Gov 2018: Lab 7 Support Vector Machine

Your name:

Tuesday March 8, 2022

Overview

This exercise is based on UC Business Analytics R Programming Guide.

```
# Load Packages
library(tidyverse)    # data manipulation and visualization
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.0.2     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(e1071)        # SVM methodology
library(ISLR)         # contains example data set "Khan"
```

Question 1 Maximal Margin Classifier

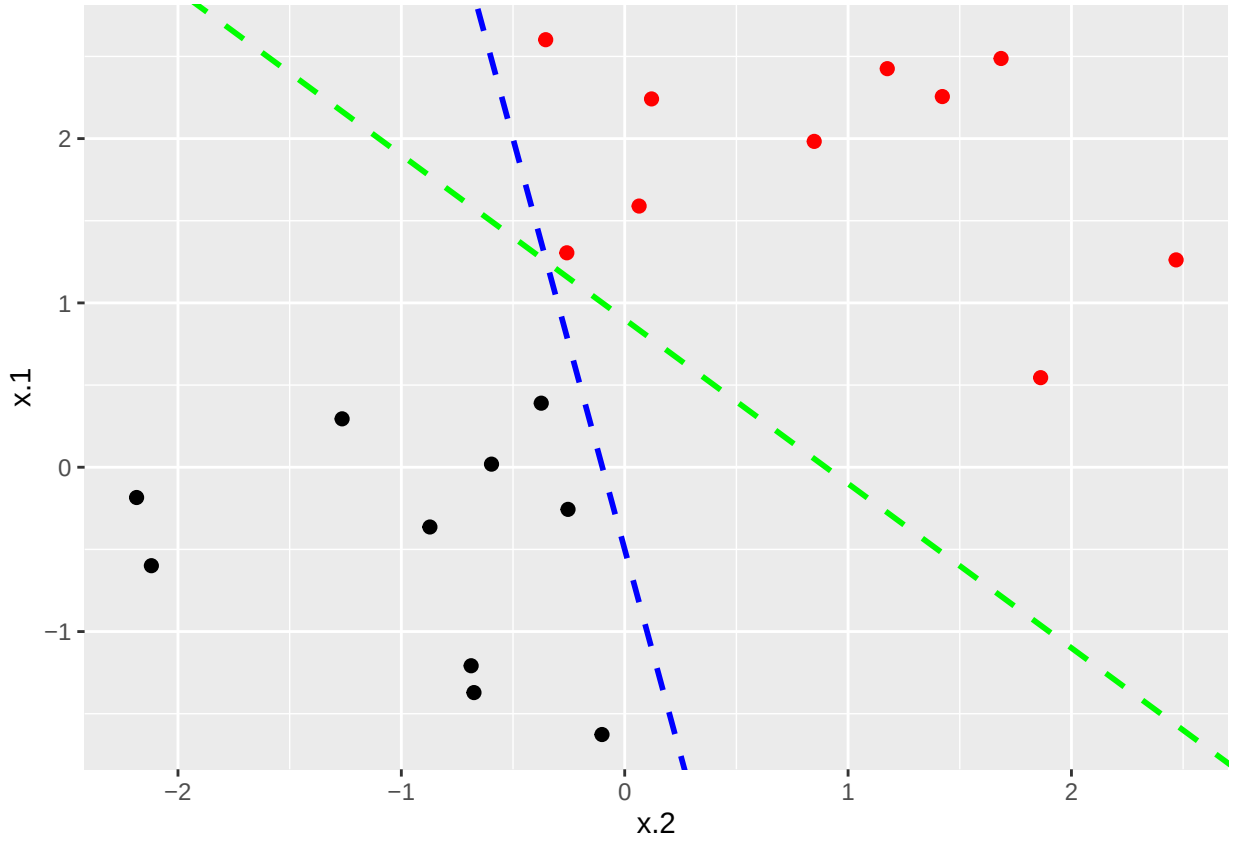
If the classes are separable by a linear boundary, we can use a Maximal Margin Classifier to find the classification boundary. To visualize an example of separated data, we generate 40 random observations and assign them to two classes.

Upon visual inspection, we can see that infinitely many lines exist that split the two classes. Below, we draw two example classification boundaries, and draw line segments from each observation to the boundary lines. The goal of the maximal margin classifier is to identify the linear boundary that maximizes the total distance between the line and the closest point in each class.

Run the codes below and observe the difference between two example classification boundaries.

