Package ‘pkgFireCARES’

June 10, 2020

Title Useful Functions for FireCARES

Version 1.0.0

Description This package provides some useful functions for estimating community risk models and community risk scores in FireCARES.

Depends R (>= 3.4.0)

Imports acs,
boot,
glmnet,
ranger,
RPostgreSQL,
utils,
cluster,
magrittr

Suggests doParallel

License unlimited

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

R topics documented:

`acs.dwnld` *Download ACS data*

Description

Downloads the ACS data needed for use in model estimation.

Usage

```
acs.dwnld(conn, year, cols = NULL, states = NULL)
```

Arguments

`conn`	DBI Connection. The connection to the database where the data will be uploaded.
`year`	Integer. The year for which to download ACS data
`cols`	Character vector. Vector of column names (from the ACS) reflecting what ACS columns to download. [optional]
`states`	Character vector. This allows me to specify a subset of states. If it is not specified, all nationwide data is downloaded. [optional]

Details

This leaves a ’temporary’ table on the database. That table will then need to be manually inserted into the main acs_est table.

Value

returns a list with the following entries:

table.name.est Name of the table on the database in which the new estimates are stored

table.name.err Name of the table on the database in which the new error values are stored

rows Number of rows added to the data set

elapsed.time Time it took to complete the download

c_test *Merge multiple fcTest objects*

Description

Merges multiple fcTest objects

Usage

```
c_test(t1, ...)
```

Arguments

t1	test object.
...	Additional test objects

Value

Returns a test object that contains all the information in the separate test objects supplied.

Examples

```
## Not run:
  c_test(test.f.L1, test.f.L2)

## End(Not run)
```

fcCluster *Clusters the counties*

Description

This function generates the county clusters for use by any estimation routine that uses them.

Usage

```
fcCluster(conn, filter = 1000, clusters = 9)
```

Arguments

conn	A DBI Connection. This is a connection to the database containing the data and model definitions. If none is entered, default connection information is obtained from the operating system environment.
filter=1000	Numeric. If the number of missing fields in a column is greater than `filter`, that column is excluded from the analysis. If `filter` is NULL, then no columns are excluded.
clusters=9	The number of clusters to use.

Details

The main purpose of this routine is to update the `nist.county_clusters` table in the FireCARES database. It does the following tasks.

- Load needed data from the FireCARES database.
- Filter out the offshore territories and any columns with too few records to be effectively usable.
- Cluster the data using the cluster package
- Upload the county clustering results back to FireCARES.

Value

Invisibly returns the revised cluster table used by later queries.

`fcEstimate` *Predict expected values*

Description

Predicts expected values based on fitted models and new data.

Usage

```
fcEstimate(input, output, new.data, subset = TRUE)
```

Arguments

`input`	character vector. This lists the names of the control objects used to generate the model(s) used for prediction.
`output`	character vector. This lists the names of the output models created from the control objects listed in 'input'.
`new.data`	data.frame This is the data that will be used to generate the predictions. It needs to contain all the input variables used in creating the model.
`subset`	name. This specifies the subset of new.data for which predictions will be made. If, as I expect, you want predictions for all rows in the new.data object, then the default value (subset=TRUE) will provide that. There is no need to screen out rows with undefined input variables since this routine already takes that into account.

Details

This function produces tract-level estimates of the modeled information based on models run and output in the "output" object, and using the data in "new.data".

Value

Returns a data frame with some identifier columns and the predictions from the model for each row. If any row is screened out by the subset parameter, then it will be returned with an NA in the prediction column.

`fcMacro` *Run a set of control objects*

Description

This function takes a list of control objects and goes through the steps needed to run the tests for those control objects, and save the outputs.

Usage

```
fcMacro(npt, conn = NULL, do.rmse = TRUE, save.tests = NULL,
  err.Log = NULL)
```

Arguments

`npt`	character vector. Lists names of the control objects to be run.
`conn`	DBI connection. Connection to the database containing the 'controls' sets. Only needed if the control objects in npt above are not on the command line. [optional]
`do.rmse`	logical. If TRUE (the default) it will calculate the out-of-sample RMS errors.
`save.tests`	environment. Where to save the test results. [optional]
`err.Log`	Either character or an err.Log structure. If this is a character vector, then it creates an err.Log structure based on that vector. If it is an err.Log structure, it uses the passed structure. If it is NULL, then no error logging is done.

Details

The object listed in `npt` need not exist in the R environment. If it does not, then this function calls the `npt` function to get the definition of the control object out of the database. That is what the `conn` variable is used for. If the objects already exist in the R environment, then there is no need to supply the `conn` variable.

This routine sequentially runs `fcRun` and `fcTest` for all the test objects supplied. It then collects summary information from the `fcTest` output for all the control objects listed and returns it in the list `rmse.sum` which is created in the global environment.

The function saves the control, output, and test objects to disk and deletes them from the R environment. This step is necessary because some of output objects are quite large.

