Package ‘FIT’

January 7, 2019

Title Transcriptomic Dynamics Models in Field Conditions
Version 0.0.6

Depends R (>= 3.2.2)
Imports methods, Rcpp (>= 0.11.2), XML, gglasso (>= 1.4), MASS
License MPL (>= 2) | file LICENSE
LazyData true
LinkingTo Rcpp, RcppEigen (>= 0.3.2.1.2)
SystemRequirements C++11
NeedsCompilation yes
RoxygenNote 6.0.1
Author Koji Iwayama [cre], Yuri Aisaka [aut]
Maintainer Koji Iwayama <fieldtranscriptome@gmail.com>
Suggests knitr, markdown
VignetteBuilder knitr
Repository CRAN
Date/Publication 2019-01-07 14:50:15 UTC

R topics documented:

convert.attribute .................................................. 2
convert.expression ............................................... 3
convert.weather .................................................. 3
**convert.attribute**

Converts attribute data from a dataframe into an object.

**Description**

Converts attribute data from a dataframe into an object.

**Usage**

```r
convert.attribute(data, sample = NULL)
```

**Arguments**

- `data` A dataframe of the attributes of microarray/RNA-seq data.
- `sample` An optional numeric array that designates the samples, that is rows, of the dataframe to be loaded.

**Value**

An object that represents the attributes of microarray/RNA-seq data. Internally, the object holds a dataframe whose number of entries (rows) equals that of the samples.
**convert.expression**

*converts expression data from a dataframe into an object.*

---

**Description**

converts expression data from a dataframe into an object.

**Usage**

```r
convert.expression(data, entries = NULL)
```

**Arguments**

- **data**: A dataframe of expression data to be loaded.
- **entries**: An optional string array that designates the entries of the dataframe to be loaded.

**Value**

An object that represents the expression data of microarray/RNA-seq. Internally, the object holds a matrix of size \( n_{samples} \times n_{genes} \).

---

**convert.weather**

*Converts weather data from a dataframe into an object.*

---

**Description**

Converts weather data from a dataframe into an object.

**Usage**

```r
convert.weather(data, entries = IO\$weather.entries)
```

**Arguments**

- **data**: A dataframe of weather data to be converted.
- **entries**: An optional string array that designates the entries of the dataframe to be loaded.

**Value**

An object that represents the timeseries data of weather factors. Internally, the object holds a dataframe of size \( n_{timepoints} \times n_{factors} \).
convert.weight  
*Converts regression weight data from a dataframe into an object.*

**Description**

Converts regression weight data from a dataframe into an object.

**Usage**

```r
convert.weight(data, entries = NULL)
```

**Arguments**

- `data`: A dataframe that contains weight data to be loaded.
- `entries`: An optional string array that designates the entries of the dataframe to be loaded.

**Value**

An object that represents the weights. Internally, the object holds a matrix of size `nsamples * ngenes`.

---

**FIT**  
*FIT: a statistical modeling tool for transcriptome dynamics under fluctuating field conditions*

**Description**


**Overview**

The FIT package is an R implementation of a class of transcriptomic models that relates gene expressions of plants and weather conditions to which the plants are exposed. (The reader is referred to [Nagano et al.] for the detail of the class of models concerned.)

By providing (a) gene expression profiles of plants brought up in a field condition, and (b) the relevant weather history (temperature etc.) of the said field, the user of the package is able to (1) construct optimized models (one for each gene) for their expressions, and (2) use them to predict the expressions for another weather history (possibly in a different field).

