Old Exam

Andreas

8 January 2019

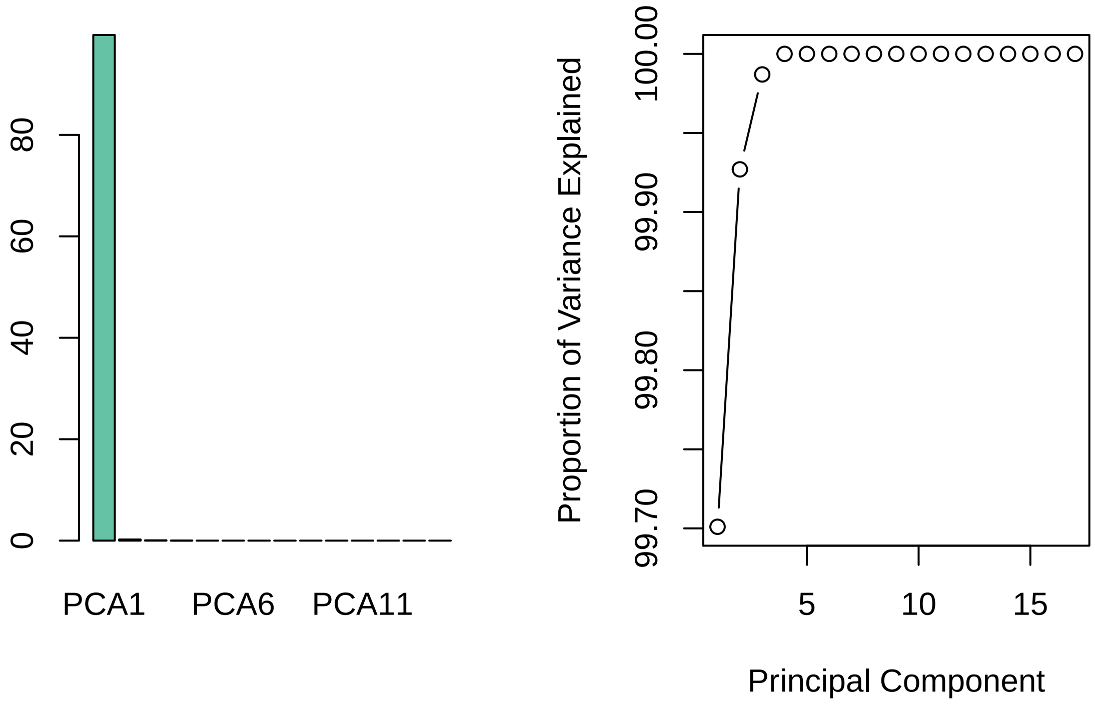
Contents

Assignment 1

1

```
## [1] "The percentage of variance for feature space "
```

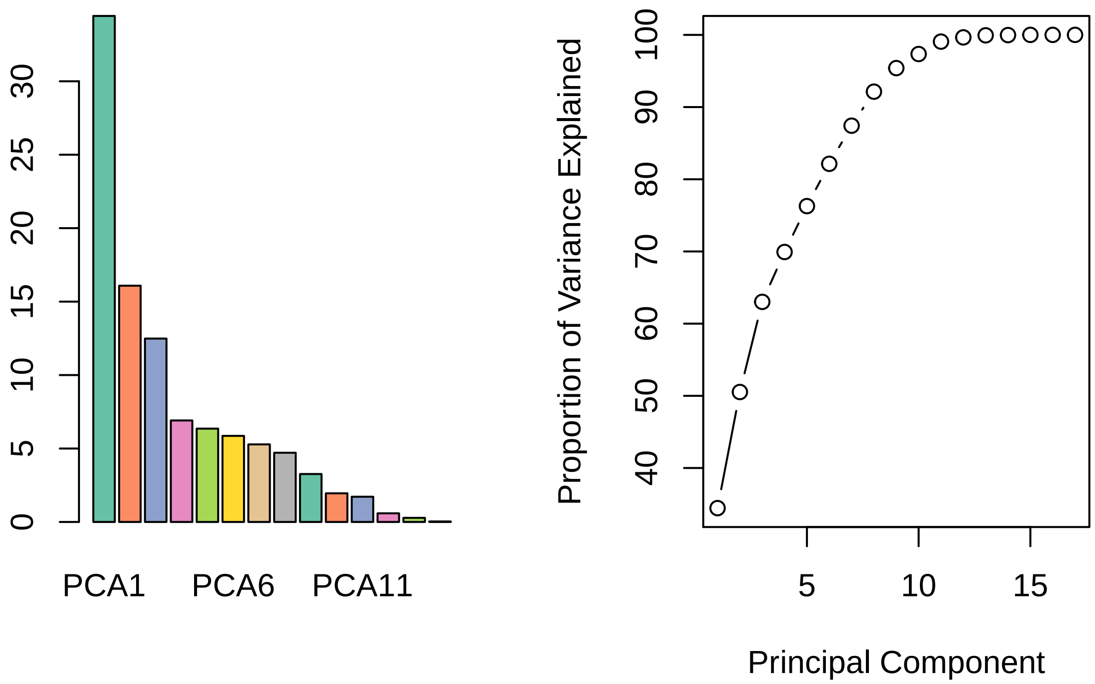
```
##  [1] 99.701  0.226  0.060  0.013  0.000  0.000  0.000  0.000  0.000  0.000
## [11]  0.000  0.000  0.000  0.000  0.000  0.000  0.000
```



