Lab-08

Ehshan Veerabangsa

29 November 2017

SVM on Linear separable dataset

Generate Linearly Separable Data

First we will create a dataset of 100 observation evenly split between two classes, positive and negative. Each observation will have 2 features. The mean of the two class will be the only differential.

```r
# Set seed
set.seed(300)

# Observations
observations <- 100

# Features
feaures <- 2

# Data points for each class
data_points <- observations * feaures

# Cluster means
pos_mean <- 0
neg_mean <- 3

postive = matrix(rnorm(data_points, mean = pos_mean), nrow = 100, ncol = 2)
negative = matrix(rnorm(data_points, mean = neg_mean), nrow = 100, ncol = 2)
```

We will now simulate labels for the dataset.

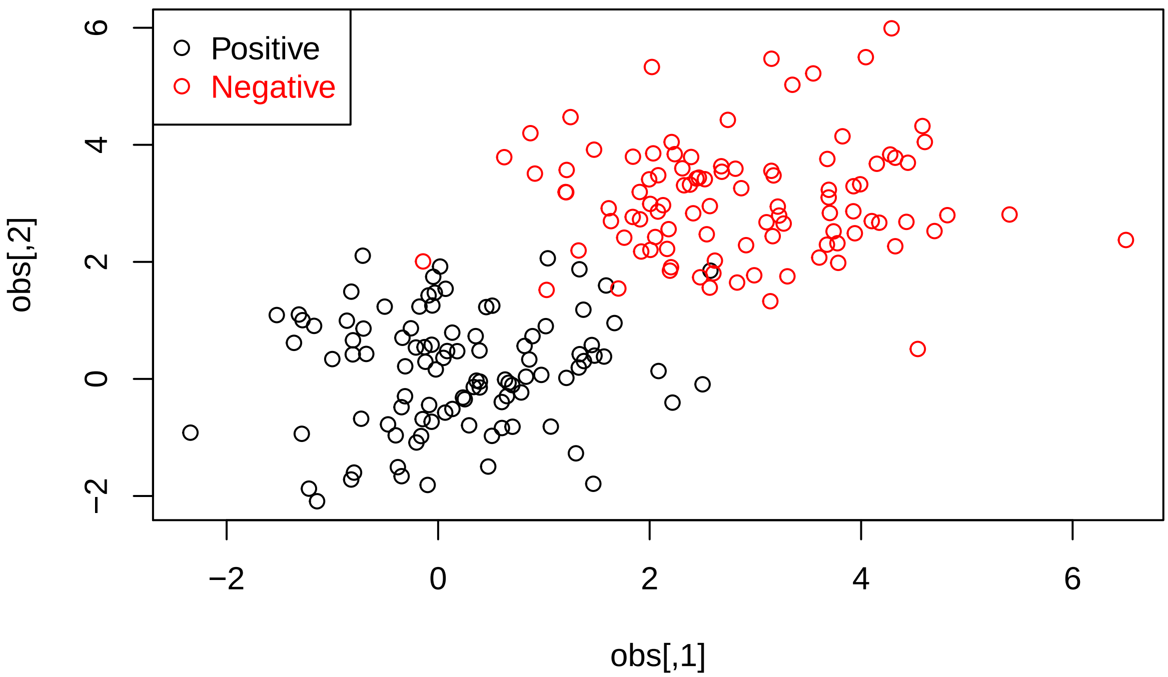
```r
labels <- c(rep(1, 100), rep(-1, 100))
```

Next, we bind the dataset to one variable and populate a dataframe.

```r
obs <- rbind(postive, negative)

data <- data.frame(x = obs, y = as.factor(labels))
```

The next step is to visualise the dataset with a plot.

```r
plot(obs, col=ifelse( labels > 0, 1, 2))

legend("topleft",c("Positive","Negative"),col=seq(2),pch=1,text.col=seq(2))
```



Finally we can split our data into training and test sets by a 70/30 split

```
# Split data into training & test sets

set.seed(300)

train_indices <- sample(200, 200 * 0.7)

training_set <- data[train_indices, ]

test_obs <- obs[-train_indices, ]
test_labels <- labels[-train_indices]
```

Part 1: Load SVM Library

Before we begin our experiements, we will need to load the apporate library.

```
#install.packages("e1071")
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 3.3.3
```

Part 2: Linear SVM with cost = 1

Now we can fit a linear SVM with a cost parameter of 1.

```r
linear_fit_1 = svm(y~.,data = training_set, kernel = "linear",cost = 1)
```

