openEBGM: An R Implementation of the Gamma-Poisson Shrinker Data Mining Model

by Travis Canida and John Ihrie

Abstract We introduce the R package **openEBGM**, an implementation of the Gamma-Poisson Shrinker (GPS) model for identifying unexpected counts in large contingency tables using an empirical Bayes approach. The Empirical Bayes Geometric Mean (EBGM) and quantile scores are obtained from the GPS model estimates. **openEBGM** provides for the evaluation of counts using a number of different methods, including the model-based disproportionality scores, the relative reporting ratio (RR), and the proportional reporting ratio (PRR). Data squashing for computational efficiency and stratification for confounding variable adjustment are included. Application to adverse event detection is discussed.

Introduction

Contingency tables are a common way to summarize events that depend on categorical factors. DuMouchel (1999) describes an application of contingency tables to adverse event reporting databases. The rows represent products, such as drugs or food, and the columns represent adverse events. Each cell in the table represents the number of reports that mention both that product and that event. Overreported product-event pairs might be of interest.

One naïve approach for analyzing counts in this table is to calculate the observed *relative reporting ratio* (*RR*)—which DuMouchel (1999) calls *relative report rate*—for each cell. While *RR* is easy to compute, it has the drawback of being highly variable, and thus unreliable, for small counts (DuMouchel, 1999; Madigan et al., 2011). To combat the high variability of *RR*, DuMouchel (1999) created an *empirical Bayes* (EB) data mining model for finding "interestingly large" counts. The model-based EB scores are measures of disproportionality, similar to *RR*. However, the model uses Bayesian shrinkage to correct for the high variability in *RR* associated with small counts.

After the EB model was introduced, Evans et al. (2001) created another disproportionality approach called the *proportional reporting ratio* (*PRR*). The *PRR* compares "the proportion of all reactions to a drug which are for a particular medical condition of interest...to the same proportion for all drugs in the database" (Evans et al., 2001). Like *RR*, *PRR* is easy to calculate, but has the same drawback of high variability (Madigan et al., 2011). Almenoff et al. (2006) found that *PRR* finds more false positives than DuMouchel's model; however, it also finds more true positives. Another difference between DuMouchel's model and the *PRR* metric is that while DuMouchel's model considers the event of interest when calculating the expected count for the pair, *PRR* does not (Duggirala et al., 2015).

The **openEBGM** (Ihrie and Canida, 2017) package implements the model described in DuMouchel (1999) and the computational efficiency improvement techniques described in DuMouchel and Pregibon (2001). Our goal was to create a general-purpose, flexible, efficient implementation of DuMouchel's model, but we also include *PRR* in case users want to compare the results. Other disproportionality approaches exist (Madigan et al., 2011), but our goal for this article is not to provide an exhaustive list or to compare DuMouchel's model to other methods. Instead, we focus on our implementation of DuMouchel's model and provide some comparisons with other R packages. Users can refer to **openEBGM**'s vignettes to follow future developments and modifications to the package.

While **openEBGM** was developed mainly for adverse event detection, we structured the code to be general enough for any similar application. DuMouchel describes multiple applications of his EB model, which can find statistical associations but not necessarily causal relationships (DuMouchel, 1999; DuMouchel and Pregibon, 2001). Thus, the model is used primarily for signal detection. Regardless of the application, subject-matter experts should investigate using other means to determine if any causal relationships might exist.

Attenuating the relative reporting ratio

The relative reporting ratio compares a table cell's actual count, N, to its expected count, E, under the assumption of independence between rows and columns: $RR = {}^{N}/_{E}$. Thus, $RR = 1$ if the actual count is equal to the expected count. When $RR > 1$, more events are observed than expected. Therefore, large RR scores may indicate interesting row-column pairs. This approach to analyzing contingency

table counts works well for large cell counts, but small cell counts result in unstable RR values. Since the expected counts can be close to zero, RR can be very large (DuMouchel, 1999) for small actual counts which could have easily occurred simply by chance. The EB approach shrinks large RRs with small Ns to a value much closer to 1. The shrinkage is less for larger counts. Shrinkage produces more reliable results than the simple RR score.

Model description

The EB model uses a Poisson(μ_{ij}) data distribution (i.e. likelihood) for contingency table cell counts in row i and column j, where $i = 1, \ldots, I$ and $j = 1, \ldots, J$. We are interested in the ratio $\lambda_{ij} = {}^{\mu_{ij}}/_{E_{ij}}$. The prior distribution on λ (Equation 1) is a mixture of two gamma distributions, resulting in gamma-mixture posterior distributions (Equation 2). Thus, the model is sometimes referred to as the *Gamma-Poisson Shrinker* (GPS) model.

The prior is a single distribution that models all cell counts; however, each cell has a separate posterior distribution determined both by that cell's actual and expected counts and by the distribution of actual and expected counts in the entire table. The λ_{ij}s are assumed to come from the prior distribution with hyperparameter $\theta_{prior} = (\alpha_1, \beta_1, \alpha_2, \beta_2, P)$, where P is the mixture fraction. The posterior distributions have parameters $\theta_{post,ij} = \left(\alpha_1 + n_{ij},\ \beta_1 + E_{ij},\ \alpha_2 + n_{ij},\ \beta_2 + E_{ij},\ Q_{n,ij}\right)$, where $Q_{n,ij}$ are the mixture fractions. The posterior distributions are, in a sense, Bayesian representations of the relative reporting ratios (note the similarity in the equations $RR_{ij} = {}^{N_{ij}}/_{E_{ij}}$ and $\lambda_{ij} = {}^{\mu_{ij}}/_{E_{ij}}$). DuMouchel (1999, Eqs. 4, 7) summarized the model with row and column subscripts suppressed:

$$prior:\ \pi(\lambda;\ \alpha_1, \beta_1, \alpha_2, \beta_2, P) = Pg(\lambda;\ \alpha_1, \beta_1) + (1 - P)g(\lambda;\ \alpha_2, \beta_2) \tag{1}$$

$$posterior:\ \lambda | N = n \sim \pi(\lambda;\ \alpha_1 + n,\ \beta_1 + E,\ \alpha_2 + n,\ \beta_2 + E,\ Q_n) \tag{2}$$

where $g(\cdot) \sim \Gamma(\alpha, \beta)$ is the gamma distribution with shape and rate parameters α and β, respectively.

Model-based scores

The EB scores are based on the posterior distributions and used in lieu of RR. The *Empirical Bayes Geometric Mean* ($EBGM$) score is the geometric mean of a posterior distribution. The 5^{th} and 95^{th} percentiles of a posterior distribution create a two-sided 90% credibility interval for the EB disproportionality score. Alternatively, since we are primarily interested in the lower bound, we could create a one-sided 95% credibility interval (Szarfman et al., 2002). Bayesian shrinkage causes the EB scores to be smaller than RR scores, but the shrinkage amount decreases as N increases (Figure 1).

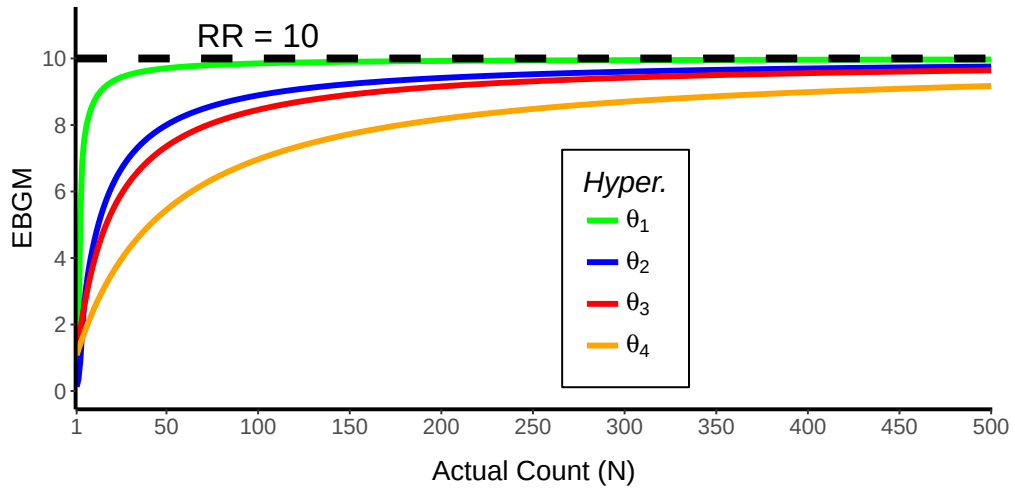

Figure 1: Asymptotic behavior of $EBGM$ for various hyperparameter values. $\theta_1 = (.01, .1, 5, 20, .5)$; $\theta_2 = (.1, 1.2, .4, 8, .05)$; $\theta_3 = (2, 2, 5, 5, .8)$; $\theta_4 = (5, 5, 5, 5, .5)$

Example application

The U.S. Food and Drug Administration (FDA) monitors regulated products (such as drugs, vaccines, food and cosmetics) for adverse events. For food, FDA's Center for Food Safety and Applied Nutrition (CFSAN) mines data from the CFSAN Adverse Events Reporting System (CAERS) (U.S. Food and Drug Administration, 2017). Adverse events are tabulated in contingency tables (separate tables for separate product types). The FDA previously used the GPS model to analyze these tables (Szarfman et al., 2002). Later, we show an example of GPS using the CAERS database.

Existing open-source implementations

Existing open-source implementations of the GPS model include the R packages **PhViD** (Ahmed and Poncet, 2016) and **mederrRank** (Venturini and Myers, 2015). Each of the existing implementations has its own feature set and drawbacks.

