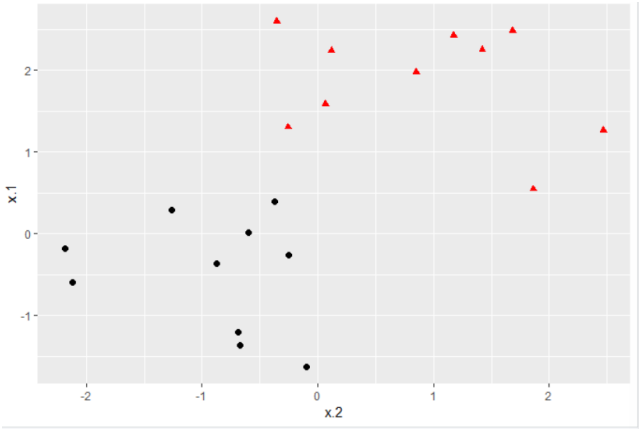
INFO523: R Exercise
Week 10 Chapter 8 Classification R Exercise: SVM

Goal: Practice basic R commands/methods for classification. If you are already familiar with some of the commands/methods, you can just practice the ones that are new to you.

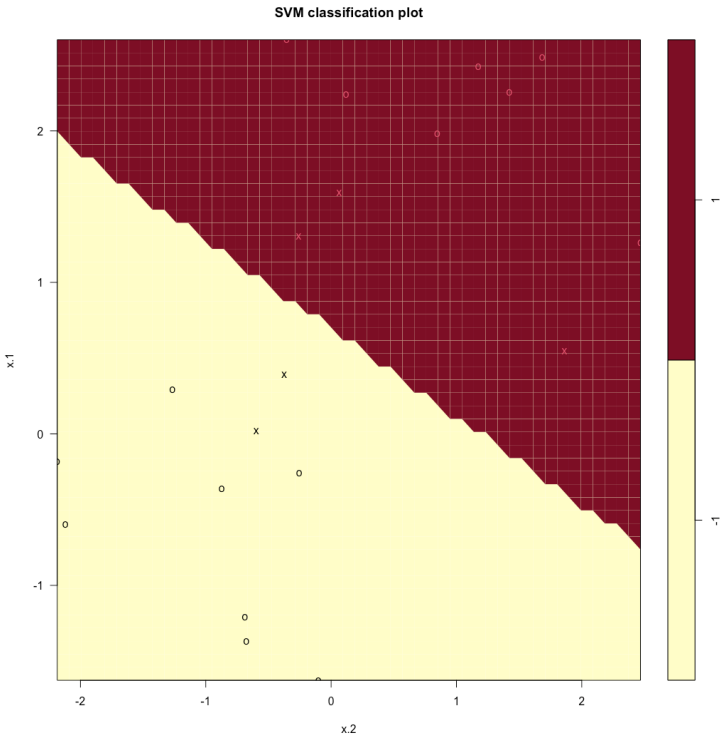
*> # Blue text are explanations.
*> Blue text are R commands/methods your will type into an R console.
*Black and Red text are output from R

```
> # SVM
> # set pseudorandom number generator
> set.seed(10)

> # Attach Packages
> install.packages("tidyverse")
> install.packages("kernlab")
> install.packages("e1071")
> install.packages("ISLR")
> install.packages("RColorBrewer")

> library(tidyverse)    # data manipulation and visualization
> library(kernlab)      # SVM methodology
> library(e1071)        # SVM methodology
> library(ISLR)         # contains example data set "Khan"
> library(RColorBrewer) # customized coloring of plots

> # set pseudorandom number generator
> set.seed(10)
> # Construct sample data set - completely separated
> # randomly generate 20 observations, one observation has two attributes
> x <- matrix(rnorm(20*2), ncol = 2)
> # created class labels: -1 and 1. They will be treated as factors, so any
two different labels would do, for example, 0 vs. 1
> y <- c(rep(-1,10), rep(1,10))
> # add 3/2 to the values in the 'y=1' class, so now the instances in the two
classes are separated
> x[y==1,] <- x[y==1,] + 3/2
> #combine x and factor y to make the dataframe for the classifier.
> dat <- data.frame(x=x, y=as.factor(y))
> # Plot data to see one can draw a line btw the two classes to separate them
> 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")
```

