Support Vector Classifier

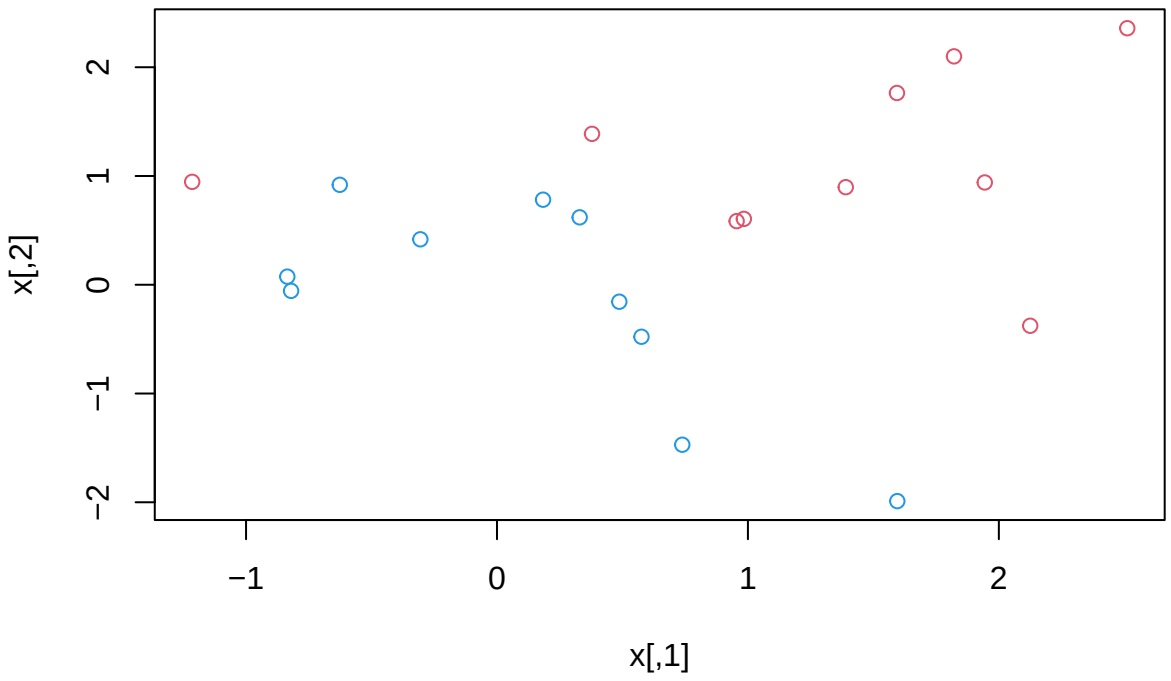
ACM

In this example we're going to look at Support Vector Classifier's with some simulated data.

```
library(printr)
```

```
## Registered S3 method overwritten by 'printr':
##   method                from
##   knit_print.data.frame rmarkdown
```

```
set.seed(1)
x <- matrix(rnorm(20*2),20,2)
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)
?svm
```