Part 3: Report Results

The next step is to check the figures from our new model.

```r
summary(linear_fit_1)
```

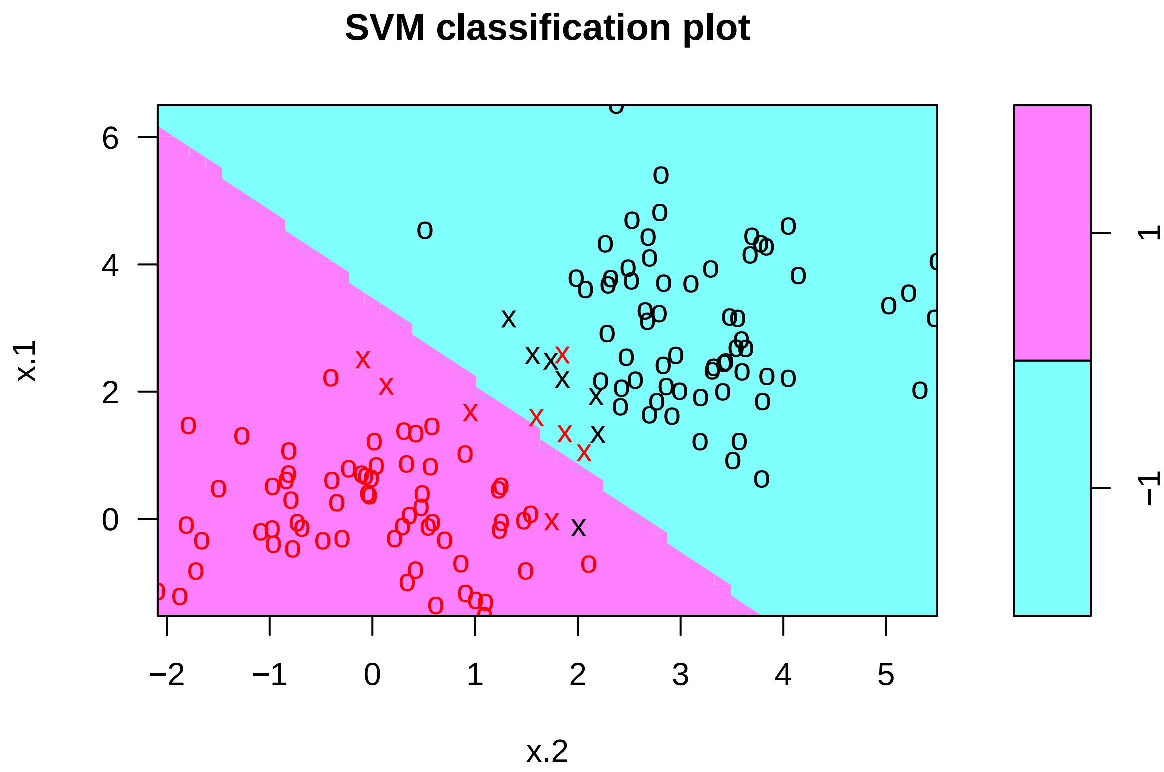
```
##
## Call:
## svm(formula = y ~ ., data = training_set, kernel = "linear",
##     cost = 1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##       gamma:  0.5
##
## Number of Support Vectors:  15
##
##  ( 7 8 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

The results shows that there are 12 spoort vetors

Part 4: Linear SVM Plot

We will now visualise our model.

```r
plot(linear_fit_1, training_set)
```



The results show that there are 3 points close to the margin, 1 on the pink side, 2 on the blue.

Part 5: Predictions on Linear Model

Finally for this model, we can check the error rate, by making predictions, and comparing them with our test set.

```
# predictions
prediction_linear_1 = predict(linear_fit_1, newdata = test_obs)

