Week 3 Entropy Project

Sandesh Sadalge

Fri, Sept 12, 2014

Dataset

I put the data into my github directory to make things easier. Need to load a package before I can get it from github with this R-Markdown.

```
## Loading required package: bitops
```

```
fileLoc <- "https://raw.githubusercontent.com/spenglerss/MSDAFall2014/master/IS%20607%20Data%20Acquisit
dataset <- read.csv(text = getURL(fileLoc))
```

A) Create the **entropy()** function:

Here is my final function code. Explanations follow.

```
entropy <- function (d)
{
  partitions <- table(d)                          # (1)
  partitions <- partitions / sum(partitions)      # (2)
  partitions <- partitions * log2(partitions)     # (3)
  entropy <- -sum(partitions)                     # (4)
}
```

(1) Determining categorical partitions and their frequencies:

Initially, the hard part for me was coming up with a way to determine a given vector's categorical partitions and their frequencies.

My first thoughts started down the path of either trying to convert to factor or using *unique()* to get the distinct categorical values and perhaps then using a loop to figure out the frequency. I realized this was probably a very *'un-R'* way to do it.

Fortunately, I came across ***table()*** which seemed to do exactly what I was looking for. The code: `partitions <- table(d)` gives you a table with the distinct values & respective frequencies of the input vector, d. As an example, in our dataset, `table(dataset$answer)` returns:

```
##
##   0   1
## 576 424
```

(2) Thanks to the fact that R is a vectorized language, `partitions <- partitions / sum(partitions)` determines the probability of each of the partitions (based on frequency counts).

Again, inr our example on `table(dataset$answer)`, when you do this, you get:

```
##
##     0     1
## 0.576 0.424
```

(3) Similarly, `partitions <- partitions * log2(partitions)` gives you the individual calculation within the summation notation of the Entropy equation, probability * log2(probability).

```
##
##       0       1
## -0.4584 -0.5249
```

(4) And lastly, we just take the negative sum of the terms: `entropy <- -sum(partitions)`

```
## [1] 0.9833
```

B) Create the **infogain()** function

Here's the code:

```
infogain <- function (d, a)
{
  if (all(d==a) == TRUE)                                   # (1)
  {
    infogain <- 0
  } else
  {
    entropy.d <- entropy(d)                                # (2)

    x <- table(a, d)                                       # (3)
    y <- (x / rowSums(x))                                  # (4)
    y <- y * ifelse(is.infinite(log2(y)),0,log2(y))        # (5)
    z <- rowSums(x) / sum(x)                               # (6)
    entropy.a <- sum(z * -1 * rowSums(y))                  # (7)

    infogain <- entropy.d - entropy.a
  }
}
```

(1) `if (all(d==a) == TRUE)` This is one of those pieces of code that one puts in after you've started testing and you realize there's a special case that needs to be taken care of. I explain this much more fully in note 3 in the next function creation, decide(). Basically, if the d and a vectors are the same, the infogain should be 0. You don't 'gain' anything for the same partition, right?

Note on explanations below: I wish I could put together a video or something to explain how I'm calculating the probabilities and entropies using tables but I'll try by best to explain using my limited R-Markdown abilities & some pictures I made in Excel:

Say you have a vector **d** with partitions ***d.i*** where ***i = 1 . . . N*** and another vector **a** with partitions ***a.j*** where ***j = 1 . . . K***. Then, the infogain, **I(d,a) = E(d) - E(d,a)**.

You can get E(d) using the first function created which is what I did in **(2)** using `entropy(d)`.

As a reminder, you get the following entropy for `entropy(dataset$answer)`:

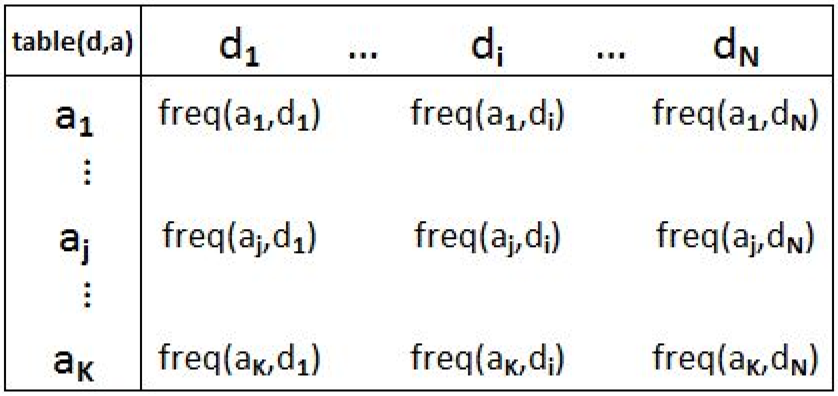
```
## [1] 0.9833
```

But how do you get E(d,a)?