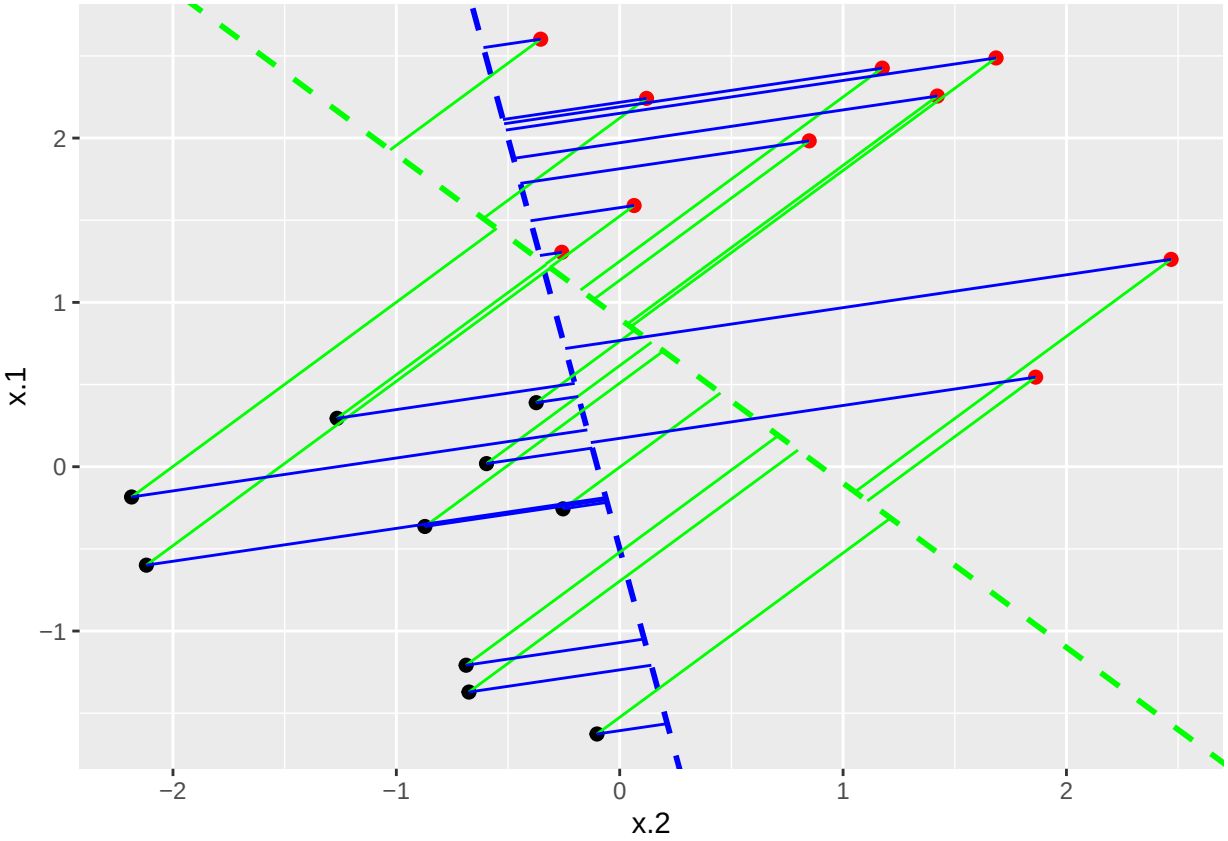
```
set.seed(10)

# Construct sample data set - completely separated
x <- matrix(rnorm(20*2), ncol = 2)
y <- c(rep(-1,10), rep(1,10))
x[y==1,] <- x[y==1,] + 3/2L
dat <- data.frame(x=x, y=as.factor(y))

# Plot data
```

```r
ggplot(data = dat, aes(x = x.2, y = x.1, color = y)) +
  geom_point(size = 2) +
  scale_color_manual(values=c("#000000", "#FF0000")) +
  theme(legend.position = "none") +
  geom_abline(intercept = 0.9, slope = -1, color = "green",
                linetype = "dashed", size = 1) + # example classification boundary 1
  geom_abline(intercept = -0.5, slope = -5, color = "blue", # example classification boundary 2
                linetype = "dashed", size = 1)
```




```r
# Draw perpenticular lines to the boundary lines
perp.segment.coord <- function(x0, y0, a=0, b=1){
 #finds endpoint for a perpendicular segment from the point (x0,y0) to the line
 # defined by lm.mod as y=a+b*x
  x1 <- (x0+b*y0-a*b)/(1+b^2)
  y1 <- a + b*x1
  list(x0=x0, y0=y0, x1=x1, y1=y1)
}

ss1 <- perp.segment.coord(dat$x.2, dat$x.1, 0.9, -1)
ss2 <- perp.segment.coord(dat$x.2, dat$x.1, -0.5, -5)

ggplot(data = dat, aes(x = x.2, y = x.1, color = y)) +
  geom_point(size = 2) +
  scale_color_manual(values=c("#000000", "#FF0000")) +
  theme(legend.position = "none") +
  geom_abline(intercept = 0.9, slope = -1, color = "green",
                linetype = "dashed", size = 1) + # example classification boundary 1
```

```
  geom_abline(intercept = -0.5, slope = -5, color = "blue", # example classification boundary 2
              linetype = "dashed", size = 1) +
  geom_segment(data=as.data.frame(ss1), aes(x = x0, y = y0, xend = x1, yend = y1), colour = "green") +
    geom_segment(data=as.data.frame(ss2), aes(x = x0, y = y0, xend = x1, yend = y1), colour = "blue")
```



Now, use the `svm()` function in the `e1071` package to find the linear boundary that maximizes the distance.

```
svmfit <- svm(..., # TODO: specify the arguments
              kernel = "linear", scale = FALSE)
print(svmfit)
```

Visualize the results using `plot()`. Recall that the points that fall on the margin and thus affect the classification line are called the **support vectors**. Which points in your plot are support vectors?

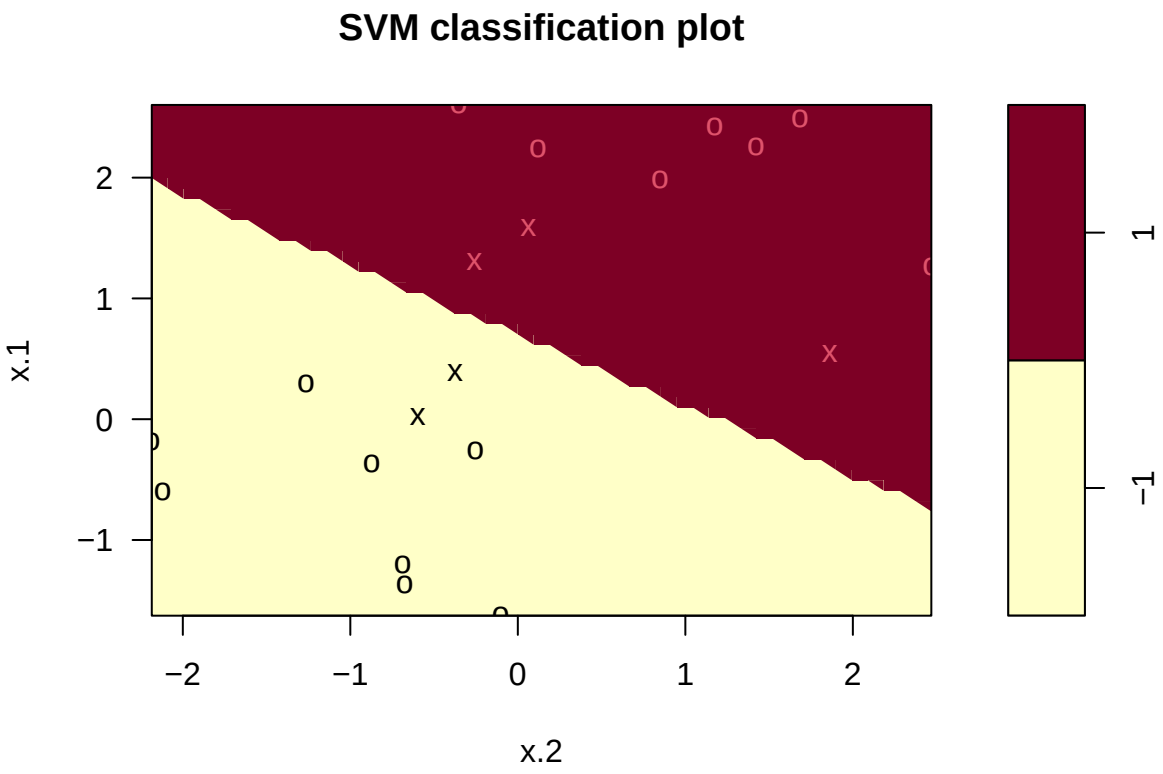
```
plot(svmfit, dat)
```

```
# Fit model

svmfit <- svm(y~., data = dat, kernel = "linear", scale = FALSE)
print(svmfit)
```

```
##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
```

```
##        cost:  1
##
## Number of Support Vectors:  5
```