# error calculation
mean(prediction_linear_1 != test_labels)
```

```
## [1] 0.01666667
```

The model has an error rate of 0.01666667.

Part 6: Linear Models with cost 0.01 & 100000

We will now repeat parts 2 - 5 for model with the cost parameter of 0.01 and 1e5

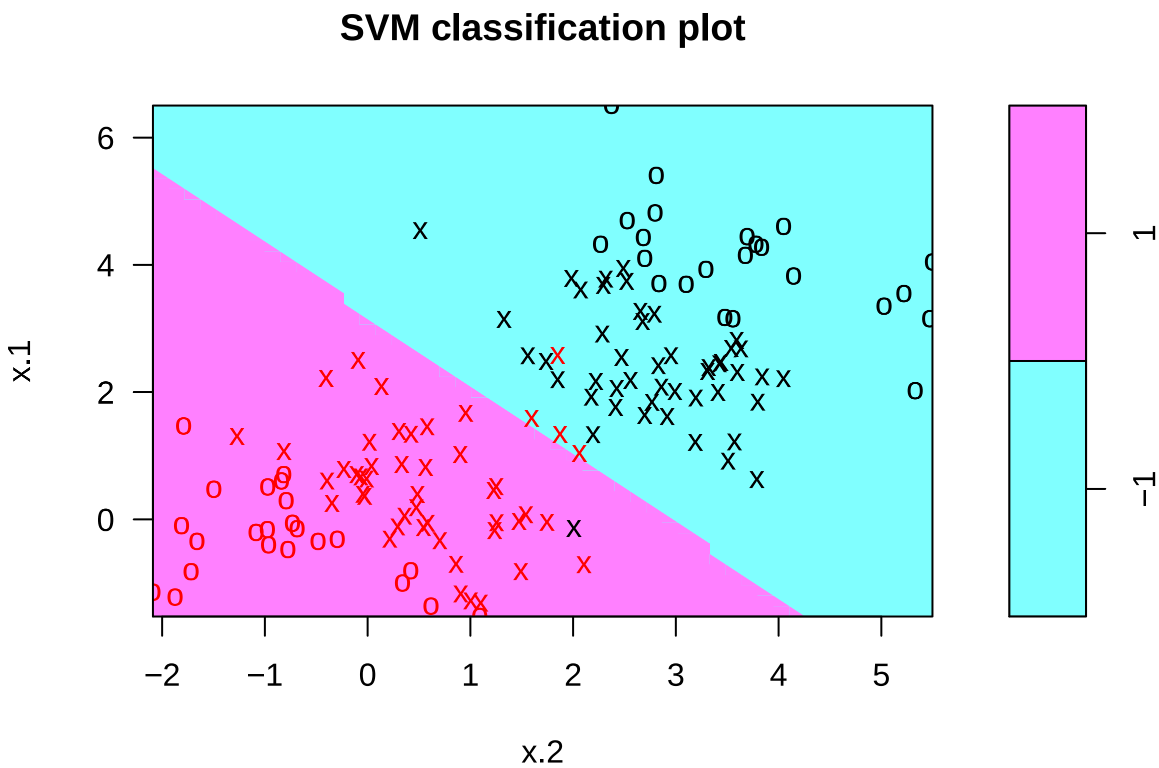
a: cost = 0.01

```
# build model
linear_fit_001 = svm(y~.,data = training_set, kernel = "linear",cost = 0.01)
```

```r
# summary
summary(linear_fit_001)
```

```
##
## Call:
## svm(formula = y ~ ., data = training_set, kernel = "linear",
##     cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##       gamma:  0.5
##
## Number of Support Vectors:  94
##
##  ( 47 47 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

```r
# 92 support vectors

# plot
plot(linear_fit_001, training_set)
```




```
# similar boundray

# predictions
prediction_linear_001 = predict(linear_fit_001, newdata = test_obs)

# error calculation
mean(prediction_linear_001 != test_labels)
```

```
## [1] 0.01666667
```

The results show that this model has the same error rate.

