LAB ASSIGNMENT -9

NAME: JYOTHI K C

REG NO:19BDS0144

SLOT: L21+L22

COURSECODE: CSE3046- Programming for Data Science

1) Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

**CODE:**

#Purity- contains only one class

IsPure <- function(data) {

length(unique(data[,ncol(data)])) == 1

}

#Entropy- measure of purity

Entropy <- function( vls ) {

res <- vls/sum(vls) \* log2(vls/sum(vls))

res[vls == 0] <- 0

-sum(res)

}

#entropy as a function of number of edible mushrooms in a set of 100 mushrooms

entropy <- function(edible) Entropy(c(edible, 100 - edible))

entropy <- Vectorize(entropy)

curve( entropy, from = 0, to = 100, xname = 'edible')

#Information Gain= difference between the entropy before the split, and the weighted sum of the entropies after the split

InformationGain <- function( tble ) {

tble <- as.data.frame.matrix(tble)

entropyBefore <- Entropy(colSums(tble))

s <- rowSums(tble)

entropyAfter <- sum (s / sum(s) \* apply(tble, MARGIN = 1, FUN = Entropy ))

informationGain <- entropyBefore - entropyAfter

return (informationGain)

}

```

```{r}

InformationGain <- function( tble ) {

tble <- as.data.frame.matrix(tble)

entropyBefore <- Entropy(colSums(tble))

s <- rowSums(tble)

entropyAfter <- sum (s / sum(s) \* apply(tble, MARGIN = 1, FUN = Entropy ))

informationGain <- entropyBefore - entropyAfter

return (informationGain)

}

```

```{r}

library(data.tree)

data(mushroom)

tble <- table(mushroom[,c('color', 'edibility')])

tble

```

InformationGain(tble)

InformationGain(table(mushroom[,c('size', 'edibility')]))

InformationGain(table(mushroom[,c('points', 'edibility')]))

#ID3 algorithm

TrainID3 <- function(node, data) {

node$obsCount <- nrow(data)

#if the data-set is pure (e.g. all toxic), then

if (IsPure(data)) {

#construct a leaf having the name of the pure feature (e.g. 'toxic')

child <- node$AddChild(unique(data[,ncol(data)]))

node$feature <- tail(names(data), 1)

child$obsCount <- nrow(data)

child$feature <- ''

} else {

#chose the feature with the highest information gain (e.g. 'color')

ig <- sapply(colnames(data)[-ncol(data)],

function(x) InformationGain(

table(data[,x], data[,ncol(data)])

)

)

feature <- names(ig)[ig == max(ig)][1]

node$feature <- feature

#take the subset of the data-set having that feature value

childObs <- split(data[,!(names(data) %in% feature)], data[,feature], drop = TRUE)

for(i in 1:length(childObs)) {

#construct a child having the name of that feature value (e.g. 'red')

child <- node$AddChild(names(childObs)[i])

#call the algorithm recursively on the child and the subset

TrainID3(child, childObs[[i]])

}

}

}

#Training the model

tree <- Node$new("mushroom")

TrainID3(tree, mushroom)

print(tree, "feature", "obsCount")

#Predicting function

Predict <- function(tree, features) {

if (tree$children[[1]]$isLeaf) return (tree$children[[1]]$name)

child <- tree$children[[features[[tree$feature]]]]

return ( Predict(child, features))

}

#Prediction

Predict(tree, c(color = 'red',

size = 'large',

points = 'yes')

)

**OUTPUT:**

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, email

Description automatically generated

Table

Description automatically generated with medium confidence

Graphical user interface, text, application, email

Description automatically generated

2) Write a program to implement the naïve Bayesian classifier for a sample training data set. Compute the accuracy of the classifier, considering few test data sets.

**CODE:**

IRIS dataset- to predict species of iris( target variable: species)

# Loading data

data(iris)

# Structure

str(iris)

# Installing Packages

install.packages("e1071")

install.packages("caTools")

install.packages("caret")

# Loading package

library(e1071)

library(caTools)

library(caret)

# Splitting data into train

# and test data

split <- sample.split(iris, SplitRatio = 0.7)

train\_cl <- subset(iris, split == "TRUE")

test\_cl <- subset(iris, split == "FALSE")

# Feature Scaling

train\_scale <- scale(train\_cl[, 1:4])

test\_scale <- scale(test\_cl[, 1:4])

# Fitting Naive Bayes Model

# to training dataset

set.seed(120) # Setting Seed

classifier\_cl <- naiveBayes(Species ~ ., data = train\_cl)

classifier\_cl

# Predicting on test data'

y\_pred <- predict(classifier\_cl, newdata = test\_cl)

# Confusion Matrix

cm <- table(test\_cl$Species, y\_pred)

cm

# Model Evaluation

confusionMatrix(cm)

TITANIC dataset – to predict survivors(target variable:survived)

train <- read.csv("C:\\Users\\kcjyo\\Desktop\\studies\\sem 5\\Programming for ds\\lab\\train\_titanic.csv")

head(train)

test <- read.csv("C:\\Users\\kcjyo\\Desktop\\studies\\sem 5\\Programming for ds\\lab\\test\_titanic.csv")

head(test)

survive <-read.csv("C:\\Users\\kcjyo\\Desktop\\studies\\sem 5\\Programming for ds\\lab\\gender\_submission.csv")

head(survive)

#Data preparation

test1 <- cbind(test, Survived = survive$Survived)

