Data Mining
Home work 08
Machine Learning Start...

Aqeel Labash
Lecturer: Jaak Vilo

29 March 2016

First Question

The quality of classifier as I understood from the page is when we can classify accurately depending on that classifier.By that I mean to have a certain point where it completely separate data into two groups.
Another meaning for quality of classifier might be if the classifier really represent a real case or just something happened with high probability in training dataset.If it's just high probability then depending on that classifier will just make our model worst at predicting with real data or unseen data.

Second Question

For this task I implemented this code in python to get information about the tree.

```
# coding: utf-8
import csv
with open('data.csv') as f:
spam = csv.DictReader(f)
trainset = list(spam)
trainset  = sorted(trainset, key=lambda k: k['Play'])
len(trainset)
class v:
def __init__(self):
self.lst={}

def AddItem(self,values):

if len(values)<=0:
return
print values[0]
if values[0] in self.lst.keys():
self.lst[values[0]].AddItem(values[1:])
else:
self.lst[values[0]]= v()
self.lst[values[0]].AddItem(values[1:])
def printeverything(self,level=0):
#print
for itm in self.lst.keys():
thespace = '----'*level
print(thespace+itm)
self.lst[itm].printeverything(level=level+1)
class Core:
def __init__(self,name=None,occurence=0):
self.name = name
self.occurence =occurence
self.sons={}
def AddItem(self,items):
if len(items)<=0:
return
if items[0] in self.sons.keys():
self.sons[items[0]].occurence+=1
self.sons[items[0]].AddItem(items[1:])
else:
self.sons[items[0]] = Core(name=items[0],occurence=1)
```

```
self.sons[items[0]].AddItem(items[1:])
def printeverything(self,level=0):
#print
thespace = '——'*level
print(thespace+str(self.name)+':'+str(self.occurence))
for itm in self.sons.keys():
self.sons[itm].printeverything(level=level+1)
root = Core()
for i in trainset:
root.AddItem((i['Outlook'],i['Temp'],i['Humidity'],i['Windy'],i['Play']))
root.printeverything()
```

Which result to the following tree :

```
None:0
——Rainy:5
————Mild:3
——————High:2
————————FALSE:1
——————————Yes:1
————————TRUE:1
——————————No:1
——————Normal:1
————————FALSE:1
——————————Yes:1
————Cool:2
——————Normal:2
————————FALSE:1
——————————Yes:1
————————TRUE:1
——————————No:1
——Overcast:5
————Hot:2
——————High:1
————————FALSE:1
——————————Yes:1
——————Normal:1
————————FALSE:1
——————————Yes:1
————Mild:1
——————High:1
————————TRUE:1
——————————Yes:1
————Cool:2
——————High:1
————————FALSE:1
——————————No:1
——————Normal:1
————————TRUE:1
——————————Yes:1
——Sunny:5
————Hot:2
——————High:2
————————TRUE:1
——————————No:1
————————FALSE:1
——————————No:1
————Mild:2
——————High:1
————————FALSE:1
——————————No:1
——————Normal:1
————————TRUE:1
——————————Yes:1
————Cool:1
——————Normal:1
————————FALSE:1
——————————Yes:1
```

Note:more clear version available here.Now we have a preview about the data. We choose the levels depending on the entropy by following these steps (source) :

1. First we calculate the **Entropy** for target:

$$E(S) = \sum_{i=1}^{c} -p_i log_2(p_i)$$

in our case we have

$$(Yes = 9, No = 6) \implies (Yes = 9/15, No = 6/15) \implies (Yes = 0.6, No = 0.4)$$

$$E(PlayTennis) = E(0.4, 0.6) = -0.4 * log_2(0.4) - 0.6 * log_2(0.6) = 0.97095059$$

2. Now we calculate the gain for each feature and go greedy to pick which one first.

$$Gain(T, X) = E(T) - E(T, X)$$

I used the following R code to get the tables :

```r
#### Second Question ######
playing = read.csv('data.csv')
table(playing$Play)
table(playing$Outlook, playing$Play)
table(playing$Temp, playing$Play)
table(playing$Humidity, playing$Play)
table(playing$Windy, playing$Play)

```

The tables as following :

Outlook \ Play Tennis	No	Yes
Overcast	1	4
Rainy	2	3
Sunny	3	2

Here I explain how to calculate the gain (After that I'll use automate way).