The function creates a message text file on disk for each control object so as to contain any errors or or messages generated while running the models. The name of the message file is 'message.nn.txt' where nn is a two-digit number. This function finds the message file with the largest such number already on the disk and names the new file with the next-largest number. Note that if a large number of such files already exist on the disk or if fcMacro is called with a long list of control objects, it is possible for the 'nn' in the name above to extend into triple digits.

If the save.tests environment is supplied, then the `fcTest` results (and only the `fcTest` results) will be retained in the environment specified. The `fcTest` objects are small enough that they can be retained without much harm. Note again that they are saved (if they are saved) in a separate environment which keeps the global environment less cluttered.

Side Effects:

- Creates `rmse.sum` in the global environment if `do.rmse` is `TRUE`.

- For each control object, creates a .RData file on the disk containing the objects created.
- For each control object, creates a message text file on disk containing any errors or messages generated in the process.
- Optionally saves the outputs of the fcTest function in a specified environment.

rmse.sum is a named list. Each control object that produces a fcTest object (many will not) has an entry in the list. The name of the entry is the name of the control object. The entry is the .$se entry in the fcTest object.

Value

Data frame listing the objects and files created for each control object.

The data.frame has the following structure:

npt.name Name of the control object.

res.name Name of the results output by fcRun.

tst.name Name of the test results output by fcTest.

msg.name Text file in which errors and messages associated with this object are output. This is obsolete, and will soon be deleted.

save.name Name of the file to which all objects are saved.

Note that if an object is not created (typically due to an error) then an NA will appear in the appropriate cell.

Examples

```
## Not run:
  fcMacro(c("mr.final", "npt.final", "npt.final.L"))

  res <- new.env()
  fcMacro(c("mr.final", "npt.final", "npt.final.L"), save.tests=res)

## End(Not run)
```

---

fcMerge *Takes two or more models and combines them.*

Description

There are cases where different models apply to different portions of the prediction set. This takes the predictions for each model and applies the prediction only to those portions of the set to which it applies.

Usage

```
fcMerge(predictions, merge.by)
```

Arguments

`predictions`	data.frame. This is a data frame of the predictions for the models that make up the constituent parts of the final prediction.
`merge.by`	vector. This vector must be the same length as the prediction frame, and determines which column of the `prediction` data frame goes in which record. See `details` below for its format.

Details

This function returns a single vector where each entry is one of the values in `predictions` from the corresponding row. Which row depends on the value in the corresponding entry in `merge.by`.

Each entry in the `merge.by` vector should refer to a specific column in the `predictions` data frame, but format is flexible. If `merge.by` is a factor or character vector, then the columns will be referenced by name. If it is an integer vector, then they will be referenced by column number. If it is a numeric vector, then the vector will be coerced to an integer vector.

If an entry in the `merge.by` vector does not correspond to a column in the `predictions` data frame, then an NA will be returned for that entry.

This is similar to the process in `fcRun` and `fcEstimate` where different portions of the domain are estimated and predicted separately. This is more flexible in that it allows different models to be applied to different portions of the domain while in those routines, the models must be the same–they are simply estimated and predicted separately.

Value

returns a list. The `cols` entry in the list is an integer vector listing the column numbers that were merged to generate the results. The `result` entry is the vector containing the merged data.

Examples

```
y <- matrix(rnorm(30), ncol=5)
y <- as.data.frame(y)
x1 <- c(1,-1,6,2,2,4)
x2 <- c("V1", "VX", "V6", "V2", "V2", "V4")
x3 <- c(1.1, 0.9, 7.4, 2.1, 2.9, 4.4)

# All three of these return the same result
fcMerge(y,x1)
fcMerge(y,x2)
fcMerge(y,x3)

## Not run:
# This throws an error because x4 is longer than y
x4 <- c(1,2,3,4,5,6,7)
fcMerge(y,x4)

## End(Not run)
```

fcRun *Fit models described in the supplied control objects*

Description

The function takes the control object specifying a series of regression models and runs those models.

Usage

```
fcRun(sets, n = 0, err.Log = NULL)
```

Arguments

`sets`	Control Object The control object describing the models to run. This will typically be generated by 'npt'
`n`	Integer. Number of bootstrap replications to run in order to estimate the confidence intervals on parameters. n=0 (the default) will not run any bootstrap replicates.
`err.Log`	Either character or an err.Log structure. If this is a character vector, then it creates an err.Log structure based on that vector. If it is an err.Log structure, it uses the passed structure. If it is NULL, then no error logging is done.

Details

Creates: an `out` object listing output of the models.

The reason that `out` is created in the global environment rather than returned as an object is that if one of the models errors out, I still get the results of the previous models.

