

CSC 522 : Automated Learning and Data Analysis

Homework 2

Roopak Venkatakrishnan - rvenkat7@ncsu.edu

February 5, 2013

1 Question 1

Devise a training set for use as input to the ID3 algorithm in order to produce a decision tree with a particular structure. We're providing the required target decision tree that should be induced using your training set. In other words, instead of the usual tree induction learning problem where we're given training data and a tree learner and we see what the program outputs, for this assignment you are given the desired output, and must design the input to produce it. To test your candidate training sets, you may use either Weka, Matlab or R.

Construct a training set having the appropriate format for the classifier you are using, which, when used with the classifier, comes as close as you can to outputting the given decision tree. The decision tree:

```
tear-prod-rate=(normal)
| astigmatism=(yes)
| | spectacle-prescrip=(hypermetrope): none
| | spectacle-prescrip!=(hypermetrope): hard
| astigmatism!=(yes)
| | age=(presbyopic): soft
| | age!=(presbyopic): soft
tear-prod-rate!=(normal): none
```

“Outputting the tree” means the tree structure is the same as above, with the same decision nodes in the same places, etc. Generating just any tree, no matter how accurate, is not sufficient. There must be no missing data in your training set. In other words, each example must have values for all attributes. For each example, the training set should provide values for all the following attributes, including the attributes that do not appear in the target tree. The list of attributes is given in Table 1.

Deliverables: Turn in the following:

1. A description of how you arrived at the dataset. An answer along the lines of I got lucky or this was simple trial and error are not acceptable.
2. A description of how to load the dataset in either Matlab, Weka or R, and the toolbox or package used.
3. A text file with the dataset you generated.

Attributes	Values
age	{young,pre-presbyopic,presbyopic}
spectacle-prescrip	{hypermetrope,myope}
astigmatism	{yes,no}
tear-prod-rate	{reduced,normal}
contact-lenses (Target Class)	{none,soft,hard}

Table 1: Table with attributes and values

Arriving At the dataset

We notice that tear-prod-rate is the root of the tree. So we try to center our values in such a way that they revolve around tear-prod-rate. The first branch says that if tear-prod-rate is reduced then the lenses required are none. We try and depict this in such a way that contact-lenses depends on nothing else but the tear-prod-rate. To achieve this we add all possible combinations of astigmatism, spectacle-prescrip and age. This way no pattern is formed here and looking at the data the ID3 algorithm is forced to say when tear-prod-rate is reduced then contact lenses is none. If we do not account for all combinations some other pattern may for unbeknown to us.

Next we work on the other branch of the tree, where tear-prod-rate is normal. Now we see that astigmatism is the next branch. Let us consider the case where astigmatism is yes. We notice that if spectacle-prescrip=hypermetrope the contact-lense is none otherwise it is hard. So we work combinations in such way that we keep tear-prod-rate=normal, astigmatism=yes and spectacle-prescrip=hypermetrope we set ccontact-lense to none for all possible values of age. This ensures that age doesnt contribute to a pattern at this point. Next we consider tear-prod-rate=normal, astigmatism=yes and spectacle-prescrip=myope and set contact-lense=hard for all possible values of age. This makes sue that at this point the tree is split only the way we want it to be split and random patterns dont develop without our notice.

Next we work on the branch where astigmatism is no. we consider tear-prod-rate=normal, astigmatism=no and age=presbyopic. For this we set spectacle-prescrip to all possible values and set the contact-lense=soft. This way the only pattern is that of tear-prod-rate-¿astigmatism-¿age(presbyopic). For the other two values of age (young and pre-presbyopic) we set all possible values of spectacle-prescrip for each of them.

We have thus constructed an ID3 algorithm in such a way that it is forced to go the way we want it to go. Because all other cases would end up with a tree of much higher length.

The exact tree generated by my algorithm is as shown below.

```
tear-prod-rate = normal
| astigmatism = yes
| | spectacle-prescrip = hypermetrope: none
| | spectacle-prescrip = myope: hard
| astigmatism = no
| | age = pre-presbyopic: none
| | age = presbyopic: soft
| | age = young: none
tear-prod-rate = reduced: none
```

Loading My Dataset This dataset was designed to be used with Weka. The steps are as follows.