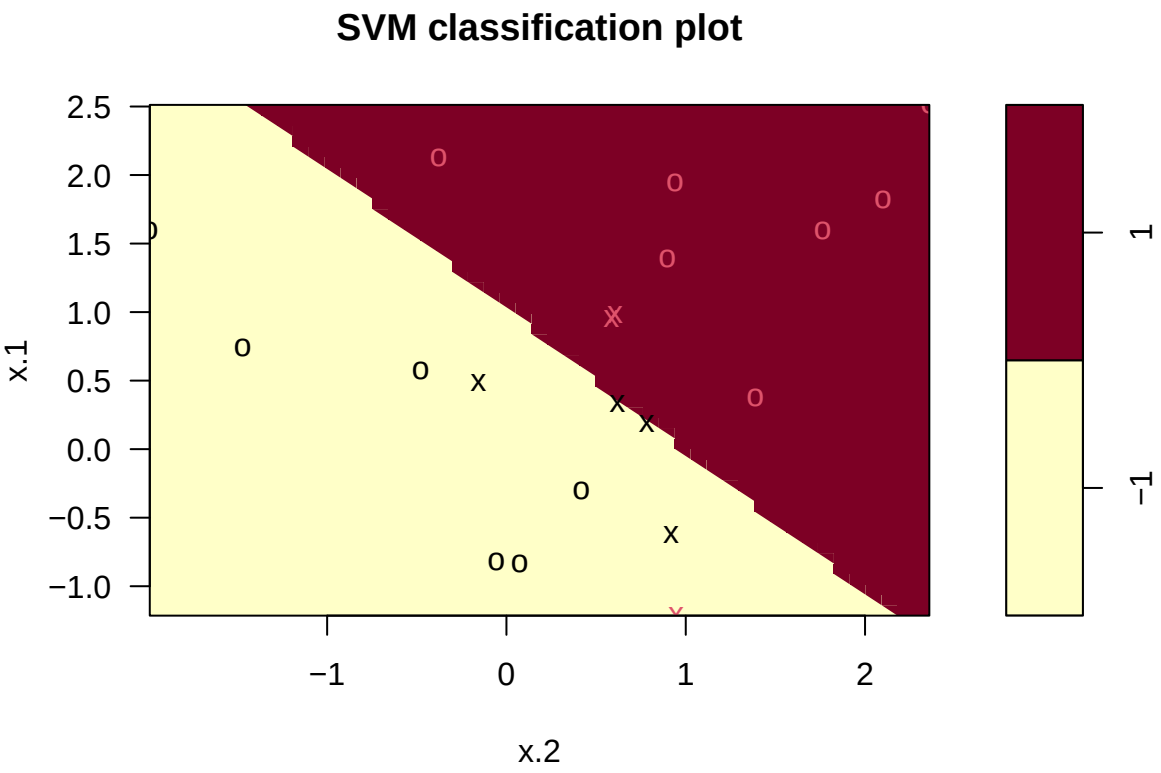
```
## Support Vector Machines
##
## Description:
##
##      'svm' is used to train a support vector machine. It can be used to
##      carry out general regression and classification (of nu and
```

```
##      epsilon-type), as well as density-estimation. A formula interface
##      is provided.
##
## Usage:
##
##      ## S3 method for class 'formula'
##      svm(formula, data = NULL, ..., subset, na.action =
##      na.omit, scale = TRUE)
##      ## Default S3 method:
##      svm(x, y = NULL, scale = TRUE, type = NULL, kernel =
##      "radial", degree = 3, gamma = if (is.vector(x)) 1 else 1 / ncol(x),
##      coef0 = 0, cost = 1, nu = 0.5,
##      class.weights = NULL, cachesize = 40, tolerance = 0.001, epsilon = 0.1,
##      shrinking = TRUE, cross = 0, probability = FALSE, fitted = TRUE,
##      ..., subset, na.action = na.omit)
##
## Arguments:
##
##  formula: a symbolic description of the model to be fit.
##
##     data: an optional data frame containing the variables in the model.
##           By default the variables are taken from the environment which
##           'svm' is called from.
##
##        x: a data matrix, a vector, or a sparse matrix (object of class
##           'Matrix' provided by the 'Matrix' package, or of class
##           'matrix.csr' provided by the 'SparseM' package, or of class
##           'simple_triplet_matrix' provided by the 'slam' package).
##
##        y: a response vector with one label for each row/component of
##           'x'. Can be either a factor (for classification tasks) or a
##           numeric vector (for regression).
##
##    scale: A logical vector indicating the variables to be scaled. If
##           'scale' is of length 1, the value is recycled as many times
##           as needed.  Per default, data are scaled internally (both 'x'
##           and 'y' variables) to zero mean and unit variance. The center
##           and scale values are returned and used for later predictions.
##
##     type: 'svm' can be used as a classification machine, as a
##           regression machine, or for novelty detection.  Depending of
##           whether 'y' is a factor or not, the default setting for
##           'type' is 'C-classification' or 'eps-regression',
##           respectively, but may be overwritten by setting an explicit
##           value.
##           Valid options are:
##
##             • 'C-classification'
##
##             • 'nu-classification'
##
##             • 'one-classification' (for novelty detection)
##
##             • 'eps-regression'
```

```
