Support Vector Machines: Summary

Abhinav Mishra, Kristin Köhler, Sanket Gosavi, Utkarsha Kandale

2022-07-19

Contents

1 Introduction

A generalized approach for *maximal margin classifier*.

1.1 Maximal Margin Classifier

1.1.1 Hyperplane

A flat affine subspace of dimension $p-1$ in a p-dimensional space. Here, affine means that the subspace need not pass through the origin.

1. 2-d space, a hyperplane is a flat 1-d space i.e. a line.
2. 3-d space, a hyperplane is a flat 2-d space i.e. a plane.

For $p > 3$, the idea remains the same.

Mathematically, it is defined as:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p = 0$$

for β's $\neq 0$. In the equation, any $X = (X_1, X_2, \ldots, X_p)^T$ for which the equation holds true lies in the hyperplane.

James et al. (2013) suggests that the hyperplane is a combination of divided halves of p-dimensional space in case when X doesn't satisfy the mathematical definition so it would be either follow > 0 or < 0. To find out on which side of the hyperplane, a point lies, calculate the sign of the left hand side.

1.1.2 Seperating hyperplane

The goal is to develop a classifier based on the training data that will correctly classify the test observation using its feature measurements.

Precisely, we want to construct a hyperplane that separates the training observations perfectly according to their class labels, having the property that:

$$y_i(\beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \cdots + \beta_p X_{ip}) > 0$$

for all $i = 1, .., n$. Here, the $y_1, ..., y_n \epsilon {-1}, 1$, where -1 represents one class and 1 the other class, and test observation, a p-vector of observed features $x^* = (x_1^*, ..., x_p^*)$.

If a *separating hyperplane* exists, we can use it to construct a very natural classifier: a test observation is assigned a class depending on which side of the hyperplane it is located which leads to a **linear decision boundary**.

1.1.3 Maximal Marginal Classifier

For constructing a classifier based upon a separating hyperplane, the natural choice would be *maximal margin hyperplane* to decide which of the infinite possible separating hyperplanes to use, *if a hyperplane exists.* It is also known as *optimal seperating hyperplane*, which is the farthest from the test observations. The minimun perpendicular distance out of each training observation to the hyperplane is called *margin.* Here, the hyperplane is the one that has the farthest minimum distance to the test observations. When we classify a test observation based on which side of the *maximal margin hyperplane* it lies, it's called as *maximal margin classifier.* To calculate the maximal margin classifier, the following maximization problem is solved.

$$\max_{\beta_0, \beta_1, \ldots, \beta_p, M} M$$

$$\text{subject to} \sum_{j=1}^{p} \beta_j^2 = 1$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip}) > M \ \forall i = 1, ..., n$$

With large p's, maximal margin classifiers tend to overfit.

If $\beta_0, \beta_1, ..., \beta_p$ are the coefficients of the maximal margin hyperplane, then the *maximal margin classifier* classifies the test observation x^* based on the sign of $f(x^*) = \beta_0 + \beta_1 x_1^* + \beta_2 x_2^* + ... + \beta_p x_p^*$, where $f(x^*)$ gives the magnitude of the hyperplane.

The equidistant training observations from the maximal margin hyperplane that indicate the width of the margin, are **Support Vectors**, and they "support" the maximal margin hyperplane, as if these points would move, it would also move the hyperplane.

The generalization of the maximal margin classifier to the non-separable case is known as *Support Vector Classifier.*

2 Support Vector Classifiers

Sometimes, the observations from two classes are not perfectly separable by a hyperplane causing the mentioned maximization problem to have no solution. Additionally, maximal margin classifiers are largely sensitive to individual observations especially to outliers. So, the addition of a single observation can create a substantial change in the margin, and if it's a small margin, then confidence for correct classification is somewhat low as the margin is partly responsible for the accuracy of the classifier. A trade-off with a hyperplane which does **NOT** perfectly seperate the two classes, in interest of:

- greater robustness to individual observations
- better classification for the most observations

The support vector classifier is a *soft margin* classifier, since we're allowing some observations to be on the incorrect side of the margin, or the hyperplane.

2.1 Slack Variables

By multiplying the maximal margin M with the error term $(1 - \epsilon_i)$, we allow some observations to be missclassified, either on the wrong side of the margin or even on the wrong side of the hyperplane. The maximization problem then changes to:

$$\max_{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n,M} M$$

$$\text{subject to} \sum_{j=1}^{p} \beta_j^2 = 1$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M(1 - \epsilon_i),$$

$$\epsilon_i \geq 0, \sum_{i=1}^{n} \epsilon_i \leq C$$

The slack variable ϵ_i tells us where the ith observation is located, relative to the hyperplane and relative to the margin.

1. If $\epsilon_i = 0$ then, the ith observation is on the correct side of the margin.
2. If $\epsilon_i > 0$ then, the ith observation is on the wrong side of the m argin, and we say that the ith observation has violated the margin.
3. If $\epsilon_i > 1$ then, it is on the wrong side of the hyperplane.

We can control the strictness of the model and the number of misclassification by specifying the hyperparameter C. An increase in C allows the model to misclassify more observations i.e. set more ϵ_i's to a value greater than 0. Larger values of C increase the margin size.

Observations that lie directly on the margin, or on the wrong side of the margin for their class, are known as **support vectors**.

3 Support Vector Machines

3.1 Non-decision linear boundaries

There must be a mechanism to convert linear boundaries to non-linear decision boundaries as the performance of linear regression will suffer when there is non-linear relationship between the outcomes and predictors.

We can enhance the feature space by using functions of the predictors, such as quadratic and cubic terms, in order to address this non-linearity.

Example:

Instead of fitting a support vector classifier using p features:

$$X_1, X_2, \dots, X_p,$$

we could fit a support vector classifier using $2p$ features:

$$X_1, X_1^2, X_2, X_2^2 \dots, X_p^2, X_p$$

3.2 Support Vector Machine

Support vector machine (SVM) is an extension of the support vector classifier that results from enlarging the feature space in a specific way, using *kernels*.

The solution to the *support vector classifier* problem involves only the *inner products* of the observations x_i, and $x_{i'}$ (not the observations):

$$\langle x_i, x_{i'} \rangle = \sum_{j=1}^{p} x_{ij} x_{i'j}$$

The linear support vector classifier can be written as:

$$f(x) = \beta_0 + \underbrace{\sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle}_{\text{Generalization}},$$

where there are n parameters α_i, $i = 1, \dots, n$, one per training observation.