We can see that only 1 component needed to explain the 95% variation in the data on the unscaled data.

```
## [1] "The percentage of variance for feature space "
```

```
##  [1] 34.446 16.083 12.486  6.910  6.355  5.860  5.281  4.713  3.264  1.950
## [11]  1.719  0.589  0.282  0.035  0.028  0.000  0.000
```



```
## [1] "The cumsum for pca in the scaled data is: "
```

```
##  [1]  34.446  50.529  63.015  69.925  76.280  82.140  87.421  92.134
##  [9]  95.398  97.348  99.067  99.656  99.938  99.973 100.001 100.001
## [17] 100.001
```

We can see that in the scaled data we need 9 components in order to have at least 95% variation in the data.

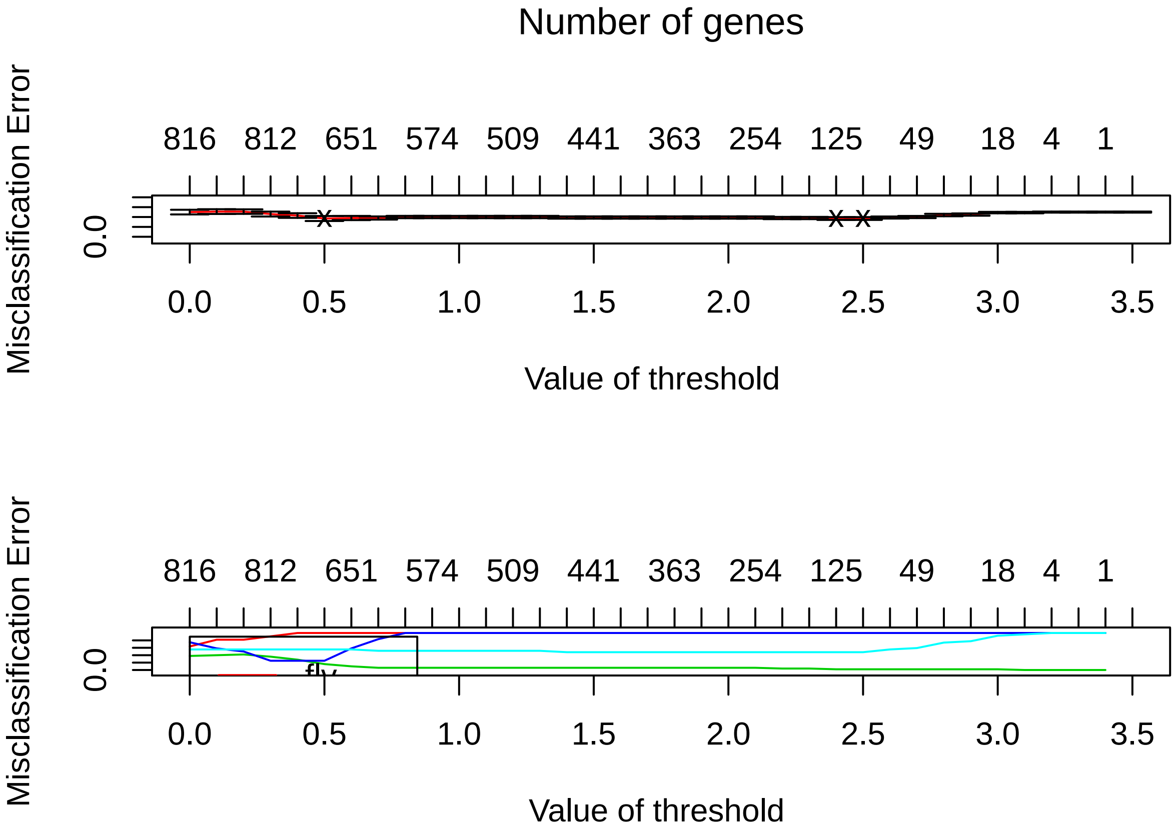
The reason in that the original data is on a very different scale -> variation in one feature dominates variation in the other features. ###2

```
## Loading required package: cluster
```

```
## Loading required package: survival
```

```
## 1234567891011121314151617181920212223242526272829303132333435363738394041
```

```
## 1234Fold 1 :123456789101112131415161718192021222324252627282930313233343536
## Fold 2 :123456789101112131415161718192021222324252627282930313233343536
## Fold 3 :123456789101112131415161718192021222324252627282930313233343536
## Fold 4 :123456789101112131415161718192021222324252627282930313233343536
## Fold 5 :123456789101112131415161718192021222324252627282930313233343536
## Fold 6 :123456789101112131415161718192021222324252627282930313233343536
## Fold 7 :123456789101112131415161718192021222324252627282930313233343536
## Fold 8 :123456789101112131415161718192021222324252627282930313233343536
## Fold 9 :123456789101112131415161718192021222324252627282930313233343536
## Fold 10 :123456789101112131415161718192021222324252627282930313233343536
```