This will typically be followed up by a run of `fcTest`

Examples

```
## Not run:
  fcRun(mr.d.00, sink="messages.13.txt")
  fcRun(mr.f.S0b, n=1000, sink="messages.08.txt")
  fcRun(mr.j.L0a)

## End(Not run)
```

fcSetup *Condition a data set for use in model estimation*

Description

Condition a data set for use in model estimation

Usage

```
fcSetup(dta, seed = 953016876)
```

Arguments

`dta`	data.frame. The data set that needs to be condition for use.
`seed`	Intger. A random seed used to ensure consistent results for the partitioning of the data set into training and test sets.

Details

This takes the 'low.risk.fires', 'med.risk.fires' and 'high.risk.fires' data.frames as pulled from the database and makes the modification needed to use them for analysis.

It also does much of the pre-processing on the 'lr_mr_pred' and 'hr_pred' data frames as well.

This function carries out the following tasks:

1. Set any nulls in the outcome variables to zero.
2. Turn any categorical predictors into factors.
3. Take the log of the income variable.
4. Ensure that there is an f_located column in the table.
5. Define any filters that are needed (only for training tables).
6. Define training and test sets for the training tables.

Value

The conditioned data frame.

`fcTest` *Compute out-of-sample RMS Errors for model output*

Description

Compute out-of-sample RMS Errors for model output

Usage

```
fcTest(input, output, subset = NULL)
```

Arguments

`input`	Control object. The input control object used by `fcRun` to generate the output.
`output`	Model Output. The model output produced by `fcRun`.
`subset`	The subset of the data over which to estimate RMS Errors. I include this because in some cases the test subset has been different from the training subset in non-random ways.

Details

This function takes output from the `fcRun` function and calculates the out-of-sample Root-Mean-Square Error values for each model in the output object.

Value

This returns a list with the following members:

lhs Name of the left-hand side variable.

subset The subset to which the results are applied.

se A named vector with the root-mean-square errors on the out-of-sample data for each model in the control object.

results A data frame with the row-by-row results.

`full_analysis` *Runs the full analysis of a set of models.*

Description

This function runs through the entire process of estimating models and developing predictions at either the census tract level or at the department level. This function can be run with NO inputs. That is, it can be called as `full_analysis()` and it will work. Default values exist in some form for all the parameters. Supply the parameters if you want different results.

Usage

```
full_analysis(conn = NULL, refresh.data = FALSE, models.run = NULL,
  merg = NULL, bypass.models = FALSE, do.predictions = TRUE,
  do.rmse = FALSE, roll.up.2.dept = TRUE, object.list = NULL,
  incl.detail = FALSE)
```

Arguments

`conn`	A DBI Connection. This is a connection to the database containing the data and model definitions. If none is entered, default connection information is obtained from the operating system environment.
`models.run`	Either a list or a data frame. This determines what models are run. Its format is given as an example below. If it is undefined, then a default set of models are run (see below). Note that multiple model objects per risk level does not present a problem.
`merg`	Either a vector or a list. A value of '0' guarantees that the merg option will not be used.
`object.list`	List of data frames. The list must contain an entry for every risk level that is run. The entry for each risk level must contain data frame with the output from `fcMacro` for that risk level. If `bypass.models` is TRUE and this parameter is undefined, the function will error out. If `bypass.models` is FALSE, then this parameter is ignored.
`refresh.data=FALSE`	Logical. [This feature has not been implemented yet] If this is TRUE, then the views on which the data for this analysis are based are refreshed and the data.frames used in this analysis are requeried.

`bypass.models=FALSE`
: Logical. If it is TRUE, then no models are estimated. If not, then the models listed in 'models.run' above are estimated first. Note that if 'bypass.models' is TRUE, then the 'objects' data frame (output of the fcMacro function) must be supplied.

`do.predictions=TRUE`
: Logical. If it is TRUE, then predictions are generated from the models run. If not, then no predictions are generated from the estimated models.

`do.rmse=FALSE`
: Logical. If TRUE, then the RMS Errors of all the models will be computed.

`roll.up.2.dept=TRUE`
: Logical. If it is TRUE, then the predictions are rolled up to the department level. If it is FALSE, then the predictions are left at the census tract level.

`incl.detail=FALSE`
: Logical. If TRUE, the `risk.results` section of the output (described below) is included. If FALSE then the `risk.results` section is not returned.

Details

The `models.run` parameter can have one of two formats, a list format or a data frame format. The list format is preferred. The data frame format has two columns: `risk` and `lst`. Both columns have character format. Each row represents a model set to be run. The `risk` column specfies the risk level associated with that model set (and can only be one of 'lr', 'mr', 'hr', 'ems500', and 'emscty'). The `lst` column is the `lst` value from the `controls` database for the model set to be run. The default value of `models.run` (in data.frame format) is listed below.

risk	**lst**
lr	npt.final
mr	mr.final
hr	hr.final
ems500	ems.5.final
emscty	ems.C.final

The list format contains a named entry for each risk level to be run. Each entry contains a character vector listing the `lst` values from the `controls` database for the model sets to be run for that risk level. The default value of `models.run` (in list format) is listed below.

```
models.run <-list(lr=c("npt.final"),mr=c("mr.final"),hr=c("hr.final"),ems500=c("ems.5.final"),ems
```

The `merg` entry is if you want to merge multiple models into a single model. It can either be a data frame or a list of data frames. The data frame contains two columns. The first column, labeled either 'geoid' or 'tr10_fid', contains the tract/spatial identifiers used for prediction. The 'group' column should refer to a specific prediction column in the preliminary output. See `fcMerge` for more details on format. If the `merg` entry is a list, then each entry in the list is treated as as separate merger and must have the appropriate format.

