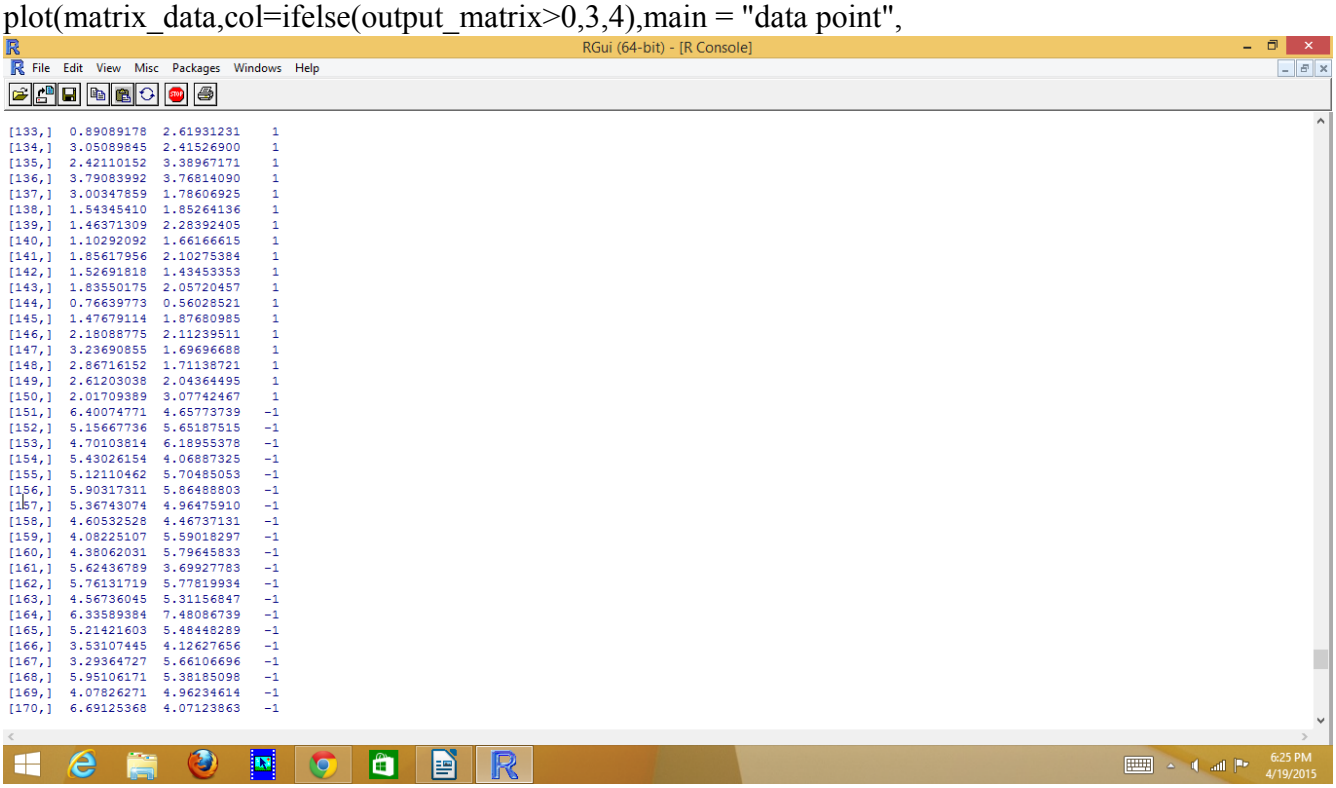
Data And Code used for Analysis

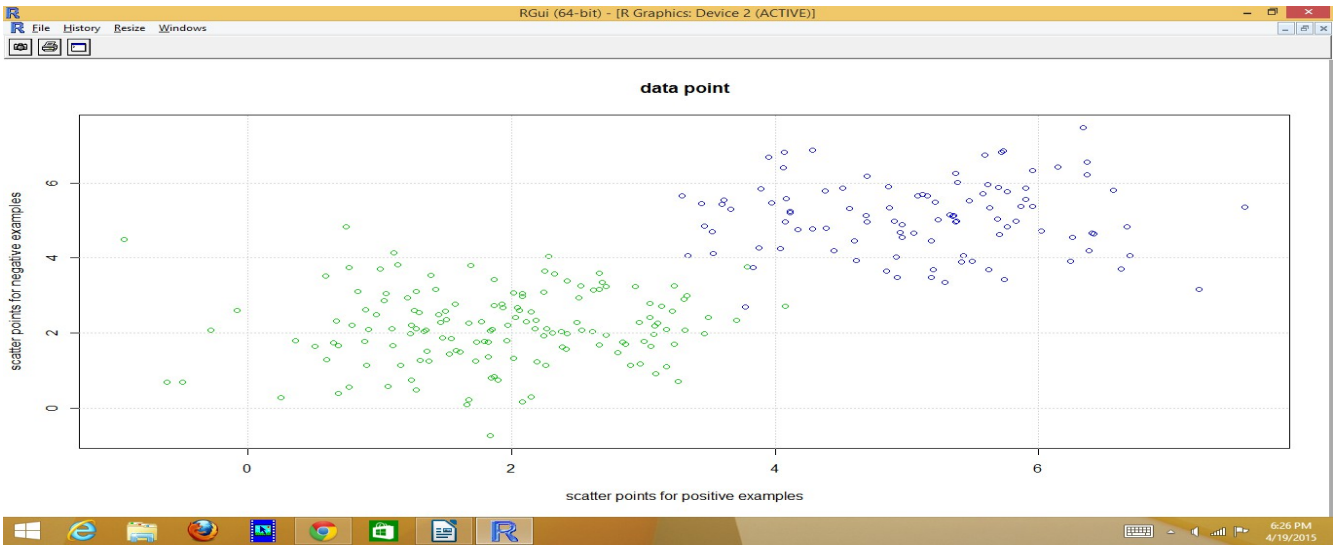
```
num <- 250 # number of data points
dimension <- 2 # dimension
num_positive = 150
num_n = 100

val1 = num_positive * dimension
val2 = num_n * dimension

matrix_p <- matrix(rnorm(val1,mean= 2,sd= 1),num_positive,dimension)
matrix_n <- matrix(rnorm(val2,mean= 5,sd=1),num_n, dimension)
matrix_data = rbind(matrix_p,matrix_n)

print(matrix_data)

one_vector= c(rep(1,num_positive))
diffone_vector = c(rep(-1, num_n))

output_matrix = matrix(c(one_vector, diffone_vector))
total_data = cbind(matrix_data, output_matrix)

print(total_data)


plot(matrix_data,col=ifelse(output_matrix>0,3,4),main = "data point",
```



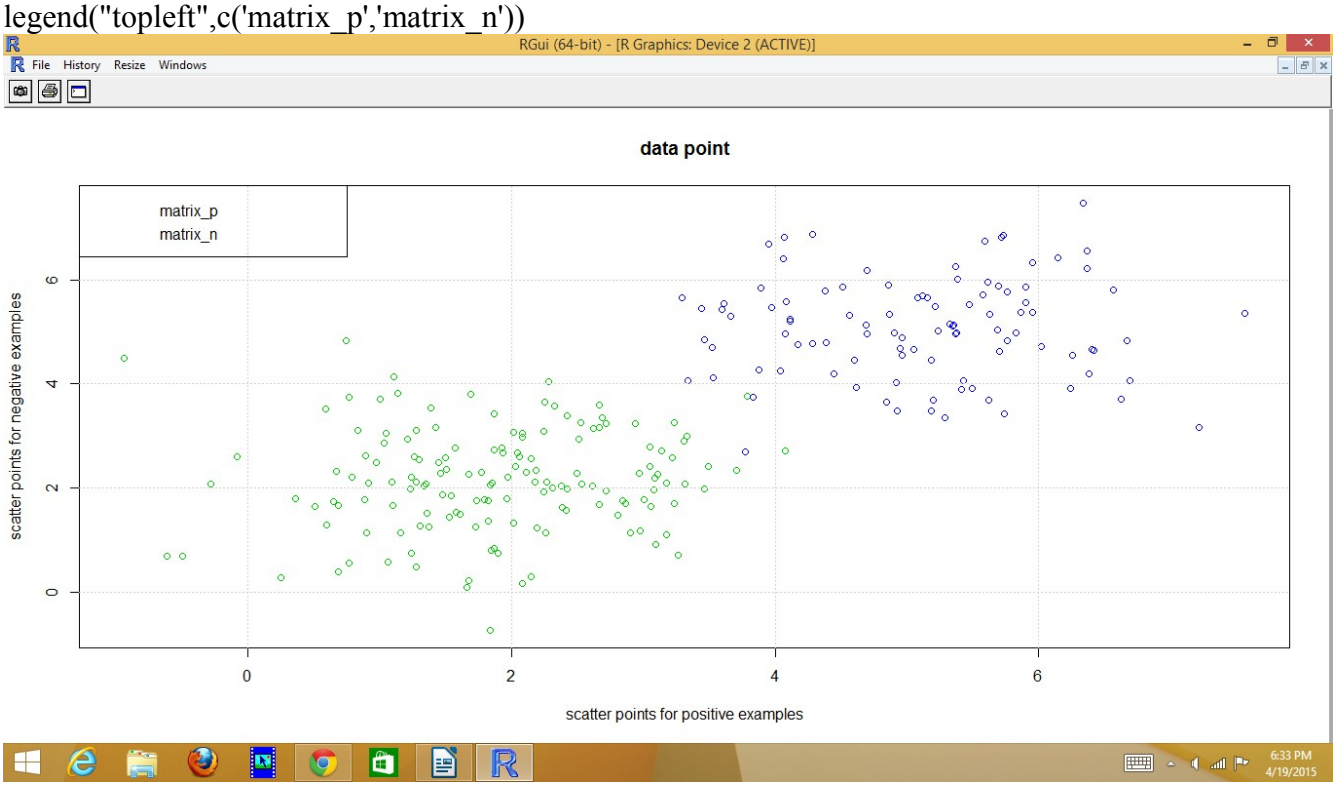

```
xlab = "scatter points for positive examples",
ylab = "scatter points for negative examples" )
```

grid()

we know analyze the scatter points and see how our positive and negative examples looks like

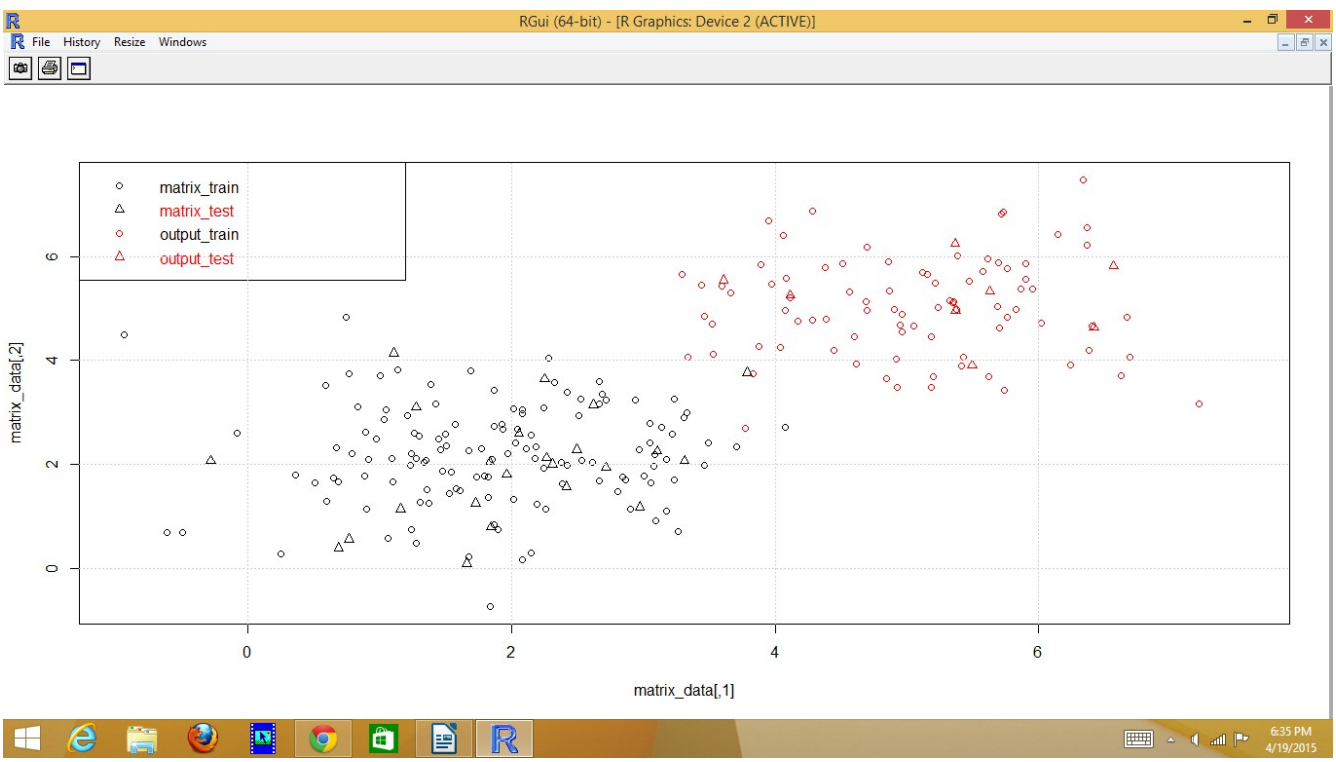


legend("topleft",c('matrix_p','matrix_n'))



