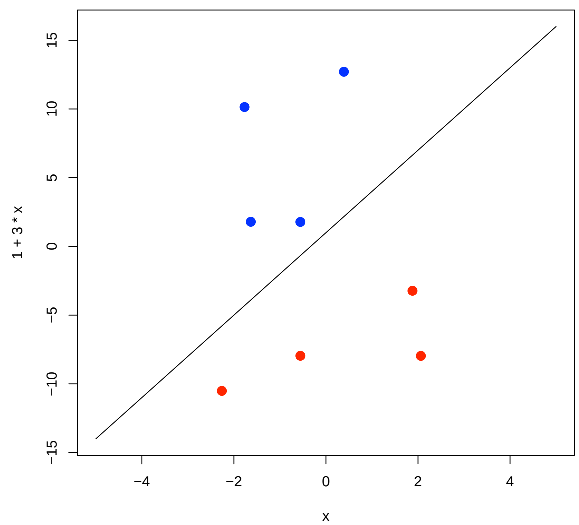
Homework # 11: Support Vector Machines (Chap. 9)

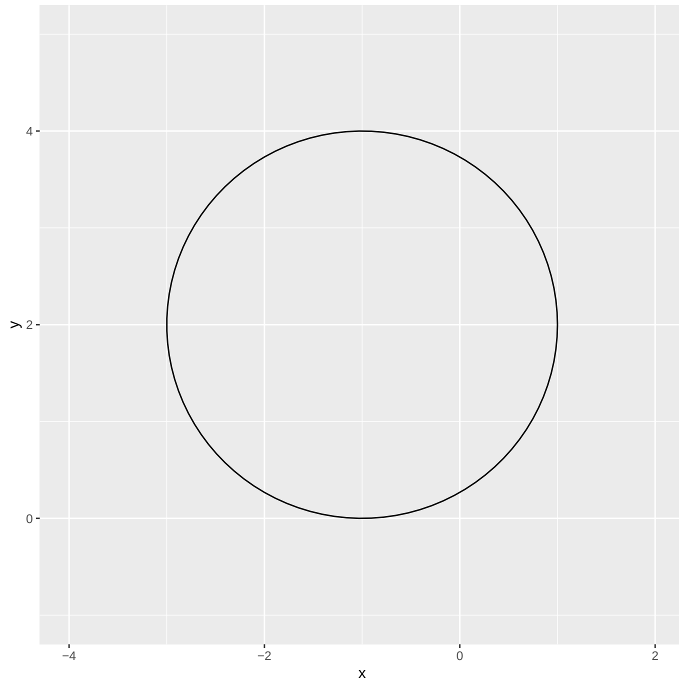
1. **(Chap. 9, # 1a, p. 368)** This problem involves a hyperplane in two dimensions. Sketch the hyperplane $1 + 3X_1 - X_2 = 0$. Indicate the set of points for which $1 + 3X_1 - X_2 > 0$, as well as the set of points for which $1 + 3X_1 - X_2 < 0$.



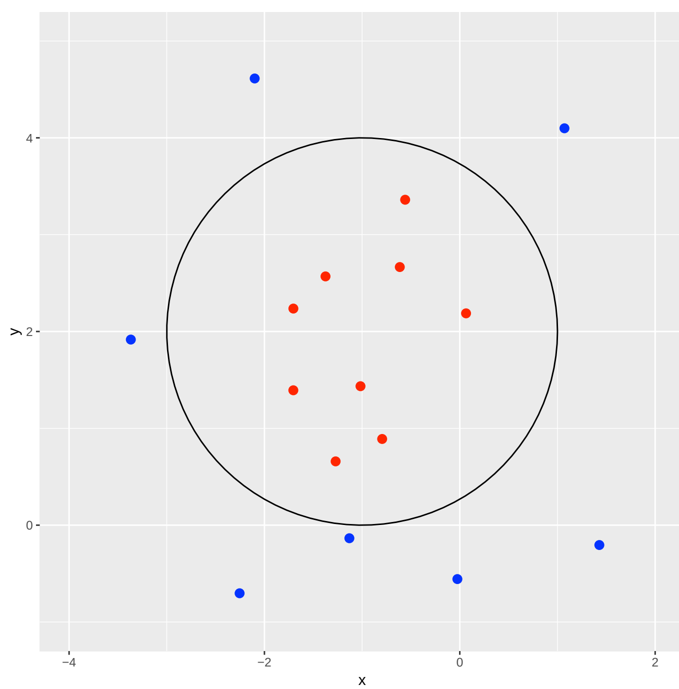
For the points which $1 + 3X_1 - X_2 < 0$, they are colored red. For the points which $1 + 3X_1 - X_2 > 0$, they are colored blue.

2. **(Chap. 9, # 2, p. 368)** 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 $(1 + X_1)^2 + (2 - X_2)^2 = 4$.