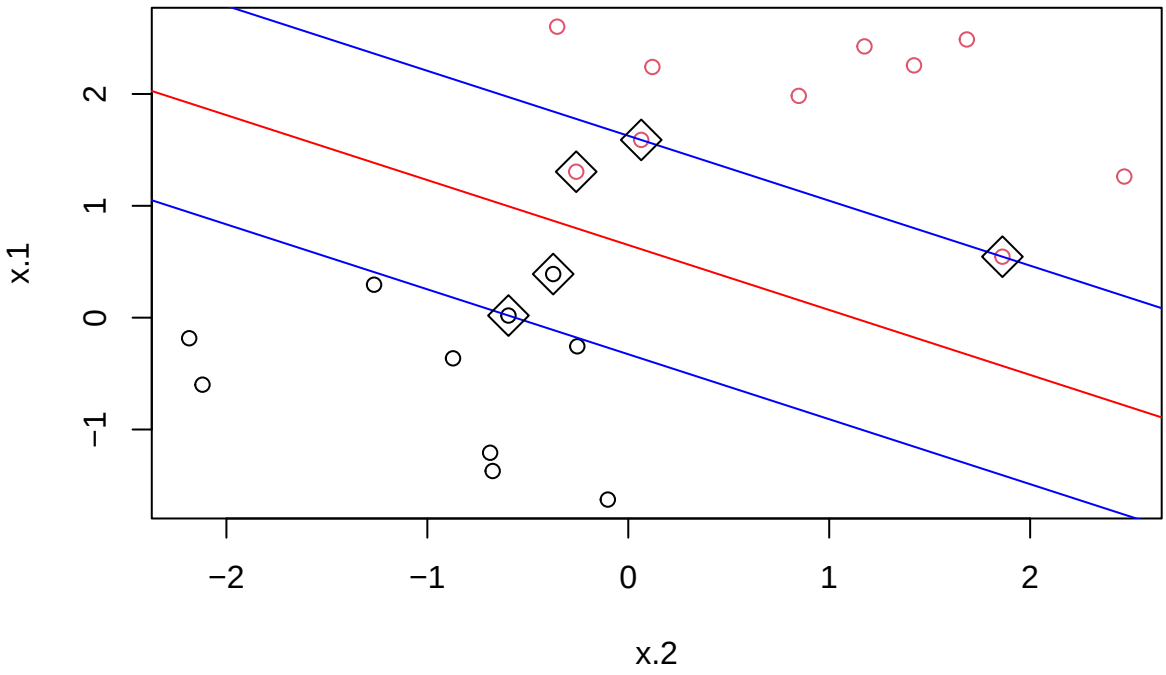
```
# plot
plot(svmfit, dat)
```




```
svmfit$SV
```

```
##           x.1         x.2
## 1  0.01874617 -0.59631064
## 6  0.38979430 -0.37366156
## 16 1.58934727  0.06448564
## 17 0.54505614  1.86208723
## 18 1.30484962 -0.25908675
```

```
# You can also draw this manually:
# plot data and separating hyperplane
plot(x.1 ~ x.2, data = dat, col=dat$y)
cf <- coef(svmfit)
abline(-cf[1]/cf[2], -cf[3]/cf[2], col = "red")

# margin and support vectors

abline(-(cf[1] + 1)/cf[2], -cf[3]/cf[2], col = "blue")
abline(-(cf[1] - 1)/cf[2], -cf[3]/cf[2], col = "blue")
points(svmfit$SV[,2], svmfit$SV[,1], pch = 5, cex = 2)
```

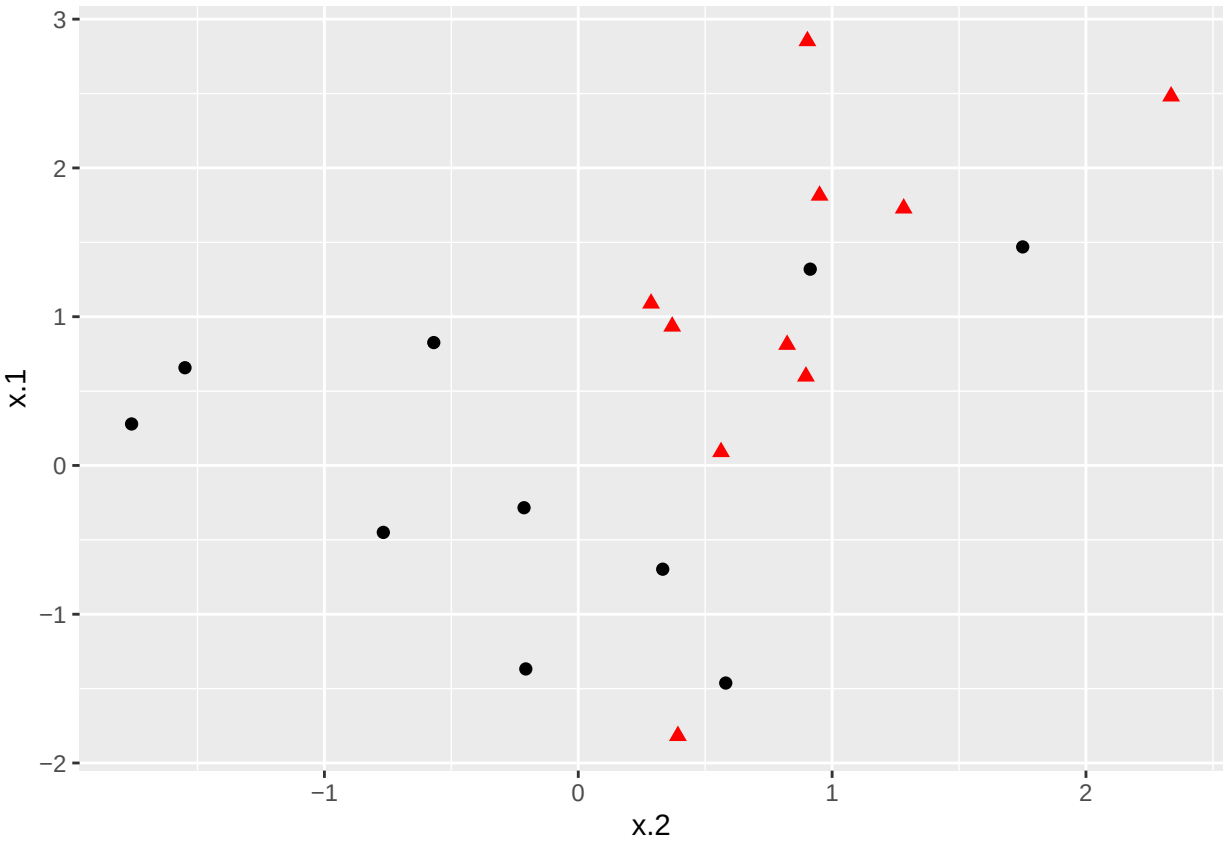


Question 2: Support Vector Classifiers

Most real data sets will not be fully separable by a linear boundary. To handle such data, we must use modified methodology. We simulate a new data set where the classes are more mixed.