Each list entry can be (read 'should be') named. If it is named, then the new column in the predictions output will have the name given the list entry. If only a data frame is supplied, then the default name of "merge" will be used. For that reason, the list format is preferred, even if only one merger is done.

As currently written, this will only merge numeric columns. If you want to merge non-numeric columns, the merger will have to be performed outside of this function.

This returns the output data frame with the new merged column and with the old constituent columns removed. However, since the removal of the old columns is performed last, it is possible to use the same source column for multiple final merged columns.

The `merg` entry is only processed if `do.predictions` is `TRUE`.

The `object.list` list object has an entry for each risk level run. That entry is a data frame with information output from `fcMacro`. The structure of that data frame is given by the following example:

npt.name	**res.name**	**tst.name**	**msg.name**	**save.name**
npt.final	npt.final.res	npt.final.tst	messages.00.txt	npt.final.RData
npt.final.L	npt.final.L.res	npt.final.L.tst	messages.01.txt	npt.final.L.RData

Note that if you are supplying the `object.list` structure while using the `bypass.models` option, you can safely leave out the `tst.name` and `msg.name` columns.

Value

returns a list with the following entries:

models.run The `models.run` input listing the models run by risk level.

bypass.models The input `bypass.models` value.

do.predictions The input `do.predictions` value.

roll.up.2.dept The input `roll.up.2.dept` value.

object.list The `object.list` object described above. If the `bypass.models` flag is set, then this is the object supplied to the function. Otherwise it is returned by the calls to `fcMacro`.

msg.name Name of the message file to which messages are sent.

prediction Data frame containing predictions for all variables requested in the models.run object. The predictions are either by census tract or by department depending on the value of the `roll.up.2.dept` flag.

risk.results This is a list, with an entry for each risk level. Each entry contains a data frame with the raw estimates for that risk level. For low and medium risk fires this contains the predictions at the census tract level (which are redundant with the results in `predictions` if `roll.up.2.dept` is FALSE). For high risk fires, this contains predictions at the parcel level.

Side Effects

The function leaves several new files on the hard drive. The exact files are listed in the `object.list` section of the output. The files fall into two groups. First, output of the estimation algorithms are saved in a set if objects listed in the `object.list` object. Second, error and log files are saved to disk.

`getContext` *Retrieve message context.*

Description

This function returns the message context.

Usage

```
getContext(err.Log = NULL)
```

Arguments

`err.Log`	The err.Log structure. If this is NULL, then the function returns NULL

Details

Message context is information that is printed with the message. The intent is that message context will provide information that will help contextualize the message.

Value

This function returns the message context in a character vector. If logging has not been set up, or if no context has been set, NULL is returned.

`getDests` *Returns the files to which messages are sent.*

Description

Returns the files to which messages are sent.

Usage

```
getDests(err.Log = NULL)
```

Arguments

`err.Log`	the err.Log structure. If this is NULL, then the function returns NULL

Details

This function returns the list of files to which messages are set. It does not return 'console' in the vector, even if messages are sent to it.

Value

This function returns a character vector containing the files to which messages are sent. If an error environment has not been set up, then it returns NULL.

`lasso` *LASSO helper function*

Description

This is a helper function that `fcRun` calls whenever a LASSO model is used.

Usage

```
lasso(formula, data, subset = NULL, ...)
```

Arguments

`formula`	Formula. This describes the model that the LASSO fits.
`data`	Data Frame. The data used for the model.
`subset`	Name. This defines the subset of the data the model is evaluated over.
`...`	Additional parameters to the `cv.glmnet` function.

Details

Basically it takes the standard inputs from the `fcRun` routine and translates them to work with the glmnet `cv.glmnet` function.

Value

Returns the glmnet.lasso object with the call slot altered to reflect the call to this function rather than the glmnet function.

`mass.npt` *Mass Building of Control Objects*

Description

This uses a pattern to collect a set of control files, and then calls ’npt’ for each of those control files.

Usage

```
mass.npt(conn, pattern = NULL, list = NULL, relocate = NULL,
  err.Log = NULL)
```

Arguments

`conn`	DBI connection. Connects to the database containing the ’controls’ sets.
`pattern`	character. Used to pattern-match the ’lst’ values in the control list.
`list`	character. List of control objects to build.
`relocate`	environment. This is an [optional] environment into which to move any existing test objects.
`err.Log`	Either character or an err.Log structure. If this is a character vector, then it creates an err.Log structure based on that vector. If it is an err.Log structure, it uses the passed structure. If it is NULL, then no error logging is done.