$$Gain = 0.97 - (P(Overcast) * E(1, 4) + P(Rainy) * E(2, 3) + P(Sunny) * E(3, 2)) =$$

$$0.97 - (1/3 * E(0.2, 0.8) + 1/3 * E(0.4, 0.6) + 1/3 * E(0.6, 0.4)) \simeq$$

$$0.97 - \frac{1}{3}(-0.2*-2.32-0.8*-0.32-0.4*-1.32-0.6*-0.73-0.4*-1.32-0.6*-0.73) = 0.97-0.884 = 0.086$$

To automate this operation I used the following code:

```r
#install.packages("entropy")
library(entropy)
#outlook Gain
table(playing$Outlook, playing$Play)
entropy(c(6,9), unit = "log2")-(5/15*entropy(c(1,4), unit = "log2")+5/15*entropy(c(2,3),
    unit = "log2")+5/15*entropy(c(3,2), unit = "log2"))
#Temp Gain
table(playing$Temp, playing$Play)
entropy(c(6,9), unit = "log2")-(5/15*entropy(c(2,3), unit = "log2")+4/15*entropy(c(2,2),
    unit = "log2")+6/15*entropy(c(2,4), unit = "log2"))
#Humidity Gain
table(playing$Humidity, playing$Play)
entropy(c(6,9), unit = "log2")-(8/15*entropy(c(5,3), unit = "log2")+7/15*entropy(c(1,6),
    unit = "log2"))
#Windy Gain
table(playing$Windy, playing$Play)
entropy(c(6,9), unit = "log2")-(9/15*entropy(c(3,6), unit = "log2")+6/15*entropy(c(3,3),
    unit = "log2"))
```

After the previous code is done we have

Feature	Gain
OutLook	0.0830075
Temp	0.0133154
Humidity	0.1858052
Windy	0.01997309

3. From the previous table we can see that the most gain is if we used **humidity** feature.Now Humidity split our data set into two parts (Y=6,N=1 , Humidity = Normal), (Y=3,N=5,Humidity =High).

Now we continue to build level two, lets decide the best feature to split when **humidity is high or Normal.**

$$Eh(PlayTennis) = Eh(5, 3) = 0.954434, EN(PlayTennis) = EN(1, 6) = 0.5916728$$

Feature	Humidity=Normal	Humidity=High
OutLook	0.1981174	0.3600731
Temp	0.1280853	0.1100731
Windy	0.1981174	0.003228944

From previous table we can go with OutLook as my next classifier.In 3rd level, I only did it for 3 groups because while drawing and calculating other groups are already classified.:

	Rainy.Normal	Rainy.High	Overcast.High
Windy	0.9182958	1	0.2516292
Temp	0.2516292	0	0.9182958

From the previous table we can see that it's best to classify with feature "windy" for branchings (Rainy.Normal,Rainy.High) and with feature "temp" for branch (overcast high).By that no more branches is required as shown in figure 1