Below, we briefly explain the construction of the optimized models (“training phase”) and the way to use them to make predictions (“prediction phase”).
Model training phase:
The model of [Nagano et al.] belongs to the class of statistical models called “linear models” and are specified by a set of “parameters” and “(linear regression) coefficients”. The former are used to convert weather conditions to the “input variables” for a regression, and the latter are then multiplied to the input variables to form the expectation values for the gene expressions. The reader is referred to the original article [Nagano et al.] for the formulas for the input variables. (See also [Iwayama] for a review.)
The training phase consists of three stages:
1. Init: fixes the initial model parameters
2. Optim: optimizes the model parameters
3. Fit: fixes the linear regression coefficients
The user can configure the training phase through a custom data structure (“recipe”), which can be constructed by using the utility function FIT::make.recipe(). The role of the first stage Init is to fix the initial values for the model parameters from which the parameter optimization is performed. At the moment two methods, ‘manual’ and ‘gridsearch’, are implemented. With the ‘manual’ method the user can simply specify the set of initial values that he thinks is promising. For the ‘gridsearch’ method the user discretizes the parameter space to a grid by providing a finite number of candidate values for each parameter. FIT then performs a search over the grid for the “best” combinations of the initial parameters.
The second stage Optim is the main step of the model training, and FIT tries to gradually improve the model parameters using the Nelder-Mead method.
This stage could be run one or more times where each can be run using the method ‘none’, ‘lm’ or ‘lasso’. The ‘none’ method passes the given parameter as-is to the next method in the Optim pipeline or to the next stage Fit. (Basically, the method is there so that the user can skip the entire Optim stage, but the method could be used for slightly warming-up the CPU as well.) The ‘lm’ method uses the a simple (weighted) linear regression to guide the parameter optimization. That is, FIT first computes the “input variables” from the current parameters and associated weather data, and then finds the set of linear coefficients that best explains the “output variables” (gene expressions). Finally, the quadratic residual is used as the measure for the error and is fed back to the Nelder-Mead method.
The ‘lasso’ method is similar to the ‘lm’ method but uses the (weighted) Lasso regression (“linear” regression with an L1-regularization for the regression coefficients) instead of the simple linear regression. FIT uses the glmnet package to perform the Lasso regression and the strength of the L1-regularization is fixed via a cross validation. (See cv.glmnet() from the glmnet package. The Lasso regression is said to suppress irrelevant input variables automatically and tends to create models with better prediction ability. On the other hand, ‘lasso’ runs considerably slower than ‘lm’.
For example, passing a vector c('lm', 'lasso') to the argument optim (of make.recipe()) creates a recipe that instructs the Optim stage to (1) first optimize using the ‘lm’ method, (2) and then fine tunes the parameters using the ‘lasso’ method.
After fixing the model parameters in the Optim stage, the Fit stage can be used to fix the linear coefficients of the models. Here, either ‘fit.lm’ or ‘fit.lasso’ can be used to find the “best” coefficients, the main difference being that the coefficients are penalized by an L1-norm for the latter. Note that it is perfectly okay to use ‘fit.lasso’ for the parameters optimized using ‘lm’.
In order to prepare for the possibly huge variations of expression data as measured by RNA-seq, FIT provides a way to weight regression penalties from each sample with different weights as in sum_{s in samples} (weight_s) (error_s)^2.
**Prediction phase:** For each gene, the trained model of the previous subsection can be thought of as a black box that maps the field conditions (weather data), to which a plant containing the gene is exposed, to its expected expression. **FIT** provides a simple function `FIT::predict()` that does just this. `FIT::predict()` takes as its argument a list of pretrained models as well as actual/hypothetical plant sample attributes and weather data, and returns the predicted values of gene expressions. When there is a set of actually measured expressions, an associated function `FIT::prediction.errors()` can be used to check the validity of the predictions made by the models.

**Namespce contamination**

The **FIT** package exports fairly ubiquitous names such as `optim`, `predict` etc\.* as its API. Users, therefore, are advised to load **FIT** via `requireNamespace('FIT')` and use its API function with a namaspace qualifier (e.g.-`FIT::optim()`) rather than loading and attaching it via `library('FIT')`.

**Basic usage**

See vignettes for examples of actual scripts that use **FIT**.

