Module 8: Compuate Aided Technique

Weixing Song

July 24, 2018

First, we use the following R-codes to install all necessary packages.

```
list.of.packages=c("tree","rpart","rattle","tidyverse","kernlab","e1071","ISLR","RColorBrewer","ggplot2
if(length(which(!list.of.packages %in% installed.packages()))){
  install.packages(list.of.packages[!list.of.packages %in% installed.packages()])}
```

1. Classification Tree

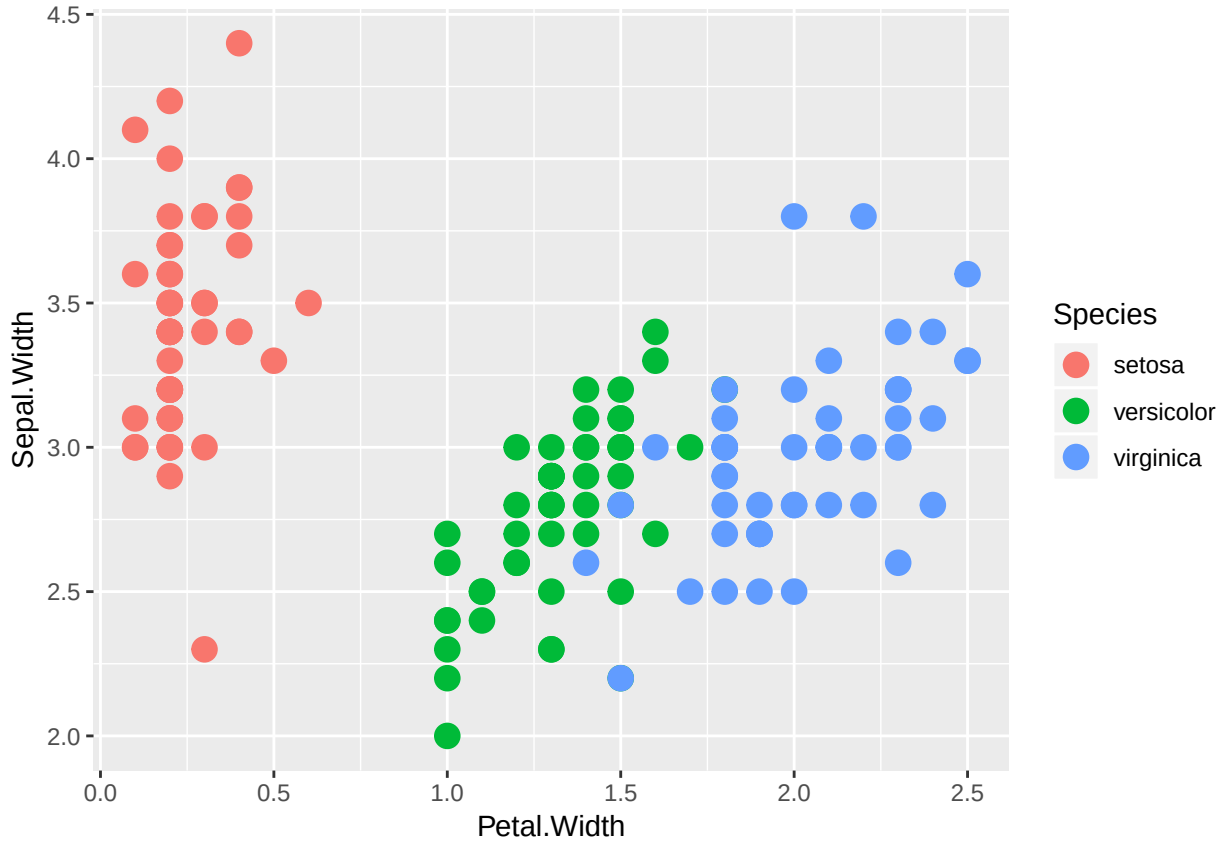
```
  data(iris)
  names(iris)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"
## [5] "Species"
```

```
  table(iris$Species)
```

```
##
##     setosa versicolor  virginica
##         50         50         50
```

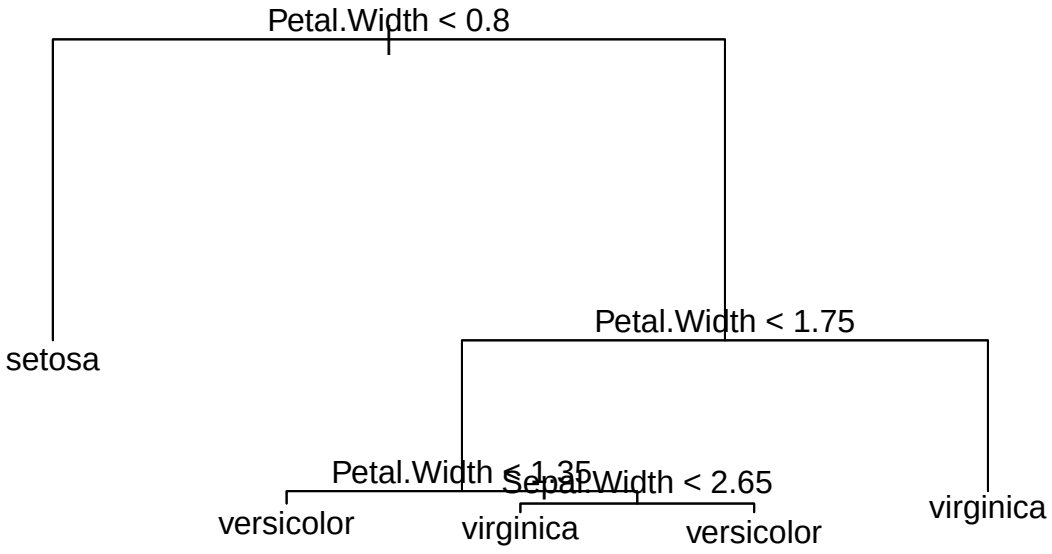
```
  library(ggplot2)
  qplot(Petal.Width,Sepal.Width,data=iris,colour=Species,size=I(4))
```