##
##             • 'nu-regression'
##
##   kernel: the kernel used in training and predicting. You might
##           consider changing some of the following parameters, depending
##           on the kernel type.
##
##           linear: u'*v
##
##           polynomial: (gamma*u'*v + coef0)^degree
##
##           radial basis: exp(-gamma*|u-v|^2)
##
##           sigmoid: tanh(gamma*u'*v + coef0)
##
##   degree: parameter needed for kernel of type 'polynomial' (default: 3)
##
##    gamma: parameter needed for all kernels except 'linear' (default:
##           1/(data dimension))
##
##    coef0: parameter needed for kernels of type 'polynomial' and
##           'sigmoid' (default: 0)
##
##     cost: cost of constraints violation (default: 1)-it is the
##           'C'-constant of the regularization term in the Lagrange
##           formulation.
##
##       nu: parameter needed for 'nu-classification', 'nu-regression',
##           and 'one-classification'
##
## class.weights: a named vector of weights for the different classes,
##           used for asymmetric class sizes. Not all factor levels have
##           to be supplied (default weight: 1). All components have to be
##           named. Specifying '"inverse"' will choose the weights
##           _inversely_ proportional to the class distribution.
##
## cachesize: cache memory in MB (default 40)
##
## tolerance: tolerance of termination criterion (default: 0.001)
##
##  epsilon: epsilon in the insensitive-loss function (default: 0.1)
##
## shrinking: option whether to use the shrinking-heuristics (default:
##           'TRUE')
##
##    cross: if a integer value k>0 is specified, a k-fold cross
##           validation on the training data is performed to assess the
##           quality of the model: the accuracy rate for classification
##           and the Mean Squared Error for regression
##
##   fitted: logical indicating whether the fitted values should be
##           computed and included in the model or not (default: 'TRUE')
##
## probability: logical indicating whether the model should allow for
```

```
##           probability predictions.
##
##      ...: additional parameters for the low level fitting function
##           'svm.default'
##
##   subset: An index vector specifying the cases to be used in the
##           training sample.  (NOTE: If given, this argument must be
##           named.)
##
## na.action: A function to specify the action to be taken if 'NA's are
##           found. The default action is 'na.omit', which leads to
##           rejection of cases with missing values on any required
##           variable. An alternative is 'na.fail', which causes an error
##           if 'NA' cases are found. (NOTE: If given, this argument must
##           be named.)
```

```
svmfit = svm(y~.,data=dat,kernel="linear",cost=10,scale=FALSE)
plot(svmfit,dat)
```



