Labs: Support Vector Machines (SVM) and continue Trees, Project Progress Update

Thilanka Munasinghe

Data Analytics

ITWS-4600/ITWS-6600/MATP- 4450/CSCI-4960

Group 4, Lab 9 - November 11th, 2022

Rpart – recursive partitioning and Conditional Inference

Reminder to go over these code snippets…

group3/lab1_rpart1.R

group3/lab1_rpart2.R

group3/lab1_rpart3.R

group3/lab1_rpart4.R

Try rpart for “Rings” on the Abalone dataset

group3/lab1_ctree1.R

group3/lab1_ctree2.R

group3/lab1_ctree3.R

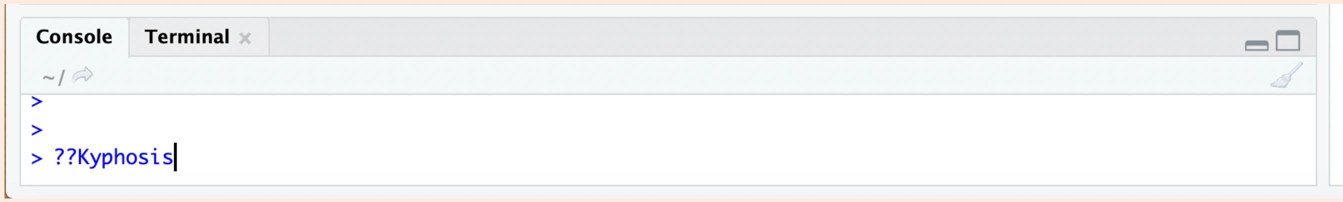
RandomForest

Go over the:

group3/lab1_randomforest1.R

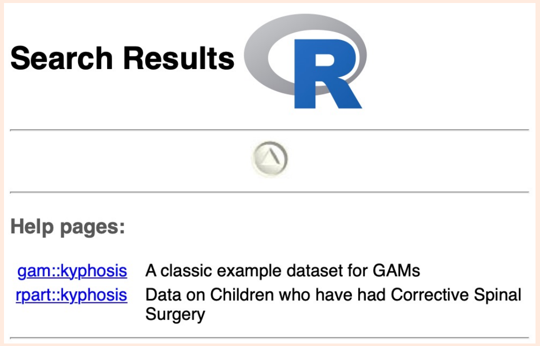
On Rstudio Consol, type:

??Kyphosis



```
>
>
> ??Kyphosis
```

Read:



Search Results

Help pages:

gam::kyphosis	A classic example dataset for GAMs
rpart::kyphosis	Data on Children who have had Corrective Spinal Surgery

rpart::kyphosis	Data on Children who have had Corrective Spinal Surgery

Trees for the Titanic

Reminder to finish this…

data(Titanic)

Conduct the,

rpart, ctree, hclust, randomForest for: Titanic dataset

Survived ~ .

