Mixture distributions

I have thoughts about mixture distributions. Sometimes we get data that could be modelled simply with classical techniques if we make assumptions about its distribution which are mostly right. But the distribution is ‘contaminated’ with another distribution. Very small departures from your assumptions can cause major challenges which are not always visually obvious.

Consider this example, drawn from Rand Wilcox’s excllent *Modern Statistics for the Social and Bahvaioral Sciences*. A standard normal distribution has been contanimated with 10% of its data coming from a second distribution, also normal but with a much higher variance. Without very large samples and/or a particularly alert approach to the modelling, this sort of data can cause serious pitfalls for inference.

Exactly what the pitfalls are, I don’t have time to talk about right now, but they are serious and involve long tails and difficulties in estimating variance.

That example was drawn with this code, which also sets us up for application next in this post to a real life example that crossed my path a week ago:

```
library(tidyverse)
library(scales)
library(ggrepel)
library(glue)
library(ggtext)
library(rstan)

#---------Example adapted from Wilcox Modern Statistics for the Social
and Behavioral Sciences---------
# Wilcox's Figure 3.8

set.seed(123)
egd <- tibble(diet = rep(c("Diet", "No diet"), c(10000, 90000))) %>%
  mutate(weight_loss = ifelse(diet == "Diet",
                              rnorm(n(), 0, 10),
                              rnorm(n(), 0, 1)))

ggplot(egd, aes(x = weight_loss)) +
  geom_density(fill = "grey", alpha = 0.5, colour = "darkblue") +
  stat_function(fun = dnorm, colour = "darkgreen", n = 1000) +
  coord_cartesian(xlim = c(-3, 3)) +
  annotate("text", x = 1, y = 0.3,
           label = "sigma^2==1", parse = TRUE,
           colour = "darkgreen", hjust = 0) +
  annotate("text", x = -1.1, y = 0.17,
           label = glue("sigma^2=={round(var(egd$weight_loss), 1)}"),
parse = TRUE,
           colour = "darkblue", hjust = 0) +
  annotate("text", x = -1.1, y = 0.13,
           label ="Mixture of standard normal (90%) and\nnormal with
sd=10 (10%)" ,
           colour = "darkblue", hjust = 0, size = 3) +

  labs(caption = "Adapted from Figure 3.8 of Wilcox, Modern Statistics
for the Social and Behavioral Sciences",
       x = "Weight loss",
       title = "The perils of a mixed or 'contanimated' distribution",
       subtitle = "Comparison of a standard normal and a mixed normal
distribution") +
    theme(plot.subtitle = element_textbox())
```

Distribution of reporting delays for deaths in Florida

The real life example came from a Twitter thread by Jason Salemi on 28 August 2020, about the distribution of delays in reporting Covid-19 deaths in Florida. I can't find the actual Twitter thread now, but the discussion in passing mentioned the interesting fact that the mean delay was noticeably smaller than the median. This is unusual for data that has a strict lower bound (report isn't possible before the actual death) and no upper bound, because typically a long rightwards tail and a few outliers will drag the mean to noticeably higher value than the median.

The example has extra interest for me in thinking about the data generating process for this sort of reporting on deaths. Recently in Victoria it has become clear that there are (at least) two different ways Covid-19 deaths are being reported – via aged care, and everything else. The aged care deaths are characterised by appearing in the official statistics only at some delay (up to a month or more), and in big clumps. This looks like the sort of process that needs to be modelled by a mixture distribution.

Dr Salemi kindly forwarded me the actual Florida data, which looks like this:

The mean delay is 23.25 days and the median is 31. I've converted one freqency of minus one in the original data into zero to make my life simpler. That chart was produced with this R code:

```
#--------Florida delay in death reporting data-----------
# From Jason Salemi 28 August 2020 via Twitter

orig_d <- tibble(freq= c(11,18,2,2,1,5,2,3,0,1,2,1,3,
1,0,3,1,1,0,2,0,0,1,
                    3,0,0,1,0,0,1,2,2,2,7,8,4,7,6,6,3,3,6,0,2,2,1,3,
                    0,2,3,0,0,1,0,0,1)) %>%
  mutate(delay = 1:n() - 1) %>%
  mutate(cumulative_freq = cumsum(freq),
         quantile = cumulative_freq / sum(freq))

orig_x <- with(orig_d, rep(delay, freq))
mean(orig_x)
median(orig_x)

ggplot(orig_d, aes(x = as.ordered(delay), y = freq)) +
  geom_vline(xintercept = mean(orig_x), colour = "blue", size = 2) +
  geom_vline(xintercept = median(orig_x), colour = "red", size = 2) +
  geom_col(width = 1, colour = "grey50", alpha = 0.9) +
  geom_text(aes(label = ifelse(freq > 0, freq, ""), y = freq + 1),
size = 3, colour = "steelblue") +
  scale_x_discrete(breaks = 0:5 * 10) +
  labs(x = "Time elapsed from death occuring to appearing in confirmed
statistics",
       y = "Frequency",
       title = "Reporting lags in Covid-19 deaths in Florida",
       caption = "Data from Jason L. Salemi",
       subtitle = "An interesting example of a mixed distribution;
with two modes, and  mean < median.") +
  theme(plot.subtitle = element_textbox())
```

It seemed to me from this visualisation that this was likely to be from a mixture of two distributions, but I was still surprised that the median and mean worked out this way. This prompted me to consider "is it easy to construct a mixture distribution, given just the overall median and mean?"

Brute force – fitting a mixture of two Poisson distributions based on just mean and median

It wasn't hard to do this if I was prepared to assume both distributions contributing to the mixture are Poisson distributions. This means I have just three parameters to estimate – the lambda (mean and variance) of each Poisson distribution, and the proportion of the data that comes from the first of the two distributions.

If you have the mean of the first distribution and the proportion of data from that distribution, you can calculate the mean of the second that is needed to provide the given total mean. Because:

So:

So really we only have two degrees of freedom to worry about. By brute force we can make a grid of possible values of and , calculate the as above in a way that the overall mean is 23.25, and then use the probability mass functions of Poisson distributions to calculate the median of the resulting mixture directly. Then we just need to see which combinations of the parameters give the correct median (31.0).

It turns out there are not one but many mixtures of two Poisson distributions that will give you this combination of mean and median. Here is a summary of which combinations of parameters can give you this:

… and here are four examples of the actual fitted distributions. These all have the same mean and median, but differ in other characteristics

To do all this, I created two functions in R:

- `dpois2()` calculates the density of a mixture of two Poisson distributions
- `median_pois2()` returns the median of a mixture of two Poisson distributions

And then just calculated the median for all combinations of a grid of possible parameters as described above. Here's the code that does that and produces those two charts:

```
#' Density of a mixture of two Poisson distributions
#'
#' @param x vector of non-negative integers
#' @param lambda1 mean of first Poisson distribution
#' @param lambda2 mean of second Poisson distribution
#' @param p1 proportion of values that come from the first
distribution
dpois2 <- function(x, lambda1, lambda2, p1){
  d <- dpois(x, lambda1) * p1 +
    dpois(x, lambda2) * (1 - p1)
  return(d)
}



#' Approximate the median of a mixture of two Poisson distributions
#'
#' @param lambda1 mean of first Poisson distribution
#' @param lambda2 mean of second Poisson distribution
#' @param p1 proportion of values that come from the first
distribution
#' @param maxx highest value data to take into account
median_pois2 <- function(lambda1, lambda2, p1, maxx = 1000){
  df <- data.frame(x = 0:maxx)
  df$dens <- dpois2(df$x, lambda1, lambda2, p1)
  df$F <- cumsum(df$dens)
  which_row <- which(abs(df$F - 0.5) == min(abs(df$F - 0.5)))[1]
  med <- df[which_row, ]$x - 1
  return(med)
}

trial_grid <- expand.grid(p1 = 0:200 / 200,
                            lambda1 = 1:30 / 6) %>%
  as_tibble() %>%
  mutate(lambda2 = (23.25 - p1 * lambda1) / (1 - p1)) %>%
  mutate(med = Vectorize(median_pois2)(lambda1, lambda2, p1))

# Visualisation of trade-off between different parameters
set.seed(42)
trial_grid %>%
  filter(round(med, 2) == 31.0) %>%
  mutate(mu2_lab = ifelse(runif(n()) > 0.9, round(lambda2, 1), ""))
%>%
  ggplot(aes(x = p1, y = lambda1)) +
  geom_smooth(se = FALSE, colour = "lightblue") +
  geom_point(size = 2, colour = "grey") +
  geom_text_repel(aes(label = mu2_lab), colour = "steelblue") +
  labs(x = "Proportion of variables from first Poisson distribution",
       y = "Mean of first Poisson distribution",
       title = "Different mixtures to give you the same mean and
median",
       subtitle = "Parameters of a mixture of two Poisson
distributions that have a mean of 23.25 and median of 31.") +
  annotate("text", x = 0.378, y = 2, label = "Labels show mean of
second Poisson\ndistribution (selected points only).",
           hjust = 0, colour = "steelblue")


# Four example distributions:
top_params <- trial_grid %>%
  arrange(abs(med - 31), runif(n())) %>%
  slice(1:4)

dens_results <- list()
x <- 0:70

# Get the actual densities:
for(i in 1:nrow(top_params)){
  d <- top_params[i, ]
  dens_results[[i]] <- with(d,
                            data.frame(x = x,
                              dens = dpois2(x, lambda1, lambda2, p1),
                              lambda1 = lambda1,
                              lambda2 = lambda2,
                              p1 = p1,
                              parameter_set = i))
}

# Not sure why I have to define my own labeller function, but I can't
see
# otherwise how I pass multi_line through to label_parsed in
facet_wrap() !
lp <- function(labels, multi_line = FALSE){
  label_parsed(labels, multi_line = multi_line)
}

# Draw the chart for the four example distributions:
bind_rows(dens_results) %>%
  as_tibble() %>%
  mutate(par_lab1 = glue("lambda[1]=={round(lambda1, 1)}"),
         par_lab2 = glue("lambda[2]=={round(lambda2, 2)}"),
         par_lab3 = glue("pi[1]=={p1}")) %>%
  ggplot(aes(x = x, y = dens)) +
  facet_wrap(~par_lab1 + par_lab2 + par_lab3, labeller = lp) +
  geom_vline(xintercept = 31, colour = "red", size = 2) +
  geom_vline(xintercept = 23.5, colour = "blue", size = 2) +
  geom_col() +
  theme(panel.spacing = unit(1.5, "lines"),
        plot.title = element_textbox()) +
  xlim(0, 60) +
  labs(x = "x",
       y = "Density",
       title = "Different mixtures to give you the same mean and
median",
       subtitle = "Parameters of a mixture of two Poisson
distributions that (taken together) have a mean of 23.25 and median of
31.")

#---------------check I'm not going mad and that I do get the right
results!-------------------
n <- 1e5
xsim <- c(rpois(n * top_params[1, ]$p1, top_params[1, ]$lambda1),
          rpois(n * (1 - top_params[1, ]$p1), top_params[1,
]$lambda2))

c(mean(xsim), median(xsim))
```

Better approach – fitting mixtures in Stan using all the data

There are two serious shortfalls with the approach to modelling above:

1. Because I am using only the mean and the median summary statistics, I am discarding a lot of extra information in the original data
2. While Poisson distributions are easy to use because they are fully charaterised by a single parameter, there is no particular reason to think that they are a good model of the number of days death by which reporting is delayed.

Shortfall 1 suggests I need to find a way to estimate a good mixture of distributions that uses the full original dataset. Shortfall 2 suggests using a different distribution, but still one that works with integers; negative binomial is my usual go-to here.

Luckily, chapter 5 of the Stan Reference Manual is dedicated to Finite Mixtures. It has a straighforward and adpatable example of a mixture of K different normal distributions.

To address my challenges in easy steps, I started by fitting a mixture of two Poisson distributions to my delays data, using the full data not just the summary statistics. Here's the simple program in Stan that sets out this model

```
// Draws heavily on https://mc-stan.org/docs/2_24/stan-users-
guide/summing-out-the-responsibility-parameter.html

data {
  int<lower=0> N;
  int x[N];
  int<lower=1> K; // number of mixture components
}

parameters {
  positive_ordered[K] lambda; // arbitrarily ordered so lower value is
first, otherwise unidentified
  simplex[K] p;
}

model {
  vector[K] log_p = log(p); // cached log calculation
  lambda ~ gamma(1.5, 0.1); // prior for the lambdas

  for (n in 1:N){
    vector[K] lps = log_p;
    for (k in 1:K)
      lps[k] += poisson_lpmf(x[n] | lambda[k]);
    target += log_sum_exp(lps);
  }

}
```

Running this from R is a one-liner, and it quickly comes back with an estimate that the data could come from two Poisson distributions with means of 3.4 and 35.4, with 38% of the data coming from the first of these. This isn't *hugely* different from my cruder estimates based on just the mean and median, but noticeably the mean of the first distribution is now quite a bit higher. Results from Stan are just pasted into the code below as comments, for reference.

```
ms1 <- stan("0192-poisson.stan", data = list(x = orig_x,
                                             N = length(orig_x),
                                             K = 2))
ms1
# Inference for Stan model: 0192-poisson.
# 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
# lambda[1]    3.42    0.01 0.30    2.84    3.21    3.41    3.61
4.02  2849    1
# lambda[2]   35.41    0.01 0.68   34.09   34.94   35.41   35.88
36.71  4107    1
# p[1]         0.38    0.00 0.04    0.30    0.35    0.38    0.41
0.47  3532    1
# p[2]         0.62    0.00 0.04    0.53    0.59    0.62    0.65
0.70  3532    1
# lp__      -580.55    0.03 1.29 -584.02 -581.09 -580.22 -579.62
-579.11  2029    1
#
# Samples were drawn using NUTS(diag_e) at Sun Sep 06 11:36:48 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
#
```