I realized later that I could've created another function called `entropy(d,a)` to do this but I opted to put the code withint the infogain() function instead.

My solution uses *table()* again to create a 2 way table whose column names are *d.i* and row names are *a.j*. the value *(a.j,d.i)* equals the *frequency of the partitions* ***a.j & d.i***

Something like this:



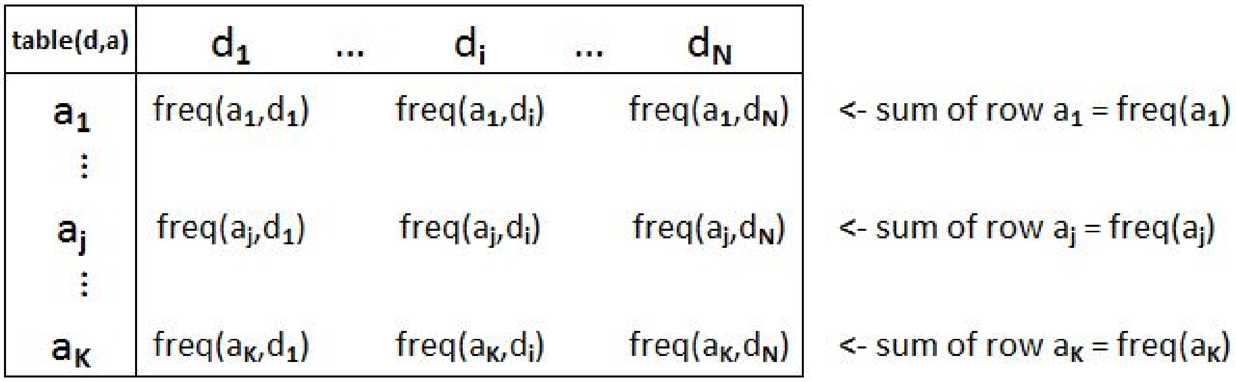
table(d,a)	d_1	...	d_i	...	d_N
a_1	freq(a_1,d_1)		freq(a_1,d_i)		freq(a_1,d_N)
⋮					
a_j	freq(a_j,d_1)		freq(a_j,d_i)		freq(a_j,d_N)
⋮					
a_K	freq(a_K,d_1)		freq(a_K,d_i)		freq(a_K,d_N)

This is what line **(3)** does using `table(a, d)`

To be less abstract, for `d <- dataset$answer` and `a <- dataset$attr1` you get the following table:

```
##
##       0   1
##   0 347 253
##   1 229 171
```

(4) In how I've constructed this table, you will notice that the sum of the rows are the total frequecy of the partitions **a**.*j*:



table(d,a)	d_1	...	d_i	...	d_N	
a_1	freq(a_1,d_1)		freq(a_1,d_i)		freq(a_1,d_N)	<- sum of row a_1 = freq(a_1)
⋮						
a_j	freq(a_j,d_1)		freq(a_j,d_i)		freq(a_j,d_N)	<- sum of row a_j = freq(a_j)
⋮						
a_K	freq(a_K,d_1)		freq(a_K,d_i)		freq(a_K,d_N)	<- sum of row a_K = freq(a_K)

I use this fact to figure out the proabilities P(aj|di) (or is the notation P(di|aj)?) simply by dividing each element by the sum of the row. Or in my code, `(x / rowSums(x))`.

Continuing our example using `dataset$answer` and `dataset$attr1`, you get:

```
##
##          0      1
##   0 0.5783 0.4217
##   1 0.5725 0.4275
```

Notice that the sum for each of the rows is 1.

(5) Now that we have the probabilities, we just multiply by the log base 2 of each one respectively as the entropy equation states.

Here you'll notice that I put a special case using `ifelse` for when log2() is infinity. One case where this will happen is if d and a are the same vector.

