previous code

Prudhvi Peddmallu

23 April 2019

neuralnet-package

```
library(neuralnet)

set.seed(1234567890)
Var <- runif(50, 0, 3)
tr <- data.frame(Var, Sin=sin(Var))
Var <- runif(50, 3, 9)
te <- data.frame(Var, Sin=sin(Var))
```

```
#setwd("your directory")
mydata <- read.csv("dividendinfo-1.csv")
#attach(mydata)
##Data Normalization-This involves adjusting the data to a common scale so as to accurately compare pre
scaleddata<-scale(mydata)
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}
maxmindf <- as.data.frame(lapply(mydata, normalize))
#We base our training data (trainset) on 80% of the observations. The test data (testset) is based on t
# Training and Test Data we have 459 observations in data so 80% is 367 are train and remaining are tes
trainset <- maxmindf[1:160, ]
testset <- maxmindf[161:200, ]
#Neural Network
#Neural Network
library(neuralnet)
nn <- neuralnet(dividend ~ fcfps + earnings_growth + de + mcap + current_ratio, data=trainset, hidden=c
nn$result.matrix
plot(nn)
#Testing The Accuracy Of The Model
#Test the resulting output
temp_test <- subset(testset, select = c("fcfps","earnings_growth", "de", "mcap", "current_ratio"))
head(temp_test)
nn.results <- compute(nn, temp_test)
results <- data.frame(actual = testset$dividend, prediction = nn.results$net.result)
```

MMs

```
# JMP

install.packages("mvtnorm")
library(mvtnorm)

set.seed(1234567890)
```

```
max_it <- 100 # max number of EM iterations
min_change <- 0.1 # min change in log likelihood between two consecutive EM iterations
N=300 # number of training points
D=2 # number of dimensions
x <- matrix(nrow=N, ncol=D) # training data

# Producing the training data
mu1<-c(0,0)
Sigma1 <- matrix(c(5,3,3,5),D,D)
dat1<-rmvnorm(n = 100, mu1, Sigma1)
mu2<-c(5,7)
Sigma2 <- matrix(c(5,-3,-3,5),D,D)
dat2<-rmvnorm(n = 100, mu2, Sigma2)
mu3<-c(8,3)
Sigma3 <- matrix(c(3,2,2,3),D,D)
dat3<-rmvnorm(n = 100, mu3, Sigma3)
plot(dat1,xlim=c(-10,15),ylim=c(-10,15))
points(dat2,col="red")
points(dat3,col="blue")
x[1:100,]<-dat1
x[101:200,]<-dat2
x[201:300,]<-dat3
plot(x,xlim=c(-10,15),ylim=c(-10,15))

K=3 # number of guessed components
z <- matrix(nrow=N, ncol=K) # fractional component assignments
pi <- vector(length = K) # mixing coefficients
mu <- matrix(nrow=K, ncol=D) # conditional means
Sigma <- array(dim=c(D,D,K)) # conditional covariances
llik <- vector(length = max_it) # log likelihood of the EM iterations

# Random initialization of the paramters
pi <- runif(K,0,1)
pi <- pi / sum(pi)
for(k in 1:K) {
  mu[k,] <- runif(D,0,5)
  Sigma[,,k]<-c(1,0,0,1)
}
pi
mu
Sigma

for(it in 1:max_it) {
  # E-step: Computation of the fractional component assignments
  llik[it] <- 0
  for(n in 1:N) {
    for(k in 1:K) {
      z[n,k] <- pi[k]*dmvnorm(x[n,],mu[k,],Sigma[,,k])
    }

    #Log likelihood computation.
    llik[it] <- llik[it] + log(sum(z[n,]))
```

```
    z[n,] <- z[n,]/sum(z[n,])
  }

  cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  flush.console()
  # Stop if the lok likelihood has not changed significantly
  if (it > 1) {
    if(abs(llik[it] - llik[it-1]) < min_change) {
      break
    }
  }

  #M-step: ML parameter estimation from the data and fractional component assignments
  for(k in 1:K) {
    pi[k] <- sum(z[,k]) / N
    for(d in 1:D) {
        mu[k, d] <- sum(x[, d] * z[, k]) / sum(z[,k])
    }
    for(d in 1:D) {
      for(d2 in 1:D) {
        Sigma[d,d2,k]<-sum((x[, d]-mu[k,d]) * (x[, d2]-mu[k,d2]) * z[, k]) / sum(z[,k])
      }
    }
  }
}
pi
mu
Sigma
```

