LAB 09

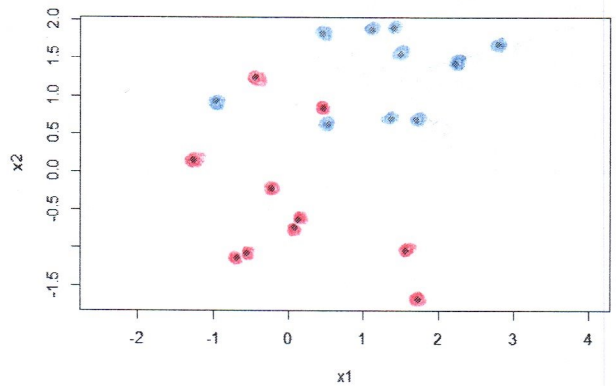
TOPICS:

- Support Vector Machines

```
library(e1071)
help(svm)
```

```
### ------------------------------------------------------------------
### Linear case
### ------------------------------------------------------------------
# Generate the data
set.seed (123)
x <- matrix (rnorm (20*2) , ncol =2)
y <- c(rep (-1,10) , rep (1 ,10) )
x[y==1,] <- x[y==1,] + 1

# The classes are not separable
x11()
plot(x, col =ifelse(y==1, 'light blue', 'salmon'),
     pch=19, xlab='x1', ylab='x2', asp=1)
```

→ there is no separating hyperplane



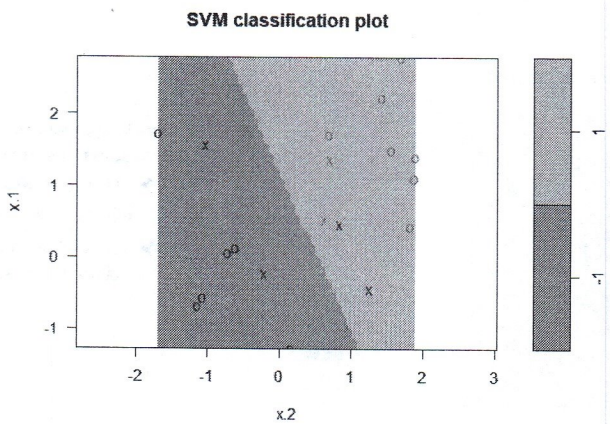

```
# Fit the Support Vector Classifier (kernel = "linear")
# given a cost C
dat <- data.frame(x=x, y=as.factor (y))
svmfit <- svm(y~., data=dat , kernel ='linear', cost =10, scale =FALSE )
summary(svmfit)
```

features = x, response variable = y (← factor!)

scale = TRUE if we want to standardize (normalize) the variables

```
##
## 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
```

```
x11()
par(mfrow=c(1,2))
plot(svmfit , dat, col =c('salmon', 'light blue'), pch=19, asp=1)
```



x = support vectors
o = other observations

```
# support vectors are indicated with crosses
# they are:
svmfit$index
```

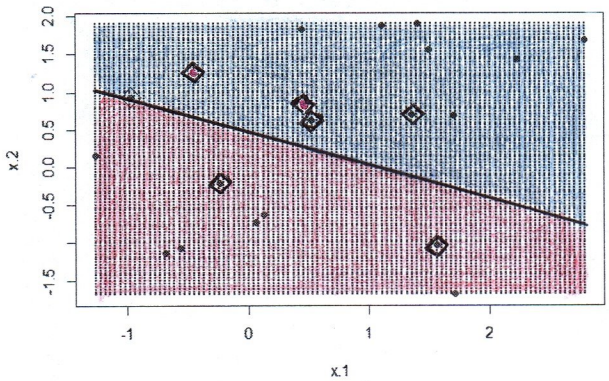
indexes of the observations that have been used for support vectors

```
## [1]  2  3  7 10 12 18 20
```

Another possibility (possibilities) to represent graphically the partition provided by the support vector machine algo.