We then perform cross validation pver the data to classify the test data set

```
#cross validation pick 85 percent initially
train_cases = floor(num * 0.85)
test_cases = num - train_cases

#randomly choose the number of training cases, we need to take 85 %of data
#for training, thus we choose train_cases to select that

index_train = sample(num, train_cases)

# pick all the index_train and store it in matrix_train
matrix_train <- matrix_data[index_train,]
output_train <- output_matrix[index_train,]

# rest will be in the test data
matrix_test <- matrix_data[-index_train,]
output_test <- output_matrix[-index_train,]
plot_train=rep(0,n)
plot_train[index_train]=1


plot(matrix_data,col=ifelse(output_matrix >0,1,2),pch=ifelse(plot_train==1,1,2))
legend("topleft",c('matrix_train','matrix_test','output_train','output_test'),
col=c(1,1,2,2),pch=c(1,2,1,2),text.col=c(1,2,1,2))
grid()
```

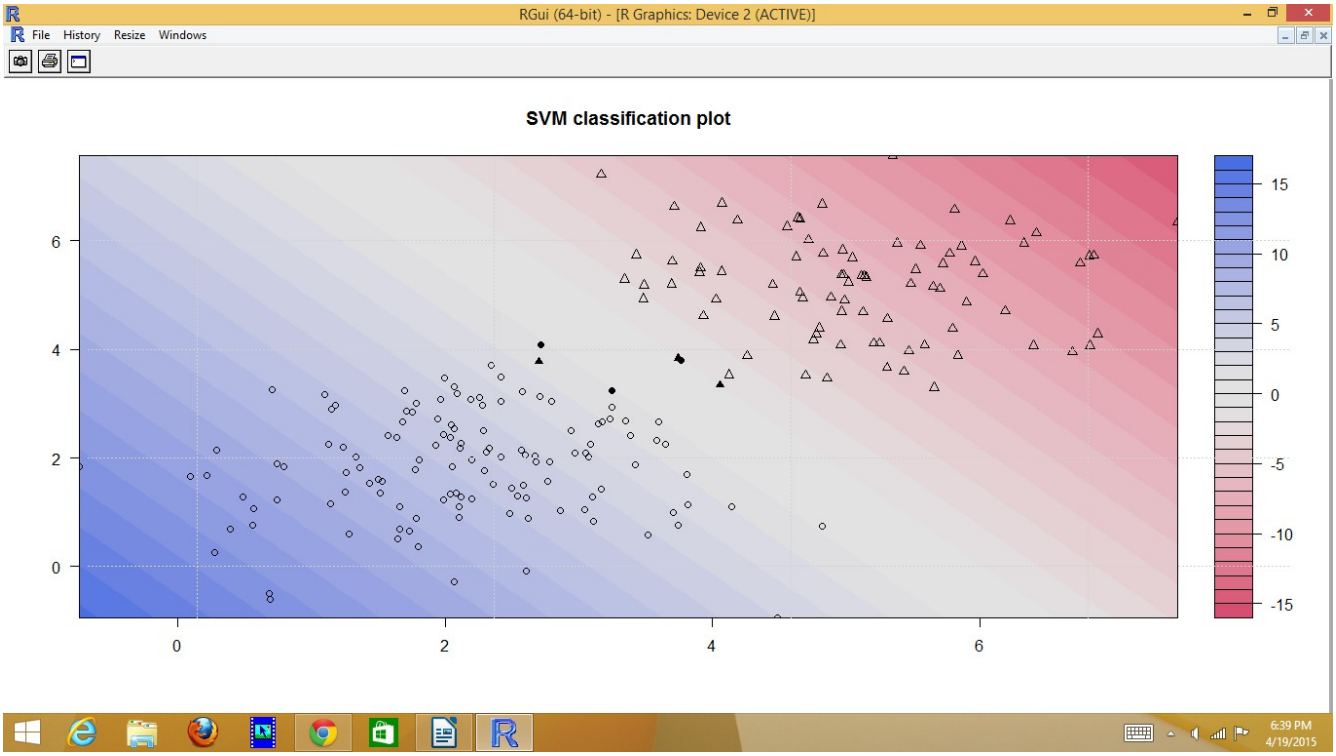


We now have scattered our input and output training data and our input and output test data set.

We now have all the data necessary to perform training to our svm

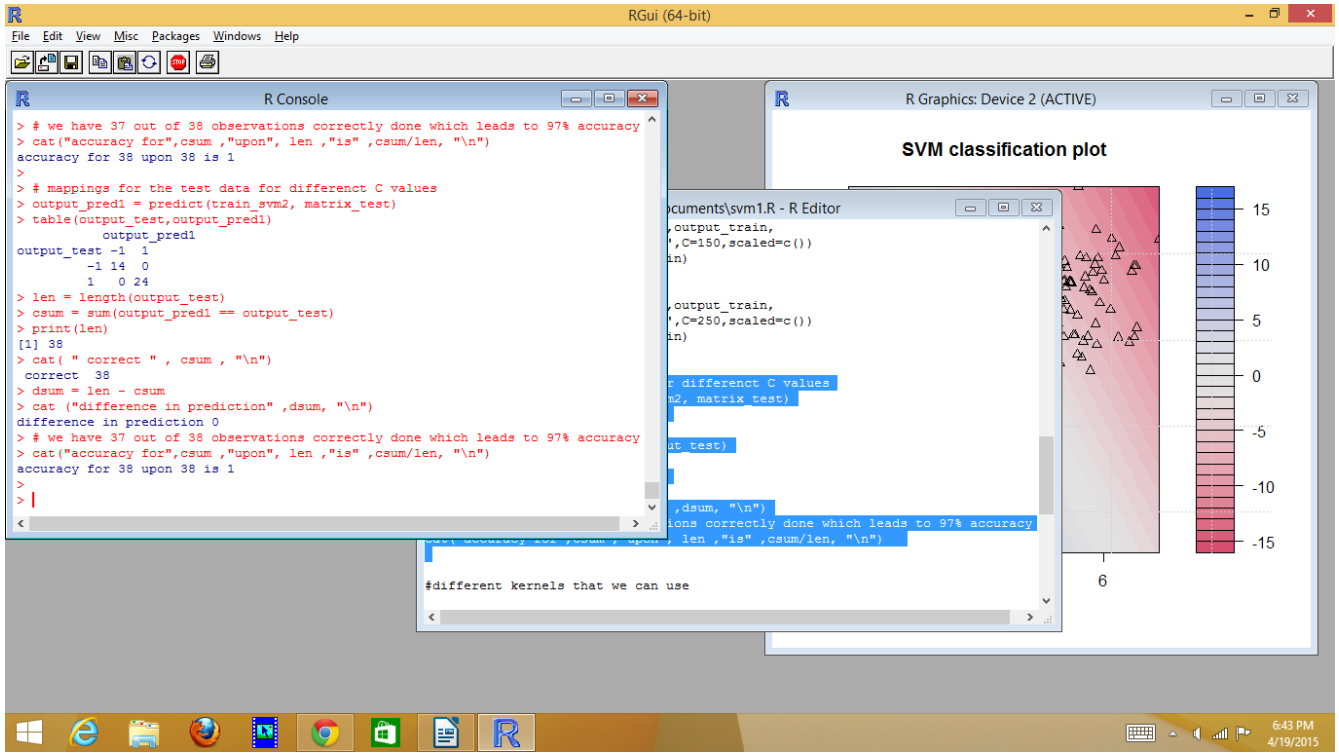
```
library(kernlab)
train_svm1 <- ksvm(matrix_train,output_train,
type="C-svc",kernel='vanilladot',C=50,scaled=c())
plot(train_svm1,data=matrix_train)
grid()
```

We should now analyze different versions of classification using different kernels and see the plot that corresponds to each of the kernel outputs.



Classification for vanilla dot kernel.

To know the accuracy and to best know the c vlaue, the below script will be useful for us



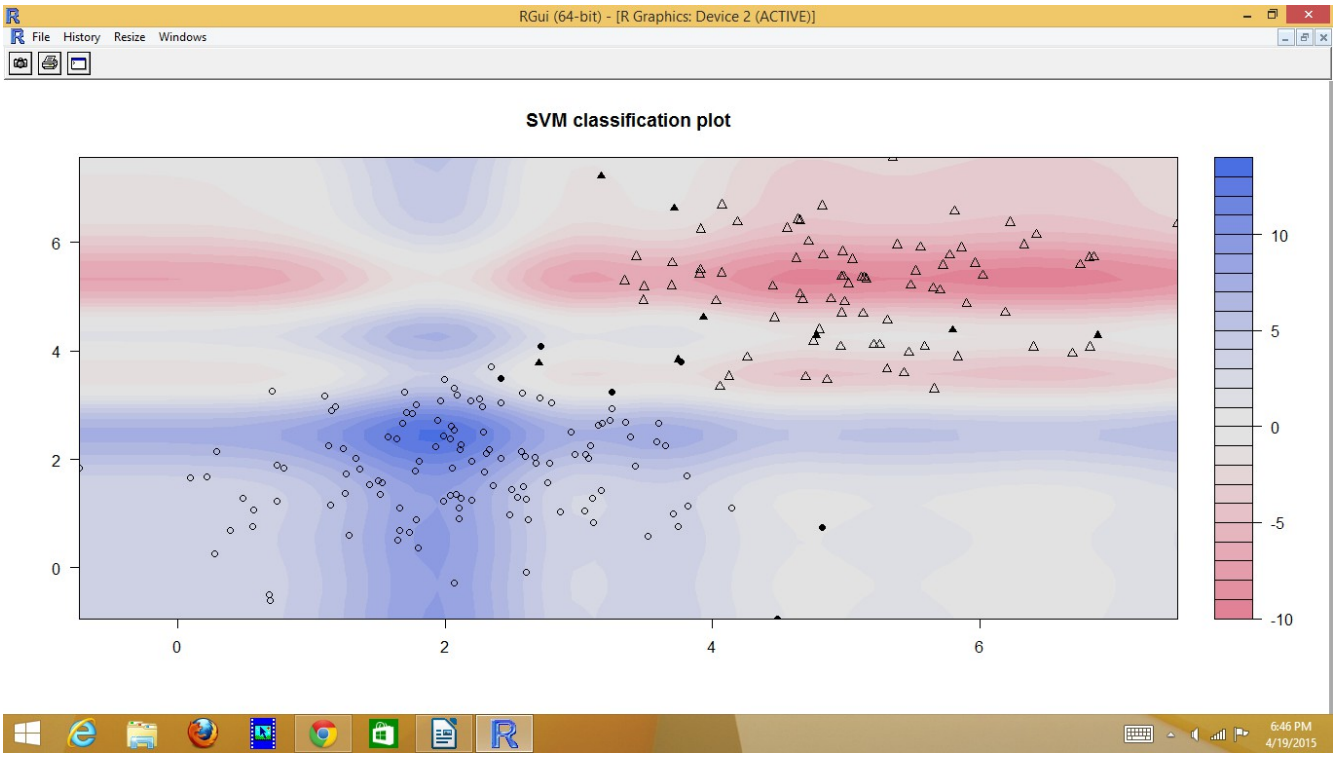
In the above screen shot, we can see that R console shows our accuracy, in our case it just happens to be 100

There are several different kernels that we can use for our analysis, we present the data and plots for all the kernels available.

```
#different kernels that we can use

train_svm1 <- ksvm(matrix_train,output_train,
type="C-svc",kernel='polydot',C=250,scaled=c())
plot(train_svm1,data=matrix_train)

train_svm1 <- ksvm(matrix_train,output_train,
type="C-svc",kernel='tanhdot',C=250,scaled=c())
plot(train_svm1,data=matrix_train)

train_svm1 <- ksvm(matrix_train,output_train,
type="C-svc",kernel='laplacedot',C=250,scaled=c())
plot(train_svm1,data=matrix_train)

train_svm1 <- ksvm(matrix_train,output_train,
type="C-svc",kernel='besseldot',C=250,scaled=c())
plot(train_svm1,data=matrix_train)

train_svm1 <- ksvm(matrix_train,output_train,
```

```
type="C-svc",kernel='anovadot',C=250,scaled=c())
plot(train_svm1,data=matrix_train)

train_svm1 <- ksvm(matrix_train,output_train,
type="C-svc",kernel='splinedot',C=250,scaled=c())
plot(train_svm1,data=matrix_train)
```

