All working Codes for ML Lab

Lab 1

-

Lab 2

```
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)


## lm is used to fit linear model
relation <- lm(y~x)


## for printing the data
print(relation)

print(summary(relation))
## predict function is used to predict new values
a <- data.frame(x= 170)
result <- predict(relation, a)
print(result)

### Visualization
#creating the predictor and respone variable
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
y <- c(63,81, 56, 91, 47, 57, 76, 72, 62, 48)
relation <- lm(y~x)

# To give the chart file a name
png(file ="linearregression.png")
# Plotting the chart

plot(x,y,col = "blue", main = "Height & weight Regression", abline(lm(y~x)), cex = 1.3, pch = 16,
xlab = "Height in cm", ylab= "Weight in kg")


#Saving the file
dev.off()
```

```
# set.seed(n) generate pseudo random numbers.
#By doing this, the random number generator generates always the same numbers

set.seed(20)
q <- seq(from=0, to=20, by=0.1)
#Predictor(q)
y <- 500 + 0.4* (q-10)^3
noise <- rnorm(length(q), mean=10, sd=80)
noisy.y <- y+noise
plot(q, noisy.y, col='deepskyblue4', xlab='q', main='Observed data')
lines(q, y, col='firebrick1',lwd=3)


model <- lm(noisy.y ~ poly(q,3))
#model <- lm(noisy.y ~ x + I(X^2) + I(X^3))

predicted.intervals <- predict(model,data.frame(x=q),interval='confidence', level=0.99)
lines(q,predicted.intervals[,1],col='green',lwd=3)
lines(q,predicted.intervals[,2],col='black',lwd=1)
lines(q,predicted.intervals[,3],col='black',lwd=1)


#adding legends to the plot
#legend("bottomright",c("Observ.","Signal","Predicted"), col=c("deepskyblue4","red","green"),
lwd=3)

#For Finding PCA
#prin_comp <- prcomp(air_data, scale. = T)
```

Lab 3

```
head(mtcars)
d01 <- mtcars[, c(1:7,10,11)]
head(d01)
d01.pca <- princomp(d01, cor=TRUE, score=TRUE)
```

```
summary(d01.pca)
plot(d01.pca)

d01.pca$loadings

d02 <- d01.pca$scores
head(d02)
```

Lab 4

```
install.packages("e1071")
library(e1071)
dataset("Titanic")
print(Titanic)
Titanic_dataFrame = as.data.frame(Titanic)
print(Titanic_dataFrame)
sequence = rep.int(seq_len(nrow(Titanic_dataFrame)), Titanic_dataFrame$Freq)
Titanic_dataset=Titanic_dataFrame[sequence,]
Titanic_dataset$Freq = NULL


model=naiveBayes(Survived ~., data=Titanic_dataset)
print(model)

prediction = predict(model, Titanic_dataset)
table( prediction, Titanic_dataset$Survived)

## using laplace smoothing:
model=naiveBayes(Survived ~., data=Titanic_dataset, laplace = 3)
print(model)
```

Lab 5

```
data(iris)
```

```
idxs <- sample(1:nrow(iris), as.integer(0.7*nrow(iris)))

#Splitting in training and testing set
trainIris <- iris[idxs,]
testIris <-iris[-idxs,]
cvIris <-iris[-idxs,]
trainIris1 <- trainIris[,-5]
testIris1 <- testIris[,-5]

# to Install kNN install.packages("class")

## A 3 nearest neightbours model with no normalization
#nn3 <- knn(Species ~ ., trainIris, testIris, norm=FALSE, k=3)
nn3 <- knn(train = trainIris1, test = testIris1, cl= trainIris$Species,k = 3)

## The resulting confusion matrix
table(testIris[,'Species'], nn3)


## Now a 5 nearest neighbours model with normalization
#nn5 <- kNN(Species ~., trainIris, testIris, norm = TRUE, k=5)
nn5 <- knn(train = trainIris1, test = testIris1, cl=trainIris$Species, k=5)

# The resulting confusion matrix
table(testIris[,'Species'], nn5)

##this function divides the correct predictions by total number of predictions that tell us how
accurate the model is.

#accuracy <- function(table){sum(diag(table)/(sum(rowSums(table)))) * 100}
#accuracy(nn3)

#install.packages("gmodels")
#library(gmodels)
#CrossTable(x=testIris, y=trainIris, prop.chisq=FALSE)
```

