Bayesian Statistics

As a beginning thing you will calculate some bayes' theorem probabilities based on prior probabilities.

Create a series of variables that describe the properties of a disease screening program.

We have a test that is 99% effective at screening a disease. I.e it is 99% effective at finding the disease in people that have it and 99% effective at saying there is no disease for people that don't have it. Suppose that 0.5% of people have this disease. If we randomly selected someone off the street what is the chance that they will actually have the disease if they test positive for it?

```
detection <- 0.99

infected <- 0.005
```

Now we can think of our Bayesian theorem formula:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Our P(A) is just how many people have the disease ie our infected people or 0.5%.

Our P(B|A) is just the chance that if you are infected it will have been detected or 99%

Our P(B) is trickier this is just the probability that anyone would test positive at all. I.e. the possibility of testing positive if you have it added to the probability of a false positive.

In this case P(B) would be 0.99* 0.005 + 0.01 * 0.995

```
positive <- 0.99 * 0.005 + 0.01*0.995

prob <- (detection * infected)/positive
```

so our probability of grabbing a random person on the street who has the disease and them testing positive for it is only 33.2%

So what happens if we alter some of these. First test out what happens if we become 100% accurate at detecting the disease but still only 99% effective at recognising someone who hasn't got it.

How about if we raise the rate of recognising someone who doesn't have it to 99.5% and the accuracy of detection in people that do have it is 99% again.

Here we have calculated a posterior probability of essentially false positives. Which can change depending on new evidence.

It would be very helpful if we could turn this into a function that can be applied to anything. Functions in R are slightly different than functions in Python.

```
Bayes <- function(PBA,PA,PB){


 PAB <- (PBA * PA)/PB
 return(PAB)


}
Bayes(detection,infected,positive)
```

This function will calculate the posterior probability of any probabilities we put into it.

Now try out working out the posterior probability of this example.

In a particular pain clinic, 10% of patients are prescribed narcotic pain killers. Overall, five percent of the clinic's patients are addicted to narcotics (including pain killers and illegal substances). Out of all the people prescribed pain pills, 8% are addicts. If a patient is an addict, what is the probability that they will be prescribed pain pills?

How about if we want to estimate the proportion of something, a problem is that some of our prior knowledge may actually be a distribution of options.

I want to know the proportion of students on the DTP that like to drink. In this situation you can make prior assumptions based on generalisations. I expect that about 85% of them like to have a drink. But I can also think about this in terms of a distribution. I doubt less than 60% of you drink and I expect not more than 95% of you do. This creates a distribution that we can turn into one of our priors. Our priors can't be a straight number 0.6 – 0.95 we need to turn it into a proper prior.

```
install.packages("LearnBayes")
library(LearnBayes)
quantile1 <- list(p=0.5, x=0.85)    # we believe the median of the prior is 0.85
quantile2 <- list(p=0.99999,x=0.95) # we believe the 99.999th percentile of the prior is 0.95
quantile3 <- list(p=0.00001,x=0.60) # we believe the 0.001st percentile of the prior is 0.60
beta.select(quantile1, quantile2)
beta.select(quantile1, quantile3)
```

There is a much more complicated way to select the better prior based on this but we can do it simply and say that our second option is the correct prior between quantile 1 and quantile 3. So our beta prior has a = 52.22 and b = 9.49. (this is almost correct for a best possible prior for this data. If you want the full code for how to get the best one go to the end of the document.)

Lets print the prior so we can take a look at our prior distribution.

```
curve(dbeta(x,52.22,9.49))
```

from the plot you should see this distribution mathematically represents our idea of it starting at 0.6 and dropping off at around 0.95

So now we have our beta prior we should take a look at likelihood of our results. This comes when we actually collect some data. So if I ask you 30 students how many of you drink and 27 of you say you do that is my collected data. Lets take a look at this.

```
calcLikelihood <- function(successes, total)
{
 curve(dbinom(successes,total,x)) # plot the likelihood
}
```

This has its peak at 0.9 which means the most liklely value of the proportion is 0.9. 27/30 = 0.9

