13 0.001  0.10      2 0.540     0.0550
## 14 0.010  0.10      2 0.540     0.0550
## 15 0.100  0.10      2 0.339     0.0929
## 16 0.001  0.50      2 0.518     0.0779
## 17 0.010  0.50      2 0.300     0.0842
## 18 0.100  0.50      2 0.227     0.0832
## 19 0.001  0.01      3 0.540     0.0550
## 20 0.010  0.01      3 0.540     0.0550
## 21 0.100  0.01      3 0.540     0.0550
## 22 0.001  0.10      3 0.540     0.0550
## 23 0.010  0.10      3 0.353     0.1037
## 24 0.100  0.10      3 0.263     0.0996
## 25 0.001  0.50      3 0.259     0.1062
## 26 0.010  0.50      3 0.124     0.0872
## 27 0.100  0.50      3 0.128     0.0703
```

```
best_auto_linear3 <- tune_out_linear3$best.model

summary(best_auto_linear3)
```

```
##
## Call:
## best.tune(method = svm, train.x = abovemedmpg ~ ., data = Auto_train,
##     ranges = list(cost = c(0.001, 0.01, 0.1), gamma = c(0.01,
##         0.1, 0.5), degree = c(1, 2, 3)), kernel = "polynomial")
##
##
```

```
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  0.1
##      degree:  1
##       gamma:  0.1
##      coef.0:  0
##
## Number of Support Vectors:  137
##
##  ( 68 69 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

The best performance for the polynomial kernel is for the model with a cost of 0.01, a gamma of 0.5, and a degree 1, with an error of 0.115. Unlike the model where we used the radial kernel, it appears that the degree had an impact.

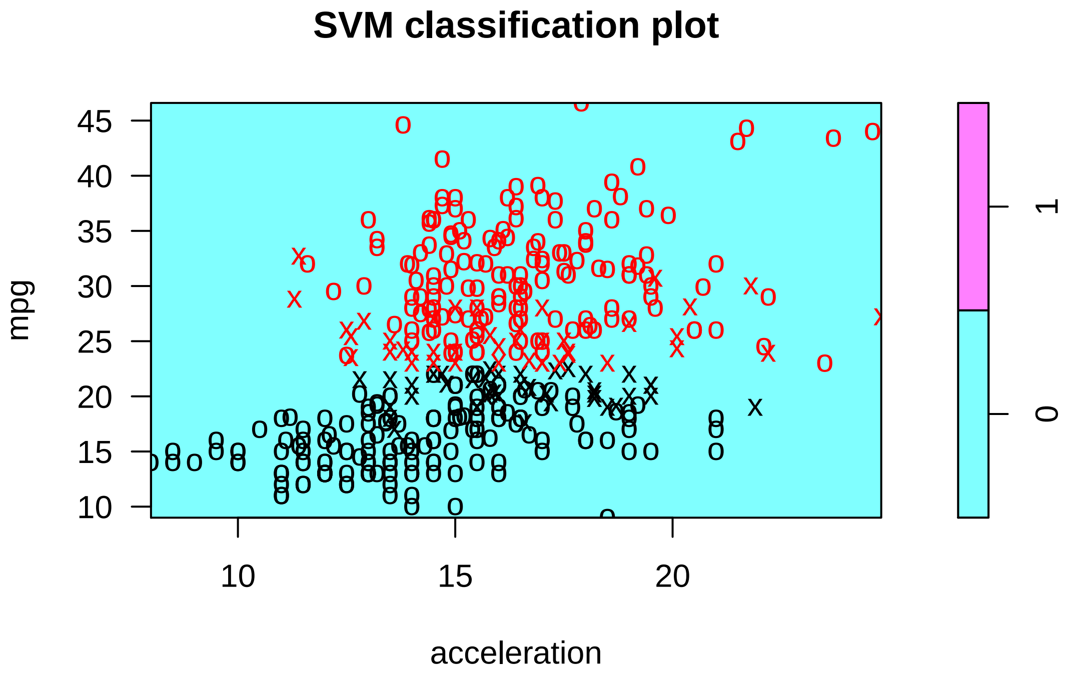
(d) Make some plots to back up your assertions in (b) and (c). Hint: In the lab, we used the plot() function for svm objects only in cases with p = 2. When p > 2, you can use the plot() function to create plots displaying pairs of variables at a time.

Essentially, instead of typing plot(svmfit , dat) where svmfit contains your fitted model and dat is a data frame containing your data, you can type plot(svmfit , dat , x1???x4) in order to plot just the first and fourth variables. However, you must replace x1 and x4 with the correct variable names. To find out more, type ?plot.svm.

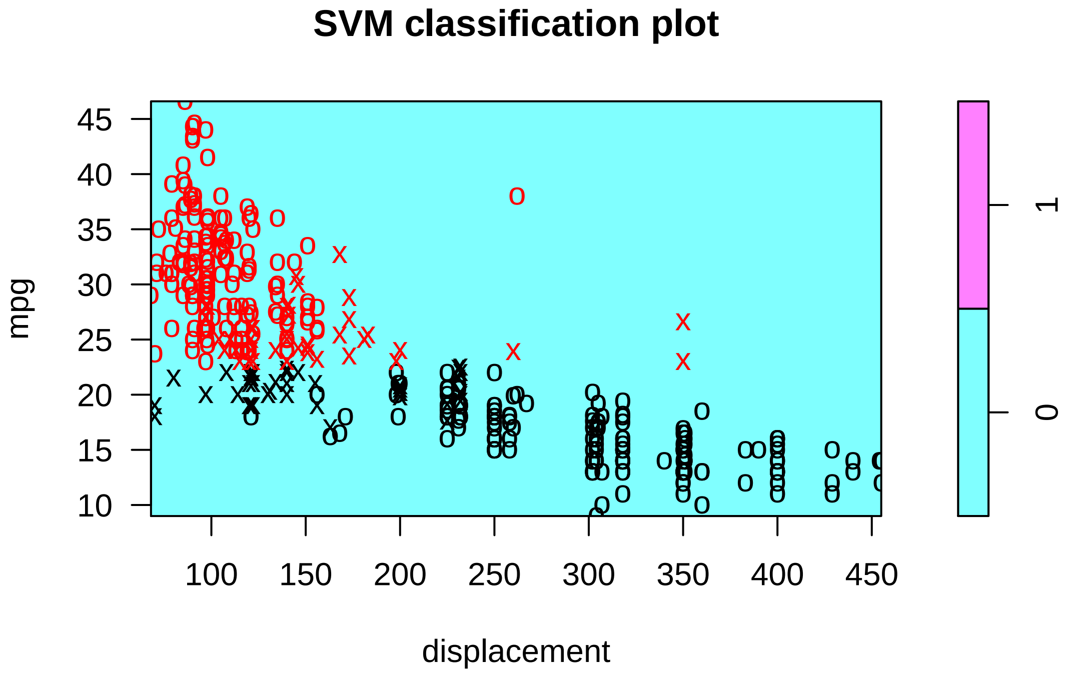
```
# Run optimal modles on full dataset
svm_linear <- svm(factor(abovemedmpg) ~ ., data = Autoset, kernel = "linear", cost = 10, gamma = 0.143)
svm_radial = svm(abovemedmpg ~ ., data = Autoset, kernel = "radial", cost = 0.1, gamma = 0.1)
svm_poly = svm(abovemedmpg ~ ., data = Autoset, kernel = "polynomial", cost = 0.01, degree = 1,
               gamma = 0.5, coef0=0)

# Add mpg back in for plotting
Autoset <- Auto[,c(2:8, 10)]
Autoset$mpg <- Auto$mpg

# For slicing the plot variables
mean_acceleration <- mean(Autoset$acceleration)
mean_weight <- mean(Autoset$weight)
mean_horsepower <- mean(Autoset$horsepower)
mean_displacement <- mean(Autoset$displacement)
mean_mpg <- mean(Autoset$mpg)
```