Lets analyse the output for anovadot kernel,

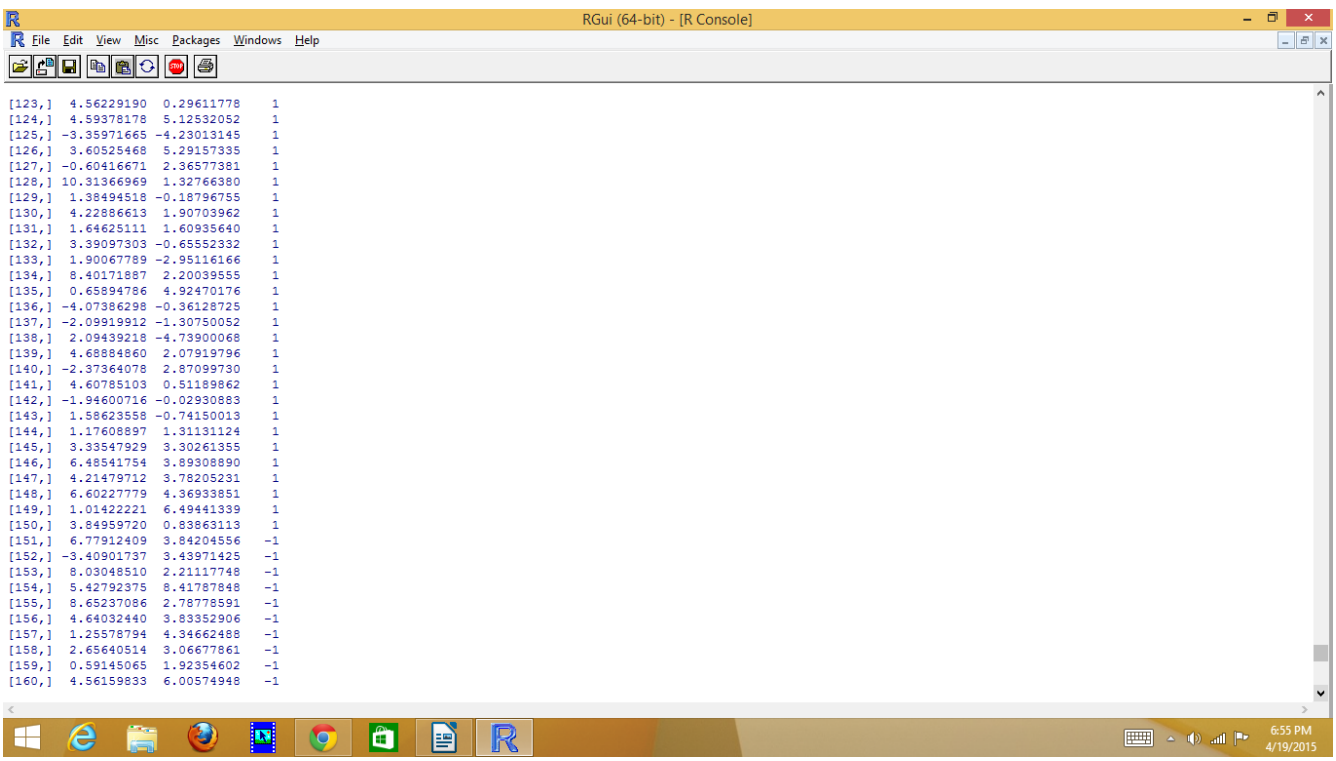


We see that we can clearly see the variance and the mean concentration of data. We can simply use the above used different kernels and analyze the output plot for each kernel.

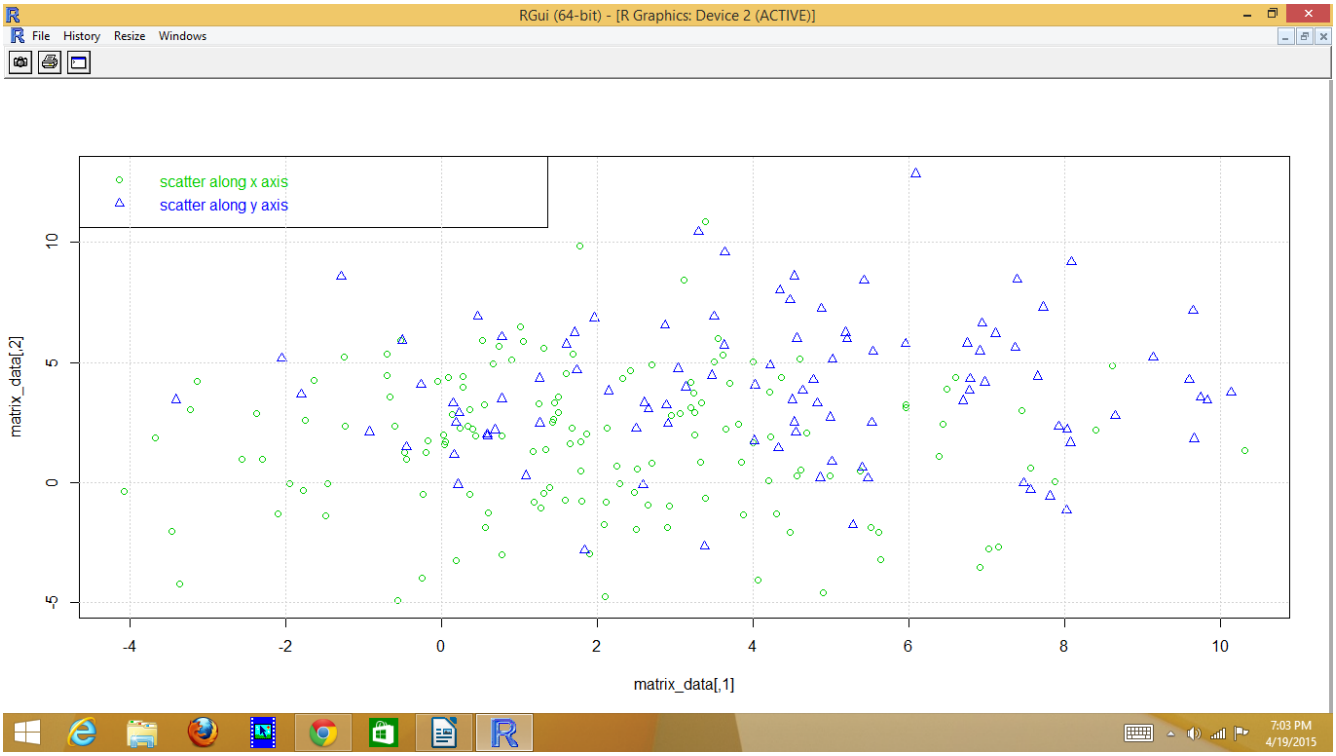
Not all the data can be binary, we may encounter many a times, where linear classification will not work, so we have to use proper svm training to do multi class classification, the below is the example for the same. We use the library kernlab for our analysis.

```
library(kernlab)

num <- 250 # number of data points
dimension <- 2 # dimension
num_positive = 150
num_n = 100

val1 = num_positive * dimension
```

```
val2 = num_n * dimension

matrix_p <- matrix(rnorm(val1,mean= 2,sd= 3),num_positive,dimension)
matrix_n <- matrix(rnorm(val2,mean= 4,sd=3),num_n, dimension)
matrix_data = rbind(matrix_p,matrix_n)

print(matrix_data)

one_vector= c(rep(1,num_positive))
diffone_vector = c(rep(-1, num_n))

output_matrix = matrix(c(one_vector, diffone_vector))
total_data = cbind(matrix_data, output_matrix)

print(total_data)
```

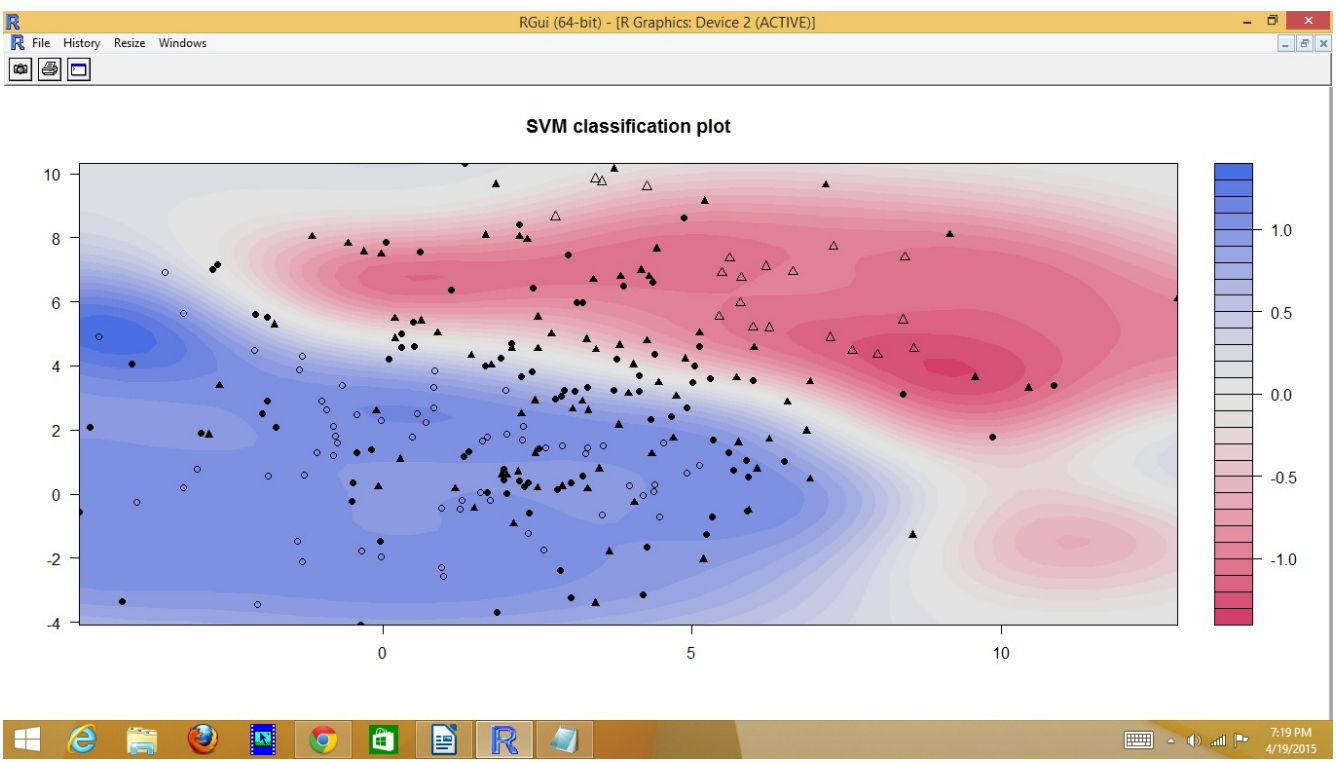



```
plot(matrix_data,col=ifelse(output_matrix>0,3,4),
pch=ifelse(output_matrix>0,1,2))
legend("topleft",c('scatter along x axis','scatter along y axis'),
col=c(3,4),pch=c(1,2),text.col=c(3,4))
grid()

# lets visualize the data and see why linear classification does not work here
```



We can clearly see that we cannot have a linear boundary that can classify the data for us

```
nonlinear_svm1 <- ksvm(matrix_data,output_matrix,type="C-svc",kernel='rbf')
plot(nonlinear_svm1,data=matrix_data, main = "rbf classification" )
```