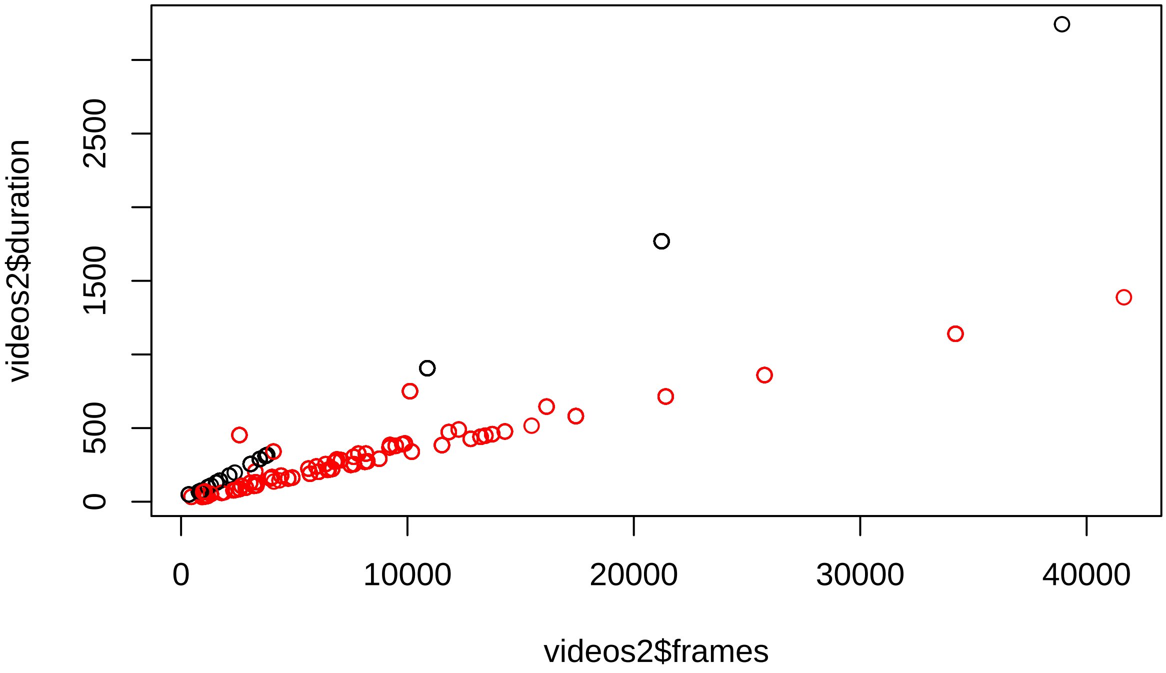


Higher threshold leads to less complex models, the higher the complexity the lower the bias and the higher is the variance.