Next we will calculate our posterior probability. This will use a incredibly complicated function that I will give you. We don't have the time to go over how this works in detail but essentially it uses our distributions to calculate a prior probability distribution exactly like we did with Bayes() but over distributions.

```
calcPosteriorProb <- function(successes, total, a, b)
 {
   # Adapted from triplot() in the LearnBayes package
   # Plot the prior, likelihood and posterior:
   likelihood_a = successes + 1; likelihood_b = total - successes + 1
   posterior_a = a + successes;  posterior_b = b + total - successes
   theta = seq(0.005, 0.995, length = 500)
   prior = dbeta(theta, a, b)
   likelihood = dbeta(theta, likelihood_a, likelihood_b)
   posterior  = dbeta(theta, posterior_a, posterior_b)
   m = max(c(prior, likelihood, posterior))
   plot(theta, posterior, type = "l", ylab = "Density", lty = 2, lwd = 3,
```

```
        main = paste("beta(", a, ",", b, ") prior, B(", total, ",", successes, ") data,",
        "beta(", posterior_a, ",", posterior_b, ") posterior"), ylim = c(0, m), col = "red")
    lines(theta, likelihood, lty = 1, lwd = 3, col = "blue")
    lines(theta, prior, lty = 3, lwd = 3, col = "green")
    legend(x=0.8,y=m, c("Prior", "Likelihood", "Posterior"), lty = c(3, 1, 2),
        lwd = c(3, 3, 3), col = c("green", "blue", "red"))
    # Print out summary statistics for the prior, likelihood and posterior:
    calcBetaMode <- function(aa, bb) { BetaMode <- (aa - 1)/(aa + bb - 2); return(BetaMode); }
    calcBetaMean <- function(aa, bb) { BetaMean <- (aa)/(aa + bb); return(BetaMean); }
    calcBetaSd   <- function(aa, bb) { BetaSd <- sqrt((aa * bb)/(((aa + bb)^2) * (aa + bb + 1))); return(BetaSd); }
    prior_mode      <- calcBetaMode(a, b)
    likelihood_mode <- calcBetaMode(likelihood_a, likelihood_b)
    posterior_mode  <- calcBetaMode(posterior_a, posterior_b)
    prior_mean      <- calcBetaMean(a, b)
    likelihood_mean <- calcBetaMean(likelihood_a, likelihood_b)
    posterior_mean  <- calcBetaMean(posterior_a, posterior_b)
    prior_sd        <- calcBetaSd(a, b)
    likelihood_sd   <- calcBetaSd(likelihood_a, likelihood_b)
    posterior_sd    <- calcBetaSd(posterior_a, posterior_b)
    print(paste("mode for prior=",prior_mode,", for likelihood=",likelihood_mode,", for posterior=",posterior_mode))
    print(paste("mean for prior=",prior_mean,", for likelihood=",likelihood_mean,", for posterior=",posterior_mean))
    print(paste("sd for prior=",prior_sd,", for likelihood=",likelihood_sd,", for posterior=",posterior_sd))
 }
```

Copy and paste this and use it

```
calcPosteriorForProportion(45, 50, 52.22, 9.49)
```

Ok so we can see our prior probability is not the same as our likelihood and is dependent on the prior.

Now we can try it with a different problem. Work out the priors and the posterior distribution of the proportion of mosquitos that carry malaria. If the proportions of mosquitos that carry malaria in an area probably range from 0 to 5% with most of them somewhere at 3%. And then when you went to survey mosquitos you found that 1234 of the 10000 you surveyed were carrying malaria.

As a final thing you will use the ability for Bayesian probabilities to be updated with new knowledge.

```
CoinToss <- rbinom(10000,1,0.5)
```

This will generate a dataset of 10000 where each one is either a 0 or a 1 with a 50/50 chance or a coin toss essentially.

Next you should run through a loop that uses more and more data of the coin toss each iteration of the loop.

In R loops work a bit differently

```
for (i in seq(2,length(CoinToss),by = ){

dat <- CoinToss[1:i]

success <- length(subset(dat,  dat == 1)

total <- length(dat)

#from here do the posterior probability estimation yourself

}
```

```
findBeta <- function(quantile1,quantile2,quantile3)
  {
     # find the quantiles specified by quantile1 and quantile2 and
quantile3
     quantile1_p <- quantile1[[1]]; quantile1_q <- quantile1[[2]]
     quantile2 p <- quantile2[[1]]; quantile2 q <- quantile2[[2]]
     quantile3_p <- quantile3[[1]]; quantile3_q <- quantile3[[2]]

     # find the beta prior using quantile1 and quantile2
     priorA <- beta.select(quantile1,quantile2)
     priorA_a <- priorA[1]; priorA_b <- priorA[2]

     # find the beta prior using quantile1 and quantile3
     priorB <- beta.select(quantile1,quantile3)
     priorB_a <- priorB[1]; priorB_b <- priorB[2]

     # find the best possible beta prior
     diff_a <- abs(priorA_a - priorB_a); diff_b <- abs(priorB_b - priorB_b)
     step_a <- diff_a / 100; step_b <- diff_b / 100
     if (priorA_a < priorB_a) { start_a <- priorA_a; end_a <- priorB_a }
     else                     { start_a <- priorB_a; end_a <- priorA_a }
     if (priorA_b < priorB_b) { start_b <- priorA_b; end_b <- priorB_b }
     else                     { start_b <- priorB_b; end_b <- priorA_b }
```

```
    steps_a <- seq(from=start_a, to=end_a, length.out=1000)
    steps_b <- seq(from=start_b, to=end_b, length.out=1000)
    max_error <- 10000000000000000000
    best_a <- 0; best_b <- 0
    for (a in steps_a)
    {
       for (b in steps_b)
       {
          # priorC is beta(a,b)
          # find the quantile1_q, quantile2_q, quantile3_q quantiles of
priorC:
          priorC_q1 <- qbeta(c(quantile1_p), a, b)
          priorC_q2 <- qbeta(c(quantile2_p), a, b)
          priorC_q3 <- qbeta(c(quantile3_p), a, b)
          priorC_error <- abs(priorC_q1-quantile1_q) +
                          abs(priorC_q2-quantile2_q) +
                          abs(priorC_q3-quantile3_q)
          if (priorC_error < max_error)
          {
            max_error <- priorC_error; best_a <- a; best_b <- b
          }
       }
    }
    print(paste("The best beta prior has a=",best_a,"b=",best_b))
  }
```