Details

Creates: A set of control objects in the global environment, as specified in the 'pattern' input.

This is usually followed up with a call to `fcMacro`

Value

Vector listing the control objects created.

Examples

```
## Not run:
  conn <- dbConnect("PostgreSQL",
                    host="hostname.com",
                    dbname="nfirs",
                    user="username",
                    password="***")
  mass.npt(conn, "npt.f")
  res <- new.env()
  mass.npt("final", conn, res)

## End(Not run)
```

`msgOut` *Send messages to the error log(s).*

Description

This function sends messages to all error logs that have been set up.

Usage

```
msgOut(err.Log = NULL, message = NULL, type = "error")
```

Arguments

`err.Log`	This is the err.Log environment that was set up earlier. If this is NULL, then the function does nothing.
`message`	character. The text of the message to be sent. [optional]
`type`	character. The type of message. One of 'error', 'warning', 'message', 'status', 'start', or 'stop'. This parameter is not case sensitive. Only the first option is used. Any unknown entry is treated as equivalent to 'status'. See `details` for what these parameters mean.

Details

The message that is posted depends on the type.

If `type` is one of 'error', 'warning', or 'message', then the message (with `context` prepended) is posted to all logs.

If `type` is 'status' (or any equivalent–any unidentified type is interpreted as 'status'), then the message (with `context` prepended) is posted to the console IF console is one of the logs set up.

If `type` is 'start', then a message (with `context` prepended) is posted to the console (IF console is one of the logs set up) stating that a process has started. The 'start' code also records the starting time.

If `type` is 'stop', then a message (with `context` prepended) is posted to the console (IF console is one of the logs set up) stating that a process has completed, with the elapsed time (if a starting time has been recorded).

If no error logs have been set up, this routine automatically calls `openLog` with a basic option to generate one log file.

Value

This function invisibly returns the deleted error information.

`naive` *Naive estimator*

Description

Generates the naive estimator for any given data set.

Usage

```
naive(test)
```

Arguments

`test`	`fcTest` object.

Details

This takes an output object from the `fcTest` function and computes the Naive predictor. The Naive predictor says that the best prediction for a tract-year is the number of outcomes (fires, injuries, etc.) that occurred for that tract the previous year. It is undefined for the first year in the data set.

Value

(modified) `fcTest` object.

`npt` *Create control objects*

Description

This function creates the specified control objects from the templates maintained in the database.

Usage

```
npt(conn, group = NULL, risk = NULL, y = NULL, mdls = NULL,
  run = "S")
```

Arguments

`conn`	DBI Connection. Connects to the database containing the controls tables.
`group`	character The entry in the 'lst' column. This determines which models get used.
`risk`	character. This along with 'y' and 'mdls' (and 'run') represent an alternative way of specifying which models get used. If 'group' is specified, this is ignored. This is risk category, and is one of 'l' (for low risk properties), 'm' (for medium risk properties), or 'h' (for
`y`	character. This is the target variable, and is one of 'f' (for fires), 'j' (for injuries), 'd' (for deaths), 'sz2' (for "size 2" fires), 'sz3' (for "size 3" fires), and 'ems'.
`mdls`	character This is a character vector of the models to be included for that target variable.
`run`	character This currently takes one of four values: '0', 'S', 'L', and 'C'. The values '0' and 'S' both run a single model for all department sizes and regions (typically with dummies for each). The value 'L'runs separate models for each combination of department size and region. The value 'C' runs separate models for each cluster. Note that '0' as an option is deprecated.

Details

As written, this creates a single control object regardless of whether multiple groups are specified.

If multiple groups are specified, only the first is processed. If y and mdl are specified, only the first y value (and the first 'runs' value) is processed. The multiple models in the y; mdl formulation are all added to a single control object, so be careful. It is easy to build a control object that will produce an output file so large it will choke the computer.

Note that this function does not check for the validity of the input to the `run` parameter. That allows me to add additional types of runs if needed without rewriting the function. On the other hand, that means invalid inputs are caught only if the queries fail.

Value

control object

Examples

```
## Not run:
  npt(conn, "npt.base")
  npt(conn, "npt.base", run="L")
  npt(conn, y="f", mdls=c("", ""))
  npt(conn, y="d", mdls=c("", ""), run="L")

## End(Not run)
```

`openLog` *Open an error log*

Description

This function opens the error log.

Usage

```
openLog(dest = c(NA, "console"))
```

Arguments

`dest`	character vector. destinations to send the error output to. Destinations are usually file names.

Details

This sets up the error logging system. The parameter `dest` lists all the output locations to which messages are to be sent. If more than one location is specified, then the messages are sent to all the specified locations.

With three exceptions, the values in `dest` are interpreted as file names. The exceptions are 'console' (case is ignored here), the empty string (''), and NA. Either of the first two cases are interpreted as requesting that output be sent to the console.