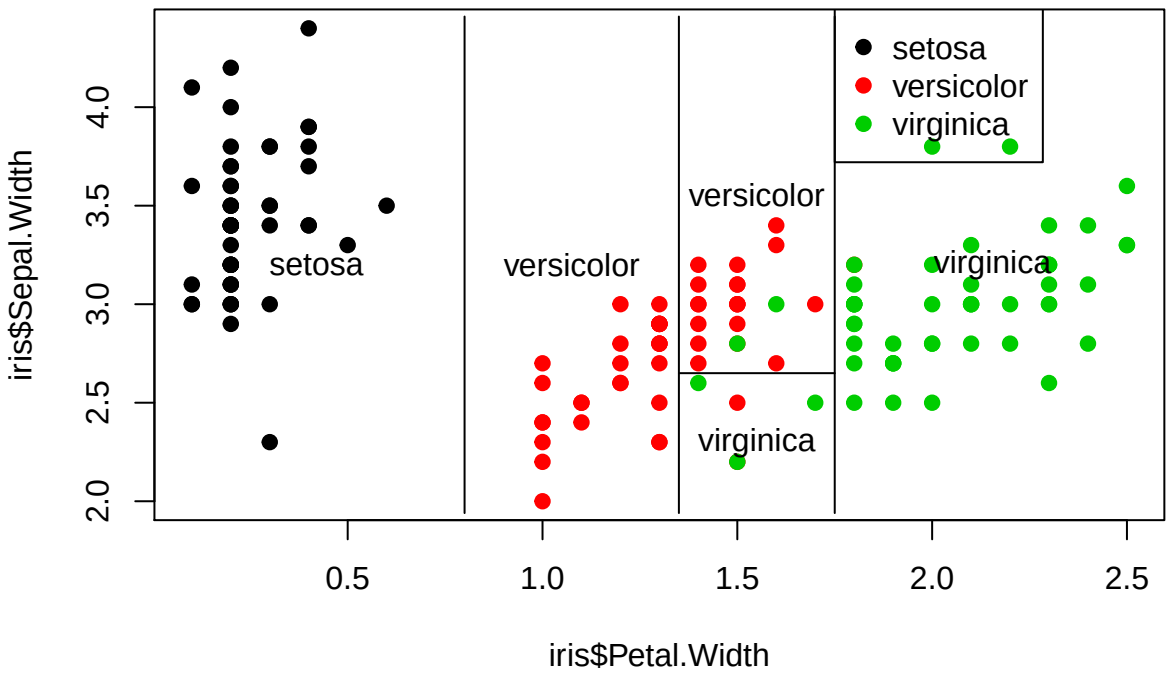

```
    # Use I(value) to indicate a specific value. For example size=z makes the size of the plotted point
    # lines proporational to the values of a variable z. In contrast, size=I(3) sets each point or line
    # three times the default size.

  # Building the classification tree

  #install.packages("tree")
  library(tree)
  tree1=tree(Species~Sepal.Width+Petal.Width,data=iris)
  summary(tree1)
```

```
##
## Classification tree:
## tree(formula = Species ~ Sepal.Width + Petal.Width, data = iris)
## Number of terminal nodes:  5
## Residual mean deviance:  0.204 = 29.57 / 145
## Misclassification error rate: 0.03333 = 5 / 150
```

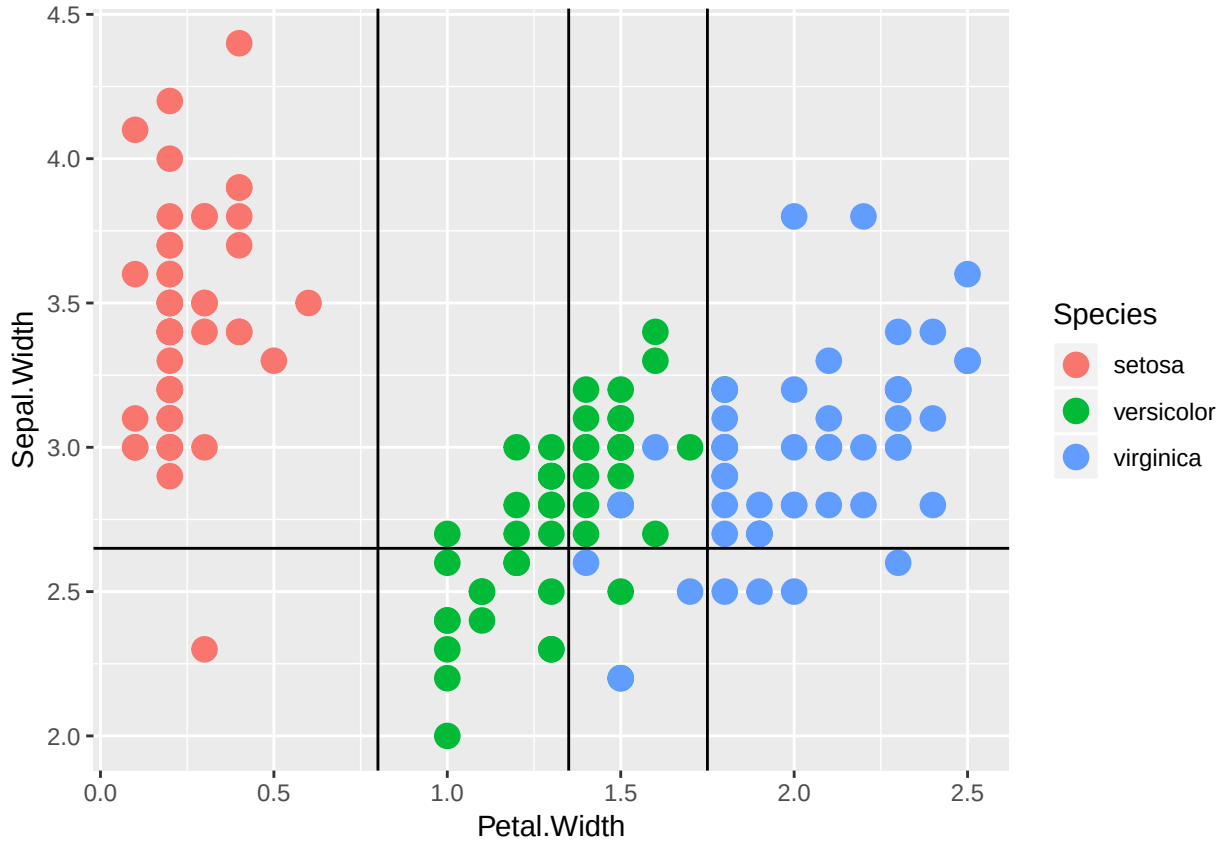
```
  # Tree plot
  # A classification tree showing at each internal node the feature property and at each terminal
  # node the species.

  plot(tree1)
  text(tree1)
```




```
#The partitions are defined by the classification tree above. For example, the first node
#partitions every species with petal width < 0.8 as setosa. Next, all species with petal width >
#1.75 are virginica and so on.
plot(iris$Petal.Width,iris$Sepal.Width,pch=19,col=as.numeric(iris$Species))
partition.tree(tree1,label="Species",add=TRUE)
legend(1.75,4.5,legend=unique(iris$Species),col=unique(as.numeric(iris$Species)),pch=19)
```