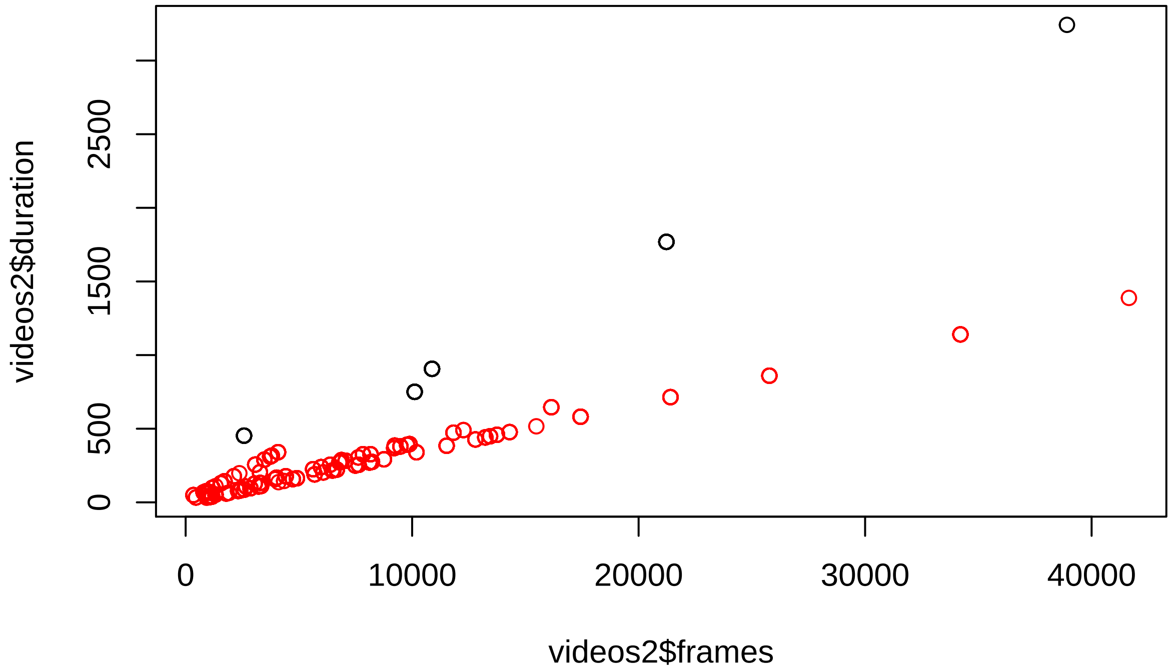
3

```
## ==============================================================
##  The theshold having the max value of log-likelihood is : 2.5
```

Multinomial likelihood is used because this is a classification problem.



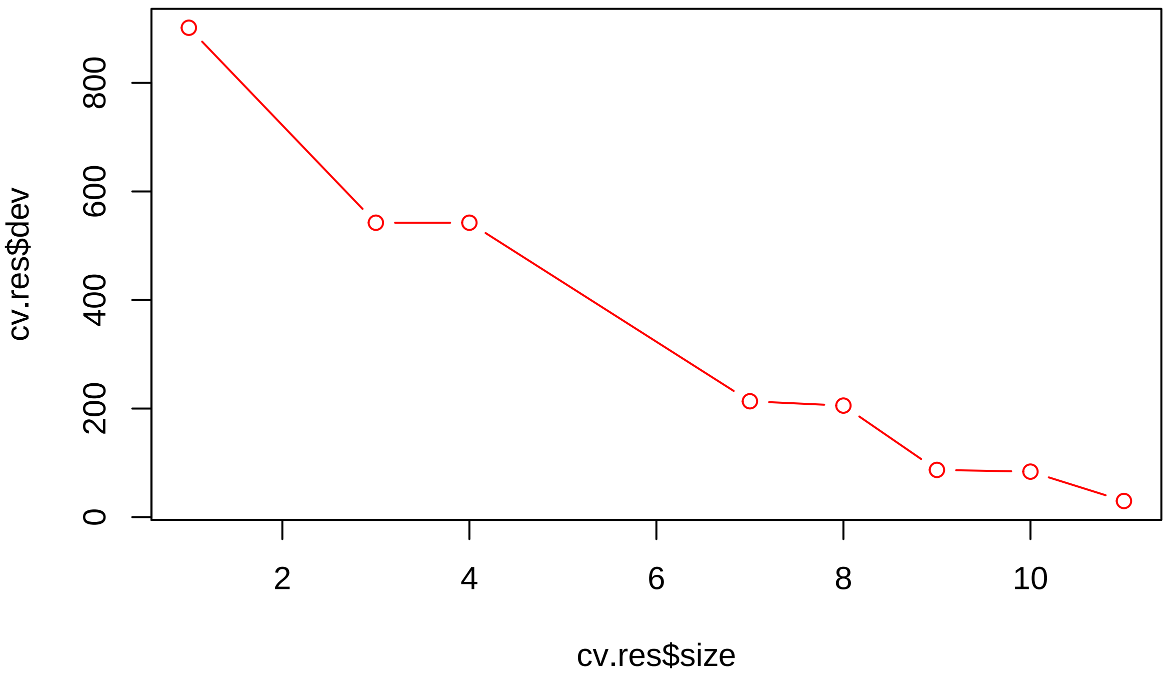
Classes seen to be rather clearly linearly separable (with exception of a few cases near to the origin ###5



```
## The misclassification error for train is: 0.172
```

The result of classification is rather bad. It is clear that covariance matrices per class are very different. In addition, class-conditional distributions do not look like multivariate normal.