Lab 6

```
#cifar_10 <-read.table( unz( "cifar_10.zip" , "cifar_10.csv" ) , nrows = 500 , header = T, quote =
"\"" , sep = "," )
# For unzipped file, write the below line
cifar_10 <-read.table( "cifar_10.csv", nrows = 500 , header = T, quote = "\"" , sep = "," )

X= cifar_10[, -ncol(cifar_10)]
dim(X)

y= cifar_10[, ncol(cifar_10)]
table(y)

#install.packages("OpenImageR")
#library(OpenImageR)

hog = HOG_apply(X, cells = 6, orientations = 9, rows = 28, columns = 28, threads = 6)
dim(hog)
#install.packages("KernelKnn")
library(KernelKnn)
fit = KernelKnnCV(as.matrix(new_x), y, k=8, folds = 4, method = 'braycurtis',weights_function =
'biweight_tricube_MULT', regression = F, threads = 6, Levels = sort(unique(y)))

#evaluation metric
acc=function(y_true, preds){
  out = table(y_true, max.col(preds, ties.method = "random"))
  acc=sum(diag(out))/sum(out)
  acc
}
acc_fit = unlist(lappy(1:length(fit$preds), function(x) acc(y[fit$folds[[x]]], fit$preds[[x]])))
acc_fit
cat('mean accuracy using cross-validation :', mean(acc_fit),'\n')
```

LAB 7

```
n <-150
p <- 2
sigma <- 1
meanpos <- 0
meanneg <- 3
npos <- round(n/2)
```

```
nneg <- n-npos

xpos <- matrix(rnorm(npos*p, mean=meanpos, sd=sigma), npos, p)
xneg <- matrix(rnorm(nneg*p, mean=meanneg, sd=sigma), npos, p)
x <- rbind(xpos, xneg)

y <- matrix(c(rep(1,npos), rep(-1,nneg)))
plot(x,col=ifelse(y>0,1,2))
legend("topleft", c("Positive", "Negative"), col=seq(2), pch=1, text.col= seq(2))

#Splitting the dataset

ntrain <- round(n*0.8)
tindex <- sample(n,ntrain)
xtrain <- x[tindex,]
xtest <- x[-tindex,]
ytrain <- y[tindex]
ytest <- y[-tindex]
istrain = rep(0,n)
istrain[tindex] = 1

#visualize
plot(x,col=ifelse(y>0,1,2), pch=ifelse(istrain==1,1,2))
legend("topleft",c("Positive Train","Positive Test", "Negative Train", "Negative Test"),
col=c(1,1,2,2), pch=c(1,2,1,2), text.col = c(1,1,2,2))

#instal e1071 and kernlab

#Training  SVM
library(kernlab)
svp <- ksvm(xtrain, ytrain, type="C-svc", kernel = "vanilladot", C=100, scaled=c())
svp
attributes(svp)
alpha(svp)
alphaindex(svp)
b(svp)

#predicting with SVM
plot(svp, data=xtrain)
ypred = predict(svp,xtest)
table(ytest,ypred)
sum(ypred==ytest)/length(ytest)
ypredscore = predict(svp,xtest,type="decision")
```

```
table(ypredscore > 0,ypred)
```

Lab 9 - Decision Trees

```
install.packages("party")
library(party)
readingSkills
input.dat <- readingSkills[c(1:105),]

#png(file= "decision_tree.png")

output.tree <- ctree(
  nativeSpeaker ~ age + shoeSize + score,
  data=input.dat)


plot(output.tree)

dev.off()


#install.packages("randomForest")
library(party)

library(randomForest)

output.forest <- randomForest(nativeSpeaker ~ age + shoeSize + score,
                              data = readingSkills)

print(output.forest)

print(importance(output.forest,type = 2))
require(rpart)
output.forest.rpart1 = rpart(nativeSpeaker ~ age + shoeSize + score,
                             data = readingSkills)
```

```
plotcp(output.forest.rpart1)

output.forest.rpart2 = prune(output.forest.rpart1, cp =0.02)
plot(output.forest.rpart2, uniform = TRUE)
text(output.forest.rpart2, use.n = TRUE, cex = 0.75)
output.forest.rpart1
readingSkills
```