Package 'ICBioMark'

March 18, 2021

Title Data-Driven Design of Targeted Gene Panels for Estimating Immunotherapy Biomarkers

Version 0.1.2

Description Implementation of the methodology proposed in 'Data-driven design of targeted gene panels for estimating immunotherapy biomarkers', Bradley and Cannings (2021) <arXiv:2102.04296>. This package allows the user to fit generative models of mutation from an annotated mutation dataset, and then further to produce tunable linear estimators of exome-wide biomarkers. It also contains functions to simulate mutation annotated format (MAF) data, as well as to analyse the output and performance of models.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

Suggests testthat (>= 2.1.0)

Imports stats, utils, glmnet, Matrix, dplyr, purrr, latex2exp, matrixStats, ggplot2, gglasso, PRROC

Depends R (>= 2.10)

NeedsCompilation no

Author Jacob R. Bradley [aut, cre] (<https://orcid.org/0000-0003-1616-4969>), Timothy I. Cannings [aut] (<https://orcid.org/0000-0002-2111-4168>)

Maintainer Jacob R. Bradley <cobrbradley@gmail.com>

Repository CRAN

Date/Publication 2021-03-18 16:10:02 UTC

R topics documented:

`ensembl_gene_lengths` *Gene Lengths from the Ensembl Database*

Description

Pre-imported length data from the Ensembl database for all genes on chromosomes 1-22, X and Y.

Usage

```
ensembl_gene_lengths
```

Format

A dataframe with three columns:

Hugo_Symbol The names of all nuclear genes in humans for which ensembl entries with coding sequence lengths exist.

max_cds The maximum coding sequence for each gene as given by the ensembl database.

Chromosome The chromosome where each gene is located.

Source

See the folder data-raw.

`example_first_pred_tmb`

First-Fit Predictive Model Fitting on Example Data

Description

An example output from the function pred_first_fit(), applied to pre-loaded example mutation data.

Usage

```
example_first_pred_tmb
```

Format

A list with six entries:

fit A gglasso fit.

panel_genes A matrix where each row corresponds to a gene, each column to an iteration of the group lasso with a different penalty factor, and the elements booleans specifying whether that gene was selected to be included in that iteration.

panel_lengths A vector giving total panel length for each gglasso iteration.

p The vector of weights used in the optimisation procedure.

K The bias penalty factor used in the optimisation procedure.

names Gene and mutation type information as used when fitting the generative model.

example_gen_model *Generative Model from Simulated Data*

Description

An example of the output produced by fit_gen_model() on simulated data.

Usage

```
example_gen_model
```

Format

A list with two entries:

fit A glmnet fit object.

dev A table containing the average deviance of each cross-validation fold, for each penalisation factor in fit$lambda.

s_min The index of the regularisation penalty minimising average deviance across folds.

example_maf_data *Simulated MAF Data*

Description

An example dataset generated by the function generate_maf_data(), with n_sample = 100 and n_genes = 20.

Usage

```
example_maf_data
```

Format

A list with two entries:

maf An annotated mutation dataframe with 3 columns and 1346 rows:

- **Tumor_Sample_Barcode** A sample id for each mutation.
- **Hugo_Symbol** The name of the gene location for each mutation.
- **Variant_Classification** The mutation type for each mutation.

gene_lengths A data frame with two rows:

- **Hugo_Symbol** The name of each gene.
- **max_cds** The length of each gene, as defined by maximum coding sequence.

example_predictions *Example Predictions*

Description

An example output from use of the function get_predictions(), applied to the pre-loaded datasets example_refit_range and example_tables$val .

Usage

```
example_predictions
```

Format

A list with two entries:

predictions A a matrix containing a row for each sample and a column for each panel.

panel_lengths A vector giving total panel lengths.

example_refit_panel *Refitted Predictive Model Fitted on Example Data*

Description

An example output from use of the function pred_refit_panel(), applied to example gene length data and generative model fit.

Usage

```
example_refit_panel
```

Format

A list with three entries:

fit A list with a single element 'beta', a matrix with prediction weights.

panel_genes A matrix (in this case with a single column) where each row corresponds to a gene, and each entry corresponds to whether the gene is included in the panel.

panel_lengths A vector of length 1 giving total panel length.

example_refit_range *Refitted Predictive Models Fitted on Example Data*

Description

An example output from use of the function pred_refit_range(), applied to example gene length data and generative model fit.

Usage

```
example_refit_range
```

Format

A list with six entries:

fit A list with a single element 'beta', a matrix with prediction weights.

panel_genes A matrix where each row corresponds to a gene, and each entry corresponds to whether the gene is included in the panel.

panel_lengths A vector giving total panel length.

example_tables *Mutation Matrices from Simulated Data*

Description

Mutation data extracted from the pre-loaded example mutation data example_maf_data, using the function get_mutation_tables().

Usage

```
example_tables
```

Format

A list with three entries:

train An object 'train'.

val An object 'val'.

test An object 'test'.

Each of these three objects is a list with the following entries (for more detail see the documentation for the function get_table_from_maf()):

matrix A sparse matrix of mutations.

sample_list A character vector of sample IDs, corresponding to the rows of the mutation matrix.

gene_list A character vector of gene names.

mut_types_list A character vector of mutation types.

colnames A character vector of gene name/mutation type combinations (in each case separated by the character "_"), corresponding to the columns of the mutation matrix.

`example_tib_tables` *Tumour Indel Burden of Example Train, Validation and Test Data.*

Description

An example output produced by using the function get_biomarker_tables(), applied to the example MAF data pre-loaded in example_maf_data$maf.

Usage

```
example_tib_tables
```

Format

A list with threeobjects: 'train', 'val' and 'test'. Each is a dataframe with two columns:

Tumor_Sample_Barcode A unique ID for each sample.

TIB The value of Tumour Indel Burden for that sample.

`example_tmb_tables` *Tumour Mutation Burden of Example Train, Validation and Test Data.*

Description

An example output produced by using the function get_biomarker_tables(), applied to the example MAF data pre-loaded in example_maf_data$maf.