Support vectors are on the margins or over the margins (the "X" points).

```
svmfit$index
```

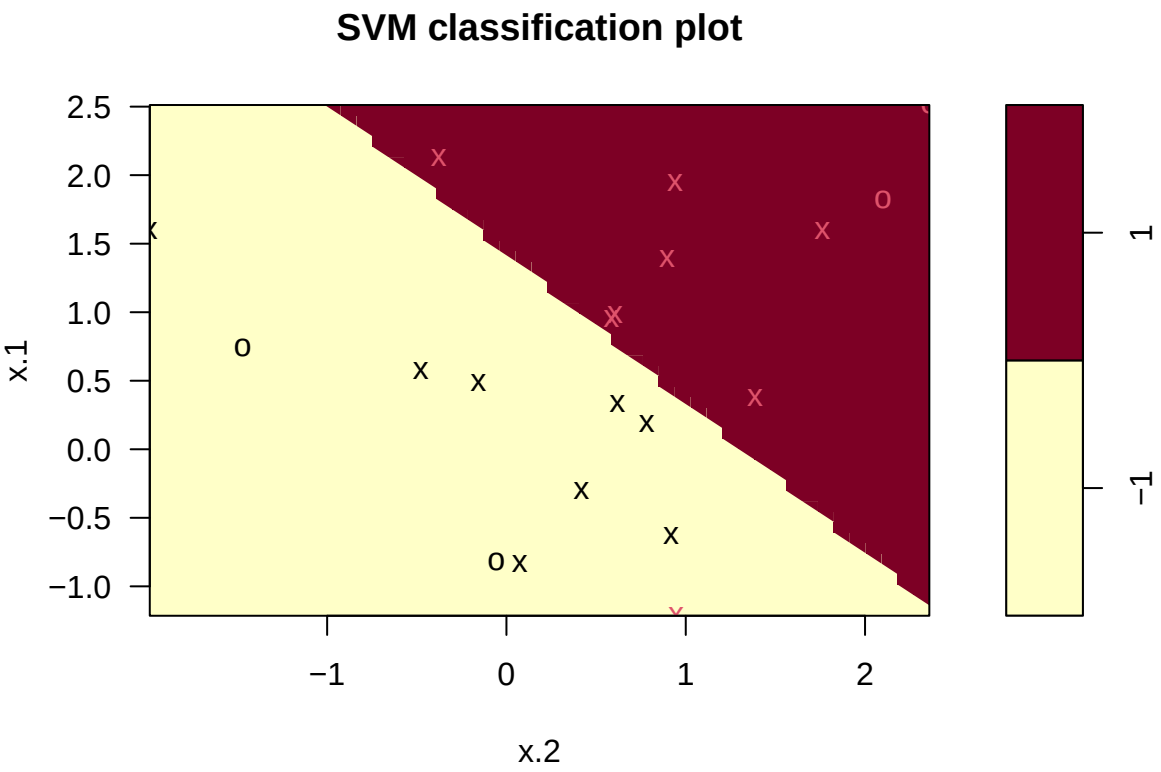
```
## [1]  1  2  5  7 14 16 17
```

```
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)
```



For the cost=0.1 almost everything is a support point (very few "O" points).

```
svmfit$index
```

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

```
summary(svmfit)
```

```
##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 0.1, scale = FALSE)
##
##
## 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
```

More support points for lower costs.

Now we'll check this out for a number of costs and see which works best. For this we'll use 10-fold cross validation (which is the default for the function below), which is why we set a seed.

```
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)
```