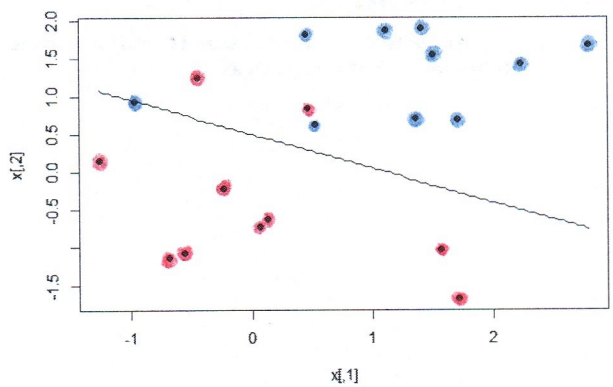
```
###
n.g <- 100

xgrid <- expand.grid(x.1=seq(from=range(dat$x.1)[1],to=range(dat$x.1)[2],length=n.g),
                     x.2=seq(from=range(dat$x.2)[1],to=range(dat$x.2)[2],length=n.g))
ygrid <- predict(svmfit,xgrid)
x11()
plot(xgrid,col=c("red","blue")[as.numeric(ygrid)],pch=20,cex=.2)
points(x,col=c("red","blue")[as.numeric(dat$y)],pch=19)
points(x[svmfit$index,],pch=5,cex=2)
```

1.

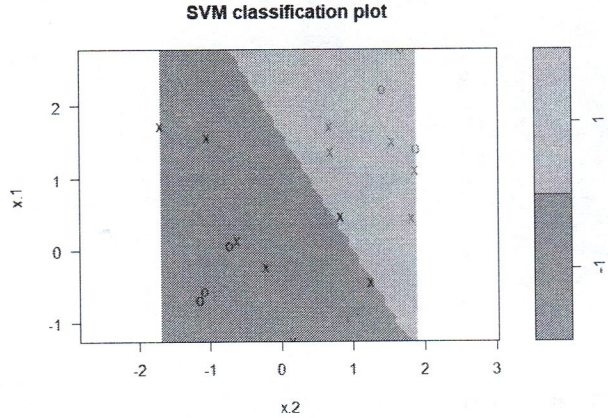



```
x11()
plot(x,col=c("red","blue")[as.numeric(dat$y)],pch=19)
contour(seq(from=range(dat$x.1)[1],to=range(dat$x.1)[2],length=n.g),
        seq(from=range(dat$x.2)[1],to=range(dat$x.2)[2],length=n.g),
        matrix(as.numeric(ygrid),n.g,n.g),level=1.5,add=TRUE,
        drawlabels=F)
```

2.




```
###

# If we try to change the cost parameter we get more support points
# (higher bias, lower variance)
svmfit <- svm(y~., data=dat , kernel ='linear', cost =0.1, scale =FALSE )
plot(svmfit , dat, col =c('salmon', 'light blue'), pch=19, asp=1)
```



← we have more points used as support vectors
⇒ the estimation is based on more points
⇒ increase the bias, decrease the variance

```
# To set the parameter C we can use the function tune(),
# which is based on cross-validation (10-fold)
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)
```

we have to provide a list of possible parameters that we can use

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
## cost
##    1
##
## - best performance: 0.15
##
## - Detailed performance results:
##    cost error dispersion
## 1 1e-03  0.55  0.4377975
## 2 1e-02  0.55  0.4377975
## 3 1e-01  0.20  0.2581989
## 4 1e+00  0.15  0.2415229
## 5 5e+00  0.20  0.2581989
## 6 1e+01  0.20  0.2581989
## 7 1e+02  0.20  0.2581989
```

this is a very common use of cross-validation: we introduced cross validation to compare classifiers and in fact here we're comparing different classifiers (each one has been determined by choosing a different value for **c**).

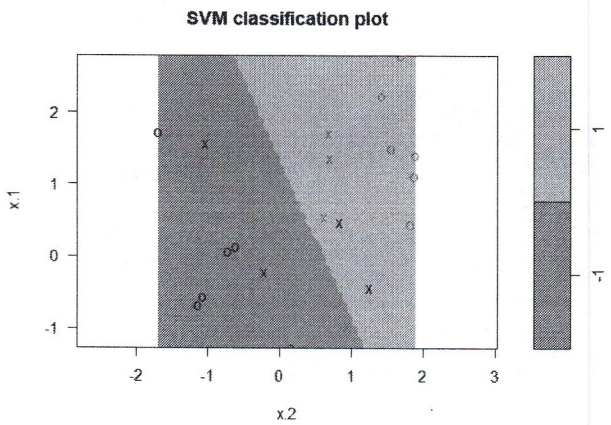
But it's more general than that: we'll see that, even when we'll talk about linear models we'll introduce penalizations and then we'll have to choose the penalizing parameter and cross-validation will come very handy (in sense of choosing the parameter for which prediction error is minimized ($\forall$ pred. error is)).

CROSS VALIDATION IS LIKE:
we have different possibilities, let's try all of them on a test set and see which one perform better, but we don't have a test set, so by cross-validation we create a test set and then we perform the testing