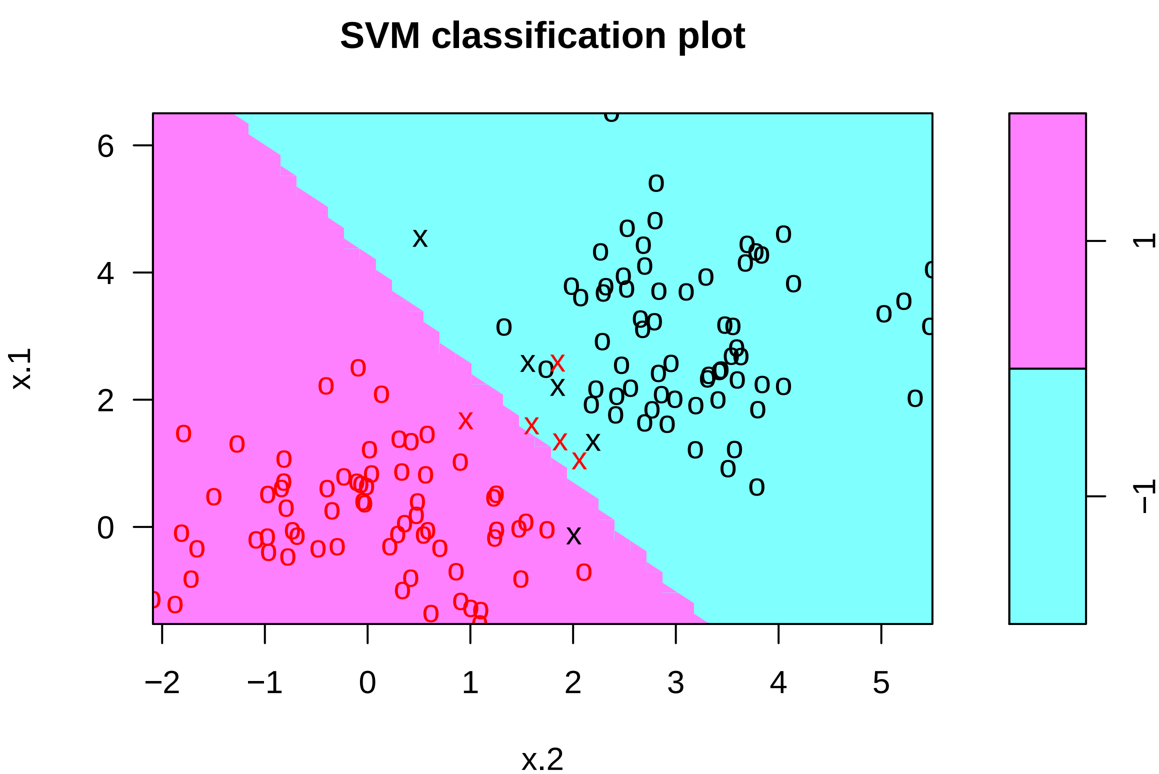
b: cost = 1e5

```
# build model
linear_fit_1e5 = svm(y~.,data = training_set, kernel = "linear",cost = 1e5)

# summary
summary(linear_fit_1e5)
```

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

```
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1e+05
##       gamma:  0.5
##
## Number of Support Vectors:  10
##
##  ( 5 5 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

```r
# 8 support vectors

# plot
plot(linear_fit_1e5, training_set)
```




```r
# Only 2 points close to boundary on pink side

# predictions
prediction_linear_1e5 = predict(linear_fit_1e5, newdata = test_obs)

# error calculation
```

```r
mean(prediction_linear_1e5 != test_labels)
```

```
## [1] 0.01666667
```

Again we have the same error rate.

Part 7: Find Optimal Model

We can the tune function to find the optimal cost parameter.

```r
set.seed(1)

# range of cost values
costs <- c(0.001,0.01,0.1,1,5,10,100,1000,10000,1e5)

tuned_fit = tune(svm, y~., data = data ,kernel="linear",ranges=list(costs))


summary(tuned_fit)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##   Var1
##  0.001
##
## - best performance: 0.03
##
## - Detailed performance results:
##     Var1 error dispersion
## 1  1e-03  0.03  0.0421637
## 2  1e-02  0.03  0.0421637
## 3  1e-01  0.03  0.0421637
## 4  1e+00  0.03  0.0421637
## 5  5e+00  0.03  0.0421637
## 6  1e+01  0.03  0.0421637
## 7  1e+02  0.03  0.0421637
## 8  1e+03  0.03  0.0421637
## 9  1e+04  0.03  0.0421637
## 10 1e+05  0.03  0.0421637
```

The optimal cost parameter is 0.001.

SVM on Linear inseparable dataset

Generate Linear inseparable dataset

For the second part of our experiements, we will generate a linear inseparable dataset, simiarly of 100 observation, with 2 feature, split 50/50. The negative class will be split into 2 clusters, each with a mean higher or lower that the positive class.

