Assignment 9 – support vector machines

Math 154, Computational Statistics
Fall 2015, Maria Martinez

Due: Tuesday, November 17, 2015, noon

Not related to HW Score:

Total hours spent on assignment: __________

Number of different "sittings" to finish assignment: _________

Summary

Support Vector Machines: one more classification technique is introduced. Note that SVMs can be extremely powerful, but they require (1) a binary response variable, and (2) numerical predictor variables (note: categorical variables can be forced to be numerical using binary representations of the different levels).

Requisites

Read relevant sections of *An Introduction to Statistical Learning*, `http://www-bcf.usc.edu/~gareth/ISL/`: chapter 9 (no ROC).

Note that the lab in section 9.6 walks through much of the needed svm code (no ROC).

Assignment

1. 9.7.2) We have seen that in $p = 2$ dimensions, a linear decision boundary takes the form $\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$. We now investigate a non-linear decision boundary.

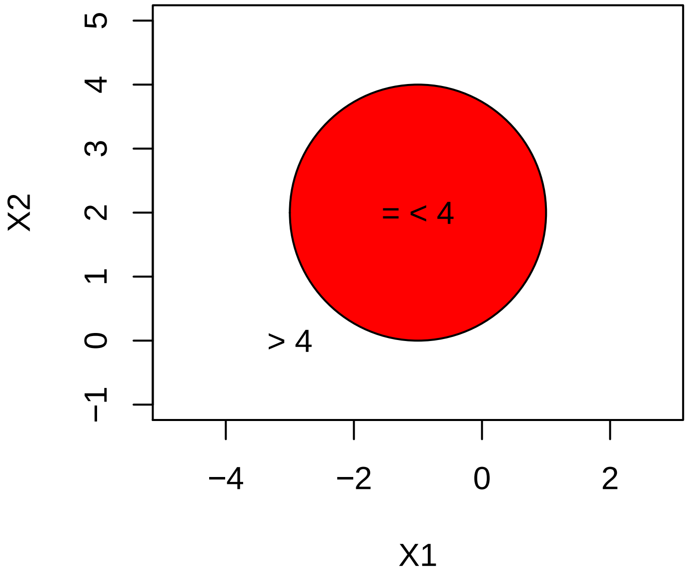
   (a) Sketch the curve $(1 + X_1)^2 + (2 - X_2)^2 = 4$.

   (b) On your sketch, indicate the set of points for which $(1 + X_1)^2 + (2 - X_2)^2 > 4$, as well as the set of points for which $(1 + X_1)^2 + (2 - X_2)^2 \leq 4$.

```r
#require(plotrix)
plot.new()
frame()
plot(-5:5, -1:5,type = "n",
     xlab = "", ylab = "", main = "Test draw.circle")
```

```
draw.circle(1,2,2,lty=1,lwd=1)

text(-1, 2, "= < 4")
text(-3, 0, "> 4")
```



(c) Suppose that a classifier assigns an observation to the blue class if $(1+X_1)^2+(2-X_2)^2 > 4$, and to the red class otherwise. To what class are the following observations classified?

i. (0, 0) ⇒ BLUE

ii. (1, 1) ⇒ RED

iii. (2, 2) ⇒ BLUE

iv. (3, 8) ⇒ BLUE

(d) Argue that while the decision boundary in (c) is not linear in terms of X_1 and X_2, it is linear in terms of X_1, X_1^2, X_2, and X_2^2.
When you expand the equation $(1 + X_1)^2 + (2 - X_2)^2 > 4$ you get:

$$1 + 2X_1 + X_1^2 + 4 - 4X_2 + X_2^2 > 4$$

$$\Rightarrow 5 + 2X_1 + X_1^2 - 4X_2 + X_2^2 > 4$$

Now we get a linear equation.

2. 9.7.3) Here we explore the maximal margin classifier on a toy data set.