svm() in e1071

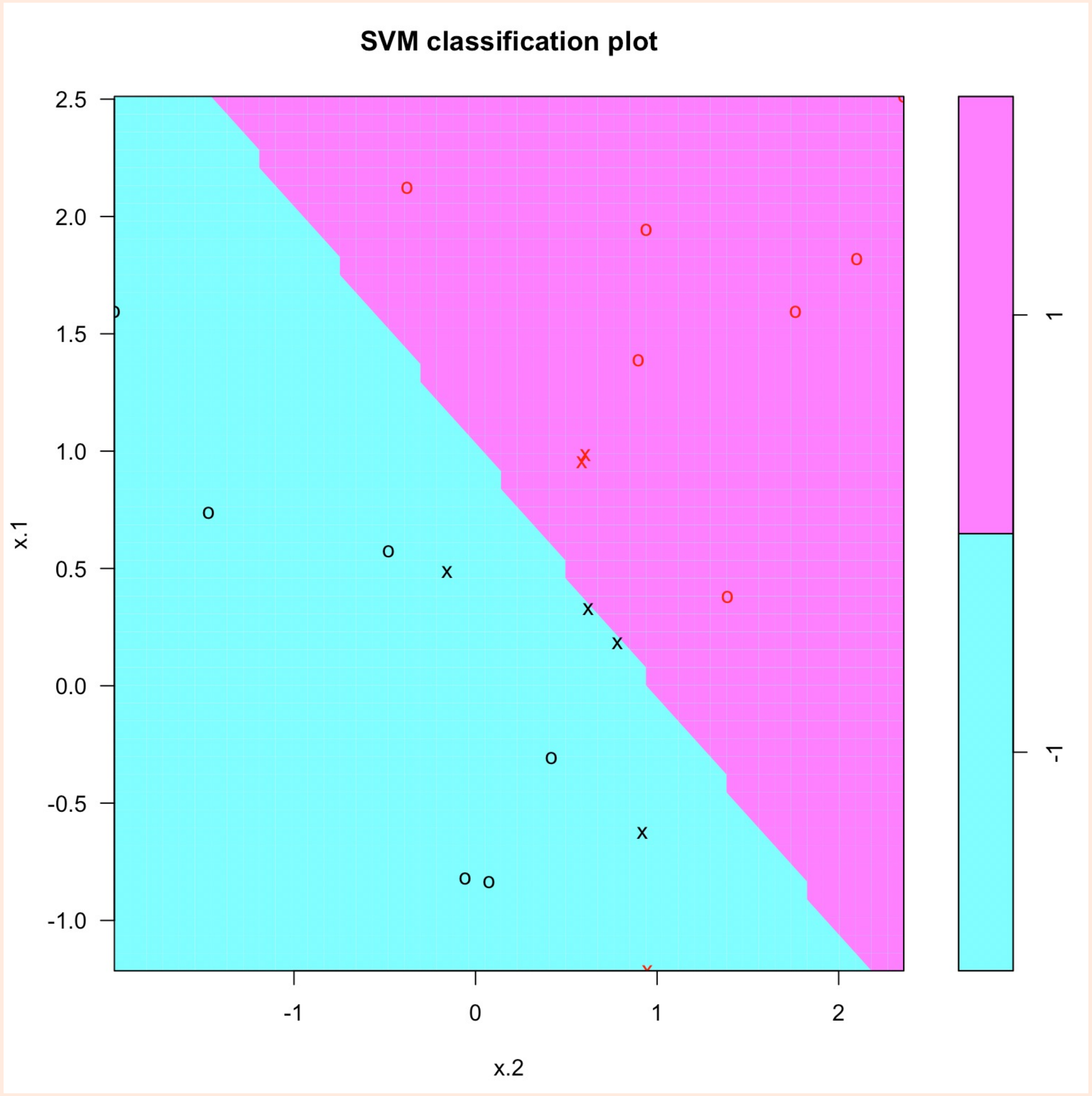
- The e1071 library contains implementations for a number of statistical learning methods
- In particular, the svm() function can be used to fit a support vector classifier when the argument kernel="linear" is used.
- A *cost* argument allows us to specify the *cost of a violation to the margin*.
- When the *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.

- We now use the svm() function to fit the support vector classifier for a given value of the cost parameter.
- Here we demonstrate the use of this function on a two-dimensional example so that we can plot the resulting decision boundary. We begin by generating the observations, which belong to two classes.

