Package 'baldur'

May 7, 2026

Title Bayesian Hierarchical Modeling for Label-Free Proteomics

Version 0.0.4

Description Statistical decision in proteomics data using a hierarchical Bayesian model. There are two regression models for describing the mean-variance trend, a gamma regression or a latent gamma mixture regression. The regression model is then used as an Empirical Bayes estimator for the prior on the variance in a peptide. Further, it assumes that each measurement has an uncertainty (increased variance) associated with it that is also inferred. Finally, it tries to estimate the posterior distribution (by Hamiltonian Monte Carlo) for the differences in means for each peptide in the data. Once the posterior is inferred, it integrates the tails to estimate the probability of error from which a statistical decision can be made.
See Berg and Popescu for details (<doi:10.1016/j.mcpro.2023.100658>).

License MIT + file LICENSE

URL https://github.com/PhilipBerg/baldur

Encoding UTF-8

Language en-US

RoxygenNote 7.3.2

Biarch true

Depends R (>= 4.2.0)

Imports dplyr (>= 1.0.9), magrittr (>= 2.0.3), methods, purrr (>= 0.3.4), Rcpp (>= 0.12.0), RcppParallel (>= 5.0.1), rstan (>= 2.26.0), rstantools (>= 2.2.0), stats, stringr (>= 1.0.4), tidyr (>= 1.2.0), rlang (>= 1.0.2), Rdpack (>= 2.4), multidplyr (>= 0.1.1), ggplot2 (>= 3.3.6), tibble (>= 3.1.7), viridisLite (>= 0.4.1), lifecycle

LinkingTo BH (>= 1.66.0), Rcpp (>= 0.12.0), RcppEigen (>= 0.3.3.3.0), RcppParallel (>= 5.0.1), rstan (>= 2.26.0), StanHeaders (>= 2.26.0)

SystemRequirements GNU make

Suggests knitr, rmarkdown

VignetteBuilder knitr

LazyData true

RdMacros Rdpack

BugReports https://github.com/PhilipBerg/baldur/issues

NeedsCompilation yes

Author Philip Berg [aut, cre] (ORCID: <https://orcid.org/0000-0002-3772-6185>)

Maintainer Philip Berg <pberg@live.se>

Repository CRAN

Date/Publication 2025-11-09 10:00:18 UTC

Contents

`baldur-package` *The 'baldur' package.*

Description

`baldur` is a Bayesian hierarchical model for statistical decision in proteomics data. It models the mean-variance trend with the option of two different regression models, a gamma regression or a latent gamma mixture regression. It then the regression model as en Empirical Bayes estimator for the prior on the variance. Further, it assumes that each measurement has an uncertainty (increased variance) associated with it that it also infers. Finally, it tries to estimate the posterior distribution (by Hamiltonian Monte Carlo) for the differences in means for each peptide in the data. Once the posterior is inferred, it integrates the tails to estimate the probability of error from which a statistical decision can be made.

Author(s)

Maintainer: Philip Berg <pberg@live.se> (ORCID)

References

Berg George Popescu (2023) "Baldur: Bayesian Hierarchical Modeling for Label-Free Proteomics with Gamma Regressing Mean-Variance Trends" Molecular & Cellular Proteomics: 2023-12. https://doi.org/10.1016/j.mcpro

Stan Development Team (2022). RStan: the R interface to Stan. R package version 2.21.5. https://mc-stan.org

See Also

Useful links:

- https://github.com/PhilipBerg/baldur
- Report bugs at https://github.com/PhilipBerg/baldur/issues

`calculate_mean_sd_trends` *Calculate the Mean-Variance trend*

Description

[Experimental]

Calculates the mean and standard deviation of each row (peptide) and adds them as new columns. Assumes that the condition names are the names in the design matrix.

Usage

```
calculate_mean_sd_trends(data, design_matrix, auxiliary_columns = c())
```

Arguments

`data`	A `tibble` or `data.frame` to annotate with mean and sd
`design_matrix`	A design matrix for the data (see example).
`auxiliary_columns`	Names of columns in the design matrix that does not have corresponding data in the data set. For example, this can be co-founding variables such as bashes.

Value

A `tibble` or `data.frame` with the mean and sd vectors

Examples

```
# Setup model matrix
design <- model.matrix(~ 0 + factor(rep(1:2, each = 3)))
colnames(design) <- paste0("ng", c(50, 100))

yeast %>%
    # Normalize data
    psrn("identifier") %>%
    # Get mean-variance trends
    calculate_mean_sd_trends(design)
```

`empirical_bayes` *Baldur's empirical Bayes Prior For The Mean In Conditions*

Description

Here we assume that the sample mean of each condition is an estimator for the center of the mean prior. In addition, it assumes that the confidence in the prior is proportional to the variance of the peptide.

$$\boldsymbol{\mu}_0 \sim \mathcal{N}(\bar{\boldsymbol{y}}, \sigma \boldsymbol{n}_R)$$
$$\boldsymbol{n}_R = [\frac{1}{\sqrt{n_1}}, \frac{1}{\sqrt{n_2}}, \ldots, \frac{1}{\sqrt{n_C}}]$$

Value

A `stanmodel` that can be used in infer_data_and_decision_model.