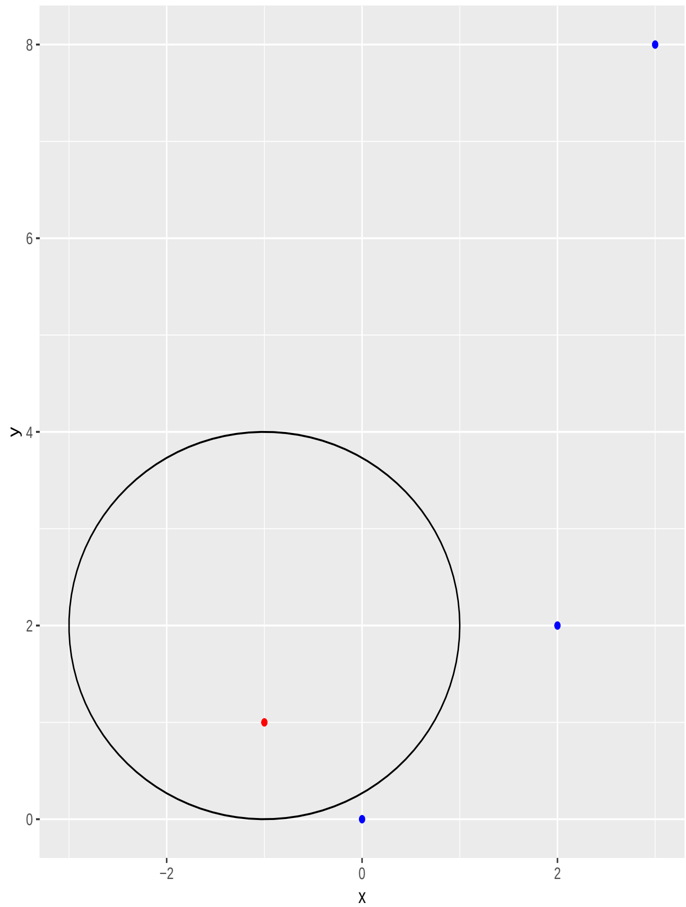


(b) On your sketch, indicate the set of points for which $(1+X_1)^2 + (2-X_2)^2 > 4$, as well as the set of points for which $(1+X_1)^2 + (2-X_2)^2 \leq 4$.



Blue points indicate $(1+X_1)^2 + (2-X_2)^2 > 4$, while red points indicate $(1+X_1)^2 + (2-X_2)^2 \leq 4$.

(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 is the observation (0, 0) classified? (-1, 1)? (2,2)? (3,8)?



The observation (0, 0), (2,2), (3,8) are classified as blue while (-1, 1) is classified as red.

(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.

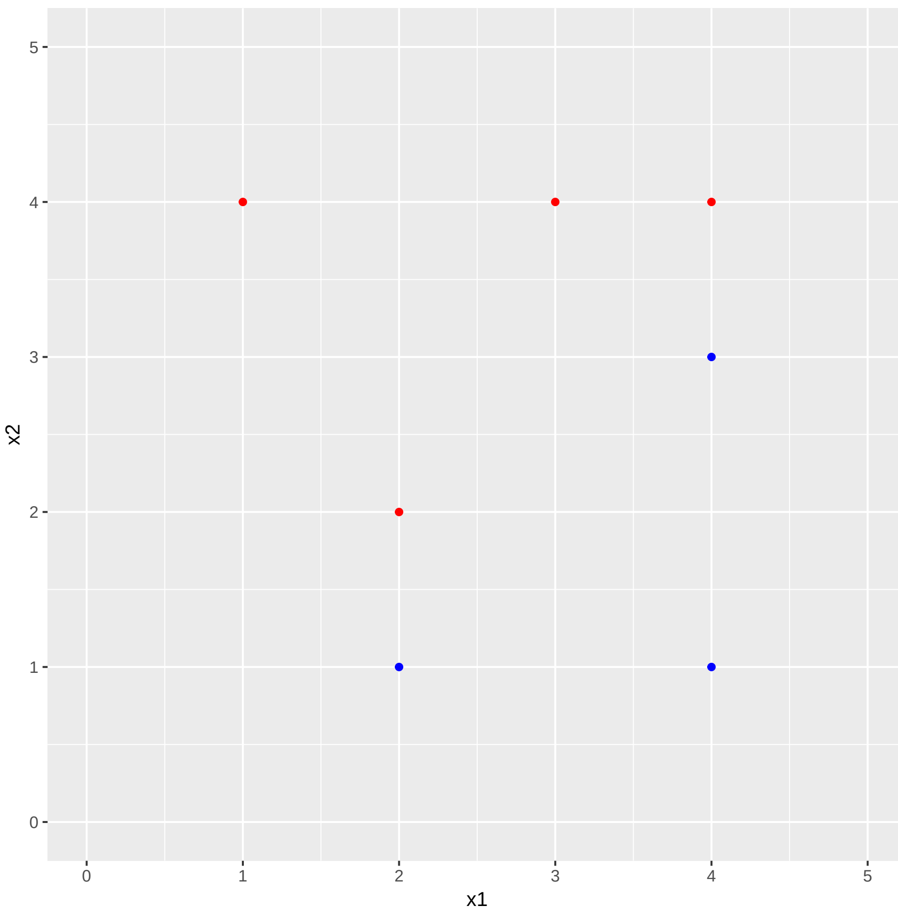
$$
\begin{aligned}
&(1+X_1)^2 + (2-X_2)^2 \\
=&1 + 2X_1 + X_1^2 + 4 - 4X_2 + X_2^2 \\
=&5 + 2X_1 + X_1^2 + 4 - 4X_2 + X_2^2
\end{aligned}
$$

3. **(Chap. 9, # 3, p. 368)** Here we explore the maximal margin classifier on a toy data set.