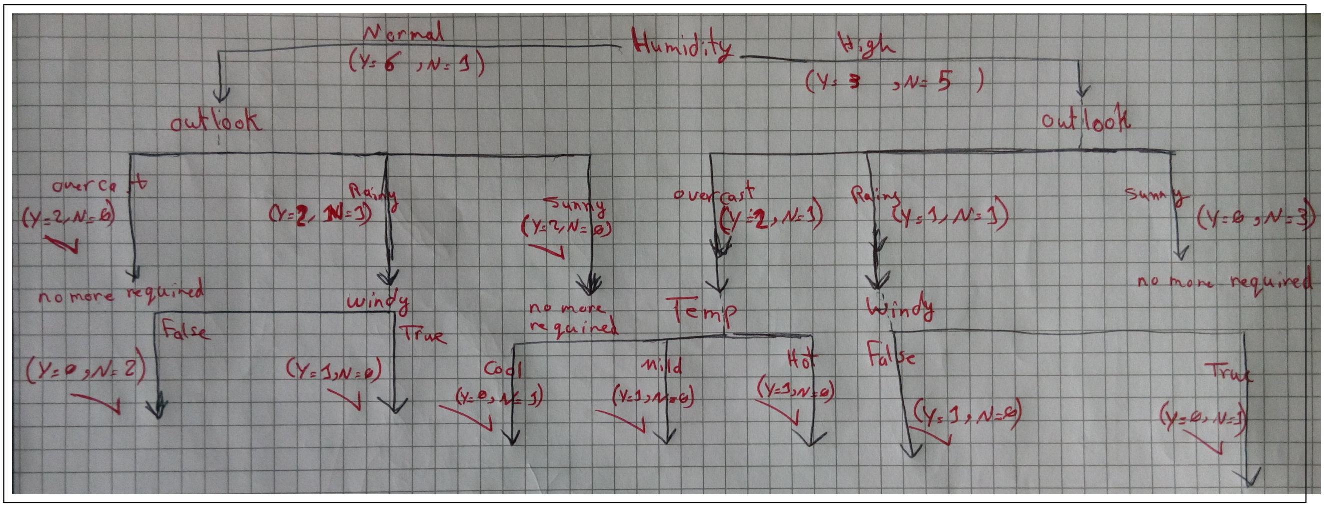

Figure 1: Decision tree for playing tennis data

Following the tree for the scenario (mild, overcast, high humidity and high wind weather) we should play tennis. The previous calculation done by this code :

```
####Level Two####
humidity.h <- subset(playing,playing$Humidity=="High")
humidity.n <- subset(playing,playing$Humidity=="Normal")
#Humidity Normal features gain
#Normal humidity
table(humidity.n$Play)
table(humidity.n$Outlook,humidity.n$Play)
entropy(c(1,6),unit = "log2")-(2/7*entropy(c(0,2),unit = "log2")+3/7*entropy(c(1,2),unit = "log2")+2/7*entropy(c(0,2),unit = "log2"))
table(humidity.n$Temp,humidity.n$Play)
entropy(c(1,6),unit = "log2")-(4/7*entropy(c(1,3),unit = "log2")+1/7*entropy(c(0,1),unit = "log2")+2/7*entropy(c(0,2),unit = "log2"))
table(humidity.n$Windy,humidity.n$Play)
entropy(c(1,6),unit = "log2")-(4/7*entropy(c(0,4),unit = "log2")+3/7*entropy(c(1,2),unit = "log2"))
#High humidity
table(humidity.h$Play)
table(humidity.h$Outlook,humidity.h$Play)
entropy(c(5,3),unit = "log2")-(3/8*entropy(c(1,2),unit = "log2")+2/8*entropy(c(1,1),unit = "log2")+3/8*entropy(c(3,0),unit = "log2"))
table(humidity.h$Temp,humidity.h$Play)
entropy(c(5,3),unit = "log2")-(1/8*entropy(c(1,0),unit = "log2")+3/8*entropy(c(2,1),unit = "log2")+4/8*entropy(c(2,2),unit = "log2"))
table(humidity.h$Windy,humidity.h$Play)
entropy(c(5,3),unit = "log2")-(5/8*entropy(c(3,2),unit = "log2")+3/8*entropy(c(2,1),unit = "log2"))
#### Level 3 #####
table(playing$Outlook)
rainy.normal<- subset(playing,playing$Humidity=="Normal" & playing$Outlook=="Rainy")
rainy.high<- subset(playing,playing$Humidity=="High" & playing$Outlook=="Rainy")
overcast.high<- subset(playing,playing$Humidity=="High" & playing$Outlook=="Overcast")
#Rainy Normal branch
table(rainy.normal$Play)
table(rainy.normal$Windy,rainy.normal$Play)
entropy(c(1,2),unit = "log2")-(2/3*entropy(c(0,2),unit = "log2")+1/3*entropy(c(1,0),unit = "
```

```
log2"))
table(rainy.normal$Temp,rainy.normal$Play)
entropy(c(1,2),unit = "log2")-(2/3*entropy(c(1,1),unit = "log2")+1/3*entropy(c(0,1),unit = "log2"))
#Rainy High branch
table(rainy.high$Play)
table(rainy.high$Windy,rainy.high$Play)
entropy(c(1,1),unit = "log2")-(1/2*entropy(c(0,1),unit = "log2")+1/2*entropy(c(1,0),unit = "log2"))
table(rainy.high$Temp,rainy.high$Play)
entropy(c(1,1),unit = "log2")-(entropy(c(1,1),unit = "log2"))
#overcast high branch
table(overcast.high$Play)
table(overcast.high$Windy,overcast.high$Play)
entropy(c(1,2),unit = "log2")-(2/3*entropy(c(1,1),unit = "log2")+1/3*entropy(c(0,1),unit = "log2"))
table(overcast.high$Temp,overcast.high$Play)
entropy(c(1,2),unit = "log2")-(1/3*entropy(c(1,0),unit = "log2")+1/3*entropy(c(0,1),unit = "log2")+1/3*entropy(c(0,1),unit = "log2"))
```

