Package ‘glmGamPoi’

October 17, 2020

Type Package

Title Fit a Gamma-Poisson Generalized Linear Model

Version 1.0.0

Description Fit linear models to overdispersed count data.
The package can estimate the overdispersion and fit repeated models
for matrix input. It is designed to handle large input datasets as they
typically occur in single cell RNA-seq experiments.

License GPL-3

Encoding UTF-8

SystemRequirements C++11

Suggests testthat (>= 2.1.0), zoo, DESeq2, edgeR, beachmat, MASS,
statmod, ggplot2, bench, BiocParallel, knitr, rmarkdown,
BiocStyle, TENxPBMCData

LinkingTo Rcpp, RcppArmadillo, beachmat (>= 2.0.0)

Imports Rcpp, pracma, DelayedMatrixStats, DelayedArray, HDF5Array,
SummarizedExperiment, methods, stats, utils

Roxygen list(markdown = TRUE)

RoxygenNote 7.0.2

URL https://github.com/const-ae/glmGamPoi

BugReports https://github.com/const-ae/glmGamPoi/issues

biocViews Regression, RNASeq, Software, SingleCell

VignetteBuilder knitr

git_url https://git.bioconductor.org/packages/glmGamPoi

git_branch RELEASE_3_11

git_last_commit ea2224c

git_last_commit_date 2020-04-27

Date/Publication 2020-10-16

Author Constantin Ahlmann-Eltze [aut, cre]
(<https://orcid.org/0000-0002-3762-068X>),
Michael Love [ctb]

Maintainer Constantin Ahlmann-Eltze <artjom31415@googlemail.com>

R topics documented:

`as.list.glmGamPoi` *Convert glmGamPoi object to a list*

Description

Convert glmGamPoi object to a list

Usage

```
## S3 method for class 'glmGamPoi'
as.list(x, ...)
```

Arguments

`x`	an object with class glmGamPoi
`...`	not used

Value

The method returns a list with the following elements:

`Beta` a matrix with dimensions nrow(data) x n_coefficients where `n_coefficients` is based on the `design` argument. It contains the estimated coefficients for each gene.

`overdispersions` a vector with length `nrow(data)`. The overdispersion parameter for each gene. It describes how much more the counts vary than one would expect according to the Poisson model.

`Mu` a matrix with the same dimensions as `dim(data)`. If the calculation happened on disk, than `Mu` is a `HDF5Matrix`. It contains the estimated mean value for each gene and sample.

`size_factors` a vector with length `ncol(data)`. The size factors are the inferred correction factors for different sizes of each sample. They are also sometimes called the exposure factor.

`model_matrix` a matrix with dimensions ncol(data) x n_coefficients. It is build based on the `design` argument.

gampoi_overdispersion_mle
Estimate the Overdispersion for a Vector of Counts

Description

Estimate the Overdispersion for a Vector of Counts

Usage

```
gampoi_overdispersion_mle(
  y,
  mean = base::mean(y),
  model_matrix = matrix(1, nrow = length(y), ncol = 1),
  do_cox_reid_adjustment = TRUE,
  subsample = FALSE,
  verbose = FALSE
)
```

Arguments

- `y` — a numeric or integer vector with the counts for which the overdispersion is estimated
- `mean` — a numeric vector of either length 1 or `length(y)` with the predicted value for that sample. Default: `mean(y)`.
- `model_matrix` — a numeric matrix that specifies the experimental design. It can be produced using `stats::model.matrix()`. Default: `matrix(1,nrow = length(y),ncol = 1)`, which is the model matrix for a 'just-intercept-model'.
- `do_cox_reid_adjustment` — the classical maximum likelihood estimator of the `overdisperion` is biased towards small values. McCarthy *et al.* (2012) showed that it is preferable to optimize the Cox-Reid adjusted profile likelihood.
  `do_cox_reid_adjustment` can be either be `TRUE` or `FALSE` to indicate if the adjustment is added during the optimization of the `overdispersion` parameter. Default: `TRUE`.
- `subsample` — the estimation of the overdispersion is the slowest step when fitting a Gamma-Poisson GLM. For datasets with many samples, the estimation can be considerably sped up without loosing much precision by fitting the overdispersion only on a random subset of the samples. Default: `FALSE` which means that the data is not subsampled. If set to `TRUE`, at most 1,000 samples are considered. Otherwise the parameter just specifies the number of samples that are considered for each gene to estimate the overdispersion.
- `verbose` — a boolean that indicates if information about the individual steps are printed while fitting the GLM. Default: `FALSE`.

