Packages included:

```
require(ggplot2)
require(e1071)
require(gridExtra)
require(neuralnet)
require(reshape)
```

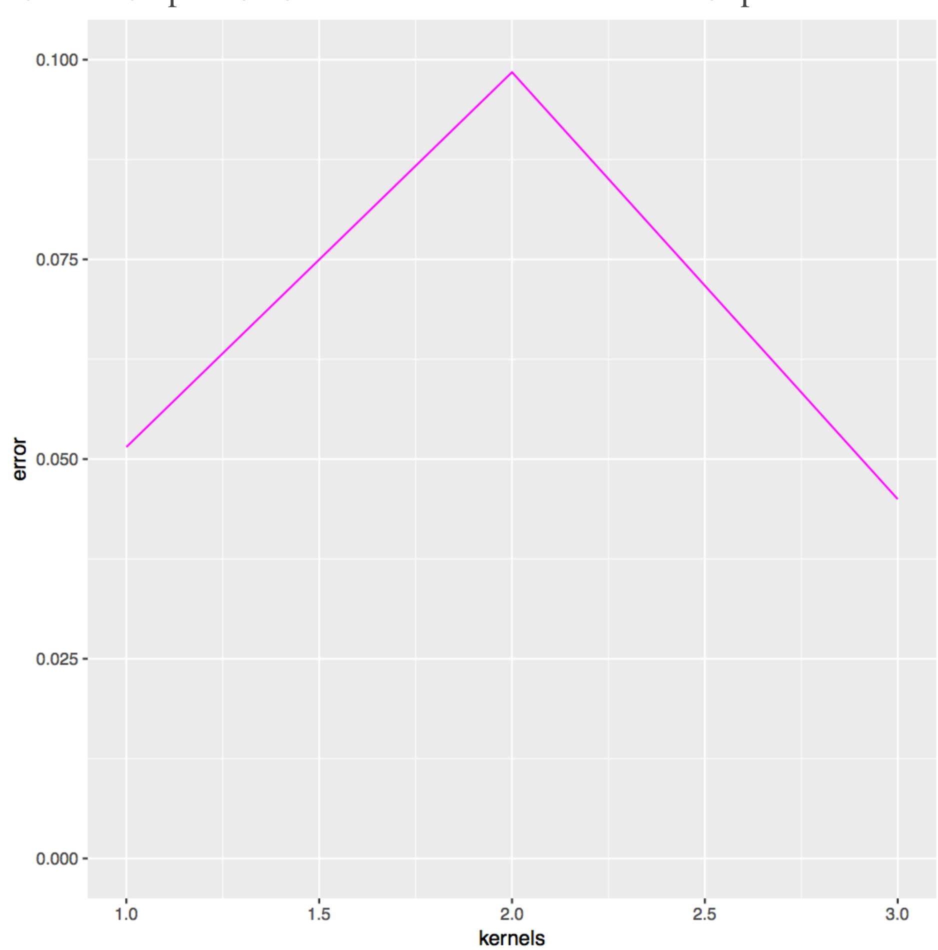
I used the log-transformed data for the analysis.

```
test=data.frame(log(1+test[, -58]), spam=factor(test[,58]))
train=data.frame(log(1+train[,-58]), spam=factor(train[,58]))
```

1. Choice of the kernels (linear, polynomial, Gaussian)

Since the test data is given and no special requirement of CV here, we use test error to compare sensitiveness of changing parameters and kernels.

For the comparison of kernels I use the default set of parameters.