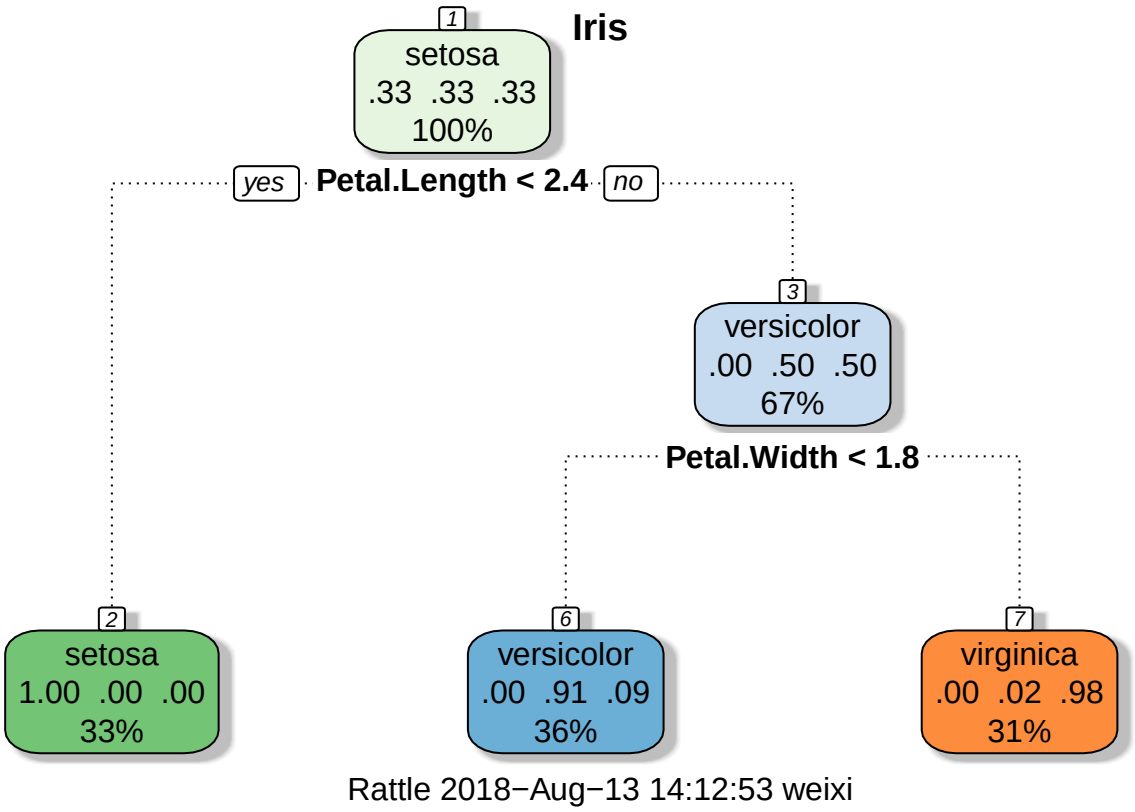

```
library(ggplot2)
graph=qplot(Petal.Width, Sepal.Width, data=iris, colour=Species, size=I(4))
graph + geom_hline(aes(yintercept=2.65)) + geom_vline(aes(xintercept=0.8)) +
    geom_vline(aes(xintercept=1.75)) + geom_vline(aes(xintercept=1.35))
```




```
  # Prettier classification trees in R using the rpart package
# install if necessary
 #install.packages('rpart')
 #install.packages('rattle')

  library(rpart)
  library(rattle)
```

```
## Rattle: A free graphical interface for data science with R.
## Version 5.1.0 Copyright (c) 2006-2017 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
  rpart=rpart(Species~., data=iris, method="class")

  # plot decision tree
  fancyRpartPlot(rpart, main="Iris")
```



2. Support Vector Machine

Maximal margin classifier

```
set.seed(10);
#install.packages(c("tidyverse","kernlab","e1071","ISLR","RColorBrewer","ggplot2"))
library("tidyverse") # data manipulation and visualization
```

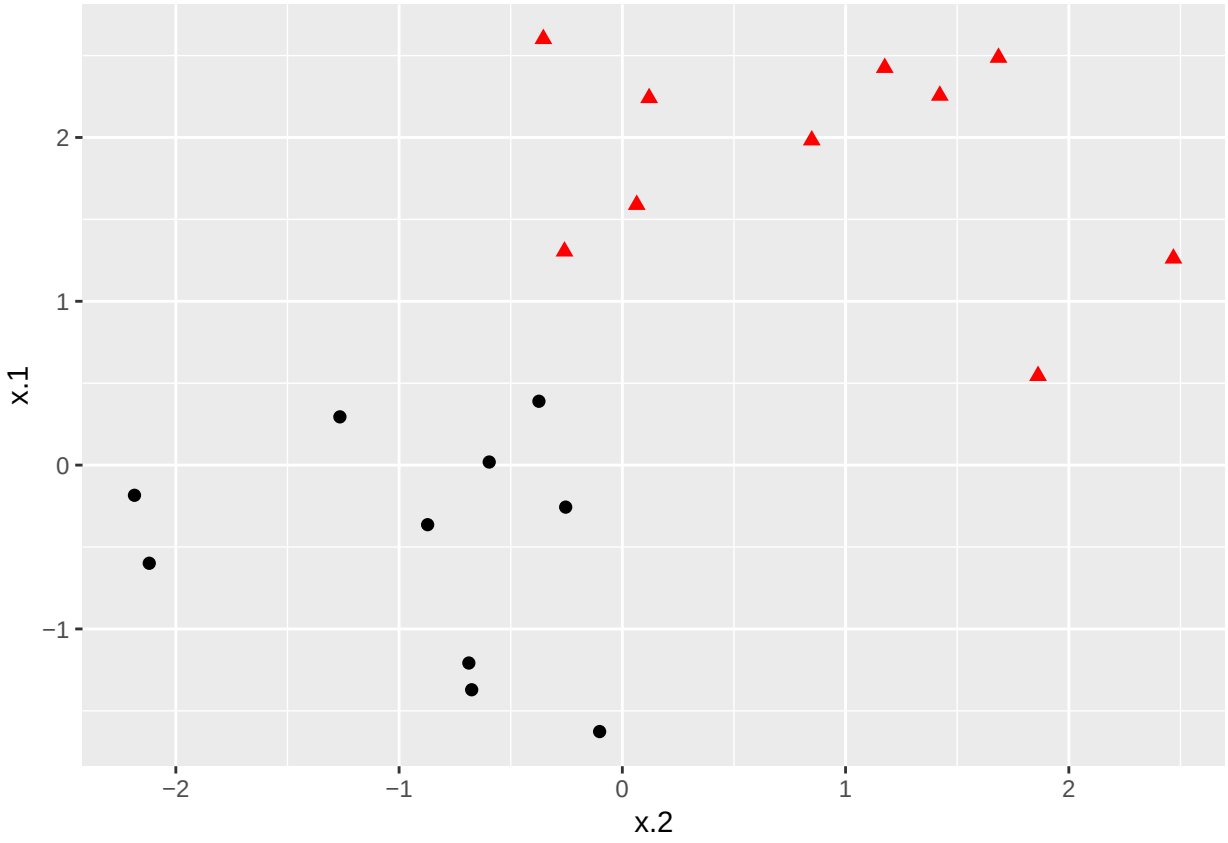
```
## -- Attaching packages ------------------------- tidyverse 1.2.1 --
## v tibble  1.4.2     v purrr   0.2.4
## v tidyr   0.8.1     v dplyr   0.7.4
## v readr   1.1.1     v stringr 1.2.0
## v tibble  1.4.2     v forcats 0.3.0
## -- Conflicts ---------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library("kernlab") # SVM methodology
```