For every NA included in the `dest` vector, a file name is generated. The file name has the form 'messages.xx.txt' where the 'xx' is a two (or more) digit number, starting with '01'. The function finds the largest such file name in the working directory and uses the next larger number for the new file name. So, I can auto-generate 3 (or any other arbitrary number) of log files simply by listing `NA` three times in the `dest` parameter vector.

Duplicates are automatically removed. Note that (aside from the 'console'), names are case sensitive. So two entries with the same name, aside from case differences, will both be retained. For operating systems that are case insensitive, that will result in duplicate messages being sent to the file.

If the session is not interactive, then the request to send messages to the console is automatically ignored.

If an error environment has been set up for a calling function, then some of the information for the calling function is imported for this function. In particular, `context` is imported, and the `dest` values are imported. If there is an error environment for a calling function, then no new file name is generated, even if `NA` is included in this call. As usual, duplicates are removed. So, any messages are sent to all log files used for calling functions, and to any new files specifically specified in this call. But no new file is automatically generated, even if `NA` is specified in this call.

If an error environment has already been set up for this function, this deletes it and starts over.

Value

This function returns the error environment `pkgFireCARES` *pkgFireCARES: A package for estimating community risk in FireCARES*

Description

This package creates a series of functions that are used to estimate community risk models and make community risk predictions.

Functions

Functions included are:

`full_analysis`: Runs through complete analysis (depending on the parameters set).

`refresh_views`: [This feature has not been implemented yet] Since the underlying data used to create the data sets for this analysis change, and the materialized views this analysis uses do not refresh automatically, this makes sure those views are current before continuing with the analysis.

`fcCluster`: Builds the cluster assignments for US Counties. So far, this is used exclusively for EMS analysis.

`fcSetup`: Takes data file (either for model estimation or prediction) and prepare it for use.

`npt`: Builds a control object from the specified templates in the database.

`mass.npt`: Builds a collection of control objects. This function calls npt to do most of the work.

`fcRun`: Uses the control object to run a set of models.

`fcTest`: Calculates the out-of-sample Root-Mean-Square error on the results for the models in the supplied test object. This function works on output from `fcRun`.

`fcMacro`: For a supplied set of control objects, sequentially `fcRun`s them, runs `fcTest` on them, summarized the `fcTest` results in a single data.frame, and saves the results to disk.

`naive`: Takes a `fcTest` output and computes the naive estimator and the RMS Error for the naive estimator for that test object.

`fcEstimate`: Takes output from the `fcRun` routine and new data and computes predictions by tract or (for high-risk fires) Assessors Parcel.

`fcMerge`: Takes two or more separate predictions (from `fcEstimate`) and combines them according to a rule specified. Primarily used with EMS.

`rollUp2Dept`: Takes output from the `fcEstimate` routine sums over census tracts to the department level.

`lasso`: Helper function for LASSO and ridge regression models.

`ranger`: Helper function for Random Forest models (using the **ranger** package).

`c_test`: Combines two test objects.

`acs.dwnld`: Downloads new ACS data from Census for import to the database. This should considerably simplify the process of keeping census data up to date. Note that it requires census API key installed (see the acs package documentation).

The following functions are used for error logging.

`openLog`: Creates the log structure, and opens log file(s) as requested.

`msgOut`: Sends log and error messages to the output pre-selected output destinations.

`setContext`: Sets a context message so that the errors and messages in the log file can be identified.

`getContext`: Retrieves the context message that was set up.

`getDests`: Gets the log and error destinations that were set up.

Database Info

This package works with data on the nfirs database on the FireCARES server. In particular, it works with the information in the nist and controls schemas. Most of that, however, is transparent to the functions in this package. Any function (except full_analysis) that accesses the database takes a DBI Connection as one of its parameters. That connection will contain all the connection parameters and must be supplied.

Note that while I assume that the database is a PostgreSQL one (as is currently the case), there is nothing in these functions (again, except for full_analysis) that is specific to PostgreSQL. So any DBI connection can be used. There are packages in R that create DBI connections for MySQL, SQLite, Oracle, the ODBC Interface, SQLServer, and others. So these functions should continue to work even if the server hosting the database is changed.

Even full_analysis allows for a DBI connection object to be supplied. So, if the database were to change, then the correct connection object could be supplied without rewriting the package.

Typical Workflow

This section takes you through the basic work flow that will typically be followed in using this package. The function full_analysis automates this process.

Build county clusters This is typically done with a call to fcCluster and (so far) is only relevant to EMS risk. Since the underlying data used to build county clusters only changes rarely, it should only have to be done once every year or two.

Refresh the views in the database. This is typically done with a call to refresh_views [which is not functional yet!!!!].

Build the definitions of the models to be estimated. That will typically be done by a call to mass.npt, although it could be done by calling npt directly. Either will leave one or more control objects in the working environment.

Download data for analysis. That will need to be done separately if full_analysis is not used.

Prepare the data for analysis. This is done by a call to fcSetup.

