Introduction to Bayesian Modeling

Section 1 - Distributions

Level-Set

In this section, we focus on distributions. In Section 2, we'll add equations *(e.g., regression)* and then explore multilevel models and pooling. This level assumes that you're familiar with basic probability *(including conditional probability)*, distributions, and parameter estimation using gradient descent and other derivative-based methods, linear algebra estimators, and maximum likelihood. Now we will look at parameter estimation using sampling.

Bayes Rule, stated in terms for parameters and data $P(\theta|Data) = \frac{P(\theta)*P(Data|\theta))}{P(Data)}$, works well for inverting probabilities in a closed system. In this case, the denominator $P(Data)$ is the marginal probability of the population. It serves as a normalizing **constant**. Recall how, in maximum likelihood, we can drop the constant portion of the binomial density function $\frac{n!}{n-h!}$ while varying p for likelihood. We can likewise drop the normalizing constant here. Then, the equation becomes:

$P(\theta|Data) \propto P(\theta) * P(Data|\theta)$, or

$Posterior(\theta|Data) \propto Prior(\theta) * Likelihood(Data|\theta)$, where $\propto$ means "proportionate to". We can normalize later.

So, that's where we begin. Now, let's discuss a few overall concepts that will be helpful going forward:

- The model's **target** is $Posterior(\theta|Data)$, i.e., we want the **parameters** *(conditioned on the data)*. Once we have a θ vector of parameters, we can simulate, project, and make inferences about, the posterior. In the business transaction space, where volume and dynamics are high, we don't want a model doing prediction tasks - we just want the model to provide the parameters, and then we'll use more performant, extensible, interoperable programs to generate predictions, and share parameters with other programs.
- The θ values across the prior and the likelihood are seldom the same structure *(i.e, different distributions)*. And since these distribution are multiplied together to get the posterior, we usually don't know what the posterior is going to look like. We can guess about the distribution *(actually, we have to do that)*, but that's just a start. Then, we use the sampler to find the parameters for us.
- Posterior parameters often become priors in subsequent analyses *(note how the prior is also defined by parameters $P(\theta)$, so there's a natural transition)*. This is an important advantage of Bayesian modeling, as we'll see. Also note that parameters can number in hundreds, or thousands *(think of a multilevel model with a couple of levels and hundreds of groups in each level - each with a unique set of parameters)*.

Now, we're going to walk through three models with conjugate distributions *(distributions of the same family)*. This is a nice, clean scenario, and a good exercise for building intuition. We're also going to walk through direct calculation of posterior parameters with two of the models *(the last one - a skew normal doesn't have a closed form solution, so we can't do that)*. Then we'll use these directly determined posteriors as a baseline for comparing and understanding sampling.

We'll also parameter a distribution to get a feel for how that works - as this is a handy skill.

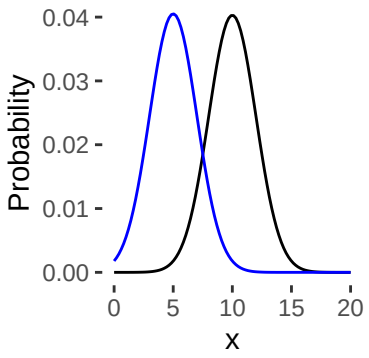
First, load the following libraries:

```
library(tidyverse)
library(rstan)
library(sn)
```

I. Normal Distributions (Conjugate Prior)

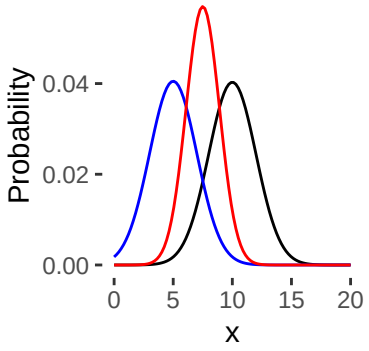
Generate some data using the following parameters for a prior distribution. We'll also *normalize* the data by dividing the density values by the sum(density) *(this keeps all the distributions on the same scale and helps with some of the direct calculations we'll do - not necessary when using a sampler)*. Show the prior, likelihood:

```
# set seed not necessary
set.seed(12)

priorMean <- 10
priorSigma <- 2
likeMean <- 5
likeSigma <- 2

x = seq(from = 0, to = 20, length.out = 100)
dfNorm <- data.frame(x=x, prior = dnorm(x, priorMean ,priorSigma))
dfNorm$prior  <- dfNorm$prior/sum(dfNorm$prior) #normalize

p <- ggplot(data = dfNorm, aes(x, prior)) +
  geom_line() +
  theme(panel.background = element_rect(fill = "white"))  +
  ylab("Probability")

dfNorm$like   <- dnorm(x, likeMean, likeSigma)
dfNorm$like   <- dfNorm$like/sum(dfNorm$like) # normalize

p <- p + geom_line(data = dfNorm, aes(x, like), color = 'blue')
p
```



Now, let's create a posterior $P(\theta) * P(Data|\theta)$:

```
dfNorm$post   <- dfNorm$like*dfNorm$prior # multiply densities
dfNorm$post   <- dfNorm$post/sum(dfNorm$post) # normalize
p <- p + geom_line(data = dfNorm, aes(x, post), color = 'red')
p
```



We often say that the posterior is a compromise between the prior and likelihood. Can you see why?