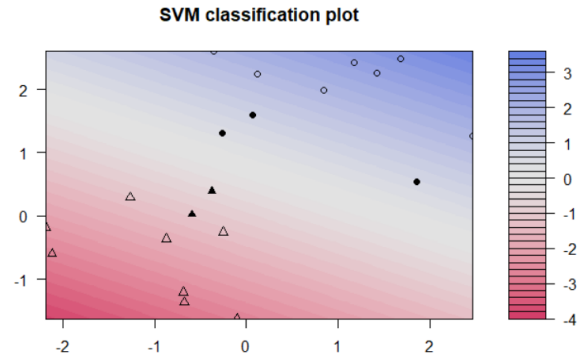



```
> # Fit Support Vector Machine model to data set
> svmfit <- svm(y~., data = dat, kernel = "linear")
> # Plot Results
> plot(svmfit, dat)
```

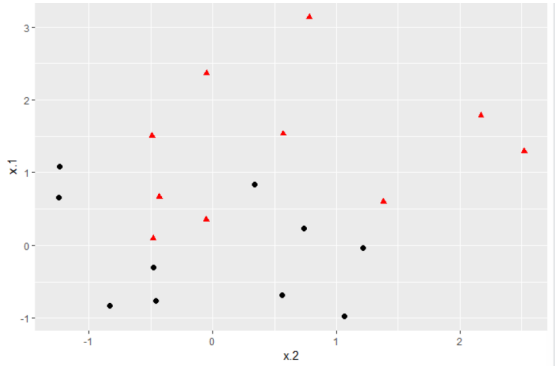



```
> #In the plot, points that are represented by an “X” are the support vectors, or the points that directly affect the classification line. The points marked with an “o” are the other points, which don’t affect the calculation of the line.
```

```
>
> # same result can be obtained using kernlab package
> # fit model and produce plot
> kernfit <- ksvm(x, y, type = "C-svc", kernel = 'vanilladot')
 Setting default kernel parameters
> plot(kernfit, data = x)
> #In this plat, a color gradient that indicates how confidently a new point
would be classified based on its features.
```

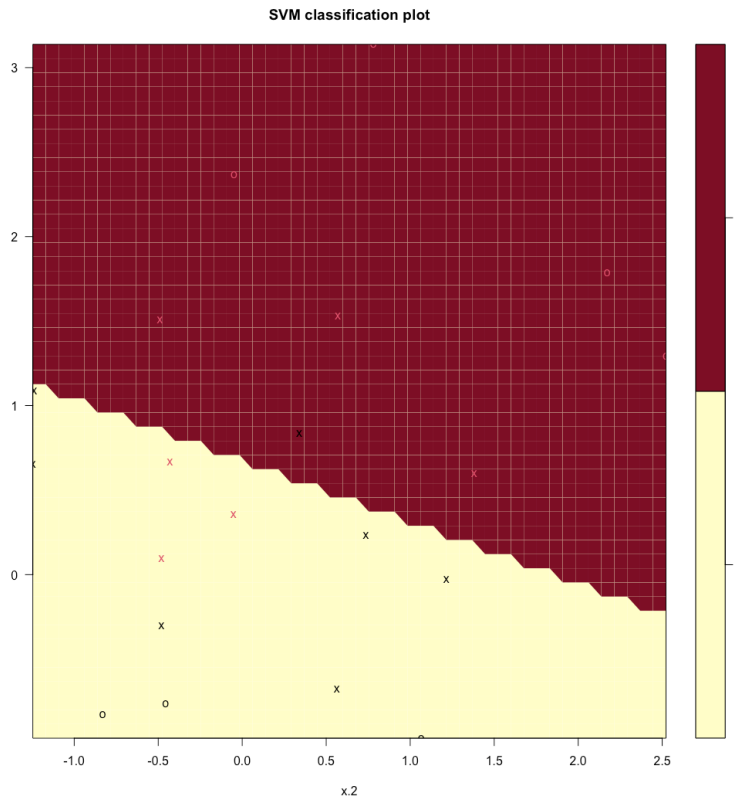



```
> #Next We simulate a new data set where the classes are more mixed.
> # Construct sample data set - not completely separated
> x <- matrix(rnorm(20*2), ncol = 2)
> y <- c(rep(-1,10), rep(1,10))
> x[y==1,] <- x[y==1,] + 1
> dat <- data.frame(x=x, y=as.factor(y))
>
> # Plot data set
> ggplot(data = 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")
```