Code

The Stan code for this model is given by:

```
empirical_bayes
S4 class stanmodel 'empirical_bayes' coded as follows:
data {
  int<lower=0> N;     // number of data items
  int<lower=0> K;     // number of conditions
  int C;              // number of comparisons to perform
  matrix[N, K] x;     // design matrix
  vector[N] y;        // data
  matrix[K, C] c;     // contrast matrix
  real alpha;         // alpha prior for gamma
  real beta;          // beta prior for gamma
  vector[N] u;        // uncertainty
  vector[K] mu_not;   // prior mu
}
transformed data{
  vector[K] n_k;      // per condition reciprocal measurements
```

```
  row_vector[C] n_c;  // per comparison measurements
  matrix[K, C] abs_c; // abs of C for n_c calculation
  for (i in 1:K) {
    for (j in 1:C) {
      abs_c[i, j] = abs(c[i, j]);
    }
  }
  for (i in 1:K) {
    n_k[i] = 1/sum(x[,i]);
  }
  n_c = n_k' * abs_c;
  n_c = sqrt(n_c);
  n_k = sqrt(2*n_k);
}
parameters {
  vector[K] mu;           // mean vector
  real<lower=0> sigma;    // error scale
  array[C] real y_diff;   // difference in coefficients
  vector[K] eta;          // Error in mean
  vector[K] prior_mu_not; // Estimation error
}
transformed parameters{
  row_vector[C] mu_diff = mu' * c;      // differences in means
  vector[K] sigma_mu_not = sigma * n_k; // variance of ybars
  vector[C] sigma_lfc = sigma * n_c';   // variance of y_diff
}
model {
  sigma        ~ gamma(alpha, beta);                      // variance
  eta         ~ normal(0, 1);                           // NCP auxilary variable
  prior_mu_not ~ normal(mu_not, sigma_mu_not);             // Prior mean
  mu           ~ normal(prior_mu_not + sigma * eta, sigma); // mean
  y            ~ normal(x * mu, sigma * u);                // data model
  y_diff      ~ normal(mu_diff, sigma_lfc);              // difference statistic
}
```

`estimate_gamma_hyperparameters` *Estimate Gamma hyperparameters for sigma*

Description

[Experimental]

Estimates the hyperparameters for the Bayesian data and decision model. `estimate_gamma_hyperparameters` is a wrapper that adds new columns to the data (one for alpha and one for betas). Note that for `lgmr` objects, the `estimate_beta` function assumes that the data is ordered as when the model was fitted.

If this is not the case, theta's will be incorrectly matched with peptides—resulting in wrong estimates of beta parameters. On the other hand, `estimate_gamma_hyperparameters` will temporarily sort the data as when fitted and the sort it back as it was input to the function.

Usage

```
estimate_gamma_hyperparameters(reg, data, ...)

## S3 method for class 'glm'
estimate_gamma_hyperparameters(reg, data, ...)

## S3 method for class 'lgmr'
estimate_gamma_hyperparameters(reg, data, id_col, ...)

estimate_beta(reg, mean, ...)

## S3 method for class 'glm'
estimate_beta(reg, mean, alpha, ...)

## S3 method for class 'lgmr'
estimate_beta(reg, mean, m, s, ...)
```

Arguments

`reg`	A `glm` Gamma regression or a `lgmr` object
`data`	A `tibble` or `data.frame` to add gamma priors to
`...`	Currently not in use
`id_col`	A character for the name of the column containing the name of the features in data (e.g., peptides, proteins, etc.)
`mean`	The mean value of the peptide
`alpha`	The alpha parameter of the peptide
`m`	The mean of the means
`s`	The sd of the means

Value

`estimate_gamma_hyperparameters` returns a `tibble` or `data.frame` with the alpha,beta hyperparameters estimates as new columns.

`estimate_beta` returns estimates of the beta parameter(s)

Examples

```
# Setup model matrix
design <- model.matrix(~ 0 + factor(rep(1:2, each = 3)))
colnames(design) <- paste0("ng", c(50, 100))

# Normalize data
```

```
yeast_norm <- yeast %>%
    psrn("identifier") %>%
    # Get mean-variance trends
    calculate_mean_sd_trends(design)

# Fit gamma regression (could also have been a lgmr model)
gam_reg <- fit_gamma_regression(yeast_norm, sd ~ mean)

# Estimate priors
gam_reg %>%
    estimate_gamma_hyperparameters(yeast_norm)

# Can also explicitly estimate the beta parameters
# Note this is order sensitive.
estimate_beta(gam_reg, yeast_norm$mean, 1/summary(gam_reg)$dispersion)
```

`estimate_uncertainty` *Estimate measurement uncertainty*

Description

[Experimental]

Estimates the measurement uncertainty for each data point using a Gamma regression. Calculated as the expected standard deviation for each measurement:

$$\mathrm{E}[s_i|\omega, y_{ij}] = \exp(f(y_{ij}, \omega))$$

where ω are the regression parameters and f is a function describing the mean relationship between s_i and y_{ij}.