Estimate the models and calcuate the RMS Error for all the models queued up for estimation. That is typically done by a call to fcMacro. However it can be done by sequentially calling fcRun and fcTest, although that is not recommended. Note that fcMacro takes all the objects created by either fcRun or fcTest, saves them to file and deletes them from the working enviroment. It leaves behind a summary data frame summarizing the RMS Errors of the models run.

Estimate the naive model for comparison. To do that, you will need the output of fcTest from one of the sets of models estimated, and will use the naive function.

Estimate predictions for each tract or parcel. That occurs in three steps. First, download the data to be used to make predictions. That will occur outside any of these functions. Second, prepare the new data for analysis. That occurs through a call to fcSetup. Finally, compute the predictions based on the selected models. That occurs through a call to fcEstimate.

Optionally combine separate predictions into one. In some cases separate predictions apply to different portions of the prediction set. A call to fcMerge will combine them.

Optionally, roll the census tract predictions up to the department level. A call to rollUp2Dept completes this task.

ACS Data

These models and estimates rely on data from the American Community Survey maintained by the Census Bureau. New data is released for the survey annually. The function acs.dwnld is a utility function that simplifies the process of downloading new data. In order to use it you will need a

Census API key installed on the server (see the **acs** package documentation for more details). It leaves a set of tables on the server (in the 'nist' schema) that are formatted the same as the master ACS tables already on the server. Those tables will need to be appended to the existing ACS tables already on the server.

IMPORTS

acs,boot,glmnet,ranger,RPostgreSQL,utils,cluster,magrittr

SUGGESTS

doParallel

Notes

These functions do assume that the information they need is in the 'controls' and 'nist' schemas, so if that changes, these functions will need to be rewritten.

The 'controls' schema is assumed to contain the definitions of all models that are used by these functions. The format for the tables in the 'controls' schema is very specific, and is hard-coded into these functions. There are three tables assumed to exist in the 'controls' schema: `models`, and `inputs` and `runs`. Their layouts are described below.

Table models

This table specifies information about each model to be run.

Name	Type	Details
index	integer	primary key.
lst	text	Typically the name of the control object.
model	text	Name of the model to be estimated.
library	text	Name of the library needed to estimate the model.
ff	text	Name of the function that estimates the model.
target	text	Name of the dependent variable estimated.
runs	text	One of '0', 'S', or 'L'. Whether the model is estimated over the whole data set ('0' or 'S') or separately

Table inputs

This table specifies the parameters for the model to be run.

Name	Type	Details
index	integer	primary key.
lst	text	Typically the name of the control object.
model	text	Name of the model to be estimated.
input	text	Name of an input parameter for the estimation function (ff above).
class	text	Class of the input parameter.
value	text	Value of the input parameter.

Note that for practical purposes, the (lst, model) pair serve as keys to the list of models, and they are a foreign key that the `inputs` table uses to link to the `models` table.

Table runs

This table specifies how the data is partitioned. Each partition will have a separate model built for it. Other than the partition, all other inputs are identical.

Name	Type	Details
grp	text	One of 'L', 'S', '0', or 'C'. This matches the `runs` column in the `models` table.
tier1	text	This combined with 'tier2' below serve as a name for the subset to be evaluated.
tier2	text	See above.
value	text	Definition of the subset to be evaluated.

Parallel Processing

Both LASSO and Random Forest (through the **ranger** package) can use parallel computation if multiple processors are available. The **ranger** package has support for multiple processors built in by default. I have made no adjustment to the defaults, so it will use them if they are there and the package supports them. LASSO (through the **glmnet** package) can also use it, but setup is required. LASSO here is set up to use the **doParallel** package if it is set up. Note that for LASSO to use multiple processors, **doParallel** must be set up separately. That is, the package must be installed and loaded (typically with a call to `library`) in advance. It that is done (and works–doParallel only works on certain types of systems) the LASSO will make use of it. If not, it will not.

Author(s)

Maintainer: Stanley Gilbert `<stanley.gilbert@nist.gov>`

Examples

```
## Not run:
conn <- dbConnect("PostgreSQL",
                  host="some.host.com",
                  dbname="nfirs",
                  user="user",
                  password="pwd")
low.risk.fires <- dbGetQuery(conn, "select * from nist.low_risk_fires")
low.risk.fires <- fcSetup(low.risk.fires)
med.risk.fires <- dbGetQuery(conn, "select * from nist.med_risk_fires")
med.risk.fires <- fcSetup(med.risk.fires)
high.risk.fires <- dbGetQuery(conn, "select * from nist.high_risk_fires")
high.risk.fires <- fcSetup(high.risk.fires)

models <- mass.npt(conn, pattern="final")
tables <- fcMacro(models)
tables

lr.mr.pred <- dbGetQuery(conn, "select * from nist.lr_mr_pred")
lr.mr.pred <- fcSetup(lr.mr.pred)

e <- new.env()
npt.final <- e$npt.final
npt.final.res <- e$npt.final.res
lr.pred <- fcEstimate("npt.final",
                      "npt.final.res",
                      lr.mr.pred,
                      quote(fd_size %in% paste("size_", 3:9, sep="")))
head(lr.pred)
```

```
## End(Not run)
```

---

ranger *ranger helper function*

Description

This is a helper function that fcRun calls whenever a ranger model is used.

