Package ‘fairml’

April 30, 2025

Type Package

Title Fair Models in Machine Learning

Version 0.9

Date 2025-04-29

Depends R (>= 3.5.0)

Imports methods, glmnet

Suggests lattice, gridExtra, parallel, cccp, CVXR, survival

Maintainer Marco Scutari <scutari@bnlearn.com>

Description Fair machine learning regression models which take sensitive attributes into account in model estimation. Currently implementing Komiyama et al. (2018) <http://proceedings.mlr.press/v80/komiyama18a/komiyama18a.pdf>, Zafar et al. (2019) <https://www.jmlr.org/papers/volume20/18-262/18-262.pdf> and my own approach from Scutari, Panero and Proissl (2022) <doi:10.1007/s11222-022-10143-w> that uses ridge regression to enforce fairness.

License MIT + file LICENSE

LazyData yes

NeedsCompilation no

Author Marco Scutari [aut, cre]

Repository CRAN

Date/Publication 2025-04-29 23:30:12 UTC

Contents

`fairml-package` *Fair Models in Machine Learning*

Description

Fair machine learning models: estimation, tuning and prediction.

Details

fairml implements key algorithms for learning machine learning models while enforcing fairness with respect to a set of observed sensitive (or protected) attributes.

Currently **fairml** implements the following algorithms (references below):

- `nclm()`: the non-convex formulation of fair linear regression model from Komiyama et al. (2018).
- `frrm()`: the fair (linear) ridge regression model from Scutari, Panero and Proissl (2022).
- `fgrrm()`: thefair generalized (linear) ridge regression model from Scutari, Panero and Proissl (2022), supporting the Gaussian, binomial, Poisson, multinomial and Cox (proportional hazards) families.
- `zlrm()`: the fair logistic regression with covariance constraints from Zafar et al. (2019).
- `zlrm()`: a fair linear regression with covariance constraints following Zafar et al. (2019).

Furthermore, different fairness definitions can be used in `frrm()` and `fgrrm()`:

- `"sp-komiyama"`: the statistical parity fairness constraint from Komiyama et al. (2018);
- `"eo-komiyama"`: the analogous equality of opportunity constraint built on the proportion of variance (or deviance) explained by sensitive attributes;
- `"if-berk"`: the individual fairness constraint from Berk et al. (2017) adapted in Scutari, Panero and Proissl (2022);
- user-provided functions for custom definitions.

In addition, **fairml** implements diagnostic plots, cross-validation, prediction and methods for most of the generics made available for linear models from `lm()` and `glm()`. Profile plots to trace key model and goodness-of-fit indicators at varying levels of fairness are available from `fairness.profile.plot()`.

Author(s)

Marco Scutari
Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA)

Maintainer: Marco Scutari <`scutari@bnlearn.com`>

References

Berk R, Heidari H, Jabbari S, Joseph M, Kearns M, Morgenstern J, Neel S, Roth A (2017). "A Convex Framework for Fair Regression". FATML.
`https://www.fatml.org/media/documents/convex_framework_for_fair_regression.pdf`

Komiyama J, Takeda A, Honda J, Shimao H (2018). "Nonconvex Optimization for Regression with Fairness Constraints". Proceedings of the 35th International Conference on Machine Learning (ICML), PMLR **80**:2737–2746.
`http://proceedings.mlr.press/v80/komiyama18a/komiyama18a.pdf`

Scutari M, Panero F, Proissl M (2022). "Achieving Fairness with a Simple Ridge Penalty". Statistics and Computing, **32**, 77.
`https://link.springer.com/content/pdf/10.1007/s11222-022-10143-w.pdf`

Zafar BJ, Valera I, Gomez-Rodriguez M, Gummadi KP (2019). "Fairness Constraints: a Flexible Approach for Fair Classification". Journal of Machine Learning Research, 30:1–42.
`https://www.jmlr.org/papers/volume20/18-262/18-262.pdf`

`adult` *Census Income*

Description

Predict whether income exceeds $50K per year using the U.S. 1994 Census data.

Usage

```
data(adult)
```

Format

The data contains 30162 observations and 14 variables. See the UCI Machine Learning Repository for details.