## Linear Plots

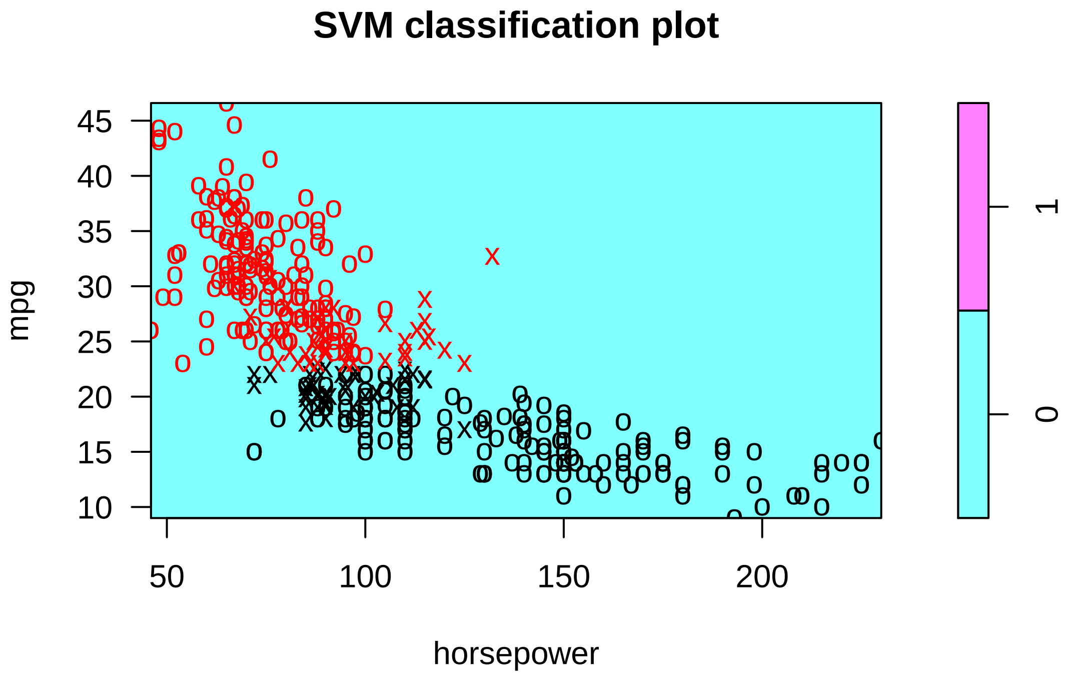
```
par(mfrow=c(2,2))
plot(svm_linear, Autoset, mpg ~ acceleration, slice = list(displacement = mean_displacement, horsepower
```



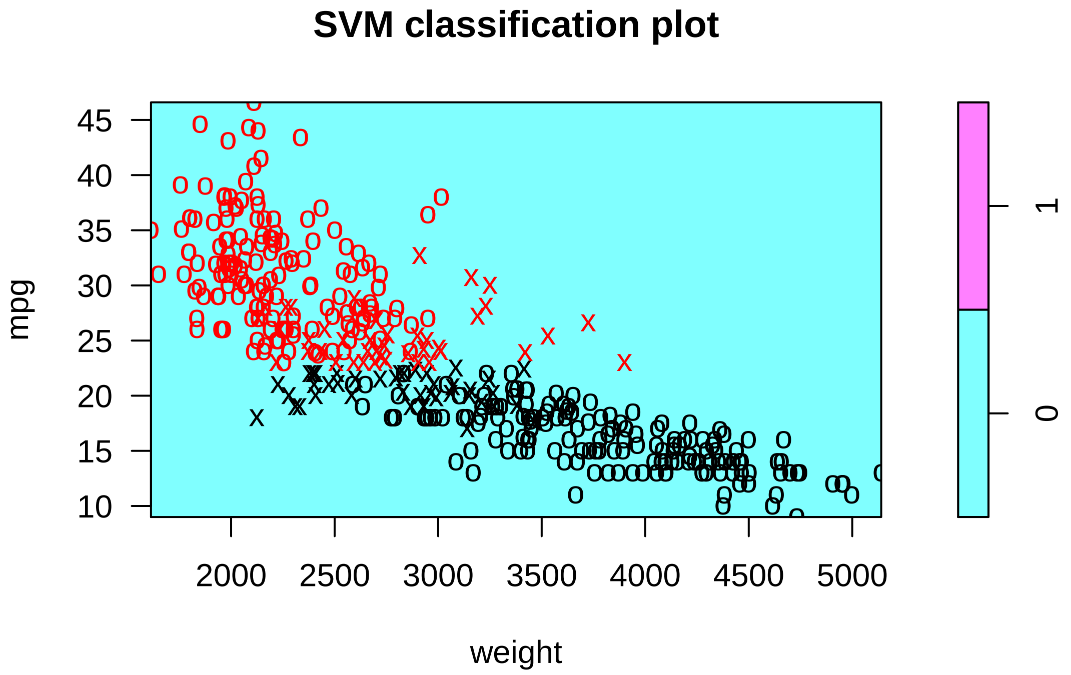
```
plot(svm_linear, Autoset, mpg ~ displacement,  slice = list(acceleration = mean_acceleration, horsepowe
```



```
plot(svm_linear, Autoset, mpg ~ horsepower, slice = list(acceleration = mean_acceleration, displacement
```

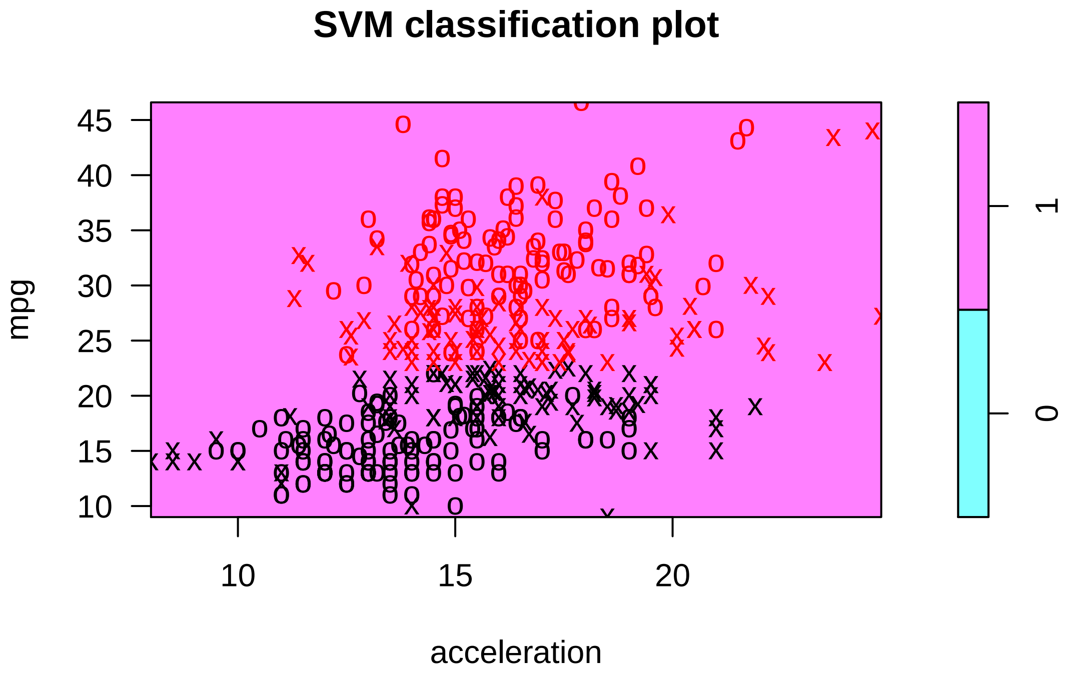


```
plot(svm_linear, Autoset, mpg ~ weight, slice = list(acceleration = mean_acceleration, displacement = me
```

