Untitled

Prudhvi Peddmallu

24 April 2019

pca-pcr

The data file video.csv contains characteristics of a sample of Youtube videos. Import data to R and divide it randomly (50/50) into training and test sets. 1. Perform principal component analysis using the numeric variables in the training data except of "utime" variable. Do this analysis with and without scaling of the features. How many components are necessary to explain more than 95% variation of the data in both cases? Explain why so few components are needed when scaling is not done. (2p) 2. Write a code that fits a principle component regression ("utime" as response and all scaled numerical variables as features) with ???????? components to the training data and estimates the training and test errors, do this for all feasible ???????? values. Plot dependence of the training and test errors on ???????? and explain this plot in terms of bias-variance tradeoff. (Hint: prediction function for principal component regression has some peculiarities, see predict.mvr) (2p) 3. Use PCR model with ????????=8 and report a fitted probabilistic model that shows the connection between the target and the principal components. (1p)

4. Use original data to create variable "class" that shows "mpeg" if variable "codec" is equal to "mpeg4", and "other" for all other values of "codec". Create a plot of "duration" versus "frames" where cases are colored by "class". Do you think that the classes are easily separable by a linear decision boundary? (1p)
5. Fit a Linear Discriminant Analysis model with "class" as target and "frames" and "duration" as features to the entire dataset (scale features first). Produce the plot showing the classified data and report the training error. Explain why LDA was unable to achieve perfect (or nearly perfect) classification in this case. (2p)
6. Fit a decision tree model with "class" as target and "frames" and "duration" as features to the entire dataset, choose an appropriate tree size by cross-validation. Report the training error. How many leaves are there in the final tree? Explain why such a complicated tree is needed to describe such a simple decision boundary. (2p)

```
data0=read.csv("video.csv")

data1=data0
data1$codec=c()

n=dim(data1)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data1[id,]
test=data1[-id,]


data11=data1
data11$utime=c()
res=prcomp(data11)
lambda=res$sdev^2
sprintf("%2.3f",cumsum(lambda)/sum(lambda)*100)

res=prcomp(scale(data11))
lambda=res$sdev^2
```

```
sprintf("%2.3f",cumsum(lambda)/sum(lambda)*100)


data11=t(apply(as.matrix(data1[1:100,1:18]), 1, combn, 3, prod))
library(pamr)
mydata=as.data.frame(scale(data11))
rownames(mydata)=1:nrow(mydata)
x=t(mydata)
y=data0$codec[1:100]
mydata1=list(x=x,y=as.factor(y),geneid=as.character(1:nrow(x)), genenames=rownames(x))
model=pamr.train(mydata1,threshold=seq(0,4, 0.1))
set.seed(12345)
cvmodel=pamr.cv(model,mydata1)
pamr.plotcv(cvmodel)

cvmodel$threshold[which.max(cvmodel$loglik)]

data2=data0
data2$class=ifelse(data2$codec=="mpeg4", "mpeg4", "other")
data2$codec=c()
data2$frames=scale(data2$frames)
data2$duration=scale(data2$duration)

plot(data2$frames,data2$duration, col=as.factor(data2$class), cex=0.5)
m3=lda(as.factor(class)~frames+duration, data=data2)

plot(data2$frames,data2$duration, col=predict(m3)$class)

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

missclass(data2$class, predict(m3, type="class")$class)

library(tree)
m4=tree(as.factor(class)~frames+duration, data=data2)
set.seed(12345)
cv.res=cv.tree(m4)
plot(cv.res$size, cv.res$dev, type="b",
     col="red")

print(m4)
plot(m4)
missclass(data2$class, predict(m4, type="class"))
```

neuralnet-twolayers

NEURAL NETWORKS (3p) Train a neural network (NN) to learn the trigonometric sine function. To do so, sample 50 points uniformly at random in the interval [0, 10]. Apply the sine function to each point. The resulting pairs are the data available to you. Use 25 of the 50 points for training and the rest for validation. The validation set is used for early stop of the gradient descent. Consider threshold values i/1000 with i = 1,