In summary, in representing the linear classifier $f(x)$, and in computing its coefficients, all we need are inner products, and the *generalization* is $K(x_i, x_{i'})$, where K is some function that we refer to as *kernel*. A *kernel* is a function that quantifies the similarity of two observations:

$$LinearKernel = K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij} x_{i'j}$$

The linear kernel essentially quantifies the similarity of a pair of observations using *Pearson* correlation.

Now, the polynomial kernel of degree d, where d is a positive integer:

$$K(x_i, x_{i'}) = (1 + \sum_{j=1}^{p} x_{ij} x_{i'j})^d$$

When the support vector classifier is combined with a non-linear kernel such as above, the resulting classifier is known as a *support vector machine*.

The radial kernel is the alternative of a possible non-linear abound:

$$K(x_i, x_{i'}) = exp(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2),$$

where $\gamma > 0$. It has a *local* behavior in the sense that only nearby training observations have an effect on the class label of a test observation, the feature space is implicit and infinite-dimensional, so we could never do the computations there anyway!.

4 SVMs (with more than two classes)

4.1 One-versus-One Classification

A one-versus-one or all-pairs approach constructs $\binom{K}{2}$ SVMs, each of which compares a pair of classes.

For example, one such one SVM might compare the kth class, coded as $+1$, to the $k^{'}$th class, coded as -1.

We classify a test observation using each of the $\binom{K}{2}$ classifiers, and we tally the number of times that the test observation is assigned to each of the K classes. The final classification is performed by assigning the test observation to the class to which it was most frequently assigned in these $\binom{K}{2}$ pairwise classifications.

4.2 One-versus-All Classifications

An alternative procedure for applying SVMs in the case of $K > 2$ classes. We fit K SVMs, each time comparing one of the K classes to the remaining $K - 1$ classes.

Let $\beta_{0k}, \beta_{1k}, \dots, \beta_{pk}$ denote the parameters that result from fitting an SVM comparing the kth class (coded as $+1$) to the others (coded as -1).

Let x^* denote a test observation. We assign the observation to the class for which $\beta_{0k} + \beta_{1k}x_1^* + \beta_{2k}x_2^* \dots, +\beta_{pk}x_p^*$ is largest, as this amounts to a high level of confidence that the test observation belongs to the kth class rather than to any of the other classes.

5 Relationship to Logistic Regression

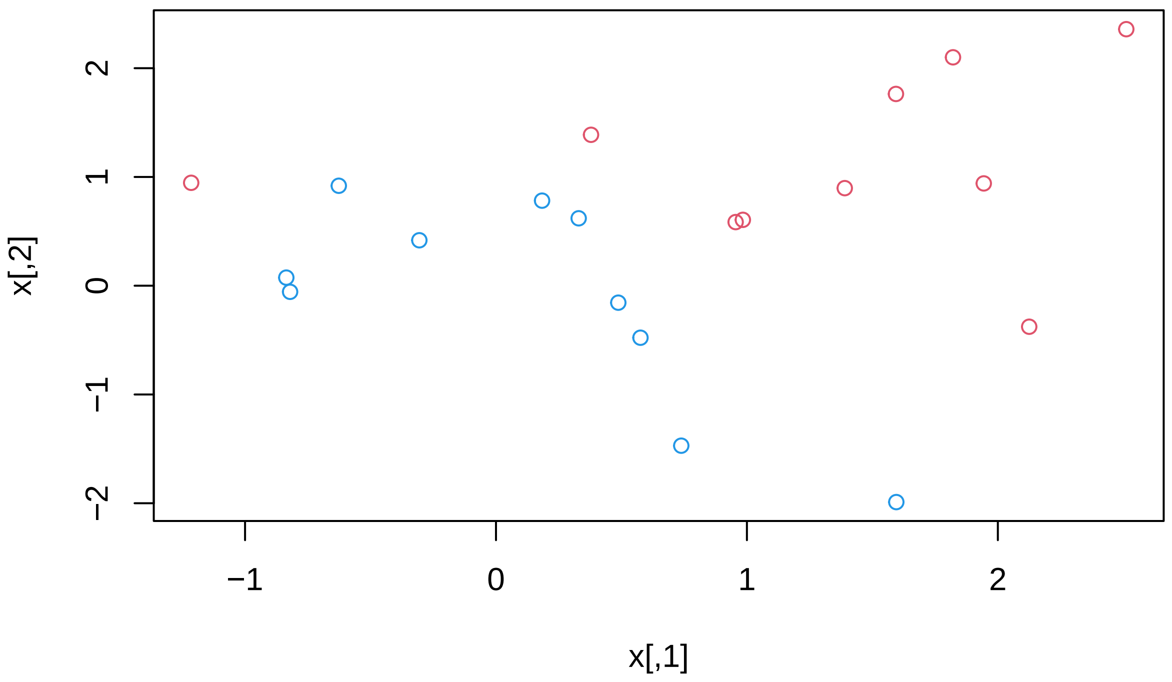
When the classes are well separated, SVMs tend to behave better than logistic regression; in more overlapping regimes, logistic regression is often preferred.

An extension of the SVM for regression (i.e. for a quantitative rather than a qualitative response), called *support vector regression*.

