SVM and KNN Classification

Question 1

I work on automated recognition. A typical binary classification problem is face recognition. Given two photos of face the system makes a determination as to whether these are two photos of the same person or different people.

Question 2 - 1

We will run a basic SVM using all the data. This is as per the notes on the assingment.

In [16]:

```
library("kernlab")
setwd("C:/Users/Jake/Documents/OMSA/Hw-1")
myData=as.matrix(read.table("credit_card_data-headers.txt",header=TRUE))
head(myData,5)

# fit model
model <- ksvm(myData[,1:10],myData[,11],type="C-svc",kernel="vanilladot",C=100,scaled=T
RUE)

# calculate each and display the weights
a <- colSums(myData[model@SVindex,1:10]* model@coef[[1]])
a0 <- sum(a*myData[1,1:10]) - model@b

a
a0

pred <- predict(model,myData[,1:10])
pred

# display accuracy
sum(pred == myData[,11]) / nrow(myData)
```

A1	A2	A3	A8	A9	A10	A11	A12	A14	A15	R1
1	30.83	0.000	1.25	1	0	1	1	202	0	1
0	58.67	4.460	3.04	1	0	6	1	43	560	1
0	24.50	0.500	1.50	1	1	0	1	280	824	1
1	27.83	1.540	3.75	1	0	5	0	100	3	1
1	20.17	5.625	1.71	1	1	0	1	120	0	1

```
Setting default kernel parameters
```

A1
-0.000466036176627327
A2
-0.0140534983606244
A3
-0.0081688661743442
A8
0.0101292226736795
A9
0.501609468692229
A10
-0.00140343386065389
A11
0.00129121684002342
A12
-0.000266898857269382
A14
-0.206754961642446
A15
558.33559056503

-41.6013574057321

```
1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 0 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1
1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

0.863914373088685

This model is clearly overfit given that we have used all the data. In the final question we will look to produce a model with a more relaible accuracy measure.

Question 2-2

We will fit a single KNN model. We will take it a step further and perform some training, comparing the best performance between 1 and 9.

In [9]:

```r
#reload data
library("kknn")
myData <- read.table("credit_card_data-headers.txt",header=TRUE)
head(myData, 5)

for(i in 1:10) {
  myData[,i] <- as.numeric(as.character(Data[,i]))
}
myData$R1 <- as.factor(Data$R1)

# remove 50 entries for assesment of model
m <- dim(Data)[1]
Sample <- sample(1:m, 50)
testing <- myData[Sample, ]
learning <- myData[-Sample, ]

dim(learning)
dim(testing)

# assess models
model_train <- train.kknn(R1 ~ ., data=learning, scale=TRUE, kmax=9)
model_train

prediction <- predict(model_train, testing[, -11])
CM <- table(testing[, 11], prediction)
CM

accuracy <- (sum(diag(CM)))/sum(CM)
accuracy
```

A1	A2	A3	A8	A9	A10	A11	A12	A14	A15	R1
1	30.83	0.000	1.25	1	0	1	1	202	0	1
0	58.67	4.460	3.04	1	0	6	1	43	560	1
0	24.50	0.500	1.50	1	1	0	1	280	824	1
1	27.83	1.540	3.75	1	0	5	0	100	3	1
1	20.17	5.625	1.71	1	1	0	1	120	0	1

604 11

50 11

```
Call:
train.kknn(formula = R1 ~ ., data = learning, kmax = 9, scale = TRUE)

Type of response variable: nominal
Minimal misclassification: 0.1440397
Best kernel: optimal
Best k: 5
```

```
   prediction
     0  1
  0 22  6
  1  5 17
```

0.78

We acheived a 0.9 accuracy with k=7. We will use this to finalize a model.

In [5]:

```
# finalize model model
model <- kknn(R1 ~ ., train=learning, test=testing, scale=TRUE, k=7)
model

summary(model)
fit <- fitted(model)


CM <- table(testing$R1, fit)
CM

accuracy <- (sum(diag(CM)))/sum(CM)
accuracy
```

```
Call:
kknn(formula = R1 ~ ., train = learning, test = testing, k = 7,     scale
 = TRUE)

Response: "nominal"
```

```
Call:
kknn(formula = R1 ~ ., train = learning, test = testing, k = 7,     scale
 = TRUE)