Usage

```
estimate_uncertainty(reg, data, id_col, design_matrix)

## S3 method for class 'glm'
estimate_uncertainty(reg, data, id_col, design_matrix)

## S3 method for class 'lgmr'
estimate_uncertainty(reg, data, id_col, design_matrix)
```

Arguments

`reg`	A `glm` gamma regression or `lgmr` object
`data`	A `tibble` or `data.frame`
`id_col`	A character for the name of the column containing the name of the features in data (e.g., peptides, proteins, etc.)
`design_matrix`	Cell mean design matrix for the data

Value

A matrix with the uncertainty

Examples

```
# Setup model matrix
design <- model.matrix(~ 0 + factor(rep(1:2, each = 3)))
colnames(design) <- paste0("ng", c(50, 100))

yeast_norm <- yeast %>%
  # Remove missing data
  tidyr::drop_na() %>%
  # Normalize data
  psrn("identifier") %>%
  # Add mean-variance trends
  calculate_mean_sd_trends(design)
# Fit the gamma regression
gam <- fit_gamma_regression(yeast_norm, sd ~ mean)
# Estimate each data point's uncertainty
estimate_uncertainty(gam, yeast_norm, 'identifier', design)
```

---

fit_gamma_regression *Function for Fitting the Mean-Variance Gamma Regression Models*

Description

[Experimental]

`fit_gamma_regression` returns a `glm` object containing the gamma regression for the mean-variance trend.

Usage

```
fit_gamma_regression(data, formula = sd ~ mean, ...)
```

Arguments

`data`	a `data.frame` to generate the mean-variance trends for. It should contain columns with conditions named as the column names in `design` (presumably with some suffix).
`formula`	a formula describing the model
`...`	only for compatibility with other functions

Value

`fit_gamma_regression` returns a glm object

Examples

```
# Generate a design matrix for the data
design <- model.matrix(~ 0 + factor(rep(1:2, each = 3)))

# Set correct colnames, this is important for calculate_mean_sd_trends
colnames(design) <- paste0("ng", c(50, 100))

# Normalize and log-transform the data
yeast_norm <- psrn(yeast, "identifier") %>%
  # Add row means and variances
  calculate_mean_sd_trends(design)

# Fit gamma regression model for the mean-variance trends
gamma_model <- fit_gamma_regression(yeast_norm, sd ~ mean)
```

---

fit_lgmr *Fit Latent Gamma Mixture Regression*

Description

[Experimental]

See lgmr_model for model details.

Usage

```
fit_lgmr(
  data,
  id_col,
  model = lgmr_model,
  iter = 6000,
  warmup = 1500,
  chains = 5,
  cores = 1,
  return_stanfit = FALSE,
  simplify = FALSE,
  ...
)

## S3 method for class 'lgmr'
print(
  x,
  simplify = x$simplify,
  pars = c("auxiliary", "coefficients"),
  digits = 3,
  ...
)
```

```
## S3 method for class 'lgmr'
coef(object, simplify = FALSE, pars = c("coefficients", "auxiliary"), ...)
```

Arguments

data	A data.frame with mean-variance trends to use in the fitting. The columns need to have the following hard-coded names: mean and sd.
id_col	A character for the name of the column containing the name of the features in data (e.g., peptides, proteins, etc.). Has to be a unique identifier for each feature.
model	Defaults to lgmr_model (see it for details on the model), can also be an user supplied stan_model
iter	Total number of samples to draw
warmup	Number of warm-up samples to draw
chains	Number of chains to run
cores	Number of cores to use per chain
return_stanfit	Should the stanfit object be returned with the model?
simplify	Should only the mean estimates of the posterior be returned?
...	Additional arguments to rstan's sampling. Does nothing for print or coef only for fit_lgmr.
x, object	An lgmr model.
pars	If you want to print/extract the regression coefficients, theta, auxiliary (alpha and NRMSE), or all
digits	Number of digits to print

Value

A fitted lgmr model.

Examples

```
# Define design matrix
design <- model.matrix(~ 0 + factor(rep(1:2, each = 3)))
colnames(design) <- paste0("ng", c(50, 100))


# Normalize data, calculate M-V trend, and fit LGMR model
yeast_lgmr <- yeast %>%
    # Remove missing values
    tidyr::drop_na() %>%
    # Normalize
    psrn("identifier") %>%
    # Add the mean-variance trends
    calculate_mean_sd_trends(design) %>%
    # Fit the model
    fit_lgmr("identifier")
# Print everything except thetas
print(yeast_lgmr, pars = c("coefficients", "auxiliary"))
```

```
# Extract the mean of the model parameters posterior
yeast_lgmr_pars <- coef(yeast_lgmr, pars = 'all', simplify = TRUE)
```

`infer_data_and_decision_model` *Sample the Posterior of the data and decision model and generate point estimates*

Description

[Experimental]

Function to sample the posterior of the Bayesian data and decision model. It first produces the needed inputs for Stan's sampling for each peptide (or protein, PTM, etc.). It then runs the sampling for the data and decision model. From the posterior, it then collects estimates and sampling statistics from the posterior of data model and integrates the decision distribution D. It then returns a tibble with all the information for each peptide's posterior (see **Value** below). There are major time gains to be made by running this procedure in parallel. `infer_data_and_decision_model` has an efficient wrapper around `multidplyr`. This will let you to evenly distribute all peptides evenly to each worker. E.g., two workers will each run half of the peptides in parallel.