The **PhViD** package does not offer data squashing, stratification, or a means of counting events from raw data. Nor does it account for unique event identifiers to eliminate double counting of reports when calculating expected counts. **PhViD** does, however, offer features (Ahmed et al., 2009) not found in **openEBGM**, such as false discovery rate decision rules. **PhViD**'s approach to implementation of the GPS model seems focused on adding multiple comparison techniques. We used the **PhViD** package as a starting point to write our own code. However, we focused on creating a tool that simply implements the GPS model in a flexible and efficient manner. See Appendix 2.10 for a timing comparison between **openEBGM** and **PhViD**.

The **mederrRank** package adapts the GPS model to a specific medication error application. Although **mederrRank** does not appear to offer data squashing or a means of counting events from raw data, it does allow for stratification by hospital (Venturini et al., 2017). Hyperparameters are estimated using an expectation-maximization (EM) algorithm, whereas **openEBGM** uses a gradient-based approach. **mederrRank** seems focused on comparing the GPS model to the Bayesian hierarchical model suggested by Venturini et al. (2017).

Main functions

Table 1 shows a listing of **openEBGM**'s main functions:

Function	Description
`processRaw()`	Counts unique reports; calculates expected counts, RR, and PRR.
`squashData()`	Squashes points (N, E) to reduce computational burden for hyperparameter estimation.
`autoHyper()`	Calculates hyperparameter estimates.
`ebgm()`	Calculates $EBGM$ scores.
`quantBisect()`	Calculates quantile scores.
`ebScores()`	Creates an object with $EBGM$ and quantile scores.

Table 1: The main functions in **openEBGM**.

Hardware and software specifications

We ran the code presented in this article on a 64-bit Windows 7 machine with 128 GB of DDR3 RAM clocked at 1866 MHz and two 6-core Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60 GHz processors. We used R v3.4.1, **openEBGM** v0.3.0, and **PhViD** v1.0.8.

Data preparation

Use of the **openEBGM** package requires that the data be formatted in a tidy way (Wickham, 2014); the data must adhere to the standards of one column per variable and one row per observation. The columns may be of class `"factor"`, `"character"`, `"integer"` or `"numeric"`. Although zero counts can

exist in the contingency table, missing values are not allowed in the unprocessed data and must be replaced with appropriate values or removed before using **openEBGM**'s functions.

Column names

The input data frame must contain some specific columns. In particular, these are: 'var1', 'var2' and 'id'. 'var1' and 'var2' are simply the row and column variables of the contingency table, respectively. The identifier ('id') column allows **openEBGM** to properly handle marginal totals and remove the effect of double counting. For example, many CAERS reports contain multiple products and events. However, if an application does use a unique cell for each event of interest, the user can then create a column of unique sequential identifiers with `df$id <-1:nrow(df)`, where `df` is the input data frame.

In addition to the columns mentioned above, a column which stratifies the data (by, for example, gender, age, race, etc.) may be of interest as it can help reduce the effects of confounding variables (DuMouchel, 1999). If stratification is used, any column whose name contains the case-sensitive substring 'strat' will be treated as a stratification variable. If a continuous variable (e.g. age) is used for stratification, it should be appropriately categorized so that it is no longer continuous. Additional columns besides those mentioned above are allowed, but ignored by **openEBGM**.

CAERS data example

One example use case of the **openEBGM** package is adverse event signal detection. This application is utilized by FDA's CFSAN with data from CAERS, which collects adverse event reports on consumer products such as foods and dietary supplements and is freely available and updated quarterly (U.S. Food and Drug Administration, 2017). An example of data preparation with the CAERS dataset is shown below. We start by downloading the data from FDA's website and selecting dietary supplement (*Industry Code 54*) reports before the year 2017.

```
> Sys.setlocale(locale = "C")  #locale can affect sorting order, etc.
> site   <- "https://www.fda.gov/downloads/Food/ComplianceEnforcement/UCM494018.csv"
> dat    <- read.csv(site, stringsAsFactors = FALSE, strip.white = TRUE)
> dat$yr <- dat$RA_CAERS.Created.Date
> dat$yr <- substr(dat$yr, start = nchar(dat$yr) - 3, stop = nchar(dat$yr))
> dat$yr <- as.integer(dat$yr)
> dat    <- dat[dat$PRI_FDA.Industry.Code == 54 & dat$yr < 2017, ]
```

Next we rename the columns:

```
> dat$var1 <- dat$PRI_Reported.Brand.Product.Name
> dat$var2 <- dat$SYM_One.Row.Coded.Symptoms
> dat$id   <- dat$RA_Report..
> dat$strat_gen <- dat$CI_Gender
> vars <- c("id", "var1", "var2", "strat_gen")
> dat  <- dat[, vars]
```

Gender needs to be recategorized:

```
> dat$strat_gen <- ifelse(dat$strat_gen %in% c("Female", "Male"),
+                         dat$strat_gen, "unknown")
```

We can remove rows with non-ASCII characters in case they cause problems in certain locales:

```
> dat2 <- dat[!dat$var1 %in% tools::showNonASCII(dat$var1), ]
```

Each row has one product, but multiple adverse events:

```
> head(dat2, 3)
```

```
     id                    var1                                        var2 strat_gen
7 65353     HERBALIFE RELAX NOW PARANOIA, PHYSICAL EXAMINATION, DELUSION    Female
8 65353 HERBALIFE TOTAL CONTROL PARANOIA, PHYSICAL EXAMINATION, DELUSION    Female
9 65354                 YOHIMBE                 BLOOD PRESSURE INCREASED      Male
```

We can use the **tidyr** (Wickham, 2017) package to reshape the data:

```
> dat_tidy <- tidyr::separate_rows(dat2, var2, sep = ", ")
> dat_tidy <- dat_tidy[, vars]
```

```
> head(dat_tidy, 3)

     id                var1                 var2 strat_gen
1 65353 HERBALIFE RELAX NOW             PARANOIA    Female
2 65353 HERBALIFE RELAX NOW PHYSICAL EXAMINATION    Female
3 65353 HERBALIFE RELAX NOW             DELUSION    Female
```

Data cleaning

As a small digression, it is worth noting that the quality of the data can greatly impact the results of the **openEBGM** package. For instance, **openEBGM** considers the drug-symptom pairs 'drug1-symp1', 'Drug1-symp1', and 'DRUG1-symp1' to be distinct *var1-var2* pairs. Even minor differences (e.g. spelling, capitalization, spacing, etc.) can influence the results. Thus, we recommend cleaning data before using **openEBGM** to help improve hyperparameter estimates and signal detection in general.

While the issue above could be fixed simply by using the R functions `tolower()` or `toupper()`, other issues such as punctuation, spacing, string permutations in the 'var1' or 'var2' variables, or some combination of these must first be vetted and repaired by the user.

Counts and simple disproportionality measures

openEBGM contains functions which take the prepared data and output *var1-var2* counts, as well as some simple disproportionality measures for these *var1-var2* pairs. As mentioned in other sections, there are a number of disproportionality measures of interest that this package concerns, including the EB scores, *RR* as well as the *PRR*. The `processRaw()` function in the **openEBGM** package takes the prepared data and outputs the *var1-var2* pair counts (N), the expected number of counts for the *var1-var2* pair (E), as well as the *RR* and *PRR* for the *var1-var2* pair.

Data processing function

The function `processRaw()` takes the prepared data and returns a data frame with one row for each *var1-var2* pair. Each row contains the simple disproportionality measures (*RR*, *PRR*) as well as the counts (N, E) for that pair. In the case that the calculation for *PRR* involves division by zero, a default value of 'Inf' is returned.

The user may decide if stratification should be used to calculate E and whether zero counts (i.e. a given *var1-var2* pair is never observed in the data) should be included. Stratification affects *RR*, but not *PRR*. For the purpose of reducing computational burden, zero counts should not typically be included for hyperparameter estimation; however, they may help when convergence issues are encountered. If included, the points should be squashed (discussed in later sections) in order to reduce the computation time. These zero counts should not typically be used for EB scores since they are meaningless for studying larger-than-expected counts (which is usually the goal).

Data processing example

Here, the **tidyr** package is only needed for the pipe (`%>%`) operator.

```
> library("tidyr")
> library("openEBGM")
> data("caers")  #small subset of publicly available CAERS data
> head(caers, 3)

      id     var1                    var2 strat1
1 147289 PREVAGEN          BRAIN NEOPLASM Female
2 147289 PREVAGEN CEREBROVASCULAR ACCIDENT Female
3 147289 PREVAGEN          RENAL DISORDER Female
```

First, we process the data without using stratification or zeroes:

```
> processRaw(caers) %>% head(3)

                    var1                  var2 N            E      RR      PRR
1         1-PHENYLALANINE  HEART RATE INCREASED 1 0.0360548272   27.74    27.96
2 11 UNSPECIFIED VITAMINS               ASTHMA 1 0.0038736591  258.15   279.58
3 11 UNSPECIFIED VITAMINS CARDIAC FUNCTION TEST 1 0.0002979738 3356.00      Inf
```

Now, using stratification:

```
> processRaw(caers, stratify = TRUE) %>% head(3)

stratification variables used: strat1
there were 3 strata

                     var1                   var2 N            E      RR      PRR
1         1-PHENYLALANINE  HEART RATE INCREASED 1 0.0287735849   34.75    27.96
2 11 UNSPECIFIED VITAMINS               ASTHMA 1 0.0047169811  212.00   279.58
3 11 UNSPECIFIED VITAMINS CARDIAC FUNCTION TEST 1 0.0004716981 2120.00      Inf
```

Finally, we use stratification and a cap on the number of strata (useful for preventing excessive data slimming, especially with uncategorized continuous variables):

```
> processRaw(caers, stratify = TRUE, max_cats = 2) %>% head(3)

stratification variables used: strat1
Error in .checkStrata_processRaw(data, max_cats) :
  at least one stratification variable contains more than 2 categories --
  did you remember to categorize stratification variables?
  if you really need more categories, increase 'max_cats'
```

Hyperparameter estimation

The marginal distribution of each N_{ij} is a negative binomial mixture (DuMouchel, 1999, p. 180, Eq. 5) and is a function of the hyperparameter (θ_{prior}), the actual observed counts (n_{ij}), and the expected counts (E_{ij}). The prior distribution's maximum likelihood hyperparameter estimate, $\hat{\theta}_{prior}$, is obtained by minimizing the negative log-likelihood from these marginal distributions. Global optimization is needed to estimate θ_{prior}. **openEBGM**'s approach to global optimization is to simply use local optimization with multiple starting points. The user can manually optimize the likelihood function using another approach if desired. **openEBGM**'s optimization functions are wrappers for functions from the **stats** package. (**Note:** Results might vary slightly by operating system and version of R.) Users are encouraged to explore many optimization approaches because the accuracy of a global optimization result is difficult to verify, convergence is not guaranteed, and some approaches may outperform others.

Data squashing

DuMouchel and Pregibon (2001) used a method of reducing computational burden they called *data squashing*, which reduces the number of data points (n_{ij}, E_{ij}) used for hyperparameter estimation. A very large table with I rows and J columns can require immense computational resources. Data squashing reduces this set of points to a much smaller set of K points (n_k, E_k, W_k), where $k = 1, \ldots, K < I \times J$ and W_k is the weight of the k^{th} squashed point.

For a given n, `squashData()` bins points with similar Es and uses the average E within each bin as the expected count for that squashed point. The new points are weighted by bin size. For example, the points $(1, 1.1)$ and $(1, 1.3)$ could be squashed to $(1, 1.2, 2)$. To minimize information loss, we recommend only squashing points in close proximity. By default, `squashData()` does not squash the points with the highest Es for a given n since those points tend to have more variability (at least for small n; see Figure 2). The user can call `squashData()` repeatedly on different values of n (e.g. $n = 1$, then $n = 2$). We recommend squashing the data to less than 20,000 points.

Likelihood functions

The likelihood function for θ_{prior} (DuMouchel, 1999, p. 181, Eq. 12) must be adjusted (DuMouchel and Pregibon, 2001) when using data squashing or removing small counts (often just zeroes) from the estimation procedure, so **openEBGM** offers 4 likelihood functions: `negLL()`, `negLLsquash()`, `negLLzero()`, and `negLLzeroSquash()`. Since the GPS model was developed to study large datasets, `negLLsquash()` will usually be used since it allows for both data squashing and the removal of smaller counts. The user will not call the likelihood functions directly if using the wrapper functions described in the next section.

Optimization wrapper functions

`exploreHypers()` requires the user to choose one or more starting points (i.e. guesses) for θ_{prior}. For each starting point, the corresponding estimate, $\hat{\theta}_{prior}$, is returned if the algorithm converges. Examining estimates from multiple starting points allows the user to study the consistency of the results and reduces the chances of false convergence or getting trapped in a local minimum. Hyperparameter estimates are calculated using an implementation of one of three Newton-like or quasi-Newton methods from the **stats** package: `nlminb()`, `nlm()`, or `optim()` (using `method = "BFGS"` for `optim()`). The `N_star` argument defines the smallest actual count (usually 1) used for hyperparameter estimation (DuMouchel and Pregibon, 2001). Setting the `std_errors` argument to `TRUE` calculates estimated standard errors using the observed Fisher information as discussed in DuMouchel (1999, p. 183).

`autoHyper()` uses a semi-automated approach that returns a list including a final $\hat{\theta}_{prior}$ after running some verification checks on the estimates returned by `exploreHypers()`. From the solutions that converge inside the parameter space, `autoHyper()` chooses the $\hat{\theta}_{prior}$ with the smallest negative log-likelihood. By default, at least one other convergent solution must be similar to the chosen $\hat{\theta}_{prior}$ (i.e. within a specified tolerance defined by the `tol` argument). Each of the three methods available in `exploreHypers()` are attempted in sequence until these conditions are satisfied. If all methods are exhausted without consistent convergence, `autoHyper()` returns an error message. Setting the `conf_ints` argument to `TRUE` returns standard errors and asymptotic normal confidence intervals. `exploreHypers()` is called internally, so `autoHyper()` may be used without first calling `exploreHypers()`.

Hyperparameter estimation example

We start by counting item pairs and squashing the counts twice:

```
> proc <- processRaw(caers)
```

Figure 2 illustrates why squashing the largest Es for each N could result in a large loss of information.

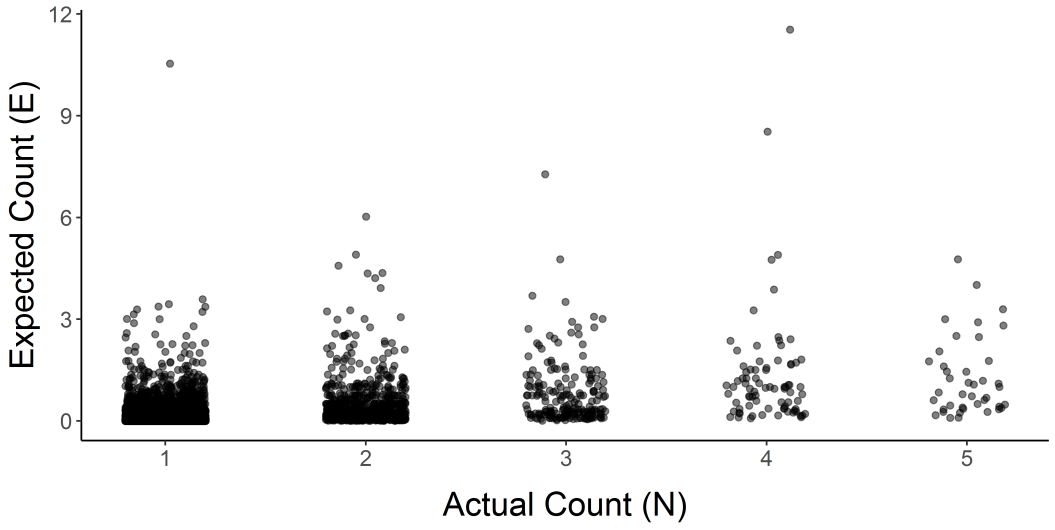

Figure 2: Larger expected counts are generally more spread out; by default, they are not squashed (to prevent information loss).

```
> squashed <- squashData(proc)
> squashed <- squashData(squashed, count = 2, bin_size = 10)
> head(squashed, 3); tail(squashed, 2)

  N            E weight
1 1 0.0002979738     50
2 1 0.0002979738     50
3 1 0.0002979738     50
     N        E weight
946 53 14.30095      1
947 54 16.05721      1
```

We can optimize the likelihood function directly:

```
> theta_init1 <- c(alpha1 = 0.2, beta1 = 0.1, alpha2 = 2, beta2 = 4, p = 1/3)
> stats::nlminb(start = theta_init1, objective = negLLsquash,
+               ni = squashed$N, ei = squashed$E, wi = squashed$weight)$par

    alpha1      beta1     alpha2      beta2          p
3.25120698 0.39976727 2.02695588 1.90892701 0.06539504
```

Or we can use **openEBGM**'s wrapper functions:

```
> theta_init2 <- data.frame(
+   alpha1 = c(0.2, 0.1, 0.5),
+   beta1  = c(0.1, 0.1, 0.5),
+   alpha2 = c(2,   10,  5),
+   beta2  = c(4,   10,  5),
+   p      = c(1/3, 0.2, 0.5)
+ )

> exploreHypers(squashed, theta_init = theta_init2, std_errors = TRUE)

$estimates
  guess_num   a1_hat    b1_hat   a2_hat   b2_hat      p_hat ... minimum
1         1 3.251207 0.3997673 2.026956 1.908927 0.06539504 ... 4161.921
2         2 3.251187 0.3997670 2.026961 1.908933 0.06539553 ... 4161.921
3         3 3.251243 0.3997702 2.026965 1.908933 0.06539509 ... 4161.921

$std_errs
  guess_num    a1_se     b1_se     a2_se     b2_se       p_se
1         1 2.280345 0.1434897 0.4515784 0.4328318 0.03575796
2         2 2.280247 0.1434851 0.4515718 0.4328231 0.03575709
3         3 2.280786 0.1435108 0.4516005 0.4328767 0.03576430

There were 11 warnings (use warnings() to see them)

> (theta_hat <- autoHyper(squashed, theta_init = theta_init2, conf_ints = TRUE))

$method
[1] "nlminb"

$estimates
    alpha1      beta1     alpha2      beta2          P
3.25118662 0.39976698 2.02696130 1.90893277 0.06539553

$conf_int
       pt_est     SE   LL_95  UL_95
a1_hat 3.2512 2.2802 -1.2180 7.7204
b1_hat 0.3998 0.1435  0.1185 0.6810
a2_hat 2.0270 0.4516  1.1419 2.9120
b2_hat 1.9089 0.4328  1.0606 2.7573
p_hat  0.0654 0.0358 -0.0047 0.1355

$num_close
[1] 2

$theta_hats
  guess_num   a1_hat    b1_hat   a2_hat   b2_hat      p_hat ... minimum
1         1 3.251207 0.3997673 2.026956 1.908927 0.06539504 ... 4161.921
2         2 3.251187 0.3997670 2.026961 1.908933 0.06539553 ... 4161.921
3         3 3.251243 0.3997702 2.026965 1.908933 0.06539509 ... 4161.921

There were 11 warnings (use warnings() to see them)
```

Warnings are often produced. Global optimization results are not guaranteed even when no warnings are produced and convergence is reached. The starting points can have an impact on the results. DuMouchel (1999) and DuMouchel and Pregibon (2001) provide some recommendations for starting points. The user must decide if the results seem reasonable.

EB disproportionality scores

The empirical Bayes scores are obtained from the posterior distributions. Thus, the posterior functions calculate the EB scores. Casual users will rarely call these functions directly since they are called internally by the `ebScores()` function described in the next section. However, we still exported these functions for users that want added flexibility.

Posterior functions

`Qn()` calculates the mixture fractions for the posterior distributions using the hyperparameter estimates ($\hat{\theta}_{prior}$) and the counts (n_{ij}, E_{ij}). The values returned by `Qn()` correspond to "the posterior probability that λ came from the first component of the mixture, given $N = n$" (DuMouchel, 1999, p. 180). Recall that a posterior distribution exists for each cell in the table. Thus, `Qn()` returns a numeric vector with the same length as the number of (usually non-zero) *var1-var2* combinations. Use the counts returned by `processRaw()`—not the squashed dataset—as inputs for `Qn()`.

`ebgm()` finds the *EBGM* scores. These scores replace the *RR* scores and represent the geometric means of the posterior distributions. Scores much larger than 1 indicate *var1-var2* pairs that occur at a higher-than-expected rate. `quantBisect()` finds the quantile scores (i.e. credibility limits) using the bisection method and can calculate any percentile between 1 and 99. Low percentiles (e.g. 5^{th} or 10^{th}) can be used as conservative disproportionality scores.

EB scores example

Continuing with the previous example:

```
> theta_hats <- theta_hat$estimates
> qn <- Qn(theta_hats, N = proc$N, E = proc$E)
> proc$EBGM <- ebgm(theta_hats, N = proc$N, E = proc$E, qn = qn)
> proc$QUANT_05 <- quantBisect(5, theta_hat = theta_hats,
+                              N = proc$N, E = proc$E, qn = qn)
> proc$QUANT_95 <- quantBisect(95, theta_hat = theta_hats,
+                              N = proc$N, E = proc$E, qn = qn)
> head(proc, 3)
```

```
                    var1                  var2 ...      RR ... EBGM QUANT_05 QUANT_95
1          1-PHENYLAL...  HEART RATE INCREASED ...   27.74 ... 2.23     0.49    13.85
2 11 UNSPECIFIED VIT...                ASTHMA ...  258.15 ... 2.58     0.52    15.78
3 11 UNSPECIFIED VIT... CARDIAC FUNCTION TEST ... 3356.00 ... 2.63     0.52    16.02
```

Object-oriented features

In addition to the capabilities described above, **openEBGM** can create S3 objects of class `"openEBGM"` to aid in the calculation and inspection of disproportionality scores, as well as reduce the number of direct function calls needed. When using an `"openEBGM"` object, the generic functions `print()`, `summary()` and `plot()` dispatch to methods written specifically for objects of this class.

Object creation

The `ebScores()` function instantiates an object, which includes the EB scores (*EBGM* and chosen posterior quantile scores). After calculating the hyperparameters, `ebScores()` can instantiate an object by calling the posterior distribution functions (previous section), thus simplifying the analyst's work. Generic functions can then be used to quickly review the results. An example is provided below.

```
> proc2 <- processRaw(caers)
> ebScores(proc2, hyper_estimate = theta_hat, quantiles = 10)$data %>% head(3)
```

```
                    var1                  var2 N ... EBGM QUANT_10
1          1-PHENYLALANINE  HEART RATE INCREASED 1 ... 2.23     0.67
2 11 UNSPECIFIED VITAMINS                ASTHMA 1 ... 2.58     0.71
3 11 UNSPECIFIED VITAMINS CARDIAC FUNCTION TEST 1 ... 2.63     0.72
```

We can also calculate upper and lower credibility limits for the EB disproportionality score:

```
> obj <- ebScores(proc2, hyper_estimate = theta_hat, quantiles = c(10, 90))
> head(obj$data, 3)

                    var1                  var2 N ... EBGM QUANT_10 QUANT_90
1         1-PHENYLALANINE  HEART RATE INCREASED 1 ... 2.23     0.67    10.79
2 11 UNSPECIFIED VITAMINS               ASTHMA 1 ... 2.58     0.71    12.62
3 11 UNSPECIFIED VITAMINS CARDIAC FUNCTION TEST 1 ... 2.63     0.72    12.83
```

Or we can specify *EBGM* scores without limits, which may reduce computation time:

```
> ebScores(proc2, hyper_estimate = theta_hat, quantiles = NULL)$data %>% head(3)

                    var1                  var2 N ... EBGM
1         1-PHENYLALANINE  HEART RATE INCREASED 1 ... 2.23
2 11 UNSPECIFIED VITAMINS               ASTHMA 1 ... 2.58
3 11 UNSPECIFIED VITAMINS CARDIAC FUNCTION TEST 1 ... 2.63
```

Like all S3 objects in R, class "openEBGM" objects are list-like. The first element, 'data', contains the *var1-var2* pair counts, simple disproportionality scores, and EB scores. Other elements describe the hyperparameter estimation results and the quantile choices, if present.

Simple descriptive analysis

As stated previously, there are some generic functions included in **openEBGM**, two of which assist with descriptive analysis. These functions provide textual summaries of the disproportionality scores. Examples are provided below.

```
> obj <- ebScores(proc2, hyper_estimate = theta_hat, quantiles = c(10, 90))
> obj

There were 157 var1-var2 pairs with a QUANT_10 greater than 2

Top 5 Highest QUANT_10 Scores
                           var1                        var2  N ... QUANT_10
13924              REUMOFAN PLUS          WEIGHT INCREASED 16 ...    17.22
8187  HYDROXYCUT REGULAR RAPID...        EMOTIONAL DISTRESS 19 ...    12.69
13886              REUMOFAN PLUS                  IMMOBILE  6 ...    11.74
4093  EMERGEN-C (ASCORBIC ACID...                     COUGH  6 ...    10.29
7793       HYDROXYCUT HARDCORE... CARDIO-RESPIRATORY DISTRESS  8 ...    10.23
```

When the `print()` function is executed on the object, the textual output gives a brief overview of the highest EB scores for the lowest quantile calculated. In the absence of quantiles, the highest *EBGM* scores are returned with their associated *var1-var2* pairs. In addition, it states how many *var1-var2* pairs with a minimal quantile score above 2 existed in the data. Two is used as a rule of thumb in determining whether a pair is observed more than would be expected.

When `summary()` is called on an "openEBGM" object, the output includes a numerical summary on the EB scores. One may use the `log.trans=TRUE` argument to $\log_2$ transform the *EBGM* scores beforehand, in order to get information on the "Bayesian version of the information statistic" (DuMouchel, 1999, p. 180).

```
> summary(obj)

Summary of the EB-Metrics
      EBGM           QUANT_10          QUANT_90
 Min.   : 0.200   Min.   : 0.0900   Min.   : 0.43
 1st Qu.: 2.010   1st Qu.: 0.6500   1st Qu.: 9.19
 Median : 2.390   Median : 0.6900   Median :11.62
 Mean   : 2.355   Mean   : 0.7266   Mean   :10.42
 3rd Qu.: 2.580   3rd Qu.: 0.7100   3rd Qu.:12.57
 Max.   :23.260   Max.   :17.2200   Max.   :31.07
```

Graphical analysis

In addition to the above descriptive analysis, plots are exceedingly helpful when analyzing disproportionality scores. There are a number of different plot types included in the **openEBGM** package, all of which utilize the **ggplot2** package (Wickham and Chang, 2016; Wickham, 2009). The plots may

be used to diagnose the performance of the package, as well as to aid in analysis and identification of interesting *var1-var2* pairs. All of the plots are created by using the generic `plot()` function when called on an `"openEBGM"` object. The plot types include histograms, bar plots and shrinkage plots, and may be specified using the `plot.type` parameter in the generic function.

For all of the plots that follow, they may be created for the entire dataset in general, or with a specified event. When one wishes to look at the *EBGM* scores (and corresponding quantiles, etc.) for 'var1' observations corresponding to a specific 'var2' variable, the argument 'event' may be used in the `plot()` function call. An example is provided for bar plots using this feature.

Bar plots

It may be of interest to the researcher to see the comparison of *var1-var2* pairs in the disproportionality scores. For this reason, the generic `plot()` function's default plot type is a bar plot showing *var1-var2* pairs by their *EBGM* scores, with error bars when appropriate. Counts for each pair are also displayed as an additional layer of information. When quantiles were requested in the `ebScores()` function call, then the error bars on the bar plot represent the bounds of the quantiles specified. That is, the lower end of the error bar is the lowest quantile requested, and the upper end is the highest quantile requested. When no quantiles or only one quantile is requested, then error bars are not printed onto the plot. The bars are colored by the magnitude of the *EBGM* score.

Continuing from the last example, 'obj' is an object of class `"openEBGM"`, with quantile specification of the 10^{th} and 90^{th} percentiles. Figure 3 then displays the bar plot on the data in general, corresponding to the highest *EBGM* scores for the *var1-var2* pairs, and Figure 4 displays the bar plot when only considering the event terms which match the regular expression 'CHOKING'.

```
> plot(obj)  #Figure 3
```

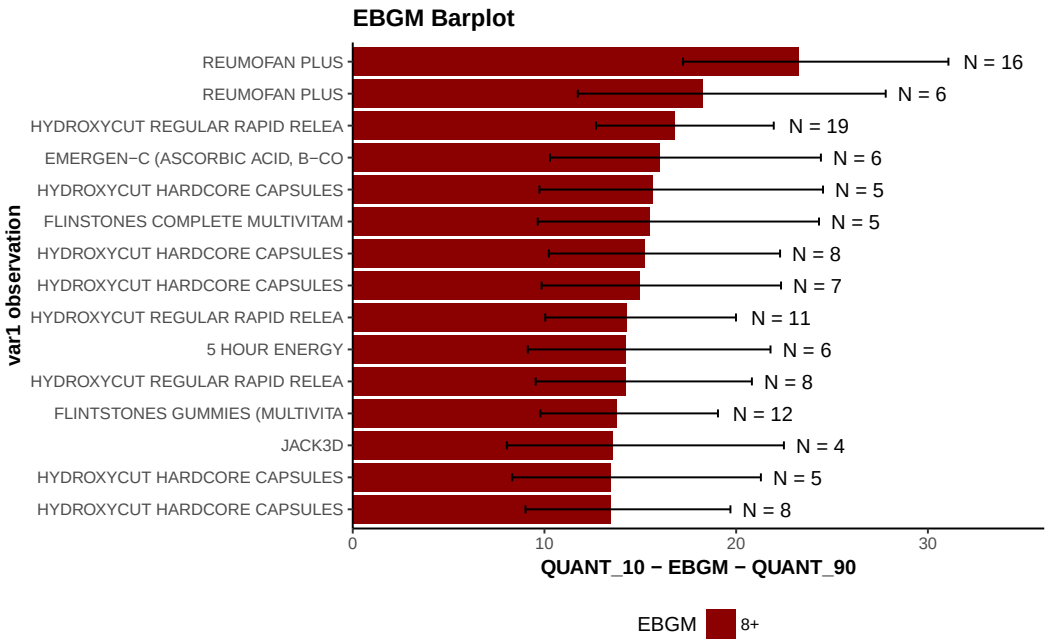

Figure 3: Top 15 product-event EBGM scores on entire dataset

```
> plot(obj, event = "CHOKING")  #Figure 4
```

Histograms

It may also be of interest to the researcher to see the distribution of *EBGM* scores. This distribution may provide the researcher with valuable insight regarding the extent of high-scoring *var1-var2* pairs, as well as their relative magnitudes. For this reason, a histogram may be created by the generic `plot()` function. The histogram output is always the *EBGM* score, regardless of whether or not quantiles were specified in the `ebScores()` function call. Figure 5 demonstrates that most of the EBGM scores are relatively small, with far fewer more interesting large scores representing unusual occurrences, illustrating why the GPS model is useful for signal detection.

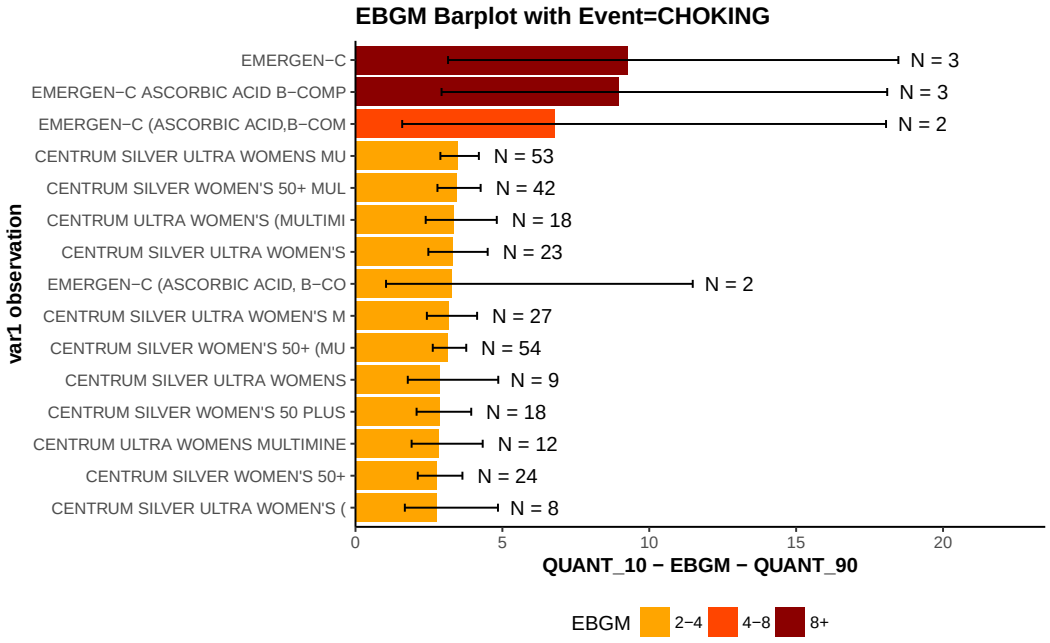

Figure 4: Top 15 product-event EBGM scores for choking events only

```
> plot(obj, plot.type = "histogram")  #Figure 5
```

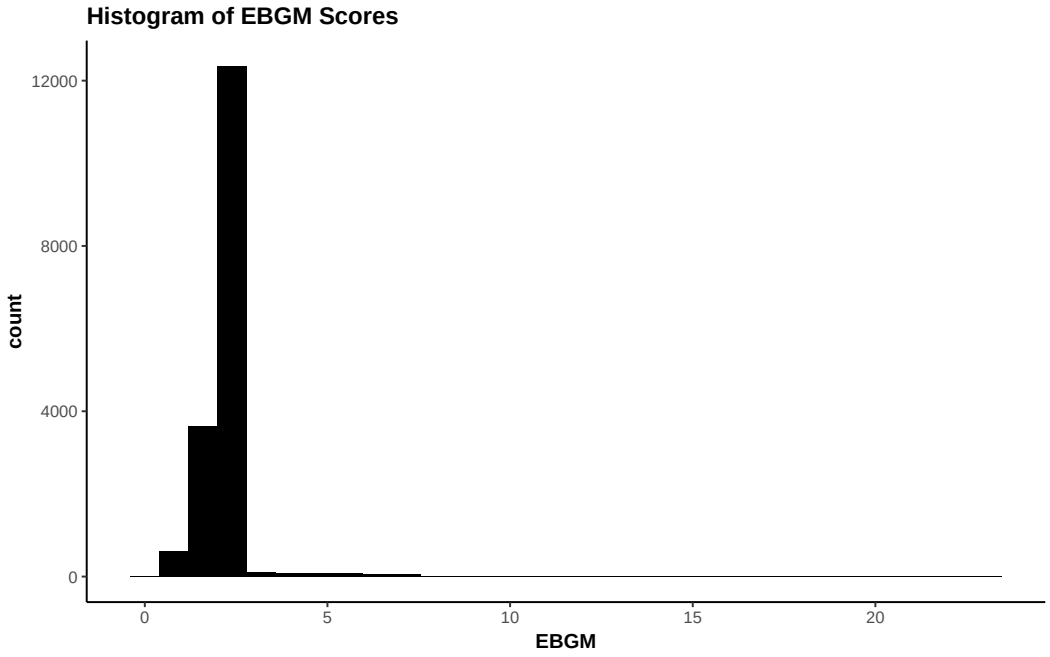

Figure 5: Histogram of EBGM scores on entire dataset

Shrinkage plots

Finally, the last plot that is included in the **openEBGM** package is a *Chirtel Squid Shrinkage Plot* (Chirtel, 2012; Duggirala et al., 2015), similar to Madigan et al. (2011, Fig.1), which shows the performance of the shrinkage algorithm on the data. In particular, it plots the *EBGM* score versus the natural log transformation of the *RR*. This plot may be useful by allowing the researcher to investigate the extent of the shrinkage that is being performed on the data, and how this shrinkage changes with varying *N* counts. The plot is colored by the count of each individually displayed *var1-var2* pair. Figure 6 illustrates that scores for product/event pairs that only occur once are greatly shrunk, but the

shrinkage lessens quickly as the number of occurrences increases. For this reason, single counts are typically not considered signals, effectively eliminating many of the false signals that occur with the simple relative reporting ratio.

```
> plot(obj, plot.type = "shrinkage")  #Figure 6
```

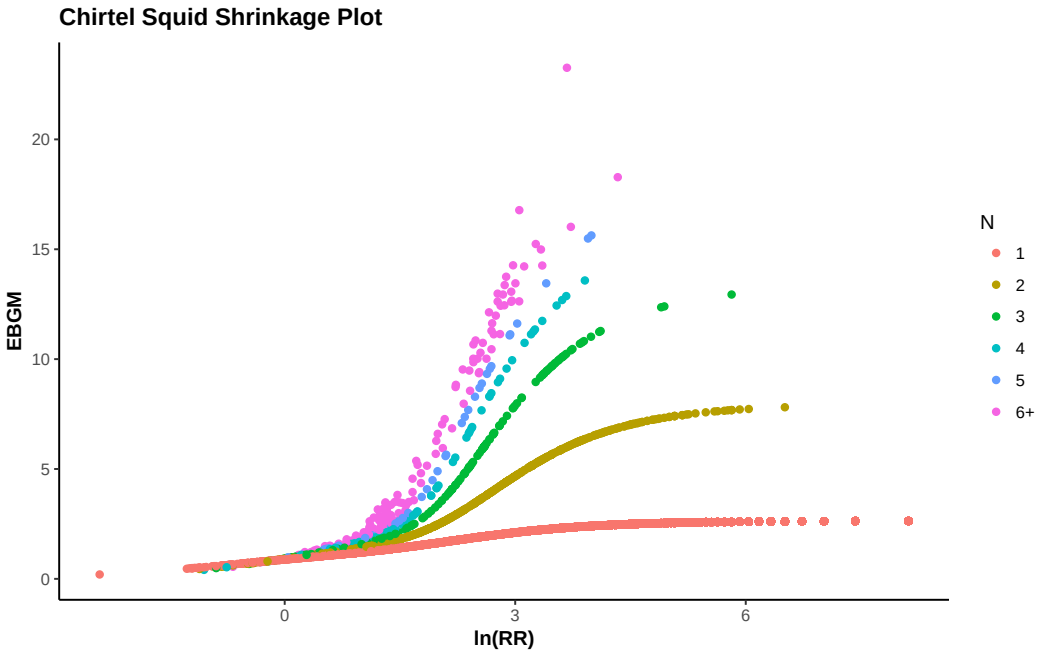

Figure 6: Shrinkage performance plot on entire dataset

Computational efficiency

As mentioned earlier, the **openEBGM** package implements the GPS model, taking into account the computational requirements that are involved when working with contingency tables, as well as when exploring a parameter space for the purposes of log-likelihood maximization (here, minimization of negative log-likelihood).

Efficient processing

As seen in the data squashing section, methodologies may be implemented that provide more efficient data processing in order to reduce overall computation time without significant loss in posterior estimates. An example is provided below showing the difference in time in hyperparameter estimation for data with and without zeroes, along with the utilization of data squashing and no data squashing.

```
> library("openEBGM")
> data("caers")
> proc_zeroes <- processRaw(caers, zeroes = TRUE)
> proc_no_zeroes <- processRaw(caers)
> squash_zeroes <- squashData(proc_zeroes, count = 0)
> squash_no_zeroes <- squashData(proc_no_zeroes)
> theta_init <- data.frame(alpha1 = c(0.2, 0.1),
+                          beta1  = c(0.1, 0.1),
+                          alpha2 = c(2,   10),
+                          beta2  = c(4,   10),
+                          p      = c(1/3, 0.2))

> system.time(hyper_zeroes <- autoHyper(squash_zeroes,
+                                       theta_init = theta_init, squashed = TRUE,
+                                       zeroes = TRUE, N_star = NULL,
+                                       max_pts = nrow(squash_zeroes)))
```

```
> system.time(hyper_no_zeroes <- autoHyper(squash_no_zeroes,
+                                          theta_init = theta_init,
+                                          squashed = TRUE, zeroes = FALSE,
+                                          N_star = 1))

> qn_zeroes <- Qn(theta_hat = hyper_zeroes$estimates,
+                 N = proc_no_zeroes$N, E = proc_no_zeroes$E)
> ebgm_zeroes <- ebgm(theta_hat = hyper_zeroes$estimates,
+                     N = proc_no_zeroes$N, E = proc_no_zeroes$E, qn = qn_zeroes)

> qn_no_zeroes <- Qn(theta_hat = hyper_no_zeroes$estimates,
+                    N = proc_no_zeroes$N, E = proc_no_zeroes$E)
> ebgm_no_zeroes <- ebgm(theta_hat = hyper_no_zeroes$estimates,
+                        N = proc_no_zeroes$N, E = proc_no_zeroes$E, qn = qn_no_zeroes)

> hyper_zeroes$estimates
> hyper_no_zeroes$estimates
```

We may look at the output of the above code to see how long it took to estimate the hyperparameters with and without zeroes, as well as their associated estimates. In the output below, for both the time elapsed in calculation as well as the estimates, the data including zeroes is displayed first.

```
   user  system elapsed
 311.73    10.09  321.83

   user  system elapsed
   2.58    0.00    2.58

   alpha1     beta1    alpha2     beta2         P
0.1874269 0.2171502 2.3409023 1.6725397 0.4582590

    alpha1      beta1     alpha2      beta2          P
3.25091549 0.39971566 2.02580114 1.90804807 0.06539048
```

A Bland-Altman plot comparing the *EBGM* scores when using zero counts vs. strictly nonzero counts for hyperparameter estimates can be seen in Figure 7. We can also compare the difference between squashed data and non-squashed data on the estimates of the hyperparameters, as well as how long it takes to estimate them.

```
> system.time(hyper_squash <- autoHyper(data = squash_no_zeroes,
+                                       theta_init = theta_init, squashed = TRUE,
+                                       zeroes = FALSE, N_star = 1))
> system.time(hyper_no_squash <- autoHyper(data = proc_no_zeroes,
+                                          theta_init = theta_init, squashed = FALSE,
+                                          zeroes = FALSE, N_star = 1))
> hyper_squash$estimates
> hyper_no_squash$estimates

   user  system elapsed
   2.54    0.00    2.54

   user  system elapsed
  24.62    0.17   24.79

    alpha1      beta1     alpha2      beta2          P
3.25091549 0.39971566 2.02580114 1.90804807 0.06539048

    alpha1      beta1     alpha2      beta2          P
3.25599765 0.39999135 2.02374652 1.90612584 0.06530695
```

As we see in the example above, the difference in *EBGM* estimates when comparing the use of squashing and not squashing for hyperparameter estimation is minimal, and by analysis of the Bland-Altman plot of the *EBGM* scores (see Figure 8), the results are nearly identical.

We see that not including zero counts makes the estimation of the hyperparameters far more computationally feasible than when one uses zeroes, which would occur if we used contingency table

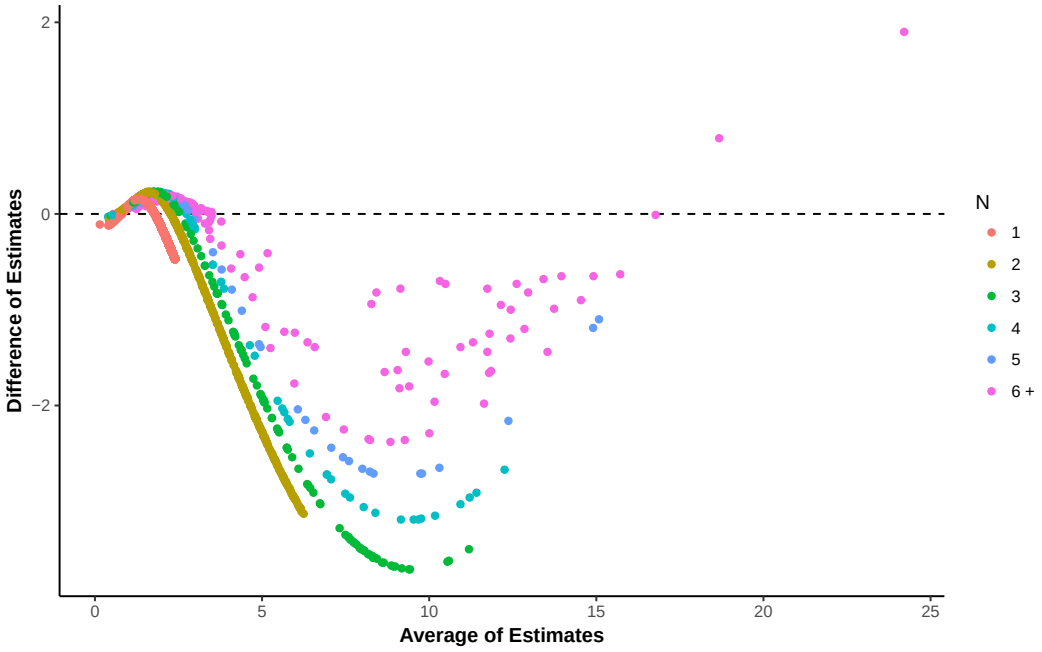

Figure 7: Bland-Altman plot of *EBGM* scores with and without zeroes (zeroes - no zeroes)

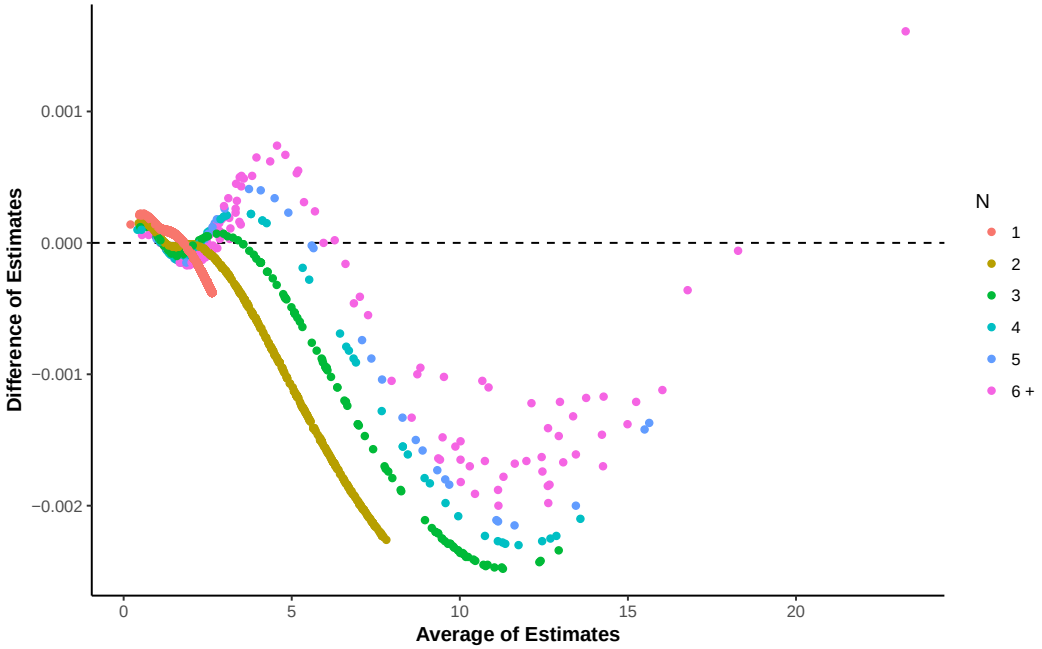

Figure 8: Bland-Altman plot of *EBGM* scores with and without data squashing (data squashing - no data squashing)

data structures instead of tidy data where zeroes are generally not needed. Appendix 2.11 further illustrates efficiency gains from removing zeroes.