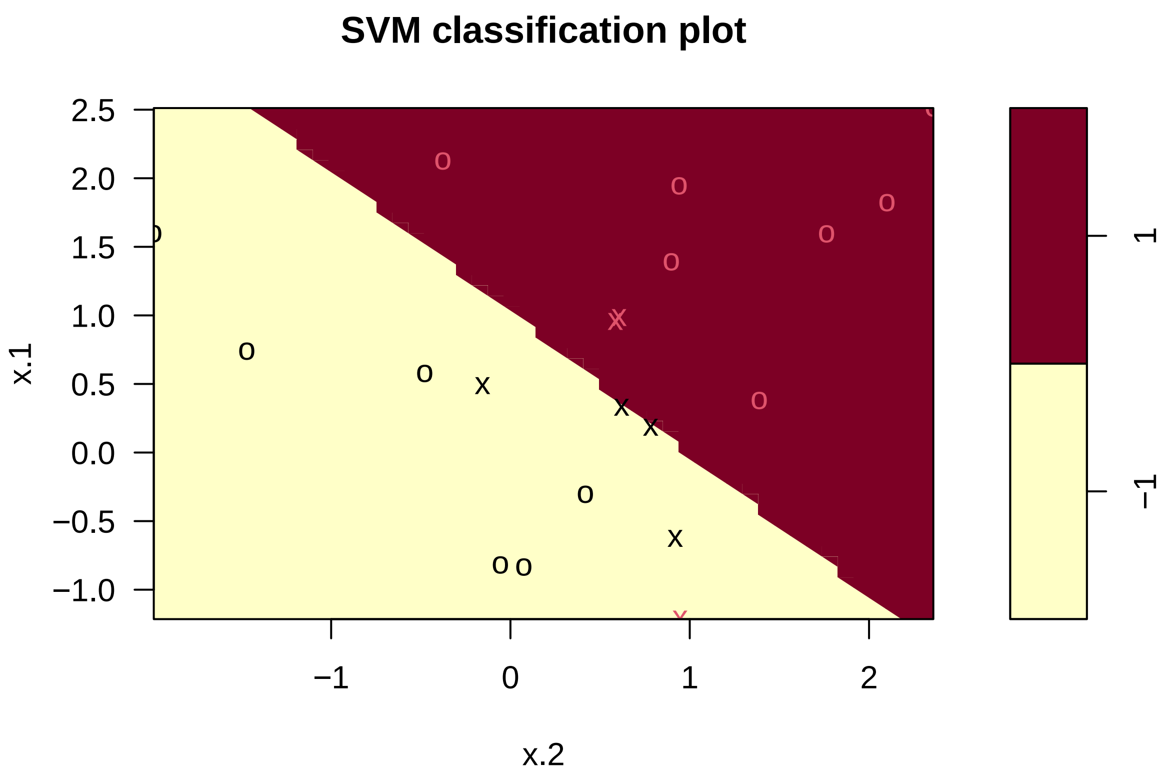
6 Lab: Support Vector Machines

6.1 Support Vector Classifier

```r
###
set.seed(1)
x <- matrix(rnorm(20 * 2), ncol = 2)
y <- c(rep(-1, 10), rep(1, 10))
x[y == 1, ] <- x[y == 1, ] + 1
plot(x, col = (3 - y))
```



```
###
dat <- data.frame(x = x, y = as.factor(y))
library(e1071)
svmfit <- svm(y ~ ., data = dat, kernel = "linear",
    cost = 10, scale = FALSE)
###
plot(svmfit, dat)
```



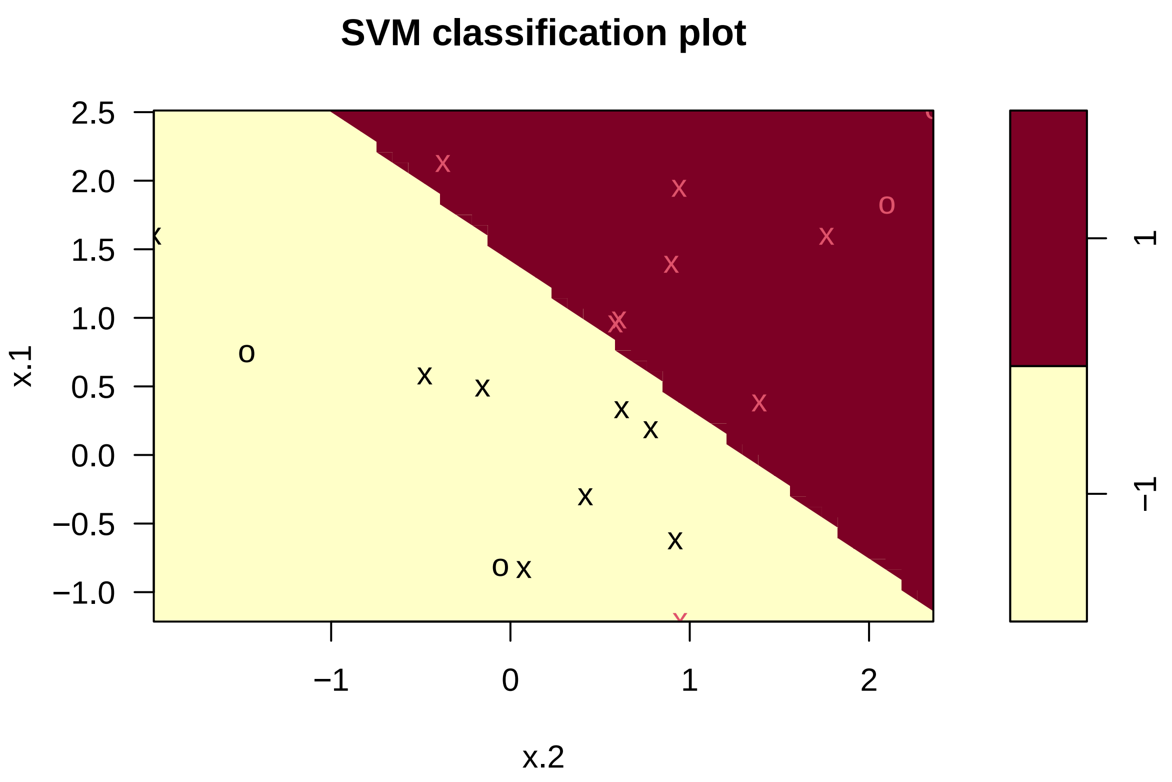
```
###
svmfit$index
```

```
## [1]  1  2  5  7 14 16 17
```

```
###
summary(svmfit)
```

```
##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##
## Number of Support Vectors:  7
##
##  ( 4 3 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

```
###
svmfit <- svm(y ~ ., data = dat, kernel = "linear",
    cost = 0.1, scale = FALSE)
plot(svmfit, dat)
```



```
svmfit$index
```

```
##  [1]  1  2  3  4  5  7  9 10 12 13 14 15 16 17 18 20
```

```
###
set.seed(1)
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)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 0.05
##
## - Detailed performance results:
##    cost error dispersion
## 1 1e-03  0.55  0.4377975
## 2 1e-02  0.55  0.4377975
## 3 1e-01  0.05  0.1581139
## 4 1e+00  0.15  0.2415229
## 5 5e+00  0.15  0.2415229
## 6 1e+01  0.15  0.2415229
## 7 1e+02  0.15  0.2415229
```

```
###
bestmod <- tune.out$best.model
summary(bestmod)
```

```
##
## 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
##
## Number of Support Vectors:  16
##
##  ( 8 8 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

