Package 'FLAME'

December 7, 2021

Type Package

Title Interpretable Matching for Causal Inference

Version 2.1.1

URL `https://almost-matching-exactly.github.io`,`https://vittorioorlandi.github.io/`

BugReports `https://github.com/vittorioorlandi/FLAME/issues`

Description Efficient implementations of the algorithms in the
Almost-Matching-Exactly framework for interpretable matching in causal
inference. These algorithms match units via a learned, weighted Hamming
distance that determines which covariates are more important to match on.
For more information and examples, see the Almost-Matching-Exactly website.

License MIT + file LICENSE

Encoding UTF-8

Imports glmnet, gmp

RoxygenNote 7.1.1

Suggests nnet, knitr, mice, rmarkdown, testthat, xgboost

VignetteBuilder knitr

NeedsCompilation no

Author Vittorio Orlandi [aut, cre],
Sudeepa Roy [aut],
Cynthia Rudin [aut],
Alexander Volfovsky [aut]

Maintainer Vittorio Orlandi <`almost.matching.exactly@gmail.com`>

Repository CRAN

Date/Publication 2021-12-07 22:50:02 UTC

R topics documented:

`AME` *Almost Matching Exactly (AME) Algorithms for Discrete, Observational Data*

Description

Almost Matching Exactly (AME) Algorithms for Discrete, Observational Data

Usage

```
FLAME(
  data,
  holdout = 0.1,
  C = 0.1,
  treated_column_name = "treated",
  outcome_column_name = "outcome",
  weights = NULL,
  PE_method = "ridge",
  user_PE_fit = NULL,
  user_PE_fit_params = NULL,
  user_PE_predict = NULL,
  user_PE_predict_params = NULL,
  replace = FALSE,
  estimate_CATEs = FALSE,
  verbose = 2,
  return_pe = FALSE,
  return_bf = FALSE,
  early_stop_iterations = Inf,
  early_stop_epsilon = 0.25,
  early_stop_control = 0,
  early_stop_treated = 0,
  early_stop_pe = Inf,
  early_stop_bf = 0,
  missing_data = c("none", "drop", "keep", "impute"),
  missing_holdout = c("none", "drop", "impute"),
  missing_data_imputations = 1,
  missing_holdout_imputations = 5,
  impute_with_treatment = TRUE,
  impute_with_outcome = FALSE
)
```

```
DAME(
  data,
  holdout = 0.1,
  treated_column_name = "treated",
  outcome_column_name = "outcome",
  weights = NULL,
  PE_method = "ridge",
  n_flame_iters = 0,
  user_PE_fit = NULL,
  user_PE_fit_params = NULL,
  user_PE_predict = NULL,
  user_PE_predict_params = NULL,
  replace = FALSE,
  estimate_CATEs = FALSE,
  verbose = 2,
  return_pe = FALSE,
  return_bf = FALSE,
  early_stop_iterations = Inf,
  early_stop_epsilon = 0.25,
  early_stop_control = 0,
  early_stop_treated = 0,
  early_stop_pe = Inf,
  early_stop_bf = 0,
  missing_data = c("none", "drop", "keep", "impute"),
  missing_holdout = c("none", "drop", "impute"),
  missing_data_imputations = 1,
  missing_holdout_imputations = 5,
  impute_with_treatment = TRUE,
  impute_with_outcome = FALSE
)

## S3 method for class 'ame'
print(x, digits = getOption("digits"), linewidth = 80, ...)
```

Arguments

`data`	Data to be matched. Either a data frame or a path to a .csv file to be read into a data frame. Treatment must be described by a logical or binary numeric column with name `treated_column_name`. If supplied, outcome must be described by a column with name `outcome_column_name`. The outcome will be treated as continuous if numeric with more than two values, as binary if a two-level factor or numeric with values 0 and 1 exclusively, and as multi-class if a factor with more than two levels. If the outcome column is omitted, matching will be performed but treatment effect estimation will not be possible. All columns not containing outcome or treatment will be treated as covariates for matching. Covariates are assumed to be categorical and will be coerced to factors, though they may be passed as either factors or numeric; if the former, unused levels will automatically be dropped. If you wish to use continuous covariates for matching, they should be binned prior to matching.

