Assignment 7 - Support Vector Classifiers and Machines

Conceptual

1. What is the intution behind SVMs and how do they work?

---*Your answer here*---

2. Are SVMs always robust regarding overfitting and noisy data? Discuss your answer considering aspects such as the choice of kernel and the degree of noise in the data.

---*Your answer here*---

Practical

Overview of the steps

1. Generate data and get an overview of the data
2. Learn and assess an support vector (soft margin) classifier
3. Learn and assess an SVM classifier
4. Learn and assess an SVM classifier for multiple classes
5. Apply SVM to Gene Expression Data

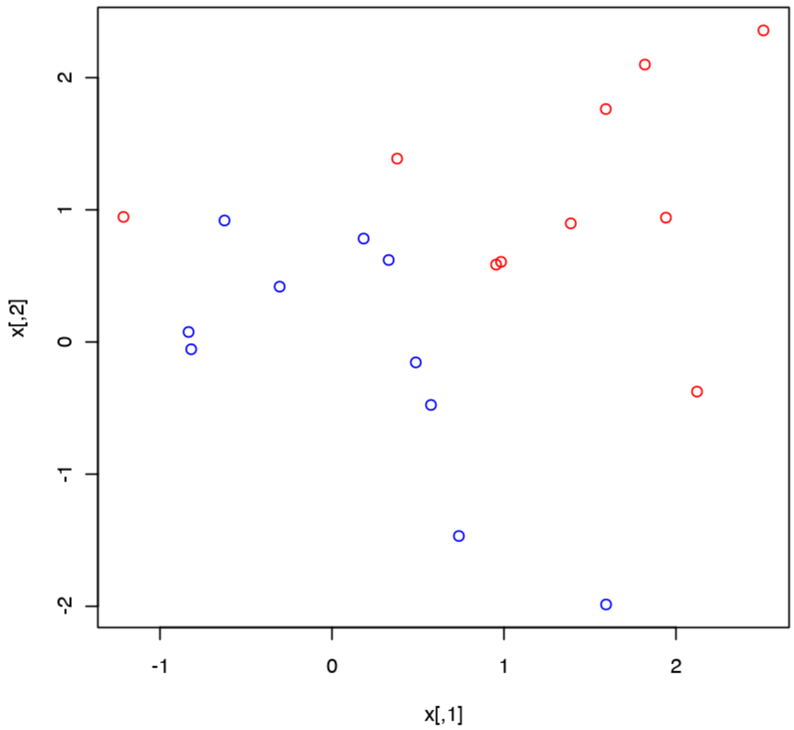
Steps in detail

Generate data and get an overview of the data

Generate the observations belonging to two classes.

Therefore, we use `rnorm` that generates a vector of `n=20*2` normally distributed random numbers. We split them into two columns in a predictor matrix `x` corresponding to two predictors and assign two classes in a response vector `y` : `-1` to the first ten observations and `1` to the last ten observations. Then we plot the data

In [1]:
```
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))
```



In [2]:
```
print(x)
print(y)
```

```
            [,1]        [,2]
 [1,] -0.6264538  0.91897737
 [2,]  0.1836433  0.78213630
 [3,] -0.8356286  0.07456498
 [4,]  1.5952808 -1.98935170
 [5,]  0.3295078  0.61982575
 [6,] -0.8204684 -0.05612874
 [7,]  0.4874291 -0.15579551
 [8,]  0.7383247 -1.47075238
 [9,]  0.5757814 -0.47815006
[10,] -0.3053884  0.41794156
[11,]  2.5117812  2.35867955
[12,]  1.3898432  0.89721227
[13,]  0.3787594  1.38767161
[14,] -1.2146999  0.94619496
[15,]  2.1249309 -0.37705956
[16,]  0.9550664  0.58500544
[17,]  0.9838097  0.60571005
[18,]  1.9438362  0.94068660
[19,]  1.8212212  2.10002537
[20,]  1.5939013  1.76317575
 [1] -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  1  1  1  1  1  1  1  1  1  1
```

Check visually whether the classes are linearly separable. They are not.

Learn and assess a support vector (soft margine) classifier

Install the necessary library.

In [3]:
```
install.packages("e1071")
library(e1071)
```

```
The downloaded binary packages are in
	/var/folders/ct/4pcck8t94sdfc73rhymq4t140000gp/T//RtmpIFcGm
6/downloaded_packages
```

Fit the support vector classifier.