```

```{r}

colSums(is.na(train))

library(dplyr)

train\_clean <- train %>%

mutate(Age = if\_else(is.na(Age), mean(Age, na.rm = TRUE), Age))

colSums(is.na(train\_clean))

colSums(is.na(test1))

test\_clean <- test1 %>%

na.omit()

```

```{r}

data\_train <- train\_clean %>%

select(-c(PassengerId, Name, Ticket, Cabin)) %>%

mutate(Survived = as.factor(Survived),

Pclass = as.factor(Pclass),

Sex = as.factor(Sex),

SibSp = as.factor(SibSp),

Parch = as.factor(Parch),

Embarked = as.factor(Embarked))

```

```{r}

data\_test <- test\_clean %>%

select(-c(PassengerId, Name, Ticket, Cabin)) %>%

mutate(Survived = as.factor(Survived),

Pclass = as.factor(Pclass),

Sex = as.factor(Sex),

SibSp = as.factor(SibSp),

Parch = as.factor(Parch),

Embarked = as.factor(Embarked))

```

```{r}

str(data\_train)

```

prop.table(table(data\_train$Survived))

set.seed(267)

data\_traind <- downSample(x = data\_train[, -1], y = as.factor(data\_train[, 1]), yname = "Survived")

rmarkdown::paged\_table(data\_traind)

```

prop.table(table(data\_traind$Survived))

model\_naive <- naiveBayes(x = data\_traind %>% select(-Survived), #prediktor

y = data\_traind$Survived, #target

laplace = 1)

```

```{r}

pred\_label\_naive <- predict(model\_naive, data\_test, type = "class")

head(data.frame(actual = data\_test$Survived, prediction = pred\_label\_naive))

mat1 <- confusionMatrix(data = pred\_label\_naive, reference = data\_test$Survived, positive = "1")

mat1

**OUTPUT:**

Dataset -iris

A picture containing table

Description automatically generated

Graphical user interface

Description automatically generated

Graphical user interface, text, email

Description automatically generated

Graphical user interface, text

Description automatically generated

Graphical user interface, text

Description automatically generated with medium confidence

Graphical user interface, text

Description automatically generated with medium confidence

Graphical user interface, text

Description automatically generated

Text

Description automatically generated with medium confidence

Graphical user interface, application, table

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, email

Description automatically generated

Graphical user interface, text

Description automatically generated with medium confidence

Graphical user interface, text, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated