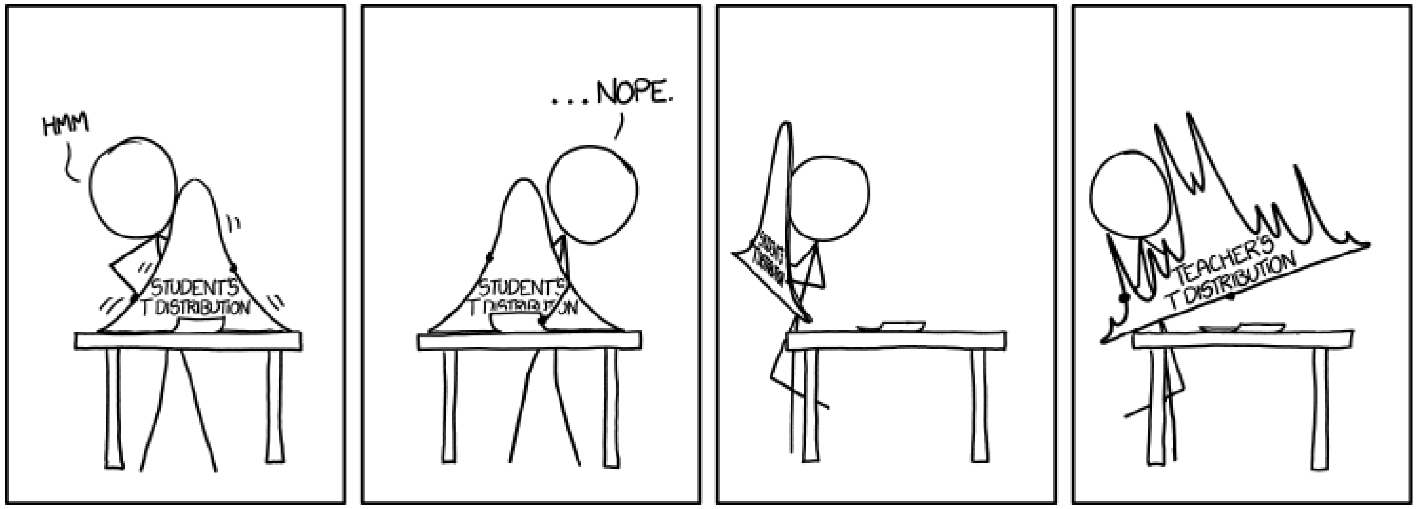
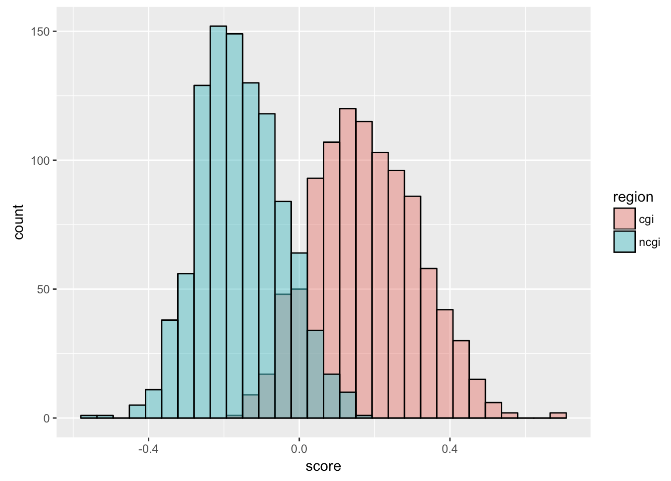
Mixture Models

Susan Holmes



(xkcd)

Why mixture models?



Heterogeneity due to hidden variables, some that we want to detect, some that are uninteresting.

The statistical sample does not come from only one population.

Types of Mixture Models

There are two types of mixture models we will discuss: finite and infinite

Simple Examples and computer experiments

Suppose we want two equally likely components, we decompose the generating process into steps:

Flip a fair coin.

- If it comes up heads + Generate a random number from a Normal with mean 1 and variance 0.25.
- If it comes up tails + Generate a random number from a Normal with mean 2 and variance 0.25.

Fair Mixture

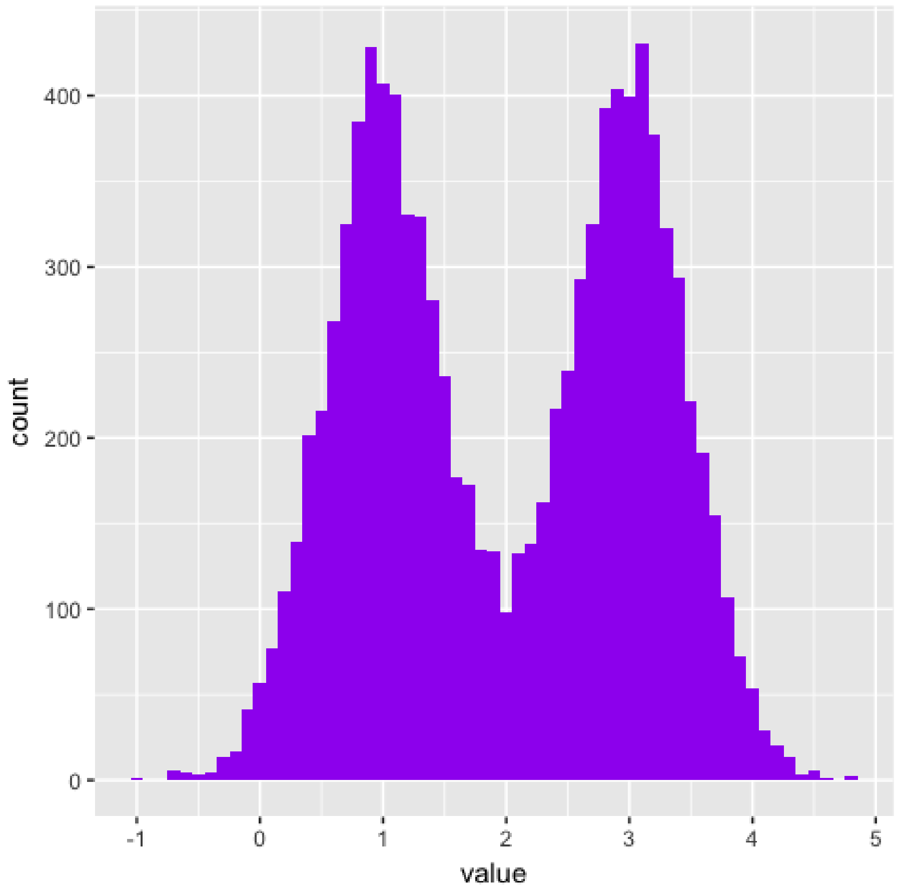
This is what the resulting histogram would look like if we did this 10,000 times.

```
require(ggplot2)
require(tibble)
require(dplyr)
coinflips = (runif(10000) > 0.5)
table(coinflips)
```

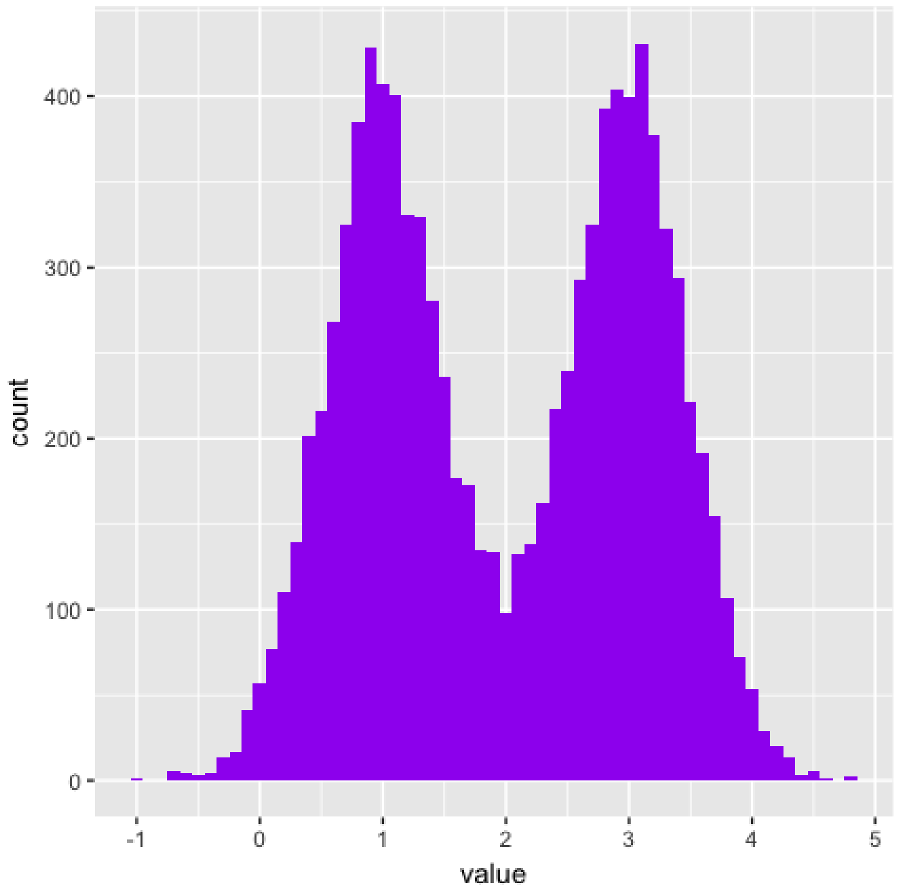
```
## coinflips
## FALSE  TRUE
##  5054  4946
```

Generate Plot

```r
oneFlip = function(fl, mean1 = 1, mean2 = 3, sd1 = 0.5, sd2 = 0.5) {
  if (fl) {
   rnorm(1, mean1, sd1)
  } else {
   rnorm(1, mean2, sd2)
  }
}
fairmix = vapply(coinflips, oneFlip, numeric(1))
ggplot(tibble(value = fairmix), aes(x = value)) +
     geom_histogram(fill = "purple", binwidth = 0.1)
```



Histogram



Simple histogram of two normals

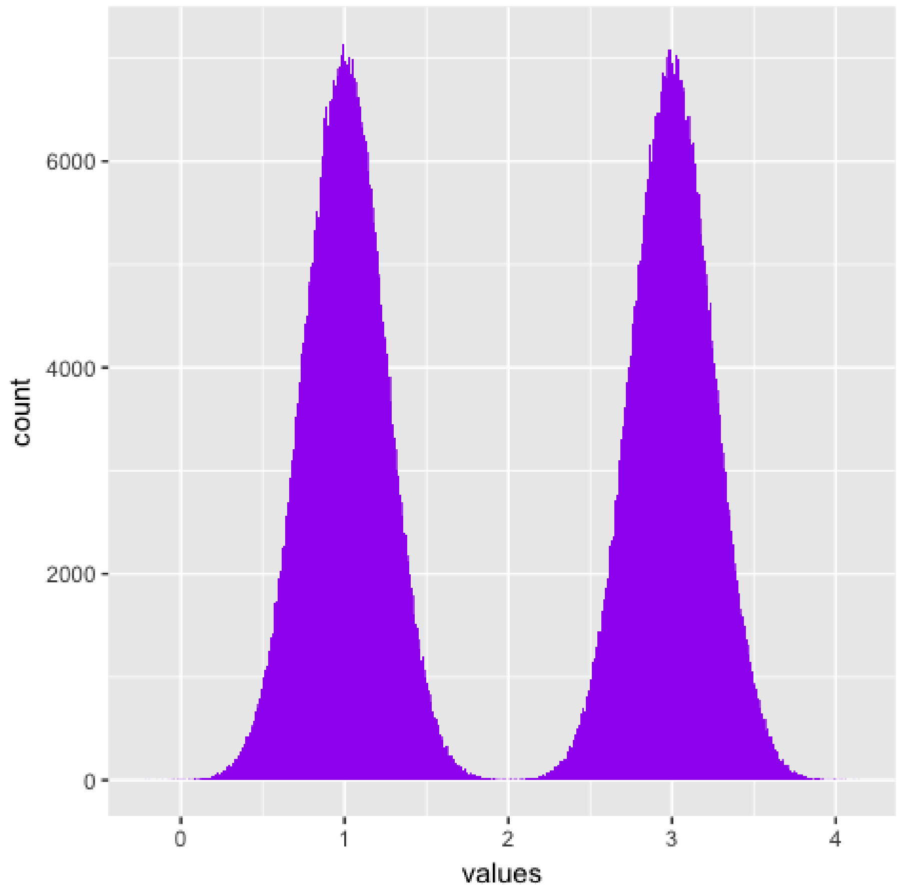
Density

In fact we can write the density (the limiting curve that the histograms tend to look like) as

$$f(x) = \frac{1}{2}\phi_1(x) + \frac{1}{2}\phi_2(x)$$

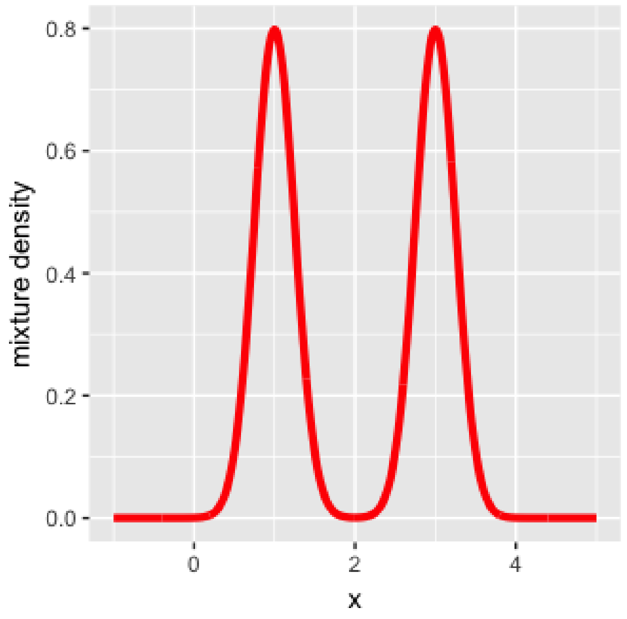
where ϕ_1 is the density of the Normal($\mu_1 = 1, \sigma^2 = 0.25$) and ϕ_2 is the density of the Normal($\mu_2 = 2, \sigma^2 = 0.25$).

