Experiment No: 3

**Aim:** To study and implement following classifiers

1. Naive Bayes Classifier
2. Decision Tree

**Part A**

**Naive Bayes Classifier**

**Theory:**

Bayesian classifiers are statistical classifiers. They can predict class membership probabilities such as the probability that a given tuple belongs to a particular class. Bayesian classification is based on Bayes’ theorem.

**Algorithm:**

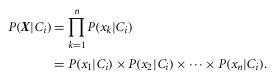
1. Let D be a training set of tuples and their associated class labels. As usual, each tuple is represented by an n-dimensional attribute vector, X = (x1 , x2 , . . . , xn ), depicting n measurements made on the tuple from n attributes, respectively, A1 , A2 , . . . , An .
2. Suppose that there are m classes, C1 , C2 , . . . , Cm. Given a tuple, X, the classifier will predict that X belongs to the class having the highest posterior probability, conditioned on X. That is, the naive Bayesian classifier predicts that tuple X belongs to the class Ci if and only if

P(Ci |X) > P(Cj |X) for 1 ≤ j ≤ m, j ≠ i.

Thus, we maximize P(Ci|X). The class Ci for which P(Ci|X) is maximized is called the maximum posteriori hypothesis. By Bayes’ theorem



1. As P(X) is constant for all classes, only P(X|Ci)P(Ci) needs to be maximized. If the class prior probabilities are not known, then it is commonly assumed that the classes are equally likely, that is, P(C1) = P(C2) = · · · = P(Cm), and we would therefore maximize P(X|Ci). Otherwise, we maximize P(X|Ci)P(Ci). Note that the class prior probabilities may be estimated by P(Ci) = |CiD|/|D|, where |Ci ,D | is the number of training tuples of class Ci in D.
2. Given data sets with many attributes, it would be extremely computationally expensive to compute P(X|Ci). To reduce computation in evaluating P(X|Ci), the naive assumption of class-conditional independence is made. This presumes that the attributes’ values are conditionally independent of one another, given the class label of the tuple (i.e., that there are no dependence relationships among the attributes). Thus,



We can easily estimate the probabilities P(x1|Ci), P(x2|Ci), . . . , P(xn|Ci) from the training tuples. Recall that here xk refers to the value of attribute Ak for tuple X.

1. To predict the class label of X, P(X|C)P(Ci) is evaluated for each class Ci. The classifier predicts that the class label of tuple X is the class Ci if and only if

P(X|Ci)P(Ci) > P(X|Ci) P(Cj) for 1 ≤ j ≤ m, j ≠ i.

In other words, the predicted class label is the class Ci for which P(X|Ci)P(Ci) is the maximum.

**Program:**

import java.util.\*;

public class btheo

{

static char outlook[]={'S','S','O','R','R','R','O','S','S','R','S','O','O','R'};

static char temperature[]={'H','H','H','M','C','C','C','M','C','M','M','M','H','M'};

static char humidity[]={'P','P','P','P','N','N','N','P','N','N','N','P','N','P'};

static char windy[]={'F','T','F','F','F','T','T','F','F','F','T','T','F','T'};

static char class1[]={'N','N','P','P','P','N','P','N','P','P','P','P','P','N'};

static double prob[][]=new double[4][2];

static double pp=9.0/14.0;

static double npp=5.0/14.0;

static int flag=0;

static int flag1=0;

static double play\_N=1;

static double notplay\_N=1;

static void cal\_N(int a)

{

if(a==1)

{

for(int i=0;i<4;++i)

play\_N\*=prob[i][0];

play\_N\*=pp;

}

else

{

for(int i=0;i<4;++i)

notplay\_N\*=prob[i][1];

notplay\_N\*=npp;

}

}

static double cal\_play\_prob(char ch)

{

double prob=0;

double count=0;

if(flag==0)

{

for(int i=0;i<14;++i)

if(outlook[i]==ch && class1[i]=='P')

++count;

prob=count/9.0;

flag=1;

}

else if(flag==1)

{

for(int i=0;i<14;++i)

if(temperature[i]==ch && class1[i]=='P')

++count;

prob=count/9.0;

flag=2;

}

else if(flag==2)

{

for(int i=0;i<14;++i)

if(humidity[i]==ch && class1[i]=='P')

++count;

prob=count/9.0;

flag=3;

}

else

if(flag==3)

{

for(int i=0;i<14;++i)

if(windy[i]==ch && class1[i]=='P')