Response: "nominal"
   fit     prob.0     prob.1
1    1 0.00000000 1.00000000
2    0 0.77080726 0.22919274
3    1 0.07275176 0.92724824
4    1 0.11329303 0.88670697
5    1 0.07275176 0.92724824
6    0 0.95945873 0.04054127
7    0 0.65944241 0.34055759
8    0 1.00000000 0.00000000
9    0 1.00000000 0.00000000
10   0 1.00000000 0.00000000
11   0 1.00000000 0.00000000
12   1 0.00000000 1.00000000
13   0 1.00000000 0.00000000
14   1 0.22919274 0.77080726
15   1 0.11136485 0.88863515
16   1 0.22919274 0.77080726
17   0 0.80313939 0.19686061
18   0 0.62686494 0.37313506
19   1 0.30194449 0.69805551
20   0 1.00000000 0.00000000
21   1 0.38946305 0.61053695
22   1 0.44588682 0.55411318
23   1 0.41367634 0.58632366
24   0 0.95945873 0.04054127
25   0 1.00000000 0.00000000
26   0 1.00000000 0.00000000
27   0 1.00000000 0.00000000
28   1 0.41367634 0.58632366
29   0 0.62686494 0.37313506
30   1 0.00000000 1.00000000
31   1 0.00000000 1.00000000
32   1 0.00000000 1.00000000
33   1 0.37313506 0.62686494
34   0 0.88863515 0.11136485
35   0 1.00000000 0.00000000
36   1 0.27163517 0.72836483
37   1 0.26973401 0.73026599
38   1 0.00000000 1.00000000
39   1 0.00000000 1.00000000
40   1 0.22919274 0.77080726
41   1 0.22919274 0.77080726
42   1 0.00000000 1.00000000
43   0 1.00000000 0.00000000
44   0 1.00000000 0.00000000
45   0 0.62686494 0.37313506
46   0 1.00000000 0.00000000
47   1 0.28247801 0.71752199
48   0 1.00000000 0.00000000
49   1 0.01274400 0.98725600
50   1 0.49724391 0.50275609

   fit
     0  1
  0 23  5
  1  0 22
```

0.9

Again we shouldn't expect this to generalise at 90%. In the next question we will use k-folds to produce more robust measures

Question 3

In [18]:

```
# I am going to use the caret libray for classfiers
# detach("package:kernlab", unload=TRUE)
# detach("package:kknn", unload=TRUE)
library(caret)

# Load data into df
myData <- read.table("credit_card_data-headers.txt",header=TRUE)

# convert to numeric data
for(i in 1:10) {
  myData[,i] <- as.numeric(as.character(myData[,i]))
}
myData$R1 <- as.factor(myData$R1)

# class distribution
cbind(freq=table(myData$R1), percentage=prop.table(table(myData$R1))*100)

# summarize correlations between input variables
complete_cases <- complete.cases(myData)
cor(myData[complete_cases,1:10])
```

	freq	percentage
0	358	54.74006
1	296	45.25994

	A1	A2	A3	A8	A9	A10
A1	1.00000000	0.04580212	-0.03475616	0.08433408	-0.02227002	0.06603888
A2	0.04580212	1.00000000	0.21573658	0.40910052	0.22151088	-0.09674680
A3	-0.03475616	0.21573658	1.00000000	0.30043424	0.23678042	-0.16580972
A8	0.08433408	0.40910052	0.30043424	1.00000000	0.33513714	-0.22853844
A9	-0.02227002	0.22151088	0.23678042	0.33513714	1.00000000	-0.42878000
A10	0.06603888	-0.09674680	-0.16580972	-0.22853844	-0.42878000	1.00000000
A11	-0.01719151	0.19245630	0.26967523	0.32758743	0.37722077	-0.56940610
A12	-0.05131107	-0.05158808	0.00626837	-0.13991523	-0.08842336	0.02145246
A14	0.07642180	-0.09013265	-0.21710129	-0.06388272	-0.05866532	0.03748592
A15	0.01296216	0.02790891	0.11972427	0.05224694	0.08418701	-0.06832104

In [22]:

```
set.seed(42)
validation_index <- createDataPartition(myData$R1, p=0.80, list=FALSE)
validation <- myData[-validation_index,]
temp <- myData[validation_index,]
test_index <- createDataPartition(temp$R1, p=0.75, list=FALSE)
testing <- temp[-test_index,]
training <- temp[test_index,]