Density : R



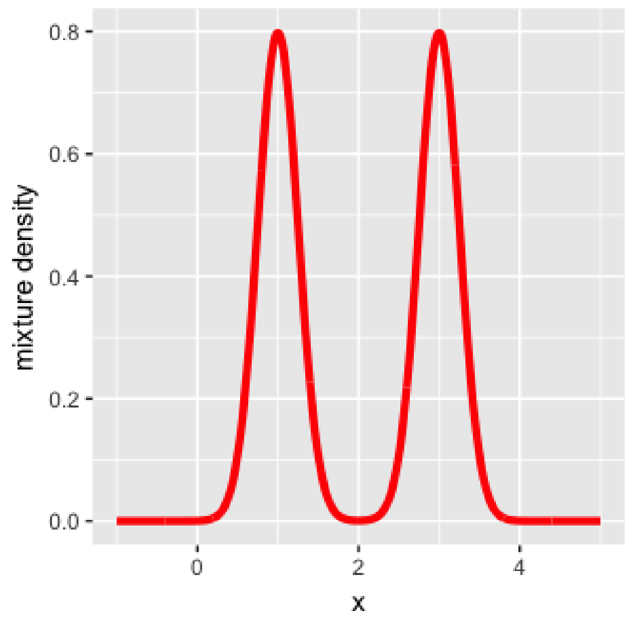
Density : Plot

```r
fairtheory = tibble(
  x = seq(-1, 5, length.out = 1000),
  f = 0.5 * dnorm(x, mean = means[1], sd = sds[1]) +
      0.5 * dnorm(x, mean = means[2], sd = sds[2]))
ggplot(fairtheory, aes(x = x, y = f)) +
  geom_line(color = "red", size = 1.5) + ylab("mixture density")
```



The theoretical density of the mixture.

Density : Plot

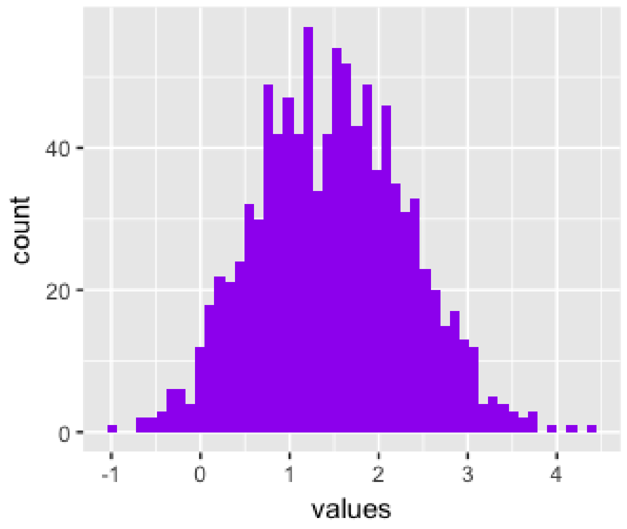


Simple histogram of two normals

Less Definite Mixtures

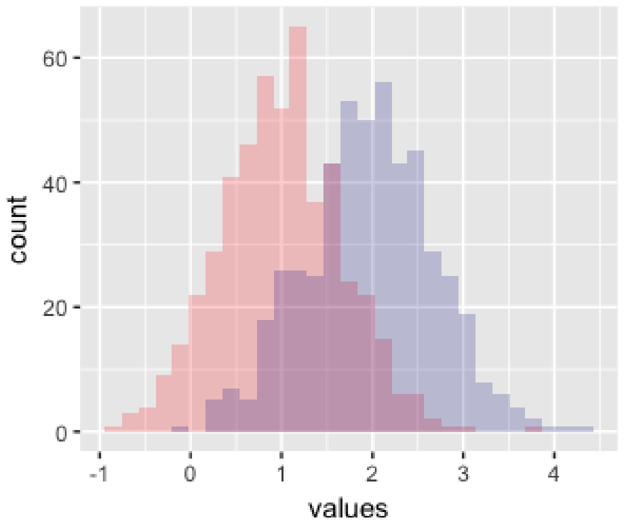
In this case of course the mixture model was extremely visible as the two distributions don't overlap, this can happen if we have two very separate populations, for instance different species of fish whose weights are very different. However if many cases the separation is not so clear.

Challenge: Here is a histogram generated by two Normals with the same variances, can you guess the two parameters for these two Normals?



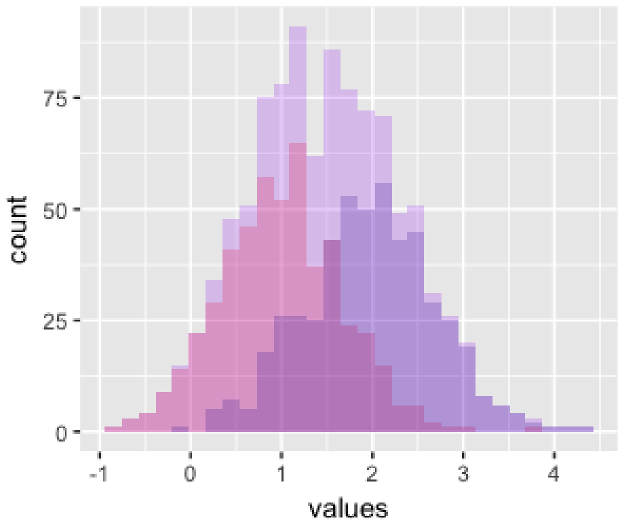
A mixture of two normals that is harder to recognize.

```
## # A tibble: 3 x 2
##   coinflips values
##   <lgl>      <dbl>
## 1 TRUE       0.343
## 2 FALSE      1.91
## 3 FALSE      1.21
```



The mixture from above, but with the two components colored in red and blue.

```
## # A tibble: 3 x 2
##   coinflips values
##   <lgl>      <dbl>
## 1 TRUE       0.343
## 2 FALSE      1.91
## 3 FALSE      1.21
```



The mixture from above, but with the two components colored in red and blue.

Latent variable: first case

Here we knew who had been generated from which component of the mixture, often this information is missing, we call the hidden variable a latent variable.

This is an important example of heterogeneity due to a hidden variable (unmeasured or **latent**).

Discovering the hidden class labels

We use the EM algorithm infer the value of the hidden groupings.

Expectation-Maximization algorithm alternates between

- pretending we know the probability with which each observation belongs to a component and estimating the distribution parameters of the components, and
- pretending we know the parameters of the component distributions and estimating the probability with which each observation belongs to the components.

Let's take a simple example. We measure a variable Y on a series of objects that we think come from two groups, although we do not know the group labels. We start by *augmenting* the data with the unobserved (latent) group label, which we'll call U.

We want to learn:
- U and
- the unknown parameters of the underlying densities.

We use the maximum likelihood approach. We will use **bivariate** distributions here because we are going to look at the distribution of "couples" (Y, U).

$$f_\theta(y, u) = f_\theta(y \mid u) f_\theta(u)$$

Suppose we have a fair mixture of two normals with parameters
$\theta = (\mu_1 = ?, \mu_2 = ?, \sigma_1 = 1, \sigma_2 = 1, \lambda = \frac{1}{2})$, μ_1 and μ_2 are unknown.

For this bivariate distribution we can define a `complete joint likelihood`, we usually work with its log

$$loglikeli(\theta) = \log f(y, u|\theta)$$

Mixture of normals

Suppose we have a mixture of two normals with mean parameters unknown and standard deviations 1; thus, $(\mu_1 = ?,\ \mu_2 = ?,\ \sigma_1 = \sigma_2 = 1)$.

Here is an example of data generated according to such a model. The labels are u.

```
mus = c(-0.5, 1.5)
u = sample(2, 100, replace = TRUE)
y = rnorm(length(u), mean = mus[u])
duy = tibble(u, y)
head(duy)
```

```
## # A tibble: 6 x 2
##         u      y
##   <int>  <dbl>
## 1     2  2.78
## 2     1  0.332
## 3     2  2.01
## 4     1 -1.64
## 5     2  1.42
## 6     2  2.30
```

If we know the labels u, we can estimate the means using separate MLEs for each group. The overall MLE is obtained by maximizing

$$f(y, u \mid \theta) = \prod_{\{i:\, u_i=1\}} \phi_1(y_i) \prod_{\{i:\, u_i=2\}} \phi_2(y_i),$$

or its logarithm.

The maximization can be split into two independent pieces and solved as if we had two different MLEs to find:

```
group_by(duy, u) %>% summarize(mean(y))
```

```
## # A tibble: 2 x 2
##         u `mean(y)`
##     <int>     <dbl>
## 1       1    -0.448
## 2       2     1.52
```

"If we try to estimate the mixture model, then, we're doing weighted maximum likelihood, with weights given by the posterior label probabilities.

These, to repeat, depend on the parameters we are trying to estimate, so there seems to be a vicious circle."

In reality we do not know the u labels, nor do we know the mixture proportions (i.e., λ).

We have to start with an initial guess for the labels, estimate the parameters and go through several iterations of the algorithm, updating at each step our current best guess of the group labels and the parameters until we see no substantial improvement in our optimizations.

Soft averaging

Replace the "hard" labels u for each observation (it is either in group 1 or 2) by membership probabilities that sum up to 1.

$p(u, x \mid \theta)$ serves as a weight or "participation" of observation x in the likelihood function

The **marginal likelihood** for the observed y is the sum over all possible values of u of the densities at (y, u):

$$\text{marglike}(\theta; y) = f(y \mid \theta) = \sum_u f(y, u \mid \theta)$$

At each iteration , the current $(*)$ best guesses for the unknown parameters (for instance μ_1^* and μ_2^*) are combined into $\theta^* = (\mu_1^*, \mu_2^*, \lambda^*)$.

We use these to compute the so-called **E**xpectation function $E^*(\theta)$

$$E^*(\theta) = E_{\theta^*,Y}[\log p(u, y \mid \theta^*)] = \sum_u p(u \mid y, \theta^*) \log p(u, y \mid \theta^*),$$

The value of θ that maximizes E^* is found in what is known as the **M**aximization step.

These two iterations (**E** and **M**) are repeated until the improvements are small.

R implementations

Several R packages provide EM implementations, including **mclust**, **EMcluster** and **EMMIXskew**. Choose one and run the EM function several times with different starting values. Then use the function `normalmixEM` from the **mixtools** package to compare the outputs.

Here we show the output from **mixtools**.

```r
library("mixtools")
y = c(rnorm(100, mean = -0.2, sd = 0.5),
      rnorm( 50, mean =  0.5, sd =   1))
gm = normalmixEM(y, k = 2, lambda = c(0.5, 0.5),
     mu = c(-0.01, 0.01), sigma = c(1, 1))
```

```
## number of iterations= 56
```

```r
gm$lambda
```

```
## [1] 0.599 0.401
```

```r
gm$mu
```

```
## [1] -0.196  0.651
```

```r
gm$sigma
```

```
## [1] 0.464 0.999
```

```r
gm$loglik
```

```
## [1] -175
```

Things to learn from the EM algorithm

1. It shows us how we can tackle a difficult problem with too many unknowns by alternating between solving simpler problems. In this way, we eventually find estimates of hidden variables.

2. It provides a first example of *soft* averaging i.e., where we don't decide whether an observation belongs to one group or another, but allow it to participate in several groups by using probabilities of membership as weights, and thus obtain more nuanced estimates.

3. The method employed here can be extended to the more general case of **model-averaging**. We can combine them together into a weighted model. The weights are provided by the likelihoods of the models.

Animation

Wiki

Zero inflated models

$$f_{zero} = \pi I_{\{0\}}(y) + (1 - \pi) f_{count}(y)$$

Zero-inflated Poisson, Hurdle models, Zero-Inflated Negative Binomial.

Example: ChIP-Seq data

Let's consider the example of ChIP-Seq data: sequences of pieces of DNA that are obtained from chromatin immunoprecipitation (ChIP).

This technology enables the mapping of the locations along genomic DNA of transcription factors, nucleosomes, histone modifications, chromatin remodeling enzymes, chaperones, polymerases and other protein.