Run the codes below to generate and visualize such data.

```
set.seed(890624)
# 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")
```



Whether the data is separable or not, the `svm` command syntax is the same. In the case of data that is not linearly separable, however, the `cost` argument takes on real importance. This quantifies the penalty associated with having an observation on the wrong side of the classification boundary. In other words, the parameter defines how much misclassified observations contribute to the overall error. Moreover, the `kernel` argument allows us to select a function based on the assumed shape of our data. For now we assume our data are linearly separable.

Use a `linear` kernel to fit an SVM to the model data set, with a cost of 10.

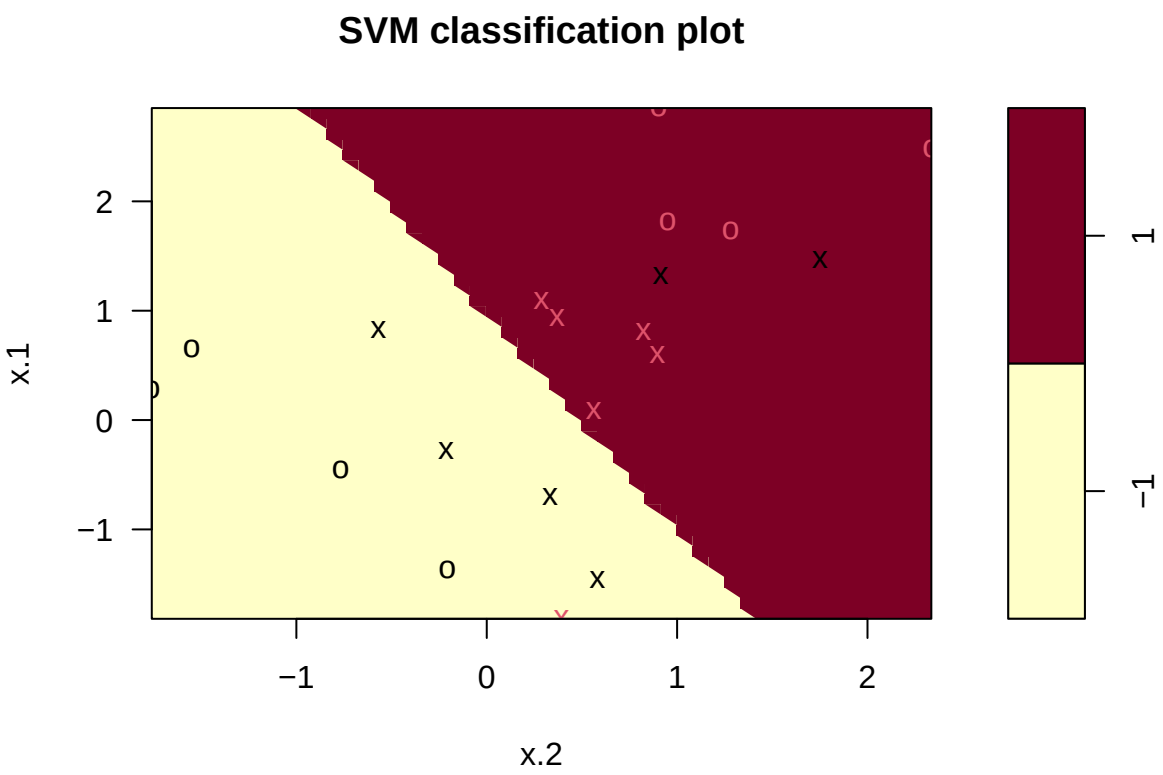
```
svmfit2 <- svm(...) # TODO: specify the arguments
```

Then, instead of specifying a cost up front, use the `tune` function from `e1071` to test various costs and identify which value produces the best fitting model. Use the `ranges` argument, which takes a list of vectors, to define the range of costs we are interested in. Utilize a cost of `list(c(0.001, 0.01, 0.1, 1, 5, 10, 100))` after setting the seed to 890624. What's the optimal cost? Create a contingency table to discern how well your predictions are under the SVM.

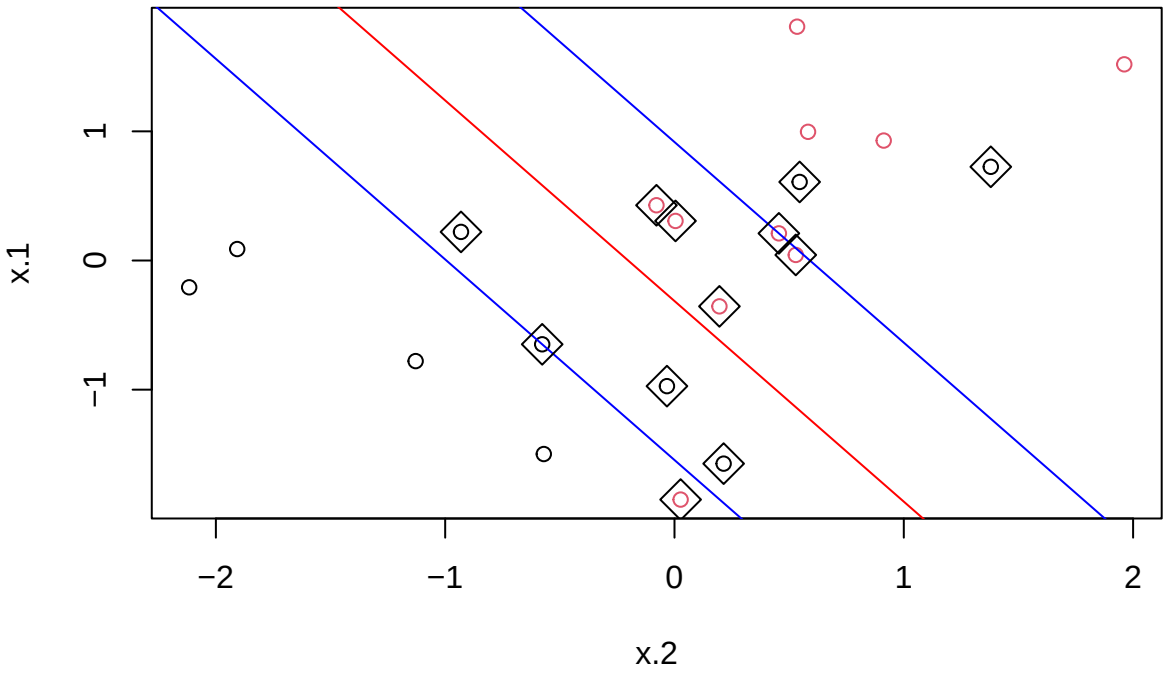
```
set.seed(890624)
# Find optimal cost of misclassification
tune.out <- tune(..., # TODO 1: specify the arguments
                 ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100)))

# TODO 2: Extract the best model

# TODO 3: Create a contingency table for in-sample data
# (Hint: `fitted` item of your sum object is your in-sample prediction)
```

```
# do the same thing as abovebut with new data
```

```r
# Fit Support Vector Machine model to data set
svmfit2 <- svm(y~., data = dat, kernel = "linear", cost = 10)
# Plot Results
plot(svmfit2, dat)
```




```r
# You can also draw this manually:
# plot data and separating hyperplane
scaled.dat <- scale(dat[,1:2]) # Note that we've scaled the predictors for svmfit2
plot(x.1 ~ x.2, data = scaled.dat, col=dat$y)
cf <- coef(svmfit2)
abline(-cf[1]/cf[2], -cf[3]/cf[2], col = "red")

# plot margin and mark support vectors
abline(-(cf[1] + 1)/cf[2], -cf[3]/cf[2], col = "blue")
abline(-(cf[1] - 1)/cf[2], -cf[3]/cf[2], col = "blue")
points(svmfit2$SV[,2], svmfit2$SV[,1], pch = 5, cex = 2)
```