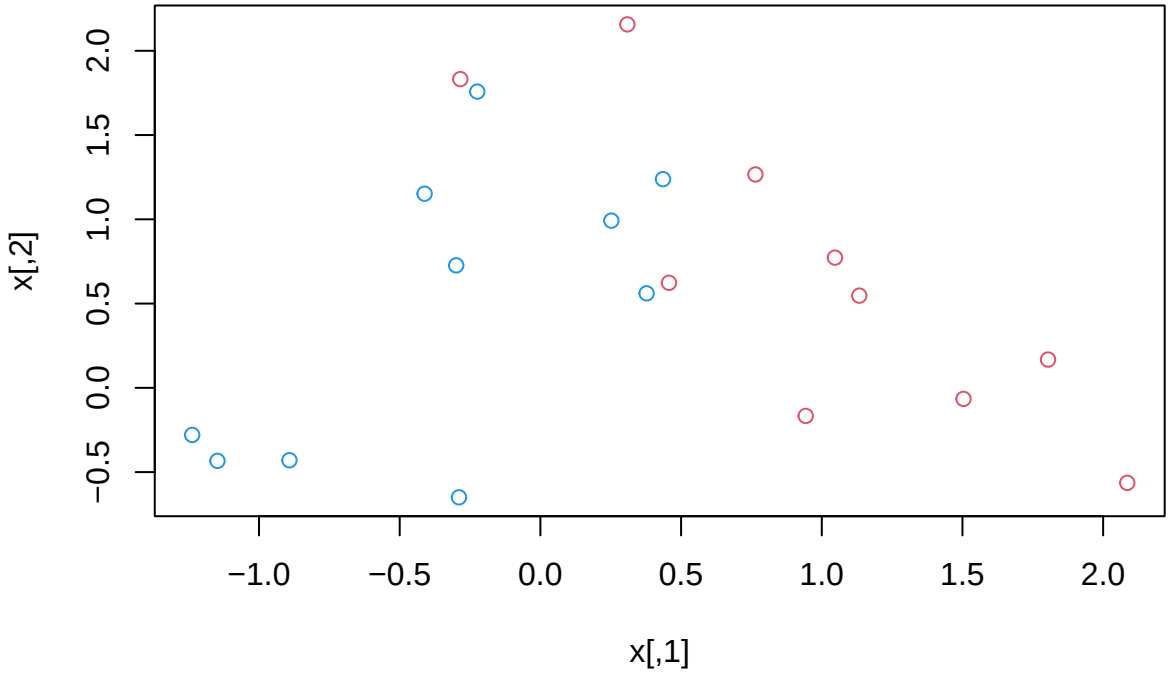
```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 0.05
##
## - Detailed performance results:
##    cost error dispersion
## 1 1e-03  0.55  0.4377975
## 2 1e-02  0.55  0.4377975
## 3 1e-01  0.05  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
```

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




```
testdat <- data.frame(x=x,y=as.factor(y))
ypred = predict(bestmod,testdat)
ypred
```

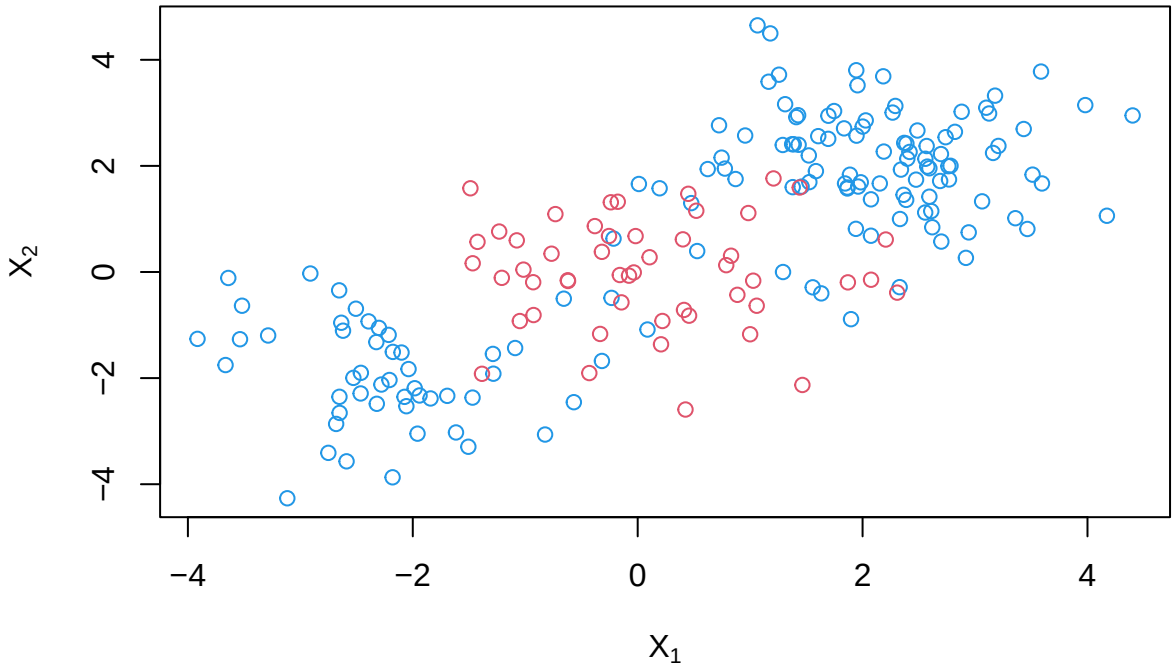
```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
## -1 -1 -1 -1 -1 -1  1 -1  1 -1  1  1 -1  1  1  1  1  1  1 -1
## Levels: -1 1
```

```
table(ypred,testdat$y)
```