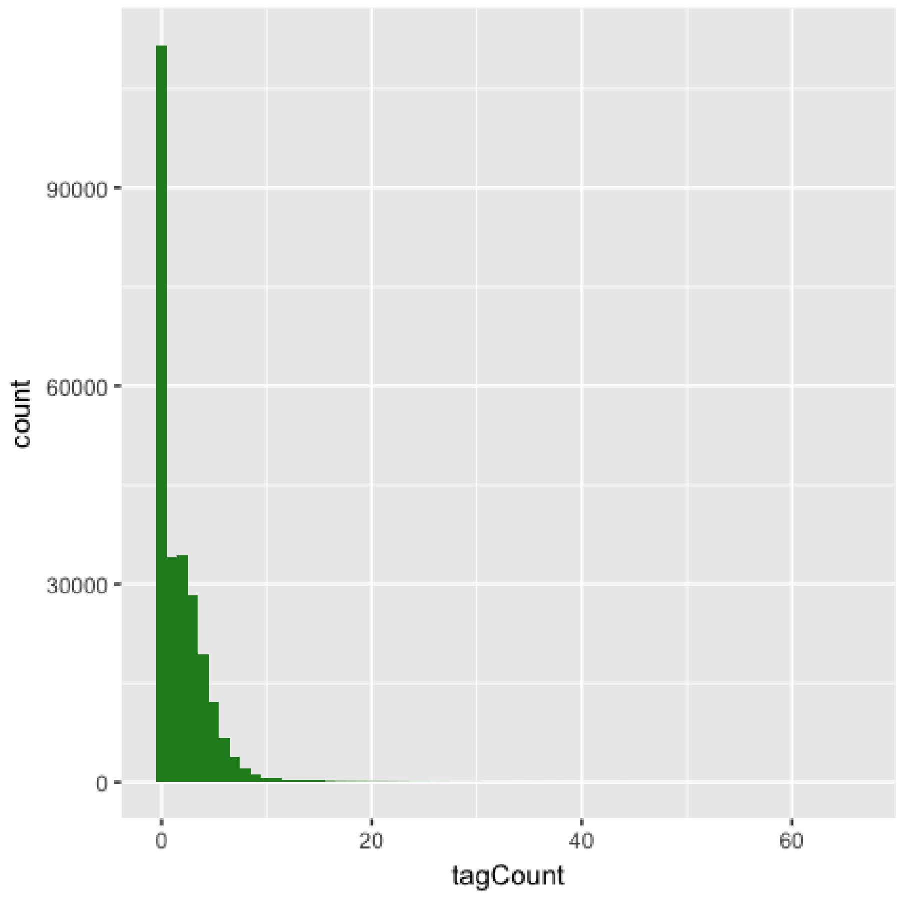
Example from the **mosaicsExample** package, which shows data measured on chromosome 22 from ChIP-Seq of antibodies for the STAT1 protein and the H3K4me3 histone modification applied to the GM12878 cell line.

```
binTFBS
```

```
## Summary: bin-level data (class: BinData)
## ----------------------------------------
## - # of chromosomes in the data: 1
## - total effective tag counts: 462479
##   (sum of ChIP tag counts of all bins)
## - control sample is incorporated
## - mappability score is NOT incorporated
## - GC content score is NOT incorporated
## - uni-reads are assumed
## ----------------------------------------
```

From this object, we can create the histogram of per-bin counts.

```r
bincts = print(binTFBS)
ggplot(bincts, aes(x = tagCount)) +
  geom_histogram(binwidth = 1, fill = "forestgreen")
```



The number of binding sites found in 200nt windows along chromosome 22 in a ChIP-Seq dataset.

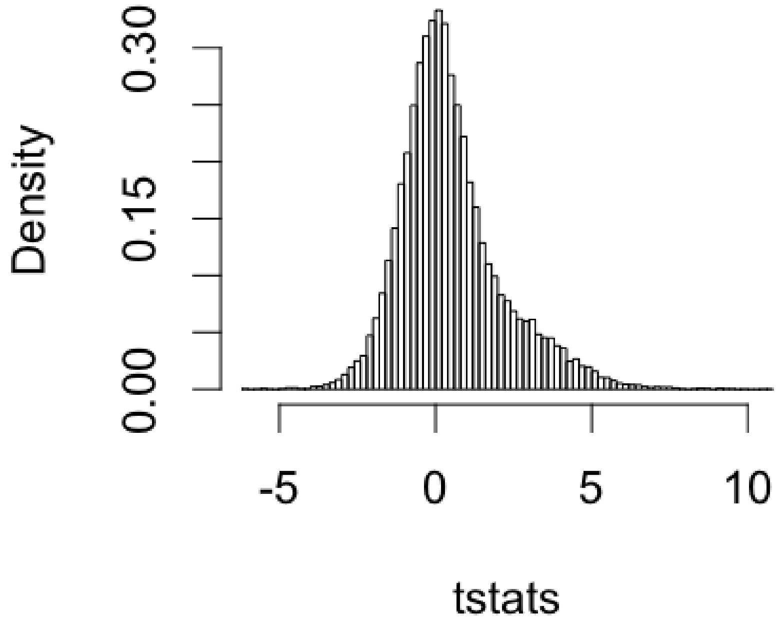
Mixture Models for testing

In a hypothesis testing framing, where we want to find genes that are `on`, mutations that `occur`

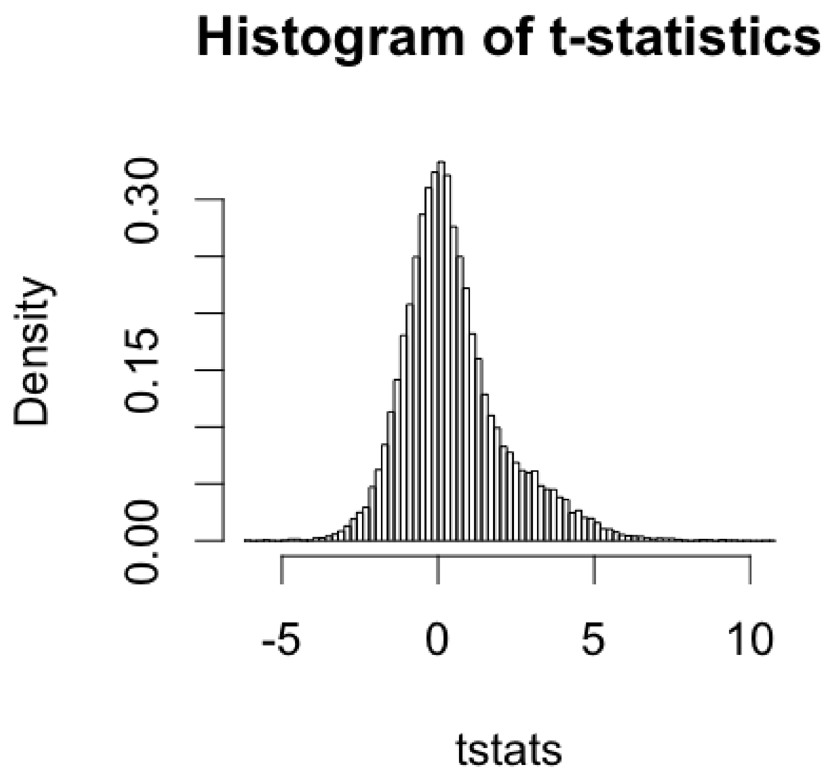
$$f(y) = p_0 f_0(y) + (1-p) f_1(y) \qquad local \quad fdr = \frac{p_0 f_0(y)}{f(y)}$$

```r
ng = 20000 # Number of "genes"
p0 = 0.85
null <- rbinom(ng, size = 1, prob = 1-p0)
# Simulate some t-scores, non-null effect:3
#Non centrality parameter of 3
tstats <- (1-null)*rt(ng, df = 9) + null*rt(ng, df = 9, ncp = 3)
hist(tstats,100,prob=T,main="Histogram of t-statistics")
```

Histogram of t-statistics

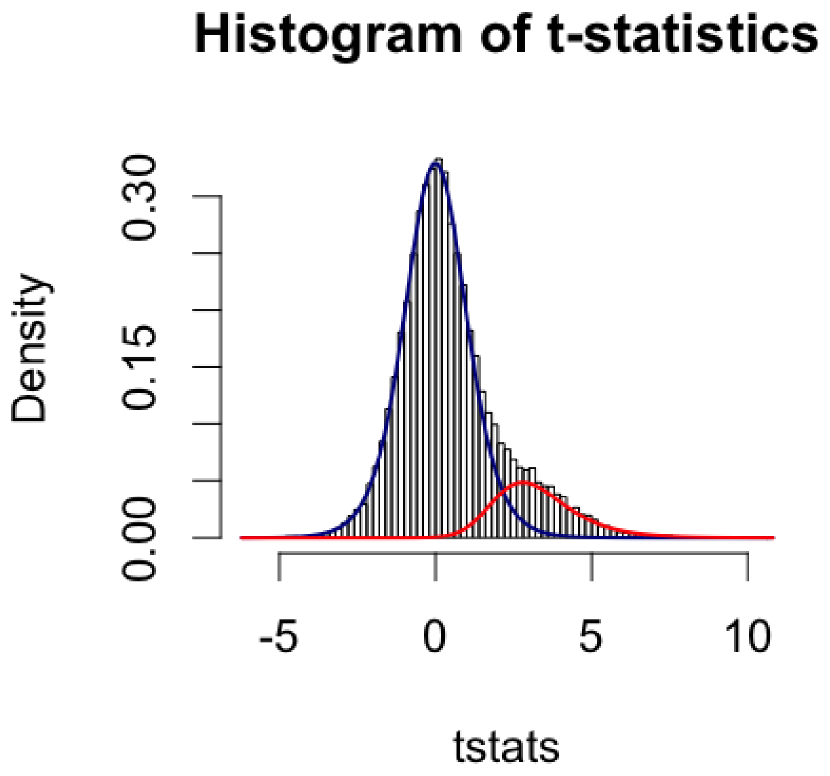


Which are expressed?



Which are on?

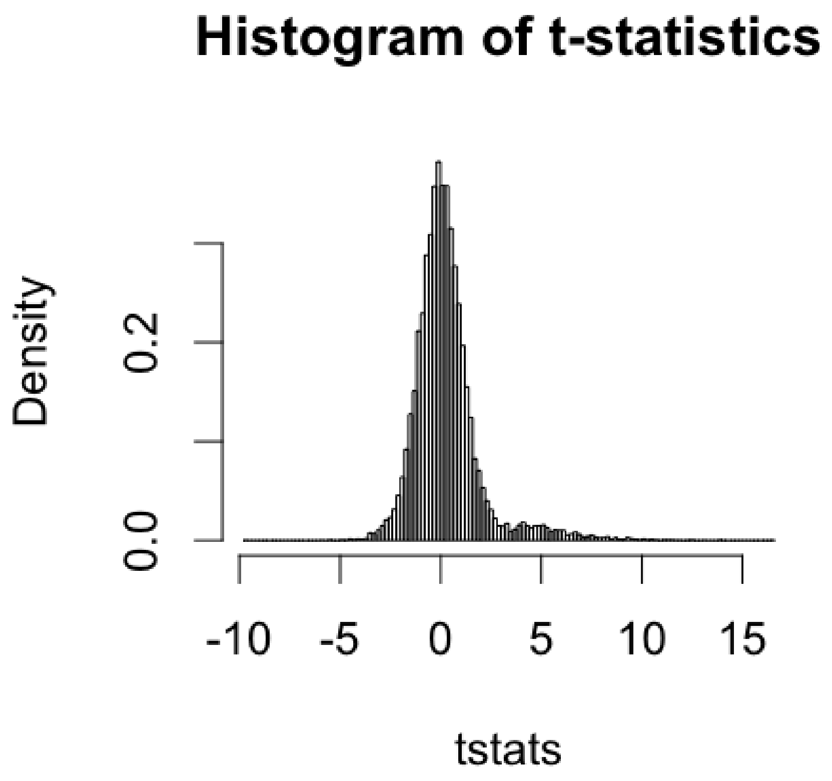
```
hist(tstats,100,prob=T,main="Histogram of t-statistics")
curve(p0*dt(x, df=9),
       col="darkblue", lwd=2, add=TRUE, yaxt="n")
 curve((1-p0)*dt(x, df=9,ncp=3),
       col="red", lwd=2, add=TRUE, yaxt="n")
```



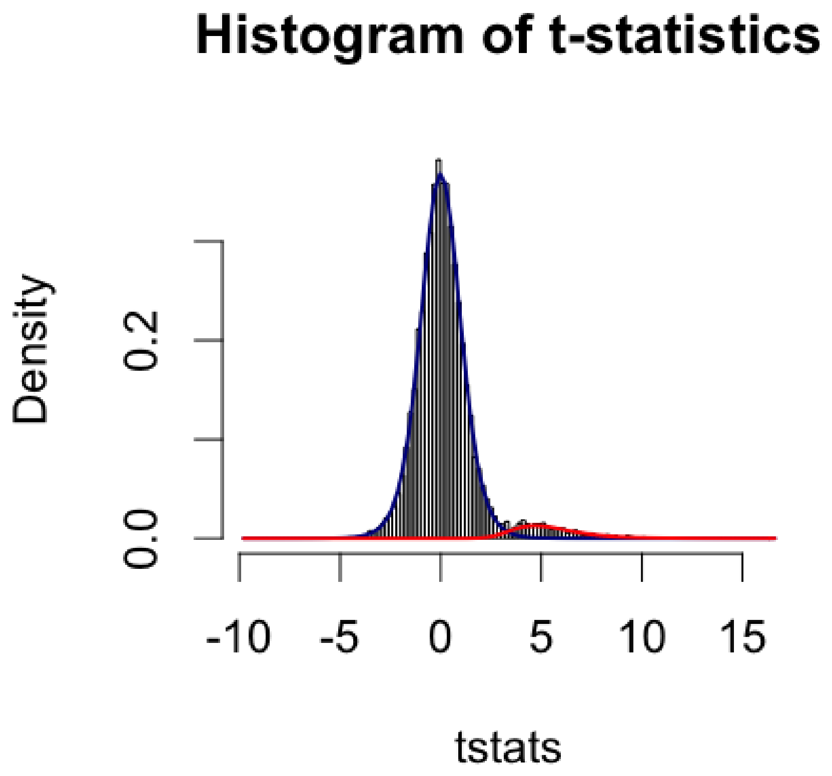
Hidden density.