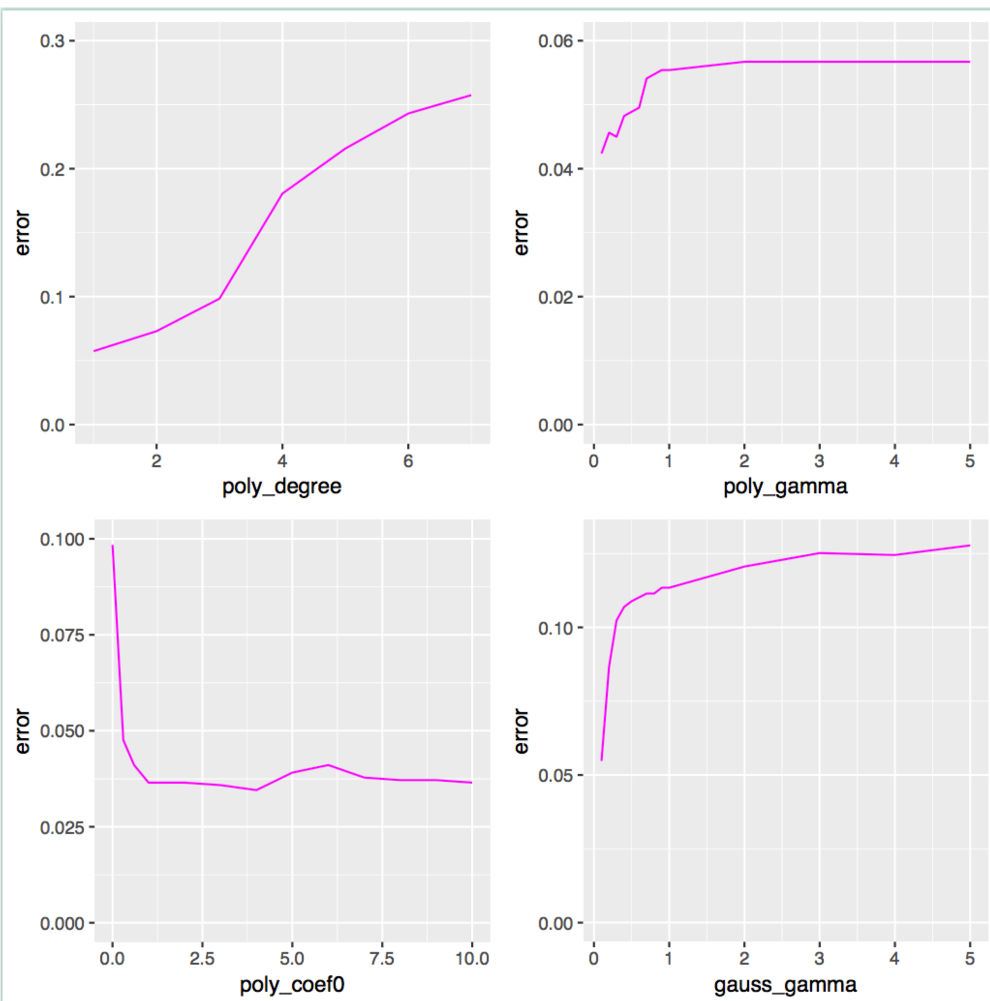


Comment: from the test error of using linear, polynomial and Gaussian kernels we can see that based on the default setting: Gaussian kernel gives the smallest test error for SVM, Linear kernel is the second smallest and polynomial is the largest.

Code:

```
errors=c()
spam_svm=svm(spam~.,data=train,kernel="linear",cost=1)
errors[1]=sum(predict(spam_svm, test)!=test$spam)/length(test[,1])
spam_svm1=svm(spam~.,data=train,kernel="polynomial",cost=1)
errors[2]=sum(predict(spam_svm1, test)!=test$spam)/length(test[,1])
spam_svm2=svm(spam~.,data=train,kernel="radial",cost=1)
errors[3]=sum(predict(spam_svm2, test)!=test$spam)/length(test[,1])
errors=as.data.frame(cbind(seq(3),errors))
colnames(errors)=c("kernels","error")
ggplot(data=errors,aes(x=kernels,y=error))+geom_line(color="Magenta")+
scale_y_continuous(limits=c(0, 0.1))
```

Choice of tuning parameters (polynomial(degree, gamma, coef0), Gaussian(gamma)). In the following problems, I have adjusted the grids to capture more important features of the curve.