Additionally, **openEBGM** utilizes other strategies to reduce the amount of unnecessary computation required to employ the GPS model. In particular, the package uses certain data structures which represent the sometimes large contingency table of *var1-var2* pairs as a `"data.table"` (Dowle and Srinivasan, 2017) in tidy form (Wickham, 2014). This results in more efficient data squashing and computation of marginal totals and expected counts. Representing data in tidy form instead of a matrix resembling a contingency table avoids inefficient row-wise operations involving many unnecessary zeroes. Furthermore, the documentation for **data.table** claims fast merging of `"data.table"` objects. While developing the code, we noticed a sizable speed improvement for the merging and aggregation methods for `"data.table"` objects over the standard methods for traditional data frames. Appendix

2.12 demonstrates that **openEBGM** can process very large data sets in a reasonable amount of time, even when including zeroes for hyperparameter estimation.

Conclusion and discussion

Our package greatly simplifies the process of generating disproportionality scores for large contingency tables. In addition, the analysis of these scores is simplified using built-in functions which summarize and display the results. Furthermore, **openEBGM**'s implementation of the GPS model aims for greater efficiency (see Appendix 2.10), flexibility and usability compared to other open-source implementations of the same model.

Further applications

Adverse event reporting is not the only application of the GPS methodology. DuMouchel (1999) provides other examples, including natural language processing. GPS can find word pairs that appear more frequently than expected, which could prove to be useful for crawling the web in search of trends (e.g. document searches). DuMouchel also suggests using the model to find supermarket products that are often purchased together (stratified by store location).

Multi-item gamma-Poisson shrinker

openEBGM currently implements the GPS model, which can only examine single *var1-var2* pairs. An extension of this model exists, allowing the study of interactions. The FDA currently uses (Duggirala et al., 2016; Szarfman et al., 2002) the *Multi-item Gamma-Poisson Shrinker* (MGPS) model (DuMouchel and Pregibon, 2001) to find higher-than-expected reporting of adverse events associated with specific products or product groups. MGPS can model Drug-Drug-Event or Drug-Event-Event triples (with higher order interactions also being possible). DuMouchel and Pregibon (2001) use a log-linear model in the MGPS approach to account for lower-order associations. More discussion on the details of MGPS can be found in Szarfman et al. (2002). Existing proprietery implementations of MGPS include Oracle Health Science's Empirica Signal (Oracle, 2017) and some versions of JMP® Clinical software (SAS Institute Inc., 2017). Future improvements to **openEBGM** could include the addition of the MGPS model.

Acknowledgements

We thank Stuart Chirtel for introducing us to the GPS model, helping us understand its use, and encouraging us to implement it in R. We thank Hugh Rand for valuable insight into numerical techniques and for encouraging us to add our code to the CRAN repository and write this article. We also thank Hugh, Stuart, John Bowers, and the journal reviewers for valuable comments and suggestions. We thank James Pettengill for helping us test the code. We thank Lyle Canida and Ella Smith for helping us obtain and understand the CAERS data; thanks also to everyone else involved in creating, maintaining, and disseminating the CAERS data. Finally, we thank the authors of the **PhViD** package, whose code gave us a starting point to develop our own code.

Bibliography

I. Ahmed and A. Poncet. *PhViD: PharmacoVigilance Signal Detection*, 2016. URL https://CRAN.R-project.org/package=PhViD. R package version 1.0.8. [p501]

I. Ahmed, F. Haramburu, A. Fourrier-Réglat, F. Thiessard, C. Kreft-Jais, G. Miremont-Salamé, B. Bégaud, and P. Tubert-Bitter. Bayesian pharmacovigilance signal detection methods revisited in a multiple comparison setting. *Statistics in Medicine*, 28(13):1774–1792, 2009. ISSN 1097-0258. URL https://doi.org/10.1002/sim.3586. [p501]

J. S. Almenoff, K. K. LaCroix, N. A. Yuen, D. Fram, and W. DuMouchel. Comparative performance of two quantitative safety signalling methods. *Drug Safety*, 29(10):875–887, October 2006. ISSN 1179-1942. URL https://doi.org/10.2165/00002018-200629100-00005. [p499]

S. Chirtel. Disproportionality analyses for detection of food adverse events. ENAR 2012 Spring Meeting, Washington, D.C., USA, 2012. URL https://www.enar.org/meetings/meetings2012/Meeting_ABSTRACTS.pdf. [p510]

M. Dowle and A. Srinivasan. *data.table: Extension of 'data.frame'*, 2017. URL https://CRAN.R-project.org/package=data.table. R package version 1.10.4. [p513]

H. Duggirala, J. Tonning, E. Smith, R. Bright, J. Baker, R. Ball, C. Bell, K. Bouri, S. J. Bright-Ponte, T. Botsis, M. Boyer, K. Burkhart, G. Condrey, J. Chen, S. Chirtel, R. Filice, H. Francis, H. Jiang, J. Levine, D. Martin, T. Oladipo, R. O'Neill, L. Palmer, A. Paredes, G. Rochester, D. Sholtes, H. Wong, Z. Xu, A. Szarfman, and T. Kass-Hout. Data mining at FDA. Technical report, U.S. Food and Drug Administration, November 2015. URL https://www.fda.gov/scienceresearch/dataminingatfda/ucm446239. [p499, 510]

H. Duggirala, J. Tonning, E. Smith, R. Bright, J. Baker, R. Ball, C. Bell, S. J. Bright-Ponte, T. Botsis, K. Bouri, M. Boyer, K. Burkhart, G. Condrey, J. Chen, S. Chirtel, R. Filice, H. Francis, H. Jiang, J. Levine, D. Martin, T. Oladipo, R. O'Neill, L. Palmer, A. Paredes, G. Rochester, D. Sholtes, A. Szarfman, H. Wong, Z. Xu, and T. Kass-Hout. Use of data mining at the Food and Drug Administration. *Journal of the American Medical Informatics Association*, 23(2):428–434, 2016. URL https://doi.org/10.1093/jamia/ocv063. [p514]

W. DuMouchel. Bayesian data mining in large frequency tables, with an application to the FDA spontaneous reporting system. *The American Statistician*, 53(3):177–190, August 1999. URL https://doi.org/10.1080/00031305.1999.10474456. [p499, 500, 502, 504, 505, 506, 507, 508, 514]

W. DuMouchel and D. Pregibon. Empirical bayes screening for multi-item associations. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pages 67–76, New York, NY, USA, 2001. ACM. ISBN 1-58113-391-X. URL https://doi.org/10.1145/502512.502526. [p499, 504, 505, 506, 514]

S. J. W. Evans, P. Waller, and S. Davis. Use of proportional reporting ratios (PRRs) for signal generation from spontaneous adverse drug reaction reports. *Pharmacoepidemiology and Drug Safety*, 10(6):483–486, 2001. ISSN 1099-1557. URL https://doi.org/10.1002/pds.677. [p499]

J. Ihrie and T. Canida. *openEBGM: EBGM Scores for Mining Large Contingency Tables*, 2017. URL https://CRAN.R-project.org/package=openEBGM. R package version 0.3.0. [p499]

D. Madigan, P. Ryan, S. Simpson, and I. Zorych. Bayesian methods in pharmacovigilance. In J. Bernardo, M. Bayarri, J. Berger, A. Dawid, D. Heckerman, A. Smith, and M. West, editors, *Bayesian Statistics 9*. Oxford University Press, 2011. ISBN 9780199694587. URL https://doi.org/10.1093/acprof:oso/9780199694587.003.0014. [p499, 510]

Oracle. Oracle Health Sciences Empirica Signal, 2017. URL https://www.oracle.com/us/products/applications/health-sciences/safety/empirica-signal/. [p514]

SAS Institute Inc. JMP Clinical: Clinical data analysis software for ensuring trial safety and efficacy, 2017. URL https://www.jmp.com/en_us/software/clinical-data-analysis-software.html. [p514]

A. Szarfman, S. G. Machado, and R. T. O'Neill. Use of screening algorithms and computer systems to efficiently signal higher-than-expected combinations of drugs and events in the US FDA's spontaneous reports database. *Drug Safety*, 25(6):381–392, 2002. ISSN 1179-1942. URL https://doi.org/10.2165/00002018-200225060-00001. [p500, 501, 514]

U.S. Food and Drug Administration. CFSAN Adverse Event Reporting System (CAERS), 2017. URL https://www.fda.gov/Food/ComplianceEnforcement/ucm494015.htm. [p501, 502]

S. Venturini and J. Myers. *mederrRank: Bayesian Methods for Identifying the Most Harmful Medication Errors*, 2015. URL https://CRAN.R-project.org/package=mederrRank. R package version 0.0.8. [p501]