More Often like this

```r
ng = 20000 # Number of "genes"
p0 = 0.95
null <- rbinom(ng, size = 1, prob = 1-p0)
# Simulate some t-scores, non-null  effect 5
#Non centrality parameter of 3
tstats <- (1-null)*rt(ng, df = 9) + null*rt(ng, df = 9, ncp = 5)
hist(tstats,100,prob=T,main="Histogram of t-statistics")
```



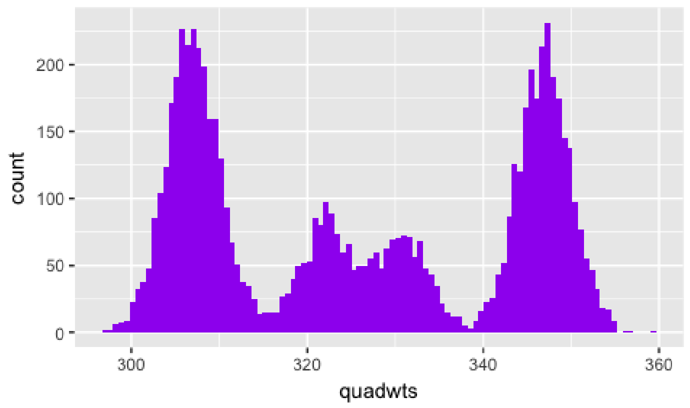
More Often like this



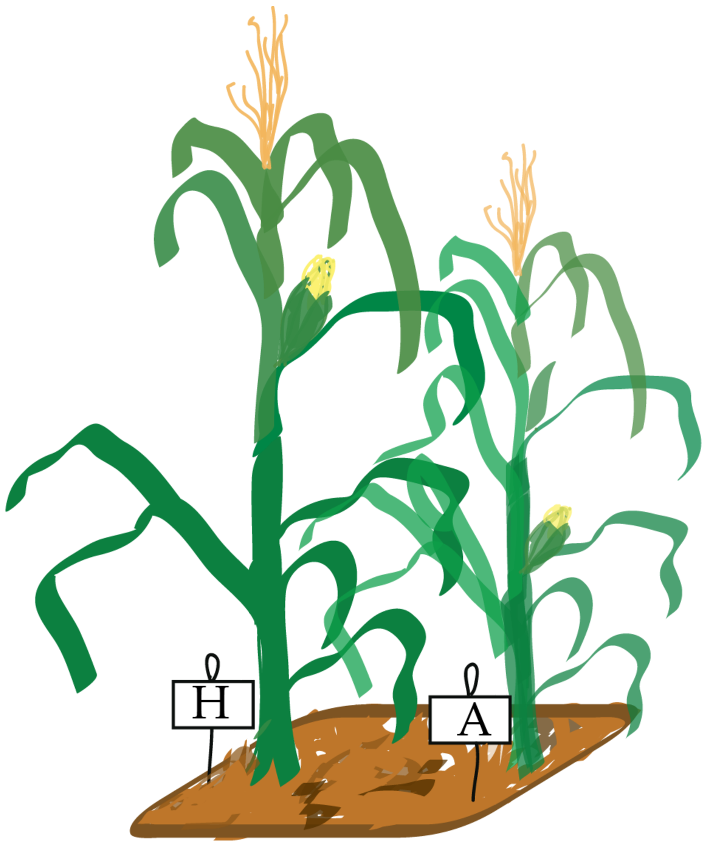
More than two components

Weighing N=7,000 nucleotides obtained from mixtures of deoxyribonucleotide monophosphates (each type has a different weight, measured with the same standard deviation sd=3), could give something like:

```r
masses = c(A =  331, C =  307, G =  347, T =  322)
probs  = c(A = 0.12, C = 0.38, G = 0.36, T = 0.14)
N  = 7000
sd = 3
nuclt   = sample(length(probs), N, replace = TRUE, prob = probs)
quadwts = rnorm(length(nuclt),
                mean = masses[nuclt],
                sd   = sd)
ggplot(tibble(quadwts = quadwts), aes(x = quadwts)) +
  geom_histogram(bins = 100, fill = "purple")
```


count
0
50
100
150
200
300
320
340
360
quadwts

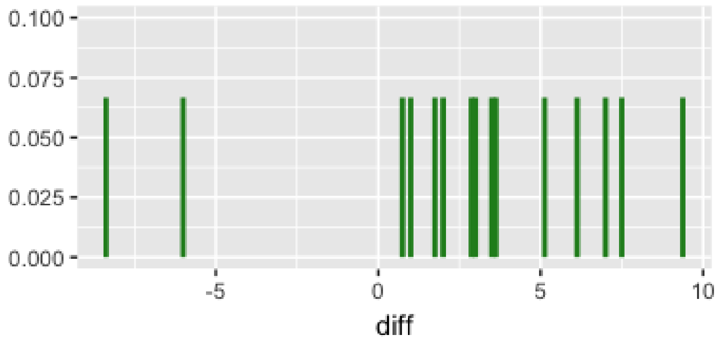
Between finite and infinite : n



```r
library("HistData")
ZeaMays$diff
```

```
##  [1]  6.12 -8.38  1.00  2.00  0.75  2.88  3.50  5.12  1.75
## [10]  3.62  7.00  3.00  9.38  7.50 -6.00
```

```r
ggplot(ZeaMays, aes(x = diff, ymax = 1/15, ymin = 0)) +
  geom_linerange(size = 1, col = "forestgreen") + ylim(0, 0.1)
```



The observed sample can be seen as a mixture of point masses at each of the values (real point masses would be bars without any width whatsoever).

$$\hat{F}_n(x) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{x \le x_i},$$

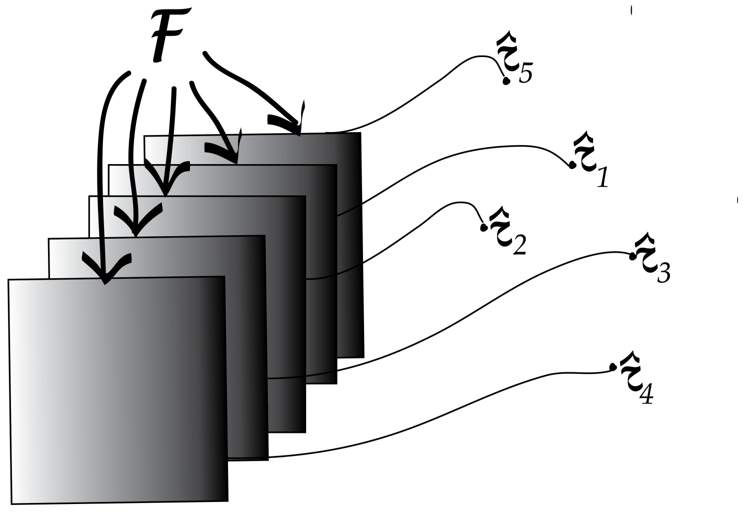
We can also write the *density* of our sample as

$$\hat{f}_n(x) = \frac{1}{n} \sum_{i=1}^{n} \delta_{x_i}(x)$$

In general, the density of a probability distribution is the derivative (if it exists) of the distribution function. We have applied this principle here: the density of the distribution.

We could do this since one can consider the function δ_a the “derivative” of the step function $\mathbb{1}_{x \le a}$: it is completely flat almost everywhere, except at the one point a where there is the step, and where its value is “infinite”

Our data sample can be considered a mixture of n **point masses** at the observed values $x_1, x_2, \ldots, x_n$.

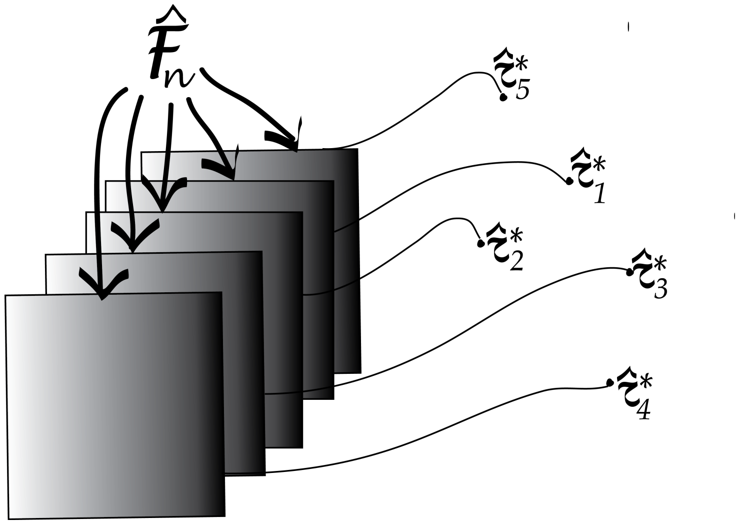


The value of a statistic τ is estimated from data (the grey matrices) generated from an underlying distribution F.

Statistics of our sample, such as the mean, minimum or median, can now be written as a function of the ECDF, for instance, $\bar{x} = \int \delta_{x_i}(x)\,\mathrm{d}x$. As another instance, if n is an odd number, the median is $x_{(\frac{n+1}{2})}$, the value right in the middle of the ordered list.

The true **sampling distribution** of a statistic $\hat{\tau}$ is often hard to know as it requires many different data samples from which to compute the statistic.

The **bootstrap** principle approximates the true sampling distribution of $\hat{\tau}$ by creating new samples drawn from the empirical distribution built from the original sample. We *reuse* the data (by considering it a mixture distribution of δs) to create new "datasets" by taking samples and looking at the sampling distribution of the statistics $\hat{\tau}^*$ computed on them. This is called the **nonparametric bootstrap** resampling approach, see for a complete reference. It is very versatile and powerful method that can be applied to basically any statistic, no matter how complicated it is.



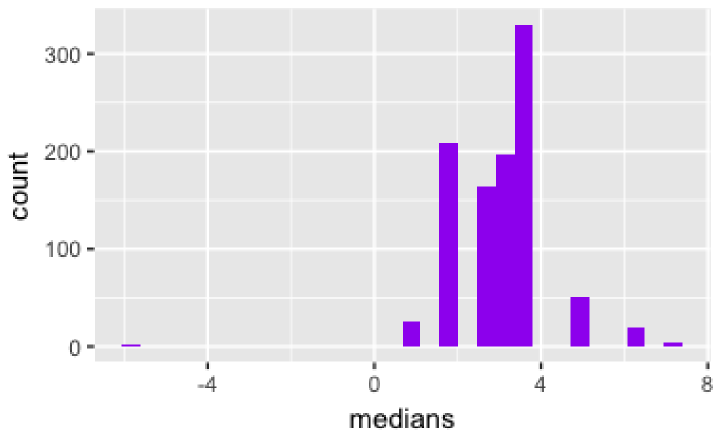
The bootstrap simulates the sampling variability by drawing samples not from the empirical distribution.

The sampling distribution of the median of the Zea Mays differences.

We use similar simulations to those in the previous sections: Draw $B = 1000$ samples of size 15 from the 15 values (each being a component in the 15-component mixture).

Then compute the 1000 medians of each of these samples of 15 values and look at their distribution: this is the bootstrap sampling distribution of the median.

```r
B = 1000
meds = replicate(B, {
  i = sample(15, 15, replace = TRUE)
  median(ZeaMays$diff[i])
})
ggplot(tibble(medians = meds), aes(x = medians)) +
  geom_histogram(bins = 30, fill = "purple")
```



The bootstrap sampling distribution of the median of the Zea Mays differences.

Estimate a 99% confidence interval for the median based on these simulations. What can you conclude from looking at the overlap between this interval and 0?

Why nonparametric?

In theoretical statistics, nonparametric methods are those that have infinitely many degrees of freedom or numbers of unknown parameters.

Despite their name, nonparametric methods are not methods that do not use parameters: all statistical methods estimate unknown quantities.

In practice, we do not wait for infinity; when the number of parameters becomes as large or larger than the amount of data available, we say the method is nonparametric. The bootstrap uses a mixture with n components, so with a sample of size n, it qualifies as a nonparametric method.

What are the two types of error that can occur when using the bootstrap?

Monte Carlo simulations of subsets of the data by resampling randomly approximate the exhaustive bootstrap.

Increasing the size of `nboot` argument in the `bootstrap` function reduces the Monte Carlo error, however the exhaustive bootstrap is still npt exact: we are still using an approximate distribution function, that of the data instead of the true distribution. If the sample size is small or the original sample biased, the approximation can still be quite poor, no matter how large we choose `nboot`.

Infinite Mixtures

Mixtures can be useful without having to find who came from which distribution.

