Package 'csDEX'

June 18, 2017

Type Package

Title Condition-specific differential exon expression.

Version 1.0

Date 2017-01-22

Author Martin Strazar

Maintainer Who to complain to <yourfault@somewhere.net>

Description More about what it does (maybe more than one line)

License What license is it under?

Depends methods, DESeq2, edgeR, aod, utils, betareg, reshape, parallel

R topics documented:

`csDEX-package` *Condition-specific differential exon expression.*

Description

More about what it does (maybe more than one line)

Details

The DESCRIPTION file: This package was not yet installed at build time.

Index: This package was not yet installed at build time.
csDEX (Condition-specific Differential Exon Expression) is a family of general linear models (GLMs), used to model differential splicing given a set of sequencing experiments based on RNA-seq. The methods assume tens to hundreds experimental conditions to be compared, and the result are changes in usage of splice sites that are specific to an experimental condition.

Capabilities:

- handling of sequence data in form of read counts or percentages (e.g. percent-spliced in, PSI),
- modelling all types of splicing events,
- discovery and ranking of condition-specific changes,
- parallel processing of multiple genes (groups),
- time-efficient model approximation with no change in false positive rate.

Author(s)

Martin Strazar

Maintainer: Who to complain to <yourfault@somewhere.net>

`csDEXdataSet` *Object to store a csDEX dataset.*

Description

Initialize and instance of the csDEXdataSet object, by constructing a feature-by-condition data matrix with associated metadata.

Usage

```
csDEXdataSet(data.dir, design.file, type = "count", col.condition = "Experiment.
```

Arguments

`data.dir`	The directory containing the replicate (data) files.
`design.file`	The design file with one line per replicate. Needs to contain at least two columns, denoting file name (default: "File.accession") and condition name (default: "Experiment.target"). An optional column can be included to denote total read counts associated to a replicate (default: "input.read.count"). Also, an optional binary column (default: "testable") can be used to test only selected conditions. Other columns can be stored, but will not be used by csDEX.
`type`	Data type, either "PSI" (percent-spliced in) or "count" (read counts).
`col.condition`	Name of the column in the design file denoting the experimental condition.
`col.replicate`	Name of the column in the design file denoting unique file identifiers (without extension).
`col.read.count`	Column denotig original input read counts.
`col.additional`	Additional columns to import (for use in e.g. complex model designs).
`col.testable`	Optional column with binary (TRUE/FALSE) information for which conditions to test.
`data.file.ext`	Replicate files extension (default: "txt").
`aggregation`	Replicate aggregation function. A pointer to a function for merging values of replicates into one per each sample (default: mean).

`min.bin.count`
: Minimum value at a feature to be considered non-zero. If defined, values below this threshold will be set to zero.

`min.gene.count`
: Minimum sum of expression values accross a gene to be considered expressed. If defined, all the features associated to the gene will be set to zero.

`zero.out`
: Use for PSI data to prevent PSI being not equal to either 0 or 1 for constitutive features. If feature value is less than min.count, it is set to NA and ignored by the aggregation function.

Details

The main task of this function is to construct a csDEXdataSet object by aggregating the replicate data into one column per sample. Arbitrary aggregation functions can be passed, the most common ones being mean, sum and max. The options min.bin.count, min.gene.count and zero.out can be used for filtering.

All metadata provided by the design file is stored in the object colData. The two mandatory columns are specified by col.condition and col.replicate, as defined above. The total experiment read count can be provided along with each replicate, to enable usage of estiamteGeneCPM function in the downstream analysis.

The row metadata is parsed as featureID (individual regions of interest) and groupID (gene).

Value

An initialized csDEXdataSet object.

Author(s)

Martin Stražar.