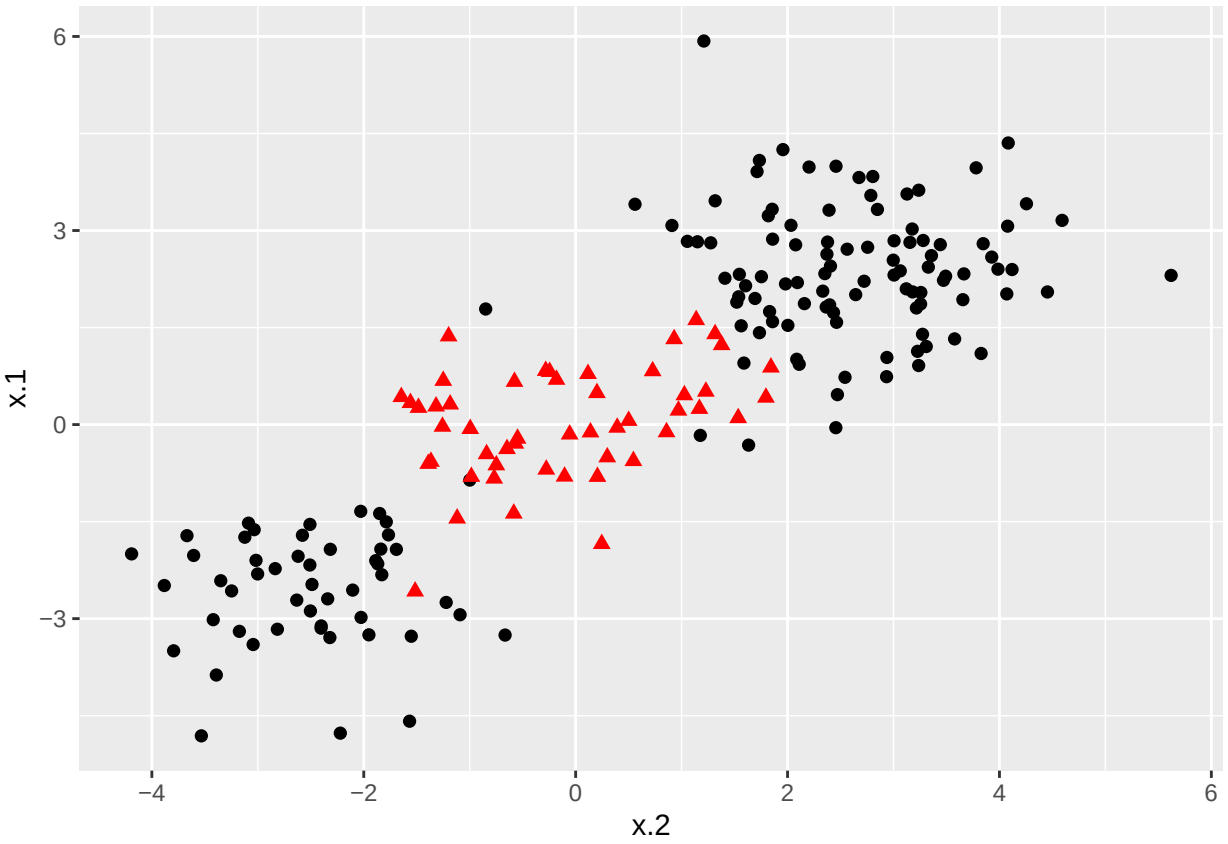


Question 3 Support Vector Machines (Nonlinear)

Support Vector Classifiers are a subset of the group of classification structures known as Support Vector Machines. Support Vector Machines can construct classification boundaries that are nonlinear in shape. The options for classification structures using the `svm` command from the `e1071` package are linear, polynomial, radial, and sigmoid. To demonstrate a nonlinear classification boundary, we will construct a new data set.

Run the codes below to generate and visualize such data.

```
set.seed(890624)
# Construct larger random data set
x <- matrix(rnorm(200*2), 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))

# 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")
```



Notice that the data is not linearly separable, and furthermore, isn't all clustered together in a single group. There are two sections of class 1 observations with a cluster of class 2 observations in between.

To demonstrate the power of SVMs, take 100 random observations from the set and use this *train* data to construct your boundary. Set `kernel = "radial"` based on the shape of our data and plot the results.

```
set.seed(890624)
# TODO 1: Randomly split the data
# TODO 2: Fit svm (Use kernel = "linear", gamma = 1, cost = 10)
```

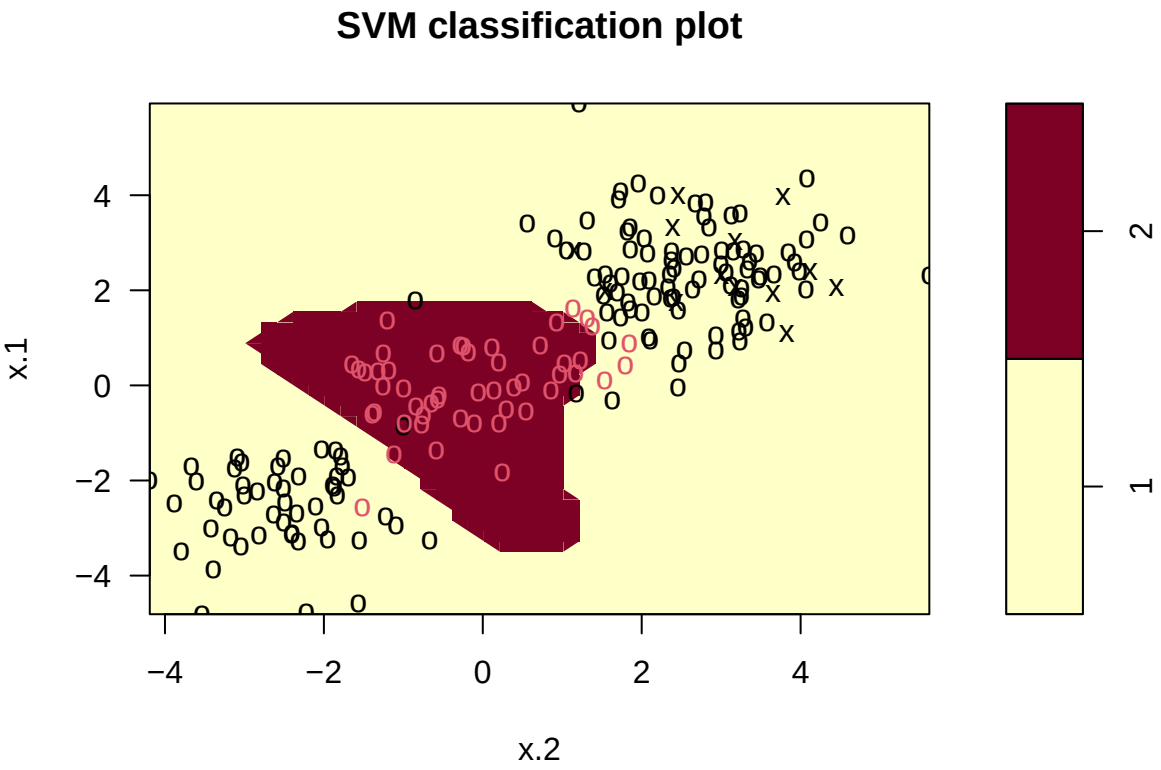
Again, use the `tune` function from `e1071` and your train data to test various costs and gamma, a scaling parameter used to fit nonlinear boundaries. What's the optimal cost and gamma?

```
set.seed(890624)
# Find optimal cost/gamma
tune.out <- tune(..., # TODO 1: specify the arguments
                 ranges = list(cost = c(0.1,1,10,20,50,100),
                 gamma = c(0.5,1,2,3,4)))

# TODO 2: Extract and print the best model
```