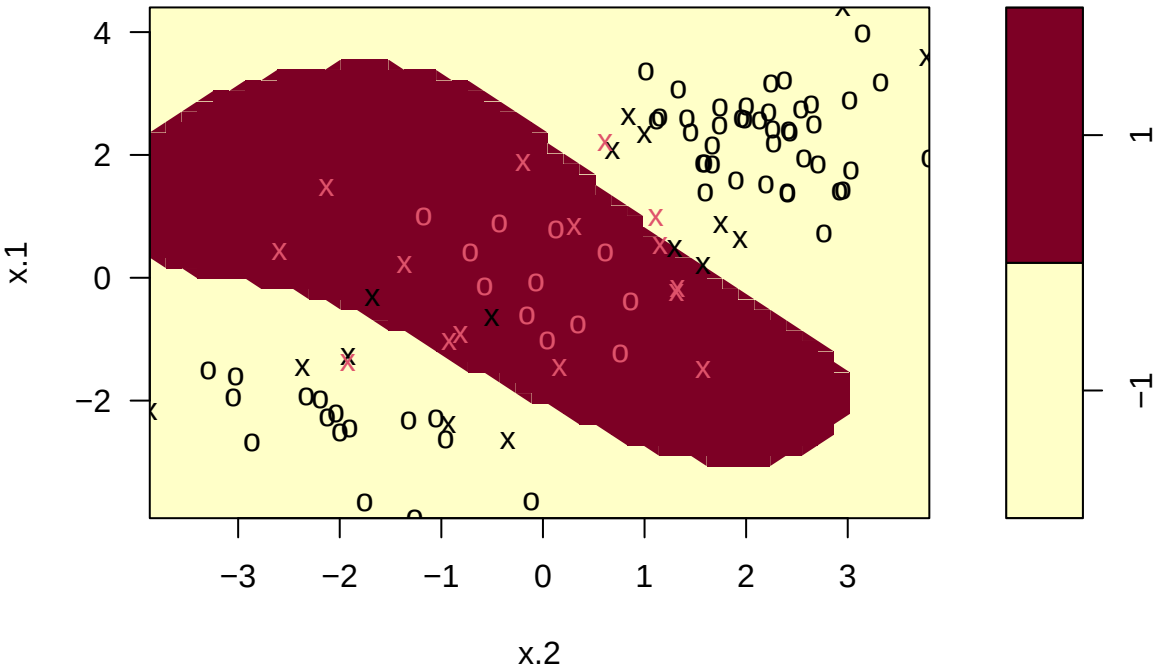
ypred/	-1	1
-1	8	2
1	2	8

Now let's look at some non-linear data.

```
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(1,50))
dat <- data.frame(x=x,y=as.factor(y))
plot(x,col=(3-y),xlab=expression(X[1]),ylab=expression(X[2]))
```