NNs

```
# JMP

library(neuralnet)
set.seed(1234567890)

Var <- runif(50, 0, 3)
tr <- data.frame(Var, Sin=sin(Var))
Var <- runif(50, 3, 9)
te <- data.frame(Var, Sin=sin(Var))
winit <- runif(10, -1, 1)
nn <- neuralnet(formula = Sin ~ Var, data = tr, hidden = 3, startweights = winit, lifesign = "full")
plot(tr,xlim=c(0,9),ylim=c(-2,2))
points(te,col="blue")
points(te$Var,compute(nn,te$Var)$net.result,col="red")
plot(nn)

# The predictions converge to -2 because the sigmoid functions saturate for large Var values.
```

svm

```
# JMP

library(kernlab)
set.seed(1234567890)

data(spam)

index <- sample(1:4601)
tr <- spam[index[1:2500], ]
va <- spam[index[2501:3500], ]
te <- spam[index[3501:4601], ]

# My first guess was that you may get an error ("No sums found") when C=0 because your data may not be
# separable. However, one gets the error even when the data is linearly separable, e.g.

ksvm(type~.,data=tr[1:2,57:58],kernel="rbfdot",kpar=list(sigma=0.05),C=0)

# It is true that theory requires C>0 but, then, I would have expected an information message telling m
# a positive C, not an error. Bottom line: I am not sure why you get an error but I do not think it is
# C must be positive. Issuing an error would be a weird way to instruct the user to raise the value of
# expected that C=0 instruct the system to try to find a linearly separable SVM.

# Model selection.

er<-NULL
myStep<-0.1
for(myC in seq(0.1,10,myStep)){
  filter <- ksvm(type~.,data=tr,kernel="rbfdot",kpar=list(sigma=0.05),C=myC)
  mailtype <- predict(filter,va[,-58])
  t <- table(mailtype,va[,58])
  er<-c(er,(t[1,2]+t[2,1])/sum(t))
}
plot(er)

# Final model.

min(er)
which.min(er)
filter<-ksvm(type~.,data=spam[index[1:3500],],kernel="rbfdot",kpar=list(sigma=0.05),C=which.min(er)*myS
mailtype <- predict(filter,te[,-58])
t <- table(mailtype,te[,58])
(t[1,2]+t[2,1])/sum(t)
filter<-ksvm(type~.,data=spam,kernel="rbfdot",kpar=list(sigma=0.05),C=which.min(er)*myStep)

# That the validation and test error are similar suggests no overfitting.

# Pseudocode.

rbfkernel <- rbfdot(sigma = 0.05)
sv<-alphaindex(filter)[[1]]
co<-coef(filter)[[1]]
```

```
k<-NULL
for(i in 1:1000){
  k2<-NULL
  for(j in 1:length(sv)){
    k2<-c(k2,co[j]*rbfkernel(unlist(tr[sv[j],-58]),unlist(va[i,-58])))
    }
  k<-c(k,sum(k2)-b(filter))
}
k
```

loss matrix 0 2 1

```
       6 0 100
       6 2 0
```

```
library("e1071")
library("dplyr")
options(scipen=999)

data <- iris

set.seed(12345)
model = naiveBayes(Species ~., data=data)

prob <- predict(model, newdata=data, type = c("raw"))
prob <- prob %>%  as.data.frame()
colnames(prob) <- c("prob_setosa", "prob_versicolor", "prob_virginica")

data2 <- cbind(data, prob)

loss_matrix <- matrix(data = c(0,2,1,6,0,100,6,2,0), byrow = TRUE, ncol=3, nrow=3)
loss_matrix

## logic consider the first row, predicted class = 1, prob_class_1 > 2 * prob_of_class_2 & prob_class_1


data2$prediced_class <- ifelse(data2$prob_setosa > 2 * data2$prob_versicolor & data2$prob_setosa > 1 *
                               ifelse(data2$prob_versicolor > 6 * data2$prob_setosa & data2$prob_versic
                                      ifelse(data2$prob_virginica > 2 * data2$prob_versicolor & data2$p

## confirmation of results
table(data2$Species, data2$prediced_class)
```