Usage

```
ranger(formula, data, subset = NULL, ...)
```

Arguments

`formula`	Formula. This describes the model that the Random Forest fits.
`data`	Data Frame. The data used for the model.
`subset`	Name. This defines the subset of the data the model is evaluated over.
`...`	Additional parameters to the ranger function.

Details

Basically it takes the standard inputs from the 'run' routine and translates them to work with the ranger function.

There are two reasons why this helper function exists: first, the default ranger function does not have a subset argument. Second, the weights, when they are used, need to be converted from symbol (or quote) to a vector.

Value

Returns the ranger object with the call slot altered to reflect the call to this function rather than the ranger function.

---

refresh_views *Refreshes the Materialized Views in the database*

Description

This function refreshes the specified materialized views in the database. Since the materialized views are the source of the data used for this modeling exercise, and the view are not refreshed automatically, this does that.

Usage

```
refresh_views(conn = NULL, tables = NULL)
```

Arguments

`conn`	A DBI Connection. This is a connection to the database containing the data and model definitions. If none is entered, default connection information is obtained from the operating system environment.
`tables`	This provides the views that will be refreshed. It can take one of two forms: either a vector of view names, or a list (or dataframe). If it is a vector of view names, then the views are assumed to be in the `public` schema. If they are not in the public schema, then the list form must be supplied. The first entry in the list will be a vector of schemas, while the second entry in the list will be a vector of view names. If this field is `Null`, then the list of views to be refreshed will be drawn from the database (see details below).

Details

If `tables` is null, then the list of views to refresh is drawn from the database. There is a table in the `public` schema in the database called `m_views`. It has three columns:

order Numeric field listing the order in which views are to be refreshed.

schema Lists the schema in which the view to be refreshed is located. If it is Null, then the schema is assumed to be `public`.

view Contains the name of the materialized view to be refreshed.

Any view with a Null entry in the `order` field will not be refreshed. All the others will be refreshed in the order listed in the `order` field.

Value

returns a data frame with the following entries:

schema schema name of the view refreshed.

view name of the view refreshed.

statement actual sql statement used to refresh the view.

complete logical value stating whether the refresh completed.

rows.aff number of rows affected–this may not actually mean anything, I do not have enough information yet to know that.

`rollUp2Dept` *Roll up census-tract predictions to the department level*

Description

This function takes output from the fcEstimate function, which is typically at the census-tract level, and rolls them up to the department level.

Usage

```
rollUp2Dept(predictions, fire.col, sz2.col, sz3.col, indx.cols)
```

Arguments

`predictions`	data frame containing predictions as output by the fcEstimate function. See details below.
`fire.col`	character vector containing names of the columns with fire predictions.
`sz2.col`	character vector containing names of the columns with predicted percentages of fires that go beyond the room of origin.
`sz3.col`	character vector containing names of the columns with predicted percentages of size 2 fires that go beyond the structure of origin.
`indx.cols`	character vector containing the names of the columns to which the data is to be rolled up.

Details

This routine is intended to work with a data frame containing multiple estimates (think low- medium- and high-risk estimates all contained in the same data frame). When that is the case, the first entry in fire.col goes with the first entry in sz2.col and sz3.col. The second entry in in fire.col goes with the second entry in sz2.col and sz3.col. And so on. See the example below.

This function takes output from the fcEstimate function, which is typically at the census-tract level, and rolls them up to the department level. What makes this more complicated than a simple call to aggregate is that the sz2 and sz3 columns are percentages rather than estimated counts. So, summing those columns produces nonsensical results.

What this function does is create temporary columns for the estimated counts for medium and large fires. It then sums over census tracts (rolling up to the department level), and back-calcuates the percentages from the accumulated estimated counts.

Value

A data frame with the department-level predictions.

Examples

```
## Not run:
dept.predict <- rollUp2Dept(predictions, c("lr.fires", "mr.fires", "hr.fires"),
                                         c("lr.sz2",   "mr.sz2",   "hr.sz2"),
                                         c("lr.sz3",   "mr.sz3",   "hr.sz3"),
                                         c("year", "fd_id"))

## End(Not run)
```

`setContext` *Sets message context.*

Description

This function sets the message context.

Usage

```
setContext(err.Log = NULL, context)
```

Arguments

`err.Log`	The err.log structure. If this is NULL, then the function does nothing.
`context`	Character Vector. A character vector representing the context information to be printed with the messages.

Details

Message context is information that is printed with the message. The intent is that message context will provide information that will help contextualize the message.

If no logging has been set up, this automatically calls the setup routine with the type set to generate one message file.

Value

This function returns the updated err.Log structure.

Index