Usage

```
example_tmb_tables
```

Format

A list with threeobjects: 'train', 'val' and 'test'. Each is a dataframe with two columns:

Tumor_Sample_Barcode A unique ID for each sample.

TMB The value of Tumour Mutation Burden for that sample.

`fit_gen_model` *Fit Generative Model*

Description

A function to fit a generative model to a mutation dataset. At its heart, requires a gene_lengths dataframe (for examples of the correct format for this see the pre-loaded datasets example_maf_data$gene_lengths and ensembl_gene_lengths), and a mutation dataset. This is best supplied through the 'table' argument, and constructed via the function get_mutation_tables().

Usage

```
fit_gen_model(
  gene_lengths,
  matrix = NULL,
  sample_list = NULL,
  gene_list = NULL,
  mut_types_list = NULL,
  col_names = NULL,
  table = NULL,
  nlambda = 100,
  n_folds = 10,
  maxit = 1e+09,
  seed_id = 1234,
  progress = FALSE,
  alt_model_type = NULL
)
```

Arguments

`gene_lengths`	(dataframe) A table with two columns: Hugo_Symbol and max_cds, providing the lengths of the genes to be modelled.
`matrix`	(Matrix::sparseMatrix) A mutation matrix, such as produced by the function get_table_from_maf().
`sample_list`	(character) The set of samples to be modelled.
`gene_list`	(character) The set of genes to be modelled.
`mut_types_list`	(character) The set of mutation types to be modelled.
`col_names`	(character) The column names of the 'matrix' parameter.
`table`	(list) Optional parameter combining matrix, sample_list, gene_list, mut_types_list, col_names, as is produced by the function get_tables().
`nlambda`	(numeric) The length of the vector of penalty weights, passed to the function glmnet::glmnet().
`n_folds`	(numeric) The number of cross-validation folds to employ.
`maxit`	(numeric) Technical parameter passed to the function glmnet::glmnet().

seed_id	(numeric) Input value for the function set.seed().
progress	(logical) Show progress bars and text.
alt_model_type	(character) Used to call an alternative generative model type such as "US" (no sample-dependent parameters) or "UI" (no gene/variant-type interactions).

Value

A list comprising three objects:

- An object 'fit', a fitted glmnet model.
- A table 'dev', giving average deviances for each regularisation penalty factor and cross-validation fold.
- An integer 's_min', the index of the regularsisation penalty minimising cross-validation deviance.
- A list 'names', containing the sample, gene, and mutation type information of the training data.

Examples

```
example_gen_model <- fit_gen_model(example_maf_data$gene_lengths, table = example_tables$train)
print(names(example_gen_model))
```

---

fit_gen_model_uninteract

Fit Generative Model Without Gene/Variant Type-Specific Interactions

---

Description

A function to fit a generative model to a mutation dataset that does not incorporate gene/variant-specific effects. Otherwise acts similarly to the function fit_gen_model().

NOTE: fits produced by this model will not be compatible with predictive model fits downstream - it is purely for comparing with full models.

Usage

```
fit_gen_model_uninteract(
  gene_lengths,
  matrix = NULL,
  sample_list = NULL,
  gene_list = NULL,
  mut_types_list = NULL,
  col_names = NULL,
  table = NULL,
  nlambda = 100,
  n_folds = 10,
  maxit = 1e+09,
```

```
  seed_id = 1234,
  progress = FALSE
)
```

Arguments

gene_lengths	(dataframe) A table with two columns: Hugo_Symbol and max_cds, providing the lengths of the genes to be modelled.
matrix	(Matrix::sparseMatrix) A mutation matrix, such as produced by the function get_table_from_maf().
sample_list	(character) The set of samples to be modelled.
gene_list	(character) The set of genes to be modelled.
mut_types_list	(character) The set of mutation types to be modelled.
col_names	(character) The column names of the 'matrix' parameter.
table	(list) Optional parameter combining matrix, sample_list, gene_list, mut_types_list, col_names, as is produced by the function get_tables().
nlambda	(numeric) The length of the vector of penalty weights, passed to the function glmnet::glmnet().
n_folds	(numeric) The number of cross-validation folds to employ.
maxit	(numeric) Technical parameter passed to the function glmnet::glmnet().
seed_id	(numeric) Input value for the function set.seed().
progress	(logical) Show progress bars and text.

Value

A list comprising three objects:

- An object 'fit', a fitted glmnet model.
- A table 'dev', giving average deviances for each regularisation penalty factor and cross-validation fold.
- An integer 's_min', the index of the regularsisation penalty minimising cross-validation deviance.
- A list 'names', containing the sample, gene, and mutation type information of the training data.

Examples

```
example_gen_model_unisamp <- fit_gen_model_unisamp(example_maf_data$gene_lengths,
                                                   table = example_tables$train)
print(names(example_gen_model))
```

fit_gen_model_unisamp *Fit Generative Model Without Sample-Specific Effects*

Description

A function to fit a generative model to a mutation dataset that does not incorporate sample-specific effects. Otherwise acts similarly to the function fit_gen_model().

NOTE: fits produced by this model will not be compatible with predictive model fits downstream - it is purely for comparing with full models.

Usage

```
fit_gen_model_unisamp(
  gene_lengths,
  matrix = NULL,
  sample_list = NULL,
  gene_list = NULL,
  mut_types_list = NULL,
  col_names = NULL,
  table = NULL,
  nlambda = 100,
  n_folds = 10,
  maxit = 1e+09,
  seed_id = 1234,
  progress = FALSE
)
```

Arguments

gene_lengths	(dataframe) A table with two columns: Hugo_Symbol and max_cds, providing the lengths of the genes to be modelled.
matrix	(Matrix::sparseMatrix) A mutation matrix, such as produced by the function get_table_from_maf().
sample_list	(character) The set of samples to be modelled.
gene_list	(character) The set of genes to be modelled.
mut_types_list	(character) The set of mutation types to be modelled.
col_names	(character) The column names of the 'matrix' parameter.
table	(list) Optional parameter combining matrix, sample_list, gene_list, mut_types_list, col_names, as is produced by the function get_tables().
nlambda	(numeric) The length of the vector of penalty weights, passed to the function glmnet::glmnet().
n_folds	(numeric) The number of cross-validation folds to employ.
maxit	(numeric) Technical parameter passed to the function glmnet::glmnet().
seed_id	(numeric) Input value for the function set.seed().
progress	(logical) Show progress bars and text.