```
# Extract the best model from the result of tune
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:  1
##
## Number of Support Vectors:  8
##
##  ( 4 4 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

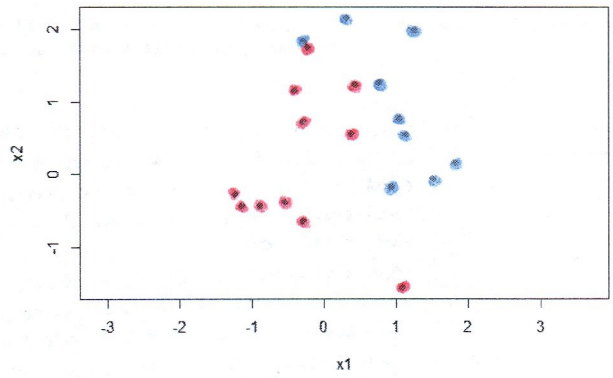
```
plot(bestmod , dat, col =c('salmon', 'light blue'), pch=19, asp=1)
```



we can use the estimated function to predict the labels of a new observation:

```
# Prediction for a new observation (command predict())
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))

plot(xtest, col =ifelse(ytest==1, 'light blue', 'salmon'),
     pch=19, xlab='x1', ylab='x2', asp=1)
```

we generate the data in the same way we generated the dataset (these are data for testing)




```
ypred <- predict(bestmod,testdat)
table(true.label=testdat$y, assigned.label =ypred )
```

Here the 2 classes were not separable

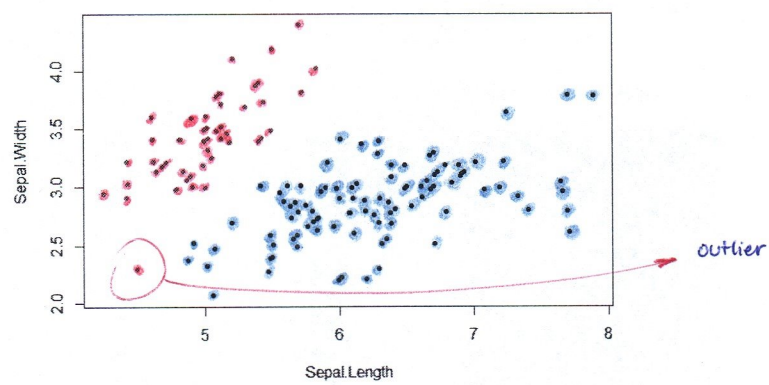
```
##            assigned.label
## true.label -1 1
##         -1  6 5
##          1  1 8
```

→ 6 misclassifications :

consider now an example where the classes are separable (∃ hyperplane)

```
# If the classes are separable, setting a high value for the cost function
# leads to the maximal margin classifier (i.e., it returns the classification
# provided by the best separating hyperplane)

species.name <- factor(iris$Species, labels=c('setosa','versicolor','virginica'))
set.seed(1)
iris2 <- iris[,1:2] + cbind(rnorm(150, sd=0.025))    # jittering

# setosa VS versicolor+virginica
y <- rep(0,150)
y[which(species.name=='setosa')] <- 1

x11()
plot(iris2[,1], iris2[,2], xlab='Sepal.Length', ylab='Sepal.Width', pch=20, col=as.character(y+1))
```