`holdout`	Holdout data to be used to compute predictive error, if `weights` is not supplied. If a numeric scalar between 0 and 1, that proportion of `data` will be made into a holdout set and only the *remaining proportion* of `data` will be matched. Otherwise, a data frame or a path to a .csv file. The holdout data must contain an outcome column with name `outcome_column_name`; other restrictions on column types are as for `data`. Covariate columns must have the same column names and order as `data`. This data will *not* be matched. Defaults to 0.1.
`C`	A finite, positive scalar denoting the tradeoff between BF and PE in the FLAME algorithm. Higher C prioritizes more matches and lower C prioritizes not dropping important covariates. Defaults to 0.1.
`treated_column_name`	Name of the treatment column in `data` and `holdout`. Defaults to 'treated'.
`outcome_column_name`	Name of the outcome column in `holdout` and also in `data`, if supplied in the latter. Defaults to 'outcome'.
`weights`	A positive numeric vector representing covariate importances. Supplying this argument prevents PE from being computed as it determines dropping order by forcing covariate subsets with lower weights to be dropped first. The weight of a covariate subset is defined to be the sum of the weights of the constituent covariates. Ties are broken at random.
`PE_method`	Denotes how predictive error (PE) is to be computed. Either a string – one of "ridge" (default) or "xgb" – or a function. If "ridge", ridge regression is used to fit a an outcome regression model via `glmnet::cv.glmnet` with default parameters. If "xgb", gradient boosting with a wide range of parameter values to cross-validate is used via `xgboost::xgb.cv` and the best parameters with respect to RMSE (for continuous outcomes) or misclassification rate (for binary/multi-class outcomes) are chosen. In both cases, the default `predict` method is used to generate in-sample predictions. If a function, denotes a user-supplied function that should be used for computing PE. This function must be passed a data frame of covariates as its first argument and a vector of outcome values as its second argument. It must return a vector of in-sample predictions, which, if the outcome is binary or multi-class, must be maximum probability class labels. See below for examples.
`user_PE_fit`	Deprecated; use argument 'PE_method' instead. An optional function supplied by the user that can be used instead of those allowed for by `PE_method` to fit a model for the outcome from the covariates. This function will be passed a data frame of covariates as its first argument and a vector of outcome values as its second argument. See below for examples. Defaults to `NULL`.
`user_PE_fit_params`	Deprecated; use argument 'PE_method' instead. A named list of optional parameters to be used by `user_PE_fit`. Defaults to `NULL`.
`user_PE_predict`	Deprecated; use argument 'PE_method' instead. An optional function supplied by the user that can be used to generate predictions from the output of `user_PE_fit`. As its first argument, must take an object of the type returned by `user_PE_fit` and as its second, a matrix of values for which to generate predictions. When the outcome is binary or multi-class, must return the maximum probability class label. If not supplied, defaults to `predict`.