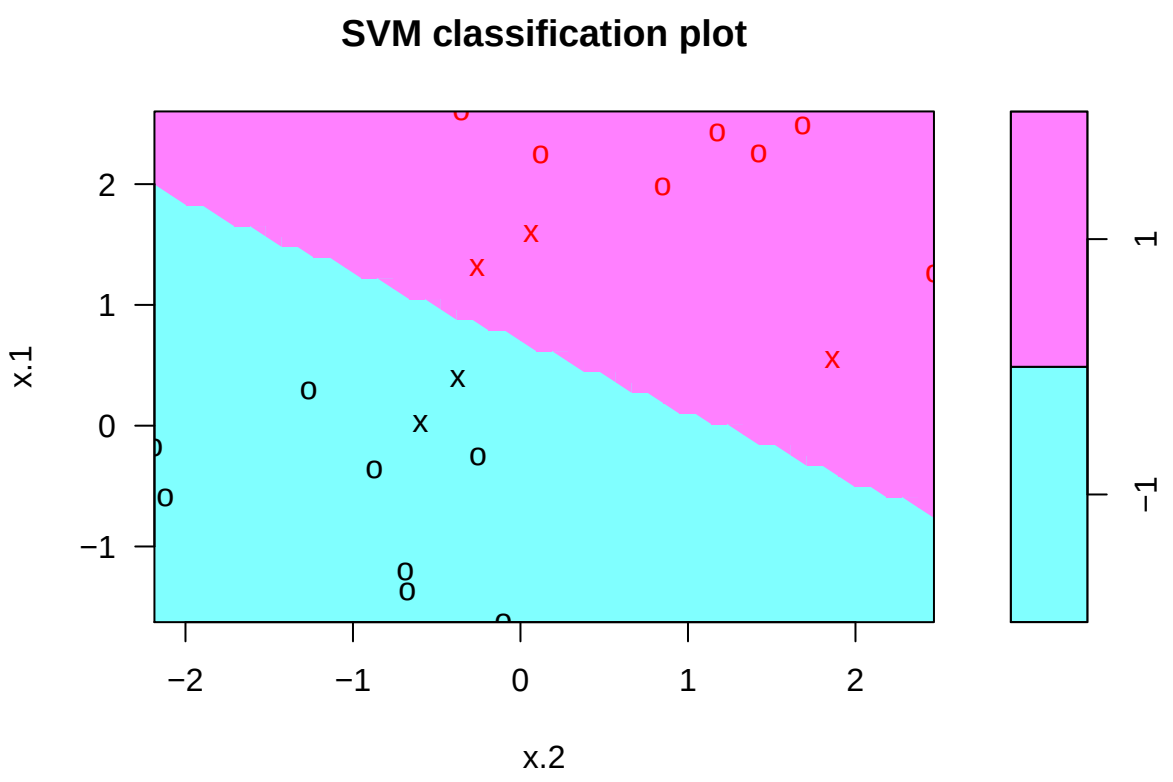
```
##
## Attaching package: 'kernlab'
## The following object is masked from 'package:purrr':
##
```

```
##     cross
## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```
library("e1071") # SVM methodology
library("ISLR") # Contains example data set "Khan"
library("RColorBrewer") # customized coloring of plots
library("ggplot2")

# Generate a data set which is linarly separable
 x=matrix(rnorm(40),ncol=2)
 x[11:20,]=x[11:20,]+1.5;
 y=c(rep(-1,10),rep(1,10))
 dat=data.frame(x=x,y=as.factor(y))
 ggplot(data=dat,aes(x=x.2,y=x.1,color=y,shape=y))+geom_point(size=2)+
    scale_color_manual(values=c("#000000","#FF0000"))+theme(legend.position = "none")
```



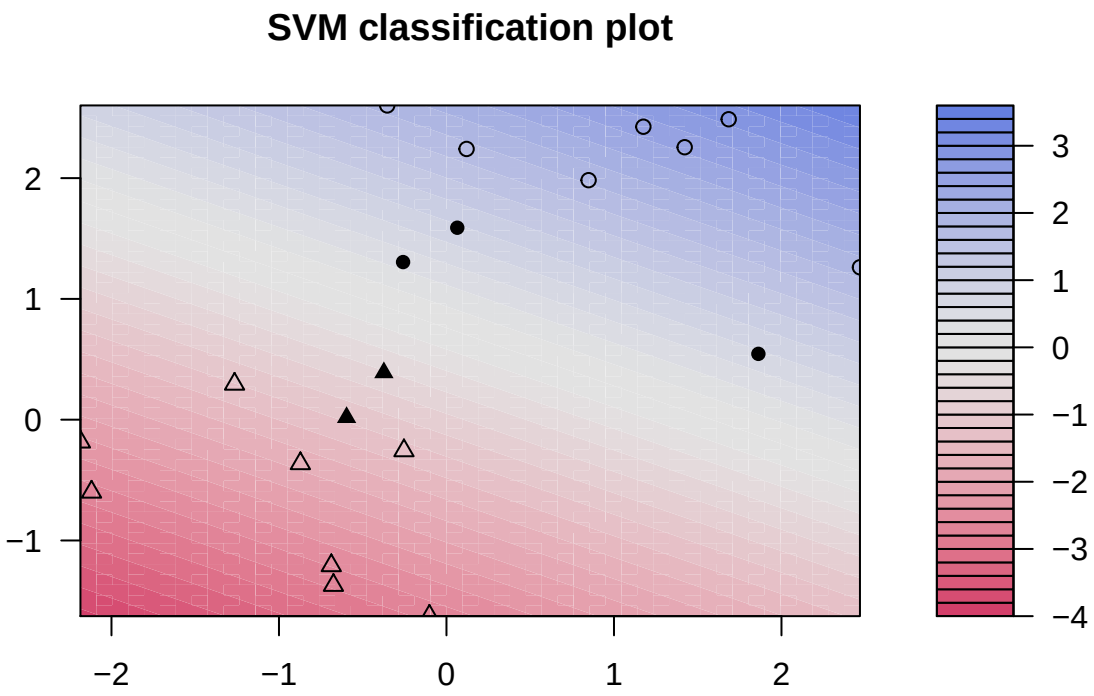

```
# Fit SVM model to data set using svm function from e1071 package
svmfit=svm(y~.,data=dat,kernel="linear",scale=F)
plot(svmfit,dat)
```




```
    #  In the plot, points that are represented by an "X" are the support vector, or the points
    #  that directly affect the classification line. The points marked with an "o" are the
    #  other points, which don't affect the calculation of the line.

  # Fit SVM model using ksum function from kernlab package
  kernfit=ksvm(x,y,type="C-svc",kernel="vanilladot");
```

```
##  Setting default kernel parameters
```