Using `dataset$answer` and `dataset$attr1` we get:

```
##
##           0       1
##   0 -0.4569 -0.5253
##   1 -0.4607 -0.5241
```

Notice, after line (5), table x has the frequencies and table y has the component pieces of E(a.j)!! Therefore, `rowSums(y)`, will now give you E(a.i). This fact will be useful in **(7)**.

(6) We need one last thing which is the weights for E(a.j). Since table x still holds the frequencies, the weights are simply `rowSums(x) / sum(x)` which I put into `z`.

(7)* What does `sum(z * -1 * rowSums(y))` do? Well, since `rowSums(y)` is the entropy for E(a.j) and z has the weights for a.j, multiplying the two, taking the negative of it and summing it up will give you the total entropy E(d,a).

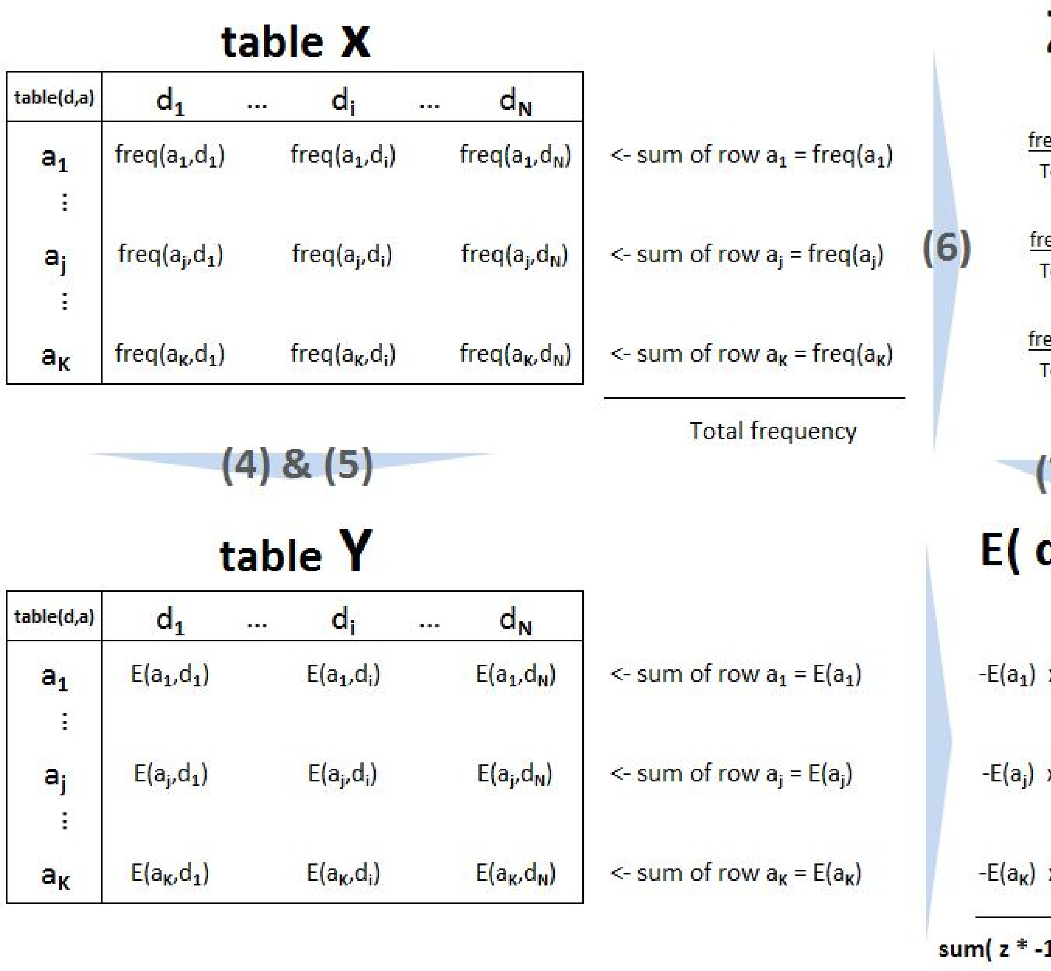


table X

table(d,a)	d_1	...	d_i	...	d_N	
a_1	freq(a_1,d_1)		freq(a_1,d_i)		freq(a_1,d_N)	<- sum of row a_1 = freq(a_1)
⋮						
a_j	freq(a_j,d_1)		freq(a_j,d_i)		freq(a_j,d_N)	<- sum of row a_j = freq(a_j)
⋮						
a_K	freq(a_K,d_1)		freq(a_K,d_i)		freq(a_K,d_N)	<- sum of row a_K = freq(a_K)
						Total frequency