. . . , 10. Initialize the weights of the neural network to random values in the interval [???1, 1]. Consider two NN architectures: A single hidden layer of 10 units, and two hidden layers with 3 units each. Choose the most appropriate NN architecture and threshold value. Motivate your choice. Feel free to reuse the code of the corresponding lab. (1p) Estimate the generalization error of the NN selected above (use any method of your choice). (1p) In the light of the results above, would you say that the more layers the better ? Motivate your answer.

```r
# JMP

library(neuralnet)

# two layers

set.seed(1234567890)

Var <- runif(50, 0, 10)
trva <- data.frame(Var, Sin=sin(Var))
tr <- trva[1:25,] # Training
va <- trva[26:50,] # Validation

# plot(trva)
# plot(tr)
# plot(va)

restr <- vector(length = 10)
resva <- vector(length = 10)
winit <- runif(22, -1, 1) # Random initializaiton of the weights in the interval [-1, 1]
for(i in 1:10) {
  nn <- neuralnet(formula = Sin ~ Var, data = tr, hidden = c(3,3), startweights = winit,
                  threshold = i/1000, lifesign = "full")

  # nn$result.matrix

  aux <- compute(nn, tr[,1])$net.result # Compute predictions for the trainig set and their squared err
  restr[i] <- sum((tr[,2] - aux)**2)/2

  aux <- compute(nn, va[,1])$net.result # The same for the validation set
  resva[i] <- sum((va[,2] - aux)**2)/2
}
plot(restr, type = "o")
plot(resva, type = "o")
restr
resva

# one layer

set.seed(1234567890)

Var <- runif(50, 0, 10)
trva <- data.frame(Var, Sin=sin(Var))
tr <- trva[1:25,] # Training
va <- trva[26:50,] # Validation

# plot(trva)
```

```r
# plot(tr)
# plot(va)

restr <- vector(length = 10)
resva <- vector(length = 10)
winit <- runif(41, -1, 1) # Random initializaiton of the weights in the interval [-1, 1]
for(i in 1:10) {
  nn <- neuralnet(formula = Sin ~ Var, data = tr, hidden = c(10), startweights = winit,
                  threshold = i/1000, lifesign = "full")

  # nn$result.matrix

  aux <- compute(nn, tr[,1])$net.result # Compute predictions for the trainig set and their squared err
  restr[i] <- sum((tr[,2] - aux)**2)/2

  aux <- compute(nn, va[,1])$net.result # The same for the validation set
  resva[i] <- sum((va[,2] - aux)**2)/2
}
plot(restr, type = "o")
plot(resva, type = "o")
restr
resva

# estimate generalization error for the best run above (one layer with threshold 4/1000)

Var <- runif(50, 0, 10)
te <- data.frame(Var, Sin=sin(Var))

winit <- runif(31, -1, 1)
nn <- neuralnet(formula = Sin ~ Var, data = trva, hidden = 10, startweights = winit,
                    threshold = 4/1000, lifesign = "full")
sum((te[,2] - compute(nn, te[,1])$net.result)**2)/2
```