So, what are the parameters of the posterior? Well, we don't know..

The nice thing about **some** conjugate distributions is that we can compute parameter estimates directly *(you can't just use moments or parameter functions - there's no data - the posterior is imaginary - we only have density factors - look at the dfNorm table).*

Here's how we can estimate parameters in the conjugate normal case:

$\sigma = \sqrt{\frac{1}{\sigma_1^{-2}+\sigma_2^{-2}}}$

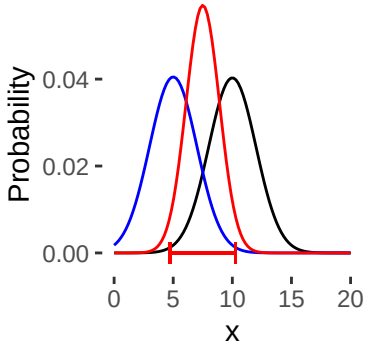
$\mu = posterior\sigma^2 * \frac{\mu_1}{\frac{\sigma^2+\mu_2}{\sigma_2^2}}$

Then we can compute our confidence intervals *(now that we have the parameters)*:

```
postVar <- 1/(priorSigma^{-2}+likeSigma^{-2})          # Posterior var
ppostSigma  <- sqrt(postVar)                  # Posterior sd
postMean  <- postVar*(priorMean/priorSigma^2+likeMean/likeSigma^2)     # Posterior mean

lowCI <- qnorm(0.025, postMean, ppostSigma)      # Posterior 95% interval
highCI <- qnorm(0.975, postMean, ppostSigma)
```

Visually:

```
p <- p + geom_errorbarh(aes(xmin = lowCI, xmax = highCI, y = 0), height = .005, color = "red")
p
```



Now we're going to do a similar estimate using Stan *(https://mc-stan.org/ - please download the user manual, etc. you'll need it).* The following text is our Stan model to sample this posterior:

```
stanMod <- '
data {
int N; // number of observations
vector[N] y; // likelihood data
real priorMu;
real<lower=0> priorSigma;
}
parameters {
real mu; // posterior mean
real<lower=0> sigma; // posterior sigma
}
model {
target += normal_lpdf(y | mu, sigma); // likelihood data
target += normal_lpdf(mu | priorMu, .2); // prior Mu
target += normal_lpdf(sigma | priorSigma, .001); // prior sigma
}
'
```

This Stan model has 3 sections data, parameters, and model *(there are more sections possible, which we'll cover later)*:

- **data**. The data section is where we set up the variables and datatypes, which are passed into the model from the stan function call *(below)*. Vectors are assumed to be numeric *(otherwise, we define the type explicitly - e.g., int x[N];)*.
- **parameters**. The parameters section is where we define the parameters we want the sampler to track - we'll get an estimate for each one. When defining scale parameters, be sure and use "real<lower=0> sigma" syntax, because a scale parameter can't be negative - otherwise you give the sampler permission to wander around looking for a negative sigmas, and that's not going to end well *(samplers are drunk)*!
- **model**. The model section is where we tell the sampler how it all fits together. To get a parameter into the sampling process, we add it to the target log probability density function *(using target +=)*, and define the distribution *(e.g., using normal_lpdf for normal log density - there are a TON of options for distributions - RTFM)*. Looking at the statements in the model section:

  target += normal_lpdf(y | mu, sigma); This statement fits a normal distribution to the likelihood data (y). y is conditioned on mu and sigma, and from there draws samples into the posterior target.

  target += normal_lpdf(mu | priorMu, .2); fits the mu, conditioned on the priorMu (which we've set to 10). The sigma of the priorMu is set to .2, which expresses confidence in that prior (this allows the sampler to move the prior between 9.6 and 10.4 (+- 2 se's for 95%) as it tries to fit the posterior parameters), which forces the posterior to settle ~ 7.5.

  target += normal_lpdf(sigma | priorSigma, .001); fits the sigma the same way. The sigma of the sigma (yes, I said that) is set to a draconian .001, so it can only move the sigma between 1.998 and 2.002 (more discipline, less bondage! "High Anxiety").