Note

The data set has been pre-processed as in Zafar et al. (2019), with the following exceptions:

- the data do not include the test sample from the UCI repository;
- the variables "capital_gain" and "capital_loss" have been scaled by 1/1000.

In that paper, income is the response variable, sex and race are the sensitive attributes and the remaining variables are used as predictors.

The data contain the following variables:

- age as a numeric variable;
- workclass, a factor with 8 levels encoding the type of employment ("Private", "Self-emp-not-inc", "Federal-gov", etc.);
- education, a factor with 10 levels from "Preschool" to "Doctorate";
- education-num, the number of years in education;
- marital-status, a factor with 7 levels from "Married-civ-spouse" to "Divorced" and "Never-married";
- occupation, a factor with 14 levels encoding the field of employment ("Tech-support", "Craft-repair", etc.);
- relationship a factor with 6 levels ("Wife", "Own-child", etc.);
- race, a factor with levels "White", "Asian-Pac-Islander", "Amer-Indian-Eskimo", "Other" and "Black";
- sex, a factor with levels "Female" and "Male";
- capital-gain as a numeric variable;
- capital-loss as a numeric variable;
- native-country as a factor with two levels "United-States" and "Non-United-States";
- hours-per-week as a numeric variable.

References

UCI Machine Learning Repository.
https://archive.ics.uci.edu/ml/datasets/adult

Examples

```
data(adult)

# short-hand variable names.
r = adult[, "income"]
s = adult[, c("sex", "race")]
p = adult[, setdiff(names(adult), c("income", "sex", "race"))]

## Not run:
m = zlrm(response = r, sensitive = s, predictors = p, unfairness = 0.05)
summary(m)

## End(Not run)
```

bank *Bank Marketing*

Description

Direct marketing campaigns (phone calls) of a Portuguese banking institution to make clients subscribe a term deposit.

Usage

```
data(bank)
```

Format

The data contains 41188 observations and 19 variables. See the UCI Machine Learning Repository for details.

Note

The data set has been pre-processed as in Zafar et al. (2019), with the following exceptions:

- the variable `duration` has been dropped in order to learn as realistic predictive model;
- the variable pdays has been dropped because it is not defined for the vast majority of samples;
- observations where `loan` is `"unknown"` have been dropped because the corresponding regression coefficient estimated by `glm()` is `NA`;
- the three observations where `default` is `"yes"` have been dropped to avoid errors in cross-validation (if all those three observations are in the test fold it is impossible to compute predictions from them).

In that paper, `subscribed` is the response variable, `age` is the sensitive attribute and the remaining variables are used as predictors.

The data contains the following variables:

- `age` as a numeric variable;
- `job`, a factor with 12 levels ranging from `"blue-collar"` to `"services"`;
- `marital`, a factor with levels `"divorced"`, `"married"`, `"single"` and `"unknown"`;
- `education`, a factor with 8 levels ranging from `"basic.4y"` to `"university.degree"`;
- `default`, a factor with levels `"no"` and `"unknown"`;
- `housing`, a factor with levels `"yes"` and `"no"`;
- `loan`, a factor with levels `"yes"` and `"no"`;
- `contact`, a factor with levels `"cellular"` and `"telephone"`;
- `month`, a factor with 12 levels for the months of the year;
- `day_of_week`, a factor with 7 levels for the days of the week;

- campaign, the number of contacts performed during this campaign;
- previous, the number of contacts performed before this campaign;
- poutcome, a factor with levels "failure", "nonexistent" and "success";
- emp_var_rate, the (numeric) quarterly employment variation rate;
- cons_price_idx, the (numeric) monthly consumer price index;
- cons_conf_idx, the (numeric) monthly consumer confidence index;
- euribor3m, the (numeric) euribor 3-month rate;
- nr_employed, a numeric variable with the number of employees in the company in that quarter;
- subscribed, a factor with levels "yes" and "no".