table Y

table(d,a)	d_1	...	d_i	...	d_N	
a_1	E(a_1,d_1)		E(a_1,d_i)		E(a_1,d_N)	<- sum of row a_1 = E(a_1)
⋮						
a_j	E(a_j,d_1)		E(a_j,d_i)		E(a_j,d_N)	<- sum of row a_j = E(a_j)
⋮						
a_K	E(a_K,d_1)		E(a_K,d_i)		E(a_K,d_N)	<- sum of row a_K = E(a_K)

The whole process looks like this:

Examples for (6) & (7):

Entropies E(a.j) or `rowSums(y)`

```
##       0       1
## -0.9822 -0.9848
```

Weights in table **z**:

```
##   0   1
## 0.6 0.4
```

Above two are multiplied & added and negated to get to the final total for E(d,a):

```
## [1] 0.9832
```

Lastly, the Infogain is E(d) - E(d,a) which is:

```
## [1] 2.412e-05
```

Examples asked for in the assignment (apaprently R-Markdown rounds a little differently than the RStudio console):

`infogain(dataset$answer,dataset$attr1)` = 2.4116 × 10-5

`infogain(dataset$answer,dataset$attr2)` = 0.2599

`infogain(dataset$answer,dataset$attr3)` = 0.0024

C) Create the **decide()** function

Here's the code:

```
decide <- function(inputDF, col)
{
  decideDF <- data.frame(colnames=colnames(inputDF))
  n <- length(decideDF$colnames)
  infogain.values <- rep(0, n)
  for (i in 1:n)
  {
    infogain.values[i] <- infogain(inputDF[,col], inputDF[,i])
  }

  decideDF <- data.frame(colnames=decideDF$colnames,infogain=infogain.values)

  decide <- list(max=which.max(infogain.values), gains=decideDF)
}
```

I'm not going to go too deeply into the explanation on this one. I think it's somewhat self-expnatory.

Basically, the function takes a data frame and the reference column to compare information gains from all the other columns against.

After setting up a helper vector and variable, the function simply iterates through each of the columns and calling "infogain(reference column, current column)"."

The function stores both the infogain values and column names in a dataframe called *decideDF*.

When it's gone through all the columns, it returns a list with the first element being the column number with the greatest infogain (using *which.max()* and the actual columns and their respective infogain values.

A few of things to mention:

1. I tried really hard to use one of the *apply()* family of functions to try and do this without the iterations but I kept running into errors. I would love to get that solution.
2. It was after writing this function that I had to go back to the original *infogain()* function I wrote and put in the special case for checking if the vectors are the same.
    - Since I iterate through all the columns, I will always try to do the infogain for the reference vector and itself. Infogain in this case should be 0. (You shouldn't 'gain' anything if you subset back on the same criteria.) What actually happened in my original infogain() function was that E(d) gave you a valid number but when I tried to do entropy(d,d) I got 0. Why? because I put a catch for the infinities associated with log2(0) which made E(d,a) = 0 . Therefore, the infogain = E(d) - E(d,a) was reverting to simply E(d) which was wrong. After puzzling through it, there were two solutions I could see: Either I do not account for the infinities and allow the infogain() function to return NaN or I could put a test in the very begining of the infogain() function to check if the vectors are exactly the same. I chose the latter.

Lastly, let's see if the outputs match the expected ones:

```r
(entropy(dataset$answer))
```

```
## [1] 0.9833
```

```r
(infogain(dataset$answer, dataset$attr1))
```

```
## [1] 2.412e-05
```

```r
(infogain(dataset$answer, dataset$attr2))
```

```
## [1] 0.2599
```

```r
(infogain(dataset$answer, dataset$attr3))
```

```
## [1] 0.002433
```

```
(decide(dataset,4))
```

```
## $max
## [1] 2
##
## $gains
##   colnames  infogain
## 1    attr1 2.412e-05
## 2    attr2 2.599e-01
## 3    attr3 2.433e-03
## 4   answer 0.000e+00
```