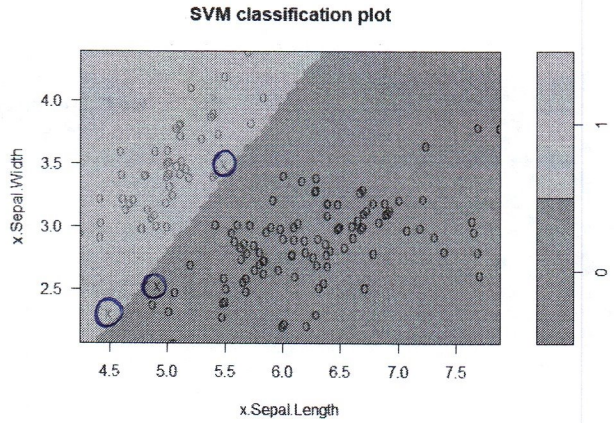
(we use just the 2 first features)




```
dat <- data.frame(x=iris2[,c(2,1)], y=as.factor (y))
svmfit <- svm(y~., data=dat , kernel ='linear', cost =100, scale =FALSE )
summary(svmfit)
```

```
##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 100, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  100
##
## Number of Support Vectors:  3
##
##  ( 2 1 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
x11()
par(mfrow=c(1,2))
plot(svmfit , dat, col =c('salmon', 'light blue'), pch=19)
```

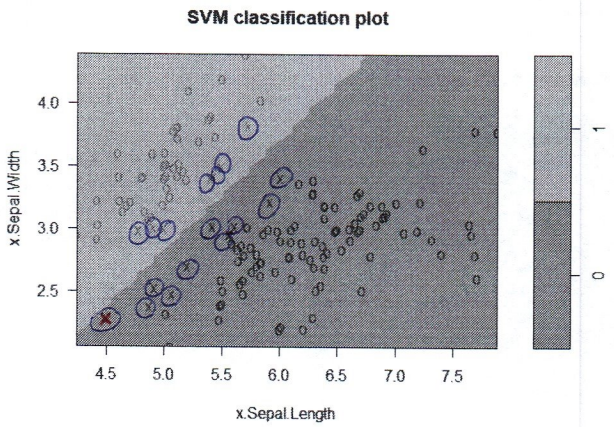


← 3 support vectors (one of them is the outlier we identified)

```
dat <- data.frame(x=iris2[,c(2,1)], y=as.factor (y))
svmfit <- svm(y~., data=dat , kernel ='linear', cost =1, scale =FALSE )
summary(svmfit)
```

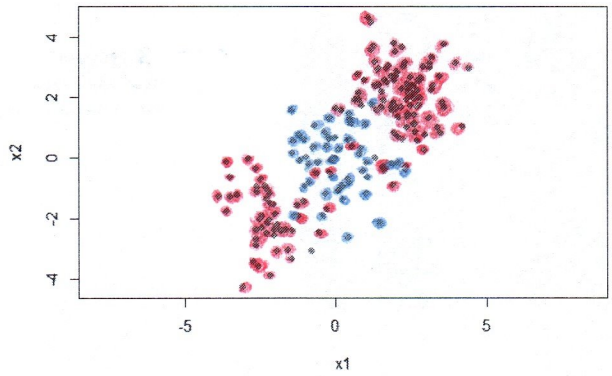
```
##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 1, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  18
##
##  ( 9 9 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
x11()
par(mfrow=c(1,2))
plot(svmfit , dat, col =c('salmon', 'light blue'), pch=19)
```



← 17 support vectors (the previous outlier now changed side and is misclassified)

```
### ------------------------------------------------------------------
### Non-Linear case
### ------------------------------------------------------------------
# Generate the 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 (2 ,50) )
dat <- data.frame(x=x,y=as.factor (y))

# The classes are not separable
x11()
plot(x, col =ifelse(y==1, 'salmon', 'light blue'), pch=19, xlab='x1', ylab='x2', asp=1)
```

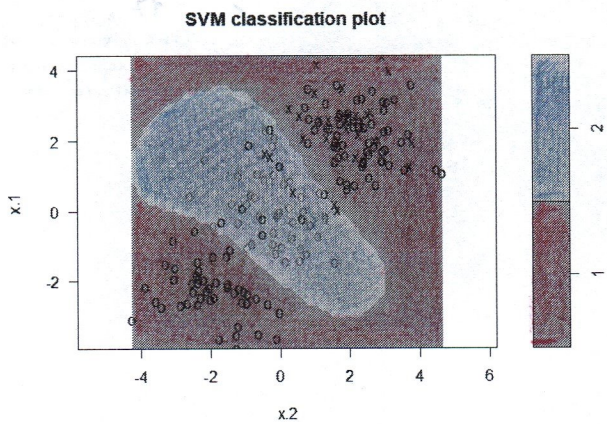


← not linearly separable

```
# Randomly split in train and test
train <- sample (200 ,100)

# Fit a Support Vector Machine (kernel = "radial") given a cost C
svmfit <- svm(y~., data=dat [train ,], kernel ='radial', gamma =1, cost =1)
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
```

```
# Plot the SVM
x11()
plot(svmfit , dat, col =c('salmon', 'light blue'), pch=19, asp=1)
```