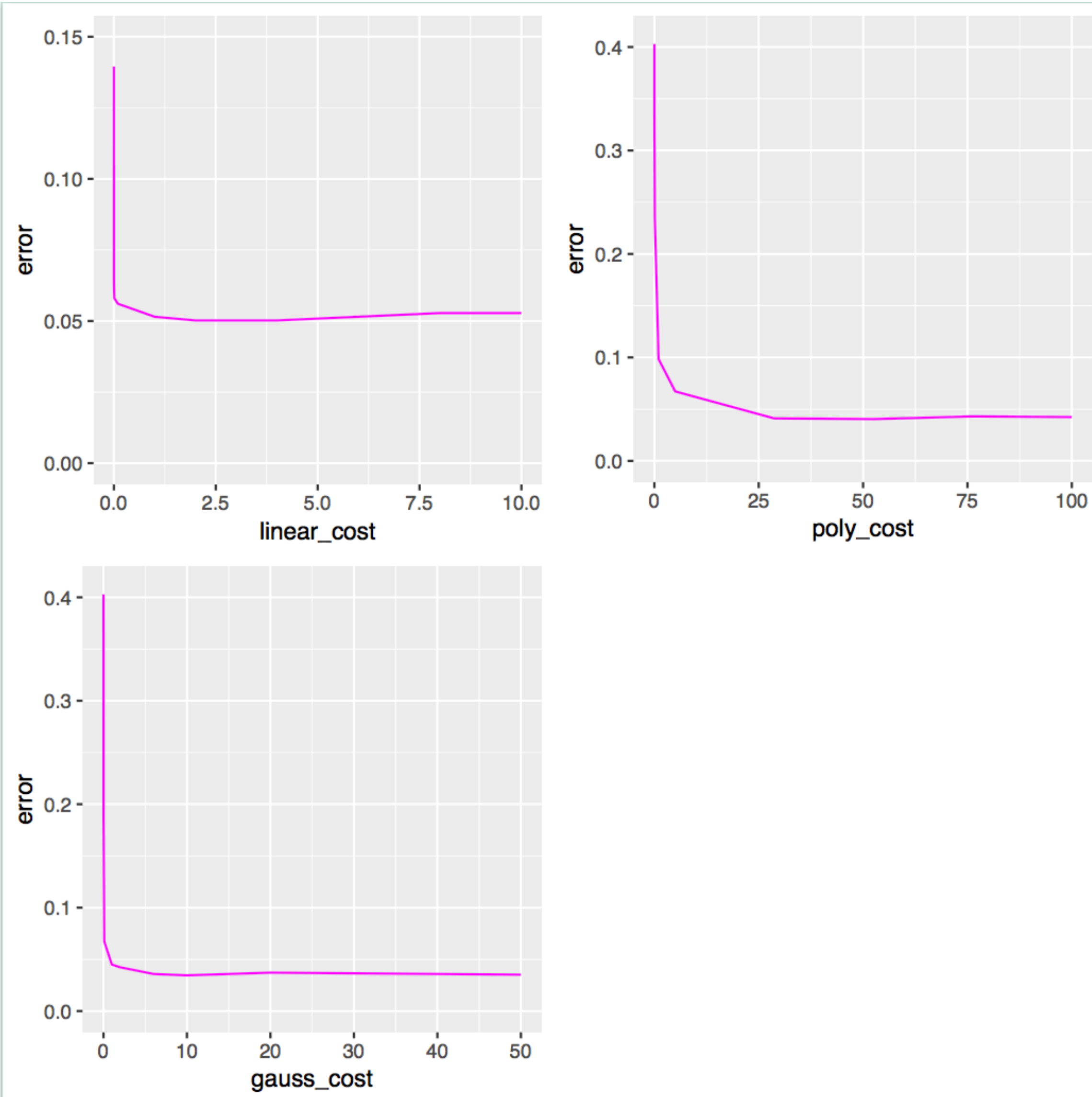
Comment: From the plot of the test error by changing tuning parameters respectively, we can see the overall trends of the sensitiveness of the model to the tuning parameters. The SVM with polynomial kernel is sensitive to degree and coef0, but insensitive to gamma. And SVM with Gaussian kernel is sensitive to gamma.