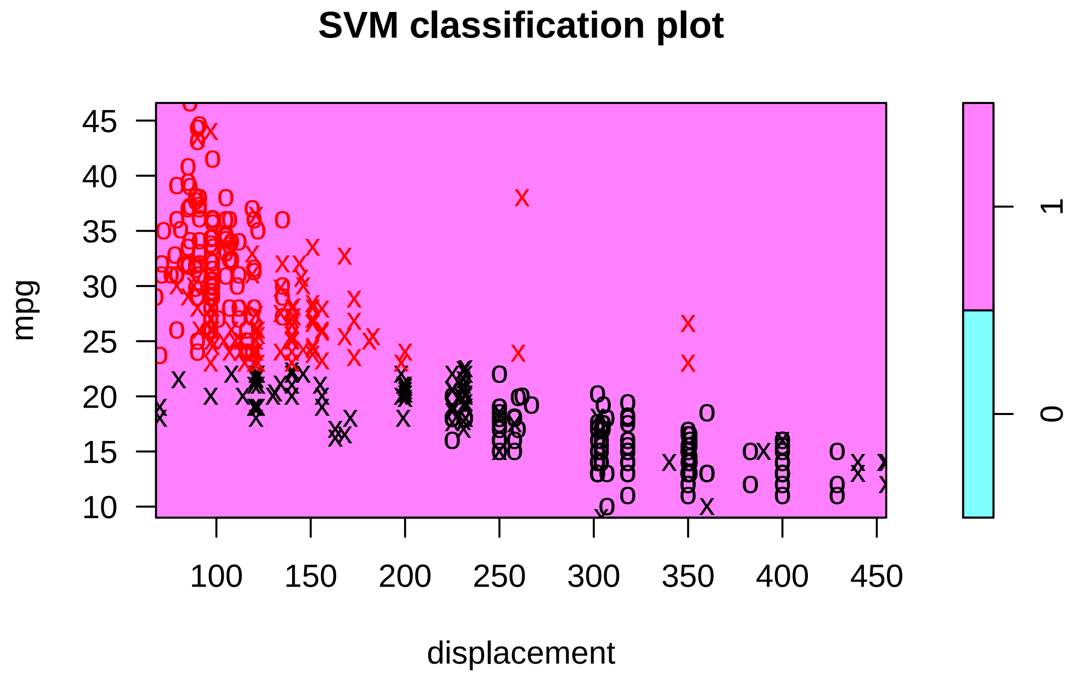


## Radial Plots

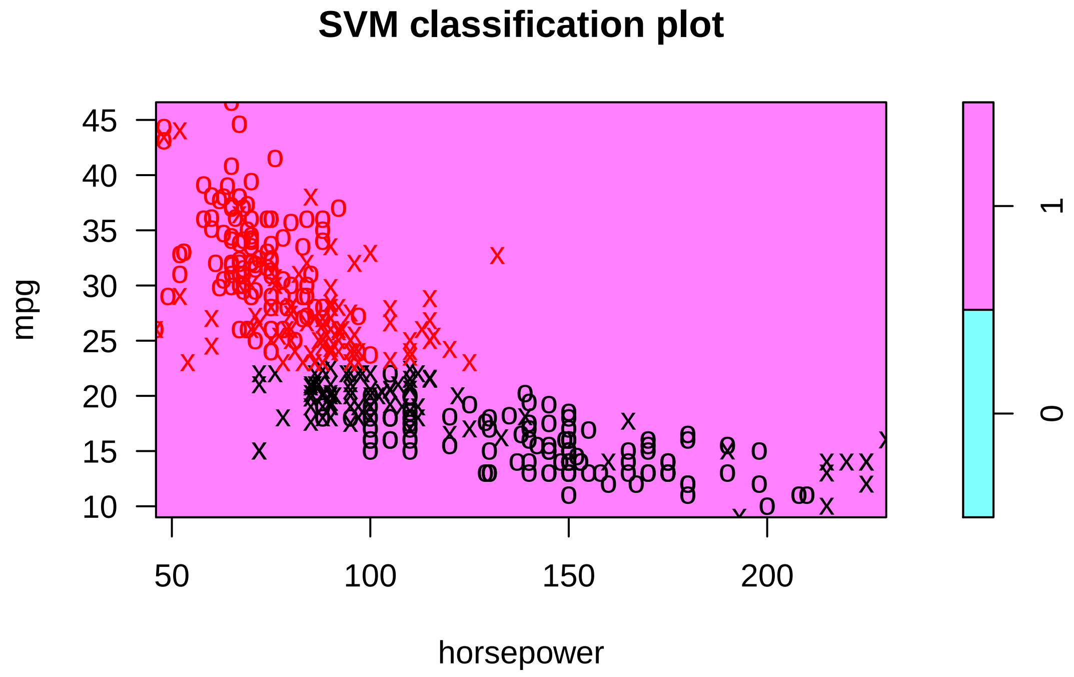
```
par(mfrow=c(2,2))
plot(svm_radial, Autoset, mpg ~ acceleration, main = "Radial: mpg~acceleration",
     slice = list(displacement = mean_displacement, horsepower = mean_horsepower))
```



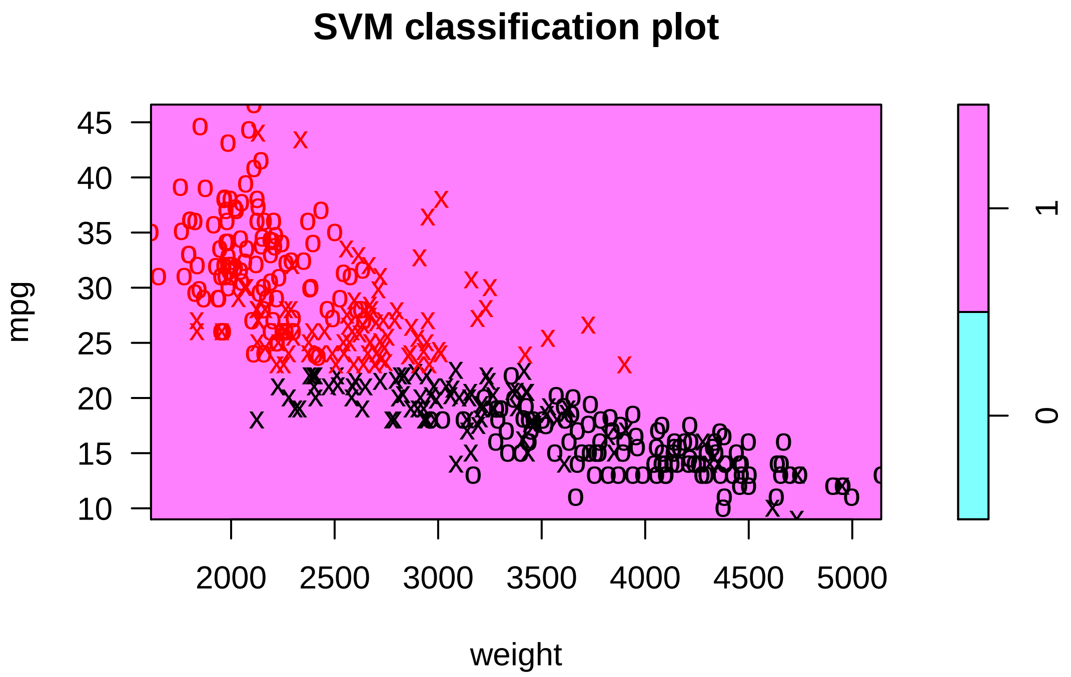
```
plot(svm_radial, Autoset, mpg ~ displacement, main = "Radial: mpg~displacement",
     slice = list(acceleration = mean_acceleration, horsepower = mean_horsepower))
```



```
plot(svm_radial, Autoset, mpg ~ horsepower, main = "Radial: mpg~horsepower",
     slice = list(acceleration = mean_acceleration, displacement = mean_displacement))
```