We decomposed the mixture model into a two step process, we extend the description to cases where there could be many more components in the model, suppose that instead of flipping a coin, we picked a ball from an urn with the mean and variances written on it, if half the balls were marked $(\mu_1 = 1, \sigma^2 = 0.1)$ and the other half $(\mu_1 = 2, \sigma^2 = 0.1)$ this would actually be equivalent to our original mixture.

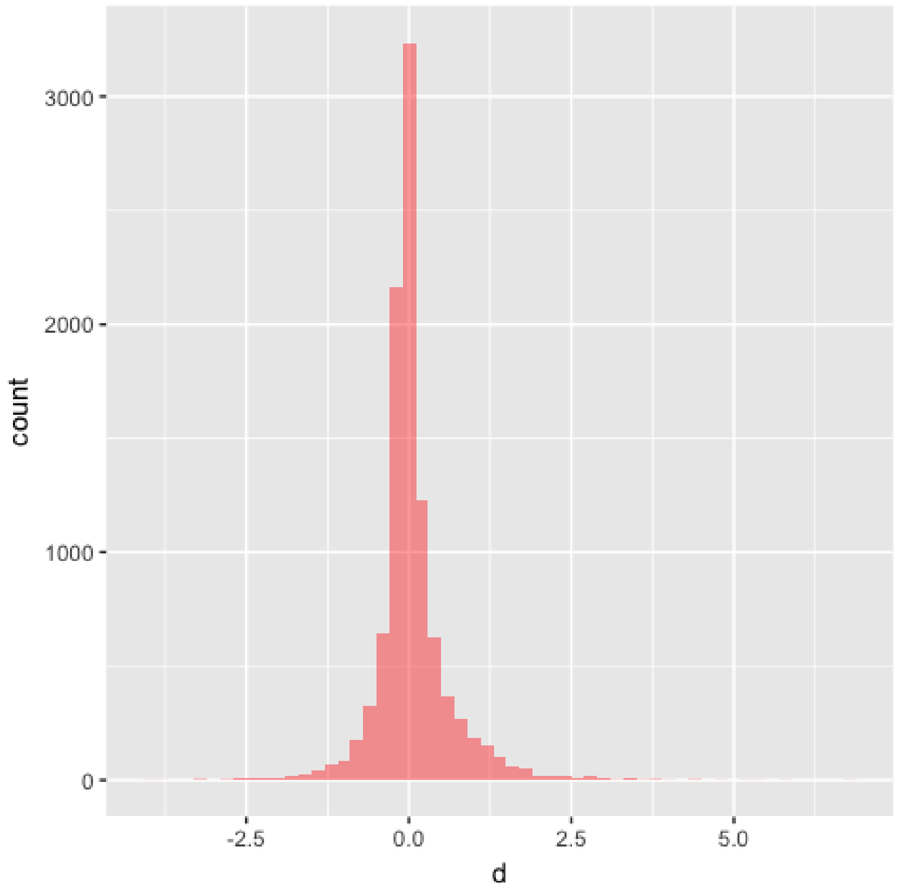
Infinite Mixtures: random effects and hierarchical models

This gives us much more freedom, all the balls could have different parameter-numbers on them and these numbers could actually be drawn at random from a special distribution. This is often called a hierarchical model because it is a two step process where one step has to be done before the next: generate the numbers on the `parameter balls', draw a ball, then draw a random number according to that parametric distribution.

By using this generating mixture we can go as far as generating a new parameter for all the picks from our distribution.

Infinite Mixture of Normals

```
theta=-0.1;mu=0.15;sigma=0.43
W=rexp(10000,1)#Choose some Wis
output=rnorm(10000,theta+mu*W,sigma*W)
dat = data.frame(d=output)
ggplot(dat,aes(x=d)) +
    geom_histogram(data=subset(dat),fill = "red", alpha = 0.4,binwidth = 0.2)
```



Laplace distribution from an infinite mixture of Normals

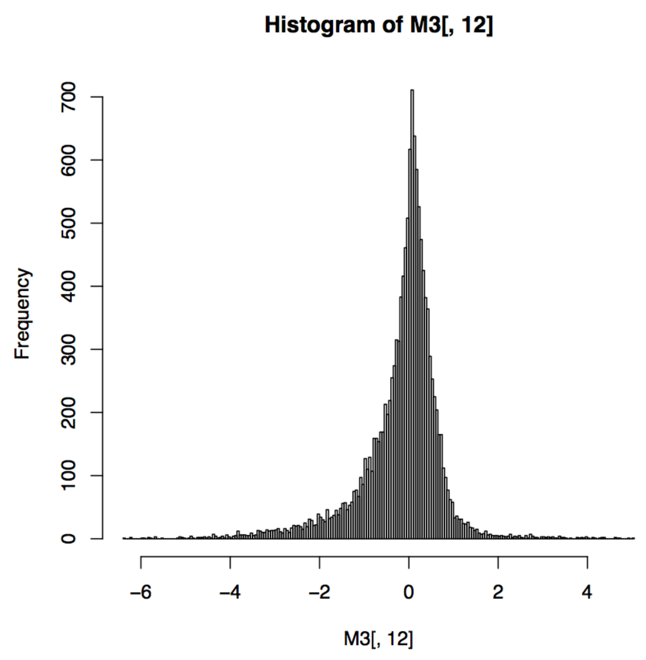
This is actually a rather useful mixture, it turns out such a mixture has a name and well known properties, it is called the Laplace distribution of errors.

Instead of using the mean and the variance to summarize it, we should use the median and the mean absolute deviation as they have better properties.

Microarray Data

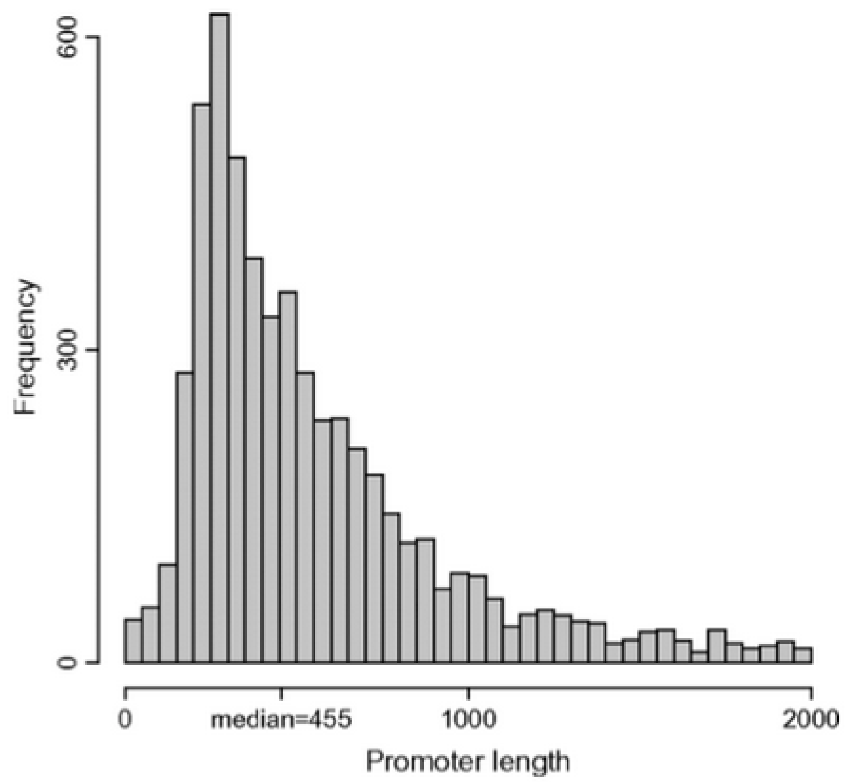
Looking at all the gene expression values from a Microarray, one gets a distribution like this:

```r
knitr::include_graphics("http://bios221.stanford.edu/images/tcellhist.png")
```



Promoter Lengths

Histogram of the 5,735 Saccharomyces cerevisiae promoters used in this study.



Kristiansson E et al. Mol Biol Evol 2009;26:1299-1307

MBE MOLECULAR BIOLOGY AND EVOLUTION

Histogram of Promoter Lengths

Historical case of preferred Median

Note:Laplace knew already that the error distribution that has the location parameter the median and the scale parameter the `mad` has density:

$$f_Y(y) = \frac{1}{2\phi} exp[-\frac{|y-\theta|}{\phi}], \qquad \phi > 0$$

This is a good example where we can look at the generative process and see why the variance and mean are linked.

The expectation and variance of an AL(θ, μ, σ) are

$$E(Y) = \theta + \mu \text{ and } var(Y) = \sigma^2 + \mu^2$$

Note the variance is not independent of the mean unless $\mu = 0$ – the case of the symmetric Laplace Distribution. This is a feature of the distribution that makes it useful.

In practical situations, when looking at gene expression in microarrays, taxa counts in 16sRNA studies, we will see that the variances depend on the mean, we will need models that can accomodate for this problem.

Representation of Laplace's First Error Distribution

Useful representation:

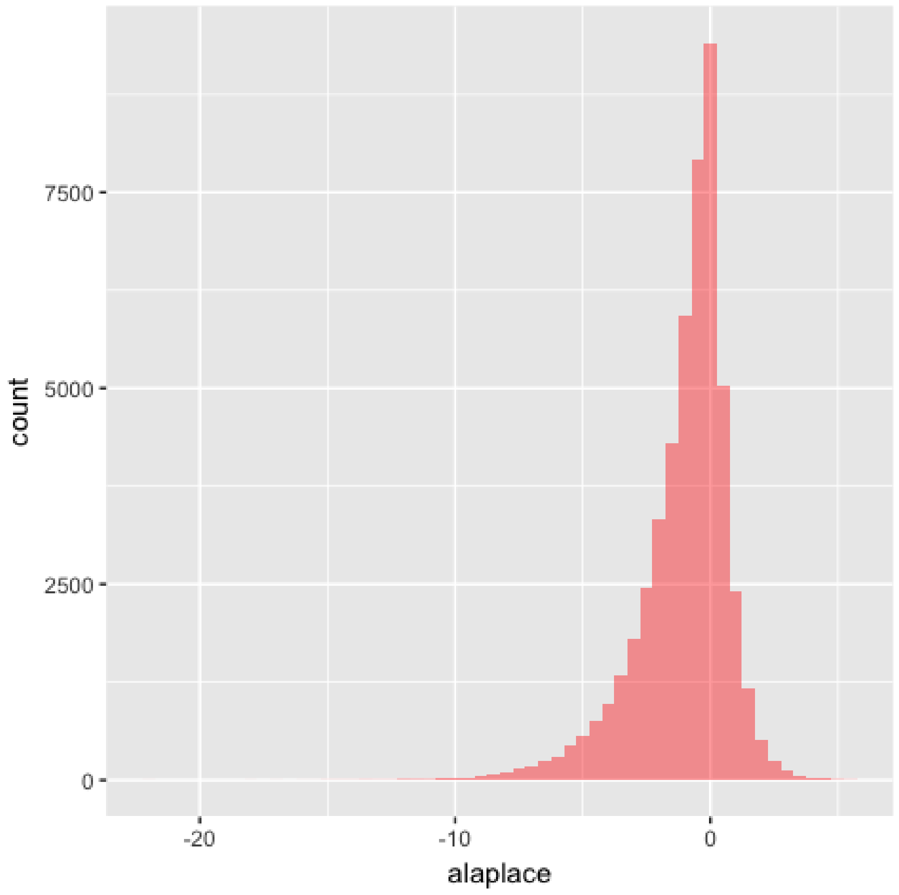
$$Y = \sqrt{X} \cdot Z, \qquad X \sim Exp(1), \qquad Z \sim N(0, 1)$$

A mixture of Normals whose scale parameters vary from an exponential.

The probe sequences are of different length and so have varying hybridization precisions.

Generating Assymetric Laplace Distributions in R

```
require(VGAM)
alaplace=ralap(50000,location=0,scale=1.5,kappa=1.6);
#hist(alaplace,100)
dat = data.frame(alaplace=alaplace)
ggplot(dat,aes(x=alaplace)) +
    geom_histogram(data=dat,fill = "red", alpha = 0.4,binwidth = 0.5)
```



Example of use: Parametric Bootstrap for Power Simulations

It is very useful when we want to design an experiment to know how well it will detect differences of different sizes, often called the effect size. Fir that, we generate data under various null hypotheses that have as many features similar to the original data:

- Same error distribution (form and moments), with all the different sources of error incorporated into the study.
- Vary some underlying tuning parameters, sample sizes or number of repetitions, numbers of reads,….

Pretend that the data come from a parametric family F_θ.

Replace in the generation of `new’ data, the unknown parameters by $F_{\hat{\theta}}$.

Now we can generate a whole set of simulated data to find the power of our analyses at various levels of sample size and effect size.

Power simulation for Microarrays

Model, after renormalization, and eventual variance stabilization.

$$m_{jk} = m_k \epsilon_k + \sqrt{\epsilon_k} Z_{kj}, \qquad Z_{kj} \sim \mathcal{N}(0, \Sigma)$$

Genes are k, $k = 1....n$. Estimate the parameters, covariance matrix, medians, median absolute deviations. This model is added onto the variance stabilization model.

One may want to change the covariance matrix to adjust for the hypotheses being tested.

Plug these into an `Assymetric  Laplace generator`.