- Run Weka
- Click on Explorer
- Navigate to the “Preprocess” tab.
- Click Open-File and navigate to the file. (file-type csv)
- Go to the classify tab, choos filter as Id3
- Make sure contact-lenses is selected in the drop down (Nom)
- leave the Test options unchanged. and click start
- Result is produced on the classifier output box in the same tab.

2 Question 2

For this exercise, we will use the Balloons dataset (<http://archive.ics.uci.edu/ml/datasets/Balloons>). Specifically, we will use the yellow-small+adult-stretch data. (<http://archive.ics.uci.edu/ml/machine-learning-databases/balloons/yellow-small+adult-stretch.data>).

The data has 16 entries each four attributes, all of which are nominal. Each entry belongs to one of the two classes T, F. Complete the following tasks.

1. Compute the tree using Gini index. Show all necessary calculations.

```

> vals<-read.csv(file.choose())
> names(vals)<- c("color","size","act","age","inflated")
> findgini <- function (vals, inflated) {
+
+
+   unq <- unique(vals)
+   gini <- c()
+   df <- data.frame(v=vals,inf=inflated)
+   gini_final<-0
+   for(i in 1:length(unq))
+   {
+     no_type_i <- length(vals[which(vals==unq[i])])
+     no_true <- length(which(inflated[which(vals==unq[i])]))
+     no_false <- no_type_i - no_true
+     tmp = 1 -(no_true/no_type_i)^2 -(no_false/no_type_i)^2
+     tlength <-length(vals)
+     gini_final<-gini_final + ((no_type_i/tlength)*tmp)
+   }
+   print(paste("The Gini is ",gini_final))
+
+
+ }
> findgini(vals$color,vals$inflated)
[1] "The Gini is  0.421875"
> findgini(vals$size,vals$inflated)
[1] "The Gini is  0.421875"
> findgini(vals$act,vals$inflated)
[1] "The Gini is  0.421875"
> findgini(vals$age,vals$inflated)
[1] "The Gini is  0.421875"
> findgini(vals$size[which(vals$color=="YELLOW")],
+ vals$inflated[which(vals$color=="YELLOW")])
[1] "The Gini is  0.1875"
> findgini(vals$act[which(vals$color=="YELLOW")],
+ vals$inflated[which(vals$color=="YELLOW")])
[1] "The Gini is  0.4375"
> findgini(vals$age[which(vals$color=="YELLOW")],
+ vals$inflated[which(vals$color=="YELLOW")])
[1] "The Gini is  0.4375"
> findgini(vals$size[which(vals$color=="PURPLE")],
+ vals$inflated[which(vals$color=="PURPLE")])
[1] "The Gini is  0.375"
> findgini(vals$act[which(vals$color=="PURPLE")],
+ vals$inflated[which(vals$color=="PURPLE")])
[1] "The Gini is  0.25"
> findgini(vals$age[which(vals$color=="PURPLE")],
+ vals$inflated[which(vals$color=="PURPLE")])
[1] "The Gini is  0.25"
> findgini(vals$act[intersect(which(vals$color=="YELLOW"),
+ which(vals$size=="SMALL"))],
+ vals$inflated[intersect(which(vals$color=="YELLOW"),
+ which(vals$size=="SMALL"))])
[1] "The Gini is  0"
> findgini(vals$act[intersect(which(vals$color=="YELLOW"),
+ which(vals$size=="LARGE"))],
+ vals$inflated[intersect(which(vals$color=="YELLOW"),
+ which(vals$size=="LARGE"))])