In `R`, we need to encode the class `as.factor`, i.e., as 'category' or 'enumerated type', and put predictors and response in a data frame. The argument `scale=FALSE` tells the `svm()` function not to scale each feature to have mean zero or standard deviation one; depending on the application, one might prefer to use `scale=TRUE`.

A `cost` argument specifies the 'cost' of a violation to the margin and it works inverse to the violation 'budget' C discussed in the lecture. When the violation `cost` argument is small, then the margins will be wide and many support vectors will be on the margin or will violate the margin. When the `cost` argument is large, then the margins will be narrow and there will be few support vectors on the margin or violating the margin.

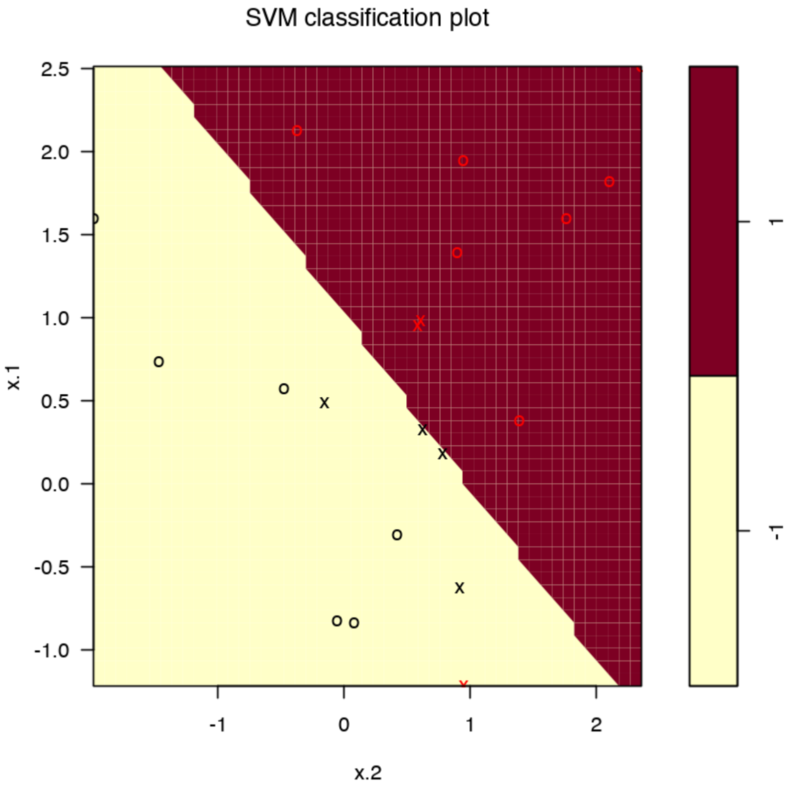
In [4]:
```
dat=data.frame(x=x, y=as.factor(y))
print(dat)
```

```
          x.1         x.2  y
1  -0.6264538  0.91897737 -1
2   0.1836433  0.78213630 -1
3  -0.8356286  0.07456498 -1
4   1.5952808 -1.98935170 -1
5   0.3295078  0.61982575 -1
6  -0.8204684 -0.05612874 -1
7   0.4874291 -0.15579551 -1
8   0.7383247 -1.47075238 -1
9   0.5757814 -0.47815006 -1
10 -0.3053884  0.41794156 -1
11  2.5117812  2.35867955  1
12  1.3898432  0.89721227  1
13  0.3787594  1.38767161  1
14 -1.2146999  0.94619496  1
15  2.1249309 -0.37705956  1
16  0.9550664  0.58500544  1
17  0.9838097  0.60571005  1
18  1.9438362  0.94068660  1
19  1.8212212  2.10002537  1
20  1.5939013  1.76317575  1
```

In [5]:
```
svmfit=svm(y~., data=dat, kernel="linear", cost=10, scale=FALSE)
```

Plot the support vector classifier obtained.

In [6]:
```
1 plot(svmfit,dat)
```



The support vectors are plotted as crosses and the remaining observations are plotted as circles; we see here that there are seven support vectors.

Determine their identities (row numbers in the data matrix).