- Elaboration and Application

One element about the model that may seem new is the definition of the variance of the prior parameters. For example, in the statement: target += normal_lpdf(mu | priorMu, .2), we're judgmentally defining the std deviation of priorMu to be .2 *(keep in mind this is not the variance of the data, it's the variance of the parameter)*. This represents the confidence that we have in the Prior - the lower the σ, the more confidence we have in the priorMu. If we increase σ, the sampler is allowed to wander more. What direction? It will wander towards the data *(likelihood)*, i.e, y. This is as it should be - if we don't have confidence in our priors,

the model trusts the data. This is a Bayesian thing, and a powerful capability - esp. in business. Consider these scenarios:

> Let's say the we're planning profit, production and inventory, and we make heavy equipment with the main components made from steel. We meet with the global sourcing staff and look at the last 2 years of data (likelihood). Several of the procurement managers believe there will be capacity limits in the EU, and a couple of managers also think there will be a global shortage, with an increase in prices. It seems obvious that, in this scenario, we need to consider the possibilities discussed, and we can do that by conditioning the data with prior estimates. As data rolls in, we can refine those priors, but we can't wait until then to build a plan - we have to start production.
>
> There is a similar scenario regrading sales of new products different markets that was posted in the Repository introduction. These scenarios are about forecasting, which eventually end up in plans and budgets. Don't get the idea that forecasts are just some doodles in a spreadsheet - they are the basis of major commitments and budgets (operating and capital), and have to be approved by the Board of Directors (as mandated in the Corporate Charter). And for assurance and audit professionals, these models explain the relationships between external drivers and business results - often they provide the best evidence for opinions.

These sigmas also give us the ability to **generalize** and **pool** specific parameters. These are powerful tools in the Bayesian kit. If you're in this course, you've hopefully studied generalization. In the machine learning world, L1 and L2 generalization are globally applied. That's just not adequate for mulitlevel enterprise scenarios. And these multiple levels will cluster in effects *(for example, in the scenario above, we might expect the effect of steel prices to be greater in EU locations than in Americas)*

So now we run the model. Stan will show progress: *(this is the only time I'll do this in a document, but it's a good practice to monitor sampler output on all models).*

```
N  = nrow(dfNorm)
# generate likelihood data
x = seq(from = 0, to = 20, length.out = N)
y <- rnorm(x, mean = likeMean, sd = likeSigma)

fit <- stan(model_code = stanMod, data = list(
  y= y,
  N = N,
  priorMu = priorMean,
  priorSigma = priorSigma)
  )
```

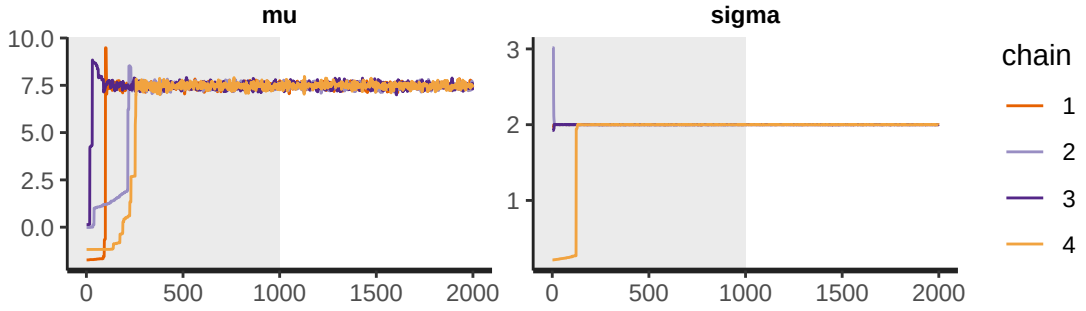
```
##
## SAMPLING FOR MODEL '575c502e6fb6b1a2db60c77a96225199' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
```

```
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.04 seconds (Warm-up)
## Chain 1:                0.028 seconds (Sampling)
## Chain 1:                0.068 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL '575c502e6fb6b1a2db60c77a96225199' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.026 seconds (Warm-up)
## Chain 2:                0.044 seconds (Sampling)
## Chain 2:                0.07 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL '575c502e6fb6b1a2db60c77a96225199' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
```

```
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.055 seconds (Warm-up)
## Chain 3:                0.034 seconds (Sampling)
## Chain 3:                0.089 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL '575c502e6fb6b1a2db60c77a96225199' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.05 seconds (Warm-up)
## Chain 4:                0.025 seconds (Sampling)
## Chain 4:                0.075 seconds (Total)
## Chain 4:
```

The output shows the progress of 4 sampling chains *(Stan sets 4 chains with 2k iterations by default)*, and the parameters. A visual representation of the sampling can be created with a trace plot. Compare this with the description of how the sampler finds parameters above *(and you can see how our choice of prior sigmas constrains the sampler)*:

```
# plot of sampling
stan_trace(fit, inc_warmup = TRUE)
```