Value

A list comprising three objects:

- An object 'fit', a fitted glmnet model.
- A table 'dev', giving average deviances for each regularisation penalty factor and cross-validation fold.
- An integer 's_min', the index of the regularsisation penalty minimising cross-validation deviance.
- A list 'names', containing the sample, gene, and mutation type information of the training data.

Examples

```
example_gen_model_unisamp <- fit_gen_model_unisamp(example_maf_data$gene_lengths,
                                                   table = example_tables$train)
print(names(example_gen_model))
```

`generate_maf_data` *Generate mutation data.*

Description

A function to randomly simulate an (abridged) annotated mutation file, containing information on sample of origin, gene and mutation type, as well as a dataframe of gene lengths.

Usage

```
generate_maf_data(
  n_samples = 100,
  n_genes = 20,
  mut_types = NULL,
  data_dist = NULL,
  sample_rates = NULL,
  gene_rates = NULL,
  gene_lengths = NULL,
  sample_rates_dist = NULL,
  gene_rates_dist = NULL,
  gene_lengths_dist = NULL,
  bmr_genes_prop = 0.7,
  output_rates = FALSE,
  seed_id = 1234
)
```

Arguments

n_samples	(numeric) The number of samples to generate mutation data for - each will have a unique value in the 'Tumor_Sample_Barcode' column of the simulated MAF table. Note that if no mutations are simulated for an example, they will not appear in the table.
n_genes	(numeric) The number of genes to generate mutation data for - each will have a unique value in the 'Hugo_Symbol' column of the simulated MAF table. A length will also be generated for each gene, and stored in the table 'gene_lengths'.
mut_types	(numeric) A vector of positive values giving the relative average abundance of each mutation type. The names of each mutation type are stored in the names attribute of the vector, and will form the entries of the column 'Variant_Classification' in the output MAF table.
data_dist	(function) Directly provide the probability distribution of mutations, as a function on n_samples, n_genes, mut_types, and gene_lengths.
sample_rates	(numeric) Directly provide sample-specific rates.
gene_rates	(numeric) Directly provide gene-specific rates.
gene_lengths	(numeric) Directly provide gene lengths, in the form of a vector of numerics with names attribute corresponding to gene names.
sample_rates_dist	(function) Directly provide the distribution of sample-specific rates, as a function of the number of samples.
gene_rates_dist	(function) Directly provide the distribution of gene-specific rates, as a function of the number of genes.
gene_lengths_dist	(function) Directly provide the distribution of gene lengths, as a function of the number of genes.
bmr_genes_prop	(numeric) The proportion of genes that follow the background mutation rate. If specified (as is automatic), this proportion of genes will have gene-specific rates equal to 1. By setting to be NULL, can avoid applying this step.
output_rates	(logical) If TRUE, will include the sample and gene rates in the output.
seed_id	(numeric) Input value for the function set.seed().

Value

A list with two elements, 'maf' and 'gene_lengths'. These are (respectively):

- A table with three columns: 'Tumor_Sample_Barcode', 'Hugo_Symbol' and 'Variant_Classification', listing the mutations occurring in the simulated example. gene_lengths (dataframe)
- A table with two rows: 'Hugo_Symbol' and 'gene_lengths'.

Examples

```
# Generate some random data
data <- generate_maf_data(n_samples = 10, n_genes = 20)
```

```
# See the first rows of the maf table.
print(head(data$maf))
# See the first rows of the gene_lengths table.
print(head(data$gene_lengths))
```

`get_auprc` *AUPRC Metrics for Predictions*

Description

A function to return AUPRC metrics for predictions vs actual values. Works well when piped to straight from get_predictions().

Usage

```
get_auprc(predictions, biomarker_values, model = "", threshold = 300)
```

Arguments

`predictions`	(list) A list with two elements, 'predictions' and 'panel_lengths', as produced by the function get_predictions().
`biomarker_values`	(dataframe) A dataframe with two columns, 'Tumor_Sample_Barcode' and a column with the name of the biomarker in question containing values.
`model`	(character) The name of the model type producing these predictions.
`threshold`	(numeric) The threshold for biomarker high/low categorisation.

Value

A dataframe with 5 columns:

- panel_length: the length of each panel.
- model: the model that produced the predictions.
- biomarker: the name of the biomarker in question.
- stat: the AUPRC values for each panel.
- metric: a constant character "AUPRC".

Examples

```
example_auprc <- get_auprc(predictions = get_predictions(example_refit_panel,
new_data = example_tables$val), biomarker_values = example_tmb_tables$val,
model = "Refitted T", threshold = 10)
```

get_biomarker_from_maf *Produce a Table of Biomarker Values from a MAF*

Description

A function to recover true biomarker values from a mutation annotation file.

Usage

```
get_biomarker_from_maf(
  maf,
  biomarker = "TIB",
  sample_list = NULL,
  gene_list = NULL,
  biomarker_name = NULL
)
```

Arguments

maf	(dataframe) A table of annotated mutations containing the columns 'Tumor_Sample_Barcode', 'Hugo_Symbol', and 'Variant_Classification'.
biomarker	(character) Which biomarker needs calculating? If "TMB" or "TIB", then appropriate mutation types will be selected. Otherwise, will be interpreted as a vector of characters denoting mutation types to include.
sample_list	(character) Vector of characters giving a list of values of Tumor_Sample_Barcode to include.
gene_list	(character) Vector of characters giving a list of genes to include in calculation of biomarker.
biomarker_name	(character) Name of biomarker. Only needed if biomarker is not "TMB" or "TIB"

Value

A dataframe with two columns, 'Tumor_Sample_Barcode' and values of the biomarker specified.