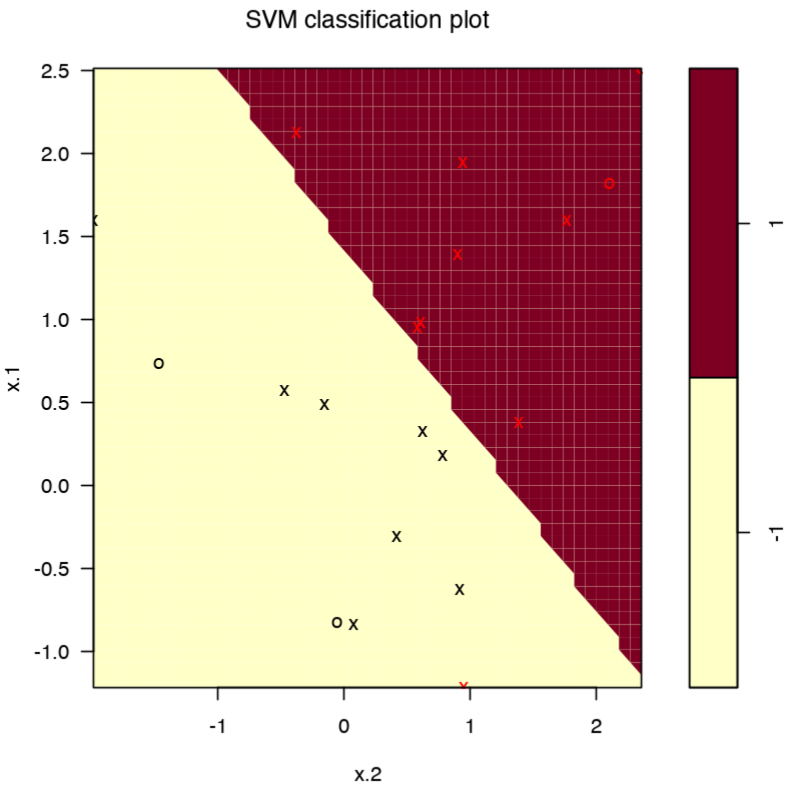
In [7]:
```
1 svmfit$index
```

1 2 5 7 14 16 17

What if we instead used a smaller value of the cost parameter?

In [8]:
```
svmfit=svm(y~., data=dat, kernel="linear", cost=0.1, scale=FALSE)
plot(svmfit,dat)
svmfit$index
```

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



Interprete the results and the variance-bias tradeoff. *Your interpretation of the results goes here!*

Perform cross-validation to determine the `cost` parameter.

The `e1071` library of `R` includes a function, `tune()` , to perform cross- validation. By default, `tune()` performs ten-fold cross-validation on a set of models of interest. Here, we want to compare SVMs with a linear kernel, using a range of values of the `cost` parameter.

In [9]:
```
set.seed(1)
tune.out=tune(svm,y~.,data=dat,kernel="linear",ranges=list(cost=c(
```

Access the cross-validation errors for each of these models.

In [10]:
```
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
```

Interprete the results. *Your interpretation of the results goes here!*

Capture the best model.

In [11]:
```
bestmod=tune.out$best.model
summary(bestmod)
```

```
Call:
best.tune(method = svm, train.x = y ~ ., data = dat, ranges = list(c
ost = 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
```

Generate test data (a sample of the same distribution) as before.

In [12]:
```
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))
```

Predict the class labels of these test observations. Use the best model obtained through cross-validation in order to make predictions.

In [13]:
```
ypred=predict(bestmod ,testdat)
table(predict=ypred, truth=testdat$y)
```