**References**


**Examples**

```r
## not run:
# the following snippet shows the structure of a typical
# driver script of the FIT package.
# See vignettes for examples of actual scripts that use FIT.

###################
## training ##
###################
## discretized parameter space (for 'gridsearch')
grid.coords <- list(
  clock.phase = seq(0, 23*60, 1*60),
  gate.radiation.amplitude = c(-5, 5)
)

## create a training recipe
recipe <- FIT::make.recipe(c('temperature', 'radiation'),
  init = 'gridsearch',
  init.data = grid.coords,
  optim = c('lm'),
  fit = 'fit.lasso',
  time.step = 10,
  opts =
)```

*FIT*
list(lm = list(maxit = 900),
lasso = list(maxit = 1000))

## names of genes to construct models
genes <- c('Os12g0189300', 'Os02g0724000')

## End(Not run)

## Not run:
## load training data
training.attribute <- fit::load.attribute('attribute.2008.txt')
training.weather <- fit::load.weather('weather.2008.dat', 'weather')
training.expression <- fit::load.expression('expression.2008.dat', 'ex', genes)

## models will be a list of trained models (length: ngenes)
models <- fit::train(training.expression,
  training.attribute,
  training.weather,
  recipe)

## End(Not run)

################
## prediction ##
################

## Not run:
## load validation data
prediction.attribute <- fit::load.attribute('attribute.2009.txt');
prediction.weather <- fit::load.weather('weather.2009.dat', 'weather')
prediction.expression <- fit::load.expression('expression.2009.dat', 'ex', genes)

## predict
prediction.result <- fit::predict(models[[1]],
  prediction.attribute,
  prediction.weather)

## End(Not run)
Description

Note: use train() unless the user is willing to accept breaking API changes in the future.

Usage

init(expression, weight, attribute, weather, recipe)

Arguments

expression  An object that represents gene expression data. The object can be created from a dumped/saved dataframe of size $n_{samples} \times n_{genes}$ using FIT::load.expression(). (At the moment it is an instance of a hidden class IOSAttribute, but this may be subject to change.)

weight  A matrix of size $n_{samples} \times n_{genes}$ that during regression penalizes errors from each sample using the formula $\sum_{s \text{ in samples}} (weight_s) (error_s)^2$. Note that, unlike for FIT::train(), this argument is NOT optional.

attribute  An object that represents the attributes of microarray/RNA-seq data. The object can be created from a dumped/saved dataframe of size $n_{samples} \times n_{attributes}$ using FIT::load.attribute(). (At the moment it is an instance of a hidden class IOSAttribute, but this may be subject to change.)

weather  An object that represents actual or hypothetical weather data with which the training of models are done. The object can be created from a dumped/saved dataframe of size $n_{timepoints} \times n_{factors}$ using FIT::load.weather(). (At the moment it is an instance of a hidden class IOSWeather, but this may be subject to change.)

recipe  An object that represents the training protocol of models. A recipe can be created using FIT::make.recipe().

models  A collection of models being trained as is returned by FIT::optim().

Value

A collection of models whose parameters and regression coefficients are optimized.

init

A raw API for initializing model parameters.

Description

Note: use train() unless the user is willing to accept breaking API changes in the future.

Usage

init(expression, weight, attribute, weather, recipe)
**load.attribute**

**Arguments**

- **expression**: An object that represents gene expression data. The object can be created from a dumped/saved dataframe of size `nSamples * nGenes` using `FIT::load.expression()`. (At the moment it is an instance of a hidden class IOSAttribute, but this may be subject to change.)

- **weight**: A matrix of size `nSamples * nGenes` that during regression penalizes errors from each sample using the formula `\sum_{s in samples} (weight_s) (error_s)^2`. Note that, unlike for `FIT::train()`, this argument is NOT optional.

- **attribute**: An object that represents the attributes of a microarray/RNA-seq data. The object can be created from a dumped/saved dataframe of size `nSamples * nAttributes` using `FIT::load.attribute()`. (At the moment it is an instance of a hidden class IOSAttribute, but this may be subject to change.)

- **weather**: An object that represents actual or hypothetical weather data with which the training of models are done. The object can be created from a dumped/saved dataframe of size `nTimepoints * nFactors` using `FIT::load.weather()`. (At the moment it is an instance of a hidden class IOSWeather, but this may be subject to change.)

- **recipe**: An object that represents the training protocol of models. A recipe can be created using `FIT::make.recipe()`.

**Value**

A collection of models whose parameters are set by using the 'init' method in the argument `recipe`.

---

**load.attribute**      Loads attribute data.

---

**Description**

Loads attribute data.

**Usage**