Details

The function employs a rough heuristic to decide if the iterative or the Bandara approach is used to calculate the overdispersion. If `max(y) < length(y)` Bandara's approach is used, otherwise the conventional one is used.

Value

The function returs a list with the following elements:

`estimate` the numerical estimate of the overdispersion.

`iterations` the number of iterations it took to calculate the result.

`method` the method that was used to calculate the overdispersion: either `"conventional"` or `"bandara"`.

`message` additional information about the fitting process.

See Also

`glm_gp()`

Examples

```
set.seed(1)
# true overdispersion = 2.4
y <- rnbinom(n = 10, mu = 3, size = 1/2.4)
# estimate = 1.7
gampoi_overdispersion_mle(y)


# true overdispersion = 0
y <- rpois(n = 10, lambda = 3)
# estimate = 0
gampoi_overdispersion_mle(y)
# with different mu, overdispersion estimate changes
gampoi_overdispersion_mle(y, mean = 15)
# Cox-Reid adjustment changes the result
gampoi_overdispersion_mle(y, mean = 15, do_cox_reid_adjustment = FALSE)


# Many very small counts, true overdispersion = 50
y <- rnbinom(n = 1000, mu = 0.01, size = 1/50)
summary(y)
# estimate = 31
gampoi_overdispersion_mle(y)
```

`glm_gp` *Fit a Gamma-Poisson Generalized Linear Model*

Description

This function provides a simple to use interface to fit Gamma-Poisson generalized linear models. It works equally well for small scale (a single model) and large scale data (e.g. thousands of rows and columns, potentially stored on disk). The function automatically determines the appropriate size factors for each sample and efficiently finds the best overdispersion parameter for each gene.

Usage

```
glm_gp(
  data,
  design = ~1,
  col_data = NULL,
  reference_level = NULL,
  offset = 0,
  size_factors = TRUE,
  overdispersion = TRUE,
  do_cox_reid_adjustment = TRUE,
  subsample = FALSE,
  on_disk = NULL,
  verbose = FALSE
)
```

Arguments

`data` any matrix-like object (e.g. matrix, DelayedArray, HDF5Matrix) or anything that can be cast to a SummarizedExperiment (e.g. `MSnSet`, `eSet` etc.) with one column per sample and row per gene.

`design` a specification of the experimental design used to fit the Gamma-Poisson GLM. It can be a `model.matrix()` with one row for each sample and one column for each coefficient.
Alternatively, `design` can be a `formula`. The entries in the formula can refer to global objects, columns in the `col_data` parameter, or the `colData(data)` of `data` if it is a `SummarizedExperiment`.
The third option is that `design` is a vector where each element specifies to which condition a sample belongs.
Default: `design = ~ 1`, which means that all samples are treated as if they belong to the same condition. Note that this is the fasted option.

`col_data` a dataframe with one row for each sample in `data`. Default: `NULL`.

`reference_level`
a single string that specifies which level is used as reference when the model matrix is created. The reference level becomes the intercept and all other coefficients are calculated with respect to the `reference_level`. Default: `NULL`.

`offset` Constant offset in the model in addition to `log(size_factors)`. It can either be a single number, a vector of length `ncol(data)` or a matrix with the same dimensions as `dim(data)`. Note that if data is a DelayedArray or HDF5Matrix, `offset` must be as well. Default: `0`.

`size_factors` in large scale experiments, each sample is typically of different size (for example different sequencing depths). A size factor is an internal mechanism of GLMs to correct for this effect.
`size_factors` can either be a single boolean that indicates if the size factor for each sample should be calculated. Or it is a numeric vector that specifies the size factor for each sample. Note that `size_factors = 1` and `size_factors = FALSE` are equivalent. Default: `TRUE`.