```
# Set seed
set.seed(300)

# half and half with different means
positive_1 <- matrix(rnorm((data_points/2), mean = pos_mean), nrow = 50, ncol = 2)

positive_2 <- matrix(rnorm((data_points/2), mean = neg_mean), nrow = 50, ncol = 2)


set.seed(300)


negative_1 = matrix(c(rnorm((data_points/2),mean = neg_mean) + 3,
                      rnorm((data_points/2),mean = neg_mean)),nrow = 50,ncol = 2)

negative_2 = matrix(c(rnorm((data_points/2),mean = pos_mean) - 3,
                      rnorm((data_points/2),mean = pos_mean)),nrow = 50,ncol = 2)
```

We will again create simulated labels for our new dataset.

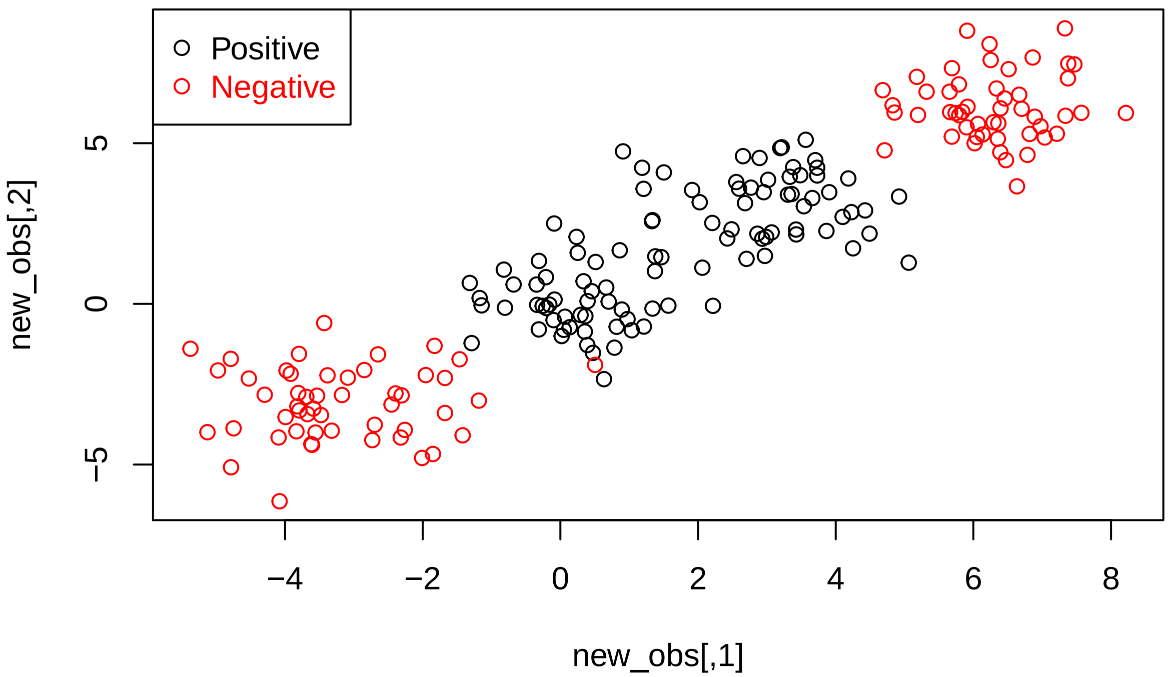
```
new_labels <- c(rep(1, 100), rep(-1, 100))
```

And again, we will bind and populate a dataframe.

```
new_obs <- rbind(positive_1, positive_2, negative_1, negative_2)

new_data <- data.frame(x = new_obs, y = as.factor(new_labels))
```

We visulaise our dataset.

```
plot(new_obs, col=ifelse(new_labels > 0, 1, 2))
legend("topleft",c("Positive","Negative"),col=seq(2),pch=1,text.col=seq(2))
```



Finally we can again split our data into training and test sets by a 70/30 split.

```
set.seed(300)

new_train_indices <- sample(200, 200 * 0.7)

new_training_set <- new_data[new_train_indices, ]

new_test_obs <- obs[-new_train_indices, ]
new_test_labels <- labels[-new_train_indices]
```

Part 8: SVM model with a radial kernel, gamma =1 and cost = 1.

To deal with our linear inseparable dataset, we will build a radial model, with a cost of 1 and gamma parameter also of 1.