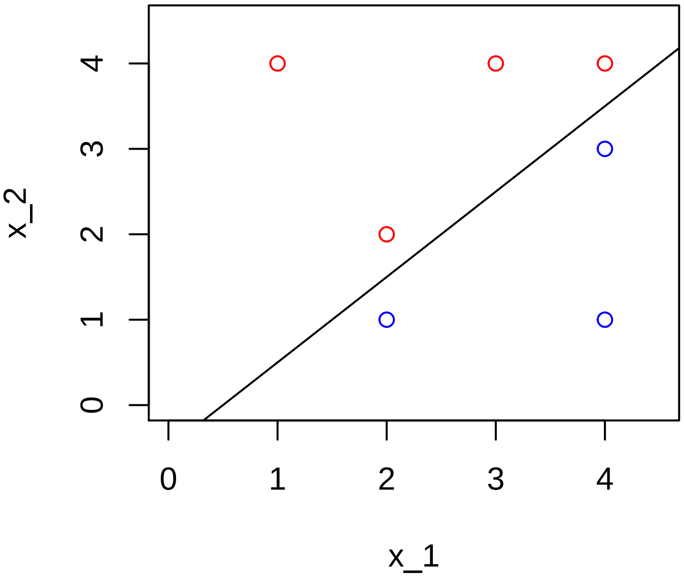
(a) We are given $n = 7$ observations in $p = 2$ dimensions. For each observation, there is an associated class label. Sketch the observations(see table).

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

$$(2,2),(4,4)\ (2,1),(4,3)$$

$$\Rightarrow (2,1.5),(4,3.5)\ b=(3.5-1.5)/(4-2)=1\ a=X_2-X_1=1.5-2=-0.5$$

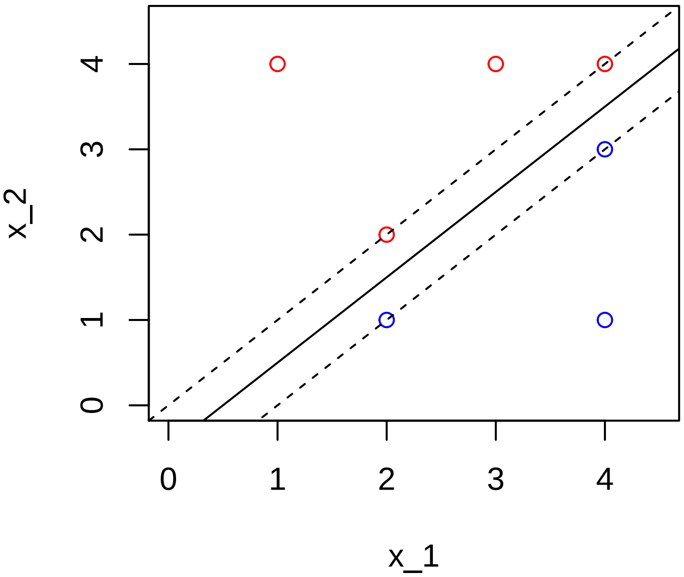
```
x_1 = c(3,2,4,1,2,4,4)
x_2 = c(4,2,4,4,1,3,1)
colors = c("red", "red", "red", "red", "blue", "blue", "blue")
plot(x_1,x_2, col = colors, xlim = c(0,4.5), ylim = c(0,4.5))
abline(-0.5, 1)
```



(c) Describe the classification rule for the maximal margin classifier. It should be something along the lines of Classify to Red if $\beta_0+\beta_1X_1+\beta_2X_2>0$, and classify to Blue otherwise. Provide the values for β_0, β_1, and β_2.

$$\text{Classification Rule} = \begin{cases} \text{RED} & \text{if 0.5 - X1 + X2 ¿ 0} \\ \text{BLUE} & \text{otherwise} \end{cases}$$