```
load.attribute(path, variable = NULL, sample = NULL)
```

**Arguments**

- **path**: A path of a file that contains attribute data to be loaded. When the file is a loadable `.Rdata`, name of the dataframe object in the `.Rdata` (that actually contains the relevant data) has to be specified as well.

- **variable**: An optional string that designates the name of a dataframe object that has been saved in an `.Rdata`. (See the description of `path`.)

- **sample**: An optional numeric array that designates the samples, that is rows, of the dataframe to be loaded.
load.expression

**Description**

Loads expression data.

**Usage**

load.expression(path, variable = NULL, entries = NULL)

**Arguments**

- `path`: A path of a file that contains attribute data to be loaded. When the file is a loadable .Rdata, name of the dataframe object in the .Rdata (that actually contains the relevant data) has to be specified as well.
- `variable`: An optional string that designates the name of a dataframe object that has been saved in an .Rdata. (See the description of `path`.)
- `entries`: An optional string array that designates the entries of the dataframe to be loaded.

**Value**

An object that represents the expression data of microarray/RNA-seq. Internally, the object holds a matrix of size `nsamples * ngenes`.

load.weather

**Description**

Loads weather data.

**Usage**

load.weather(path, variable = NULL, entries = IO$weather.entries)
**load.weight**

**Arguments**

- **path**: A path of a file that contains weather data to be loaded. When the file is a loadable `.Rdata`, name of the dataframe object in the `.Rdata` (that actually contains the relevant data) has to be specified as well.
- **variable**: An optional string that designates the name of a dataframe object that has been saved in an `.Rdata`. (See the description of path.)
- **entries**: An optional string array that designates the entries of the dataframe to be loaded.

**Value**

An object that represents the timeseries data of weather factors. Internally, the object holds a dataframe of size \( \text{ntimepoints} \times \text{nfacators} \).

---

**load.weight**

*Loads regression weight data.*

**Description**

Loads regression weight data.

**Usage**

```r
load.weight(path, variable = NULL, entries = NULL)
```

**Arguments**

- **path**: A path of a file that contains weight data to be loaded. When the file is a loadable `.Rdata`, name of the dataframe object in the `.Rdata` (that actually contains the relevant data) has to be specified as well.
- **variable**: An optional string that designates the name of a dataframe object that has been saved in an `.Rdata`. (See the description of path.)
- **entries**: An optional string array that designates the entries of the dataframe to be loaded.

**Value**

An object that represents the weights. Internally, the object holds a matrix of size \( \text{nsamples} \times \text{ngenes} \).
make.recipe

Creates a recipe for training models.

Description

Creates a recipe for training models.

Usage

make.recipe(envs, init, optim, fit, init.data, time.step, gate.open.min = 0, opts = NULL)

Arguments

envs  
An array of weather factors to be taken into account during the construction of models. At the moment, the array envs can only contain a single weather factor from weather.entries, though there is a plan to remove the restriction in a future version.

init  
A string to specify the method to choose the initial parameters. (One of 'gridsearch' or 'manual'.)

optim  
A string to specify the method to be used for optimizing the model parameters. (One of 'none', 'lm' or 'lasso'.)

fit  
A string to specify the method to be used for fixing the linear regression coefficients. (One of 'fit.lm' or 'fit.lasso'.)

init.data  
Auxiliary data needed to perform the Init stage using the method specified by the init argument. When init is 'gridsearch', it should be a list representing a discretized parameter space. When init is 'manual', it should be a list of parameter values that is used as the initial values for the parameters in the Optim stage.

time.step  
An integer to specify the basic unit of time (in minute) for the transcriptomic models. Must be a multiple of the time step of weather data.

gate.open.min  
The minimum opening length in minutes of the gate function for environmental inputs.

opts  
An optional named list that specifies the arguments to be passed to methods that constitute each stage of the model training. Each key of the list corresponds to a name of a method.

See examples for the supported options.

Value

An object representing the procedure to construct models.
Examples

```R
## not run:
initNparams <- NN # choose them wisely
# Defined in Train.R:
# defaultNopts <- list(
# none = list(),
# lm = list(maxit=1500, nfolds=-1), # nfolds for lm is simply ignored
# lasso = list(maxit=1000, nfolds=10)
# )
recipe <- fit::makeNrecipe(c("wind", "temperature"),
                           init = 'manual',
                           init.data = initNparams,
                           optim = c('lm', 'none', 'lasso'),
                           fit = 'fit.lasso',
                           time.step = 10,
                           opts =
                           list(lm = list(maxit = 900),
                                lasso = list(maxit = 1000)))