Code:

```
gam=c(seq(0.1,1,0.1),seq(2,5))

deg=seq(1,7)

coef=c(0,0.3,0.6,seq(1,10))

errors=c()

for(i in seq(length(deg))){

   spam_svm1=svm(spam~.,data=train,kernel="polynomial",cost=1,
degree=deg[i])

   errors[i]=sum(predict(spam_svm1, test)!=test$spam)/length(test[,1])

}

errors=as.data.frame(cbind(deg,errors))

colnames(errors)=c("poly_degree","error")

x=ggplot(data=errors,aes(x=poly_degree,y=error))+geom_line(color="Magenta")
+scale_y_continuous(limits=c(0, 0.3))

errors=c()

for(i in seq(length(gam))){

   spam_svm1=svm(spam~.,data=train,kernel="polynomial",cost=1,
gamma=gam[i])

   errors[i]=sum(predict(spam_svm1, test)!=test$spam)/length(test[,1])

}

errors=as.data.frame(cbind(gam,errors))

colnames(errors)=c("poly_gamma","error")
```

```
y=ggplot(data=errors,aes(x=poly_gamma,y=error))+geom_line(color="Magenta")
+scale_y_continuous(limits=c(0, 0.06))

errors=c()

for(i in seq(length(coef))){

    spam_svm1=svm(spam~.,data=train,kernel="polynomial",cost=1,
coef0=coef[i])

    errors[i]=sum(predict(spam_svm1, test)!=test$spam)/length(test[,1])

}

errors=as.data.frame(cbind(coef,errors))

colnames(errors)=c("poly_coef0","error")

z=ggplot(data=errors,aes(x=poly_coef0,y=error))+geom_line(color="Magenta")+s
cale_y_continuous(limits=c(0, 0.1))

errors=c()

for(i in seq(length(gam))){

    spam_svm2=svm(spam~.,data=train,kernel="radial",cost=1, gamma=gam[i])

    errors[i]=sum(predict(spam_svm2, test)!=test$spam)/length(test[,1])

}

errors=as.data.frame(cbind(gam,errors))

colnames(errors)=c("gauss_gamma","error")

t=ggplot(data=errors,aes(x=gauss_gamma,y=error))+geom_line(color="Magenta")
+scale_y_continuous(limits=c(0, 0.13))

grid.arrange(x,y,z,t,ncol=2)
```

Choice of the cost (Linear, polynomial, Gaussian respectively)
#For different kernels, the allowed range of cost are different. I have adjusted the grids.



Comment: we can see when the cost is small, the performance of SVM is very bad no matter for which kernel. And as the increasing of the cost, the performance would increase and reach a roughly best point, then becomes steady. Their best costs are around 2.5, 26, 10 respectively.

Code:

```
co=c(10^(-4:0),seq(2,10,2))

errors=c()

for(i in seq(length(co))){

  spam_svm=svm(spam~.,data=train,kernel="linear",cost=co[i])
```