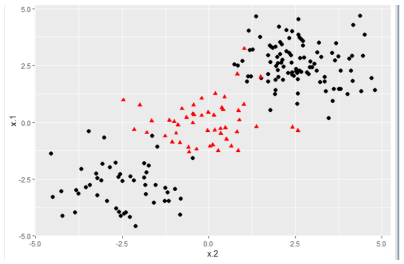

```
> # Fit Support Vector Machine model to data set using svm
> svmfit <- svm(y~., data = dat, kernel = "linear")
```

```
> # Plot Results
> plot(svmfit, dat)
> # We can see data points that are mis-classfied (both sides has Os and Xs)
```



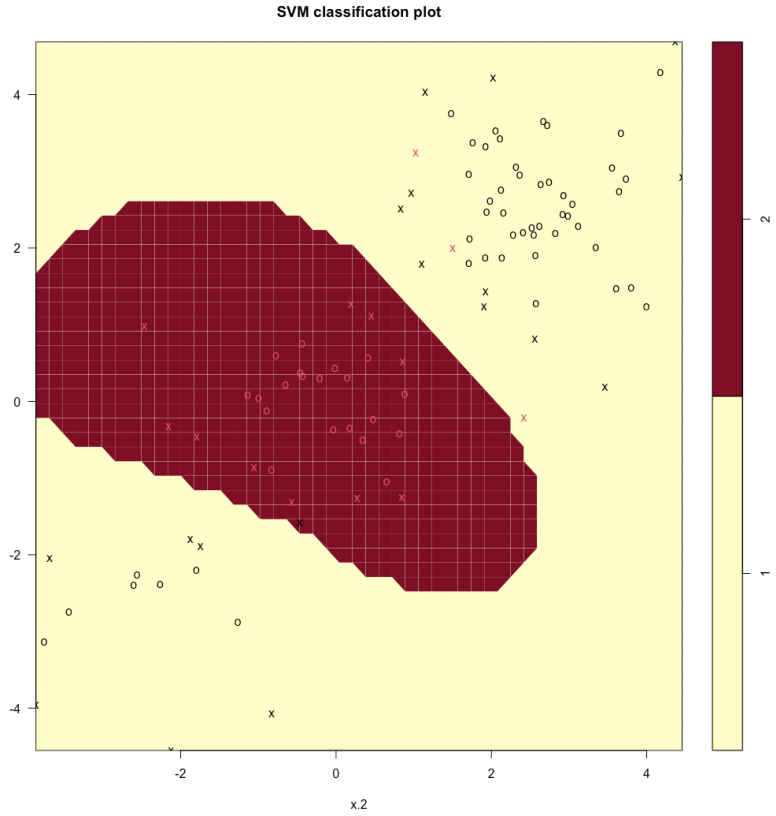

```
> #We can adjust the cost parameter to influence where the boundary is drawn
> # use tune() to test various costs and find the best fitting model
> # find optimal cost of misclassification
> tune.out <- tune(svm, y~., data = dat, kernel = "linear",
+                  ranges = list(cost = c(0.001, 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

Number of Support Vectors:  16

> # Create a table of misclassified observations
> ypred <- predict(bestmod, dat)
```

```
> (misclass <- table(predict = ypred, truth = dat$y))
       truth
predict -1 1
     -1  9 3
     1   1 7

> # Work further to use svm() to classify nonlinear data
> set.seed(123)
> # construct larger random data set
> x <- matrix(rnorm(200*2), ncol = 2)
> #first 100 data points are moved up by adding 2.5
> x[1:100,] <- x[1:100,] + 2.5
> #another 50 data points are moved down by taking away 2.5
> x[101:150,] <- x[101:150,] - 2.5
> #the last 50 didn't change their values
> #this creates a data "sandwich" as shown later in the scattered plot.
> #you can't draw a straight line to separate class 1 from class 2
>
> #we are now labeling the classes as 1 and 2.
> y <- c(rep(1,150), rep(2,50))
> dat <- data.frame(x=x,y=as.factor(y))
>
> # Plot data
> 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")
```