```

```

[1] "The Gini is 0.25"
> findgini(vals$age[intersect(which(vals$color=="YELLOW"),
+ which(vals$size=="LARGE"))],
+ vals$inflated[intersect(which(vals$color=="YELLOW"),
+ which(vals$size=="LARGE"))])
[1] "The Gini is 0.25"
> findgini(vals$age[intersect(which(vals$color=="YELLOW"),
+ which(vals$size=="SMALL"))],
+ vals$inflated[intersect(which(vals$color=="YELLOW"),
+ which(vals$size=="SMALL"))])
[1] "The Gini is 0"
> findgini(vals$age[intersect(which(vals$color=="YELLOW"),
+ which(vals$size=="LARGE"))],
+ vals$inflated[intersect(which(vals$color=="YELLOW"),
+ which(vals$size=="LARGE"))])
[1] "The Gini is 0.25"
> findgini(vals$size[intersect(which(vals$color=="PURPLE"),
+ which(vals$act=="STRETCH"))],
+ vals$inflated[intersect(which(vals$color=="PURPLE"),
+ which(vals$act=="STRETCH"))])
[1] "The Gini is 0.5"
> findgini(vals$age[intersect(which(vals$color=="PURPLE"),
+ which(vals$act=="STRETCH"))],
+ vals$inflated[intersect(which(vals$color=="PURPLE"),
+ which(vals$act=="STRETCH"))])
[1] "The Gini is 0"
> findgini(vals$age[intersect(which(vals$color=="PURPLE"),
+ which(vals$act=="DIP"))],
+ vals$inflated[intersect(which(vals$color=="PURPLE"),
+ which(vals$act=="DIP"))])
[1] "The Gini is 0"
> findgini(vals$act[intersect(which(vals$color=="PURPLE"),
+ which(vals$act=="DIP"))],
+ vals$inflated[intersect(which(vals$color=="PURPLE"),
+ which(vals$act=="DIP"))])
[1] "The Gini is 0"

```

Let us use the above function and the data to construct a decision tree. This function computes the Gini value given the corresponding values and the respective output expected.

First we see where we get the lowest $GINI_{split}$ trying all columns. In this case we notice that all the columns give us a value of $GINI_{split} = 0.421875$. This value is calculated by the above written function.

Let us run through how the function calculates the $GINI_{split}$. First it finds GINI for each of the different values of the column passed. Let us consider as an example. It has two different values, YELLOW and PURPLE. Let us now calculate $GINI(\text{color}=\text{YELLOW})$.

This can be found as

$$1 - \left(\frac{5}{8}\right)^2 - \left(\frac{3}{8}\right)^2$$

Similarly we find the GINI value for PURPLE

Combining both the values given we find the value for $GINI_{split}(\text{Color})$

Since our function does the above steps we can use the value it gives us and move forward quicker.

Since all seem to have the same level of impurity in the first level, let us just pick color and move on. Now we try and see what happens for $\text{color} = \text{YELLOW}$ and $\text{size}, \text{act}, \text{age}$. We see the following.

Given COLOR=YELLOW, $GINI_{split}$ is given to be

$$\begin{cases} 0.1875 & size \\ 0.4375 & act \\ 0.4375 & age \end{cases}$$

Given COLOR =PURPLE, $GINI_{split}$ is given to be

$$\begin{cases} 0.375 & size \\ 0.25 & act \\ 0.25 & age \end{cases}$$

Given the above we can see the best partition would be along size for COLOR=YELLOW and either of act/age for COLOR=PURPLE. Let us choose to proceed with act for PURPLE

Given COLOR=YELLOW, SIZE=LARGE, $GINI_{split}$ is found to be

$$\begin{cases} 0.25 & act \\ 0.25 & age \end{cases}$$

Thus since both are identical we can choose either and proceed. From this point onward we find the GINI values to be 0. So we proceed to draw the tree.

Given COLOR=PURPLE,act=STRETCH, $GINI_{split}$ is found to be

$$\begin{cases} 0.5 & size \\ 0.0 & age \end{cases}$$

Here we choose age and finalize the tree.

The final tree we construct looks as follows.