References

UCI Machine Learning Repository.
https://archive.ics.uci.edu/ml/datasets/bank+marketing

Examples

```
data(bank)

# remove loans with unknown status, the corresponding coefficient is NA in glm().
bank = bank[bank$loan != "unknown", ]

# short-hand variable names.
r = bank[, "subscribed"]
s = bank[, c("age")]
p = bank[, setdiff(names(bank), c("subscribed", "age"))]

## Not run:
m = zlrm(response = r, sensitive = s, predictors = p, unfairness = 0.05)
summary(m)

## End(Not run)
```

---

communities.and.crime *Communities and Crime Data Set*

Description

Combined socio-economic data from the 1990 Census, law enforcement data from the 1990 LEMAS survey, and crime data from the 1995 FBI UCR for various communities in the United States.

Usage

```
data(communities.and.crime)
```

Format

The data contains 1969 observations and 104 variables. See the UCI Machine Learning Repository for details.

Note

The data set has been pre-processed as in Komiyama et al. (2018), with the following exceptions:

- the variable community has been dropped, as it is non-predictive and contains a sizeable number of missing values;
- the variables LemasSwornFT, LemasSwFTPerPop, LemasSwFTFieldOps, LemasSwFTFieldPerPop, LemasTotalReq, LemasTotReqPerPop, PolicReqPerOffic, PolicPerPop, RacialMatchCommPol, PctPolicWhite, PctPolicBlack, PctPolicHisp, PctPolicAsian, PctPolicMinor, OfficAssgnDrugUnits, NumKindsDrugsSeiz, PolicAveOTWorked, PolicCars, PolicOperBudg, LemasPctPolicOnPatr, LemasGangUnitDeploy and PolicBudgPerPop have been dropped because they have more than 80% missing values.

In that paper, ViolentCrimesPerPop is the response variable, racepctblack and PctForeignBorn are the sensitive attributes and the remaining variables are used as predictors.

The data contain too many variable to list them here: we refer the reader to the documentation on the UCI Machine Learning Repository.

References

UCI Machine Learning Repository:
http://archive.ics.uci.edu/ml/datasets/communities+and+crime

Examples

```
data(communities.and.crime)

# short-hand variable names.
cc = communities.and.crime[complete.cases(communities.and.crime), ]
r = cc[, "ViolentCrimesPerPop"]
s = cc[, c("racepctblack", "PctForeignBorn")]
p = cc[, setdiff(names(cc), c("ViolentCrimesPerPop", names(s)))]

m = nclm(response = r, sensitive = s, predictors = p, unfairness = 0.05)
summary(m)

m = frrm(response = r, sensitive = s, predictors = p, unfairness = 0.05)
summary(m)
```

compas *Criminal Offenders Screened in Florida*

Description

A collection of criminal offenders screened in Florida (US) during 2013-14.

Usage

```
data(compas)
```

Format

The data contains 5855 observations and the following variables:

- `age`, a continuous variable containing the age (in years) of the person;
- `juv_fel_count`, a continuous variable containing the number of juvenile felonies;
- `decile_score`, a continuous variable, the decile of the COMPAS score;
- `juv_misd_count`, a continuous variable containing the number of juvenile misdemeanors;
- `juv_other_count`, a continuous variable containing the number of prior juvenile convictions that are not considered either felonies or misdemeanors;
- `v_decile_score`, a continuous variable containing the predicted decile of the COMPAS score;
- `priors_count`, a continuous variable containing the number of prior crimes committed;
- `sex`, a factor with levels `"Female"` and `"Male"`;
- `two_year_recid`, a factor with two levels `"Yes"` and `"No"` (if the person has recidivated within two years);
- `race`, a factor encoding the race of the person;
- `c_jail_in`, a numeric variable containing the date in which the person entered jail (normalized between 0 and 1);
- `c_jail_out`, a numeric variable containing the date in which the person was released from jail (normalized between 0 and 1);
- `c_offense_date`, a numeric variable containing the date the offense was committed;
- `screening_date`, a numeric variable containing the date in which the person was screened (normalized between 0 and 1);
- `in_custody`, a numeric variable containing the date in which the person was placed in custody (normalized between 0 and 1);
- `out_custody`, a numeric variable containing the date in which the person was released from custody (normalized between 0 and 1);

Note

The data set has been pre-processed as in Komiyama et al. (2018), with the following exceptions:

- the `race` variable has not been reduced to a binary factor with levels `"African-American"` and `"not African-American"`;
- the variables `type_of_assessment`, `v_type_of_assessment` have been dropped from the analysis because they take the same value for all observations;
- variables like `c_jail_in` and `c_jail_out` that encode dates have been jointly rescaled to preserve the temporal ordering of events.