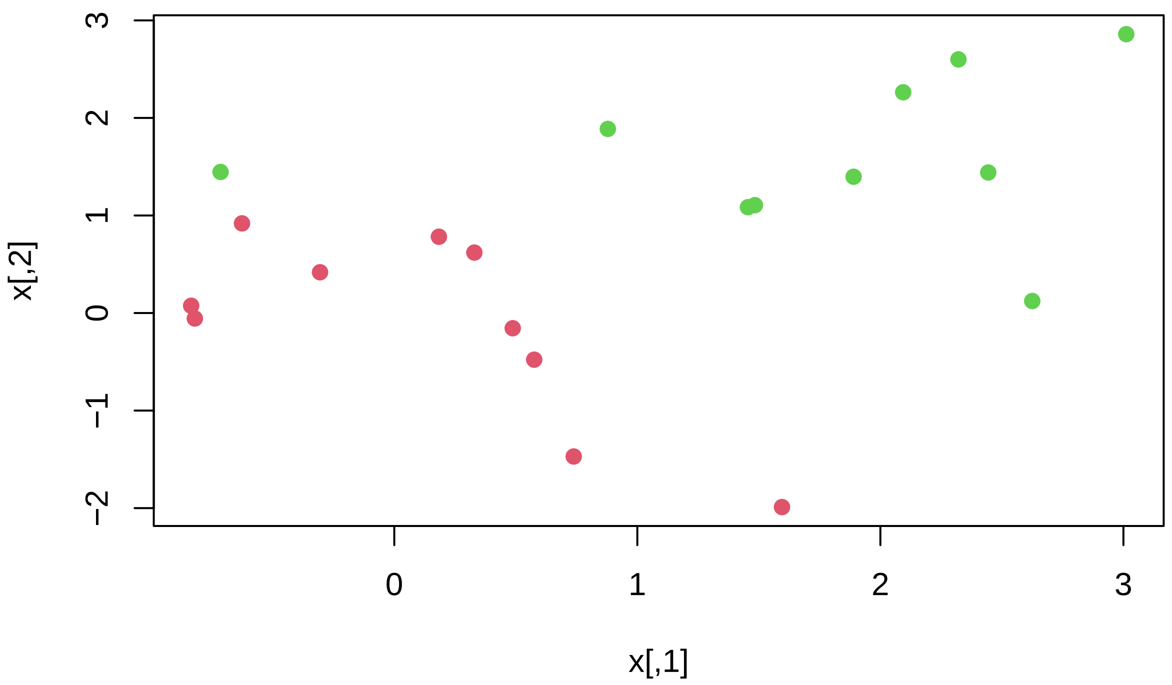
```
###
xtest <- matrix(rnorm(20 * 2), ncol = 2)
ytest <- sample(c(-1, 1), 20, rep = TRUE)
xtest[ytest == 1, ] <- xtest[ytest == 1, ] + 1
testdat <- data.frame(x = xtest, y = as.factor(ytest))
###
ypred <- predict(bestmod, testdat)
table(predict = ypred, truth = testdat$y)
```

```
##        truth
## predict -1 1
##      -1  9 1
##      1   2 8
```

```
###
svmfit <- svm(y ~ ., data = dat, kernel = "linear",
    cost = .01, scale = FALSE)
ypred <- predict(svmfit, testdat)
table(predict = ypred, truth = testdat$y)
```

```
##        truth
## predict -1  1
##      -1 11  6
##      1   0  3
```

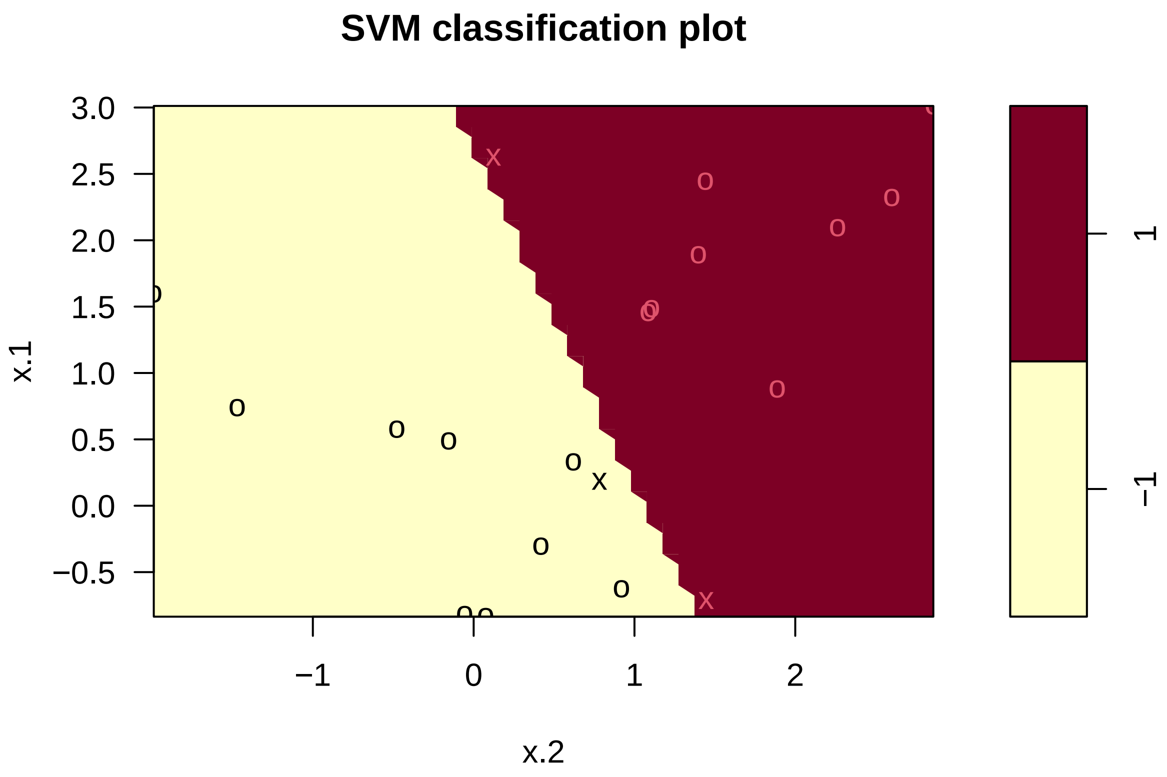
```
###
x[y == 1, ] <- x[y == 1, ] + 0.5
plot(x, col = (y + 5) / 2, pch = 19)
```



```
###
dat <- data.frame(x = x, y = as.factor(y))
svmfit <- svm(y ~ ., data = dat, kernel = "linear",
    cost = 1e5)
summary(svmfit)
```