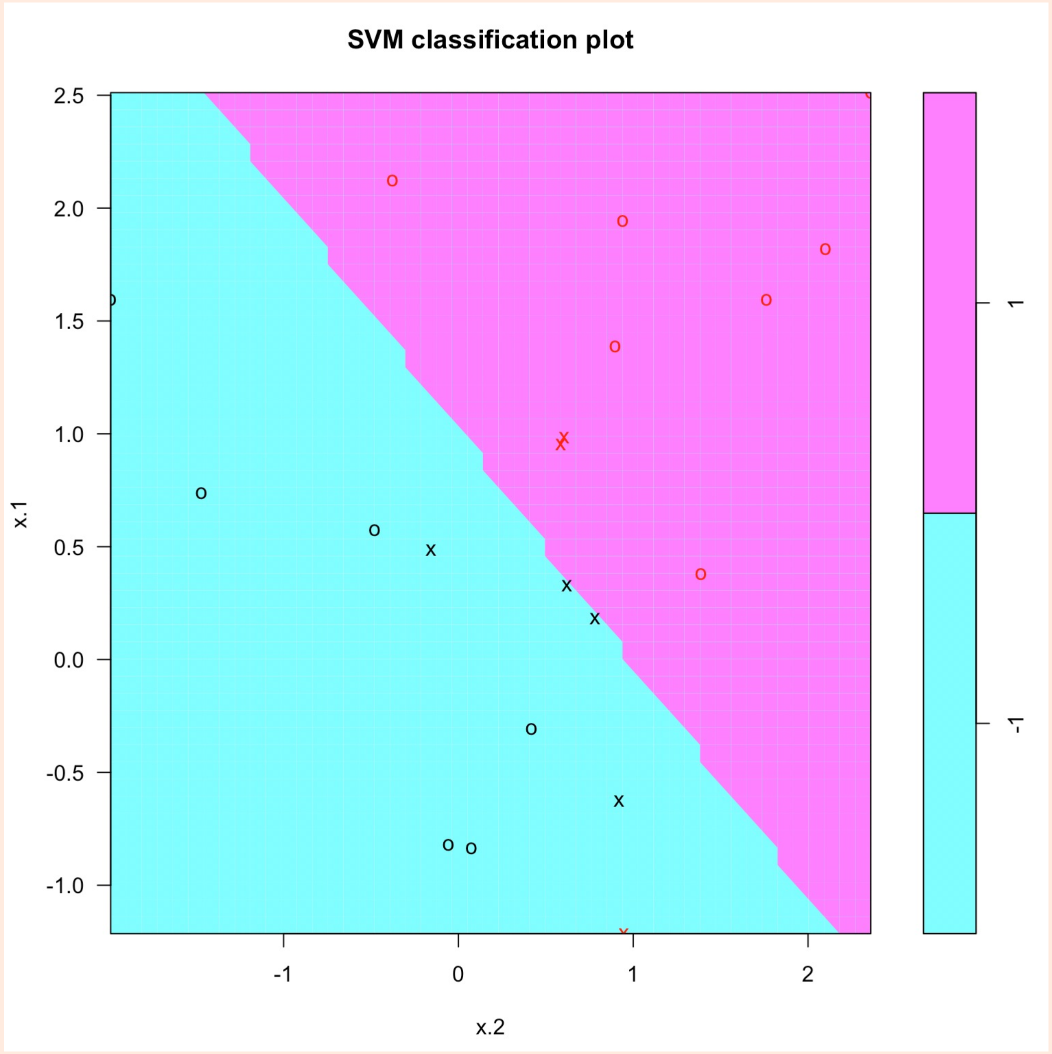
```
library(e1071)
set.seed (1)
# We now use the svm() function to fit the support vector classifier for a given value of the cost parameter.
# Here we demonstrate the use of this function on a two-dimensional example so that we can plot the resulting
# decision boundary.
# We begin by generating the observations, which belong to two classes.
x=matrix(rnorm(20*2), ncol=2)
y=c(rep(-1,10), rep(1,10))
x[y==1,]=x[y==1,] + 1
x
y
# We begin by checking whether the classes are linearly separable.
plot(x, col=(3-y))
# They are not. Next, we fit the support vector classifier.
# Note that in order for the svm() function to perform classification
# we must encode the response as a factor variable.
# We now create a data frame with the response coded as a factor.
dat <- data.frame(x = x,y  = as.factor(y))
svmfit <- svm(y ~., data=dat, kernel="linear", cost=10,scale=FALSE)
# 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.

# We can now plot the support vector classifier obtained:
plot(svmfit , dat)
# Note that the two arguments to the plot.svm() function are the output of the call to svm(),
#as well as the data used in the call to svm().
# The region of feature space that will be assigned to the −1 class is shown in light blue,
# and the region that will be assigned to the +1 class is shown in purple.
```