```
non_linear_fit_1 = svm(y~.,data = new_training_set, kernel = "radial",gamma = 1, cost = 1)
```

a: Summary

```
summary(non_linear_fit_1)
```

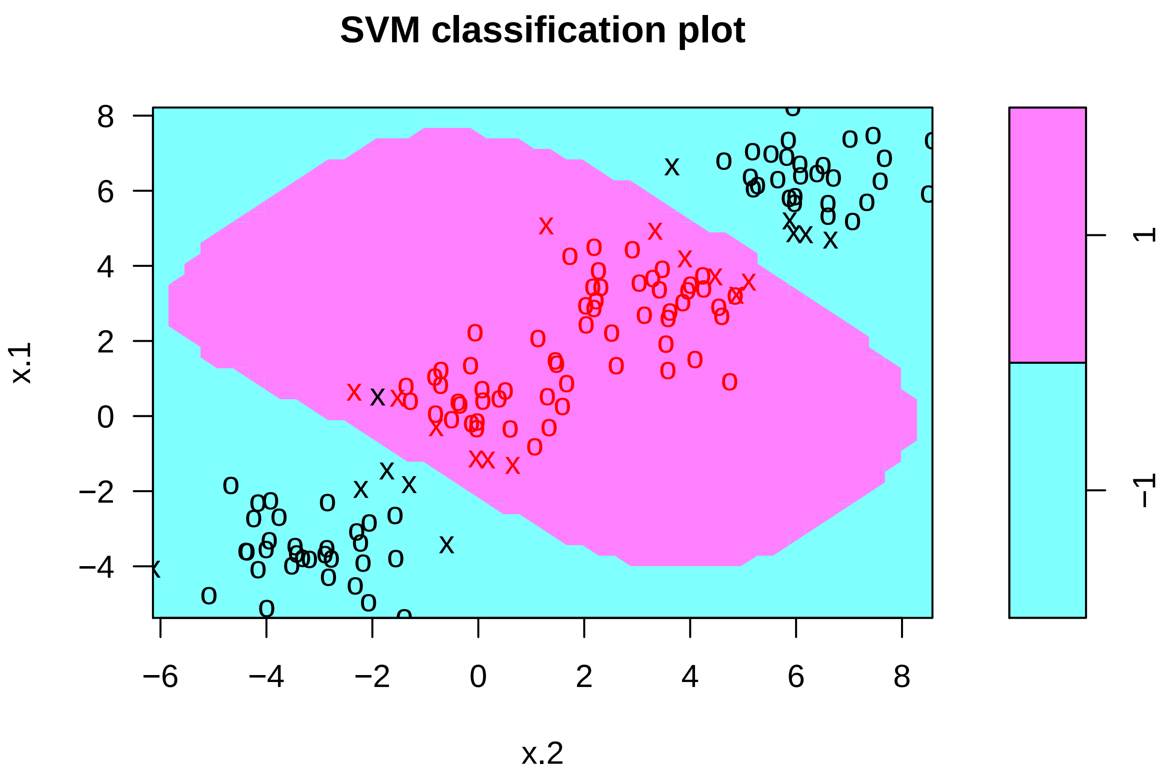
```
##
## Call:
```

```
## svm(formula = y ~ ., data = new_training_set, kernel = "radial",
##     gamma = 1, cost = 1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  1
##
## Number of Support Vectors:  23
##
##  ( 11 12 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

The model has 28 support vectors.

b: Plot

```r
plot(non_linear_fit_1, new_training_set)
```



C: Test error

```r
# predictions
prediction_non_linear_1 = predict(non_linear_fit_1, newdata = new_test_obs)

# error calculation
mean(prediction_non_linear_1 != new_test_labels)
```

```
## [1] 0.5
```

The model has and error of 0.5.

Part 9: SVM model with a radial kernel, gamma =1 and cost = 1e5.

We will follow the same process as part 8, build instead use a model with a cost of 1e5 and a gamma value of -1

```r
# build model
non_linear_fit_1e5 = svm(y~.,data = new_training_set, kernel = "radial",gamma = 1,cost = 1e5)
```