In that paper, `two_year_recid` is the response variable, `sex` and `race` are the sensitive attributes and the remaining variables are used as predictors.

References

Angwin J, Larson J, Mattu S, Kirchner L (2016). "Machine Bias: Theres Software Used Around the Country to Predict Future Criminals."
`https://www.propublica.org`

Examples

```
data(compas)

# convert the response back to a numeric variable.
compas$two_year_recid = as.numeric(compas$two_year_recid) - 1

# short-hand variable names.
r = compas[, "two_year_recid"]
s = compas[, c("sex", "race")]
p = compas[, setdiff(names(compas), c("two_year_recid", "sex", "race"))]

m = nclm(response = r, sensitive = s, predictors = p, unfairness = 0.05)
summary(m)

m = frrm(response = r, sensitive = s, predictors = p, unfairness = 0.05)
summary(m)
```

`confint.fair.model` *Confidence Intervals for Fair Models*

Description

Confidence intervals for the parameters of the models in the **fairml** package.

Usage

```
## S3 method for class 'fair.model'
confint(object, parm, level = 0.95, method = "boot",
  method.args = list(), ...)

## S3 method for class 'fair.confint'
plot(x, support = FALSE, ...)
```

Arguments

`object`	an object of class `fair.model`.
`parm`	a character vector, the names of the parameters to compute the confidence intervals for. The default is to do that for all parameters.
`level`	a number between 0 and 1, the coverage of the confidence intervals.
`method`	a character string, the method used to compute the confidence intervals. See below for details.
`method.args`	optional arguments passed to the `method`.
`...`	additional arguments (unused).
`x`	an object of class `fair.confint`.
`support`	a logical value, whether to draw a vertical line at zero.

Details

The only available `method` is `"boot"`, which implements nonparametric bootstrap with observation resampling. It has the following optional arguments:

- `response`, `predictors`, `sensitive`: the data the model was fitted from.
- `R`: a positive integer number, the number of bootstrap samples. The default value is `200`.
- `m`: a positive integer number, the size of bootstrap samples. The default value is the size of the original sample.

Value

`confint()` returns an object of class `fair.confint` which wraps a two- or three-dimensinal matrix. The upper and lower bounds of the confidence intervals in the columns, the variables are in the rows.

Author(s)

Marco Scutari

Examples

```
mgaus = fgrrm(response = vu.test$gaussian, predictors = vu.test$X,
          sensitive = vu.test$S, unfairness = 0.05, family = "gaussian")
ci = confint(mgaus, method = "boot",
       method.args = list(response = vu.test$gaussian, predictors = vu.test$X,
                          sensitive = vu.test$S, R = 20))
```

```
ci
plot(ci)
```

drug.consumption *Drug Consumption*

Description

Predict drug consumption based on psychological scores and demographics.

Usage

```
data(drug.consumption)
```

Format

The data contains 1885 observations and 31 variables. See the UCI Machine Learning Repository for details.

Note

The data set has been minimally pre-processed following the instructions on the UCI Machine Learning Repository to re-encode the variables. Categorical variables are stored as factors and the psychological scores are stored as numeric variables on their original scales.

Any of the drug use variables can be used as the response variable (with 7 different levels); `Age`, `Gender` and `Race` are the sensitive attributes. The remaining variables are used as predictors.

The data contain the following variables:

- `Age`, a factor with 6 10-years age brackets;
- `Gender`, as a factor;
- `Education`, a factor with 9 levels from `"Left school before 16"` to `"Doctorate degree"`;
- `Country`, a factor with 7 different levels for `"USA"`, `"New Zealand"`, `"Other"`, `"Australia"`, `"Republic of Ireland"` `"Canada"` and `"UK"`;
- `Race` a factor with 7 levels comprising mixed backgrounds as well;
- `Nscore`, `Escore`, `Oscore`, `Ascore`, `Cscore`, numeric scores from the five-factor model for personality traits;
- `Impulsive`, a numeric score for impulsivity;
- `SS`, a numeric score for sensation seeking;
- `Alcohol`, `Amphet`, `Amyl`, `Benzos`, `Caff`, `Cannabis`, `Choc`, `Coke`, `Crack`, `Ecstasy`, `Heroin`, `Ketamine`, `Legalh`, `LSD`, `Meth`, `Mushrooms`, `Nicotine`, `Semer` and `VSA`: factors with 7 levels ranging from `"Never Used"` to `"Used in Last Day"`.