6

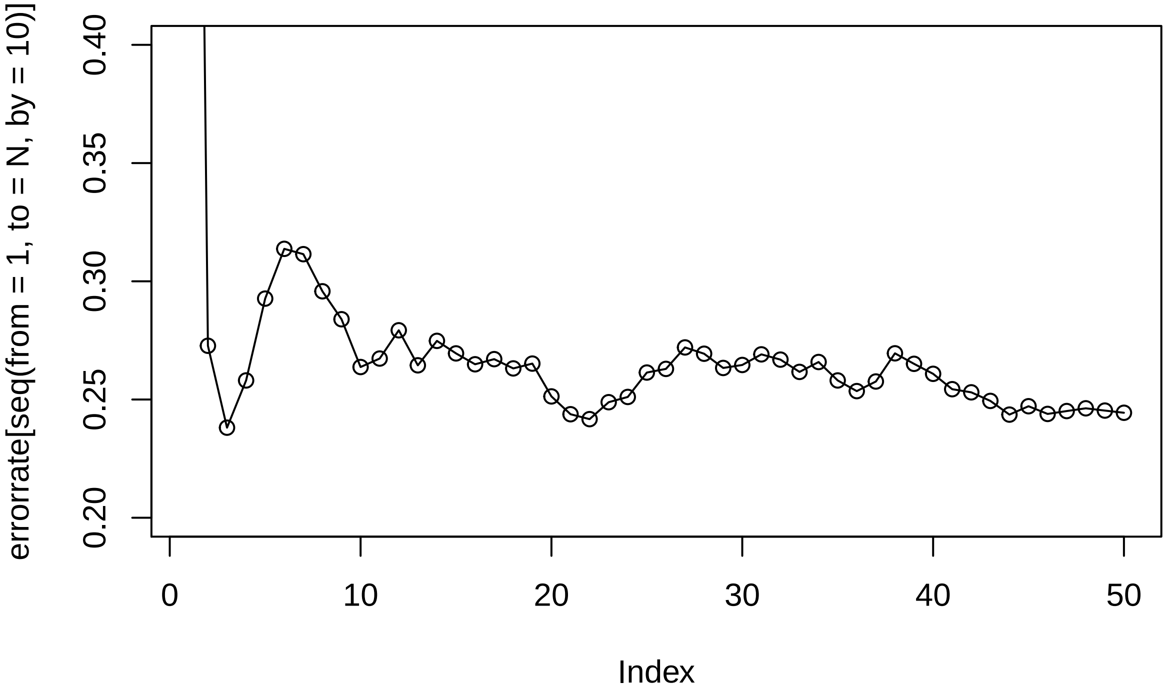


```
## =====================================================
##  The size of the best tree is : 11
##  The misclassification error is : 0.001
```

According to the cross-validation plot, the optimal tree is the largest one among the ones that were grown with default settings. The optimal tree among these has 11 leaves. Such a complicated tree is needed because the optimal decision boundary is linear but not parallel to any of the coordinate axes. Accordingly, decision tree would need to make many splits and produce a stair-kind of decision boundary that would approximate this linear decision boundary.

Assignment 2

```
## ...............................................................................................
```



```
## [1] 50
```

```
## [1] 0
```

```
## [1] 50
```