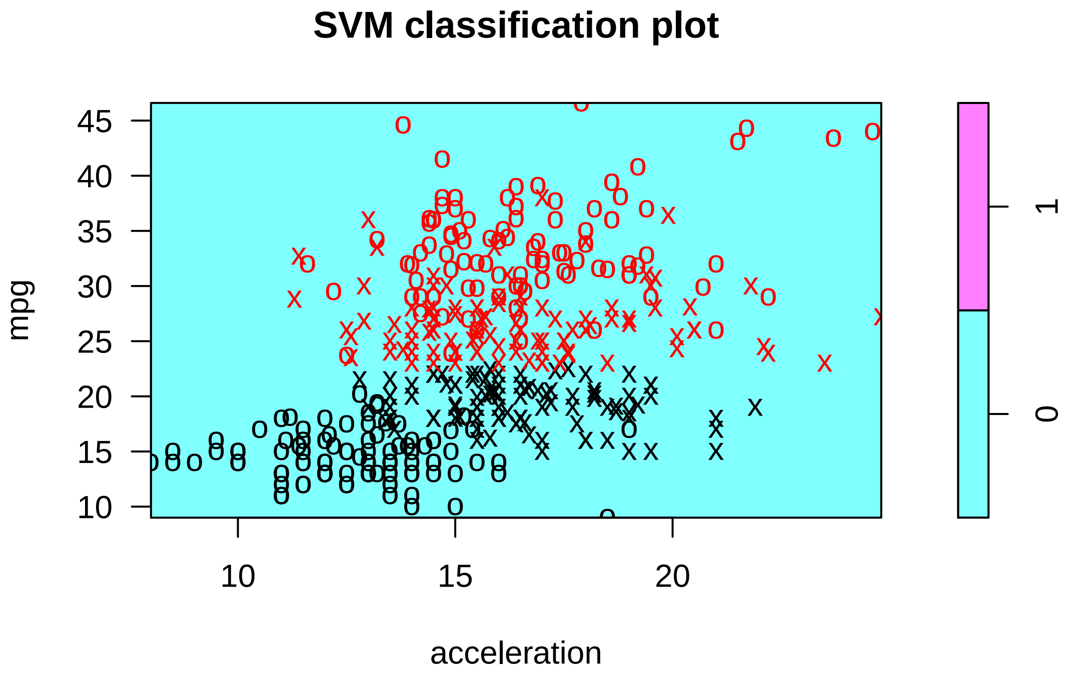


```
plot(svm_radial, Autoset, mpg ~ weight, main = "Radial: mpg~weight",
     slice = list(acceleration = mean_acceleration, displacement = mean_displacement))
```

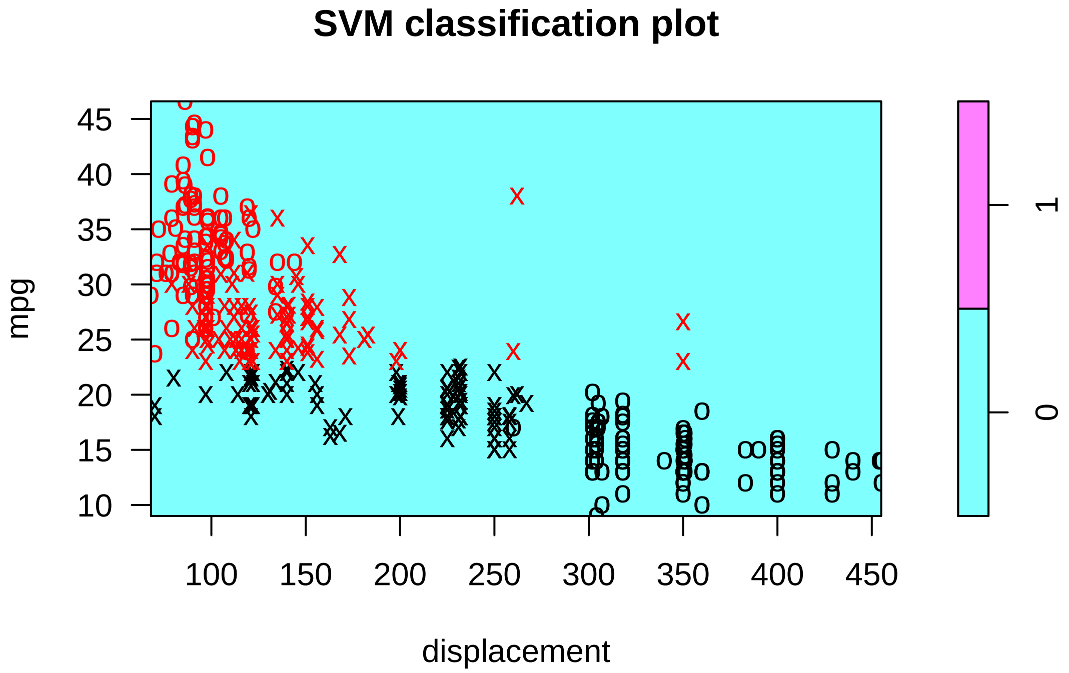


## Polynomial Plots

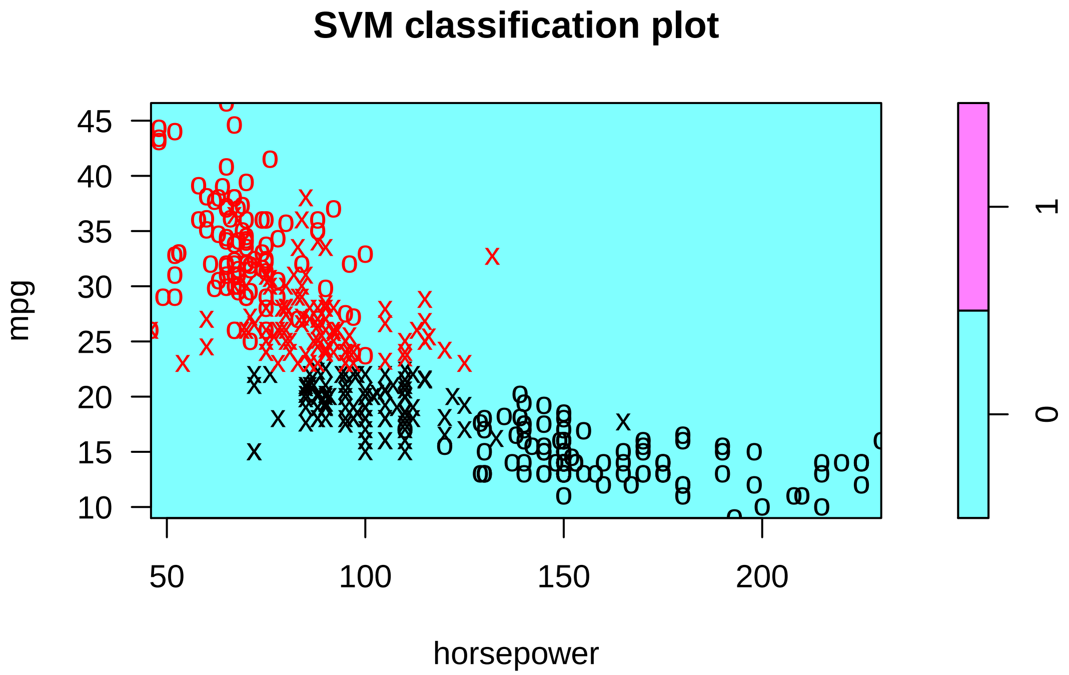
```
par(mfrow=c(2,2))
plot(svm_poly, Autoset, mpg ~ acceleration, slice = list(displacement = mean_displacement, horsepower =
```