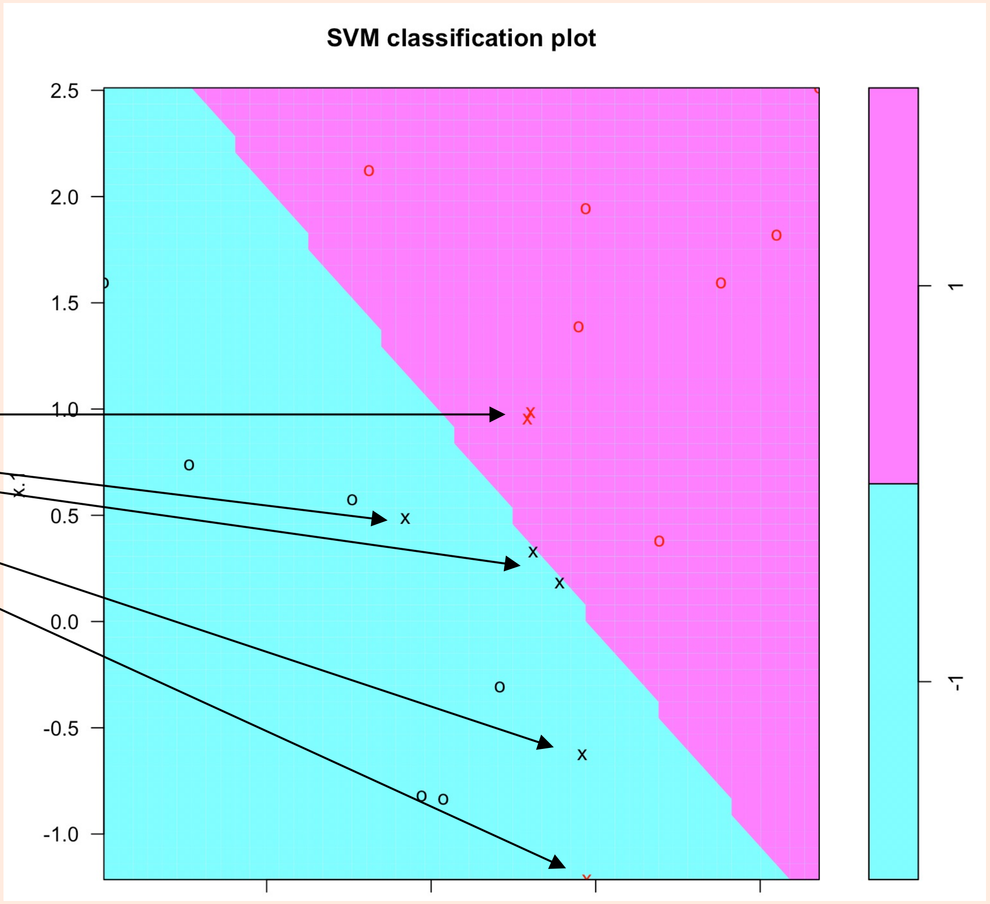
plot(svmfit , dat)



- The decision boundary between the two classes is linear (because we used the argument kernel="linear"), though due to the way in which the plotting function is implemented in this library the decision boundary looks somewhat jagged in the plot. We see that in this case only one observation is misclassified. (Note that here the second feature is plotted on the x-axis and the first feature is plotted on the y-axis, in contrast to the behavior of the usual plot() function in R.)



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



```
# We can determine the identities of those support vectors by:
svmfit$index
# You can see 1,2,5,7,14,16 and 17

# We can obtain some basic information about the support vector classifier fit using the summary()
command:
summary(svmfit)
```