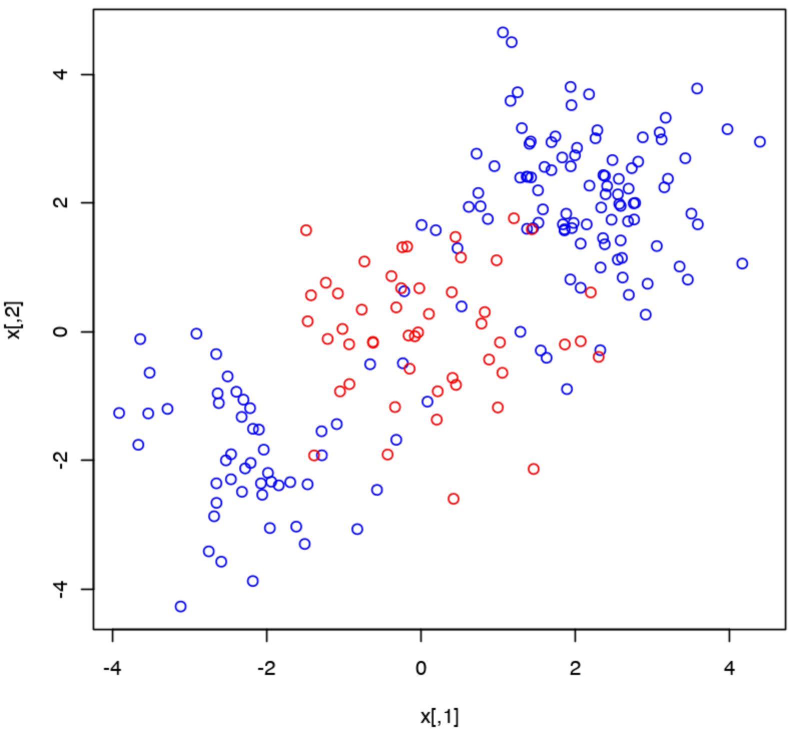
```
       truth
predict -1 1
     -1  9 1
     1   2 8
```

Interprete the results. *Your interpretation of the results goes here!*

Learn and assess an SVM classifier

First, generate some data with a non-linear class boundary, as before. Generate `200` instead of `20` observations.

In [14]:
```
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))
```

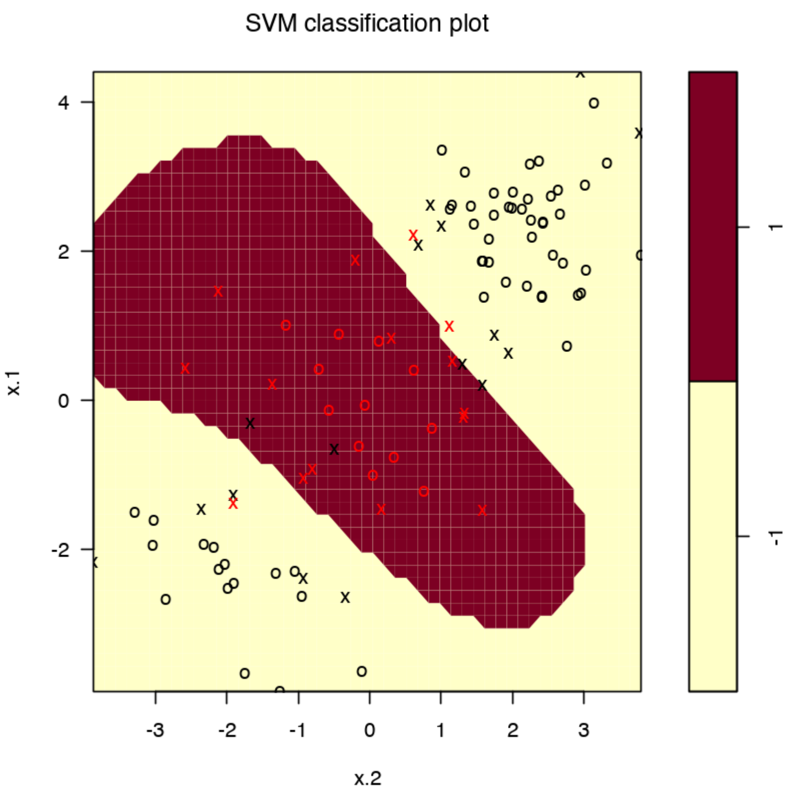


In [15]:
```
print(dat)
```

```
            x.1           x.2  y
1    1.37354619  2.4094018397 -1
2    2.18364332  3.6888732862 -1
3    1.16437139  3.5865884334 -1
4    3.59528080  1.6690921993 -1
5    2.32950777 -0.2852355353 -1
6    1.17953162  4.4976615898 -1
7    2.48742905  2.6670661668 -1
8    2.73832471  2.5413273360 -1
9    2.57578135  1.9866004769 -1
10   1.69461161  2.5101084230 -1
11   3.51178117  1.8356241682 -1
12   2.38984324  2.4206946433 -1
13   1.37875942  1.5997532560 -1
14  -0.21469989  0.6297921225 -1
15   3.12493092  2.9878382675 -1
16   1.95506639  3.5197450255 -1
17   1.98380974  1.6912594308 -1
18   2.94383621  0.7467102444 -1
```

Split the data randomly into training and testing groups. Then fit the training data using an SVM model with a radial kernel and $\gamma = 1$.