References

UCI Machine Learning Repository.
`https://archive-beta.ics.uci.edu/dataset/373/`

Examples

```
data(drug.consumption)

# short-hand variable names.
r = drug.consumption[, "Meth"]
s = drug.consumption[, c("Age", "Gender", "Race")]
p = drug.consumption[, c("Education", "Nscore", "Escore", "Oscore", "Ascore",
                         "Cscore", "Impulsive", "SS")]

# collapse levels with low observed frequencies.
levels(p$Education) =
  c("at.most.18y", "at.most.18y", "at.most.18y", "at.most.18y", "university",
    "diploma", "bachelor", "master", "phd")

## Not run:
m = fgrrm(response = r, sensitive = s, predictors = p, ,
      family = "multinomial", unfairness = 0.05)
summary(m)

HH = drug.consumption$Heroin
levels(HH) = c("Never Used", "Used", "Used", "Used", "Used Recently",
               "Used Recently", "Used Recently")

m = fgrrm(response = HH, sensitive = s, predictors = p, ,
      family = "multinomial", unfairness = 0.05)
summary(m)

## End(Not run)
```

`fairml.cv` *Cross-Validation for Fair Models*

Description

Cross-validation for the models in the **fairml** package.

Usage

```
fairml.cv(response, predictors, sensitive, method = "k-fold", ..., unfairness,
  model, model.args = list(), cluster)

cv.loss(x)
cv.unfairness(x)
cv.folds(x)
```

Arguments

`response` a numeric vector, the response variable.

`predictors` a numeric matrix or a data frame containing numeric and factor columns; the predictors.

`sensitive` a numeric matrix or a data frame containing numeric and factor columns; the sensitive attributes.

`method` a character string, either `k-fold`, `custom-folds` or `hold-out`. See below for details.

`...` additional arguments for the cross-validation `method`.

`unfairness` a positive number in [0, 1], the proportion of the explained variance that can be attributed to the sensitive attributes.

`model` a character string, the label of the model. Currently `"nclm"`, `"frrm"`, `"fgrrm"`, `"zlm"` and `"zlrm"` are available.

`model.args` additional arguments passed to model estimation.

`cluster` an optional cluster object from package **parallel**, to process folds or subsamples in parallel.

`x` an object of class `fair.kcv` or `fair.kcv.list`.

Details

The following cross-validation methods are implemented:

- *k-fold*: the data are split in `k` subsets of equal size. For each subset in turn, `model` is fitted on the other `k - 1` subsets and the loss function is then computed using that subset. Loss estimates for each of the `k` subsets are then combined to give an overall loss for data.
- *custom-folds*: the data are manually partitioned by the user into subsets, which are then used as in k-fold cross-validation. Subsets are not constrained to have the same size, and every observation must be assigned to one subset.
- *hold-out*: `k` subsamples of size `m` are sampled independently without replacement from the data. For each subsample, `model` is fitted on the remaining `m - length(response)` samples and the loss function is computed on the `m` observations in the subsample. The overall loss estimate is the average of the `k` loss estimates from the subsamples.

Cross-validation methods accept the following optional arguments:

- `k`: a positive integer number, the number of groups into which the data will be split (in k-fold cross-validation) or the number of times the data will be split in training and test samples (in hold-out cross-validation).
- `m`: a positive integer number, the size of the test set in hold-out cross-validation.
- `runs`: a positive integer number, the number of times k-fold or hold-out cross-validation will be run.
- `folds`: a list in which element corresponds to one fold and contains the indices for the observations that are included to that fold; or a list with an element for each run, in which each element is itself a list of the folds to be used for that run.

If cross-validation is used with multiple `runs`, the overall loss is the average of the loss estimates from the different runs.

The predictive performance of the models is measured using the mean square error as the loss function.