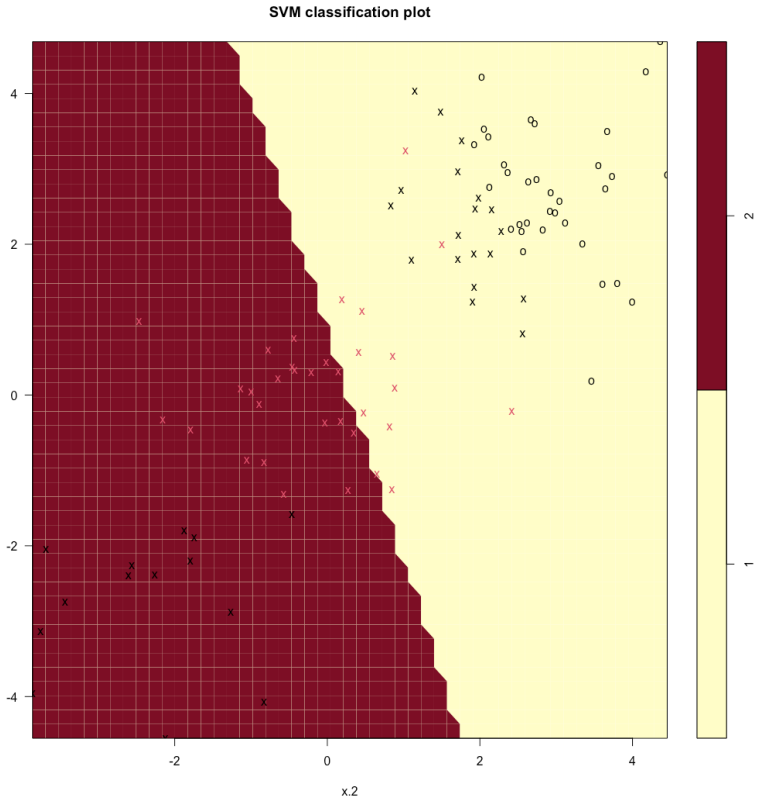


```
> # we'll take 100 random observations from the set as training example and
use them to
> #construct our boundary. We set kernel = "radial" based on the shape of our
data and plot the results.
>
> # set pseudorandom number generator
> set.seed(123)
> # sample training data and fit model
> # generate 100 random indices to select training data
```

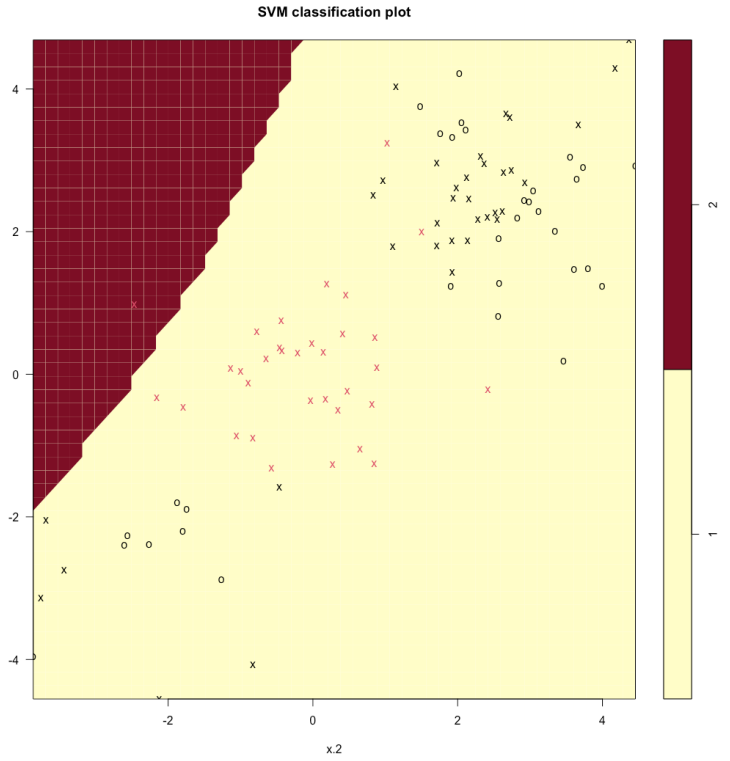
```
> train <- base::sample(200,100, replace = FALSE)
> svmfit <- svm(y~., data = dat[train,], kernel = "radial", gamma = 1, cost =
1)
> # plot classifier: looks good
> plot(svmfit, dat[train,])
```