## end(not run)
```

### Description
Makes trivial weight data

### Usage

```R
make.trivial.weights(samplesNnL genes)
```

### Arguments

- `samplesNn` A number of samples.
- `genes` A list of genes.

### Value
An object that represents the trivial weights. Internally, the object holds an identity matrix of size nsamples * ngenes.
A raw API for optimizing model parameters.

**Description**

Note: use `train()` unless the user is willing to accept breaking API changes in the future.

**Usage**

```r
optim(expression, weight, attribute, weather, recipe, models, maxit = NULL, nfolds = NULL)
```

**Arguments**

- `expression` An object that represents gene expression data. The object can be created from a dumped/saved dataframe of size `nsamples * ngenes` using `FIT::load.expression()`. (At the moment it is an instance of a hidden class `IOSAttribute`, but this may be subject to change.)
- `weight` A matrix of size `nsamples * ngenes` that during regression penalizes errors from each sample using the formula `sum_{s in samples} (weight_s) (error_s)^2`. Note that, unlike for `FIT::train()`, this argument is NOT optional.
- `attribute` An object that represents the attributes of microarray/RNA-seq data. The object can be created from a dumped/saved dataframe of size `nsamples * nattributes` using `FIT::load.attribute()`. (At the moment it is an instance of a hidden class `IOSAttribute`, but this may be subject to change.)
- `weather` An object that represents actual or hypothetical weather data with which the training of models are done. The object can be created from a dumped/saved dataframe of size `ntimepoints * nfactors` using `FIT::load.weather()`. (At the moment it is an instance of a hidden class `IOSWeather`, but this may be subject to change.)
- `recipe` An object that represents the training protocol of models. A recipe can be created using `FIT::make.recipe()`.
- `models` A collection of models being trained as is returned by `FIT::init()`. At this moment, it must be a list (genes) of a list (envs) of models and must contain at least one model. (THIS MIGHT CHANGE IN A FUTURE.)
- `maxit` An optional number that specifies the maximal number of times that the parameter optimization is performed. The user can control this parameter by using the `opts` argument for `FIT::train()`.
- `nfolds` An optional number that specifies the order of cross validation when `optim` method is 'lasso'. This is simply ignored when `optim` method is 'lm'.

**Value**

A collection of models whose parameters are optimized by using the 'optim' pipeline in the argument `recipe`. 
predict

Predicts gene expressions using pretrained models.

Description

Predicts gene expressions using pretrained models.

Usage

predict(models, attribute, weather)

Arguments

models A list of trained models for the genes of interest. At the moment the collection of trained models returned by FIT::train() cannot be directly passed to FIT::predict(): the user has to explicitly convert it to an appropriate format by using FIT::train.to.predict.adaptor(). (This restriction might be removed in a future.)

attribute An object that represents the attributes of microarray/RNA-seq data. The object can be created from a dumped/saved dataframe of size nsamples * nattributes using FIT::load.attribute(). (At the moment it is an instance of a hidden class IOSAttribute, but this may be subject to change.)

weather An object that represents actual or hypothetical weather data with which predictions of gene expressions are made. The object can be created from a dumped/saved dataframe of size ntimepoints * nfactors using FIT::load.weather(). (At the moment it is an instance of a hidden class IOSWeather, but this may be subject to change.)

Value

A list of prediction results as returned by the models.

Examples

## Not run:
# prepare models
# NOTE: FIT::train() returns a nested list of models
# so we have to flatten it using FIT::train.to.predict.adaptor()
# before passing it to FIT::predict().
models <- FIT::train(.)
models.flattened <- FIT::train.to.predict.adaptor(models)

# load data used for prediction
prediction.attribute <- FIT::load.attribute('attribute.2009.txt')
prediction.weather <- FIT::load.weather('weather.2009.dat', 'weather')
prediction.expression <- FIT::load.expression('expression.2009.dat', 'ex', genes)

prediction.results <- FIT::predict(models.flattened,
## prediction.errors

### Description

Computes the prediction errors using the trained models.

### Usage