SUPPORT VECTOR MACHINES-gaussian

(3p) Implement the budget online support vector machine (SVM). Check the course slides for the pseudo-code. Feel free to use the template below. Note that you are not using all the attributes and points in the file. Run your code on the spambase.csv file for the (M, ??) values (500,0) and (500,-0.05). Plot the error rate as a function of the number of training point. (2p) Analyze the results obtained. In particular, ??? explain why (500,0) gives better results than (500,-0.05), and ??? explain why the setting (50,0) is the slowest (you do not need to run your code until completion for this setting). #GIVEN CODE IN QUESTION

```r
set.seed(1234567890)
spam <- read.csv2("spambase.csv")
ind <- sample(1:nrow(spam))
spam <- spam[ind,c(1:48,58)]
h <- 1
beta <- # Your value here
M <- # Your value here
N <- 500 # number of training points
gaussian_k <- function(x, h) { # Gaussian kernel
# Your code here
```

```
}
SVM <- function(sv,i) { # SVM on point i with support vectors sv

# Your code here
# Note that the labels in spambase.csv are 0/1 and SVMs need -1/+1
# Then, use 2*label-1 to convert from 0/1 to -1/+1
# Do not include the labels when computing the Euclidean distance between
# the point i and each of the support vectors. This is the distance to use
# in the kernel function. You can use dist() to compute the Euclidean distance
}
errors <- 1
errorrate <- vector(length = N)
errorrate[1] <- 1
sv <- c(1)
for(i in 2:N) {
# Your code here
}
plot(errorrate[seq(from=1, to=N, by=10)], ylim=c(0.2,0.4), type="o")
length(sv)
errorrate[N]
```

CODE

```
set.seed(1234567890)
spam <- read.csv2("../Lab2aBlock2/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]
```

SVM

You are asked to use the function ksvm from the R package kernlab to learn a support vector machine (SVM) for classifying the spam dataset that is included with the package. Consider the radial basis function kernel (also known as Gaussian) with a width of 0.05. For the C parameter, consider values 0.5, 1 and 5. This implies that you have to consider three models. (2p) Perform model selection, i.e. select the most promising of the three models (use any method of your choice except cross-validation or nested cross-validation). (1p) Estimate the generalization error of the SVM selected above (use any method of your choice except cross-validation or

nested cross-validation). (1p) Produce the SVM that will be returned to the user, i.e. show the code. (1p) What is the purpose of the parameter C ?

```
library(kernlab)
set.seed(1234567890)

data(spam)

# Model selection

index <- sample(1:4601)
tr <- spam[index[1:2500], ]
va <- spam[index[2501:3501], ]
te <- spam[index[3502:4601], ]

filter <- ksvm(type~.,data=tr,kernel="rbfdot",kpar=list(sigma=0.05),C=0.5)
mailtype <- predict(filter,va[,-58])
t <- table(mailtype,va[,58])
(t[1,2]+t[2,1])/sum(t)

filter <- ksvm(type~.,data=tr,kernel="rbfdot",kpar=list(sigma=0.05),C=1)
mailtype <- predict(filter,va[,-58])
t <- table(mailtype,va[,58])
(t[1,2]+t[2,1])/sum(t)

filter <- ksvm(type~.,data=tr,kernel="rbfdot",kpar=list(sigma=0.05),C=5)
mailtype <- predict(filter,va[,-58])
t <- table(mailtype,va[,58])
(t[1,2]+t[2,1])/sum(t)

# Error estimation

filter <- ksvm(type~.,data=spam[index[1:3501], ],kernel="rbfdot",kpar=list(sigma=0.05),C=1)
mailtype <- predict(filter,te[,-58])
t <- table(mailtype,te[,58])
(t[1,2]+t[2,1])/sum(t)

# Final model

filter <- ksvm(type~.,data=spam,kernel="rbfdot",kpar=list(sigma=0.05),C=1)
```

GLM

```
library(glmnet)
library(dplyr)
library(ggplot2)
crabs<-read.csv("australian-crabs.csv")
crabss<-crabs[,c(1,7,8)]
model<-glm(data=crabss, formula=species ~ CW + BD, family=binomial(link='logit'))

ggplot
```

```
slope <- coef(crabs_logit)[2]/(-coef(crabs_logit)[3])
intercept <- coef(crabs_logit)[1]/(-coef(crabs_logit)[3])

# Fit the decision boundary
plot_x <- c(min(crabss$CW)-2, max(crabss$CW)+2)
plot_y <- (-1 /coef(model)[3]) * (coef(model)[1] * plot_x + coef(model)[2])
db.data1 <- data.frame(rbind(model, crabss))
colnames(db.data) <- c('x','y')

ggplot()+geom_line(data=db.data, aes(x=x, y=y))


set.seed(1234)
crabs<- rnorm(6,7)
x2 <- rnorm(20)

y <- sign(-1 - 2 * x1 + 4 * x2 )

y[ y == -1] <- 0

df <- cbind.data.frame( y, x1, x2)

mdl <- glm( y ~ . , data = df , family=binomial)

slope <- coef(mdl)[2]/(-coef(mdl)[3])
intercept <- coef(mdl)[1]/(-coef(mdl)[3])

library(lattice)
xyplot( x2 ~ x1 , data = df, groups = y,
   panel=function(...){
       panel.xyplot(...)
       panel.abline(intercept , slope)
       panel.grid(...)
       })
model <- glm(Species~.,family=binomial(link='logit'),data=crabss)
```

