Package: diffdriver (via r-universe)

November 23, 2026

Type Package

Title Identify Differential Selection

Version 0.1.8

Description Tests for context-dependent selection on cancer driver genes using somatic mutation data. The package implements the DiffDriver statistical framework to assess whether the strength of positive selection on mutations in a driver gene is associated with tumor- or individual-level context variables, such as clinical traits, genomic features, or immune microenvironment subtypes. DiffDriver estimates individual- and position-specific background mutation rates, models selection as a deviation from the background rate using functional annotations, and tests context effects through a latent-variable logistic model. It provides utilities for preparing mutation and annotation data, fitting differential-selection models, running gene-level association tests, summarizing candidate genes, and visualizing mutation patterns. The method is described in Zhou et al. (2026) ``Detecting context-dependent selection on cancer driver genes with DiffDriver'' <doi:10.64898/2026.04.06.716771>.

URL https://szhaolab.github.io/diffdriver/, https://github.com/szhaolab/diffdriver

License MIT + file LICENSE

Encoding UTF-8

Imports Matrix, stats, data.table, brglm, fastTopics, SQUAREM

Suggests knitr, rmarkdown, logging, testthat (>= 3.0.0)

Config/testthat/edition 3

Depends R (>= 3.5.0)

VignetteBuilder knitr

Config/roxygen2/version 8.1.1

NeedsCompilation no

Author Siming Zhao [aut, cre] (Additional email:
siming.zhao06@gmail.com), Jie Zhou [aut], Qirui Zhang [aut]

Maintainer Siming Zhao <siming.zhao@dartmouth.edu>

Repository https://cran.r-universe.dev

Date/Publication 2026-11-23 14:58:41 UTC

RemoteUrl https://github.com/cran/diffdriver

RemoteRef HEAD

RemoteSha 6efefcb737fe88b154fa10b3282556928f45b9f4

Contents

ddmcode 2
ddmodel 3
ddmodel_binary 5
ddmodel_binary_simple 6
diffdriver 7
genebinom 10
genefisher 11
genelr 12
mlr 13
mlr.v2 14
optifix 15
plot_mut 17

Index **19**

`ddmcode` *Add an Intercept and Encode Functional Annotations (Codes 7 and 8)*

Description

Process annotation tables within a list of mutation data. An intercept column is added to each annotation table. When `functypecode` is included in `selectvars`, functional annotation codes 7 and 8 are converted to a binary indicator named `functypecode8`, with values 0 and 1, respectively. When recoding is enabled, the binary indicator is placed immediately after the intercept, before the remaining annotation columns. Unlike driverMAPS, this implementation explicitly handles only functional annotation codes 7 and 8 when `functypecode` is selected. Other codes are not assigned an indicator value.

Usage

```
ddmcode(matrixlist, selectvars, functypecodelevel = NULL)
```

Arguments

`matrixlist`	A list of mutation data entries. The first element of each entry contains an annotation table, and the second element contains corresponding mutation data used to determine row counts.
`selectvars`	A character vector of selected annotation variables. The variable `nttypecode` is excluded. If `functypecode` is included, it is replaced by the binary indicator `functypecode8`.
`functypecodelevel`	An optional argument that is not used by the current implementation.

Value

The updated `matrixlist`, with an intercept column added to each annotation table and functional codes recoded when applicable.

`ddmodel` *diffDriver model*

Description

Fit the DiffDriver model for a single gene, incorporating both sample-level phenotype information and position-level functional annotations. The position-level functional effects are supplied as inputs, while model parameters describing selection and the phenotype effect are estimated using an expectation-maximization (EM) algorithm with SQUAREM acceleration. The model tests whether the phenotype is associated with positive selection, optionally adjusting for sample-level covariates.