Third Question

For this task I used R with rpart to generate the decision tree, and here is the code :

```
png('tree.png',width = 1600,height = 800)
plot(result, uniform=TRUE,
main="Classification Tree for Cars")
text(result, use.n=TRUE, all=TRUE, cex=.8)
dev.off()
```

From the previous code we get the following tree :

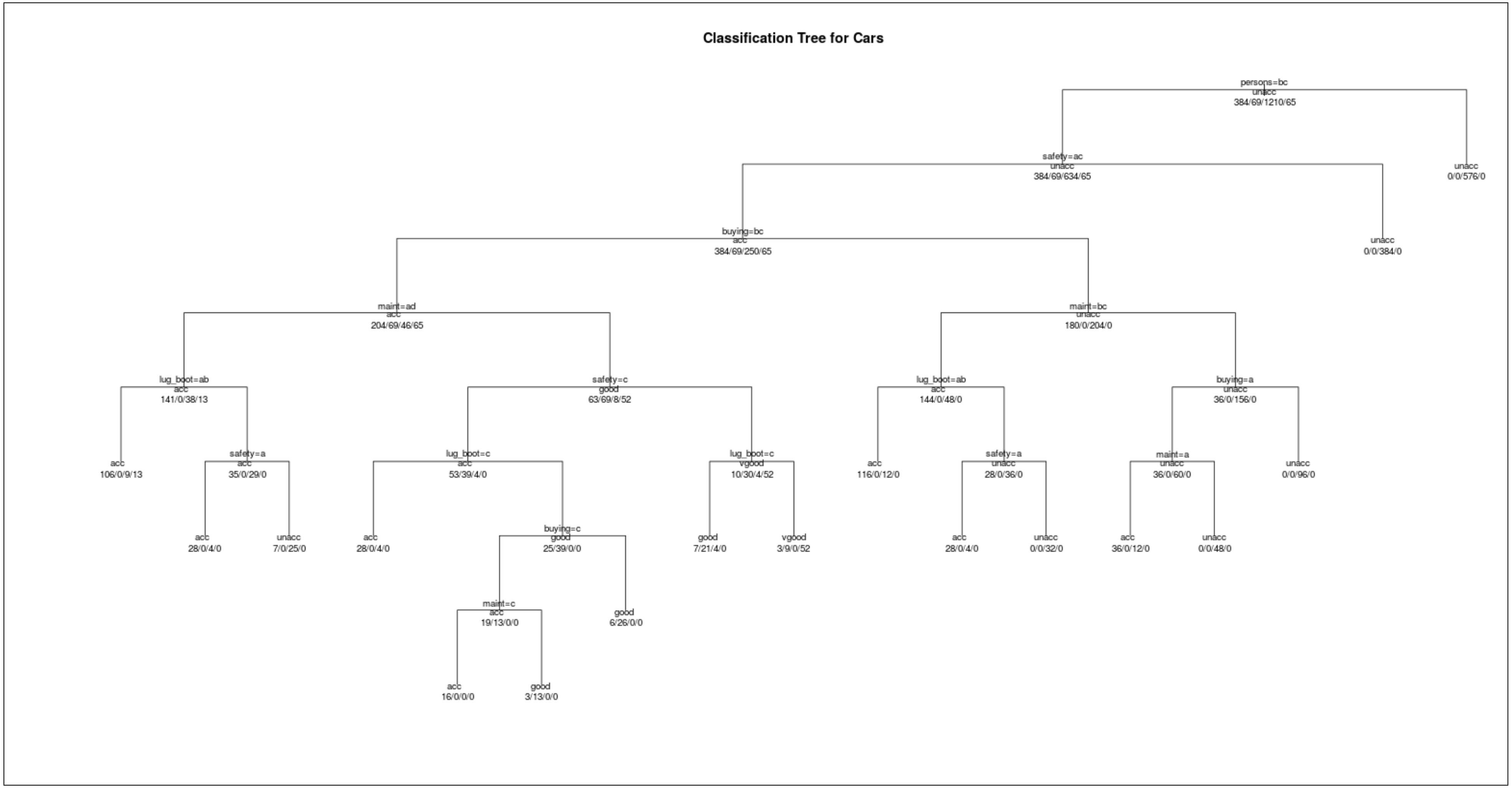

Figure 2: Tree of decision tree classification

Under each node in the tree we can see 4 numbers which represent the number of objects in each category at the following order (acc,good,unacc,vgood). After that I used this code to get rules list :

```
library(arules)
rules = apriori(cars )
inspect(rules)
```

	lhs	rhs	support	confidence	lift
1	buying=vhigh	acceptability=unacc	0.2083333	0.8333333	1.190083
2	maint=vhigh	acceptability=unacc	0.2083333	0.8333333	1.190083
3	safety=low	acceptability=unacc	0.3333333	1.0	1.428099
4	persons=2	acceptability=unacc	0.3333333	1.0	1.428099
5	lug_boot=big,safety=low	acceptability=unacc	0.1111111	1.0	1.428099
6	persons=2,lug_boot=big	acceptability=unacc	0.1111111	1.0	1.428099
7	persons=4,safety=low	acceptability=unacc	0.1111111	1.0	1.428099
8	persons=more,safety=low	acceptability=unacc	0.1111111	1.0	1.428099
9	lug_boot=med,safety=low	acceptability=unacc	0.1111111	1.0	1.428099
10	persons=2,lug_boot=med	acceptability=unacc	0.1111111	1.0	1.428099
11	persons=2,safety=high	acceptability=unacc	0.1111111	1.0	1.428099
12	persons=2,safety=med	acceptability=unacc	0.1111111	1.0	1.428099
13	lug_boot=small,safety=low	acceptability=unacc	0.1111111	1.0	1.428099
14	persons=2,safety=low	acceptability=unacc	0.1111111	1.0	1.428099
15	persons=2,lug_boot=small	acceptability=unacc	0.1111111	1.0	1.428099