Usage

```
infer_data_and_decision_model(
  data,
  id_col,
  design_matrix,
  contrast_matrix,
  uncertainty_matrix,
  stan_model = empirical_bayes,
  clusters = 1,
  h_not = 0,
  auxiliary_columns = c(),
  ...
)
```

Arguments

`data`	A `tibble` or `data.frame` with alpha,beta priors annotated
`id_col`	A character for the name of the column containing the name of the features in data (e.g., peptides, proteins, etc.)
`design_matrix`	A design matrix for the data. For the empirical_bayes prior only mean models are allowed (see example). For the weakly_informative prior more complicated design can be used.

`contrast_matrix`
: A contrast matrix of the decisions to test. Columns should sum to `0` and only mean comparisons are allowed. That is, the absolute value of the positive and negative values in each column has to sum to 2. E.g., a column can be `[0.5, 0.5, -1]`^{T} but not `[1, 1, -1]`^{T} or `[0.5, 0.5, -2]`^{T}. That is, `sum(abs(x))=2` where `x` is a column in the contrast matrix.

`uncertainty_matrix`
: A matrix of observation specific uncertainties

`stan_model`
: Which Bayesian model to use. Defaults to empirical_bayes but also allows weakly_informative, or an user supplied function.

`clusters`
: The number of parallel threads/workers to run on.

`h_not`
: The value of the null hypothesis for the difference in means

`auxiliary_columns`
: Names of columns in the design matrix that does not have corresponding data in the data set. For example, this can be co-founding variables such as bashes.

`...`
: Additional arguments passed to sampling. Note that verbose will always be forced to `FALSE` to avoid console flooding.

Details

The data model of Baldur assumes that the observations of a peptide, $\boldsymbol{Y}$, is a normally distributed with a standard deviation, σ, common to all measurements. In addition, it assumes that each measurement has a unique uncertainty u. It then models all measurements in the same condition with a common mean μ. It then assumes that the common variation of the peptide is caused by the variation in the μ As such, it models μ with the common variance σ and a non-centered parametrization for condition level effects.

$$\boldsymbol{Y} \sim \mathcal{N}(\boldsymbol{X}\boldsymbol{\mu}, \sigma\boldsymbol{u}) \quad \boldsymbol{\mu} \sim \mathcal{N}(\mu_0 + \boldsymbol{\eta}\sigma, \sigma)$$

It then assumes σ to be gamma distributed with hyperparameters infered from either a gamma regression fit_gamma_regression or a latent gamma mixture regression fit_lgmr.

$$\sigma \sim \Gamma(\alpha, \beta)$$

For details on the two priors for μ_0 see empirical_bayes or weakly_informative. Baldur then builds a posterior distribution of the difference(s) in means for contrasts of interest. In addition, Baldur increases the precision of the decision as the number of measurements increase. This is done by weighting the sample size with the contrast matrix. To this end, Baldur limits the possible contrast of interest such that each column must sum to zero, and the absolute value of each column must sum to two. That is, only mean comparisons are allowed.

$$\boldsymbol{D} \sim \mathcal{N}(\boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{K}, \sigma\boldsymbol{\xi}), \quad \xi_i = \sqrt{\sum_{c=1}^{C} |k_{cm}| n_c^{-1}}$$

where $\boldsymbol{K}$ is a contrast matrix of interest and k_{cm} is the contrast of the c:th condition in the m:th contrast of interest, and n_c is the number of measurements in the c:th condition. Baldur then integrates the tails of $\boldsymbol{D}$ to determine the probability of error.

$$P(\textbf{error}) = 2\Phi(-|\boldsymbol{\mu_D} - H_0| \odot \boldsymbol{\tau_D})$$

where H_0 is the null hypothesis for the difference in means, Φ is the CDF of the standard normal, $\boldsymbol{\mu_D}$ is the posterior mean of $\boldsymbol{D}$, $\boldsymbol{\tau_D}$ is the posterior precision of $\boldsymbol{D}$, and $\odot$ is the Hadamard product.

Value

A tibble or data.frame() annotated with statistics of the posterior and p(error). err is the probability of error, i.e., the two tail-density supporting the null-hypothesis, lfc is the estimated $\log_2$-fold change, sigma is the common variance, and lp is the log-posterior. Columns without suffix shows the mean estimate from the posterior, while the suffixes _025, _50, and _975, are the 2.5, 50.0, and 97.5, percentiles, respectively. The suffixes _eff and _rhat are the diagnostic variables returned by Stan. In general, a larger _eff indicates effective sample size and _rhat indicates the mixing within chains and between the chains and should be smaller than 1.05 (please see the Stan manual for more details). In addition it returns a column warnings with potential warnings given by the sampler.

Examples

```
# (Please see the vignette for a tutorial)
# Setup model matrix
design <- model.matrix(~ 0 + factor(rep(1:2, each = 3)))
colnames(design) <- paste0("ng", c(50, 100))

yeast_norm <- yeast %>%
# Remove missing data
  tidyr::drop_na() %>%
  # Normalize data
  psrn('identifier') %>%
  # Add mean-variance trends
  calculate_mean_sd_trends(design)

# Fit the gamma regression
gam <- fit_gamma_regression(yeast_norm, sd ~ mean)

# Estimate each data point's uncertainty
unc <- estimate_uncertainty(gam, yeast_norm, 'identifier', design)

yeast_norm <- gam %>%
   # Add hyper-priors for sigma
   estimate_gamma_hyperparameters(yeast_norm)
# Setup contrast matrix
contrast <- matrix(c(-1, 1), 2)

yeast_norm %>%
  head() %>% # Just running a few for the example
  infer_data_and_decision_model(
    'identifier',
    design,
    contrast,
    unc,
    clusters = 1 # I highly recommend increasing the number of parallel workers/clusters
                 # this will greatly reduce the time of running baldur
  )
```

`lgmr_model` *Latent Gamma Regression Model*

Description

Baldur has the option to use the Latent Gamma Regression Model (LGMR). This model attempts to estimate local Mean-Variance (M-V) trend for each peptide (or protein, PTM, etc.). To this end is describes the M-V trend as a regression with a common link function and a latent one. While there may not be a latent trend in real data, it allows for a mathematical description that can estimate the local trends of each peptide. The model describes the standard deviation (S) as a gamma random variable with mean dependency described by the sample mean ($\bar{y}$):

$$\boldsymbol{S} \sim \Gamma(\alpha, \beta), \quad \boldsymbol{\beta} = \frac{\alpha}{\boldsymbol{\mu}}$$

$$\boldsymbol{\mu} = \exp(\gamma_0 - \gamma_{\bar{y}}\, f(\bar{\boldsymbol{y}})) + \kappa \exp(\boldsymbol{\theta}(\gamma_{0L} - \gamma_{\bar{y}L}\, f(\bar{\boldsymbol{y}})), \quad f(\boldsymbol{x}) = \frac{\boldsymbol{x} - \mu_{\bar{y}}}{\sigma_{\bar{y}}}$$

Here, $\exp(\gamma_0 - \gamma_{\bar{y}} f(\bar{\boldsymbol{y}}))$ is the common trend of the regression. Then, the mixing variable, $\boldsymbol{\theta}$, describes the proportion of the mixture that each peptide has, and κ is just some small constant such that when θ is zero the latent term is small.

Details

Next we will describe the hierarchical prior setup for the regression variables. For α `baldur` uses a half-Cauchy as a weakly informative prior:

$$\alpha \sim \text{Half-Cauchy}(25)$$

For the latent contribution to the i:th peptide, θ_i, has an uniform distribution:

$$\theta_i \sim \mathcal{U}(0, 1)$$

Further, it can be seen that Baldur assumes that S always decreases (on average) with the sample mean. To this end, both slopes ($\gamma_{\bar{y}}, \gamma_{\bar{y}L}$) are defined on the real positive line. Hence, we used Half-Normal (HN) distributions to describe the slope parameters:

$$\gamma_{\bar{y}} \sim HN(1)$$

$$\gamma_{\bar{y}L} \sim HN(1)$$

For the intercepts, we assume a standard normal prior for the common intercept. Then, we use a skew-normal to model the latent intercept. The reason for this is two-fold, one, κ will decrease the value of the latent term and, two, to push the latent trend upwards. The latter reason is so that the latent intercept is larger than the common and so that the priors prioritize a shift in intercept over a increase in slope. For the intercepts, Baldur uses a standard normal prior for the common intercept.

$$\gamma_0 \sim \mathcal{N}(0, 1)$$

While for the latent trend, it uses a skew-normal (SN) to push up the second trend and to counteract the shrinkage of κ.

$$\gamma_{0L} \sim \mathcal{SN}(2, 15, 35)$$

Value

A `stanmodel` that can be used in fit_lgmr.

Code

The `Stan` code for this model is given by:

```
lgmr_model
S4 class stanmodel 'lgmr_model' coded as follows:
functions {
  // mu
  vector reg_function(
    vector x,
    vector theta,
    real I,
    real I_L,
    real S,
    real S_L,
    int N
    ) {
    vector[N] exp_beta  = .001*exp(theta .* (I_L - S_L*x));
              exp_beta +=      exp(I - S*x);
    return exp_beta;
  }
}
data {
  int<lower=0> N;       // Number of observations
  vector<lower=0>[N] y; // sd
  vector[N] x;          // mean
}
transformed data {
  real v_y = variance(y);                      // for NRMSE
  vector[N] x_star = (x - mean(x))/sd(x);      // f(y_bar)
  vector[3] lb = [0, 0, negative_infinity()]'; // Boundaries normal coefs.
}
parameters {
  real<lower = 0> alpha;                 // Shape
  vector<lower = lb>[3] eta;             // For S, S_L, I
  vector<lower = 0, upper = 1>[N] theta; // Mixture parameter
  real I_L;                              // Intercept Latent
}
transformed parameters {
  real<lower = 0> S   = eta[1]; // Slope common
  real<lower = 0> S_L = eta[2]; // Slope Latent
  real I              = eta[3]; // Intercep common
}
model {
  //Priors
```

```
  alpha         ~ cauchy(0, 25);
  eta           ~ std_normal();
  I_L           ~ skew_normal(2, 15,  35);
  theta         ~ beta(1, 1);
  {
    vector[N] exp_beta = reg_function(x_star, theta, I, I_L, S, S_L, N);
    // Likelihood
    y ~ gamma(alpha, alpha ./ exp_beta);
  }
}
generated quantities {
  // NRMSE calculations
  real nrmse;
  {
    vector[N] se = reg_function(x_star, theta, I, I_L, S, S_L, N);
    se -= y;
    se  = square(se);
    nrmse = mean(se) / v_y;
  }
  nrmse = sqrt(nrmse);
}
```

`lgmr_plotting` *Visualization of LGMR models*

Description

[Experimental] Options to plot the LGMR model. `plot_lgmr_regression` will plot the data colored by the amount of latent trend they have as well as the two extreme regression cases when θ is zero or one. `plot_regression_field` will plot the local regression trend for each data point as a vector field and color the vector based on the derivative at the mean of the peptide.

Usage

```
plot_lgmr_regression(model)

plot_regression_field(model, n = 10, rng = 10)
```

Arguments

`model`	An LGMR model object
`n`	Number of interpolation points for each peptides regression
`rng`	The proportional range of each peptides regression. E.g., a value of 10 will make each line span 1 % of the x-axis.

Value

A ggplot object

Examples

```
#' # Define design matrix
design <- model.matrix(~ 0 + factor(rep(1:2, each = 3)))
colnames(design) <- paste0("ng", c(50, 100))


# Normalize data, calculate M-V trend, and fit LGMR model
yeast_lgmr <- yeast %>%
    # Remove missing values
    tidyr::drop_na() %>%
    # Normalize
    psrn("identifier") %>%
    # Add the mean-variance trends
    calculate_mean_sd_trends(design) %>%
    # Fit the model
    fit_lgmr("identifier")
# Print everything except thetas
print(yeast_lgmr, pars = c("coefficients", "auxiliary"))
# Extract the mean of the model parameters posterior
yeast_lgmr_pars <- coef(yeast_lgmr, pars = 'all', simplify = TRUE)
plot_lgmr_regression(yeast_lgmr)
plot_regression_field(yeast_lgmr)
```

---

plot_gamma *Function for plotting the gamma regression for the mean-variance trend*

Description

Generates a scatter plot with the gamma regressions of the mean-variance trends without partitioning

Usage

```
plot_gamma(data)
```

Arguments

data	The data to use for producing the plots.

Value

a plot with the estimated mean-variance trend

Examples

```
# Produce a design matrix
design <- model.matrix(~ 0 + factor(rep(1:2, each = 3)))
colnames(design) <- paste0("ng", c(50, 100))

# Normalize and log transform the data
yeast_norm <- psrn(yeast, "identifier")

# Generate the plot
yeast_norm %>%
  calculate_mean_sd_trends(design) %>%
  plot_gamma()
```

---

plot_gamma_regression *Function for plotting the mean-variance gamma regressions*

Description

[Experimental]

Generates a scatter plot with the gamma regressions of the mean-variance trend. Can either be ran directly with `plot_gamma_regression` and inputing a design matrix, or with `plot_gamma` if the M-V trend has been added to the data with `calculate_mean_sd_trends()`.

Usage

```
plot_gamma_regression(data, design)
```

Arguments

`data`	The data to use for producing the plots.
`design`	A design matrix as produced by `model.matrix`.

Value

a plot with the mean-variance trend before partitioning on the left side, and the right side after.

Examples

```
# Produce a design matrix
design <- model.matrix(~ 0 + factor(rep(1:2, each = 3)))
colnames(design) <- paste0("ng", c(50, 100))

# Normalize and log transform the data
yeast %>%
    # Remove missing data
    # Note that this could be replaced with imputation
    tidyr::drop_na() %>%
    # Normalize
    psrn("identifier") %>%
    plot_gamma_regression(design)
```

plot_sa *Plot the trend between the log fold-change and sigma, coloring significant hits*

Description

[Experimental]

plot_sa returns a ggplot with a graphical representation between the log fold-change and sigma.

Usage

```
plot_sa(results, alpha = 0.05, lfc = NULL)
```

Arguments

results	Output generated by baldur::infer_data_and_decision_model
alpha	Significance cut-off; used to draw a line indicating where significance starts
lfc	LFC cut-off; used to draw lines for abs(lfc), if NULL no lines are drawn

Value

plot_sa returns a ggplot object

Examples

```
# Setup model matrix
design <- model.matrix(~ 0 + factor(rep(1:2, each = 3)))
colnames(design) <- paste0("ng", c(50, 100))

yeast_norm <- yeast %>%
# Remove missing data
  tidyr::drop_na() %>%
  # Normalize data
  psrn('identifier') %>%
  # Add mean-variance trends
  calculate_mean_sd_trends(design)
# Fit the gamma regression
gam <- fit_gamma_regression(yeast_norm, sd ~ mean)
# Estimate each data point's uncertainty
unc <- estimate_uncertainty(gam, yeast_norm, "identifier", design)
yeast_norm <- gam %>%
   # Add hyper-priors for sigma
   estimate_gamma_hyperparameters(yeast_norm)
# Setup contrast matrix
contrast <- matrix(c(-1, 1), 2)

results <- yeast_norm %>%
  head() %>% # Just run a few for the example
```

```
infer_data_and_decision_model(
  'identifier',
  design,
  contrast,
  unc,
  clusters = 1 # I highly recommend increasing the number of parallel workers/clusters
               # this will greatly reduce the speed of running baldur
)
# Plot with alpha = 0.05
plot_sa(results, alpha = 0.05)
# Plot with alpha = 0.01 and show LFC = 1
plot_sa(results, alpha = 0.01, 1)
```

`plot_volcano` *Plot the -log10(err) against the log fold-change*

Description

[Experimental]

`plot_volcano` returns a `ggplot` with a graphical representation between the -log10(err) and LFC.

Usage

```
plot_volcano(results, alpha = 0.05, lfc = NULL)
```

Arguments

`results`	Output generated by `baldur::infer_data_and_decision_model`
`alpha`	Significance cut-off; used to draw a line indicating where significance starts
`lfc`	LFC cut-off; used to draw lines for `abs(lfc)`, if `NULL` no lines are drawn

Value

`plot_volcano` returns a `ggplot` object

Examples

```
# Setup model matrix
design <- model.matrix(~ 0 + factor(rep(1:2, each = 3)))
colnames(design) <- paste0("ng", c(50, 100))

yeast_norm <- yeast %>%
# Remove missing data
  tidyr::drop_na() %>%
  # Normalize data
  psrn('identifier') %>%
  # Add mean-variance trends
  calculate_mean_sd_trends(design)
```

```
# Fit the gamma regression
gam <- fit_gamma_regression(yeast_norm, sd ~ mean)
# Estimate each data point's uncertainty
unc <- estimate_uncertainty(gam, yeast_norm, 'identifier', design)
yeast_norm <- gam %>%
   # Add hyper-priors for sigma
   estimate_gamma_hyperparameters(yeast_norm)
# Setup contrast matrix
contrast <- matrix(c(-1, 1), 2)

results <- yeast_norm %>%
  head() %>% # Just run a few for the example
  infer_data_and_decision_model(
    'identifier',
    design,
    contrast,
    unc,
    clusters = 1 # I highly recommend increasing the number of parallel workers/clusters
                 # this will greatly reduce the speed of running baldur
  )
  # Plot indicating where alpha = 0.05
  plot_volcano(results, alpha = 0.05)
  # Plot indicating where alpha = 0.01 and where LFC = 1
  plot_volcano(results, alpha = 0.01, 1)
```

---

psrn *Normalize data to a pseudo-reference*

Description

[Experimental]

This function generates a pseudo-reference by taking the geometric mean of each peptide across all samples. Each peptide in each sample is then divided by the pseudo-reference. Then, the median ratio of all ratios is used as an estimate to use for normalizing differences in loading concentration. All features in each sample is then divided by their corresponding estimate. All estimates are based on features without missing values. For details see Anders and Huber (2010).

Usage

```
psrn(data, id_col, log = TRUE, load_info = FALSE, target = NULL)
```

Arguments

data	data.frame
id_col	a character for the name of the column containing the name of the features in data (e.g., peptides, proteins, etc.)
log	boolean variable indicating if the data should be log transformed after normalization

load_info	logical; should the load information be output?
target	target columns to normalize, supports tidyselect-package syntax. By default, all numerical columns will be used in the normalization if not specified.

Value

data frame with normalized values if load_info=FALSE, if it is TRUE then it returns a list with two tibbles. One tibble containing the normalized data and one containing the loading info as well as the estimated normalization factors.

Source

https://www.nature.com/articles/npre.2010.4282.1

References

Simon Anders, Wolfgang Huber (2010). "Differential expression analysis for sequence count data." *Nature Precedings*, 1–1.