Usage

```
ddmodel(mut, e, mr, fe, label, cov = NULL, ...)
```

Arguments

`mut`	A matrix of mutation status indicators (0 or 1), with rows corresponding to mutation positions and columns to samples. Rows sharing the same label are aggregated during model fitting.
`e`	A numeric vector of sample-level phenotype values, aligned with the columns of `mut`.
`mr`	A matrix of log-scale background mutation rates for each mutation position and sample, not dependent on the sample-level phenotype factor. It has the same dimensions as `mut`.
`fe`	A vector of log-scale increases in mutation rate due to position-level functional effects, aligned with the rows of `mut` and `mr`. The model uses `exp(fe)` to adjust the background mutation rate before estimating selection effects.
`label`	A vector or factor of length `nrow(mut)` identifying mutation-position rows to be grouped. Mutation counts and mutation rates are summed within each group before model fitting.

cov	an optional numeric matrix of sample-level covariates (e.g. age, sex), with rows matching the columns (samples) of mut. When supplied, the covariates are added to the logistic regression of the EM responsibilities under both the null and alternative models, so the phenotype effect is tested conditional on the covariates. Default NULL (no covariates).
...	Additional arguments passed to dd_squarEM(), including maxit (maximum iterations) and tol (convergence tolerance).

Value

A list containing:

pvalue Likelihood-ratio test p-value comparing the null and alternative models for the phenotype effect.

res.null Null-model fitting results, including loglikelihood, beta0, and alpha.

res.alt Alternative-model fitting results, including loglikelihood, beta0, and alpha.

Examples

```
# Synthetic mutation matrix: rows are positions and columns are samples.
mut <- matrix(0, nrow = 6, ncol = 20)

mut[1, 1] <- 1
mut[2, 5] <- 1
mut[c(1, 2), 11] <- 1
mut[c(2, 3), 12] <- 1
mut[c(3, 4), 13] <- 1
mut[c(4, 5), 14] <- 1
mut[c(5, 6), 15] <- 1
mut[c(1, 6), 16] <- 1
mut[c(1, 3), 17] <- 1
mut[c(2, 4), 18] <- 1
mut[c(3, 5), 19] <- 1
mut[c(4, 6), 20] <- 1

e <- c(rep(0, 10), rep(1, 10))

# Mutation rates and functional effects are on the log scale.
mr <- matrix(log(0.002), nrow = nrow(mut), ncol = ncol(mut))
fe <- rep(log(3), nrow(mut))

# Each row represents a separate mutation position.
label <- factor(seq_len(nrow(mut)))

result <- ddmodel(mut, e, mr, fe, label = label)
result$pvalue

# Adjust for two sample-level covariates.
cov_matrix <- cbind(
  age = rep(c(-1, 0, 1, 0, -1), 4),
  sex = rep(c(0, 1), 10)
```

```
)

result_cov <- ddmodel(
  mut, e, mr, fe,
  label = label,
  cov = cov_matrix
)
result_cov$pvalue
```

`ddmodel_binary` *diffDriver model only for binary phenotype.*

Description

This function uses the model as cmodel.frac, but generalizes to take more than 1 functional categories. This model is applied on data of a single gene

Usage

```
ddmodel_binary(mut, e, bmr, fe)
```

Arguments

`mut`	a matrix of mutation status 0 or 1
`e`	a vector,phenotype of each sample, should match the columns of `mut` and `bmr`
`bmr`	a matrix, background mutation rate of each sample at each mutation (log scale)
`fe`	a vector, increased mutation rate at each mutation, due to functional effect (log scale), should match the rows of `mut` and `bmr`

Value

A list with the following elements:

`pvalue` The likelihood-ratio test p-value comparing the null and alternative binary-phenotype models.

`null.eta0` The fitted null-model parameter.

`alt.eta` The fitted alternative-model parameters for the two phenotype groups.

`null.ll` The maximized log-likelihood under the null model.

`alt.ll` The maximized log-likelihood under the alternative model.

`all` A named numeric vector containing `pvalue`, `null_para`, `e=0`, `e=1`, `null_likelihood`, and `alt_likelihood`.

Examples

```
# Synthetic mutation matrix: rows are positions and columns are samples.
mut <- matrix(
  c(
    1, 0, 0, 0, 0, 0, 1, 0,
    0, 0, 0, 1, 0, 0, 0, 0,
    0, 1, 0, 0, 0, 0, 0, 1
  ),
  nrow = 3,
  byrow = TRUE
)
e <- c(0, 0, 0, 0, 1, 1, 1, 1)
# Background mutation rates and functional effects are on the log scale.
bmr <- matrix(log(0.001), nrow = nrow(mut), ncol = ncol(mut))
fe <- log(c(2, 2, 2))

result <- ddmodel_binary(mut, e, bmr, fe)
result$pvalue
```

---

ddmodel_binary_simple *diffDriver model only for binary phenotype, assuming bmr the same across samples.*

Description

This function uses the model as cmodel.frac, but generalizes to take more than 1 functional categories. This model is applied on data of a single gene. This should give the same results as ddmodel_binary defined above.

Usage

```
ddmodel_binary_simple(mut, e, bmr, fe)
```

Arguments

`mut`	a matrix of mutation status 0 or 1
`e`	a vector,phenotype of each sample, should match the columns of `mut` and `bmr`
`bmr`	a matrix, background mutation rate of each sample at each mutation (log scale), as we assume bmr the same across samples, only the first column will be used.
`fe`	a vector, increased mutation rate at each mutation, due to functional effect (log scale), should match the rows of `mut` and `bmr`

Value

A list with the following elements:

`pvalue` The likelihood-ratio test p-value comparing the null and alternative binary-phenotype models.

`null.eta0` The fitted null-model parameter.

`alt.eta` The fitted alternative-model parameters for the two phenotype groups.

`null.ll` The maximized log-likelihood under the null model.

`alt.ll` The maximized log-likelihood under the alternative model.

Examples

```
# Synthetic mutation matrix: rows are positions and columns are samples.
mut <- matrix(
  c(
    1, 0, 0, 0, 0, 0, 1, 0,
    0, 0, 0, 1, 0, 0, 0, 0,
    0, 1, 0, 0, 0, 0, 0, 1
  ),
  nrow = 3,
  byrow = TRUE
)
e <- c(0, 0, 0, 0, 1, 1, 1, 1)
# Background mutation rates and functional effects are on the log scale.
bmr <- matrix(log(0.001), nrow = nrow(mut), ncol = ncol(mut))
fe <- log(c(2, 2, 2))

result <- ddmodel_binary_simple(mut, e, bmr, fe)
result$pvalue
```

`diffdriver` *Run diffDriver with Input Files*

Description

This function runs diffDriver.

Usage

```
diffdriver(
  gene,
  mut,
  pheno,
  covariates = NULL,
  anno_dir = ".",
  k = 6,
  totalnttype = 96,
  BMRmode = c("signature", "regular"),
  output_dir = tempdir(),
  output_prefix = "diffdriver_results"
)
```

Arguments

gene	A vector of genes to be included in the analysis.

mut	A data frame containing all somatic mutations from the cohort. The format is:
	Chromosome <int>
	Position <int>
	Ref <chr>
	Alt <chr>
	SampleID <chr>
	Example:
	`Chromosome Position Ref Alt SampleID`
	`1 19 55653236 C T TCGA-N6-A4VE-01A-11D-A28R-08`

pheno	A data frame containing sample phenotypes. The format is: #'
	SampleID <chr>
	Phenotype1 <dbl>
	Phenotype2 <dbl>
	... ...
	Example:
	`SampleID SmokingCessation BMI`
	`TCGA-N5-A4R8-01A-11D-A28R-08 0.5319630 20.0`
	`TCGA-N5-A4RD-01A-11D-A28R-08 0.0448991 24.4`

covariates	Optional sample-level covariates (e.g. age, sex) to adjust for. Either a path to a delimited file or a data frame. It must contain a `SampleID` column plus one or more covariate columns, which may be numeric or categorical (categorical columns are expanded into dummy variables). Only samples with complete covariate information are kept, and these are intersected with the mutation and phenotype samples. Any covariate that is single-level (has no variance) among the retained samples is dropped with a warning, since it carries no information; if no covariate with variance remains, the analysis proceeds without covariate adjustment. When supplied, the covariates are added to the regression inside `ddmodel`/`ddmodel_nl` under both the null and alternative models, so the phenotype effect is tested conditional on the covariates. The format is:
	SampleID <chr>
	Covariate1 <dbl> or <chr>
	Covariate2 <dbl> or <chr>
	... ...
	Example:
	`SampleID Age Sex`
	`TCGA-N5-A4R8-01A-11D-A28R-08 56 M`
	`TCGA-N5-A4RD-01A-11D-A28R-08 63 F`
	The default is `NULL` (no covariates).

anno_dir	The path to the directory with all the annotation files. Please download from Zenodo.The default is current folder

k	The number of topics used in modeling background mutation rate. The default is 6.
totalnttype	either 9 or 96. Will look for annotation files anno9_ntypexxx_annodata.txt when totalnttype is 9 or anno96_ntypexxx_annodata when totalnttype is 96.
BMRmode	There are two modes to run diffdriver. One is "signature", this will model individual level BMR, this is the default. The second one is "regular", this assumes BMR is the same across individuals, only models position-level difference.
output_dir	The path to the output directory. Defaults to a temporary directory.
output_prefix	The prefix being added to the output file names.

Value

A data frame with one row per tested gene. The returned columns include `dd.p`, `mlr.p`, `mlr.v2.p`, `fisher.p`, `binom.p`, and `lr.p`, which are p-values from DiffDriver and comparator methods; the corresponding `.fdr` columns, which contain Benjamini-Hochberg adjusted p-values; and `mut.E1`, `mut.E0`, `E1`, and `E0`, which summarize mutation and sample counts in the two phenotype groups used by the Fisher test.

Examples

```
phenof <- system.file(
  "extdata", "example_phenotypes.txt",
  package = "diffdriver"
)
genef <- system.file(
  "extdata", "example_gene.txt",
  package = "diffdriver"
)
mutf <- system.file(
  "extdata", "example_mutations.txt",
  package = "diffdriver"
)

pheno <- read.table(phenof, header = TRUE)
gene <- read.table(genef, header = FALSE)
mut <- read.table(mutf, header = TRUE)

## Not run:
# Download the annotation files and replace this with their directory.
annodir <- "/path/to/annodir96"

result <- diffdriver(
  gene = gene,
  mut = mut,
  pheno = pheno,
  anno_dir = annodir,
  k = 6,
  totalnttype = 96,
  BMRmode = "signature",
  output_dir = tempdir(),
```

```
  output_prefix = "diffdriver_example"
)
result

## End(Not run)
```

---

genebinom *Gene-Level Binomial Test*

Description

Perform a gene-level binomial test comparing mutation counts between two phenotype groups. The expected proportion of mutations in the group coded as 1 is determined by the proportion of samples in that group.

Usage

```
genebinom(mut, e)
```

Arguments

mut	A numeric matrix of mutation counts or indicators, with rows corresponding to mutation positions and columns corresponding to samples.
e	A binary phenotype or group indicator containing 0 and 1, with one value for each column of mut.

Value

A list containing:

pvalue The p-value from a binomial test comparing the total mutation counts in the two phenotype groups, using their relative sample sizes to determine the expected proportion.

Examples

```
# Synthetic mutation matrix: rows are positions and columns are samples.
mut <- matrix(
  c(
    1, 0, 0, 1, 0, 0,
    0, 1, 0, 0, 1, 0
  ),
  nrow = 2,
  byrow = TRUE
)
e <- c(0, 0, 0, 1, 1, 1)

result <- genebinom(mut, e)
result$pvalue
```

genefisher *Gene-Level Fisher's Exact Test*

Description

Perform a gene-level Fisher's exact test using the total mutation counts and sample counts from two phenotype groups. The function constructs a 2-by-2 table from these counts and obtains the p-value using `fisher.test()`.

Usage

```
genefisher(mut, e)
```

Arguments

`mut`	A numeric matrix of mutation counts or indicators, with rows corresponding to mutation positions and columns corresponding to samples.
`e`	A binary phenotype or group indicator containing 0 and 1, with one value for each column of `mut`.

Value

A list containing:

`pvalue` The p-value from the gene-level Fisher's exact test.

`count` A numeric vector `c(gmutc, gmutn, mu.c, mu.n)`, where `gmutc` and `gmutn` are the total mutation counts in groups 1 and 0, respectively, and `mu.c` and `mu.n` are the corresponding numbers of samples.

Examples

```
# Synthetic mutation matrix: rows are positions and columns are samples.
mut <- matrix(
  c(
    1, 0, 0, 1, 0, 0,
    0, 1, 0, 0, 1, 0
  ),
  nrow = 2,
  byrow = TRUE
)
e <- c(0, 0, 0, 1, 1, 1)

result <- genefisher(mut, e)
result$pvalue
result$count
```

genelr *Gene-Level Logistic Regression*

Description

Perform gene-level logistic regression to assess the association between mutation status and a binary phenotype, optionally adjusting for sample-level covariates. Mutation status is defined as 1 if a sample has at least one mutation in the gene, and 0 otherwise.