For comparison I think the association rules some how looks like sub category of decision tree.Because we can recreate the tree as an association rules.Another thing, I think build a tree of association rules may help as well in shortening the rules number. In the rules table row 4 is the right branch of the root node in the tree.

Fourth Question

We already find the decision tree in the third question, but to find Naive Bayes I used the following code :

```
########## Fourth Question #############
#install.packages("klaR")
library(klaR)
nbayestree<- NaiveBayes(acceptability~.,data = cars)
summary(nbayestree)
nbayestree$tables$buying
nbayestree$tables$maint
nbayestree$tables$doors
nbayestree$tables$persons
nbayestree$tables$lug_boot
nbayestree$tables$safety
```

Using the previous tool generate 6 tables, each table has the acceptability as rows and categories as columns and the table filled with probabilities.Here is the tables :

Acceptability probability with buying categories

grouping	high	low	med	vhigh
acc	0.2812500	0.2317708	0.2994792	0.1875000
good	0.0000000	0.6666667	0.3333333	0.0000000
unacc	0.2677686	0.2132231	0.2214876	0.2975207
vgood	0.0000000	0.6000000	0.4000000	0.0000000

Acceptability probability with maint categories

grouping	high	low	med	vhigh
acc	0.2734375	0.2395833	0.2994792	0.1875000
good	0.0000000	0.6666667	0.3333333	0.0000000
unacc	0.2595041	0.2214876	0.2214876	0.2975207
vgood	0.2000000	0.4000000	0.4000000	0.0000000

Acceptability probability with doors categories

grouping	2	3	4	5more
acc	0.2109375	0.2578125	0.2656250	0.2656250
good	0.2173913	0.2608696	0.2608696	0.2608696
unacc	0.2694215	0.2479339	0.2413223	0.2413223
vgood	0.1538462	0.2307692	0.3076923	0.3076923

Acceptability probability with persons categories

grouping	2	4	more
acc	0.0000000	0.5156250	0.4843750
good	0.0000000	0.5217391	0.4782609
unacc	0.4760331	0.2578512	0.2661157
vgood	0.0000000	0.4615385	0.5384615