Now, we will validate this best model using the remaining *test* data. Predict the label of the test data using the best model and create a contingency table to check the results.

```
# TODO 3: Prediction based on the best model
ypred <- predict(your-best-svm-object, newx = your-test-data)

# TODO 4: Create a contingency table for out-of-sample
```

```r
# Set pseudorandom number generator
set.seed(890624)

# Sample training data and fit model
train <- base::sample(200,100, replace = FALSE)
svmfit3 <- svm(y~., data = dat[train,], kernel = "radial", gamma = 1, cost = 10)

# Plot classifier
plot(svmfit3, dat)
```




```r
set.seed(890624)
# Tune model to find optimal cost, gamma values
tune.out <- tune(svm, y~., data = dat[train,], kernel = "radial",
                 ranges = list(cost = c(0.1,1,10,20,50,100), # A list of parameter vectors for cost
                 gamma = c(0.5,1,2,3,4)))

# Show best model
tune.out$best.model
```

```
##
## Call:
## best.tune(method = svm, train.x = y ~ ., data = dat[train, ], ranges = list(cost = c(0.1,
##     1, 10, 20, 50, 100), gamma = c(0.5, 1, 2, 3, 4)), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
```

```
## Number of Support Vectors:  24
```

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

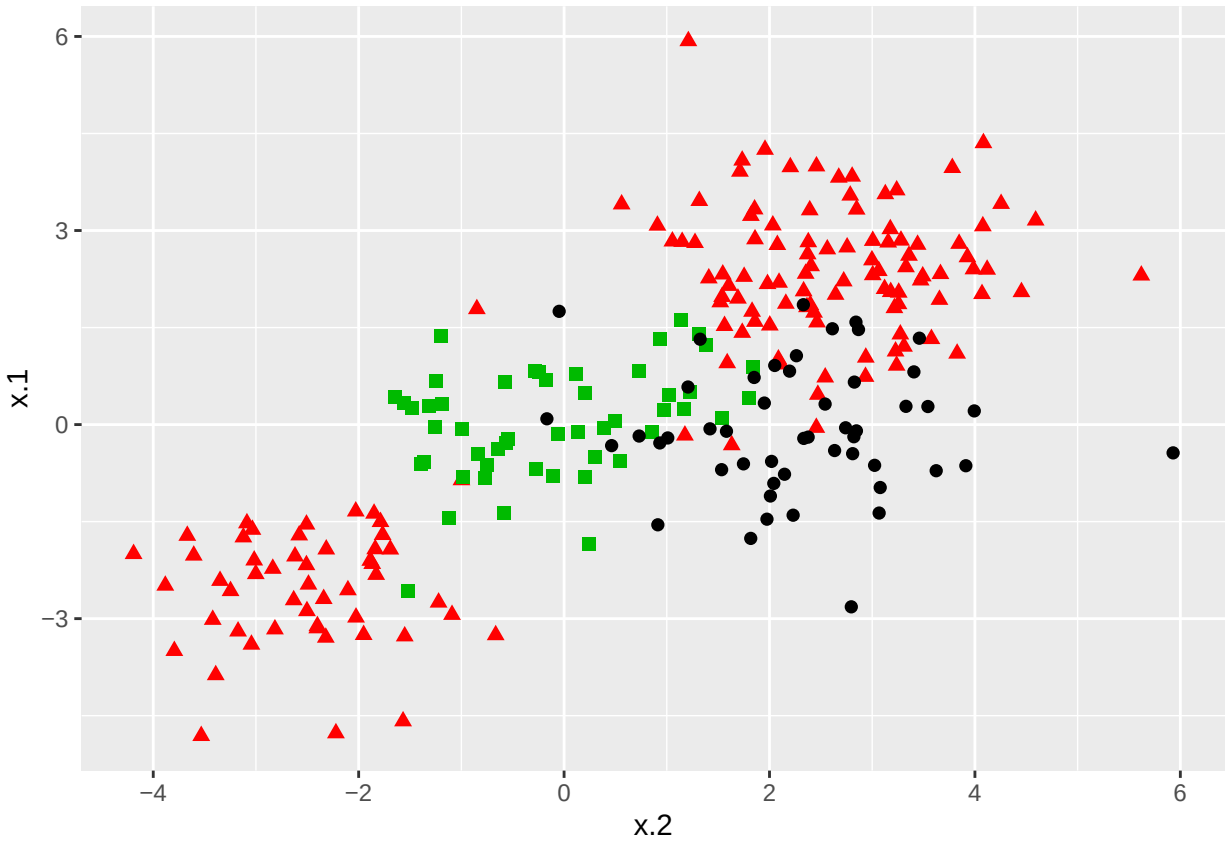
```
##     pred
## true  1  2
##    1 52 19
##    2 25  4
```

Question 4 SVMs for Multiple Classes

The procedure does not change for data sets that involve more than two classes of observations. We construct our data set the same way as we have previously, only now specifying three classes instead of two:

Run the codes below to generate and visualize such data.