++count;

prob=count/9.0;

}

return prob;

}

static double cal\_noplay\_prob(char ch)

{

double prob=0;

double count=0;

if(flag1==0)

{

for(int i=0;i<14;++i)

if(outlook[i]==ch && class1[i]=='N')

++count;

prob=count/5.0;

flag1=1;

}

else if(flag1==1)

{

for(int i=0;i<14;++i)

if(temperature[i]==ch && class1[i]=='N')

++count;

prob=count/5.0;

flag1=2;

}

else if(flag1==2)

{

for(int i=0;i<14;++i)

if(humidity[i]==ch && class1[i]=='N')

++count;

prob=count/5.0;

flag1=3;

}

else

if(flag1==3)

{

for(int i=0;i<14;++i)

if(windy[i]==ch && class1[i]=='N')

++count;

prob=count/5.0;

}

return prob;

}

public static void main(String args[])

{

Scanner scr=new Scanner(System.in);

System.out.println("Table\n");

System.out.println("Outlook\t Temperature\t Humidity\t Windy \tClass");

for(int i=0;i<14;++i){

System.out.print(outlook[i]+"\t\t"+temperature[i]+"\t\t"+humidity[i]+"\t\t"+windy[i]+"\t\t"+class1[i]);

System.out.println();

}

System.out.println("Menu:\nOutlook: Sunny=S Overcast=O Rain=R\tTemperature: Hot=H Mild=M Cool=C\n");

System.out.println("Humidity: Peak=P Normal=N\t\tWindy: True=T False=F\n\nYour input should belong to one of these classes.\n");

System.out.println("class1: Play=P class2:Not Play=NP");

System.out.println("\nEnter your input: example. t={rain,hot,peak,false} input will be R,H,P,F");

String s=scr.nextLine();

char ch;

int count=0;

for(int i=0;i<8;i+=2)

{

ch=s.charAt(i);

prob[count][0]=cal\_play\_prob(ch);

prob[count][1]=cal\_noplay\_prob(ch);

++count;

}

cal\_N(1);

cal\_N(2);

double pt=play\_N+notplay\_N;

double prob\_of\_play=0;

double prob\_of\_noplay=0;

prob\_of\_play=play\_N/pt;

prob\_of\_noplay=notplay\_N/pt;

System.out.println("\nProbability of play "+prob\_of\_play);

System.out.println("\nProbability of NO play "+prob\_of\_noplay );

if(prob\_of\_play>prob\_of\_noplay)

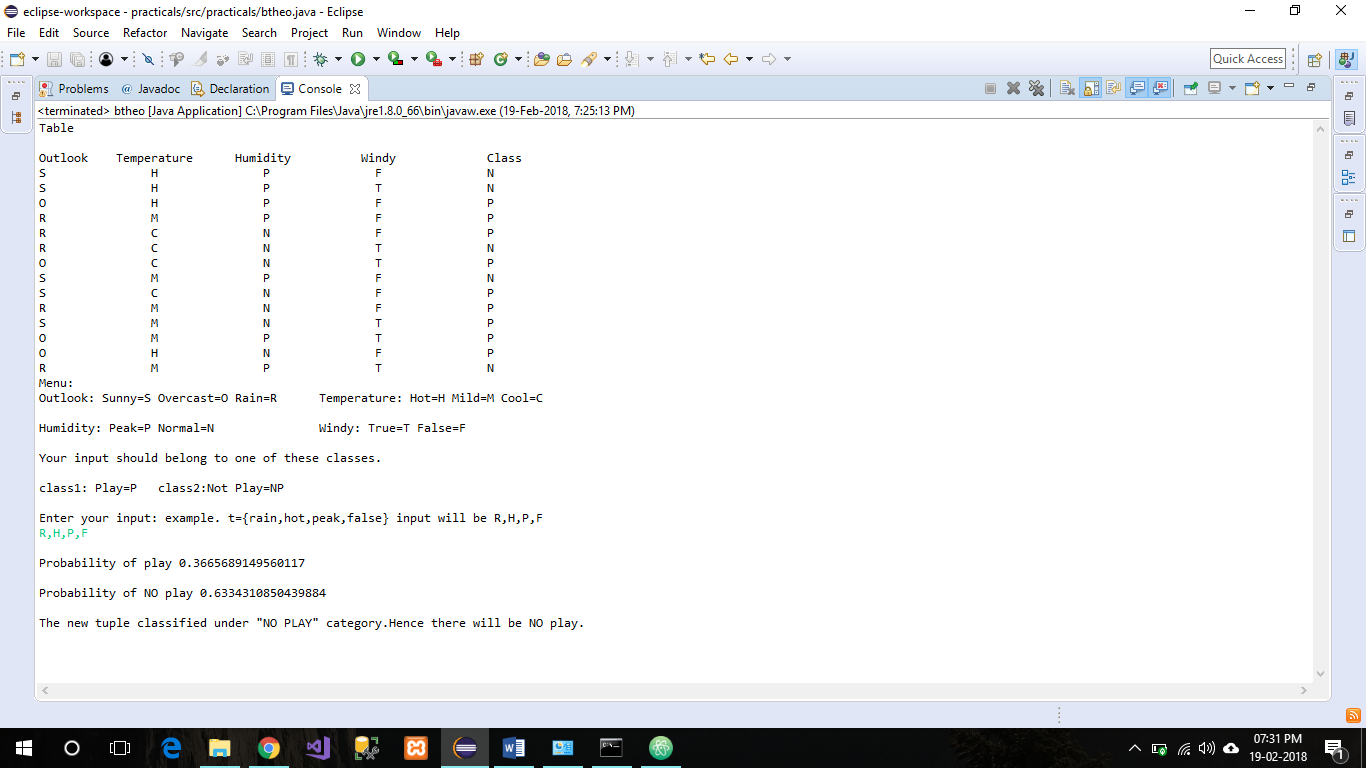
System.out.println("\nThe new tuple classified under \"PLAY\" category.Hence there will be play!!!");