Examples

```
print(head(get_biomarker_from_maf(example_maf_data$maf, sample_list = paste0("SAMPLE_", 1:100))))
```

`get_biomarker_tables` *Get True Biomarker Values on Training, Validation and Test Sets*

Description

A function, similar to get_mutation_tables(), but returning the true biomarker values for a training, validation and test sets.

Usage

```
get_biomarker_tables(
  maf,
  biomarker = "TIB",
  sample_list = NULL,
  gene_list = NULL,
  biomarker_name = NULL,
  tables = NULL,
  split = c(train = 0.7, val = 0.15, test = 0.15),
  seed_id = 1234
)
```

Arguments

`maf`	(dataframe) A table of annotated mutations containing the columns 'Tumor_Sample_Barcode', 'Hugo_Symbol', and 'Variant_Classification'.
`biomarker`	(character) Which biomarker needs calculating? If "TMB" or "TIB", then appropriate mutation types will be selected. Otherwise, will be interpreted as a vector of characters denoting mutation types to include.
`sample_list`	(character) Vector of characters giving a list of values of Tumor_Sample_Barcode to include.
`gene_list`	(character) Vector of characters giving a list of genes to include in calculation of biomarker.
`biomarker_name`	(character) Name of biomarker. Only needed if biomarker is not "TMB" or "TIB"
`tables`	(list) Optional parameter, the output of a call to get_mutation_tables(), which already has a train/val/test split.
`split`	(numeric) Optional parameter directly specifying the proportions of a train/test/val split.
`seed_id`	(numeric) Input value for the function set.seed().

Value

A list of three objects: 'train', 'val' and 'test. Each comprises a dataframe with two columns, denoting sample ID and biomarker value.

Examples

```
print(head(get_biomarker_tables(example_maf_data$maf, sample_list = paste0("SAMPLE_", 1:100))))
```

`get_gen_estimates` *Investigate Generative Model Comparisons*

Description

Given a generative model of the type we propose, and an alternate version (saturated "S", sample-independent "US", gene-independent "UG" or gene/variant interaction independent "UI"), either produces the estimated observations on the training dataset or calculates residual deviance between models.

Usage

```
get_gen_estimates(
  training_data,
  gen_model,
  alt_gen_model = NULL,
  alt_model_type = "S",
  gene_lengths = NULL,
  calculate_deviance = FALSE
)
```

Arguments

`training_data`	(list) Likely the 'train' component of a call to get_mutation_tables().
`gen_model`	(list) A generative model - result of a call to fit_gen_model*().
`alt_gen_model`	(list) An alternative generative model.
`alt_model_type`	(character) One of "S" (saturated), "US" (sample-independent), "UG", (gene-independent), "UI" (gene/variant-interaction independent).
`gene_lengths`	(dataframe) A gene lengths data frame.
`calculate_deviance`	(logical) If TRUE, returns residual deviance statistics. If FALSE, returns training data predictions.

Value

If calculate_deviance = FALSE:

A list with two entries, est_mut_vec and alt_est_mut_vec, each of length n_samples x n_genes x n_mut_types, giving expected mutation value for each combination of sample, gene and variant type in the training dataset under the two models being compared.

If calculate_deviance = TRUE:

A list with two entries, deviance and df, corresponding to the residual deviance and residual degrees of freedom between the two models on the training set.

Examples

```
sat_dev <- get_gen_estimates(training_data = example_tables$train,
                                        gen_model = example_gen_model,
                                        alt_model_type = "S",
                                        gene_lengths = example_maf_data$gene_lengths,
                                        calculate_deviance = TRUE)
```

---

get_K *Construct Bias Penalisation*

Description

An internal function, producing the correct bias penalisation for use in predictive model fitting.

Usage

```
get_K(
  gen_model,
  p_norm,
  training_matrix,
  marker_training_values = NULL,
  method = max
)
```

Arguments

gen_model	(list) A generative mutation model, fitted by fit_gen_model().
p_norm	(numeric) Scaling factor between coefficients of p and parameters of generative model (see paper for details).
training_matrix	(sparse matrix) A sparse matrix of mutations in the training dataset, produced by get_mutation_tables().
marker_training_values	(dataframe) A dataframe containing training values for the biomarker in question.
method	(function) How to select a representative biomarker value from the training dataset. Defaults to max().

Value

A numerical value, to be used as a penalty weighting in the subsequent group lasso optimisation.

Examples

```
K <- get_K(example_gen_model, 1, example_tables$train$matrix)
print(K)
```

`get_mutation_dictionary` *Group and Filter Mutation Types*

Description

A function to create a mutation dictionary to group and filter mutation types: this can be useful for computational practicality. It is often not practical to model each distinct mutation type together, so for practicality one may group multiple classes together (e.g. all indel mutations, all nonsynonymous mutations). Additionally, some mutation types may be excluded from modelling (for example, one may wish not to use synonymous mutations in the model fitting process).

Usage

```
get_mutation_dictionary(
  for_biomarker = "TIB",
  include_synonymous = TRUE,
  maf = NULL,
  dictionary = NULL
)
```

Arguments

`for_biomarker`	(string) Specify some standard groupings of mutation types, corresponding the the coarsest groupings of nonsynonymous mutations required to evaluate the biomarkers TMB and TIB. If "TMB", groups all nonsynonymous mutations together, if "TIB" groups indel mutations together and all other mutations together.
`include_synonymous`	(logical) Determine whether synonymous mutations should be included in the dictionary.
`maf`	(dataframe) An annotated mutation table containing the column 'Variant_Classification', only used to check if the dictionary specified does not contain all the variant types in your dataset.
`dictionary`	(character) Directly specify the dictionary, in the form of a vector of grouping values. The names of the vector should correspond to the set of variant classifications of interest in the mutation annotated file (MAF).

Value

A vector of characters, with values corresponding to the grouping labels for mutation types, and with names corresponding to the mutation types as they will be referred to in a mutation annotated file (MAF). See examples.