# manually running c through ksvm, I found insensitivity with the vanilladot kernel and
 with rbf diminished
# as I moved away from 1.
x <- training[,1:10]
x <- as.matrix(x)
y <- as.numeric(training$R1)
x_test <- testing[,1:10]
x_test <- as.matrix(x_test)
y_test <- as.numeric(testing$R1)
c=seq(1,200, by=20)
for(i in 1:10) {
  model<-ksvm(x,y,type="C-svc",kernel="rbf",C=c[i],scaled=TRUE)
  pred_svm <- predict(model,x_test)
  CM <- table(y_test, pred_svm)
  accuracy <- (sum(diag(CM)))/sum(CM)
  print(c[i])
  print(CM)
  print(accuracy)
}

# Convert DF values to numeric
for(i in 1:10) {
  myData[,i] <- as.numeric(as.character(myData[,i]))
}
myData$R1 <- as.factor(myData$R1)

# manually running k through kknn
k=seq(1,20,by=1)
for(i in 1:20) {
  model_knn <- kknn(R1 ~ ., train=training, test=testing, scale=TRUE, k=k[i])
  fit <- fitted(model_knn)
  CM <- table(testing$R1, fit)
  print(k[i])
  print(CM)
  accuracy <- (sum(diag(CM)))/sum(CM)
  print(accuracy)
}
```

```
[1] 1
      pred_svm
y_test  1  2
     1 58 13
     2  6 53
[1] 0.8538462
[1] 21
      pred_svm
y_test  1  2
     1 62  9
     2 16 43
[1] 0.8076923
[1] 41
      pred_svm
y_test  1  2
     1 61 10
     2 15 44
[1] 0.8076923
[1] 61
      pred_svm
y_test  1  2
     1 61 10
     2 20 39
[1] 0.7692308
[1] 81
      pred_svm
y_test  1  2
     1 61 10
     2 19 40
[1] 0.7769231
[1] 101
      pred_svm
y_test  1  2
     1 61 10
     2 20 39
[1] 0.7692308
[1] 121
      pred_svm
y_test  1  2
     1 61 10
     2 20 39
[1] 0.7692308
[1] 141
      pred_svm
y_test  1  2
     1 60 11
     2 20 39
[1] 0.7615385
[1] 161
      pred_svm
y_test  1  2
     1 60 11
     2 20 39
[1] 0.7615385
[1] 181
      pred_svm
y_test  1  2
     1 60 11
     2 21 38
[1] 0.7538462
[1] 1
```

```
   fit
     0  1
  0 61 10
  1 17 42
[1] 0.7923077
[1] 2
   fit
     0  1
  0 61 10
  1 17 42
[1] 0.7923077
[1] 3
   fit
     0  1
  0 61 10
  1 17 42
[1] 0.7923077
[1] 4
   fit
     0  1
  0 61 10
  1 17 42
[1] 0.7923077
[1] 5
   fit
     0  1
  0 62  9
  1 16 43
[1] 0.8076923
[1] 6
   fit
     0  1
  0 63  8
  1 16 43
[1] 0.8153846
[1] 7
   fit
     0  1
  0 63  8
  1 17 42
[1] 0.8076923
[1] 8
   fit
     0  1
  0 63  8
  1 17 42
[1] 0.8076923
[1] 9
   fit
     0  1
  0 63  8
  1 18 41
[1] 0.8
[1] 10
   fit
     0  1
  0 62  9
  1 15 44
[1] 0.8153846
[1] 11
   fit
```