ordinary least squares regression-predection

```
mydata=read.csv2("Bilexempel.csv")
fit1=lm(Price~Year, data=mydata)
summary(fit1)
fit2=lm(Price~Year+Mileage+Equipment,
data=mydata)
```

```
summary(fit2)
#Prediction
fitted <- predict(fit1, interval =
"confidence")
# plot the data and the fitted line
attach(mydata)
plot(Year, Price)
lines(Year, fitted[, "fit"])
# plot the confidence bands
lines(Year, fitted[, "lwr"], lty = "dotted",
col="blue")
lines(Year, fitted[, "upr"], lty = "dotted",
col="blue")
detach(mydata)
```

Ridge regression-crossvalidation

```
#usepackageglmnetwithalpha=0 (Ridge regression)
data=read.csv("machine.csv", header=F)
covariates=scale(data[,3:8])
response=scale(data[, 9])
model0=glmnet(as.matrix(covariates),
response, alpha=0,family="gaussian")
plot(model0, xvar="lambda", label=TRUE)
#Choosingthe best modelby cross-validation
model=cv.glmnet(as.matrix(covariates),
response, alpha=0,family="gaussian")
model$lambda.min
plot(model)
coef(model, s="lambda.min")
#Howgoodis thismodelin prediction?
ind=sample(209, floor(209*0.5))
data1=scale(data[,3:9])
train=data1[ind,]
test=data1[-ind,]
covariates=train[,1:6]
response=train[, 7]
model=cv.glmnet(as.matrix(covariates), response, alpha=1,family="gaussian",
lambda=seq(0,1,0.001))
y=test[,7]
ynew=predict(model, newx=as.matrix(test[, 1:6]), type="response")
#Coefficient of determination
sum((ynew-mean(y))^2)/sum((y-mean(y))^2)
sum((ynew-y)^2)
```

Variableselection-stepAIC

```
library(MASS)
fit <- lm(V9~.,data=data.frame(data1))
```

```
step <- stepAIC(fit, direction="both")
step$anova
summary(step)
```

Manipulate for mixture models

```
install.packages("manipulate")
library(manipulate)

data(faithful)
hist(faithful$waiting,freq = FALSE)

NormalPlot <- function(mu1,sigma1,mu2,sigma2,pi1){
  xGrid <- seq(40, 100, by=0.001)
  pdf = dnorm(xGrid, mean=mu1, sd=sigma1) * pi1 + (1-pi1) * dnorm(xGrid, mean=mu2, sd=sigma2)
  hist(faithful$waiting,freq = FALSE)
  lines(xGrid, pdf, type = 'l', lwd = 3, col = "blue")
}

manipulate(
  NormalPlot(mu1,sigma1,mu2,sigma2,pi1),
  mu1 = slider(40, 100, step=.1, initial = 70, label = "mu1"),
  sigma1 = slider(0, 10, step=.1, initial = 1, label = "sigma1"),
  mu2 = slider(40, 100, step=.1, initial = 70, label = "mu2"),
  sigma2 = slider(0, 10, step=.1, initial = 1, label = "sigma2"),
  pi1 = slider(0, 1, step=.01, initial = .5, label = "pi1")
)
```

Holdout-diveding the data

```
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.7))
train=data[id,]
test=data[-id,]
#Howtopartition intotrain/valid/test?
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.4))
train=data[id,]
id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.3))
valid=data[id2,]
id3=setdiff(id1,id2)
test=data[id3,]
```

Cross-validatation-Try modelswithdifferent predictorsets

```
data=read.csv("machine.csv", header=F)
library(cvTools)
fit1=lm(V9~V3+V4+V5+V6+V7+V8, data=data)
fit2=lm(V9~V3+V4+V5+V6+V7, data=data)
fit3=lm(V9~V3+V4+V5+V6, data=data)
f1=cvFit(fit1, y=data$V9, data=data,K=10,
foldType="consecutive")
f2=cvFit(fit2, y=data$V9, data=data,K=10,
foldType="consecutive")
f3=cvFit(fit3, y=data$V9, data=data,K=10,
foldType="consecutive")
res=cvSelect(f1,f2,f3)
plot(res)
```