else

System.out.println("\nThe new tuple classified under \"NO PLAY\" category.Hence there will be NO play.");

}}

**Output:**



**Part B**

**Decision Tree**

**Theory :**

Decision tree induction is the learning of decision trees from class-labeled training tuples. A decision tree is a flowchart-like tree structure, where each internal node (non leaf node) denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (or terminal node) holds a class label. The topmost node in a tree is the root node.

**Algorithm:**

**Input:** Data partition, D,

**Output:** A decision tree.

1. create a node N ;
2. if tuples in D are all of the same class, C, then
3. return N as a leaf node labeled with the class C;
4. if attribute list is empty then
5. return N as a leaf node labeled with the majority class in D; // majority voting
6. apply Attribute selection method(D, attribute list) to find the “best” splitting criterion;
7. label node N with splitting criterion;
8. if splitting attribute is discrete-valued and
9. multiway splits allowed then // not restricted to binary trees
10. attribute list ← attribute list − splitting attribute; // remove splitting attribute
11. for each outcome j of splitting criterion
12. // partition the tuples and grow subtrees for each partition.
13. let Dj be the set of data tuples in D satisfying outcome j; // a partition
14. if Dj is empty then
15. attach a leaf labeled with the majority class in D to node N ;
16. else attach the node returned by Generate decision tree(Dj , attribute list) to node N ;
17. endfor
18. return N

**Program:**

***DecisionTree.java***

import java.io.\*;

class DecisionTree {

private class BinTree {

private int nodeID;

private String questOrAns = null;

private BinTree yesBranch = null;

private BinTree noBranch = null;

public BinTree(int newNodeID, String newQuestAns) {

nodeID = newNodeID;

questOrAns = newQuestAns;

}

}

static BufferedReader keyboardInput = new BufferedReader(new InputStreamReader(System.in));

BinTree rootNode = null;

public DecisionTree() {

}

public void createRoot(int newNodeID, String newQuestAns) {

rootNode = new BinTree(newNodeID,newQuestAns);

System.out.println("Created root node " + newNodeID);

}

public void addYesNode(int existingNodeID, int newNodeID, String newQuestAns) {

if (rootNode == null) {

System.out.println("ERROR: No root node!");

return;

}

if (searchTreeAndAddYesNode(rootNode,existingNodeID,newNodeID,newQuestAns)) {

System.out.println("Added node " + newNodeID +" onto \"yes\" branch of node " + existingNodeID);

}

else System.out.println("Node " + existingNodeID + " not found");