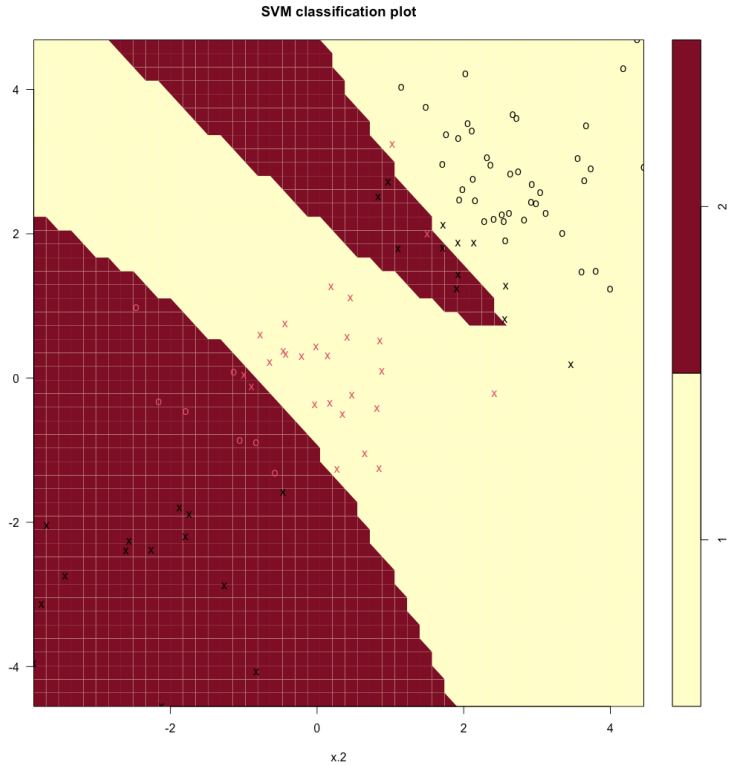

```
> # try other kernels, they don't work as well as the radial (which is good
for "sandwich" like data)
> svmfit <- svm(y~., data = dat[train,], kernel = "linear", gamma = 1, cost =
1)
> plot(svmfit, dat[train,]) #predicted all points as class 1
```




```
>
> svmfit <- svm(y~., data = dat[train,], kernel = "polynomial", gamma = 1,
cost = 1)
> plot(svmfit, dat[train,]) #predicted all points as class 1
```