```
prediction.errors(models, expression, attribute, weather)
```

### Arguments

- **models**: A list of trained models for the genes of interest. At the moment the collection of trained models returned by `FIT::train()` cannot be directly passed to `FIT::predict()`: the user has to explicitly convert it to an appropriate format by using `FIT::train.to.predict.adaptor()`. (This restriction might be removed in a future.)
- **expression**: An object that represents the actual measured data of gene expressions. The object can be created from a dumped/saved dataframe of size `nsamples * ngenes` using `FIT::load.expression()`. (At the moment it is an instance of a hidden class `IO$Attribute`, but this may be subject to change.)
- **attribute**: An object that represents the attributes of microarray/RNA-seq data. The object can be created from a dumped/saved dataframe using `FIT::load.attribute()`. (At the moment it is an instance of a hidden class `IO$Attribute`, but this may be subject to change.)
- **weather**: An object that represents actual or hypothetical weather data with which predictions of gene expressions are made. The object can be created from a dumped/saved dataframe using `FIT::load.weather()`. (At the moment it is an instance of a hidden class `IO$Weather`, but this may be subject to change.)

### Value

A list of deviance (a measure of validity of predictions, as is defined by each model) between the prediction results and the measured results (as is provided by the user through `expression` argument).

### Examples

```
## Not run:
# see the usage of FIT::predict()

## End(Not run)
```
**train**

Constructs models following a recipe.

**Description**

Constructs models following a recipe.

**Usage**

```r
train(expression, attribute, weather, recipe, weight = NULL,
      min.expressed.rate = 0.01)
```

**Arguments**

- **expression**: An object that represents gene expression data. The object can be created from a dumped/saved dataframe of size `nsamples * ngenes` using `fit::load.expression()`. (At the moment it is an instance of a hidden class `IO$Expression`, but this may be subject to change.)

- **attribute**: An object that represents the attributes of microarray/RNA-seq data. The object can be created from a dumped/saved dataframe of size `nsamples * nattributes` using `fit::load.attribute()`. (At the moment it is an instance of a hidden class `IO$Attribute`, but this may be subject to change.)

- **weather**: An object that represents actual or hypothetical weather data with which the training of models are done. The object can be created from a dumped/saved dataframe of size `ntimepoints * nfactors` using `fit::load.weather()`. (At the moment it is an instance of a hidden class `IO$Weather`, but this may be subject to change.)

- **recipe**: An object that represents the training protocol of models. A recipe can be created using `fit::make.recipe()`.

- **weight**: An optional numerical matrix of size `nsamples * ngenes` that during regression penalizes errors from each sample using the formula `\sum_{s \in \text{samples}} (weight_s \cdot (error_s)^2)`. This argument is optional for a historical reason, and when it is omitted, all samples are equally penalized.

- **min.expressed.rate**: A number used to A gene with `\text{var(expr)} < \text{thres.expr}` is regarded as unexpressed, and `FIT` sets its model as: `expr = \log(\text{offset}) + 0*\text{inputs}`.

**Value**

A collection of trained models.
Examples

```r
## Not run:
# create recipe
recipe <- FIT::make.recipe(..)

# load training data
training.attribute <- FIT::load.attribute('attribute.2008.txt');
training.weather <- FIT::load.weather('weather.2008.dat', 'weather')
training.expression <- FIT::load.expression('expression.2008.dat', 'ex', genes)
training.weight <- FIT::load.weight('weight.2008.dat', 'weight', genes)

# train models
models <- FIT::train(training.expression,
                    training.attribute,
                    training.weather,
                    recipe,
                    training.weight)

## End(Not run)
```

---

**weather.entries**

**Supported weather factors.**

### Description

Supported weather factors.

### Usage

```r
weather.entries
```

### Format

An object of class character of length 6.

### Examples

```r
length(FIT::weather.entries)
```
Index

*Topic datasets
  weather.entries, 18

convert.attribute, 2
convert.expression, 3
convert.weather, 3
convert.weight, 4

FIT, 4
FIT-package (FIT), 4
fit.models, 7

init, 8

load.attribute, 9
load.expression, 10
load.weather, 10
load.weight, 11

make.recipe, 12
make.trivial.weights, 13

optim, 14

predict, 15
prediction.errors, 16

train, 17

weather.entries, 18