}

private boolean searchTreeAndAddYesNode(BinTree currentNode,

int existingNodeID, int newNodeID, String newQuestAns) {

if (currentNode.nodeID == existingNodeID) {

if (currentNode.yesBranch == null) currentNode.yesBranch = new BinTree(newNodeID,newQuestAns);

else {

System.out.println("WARNING: Overwriting previous node " +"(id = " + currentNode.yesBranch.nodeID +") linked to yes branch of node " +existingNodeID);

currentNode.yesBranch = new BinTree(newNodeID,newQuestAns);

}

return(true);

}

else {

if (currentNode.yesBranch != null) {

if (searchTreeAndAddYesNode(currentNode.yesBranch,existingNodeID,newNodeID,newQuestAns)) {

return(true);

}

else {

if (currentNode.noBranch != null) {

return(searchTreeAndAddYesNode(currentNode.noBranch,existingNodeID,newNodeID,newQuestAns));

}

else return(false);

}

}

return(false);

}

}

public void addNoNode(int existingNodeID, int newNodeID, String newQuestAns) {

if (rootNode == null) {

System.out.println("ERROR: No root node!");

return;

}

if (searchTreeAndAddNoNode(rootNode,existingNodeID,newNodeID,newQuestAns)) {

System.out.println("Added node " + newNodeID +" onto \"no\" branch of node " + existingNodeID);

}

else System.out.println("Node " + existingNodeID + " not found");

}

private boolean searchTreeAndAddNoNode(BinTree currentNode,

int existingNodeID, int newNodeID, String newQuestAns) {

if (currentNode.nodeID == existingNodeID) {

if (currentNode.noBranch == null) currentNode.noBranch = new

BinTree(newNodeID,newQuestAns);

else {

System.out.println("WARNING: Overwriting previous node " +"(id = " + currentNode.noBranch.nodeID +") linked to yes branch of node " +existingNodeID);

currentNode.noBranch = new BinTree(newNodeID,newQuestAns);

}

return(true);

}

else {

if (currentNode.yesBranch != null) {

if (searchTreeAndAddNoNode(currentNode.yesBranch,

existingNodeID,newNodeID,newQuestAns)) {

return(true);

}

else {

if (currentNode.noBranch != null) {

return(searchTreeAndAddNoNode(currentNode.noBranch,

existingNodeID,newNodeID,newQuestAns));

}

else return(false);

}

}

else return(false);

}

}

public void queryBinTree() throws IOException {

queryBinTree(rootNode);

}

private void queryBinTree(BinTree currentNode) throws IOException {

if (currentNode.yesBranch==null) {

if (currentNode.noBranch==null) System.out.println(currentNode.questOrAns);

else System.out.println("Error: Missing \"Yes\" branch at \"" +currentNode.questOrAns + "\" question");

return;

}

if (currentNode.noBranch==null) {

System.out.println("Error: Missing \"No\" branch at \"" +currentNode.questOrAns + "\" question");

return;

}

askQuestion(currentNode);

}

private void askQuestion(BinTree currentNode) throws IOException {

System.out.println(currentNode.questOrAns + " (enter \"Yes\" or \"No\")");

String answer = keyboardInput.readLine();

if (answer.equals("Yes")) queryBinTree(currentNode.yesBranch);

else {

if (answer.equals("No")) queryBinTree(currentNode.noBranch);

else {

System.out.println("ERROR: Must answer \"Yes\" or \"No\"");

askQuestion(currentNode);

}

}

}

public void outputBinTree() {

outputBinTree("1",rootNode);

}

private void outputBinTree(String tag, BinTree currentNode) {

if (currentNode == null) return;

System.out.println("[" + tag + "] nodeID = " + currentNode.nodeID +", question/answer = " + currentNode.questOrAns);

outputBinTree(tag + ".1",currentNode.yesBranch);

outputBinTree(tag + ".2",currentNode.noBranch);

}

}

***DecisionTreeApp.java***

import java.io.\*;

class DecisionTreeApp {

static BufferedReader keyboardInput = new BufferedReader(new InputStreamReader(System.in));

static DecisionTree newTree;

public static void main(String[] args) throws IOException {

newTree = new DecisionTree();

generateTree();

System.out.println("\nOUTPUT DECISION TREE");

System.out.println("====================");

newTree.outputBinTree();

queryTree();

}

static void generateTree() {

System.out.println("\nGENERATE DECISION TREE");