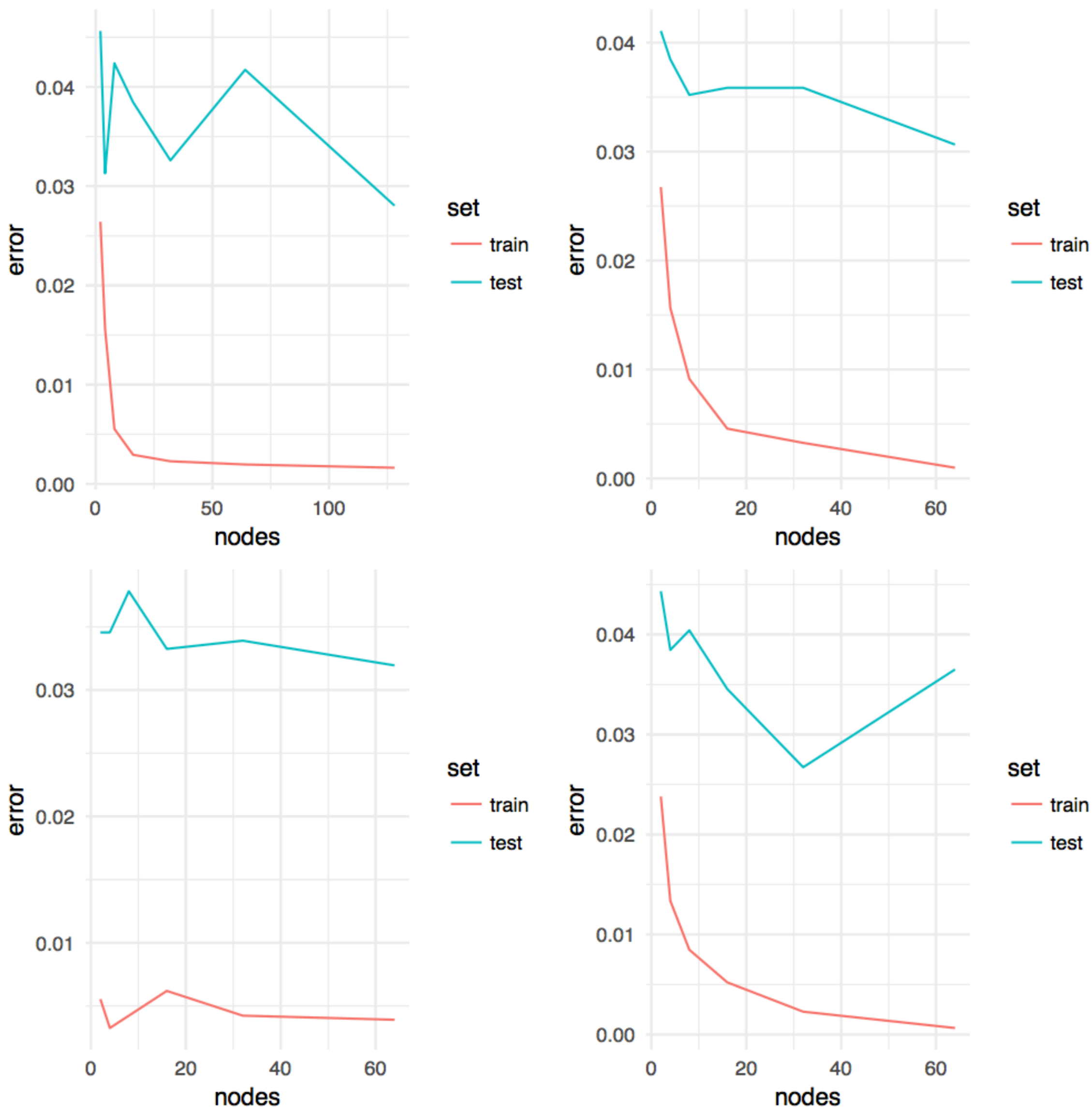
```
    errors[i]=sum(predict(spam_svm, test)!=test$spam)/length(test[,1])

}

errors=as.data.frame(cbind(co,errors))

colnames(errors)=c("linear_cost","error")

x=ggplot(data=errors,aes(x=linear_cost,y=error))+geom_line(color="Magenta")+scale_y_continuous(limits=c(0, 0.15))


co=c(10^(-4:0),seq(5,100,length.out = 5))

errors=c()

for(i in seq(length(co))){

    spam_svm1=svm(spam~.,data=train,kernel="polynomial",cost=co[i])

    errors[i]=sum(predict(spam_svm1, test)!=test$spam)/length(test[,1])

}

errors=as.data.frame(cbind(co,errors))

colnames(errors)=c("poly_cost","error")

y=ggplot(data=errors,aes(x=poly_cost,y=error))+geom_line(color="Magenta")+scale_y_continuous(limits=c(0, 0.41))


co=c(10^(-4:0),seq(2,10,4),20,50)

errors=c()

for(i in seq(length(co))){

    spam_svm=svm(spam~.,data=train,kernel="radial",cost=co[i])

    errors[i]=sum(predict(spam_svm, test)!=test$spam)/length(test[,1])

}

errors=as.data.frame(cbind(co,errors))
```