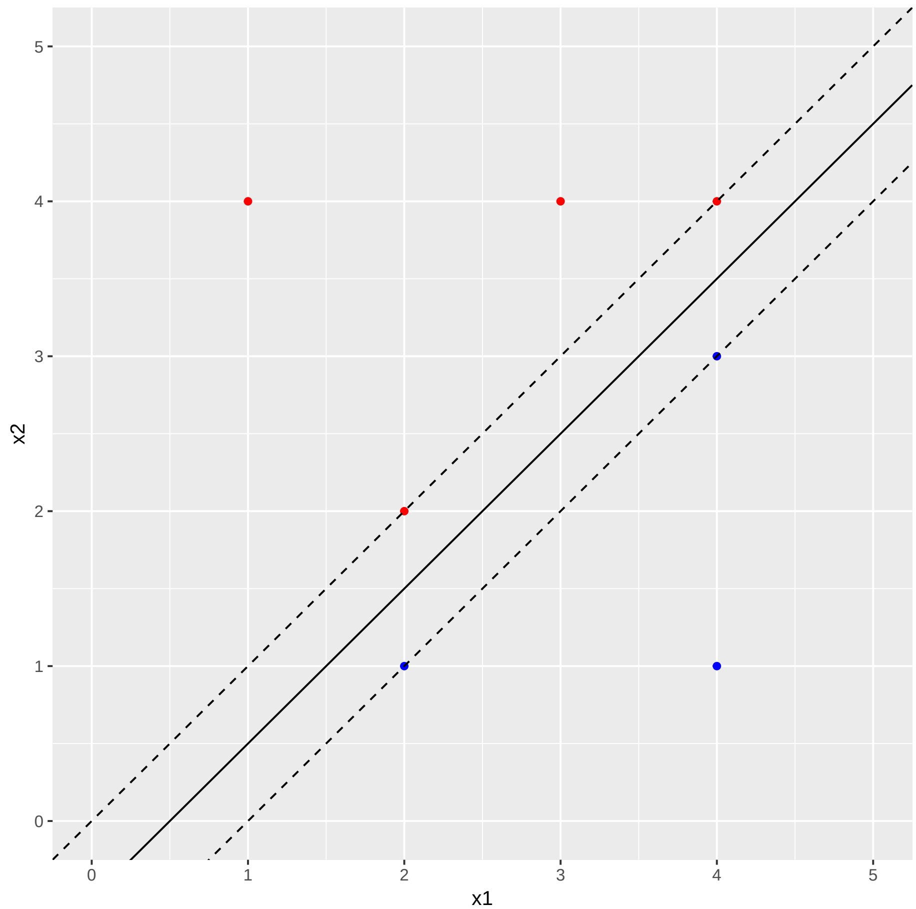
(a) We are given $n = 7$ observations in $p = 2$ dimensions. For each observation, there is an associated class label.

Obs.	X_1	X_2	Y
1	3	4	Red
2	2	2	Red
3	4	4	Red
4	1	4	Red
5	2	1	Blue
6	4	3	Blue
7	4	1	Blue

Sketch the observations.



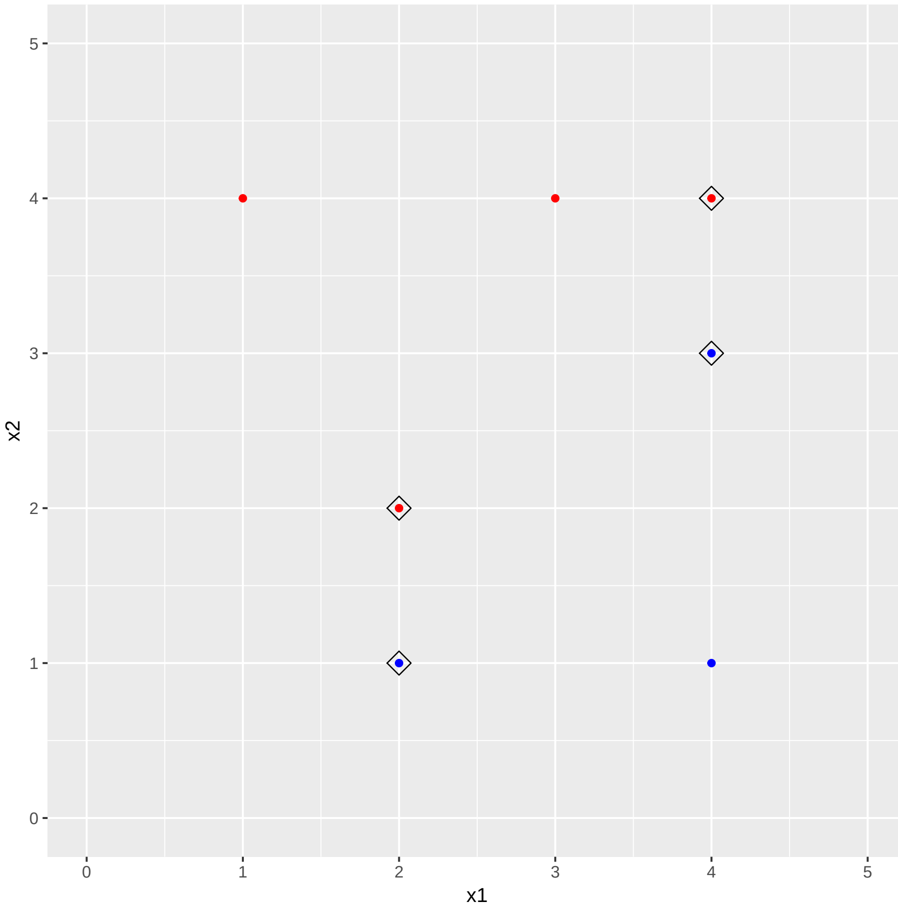
(b) Sketch the optimal separating hyperplane, and provide the equation for this hyperplane such as in exercise #1.



(c) Describe the classification rule for the maximal margin classifier. It should be something along the lines of "Classify to Red if $\beta_0 + \beta_1 X_1 + \beta_2 X_2 > 0$, and classify to Blue otherwise". Provide the values for $beta_0$, $beta_1$, and $beta_2$.
The observation is classified to Red if $0.5 - X_1 + X_2 > 0$, and classified to Blue otherwise. $beta_0 = 0.5$, $beta_1 = -1$, and $beta_2 = 1$.