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



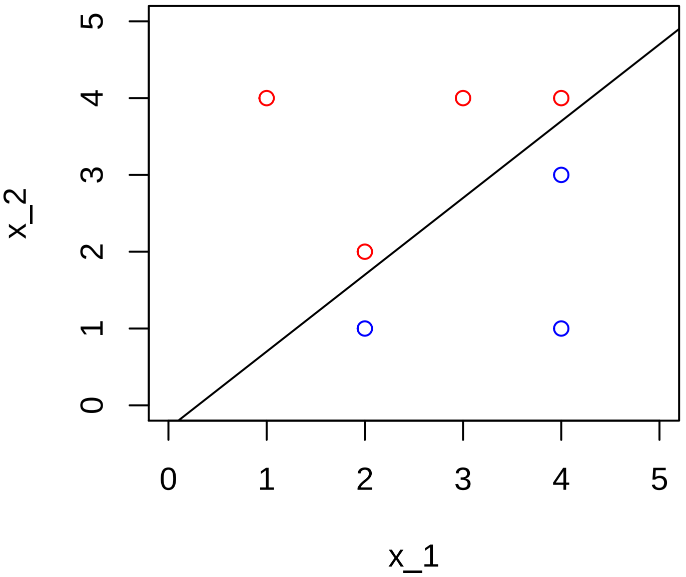
(e) Indicate the support vectors for the maximal margin classifier.

```
#plot(x_1,x_2,col=colors,xlim=c(0,4.5),ylim=c(0,4.5))
#abline(-0.5, 1)
```

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

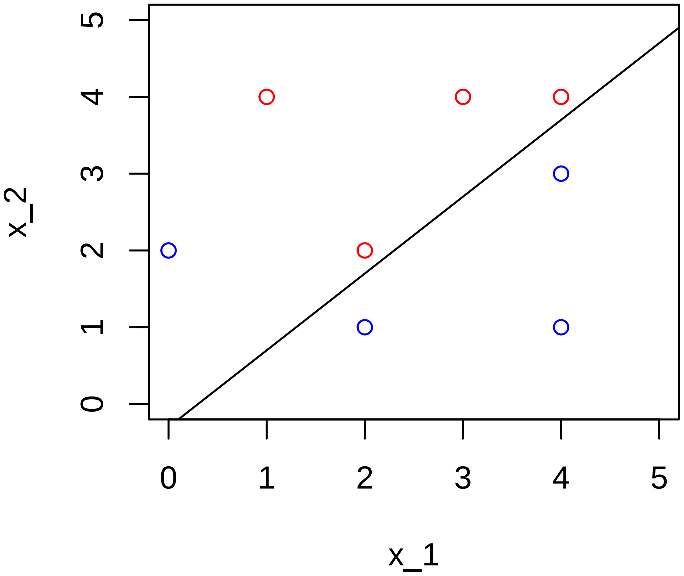
(g) Sketch a hyperplane that is not the optimal separating hyperplane, and provide the equation for this hyperplane.
Solution:$-0.3 - X_1 + X_2 > 0$

```
plot(x_1,x_2,col=colors,xlim=c(0,5),ylim=c(0,5))
abline(-0.3, 1)
```



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

```
plot(x_1,x_2,col=colors,xlim=c(0,5),ylim=c(0,5))
points(0, 2, col=c("blue"))
abline(-0.3, 1)
```



3. 9.7.7) 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.

   (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.

```
  require(ISLR)
  require(e1071)

  gasMed = median(Auto$mpg)
  dat    = data.frame(x = Auto$mpg,
                      y = as.factor(ifelse(Auto$mpg > gasMed, 1, 0)))
```

   (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.