colnames(errors)=c("gauss_cost","error")

z=ggplot(data=errors,aes(x=gauss_cost,y=error))+geom_line(color="Magenta")+scale_y_continuous(limits=c(0, 0.41))

grid.arrange(x,y,z,ncol=2)

2. Neural Network comparison:



They are test and train errors by: one layer, different nodes; two layers, and two layers' nodes change synchronously; two layers, and fix the first layer as 20 nodes, change the second layer; two layers, and fix the second layer as 20 nodes, change the first layer.

Comment: we can see the comparison of one-layer and two-layer NN with different choice of nodes. And just for these two different neural network models, the best performances they can reach are similar.

Code:

```
train_nn=data.frame(train[,-58], model.matrix(~train$spam-1))

colnames(train_nn)[58:59]=c("no","yes")

pred=function(nn, dat){

   yhat = compute(nn, dat)$net.result

   yhat = apply(yhat, 1, which.max)-1

   return(yhat)

}

set.seed(123)

nodes=2^(1:7)

errors=c()

errors_t=c()

for(i in seq(length(nodes))){

   nn = neuralnet(

   paste("no+yes ~", paste(colnames(train)[-58], collapse=" + ")),

   data=train_nn,

   hidden=nodes[i],

   linear.output=F)      #To reduce the computing time, no smoothing here

   errors[i]=mean(pred(nn, train[-58]) != train$spam)

   errors_t[i]=mean(pred(nn, test[-58]) != test$spam)
```

```
}

errors=data.frame(nodes=nodes, train=errors, test=errors_t)

errors=melt(errors, id.vars = 1)

colnames(errors)=c("nodes", "set", "error")

x=ggplot(data=errors,aes(x=nodes,y=error,color=set))+geom_line()+theme_mini
mal()

###Two layers, and two layers' nodes change synchronously

set.seed(666)

nodes=2^(1:6)

errors=c()

errors_t=c()

for(i in seq(length(nodes))){

  nn = neuralnet(

  paste("no+yes ~", paste(colnames(train)[-58], collapse=" + ")),

  data=train_nn,

  hidden=c(nodes[i], nodes[i]),

  linear.output=F)

  errors[i]=mean(pred(nn, train[-58]) != train$spam)

  errors_t[i]=mean(pred(nn, test[-58]) != test$spam)

}

errors=data.frame(nodes=nodes, train=errors, test=errors_t)

errors=melt(errors, id.vars = 1)

colnames(errors)=c("nodes", "set", "error")

y=ggplot(data=errors,aes(x=nodes,y=error,color=set))+geom_line()+theme_mini
mal()
```

```
###Two layers, and fix the first layer as 20 nodes, change the second layer

set.seed(123)

nodes=2^(1:6)

errors=c()

errors_t=c()

for(i in seq(length(nodes))){

  nn = neuralnet(

  paste("no+yes ~", paste(colnames(train)[-58], collapse=" + ")),

  data=train_nn,

  hidden=c(20, nodes[i]),

  linear.output=F)

  errors[i]=mean(pred(nn, train[-58]) != train$spam)

  errors_t[i]=mean(pred(nn, test[-58]) != test$spam)

}

errors=data.frame(nodes=nodes, train=errors, test=errors_t)

errors=melt(errors, id.vars = 1)

colnames(errors)=c("nodes", "set", "error")

z=ggplot(data=errors,aes(x=nodes,y=error,color=set))+geom_line()+theme_minim
al()

###Two layers, and fix the second layer as 20 nodes, change the first layer

nodes=2^(1:6)

errors=c()

errors_t=c()

for(i in seq(length(nodes))){
```

```
    nn = neuralnet(
    paste("no+yes ~", paste(colnames(train)[-58], collapse=" + ")),
    data=train_nn,
    hidden=c(nodes[i], 20),
    linear.output=F)
    errors[i]=mean(pred(nn, train[-58]) != train$spam)
    errors_t[i]=mean(pred(nn, test[-58]) != test$spam)
}
errors=data.frame(nodes=nodes, train=errors, test=errors_t)
errors=melt(errors, id.vars = 1)
colnames(errors)=c("nodes", "set", "error")
t=ggplot(data=errors,aes(x=nodes,y=error,color=set))+geom_line()+theme_minim
al()
grid.arrange(x,y,z,t,ncol=2)
```