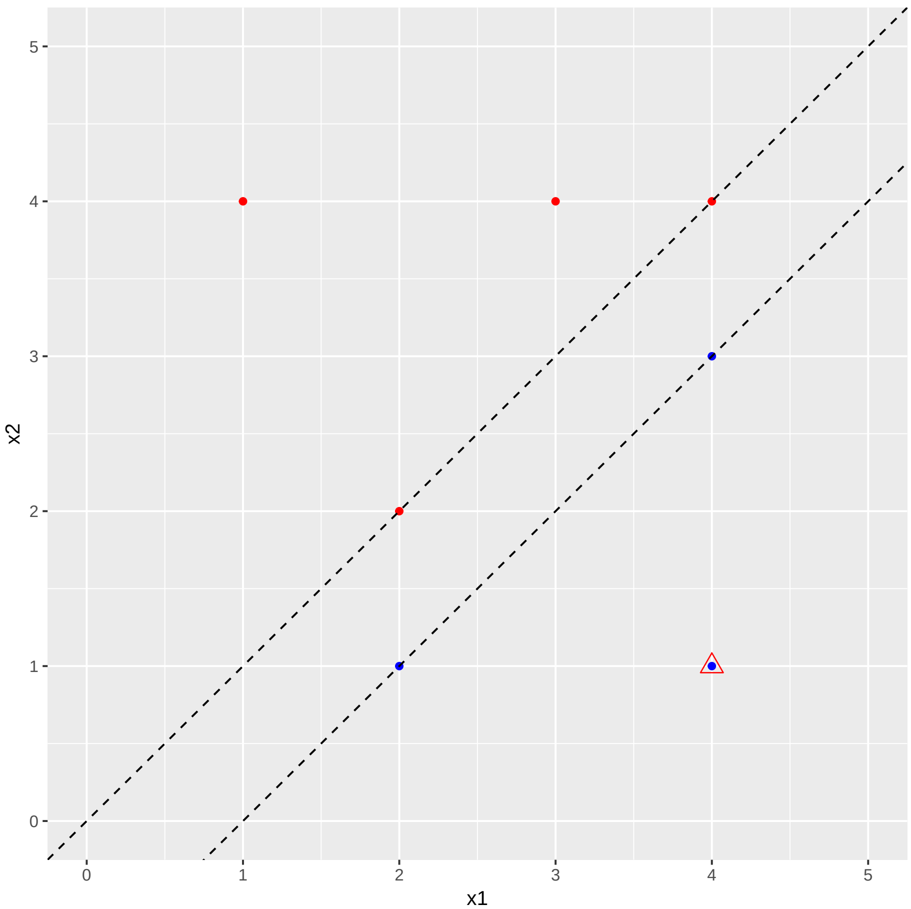
(d) On your sketch, indicate the margin for the maximal margin hyperplane.
Maximal margin: $M = \frac{\sqrt{0.5^2+0.5^2}}{2} = \frac{\sqrt{2}}{4}$.

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



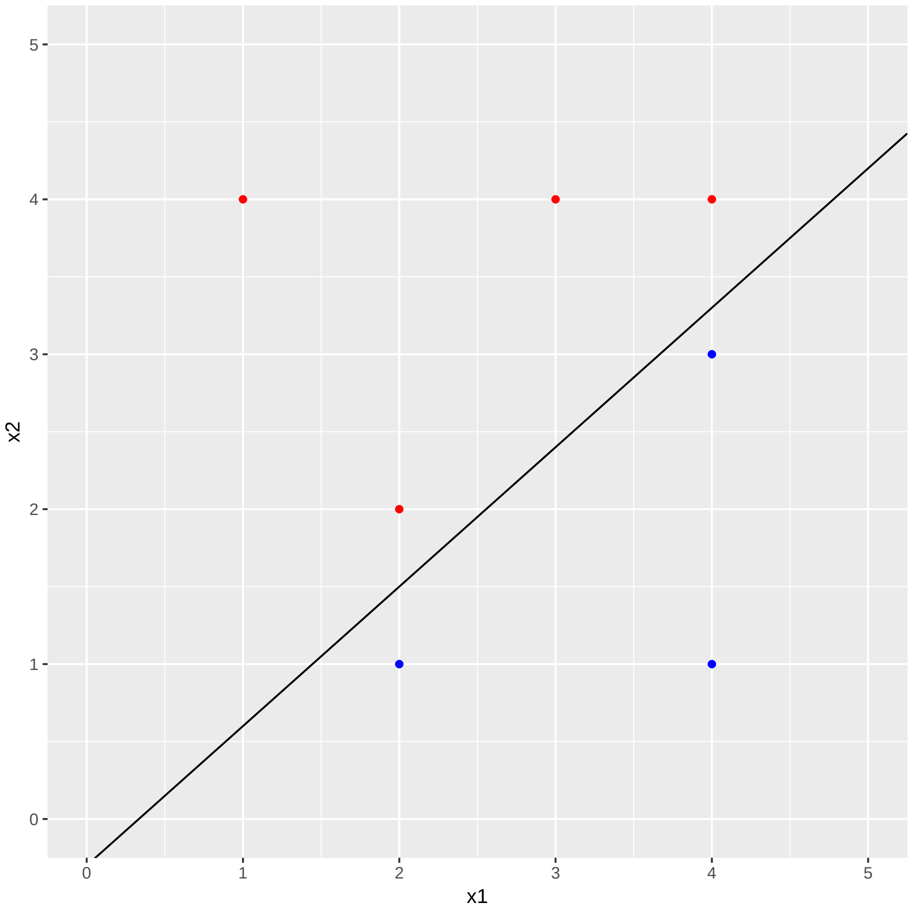
The points (2,1), (2,2), (4,3) and (4,4) are the support vectors.

