Probabilistic programming: a tentative first encounter

This my first ever Julia code :) It is a julia version of this Matlab code I did to try to learn some ideas about probabilistic programming http://www.inferencelab.com/probabilistic-programming-01/ (http://www.inferencelab.com/probabilistic-programming-01/). I've just copy/pasted the text from my blog post, but obviously have updated the code from Matlab to Julia.

In [1]:

```
# compact definition of a coin flip function
flipCoin(p) = rand() < p
```

Out[1]:

```
flipCoin (generic function with 1 method)
```

Generative Model

The example is a Bayesian Network which describes the joint distribution P(flu, TB, cough).

In [2]:

```
function model()
    flu = flipCoin(0.2)
    TB = flipCoin(0.1)
    if flu | TB
        cough = flipCoin(0.8)
    else
        cough = flipCoin(0.1)
    end
    return Dict("flu" =>flu, "TB"=>TB, "cough"=>cough)
end
```

Out[2]:

```
model (generic function with 1 method)
```

When we call the `model()` function the generative model will run once and the state will be returned in a `dict`. We can run the model many times and look at the distributions of states of the world according to our generative model.

Assessing conditional probabilities, or making ‘queries’

Another thing we might want to do is to evaluate conditional probability distributions. For example, we might want to know the probability of having the flu given we observe a cough, $P(flu|cough)$. The way how we can do this is conceptually very easy to grasp, especially with the use of a rejection sampler.

What we do is to run the model many times. Each time the model will have a state. We are only interested in occasions where the particular states we specify as observed (i.e. cough=true) occur. So what we store is the state of the model on all occasions where the condition we specify is true. Here is the code for a rejection sampler:

In [3]:

```
function rejectionQuery(model, queryFunc, conditionFunc)
    queryReturn = Bool[]
    while length(queryReturn) < 10^5
        modelState = model()
        if conditionFunc(modelState)
            push!(queryReturn, queryFunc(modelState))
            #queryReturn[end+1] = queryFunc(modelState)
        end
    end
    return queryReturn
end
```

Out[3]:

```
rejectionQuery (generic function with 1 method)
```

We should remember that the rejection sampler will be inefficient for many situations. Once we step away from finite state spaces (i.e. discreet variables in our model) into continuous state spaces, then this approach will just not work. The proportion of the time where the state of the generative model will conform to the condition given will be near zero. This is not a problem of the approach here, but simply of the nature of the sampling algorithm. There are others.

Having said that, there was a bit of a lightbulb moment when I understood how this was working. I'd say it's well worth understanding this. Even though this sampler might be impractical for many situations, it still gives you a backup position in terms of your conceptual understanding. A lot of the details further down the line (I think) come down to how to achieve this functionality in a computationally efficient manner.

P(flu|cough)

Let's run this to calculate $P(flu|cough)$. We can do this by calling the `rejectionQuery()` function, passing in the model, the query and the condition.

In [4]:

```
queryFunc(state) = state["flu"]
conditionFunc(state) = state["cough"]

queryReturn = rejectionQuery( model, queryFunc, conditionFunc )
```

Out[4]:

```
100000-element Array{Bool,1}:
 false
  true
  true
  true
 false
  true
  true
  true
  true
 false
 false
 false
  true
     ⋮
  true
  true
  true
 false
  true
  true
  true
 false
  true
  true
  true
```

The result is a boolean vector of observations (of flu) given cough=true. The estimate of $P(flu|cough)$ is:

In [5]:

```
(sum(queryReturn) / length(queryReturn)) *100
```

Out[5]:

```
53.89699999999999
```

P(flu|cough&TB)

But how do our beliefs in having the flu change if we observe a cough and TB. We can test this by changing the condition expression

In [6]:

```
queryFunc(state) = state["flu"]
conditionFunc(state) = state["cough"] & state["TB"]

queryReturn = rejectionQuery( model, queryFunc, conditionFunc )
```

Out[6]:

```
100000-element Array{Bool,1}:
 false
 false
 false
 false
 false
 false
  true
 false
  true
 false
  true
 false
 false
     ⋮
 false
 false
  true
 false
 false
  true
  true
 false
 false
  true
 false
 false
```

Running this gave me $P(flu|cough\&TB)$ of

In [7]:

```
(sum(queryReturn) / length(queryReturn)) *100
```

Out[7]:

```
19.866
```