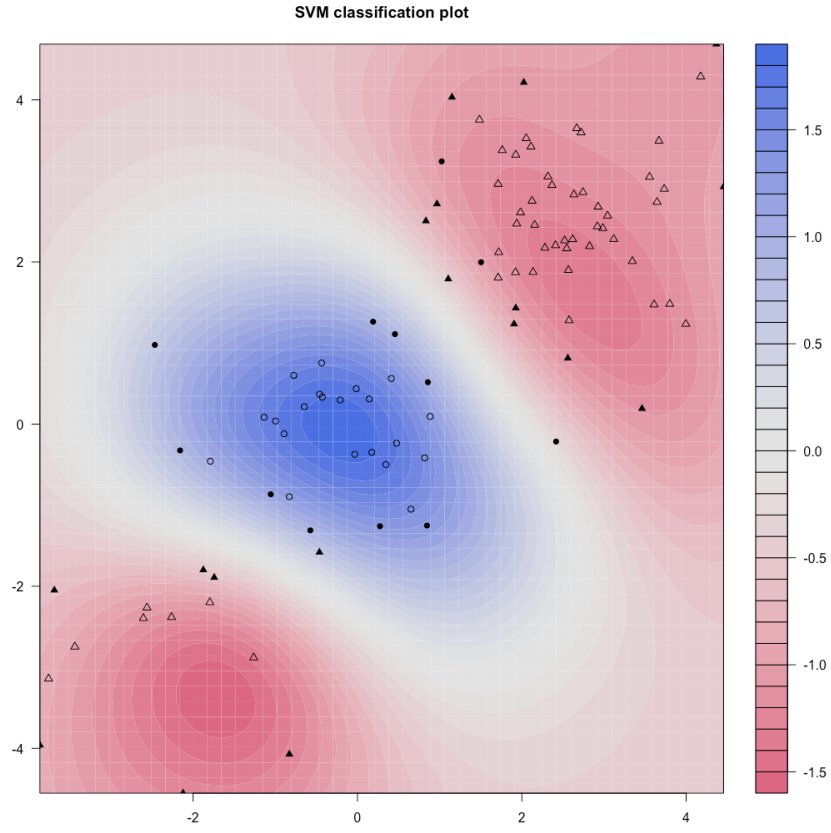



```
> svmfit <- svm(y~., data = dat[train,], kernel = "sigmoid", gamma = 1, cost =
1)
> plot(svmfit, dat[train,])
```

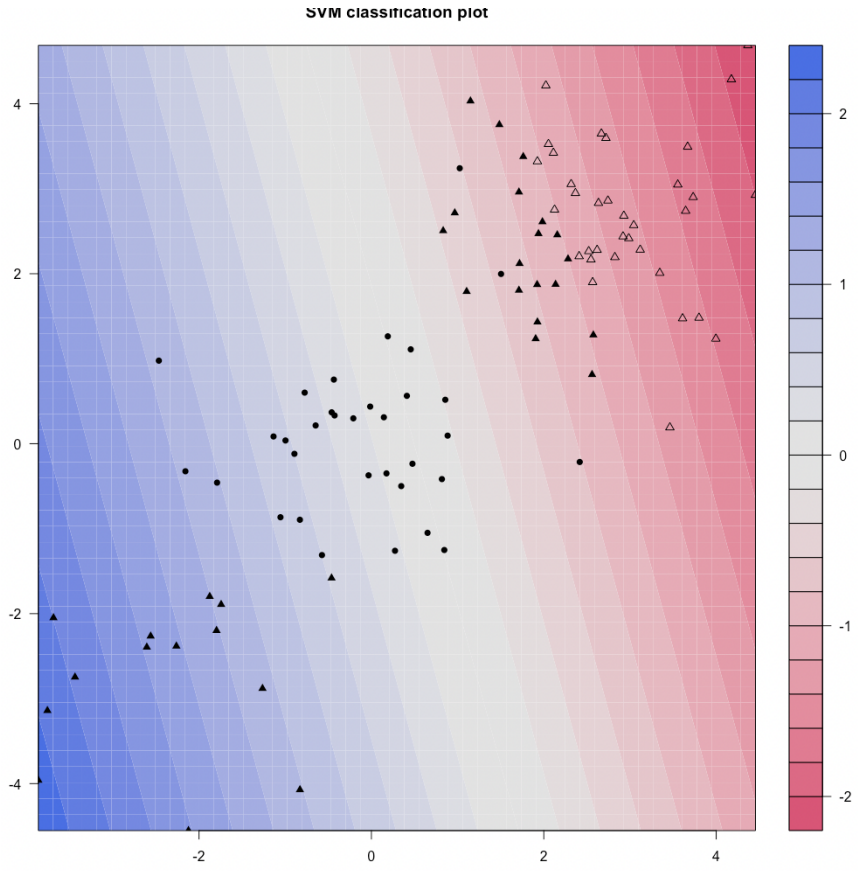



```
> # Fit radial-based SVM in kernlab. ksvm has more kernels to choose from
```