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



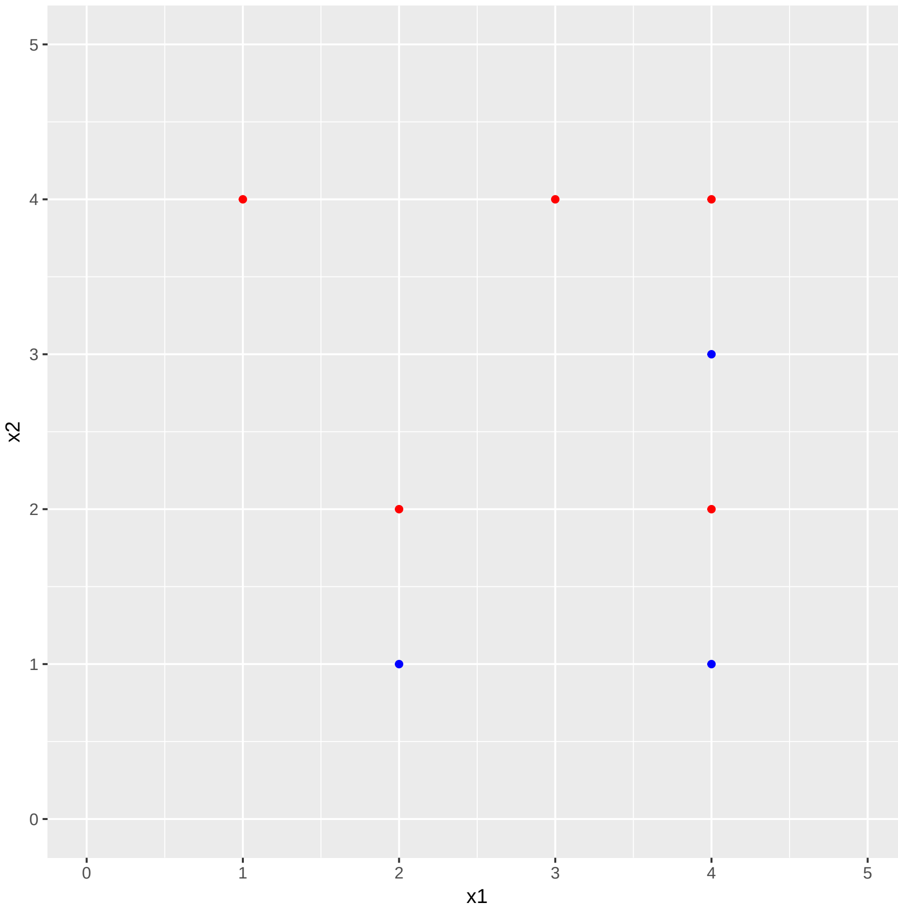
For the seventh observation, (4,1), a slight movement would not affect the maximal margin hyperplane as long as it does not enter the existed hyperplane.

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



This hyperplane is not optimal since it does not generate the largest margin, and the equation for this hyperplane is $0.3 - 0.9X_1 + X_2 = 0$

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



4. **(Chap. 9, # 7, p. 371)** 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 (from `ISLR`).

   i. 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$mileage <- rep(1,nrow(Auto))
Auto$mileage[Auto$mpg < median(Auto$mpg)] <- 0
Auto$mileage <- as.factor(Auto$mileage)
```

   ii. 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.

```
set.seed(444)
s.b <- tune(svm, mileage ~ weight + year, data = Auto,
                 ranges = list(cost = 10^seq(-3,3,1), kernel = 'linear'))
summary(s.b)

##
```

```
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost kernel
##   0.1 linear
##
## - best performance: 0.09192308
##
## - Detailed performance results:
##    cost kernel      error dispersion
## 1 1e-03 linear 0.41846154 0.09093521
## 2 1e-02 linear 0.10461538 0.05974185
## 3 1e-01 linear 0.09192308 0.04399032
## 4 1e+00 linear 0.09955128 0.05461402
## 5 1e+01 linear 0.09698718 0.04946574
## 6 1e+02 linear 0.09955128 0.05461402
## 7 1e+03 linear 0.09955128 0.05461402
```

```
s.b.2 <- tune(svm, mileage ~ weight + year, data = Auto,
              ranges = list(cost = seq(0.01,0.2,0.005), kernel = 'linear'))
summary(s.b.2)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost kernel
##  0.02 linear
##
## - best performance: 0.08935897
##
## - Detailed performance results:
##     cost kernel      error dispersion
## 1  0.010 linear 0.10205128 0.07645441
## 2  0.015 linear 0.09448718 0.08027832
## 3  0.020 linear 0.08935897 0.05832057
## 4  0.025 linear 0.09448718 0.05420465
## 5  0.030 linear 0.09705128 0.05388701
## 6  0.035 linear 0.09455128 0.05431324
## 7  0.040 linear 0.09455128 0.05431324
## 8  0.045 linear 0.09455128 0.05431324
```