`user_PE_predict_params`	Deprecated; use argument 'PE_method' instead. A named list of optional parameters to be used by `user_PE_predict`. Defaults to `NULL`.
`replace`	A logical scalar. If `TRUE`, allows the same unit to be matched multiple times, on different sets of covariates. In this case, the balancing factor for `FLAME` is computing by dividing by the total number of treatment (control) units, instead of the number of unmatched treatment (control) units. Defaults to `FALSE`.
`estimate_CATEs`	A logical scalar. If `TRUE`, CATEs for each unit are estimated throughout the matching procedure, which will be much faster than computing them after a call to `FLAME` or `DAME` for very large inputs. Defaults to `FALSE`.
`verbose`	Controls how FLAME displays progress while running. If 0, no output. If 1, only outputs the stopping condition. If 2, outputs the iteration and number of unmatched units every 5 iterations, and the stopping condition. If 3, outputs the iteration and number of unmatched units every iteration, and the stopping condition. Defaults to 2.
`return_pe`	A logical scalar. If `TRUE`, the predictive error (PE) at each iteration will be returned. Defaults to `FALSE`.
`return_bf`	A logical scalar. If `TRUE`, the balancing factor (BF) at each iteration will be returned. Defaults to `FALSE`.
`early_stop_iterations`	A positive integer, denoting an upper bound on the number of matching rounds to be performed. If 1, one round of exact matching is performed before stopping. Defaults to `Inf`.
`early_stop_epsilon`	A nonnegative numeric. If fixed covariate weights are passed via `weights`, then the algorithm will stop before matching on a covariate set whose error is above `early_stop_epsilon`, where in this case the error is defined as: $1 - weight(covariatesetmatchedon)/weight(allcovariates)$. Otherwise, if `weights` is `NULL`, if FLAME or DAME attempts to drop a covariate set that would raise the PE above (1 + `early_stop_epsilon`) times the baseline PE (the PE before any covariates have been dropped), the algorithm will stop. Defaults to 0.25.
`early_stop_control, early_stop_treated`	If the proportion of control, treated units, respectively, that are unmatched falls below this value, the matching algorithm will stop. Default to 0.
`early_stop_pe`	Deprecated. A positive numeric. If FLAME attempts to drop a covariate that would lead to a PE above this value, FLAME stops. Defaults to `Inf`.
`early_stop_bf`	Deprecated. A numeric value between 0 and 2. If FLAME attempts to drop a covariate that would lead to a BF below this value, FLAME stops. Defaults to 0.
`missing_data`	Specifies how to handle missingness in `data`. If 'none' (default), assumes no missing data. If 'drop', effectively drops units with missingness from the data and does not match them (they will still appear in the matched dataset that is returned, however). If 'keep', keeps the missing values in the data; in this case, a unit can only match on sets containing covariates it is not missing. If 'impute', imputes the missing data via `mice::mice`.

`missing_holdout`	Specifies how to handle missingness in `holdout`. If 'none' (default), assumes no missing data; if 'drop', drops units with missingness and does not use them to compute PE; and if 'impute', imputes the missing data via `mice::mice`. In this last case, the PE at an iteration will be given by the average PE across all imputations.
`missing_data_imputations`	Defunct. If `missing_data` = 'impute', one round of imputation will be performed on `data` via `mice::mice`. To view results for multiple imputations, please wrap calls to `FLAME` or `DAME` in a loop. This argument will be removed in a future release.
`missing_holdout_imputations`	If `missing_holdout` = 'impute', performs this many imputations of the missing data in `holdout` via `mice::mice`. Defaults to 5.
`impute_with_treatment, impute_with_outcome`	If `TRUE`, use treatment, outcome, respectively, to impute covariates when either `missing_data` or `missing_holdout` is equal to `'impute'`. Default to `TRUE`, `FALSE`, respectively.
`n_flame_iters`	Specifies that this many iterations of FLAME should be run before switching to DAME. This can be used to speed up the matching procedure as FLAME rapidly eliminates irrelevant covariates, after which DAME will make higher quality matches on the remaining variables.
`x`	An object of class `ame`, returned by a call to `FLAME` or `DAME`.
`digits`	Number of significant digits for printing the average treatment effect.
`linewidth`	Maximum number of characters on line; output will be wrapped accordingly.
`...`	Additional arguments to be passed to other methods.

Value

An object of type `ame`, which by default is a list of 4 entries:

data The original data frame with several modifications:

1. An extra logical column, `data$matched`, that indicates whether or not a unit was matched.
2. An extra numeric column, `data$weight`, that denotes on how many different sets of covariates a unit was matched. This will only be greater than 1 when `replace = TRUE`.
3. The columns denoting treatment and outcome will be moved after all covariate columns.
4. If `replace` is `FALSE`, a column containing a matched group identifier for each unit.
5. If, `estimate_CATEs = TRUE`, a column containing the CATE estimate for each unit.

MGs A list whose i'th entry contains the indices of units in the main matched group of the i'th unit.

cov_sets A list whose i'th entry contains the covariates set **not** matched on in the i'th iteration.

info A list containing miscellaneous information about the data and matching specifications. Primarily for use by `*.ame` methods.

Introduction

FLAME and DAME are matching algorithms for observational causal inference on data with discrete (categorical) covariates. They match units that share identical values of certain covariates, as follows. The algorithms first make any possible *exact* matches; that is, they match units that share identical values of all covariates (this is possible because covariates are discrete). They then iteratively drop a set of covariates and make any possible matches on the remaining covariates, until stopping. For each unit, DAME solves an optimization problem that finds the highest quality set of covariates the unit can be matched to others on, where quality is determined by how well that set of covariates predicts the outcome. FLAME approximates the solution to the problem solved by DAME; at each step, it drops the covariate leading to the smallest drop in match quality MQ, defined as $MQ = CBF - PE$. Here, PE denotes the predictive error, which measures how important the dropped covariate is for predicting the outcome. The balancing factor BF measures the number of matches formed by dropping that covariate. In this way, FLAME encourages matching on covariates more important to the outcome and also making many matches. The hyperparameter C controls the balance between these two objectives. In both cases, a machine learning algorithm trained on a holdout dataset is responsible for learning the quality / importance of covariates. For more details on the algorithms, please see the vignette, the FLAME paper here and/or the DAME paper here.

Stopping Rules

By default, both `FLAME` and `DAME` stop when 1. all covariates have been dropped or 2. all treatment or control units have been matched. This behavior can be modified by the arguments whose prefix is "early_stop". With the exception of `early_stop_iterations`, all the rules come into play *before* the offending covariate set is dropped. For example, if `early_stop_control = 0.2` and at the current iteration, dropping the covariate leading to highest match quality is associated with a unmatched control proportion of 0.1, FLAME will stop *without* dropping this covariate.

Missing Data

`FLAME` and `DAME` offer functionality for handling missing data in the covariates, for both the `data` and `holdout` sets. This functionality can be specified via the arguments whose prefix is "missing" or "impute". It allows for ignoring missing data, imputing it, or (for `data`) not matching on missing values. If `data` is imputed, imputation will be done once and the matching algorithm will be run on the imputed dataset. If `holdout` is imputed, the predictive error at an iteration will be the average of predictive errors across all imputed `holdout` datasets. Units with missingness in the treatment or outcome will be dropped.

Examples

```
## Not run:
data <- gen_data()
holdout <- gen_data()
# FLAME with replacement, stopping after dropping a single covariate
FLAME_out <- FLAME(data = data, holdout = holdout,
                   replace = TRUE, early_stop_iterations = 2)

# Use a linear model to compute predictive error. Call DAME without
# replacement, returning predictive error at each iteration.
```

```
my_PE <- function(X, Y) {
  return(lm(Y ~ ., as.data.frame(cbind(X, Y = Y)))$fitted.values)
}
DAME_out <- DAME(data = data, holdout = holdout,
                 PE_method = my_PE, return_PE = TRUE)

## End(Not run)
```

ATE *Average Treatment Effect estimates*

Description

These functions are deprecated and will be made defunct at a later release. See `summary.ame` for average treatment effects estimates and their variance.

Usage

```
ATE(ame_out)

ATT(ame_out)

ATC(ame_out)
```

Arguments

`ame_out`	An object of class `ame`.

Details

`ATE`, `ATT`, and `ATC` estimate the average treatment effect (ATE), average treatment effect on the treated (ATT), and average treatment effect on the controls (ATC), respectively, of a matched dataset.

The ATE is estimated as the average CATE estimate across all matched units in the data, while the ATT and ATC average only across matched treated or matched control units, respectively.

See Also

`CATE`

`CATE`	*Conditional Average Treatment Effects*

Description

`CATE` returns an estimate of the conditional average treatment effect for the subgroup defined by `units`.

Usage

```
CATE(units, ame_out)
```

Arguments

`units`	A vector of units whose CATE estimates are desired.
`ame_out`	An object of class `ame`.

Details

This function returns CATE estimates and the estimated variances of such estimates for `units` of interest. The CATE of a given unit is estimated by the difference between the average treated and the average control outcome in that unit's main matched group. As shown by Morucci 2021, under standard regularity conditions, such an estimator is asymptotically normal and unbiased for the true CATE, with a variance that can be estimated by the sum of the variance of treated and control outcomes in the matched group, each normalized by the number of treated and control units in the matched group, respectively. Note that CATEs cannot be estimated for unmatched units and estimator variances cannot be computed for units whose main matched group only contains a single treated or control unit. Note also that these CATE estimates differ from those that are used to compute average treatment effects in `print.ame` and `summary.ame` and from those that will be returned in `ame_out$data$CATE` if `estimate_CATEs = TRUE`. For a treated (control) unit i, the latter estimate the treated (control) outcome conditioned on $X = x_i$ simply as Y_i, and do not average across other treated (control) units in the matched group as is done here. This averaging is necessary in order to compute variance estimates. The different estimates can always be manually compared, though they are the same in expectation (assuming mean 0 noise) and we expect them to be similar in practice, in the absence of large noise.

Lastly, note that the `units` argument refers to units with respect to `rownames(ame_out$data)`. Typically, this will also correspond to the indexing of the data (i.e. passing `units = 3` will return the matched group of the 3rd unit in the matching data). However, if a separate holdout set was not passed to the matching algorithm or if the original matching data had rownames other than `1:nrow(data)`, then this is not the case.

Value

A matrix whose columns correspond to CATE estimates and their variances and whose rows correspond to queried units. `NA`'s therein correspond to inestimable quantities.

See Also

FLAME, DAME

Examples

```
## Not run:
data <- gen_data()
holdout <- gen_data()
FLAME_out <- FLAME(data = data, holdout = holdout)
CATE(1:5, FLAME_out)

## End(Not run)
```

---

gen_data *Generate Toy Data for Matching*

Description

`gen_data` generates toy data that can be used to explore FLAME and DAME functionality.

Usage

```
gen_data(n = 250, p = 5, write = FALSE, path = getwd(), filename = "AME.csv")
```

Arguments

`n`	Number of units desired in the data set. Defaults to 250.
`p`	Number of covariates in the data set. Must be greater than 2. Defaults to 5.
`write`	A logical scalar. If `TRUE`, the resulting data is stored as a .csv file as specified by arguments `path` and `filename`. Defaults to `FALSE`.
`path`	The path to the location where the data should be written if `write = TRUE`. Defaults to `getwd()`.
`filename`	The name of the file to which the data should be written if `write = TRUE`. Defaults to AME.csv.

Details

`gen_data` simulates data in the format accepted by FLAME and `link{DAME}`. Covariates X_i and treatment T are both independently generated according to a Bernoulli(0.5) distribution. The outcome Y is generated according to $Y = 15X_1 - 10X_2 + 5X_3 + 5T + \epsilon$, where $\epsilon \sim N(0, I_n)$. Thus, the value of `p` must be at least 3 and any additional covariates beyond X_1, X_2, X_3 are irrelevant.

Value

A data frame that may be passed to FLAME or DAME. Covariates are numeric, treatment is binary numeric and outcome is numeric.

MG *Matched Groups*

Description

MG returns the matched groups of the supplied units.

Usage

```
MG(units, ame_out, multiple = FALSE, id_only = FALSE, index_only)
```

Arguments

units	A vector of units whose matched groups are desired.
ame_out	An object of class ame.
multiple	A logical scalar. If FALSE (default), then MG will only return the main matched group for each unit. See below for details. Cannot be set to TRUE if ame_out was generated without replacement.
id_only	A logical scalar. If TRUE, then only the IDs of the units in each matched group are returned, and not their treatment, outcome, or covariate information.
index_only	Defunct. Use 'id_only' instead.