Value

fairml.cv() returns an object of class fair.kcv.list if runs is at least 2, an object of class fair.kcv if runs is equal to 1.

cv.loss() returns a numeric vector or a numeric matrix containing the values of the loss function computed for each run of cross-validation.

cv.unfairness() returns a numeric vectors containing the values of the unfairness criterion computed on the validation folds for each run of cross-validation.

cv.folds() returns a list containing the indexes of the observations in each of the cross-validation folds. In the case of k-fold cross-validation, if runs is larger than 1, each element of the list is itself a list with the indexes for the observations in each fold in each run.

Author(s)

Marco Scutari

Examples

```
kcv = fairml.cv(response = vu.test$gaussian, predictors = vu.test$X,
        sensitive = vu.test$S, unfairness = 0.10, model = "nclm",
        method = "k-fold", k = 10, runs = 10)
kcv
cv.loss(kcv)
cv.unfairness(kcv)

# run a second cross-validation with the same folds.
fairml.cv(response = vu.test$gaussian, predictors = vu.test$X,
        sensitive = vu.test$S, unfairness = 0.10, model = "nclm",
        method = "custom-folds", folds = cv.folds(kcv))

# run cross-validation in parallel.
## Not run:
library(parallel)
cl = makeCluster(2)
fairml.cv(response = vu.test$gaussian, predictors = vu.test$X,
  sensitive = vu.test$S, unfairness = 0.10, model = "nclm",
  method = "k-fold", k = 5, runs = 5, cluster = cl)
stopCluster(cl)

## End(Not run)
```

fairness.profile.plot *Profile Fair Models with Respect to Tuning Parameters*

Description

Visually explore various aspect of a model over the range of possible values of the tuning parameters that control its fairness.

Usage

```
fairness.profile.plot(response, predictors, sensitive, unfairness,
  legend = FALSE, type = "coefficients", model, model.args = list(), cluster)
```

Arguments

`response` a numeric vector, the response variable.

`predictors` a numeric matrix or a data frame containing numeric and factor columns; the predictors.

`sensitive` a numeric matrix or a data frame containing numeric and factor columns; the sensitive attributes.

`unfairness` a vector of positive numbers in [0, 1], how unfair is the model allowed to be. The default value is `seq(from = 0.00, to = 1, by = 0.02)`.

`legend` a logical value, whether to add a legend to the plot.

`type` a character string, either `"coefficients"` (the default), `"constraints"`, `"precision-recall"` or `"rmse"`.

`model` a character string, the label of the model. Currently `"nclm"`, `"frrm"`, `"fgrrm"`, `"zlm"` and `"zlrm"` are available.

`model.args` additional arguments passed to model estimation.

`cluster` an optional cluster object from package **parallel**, to fit models in parallel.

Details

`fairness.profile.plot()` fits the model for all the values of the argument `unfairness`, and produces a profile plot of the regression coefficients or the proportion of explained variance.

If `type = "coefficients"`, the coefficients of the model are plotted against the values of `unfairness`.

If `type = "constraints"`, the following quantities are plotted against the values of `unfairness`:

1. For model `"nclm"`, and model `"frrm"` with `definition = "sp-komiyama"`:
   (a) the proportion of variance explained by the sensitive attributes (with respect to the response);
   (b) the proportion of variance explained by the predictors (with respect to the response);
   (c) the proportion of variance explained by the sensitive attributes (with respect to the combined sensitive attributes and predictors).
2. For model `"frrm"` with `definition = "eo-komiyama"`:
   (a) the proportion of variance explained by the sensitive attributes (with respect to the fitted values);
   (b) the proportion of variance explained by the response (with respect to the fitted values);
   (c) the proportion of variance explained by the sensitive attributes (with respect to the combined sensitive attributes and response).
3. For model `"frrm"` with `definition = "if-berk"`, the ratio between the individual fairness loss computed for a given values of the constraint and that of the unrestricted model with `unfairness = 1`.

4. For model "fgrrm": same as for "frrm" for each definition.
5. For models "zlm" and "zlrm": the correlations between the fitted values (from fitted() with type = "link") and the sensitive attributes.

If type = "precision-recall" and the model is a classifier, the precision, recall and F1 measures are plotted against the values of unfairness.

If type = "rmse" and the model is a linear regression, the residuals mean square error are plotted against the values of unfairness.