System.out.println("======================");

newTree.createRoot(1,"Does animal eat meat?");

newTree.addYesNode(1,2,"Does animal have stripes?");

newTree.addNoNode(1,3,"Does animal have stripes?");

newTree.addYesNode(2,4,"Animal is a Tiger");

newTree.addNoNode(2,5,"Animal is a Leopard");

newTree.addYesNode(3,6,"Animal is a Zebra");

newTree.addNoNode(3,7,"Animal is a Horse");

}

static void queryTree() throws IOException {

System.out.println("\nQUERY DECISION TREE");

System.out.println("===================");

newTree.queryBinTree();

optionToExit();

}

static void optionToExit() throws IOException {

System.out.println("Exit? (enter \"Yes\" or \"No\")");

String answer = keyboardInput.readLine();

if (answer.equals("Yes")) return;

else {

if (answer.equals("No")) queryTree();

else {

System.out.println("ERROR: Must answer \"Yes\" or \"No\"");

optionToExit();

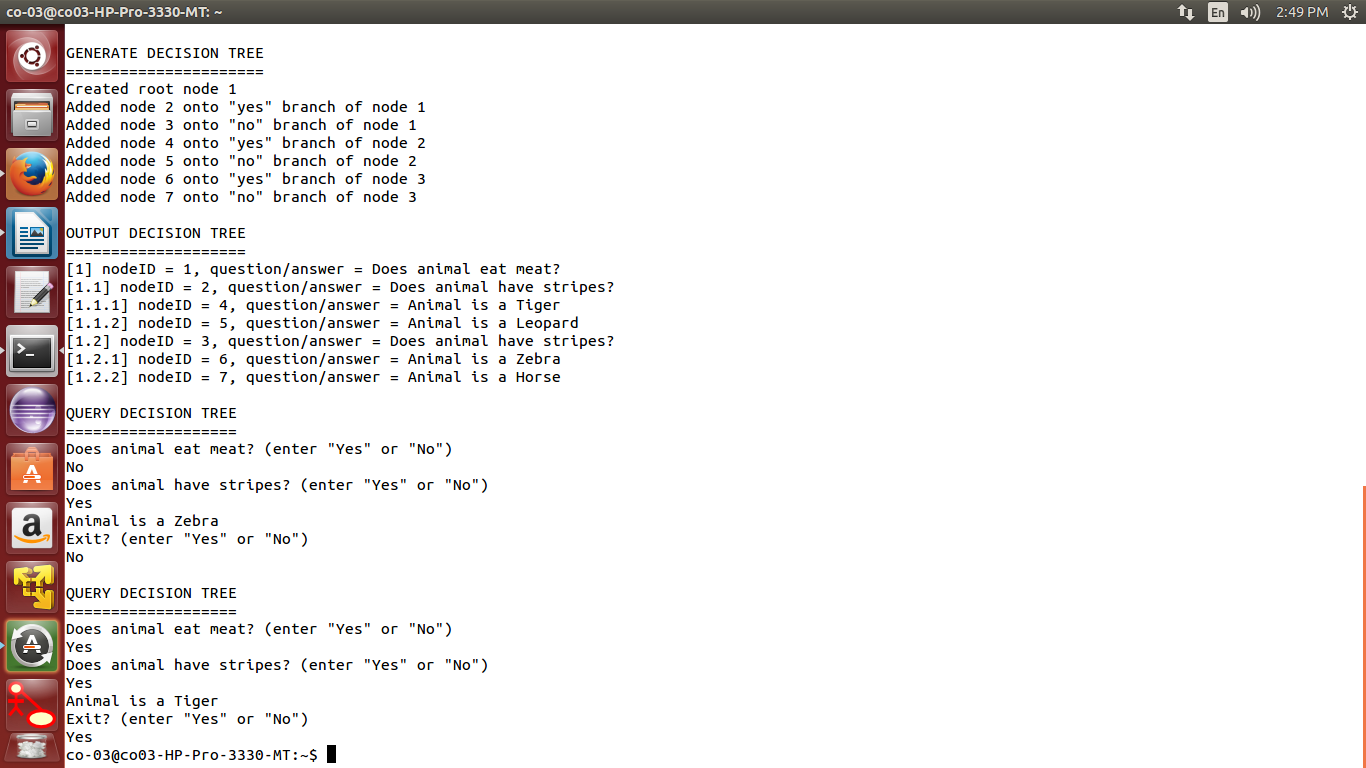
}

}

}

}

**Output:**

****

**Conclusion:**

Thus, Naive Bayes Classifier and Decision Tree has been studied and implemented Successfully.