Usage

```
genelr(mut, e, covariates = rep(1, length(e)))
```

Arguments

`mut`	A numeric matrix of mutation counts or indicators, with rows corresponding to mutation positions and columns corresponding to samples.
`e`	A binary phenotype vector containing 0 and 1, with one value for each column of `mut`.
`covariates`	Optional sample-level covariates included in the logistic regression model, aligned with the columns of `mut`. The default is a vector of ones, which does not introduce an additional varying covariate.

Value

A list containing:

`res` The summary of the fitted logistic regression model `glm(e ~ dstatus + covariates, family = binomial(link = "logit"))`.

`pvalue` The p-value for the mutation-status coefficient `dstatus`.

Examples

```
# Synthetic mutation matrix: rows are positions and columns are samples.
mut <- matrix(
  c(
    1, 0, 0, 1, 0, 0,
    0, 1, 0, 0, 1, 0
  ),
  nrow = 2,
  byrow = TRUE
)
e <- c(0, 0, 0, 1, 1, 1)

# Logistic regression without additional varying covariates.
result <- genelr(mut, e)
result$pvalue
```

```
# Logistic regression adjusting for a sample-level covariate.
covariates <- c(0, 1, 0, 0, 1, 0)
result_cov <- genelr(mut, e, covariates = covariates)
result_cov$pvalue
```

`mlr` *Gene-Level Multiple Linear Regression*

Description

Perform gene-level linear regression to assess the association between mutation status and a numeric phenotype, optionally adjusting for sample-level covariates. Mutation status is defined as 1 if a sample has at least one mutation in the gene, and 0 otherwise.

Usage

```
mlr(mut, e, covariates = rep(1, length(e)))
```

Arguments

`mut`	A numeric matrix of mutation counts or indicators, with rows corresponding to mutation positions and columns corresponding to samples.
`e`	A numeric phenotype vector, with one value for each column of `mut`.
`covariates`	Optional sample-level covariate values included in the linear regression model, aligned with the columns of `mut`. The default is a vector of ones, which does not introduce an additional varying covariate.

Value

A list containing:

`res` The summary of the fitted linear regression model `lm(e ~ dstatus + covariates)`.

`pvalue` The p-value for the mutation-status coefficient `dstatus`.

Examples

```
# Synthetic mutation matrix: rows are positions and columns are samples.
mut <- matrix(
  c(
    1, 0, 0, 1, 0, 0,
    0, 1, 0, 0, 1, 0
  ),
  nrow = 2,
  byrow = TRUE
)
e <- c(0, 0, 0, 1, 1, 1)

# Linear regression without additional varying covariates.
```

```
result <- mlr(mut, e)
result$pvalue

# Linear regression adjusting for a sample-level covariate.
covariates <- c(0, 1, 0, 0, 1, 0)
result_cov <- mlr(mut, e, covariates = covariates)
result_cov$pvalue
```

`mlr.v2` *Gene-Level Multiple Linear Regression Adjusting for Total Mutation Count*

Description

Perform gene-level linear regression to assess the association between mutation status and a numeric phenotype, adjusting for the total number of mutations per sample. Mutation status is defined as 1 if a sample has at least one mutation in the gene, and 0 otherwise.

Usage

```
mlr.v2(mut, e, nmut, covariates = 1)
```

Arguments

`mut`	A numeric matrix of mutation counts or indicators, with rows corresponding to mutation positions and columns corresponding to samples.
`e`	A numeric phenotype vector, with one value for each column of `mut`.
`nmut`	A numeric vector containing the total number of mutations for each sample, aligned with the columns of `mut`.
`covariates`	An optional argument for additional covariates. The default value `1` indicates no additional varying covariate. Sample-level covariate vectors are not supported by the current implementation.

Value

A list containing:

`res` The summary of the fitted linear regression model `lm(e ~ dstatus + nmut + covariates)`.

`pvalue` The p-value for the mutation-status coefficient `dstatus`, after adjusting for total mutation count.

Examples

```
# Synthetic mutation matrix: rows are positions and columns are samples.
mut <- matrix(
  c(
    1, 0, 0, 1, 0, 0,
    0, 1, 0, 0, 1, 0
  ),
  nrow = 2,
  byrow = TRUE
)
e <- c(0, 0, 0, 1, 1, 1)

# Synthetic per-sample total mutation counts.
nmut <- c(3, 4, 5, 6, 7, 8)

result <- mlr.v2(mut, e, nmut)
result$pvalue
```

---

optifix *optifix. Optimise with fixed parameters*

Description

its like optim, but with fixed parameters.

Usage

```
optifix(
  par,
  fixed,
  fn,
  gr = NULL,
  ...,
  method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN"),
  lower = -Inf,
  upper = Inf,
  control = list(),
  hessian = FALSE
)
```

Arguments

par	A numeric vector of initial parameter values.
fixed	A logical vector of the same length as par. TRUE indicates a parameter that remains fixed; FALSE indicates a parameter to be optimized.
fn	The objective function to be minimized. It receives the full parameter vector, including fixed parameters.

gr	An optional gradient function. It receives the full parameter vector.
...	Additional arguments passed through optim() to fn and gr, as applicable.
method	The optimization method passed to optim().
lower	Lower parameter bounds passed to optim().
upper	Upper parameter bounds. This argument is present in the function interface but is not passed to optim() by the current implementation.
control	A list of control parameters passed to optim().
hessian	Logical; whether to request a Hessian matrix from optim().

Details

optifix. Optimise with fixed parameters

its like optim, but with fixed parameters.

specify a second argument 'fixed', a vector of TRUE/FALSE values. If TRUE, the corresponding parameter in fn() is fixed. Otherwise its variable and optimised over.

The return thing is the return thing from optim() but with a couple of extra bits - a vector of all the parameters and a vector copy of the 'fixed' argument.

Written by Barry Rowlingson <b.rowlingson@lancaster.ac.uk> October 2011

This file released under a CC By-SA license: http://creativecommons.org/licenses/by-sa/3.0/

and must retain the text: "Originally written by Barry Rowlingson" in comments.

Value

A list containing the results returned by optim(), with two additional elements:

fullpars The complete parameter vector, including both optimized and fixed parameters.

fixed The logical vector identifying which parameters were held fixed.

The par element contains the optimized free parameters.

Examples

```
# Minimize a quadratic function while fixing the second parameter.
fn <- function(p) {
  (p[1] - 3)^2 + (p[2] - 5)^2
}

optifix_fun <- getFromNamespace("optifix", "diffdriver")
fit <- optifix_fun(
  par = c(0, 5),
  fixed = c(FALSE, TRUE),
  fn = fn,
  method = "BFGS"
)

fit$fullpars
fit$value
```

plot_mut *Plot Phenotype, Mutation, and Annotation Summaries for a Gene Across Samples*

Description

Visualize sample-level phenotype values, mutation counts, and functional mutation annotations for a selected gene. The function draws a series of bar plots on the current graphics device using the supplied mutation and phenotype data.

Usage

```
plot_mut(
  gene_name,
  mut,
  pheno,
  totalnttype = 96,
  anno_dir = ".",
  output_prefix = "plot",
  output_dir = tempdir()
)
```

Arguments

gene_name	A character string specifying the gene to plot.
mut	A mutation data table containing sample identifiers and mutation information, including chromosome, position, reference allele, and alternative allele.
pheno	A table of sample-level phenotype or context data corresponding to the mutation data.
totalnttype	The number of nucleotide context types used for the annotation data. Default is 96.
anno_dir	Directory containing the positional annotation files required by prep_positional_data(). Default is the current working directory.
output_prefix	Prefix passed to the positional data preprocessing function. Default is "plot".
output_dir	Directory passed to the positional data preprocessing function. Defaults to tempdir(). The plots themselves are drawn on the current graphics device.

Value

A numeric vector of bar midpoints returned by the final barplot() call. The function is primarily used for its side effect of drawing phenotype and mutation summary plots.

Examples

```
phenof <- system.file(
  "extdata", "example_phenotypes.txt",
  package = "diffdriver"
)
mutf <- system.file(
  "extdata", "example_mutations.txt",
  package = "diffdriver"
)

pheno <- read.table(phenof, header = TRUE)
mut <- read.table(mutf, header = TRUE)

## Not run:
# Download the annotation files and replace this with their directory.
annodir <- "/path/to/annodir9"

plot_mut(
  gene_name = "PIK3CA",
  mut = mut,
  pheno = pheno,
  totalnttype = 9,
  anno_dir = annodir,
  output_dir = tempdir()
)

## End(Not run)
```

Index

ddmcode, 2
ddmodel, 3
ddmodel_binary, 5
ddmodel_binary_simple, 6
diffdriver, 7

genebinom, 10
genefisher, 11
genelr, 12

mlr, 13
mlr.v2, 14

optifix, 15

plot_mut, 17