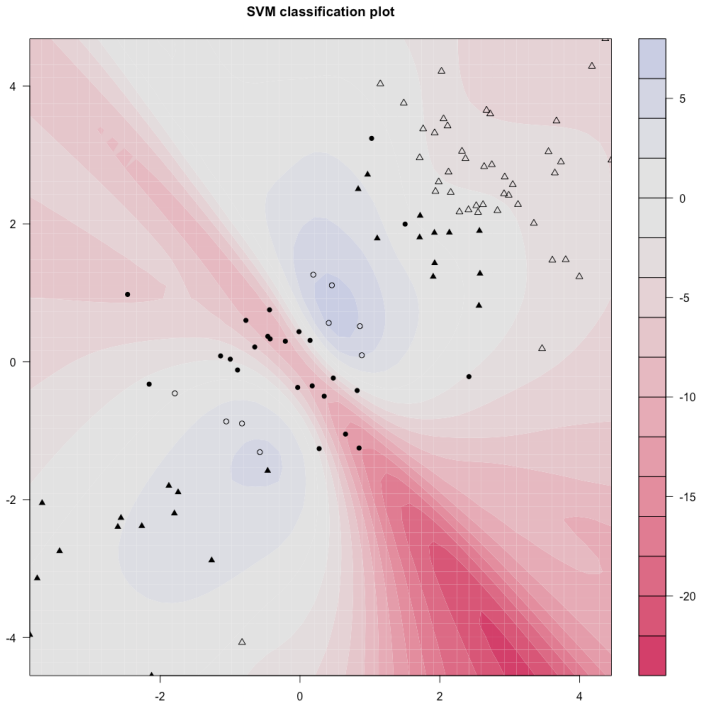
```
> kernfit <- ksvm(x[train,],y[train], type = "C-svc", kernel = 'rbfdot', C =
1, scaled = c())
> # Plot training data
> plot(kernfit, data = x[train,])
```



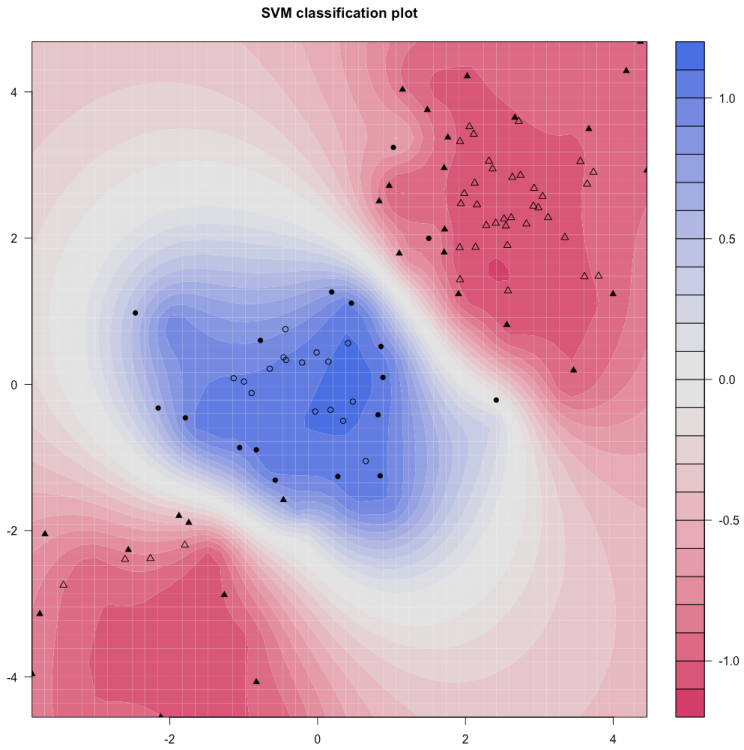

```
> #try other kernels, most bad except for lapacedot
> kernfit <- ksvm(x[train,],y[train], type = "C-svc", kernel = 'polydot', C =
1, scaled = c())
 Setting default kernel parameters
> plot(kernfit, data = x[train,])
```