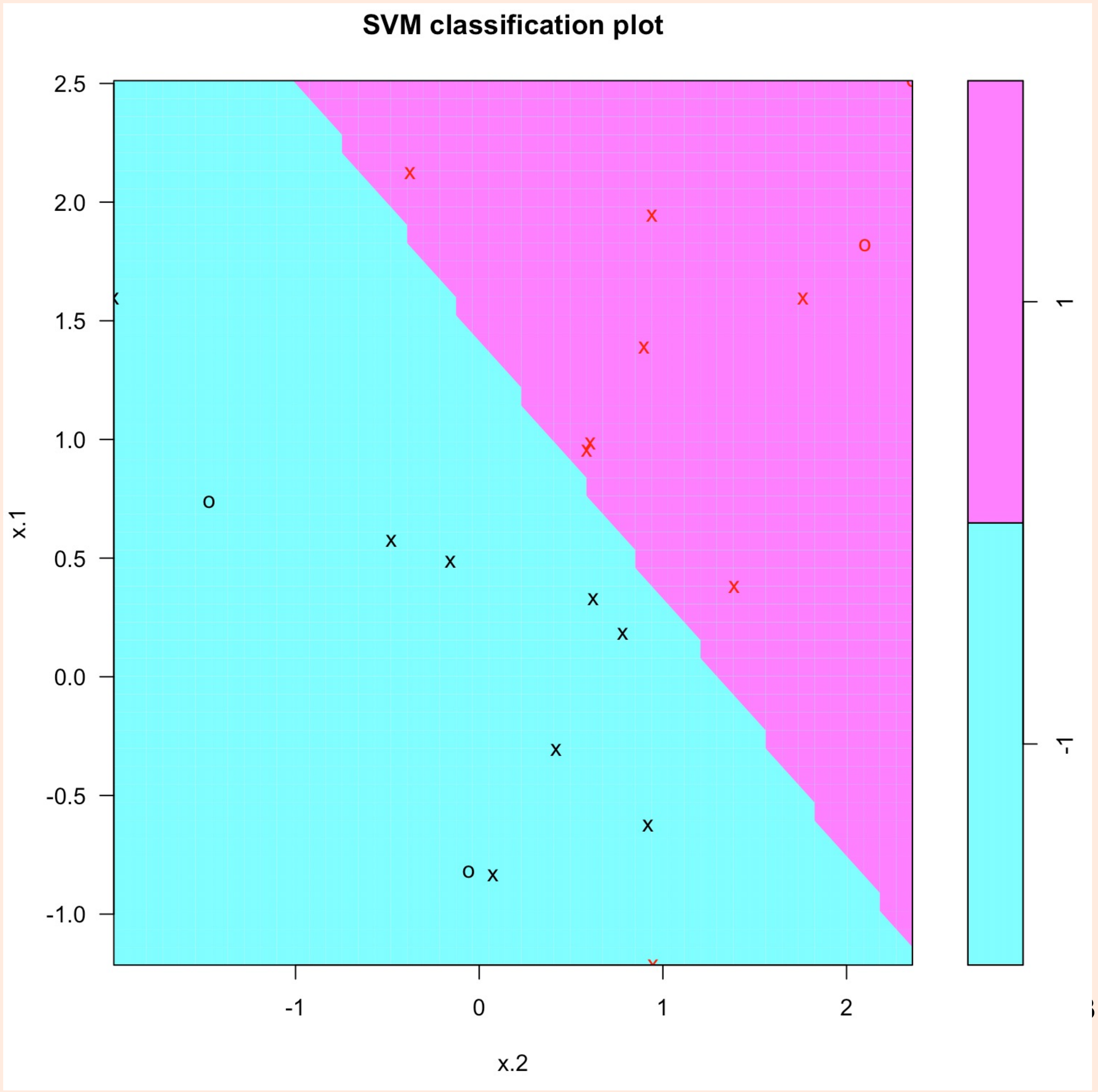
This tells us, for instance, that a linear kernel was used with cost=10,
and that there were seven support vectors, four in one class and three in the other.

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

# now cost = 0.1

```
svmfit <- svm(y ~., data=dat, kernel="linear", cost = 0.1, scale=FALSE)
plot(svmfit , dat)
svmfit$index
```

# cost = 0.1



- Now that a smaller value of the cost (cost =0.1) parameter is being used, we obtain a larger number of support vectors, because the margin is now wider.
- Unfortunately, the svm() function does not explicitly output the coefficients of the linear decision boundary obtained when the support vector classifier is fit, nor does it output the width of the margin.

tune() function

- The e1071 library includes a built-in function, tune(), to perform cross-validation.
- By default, tune() performs ten-fold cross-validation on a set of models of interest. In order to use this function, we pass in relevant information about the set of models that are under consideration.
- The following command indicates that we want to compare SVMs with a linear kernel, using a range of values of the cost parameter.

Continue…

```
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)))
# We can easily access the cross-validation errors for each of these models using the summary() command:
summary(tune.out)
# We see that cost=0.1 results in the lowest cross-validation error rate.
# The tune() function stores the best model obtained, which can be accessed as follows:
bestmod=tune.out$best.model
summary(bestmod)
# The predict() function can be used to predict the class label on a set of test observations,
# at any given value of the cost parameter. We begin by generating a test data set.
xtest=matrix(rnorm(20*2), ncol=2)
ytest=sample(c(-1,1), 20, rep=TRUE)
xtest[ytest==1,]=xtest[ytest==1,] + 1
testdat=data.frame(x=xtest, y=as.factor(ytest))
# Now we predict the class labels of these test observations.
# Here we use the best model obtained through cross-validation in order to make predictions.
ypred <-predict(bestmod ,testdat)
table(predict=ypred, truth=testdat$y)

#Thus, with this value of cost, 19 of the test observations are correctly classified.
# What if we had instead used cost= 0.01?
svmfit <- svm(y~., data=dat, kernel="linear", cost=.01, scale=FALSE)
ypred=predict(svmfit ,testdat)
table(predict=ypred, truth=testdat$y)

# In this case one additional observation is misclassified.
# Now consider a situation in which the two classes are linearly separable.
# Then we can find a separating hyperplane using the svm() function.
# We first further separate the two classes in our simulated data so that they are linearly separable:
x[y==1,]=x[y==1,]+0.5
plot(x, col=(y+5)/2, pch=19)
```