Details

The units argument refers to units with respect to rownames(ame_out$data). Typically, this will also correspond to the indexing of the data (i.e. passing units = 3 will return the matched group of the 3rd unit in the matching data). However, if a separate holdout set was not passed to the matching algorithm or if the original matching data had rownames other than 1:nrow(data), then this is not the case.

The multiple argument toggles whether only a unit's main matched group (MMG) or all matched groups a unit is part of should be returned. A unit's MMG contains its highest quality matches (that is, the units with which it first matched in the sequence of considered covariate sets). If the original call that generated ame_out specified replace = FALSE then units only are part of one matched group (which is also their MMG) and multiple must be set to FALSE.

Value

A list of length length(units), each entry of which corresponds to a different unit in units. For matched units, if multiple = FALSE, each entry is 1. a data frame containing the treatment and outcome information of members of the matched group, along with covariates they were matched on if id_only = FALSE or 2. a vector of the IDs of matched units if id_only = TRUE . If multiple = TRUE, each entry of the returned list is a list containing the previously described information, but with each entry corresponding to a different matched group. In either case, entries corresponding to unmatched units are NULL.

Examples

```
## Not run:
data <- gen_data()
holdout <- gen_data()
FLAME_out <- FLAME(data = data, holdout = holdout, replace = TRUE)

# Only the main matched group of unit 1
MG(1, FLAME_out, multiple = F)

# All matched groups of unit 1
MG(1, FLAME_out, multiple = T)

## End(Not run)
```

plot.ame *Plot a summary of FLAME or DAME*

Description

Plot information about numbers of covariates matched on, CATE estimates, and covariate set dropping order after a call to FLAME or DAME.

Usage

```
## S3 method for class 'ame'
plot(x, which_plots = c(1, 2, 3, 4), ...)
```

Arguments

x	An object of class ame, returned by a call to FLAME or link{DAME}.
which_plots	A vector describing which plots should be displayed. See details.
...	Additional arguments to passed on to other methods.

Details

plot.ame displays four plots by default. The first contains information on the number of covariates that matched groups were formed on, and thereby gives some indication of the quality of matched groups across the matched data. The second plots matched group sizes against CATEs, which can be useful for determining whether higher quality matched groups yield different treatment effect estimates than lower quality ones. The third plots a density estimate of the estimated CATE distribution. The fourth displays a heatmap showing which covariates were dropped (shown in black) throughout the matching procedure.

`summary.ame` *Summarize the output of FLAME or DAME*

Description

These methods create and print objects of class `summary.ame` containing information on the numbers of units matched by the AME algorithm, matched groups formed, and, if applicable, average treatment effects.

Usage

```
## S3 method for class 'ame'
summary(object, ...)

## S3 method for class 'summary.ame'
print(x, digits = 3, ...)
```

Arguments

`object`	An object of class `ame`, returned by a call to `FLAME` or `DAME`
`...`	Additional arguments to be passed on to other methods.
`x`	An object of class `summary.ame`, returned by a call to `summary.ame`
`digits`	Number of significant digits for printing the average treatment effect estimates and their variances.

Details

The average treatment effect (ATE) is estimated as the average CATE estimate across all matched units in the data, while the average treatment effect on the treated (ATT) and average treatment effect on controls (ATC) average only across matched treated or matched control units, respectively. Variances of these estimates are computed as in Abadie, Drukker, Herr, and Imbens (The Stata Journal, 2004) assuming constant treatment effect and homoscedasticity. Note that the implemented estimator is **not** =asymptotically normal and so in particular, asymptotically valid confidence intervals or hypothesis tests cannot be conducted on its basis. In the future, the estimation procedure will be changed to employ the nonparametric regression bias adjustment estimator of Abadie and Imbens 2011 which is asymptotically normal.

Value

A list of type `summary.ame` with the following entries:

MG A list with the number and median size of matched groups formed. Additionally, two of the highest quality matched groups formed. Quality is determined first by number of covariates matched on and second by matched group size.

n_matches A matrix detailing the number of treated and control units matched.

TEs If the matching data had a continuous outcome, estimates of the ATE, ATT, and ATC and the corresponding variance of the estimates.

Index