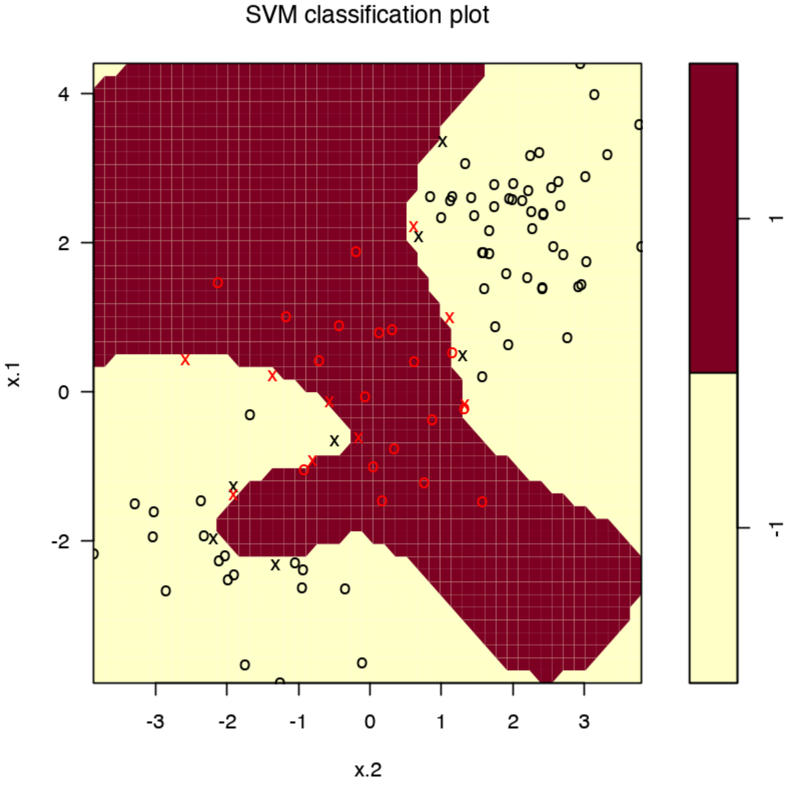
In [16]:
```
train=sample(200,100)
svmfit=svm(y~., data=dat[train,], kernel="radial", gamma=1, cost =
plot(svmfit, dat[train,])
```



We can see from the figure that there are a fair number of training errors in this SVM fit.

Increase the `cost` parameter, to reduce the number of training errors. However, this comes

In [17]:
```
svmfit=svm(y~., data=dat[train,], kernel="radial",gamma=1, cost=1e
plot(svmfit ,dat[train ,])
```



Use cross-validation to select the best choice of `cost` and γ.

In [18]:
```
set.seed(1)
tune.out=tune(svm, y~., data=dat[train,], kernel="radial", ranges=
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
```

Interprete the results. *Your interpretation of the results goes here!*

Assess the test set predictions for the best model.

In `R` we apply the `predict()` function on the subset of the dataframe `dat` consisting of all but the training data, i.e., `-train` as an index set.

In [19]:

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

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

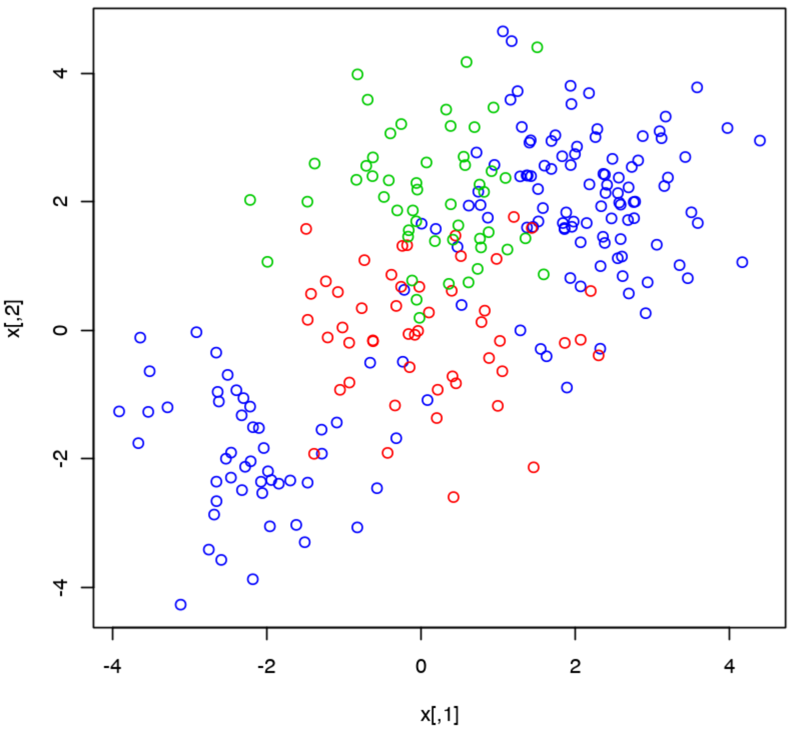
Interprete the results. *Your interpretation of the results goes here!*