   We see that `cost` = `1` results in the lowest cross-validation error rate.

```
#e1071 library
  tune.out = tune(svm ,y ~.,data= dat ,kernel ="linear",
                ranges=list(cost = c(0.001, 0.01, 0.1, 1,5,10,100) ))
 summary(tune.out$best.model)

##
```

```
## 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
##       gamma:  1
##
## Number of Support Vectors:  102
##
##  ( 51 51 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

(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.

We see that for polynomial basis kernels `cost` = `10` and `degree` = `2` results in the lowest cross-validation error rate.
We see that for radial basis kernels `cost` = `10` and `gamma` = `0.01` results in the lowest cross-validation error rate.

```
tune.out.poly = tune(svm, y ~.,data= dat,
                kernel = "polynomial",
                ranges = list(cost=c(0.1, 1, 5, 10),
                              degree = c(2, 3, 4)))
summary(tune.out.poly$best.model)
```

```
##
## Call:
## best.tune(method = svm, train.x = y ~ ., data = dat, ranges = list(cost = c(0.1,
##     1, 5, 10), degree = c(2, 3, 4)), kernel = "polynomial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  10
##      degree:  3
##       gamma:  1
##      coef.0:  0
##
## Number of Support Vectors:  90
##
##  ( 45 45 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
tune.out.rad = tune(svm, y ~.,data= dat,
                    kernel = "radial",
                    ranges = list(cost=c(0.1, 1, 5, 10),
                                  gamma=c(0.01, 0.1, 1, 5)))
summary(tune.out.rad$best.model)
```

```
##
## Call:
## best.tune(method = svm, train.x = y ~ ., data = dat, ranges = list(cost = c(0.1,
##     1, 5, 10), gamma = c(0.01, 0.1, 1, 5)), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  10
##       gamma:  0.1
##
## Number of Support Vectors:  34
##
##  ( 17 17 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

(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 , x1x4)
in order to plot just the first and fourth variables. However, you must replace x_1 and x_4 with the correct variable names. To find out more, type ?plot.svm.

```
  svm.linear = svm(y ~.,data= dat, kernel="linear", cost=1)
  svm.poly   = svm(y ~.,data= dat, kernel="polynomial",
                   cost=10, degree=2)
  svm.rad    = svm(y ~.,data= dat, kernel="radial",
                   cost=10, gamma=0.01)

  #plot(svm.linear, Auto2, as.formula(paste("mpg~", "name", sep="")))
  #plot(svm.poly, dat)
  #plot(svm.rad, dat)
```

4. 9.7.8) 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.

   ```
     #set.eed(45)
     train = sample(dim(OJ)[1], 800)
     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.linear = svm(Purchase ~ ., kernel = "linear",
                      data = OJ.train, cost = 0.01)
     summary(svm.linear)

   ##
   ## Call:
   ## svm(formula = Purchase ~ ., data = OJ.train, kernel = "linear",
   ##     cost = 0.01)
   ##
   ##
   ## Parameters:
   ##    SVM-Type:  C-classification
   ##  SVM-Kernel:  linear
   ##        cost:  0.01
   ##       gamma:  0.05555556
   ##
   ## Number of Support Vectors:  436
   ```

```
##
##  ( 218 218 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

Creates 439 support vectors. 218 belong in CH and 221 in MM.

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

```
  train.pred = predict(svm.linear, OJ.train)
  tabl = table(OJ.train$Purchase, train.pred)
  1 - sum(diag(tabl))/sum(tabl)

## [1] 0.16625

  test.pred = predict(svm.linear, OJ.test)
  tabl = table(OJ.test$Purchase, test.pred)
  1-sum(diag(tabl))/sum(tabl)

## [1] 0.1703704
```

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

```
  tune.out = tune(svm, Purchase ~ ., data = OJ.train, kernel = "linear",
                  ranges = list(cost = c(0.01,0.03,0.1,1,5,10)))
  summary(tune.out$best.mod)

##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = OJ.train,
##     ranges = list(cost = c(0.01, 0.03, 0.1, 1, 5, 10)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  5
##       gamma:  0.05555556
##
## Number of Support Vectors:  332
##
##  ( 166 166 )
##
```