Acceptability probability with lug_boot categories

grouping	big	med	small
acc	0.3750000	0.3515625	0.2734375
good	0.3478261	0.3478261	0.3043478
unacc	0.3041322	0.3239669	0.3719008
vgood	0.6153846	0.3846154	0.0000000

Acceptability probability with safety categories

grouping	high	low	med
acc	0.5312500	0.0000000	0.4687500
good	0.4347826	0.0000000	0.5652174
unacc	0.2289256	0.4760331	0.2950413
vgood	1.0000000	0.0000000	0.0000000

Up to this point to be able to judge maybe it'll need a long staring at those table, but I think to be able to judge for real is to put the models to the test and check which one predict better with higher accuracy. Now for 10 folds I'll divide the data 90 %for training-10 % for testing.To be able to compare Naive Bayes with decision tree I'll use the same data for both (first pick them randomly then use them).Here is the code I used for this task :

```
##Testing.
#install.packages("ROCR")
#install.packages("getopt")
library(gplots)
library(ROCR)
library(e1071)
library(getopt)
library(rpart)
Compare<-function()
{
bcmVector<- (c(OverallAccuracy=0,recallAcc=0,recallGood=0,recallunacc=0,recallVgood=0,precisionAcc=0,
precisionGood=0,precisionunacc=0,precisionVgood=0))
dtcmVector<- (c(OverallAccuracy=0,recallAcc=0,recallGood=0,recallunacc=0,recallVgood=0,precisionAcc=0,
precisionGood=0,precisionunacc=0,precisionVgood=0))
for(i in seq_len(10))
{
# 1728 * 0.9 ~ 1556 , 1728*0.1~172
#Shuffle the list To use it later
tmp<-cars[sample(nrow(cars)),]
nbp<-predict(NaiveBayes(acceptability~.,data = tmp[c(1:1556),]),tmp[c(1557:1728),])
dtreep<-predict(rpart(acceptability~.,data = tmp[c(1:1556),]),newdata = tmp[c(1557:1728),],type = 'class')
bcm<-table(nbp$class,tmp[c(1557:1728),]$acceptability)
dtcm<-table(dtreep,tmp[c(1557:1728),]$acceptability)
#print (bcm)
bcmVector<-bcmVector +c(OverallAccuracy=sum(diag(bcm))/sum(bcm),recallAcc=bcm[1,1]/sum(bcm[1,]),
recallGood=bcm[2,2]/sum(bcm[2,]),recallunacc=bcm[3,3]/sum(bcm[3,]),
recallVgood=bcm[4,4]/sum(bcm[4,]),precisionAcc=bcm[1,1]/sum(bcm[,1]),
precisionGood=bcm[2,2]/sum(bcm[,2]),precisionunacc=bcm[3,3]/sum(bcm[,3]),
precisionVgood=bcm[4,4]/sum(bcm[,4]))
#print (dtcm)
dtcmVector<-dtcmVector+c(OverallAccuracy=sum(diag(dtcm))/sum(dtcm),recallAcc=dtcm[1,1]/sum(dtcm[1,]),
recallGood=dtcm[2,2]/sum(dtcm[2,]),recallunacc=dtcm[3,3]/sum(dtcm[3,]),
recallVgood=dtcm[4,4]/sum(dtcm[4,]),precisionAcc=dtcm[1,1]/sum(dtcm[,1]),
precisionGood=dtcm[2,2]/sum(dtcm[,2]),precisionunacc=dtcm[3,3]/sum(dtcm[,3]),
precisionVgood=dtcm[4,4]/sum(dtcm[,4]))
}
print(bcmVector/10)
print (dtcmVector/10)
}
```

The result would for confusion matrix will look like this :

	acc	good	unacc	vgood
acc	26	4	5	4
good	2	3	1	0
unacc	9	0	114	0
vgood	0	1	0	3

(I just put a real example of output data , didn't think that we have to put the output for 10 X 2, folds X ways(Naive Bayes,Decision Tree)) As a reminder

$$Accuracy = \frac{TP + TN}{Totall}, Precision = \frac{TP}{TP + FP}, Recall = \frac{TP}{TP + FN}$$

The previous function will output the average measurements for 10 folds for Naive Bayes , decision tree table. and the output as following :
Naive Bayes

Accuracy	recallAcc	recallGood	recallunacc	recallVgood	precisionAcc	precisionGood	precisionunacc	precisionVgood
0.8581395	0.6705130	0.6916667	0.9178775	NaN	0.6936558	0.3139791	0.9592180	0.5475000

Decision tree model

OverallAccuracy	recallAcc	recallGood	recallunacc	recallVgood	precisionAcc	precisionGood	precisionunacc	precisionVgood
0.9366279	0.8489693	0.7262748	0.9881618	0.7643146	0.8967532	0.8830556	0.9577577	0.7975000

From the previous two tables now I can say that in our case the decision tree is better model than Naive Bayes for this data.