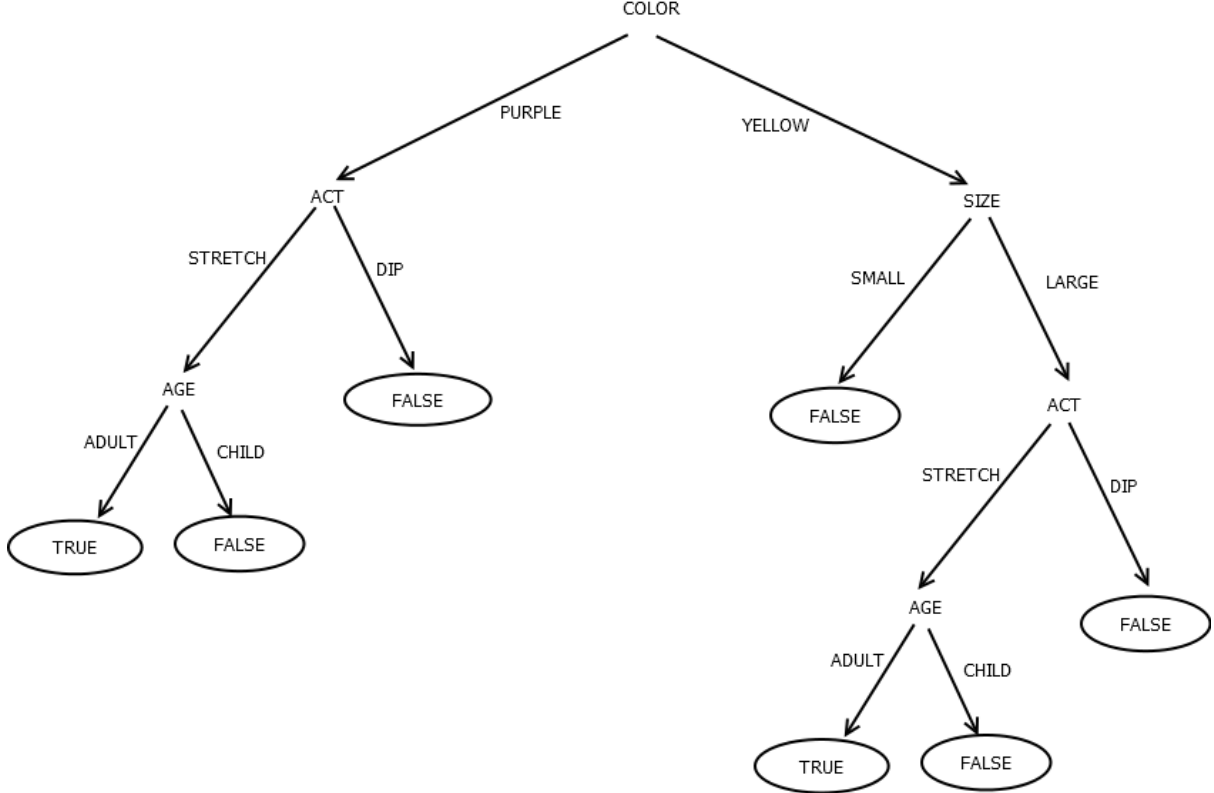


Figure 1: Decision Tree - GINI index

3 Question 3

Classifiers such as decision trees, neural networks, naive bayes, and many others estimate the conditional probability of class membership. For standard two-class classification problems, the goal is to create a decision procedure that minimizes some notion of prediction error. There are many metrics that measure this, but the naive optimal decision for all metrics in this case is:

$$\hat{y} = \begin{cases} 1 & : P(y = 1|X) > 0.5 \\ 0 & : P(y = 1|X) \leq 0.5 \end{cases}$$

where, \hat{y} is the predicted class, ‘0’ or ‘1’, and $P(y = 1|X)$ is the estimated probability of class membership provided by the classifier.

This is just a technical way of saying that we will predict the most likely class. This is the optimal rule if the cost of making any decision is the same or if the cost function is “symmetric”. In this problem we will investigate what happens if the cost function is not symmetric. In a non-symmetric cost function, the penalty associated with misclassification can be different for false negative and false positive predictions, depending on the domain. Pretend you have been approached by a large credit card company. They would like for you to help improve their current fraud detection scheme. They have a wealth of data on consumer transactions that they have used to develop a fraud classifier. This classifier takes as input data on the customer requesting the transaction and the specifics of the transaction such as time, location, and many other details. The classifier outputs an estimated probability that the transaction is fraudulent.

However, for the credit card company the cost of every decision is not the same. These costs are summarized in Table 2 below. If the transaction is fraudulent and is not caught, the credit card company loses the entire transaction amount. Similarly, if the company decides to decline a transaction because they think it is fraudulent but it is not, the customer may be upset and close their account or use it less in the future, costing the credit card company money.

	Prediction	Truth	Cost
1.	Predict No Fraud	Transaction Fraudulent	Transaction Amount (T_i)
2.	Predict Fraud	Transaction Legitimate	$10 + 0.05 * T_i$
3.	Fraud or No Fraud	Correct	0

Table 2: Cost of Classification

Using this setup, you will be asked to derive the optimal decision rule and compare it to the naive rule. Download the file `fraud_probs.csv`. This file contains information on 152880 transactions. The 3 columns are the transaction amount, the estimated probability of fraud from the classifier, and whether or not the transaction was later reported to be fraudulent. A ‘1’ in the 3rd columns means there was fraud while a ‘0’ means there was none.