We can compare the number of genes chosen with the parametric model with a given covariance structure as the one we get with the data.

The Poisson Gamma Mixture Model

Count data are often messier than simple Poisson and Binomial distributions serve as building blocks for more sophisticated models called mixtures.

What’s a Poisson-Gamma mixture model used for?

- Overdispersion (in Ecology).
- Simplest Mixture Model for Counts.
- Different evolutionary mutation rates.
- Abundance data of 16sRNA measured taxa.

Gamma Distribution: two parameters (shape and scale)

wikigamma

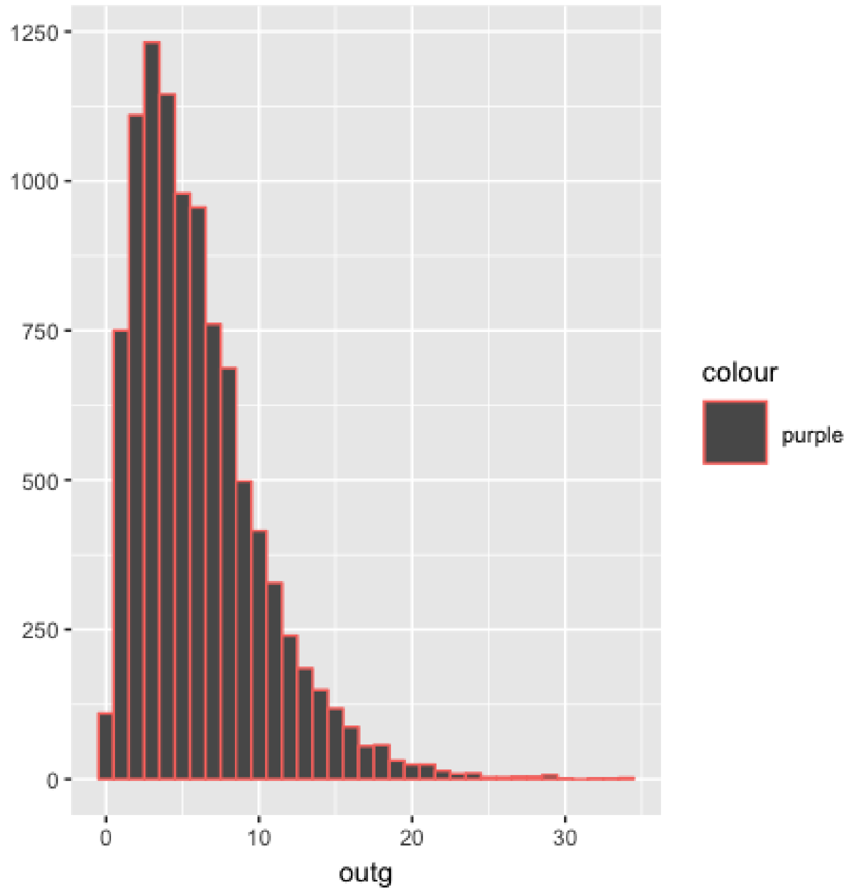
The Gamma distribution is used to model continuous variables with any range of positive values. Typical quantities that follow this distribution are waiting times and survival times.

It is often used in Bayesian inference to model the variability of the Poisson or Exponential parameters (conjugate family).

This is not unrelated to why we use it for mixture modeling.

Simulations and Examples

```r
require(ggplot2)
nr=10000;set.seed(20130607)
outg=rgamma(nr,shape=2,scale=3)
#
p=qplot(outg,geom="histogram",binwidth=1,color="purple")
p
```



Gamma(2,3)

Fitting distributions with R:

```
require(MASS)
## avoid spurious accuracy
op = options(digits = 3)
set.seed(123)
x = rgamma(100, shape = 5, rate = 0.1)
fitdistr(x, "gamma")
```

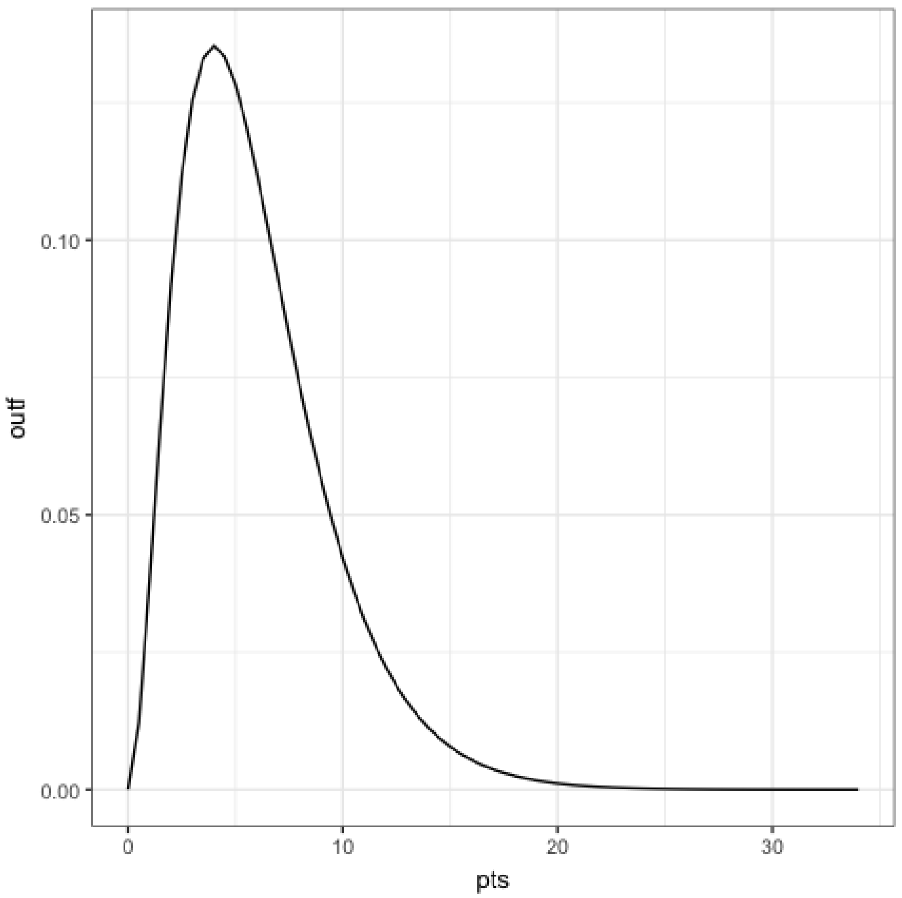
```
##     shape      rate
##    6.4870    0.1365
##   (0.8946)  (0.0196)
```

```
## now do this directly with more control.
fitdistr(x, dgamma, list(shape = 1, rate = 0.1), lower = 0.001)
```

```
##     shape      rate
##    6.4869    0.1365
##   (0.8944)  (0.0196)
```

#Gamma Density

```
require(ggplot2)
pts=seq(0,max(outg),0.5)
outf=dgamma(pts,shape=3,scale=2)
p=qplot(pts,outf,geom="line")
p+ theme_bw(10)
```



Theoretical Gamma(2,3)

We are going to use this type of variability for the variation in our Poisson parameters.

Gamma mixture of Poissons: a hierarchical model

This is a two step process:

1. Generate a whole set of Poisson parameters: $\lambda_1, \lambda_2, \ldots \lambda_{90}$ from a Gamma(2,3) distribution.

2. Generate a set of Poisson(λ_i) random variables.

Two step generation with R

```
ng=90;set.seed(1001015)
lambdas=rgamma(ng,shape=2,scale=3)
####Rate is usually the second it is 1/scale
veco=rep(0,ng)
for (j in (1:ng)){
  veco[j]=rpois(1,lambda=lambdas[j]) }
require(vcd)
goodnb=goodfit(veco,"nbinomial")
goodnb
```

```
##
## Observed and fitted values for nbinomial distribution
## with parameters estimated by `ML'
##
##  count observed fitted pearson residual
##      0       10  7.673           0.8399
##      1        6  9.895          -1.2383
##      2       12 10.191           0.5668
##      3        9  9.613          -0.1977
##      4        9  8.652           0.1184
##      5        6  7.562          -0.5681
##      6        6  6.479          -0.1881
##      7        6  5.471           0.2264
##      8        6  4.568           0.6698
##      9        3  3.782          -0.4022
##     10        2  3.109          -0.6291
##     11        2  2.542          -0.3397
##     12        2  2.067          -0.0469
##     13        2  1.675           0.2512
##     14        3  1.352           1.4171
##     15        3  1.088           1.8327
##     16        1  0.873           0.1354
##     17        2  0.699          -0.7622
```

`nbinomial` stands for the Negative Binomial and is another distribution for count data. In general it is used to model the number of trials until we obtain a success in a Binomial (p) experiment.

`rnegbin`, `dnegbin`,`pnegbin` are the corresponding functions.

Fitting a Negative Binomial with `fitdistr`:

```
set.seed(123)
x4 = rnegbin(500, mu = 5, theta = 4)
fitdistr(x4, "Negative Binomial")
```

```
##     size      mu
##   4.216   4.945
##  (0.504) (0.147)
```

The Mathematical explanation

The Negative Binomial probability distribution function

$$dnbinom(k, size = r, prob = p) = \binom{r+k-1}{k} p^r (1-p)^k$$

This can be interpreted as the probability of waiting to have k failures until the rth success occurs. Success having probability p

Negative Binomial distribution

We can compute the theoretical fit of the Negative Binomial with the data generated as Poisson-gamma using a rootogram.

```
summary(goodnb)
```

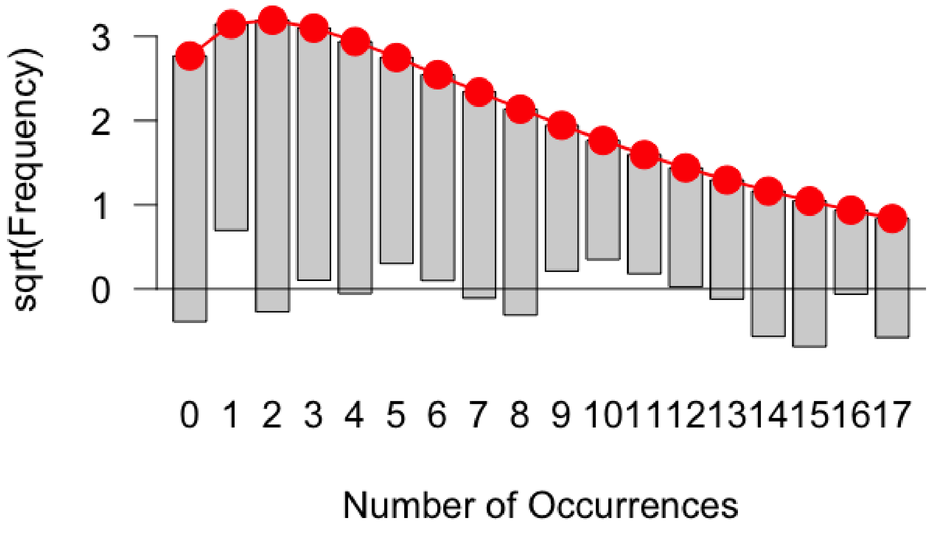
```
##
##   Goodness-of-fit test for nbinomial distribution
##
##                       X^2 df P(> X^2)
## Likelihood Ratio 15.2 15    0.435
```

```
goodnb$par
```

```
## $size
## [1] 1.67
##
## $prob
## [1] 0.23
```

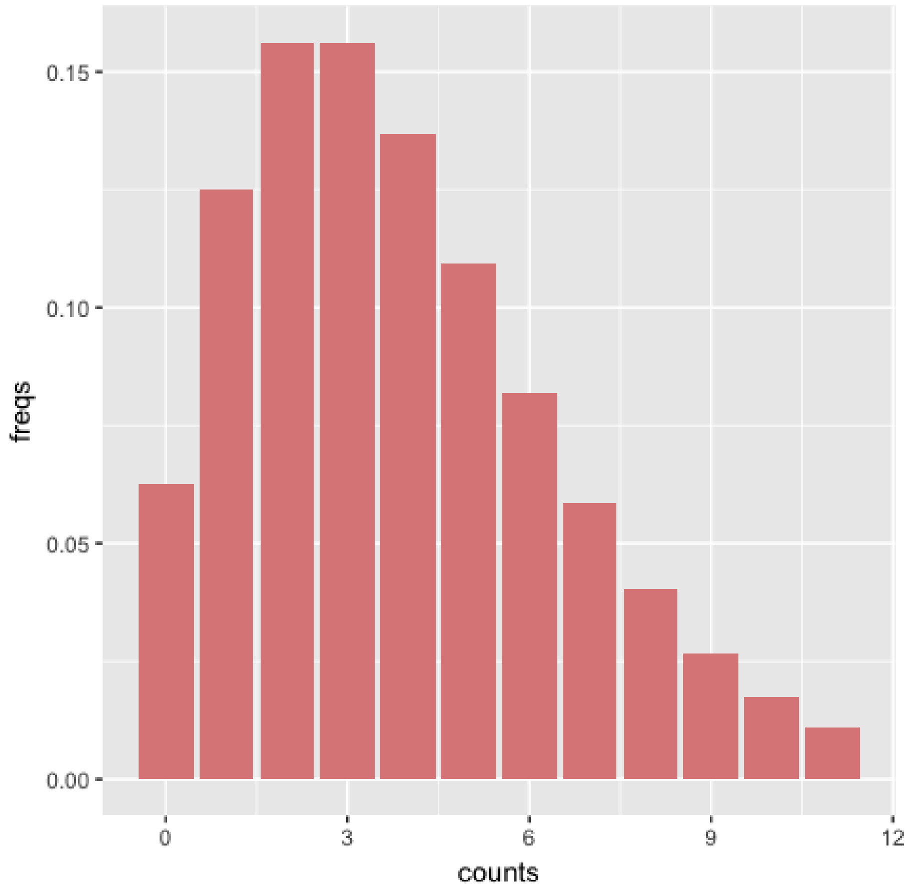
#Visual Goodness of Fit Test

```
#table(veco)
plot(goodnb)
```