2NAIVE BAYES

```
#navies bayes
library(MASS)
library(e1071)
n=dim(crabs)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=crabs[id,]
test=crabs[-id,]
fitmodel_navitrain=naiveBayes(species~ CW+BD, data=train)
predection_trainN <-predict(fitmodel_navitrain,newdata=test,
                            type="class")
confusionmatrix_trainN<-table( test$species , predection_trainN)
confusionmatrix_trainN
```

```
diagonal_trainN<-diag(confusionmatrix_trainN)
summ_trainN<-sum(confusionmatrix_trainN)
misclassificationrate_trainN<-1-(sum(diagonal_trainN)/sum(summ_trainN))
misclassificationrate_trainN
```

PC

```
#PC
library(dplyr)
crabsP<-read.csv("australian-crabs.csv")
b1<-crabsP[,c(7,8)]
b2<-scale(b1)
#b3<-as.vector(b2)
#pca
#plot on viscosity
data_NIR=b2
res=prcomp(data_NIR)
lambda=res$sdev^2
#eigenvalues
lambda
#proportion of variation
sprintf("%2.3f",lambda/sum(lambda)*100)
screeplot(res)
```

BOOTSTRAP

```
dataB = read.csv('bank.csv', sep=';', dec=',')
data_bank<-read.csv2('bank.csv')
model_glm = glm('Visitors ~ .', data=data, family=poisson(link='log'))

library(ggplot2)
library(boot)
# Creating a copy of the original data.
original_data = data
# Data to be predicted.
data_seq = list(Time=seq(12, 13, 0.05))
# Creating the sampling function.
rng = function(data, model)
{
# Getting the parameters for the poisson distribution.
# In this case y_hat is the mean of the poisson distribution
# aka lambda.
lambda_hat = predict(model, newdata=data, type='response')
nobs = nrow(data)
# Sampling from a poisson distribution.
sample = rpois(nobs, lambda_hat)
data$Visitors = sample
return(data)
}
```

```r
# Function for the prediction interval.
pi_glm = function(data)
{
# Fitting a new model given the sample generated.
new_model = glm('Visitors ~ .',
data=data,
family=poisson(link='log'))
# Estimating E[Y|X] given the bootsrap model

# on the original data.
lambda_hat = predict(new_model, newdata=data_seq, type='response')
# Generating a sample from the bootstraped model.
nobs = length(data_seq$Time)
sample = rpois(nobs, lambda_hat)
return(sample)
}
# Running the bootstrap for the prediction interval.
results = boot(data,
statistic=pi_glm,
R=1000,
mle=model_glm,
ran.gen=rng,
sim='parametric')
p_results = envelope(results)
lambda_hat = predict(model_glm, newdata=data_seq, type='response')
p = ggplot() +
geom_line(aes(x=data_seq$Time, y=p_results$point[1, ], color="Prediction Interval")) +
geom_line(aes(x=data_seq$Time, y=p_results$point[2, ], color="Prediction Interval")) +
geom_point(aes(x=data_seq$Time, y=lambda_hat)) +
scale_colour_manual(values=c("#604dc5", "#020c0b"))
print(p)
```

```r
# plot dependence of CW vs BD where the points are colored by species.
library(dplyr)
library(ggplot2)
crabs %>%
ggplot(aes(x = BD, y = CW))+
geom_point(aes(color = species))+
theme_classic()
# split data into train and test
set.seed(12345)
n <- nrow(crabs)
id <- sample(1:n, floor(0.5*n))
crabs_train <- crabs[id,]
crabs_test <- crabs[-id,]
# Naive bayes classifier
library(e1071)
crabs_model <- naiveBayes(
species ~ BD + CW,
data = crabs_train
)
# prediction
crabs_pred <- predict(crabs_model, newdata = crabs_test,
type = "class"
```

```
)
# confusion matrix
table(crabs_test$species, crabs_pred)