Learn and assess an SVM classifier for multiple classes

Generate data as before. We simply extend the matrix `x` with `50` new rows and assign these rows a new class.

In [20]:

```
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=(3-y))
```

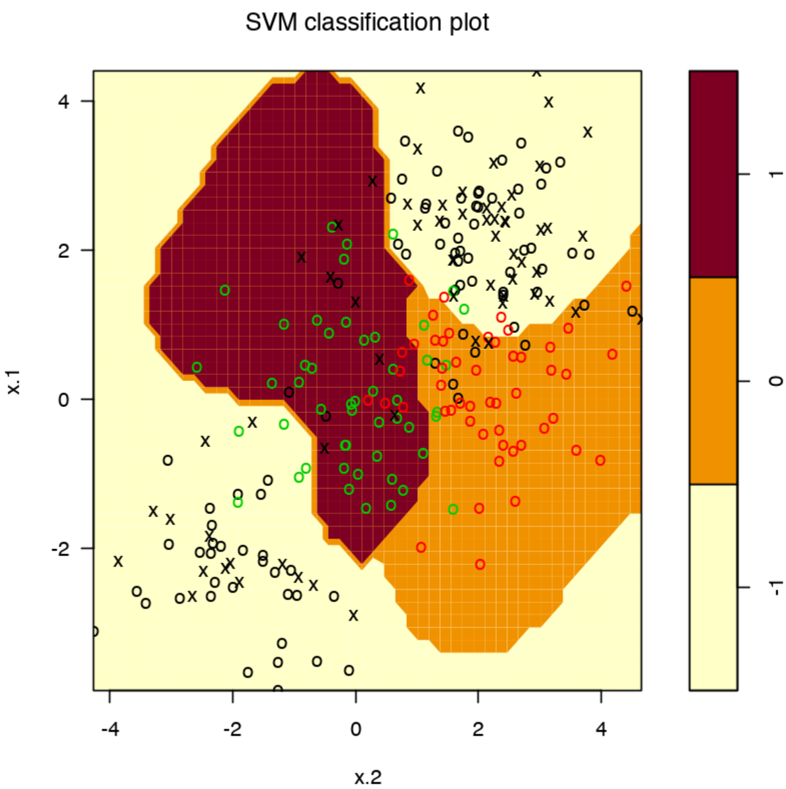


In [21]:
```
print(dat)
```

```
           x.1           x.2  y
1   1.37354619  2.4094018397 -1
2   2.18364332  3.6888732862 -1
3   1.16437139  3.5865884334 -1
4   3.59528080  1.6690921993 -1
5   2.32950777 -0.2852355353 -1
6   1.17953162  4.4976615898 -1
7   2.48742905  2.6670661668 -1
8   2.73832471  2.5413273360 -1
9   2.57578135  1.9866004769 -1
10  1.69461161  2.5101084230 -1
11  3.51178117  1.8356241682 -1
12  2.38984324  2.4206946433 -1
13  1.37875942  1.5997532560 -1
14 -0.21469989  0.6297921225 -1
15  3.12493092  2.9878382675 -1
16  1.95506639  3.5197450255 -1
17  1.98380974  1.6912594308 -1
18  2.94383621  0.7467102444 -1
```

Fit an SVM to the training data.

In [22]:
```
train=sample(250,125)
svmfit=svm(y~., data=dat[train,], kernel="radial", cost=10, gamma=
plot(svmfit , dat)
```



Find the right parameters using cross validation.