```
## 9  0.050 linear 0.09711538 0.05399286
## 10 0.055 linear 0.09967949 0.05488189
## 11 0.060 linear 0.09967949 0.05488189
## 12 0.065 linear 0.09967949 0.05619719
## 13 0.070 linear 0.09711538 0.05399286
## 14 0.075 linear 0.09711538 0.05399286
## 15 0.080 linear 0.09967949 0.05619719
## 16 0.085 linear 0.09711538 0.05399286
## 17 0.090 linear 0.09711538 0.05399286
## 18 0.095 linear 0.09711538 0.05399286
## 19 0.100 linear 0.09711538 0.05399286
## 20 0.105 linear 0.09455128 0.05431324
## 21 0.110 linear 0.09711538 0.05399286
## 22 0.115 linear 0.09711538 0.05399286
## 23 0.120 linear 0.09711538 0.05399286
## 24 0.125 linear 0.09711538 0.05399286
## 25 0.130 linear 0.09711538 0.05399286
## 26 0.135 linear 0.09711538 0.05399286
## 27 0.140 linear 0.09711538 0.05399286
## 28 0.145 linear 0.09711538 0.05399286
## 29 0.150 linear 0.09711538 0.05399286
## 30 0.155 linear 0.09967949 0.05619719
## 31 0.160 linear 0.09967949 0.05619719
## 32 0.165 linear 0.09711538 0.05663421
## 33 0.170 linear 0.09455128 0.05431324
## 34 0.175 linear 0.09455128 0.05431324
## 35 0.180 linear 0.09455128 0.05431324
## 36 0.185 linear 0.09455128 0.05431324
## 37 0.190 linear 0.09455128 0.05431324
## 38 0.195 linear 0.09711538 0.05399286
## 39 0.200 linear 0.09455128 0.05431324
```

The best cost from the output of the 10-fold cross validation is 0.02, with the lowest cross-validation error 0.08935897.

iii. Now repeat (b), this time using SVMs with radial and polynomial basis kernels. Comment on your results.

```
set.seed(444)
s.c <- tune(svm, mileage ~ weight + year, data = Auto, ranges = list(
  cost = seq(0.01,0.2,0.005), kernel = c('linear','radial','polynomial')))
summary(s.c)