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

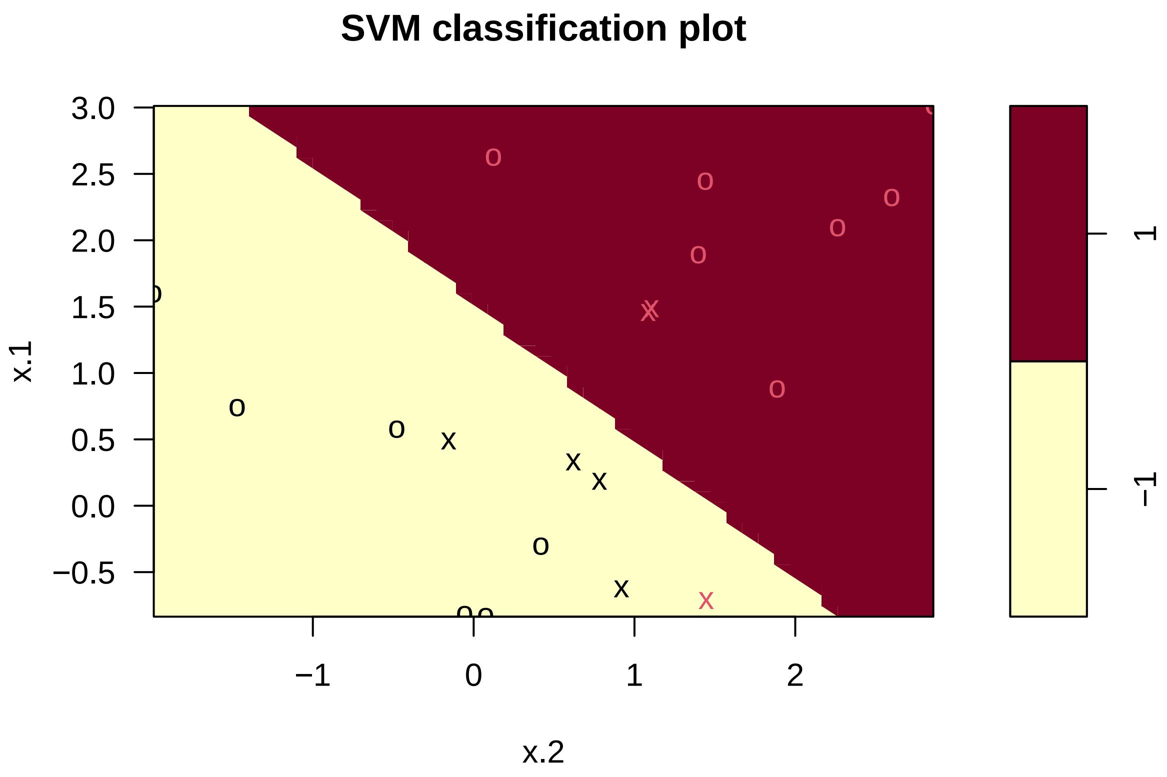
```
plot(svmfit, dat)
```



```r
###
svmfit <- svm(y ~ ., data = dat, kernel = "linear", cost = 1)
summary(svmfit)
```

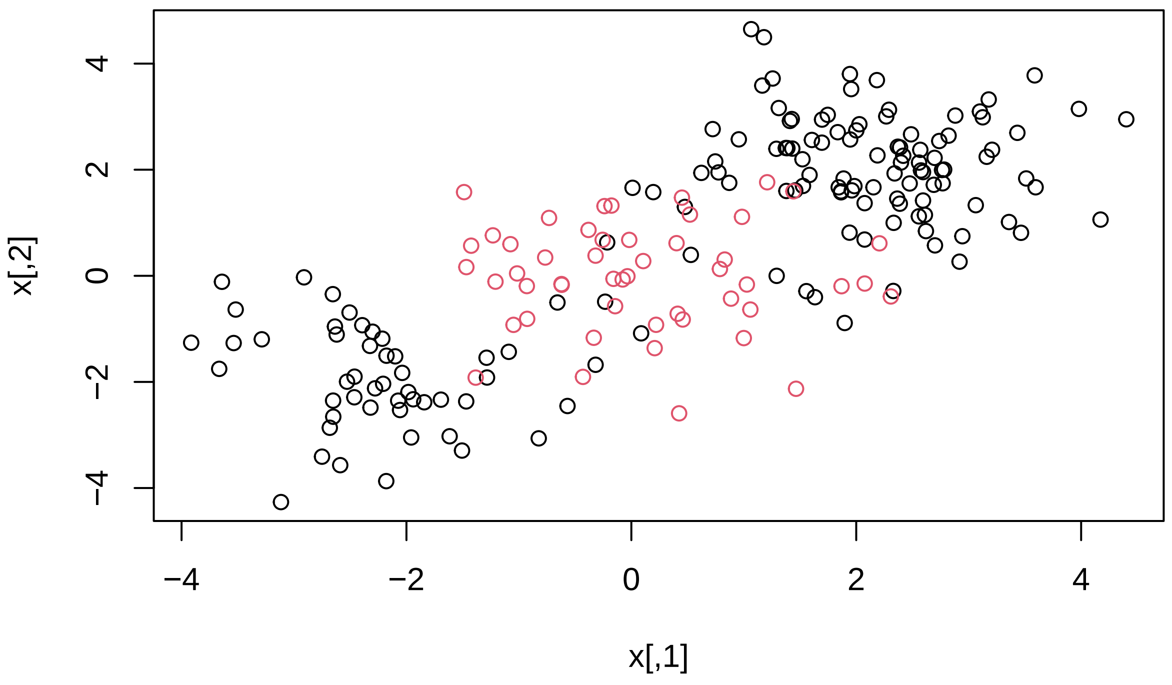
```
##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  7
##
##  ( 4 3 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

```r
plot(svmfit, dat)
```

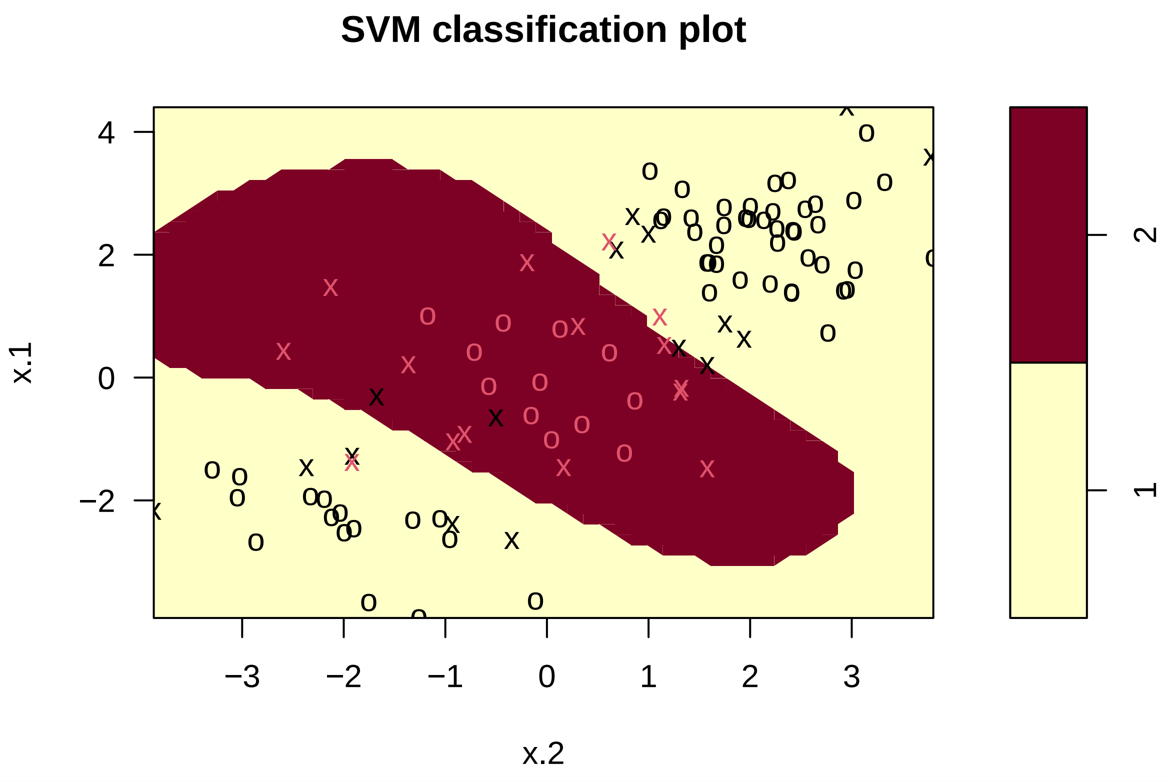


6.2 Support Vector Machine