# misclassification rate
mean(crabs_test$species != crabs_pred)
# test for independence between features.
# note it is not required only to prove a point.
cor.test(crabs$CW, crabs$BD)
```

```
# fit a logistic model
crabs_logit <- glm(species ~ BD + CW,
family = binomial(link = "logit"),
data = crabs_train
)
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
# prediction
crabs_pred2 <- predict(crabs_logit, newdata = crabs_test,
type = "response"
)
# threshold 0.5
crabs_pred2 <- ifelse(crabs_pred2 <0.5, 0,1)
table(crabs_test$species, crabs_pred2)
crabs_class <- ifelse(crabs_pred2 == 0, "Blue", "Orange")
# misclassification
mean(crabs_test$species != crabs_class)
# slope and intercept to get the decision boundary
# note this coefficients come from the trained model.
# [1] is the intercept coefficient in the crabs_logit
# [2] is coefficient for BD
# [3] is coefficient for CW
slope <- coef(crabs_logit)[2]/(-coef(crabs_logit)[3])
intercept <- coef(crabs_logit)[1]/(-coef(crabs_logit)[3])
#Plot the classification with the decision boundary
# plot
crabs_test %>%
ggplot(aes(x = BD, y = CW))+
geom_point(aes(color = species))+
geom_abline(slope = slope, intercept = intercept)+
theme_classic()
```

```
library(geosphere)
# A general gaussian kernel that takes one value
k.gaussian <- function(u) {
return(exp(-abs(u)^2))
}
# A kernel that preprocess two longitude, latitude argumnets
# for two position and find the distance between them
# with the help of distHaversine from the package geosphere.
# The calculated value is then passed throught to the general
# gaussian kernel
k.dist.station.poi <- function(s.longitude, s.latitude,
poi.longitude, poi.latitude) {
return(k.gaussian(distHaversine(c(s.longitude, s.latitude),
```

```
c(poi.longitude, poi.latitude))/h_distance))
}
# A kernel that calculates how many days is between two dates.
# But ignores the year difference between them, this is so that we get
# a periodic kernel. But it has to use 365.25 because of each 4 years we
# have a leap year.
# When the distance is calculated it is passed throught to the
# general gaussian kernel.
k.dist.day.poi <- function(d, poi) {
daydiff <- min(abs(as.numeric(difftime(d, poi))%%365.25),
abs(as.numeric(difftime(poi,d))%%365.25))
return(k.gaussian(daydiff/h_date))
}
# A kernel that fins the time difference between two times.
# We have to check both t2-t1 and t2-t1-24 beacuse of the
# reason if we have the times t2 = 22 and t1 = 1 we get
# 21 when we want to have the time difference of 3, which we get
# from the second case.
# This is so that the maximun time difference we can have
# is 12 hours and not 24
k.dist.hour.poi <- function(h, poi) {
t <- as.difftime(c(toString(h),
toString(poi)), units = "hours")
t <- min(abs(as.numeric(t[2]-t[1])), abs(as.numeric(t[2]-t[1]-24)))
return(k.gaussian(t/h_time))
}
set.seed(1234567890)
# Read in the data
stations <- read.csv("stations.csv", fileEncoding = "latin1")
temps <- read.csv("temps50k.csv")
st <- merge(stations,temps,by="station_number")
# Is somewhere near Vadstena
a <- 58.4274 # The point to predict
b <- 14.826
#date <- "1970-04-04" # The date to predict (up to the students)
h_distance <- 100000 # How long distance away to consider. The unit is meters