3. For the last problem, I chose the relatively better performance models for each method by performance curves above.
For SVM, choose the Gaussian with gamma 0.2; single layer NN with 15 nodes; double layer NN with first layer 60, second layer 20.

```
m_svm

##                     0             1
## train 0.0005408328826 0.01149425287
## test  0.0076419213974 0.20388349515

m_one

##                    0              1
## train 0.003244997296 0.005747126437
## test  0.026200873362 0.042071197411

m_two

##                    0              1
## train 0.001622498648 0.004926108374
## test  0.027292576419 0.046925566343
```

Comment: From the result of picked models for each method, the overall performance of SVM is the worst; one-layer NN is similar to two-layer. SVM has very bad performance for class 1 with the test data. The two-layer doesn't have too much overfit compared to one-layer, since I choose the model by test error. Both SVM and one-layer NN have very small train-error for class 0, so I feel they have some kind of overfit. SVM can also be regarded as one-layer NN, so sometimes they have similar performances. And based on the result, I feel there's a trade-off between the errors for different classes, and the trade-off can be realized by changing parameters.
Because of the Stochastic Gradient Descent used in finding the solution of NN, the result of the models are unstable, even for the same parameters we use.

Code:

```
spam_svm=svm(spam~.,data=train,kernel="radial", gamma=0.2)

a=table(test$spam, predict(spam_svm, test))

nn = neuralnet(

   paste("no+yes ~", paste(colnames(train)[-58], collapse=" + ")),

   data=train_nn,

   hidden=15,

   linear.output=F)

b=table(test$spam, pred(nn, test[,-58]))
```

```
set.seed(9999)

nn2 = neuralnet(

    paste("no+yes ~", paste(colnames(train)[-58], collapse=" + ")),

    data=train_nn,

    hidden=c(60, 20),

    linear.output=F)

c=table(test$spam, pred(nn2, test[,-58]))

d=table(train$spam, predict(spam_svm, train))

e=table(train$spam, pred(nn, train[,-58]))

f=table(train$spam, pred(nn2, train[,-58]))

m_svm=array(c(d[1,2]/sum(d[1,]),a[1,2]/sum(a[1,]),d[2,1]/sum(d[2,]),a[2,1]/sum(a
[2,])),c(2,2))

rownames(m_svm)=c("train","test")

colnames(m_svm)=c("0","1")

m_one=array(c(e[1,2]/sum(e[1,]),b[1,2]/sum(b[1,]),e[2,1]/sum(e[2,]),b[2,1]/sum(b
[2,])),c(2,2))

rownames(m_one)=c("train","test")

colnames(m_one)=c("0","1")

m_two=array(c(f[1,2]/sum(f[1,]),c[1,2]/sum(c[1,]),f[2,1]/sum(f[2,]),c[2,1]/sum(c[2
,])),c(2,2))

rownames(m_two)=c("train","test")

colnames(m_two)=c("0","1")
```