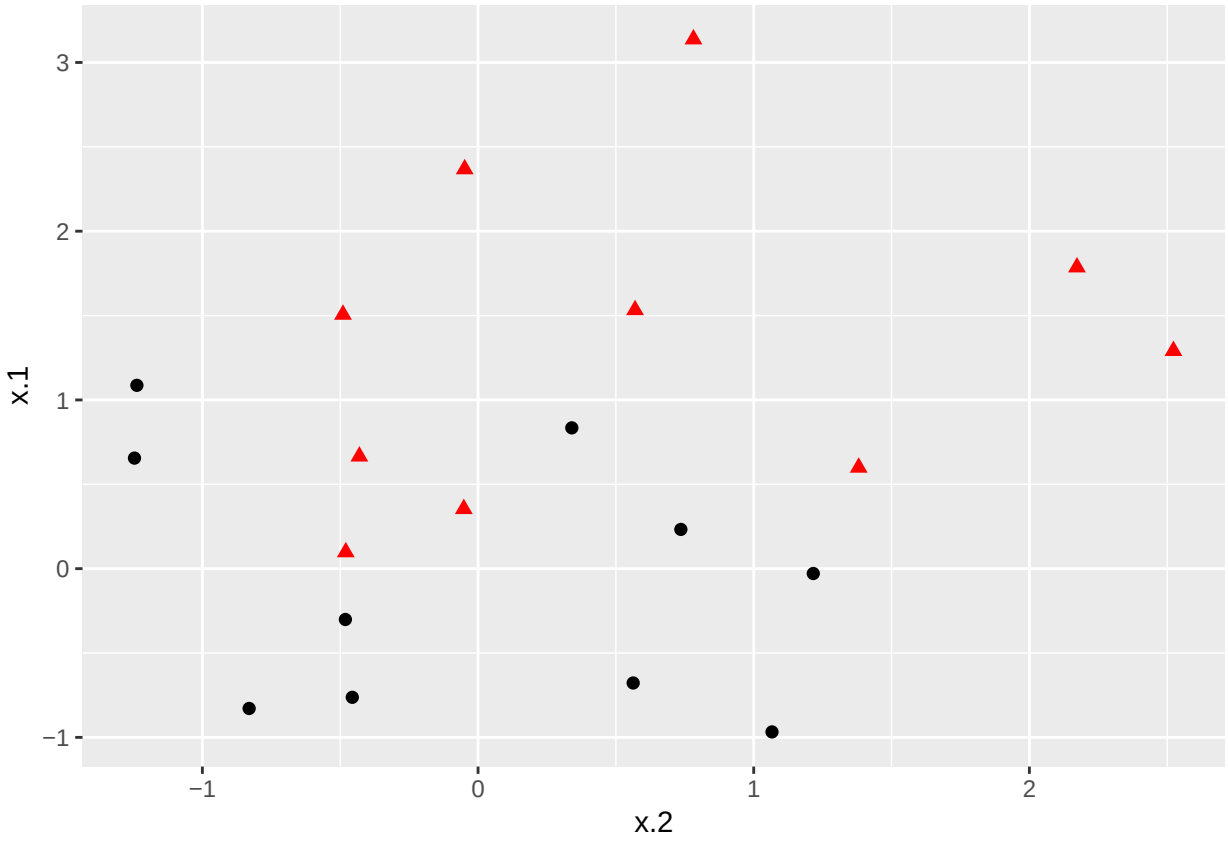
```
  plot(kernfit,data=x)
```



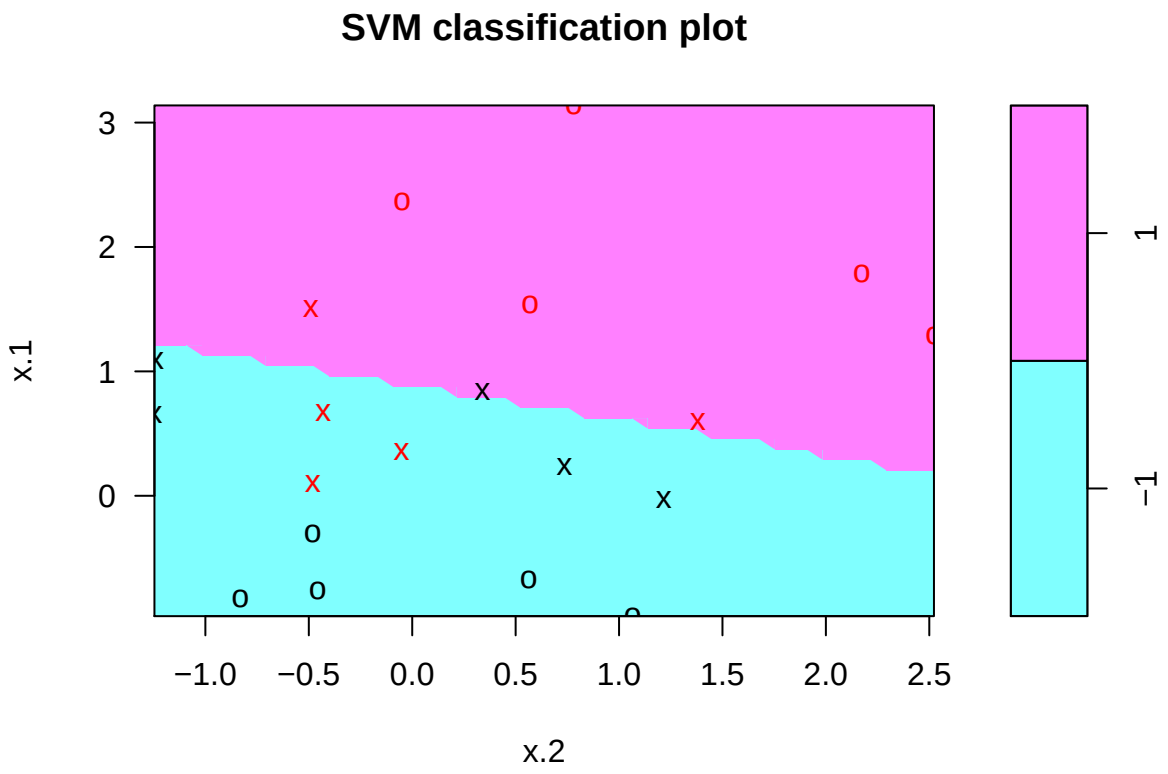

```
    # The color gradient indicates how confidently a new point would be classified based on its
    # features.
```

Support Vector Classifiers

```
# Generate sample data not completely separated
 x=matrix(rnorm(40),ncol=2);
 x[11:20,]=x[11:20,]+1
 dat=data.frame(x=x,y=as.factor(y));
 ggplot(data=dat,aes(x=x.2,y=x.1,color=y,shape=y))+geom_point(size=2)+
    scale_color_manual(values=c("#000000","#FF0000"))+theme(legend.position = "none")
```



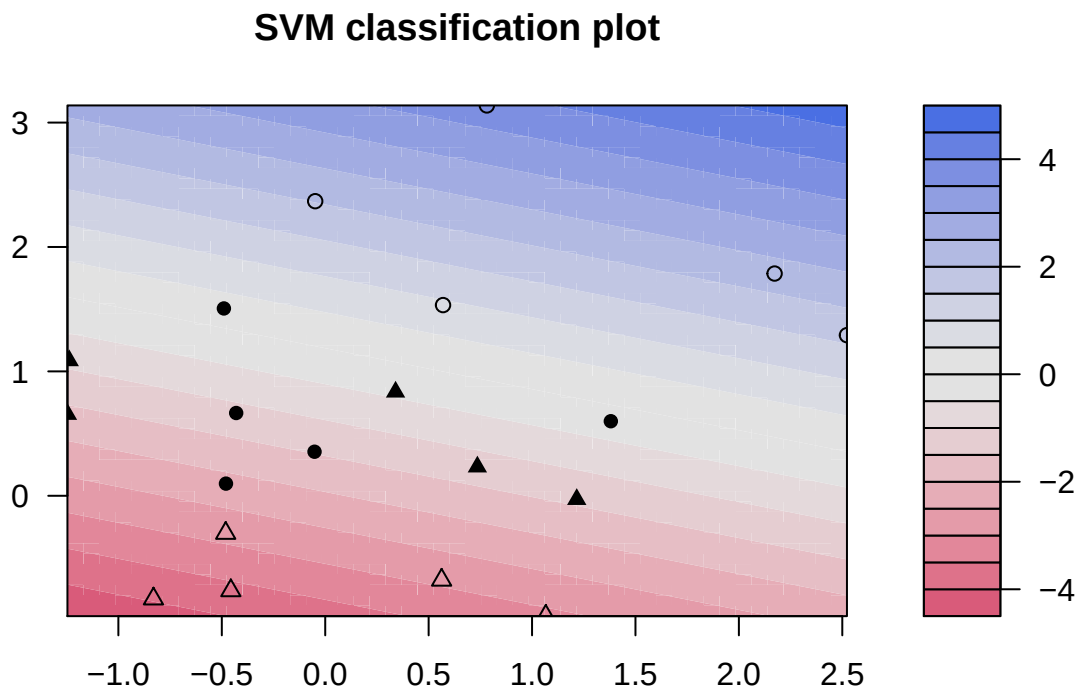

```
# Fit SVM model to data set using sum function from e1071 package
 svmfit=svm(y~.,data=dat,kernel="linear",scale=F,cost=10)
 plot(svmfit,dat)
```




```
  # Fit SVM model using ksum function from kernlab package
  kernfit=ksvm(x,y,type="C-svc",kernel="vanilladot",C=100);
```

```
##  Setting default kernel parameters
```

```
  plot(kernfit,data=x)
```




```
    # The color gradient indicates how confidently a new point would be classified based on its
    # features.

# Instead of specifying a cost up front, we can use the tune() function from e1071 to test
# various costs and identify which value produces the best fitting model.

# find optimal cost of misclassification
tune.out=tune(svm,y~., data=dat,
              kernel="linear",ranges=list(cost=c(0.001,0.01,0.1,1,5,10,100)))

# extract the best model
(bestmod=tune.out$best.model)
```

```
##
## 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
##       gamma:  0.5
##
## Number of Support Vectors:  16
```