And now we can get what we're after:

```
fit
```

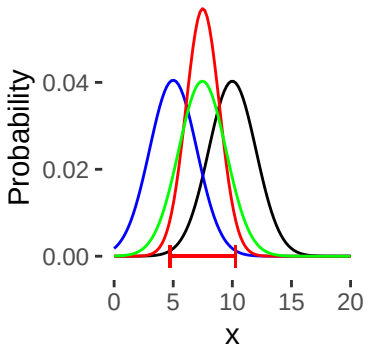
```
## Inference for Stan model: 575c502e6fb6b1a2db60c77a96225199.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##          mean se_mean   sd    2.5%     25%     50%     75%   97.5% n_eff Rhat
## mu       7.46    0.00 0.14    7.20    7.37    7.47    7.56    7.75   806 1.01
## sigma    2.00    0.00 0.00    2.00    2.00    2.00    2.00    2.00  3788 1.00
## lp__  -352.03    0.03 0.98 -354.57 -352.44 -351.73 -351.33 -351.07  1096 1.00
##
## Samples were drawn using NUTS(diag_e) at Thu Apr 02 16:01:19 2020.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

Note: we usually get the output of the model by using a summary() function on the model fit, as follows *(there are ways to tune this as we'll see later)*.

```
# get parameters and plot mu on model
sumFit <- summary(fit)
postMu <- sumFit$summary[1,1]
postSigma <- sumFit$summary[2,1]
```

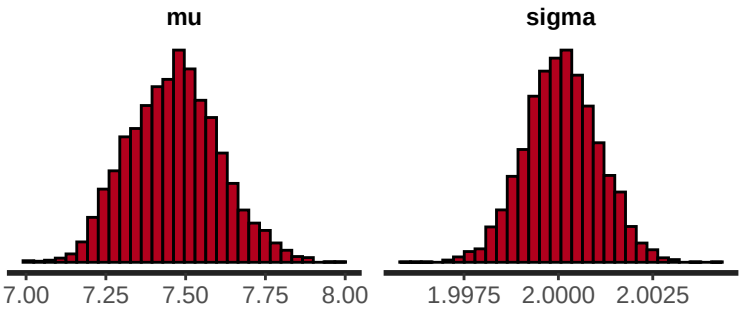
And we can add our estimated μ to the previous posterior for comparison:

```
dfNormStan = data.frame(x = x, density = dnorm(x, mean = postMu, sd = postSigma ))
dfNormStan$density = dfNormStan$density/sum(dfNormStan$density) #normalize

p <- p + geom_line(data = dfNormStan, aes(x, density), color = 'green')
p
```



and the following is a plot of the fit *(remember, this is the posterior parameters, not the prior)*:

```
stan_hist(fit)
```



Again, we generally transfer the parameter values to a dataframe where we can manage them - there can be a LOT!

The most comprehensive tool for diagnosing sampling is shinystan. I don't run this in a markdown document, as it stops the knit function, but please uncomment and run from an r file *(it will open up your browser, and r execution will halt until you shut it down)*

```
# library(shinystan)
# launch_shinystan(fit)
# don't launch in a rmd
```

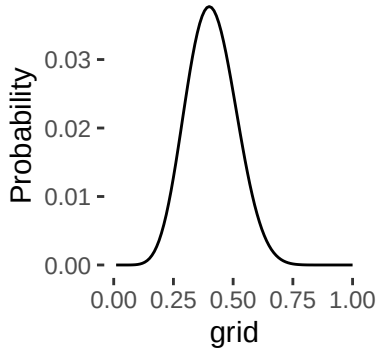
Perspective

Sampling is a complex and fragile process. We're treating it as a "black box" here, but you will have to debug sampling, which means you'll need to have some intuition, built on hands-on experience and reading everything you can find. Diagnosis level expertise is art and science and requires commitment. This is No BS territory.

II. Beta Binomial Distributions (Conjugate Prior)

If you've worked with logistic regression, you're familiar with a binomial distribution. In this example, we'll generate likelihood data assuming 8 successes out of 20 trials *(so a mean of .4 for a continuous probability distribution)*:

```
h <- 8
# n number of trials
n <- 20
# p probability of success
grid <- seq(from = .01, to = 1, by = .01)

dfData <- data.frame(h, n, grid)
dfData$like <- dbinom(h, size = n, prob = grid)
dfData$like <- dfData$like/sum(dfData$like) # normalize

p1 <- ggplot(dfData, aes(x = grid, y = like)) +
  geom_line() +
  ylab("Probability") +
  theme(panel.background = element_rect(fill = "white"))
p1
```



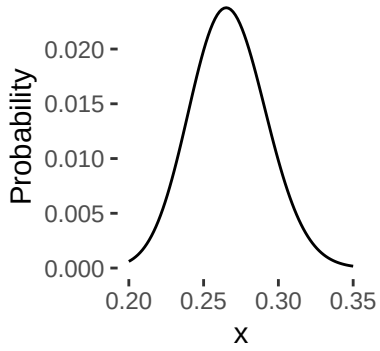
Reparameterization: Binomial => Beta

The Binomial distribution can be **reparameterized** as a Beta distribution, which like the binomial, represents a density of *probabilities (it can also be reparameterized as a Normal)*

Diversion to discuss Beta distributions

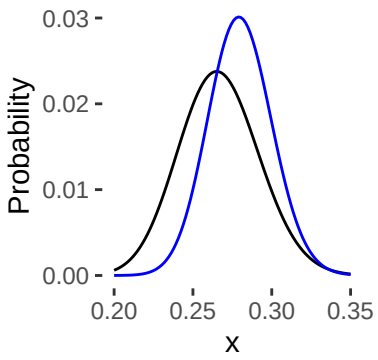
We're going to break for a minute to talk about the advantages of the Beta distribution.The classic illustration is baseball batters (recommend David Robinson's "Introduction to Empirical Bayes" - very approachable read). Let's say a batter has 300 AB's and gets 80 hits, so he's batting ~ 267 and our expectation for batting average is in the range of .2-.35 (this could be a prior based on a century of experience). All we need to tell the density function is the successes and failures:

```
a = 80
b = 220

bgrid = seq(from = .2, to = .35, length.out = 100)
dfBeta <- data.frame(x <- bgrid, db = dbeta(bgrid, a, b))
# note to ellen - go back and check earlier beta pdf for consistency
dfBeta$db <- dfBeta$db/sum(dfBeta$db) # normalize
p1 <- ggplot(data = dfBeta, aes(x, db)) +
  geom_line() +
  ylab("Probability") +
  theme(panel.background = element_rect(fill = "white"))
p1
```



Now let's say he gets up to bat 200 more times and gets 60 hits. Updating the parameters is really easy, we just add:

```
a = (80 + 60)
b = (220 + 140)

dfBeta2 <- data.frame(x <- bgrid, db = dbeta(bgrid, a, b))
dfBeta2$db <- dfBeta2$db/sum(dfBeta2$db) # standardize
p1 <- p1 +
  geom_line(data = dfBeta2, aes(x, db), color = "blue") +
  ylab("Probability") +
  theme(panel.background = element_rect(fill = "white"))
p1
```



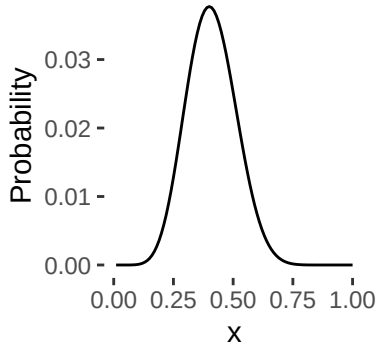
How easy is that? Also notice that the scale has decreased slightly. Why? Because we have more data (Doh!).

OK, back to Bayesian Modeling. We're going to reparameterize the Binomial as a Beta *(because it's easier to work with)*, using the following formulas:

$\alpha = h + 1$ *(remember h is number of successes in binomial)*
$\beta = n - h + 1$ *(and n is number of trials in binomial)*

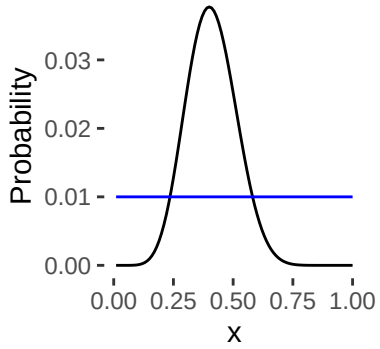
So:

```
a = h+1
b = (n-h+1)

dfBeta <- data.frame(x <- grid, db = dbeta(grid, a, b))
# note to ellen - go back and check earlier beta pdf for consistency
dfBeta$db <- dfBeta$db/sum(dfBeta$db) # standardize
p1 <- ggplot(data = dfBeta, aes(x, db)) +
  geom_line() +
  ylab("Probability") +
  theme(panel.background = element_rect(fill = "white"))
p1
```



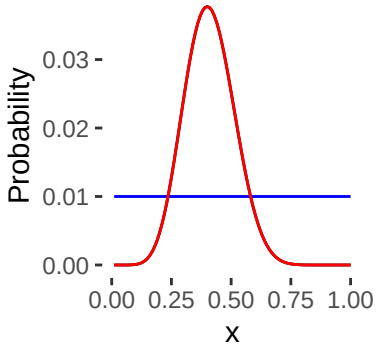
This time, we'll assume that we don't have any experience to add to our analysis - so the likelihood data is all we know. In this case, in the Bayesian world, we use a *non-informative*, or *uniform* prior *(in the Beta distribution with two parameters, a flat distribution is where a and b, are 1)*:

```
# now we'll set up a uniform prior (more on uniform priors later)

aPrior <- 1
bPrior <- 1
prior  <- dbeta(grid,aPrior,bPrior)
dfBeta$prior  <- prior/sum(prior) #standardize
p1 <- p1 +
  geom_line(data = dfBeta, aes(x, prior), color = "blue") +
  ylab("Probability")
p1
```



Now, we'll estimate the posterior parameters using a direct method *(like above)*:

```
post   <- dfData$like*prior
dfBeta$post   <- post/sum(post)
p1 <- p1 +
  geom_line(data = dfBeta, aes(x, post), color = 'red') +
  ylab("Probability")
p1
```



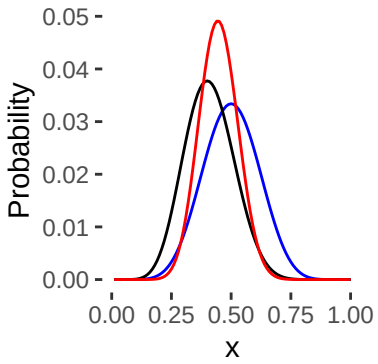
So if we use a uniform prior, the posterior reflects the likelihood data *(should make sense)*.

Now, let's change the prior as see what happens *(lets set success rate at .5 in the prior)*:

```
a <- h+1
b <- h+1 # setting success and failures equal, so 50% prob

prior  <- dbeta(grid,a,b)
dfBeta$prior  <- prior/sum(prior) #normalize

p2 <- ggplot(data = dfBeta, aes(x, prior)) +
  geom_line(color = "blue") +   # prior
  geom_line(aes(x, db)) +
  ylab("Probability") +
  theme(panel.background = element_rect(fill = "white"))
```

Now let's see how that affects the posterior"

```
# now recalculate a posterior
post   <- dfData$like*prior
dfBeta$post   <- post/sum(post) #normalize
# and redraw
p2 <- p2 +
  geom_line(data = dfBeta, aes(x, post), color = "red")
p2
```

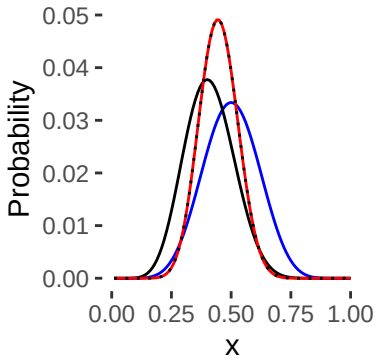


And here's how we estimate the parameters using direct calculation:

$PosteriorBetaParameters = [(h + a), (n - h + b)]$

(note that the h and n came from the likelihood data, and the a and b came from the prior - different distributions that can easily be combined and updated in the Beta distribution - see why we reparameterized?):

```
# EXACT CALCUATION  -------#
dfBeta$post2 <- dbeta(grid,h+a,n-h+b)
dfBeta$post2 <- dfBeta$post2/sum(dfBeta$post2) #normalize
p2 <- p2 +
  geom_line(data = dfBeta, aes(x, post2), linetype = "dotted", color = "black") +
  ylab("Probability")
p2
```



And calculating the confidence interval:

```
# extending calculation

(h+a)/(n+a+b)                       # Posterior mean
```

```
## [1] 0.4473684
```

```
qbeta(c(0.05,0.95),h+a,n-h+b)      # Posterior 90% interval
```

```
## [1] 0.3178115 0.5801185
```

Now, like before, we'll model this in Stan. Notice that the distribution has changed to a beta.

```
# The Stan model as a string.

model_string <- "
// Bernoulli model
data {
  int<lower=0> n; // number of observations
  int<lower=0> h; // likelihood (successes)
  int<lower=0> priorA;
  int<lower=0> priorB;
}
parameters {
  real<lower=0,upper=1> theta;
}
model {

  target += beta_lpdf(theta | priorA, priorB);
  target += binomial_lpmf(h | n, theta);
}
"
d_bin <- list(
  n = n,
  h = h,
```

```
  priorA = a,
  priorB = b)

# fit_bin <- stan(model_code = model_string, data = d_bin)
fit_bin <- stan(model_code = model_string, data = d_bin, refresh = 0) # doesn't show ouput
print(fit_bin,digits=2)
```