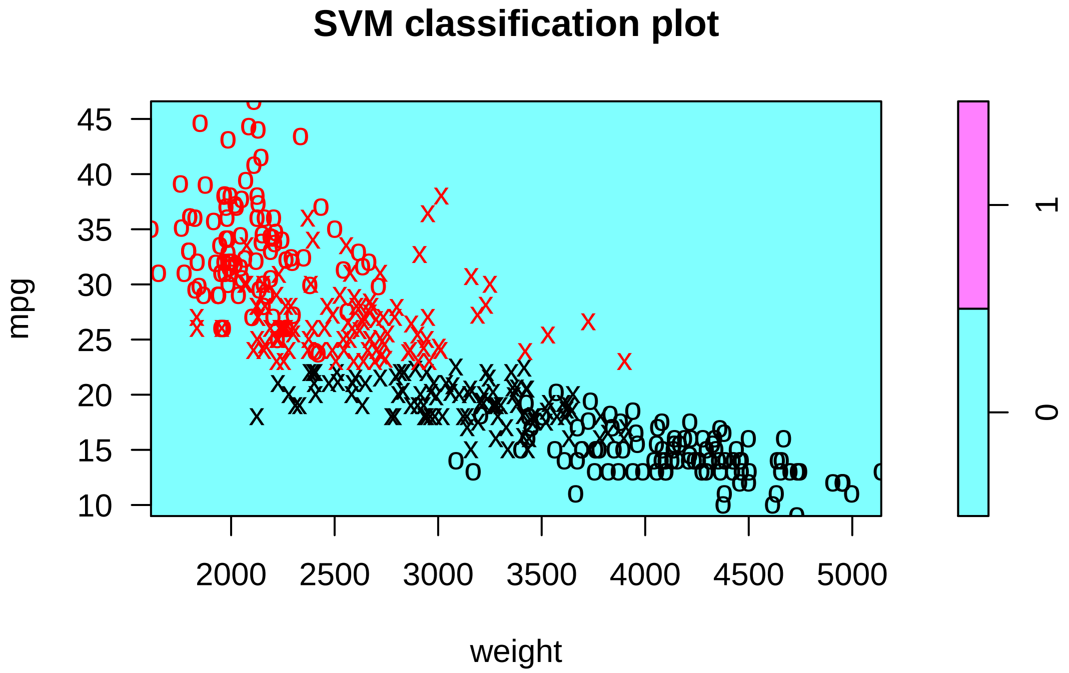
```
plot(svm_poly, Autoset, mpg ~ displacement, slice = list(acceleration = mean_acceleration, horsepower =
```



```
plot(svm_poly, Autoset, mpg ~ horsepower, slice = list(acceleration = mean_acceleration, displacement =
```



```
plot(svm_poly, Autoset, mpg ~ weight, slice = list(acceleration = mean_acceleration, displacement = mean
```



# Book 8 (5 points)

8. This problem involves the OJ data set which is part of the ISLR package.

(a) Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations.

```
data('OJ')
head(OJ)
```

```
##   Purchase WeekofPurchase StoreID PriceCH PriceMM DiscCH DiscMM SpecialCH
## 1       CH            237       1    1.75    1.99   0.00    0.0         0
## 2       CH            239       1    1.75    1.99   0.00    0.3         0
## 3       CH            245       1    1.86    2.09   0.17    0.0         0
## 4       MM            227       1    1.69    1.69   0.00    0.0         0
## 5       CH            228       7    1.69    1.69   0.00    0.0         0
## 6       CH            230       7    1.69    1.99   0.00    0.0         0
##   SpecialMM LoyalCH SalePriceMM SalePriceCH PriceDiff Store7 PctDiscMM
## 1         0   0.500        1.99        1.75      0.24     No     0.000
## 2         1   0.600        1.69        1.75     -0.06     No     0.151
## 3         0   0.680        2.09        1.69      0.40     No     0.000
## 4         0   0.400        1.69        1.69      0.00     No     0.000
## 5         0   0.957        1.69        1.69      0.00    Yes     0.000
## 6         1   0.965        1.99        1.69      0.30    Yes     0.000
##   PctDiscCH ListPriceDiff STORE
## 1    0.0000          0.24     1
## 2    0.0000          0.24     1
## 3    0.0914          0.23     1
## 4    0.0000          0.00     1
## 5    0.0000          0.00     0
## 6    0.0000          0.30     0
```

```
names(OJ)
```

```
##  [1] "Purchase"       "WeekofPurchase" "StoreID"        "PriceCH"
##  [5] "PriceMM"        "DiscCH"         "DiscMM"         "SpecialCH"
##  [9] "SpecialMM"      "LoyalCH"        "SalePriceMM"    "SalePriceCH"
## [13] "PriceDiff"      "Store7"         "PctDiscMM"      "PctDiscCH"
## [17] "ListPriceDiff"  "STORE"
```

```
# Factorize Purchase Variable
OJ$Purchase <- as.factor(OJ$Purchase)
# Create Train and Test
set.seed(1)
train <- sample(nrow(OJ), 800, replace = FALSE)
OJ_train <- OJ[train,]
OJ_test <- OJ[-train,]
```

(b) Fit a support vector classifier to the training data using cost=0.01, with Purchase as the response and the other variables as predictors. Use the summary() function to produce summary statistics, and describe the results obtained.

```
svm.model1 = svm(Purchase ~., data = OJ_train, kernel="linear",cost=0.01,scale=FALSE)
summary(svm.model1)
```

```
##
## Call:
```

```
## svm(formula = Purchase ~ ., data = OJ_train, kernel = "linear",
##     cost = 0.01, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##       gamma:  0.0556
##
## Number of Support Vectors:  615
##
##  ( 306 309 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

According to the summary statistics, when we use a linear kernal, we end up with 615 support vectors in the process of classifying the two purchase groups.

(c) What are the training and test error rates?

```
fitted_train <- attributes(predict(svm.model1, OJ_train, decision.values=TRUE))$decision.values
auc(OJ_train$Purchase, fitted_train)
```

```
## Area under the curve: 0.866
```

```
fitted_test <- attributes(predict(svm.model1, OJ_test, decision.values=TRUE))$decision.values
auc(OJ_test$Purchase, fitted_test)
```

```
## Area under the curve: 0.855
```

(d) Use the tune() function to select an optimal cost. Consider values in the range 0.01 to 10.

```
tune_out_linear <- tune(svm, Purchase ~ ., data = OJ_train, kernel = "linear", ranges = list(cost = c(0

tune_out_linear$performances
```

```
##    cost error dispersion
## 1  0.01 0.166     0.0514
## 2  0.10 0.163     0.0489
## 3  1.00 0.169     0.0472
## 4  5.00 0.168     0.0504
## 5 10.00 0.165     0.0499
```

```
best_oj_linear <- tune_out_linear$best.model

summary(best_oj_linear)
```

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = OJ_train,
##     ranges = list(cost = c(0.01, 0.1, 1, 5, 10)), kernel = "linear")
##
##
```

```
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.1
##       gamma:  0.0556
##
## Number of Support Vectors:  343
##
##  ( 171 172 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

The best model is the model with a cost of 0.1.

(e) Compute the training and test error rates using this new value for cost.

```
fitted_train <- attributes(predict(best_oj_linear, OJ_train, decision.values=TRUE))$decision.values
auc(OJ_train$Purchase, fitted_train)
```

```
## Area under the curve: 0.906
```

```
fitted_test <- attributes(predict(best_oj_linear, OJ_test, decision.values=TRUE))$decision.values
auc(OJ_test$Purchase, fitted_test)
```

```
## Area under the curve: 0.879
```

(f) Repeat parts (b) through (e) using a support vector machine with a radial kernel. Use the default value for gamma.