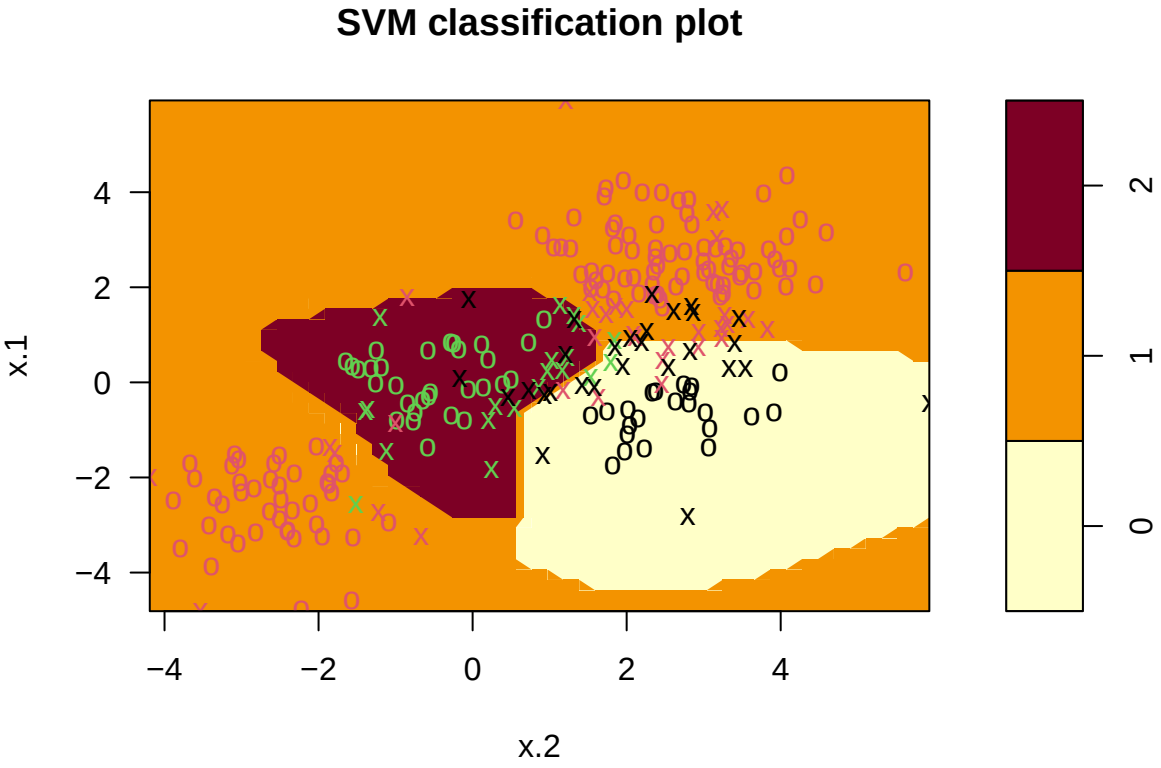
```
set.seed(890624)
# Construct data set
x <- rbind(x, matrix(rnorm(50*2), ncol = 2))
y <- c(rep(1,150), rep(2,50), rep(0,50))
x[y==0,2] <- x[y==0,2] + 2.5
dat <- data.frame(x=x, y=as.factor(y))

# 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","#00BA00")) +
  theme(legend.position = "none")
```



Fit a support vector machine with following specification: `kernel = "radial", cost = 10, gamma = 1`. See how well our model fit the data by using the `predict` command and creating a contingency table.

```
# Fit model
svmfit <- svm(...) # TODO 1: specify the arguments

# Plot results
plot(svmfit, dat)

# TODO 2: Prediction based on the fitted model

# TODO 3: Create a contingency table for in-sample data
```

```
# Fit model
svmfit <- svm(y~., data = dat, kernel = "radial", cost = 10, gamma = 1)

# Plot results
plot(svmfit, dat)
```




```
# Construct table
# ypred <- predict(svmfit, dat)
table(predict = svmfit$fitted, truth = dat$y)
```

```
##         truth
## predict   0   1   2
##       0  33   4   2
##       1   9 143   2
##       2   8   3  46
```

Question 5 Application 1

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.

```
dat_khan.train <- data.frame(x = Khan$xtrain, y=as.factor(Khan$ytrain))
dat_khan.test <- data.frame(x=Khan$xtest, y=as.factor(Khan$ytest))
```

We choose the simplest classifier (linear) to construct our model. Use `tune` and `ranges = list(cost = c(1, 5, 10, 20, 30))` to identify the best fitting model. Validate this best model with the test set and create a contingency table.

```
# TODO 1: Tune model to find optimal cost values

# TODO 2: Extract the best model

# TODO 3: Prediction based on the best model using test data
ypred <- predict(your-best-svm-object, newx = dat_khan.test)
```

```
# TODO 4: Create a contingency table
```

```
# Tune model to find optimal cost values
tune_out_khan <- tune(svm, y~., data = dat_khan.train, kernel = "linear",
                  ranges = list(cost = c(1, 5, 10, 20, 30))) # A list of parameter vectors for cost


# Show best model
out_khan_best <-  tune_out_khan$best.model
print(out_khan_best)
```

```
##
## Call:
## best.tune(method = svm, train.x = y ~ ., data = dat_khan.train, ranges = list(cost = c(1,
##     5, 10, 20, 30)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  58
```

```
# Check model performance on training set
table(out_khan_best$fitted, dat_khan.train$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 on test set
pred.test <- predict(out_khan_best, newdata=dat_khan.test)

table(pred.test, dat_khan.test$y)
```