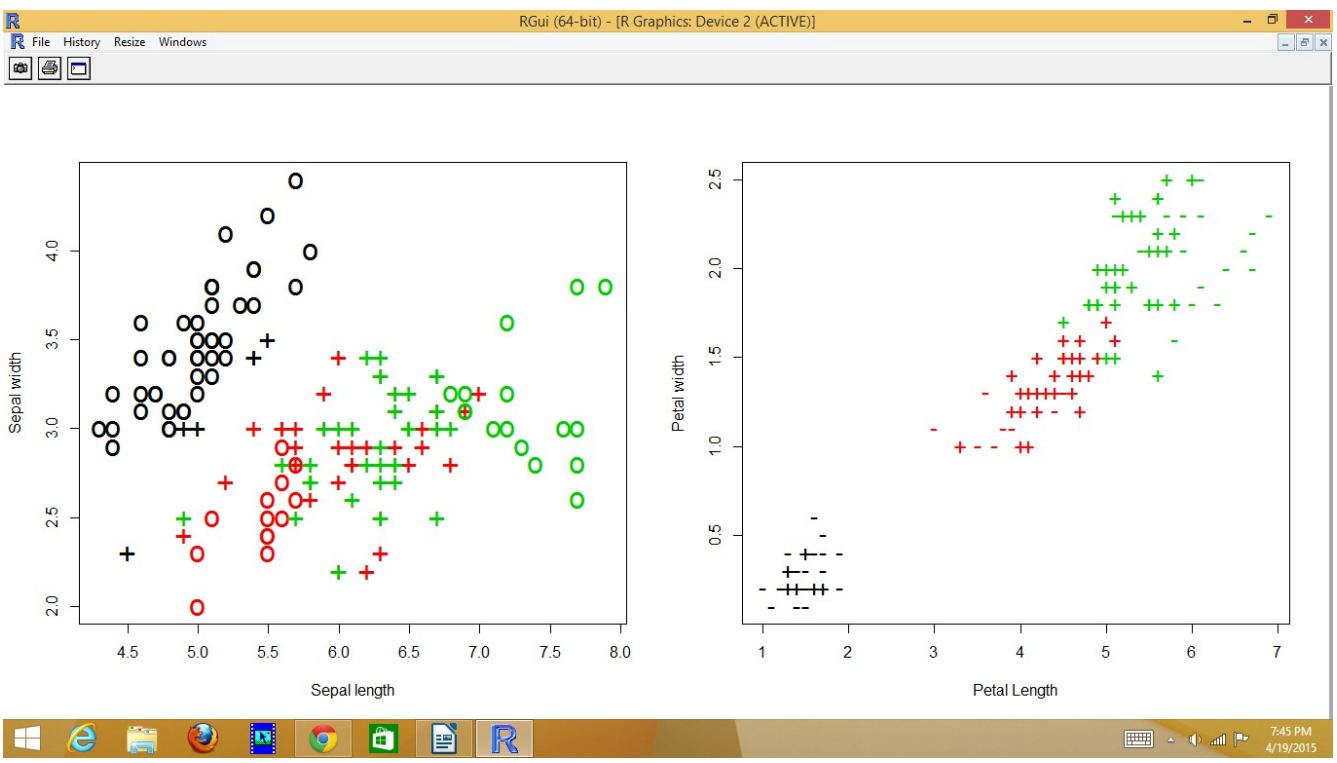


we get a non linear boundary, since the data was not uniform.

Now we use svm methods on some real data sets and then analyze the data and then we conclude by showing the kernels that svm uses for classification.

```
data(iris)
svm.model <- svm(Species ~ Sepal.Length + Sepal.Width, data = iris, kernel = "linear")
# the + are support vectors

   par(mfrow=c(1,2))
        plot(iris$Sepal.Length, iris$Sepal.Width, col = as.integer(iris[, 5]),
        pch = c("o","+")[1:150 %in% svm.model$index + 1], cex = 2,
        xlab = "Sepal length", ylab = "Sepal width")

plot(iris$Petal.Length, iris$Petal.Width,
col = as.integer(iris[, 5]),
pch = c("-","+")[1:150 %in% svm.model$index + 1], cex = 1.5,
xlab = "Petal Length", ylab = "Petal width")
```

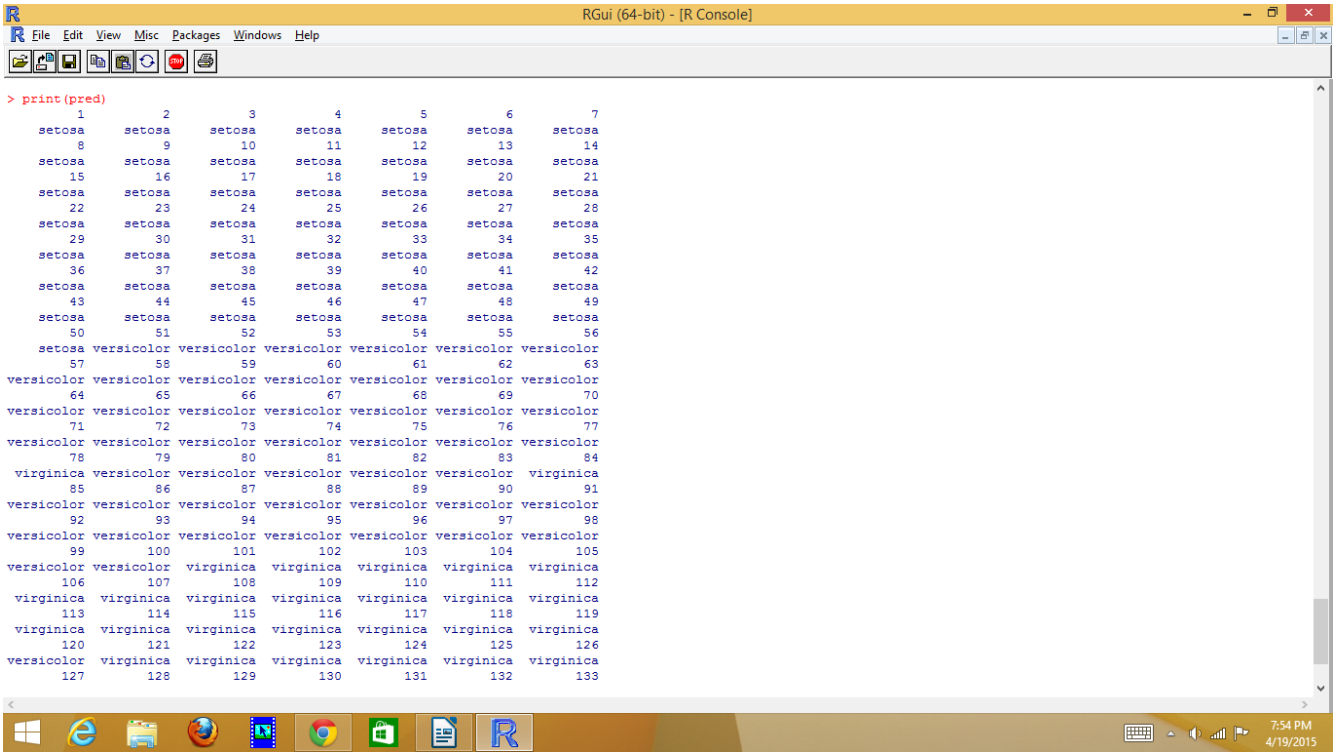
Lets visualize the distribution of data

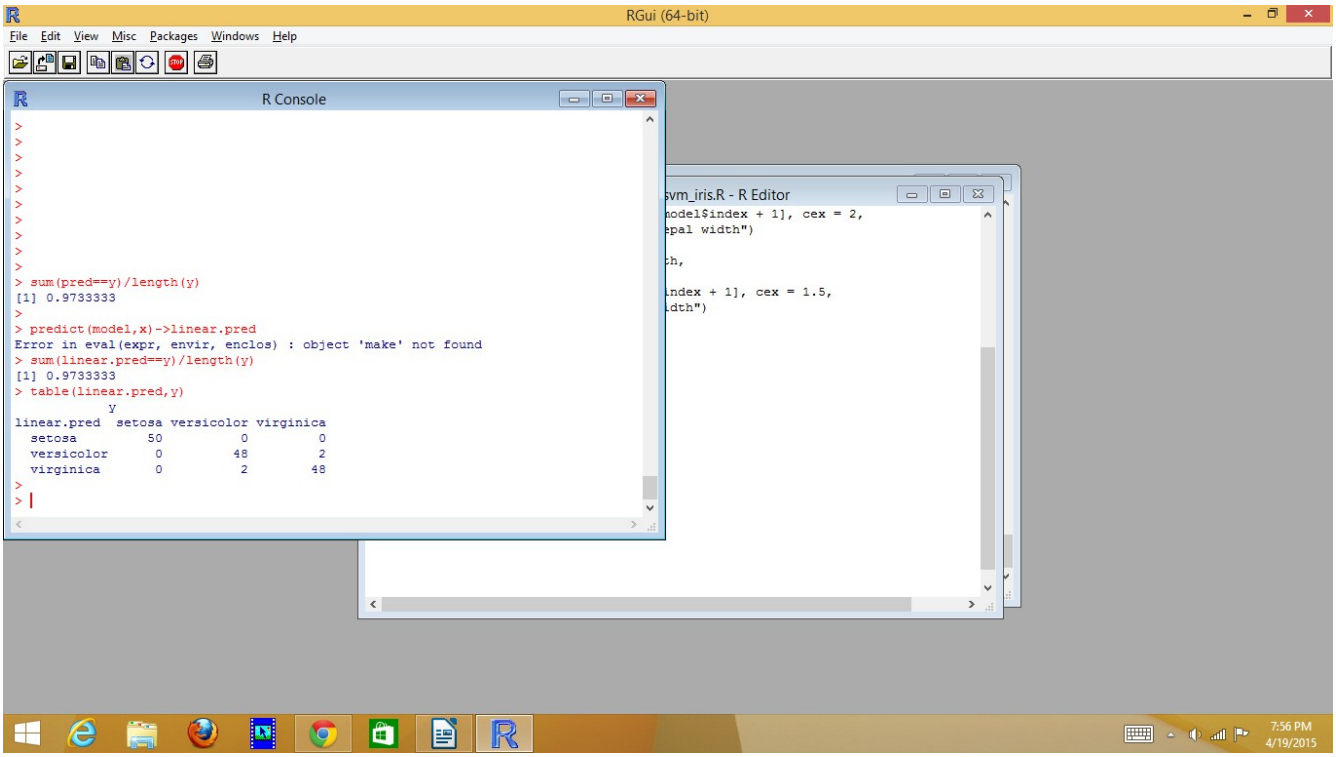


Now we have to see the prediction accuracy of svm, that can be done using the below script.

```
x<-iris[,-5]
y<-iris[,5]
predict(model,x)->pred
```

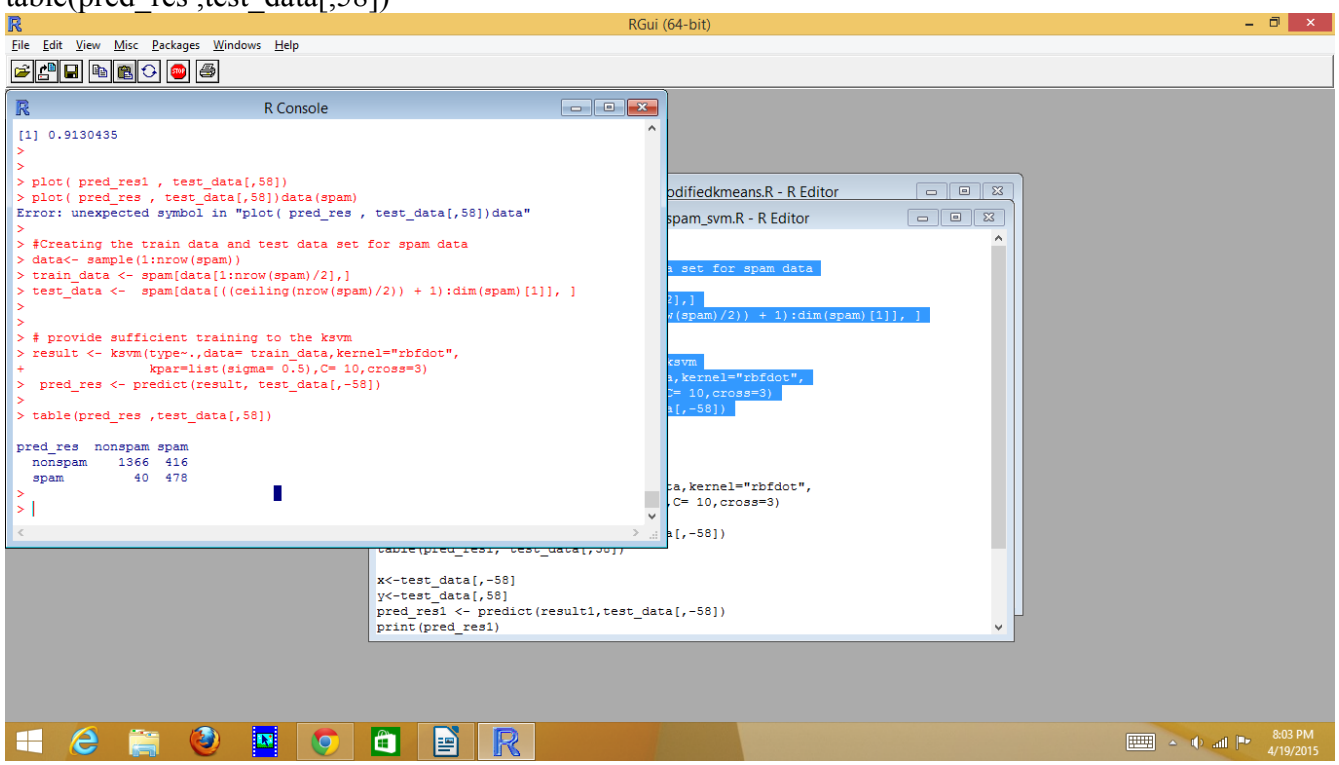
print(pred)