Value

A trellis object containing a **lattice** plot.

Author(s)

Marco Scutari

Examples

```
data(vu.test)
fairness.profile.plot(response = vu.test$gaussian, predictors = vu.test$X,
  sensitive = vu.test$S, type = "coefficients", model = "nclm", legend = TRUE)
fairness.profile.plot(response = vu.test$gaussian, predictors = vu.test$X,
  sensitive = vu.test$S, type = "constraints", model = "nclm", legend = TRUE)
fairness.profile.plot(response = vu.test$gaussian, predictors = vu.test$X,
  sensitive = vu.test$S, type = "rmse", model = "nclm", legend = TRUE)

# profile plots fitting models in parallel.
## Not run:
library(parallel)
cl = makeCluster(2)
fairness.profile.plot(response = vu.test$gaussian, predictors = vu.test$X,
  sensitive = vu.test$S, model = "nclm", cluster = cl)
stopCluster(cl)

## End(Not run)
```

flchain *Obesity Levels*

Description

A re-analysis of the flchain data set in the **survival** package.

References

Keya KN, Islam R, Pan S, Stockwell I, Foulds J (2020). Equitable Allocation of Healthcare Resources with Fair Cox Models. Proceedings of the 2021 SIAM International Conference on Data Mining (SDM), 190–198.
https://epubs.siam.org/doi/pdf/10.1137/1.9781611976700.22

Examples

```
library(survival)
data(flchain)

# complete data analysis.
flchain = flchain[complete.cases(flchain), ]
# short-hand variable names.
r = cbind(time = flchain$futime + 1, status = flchain$death)
s = flchain[, c(”age”, ”sex”)]
p = flchain[, c(”sample.yr”, ”kappa”, ”lambda”, ”flc.grp”, ”creatinine”, ”mgus”,
                ”chapter”)]

## Not run:
m = fgrrm(response = r, sensitive = s, predictors = p, family = ”cox”,
          unfairness = 0.05)
summary(m)

## End(Not run)
```

`frrm` *Fair Ridge Regression Model*

Description

A regression model enforcing fairness with a ridge penalty.

Usage

```
# a fair ridge regression model.
frrm(response, predictors, sensitive, unfairness,
  definition = "sp-komiyama", lambda = 0, save.auxiliary = FALSE)
# a fair generalized ridge regression model.
fgrrm(response, predictors, sensitive, unfairness,
  definition = "sp-komiyama", family = "binomial", lambda = 0,
  save.auxiliary = FALSE)
```

Arguments

`response` a numeric vector, the response variable.

`predictors` a numeric matrix or a data frame containing numeric and factor columns; the predictors.

`sensitive` a numeric matrix or a data frame containing numeric and factor columns; the sensitive attributes.

`unfairness` a positive number in [0, 1], how unfair is the model allowed to be. A value of `0` means the model is completely fair, while a value of `1` means the model is not constrained to be fair at all.

`definition` a character string, the label of the definition of fairness used in fitting the model. Currently either `"sp-komiyama"`, `"eo-komiyama"` or `"if-berk"`. It may also be a function: see below for details.

`family` a character string, either `"gaussian"` to fit a linear regression, `"binomial"` to fit a logistic regression, `"poisson"` to fit a log-linear regression, `"cox"` to fit a Cox proportional hazards regression of `"multinomial"` to fit a multinomial logistic regression.

`lambda` a non-negative number, a ridge-regression penalty coefficient. It defaults to zero.

`save.auxiliary` a logical value, whether to save the fitted values and the residuals of the auxiliary model that constructs the decorrelated predictors. The default value is `FALSE`.

Details

`frrm()` and `fgrrm()` can accommodate different definitions of fairness, which can be selected via the `definition` argument. The labels for the built-in definitions are:

- `"sp-komiyama"` for the same definition of fairness as `nclm()`: the model bounds the proportion of the variance that is explained by the sensitive attributes over the total explained variance. This falls within the definition of statistical parity.
- `"eo-komiyama"` enforces equality of opportunity in a similar way: it regresses the fitted values against the sensitive attributes and the response, and it bounds the proportion of the variance explained by the sensitive attributes over the total explained variance in that model.
- `"if-berk"` enforces individual fairness by penalizing the model for each pair of observations with different values of the sensitive attributes and different responses.