Examples

```
# To understand the dictionary format, note that the following code
dictionary <- get_mutation_dictionary(for_biomarker = "TMB")
# is equivalent to
dictionary <- c(rep("NS",9), rep("S", 8))
names(dictionary) <- c('Missense_Mutation', 'Nonsense_Mutation',
'Splice_Site', 'Translation_Start_Site',
'Nonstop_Mutation', 'In_Frame_Ins',
'In_Frame_Del', 'Frame_Shift_Del',
'Frame_Shift_Ins', 'Silent',
'Splice_Region', '3\'Flank', '5\'Flank',
'Intron', 'RNA', '3\'UTR', '5\'UTR')
# where the grouping levels are chosen to be "NS" and "S" for
# nonsynonymous and synonymous mutations respectively.
# the code
dictionary <- get_mutation_dictionary(for_biomarker = "TIB", include_synonymous = FALSE)
# is equivalent to
dictionary <- dictionary <- c(rep("NS",7), rep("I", 2))
names(dictionary) <- c('Missense_Mutation', 'Nonsense_Mutation',
                       'Splice_Site', 'Translation_Start_Site',
                      'Nonstop_Mutation', 'In_Frame_Ins',
                      'In_Frame_Del', 'Frame_Shift_Del',
                      'Frame_Shift_Ins')
# where now "I" is used as a label to refer to indel mutations,
# and synonymous mutations are filtered out.
```

---

get_mutation_tables *Produce Training, Validation and Test Matrices*

Description

This function allows for i) separation of a mutation dataset into training, validation and testing components, and ii) conversion from annotated mutation format to sparse mutation matrices, as described in the function get_table_from_maf().

Usage

```
get_mutation_tables(
  maf,
  split = c(train = 0.7, val = 0.15, test = 0.15),
  sample_list = NULL,
  gene_list = NULL,
  acceptable_genes = NULL,
  for_biomarker = "TIB",
  include_synonymous = TRUE,
  dictionary = NULL,
  seed_id = 1234
)
```

Arguments

maf	(dataframe) A table of annotated mutations containing the columns 'Tumor_Sample_Barcode', 'Hugo_Symbol', and 'Variant_Classification'.
split	(double) A vector of three positive values with names 'train', 'val' and 'test'. Specifies the proportions into which to split the dataset.
sample_list	sample_list (character) Optional parameter specifying the set of samples to include in the mutation matrices.
gene_list	(character) Optional parameter specifying the set of genes to include in the mutation matrices.
acceptable_genes	(character) Optional parameter specifying a set of acceptable genes, for example those which are in an ensembl databse.
for_biomarker	(character) Used for defining a dictionary of mutations. See the function get_mutation_dictionary() for details.
include_synonymous	(logical) Optional parameter specifying whether to include synonymous mutations in the mutation matrices.
dictionary	(character) Optional parameter directly specifying the mutation dictionary to use. See the function get_mutation_dictionary() for details.
seed_id	(numeric) Input value for the function set.seed().

Value

A list of three items with names 'train', 'val' and 'test'. Each element will contain a sparse mutation matrix for the samples in that branch, alongside other information as described as the output of the function get_table_from_maf().

Examples

```
tables <- get_mutation_tables(example_maf_data$maf, sample_list = paste0("SAMPLE_", 1:100))

print(names(tables))
print(names(tables$train))
```

---

get_p *Construct Optimisation Parameters.*

Description

An internal function. From the learned generative model and training data, produces a vector of weights p to be used in the subsequent group lasso optimisation, alongside a biomarker-dependent normalisation quantity p_norm.

Usage

```
get_p(gen_model, training_matrix, marker_mut_types, gene_lengths)
```

Arguments

gen_model	(list) A generative mutation model, fitted by fit_gen_model().
training_matrix	(sparse matrix) A sparse matrix of mutations in the training dataset, produced by get_mutation_tables().
marker_mut_types	(character) A character vector listing which mutation types (of the set specified in the generative model attribute 'names') constitute the biomarker in question.
gene_lengths	(dataframe) A table with two columns: Hugo_Symbol and max_cds, providing the lengths of the genes to be modelled.

Value

A list with three entries:

- A vector p, with an entry corresponding to each combination of gene and mutation type specified in the generative model fitted. Each component is a non-negative value corresponding to a weighting p to be supplied to a group lasso optimisation.
- A numeric p_norm, giving the factor between p_gs and phi_0gs (see paper for details).
- A vector biomarker_columns, detailing which of the elements of p correspond to gene/mutation type combinations contributing to the biomarker in question.

Examples

```
p <- get_p(example_gen_model, example_tables$train$matrix,
           marker_mut_types = c("I"), gene_lengths = example_maf_data$gene_lengths)
print(p$p[1:5])
print(p$p_norm)
print(p$bc[1:5])
```

---

get_panels_from_fit *Extract Panel Details from Group Lasso Fit*

Description

An internal function for analysing a group Lasso fit as part of the predictive model learning procedure, which returns the sets of genes identified by different iterations of the group Lasso algorithm.

Usage

```
get_panels_from_fit(gene_lengths, fit, gene_list, mut_types_list)
```

Arguments

gene_lengths	(dataframe) A table with two columns: Hugo_Symbol and max_cds, providing the lengths of the genes to be modelled.
fit	(list) A fit from the group lasso algorithm, produced by the function gglasso (package: gglasso).
gene_list	(character) A character vector of genes listing the genes (in order) included in the model pred_fit.
mut_types_list	(character) A character vector listing the mutation type groupings (in order) included in the model pred_fit.

Value

A list of two elements:

- panel_genes: A matrix where each row corresponds to a gene, each column to an iteration of the group lasso with a different penalty factor, and the elements booleans specifying whether that gene was selected to be included in that iteration.
- panel_lengths:

Examples

```
panels <- get_panels_from_fit(example_maf_data$gene_lengths, example_first_pred_tmb$fit,
example_gen_model$names$gene_list, mut_types_list = example_gen_model$names$mut_types_list)

print(panels$fit)
```

---

get_predictions *Produce Predictions on an Unseen Dataset*

Description

A function taking a predictive model(s) and new observations, and applying the predictive model to them to return predicted biomarker values.

Usage

```
get_predictions(pred_model, new_data, s = NULL, max_panel_length = NULL)
```

Arguments

pred_model	(list) A predictive model as fitted by pred_first_fit(), pred_refit_panel() or pred_refit_range().
new_data	(list) A new dataset, containing a matrix of observations and a list of sample IDs. Likely comes from the 'train', 'val' or 'test' argument of a call to get_mutation_tables().
s	(numeric) If producing predictions for a single panel, s chooses which panel (column in a pred_fit object) to produce predictions for.

`max_panel_length`
: (numeric) If producing predictions for a single panel, maximum panel length to specify that panel.

Value

A list with two elements:

- predictions, a matrix containing a row for each sample and a column for each panel.
- panel_lengths, a vector containing the length of each panel.

Examples

```
example_predictions <- get_predictions(example_refit_range, new_data =
example_tables$val)
```

---

get_r_squared *R Squared Metrics for Predictions*

Description

A function to return R^2 metrics for predictions vs actual values. Works well when piped to straight from get_predictions().

Usage

```
get_r_squared(predictions, biomarker_values, model = "", threshold = 10)
```

Arguments

`predictions`
: (list) A list with two elements, 'predictions' and 'panel_lengths', as produced by the function get_predictions().

`biomarker_values`
: (dataframe) A dataframe with two columns, 'Tumor_Sample_Barcode' and a column with the name of the biomarker in question containing values.

`model`
: (character) The name of the model type producing these predictions.

`threshold`
: (numeric) Unusued in this function: present for calls to get_stats().

Value

A dataframe with 5 columns:

- panel_length: the length of each panel.
- model: the model that produced the predictions.
- biomarker: the name of the biomarker in question.
- stat: the R squared values for each panel.
- metric: a constant character "R" for R squared.

Examples

```
example_r <- get_r_squared(predictions = get_predictions(example_refit_panel, new_data =
  example_tables$val), biomarker_values = example_tmb_tables$val, model = "Refitted T")
```

`get_stats` *Metrics for Predictive Performance*

Description

A function to return a variety metrics for predictions vs actual values. Works well when piped to straight from get_predictions().

Usage

```
get_stats(
  predictions,
  biomarker_values,
  model = "",
  threshold = 300,
  metrics = c("R", "AUPRC")
)
```

Arguments

`predictions`	(list) A list with two elements, 'predictions' and 'panel_lengths', as produced by the function get_predictions().
`biomarker_values`	(dataframe) A dataframe with two columns, 'Tumor_Sample_Barcode' and a column with the name of the biomarker in question containing values.
`model`	(character) The name of the model type producing these predictions.
`threshold`	(numeric) The threshold for biomarker high/low categorisation.
`metrics`	(character) A vector of the names of metrics to calculate.

Value

dataframe with 5 columns:

- panel_length: the length of each panel.
- model: the model that produced the predictions.
- biomarker: the name of the biomarker in question.
- stat: the metric values for each panel.
- metric: the name of the metric.

Examples

```
example_stat <- get_stats(predictions = get_predictions(example_refit_panel,
new_data = example_tables$val), biomarker_values = example_tmb_tables$val,
model = "Refitted T", threshold = 10)
```

`get_table_from_maf` *Produce a Mutation Matrix from a MAF*

Description

A function to, given a mutation annotation dataset with columns for sample barcode, gene name and mutation type, to reformulate this as a mutation matrix, with rows denoting samples, columns denoting gene/mutation type combinations, and the individual entries giving the number of mutations observed. This will likely be very sparse, so we save it as a sparse matrix for efficiency.

Usage

```
get_table_from_maf(
  maf,
  sample_list = NULL,
  gene_list = NULL,
  acceptable_genes = NULL,
  for_biomarker = "TIB",
  include_synonymous = TRUE,
  dictionary = NULL
)
```

Arguments

`maf`	(dataframe) A table of annotated mutations containing the columns 'Tumor_Sample_Barcode', 'Hugo_Symbol', and 'Variant_Classification'.
`sample_list`	(character) Optional parameter specifying the set of samples to include in the mutation matrix.
`gene_list`	(character) Optional parameter specifying the set of genes to include in the mutation matrix.
`acceptable_genes`	(character) Optional parameter specifying a set of acceptable genes, for example those which are in an ensembl databse.
`for_biomarker`	(character) Used for defining a dictionary of mutations. See the function get_mutation_dictionary() for details.
`include_synonymous`	(logical) Optional parameter specifying whether to include synonymous mutations in the mutation matrix.
`dictionary`	(character) Optional parameter directly specifying the mutation dictionary to use. See the function get_mutation_dictionary() for details.

Value

A list with the following entries:

- matrix: A mutation matrix, a sparse matrix showing the number of mutations present in each sample, gene and mutation type.

- sample_list: A vector of characters specifying the samples included in the matrix: the rows of the mutation matrix correspond to each of these.
- gene_list: A vector of characters specifying the the genes included in the matrix.
- mut_types_list: A vector of characters specifying the mutation types (as grouped into an appropriate dictionary) to be included in the matrix.
- col_names: A vector of characters identifying the columns of the mutation matrix. Each entry will be comprised of two parts separated by the character '_', the first identifying the gene in question and the second identifying the mutation type. E.g. 'GENE1_NS" where 'GENE1' is an element of gene_list, and 'NS' is an element of the dictionary vector.

Examples

```
# We use the preloaded maf file example_maf_data
# Now we make a mutation matrix
table <- get_table_from_maf(example_maf_data$maf, sample_list = paste0("SAMPLE_", 1:100))

print(names(table))
print(table$matrix[1:10,1:10])
print(table$col_names[1:10])
```

`ICBioMark` *ICBioMark: A package for cost-effective design of gene panels to predict exome-wide biomarkers.*

Description

This package implements the methodology proposed in 'Data-driven design of targeted gene panels for estimating immunotherapy biomarkers', (Bradley and Cannings, 2021, preprint). It allows the user to fit generative models of mutation from an annotated mutation dataset, and then further to produce tunable linear estimators of exome-wide biomarkers. It also contains functions to simulate mutation annotated format (MAF) data, as well as to analyse the output and performance of models.