Fifth Question

For this question I used the following code in R :

```r
############## 5th Question #############
rm(list=ls())
setwd('/home/aqeel/Study/DM/HW08/')
titanic <- read.csv('titanic.txt')
library(klaR)
library(gplots)
library(ROCR)
library(arules)
library(rpart)
#Shuffle the list To use it later
tmp<-titanic[sample(nrow(titanic)),]
nbmodel<-NaiveBayes(Survived~.,data = tmp[c(1:1981),])
print(nbmodel$tables)
dtmodel<-rpart(Survived~.,data = tmp[c(1:1981),])
png('treetitanic.png',width = 1600,height = 800)
plot(dtmodel, uniform=TRUE,
main="Classification Tree for Titanic survivres")
text(dtmodel, use.n=TRUE, all=TRUE, cex=.8)
dev.off()

therules<- apriori(tmp[c(1:1981),],parameter = list(supp=0.05,conf=0.7),
appearance = list(rhs=c("Survived=Yes","Survived=No"),default="lhs"))
print (inspect(therules))
```

From the previous code we get the following tables :

	lhs	rhs	support	confidence	lift
1	Sex=Female	Survived=Yes	0.15648662	0.7434053	2.272663
2	Class=3rd	Survived=No	0.23220596	0.7419355	1.102606
3	Class=Crew	Survived=No	0.30792529	0.7540173	1.120561
4	Sex=Male	Survived=No	0.61887935	0.7838875	1.164952
5	Class=2nd,Sex=Male	Survived=No	0.07016658	0.8633540	1.283049
6	Class=1st,Sex=Female	Survived=Yes	0.06208985	0.9685039	2.960812
7	Sex=Female,Age=Adult	Survived=Yes	0.14336194	0.7513228	2.296868
8	Class=3rd,Sex=Male	Survived=No	0.18727915	0.8244444	1.225225
9	Class=3rd,Age=Adult	Survived=No	0.21100454	0.7572464	1.125360
10	Class=Crew,Sex=Male	Survived=No	0.30742049	0.7728426	1.148538
11	Class=Crew,Age=Adult	Survived=No	0.30792529	0.7540173	1.120561
12	Sex=Male,Age=Adult	Survived=No	0.60424028	0.7937666	1.179634
13	Class=2nd,Sex=Male,Age=Adult	Survived=No	0.07016658	0.9144737	1.359019
14	Class=1st,Sex=Female,Age=Adult	Survived=Yes	0.06208985	0.9685039	2.960812
15	Class=3rd,Sex=Male,Age=Adult	Survived=No	0.17264008	0.8382353	1.245720
16	Class=Crew,Sex=Male,Age=Adult	Survived=No	0.30742049	0.7728426	1.148538

```
$Class
var
grouping        1st       2nd       3rd      Crew
No  0.08177044 0.1065266 0.3555889 0.4561140
Yes 0.29320988 0.1620370 0.2546296 0.2901235

$Sex
var
grouping     Female      Male
No  0.08627157 0.9137284
Yes 0.48765432 0.5123457

$Age
var
grouping     Adult      Child
No  0.9639910 0.03600900
Yes 0.9228395 0.07716049
```