RGui (64-bit) - [R Console]
File Edit View Misc Packages Windows Help
> print(pred)
1 2 3 4 5 6 7
setosa setosa setosa setosa setosa setosa setosa
8 9 10 11 12 13 14
setosa setosa setosa setosa setosa setosa setosa
15 16 17 18 19 20 21
setosa setosa setosa setosa setosa setosa setosa
22 23 24 25 26 27 28
setosa setosa setosa setosa setosa setosa setosa
29 30 31 32 33 34 35
setosa setosa setosa setosa setosa setosa setosa
36 37 38 39 40 41 42
setosa setosa setosa setosa setosa setosa setosa
43 44 45 46 47 48 49
setosa setosa setosa setosa setosa setosa setosa
50 51 52 53 54 55 56
setosa versicolor versicolor versicolor versicolor versicolor versicolor
57 58 59 60 61 62 63
versicolor versicolor versicolor versicolor versicolor versicolor versicolor
64 65 66 67 68 69 70
versicolor versicolor versicolor versicolor versicolor versicolor versicolor
71 72 73 74 75 76 77
versicolor versicolor versicolor versicolor versicolor versicolor versicolor
78 79 80 81 82 83 84
virginica versicolor versicolor versicolor versicolor versicolor virginica
85 86 87 88 89 90 91
versicolor versicolor versicolor versicolor versicolor versicolor versicolor
92 93 94 95 96 97 98
versicolor versicolor versicolor versicolor versicolor versicolor versicolor
99 100 101 102 103 104 105
versicolor versicolor virginica virginica virginica virginica virginica
106 107 108 109 110 111 112
virginica virginica virginica virginica virginica virginica virginica
113 114 115 116 117 118 119
virginica virginica virginica virginica virginica virginica virginica
120 121 122 123 124 125 126
versicolor virginica virginica virginica virginica virginica virginica
127 128 129 130 131 132 133
7:54 PM
4/19/2015

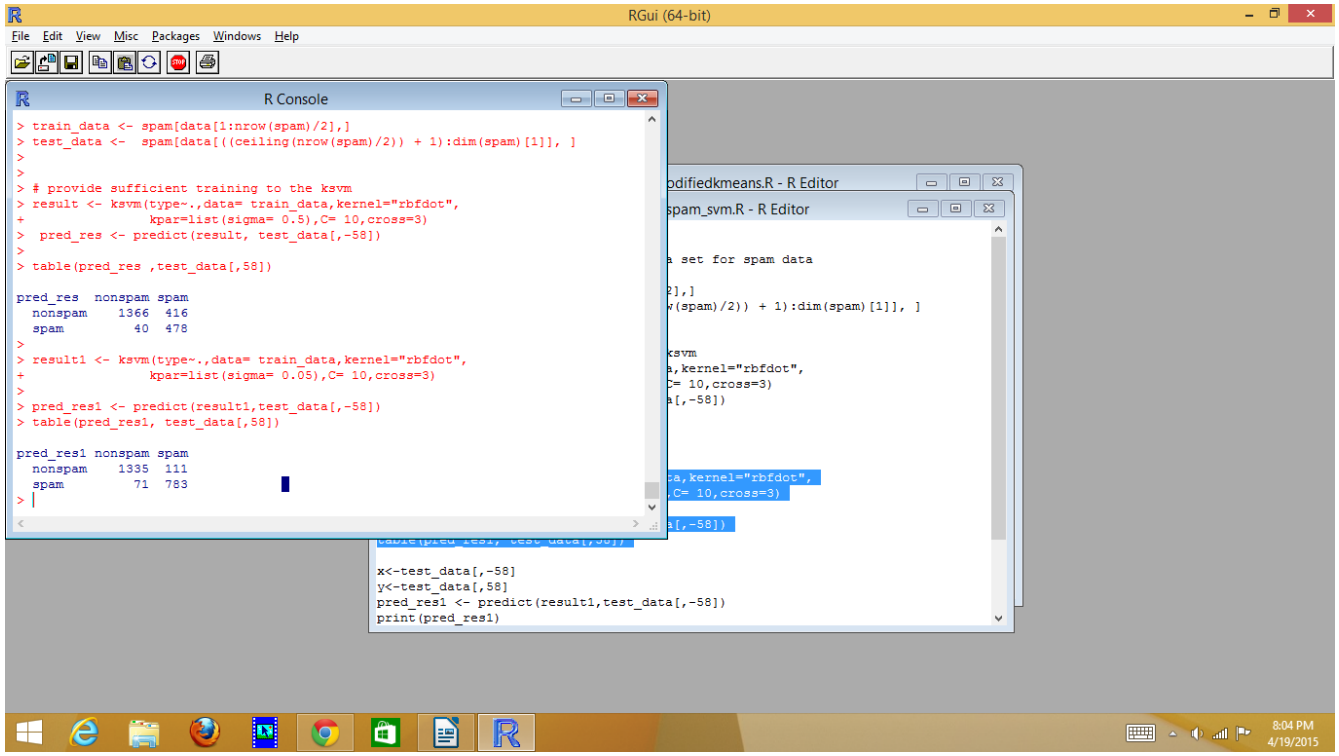

RGui (64-bit)
File Edit View Misc Packages Windows Help
R Console
>
> sum(pred==y)/length(y)
[1] 0.9733333
>
> predict(model,x)->linear.pred
Error in eval(expr, envir, enclos) : object 'make' not found
> sum(linear.pred==y)/length(y)
[1] 0.9733333
> table(linear.pred,y)
y
linear.pred setosa versicolor virginica
setosa 50 0 0
versicolor 0 48 2
virginica 0 2 48
svm_iris.R - R Editor
7:56 PM
4/19/2015

Lets Analyze the Spam data set:

```
#Creating the train data and test data set for spam data
data<- sample(1:nrow(spam))
train_data <- spam[data[1:nrow(spam)/2],]
test_data <-  spam[data[((ceiling(nrow(spam)/2)) + 1):dim(spam)[1]], ]


# provide sufficient training to the ksvm
result <- ksvm(type~.,data= train_data,kernel="rbfdot",
            kpar=list(sigma= 0.5),C= 10,cross=3)
 pred_res <- predict(result, test_data[,-58])

table(pred_res ,test_data[,58])
```

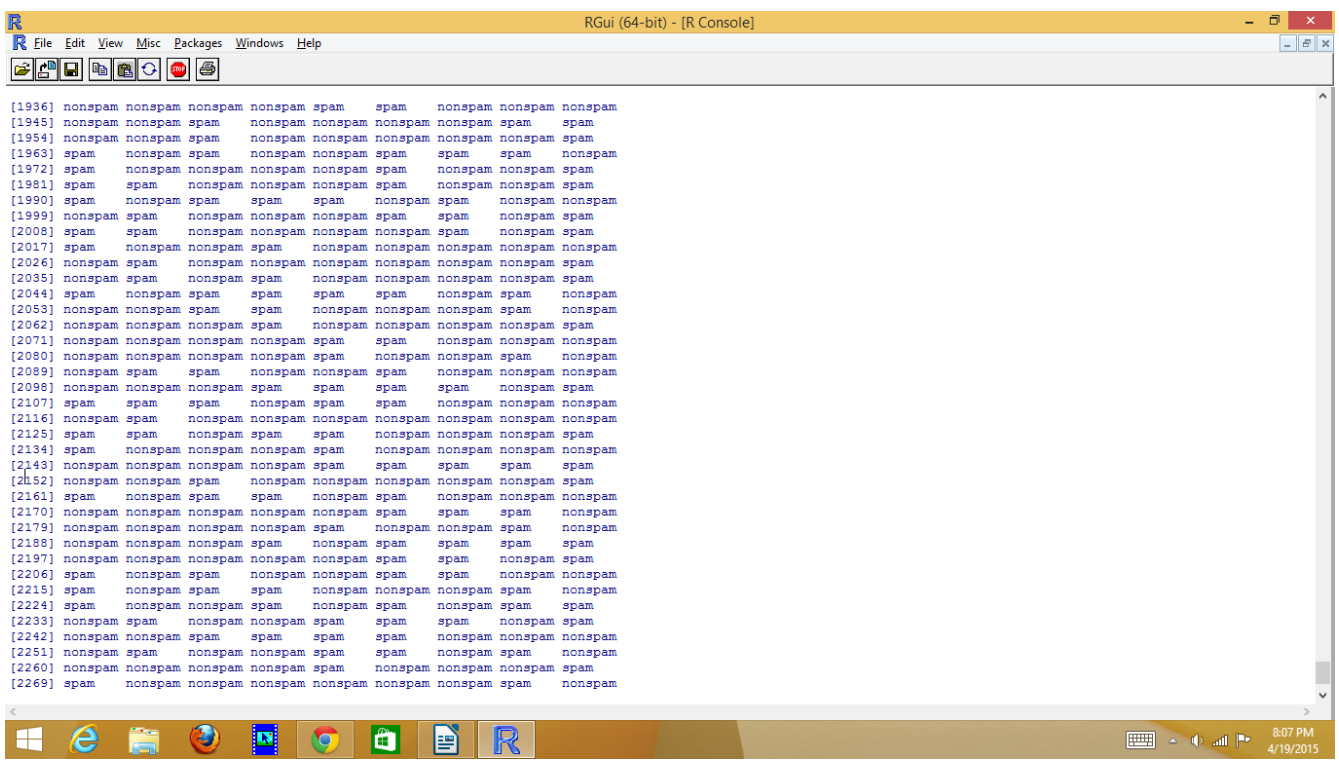



```
result1 <- ksvm(type~.,data= train_data,kernel="rbfdot",
            kpar=list(sigma= 0.05),C= 10,cross=3)

pred_res1 <- predict(result1,test_data[,-58])
table(pred_res1, test_data[,58])
```



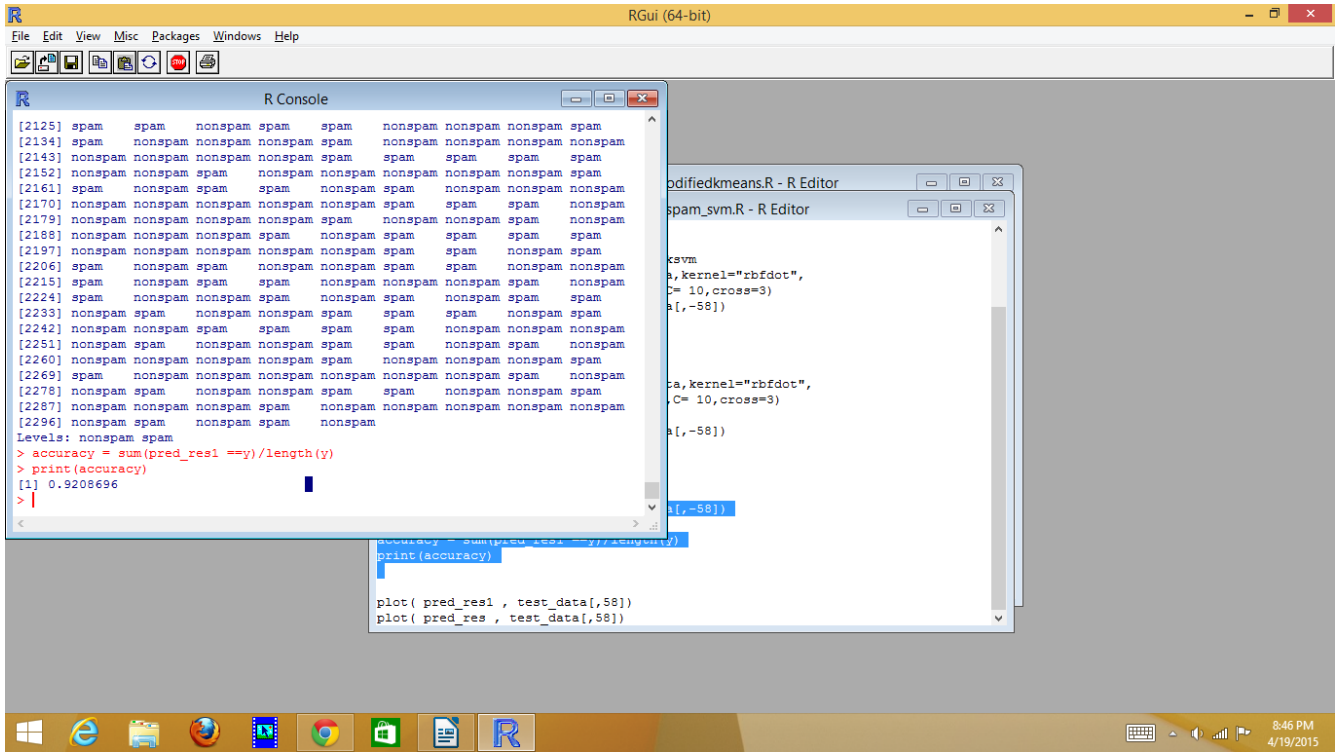
It is now clear that sigma, the variance parameter has a direct effect on the training for svm. We can clearly see that in the difference of observations we get for spam and non spam.