```
svm.model2 = svm(Purchase ~., data = OJ_train, kernel="radial", cost=0.01,scale=FALSE)
summary(svm.model2)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ_train, kernel = "radial",
##     cost = 0.01, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.01
##       gamma:  0.0556
##
## Number of Support Vectors:  628
##
##  ( 306 322 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

According to the summary statistics, when we use a radial kernal, we end uo with 628 support vectors in the process of classifying the two purchase groups.

```
train_error_radial <- attributes(predict(svm.model2, OJ_train, decision.values=TRUE))$decision.values
auc(OJ_train$Purchase, train_error_radial)
```

```
## Area under the curve: 0.799
```

```
test_error_radial <- attributes(predict(svm.model2, OJ_test, decision.values=TRUE))$decision.values
auc(OJ_test$Purchase, test_error_radial)
```

```
## Area under the curve: 0.762
```

```
tune_out_radial <- tune(svm, Purchase ~ ., data = OJ_train, kernel = "radial", ranges = list(cost = c(0

tune_out_radial$performances
```

```
##    cost error dispersion
## 1  0.01 0.383     0.0680
## 2  0.10 0.182     0.0392
## 3  1.00 0.176     0.0498
## 4  5.00 0.171     0.0396
## 5 10.00 0.182     0.0345
```

```
best_oj_radial <- tune_out_radial$best.model

summary(best_oj_radial)
```

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = OJ_train,
##     ranges = list(cost = c(0.01, 0.1, 1, 5, 10)), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  5
##       gamma:  0.0556
##
## Number of Support Vectors:  331
##
##  ( 163 168 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

The best model is the one with a cost of 10.

```
best_train_error_radial <- attributes(predict(best_oj_radial, OJ_train, decision.values=TRUE))$decision
auc(OJ_train$Purchase, best_train_error_radial)
```

```
## Area under the curve: 0.927
```

```
best_test_error_radial <- attributes(predict(best_oj_radial, OJ_test, decision.values=TRUE))$decision.va
auc(OJ_test$Purchase, best_test_error_radial)
```

```
## Area under the curve: 0.874
```

(g) Repeat parts (b) through (e) using a support vector machine with a polynomial kernel. Set degree=2.

```
svm.model3 = svm(Purchase ~., data = OJ_train, kernel="polynomial", cost=0.01, degree = 2, scale=FALSE)
summary(svm.model3)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ_train, kernel = "polynomial",
##     cost = 0.01, degree = 2, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  0.01
##      degree:  2
##       gamma:  0.0556
##      coef.0:  0
##
## Number of Support Vectors:  331
##
##  ( 165 166 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

According to the summary statistics, when we use a polynomial kernal, we end uo with 331 support vectors in the process of classifying the two purchase groups.

```
train_error_polynomial <- attributes(predict(svm.model3, OJ_train, decision.values=TRUE))$decision.valu
auc(OJ_train$Purchase, train_error_polynomial)
```

```
## Area under the curve: 0.906
```

```
test_error_polynomial <- attributes(predict(svm.model3, OJ_test, decision.values=TRUE))$decision.values
auc(OJ_test$Purchase, test_error_polynomial)
```

```
## Area under the curve: 0.881
```

```
tune_out_polynomial <- tune(svm, Purchase ~ ., data = OJ_train, kernel = "polynomial", degree=2, ranges

tune_out_polynomial$performances
```

```
##    cost error dispersion
## 1  0.01 0.383     0.0387
## 2  0.10 0.330     0.0426
## 3  1.00 0.198     0.0316
## 4  5.00 0.174     0.0402
## 5 10.00 0.171     0.0373
```

```
best_oj_polynomial <- tune_out_polynomial$best.model
```

```
summary(best_oj_polynomial)
```

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = OJ_train,
##     ranges = list(cost = c(0.01, 0.1, 1, 5, 10)), kernel = "polynomial",
##     degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  10
##      degree:  2
##       gamma:  0.0556
##      coef.0:  0
##
## Number of Support Vectors:  342
##
##  ( 170 172 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

The best model is the model with a cost of 10.

```
best_train_error_polynomial <- attributes(predict(best_oj_polynomial, OJ_train, decision.values=TRUE))$d
auc(OJ_train$Purchase, best_train_error_polynomial)
```

```
## Area under the curve: 0.912
```

```
best_test_error_polynomial <- attributes(predict(best_oj_polynomial, OJ_test, decision.values=TRUE))$dec
auc(OJ_test$Purchase, best_test_error_polynomial)
```

```
## Area under the curve: 0.887
```

(h) Overall, which approach seems to give the best results on this data?

The error for the train and test for the linear, radial, and polynomial with the optimal cost are 0.906 and 0.879, 0.931 and 0.868, and 0.912 and 0.887, respectively. Given that we are using auc for the error rate, the best results appear to come from the polynomial model. While there may bne some overfitting given the discrepancy between the train and test error, the test error is the highest on the polynomial model.

## Extra 63 (5 points)

In this problem, we use the BreastCancer data, which comes as part of the package mlbench. Install the package and read the description of the data.

We want to predict the Class of an observation (benign or malignant).

```
data('BreastCancer')
head(BreastCancer)
```

```
##        Id Cl.thickness Cell.size Cell.shape Marg.adhesion Epith.c.size
## 1 1000025            5         1          1             1            2
## 2 1002945            5         4          4             5            7
## 3 1015425            3         1          1             1            2
## 4 1016277            6         8          8             1            3
## 5 1017023            4         1          1             3            2
## 6 1017122            8        10         10             8            7
##   Bare.nuclei Bl.cromatin Normal.nucleoli Mitoses     Class
## 1           1           3               1       1    benign
## 2          10           3               2       1    benign
## 3           2           3               1       1    benign
## 4           4           3               7       1    benign
## 5           1           3               1       1    benign
## 6          10           9               7       1 malignant
```

```
BreastCancer$Id <- NULL

# Create Train and Test
set.seed(1)
train <- sample(nrow(BreastCancer),(nrow(BreastCancer) * .70), replace = FALSE)
BC_train <- BreastCancer[train,]
BC_test <- BreastCancer[-train,]
```

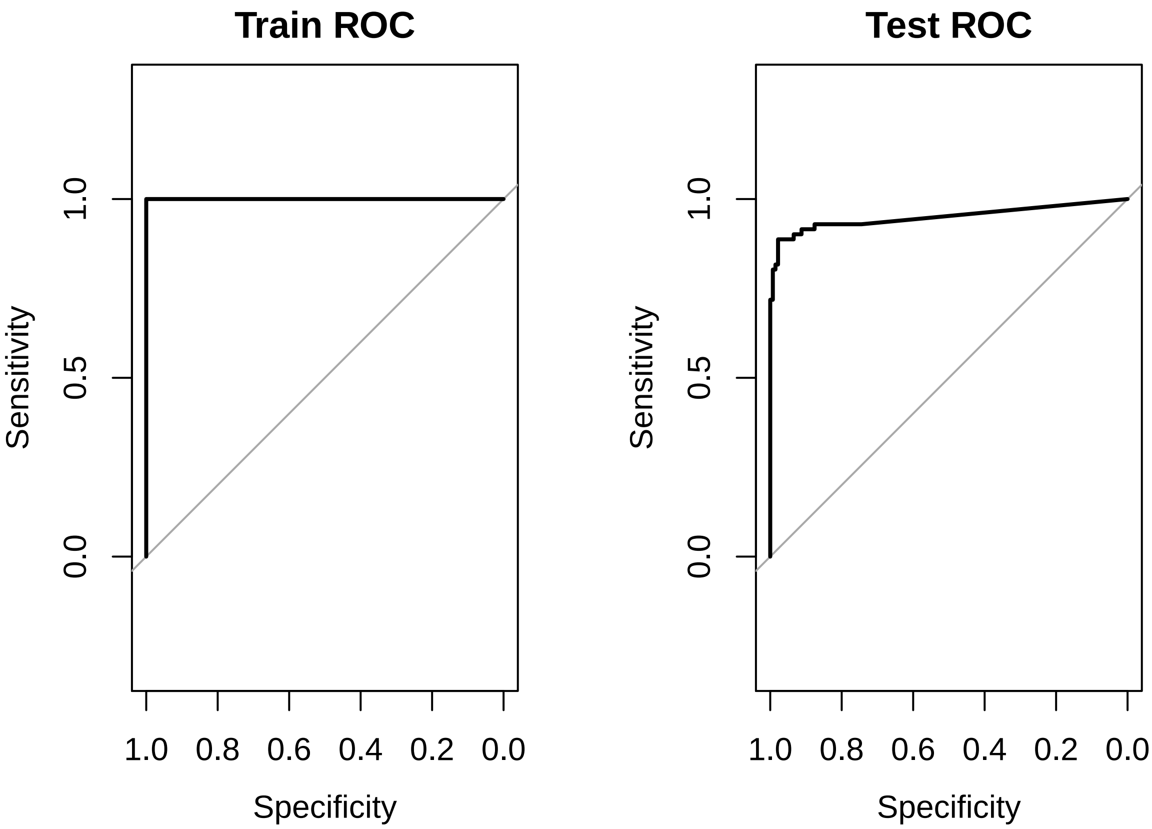
a) Fit a logistic model to the data. Plot the ROC curve.