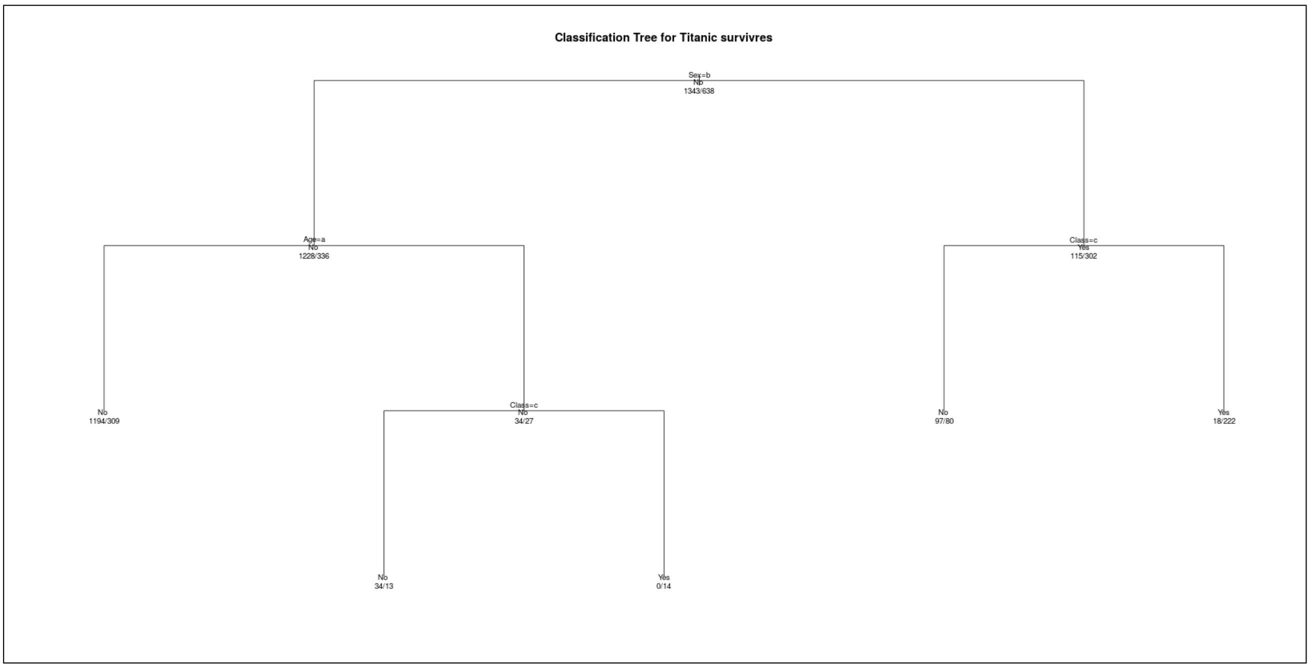

Figure 3: Shows the decision tree for titanic 0.9 of the data1

From the previous tables and figures we can notice for example if we have the following : Crew,Adult,Male. All previous method will classify as not survived.
Actually I tried many example for for this data they reach to the same decision.
To characterize and interpret those methods : I think decision tree with the long use it might get stuck and talk long time to change. While naive Bayes will change by each record (enhance the probabilities values).I think apriori algorithm some how close to decision table and it'll take long time to converge in case changes happened in the environment.

Sixth Question

To minimize over fitting there is many ways.

1. Add noise layer so we can decrease the over fitting.source
2. Use different data for training and test each time.
3. Idea: we train the model till the moment where the train accuracy increase and the test accuracy decreased.

I used the Connect4 dataset , I trained it on 80 % and tested on 20%. I used this code to generate the confusion matrix :

```
########## Sixth Question ##########
rm(list=ls())
connect4<-read.csv('connect-4.data',header = FALSE,sep = ",")
colnames(connect4)<-c('a1','a2','a3','a4','a5','a6','b1','b2','b3','b4','b5','b6','c1','c2','c3','c4','c5','c6','d1','d2','d3','d4','d5','d6','e1','e2','e3','e4','e5','e6','f1','f2','f3','f4','f5','f6','g1','g2','g3','g4','g5','g6','result')

##Over fitting example ##
connect4<-connect4[sample(nrow(connect4)),]
# 67557*0.8 ~ 54046 , 67557*0.2~13511
nbmodel<-NaiveBayes(result~.,connect4[c(0:54046),])
nbp<-predict(nbmodel,connect4[c(54047:67557),])
table(nbp$class,connect4[c(54047:67557),]$result)
```

And here is the confusion matrix.

	draw	loss	win
draw	88	112	143
loss	157	1372	443
win	1079	1778	8339

The previous table Over all Accuracy was 0.7252609 , where the cars dataset over all accuracy was : 0.8581395.
Note:All R,python,ipython,tex,pdf,etc.. files exist on github

E.O.F