Prediction of svm for spam data set:



Lets now evaluate the performance of svm for the spam data set

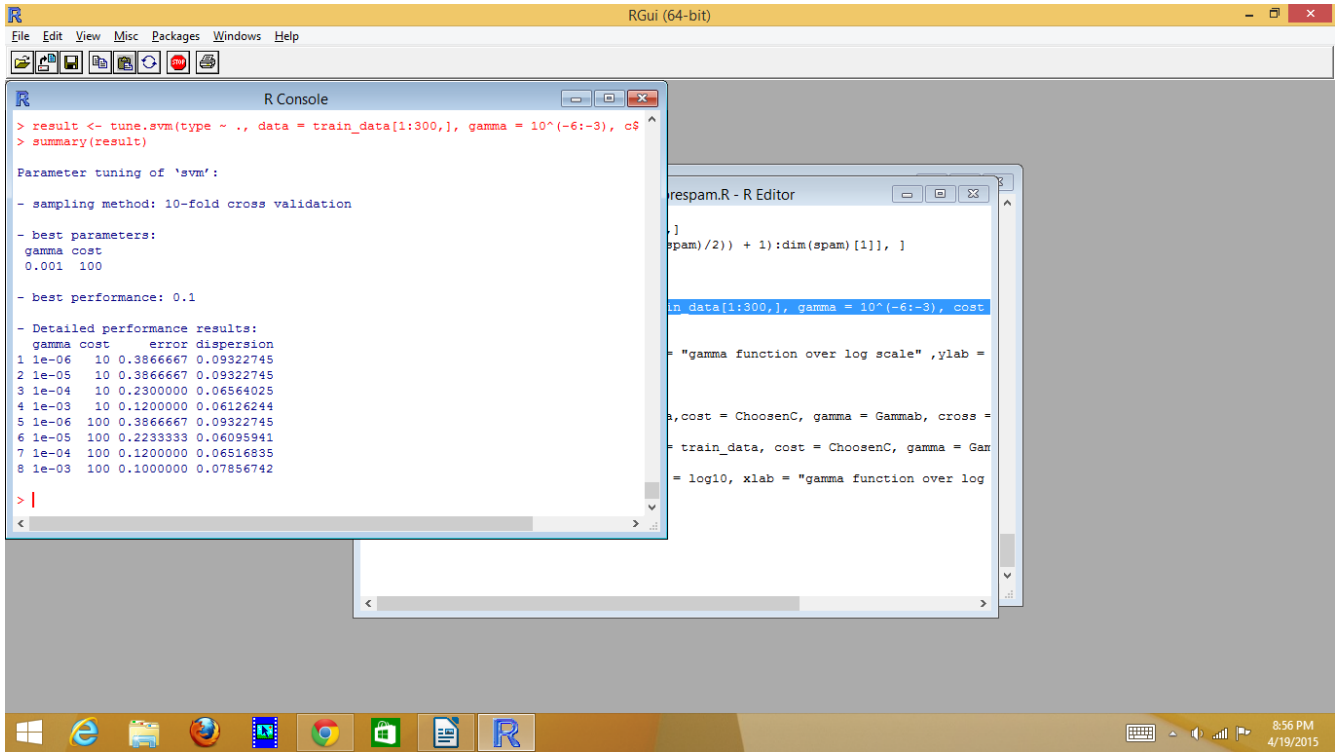
```
x<-test_data[,-58]
y<-test_data[,58]
pred_res1 <- predict(result1,test_data[,-58])
print(pred_res1)
accuracy = sum(pred_res1 ==y)/length(y)
print(accuracy)
```

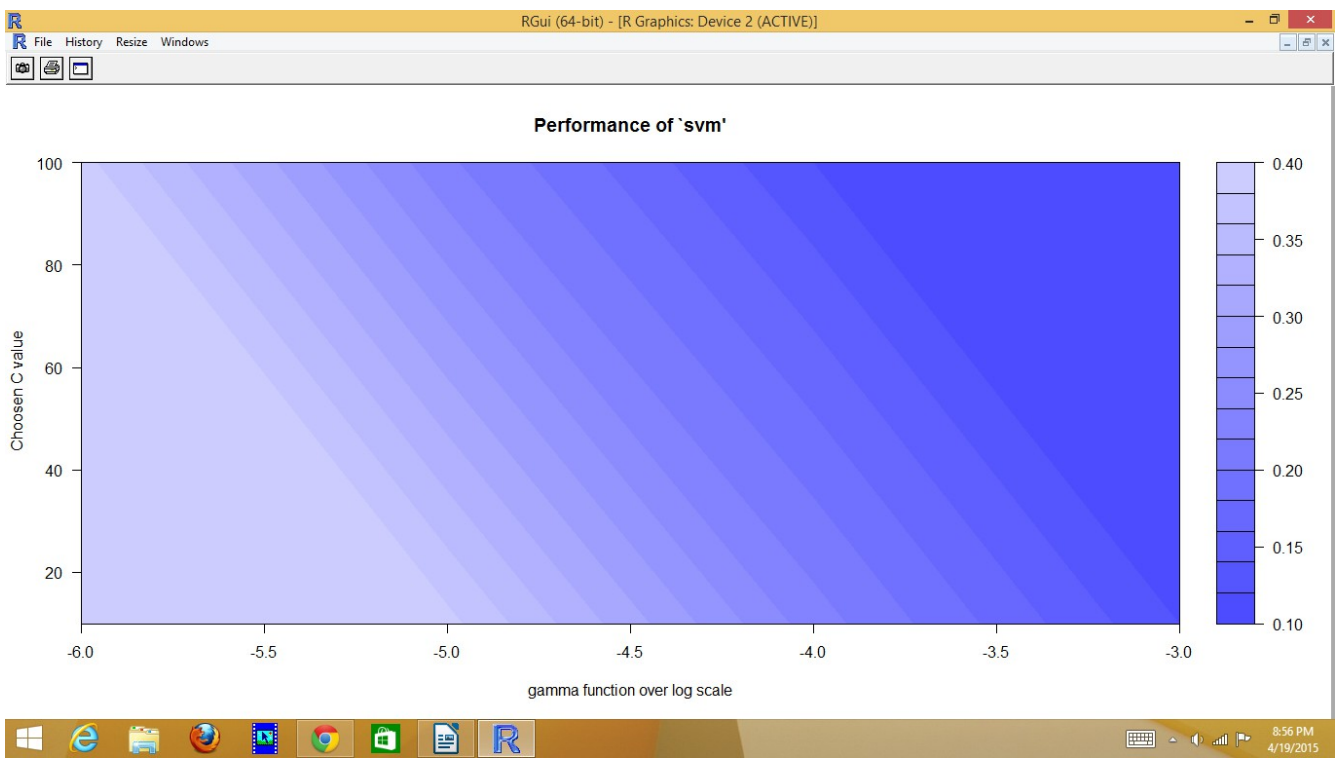


The accuracy for spam data set using the current sigma and kernel is 0.92

Tuning the svm for spam data set to improve the accuracy:

```
result <- tune.svm(type ~ ., data = train_data[1:300,], gamma = 10^(-6:-3), cost = 10^(1:2))
summary(result)
```

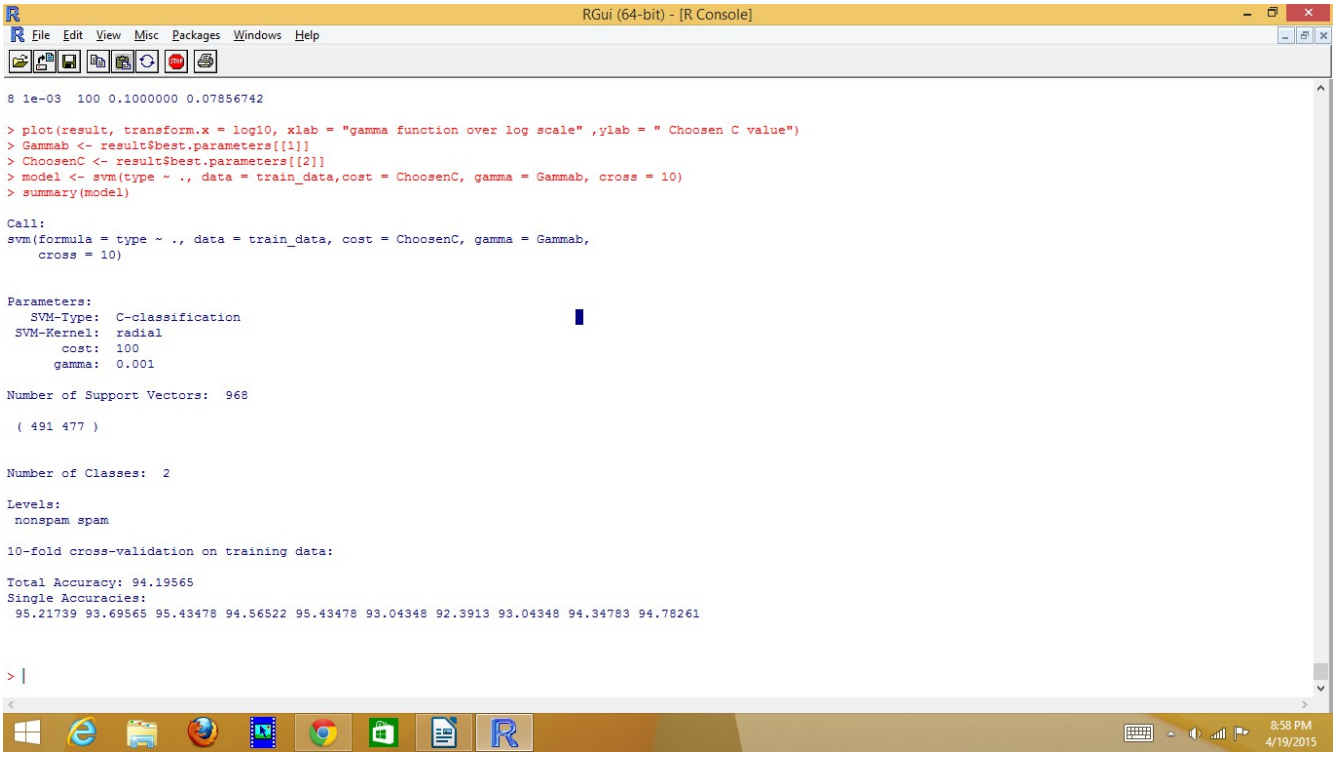




Performance over the gamma function choosen over log scale and the choosen C value.

Now we train the svm after tuning it and compare the result.