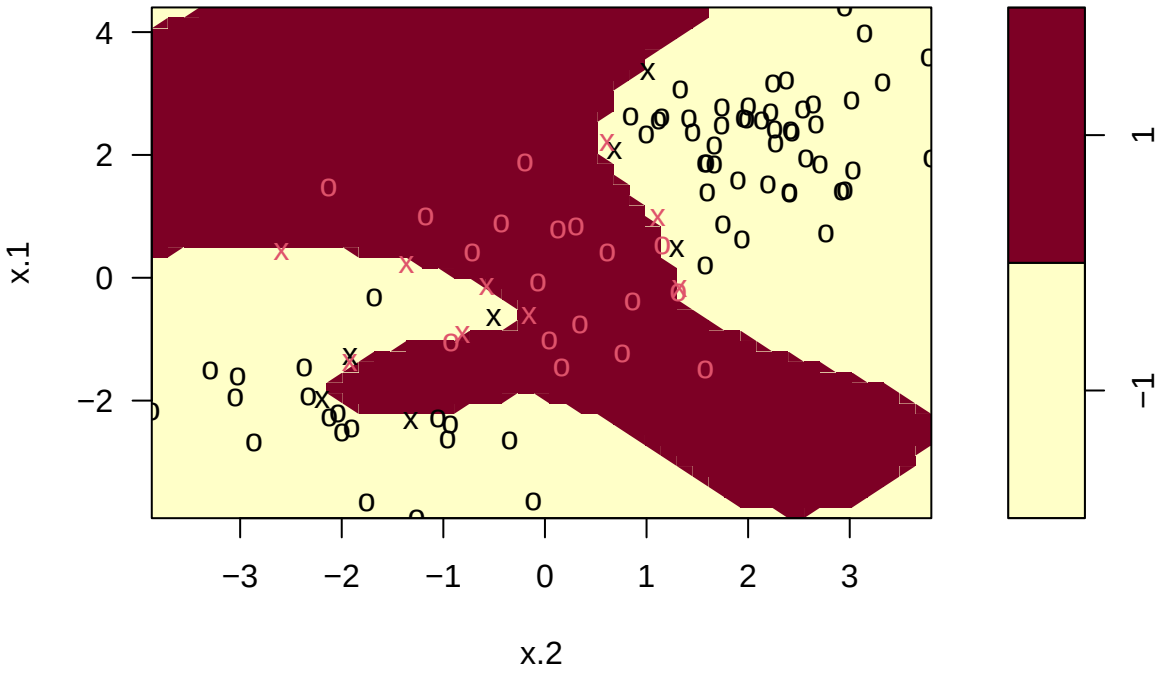

```
train=sample(200,100)
svmfit = svm(y~.,data=dat[train,],kernel="radial",cost=1,gamma=1)
par(mfrow=c(1,2))
plot(svmfit,dat[train,])
```




```
summary(svmfit)
```

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

```
svmfit2 = svm(y~.,data=dat[train,],kernel="radial",cost=1e4,gamma=1)
plot(svmfit2,dat[train,])
```




```
tune.out <- tune(svm, y~., data=dat[train,], kernel="radial",
                 ranges=list(cost=c(0.001,0.01,0.1,1,5,10,100),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: 0.06
##
## - Detailed performance results:
##     cost gamma error dispersion
## 1  1e-03   0.5  0.27 0.13374935
## 2  1e-02   0.5  0.27 0.13374935
## 3  1e-01   0.5  0.27 0.14944341
## 4  1e+00   0.5  0.06 0.08432740
## 5  5e+00   0.5  0.07 0.08232726
## 6  1e+01   0.5  0.07 0.08232726
## 7  1e+02   0.5  0.10 0.09428090
## 8  1e-03   1.0  0.27 0.13374935
## 9  1e-02   1.0  0.27 0.13374935
## 10 1e-01   1.0  0.18 0.16865481
## 11 1e+00   1.0  0.07 0.09486833
## 12 5e+00   1.0  0.08 0.10327956
## 13 1e+01   1.0  0.09 0.09944289
```