`nsclc_maf` *Non-Small Cell Lung Cancer MAF Data*

Description

A pre-loaded mutation dataset from Campbell et. al (2016), downloaded from The Cancer Genome Atlas.

Usage

```
nsclc_maf
```

Format

An annotated mutation dataframe with 6 columns and 299855 rows:

Tumor_Sample_Barcode A sample id for each mutation.

Hugo_Symbol The name of the gene location for each mutation.

Variant_Classification The mutation type for each mutation.

Chromosome Chromosome on which the mutation occurred.

Start_Position Start nucleotide location for mutation.

End_Position End nucleotide location for mutation.

Source

https://www.cbioportal.org/study/summary?id=nsclc_tcga_broad_2016

---

nsclc_survival *Non-Small Cell Lung Cancer Survival and Clinical Data*

Description

A pre-loaded clinical dataset containing survival and clinical data from Campbell et. al (2016), downloaded from The Cancer Genome Atlas.

Usage

```
nsclc_survival
```

Format

An annotated mutation dataframe with 23 columns and 1144 rows. Each row corresponds to a sample, and details clinical and survival information about the patient from whom the sample was derived. Its columns are as follows:

CASE_ID

AGE

AGE_AT_SURGERY

CANCER_TYPE

CANCER_TYPE_DETAILED

DAYS_TO_DEATH

DAYS_TO_LAST_FOLLOWUP

FRACTION_GENOME_ALTERED

HISTORY_NEOADJUVANT_TRTYN

HISTORY_OTHER_MALIGNANCY

MUTATION_COUNT

M_STAGE

N_STAGE

ONCOTREE_CODE

OS_MONTHS

OS_STATUS

SAMPLE_COUNT

SEX

SMOKING_HISTORY

SMOKING_PACK_YEARS

SOMATIC_STATUS

STAGE

T_STAGE

Source

https://www.cbioportal.org/study/clinicalData?id=nsclc_tcga_broad_2016

`pred_first_fit` *First-Fit Predicitve Model with Group Lasso*

Description

This function implements the first-fit procedure described in Bradley and Cannings, 2021. It requires at least a generative model and a dataframe containing gene lengths as input.

Usage

```
pred_first_fit(
  gen_model,
  lambda = exp(seq(-16, -24, length.out = 100)),
  biomarker = "TMB",
  marker_mut_types = c("NS", "I"),
  training_matrix,
  gene_lengths,
  marker_training_values = NULL,
  K_method = max,
  free_genes = c()
)
```

Arguments

gen_model	(list) A generative mutation model, fitted by fit_gen_model().
lambda	(numeric) A vector of penalisation weights for input to the group lasso optimiser gglasso.
biomarker	(character) The biomarker in question. If "TMB" or "TIB", then automatically defines the subsequent variable marker_mut_types.
marker_mut_types	(character) The set of mutation type groupings constituting the biomarker being estimated. Should be a vector comprising of elements of the mut_types_list vector in the 'names' attribute of gen_model.
training_matrix	(sparse matrix) A sparse matrix of mutations in the training dataset, produced by get_mutation_tables().
gene_lengths	(dataframe) A table with two columns: Hugo_Symbol and max_cds, providing the lengths of the genes to be modelled.
marker_training_values	(dataframe) A dataframe containing two columns: 'Tumor_Sample_Barcode', containing the sample IDs for the training dataset, and a second column containing training values for the biomarker in question.
K_method	(function) How to select a representative biomarker value from the training dataset. Defaults to max().
free_genes	(character) Which genes should escape penalisation (for example when augmenting a pre-existing panel).

Value

A list of six elements:

- fit: Output of call to gglasso.
- panel_genes: A matrix where each row corresponds to a gene, each column to an iteration of the group lasso with a different penalty factor, and the elements booleans specifying whether that gene was selected to be included in that iteration.
- panel_lengths: A vector giving total panel length for each gglasso iteration.
- p: The vector of weights used in the optimisation procedure.
- K: The bias penalty factor used in the optimisation procedure.
- names: Gene and mutation type information as used when fitting the generative model.

Examples

```
example_first_fit <- pred_first_fit(example_gen_model, lambda = exp(seq(-9, -14, length.out = 100)),
                                    training_matrix = example_tables$train$matrix,
                                    gene_lengths = example_maf_data$gene_lengths)
```

pred_intervals *Produce Error Bounds for Predictions*

Description

A function to produce a confidence region for a linear predictor. In upcoming versions will (hopefully) be greatly simplified.

Usage

```
pred_intervals(
  predictions,
  pred_model,
  gen_model,
  training_matrix,
  gene_lengths,
  biomarker_values,
  alpha = 0.1,
  range_factor = 1.1,
  s = NULL,
  max_panel_length = NULL,
  biomarker = "TMB",
  marker_mut_types = c("NS", "I"),
  model = "Refitted T"
)
```

Arguments

predictions	(list) A predictions object, as produced by get_predictions().
pred_model	(list) A predictive model, as produced by pred_first_fit(), pred_refit_panel() or pred_refit_range().
gen_model	(list) A generative model, as produce by fit_gen_model
training_matrix	(sparse matrix) A training matrix, as produced by get_tables()$matrix or get_table_from_maf()$matrix.
gene_lengths	(data frame) A data frame with columns 'Hugo_Symbol' and 'max_cds'. See example_maf_data$gene_lengths, or ensembl_gene_lengths for examples.
biomarker_values	(data frame) A data frame containing the true values of the biomarker in question.
alpha	(numeric) Confidence level for error bounds.
range_factor	(numeric) Value specifying how far beyond the range of max(biomarker) to plot confidence region.
s	(numeric) If input predictions are for a range of panels, s chooses which panel (column in a pred_fit object) to produce predictions for.

max_panel_length
: (numeric) Select panel by maximum length.

biomarker
: (character) Which biomarker is being predicted.

marker_mut_types
: (character) If biomarker is not one of "TMB" or "TIB", then this is required to specify which mutation type groups constitute the biomarker.

model
: (character) The model (must be based on a linear estimator) for which prediction intervals are being generated.