`overdispersion` the simplest count model is the Poisson model. However, the Poisson model assumes that $variance = mean$. For many applications this is too rigid and the Gamma-Poisson allows a more flexible mean-variance relation ($variance = mean + mean^2 * overdispersion$).

overdispersion can either be a single boolean that indicates if an overdispersion is estimated for each gene. Or it can be a numeric vector of length nrow(data). Note that overdispersion = 0 and overdispersion = FALSE are equivalent and both reduce the Gamma-Poisson to the classical Poisson model. Default: TRUE.

do_cox_reid_adjustment
: the classical maximum likelihood estimator of the overdisperion is biased towards small values. McCarthy *et al.* (2012) showed that it is preferable to optimize the Cox-Reid adjusted profile likelihood.
do_cox_reid_adjustment can be either be TRUE or FALSE to indicate if the adjustment is added during the optimization of the overdispersion parameter. Default: TRUE.

subsample
: the estimation of the overdispersion is the slowest step when fitting a Gamma-Poisson GLM. For datasets with many samples, the estimation can be considerably sped up without loosing much precision by fitting the overdispersion only on a random subset of the samples. Default: FALSE which means that the data is not subsampled. If set to TRUE, at most 1,000 samples are considered. Otherwise the parameter just specifies the number of samples that are considered for each gene to estimate the overdispersion.

on_disk
: a boolean that indicates if the dataset is loaded into memory or if it is kept on disk to reduce the memory usage. Processing in memory can be significantly faster than on disk. Default: NULL which means that the data is only processed in memory if data is an in-memory data structure.

verbose
: a boolean that indicates if information about the individual steps are printed while fitting the GLM. Default: FALSE.

Details

The method follows the following steps:

1. The size factors are estimated.
The code is a slightly adapted version of the procedure proposed by Anders and Huber (2010) in equation (5). To handle the large number of zeros the geometric means are calculated for $Y + 0.5$ and ignored during the calculation of the median. Columns with all zeros get a default size factor of 0.001.
2. The dispersion estimates are initialized based on the moments of each row of Y.
3. The coefficients of the model are estimated.
If all samples belong to the same condition (i.e. design = ~ 1), the betas are estimated using a quick Newton-Raphson algorithm. This is similar to the behavior of edgeR. For more complex designs, the general Fisher-scoring algorithm is used. Here, the code is based on a fork of the internal function fitBeta() from DESeq2. It does however contain some modification to make the fit more robust and faster.
4. The mean for each gene and sample is calculate.
Note that this step can be very IO intensive if data is or contains a DelayedArray.
5. The overdispersion is estimated.
The classical method for estimating the overdispersion for each gene is to maximize the Gamma-Poisson log-likelihood by iterating over each count and summing the the corresponding log-likelihood. It is however, much more efficient for genes with many small counts to work on the contingency table of the counts. Originally, this approach had already been used by Anscombe (1950), but only recently it has been formulated with an efficient Newton-Raphson approach by Bandara *et al.* (2019). In this package, I have implemented an extension

of their method that can handle general offsets.
See also `gampoi_overdispersion_mle()`.

6. The beta coefficients are estimated once more with the updated overdispersion estimates
7. The mean for each gene and sample is calculated again.

This method can handle not just in memory data, but also data stored on disk. This is essential for large scale datasets with thousands of samples, as they sometimes encountered in modern single-cell RNA-seq analysis. `glmGamPoi` relies on the `DelayedArray` and `beachmat` package to efficiently implement the access to the on-disk data.

Value

The method returns a list with the following elements:

`Beta` a matrix with dimensions nrow(data) x n_coefficients where `n_coefficients` is based on the `design` argument. It contains the estimated coefficients for each gene.

`overdispersions` a vector with length `nrow(data)`. The overdispersion parameter for each gene. It describes how much more the counts vary than one would expect according to the Poisson model.

`Mu` a matrix with the same dimensions as `dim(data)`. If the calculation happened on disk, than `Mu` is a `HDF5Matrix`. It contains the estimated mean value for each gene and sample.

`size_factors` a vector with length `ncol(data)`. The size factors are the inferred correction factors for different sizes of each sample. They are also sometimes called the exposure factor.

`model_matrix` a matrix with dimensions ncol(data) x n_coefficients. It is build based on the `design` argument.