S. Venturini, J. M. Franklin, L. Morlock, and F. Dominici. Random effects models for identifying the most harmful medication errors in a large, voluntary reporting database. *The Annals of Applied Statistics*, 11(2):504–526, 2017. URL https://doi.org/10.1214/16-AOAS974. [p501]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2009. ISBN 978-0-387-98140-6. URL http://ggplot2.org. [p508]

H. Wickham. Tidy data. *Journal of Statistical Software, Articles*, 59(10):1–23, 2014. ISSN 1548-7660. URL https://doi.org/10.18637/jss.v059.i10. [p501, 513]

H. Wickham. *tidyr: Easily Tidy Data with 'spread()' and 'gather()' Functions*, 2017. URL https://CRAN.R-project.org/package=tidyr. R package version 0.6.3. [p502]

H. Wickham and W. Chang. *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*, 2016. URL https://CRAN.R-project.org/package=ggplot2. R package version 2.2.1. [p508]

Travis Canida
U.S. Food and Drug Administration
Center for Food Safety and Applied Nutrition
Biostatistics and Bioinformatics Staff
5001 Campus Drive
College Park, MD 20740
United States
Travis.Canida@fda.hhs.gov

John Ihrie
U.S. Food and Drug Administration
Center for Food Safety and Applied Nutrition
Biostatistics and Bioinformatics Staff
5001 Campus Drive
College Park, MD 20740
United States
John.Ihrie@fda.hhs.gov

Appendix A

```
# Purpose: To compare computing times for openEBGM vs. PhViD.
# Notes:   We continue from the data set created in the CAERS data set example.
#          PhViD also computes many multiplicity-adjusted values not shown here.
#          The pre-2017 CAERS data set might change slightly over time.

> library("PhViD")
> dat_tidy$id <- 1:nrow(dat_tidy)  #since PhViD cannot count unique reports
> counts <- processRaw(dat_tidy)
> nrow(counts)  #number of points/var1-var2 pairs

[1] 145257

> theta_init <- c(alpha1 = 0.2, beta1 = 0.06, alpha2 = 1.4, beta2 = 1.8, P = 0.1)

#Start with the PhViD package
> system.time({
+   dat_phvid <- as.PhViD(counts[, 1:3])
+   results_phvid <- GPS(dat_phvid, RANKSTAT = 3, TRONC = TRUE, PRIOR.INIT = theta_init)
+ })

   user  system elapsed
 342.08    4.32  346.46

> results_phvid <- results_phvid$ALLSIGNALS[, 1:6]
> results_phvid <- results_phvid[order(results_phvid$drug, results_phvid$event), ]
> results_phvid$EBGM <- round(2 ^ results_phvid[, 'post E(Lambda)'], 3)
> row.names(results_phvid) <- NULL
> head(results_phvid, 3)

                            drug             event count ...    n11/E  EBGM
1  CENTRUM SILVER WOMEN'S 50+...           CHOKING     2 ... 14.27370 1.639
2  CENTRUM SILVER WOMEN'S 50+...         DYSPHAGIA     1 ... 13.53379 1.153
3  CENTRUM SILVER WOMEN'S 50+... THROAT IRRITATION     1 ... 48.00568 1.187

#Now for the openEBGM package
> theta_init_df <- t(data.frame(theta_init))
> system.time({
+   theta1 <- exploreHypers(counts, theta_init = theta_init_df, squashed = FALSE,
+                           method = "nlm", max_pts = nrow(counts))$estimates
```

```
+   theta1 <- as.numeric(theta1[1, 2:6])
+   results_open1 <- ebScores(counts, list(estimates = theta1), quantiles = NULL)
+ })

   user  system elapsed
  84.18    0.02   84.19

> dat_open1 <- results_open1$data
> head(dat_open1, 3)

                            var1              var2 N ...    RR ... EBGM
1  CENTRUM SILVER WOMEN'S 50+...           CHOKING 2 ... 14.27 ... 1.64
2  CENTRUM SILVER WOMEN'S 50+...         DYSPHAGIA 1 ... 13.53 ... 1.15
3  CENTRUM SILVER WOMEN'S 50+... THROAT IRRITATION 1 ... 48.01 ... 1.19

#Now with data squashing
> system.time({
+   squashed <- squashData(counts, bin_size = 100)
+   squashed <- squashData(squashed, count = 2, bin_size = 10)
+   theta2 <- exploreHypers(squashed, theta_init = theta_init_df,
+                           method = "nlm")$estimates
+   theta2 <- as.numeric(theta2[1, 2:6])
+   results_open2 <- ebScores(counts, list(estimates = theta2), quantiles = NULL)
+ })

   user  system elapsed
   5.64    0.03    5.68

> dat_open2 <- results_open2$data
> head(dat_open2, 3)

                            var1              var2 N ...    RR ... EBGM
1  CENTRUM SILVER WOMEN'S 50+...           CHOKING 2 ... 14.27 ... 1.64
2  CENTRUM SILVER WOMEN'S 50+...         DYSPHAGIA 1 ... 13.53 ... 1.15
3  CENTRUM SILVER WOMEN'S 50+... THROAT IRRITATION 1 ... 48.01 ... 1.19
```

Appendix B

Rows	Without zeroes			With zeroes		
	Unstratified	Stratified	Points	Unstratified	Stratified	Points
50K	0.6	0.8	40K	32	47	18M
100K	1.3	1.9	80K	84	122	44M
150K	2.2	2.9	118K	157	226	72M
200K	3.1	3.8	156K	205	308	103M
250K	3.0	3.9	194K	273	430	139M
300K	3.6	4.6	227K	352	519	171M

Elapsed time (in seconds) and approximate number of points (N, E) for `processRaw()` on various sizes of raw data. Using gender only when stratifying. K = 1,000; M = 1,000,000

Appendix C

```
# Purpose: To assess how quickly openEBGM can process a very large data set.
# Notes: We include all industry codes. We also include zero counts in the
#        hyperparameter estimation step to further illustrate how quickly openEBGM
#        can process a very large number of points (~177 million).
```

```
> site <- "https://www.fda.gov/downloads/Food/ComplianceEnforcement/UCM494018.csv"
> dat  <- read.csv(site, stringsAsFactors = FALSE, strip.white = TRUE)
> dat$yr <- dat$RA_CAERS.Created.Date
> dat$yr <- substr(dat$yr, start = nchar(dat$yr) - 3, stop = nchar(dat$yr))
> dat$yr <- as.integer(dat$yr)
> dat <- dat[dat$yr < 2017, ]  #using all industry codes
> dat$var1 <- dat$PRI_Reported.Brand.Product.Name
> dat$var2 <- dat$SYM_One.Row.Coded.Symptoms
> dat$id   <- dat$RA_Report..
> dat$strat_gen <- dat$CI_Gender
> dat$strat_gen <- ifelse(dat$strat_gen %in% c("Female", "Male"),
+                         dat$strat_gen, "unknown")
> vars <- c("id", "var1", "var2", "strat_gen")
> dat <- dat[, vars]
> dat <- dat[!dat$var1 %in% tools::showNonASCII(dat$var1), ]
> dat_tidy <- tidyr::separate_rows(dat, var2, sep = ", ")
> dat_tidy <- dat_tidy[dat_tidy$var2 != "", ]
> nrow(dat_tidy)  #rows in raw data

 [1] 307719

> system.time(counts <- processRaw(dat_tidy, zeroes = TRUE))

 user system elapsed
 289.07 63.22 352.33

> nrow(counts)  #number of points/var1-var2 pairs in processed data

 [1] 176739728

> system.time({
+   squashed <- squashData(counts, count = 0, bin_size = 50000, keep_bins = 0)
+ })

 user system elapsed
 75.38 12.40 87.80

> nrow(squashed)

 [1] 235734

> system.time({
+   squashed <- squashData(squashed, bin_size = 500, keep_bins = 1)
+   squashed <- squashData(squashed, count = 2, bin_size = 100, keep_bins = 1)
+   squashed <- squashData(squashed, count = 3, bin_size = 50, keep_bins = 1)
+   squashed <- squashData(squashed, count = 4, bin_size = 25, keep_bins = 1)
+   squashed <- squashData(squashed, count = 5, bin_size = 10, keep_bins = 1)
+   squashed <- squashData(squashed, count = 6, bin_size = 10, keep_bins = 1)
+ })

 user system elapsed
 0.16 0.00 0.16

> nrow(squashed)  #number of points used for hyperparameter estimation

 [1] 7039

> theta_init <- c(alpha1 = 0.2, beta1 = 0.06, alpha2 = 1.4, beta2 = 1.8, P = 0.1)
> theta_init_df <- t(data.frame(theta_init))

> system.time({
+   theta_est <- exploreHypers(squashed, theta_init = theta_init_df,
+                              squashed = TRUE, zeroes = TRUE, N_star = NULL,
```

```
+                              method = "nlminb", std_errors = TRUE)
+   theta_hat <- theta_est$estimates
+   theta_hat <- as.numeric(theta_hat[1, 2:6])
+ })

   user  system elapsed
   4.63    0.09    4.72

> theta_hat

[1] 0.036657366 0.006645404 0.681479849 0.605705800 0.020295102

> theta_est$std_errs  #standard errors of hyperparameter estimates

  guess_num       a1_se       b1_se      a2_se       b2_se       p_se
1         1 0.007072148 0.000275861 0.00964448 0.008095735 0.00389658

> system.time({
+   counts_sans0 <- counts[counts$N != 0, ]  #do not need EB scores for zero counts
+   results <- ebScores(counts_sans0, list(estimates = theta_hat), quantiles = NULL)
+ })

 user system elapsed
 2.95 0.02 2.97

> nrow(results$data)  #number of nonzero counts

 [1] 232203
```