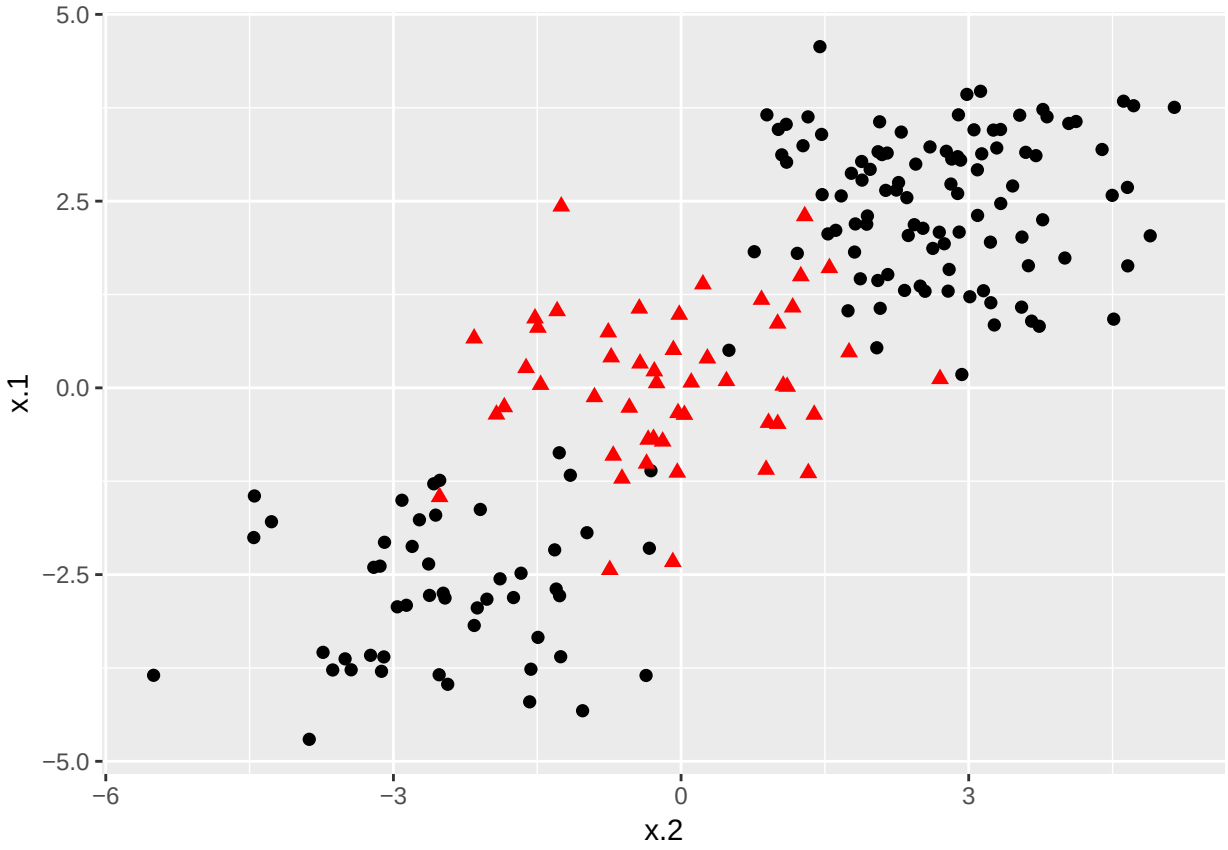
```
# Confusion matrix and classification error rate
ypred=predict(bestmod,dat)
conf.matrix=table(predict=ypred,truth=dat$y)
cer=mean(ypred!=dat$y)
```

Support Vector Machine

Two classes

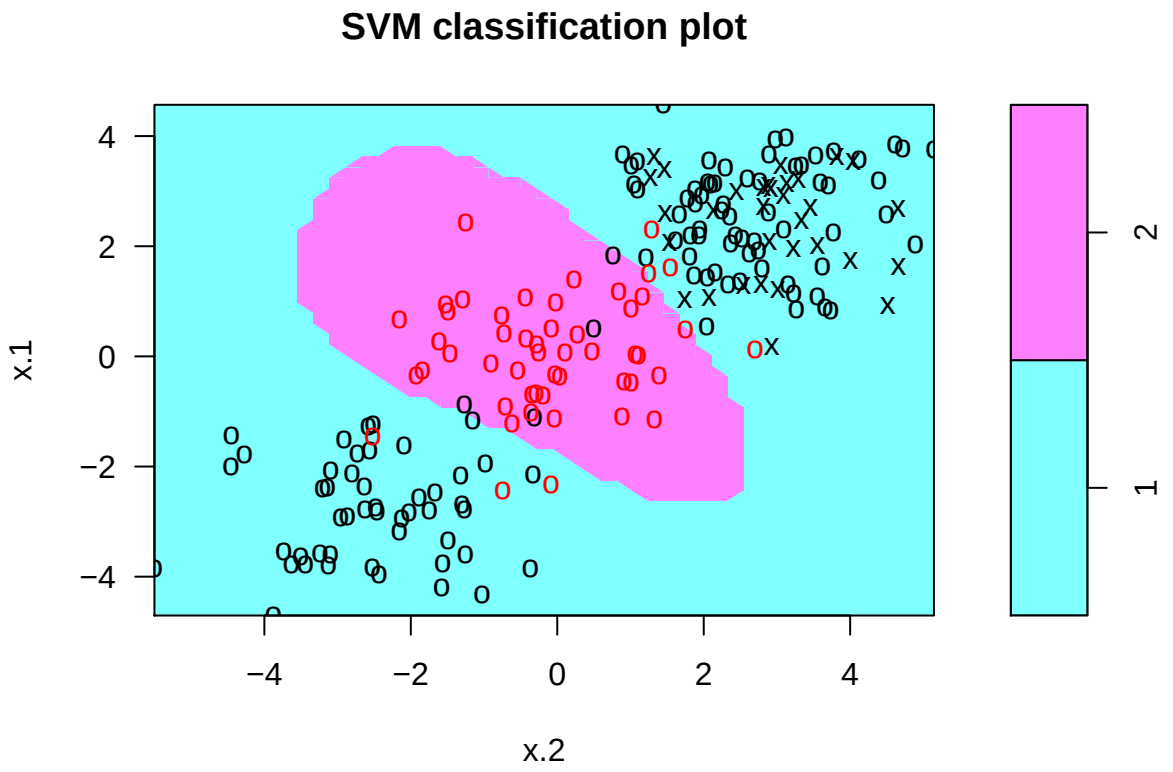
We will demonstrate a nonlinear classification boundary.

```
x=matrix(rnorm(400),ncol=2)
x[1:100,]=x[1:100,]+2.5
x[101:150,]=x[101:150,]-2.5
y=c(rep(1,150),rep(2,50))
dat=data.frame(x=x,y=as.factor(y))
ggplot(data=dat,aes(x=x.2,y=x.1,color=y,shape=y))+geom_point(size=2)+
    scale_color_manual(values=c("#000000","#FF0000"))+theme(legend.position = "none")
```