References

- McCarthy, D. J., Chen, Y., & Smyth, G. K. (2012). Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation. Nucleic Acids Research, 40(10), 4288–4297. https://doi.org/10.1093/nar/gks042.
- Anders Simon, & Huber Wolfgang. (2010). Differential expression analysis for sequence count data. Genome Biology. https://doi.org/10.1016/j.jcf.2018.05.006.
- Love, M. I., Huber, W., & Anders, S. (2014). Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. Genome Biology, 15(12), 550. https://doi.org/10.1186/s13059-014-0550-8.
- Robinson, M. D., McCarthy, D. J., & Smyth, G. K. (2009). edgeR: A Bioconductor package for differential expression analysis of digital gene expression data. Bioinformatics, 26(1), 139–140. https://doi.org/10.1093/bioinformatics/btp616.
- Bandara, U., Gill, R., & Mitra, R. (2019). On computing maximum likelihood estimates for the negative binomial distribution. Statistics and Probability Letters, 148, 54–58. https://doi.org/10.1016/j.spl.2019.01.009
- Lun ATL, Pagès H, Smith ML (2018). "beachmat: A Bioconductor C++ API for accessing high-throughput biological data from a variety of R matrix types." PLoS Comput. Biol., 14(5), e1006135. doi: 10.1371/journal.pcbi.1006135..

See Also

`glm_gp_impl()` and `gampoi_overdispersion_mle()` for the internal functions that do the work.

Examples

```
set.seed(1)
# The simplest example
y <- rnbinom(n = 10, mu = 3, size = 1/2.4)
c(glm_gp(y, size_factors = FALSE))

# Fitting a whole matrix
model_matrix <- cbind(1, rnorm(5))
true_Beta <- cbind(rnorm(n = 30), rnorm(n = 30, mean = 3))
sf <- exp(rnorm(n = 5, mean = 0.7))
model_matrix
Y <- matrix(rnbinom(n = 30 * 5, mu = sf * exp(true_Beta %*% t(model_matrix)), size = 1/2.4),
            nrow = 30, ncol = 5)

fit <- glm_gp(Y, design = model_matrix, size_factors = sf, verbose = TRUE)
summary(fit)

# Fitting a model with covariates
data <- data.frame(fav_food = sample(c("apple", "banana", "cherry"), size = 50, replace = TRUE),
city = sample(c("heidelberg", "paris", "new york"), size = 50, replace = TRUE),
age = rnorm(n = 50, mean = 40, sd = 15))
Y <- matrix(rnbinom(n = 100 * 50, mu = 3, size = 1/3.1), nrow = 100, ncol = 50)
fit <- glm_gp(Y, design = ~ fav_food + city + age, col_data = data)
summary(fit)
```

---

print.glmGamPoi *Pretty print the result from glm_gp()*

Description

Pretty print the result from glm_gp()

Usage

```
## S3 method for class 'glmGamPoi'
print(x, ...)

## S3 method for class 'glmGamPoi'
format(x, ...)

## S3 method for class 'glmGamPoi'
summary(object, ...)

## S3 method for class 'summary.glmGamPoi'
print(x, ...)

## S3 method for class 'summary.glmGamPoi'
format(x, ...)
```

Arguments

`x`	the glmGamPoi object
`...`	additional parameters, currently ignored
`object`	the glmGamPoi object that is summarized

Value

The `print()` methods return the object `x`. The `format()` method returns a string. The `summary()` method returns an object of class `summary.glmGamPoi`.

`residuals.glmGamPoi` *Extract Residuals of Gamma Poisson Model*

Description

Extract Residuals of Gamma Poisson Model

Usage

```
## S3 method for class 'glmGamPoi'
residuals(
  object,
  Y,
  type = c("deviance", "pearson", "randomized_quantile", "working", "response"),
  ...
)
```

Arguments

`object`	a fit of type `glmGamPoi`. It is usually produced with a call to `glm_gp()`.
`Y`	any matrix-like object (e.g. `matrix()`, `DelayedArray()`, `HDF5Matrix()`) with one column per sample and row per gene.
`type`	the type of residual that is calculated. See details for more information. Default: `"deviance"`.
`...`	currently ignored.

Details

This method can calculate a range of different residuals:

deviance The deviance for the Gamma-Poisson model is

$$dev = 2*(1/theta*log((1+m*theta)/(1+y*theta)) - ylog((m+y*theta)/(y+y*m*theta)))$$

and the residual accordingly is

$$res = sign(y-m)sqrt(dev).$$

pearson The Pearson residual is $res = (y-m)/sqrt(m + m^2 * theta)$

randomized_quantile The randomized quantile residual was originally developed by Dunn & Smyth, 1995. Please see that publication or `statmod::qresiduals()` for more information.

working The working residuals are $res = (y-m)/m$.

response The response residuals are $res = y - m$

Value

a matrix with the same size as `Y`. If `Y` is a `DelayedArray` than the result will be as well.

See Also

`glm_gp()` and ‘stats::residuals.glm()

Index