Continue…

```
# Now the observations are just barely linearly separable.
# We fit the support vector classifier and plot the resulting hyperplane,
# using a very large value of cost so that no observations are misclassified.
dat=data.frame(x=x,y=as.factor(y))
svmfit <-svm(y~., data=dat, kernel="linear", cost=1e5)
summary(svmfit)
plot(svmfit,dat)

# No training errors were made and only three support vectors were used.
# However, we can see from the figure that the margin is
# very narrow (because the observations that are not support vectors, indicated as circles, are very
# close to the decision boundary). It seems likely that this model will perform poorly on test data.
# We now try a smaller value of cost:

svmfit <- svm(y~., data=dat, kernel="linear", cost=1)
summary(svmfit)
plot(svmfit ,dat)
# Using cost=1, we misclassify a training observation, but we also obtain a much wider margin and make
# use of seven support vectors.
# It seems likely that this model will perform better on test data than the model with cost=1e5.
```

SVM Application to Gene Expression Dataset

```
# We now examine the Khan data set, which consists of a number of tissue samples
# corresponding to four distinct types of small round blue cell tumors
# For each tissue sample, gene expression measurements are available.
#The data set consists of training data, xtrain and ytrain, and testing data, xtest and ytest.
library(e1071)
library(ISLR)
names(Khan)
# Let's examine the dimension of the data:
# This data set consists of expression measurements for 2,308 genes.
# The training and test sets consist of 63 and 20 observations respectively
dim(Khan$xtrain )
dim(Khan$xtest )


length(Khan$ytrain )
length(Khan$ytest )
table(Khan$ytrain )
table(Khan$ytest )

# We will use a support vector approach to predict cancer subtype 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.
dat <- data.frame(x=Khan$xtrain , y = as.factor(Khan$ytrain ))
out <- svm(y ~., data=dat, kernel="linear",cost=10)
summary(out)

# 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.

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)
# We see that using cost=10 yields two test set errors on this data.
```

Work through these…

- lab1_svm1.R –> lab1_svm11.R

  script fragments in R available in course repository inside the RPI Box in Group 3: **https://rpi.box.com/s/kk6f5gp9oq56wicgdvilgw5cokibh7uu**

- Karatzoglou et al. 2006 - https://rpi.box.com/s/nowbkb2ursqlsoj86lczmcjwjwl5tl9o

kernlab

Read this article that is available in the course repository box.

https://rpi.box.com/s/2rp1gvuy2a9yjj0q4o53ox91d8jpgcuy

Read the kernlab documentation: https://www.rdocumentation.org/packages/kernlab/versions/0.9-27

Read the documentation for ksvm() function

https://www.rdocumentation.org/packages/kernlab/versions/0.9-27/topics/ksvm

- Work on the SVM example shared in the course repository box:
- Linear SVM example using kernlab package:

```
# load the kernlab package
library(kernlab)
```

https://rpi.box.com/s/2rp1gvuy2a9yjj0q4o53ox91d8jpgcuy

- Read the documentation for svmpath() function

https://www.rdocumentation.org/packages/svmpath/versions/0.955/topics/svmpath

kernlab, svmpath and klaR

Support Vector Machines in R:

- https://rpi.box.com/s/nowbkb2ursqlsoj86lczmcjwjwl5tl9o
- Start at page 9 (bottom)
- Work on the examples that available in the above article by running them on Rstudio.

Project: One-on-One Session

Project Check-in during the Lab:

One-on-One session with the Instructor Project Progress Updates during the Labs. <u>You Must check-in with the instructor before end of the class.</u>

- You need to meet with the instructor and <u>**show your current progress and the development**</u> of your term project (Assignment6).
- We will document your current progress

Ukraine Twitter Project

- We are labeling Ukraine Twitter dataset
- If you are interested help labeling ~40-50 Tweets, please contact me (instructor) via email.
- We appreciate your help with labeling process! ☺