What about addressing shortfall 2? It's straightforward to adapt my Stan program to work with a mixture of K negative binomial distributions. Note that there are two different popular ways to characterise a negative binomial distribution, I am choosing the method that specifies the mean and a dispersion parameter:

```
// Draws heavily on https://mc-stan.org/docs/2_24/stan-users-
guide/summing-out-the-responsibility-parameter.html
// Save as 0192-negbin.stan

data {
  int<lower=0> N;
  int x[N];
  int<lower=1> K; // number of mixture components
}

parameters {
  positive_ordered[K] mu; // arbitrarily ordered so lower value is
first, otherwise unidentified
  real<lower=0> phi[K];   // dispersion parameter for the negative
binomials
  simplex[K] p;
}

model {
  vector[K] log_p = log(p); // cached log calculation
  mu ~ gamma(1.5, 0.1); // prior for the means of the negative
binomial distribution
  phi ~ gamma(2, 1);     // prior for the phi (dispersion) of the neg
bin distribution
  for (n in 1:N){
    vector[K] lps = log_p;
    for (k in 1:K)
      lps[k] += neg_binomial_2_lpmf(x[n] | mu[k], phi[k]);
    target += log_sum_exp(lps);
  }
}
```

Fitting this gives me a mixture with 40% of the data from a negative binomial distribution with (mean, size) parameters of (4.8, 0.77); and 60% from a distribution with (mean, size) of (35.8, 13.7). Closing the circle, if I simulate data in R from that mixture, this is what it looks like:

It's noticeably not a perfect fit to the original Florida data, suggesting that even this model is perhaps under-fit and more complex things are going on. Also that more data (as always) would be helpful. But I think I'm onto a good approach. As an aside, I find it extraordinary how well Stan works in this situation.

Here's the final R code for controlling the Stan program with the negative binomial mixture, and simulating data from the result. Again, results from Stan are just pasted into the code below as comments, for reference.

```
ms2 <- stan("0192-negbin.stan", data = list(x = orig_x,
                                            N = length(orig_x),
                                            K = 2),
            cores = 4, chains = 4)
ms2
# Inference for Stan model: 0192-negbin.
# 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[1]     4.83    0.04 1.64    2.52    3.70    4.49    5.58    9.04
1423    1
# mu[2]    35.82    0.02 1.50   32.87   34.86   35.80   36.81   38.79
4234    1
# phi[1]    0.77    0.00 0.23    0.44    0.61    0.74    0.89    1.30
2190    1
# phi[2]   13.71    0.06 3.22    8.18   11.39   13.48   15.71   20.74
3032    1
# p[1]      0.40    0.00 0.05    0.31    0.37    0.40    0.44    0.51
2152    1
```

```
# p[2]       0.60    0.00 0.05    0.49    0.56    0.60    0.63    0.69
2152    1
# lp__    -526.41    0.04 1.64 -530.64 -527.20 -526.08 -525.21 -524.25
1524    1
#
# Samples were drawn using NUTS(diag_e) at Sun Sep 06 12:04:16 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).

#---------------simulated data from that fitted mixed negative
binomial distribution-------------------
n <- 1e5
xsim <- c(rnbinom(n * 0.4, mu = 4.83, size = 0.77),
          rnbinom(n * 0.6, mu = 35.82, size = 13.71)
)

# Plot of simulations
data.frame(x = xsim) %>%
  ggplot(aes(x = x)) +
  geom_vline(xintercept = mean(xsim), colour = "blue", size = 2) +
  geom_vline(xintercept = median(xsim), colour = "red", size = 2) +
  geom_histogram(binwidth = 1, aes(y = ..density..), colour =
"grey50", alpha = 0.9) +
  scale_y_continuous(label = comma) +
  coord_cartesian(xlim = range(orig_d$delay)) +
  theme(plot.subtitle = element_textbox_simple()) +
  labs(y = "Relative frequency",
       title = "Simulations from fitted mixture of two negative
binomial distributions",
       subtitle = "Fit is ok and median > mean, but not by as much as
in original.")
```