h_date <- 15 # How many days away to consider. The unit is days
h_time <- 3 # How many hours away to consider. The unit is hours
#a <- 67.827
#b <- 20.3387
date <- "2000-07-04"
# Set up a the times that we shall predict
times <- paste(seq(4, 24, 2), ":00:00", sep = '')
# Create the vector where we store the predicted times
temp.sum <- vector(length=length(times))
temp.prod <- temp.sum
# Remove all dates after the predicted date, including the prediction date
st <- st[which(difftime(st[,"date"], date, units = "days") <= 0),]

library(geosphere)
set.seed(1234567890)
# Create the data frame that we shall pass to our kernels
poi <- data.frame(b, a, date, times)
```

```r
colnames(poi) <- c("longitude", "latitude", "date", "time")
vec.kern.dist <- 1:nrow(st)
vec.kern.day <- vec.kern.dist
vec.kern.hour <- vec.kern.dist
# Precalculate the distance and day kernel
# We do this to avoid calculating this for every time
# that we want to predict. This is beacuse the distance
# and day difference does not change when we change
# the time.
vec.kern.dist <- mapply(k.dist.station.poi, st$longitude, st$latitude,
poi[1,]$longitude, poi[1,]$latitude)
vec.kern.day <- mapply(k.dist.day.poi, st$date, poi[1,]$date)
# Calculate the time difference kernel for every data point and
# time. Then for each time apply sum or multiple the kernels
# with each other to then multiple with the correct temperature
# for the day and divide by the sum&multiplication.
# Now we have predicted the temperature for the given
# place, day and time.
for (i in 1:nrow(poi)) {
vec.kern.hour <- mapply(k.dist.hour.poi, st$time, poi[i,]$time)
vec.kern.sum <- vec.kern.dist+vec.kern.day+vec.kern.hour
temp.sum[i] <- sum(vec.kern.sum*st$air_temperature)/sum(vec.kern.sum)
vec.kern.prod <- vec.kern.dist*vec.kern.day*vec.kern.hour
temp.prod[i] <- sum(vec.kern.prod*st$air_temperature)/sum(vec.kern.prod)
}
# Plot the difference predicted temperature
plot(temp.sum, type="o")
plot(temp.prod, type="o")
x <- seq(0, 500000, 10000)
y <- k.gaussian(x/h_distance)
plot(x=x, y=y)
# ==============
# Assignment 2
# ==============
library(neuralnet)
set.seed(1234567890)
var <- runif(50, 0, 10)
trva <- data.frame(var, sin=sin(var))
tr <- trva[1:25,]
va <- trva[26:50,]
mse <- rep(0, 10)
# Random initial weights
winit <- runif(31, -1, 1)
for(i in 1:10){
# Train neural net
nn <- neuralnet(sin~var, data=tr, hidden=c(10), threshold = i/1000, startweights = winit)
# Predict for test data
pred <- compute(nn, va$var)$net.result
# Compute error (mean squared)
mse[i] <- mean((va$sin - pred)^2)
}
# Threshold which produced minimal error
threshold <- which.min(mse)/1000
```

```r
# Plot net corresponding to minimal error
plot(nn <- neuralnet(sin~var, data=tr, hidden=c(10), threshold=threshold, startweights=winit),
rep="best", main="The Obtained Network")
# Plot prediction and true values
invisible(capture.output(res <- prediction(nn)$rep1))
library(ggplot2)
vis_res <- data.frame(x=res[,1], y=res[,2], Set="Prediction")
vis_orig <- data.frame(x=trva$var, y=trva$sin, Set="Original")
visuals <- rbind(vis_res, vis_orig)
p <- ggplot(visuals, aes(x=x, y=y, color=Set)) + geom_point(shape = 21, size=3, stroke=1) +
labs(x="x", y="Sin(x)") +
ggtitle("Orignal and predicted data")
print(p)
```