##
## Parameter tuning of 'svm':
##
```

```
## - sampling method: 10-fold cross validation
##
## - best parameters:
##   cost kernel
##  0.025 linear
##
## - best performance: 0.08423077
##
## - Detailed performance results:
##       cost     kernel      error dispersion
## 1   0.010     linear 0.10461538 0.05974185
## 2   0.015     linear 0.09705128 0.05388701
## 3   0.020     linear 0.09192308 0.05019521
## 4   0.025     linear 0.08423077 0.04689205
## 5   0.030     linear 0.08679487 0.04867313
## 6   0.035     linear 0.08935897 0.04724900
## 7   0.040     linear 0.09192308 0.04871814
## 8   0.045     linear 0.08935897 0.04724900
## 9   0.050     linear 0.08935897 0.04724900
## 10  0.055     linear 0.08935897 0.04724900
## 11  0.060     linear 0.09192308 0.05163005
## 12  0.065     linear 0.09192308 0.05163005
## 13  0.070     linear 0.09192308 0.04399032
## 14  0.075     linear 0.09192308 0.04399032
## 15  0.080     linear 0.09192308 0.04399032
## 16  0.085     linear 0.09192308 0.04399032
## 17  0.090     linear 0.09192308 0.04399032
## 18  0.095     linear 0.09192308 0.04399032
## 19  0.100     linear 0.09192308 0.04399032
## 20  0.105     linear 0.09192308 0.04399032
## 21  0.110     linear 0.09192308 0.04399032
## 22  0.115     linear 0.09192308 0.04399032
## 23  0.120     linear 0.09192308 0.04399032
## 24  0.125     linear 0.09192308 0.04399032
## 25  0.130     linear 0.09192308 0.04399032
## 26  0.135     linear 0.09192308 0.04399032
## 27  0.140     linear 0.09192308 0.04399032
## 28  0.145     linear 0.09442308 0.04519425
## 29  0.150     linear 0.09442308 0.04519425
## 30  0.155     linear 0.09192308 0.04399032
## 31  0.160     linear 0.09192308 0.04399032
## 32  0.165     linear 0.09192308 0.04399032
## 33  0.170     linear 0.09192308 0.04399032
## 34  0.175     linear 0.09192308 0.04399032
```

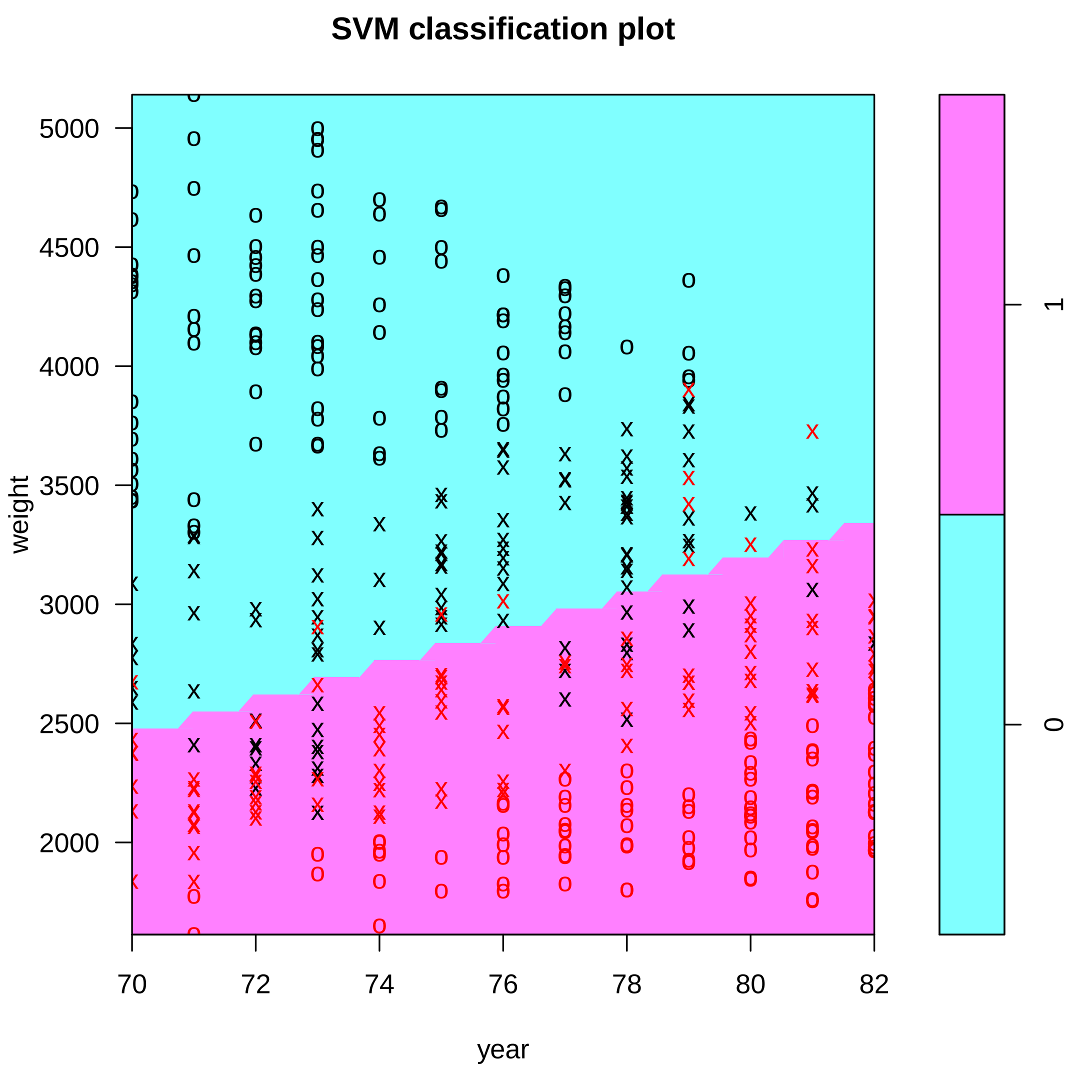
```
## 35  0.180     linear 0.09192308 0.04399032
## 36  0.185     linear 0.09192308 0.04399032
## 37  0.190     linear 0.09192308 0.04399032
## 38  0.195     linear 0.09192308 0.04399032
## 39  0.200     linear 0.09192308 0.04399032
## 40  0.010     radial 0.14051282 0.06110723
## 41  0.015     radial 0.09955128 0.05984588
## 42  0.020     radial 0.09442308 0.05679586
## 43  0.025     radial 0.08935897 0.05174930
## 44  0.030     radial 0.08935897 0.05174930
## 45  0.035     radial 0.09185897 0.05158214
## 46  0.040     radial 0.09442308 0.05139415
## 47  0.045     radial 0.09698718 0.05247373
## 48  0.050     radial 0.09955128 0.04911933
## 49  0.055     radial 0.09705128 0.05110386
## 50  0.060     radial 0.09448718 0.04698543
## 51  0.065     radial 0.09705128 0.04502436
## 52  0.070     radial 0.09705128 0.04502436
## 53  0.075     radial 0.09448718 0.04698543
## 54  0.080     radial 0.09705128 0.04502436
## 55  0.085     radial 0.09705128 0.04502436
## 56  0.090     radial 0.09961538 0.04915849
## 57  0.095     radial 0.09961538 0.04915849
## 58  0.100     radial 0.09961538 0.04915849
## 59  0.105     radial 0.09961538 0.04915849
## 60  0.110     radial 0.09961538 0.04915849
## 61  0.115     radial 0.09961538 0.04915849
## 62  0.120     radial 0.09961538 0.04915849
## 63  0.125     radial 0.10211538 0.04694056
## 64  0.130     radial 0.10467949 0.04766255
## 65  0.135     radial 0.10211538 0.04694056
## 66  0.140     radial 0.10211538 0.04694056
## 67  0.145     radial 0.10211538 0.04694056
## 68  0.150     radial 0.10211538 0.04694056
## 69  0.155     radial 0.10211538 0.04694056
## 70  0.160     radial 0.10211538 0.04694056
## 71  0.165     radial 0.10211538 0.04694056
## 72  0.170     radial 0.10467949 0.04766255
## 73  0.175     radial 0.10467949 0.04766255
## 74  0.180     radial 0.10467949 0.04766255
## 75  0.185     radial 0.10467949 0.04766255
## 76  0.190     radial 0.10467949 0.04766255
## 77  0.195     radial 0.10467949 0.04766255
## 78  0.200     radial 0.10980769 0.04219104
```