```
###
set.seed(1)
x <- matrix(rnorm(200 * 2), ncol = 2)
x[1:100, ] <- x[1:100, ] + 2
x[101:150, ] <- x[101:150, ] - 2
y <- c(rep(1, 150), rep(2, 50))
dat <- data.frame(x = x, y = as.factor(y))
###
plot(x, col = y)
```



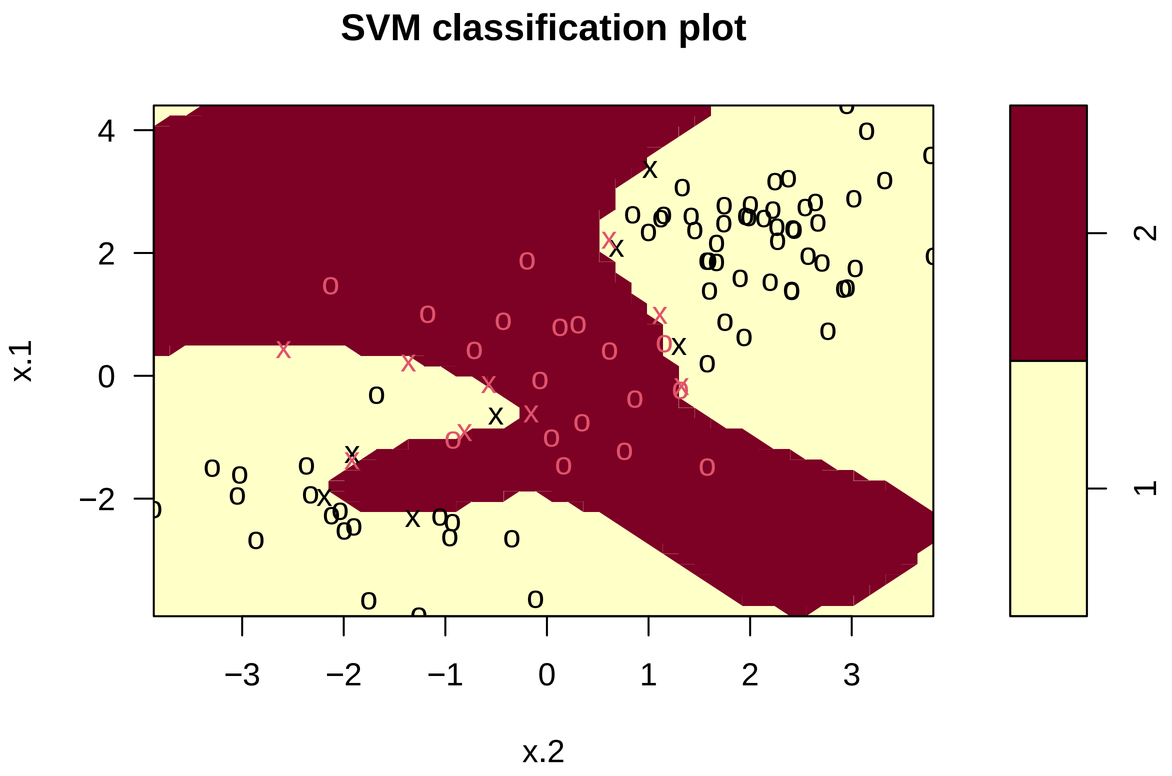
```
###
train <- sample(200, 100)
svmfit <- svm(y ~ ., data = dat[train, ], kernel = "radial",
    gamma = 1, cost = 1)
plot(svmfit, dat[train, ])
```



```
###
summary(svmfit)
```

```
##
## Call:
## svm(formula = y ~ ., data = dat[train, ], kernel = "radial", gamma = 1,
##     cost = 1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  31
##
##  ( 16 15 )
##
##
## Number of Classes:  2
##
## Levels:
##  1 2
```

```
###
svmfit <- svm(y ~ ., data = dat[train, ], kernel = "radial",
    gamma = 1, cost = 1e5)
plot(svmfit, dat[train, ])
```



```
###
set.seed(1)
tune.out <- tune(svm, y ~ ., data = dat[train, ],
    kernel = "radial",
```