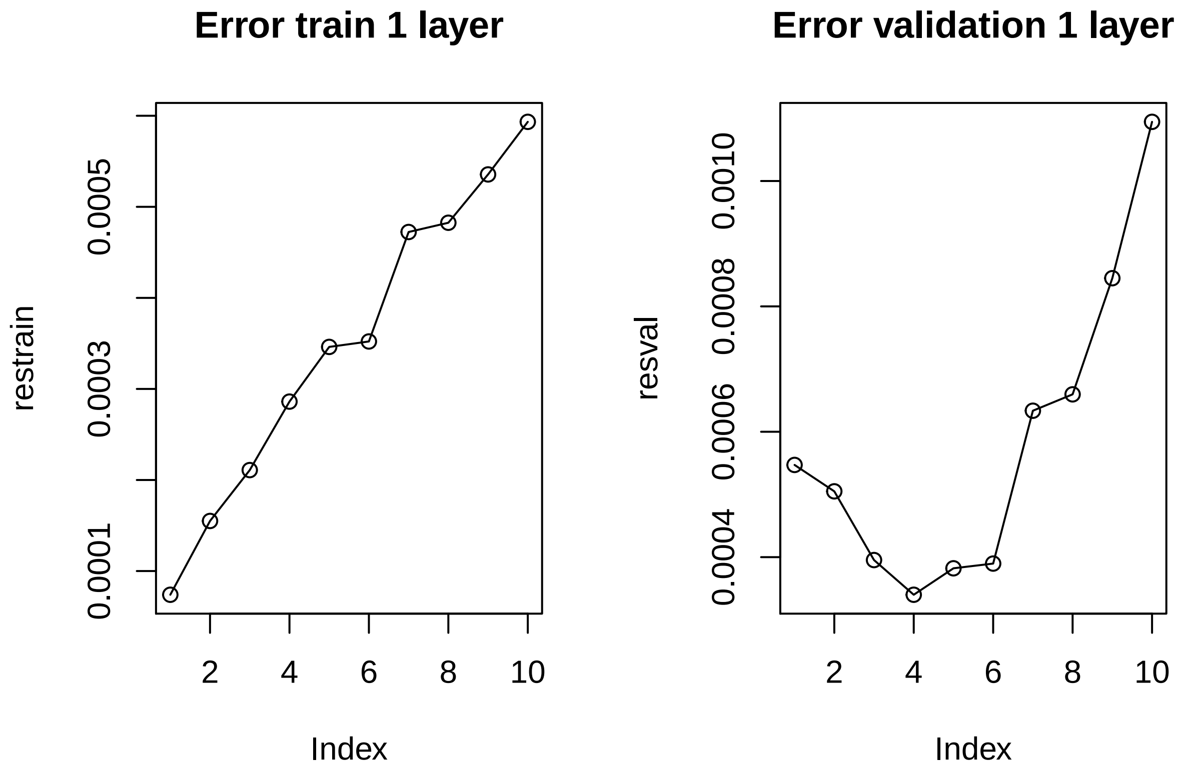
```
## [1] 0.242
```

Assignmet 3

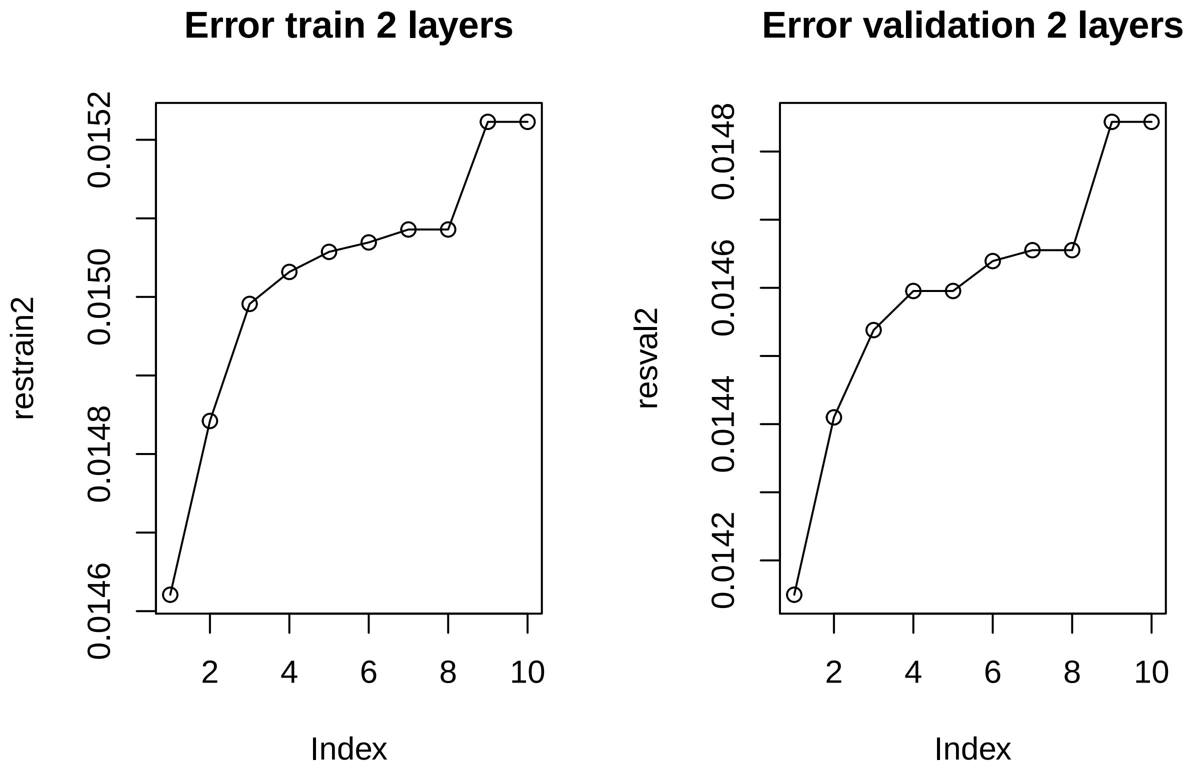
First we try the NN with one layer

```
## Loading required package: neuralnet
```

```
## Warning: package 'neuralnet' was built under R version 3.5.2
```



Next we move to training the NN with 2 layers



We can see that the model with one layer and threshold=4/1000 provides the best architecture

```
## [1] "The generalization squared error is:"
```

```
## [1] 0.007652619204
```