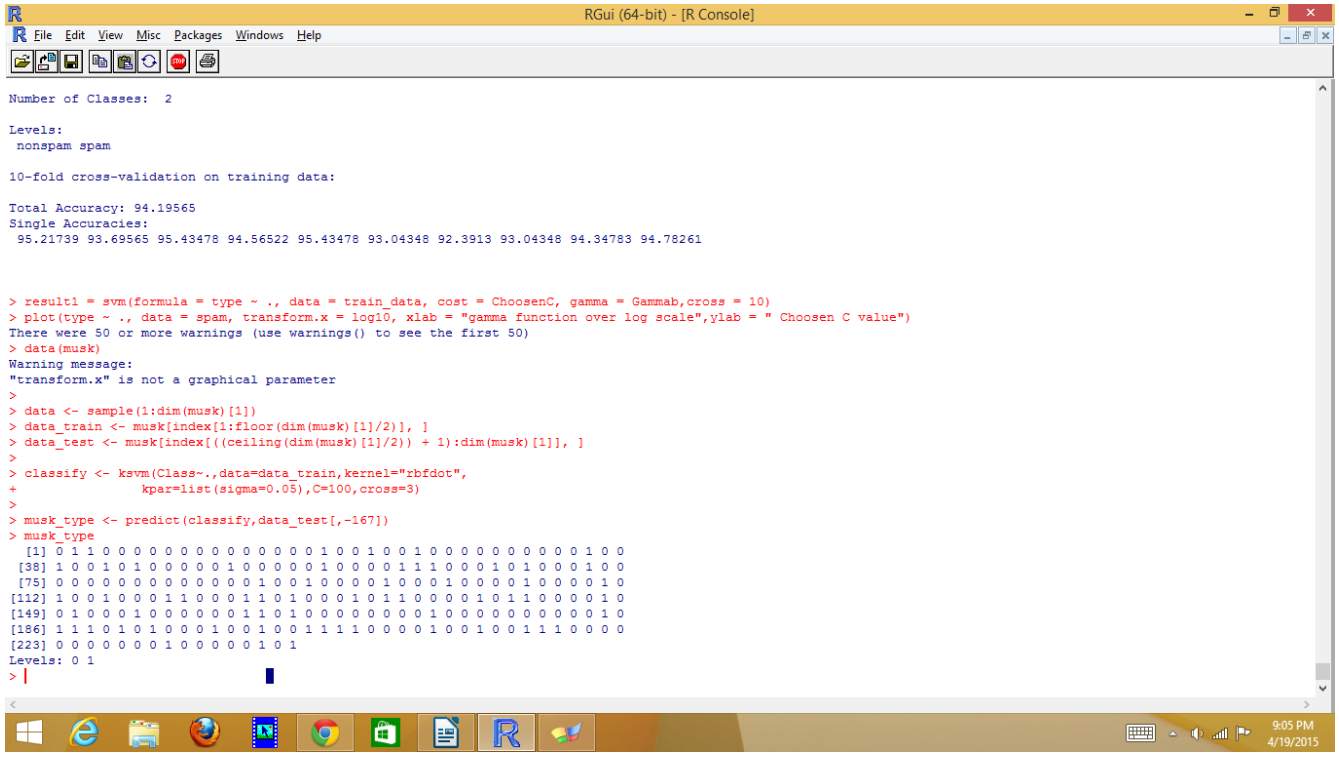
```
Gammab <- result$best.parameters[[1]]
ChoosenC <- result$best.parameters[[2]]
model <- svm(type ~ ., data = train_data,cost = ChoosenC, gamma = Gammab, cross = 10)
summary(model)
```



We have seen using the 10 fold cross validation and tuning our performance was increased by 2 and the new accuracy is 94.

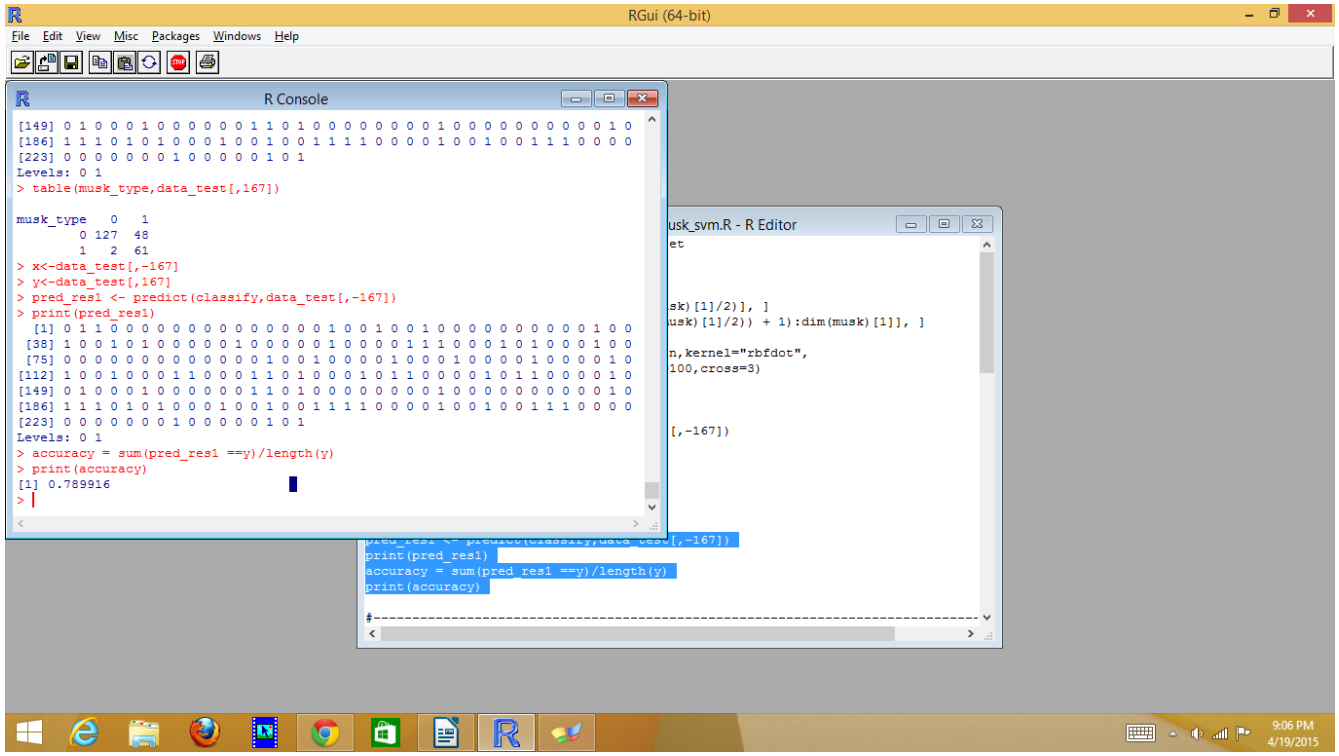
Let us analyze the musk data set.

```
data <- sample(1:dim(musk)[1])
data_train <- musk[index[1:floor(dim(musk)[1]/2)], ]
data_test <- musk[index[((ceiling(dim(musk)[1]/2)) + 1):dim(musk)[1]], ]

classify <- ksvm(Class~.,data=data_train,kernel="rbfdot",
            kpar=list(sigma=0.05),C=100,cross=3)

#musk data prediction on the test set
musk_type <- predict(classify,data_test[,-167])
```



Check results
table(musk_type,data_test[,167])

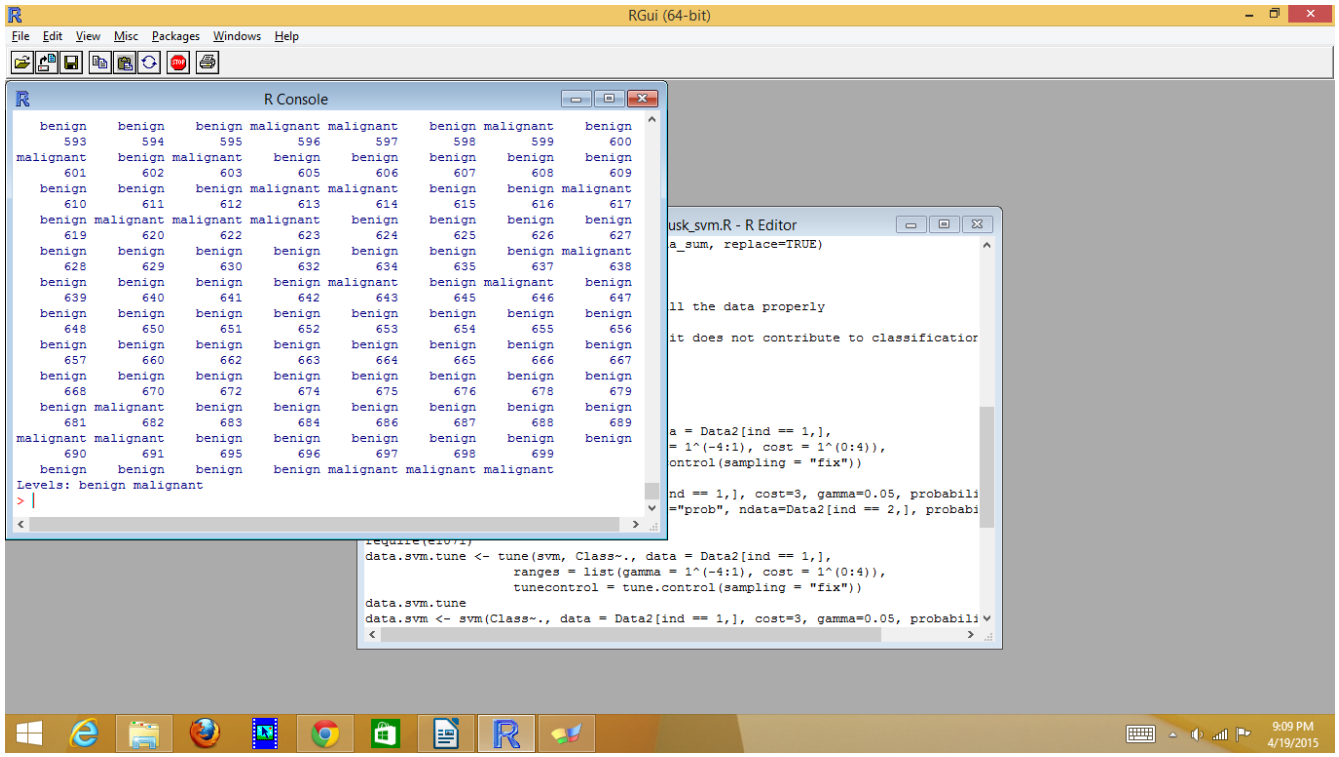
x<-data_test[,-167]
y<-data_test[,167]
pred_res1 <- predict(classify,data_test[,-167])
print(pred_res1)
accuracy = sum(pred_res1 ==y)/length(y)
print(accuracy)



We may need to perform tuning to improve the accuracy.

Breast Cancer Data:

```
#Just to be sure that we are removing all the data properly
Data2 <- na.omit(BreastCancer)
# We set the Id field to null because, it does not contribute to classification
Data2$Id =  NULL


# Performing the svm
require(e1071)
data.svm.tune <- tune(svm, Class~., data = Data2[ind == 1,],
                ranges = list(gamma = 1^(-4:1), cost = 1^(0:4)),
                tunecontrol = tune.control(sampling = "fix"))
data.svm.tune
data.svm <- svm(Class~., data = Data2[ind == 1,], cost=3, gamma=0.05, probability = TRUE)
data.svm.prob <- predict(data.svm, type="prob", ndata=Data2[ind == 2,], probability = TRUE)
```




```
data.svm <- svm(Class~., data = Data2[ind == 1,], cost=3, gamma=0.05, probability = TRUE)
data.svm.prob <- predict(data.svm, type="prob", ndata=Data2[ind == 2,], probability = TRUE)
# Performing comparision with 5 fold cross validation
cross_tab <- svm(Class ~ ., data=Data2, type="C-classification",
kernel="linear", cost=1, cross=5)
crosstab <- fitted(cross_tab)

#comparision over 5 cross validation
```

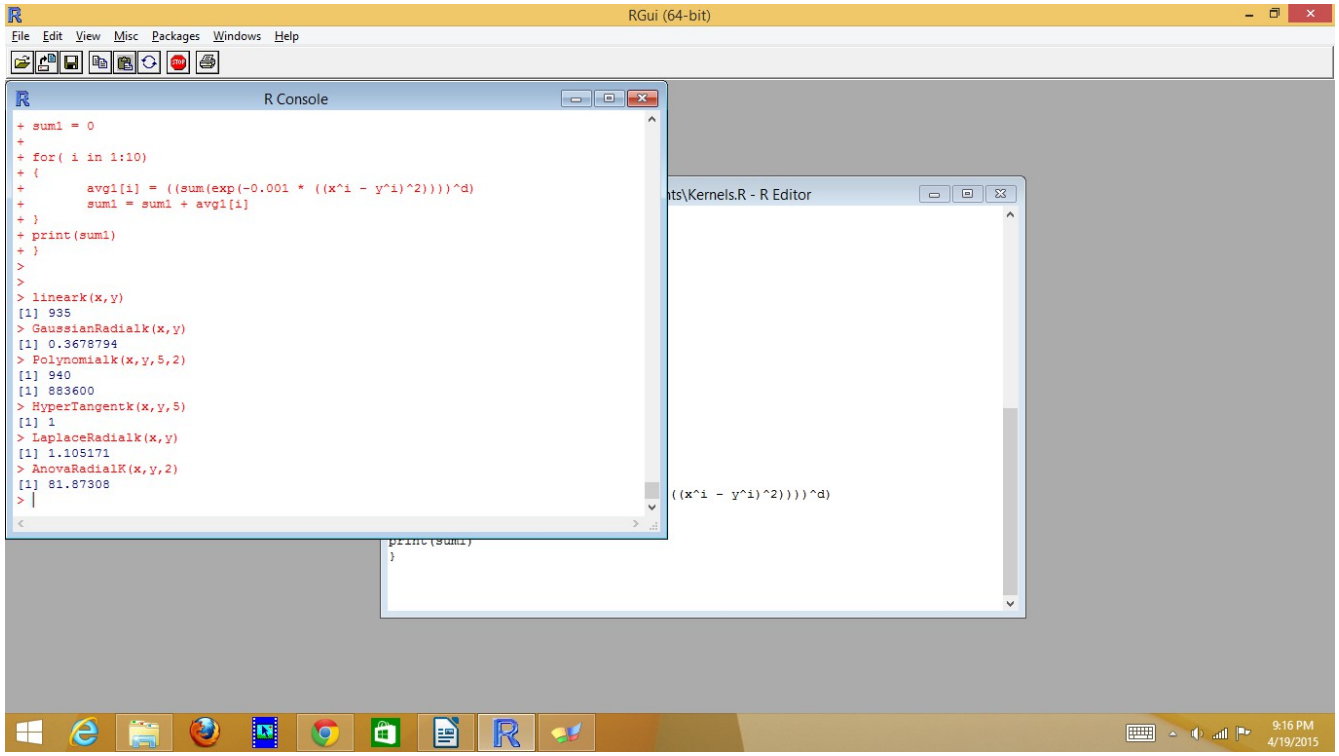
Finally let us see all the different kernels that SVM uses, all of them are almost single line of code but it is worth examining it.