Examples

```
yeast_psrn <- psrn(yeast, "identifier")
yeast_psrn_with_load <- psrn(yeast, "identifier", load_info = TRUE)
yeast_ng50_only <- psrn(yeast, "identifier", target = matches('ng50'))
```

---

ups *Spiked-in data set of peptides*

Description

A dataset containing quantification of peptides using Progenesis. True positives peptides spiked-in from the Universal Proteomics Standard Set 1 (UPS1) at three different concentrations and true negatives from *Chlamydomonas reinhardtii* with the same concentration in all samples. You can find true positives with stringr::str_detect(ups$identifier, 'UPS'). For details see Berg et al. (2019) and if you use this dataset please cite the same paper.

Usage

```
ups
```

Format

A data frame with 10599 rows and 13 variables:

identifier id column for features, true positives contains UPS and true negatives contains Cre

fmol25_1,fmol25_2,fmol25_3,fmol25_4 Technical replicates with true positives spiked-in from 25 fmol UPS1 peptides

fmol50_1,fmol50_2,fmol50_3,fmol50_4 Technical replicate with true positives spiked-in from 50 fmol UPS1 peptides

fmol100_1,fmol100_2,fmol100_3,fmol100_4 Technical replicate with true positives spiked-in from 100 fmol UPS1 peptides

Source

https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-019-2619-6

References

Philip Berg, Evan W McConnell, Leslie M Hicks, Sorina C Popescu, George V Popescu (2019). "Evaluation of linear models and missing value imputation for the analysis of peptide-centric proteomics." *BMC bioinformatics*, **20**(2), 7–16.

---

weakly_informative *Baldur's weakly informative prior for the mean in conditions*

Description

Here we will model the mean of the prior with a weakly informative (WI) prior. We will assume that, in essence, nothing is know about the mean. As such, for the WI prior, we use a normal prior on $\boldsymbol{\mu}_0$ centered at zero and with a very large variance.

$$\boldsymbol{\mu}_0 \sim \mathcal{N}(0, 100)$$

Value

A `stanmodel` that can be used in infer_data_and_decision_model.

Code

The `Stan` code for this model is given by:

```
weakly_informative
S4 class stanmodel 'weakly_informative' coded as follows:
data {
  int<lower=0> N;     // number of data items
  int<lower=0> K;     // number of conditions
  int C;              // number of comparisons to perform
  matrix[N, K] x;     // design matrix
  vector[N] y;        // data
  matrix[K, C] c;     // contrast matrix
  real alpha;         // alpha prior for gamma
  real beta;          // beta prior for gamma
  vector[N] u;        // uncertainty
}
```

```
transformed data{
  vector[K] n_k;      // per condition measurements
  row_vector[C] n_c;  // per comparison measurements
  matrix[K, C] abs_c; // abs of C for n_c calculation
  for (i in 1:K) {
    for (j in 1:C) {
      abs_c[i, j] = abs(c[i, j]);
    }
  }
  for (i in 1:K) {
    n_k[i] = 1/sum(x[,i]);
  }
  n_c = n_k' * abs_c;
  n_c = sqrt(n_c);
}
parameters {
  vector[K] mu;           // coefficients for predictors
  real<lower=0> sigma;    // error scale
  array[C] real y_diff;   // difference in coefficients
  vector[K] eta;          // Error in mean
  vector[K] prior_mu_not; // Estimation error
}
transformed parameters{
  row_vector[C] mu_diff = mu' * c;        // differences in means
  vector[C] sigma_lfc = sigma * n_c';     // variance of y_diff
}
model {
  sigma        ~ gamma(alpha, beta);                    // variance
  eta         ~ normal(0, 1);                          // NCP auxilary variable
  prior_mu_not ~ normal(0, 10);                          // prior mean
  mu           ~ normal(prior_mu_not + sigma*eta, sigma); // mean
  y            ~ normal(x * mu, sigma*u);                // data model
  y_diff      ~ normal(mu_diff, sigma_lfc);             // difference statistic
}
```

`yeast` *Spiked-in data set of reversibly oxidized cysteines*

Description

A dataset containing quantification of reversibly oxidized cysteines using Progenesis. True positives cysteines spiked-in from yeast at two different concentrations and true negatives from *Chlamydomonas reinhardtii* with the same concentration in all samples. To identify true positives one can use `stringr::str_detect(yeast$identifier, 'YEAST')`. For details see Berg et al. (2019) and if you use this dataset please cite the same paper.

Usage

```
yeast
```

Format

A data frame with 2235 rows and 7 variables:

identifier id column for features, true positives contains YEAST and true negatives contains Cre

ng50_1,ng50_2,ng50_3 Biological replicates with true positives spiked-in from 50 ng yeast cells

ng100_1,ng100_2,ng100_3 Biological replicates with true positives spiked-in from 100 ng yeast cells

Source

https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-019-2619-6

References

Philip Berg, Evan W McConnell, Leslie M Hicks, Sorina C Popescu, George V Popescu (2019). "Evaluation of linear models and missing value imputation for the analysis of peptide-centric proteomics." *BMC bioinformatics*, **20**(2), 7–16.

Index