a: Summary

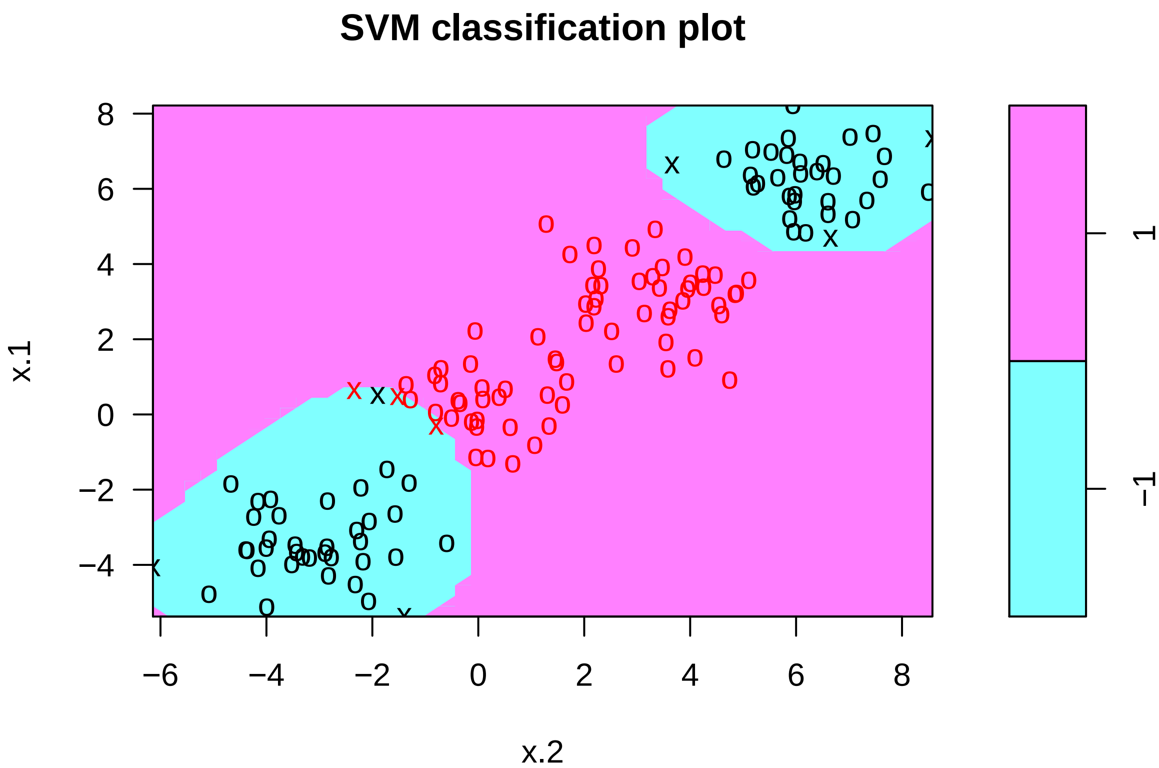
```r
summary(non_linear_fit_1e5)
```

```
##
## Call:
## svm(formula = y ~ ., data = new_training_set, kernel = "radial",
##     gamma = 1, cost = 1e+05)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1e+05
##       gamma:  1
##
## Number of Support Vectors:  9
##
##  ( 6 3 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

The model has used 12 support vectors.

b: Plot

```r
plot(non_linear_fit_1e5, new_training_set)
```



C: Test error

```
# predictions
prediction_non_linear_1e5 = predict(non_linear_fit_1e5, newdata = new_test_obs)

# error calculation
mean(prediction_non_linear_1e5 != new_test_labels)
```

```
## [1] 0.5833333
```

The model gives an error of 0.4833333 - a slight inprovement.

Part 10: Optimal values for cost and gamma

We can use the tune function to find the optimal values for both parameters.

```
set.seed(1)

costs <- c(0.001,0.01,0.1,1,5,10,100,1000,10000,1e5)

gammas <- c(0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4)

tuned_radial <- tune(svm,y~.,data = new_training_set, kernel = "radial",ranges = list(cost = costs, gamm
```

Summarise the results.