LDA

```
resLDA=lda(Equipment~Mileage+Year, data=mydata)
print(resLDA)
#plot
plot(mydata$Year, mydata$Mileage,
col=as.numeric(Pred$class)+1, pch=21,
bg=as.numeric(Pred$class)+1,
main="Prediction")
#Misclassified items
```

Splines

.Smoothingsplines: smooth.spline() .Naturalcubicsplines: ns() in splines .Thinplatesplines: Tps() in fields

```
res1=smooth.spline(data$Time,data$RSS_anchor2,df=10)
predict(res1,x=data$Time)$y
```

Generalizedadditivemodels

```
library(mgcv)
library(akima)
library(plotly)
river=read.csv2("Rhine.csv")
res=gam(TotN_conc~Year+Month+s(Year,
k=length(unique(river$Year)))+
s(Month, k=length(unique(river$Month))),
data=river)
s=interp(river$Year,river$Month, fitted(res))
print(res)
summary(res)
```

```
res$sp
plot(res)
plot_ly(x=~s$x, y=~s$y, z=~s$z, type="surface")
```

Naive Bayes

.naiveBayesin packagee1071

```
library(MASS)
library(e1071)
n=dim(housing)[1]
ind=rep(1:n, housing[,5])
housing1=housing[ind,-5]
fit=naiveBayes(Sat~., data=housing1)
fit
Yfit=predict(fit, newdata=housing1)
table(Yfit,housing1$Sat)
```

Decision trees

```
#treepackage
#Alternative: rpart
#tree(formula, data, weights, control, split = c("deviance", "gini"), .)
#print(), summary(), plot(), text()
library(tree)
n=dim(biopsy)[1]
fit=tree(class~., data=biopsy)
plot(fit)
text(fit, pretty=0)
fit
summary(fit)
#.Misclassificationresults-tree
Yfit=predict(fit, newdata=biopsy, type="class")
table(biopsy$class,Yfit)
#.Selectingoptimal treeby penalizing
#Cv.tree()
set.seed(12345)
ind=sample(1:n, floor(0.5*n))
train=biopsy[ind,]
valid=biopsy[-ind,]
fit=tree(class~., data=train)
set.seed(12345)
cv.res=cv.tree(fit)
plot(cv.res$size, cv.res$dev, type="b",
col="red")
plot(log(cv.res$k), cv.res$dev,
type="b", col="red")
#.Selectingoptimal treeby train/validation
fit=tree(class~., data=train)
trainScore=rep(0,9)
```

```
testScore=rep(0,9)
for(i in 2:9) {
prunedTree=prune.tree(fit,best=i)
pred=predict(prunedTree, newdata=valid,
type="tree")
trainScore[i]=deviance(prunedTree)
testScore[i]=deviance(pred)
}
plot(2:9, trainScore[2:9], type="b", col="red",
ylim=c(0,250))
points(2:9, testScore[2:9], type="b", col="blue")
#.Final tree: 5 leaves
finalTree=prune.tree(fit, best=5)
Yfit=predict(finalTree, newdata=valid,
type="class")
table(valid$class,Yfit)
```

Nearest ShrunkenCentroids-NSC

```
#Packagepamr
#pamr.train()
#pamr.cv
data0=read.csv2("voice.csv")
data=data0
data=as.data.frame(scale(data))
data$Quality=as.factor(data0$Quality)
library(pamr)
rownames(data)=1:nrow(data)
x=t(data[,-311])
y=data[[311]]
mydata=list(x=x,y=as.factor(y),geneid=as.character(1:nrow(x)), genenames=rownames(x))
model=pamr.train(mydata,threshold=seq(0,4, 0.1))
pamr.plotcen(model, mydata, threshold=1)
pamr.plotcen(model, mydata, threshold=2.5)
a=pamr.listgenes(model,mydata,threshold=2.5)
cat( paste( colnames(data)[as.numeric(a[,1])], collapse='\n' ) )
cvmodel=pamr.cv(model,mydata)
print(cvmodel)
pamr.plotcv(cvmodel)
```