Value

A list with two entries:

- prediction_intervals:
- confidence_region:

Examples

```
example_intervals <- pred_intervals(predictions = get_predictions(example_refit_range,
               new_data = example_tables$val),
             pred_model = example_refit_range, biomarker_values = example_tmb_tables$val,
            gen_model = example_gen_model, training_matrix = example_tables$train$matrix,
               max_panel_length = 15000, gene_lengths = example_maf_data$gene_lengths)

example_confidence_plot <- ggplot2::ggplot() +
  ggplot2::geom_point(data = example_intervals$prediction_intervals,
              ggplot2::aes(x = true_value, y = estimated_value)) +
        ggplot2::geom_ribbon(data = example_intervals$confidence_region,
          ggplot2::aes(x = x, ymin = y_lower, ymax = y_upper),
                    fill = "red", alpha = 0.2) +
        ggplot2::geom_line(data = example_intervals$confidence_region,
          ggplot2::aes(x = x, y = y), linetype = 2) +
        ggplot2::scale_x_log10() + ggplot2::scale_y_log10()

plot(example_confidence_plot)
```

---

pred_refit_panel *Refitted Predictive Model for a Given Panel*

Description

A function taking the output of a call to pred_first_fit(), as well as gene length information, and a specified panel (list of genes), and producing a refitted predictive model on that given panel.

Usage

```
pred_refit_panel(
  pred_first = NULL,
  gene_lengths = NULL,
  model = "T",
  genes,
  biomarker = "TMB",
  marker_mut_types = c("NS", "I"),
  training_data = NULL,
  training_values = NULL,
  mutation_vector = NULL,
  t_s = NULL
)
```

Arguments

pred_first	(list) A first-fit predictive model as produced by pred_first_fit().
gene_lengths	(dataframe) A dataframe of gene lengths (see example_maf_data$gene_lengths for format).
model	(character) A choice of "T", "OLM" or "Count" specifying how predictions should be made.
genes	(character) A vector of gene names detailing the panel being used.
biomarker	(character) If "TMB" or "TIB", automatically defines marker_mut_types, otherwise this will need to be specified separately.
marker_mut_types	(character) A vector specifying which mutation types groups determine the biomarker in question.
training_data	(list) Training data, as produced by get_mutation_tables() (select train, val or test).
training_values	(dataframe) Training true values, as produced by get_biomarker_tables() (select train, val or test).
mutation_vector	(numeric) Optional vector specifying the values of the training matrix (training_data$matrix) in vector rather than matrix form.
t_s	(numeric) Optional vector specifying the frequencies of different mutation types.

Value

A list with three elements:

- fit, a list including a sparse matrix 'beta' giving prediction weights.
- panel_genes, a sparse (logical) matrix giving the genes included in prediction.
- panel_lengths, a singleton vector giving the length of the panel used.

Examples

```
example_refit_panel <- pred_refit_panel(pred_first = example_first_pred_tmb,
  gene_lengths = example_maf_data$gene_lengths, genes = paste0("GENE_", 1:10))
```

`pred_refit_range` *Get Refitted Predictive Models for a First-Fit Range of Panels*

Description

A function producing a refitted predictive model for each panel produced by usage of the function pred_first_fit(), by repeatedly applying the function pred_refit_panel().

Usage

```
pred_refit_range(
  pred_first = NULL,
  gene_lengths = NULL,
  model = "T",
  biomarker = "TMB",
  marker_mut_types = c("NS", "I"),
  training_data = NULL,
  training_values = NULL,
  mutation_vector = NULL,
  t_s = NULL,
  max_panel_length = NULL
)
```

Arguments

`pred_first`	(list) A first-fit predictive model as produced by pred_first_fit().
`gene_lengths`	(dataframe) A dataframe of gene lengths (see example_maf_data$gene_lengths for format).
`model`	(character) A choice of "T", "OLM" or "Count" specifying how predictions should be made.
`biomarker`	(character) If "TMB" or "TIB", automatically defines marker_mut_types, otherwise this will need to be specified separately.
`marker_mut_types`	(character) A vector specifying which mutation types groups determine the biomarker in question.
`training_data`	(sparse matrix) Training matrix, as produced by get_mutation_tables() (select train, val or test).
`training_values`	(dataframe) Training true values, as produced by get_biomarker_tables() (select train, val or test).

mutation_vector
: (numeric) Optional vector specifying the values of the training matrix (training_data$matrix) in vector rather than matrix form.

t_s
: (numeric) Optional vector specifying the frequencies of different mutation types.

max_panel_length
: (numeric) Upper bound for panels to fit refitted models to. Most useful for "OLM" and "Count" model types.

Value

A list with three elements:

- fit, a list including a sparse matrix 'beta' giving prediction weights for each first-fit panel (one panel per column).
- panel_genes, a sparse (logical) matrix giving the genes included in prediction for each first-fit panel.
- panel_lengths, a vector giving the length of each first-fit panel.

Examples

```
example_refit_range <- pred_refit_range(pred_first = example_first_pred_tmb,
  gene_lengths = example_maf_data$gene_lengths)
```

---

vis_model_fit *Visualise Generative Model Fit*

Description

A function to visualise how well a general model has fitted to a mutation dataset across cross-validation folds. Designed to produce a similar output to glmnet's function plot.cv.glmnet.

Usage

```
vis_model_fit(
  gen_model,
  x_sparsity = FALSE,
  y_sparsity = FALSE,
  mut_type = NULL
)
```

Arguments

gen_model	(list) A generative model fitted by fit_gen_model()
x_sparsity	Show model sparsity on x axis rather than lambda.
y_sparsity	Show model sparsity on y axis rather than deviance.
mut_type	Produce separate plots for each mutation type.

Value

Summary plot of the generative model fit across folds.

Examples

```
p <- vis_model_fit(example_gen_model)
```

Index