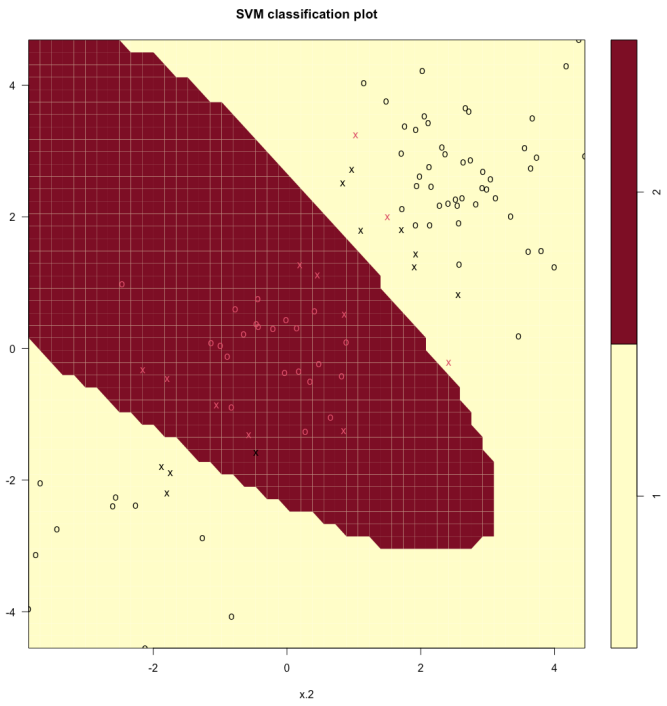

```
> kernfit <- ksvm(x[train,],y[train], type = "C-svc", kernel = 'vanilladot', C
= 1, scaled = c())
 Setting default kernel parameters
> plot(kernfit, data = x[train,])

> kernfit <- ksvm(x[train,],y[train], type = "C-svc", kernel = 'tanhdot', C =
1, scaled = c())
 Setting default kernel parameters
> plot(kernfit, data = x[train,])
```




```
> #laplacedot looks good
> kernfit <- ksvm(x[train,],y[train], type = "C-svc", kernel = 'laplacedot', C
= 1, scaled = c())
> plot(kernfit, data = x[train,])
```



```
> kernfit <- ksvm(x[train,],y[train], type = "C-svc", kernel = 'anovadot', C =
1, scaled = c())
 Setting default kernel parameters
> plot(kernfit, data = x[train,])

> kernfit <- ksvm(x[train,],y[train], type = "C-svc", kernel = 'splinedot', C
= 1, scaled = c())
 Setting default kernel parameters
> plot(kernfit, data = x[train,])


> kernfit <- ksvm(x[train,],y[train], type = "C-svc", kernel = 'besseldot', C
= 1, scaled = c())
 Setting default kernel parameters
> plot(kernfit, data = x[train,])

> # can we tune it and make it better?
> 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(0.001, 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[train, ], ranges = list(cost = c(0.001,
    0.01, 0.1, 1, 5, 10, 100), gamma = c(0.001, 0.01, 0.1, 1, 5, 10, 100)), kernel = "radial")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  10

Number of Support Vectors:  22

> plot(bestmod, dat[train, ])
```




```
> # validate model performance on test data
> ypred <- predict(bestmod, dat[-train, ])
> (valid <- table(true = dat[-train,"y"], pred = ypred))
    pred
true  1  2
   1 80  3
   2  2 15

>
>
> # mutiple class classification using SVM is very similar to the binary
classification examples shown above
> #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
start with the simplest classifier (linear)
> #to construct our model. We will use the svm command from e1071 to conduct
our analysis.
>
> # fit model (you can always use tune() to find the best parameter)
> dat <- data.frame(x = Khan$xtrain, y=as.factor(Khan$ytrain))
> (out <- svm(y~., data = dat, kernel = "linear", cost=10))
```

```
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


> # check model performance on training set: 100% precision and recall
> 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
>
> # validate model performance
> 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) #classified 2 instances wrong -- good performance

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
```

[ADVANCED]

1. Try out SVM on your dataset and report performances using two different sets of parameters.
2. Use Performance Estimates learned last week to find a set of best parameters for your SVM.