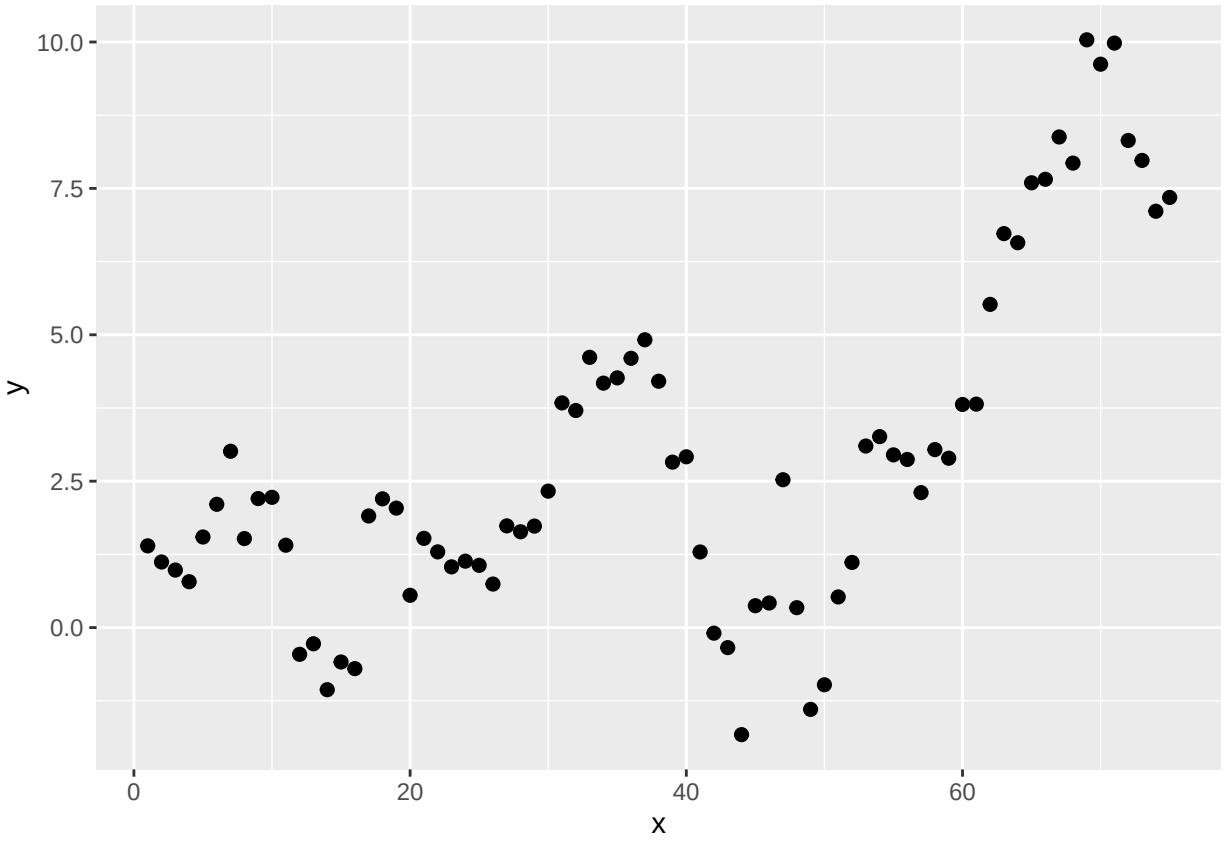
```
##
## pred.test 1 2 3 4
##         1 3 0 0 0
##         2 0 6 2 0
##         3 0 0 4 0
##         4 0 0 0 5
```

[Optional] Application 2

In this example, we will use a continuous y variable. Generate a random data:

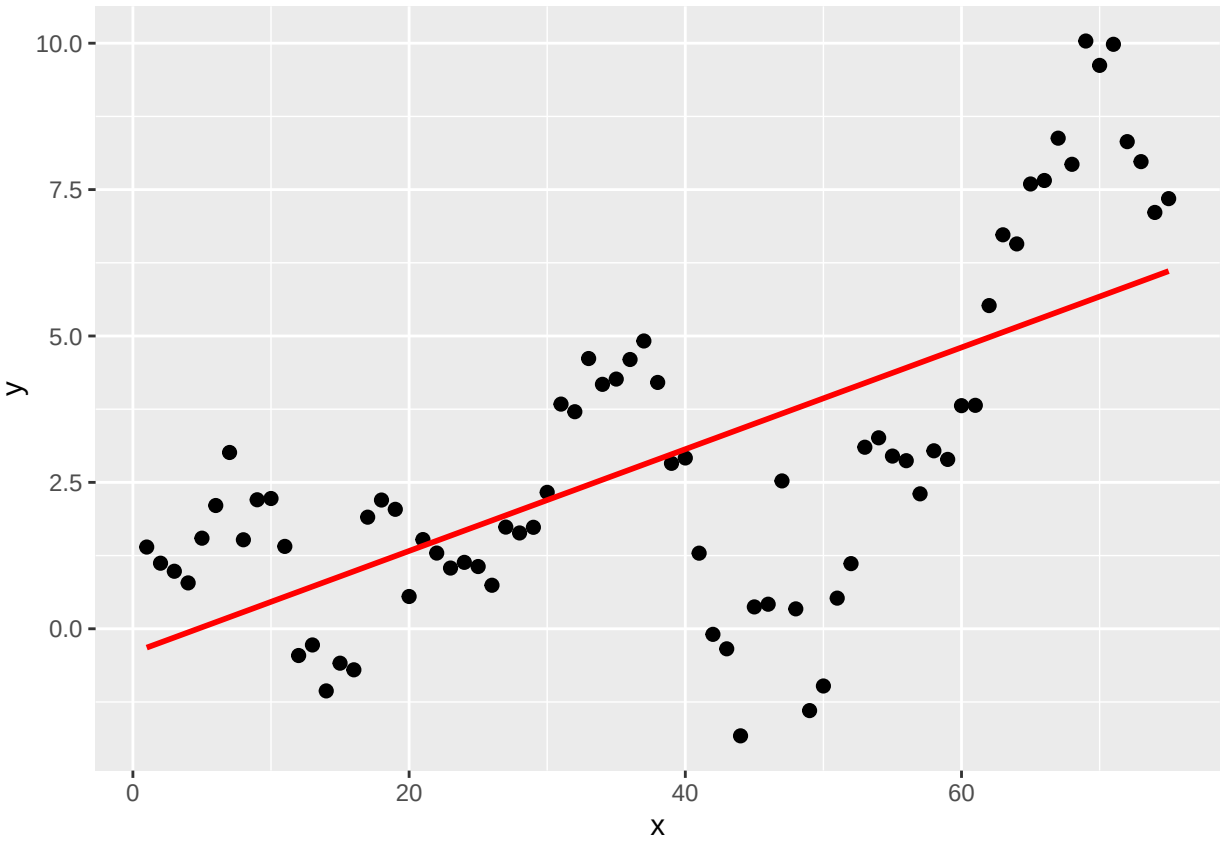
```
# Generate random data
set.seed(910328)
x = 1:75
y = cumsum((rnorm(length(x))))
```

```r
dat <- data.frame(x=x, y=y)

# Plot data
ggplot(data = dat, aes(x = x, y = y)) +
  geom_point(size = 2) +
  scale_color_manual(values=c("#000000", "#FF0000")) +
  theme(legend.position = "none")
```



We will first use linear regression to compare its performance with that of SVM regression.

```r
# Fit a linear regression model
fit_linear <- lm(y ~ x, data = dat)

# Make predictions for regression model for each x val
predict_linear <- predict(fit_linear, dat)

# Show predictions with orignal data
ggplot(data = dat, aes(x = x, y = y)) +
  geom_point(size = 2) +
  scale_color_manual(values=c("#000000", "#FF0000")) +
  theme(legend.position = "none") +
  geom_smooth(method = "lm", se = FALSE, color = "red", formula = y ~ x)
```



Now we use support vector regression.

```
# TODO 1: Fit SVM model

# TODO 2: Plot the results along with the linear regression line
```

Now we will tune the cost parameter to identify the best model.

```
# TODO 1: Perform a grid search (this might take a few seconds, adjust how fine of grid if taking too l
tuneResult <- tune(...,
              ranges = list(epsilon = seq(0,1,0.1), cost = 2^(seq(0.5,8,.5)))
)

# Map tuning results
plot(tuneResult)
```

Let's plot the three models together for comparison

Alternatively, we can also compare the root mean squared errors (RMSE) of different models.

```
rmse <- function(errval)
{
  val = sqrt(mean(errval^2))
  return(val)
}

# TODO: Compare the models
```