Kernel PCA

```
library(kernlab)
K <- as.kernelMatrix(crossprod(t(x)))
res=kpca(K)
barplot(res@eig)
plot(res@rotated[,1], res@rotated[,2], xlab="PC1",
ylab="PC2")
```

Feature assessment

```
res=t.test(MFCC_2nd.coef~Quality,data=data,
alternative="two.sided")
res$p.value
res=oneway_test(MFCC_2nd.coef~as.factor(Qu
ality), data=data,paired=FALSE)
pvalue(res)
```

Nonparametricbootstrap:

```
#Writea functionstatisticthatdependson dataframeand index and returnsthe estimator
library(boot)
data2=data[order(data$Area),]#reordering data according to Area
# computing bootstrap samples
f=function(data, ind){
data1=data[ind,]# extract bootstrap sample
res=lm(Price~Area, data=data1) #fit linear model
#predict values for all Area values from the original data
priceP=predict(res,newdata=data2)
return(priceP)
}
res=boot(data2, f, R=1000) #make bootstrap
```

Parametricbootstrap:

.Computevaluemlethatestimatesmodelparameters from the data .Writefunctionran.genthatdependson dataand mleand whichgenerates new data .Writefunctionstatisticthatdependon datawhichwillbe generatedby ran.genand shouldreturnthe estimator

```
mle=lm(Price~Area, data=data2)
rng=function(data, mle) {
data1=data.frame(Price=data$Price, Area=data$Area)
n=length(data$Price)
#generate new Price
data1$Price=rnorm(n,predict(mle, newdata=data1),sd(mle$residuals))
return(data1)
}
f1=function(data1){
res=lm(Price~Area, data=data1) #fit linear model
#predict values for all Area values from the original data
priceP=predict(res,newdata=data2)
return(priceP)
}
res=boot(data2, statistic=f1, R=1000, mle=mle,ran.gen=rng, sim="parametric")
```

Uncertaintyestimation

```
#Bootstrapcofidencebands for linearmodel
e=envelope(res) #compute confidence bands
fit=lm(Price~Area, data=data2)
priceP=predict(fit)
plot(Area, Price, pch=21, bg="orange")
points(data2$Area,priceP,type="l") #plot fitted line
#plot cofidence bands
points(data2$Area,e$point[2,], type="l", col="blue")
points(data2$Area,e$point[1,], type="l", col="blue")
```

Estimationofthe modelquality-parametricbootstrap

```
mle=lm(Price~Area, data=data2)
f1=function(data1){
res=lm(Price~Area, data=data1) #fitlinear model
#predict values for all Area valuesfrom the original data
priceP=predict(res,newdata=data2)
n=length(data2$Price)
predictedP=rnorm(n,priceP,
sd(mle$residuals))
return(predictedP)
}
res=boot(data2, statistic=f1, R=10000,
mle=mle,ran.gen=rng, sim="parametric")
```

PCA-principle

```
mydata=read.csv2("tecator.csv")
data1=mydata
data1$Fat=c()
res=prcomp(data1)
lambda=res$sdev^2
#eigenvalues
lambda
#proportion of variation
sprintf("%2.3f",lambda/sum(lambda)*100)
screeplot(res)
#Principal componentloadings(U)
U=res$rotation
head(U)
#Data in (PC1, PC2) -scores (Z)
plot(res$x[,1], res$x[,2], ylim=c(-5,15))
#Traceplots
U=loadings(res)
plot(U[,1], main="Traceplot, PC1")
plot(U[,2],main="Traceplot, PC2")
```

Independent component analysis-ICA

```
S <- cbind(sin((1:1000)/20), rep((((1:200)-100)/100), 5)) #Generate data
A <- matrix(c(0.291, 0.6557, -0.5439, 0.5572), 2, 2)
X <- S %*% A
a <- fastICA(X, 2, alg.typ = "parallel", fun = "logcosh", alpha = 1,
method = "R", row.norm = FALSE, maxit = 200, tol = 0.0001, verbose = TRUE) #ICA
```