Appendix

```
#load data
videos<-read.csv("video.csv",sep=",",header=T)

#dvivde data to 50/50
n=dim(videos)[1]
set.seed(12345)
id<-sample(1:n,floor(n*0.5))
train<-videos[id,]
test<-videos[-id,]

#numeric<-names(dplyr::select_if(train,is.numeric))
#train_num<-train[,numeric]
#names(dplyr::select_if(train,is.numeric))

#select only numeric and integer columns
train_num<-train[, sapply(train, class) %in% c("integer","numeric") ]
#discurd the utime column from data
train_pca<-train_num[,!names(train_num)%in%c("utime")]
```

```
#we perform pca for unscaled data
pca_unscaled<-prcomp(train_pca)
pca.var<-pca_unscaled$sdev^2
# calculate percentage of variance of PCAs
pca.var.per<-round(pca.var/sum(pca.var)*100,3)

print("The percentage of variance for feature space ")
pca.var.per

library(RColorBrewer)
par(mfrow=c(1,2))

barplot(pca.var.per[1:14],names.arg=c("PCA1","PCA2","PCA3","PCA4","PCA5","PCA6",
                                      "PCA7","PCA8","PCA9","PCA10","PCA11","PCA12","PCA13","PCA14"),
        col=brewer.pal(8, "Set2"))

plot(cumsum(pca.var.per), xlab = "Principal Component",
     ylab = "Proportion of Variance Explained",
     type = "b")


#we perform pca for scaled data
pca_scaled<-prcomp(train_pca,scale=T)
pca.var1<-pca_scaled$sdev^2
# calculate percentage of variance of PCAs
pca.var.per1<-round(pca.var1/sum(pca.var1)*100,3)

print("The percentage of variance for feature space ")
pca.var.per1

library(RColorBrewer)
par(mfrow=c(1,2))

barplot(pca.var.per1[1:14],names.arg=c("PCA1","PCA2","PCA3","PCA4","PCA5","PCA6",
                                       "PCA7","PCA8","PCA9","PCA10","PCA11","PCA12","PCA13","PCA14"),
        col=brewer.pal(8, "Set2"))

plot(cumsum(pca.var.per1), xlab = "Principal Component",
     ylab = "Proportion of Variance Explained",
     type = "b")




#print cumsum in the scaled data
print("The cumsum for pca in the scaled data is: ")
cumsum(pca.var.per1)
#make dataset with only the numeric and integer columns
#with the 100 first rows
data_num<-videos[1:100, sapply(videos, class) %in% c("integer","numeric") ]
#create interaction variables up to the third order
data=t(apply(as.matrix(data_num), 1, combn, 3, prod))
#convert to data frame and scale them
```

```r
df<-as.data.frame(scale(data))

library(pamr)
x <- t(df)
#make the codec as factor
y <- as.factor(videos$codec[1:100])
#make a list to insert to pamr.train function
mydata<-list(x=x,y=y,geneid=as.character(1:nrow(x)), genenames=rownames(x))
cen_fit<-pamr.train(mydata,threshold=seq(0,4, 0.1))
set.seed(12345)
cen_cv<-pamr.cv(cen_fit,mydata)
#plot the cv model
pamr.plotcv(cen_cv)


#calculate threshold having max loglikelohood
thres_max<-cen_cv$threshold[which(cen_cv$loglik==max(cen_cv$loglik))]
cat("==============================================================\n",
    "The theshold having the max value of log-likelihood is :",thres_max )

videos2=videos
#make new column class with condition given
videos2$class=ifelse(videos2$codec=="mpeg4","mpeg4","other")
#make plot
plot(videos2$frames,videos2$duration,col=as.factor(videos2$class))


library(MASS)
#lda model
lda.fit<-lda(as.factor(class)~frames+duration,data=videos2)
#predictions with lda
preds_lda<-predict(lda.fit,videos2,type="class")
#plot with the predicitons
plot(videos2$frames,videos2$duration, col=preds_lda$class)


# missclass=function(X,X1){
#   n=length(X)
#   return(1-sum(diag(table(X,X1)))/n)
# }

cat("The misclassification error for train is:",mean(videos2$class!=preds_lda$class))


library(tree)
#fit tree model
tree_fit<-tree(as.factor(class)~frames+duration,data=videos2)
#perform cv to find the optimal tree size
set.seed(12345)
cv.res=cv.tree(tree_fit)
#plot the size vs deviance
plot(cv.res$size, cv.res$dev, type="b",
     col="red")
```

```r
#find which is the best tree size
best_size<-cv.res$size[which.min(cv.res$dev)]
#prune tree to this size
tree_best<-prune.tree(tree_fit,best=11)
#make predictions on data with tree model
preds_tree<-predict(tree_best,videos2,type="class")
#calculate misclassification error

mis_error_tree<-mean(videos2$class!=predict(tree_fit,type="class"))