```
## 79  0.010 polynomial 0.23448718 0.09030846
## 80  0.015 polynomial 0.20121795 0.08974585
## 81  0.020 polynomial 0.18615385 0.08766790
## 82  0.025 polynomial 0.14801282 0.07728810
## 83  0.030 polynomial 0.13782051 0.07278833
## 84  0.035 polynomial 0.12756410 0.06833666
## 85  0.040 polynomial 0.11243590 0.07583767
## 86  0.045 polynomial 0.09448718 0.04999836
## 87  0.050 polynomial 0.09448718 0.05553599
## 88  0.055 polynomial 0.10217949 0.05934496
## 89  0.060 polynomial 0.09961538 0.05204577
## 90  0.065 polynomial 0.10474359 0.05868270
## 91  0.070 polynomial 0.10467949 0.05191916
## 92  0.075 polynomial 0.10474359 0.05326919
## 93  0.080 polynomial 0.10474359 0.05045197
## 94  0.085 polynomial 0.10730769 0.05510932
## 95  0.090 polynomial 0.10730769 0.05510932
## 96  0.095 polynomial 0.10730769 0.05510932
## 97  0.100 polynomial 0.10987179 0.05927953
## 98  0.105 polynomial 0.10730769 0.05510932
## 99  0.110 polynomial 0.10730769 0.05510932
## 100 0.115 polynomial 0.10730769 0.05510932
## 101 0.120 polynomial 0.10730769 0.05510932
## 102 0.125 polynomial 0.10730769 0.05510932
## 103 0.130 polynomial 0.10730769 0.05510932
## 104 0.135 polynomial 0.10730769 0.05510932
## 105 0.140 polynomial 0.10987179 0.05927953
## 106 0.145 polynomial 0.10987179 0.05927953
## 107 0.150 polynomial 0.11243590 0.05824145
## 108 0.155 polynomial 0.11243590 0.05824145
## 109 0.160 polynomial 0.11500000 0.06313457
## 110 0.165 polynomial 0.10987179 0.05927953
## 111 0.170 polynomial 0.11243590 0.06420732
## 112 0.175 polynomial 0.11243590 0.05824145
## 113 0.180 polynomial 0.11500000 0.06313457
## 114 0.185 polynomial 0.11500000 0.06313457
## 115 0.190 polynomial 0.11500000 0.06313457
## 116 0.195 polynomial 0.11243590 0.05824145
## 117 0.200 polynomial 0.11500000 0.05832292
```

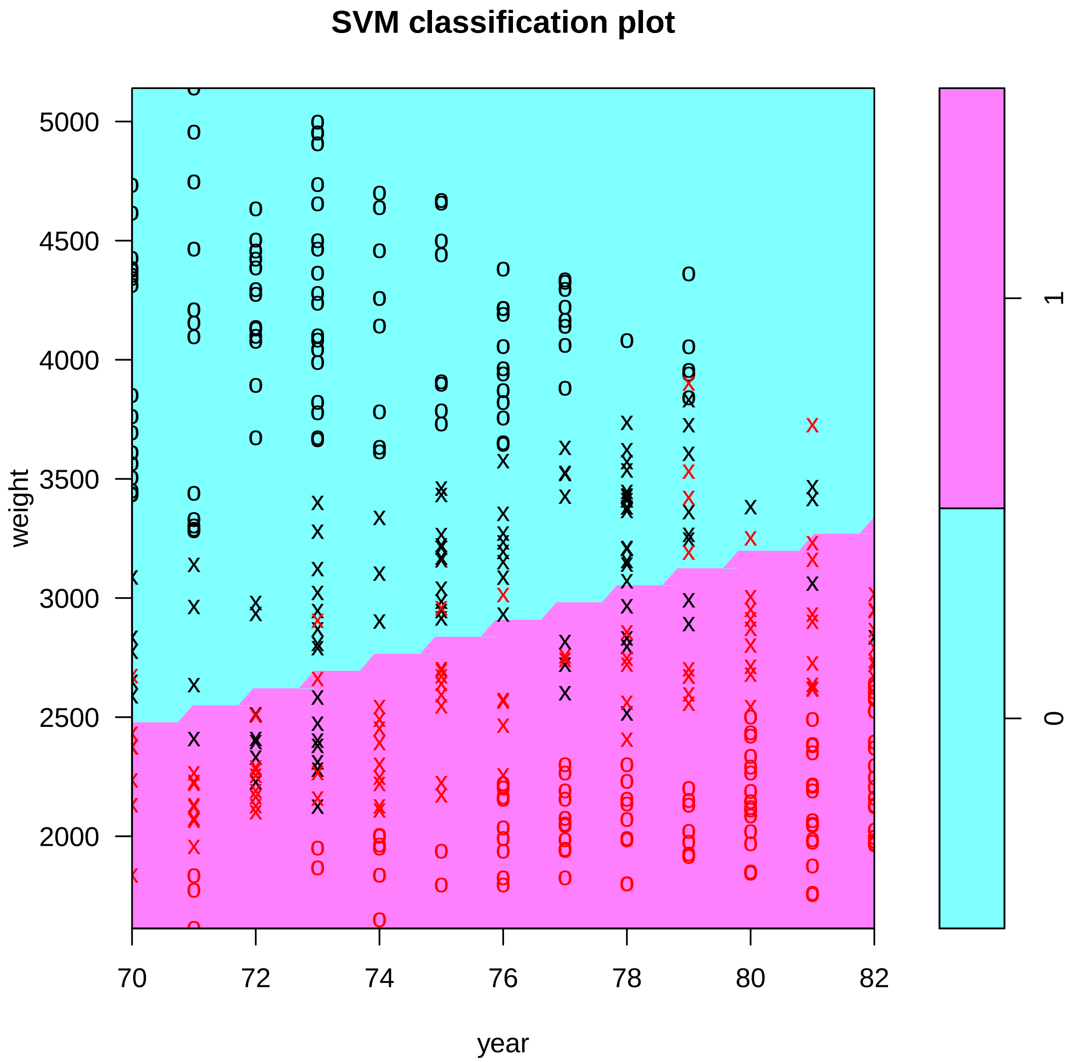
The best tuning parameters from the output of the 10-fold cross validation are 0.025 for cost and linear kernel, with the lowest cross-validation error 0.08423077.

iv. Make some plots to back up your assertions in (b) and (c). When $p > 2$, you can use the `plot()` function to create plots displaying pairs of variables at a time. For example, to plot horsepower and year, the syntax is `plot(svm.result, data=Auto, horsepower~year)`.

```
svm.b <- svm(mileage~weight+year, data = Auto, kernel = 'linear', cost = 0.02)
svm.c <- svm(mileage~weight+year, data = Auto, kernel = 'linear', cost = 0.025)
plot(svm.b, data=Auto, weight~year)
```




```
plot(svm.c, data=Auto, weight~year)
```


SVM classification plot
weight
year
5000
4500
4000
3500
3000
2500
2000
70
72
74
76
78
80
82
1
0