In [23]:
```
set.seed(1)
tune.out=tune(svm, y~., data=dat[train,], kernel="radial", ranges=
summary(tune.out)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
    1   0.5

- best performance: 0.2166667

- Detailed performance results:
    cost gamma     error dispersion
1  1e-01   0.5 0.4493590  0.1337631
2  1e+00   0.5 0.2166667  0.1562257
3  1e+01   0.5 0.2320513  0.1650923
4  1e+02   0.5 0.2480769  0.1448916
5  1e+03   0.5 0.2878205  0.1364664
6  1e-01   1.0 0.4160256  0.1136081
7  1e+00   1.0 0.2166667  0.1562257
8  1e+01   1.0 0.2320513  0.1451986
9  1e+02   1.0 0.2878205  0.1502260
10 1e+03   1.0 0.2641026  0.1387343
11 1e-01   2.0 0.4339744  0.1454577
12 1e+00   2.0 0.2320513  0.1650923
13 1e+01   2.0 0.2724359  0.1378247
14 1e+02   2.0 0.2724359  0.1456836
15 1e+03   2.0 0.2653846  0.1599341
16 1e-01   3.0 0.4576923  0.1165343
17 1e+00   3.0 0.2403846  0.1635209
18 1e+01   3.0 0.2961538  0.1480940
19 1e+02   3.0 0.3044872  0.1526918
20 1e+03   3.0 0.2897436  0.1604073
21 1e-01   4.0 0.4500000  0.1241819
22 1e+00   4.0 0.2641026  0.1486786
23 1e+01   4.0 0.2961538  0.1608621
24 1e+02   4.0 0.2967949  0.1448916
25 1e+03   4.0 0.3057692  0.1268824
```

Assess the test set predictions for the best model.

In [24]:
```
table(true=dat[-train,"y"], pred=predict(tune.out$best.model, newd
```

```
    pred
true -1  0  1
  -1 73  4  4
  0   3 17  4
  1   3  2 15
```

Interprete the results. *Your interpretation of the results goes here!*

Apply SVM to Gene Expression Data

The `Khan` data set consists of gene expression measurements for $2,308$ genes and the forresponding 4 cancer subtypes. The training and test sets consist of 63 and 20 data points, respectively.

Load the data and get yourself an overview.

In [28]:
```
load(file = "../ISLR/data/Khan.rda")
names(Khan)
dim(Khan$xtrain)
dim(Khan$xtest)
Khan$xtrain
Khan$ytrain
```

'xtrain' 'xtest' 'ytrain' 'ytest'

63 2308

20 2308

A matrix: 63 × 2308 of type dbl

V1	0.773343700	-2.4384050	-0.482562200	-2.72113500	-1.2170580	0.82780920	1.3426040
V2	-0.078177780	-2.4157540	0.412771700	-2.82514600	-0.6262365	0.05448819	1.4294980
V3	-0.084469160	-1.6497390	-0.241307500	-2.87528600	-0.8894054	-0.02747398	1.1593000
V4	0.965614000	-2.3805470	0.625296500	-1.74125600	-0.8453664	0.94968680	1.0938010
V5	0.075663900	-1.7287850	0.852626500	0.27269530	-1.8413700	0.32793590	1.2512190
V6	0.458816300	-2.8752860	0.135841200	0.40539840	-2.0826470	0.13784710	1.7335300

Use a support vector approach to predict the cancer subtypes using gene expression measurements.

In this data set, there are a very large number of features relative to the number of observations. This suggests that we should use a linear kernel, because the additional flexibility that will result from using a polynomial or radial kernel is unnecessary.

In [29]:
```
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
```

Assess the training error.

In [30]:
```
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
```

We see that there are no training errors. In fact, this is not surprising, because the large number of variables relative to the number of observations implies that it is easy to find hyperplanes that fully separate the classes. We are most interested not in the support vector classifier's performance on the training observations, but rather its performance on the test observations.

Assess the test error.

In [31]:
```
dat.test=data.frame(x=Khan$xtest, y=as.factor(Khan$ytest))
pred.test=predict(out, newdata=dat.test)
table(pred.test, dat.test$y)
```

```
pred.test 1 2 3 4
        1 3 0 0 0
        2 0 6 2 0
        3 0 0 4 0
        4 0 0 0 5
```

Interprete the results. *Your interpretation of the results goes here!*