```r
summary(tuned_radial)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##   0.1   0.5
##
## - best performance: 0.007142857
##
## - Detailed performance results:
##     cost gamma       error dispersion
## 1  1e-03   0.5 0.571428571 0.19047619
## 2  1e-02   0.5 0.571428571 0.19047619
## 3  1e-01   0.5 0.007142857 0.02258770
## 4  1e+00   0.5 0.007142857 0.02258770
## 5  5e+00   0.5 0.007142857 0.02258770
## 6  1e+01   0.5 0.014285714 0.03011693
## 7  1e+02   0.5 0.014285714 0.03011693
## 8  1e+03   0.5 0.014285714 0.03011693
## 9  1e+04   0.5 0.064285714 0.06254250
## 10 1e+05   0.5 0.064285714 0.06254250
## 11 1e-03   1.0 0.571428571 0.19047619
## 12 1e-02   1.0 0.571428571 0.19047619
## 13 1e-01   1.0 0.007142857 0.02258770
## 14 1e+00   1.0 0.007142857 0.02258770
## 15 5e+00   1.0 0.007142857 0.02258770
## 16 1e+01   1.0 0.014285714 0.03011693
## 17 1e+02   1.0 0.014285714 0.03011693
## 18 1e+03   1.0 0.050000000 0.05880519
## 19 1e+04   1.0 0.064285714 0.06254250
## 20 1e+05   1.0 0.064285714 0.06254250
## 21 1e-03   1.5 0.571428571 0.19047619
## 22 1e-02   1.5 0.571428571 0.19047619
## 23 1e-01   1.5 0.007142857 0.02258770
## 24 1e+00   1.5 0.007142857 0.02258770
## 25 5e+00   1.5 0.007142857 0.02258770
## 26 1e+01   1.5 0.014285714 0.03011693
## 27 1e+02   1.5 0.014285714 0.03011693
## 28 1e+03   1.5 0.050000000 0.05880519
## 29 1e+04   1.5 0.064285714 0.05270463
## 30 1e+05   1.5 0.064285714 0.05270463
## 31 1e-03   2.0 0.571428571 0.19047619
## 32 1e-02   2.0 0.571428571 0.19047619
## 33 1e-01   2.0 0.007142857 0.02258770
## 34 1e+00   2.0 0.007142857 0.02258770
## 35 5e+00   2.0 0.014285714 0.03011693
## 36 1e+01   2.0 0.014285714 0.03011693
## 37 1e+02   2.0 0.014285714 0.03011693
## 38 1e+03   2.0 0.057142857 0.05634362
```

```
## 39 1e+04   2.0 0.071428571 0.05832118
## 40 1e+05   2.0 0.071428571 0.05832118
## 41 1e-03   2.5 0.571428571 0.19047619
## 42 1e-02   2.5 0.571428571 0.19047619
## 43 1e-01   2.5 0.007142857 0.02258770
## 44 1e+00   2.5 0.007142857 0.02258770
## 45 5e+00   2.5 0.014285714 0.03011693
## 46 1e+01   2.5 0.014285714 0.03011693
## 47 1e+02   2.5 0.021428571 0.03450328
## 48 1e+03   2.5 0.057142857 0.05634362
## 49 1e+04   2.5 0.057142857 0.05634362
## 50 1e+05   2.5 0.057142857 0.05634362
## 51 1e-03   3.0 0.571428571 0.19047619
## 52 1e-02   3.0 0.571428571 0.19047619
## 53 1e-01   3.0 0.007142857 0.02258770
## 54 1e+00   3.0 0.007142857 0.02258770
## 55 5e+00   3.0 0.014285714 0.03011693
## 56 1e+01   3.0 0.014285714 0.03011693
## 57 1e+02   3.0 0.028571429 0.04994328
## 58 1e+03   3.0 0.050000000 0.05880519
## 59 1e+04   3.0 0.050000000 0.05880519
## 60 1e+05   3.0 0.050000000 0.05880519
## 61 1e-03   3.5 0.571428571 0.19047619
## 62 1e-02   3.5 0.571428571 0.19047619
## 63 1e-01   3.5 0.007142857 0.02258770
## 64 1e+00   3.5 0.007142857 0.02258770
## 65 5e+00   3.5 0.014285714 0.03011693
## 66 1e+01   3.5 0.014285714 0.03011693
## 67 1e+02   3.5 0.028571429 0.04994328
## 68 1e+03   3.5 0.042857143 0.06023386
## 69 1e+04   3.5 0.042857143 0.06023386
## 70 1e+05   3.5 0.042857143 0.06023386
## 71 1e-03   4.0 0.571428571 0.19047619
## 72 1e-02   4.0 0.571428571 0.19047619
## 73 1e-01   4.0 0.007142857 0.02258770
## 74 1e+00   4.0 0.007142857 0.02258770
## 75 5e+00   4.0 0.014285714 0.03011693
## 76 1e+01   4.0 0.014285714 0.03011693
## 77 1e+02   4.0 0.028571429 0.04994328
## 78 1e+03   4.0 0.028571429 0.04994328
## 79 1e+04   4.0 0.028571429 0.04994328
## 80 1e+05   4.0 0.028571429 0.04994328
```

We can see that the best value for cost is 0.1 and the best value for gamma is 0.5.

Part 11: Test error on optimal model

Finally, we can check the test error on our optimal model.

```r
prediction_optimal_radial = predict(tuned_radial$best.model, newdata = new_test_obs)

mean(prediction_optimal_radial != new_test_labels)
```

```
## [1] 0.5
```