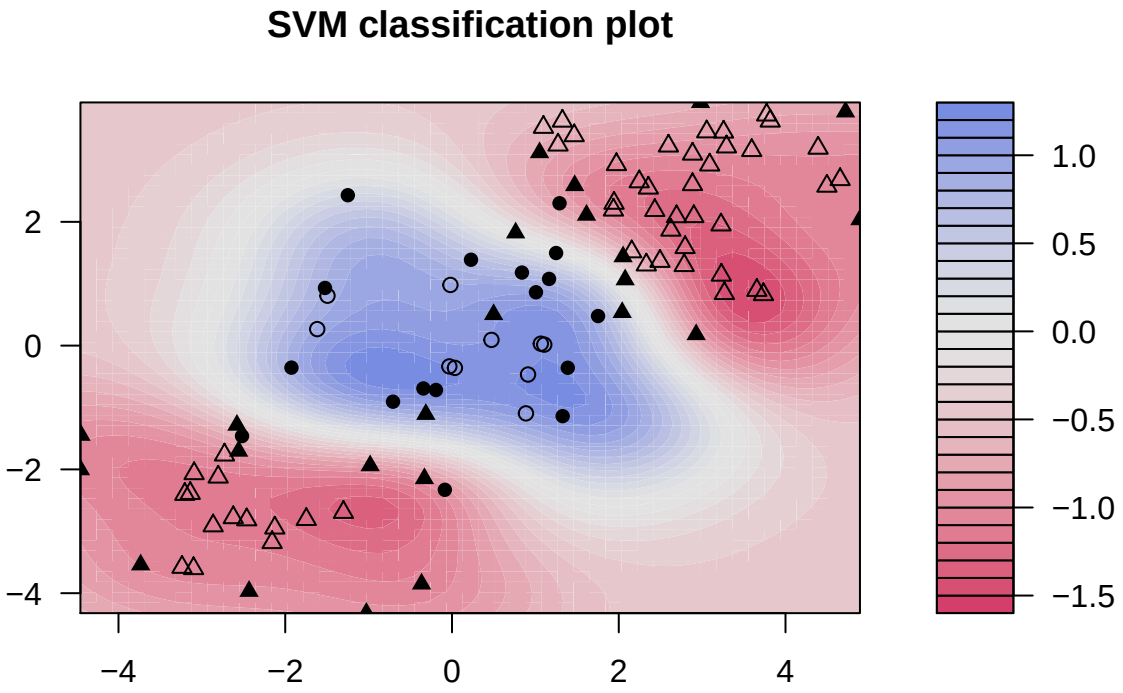



```
set.seed(123)
# sample training data and fit model
train=base::sample(200,100,replace=F);

# svm from e1071
svmfit=svm(y~.,data=dat[train,],kernel="radial",gamma=1,cost=1);
```

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




```
# ksvm from kernlab
kernfit=ksvm(x[train,],y[train],type="C-svc",kernel="rbfdot",C=1,scaled=c())
plot(kernfit,data=x[train,])
```



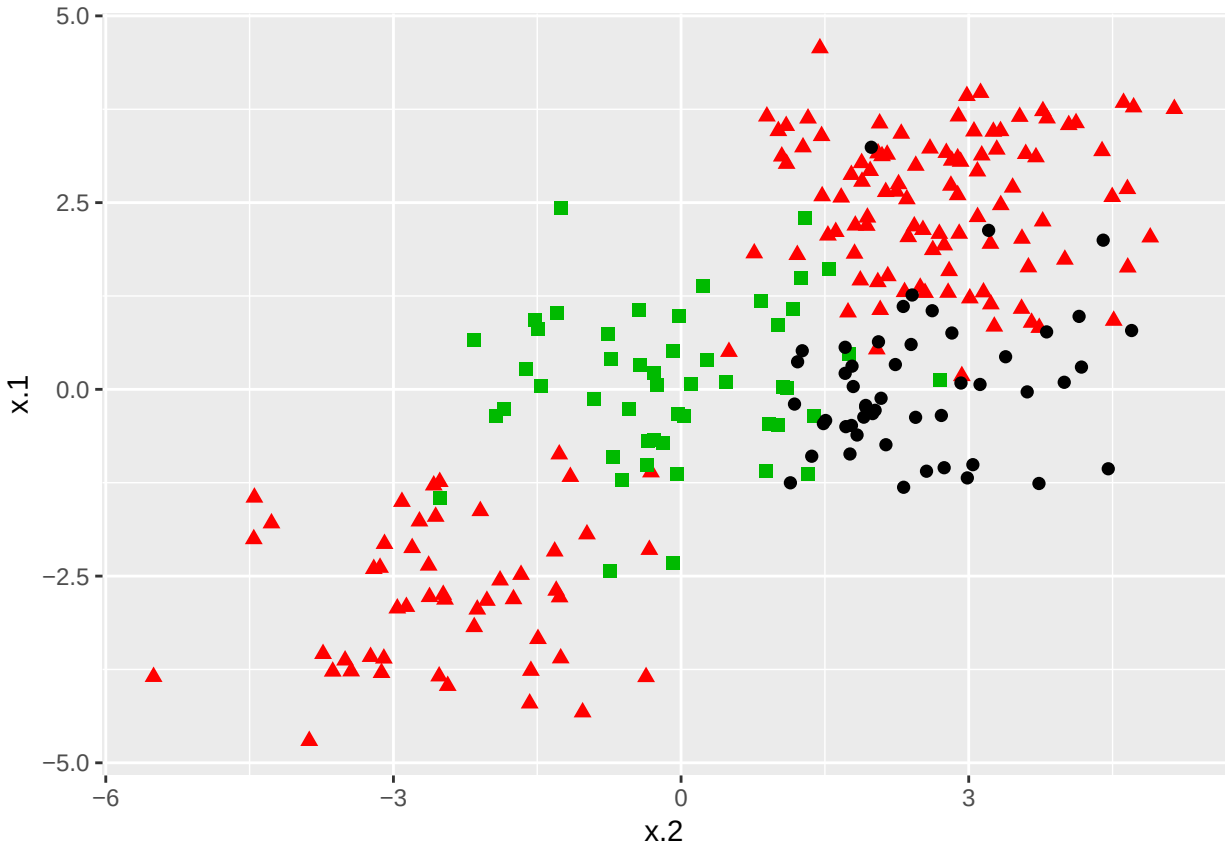

```
# The optimal cost
tune.out=tune(svm,y~., data=dat[train,],kernel="radial",
              ranges=list(cost=c(0.1,1,10,100,1000),gamma=c(0.5,1,2,3,4)))
tune.out$best.model
```

```
##
## Call:
## best.tune(method = svm, train.x = y ~ ., data = dat[train, ],
##     ranges = list(cost = c(0.1, 1, 10, 100, 1000), gamma = c(0.5,
##         1, 2, 3, 4)), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  0.5
##
## Number of Support Vectors:  34
```