```
## Inference for Stan model: 64634405d6cf3dfab515119ba9b8e66a.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##        mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
## theta  0.45    0.00 0.08  0.29  0.39  0.45  0.50  0.60  1395    1
## lp__  -2.59    0.02 0.71 -4.63 -2.75 -2.31 -2.14 -2.09  1817    1
##
## Samples were drawn using NUTS(diag_e) at Thu Apr 02 16:02:06 2020.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```
sumFit <- summary(fit_bin)
mu <- sumFit$summary[1,1]
sd <- sumFit$summary[1,3] # this is NOT the sd for the posterior (we didn't estimate that in the model)
```

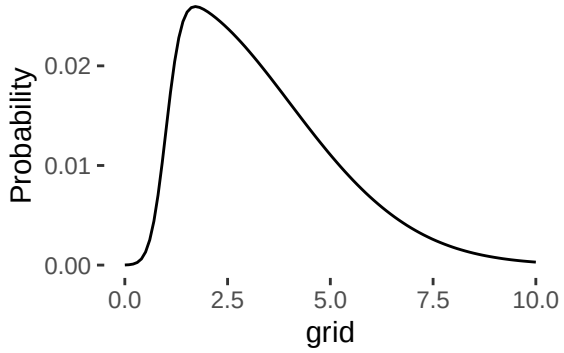
Looking at the model section *(switching order here for discussion purposes)*

> target += binomial_lpmf(h | n, theta); The theta is the probability parameter for the likelihood data which has n of 20 and h of 8. Those are the data - the likelihood - so p wants to go to .4.
>
> target += beta_lpdf(theta | priorA, priorB); But theta has priors that must be rationalized within a beta distribution. The PiorA is 9 and the PriorB is 9, so the prior wants to pull towards .5. The sampler finds the balance

III. Skew Normal

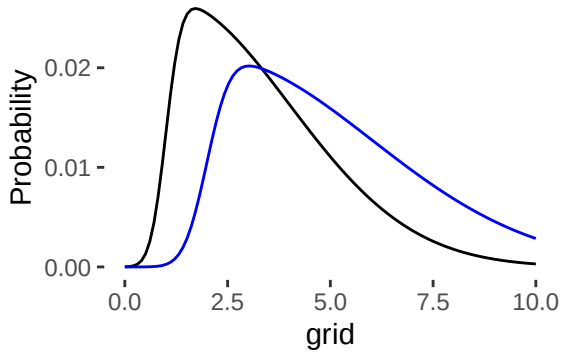
The Skew Normal doesn't have a closed form solution, so we won't be able to get exact calculations for comparison *(which will be the case in 99% of your work in business scenarios)*.

```
grid = seq(from = 0, to = 10, length.out = 100)
dfData <- data.frame(grid = grid, like = dsn(grid, 1, 3, 10))
dfData$like <- dfData$like/sum(dfData$like)

p3 <- ggplot(data = dfData) +
  geom_line(aes(grid, like)) +
  ylab("Probability") +
  theme(panel.background = element_rect(fill = "white"))
p3
```



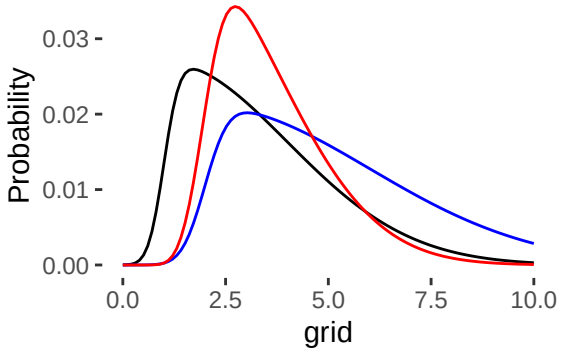
Setting the priors:

```
priorLocat <- 2
priorScale <- 4
priorSkew <-  9

dfData$prior <- dsn(grid, priorLocat, priorScale, priorSkew)
dfData$prior <- dfData$prior/sum(dfData$prior)

p3 <- p3 +
  geom_line(data = dfData, aes(grid, prior), color = "blue") +
  theme(panel.background = element_rect(fill = "white"))
p3
```



Even though there's no closed form solution, we're still working with $P(\theta|Data) \propto P(\theta) * P(Data|\theta)$. To the sampler, it really doesn't matter *(although it does affect computation times)*

```
dfData$post <- dfData$like*dfData$prior
dfData$post   <- dfData$post/sum(dfData$post)

# there is no closed form solution to SN distributions
# we can estimate using likelihood * prior

p3 <- p3 +
  geom_line(data = dfData, aes(grid, post), color = "red") +
  theme(panel.background = element_rect(fill = "white"))
p3
```



Now for Stan