← with the same algorithm we can get a separation which is not linear

```
# Misclassification error on the training set
table(true=dat[train ,"y"], pred=predict (svmfit ,
                                          newdata =dat[train ,]))
```

→ we use only "train"

```
##     pred
## true  1  2
##    1 69  4
##    2  3 24
```

```
(3+4)/100
```

```
## [1] 0.07
```

```
# Misclassification error on the test set
table(true=dat[-train ,"y"], pred=predict (svmfit ,
                                          newdata =dat[-train ,]))
```

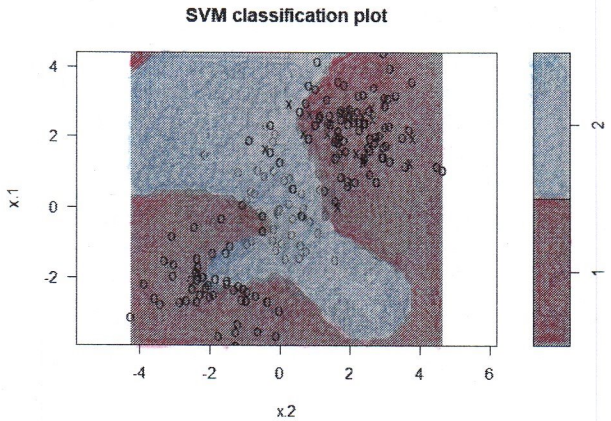
→ we remove "train"

```
##     pred
## true  1  2
##    1 67 10
##    2  3 20
```

```
(3+10)/100
```

```
## [1] 0.13
```

```
# Increasing the cost decreases the errors on the training set,
# at the expense of a more irregular boundary
svmfit <- svm(y~., data=dat [train ,], kernel ='radial',gamma =1,cost=1e5)

x11()
plot(svmfit , dat, col =c('salmon', 'light blue'), pch=19, asp=1)
```




```
# Misclassification error on the training set
table(true=dat[train ,"y"], pred=predict (svmfit ,
                                          newdata =dat[train ,]))
```

```
##     pred
## true  1  2
##    1 73  0
##    2  0 27
```

```
0
```

```
## [1] 0
```

← Smaller : perfect fit of the data (overfitting)

```
# Misclassification error on the test set
table(true=dat[-train ,"y"], pred=predict (svmfit ,
                                          newdata =dat[-train ,]))
```

```
##     pred
## true  1  2
##    1 57 20
##    2  5 18
```

```
(5+20)/100
```

```
## [1] 0.25
```

← larger : we are overfitting !

```
# Set parameters via CV:
set.seed (1)
tune.out <- tune(svm , y~., data=dat[train ,], kernel ='radial',
                 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
```

related to the kernel that we're using

```
# Misclassification error with best model on the training set
table(true=dat[train ,"y"], pred=predict (tune.out$best.model ,
                                          newdata =dat[train ,]))
```

```
##     pred
## true  1  2
##    1 69  4
##    2  2 25
```

```
(2+4)/100
```

```
## [1] 0.06
```

```
# Misclassification error with best model on the test set
table(true=dat[-train ,"y"], pred=predict (tune.out$best.model ,
                                          newdata =dat[-train ,]))
```

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

```
(2+10)/100
```

```
## [1] 0.12
```

```
###-----------------------------------------------------------------
### Application to Gene Expression Data (multiclass classification)
###-----------------------------------------------------------------
library (ISLR)
help(Khan)
```

data concern the gene expression. The data consists of tissue samples from cell tumors and there are different types of tumors (4 types) and there are a lot of features (in the gene expression)

```
is(Khan)
```

```
## [1] "list"   "vector"
```

```
names(Khan)
```

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

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

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

features # patients

```
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
```

← we have labels in Y (both training and test)

```
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
```

the error on the training is 0
(we have a very large dataset :
very large features space and very few observations)

```
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
```

also the test set performs quite good
(only 2 misclassifications)