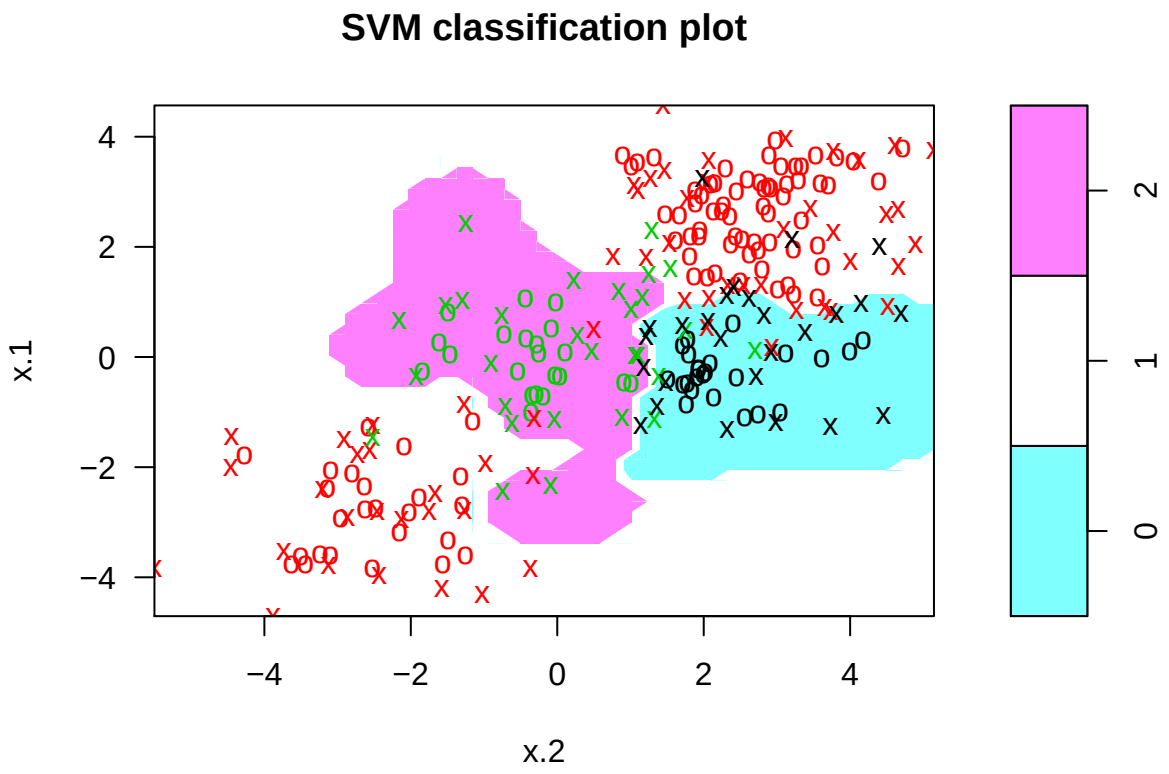
```
# Validate model performance
valid=table(true=dat[-train,"y"],pred=predict(tune.out$best.model))
```

Multiple Classes

```
x=rbind(x,matrix(rnorm(100),ncol=2))
y=c(y,rep(0,50))
x[y==0,2]=x[y==0,2]+2.5;
dat=data.frame(x=x,y=as.factor(y))
ggplot(data=dat,aes(x=x.2,y=x.1,color=y,shape=y))+geom_point(size=2)+
    scale_color_manual(values=c("#000000","#FF0000","#00BA00"))+theme(legend.position = "none")
```




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




```
# Check the performance
ypred=predict(svmfit,dat)
table(predict=ypred,truth=dat$y)
```

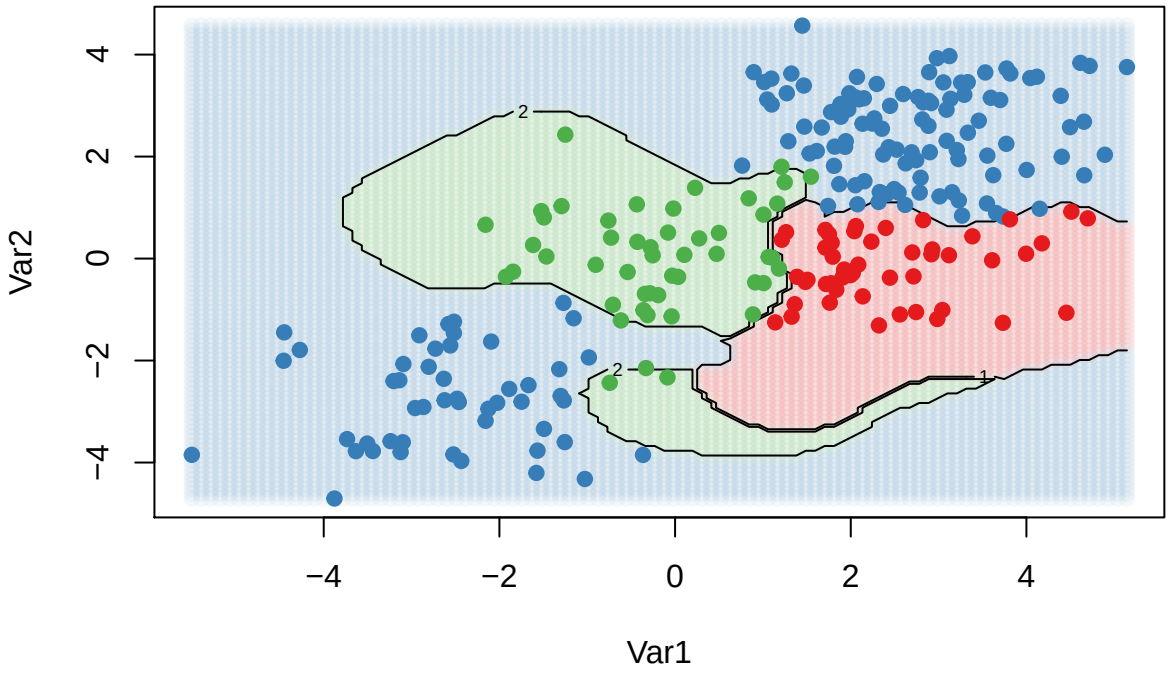
```
##         truth
## predict   0   1   2
##       0  43   3   4
##       1   5 144   3
##       2   2   3  43
```

```
kernfit=ksvm(x,y,type="C-svc",kernel="vanilladot",C=100);
```

```
##  Setting default kernel parameters
```

```
#plot(kernfit,data=x)

# The ksvm can fit more than 2 classes, but cannot plot the results. We have to create a grid of  # po

kernfit=ksvm(as.matrix(dat[,2:1]),dat$y,type="C-svc",kernel="rbfdot",C=100,scaled=c());

# Create a fine grid of the feature space
x.1=seq(min(dat$x.1),max(dat$x.1),length=100)
x.2=seq(min(dat$x.2),max(dat$x.2),length=100)
x.grid=expand.grid(x.2,x.1)

# predictions over grid points
pred=predict(kernfit,newdata=x.grid)
```

```
# plot the results
cols=brewer.pal(3,"Set1")
plot(x.grid, pch=19,col=adjustcolor(cols[pred],alpha.f=0.05))

classes=matrix(pred,nrow=100,ncol=100)
contour(x=x.2,y=x.1,z=classes,levels=1:3,lables="",add=T)
points(dat[,2:1],pch=19,col=cols[predict(kernfit)])
```



An application

The Khan data set contains data on 83 tissue samples with 2308 gene expression measurements on each sample. These were split into 63 training observations and 20 testing observations, and there are four distinct classes in the set. It would be impossible to visualize such data, so we choose the simplest classifier (linear) to construct our model. We will use the svm command from e1071 to conduct our analysis.

```
dat=data.frame(x=Khan$xtrain,y=as.factor(Khan$ytrain))
out=svm(y~.,data=dat,kernel="linear",cost=10)
# check model performance on training set
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
```

```
# Validation on test data

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