```
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

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

```
  svm.linear = svm(Purchase ~ ., kernel = "linear", data = OJ.train,
                   cost = tune.out$best.parameters$cost)

  train.pred = predict(svm.linear, OJ.train)
  tabl = table(OJ.train$Purchase, train.pred); tabl

##     train.pred
##       CH  MM
##   CH 436  56
##   MM  70 238

  1 - sum(diag(tabl))/sum(tabl)

## [1] 0.1575

  test.pred = predict(svm.linear, OJ.test)
  tabl = table(OJ.test$Purchase, test.pred)
  1 - sum(diag(tabl))/sum(tabl)

## [1] 0.1703704
```

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

```
  svm.rad = svm(Purchase ~ ., data = OJ.train, kernel = "radial")

  train.pred = predict(svm.rad, OJ.train)
  tabl = table(OJ.train$Purchase, train.pred); tabl

##     train.pred
##       CH  MM
##   CH 451  41
##   MM  78 230

  1 - sum(diag(tabl))/sum(tabl)

## [1] 0.14875

  test.pred = predict(svm.rad, OJ.test)
  tabl = table(OJ.test$Purchase, test.pred); tabl
```

```
##     test.pred
##       CH  MM
##   CH 144  17
##   MM  27  82
```

```
  1 - sum(diag(tabl))/sum(tabl)
```

```
## [1] 0.162963
```

```
  tune.out = tune(svm, Purchase ~ ., data = OJ.train, kernel = "radial",
                  ranges = list(cost = c(0.01,0.03,0.5,0.1,1,5,10)))
  summary(tune.out$best.mod)
```

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = OJ.train,
##     ranges = list(cost = c(0.01, 0.03, 0.5, 0.1, 1, 5, 10)),
##     kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  0.05555556
##
## Number of Support Vectors:  373
##
##  ( 189 184 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
  svm.rad = svm(Purchase ~ ., kernel = "radial", data = OJ.train,
                cost = tune.out$best.parameters$cost,
                gamma = tune.out$best.mod$gamma)
  train.pred = predict(svm.rad, OJ.train)
  tabl = table(OJ.train$Purchase, train.pred)
  1 - sum(diag(tabl))/sum(tabl)
```

```
## [1] 0.14875
```

```
  test.pred = predict(svm.rad, OJ.test)
  tabl = table(OJ.test$Purchase, test.pred)
  1 - sum(diag(tabl))/sum(tabl)
```

```
## [1] 0.162963
```

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

```
  svm.poly = svm(Purchase ~ ., data = OJ.train,
                 kernel = "polynomial", degree = 2)

  train.pred = predict(svm.poly, OJ.train)
  tabl = table(OJ.train$Purchase, train.pred)
  1 - sum(diag(tabl))/sum(tabl)
```

```
## [1] 0.18375
```

```
  test.pred = predict(svm.poly, OJ.test)
  tabl = table(OJ.test$Purchase, test.pred)
  1 - sum(diag(tabl))/sum(tabl)
```

```
## [1] 0.2
```

```
  tune.out = tune(svm, Purchase ~ ., data = OJ.train,
                  kernel = "polynomial",
                  ranges = list(cost = c(0.01,0.03,0.5,0.1,1,5,10)),
                  degree = 2)

  svm.poly = svm(Purchase ~ ., kernel = "polynomial", data = OJ.train,
                 cost = tune.out$best.mod$cost,
                 degree = tune.out$best.mod$degree)
  train.pred = predict(svm.poly, OJ.train)
  tabl = table(OJ.train$Purchase, train.pred)
  1 - sum(diag(tabl))/sum(tabl)
```

```
## [1] 0.1575
```

```
  test.pred = predict(svm.poly, OJ.test)
  tabl = table(OJ.test$Purchase, test.pred)
  1 - sum(diag(tabl))/sum(tabl)
```

```
## [1] 0.1962963
```

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