```r
    ranges = list(
      cost = c(0.1, 1, 10, 100, 1000),
      gamma = c(0.5, 1, 2, 3, 4)
    )
  )
summary(tune.out)
```

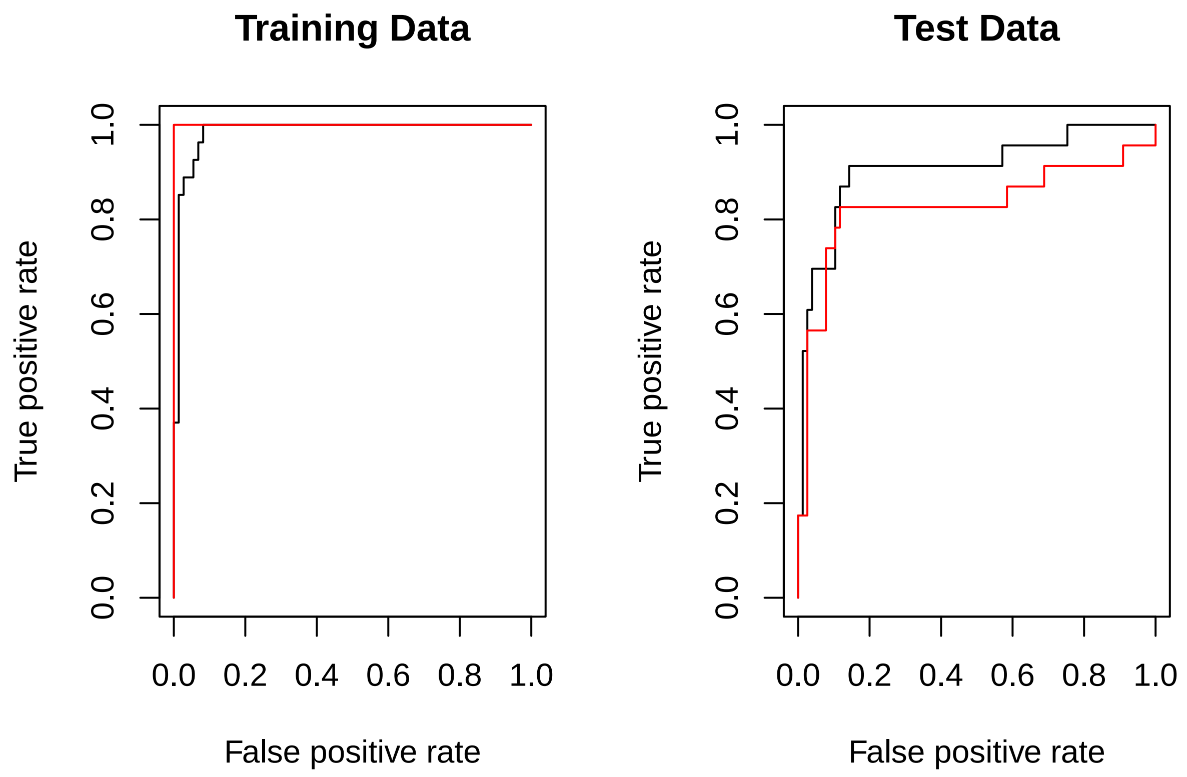
```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1   0.5
##
## - best performance: 0.07
##
## - Detailed performance results:
##     cost gamma error dispersion
## 1  1e-01   0.5  0.26 0.15776213
## 2  1e+00   0.5  0.07 0.08232726
## 3  1e+01   0.5  0.07 0.08232726
## 4  1e+02   0.5  0.14 0.15055453
## 5  1e+03   0.5  0.11 0.07378648
## 6  1e-01   1.0  0.22 0.16193277
## 7  1e+00   1.0  0.07 0.08232726
## 8  1e+01   1.0  0.09 0.07378648
## 9  1e+02   1.0  0.12 0.12292726
## 10 1e+03   1.0  0.11 0.11005049
## 11 1e-01   2.0  0.27 0.15670212
## 12 1e+00   2.0  0.07 0.08232726
## 13 1e+01   2.0  0.11 0.07378648
## 14 1e+02   2.0  0.12 0.13165612
## 15 1e+03   2.0  0.16 0.13498971
## 16 1e-01   3.0  0.27 0.15670212
## 17 1e+00   3.0  0.07 0.08232726
## 18 1e+01   3.0  0.08 0.07888106
## 19 1e+02   3.0  0.13 0.14181365
## 20 1e+03   3.0  0.15 0.13540064
## 21 1e-01   4.0  0.27 0.15670212
## 22 1e+00   4.0  0.07 0.08232726
## 23 1e+01   4.0  0.09 0.07378648
## 24 1e+02   4.0  0.13 0.14181365
## 25 1e+03   4.0  0.15 0.13540064
```

```r
###
table(
    true = dat[-train, "y"],
    pred = predict(
      tune.out$best.model, newdata = dat[-train, ]
      )
    )
```

```
##     pred
## true  1  2
##    1 67 10
##    2  2 21
```

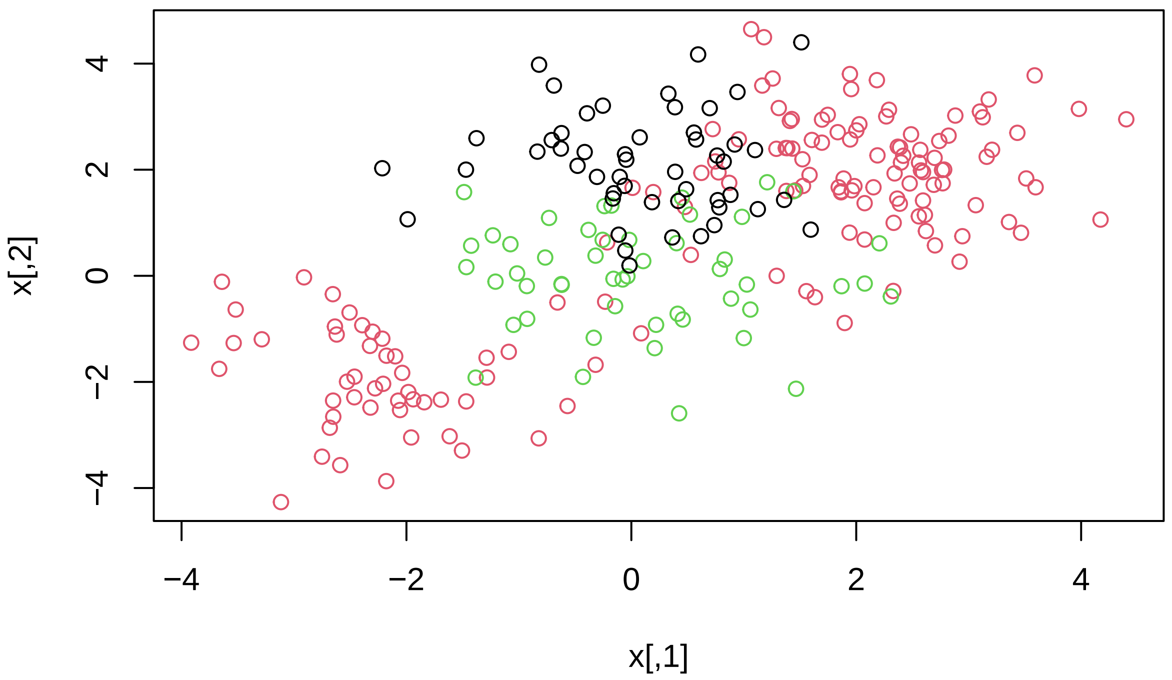
6.3 ROC Curves