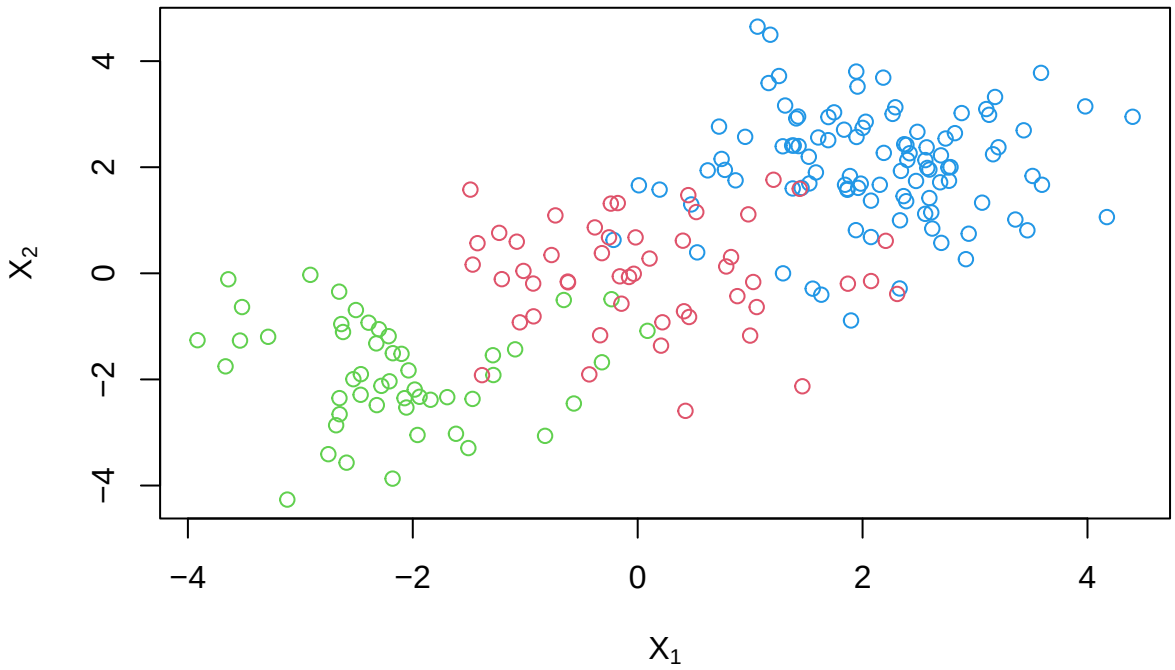
```
## 14 1e+02   1.0  0.10 0.08164966
## 15 1e-03   2.0  0.27 0.13374935
## 16 1e-02   2.0  0.27 0.13374935
## 17 1e-01   2.0  0.26 0.15776213
## 18 1e+00   2.0  0.08 0.11352924
## 19 5e+00   2.0  0.11 0.11972190
## 20 1e+01   2.0  0.12 0.12292726
## 21 1e+02   2.0  0.12 0.13165612
## 22 1e-03   3.0  0.27 0.13374935
## 23 1e-02   3.0  0.27 0.13374935
## 24 1e-01   3.0  0.27 0.13374935
## 25 1e+00   3.0  0.09 0.11005049
## 26 5e+00   3.0  0.12 0.12292726
## 27 1e+01   3.0  0.10 0.13333333
## 28 1e+02   3.0  0.13 0.10593499
## 29 1e-03   4.0  0.27 0.13374935
## 30 1e-02   4.0  0.27 0.13374935
## 31 1e-01   4.0  0.27 0.13374935
## 32 1e+00   4.0  0.09 0.12866839
## 33 5e+00   4.0  0.10 0.13333333
## 34 1e+01   4.0  0.10 0.13333333
## 35 1e+02   4.0  0.15 0.13540064
```

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

true/pred	-1	1
-1	67	10
1	2	21

Now we'll fit an SVM to some multiclass data.

```
set.seed(1)
y[101:150] <- 0
par(mfrow=c(1,1))
plot(x,col=(3-y),xlab=expression(X[1]),ylab=expression(X[2]))
```




```
dat <- data.frame(x=x,y=as.factor(y)) #the as.factor is important so we don't do regression.
svmfit = svm(y~.,data=dat,kernel="radial",cost=10,gamma=1)
plot(svmfit,dat)
```