```
fit <- glm(Class ~ ., data = BC_train, family=binomial)
```

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

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

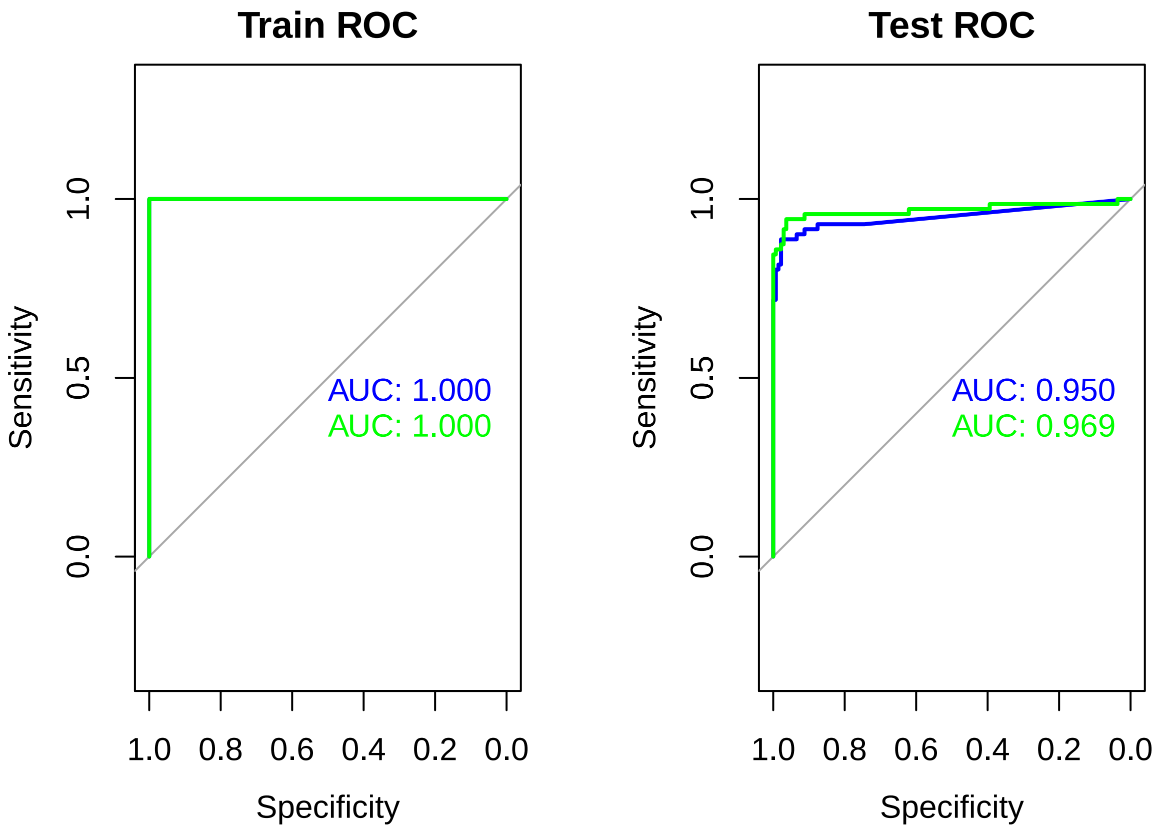
```
BC_train$pred <- predict(fit, BC_train, type = "response")

r1 <- roc(as.numeric(BC_train$Class), BC_train$pred)

BC_test$pred <- predict(fit, newdata = BC_test, type = "response")

r2 <- roc(as.numeric(BC_test$Class), BC_test$pred)

par(mfrow=c(1,2))
plot(r1, main = "Train ROC")
plot(r2, main = "Test ROC")
```



```
# Is 1 what we expected for AUC here, seems ods
```

b) Use a support vector classifier (linear kernels) to classify. Plot the ROC curve in the plot you made in part a).

```
svm.model <- svm(Class ~ ., data = BC_train, kernel="linear", scale=FALSE)

BC_train$pred <- attributes(predict(svm.model, BC_train, decision.values=TRUE, na.action = na.exclude))$

r3 <- roc(as.numeric(BC_train$Class), BC_train$pred)

BC_test$pred <- attributes(predict(svm.model, BC_test, decision.values=TRUE, na.action = na.exclude))$de

r4 <- roc(as.numeric(BC_test$Class), BC_test$pred)

par(mfrow=c(1,2))
roc_train <- plot(r1, print.auc = TRUE, col = "blue", main = "Train ROC")
#And for the second ROC curve you need to change the y position of the AUC and use add the plot the two
roc_train <- plot(r3, print.auc = TRUE, col = "green", print.auc.y = .4, add = TRUE)

roc_test <- plot(r2, print.auc = TRUE, col = "blue", main = "Test ROC")
#And for the second ROC curve you need to change the y position of the AUC and use add the plot the two
roc_test <- plot(r4, print.auc = TRUE, col = "green", print.auc.y = .4, add = TRUE)
```



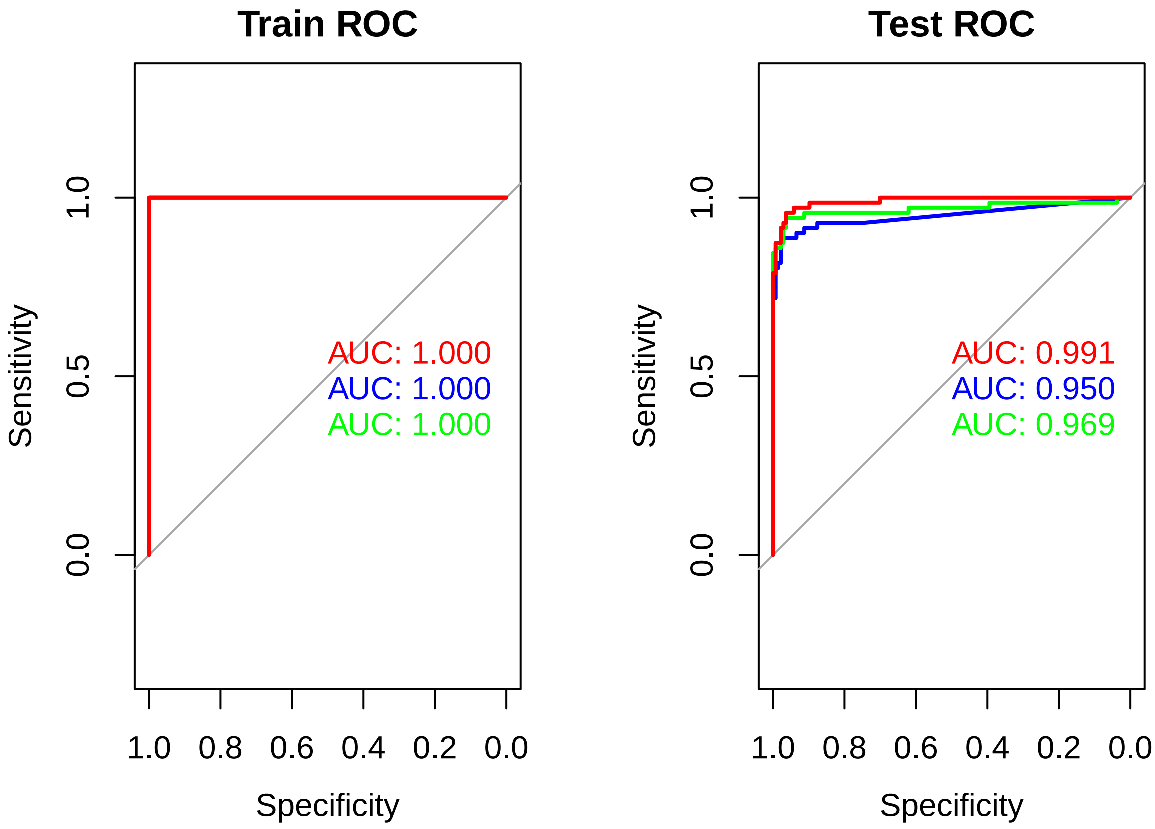
c) Use SVMs with polynomial kernels for the same classification task, with several different degrees and oterwise using the same (hyper)parameters. Plot the ROC curves.