```
stanMod <- '
data {
  int N;
  vector[N] y;
  real xi_mu;
  real<lower=0> xi_sigma;
  real omega_mu;
  real<lower=0> omega_sigma;
  real alpha_mu;
  real<lower=0> alpha_sigma;
}
parameters {
  real xi;
  real<lower=0> omega;
  real alpha;
}
model {
  target += skew_normal_lpdf(y | xi, omega, alpha);
  target += normal_lpdf(xi | xi_mu, xi_sigma);
  target += normal_lpdf(omega | omega_mu, omega_sigma);
  target += normal_lpdf(alpha | alpha_mu, alpha_sigma);
}
'
locat <- 1
scale <- 3
skew <- 10
N = 100
y <- rsn( N , xi= locat, omega= scale , alpha= skew ) # using data sample now

mod <- stan_model(model_code = stanMod)

priorLocat <- 2
priorScale <- 4
priorSkew <-  9

fit <- stan(model_code = stanMod,
            data = list(y = y,
            xi_mu = priorLocat,
            xi_sigma = .5,
            omega_mu = priorScale,
```

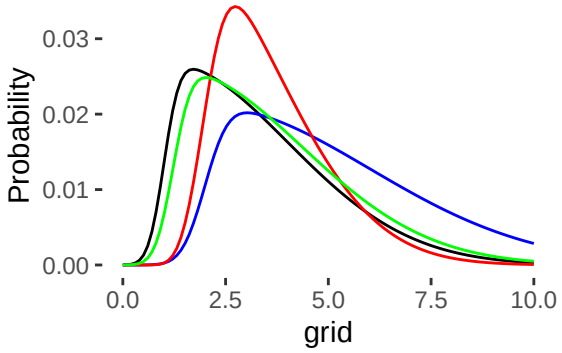
```
            omega_sigma = .5,
            alpha_mu = priorSkew,
            alpha_sigma = .5,
            N = 100), refresh = 0)
print(fit)
```

```
## Inference for Stan model: caf868e851a61cbc4617fa598b78b6f1.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##          mean se_mean   sd    2.5%     25%     50%     75%   97.5% n_eff Rhat
## xi       1.23    0.00 0.11    0.99    1.15    1.23    1.30    1.44  2894    1
## omega    3.12    0.00 0.22    2.73    2.97    3.11    3.27    3.59  3005    1
## alpha    9.01    0.01 0.49    8.03    8.69    9.02    9.34   10.00  3352    1
## lp__  -191.90    0.03 1.21 -194.93 -192.49 -191.60 -191.00 -190.50  1694    1
##
## Samples were drawn using NUTS(diag_e) at Thu Apr 02 16:02:52 2020.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

The model here is pretty straightforward:

> target += skew_normal_lpdf(y | xi, omega, alpha); The likelihood data (y) is conditioned on three parameters xi, omega and alpha, which is 1, 3, and 10.
>
> target += normal_lpdf(xi | xi_mu, xi_sigma); xi of 1 is conditioned on a prior of 2 with a xi_sigma of .5
>
> target += normal_lpdf(omega | omega_mu, omega_sigma); omega of 3 is conditioned on a prior of 4 with a omega_sigma of .5
>
> target += normal_lpdf(alpha | alpha_mu, alpha_sigma); alpha of 10 is conditioned on a prior of 9 with a alpha_sigma of .5

```
sumFit <- summary(fit)
postLocat <- sumFit$summary[1,1]
postScale <- sumFit$summary[2,1]
postSkew <- sumFit$summary[3,1]

dfData$post2 <- dsn(grid, postLocat, postScale, postSkew)
dfData$post2 <- dfData$post2/sum(dfData$post2)

p3 <- p3 +
  geom_line(data = dfData, aes(grid, post2), color = "green") +
  theme(panel.background = element_rect(fill = "white"))
p3
```



Take-Aways

The Bayesian modeling process is really quite natural. We collect data for a current event *(likelihood)*, and many times we have prior knowledge that can, and should influence our analysis of that current event *(the batter example is classic - what if the batter got a hit his first 10 times up - is he likely to bat 1.0?)*. This results in a posterior that balances evidence and experience to produce a posterior. Now for inference: if we don't have distributions, we don't have probability, if we don't have probability, we can't make an inference. So, we need distributions. We've looked at 3-4 distributions here, but there are many, many others *(RTFM)*. The art and science of fitting distributions and setting parameters is where experience and intuition come in.

Bayesian analysis is an established tool and most Data Science groups will expect you to apply Bayesian analysis and multilevel, pooled models as a default approach in business analysis. That's where you start. Machine learning is really a tool for tuning Bayesian models. We'll look at equations *(within the Stan model)* next.

Homework

1. Create an R file that walks through the 3 scenarios, but use your own parameters *(different from mine - no copy and paste - THINK through it, try different values and see how it affects the outcome)*

2. Include conclusions and comments in the file

3. Make sure it runs end-to-end. I'm not debugging - no run, no credit.