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (data.dir, design.file, type = "count", col.condition = "Experiment.target",
    col.replicate = "File.accession", data.file.ext = "txt",
    aggregation = NULL, min.bin.count = NULL, min.gene.count = NULL,
    zero.out = NULL)
{
    if (!(type %in% c("count", "PSI")))
        stop("type must be one of ('count', 'PSI')")
    if (is.null(min.bin.count))
        min.bin.count = 0
    if (type == "PSI") {
        if (is.null(aggregation))
            aggregation = mean
        if (is.null(zero.out))
            zero.out = TRUE
        if (is.null(min.gene.count))
            min.gene.count = 1
    }
    else if (type == "count") {
```

```
    if (is.null(aggregation))
        aggregation = sum
    if (is.null(zero.out))
        zero.out = FALSE
    if (is.null(min.gene.count))
        min.gene.count = 0
}
zeroOut <- function(repData, min.count = 0) {
    genes = unlist(lapply(row.names(repData), function(x) strsplit(x,
        ":")[[1]][1]))
    for (g in unique(genes)) {
        inxs = genes == g
        zeros = which(apply(repData[inxs, ], 2, sum) < min.count)
        repData[inxs, zeros] = NA
    }
    return(repData)
}
design = read.csv(design.file, sep = "\t", header = TRUE)
stopifnot(col.condition %in% colnames(design))
stopifnot(col.replicate %in% colnames(design))
conditions = sort(unique(design[, col.condition]))
n.con = length(conditions)
message("Processing expression data")
exprData = NULL
lib.sizes = NULL
for (i in 1:length(conditions)) {
    cond = conditions[i]
    message(sprintf("Condition %s", cond))
    replicates = design[design[, col.condition] == cond,
        col.replicate]
    cond.lib.size = NULL
    if (!is.null(design$input.read.count)) {
        cond.lib.size = aggregation(design[design[, col.condition] ==
            cond, "input.read.count"])
        lib.sizes = c(lib.sizes, cond.lib.size)
    }
    repData = NULL
    n.rep = length(replicates)
    for (j in 1:length(replicates)) {
        rep = replicates[j]
        rep.path = file.path(data.dir, paste(rep, data.file.ext,
            sep = "."))
        y = read.table(rep.path, header = FALSE, comment.char = "_")
        y$V1 = as.character(y$V1)
        n.row = nrow(y)
        message(sprintf("     Replicate %s, num. rows: %d",
            rep.path, n.row))
        if (is.null(repData)) {
            repData = matrix(0, ncol = n.rep, nrow = n.row)
            row.names(repData) = y$V1
        }
        repData[y$V1, j] = y$V2
    }
    repData[repData < min.bin.count] = 0
    if (zero.out || min.gene.count > 0)
        repData = zeroOut(repData, min.gene.count)
    repVec = suppressWarnings(apply(repData, 1, aggregation,
```

```
            na.rm = TRUE))
        repVec[is.infinite(repVec)] = 0
        repVec[is.na(repVec)] = 0
        if (is.null(exprData)) {
            exprData = matrix(0, ncol = n.con, nrow = n.row)
            row.names(exprData) = row.names(repData)
            colnames(exprData) = conditions
        }
        exprData[, i] = repVec
    }
    rowData = data.frame(featureID = row.names(exprData), groupID = unlist(lapply(row.nam
        function(x) strsplit(x, ":")[[1]][1])), binID = unlist(lapply(row.names(repData),
        function(x) strsplit(x, ":")[[1]][2])))
    rowData$featureID = as.character(rowData$featureID)
    rowData$groupID = as.character(rowData$groupID)
    rowData$binID = as.character(rowData$binID)
    row.names(rowData) = rowData$featureID
    colData = data.frame(condition = colnames(exprData))
    if (!is.null(lib.sizes))
        colData$lib.size = lib.sizes
    new("csDEXdataSet", exprData = exprData, rowData = rowData,
        colData = colData, dataType = type)
}
```

`csDEXdataSet-class` *Class* `"csDEXdataSet"`

Description

A data structure with the features-by-condition matrix, storing the expression values for percentage-spliced in or read counts, along with metadata for both rows and columns. Optionally, gene expression is stored as counts-per-million reads (CPM).

Objects from the Class

Objects can be created by calls of the form `new("csDEXdataSet", ...)`.

Slots

`dataType`**:** Object of class `"character"` Either "PSI" (percentage-spliced in) or "count" (read counts), determining the underlying data distribution.

`exprData`**:** Object of class `"matrix"` Feature-by-condition matrix containing expression data.

`rowData`**:** Object of class `"data.frame"` Row (feature) metadata.

`colData`**:** Object of class `"data.frame"` Column (condition) metadata, the experiment design.

`cpmData`**:** Object of class `"matrix"` Optional. Counts-per-million reads (CPM) associated to a group (gene).

Methods

dataType `signature(obj = "csDEXdataSet")`: The accessor (getter) of the `csDEXdataSet` object data type (PSI or count).

colData `signature(obj = "csDEXdataSet")`: The accessor (getter) of column (condition) metadata provided by the experiment design file.

colData<- `signature(obj = "csDEXdataSet")`: The accessor (setter) of column (condition) metadata provided by the experiment design file.

cpmData `signature(obj = "csDEXdataSet")`: The accessor (getter) of the counts per million reads (CPM) associated to each gene (`groupID`). Computable if the `input.read.count` field is set and provided in the experiment design file.

cpmData<- `signature(obj = "csDEXdataSet")`: The accessor (setter) of the counts per million reads (CPM) associated to each gene (`groupID`).

exprData `signature(obj = "csDEXdataSet")`: The accessor (getter) of the expression data matrix constructed by merging the experimental data in samples.

exprData<- `signature(obj = "csDEXdataSet")`: The accessor (setter) of the expression data matrix constructed by merging the experimental data in samples.

rowData `signature(obj = "csDEXdataSet")`: The accessor (getter) of row (feature) metadata provided by the experiment data files.

rowData<- `signature(obj = "csDEXdataSet")`: The accessor (setter) of row (feature) metadata provided by the experiment data files.

estimatePrecisions `signature(obj = "csDEXdataSet")`: Estimate the precisions hyperparameter for each feature by using the external edgeR::estimateDisp function. Required for fitting read-count based models. The function estimateSizeFactors should be called on the cdx object prior to estimatePrecisions call.

estimateGeneCPM `signature(obj = "csDEXdataSet")`: Estimate counts per million reads (CPM) associated to each gene (groupID). An integer field named input.read.count is required in the design file, specifying the sequencing depth measured by the number of sequenced reads / read pairs. Can be used by both PSI and count models to skip testing for low-expressed genes and avoid Type I errors that are dut to changes in gene expression (and not differential splicing).

estimateSizeFactors `signature(obj = "csDEXdataSet")`: Estimate the size factors hyperparameter for each feature by using the external DESeq2::estimateSizeFactorsForMatrix function. Required for normalization of samples with respect to sequencing depth and mandatory when fitting count models.

Examples

```
showClass("csDEXdataSet")
```

`nbreg.fit` *Negative Binomial regression.*

Description

Fit a Negative Binomial regression model. Called internally by the geneModel function.

Usage

```
nbreg.fit(X, y, phi, beta.init = NULL, offset = 0, max.iter = 100, tol = 1e-
lambda.0 = 1e-04, lambda.max = 100, verbose = FALSE)
```

Arguments

X	The design matrix with n rows and p columns.
y	A vector of n target values.
phi	Precision hyperparameter. Either a scalar or a vector of n values.
beta.init	Optional. Initial estimate of parameter values.
offset	Optional. Known offset of target values from a negative binomially distributed data.
max.iter	Number of optimization iteration.
tol	Relative change in log-likelihood to declare convergence.
lambda.0	Initial Levenberg-Marquardt damping factor.
lambda.max	Maximal Levenberg-Marquardt damping factor.
verbose	Print diagnostic messages.

Details

Custom implementation of the Negative Binomial regression model fitting with Levenberg-Marquardt update, enabling custom initialization values.

Value

coefficients	Parameter values for the mean of each fitted value.
fitted.values	Means of n fitted values.
converged	Binary convergence factor. If FALSE, use verbose=TRUE to determine possible reasons.
phi	Precision.
loglik	Log-likelihood of model parameters given data.
iter	Number of executed gradient steps.
vcov	Parameter variance-covariance matrix.

Author(s)

Martin Stražar

---

testForDEU *Test for differential usage of exons using the PSI or the count models.*

Description

Difference of deviance tests for interactions between each feature and condition for an instance of a gene. Produces a ranked list of significant candidate interactions. The function geneModel invoke by testForDEU and possibly dispatched to multiple cores. The waldTest is invoked by geneModel if model approximation is used.

Usage

```
testForDEU(cdx, workers = 1, tmp.dir = NULL, min.cpm = NULL,
  alpha.wald = NULL, formula = y ~ featureID + condition)
geneModel(input, min.cpm = NULL, tmp.dir = NULL, dist="count",
  alpha.wald = NULL, formula = y ~ featureID + condition)
waldTest(mm0, model0, alpha=0.05)
```

Arguments

cdx	A csDEXdataSet object.
workers	Number of cores.
tmp.dir	Temporary directory. If set, results for individual genes are stored immediately as they are computed.
min.cpm	Minimal counts per million reads (CPM) for a gene to be considered. No positions within a gene with a CPM below the set threshold will be tested. An output data frame is still produced, with appropriate message in the msg field.
alpha.wald	Significance threshold for model approximation based on the Wald test for parameter significance. If set, any parameters non-significantly different from zero in the null model are merged to a single "error" parameter. Can significantly reduce computation time for large and sparse datasets.
formula	Formula object describing the null model. For the alternative model, an interaction term for the current '(featureID, condition)' pair is appended.

Details

The core function of the package. Evaluate significance of all possible pairs of interactions between features and conditions by means of difference in deviance model selection. Generalized linear models (GLMs) based on the Negative Binomial distribution are assumed for the quantification based on read counts. The Beta distribution GLM is used to model data based on percentage-spliced (PSI) quantification.

The returned data frame containing interactions will be sorted according to statistical significance (determined by the p-value and padj). The residual value can be used to test the absolute deviation of the null model from the expected value. In the event of an optimization error (numerical problem in matrix inversion or non-convergence) in null or alternative model fitting, the feature-condition pair will be declared non-testable, with corresponding error message stored in the message field.

The read count model requires additional hyperparameters, to be stored in the csDEXDataSet object, which need to be precomputed by package methods estimateSizeFactors, estimateDispersions, estimateGeneCPM, in that order. The PSI model can be run on a constructed csDEXDataSet object directly.

The time complexity of GLM fitting is cubic in the number of parameters, which equal the sum of number of features and conditions. A model approximation based on Wald test of parameter significance can be used by setting the alpha.wald argument to a significance threshold in (0, 1), e.g. 0.05. The internal method waldTest is invoked for model approximation. This leads to faster execution, without affecting the probability of Type I error (i.e. false-positive rate), but affecting the Type II error (i.e. false-negative rate). The returned values nrow, ncol and time can be used to compare the savings with respect to the full model. See references for a detailed description of the approximation.

Value

`featureID`	Feature identifier.
`groupID`	Gene identifier.
`condition`	Condition identifier.
`y`	The expression value of feature in the given experimental condition. A number in open interval (0, 1) for the PSI model or a non-negative integer for the count model.
`cpm`	Gene count per million reads (CPM), used by the count model.
`pvalue`	Statistical significance of the interaction (probability of type 1 error).
`padj`	Bonferroni-adjusted p-value on a gene-level.
`residual`	Model residual. The difference between actual value y and the fitted value of the null model.
`testable`	Whether the interaction complies to used defined threshold and model optimization converged without error.
`loglik`	Alternative model log-likelihood.
`LR`	Difference of deviance.
`time`	Time for alternative model fitting.
`nrow`	Number of rows in the model design matrix.
`ncol`	Number of columns in the model design matrix.
`msg`	Error/warning message, if any.

Note

See the vignette for examples.

Author(s)

Martin Stražar

Index