```
###
library(ROCR)
rocplot <- function(pred, truth, ...) {
  predob <- prediction(pred, truth)
  perf <- performance(predob, "tpr", "fpr")
  plot(perf, ...)
}
###
svmfit.opt <- svm(y ~ ., data = dat[train, ],
    kernel = "radial", gamma = 2, cost = 1,
    decision.values = T)
fitted <- attributes(
    predict(svmfit.opt, dat[train, ], decision.values = TRUE)
  )$decision.values
###
par(mfrow = c(1, 2))
rocplot(-fitted, dat[train, "y"], main = "Training Data")
###
svmfit.flex <- svm(y ~ ., data = dat[train, ],
    kernel = "radial", gamma = 50, cost = 1,
    decision.values = T)
fitted <- attributes(
    predict(svmfit.flex, dat[train, ], decision.values = T)
  )$decision.values
rocplot(-fitted, dat[train, "y"], add = T, col = "red")
###
fitted <- attributes(
    predict(svmfit.opt, dat[-train, ], decision.values = T)
  )$decision.values
rocplot(-fitted, dat[-train, "y"], main = "Test Data")
fitted <- attributes(
    predict(svmfit.flex, dat[-train, ], decision.values = T)
  )$decision.values
rocplot(-fitted, dat[-train, "y"], add = T, col = "red")
```

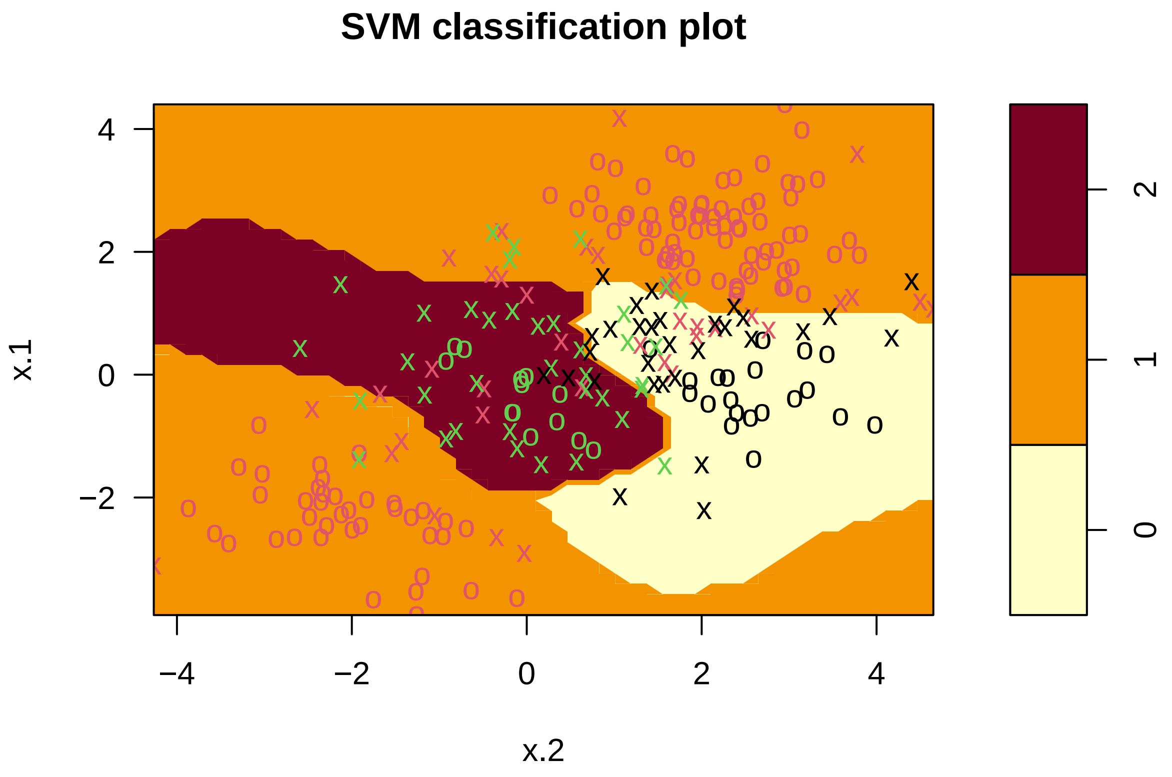


6.4 SVM with Multiple Classes

```
###
set.seed(1)
x <- rbind(x, matrix(rnorm(50 * 2), ncol = 2))
y <- c(y, rep(0, 50))
x[y == 0, 2] <- x[y == 0, 2] + 2
dat <- data.frame(x = x, y = as.factor(y))
par(mfrow = c(1, 1))
plot(x, col = (y + 1))
```



```
###
svmfit <- svm(y ~ ., data = dat, kernel = "radial",
    cost = 10, gamma = 1)
plot(svmfit, dat)
```



6.5 Application to Gene Expression Data

```
library(ISLR2)
names(Khan)
```

```
## [1] "xtrain" "xtest"  "ytrain" "ytest"
```

```
dim(Khan$xtrain)
```

```
## [1]   63 2308
```

```
dim(Khan$xtest)
```

```
## [1]   20 2308
```

```
length(Khan$ytrain)
```

```
## [1] 63
```

```
length(Khan$ytest)
```

```
## [1] 20
```

```
table(Khan$ytrain)
```

```
## 
##  1  2  3  4 
##  8 23 12 20
```

```
table(Khan$ytest)
```

```
## 
## 1 2 3 4 
## 3 6 6 5
```

```
dat <- data.frame(
    x = Khan$xtrain,
    y = as.factor(Khan$ytrain)
  )
out <- svm(y ~ ., data = dat, kernel = "linear",
    cost = 10)
summary(out)
```

```
## 
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10)
## 
## 
## Parameters:
##    SVM-Type:  C-classification 
##  SVM-Kernel:  linear 
##        cost:  10 
## 
## Number of Support Vectors:  58
## 
##  ( 20 20 11 7 )
## 
## 
## Number of Classes:  4 
## 
```

```
## Levels:
##  1 2 3 4
```

```
table(out$fitted, dat$y)
```

```
##
##      1  2  3  4
##   1  8  0  0  0
##   2  0 23  0  0
##   3  0  0 12  0
##   4  0  0  0 20
```

```
dat.te <- data.frame(
    x = Khan$xtest,
    y = as.factor(Khan$ytest))
pred.te <- predict(out, newdata = dat.te)
table(pred.te, dat.te$y)
```

```
##
## pred.te 1 2 3 4
##       1 3 0 0 0
##       2 0 6 2 0
##       3 0 0 4 0
##       4 0 0 0 5
```

7 References

(James et al. 2013)

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2013. "Support Vector Machines." In, 337–72. Springer New York. https://doi.org/10.1007/978-1-4614-7138-7_9.