#mean(videos2$class!=preds_tree) #alternative code
cat("=======================================================\n",
    "The size of the best tree is :",best_size,"\n",
    "The misclassification error is :",mis_error_tree)

set.seed(1234567890)
spam <- read.csv2("spambase.csv")
ind <- sample(1:nrow(spam))
spam <- spam[ind,c(1:48,58)]
h <- 1
beta <- 0
M <- 50
N <- 500 # number of training points

gaussian_k <- function(x, h) { # It is fine if students use exp(-x**2)/h instead
  return (exp(-(x**2)/(2*h*h)))
}

SVM <- function(sv,i) { #SVM on point i with support vectors sv
  yi <- 0
  for(m in 1:length(sv)) {
    xixm <- rbind(spam[i,-49],spam[sv[m],-49]) # do not use the true label when computing the distance
    tm <- 2 * spam[sv[m],49] - 1 # because the true labels must be -1/+1 and spambase has 0/1
    yi <- yi + tm * gaussian_k(dist(xixm, method="euclidean"), h)
  }
  return (yi)
}

errors <- 1
errorrate <- vector(length = N)
errorrate[1] <- 1
sv <- c(1)
for(i in 2:N) {
  yi <- SVM(sv,i)
  ti <- 2 * spam[i,49] - 1

  if(ti * yi < 0) {
    errors <- errors + 1
  }
  errorrate[i] <- errors/i

  cat(".") # iteration ", i, "error rate ", errorrate[i], ti * yi, "sv ", length(sv), "\n")
  flush.console()
  if(ti * yi <= beta) {
    sv <- c(sv, i)

    if (length(sv) > M) {
      for(m in 1:length(sv)) { # remove the support vector that gets classified best without itself
        sv2 <- sv[-m]
        ym2 <- SVM(sv2,sv[m])
        tm <- 2 * spam[sv[m],49] - 1

        if(m==1) {
          max <- tm * ym2
          ind <- 1
        }
        else {
          if(tm * ym2 > max) {
            max <- tm * ym2
            ind <- m
          }
        }
      }
      sv <- sv[-ind]

      # cat("removing ", ind, max, "\n")
      # flush.console()
    }
  }
}
plot(errorrate[seq(from=1, to=N, by=10)], ylim=c(0.2,0.4), type="o")
M
beta
length(sv)
errorrate[N]


require(neuralnet)

set.seed(1234567890)

##one layer


points<-runif(50,0,10)
dat<-data.frame(points,sin(points))
train_dat<-dat[1:25,]
val_dat<-dat[26:50,]

restrain <- double(10)
resval <- double(10)
weights<-runif(41, -1, 1)


for (i in 1:10){
  nn<-neuralnet(sin.points.~points,data=train_dat,
```

```r
                hidden=c(10),startweights = weights,
                threshold = i/1000,lifesign="none")

  res1<-compute(nn,train_dat[,1])$net.result
  restrain[i]<-mean((train_dat[,2]-res1)^2)

  res2<-compute(nn,val_dat[,1])$net.result
  resval[i]<-mean((val_dat[,2]-res2)^2)

}

par(mfrow=c(1,2))
plot(restrain, type = "o",main="Error train 1 layer")
plot(resval, type = "o",main="Error validation 1 layer")

#restrain
#resval



##2 layers

set.seed(1234567890)
points<-runif(50,0,10)
dat<-data.frame(points,sin(points))
train_dat<-dat[1:25,]
val_dat<-dat[26:50,]

restrain2 <- double(10)
resval2 <- double(10)
# Random initializaiton of the weights in the interval [-1, 1]
winit1 <- runif(22, -1, 1)
for(i in 1:10) {
  nn <- neuralnet(formula = sin.points. ~ points, data = train_dat,
                  hidden = c(3,3), startweights = winit1,
                  threshold = i/1000, lifesign = "none")

  # nn$result.matrix
  # Compute predictions for the trainig set and their mean squared error
  res3 <- compute(nn, train_dat[,1])$net.result
  restrain2[i] <- mean((train_dat$sin.points.-res3)^2)
  # The same for the validation set
  res4 <- compute(nn, val_dat[,1])$net.result
  resval2[i] <-mean ((val_dat$sin.points.-res4)^2)
}

par(mfrow=c(1,2))
plot(restrain2, type = "o",main="Error train 2 layers")
plot(resval2, type = "o",main="Error validation 2 layers")
#restrain2
#resval2
```

```r
#generate new data and weights
p <- runif(50, 0, 10)
d <- data.frame(p, Sin=sin(p))

weights_init <- runif(31, -1, 1)
#train nn with the best model and threshold
nn <- neuralnet(formula = sin.points. ~ points, data = dat,
                hidden = 10, startweights = weights_init,
                threshold = 4/1000, lifesign = "none")
#calculate predictions
y_nn<-compute(nn,d[,1])$net.result

print("The generalization squared error is:")
sum((d[,2] - y_nn)^2)/2 #squared error
#mean((d[,2]-y_nn)^2) #mean squared error
```