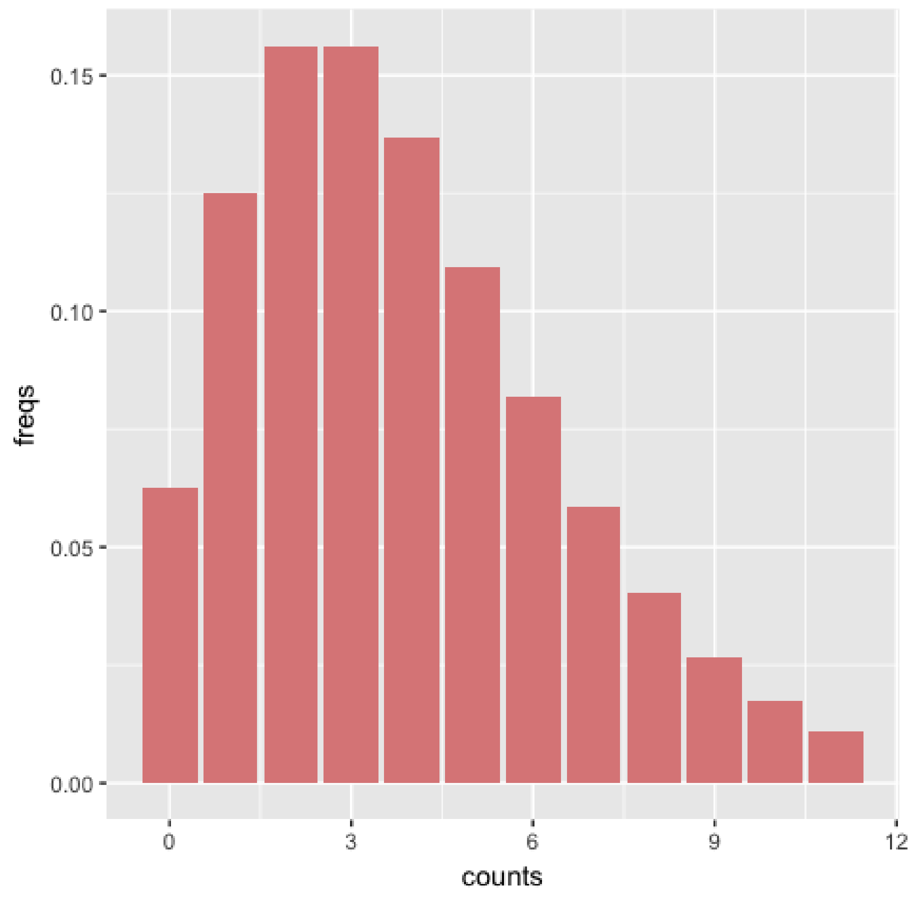


ggplot version of negative binomial histogram

```r
set.seed(321)
cts=0:11
out=dnbinom(cts,size=4,p=0.5)
dfnb=data.frame(counts=cts,freqs=out)
ggplot(data=dfnb, aes(x=counts, y=freqs)) +
  geom_bar(stat="identity",fill="#DD8888")
```



#Barplot from a theoretical negative binomial with p=0.5



Gamma Mixture of Poissons: the densities

Theoretically taking a mixture of Poisson(μ) variables where $\mu \sim Gamma(\alpha = k, \beta)$. The final distribution is the result of a two step process: 1. Generate a Gamma(α, β) distributed number, call it z from density

$$dgamma(z, \alpha, \beta) = \frac{\beta^{\alpha}}{\Gamma(\alpha)} z^{\alpha-1} e^{-\beta z}$$

2. Generate a number from the Poisson(z) distribution with parameter z, call it y.

$$dpois(y, \lambda = z) = \frac{z^{y} e^{-z}}{y!}$$

If z only took on integer numbers from 0 to 10 then we would write

$$P(Y = y) = P(Y = y|z = 0)P(z = 0) + P(Y = y|z = 1)P(z = 1)$$

$$\ldots + P(Y = y|z = 10)P(z = 10)$$

It's not quite that simple and we have to write it out as an integral sum instead of a discrete sum.

Gamma-Poisson is Negative Binomial

We call the distribution of Y the marginal and it is given by

$$P(Y=y)=\int dgamma(z,a,b)dpois(y,z)dz=\int \frac{b^a}{\Gamma(a)}z^{a-1}e^{-bz}\frac{z^y e^{-z}}{y!}dz$$

Remembering that $\Gamma(a)=(a-1)!$

$$P(Y=y)=\frac{b^a}{(a-1)!y!}\int z^{y+a-1}e^{-z(b+1)}dz$$

Now we use that the integral

$$\int z^{r-1}e^{-wz}dz=\frac{\Gamma(r)}{w^r} \qquad \text{so}$$

$$P(Y=y)=\frac{(y+a-1)!}{(a-1)!y!}\frac{b^a}{(b+1)^a(b+1)^y}$$

$$=\binom{y+a-1}{y}(\frac{b}{b+1})^a(1-\frac{b}{b+1})^y$$

giving the negative binomial with size parameter a and $p=\frac{b}{b+1}$.

Example of use: Gamma Exponential Noise Models (used for PCR)

We will see another example of using a Gamma to generate a mixture, but this time for the error model for PCR where the second distribution is exponential with the parameter generated by a Gamma. (See link Gamma-Exponential Noise Models useful to model noise in PCR experiments.)

Read Noise Modeling

Negative Binomial :hierarchical mixture for reads

In biological contexts such as RNA-seq and microbial count data the negative binomial distribution arises as a hierarchical mixture of Poisson distributions. This is due to the fact that if we had technical replicates with the same read counts, we would see Poisson variation with a given mean. However, the variation among biological replicates and library size differences both introduce additional sources of variability.

To address this, we take the means of the Poisson variables to be random variables themselves having a Gamma distribution with (hyper)parameters shape r and scale $p/(1-p)$. We first generate a random mean, λ, for the Poisson from the Gamma, and then a random variable, k, from the Poisson(λ).

#The marginal distribution:

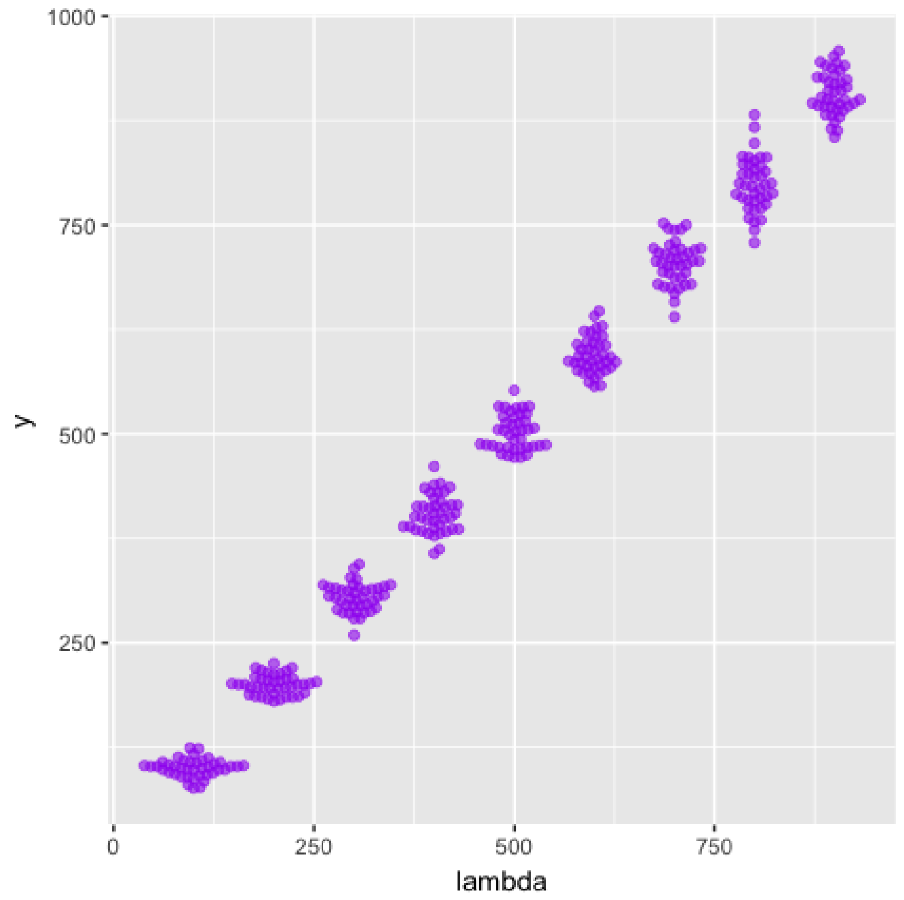
$$
\begin{aligned}
P(X=k) &= \int_0^\infty Po_\lambda(k) \times \gamma_{(r,\frac{p}{1-p})} d\lambda \\
&= \int_0^\infty \frac{\lambda^k}{k!} e^{-\lambda} \times \frac{\lambda^{r-1} e^{-\lambda \frac{1-p}{p}}}{(\frac{p}{1-p})^r \Gamma(r)} d\lambda \\
&= \frac{(1-p)^r}{p^r k! \Gamma(r)} \int_0^\infty \lambda^{r+k-1} e^{-\lambda/p} d\lambda \\
&= \frac{(1-p)^r}{p^r k! \Gamma(r)} p^{r+k} \Gamma(r+k) \\
&= \frac{\Gamma(r+k)}{k! \Gamma(r)} p^k (1-p)^r
\end{aligned}
$$

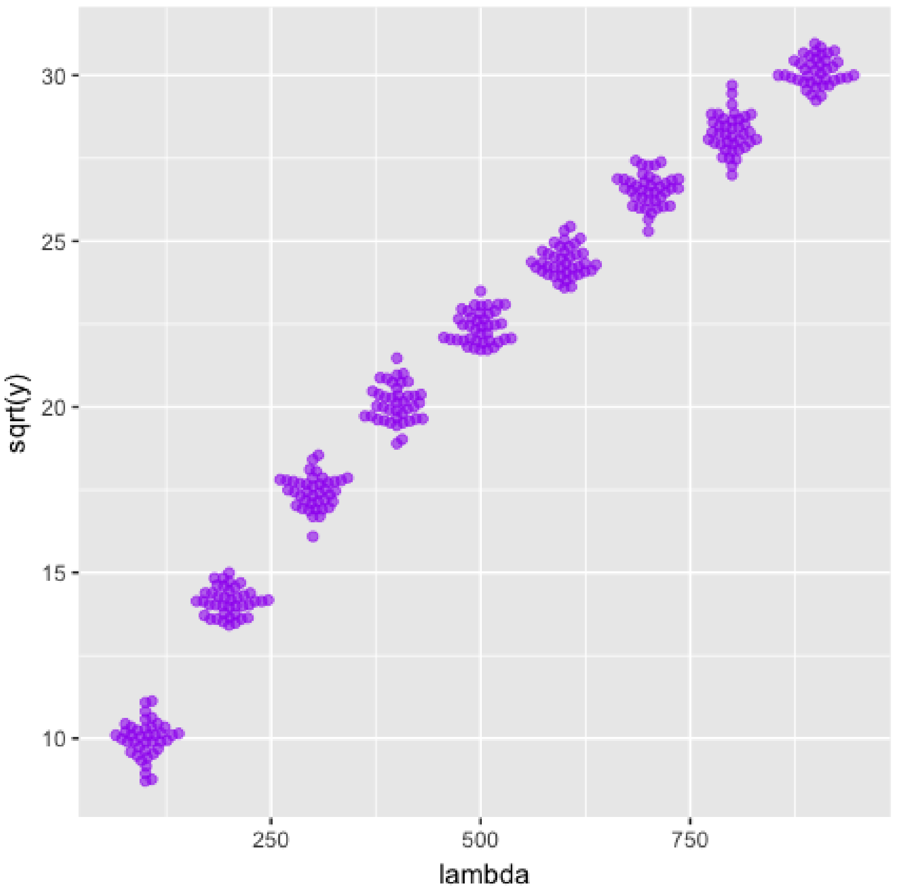
Variance Stabilization

Take for instance different Poisson variables with mean μ_i.