```
     0  1
  0 62  9
  1 15 44
[1] 0.8153846
[1] 12
   fit
     0  1
  0 62  9
  1 15 44
[1] 0.8153846
[1] 13
   fit
     0  1
  0 61 10
  1 14 45
[1] 0.8153846
[1] 14
   fit
     0  1
  0 61 10
  1 12 47
[1] 0.8307692
[1] 15
   fit
     0  1
  0 61 10
  1 12 47
[1] 0.8307692
[1] 16
   fit
     0  1
  0 61 10
  1 12 47
[1] 0.8307692
[1] 17
   fit
     0  1
  0 61 10
  1 13 46
[1] 0.8230769
[1] 18
   fit
     0  1
  0 61 10
  1 13 46
[1] 0.8230769
[1] 19
   fit
     0  1
  0 61 10
  1 13 46
[1] 0.8230769
[1] 20
   fit
     0  1
  0 61 10
  1 14 45
[1] 0.8153846
```

The classes are marginally unbalanced, the corrolation looks good. I am not going to remove any data.

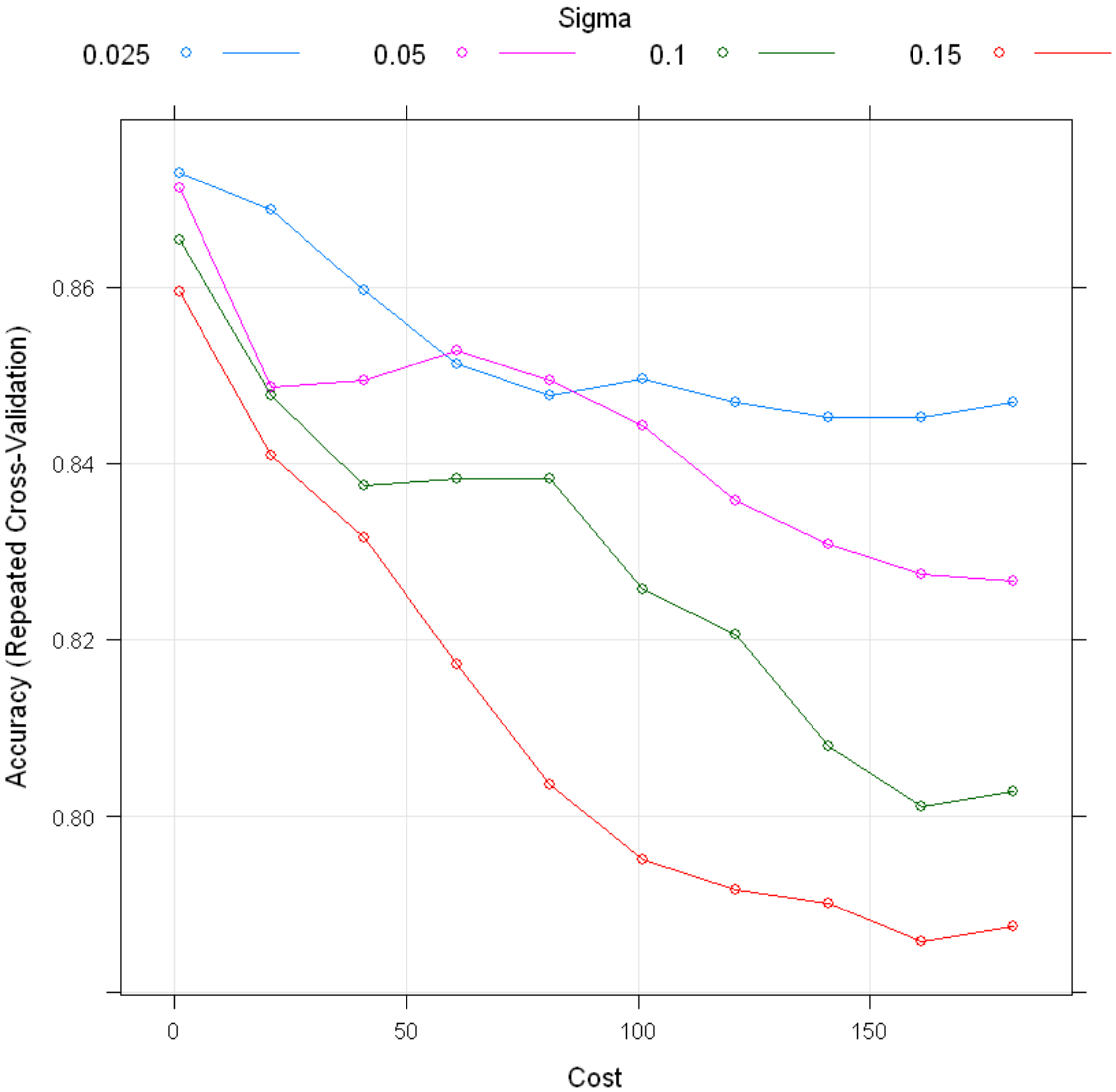
The approach before used k = 10 for k-folds cross validation with 3 repeats. We then run it through a grid search to find the best performing data. Following this we will assess the model accuracy on the validation data.

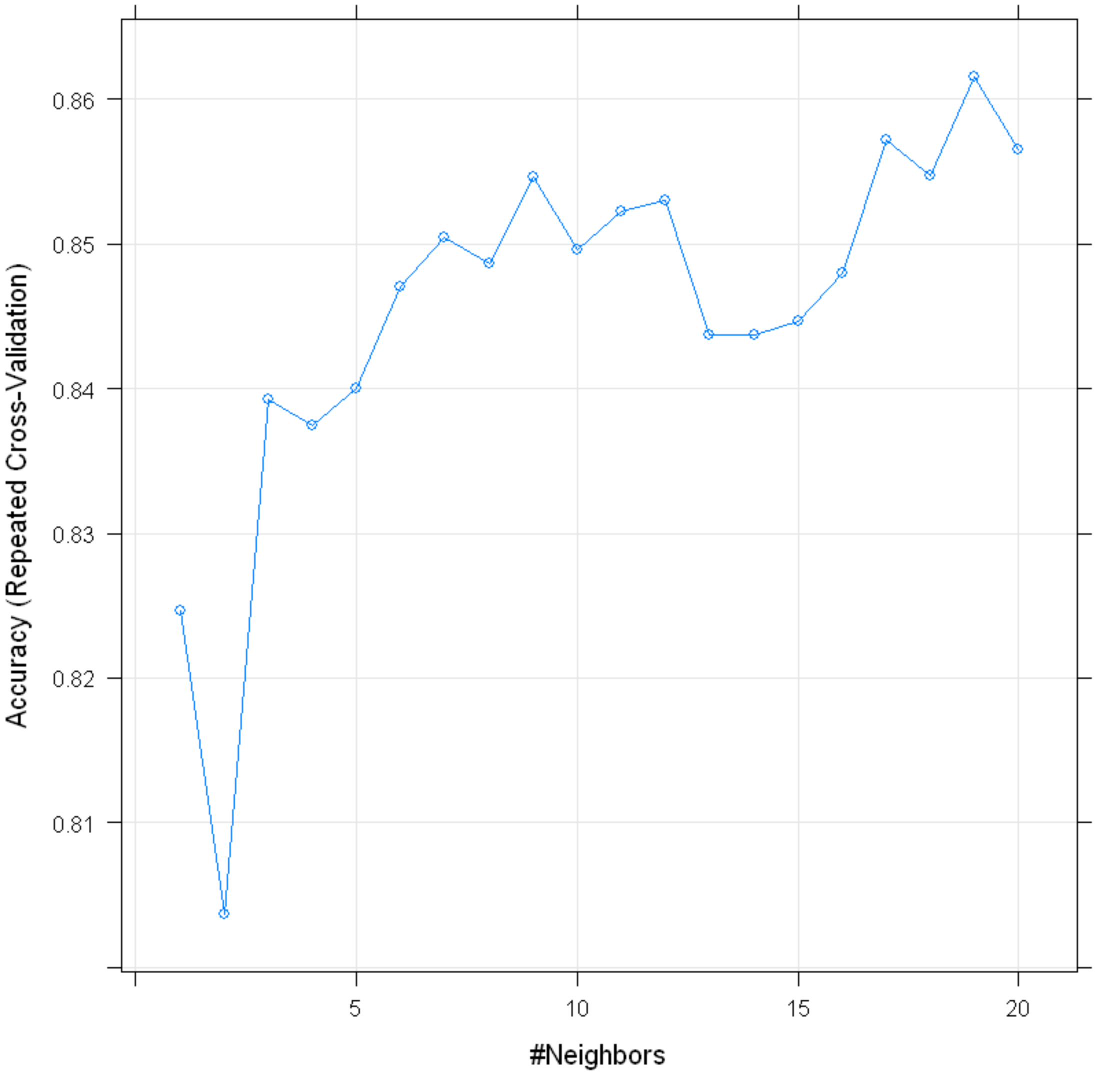
In [23]:

```
# Tune SVM
# 10-fold cross validation with 3 repeats
control <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "Accuracy"
set.seed(7)
grid <- expand.grid(.sigma=c(0.025, 0.05, 0.1, 0.15), .C=seq(1, 200, by=20))
fit.svm <- train(R1~., data=training, method="svmRadial", metric=metric, tuneGrid=grid,
 preProc=c("scale"), trControl=control)
print(fit.svm)
plot(fit.svm)


# Tune kNN
# 10-fold cross validation with 3 repeats
control <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "Accuracy"
set.seed(7)
grid <- expand.grid(.k=seq(1,20,by=1))
fit.knn <- train(R1~., data=training, method="knn", metric=metric, tuneGrid=grid, prePr
oc=c("scale"), trControl=control)
print(fit.knn)
plot(fit.knn)
```

```
Support Vector Machines with Radial Basis Function Kernel

394 samples
 10 predictor
  2 classes: '0', '1'

Pre-processing: scaled (10)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 354, 354, 355, 355, 355, 355, ...
Resampling results across tuning parameters:

  sigma  C    Accuracy   Kappa
  0.025    1  0.8729341  0.7484916
  0.025   21  0.8687910  0.7389673
  0.025   41  0.8596457  0.7194125
  0.025   61  0.8512045  0.7015536
  0.025   81  0.8477215  0.6940734
  0.025  101  0.8494962  0.6976564
  0.025  121  0.8469523  0.6924786
  0.025  141  0.8452429  0.6889450
  0.025  161  0.8452002  0.6888132
  0.025  181  0.8468882  0.6920568
  0.050    1  0.8712461  0.7450740
  0.050   21  0.8485751  0.6959329
  0.050   41  0.8494309  0.6974511
  0.050   61  0.8527429  0.7036564
  0.050   81  0.8493893  0.6965780
  0.050  101  0.8443252  0.6858923
  0.050  121  0.8357973  0.6686327
  0.050  141  0.8307760  0.6583348
  0.050  161  0.8274438  0.6512451
  0.050  181  0.8265891  0.6492818
  0.100    1  0.8653261  0.7331355
  0.100   21  0.8476361  0.6925564
  0.100   41  0.8374629  0.6715271
  0.100   61  0.8382535  0.6723041
  0.100   81  0.8383187  0.6724730
  0.100  101  0.8256455  0.6466661
  0.100  121  0.8205601  0.6364893
  0.100  141  0.8078464  0.6101618
  0.100  161  0.8010717  0.5967633
  0.100  181  0.8027598  0.6004378
  0.150    1  0.8595153  0.7211370
  0.150   21  0.8408176  0.6782704
  0.150   41  0.8315879  0.6592319
  0.150   61  0.8171413  0.6305625
  0.150   81  0.8035279  0.6025607
  0.150  101  0.7950900  0.5853625
  0.150  121  0.7916700  0.5782091
  0.150  141  0.7900259  0.5741907
  0.150  161  0.7857951  0.5649980
  0.150  181  0.7875270  0.5688254

Accuracy was used to select the optimal model using  the largest value.
The final values used for the model were sigma = 0.025 and C = 1.
```

```
k-Nearest Neighbors

394 samples
 10 predictor
  2 classes: '0', '1'

Pre-processing: scaled (10)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 354, 354, 355, 355, 355, 355, ...
Resampling results across tuning parameters:

  k   Accuracy   Kappa
   1  0.8246188  0.6449127
   2  0.8037213  0.6025591
   3  0.8392578  0.6761542
   4  0.8374640  0.6727079
   5  0.8400270  0.6790351
   6  0.8469984  0.6925101
   7  0.8504161  0.6995952
   8  0.8486640  0.6961528
   9  0.8546064  0.7075619
  10  0.8495625  0.6973627
  11  0.8522143  0.7017616
  12  0.8530061  0.7032534
  13  0.8437101  0.6839648
  14  0.8437303  0.6839817
  15  0.8446278  0.6850539
  16  0.8479791  0.6917892
  17  0.8572323  0.7103834
  18  0.8546907  0.7048272
  19  0.8615081  0.7187452
  20  0.8565092  0.7086244

Accuracy was used to select the optimal model using  the largest value.
The final value used for the model was k = 19.
```


Sigma
0.025
0.05
0.1
0.15
Accuracy (Repeated Cross-Validation)
0.86
0.84
0.82
0.80
0
50
100
150
Cost



We found k=19 to be best with 86% accuracy. SVM experienced similiar performance with sigma = 0.025 and C = 1 I will build each model we can compare each against the validation data.

In [24]:

```
# build final model svm
x <- training[,1:10]
x <- as.matrix(x)
y <- as.numeric(training$R1)
model_svm <- ksvm(x, y,type="C-svc",kernel="rbf",simga=0.025,C=1)

x_valid <- as.matrix(validation[,1:10])
y_valid <- as.numeric(validation$R1)
pred_svm <- predict(model,x_valid)
CM <- table(y_valid, pred_svm)
CM
accuracy <- (sum(diag(CM)))/sum(CM)
accuracy

# Acheived 83.8 accuracy on the validation set
```

```
       pred_svm
y_valid  1  2
      1 59 12
      2 13 46
```

0.807692307692308

In [25]:

```
# build final model knn
model_knn <- kknn(R1 ~ ., train=training, test=validation, scale=TRUE, k=19)
fit <- fitted(model_knn)
CM <- table(validation$R1, fit)
CM
accuracy <- (sum(diag(CM)))/sum(CM)
accuracy
```

```
   fit
     0  1
  0 56 15
  1  8 51
```

0.823076923076923

The accuracy on the validation suggest our accuracy with the algorithims is 80.7% for the svm and 82.3% for knn.