Created a mock vectors for verification of data:

```
x = c(1:10)
y = c(11:20)

#linear kernel just returns the inner product between 2 points
#in a suitable inner space

lineark = function(x,y)
{
print (sum(x * y))
}

# here 0.001 is the variance of distribution it can be adjusted to suit the
# requirement over the feature space

GaussianRadialk <- function(x, y) {
```

```
print (exp(-0.001 * sum((x -y)^2)))
}

#polynomial kernel  k(x,y) = ((x transpose*y) +c)^d
#vectors of features computed from training or test samples
#c = 0 is the cost parameter for the higher order vs the lower order polynomial
#terms

Polynomialk <- function(x, y,c,d) {
print (sum (t(x) * y) + c)^ d
}

# Hyperbolic Tangent Kernel
HyperTangentk <- function(x, y,c) {
print (tanh(sum(t(x) * y) + c))
}

LaplaceRadialk <- function(x, y) {
print (exp(-0.001 * sum(x -y)))
}

#AnovaRadialK
AnovaRadialK <- function(x,y,d)
{
avg1 = c(1: length(x))
sum1 = 0

        for( i in 1:10)
        {
    avg1[i] = ((sum(exp(-0.001 * ((x^i - y^i)^2))))^d)
    sum1 = sum1 + avg1[i]
        }
print(sum1)
}
```

Sample outputs for each of the function for the mock data set is as follows.



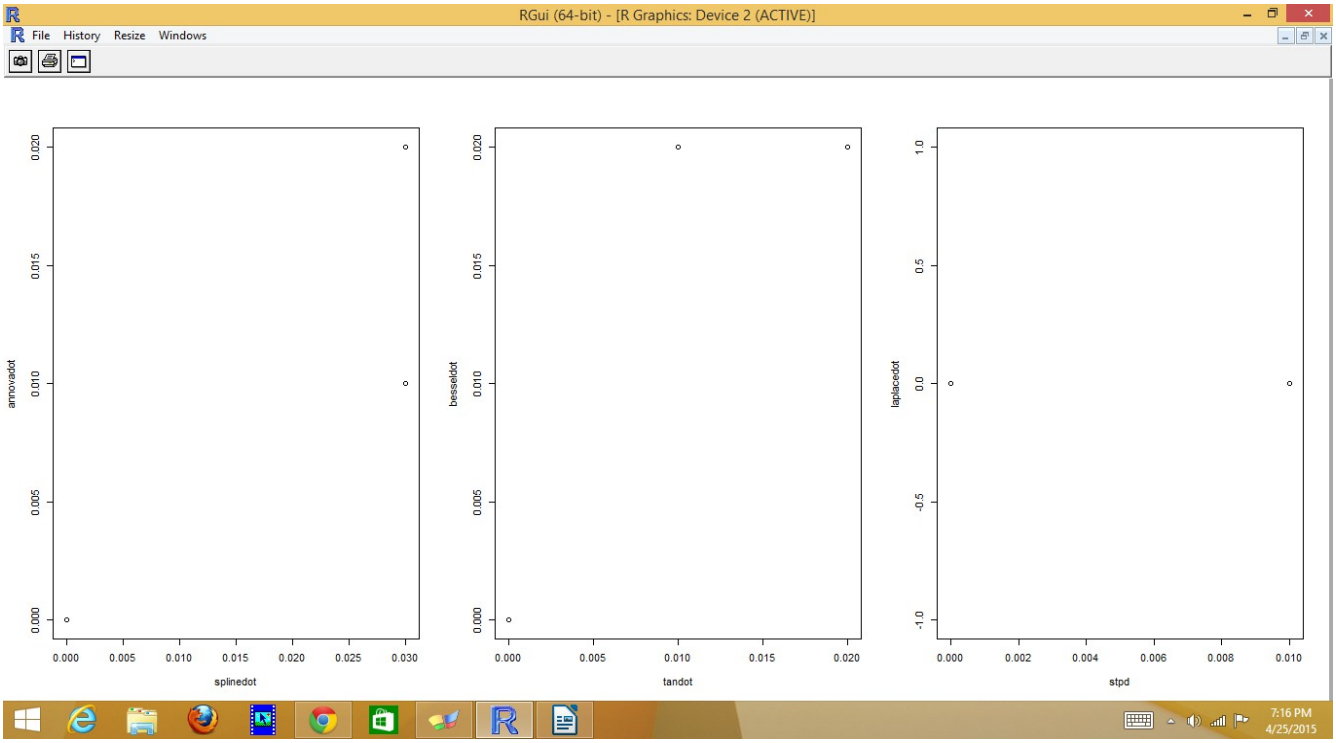
Time to Train for each data set and some plots:

Mock Data:

```
#different kernels that we can use

stpd = system.time(
train_svm1 <- ksvm(matrix_train,output_train,
type="C-svc",kernel='polydot',C=250,scaled=c()))


plot(train_svm1,data=matrix_train)

tandot = system.time(
train_svm1 <- ksvm(matrix_train,output_train,
type="C-svc",kernel='tanhdot',C=250,scaled=c()))

print(tandot)
plot(train_svm1,data=matrix_train)

laplacedot = system.time(
train_svm1 <- ksvm(matrix_train,output_train,
type="C-svc",kernel='laplacedot',C=250,scaled=c()))
print(laplacedot)
plot(train_svm1,data=matrix_train)
```

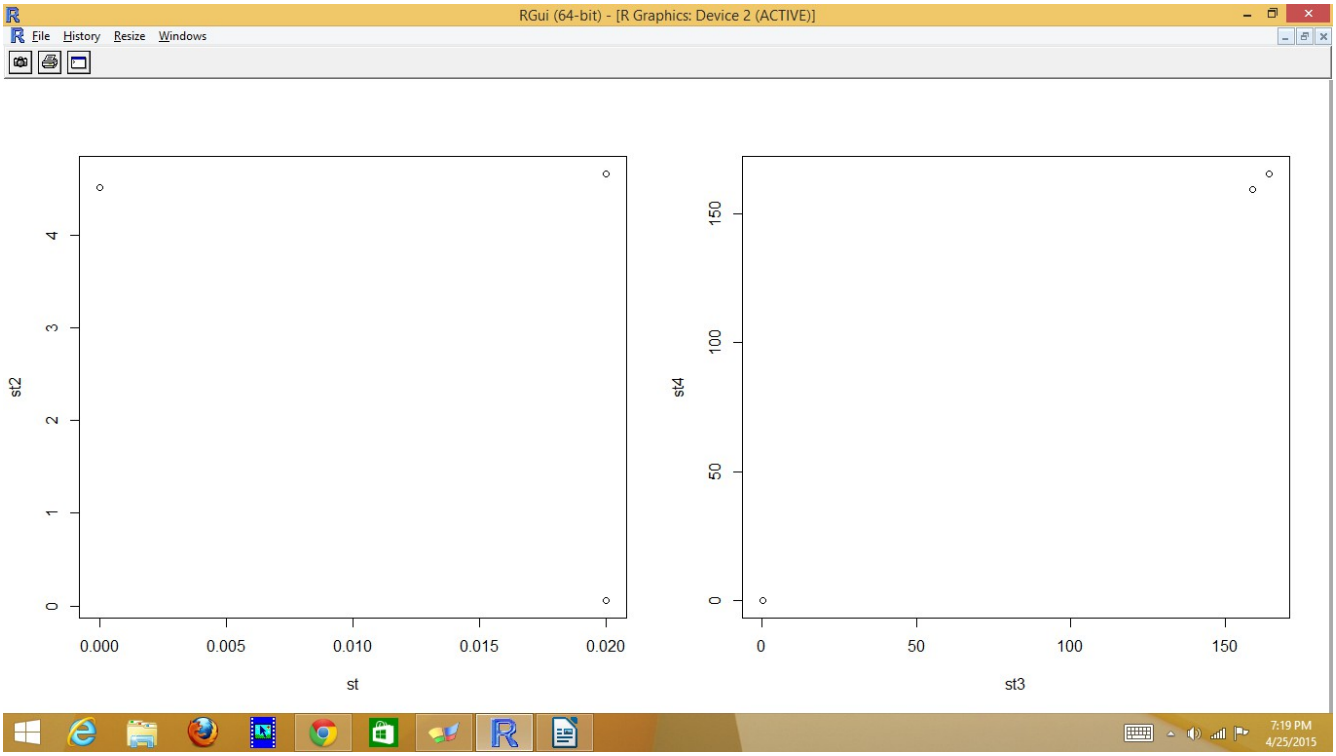
```
besseldot = system.time(
train_svm1 <- ksvm(matrix_train,output_train,
type="C-svc",kernel='besseldot',C=250,scaled=c()))

print(besseldot)

plot(train_svm1,data=matrix_train)

annovadot = system.time(
train_svm1 <- ksvm(matrix_train,output_train,
type="C-svc",kernel='anovadot',C=250,scaled=c()))

print (annovadot)
plot(train_svm1,data=matrix_train)

splinedot = system.time(
train_svm1 <- ksvm(matrix_train,output_train,
type="C-svc",kernel='splinedot',C=250,scaled=c()))
print (splinedot)

plot(train_svm1,data=matrix_train)


par(mfrow=c(1,3))
plot(splinedot,annovadot)
plot(tandot, besseldot)
plot(stpd, laplacedot)
```

Plot:



Spam Data set:

```
# provide sufficient training to the ksvm
st <- system.time(
result <- ksvm(type~.,data= train_data,kernel="rbfdot",
            kpar=list(sigma= 0.5),C= 250,cross=3))
 pred_res <- predict(result, test_data[,-58])

st2 <- system.time(
result <- ksvm(type~.,data= train_data,kernel="tanhdot",
            C= 250,cross=3))

st3 <- system.time(
result <- ksvm(type~.,data= train_data,kernel="polydot",
            C= 250,cross=3))

st4 <- system.time(
result <- ksvm(type~.,data= train_data,kernel="vanilladot",
            C= 250,cross=3))

par(mfrow=c(1,2))
plot(st, st2)
plot(st3, st4)
```



Code for Different Kernels:

```
x = c(1:10)
y = c(11:20)

#linear kernel just returns the inner product between 2 points
#in a suitable inner space

lineark = function(x,y)
{
print (sum(x * y))
}

# here 0.001 is the variance of distribution it can be adjusted to suit the
# requirement over the feature space

GaussianRadialk <- function(x, y) {
print (exp(-0.001 * sum((x -y)^2)))
}

#polynomial kernel  k(x,y) = ((x transpose*y) +c)^d
#vectors of features computed from training or test samples
#c = 0 is the cost parameter for the higher order vs the lower order polynomial
#terms

Polynomialk <- function(x, y,c,d) {
print (sum (t(x) * y) + c)^ d
}
```

```
# Hyperbolic Tangent Kernel
HyperTangentk <- function(x, y,c) {
print (tanh(sum(t(x) * y) + c))
}

LaplaceRadialk <- function(x, y) {
print (exp(-0.001 * sum(x -y)))
}

#AnovaRdialK
AnovaRadialK <- function(x,y,d)
{
avg1 = c(1: length(x))
sum1 = 0

        for( i in 1:10)
        {
    avg1[i] = ((sum(exp(-0.001 * ((x^i - y^i)^2))))^d)
    sum1 = sum1 + avg1[i]
        }
print(sum1)
}
```