However, if the square root transformation is applied to each of the variables, then the transformed variables will have approximately constant variance

```r
lambdas = seq(100, 900, by = 100)
simdat = lapply(lambdas, function(l)
    tibble(y = rpois(n = 40, lambda=l), lambda = l)
  ) %>% bind_rows
library("ggbeeswarm")
ggplot(simdat, aes(x = lambda, y = y)) +
  geom_beeswarm(alpha = 0.6, color = "purple")
ggplot(simdat, aes(x = lambda, y = sqrt(y))) +
  geom_beeswarm(alpha = 0.6, color = "purple")
```

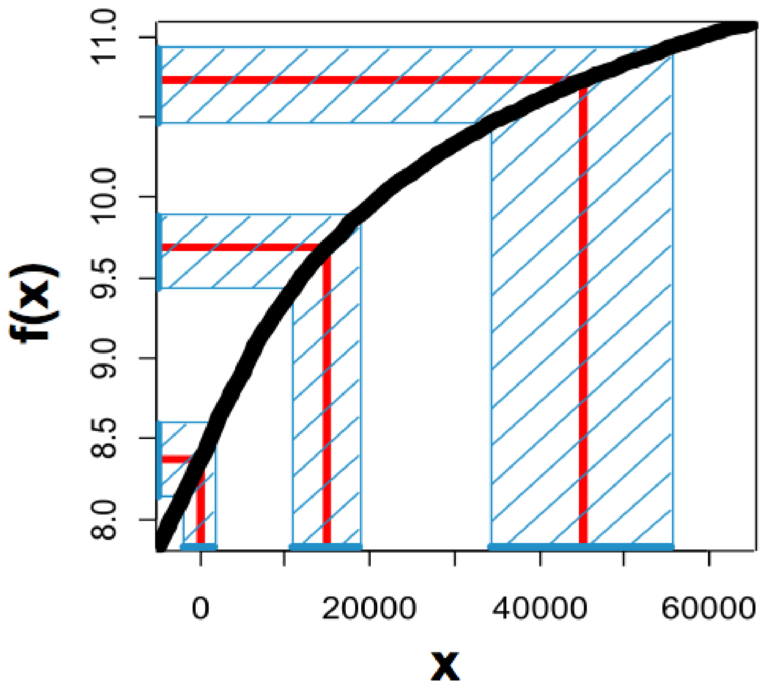

1000
750
500
250
y
0
250
500
750
lambda


sqrt(y)
30
25
20
15
10
250
500
750
lambda

```r
summarise(group_by(simdat, lambda), sd(y), sd(2*sqrt(y)))
```

```
## # A tibble: 9 x 3
##   lambda `sd(y)` `sd(2 * sqrt(y))`
##    <dbl>   <dbl>             <dbl>
## 1    100    10.8             1.09
## 2    200    11.7             0.824
## 3    300    16.9             0.973
## 4    400    22.5             1.12
## 5    500    21.0             0.936
## 6    600    21.9             0.893
## 7    700    25.6             0.967
## 8    800    32.0             1.13
## 9    900    25.7             0.852
```

▶ variance stabilizing transformation



#Delta Method

Transformation function g, and assume it's differentiable. Random variables X_i, with means μ_i and variances v_i, v_i and μ_i are related by a functional relationship $v_i = v(\mu_i)$. For values of X_i in the neighborhood of its mean μ_i,

$$g(X_i) = g(\mu_i) + g'(\mu_i)(X_i - \mu_i) + \ldots$$

where the dots stand for higher order terms that we can neglect. The variances of the transformed values are then

$$\mathrm{Var}(g(X_i)) \simeq g'(\mu_i)^2 \ \mathrm{Var}(X_i) = g'(\mu_i)^2 \ v(\mu_i),$$

where we have used the rules $\mathrm{Var}(X - c) = \mathrm{Var}(X)$ and $\mathrm{Var}(cX) = c^2 \ \mathrm{Var}(X)$ that hold whenever c is a constant number. Requiring that this be constant leads to the differential equation

Delta method transformation (analytic)

$$g'(x) = \frac{1}{\sqrt{v(x)}}.$$

For any given mean-variance relationship $v(\mu)$, we can solve this for the function g. Let's check this for some simple cases:

- if $v(\mu) = \mu$ (Poisson), we recover $g(x) = \sqrt{x}$, the square root transformation.
- If $v(\mu) = c\,\mu^2$, solving the differential equation above gives $g(x) = \log(x)$.

The logarithm transformation is so popular:
- it acts as a variance stabilizing transformation whenever the data have a constant coefficient of variation, (when the standard deviation is proportional to the mean).

Variance Stabilization for Negative Binomial

Anscombe showed that there are several transformations that stabilize the variance of the Negative Binomial depending on the values of the parameters m and r, where $r = \frac{1}{\phi}$, sometimes called the of the Negative Binomial. For large m and constant $m\phi$, the transformation

$$arcsinh\sqrt{(\frac{1}{\phi} - \frac{1}{2})\frac{x + \frac{3}{8}}{\frac{1}{\phi} - \frac{3}{4}}}$$

gives a constant variance around $\frac{1}{4}$. Whereas for m large and $\frac{1}{\phi}$ not substantially increasing, the following simpler transformation is preferable

$$log(x + \frac{1}{2\phi})$$

Modeling read counts

If we have technical replicates with the same number of reads s_j, we expect to see Poisson variation with mean $\mu = s_j u_i$, for each gene or taxa or locus i whose incidence proportion we denote by u_i.

Thus the number of reads for the sample j and taxa i would be

$$K_{ij} \sim \text{ Poisson } (s_j u_i)$$

We use the notational convention that lower case letters designate fixed or observed values whereas upper case letters designate random variables.

For biological replicates within the same group – such as treatment or control groups or the same environments – the proportions u_i will be variable between samples.

Model for abundances, RNA-seq etc..

Variability of Poisson taken to be the Gamma distribution, as it has two parameters and can be adapted to many distributional shapes.

Call the two parameters r_i and $\frac{p_i}{1-p_i}$. So that U_{ij} the proportion of taxa i in sample j is distributed according to Gamma$(r_i, \frac{p_i}{1-p_i})$.

Thus we obtain that the read counts K_{ij} have a Poisson-Gamma mixture of different Poisson variables. As shown above we can use the Negative Binomial with parameters $(m = u_i s_j)$ and ϕ_i as a satisfactory model of the variability.

The counts for the gene/taxa i and sample j in condition c having a Negative Binomial distribution with $m_c = u_{ic} s_j$ and ϕ_{ic} so that the variance is written

$$u_{ic} s_j + \phi_{ic} s_j^2 u_{ic}^2.$$

We can estimate the parameters u_{ic} and ϕ_{ic} from the data.

Applications

Mixtures occur naturally because of heterogeneous data, an experiment may be run by different labs, use differing technologies.

There are often differing binding propensities in different parts of the genome, PCR biases can occur when different operators use different protocols.

The most common problems involve different distributions because both the means *and* the variances are different. This requires variance stabilization to do statistical testing.

Mixture models can often lead us to be able to use data transformations are actually used in what is often known as a *generalized logarithmic* transformation applied in microarray variance stabilizing transformations and RNA-seq normalization and which also proves useful in the normalization of next generation reads in microbial ecology and Chip-SEQ analysis .

Mixtures are everywhere (There is mathematical reason) ....

It is actually not a mystery why mixtures are so ubiquitous, there is even a theorem that says that if the order in which the observations are made doesn't matter, then they are from a mixture....

de Finetti's theorem

If we don't know much about the noise but we know that the order in which we collect the data doesn't matter, that is called exchangeability:

$$D(X_1, X_2, X_3, \dots, X_n) = D(X_{\pi(1)}, X_{\pi(2)}, X_{\pi(3)}, \dots, X_{\pi(n)})$$

For pi any permutation of size n.

If the random variables $(X_1, X_2, X_3, \dots, X_n)$ are independent then

$$P(X_1 = x_1, X_2 = x_2, X_3 = x_3, \dots, X_n = x_n) = \prod_{i=1}^{n} P(X_1 = x_1)P(X_2 = x_2)P(X_3 = x_3)\dots P(X_n = x$$

commutativity of multiplication gives us exchangeability.

The converse is not true, but conditional independence through mixing.

Binary Data Example:

$$p(1, 1, 0, 0, 0, 1, 0, 0) = p(1, 0, 1, 0, 0, 0, 0, 1)$$

Exchangeable but not independent: Polya's urn and the restaurant.

Stefan Lauritzen Lecture

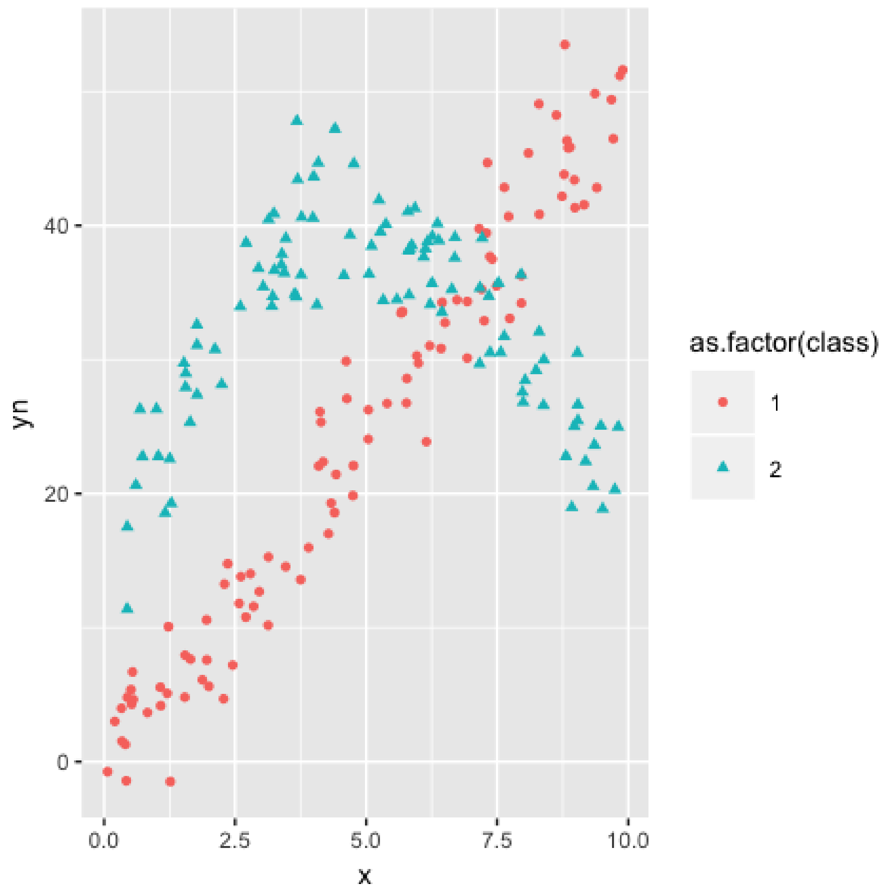
Mixture Modeling Examples for Regressions

The ‘flexmix’ package allows to cluster and fit regressions to the data at the same time. The standard M-step `FLXMRglm()` of FlexMix is an interface to R’s generalized linear modelling facilities - `glm()` function.

```r
require(flexmix)
data(NPreg)
```

Scatterplot

```
ggplot(NPreg,aes(x,yn)) +geom_point(aes(colour = as.factor(class),shape=as.factor(class)))
```



Two latent classes

As a simple example we use artificial data with two latent classes of size 100 each:

$$\begin{aligned} Class\ 1: &\quad y = 5x + \epsilon \\ Class\ 2: &\quad y = 15 + 10x - x^2 + \epsilon \end{aligned}$$

with $\epsilon \sim N(0, 9)$ and prior class probabilities $\pi_1 = \pi_2 = 0.5$.

We can fit this model in R using the commands

```
library("flexmix")
data("NPreg")
m1 = flexmix(yn ~ x + I(x^2), data = NPreg, k = 2)
m1
```

```
##
## Call:
## flexmix(formula = yn ~ x + I(x^2), data = NPreg,
##     k = 2)
##
## Cluster sizes:
##   1   2
## 100 100
##
## convergence after 16 iterations
```

and get a first look at the estimated parameters of mixture component~1 by

```
parameters(m1, component = 1)
```

```
##                    Comp.1
## coef.(Intercept) -0.2095
## coef.x            4.8177
## coef.I(x^2)       0.0362
## sigma             3.4766
```

and

```
parameters(m1, component = 2)
```

```
##                    Comp.2
## coef.(Intercept) 14.717
## coef.x            9.847
## coef.I(x^2)      -0.968
## sigma             3.480
```

for component 2. The parameter estimates of both components are close to the true values. A cross-tabulation of true classes and cluster memberships can be obtained by

```
table(NPreg$class, clusters(m1))
```

```
##
##      1  2
##   1 95  5
##   2  5 95
```

Output from flexmix

```r
summary(m1)
```

```
##
## Call:
## flexmix(formula = yn ~ x + I(x^2), data = NPreg,
##     k = 2)
##
##        prior size post>0 ratio
## Comp.1 0.494  100    145 0.690
## Comp.2 0.506  100    141 0.709
##
## 'log Lik.' -643 (df=9)
## AIC: 1303   BIC: 1333
```

Summary: One Dimensional Mixture Models

Finite Mixture Models

- Mixture of Normals with different means and variances.
- Decomposing the mixtures using the EM algorithm.
- Using a two component mixture model enables us to use a local fdr estimate for multiple hypotheses testing.

Empirical Distribution is a mixture model

- Small point mass on each observed value leads to bootstrap method.

Common Infinite Mixture Models

- Mixtures of Normals (often with a hierarchical model on the variances).
- Gamma-Poisson for read counts.
- Gamma-Exponential for PCR.
- Dirichlet-Multinomial (Birthday problem and the Bayesian setting).