```
tune_out_polynomial <- tune(svm, Class ~ ., data = BC_train, kernel = "polynomial", ranges = list(degre

tune_out_polynomial$performances
```

```
##   degree error dispersion
## 1      1 0.000     0.0000
## 2      2 0.354     0.0509
## 3      3 0.354     0.0509
## 4      4 0.354     0.0509
```

```
#
best_bc_polynomial <- tune_out_polynomial$best.model

BC_train$pred_polynomial <- attributes(predict(best_bc_polynomial, BC_train, decision.values=TRUE, na.a

BC_train$pred <- attributes(predict(svm.model, BC_train, decision.values=TRUE, na.action = na.exclude))$

r5 <- roc(as.numeric(BC_train$Class), BC_train$pred_polynomial)

BC_test$pred_polynomial <- attributes(predict(best_bc_polynomial, BC_test, decision.values=TRUE, na.acti

r6 <- roc(as.numeric(BC_test$Class), BC_test$pred_polynomial)

par(mfrow=c(1,2))
```

```
roc_train <- plot(r1, print.auc = TRUE, col = "blue", main = "Train ROC")
#And for the second ROC curve you need to change the y position of the AUC and use add the plot the two
roc_train <- plot(r3, print.auc = TRUE, col = "green", print.auc.y = .4, add = TRUE)
roc_train <- plot(r5, print.auc = TRUE, col = "red", print.auc.y = .6, add = TRUE)


roc_test <- plot(r2, print.auc = TRUE, col = "blue", main = "Test ROC")
#And for the second ROC curve you need to change the y position of the AUC and use add the plot the two
roc_test <- plot(r4, print.auc = TRUE, col = "green", print.auc.y = .4, add = TRUE)
roc_test <- plot(r6, print.auc = TRUE, col = "red", print.auc.y = .6, add = TRUE)
```



d) Compare the results of a), b), and c). Are the support vector classifier and logistic regression comparable?

On the train dataset, the SVC linear and polynomial and the logistic regression are identical. They are also very similar on the test ROC, though the polynomial SVM seems to perform the best with an AUC of 0.988.

## Extra 66 (5)

This problem uses the MNIST image classification data, available as mnist_all.RData that were used earlier. We want to distinguish between 3 and 8. Extract the relevant training and test data and place them in suitable data frames. Remove all variables (pixels) with zero variance from the training data and remove these also from the test data.

```
mnist <-load('mnist_all.RData')
```

```r
mnist_train <- data.frame(train$x, train$y)
mnist_train <- mnist_train[mnist_train$train.y == 3 | mnist_train$train.y == 8,]
# Rremove zero variance variables (161 in total)
zerovalist <- which(apply(mnist_train, 2, var) == 0)
mnist_train <- mnist_train[ - zerovalist]

mnist_test <- data.frame(test$x, test$y)
mnist_test <- mnist_test[mnist_test$test.y == 3 | mnist_test$test.y == 8,]
mnist_test <- mnist_test[ - zerovalist]
```

a) Fit a random forest model to the training data. Experiment with hyperparameters until you get a very good classification on the training data. Then evaluate the model on the test data and make a confusion matrix.

```r
rf.mnist <- randomForest(factor(train.y) ~ .,data= mnist_train, mtry=100, ntree=25, importance =TRUE)

yhat.rf = predict(rf.mnist ,newdata= mnist_test)

table(mnist_test$test.y, yhat.rf)
```

```
##    yhat.rf
##       3   8
##   3 995  15
##   8   6 968
```

```r
confusionMatrix(yhat.rf, mnist_test$test.y)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   8
##          3 995   6
##          8  15 968
##
##                Accuracy : 0.989
##                  95% CI : (0.984, 0.993)
##     No Information Rate : 0.509
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.979
##  Mcnemar's Test P-Value : 0.0809
##
##             Sensitivity : 0.985
##             Specificity : 0.994
##          Pos Pred Value : 0.994
##          Neg Pred Value : 0.985
##              Prevalence : 0.509
##          Detection Rate : 0.502
##    Detection Prevalence : 0.505
##       Balanced Accuracy : 0.989
##
##        'Positive' Class : 3
##
```

b) Can you get a similar or better performance on the training data if you use an SVM classifier with kernel = “radial”? Experiment with hyperparameters. Do not report all trials, only the best result.

Then evaluate the model on the test data and make a confusion matrix.

```
# Downsample to decrease svm runtime

downsamp <- seq(1, ncol(mnist_train), 4)
mnist_train2 <- mnist_train[ downsamp]
mnist_train2$train.y <- mnist_train$train.y
mnist_test2 <- mnist_test[ downsamp]
mnist_test2$test.y <- mnist_test$test.y
```

```
# Tune function took far too long with this many, so just did default svm here
svm_radial <- svm(factor(train.y) ~ ., data= mnist_train2, kernel = "radial", probability = TRUE)
svm_radial
```

```
##
## Call:
## svm(formula = factor(train.y) ~ ., data = mnist_train2, kernel = "radial",
##     probability = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  0.00641
##
## Number of Support Vectors:  1918
```

```
mnist_test2$Pred_Class <- predict(svm_radial, mnist_test2, probability = TRUE)

confusionMatrix(mnist_test2$test.y, mnist_test2$Pred_Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   8
##          3 994  16
##          8  22 952
##
##                Accuracy : 0.981
##                  95% CI : (0.974, 0.986)
##     No Information Rate : 0.512
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.962
##  Mcnemar's Test P-Value : 0.417
##
##             Sensitivity : 0.978
##             Specificity : 0.983
##          Pos Pred Value : 0.984
##          Neg Pred Value : 0.977
##              Prevalence : 0.512
##          Detection Rate : 0.501
##    Detection Prevalence : 0.509
##       Balanced Accuracy : 0.981
##
```

```
##        'Positive' Class : 3
##
```

c) Compare the results of a) and b). Is there a clear difference in performance? Do any of these methods tend to overfit? Do their runtimes differ substantially?

The runtimes differ substantially. The svm, especially when we use tune to test out different parameters, takes significantly longer than the logistic regressions. In terms of performance, the random forest model shows an accuracy rate of 0.989 compared to the accuracy rate of 0.981 for the radial model with a cost of 1 and a gamma of 0.00641, meaning the random forest model performs slightly better, though the difference in performance is not substantial.