Users may also pass a function via the `definition` argument to plug custom fairness definitions. This function should have signature `function(model, y, S, U, family)` and return an array with an element called `"value"` (optionally along with others). The arguments will contain the model fitted for the current level of fairness (`model`), the sanitized response variable (`y`), the design matrix for the sanitized sensitive attributes (`S`), the design matrix for the sanitized decorrelated predictors (`U`) and the character string identifying the family the model belongs to (`family`).

The algorithm works like this:

1. regresses the predictors against the sensitive attributes;
2. constructs a new set of predictors that are decorrelated from the sensitive attributes using the residuals of this regression;
3. regresses the response against the decorrelated predictors and the sensitive attributes; while
4. using a ridge penalty to control the proportion of variance the sensitive attributes can explain with respect to the overall explained variance of the model.

Both `sensitive` and `predictors` are standardized internally before estimating the regression coefficients, which are then rescaled back to match the original scales of the variables.

`fgrrm()` is the extension of `frrm()` to generalized linear models, currently implementing linear (`family = "gaussian"`) and logistic (`family = "binomial"`) regressions. `fgrrm()` is equivalent to `frrm()` with `family = "gaussian"`. The definition of fairness are identical between `frrm()` and `fgrrm()`.

Value

`frrm()` returns an object of class `c("frrm", "fair.model")`. `fgrrm()` returns an object of class `c("fgrrm", "fair.model")`.

Author(s)

Marco Scutari

References

Scutari M, Panero F, Proissl M (2022). "Achieving Fairness with a Simple Ridge Penalty". Statistics and Computing, **32**, 77.
`https://link.springer.com/content/pdf/10.1007/s11222-022-10143-w.pdf`

See Also

nclm, zlm, zlrm

`german.credit` *German Credit Data*

Description

A credit scoring data set that can be used to predict defaults on consumer loans in the German market.

Usage

```
data(german.credit)
```

Format

The data contains 1000 observations (700 good loans, 300 bad loans) and the following variables:

- `Account_status`: a factor with four levels representing the amount of money in the account or `"no chcking account"`.
- `Duration`: a continuous variable, the duration in months.
- `Credit_history`: a factor with five levels representing possible credit history backgrounds.
- `Purpose`: a factor with ten levels representing possible reasons for taking out a loan.
- `Credit_amount`: a continuous variable.
- `Savings_bonds`: a factor with five levels representing amount of money available in savings and bonds or `"unknown / no savings account"`.
- `Present_employment_since`: a factor with five levels representing the length of tenure in the current employment or `"unemployed"`.
- `Installment_rate`: a continuous variable, the installment rate in percentage of disposable income.

- Other_debtors_guarantors: a factor with levels "none", "co-applicant" and "guarantor".
- Resident_since: a continuous variable, number of years in the current residence.
- Property: a factor with four levels describing the type of property to be bought or "unknown / no property".
- Age: a continuous variable, the age in years.
- Other_installment_plans: a factor with levels "bank", "none" and "stores".
- Housing: a factor with levels "rent", "own" and "for free".
- Existing_credits: a continuous variable, the number of existing credit lines at this bank.
- Job: a factor with four levels for different job descriptions.
- People_maintenance_for: a continuous variable, the number of people being liable to provide maintenance for.
- Telephone: a factor with levels "none" and "yes".
- Foreign_worker: a factor with levels "no" and "yes".
- Credit_risk: a factor with levels "BAD" and "GOOD".
- Gender: a factor with levels "Male" and "Female".

Note

The variable "Personal status and sex" in the original data has been transformed into Gender by dropping the personal status information.

References

UCI Machine Learning Repository:
https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data)

---

health.retirement *Health and Retirement Survey*

Description

The University of Michigan Health and Retirement Study (HRS) longitudinal dataset.

Usage

```
data(health.retirement)
```

Format

The data contains 38653 observations and 27 variables.

Note

The data set has been minimally pre-processed: the redundant variables `HISPANIC` and `BITHYR` were removed, along with the patient ID `PID`. A single patient was recorded twice: the duplicate has been removed. However, incomplete observations have been left in the data set.

The number of dependencies in daily activities `score` is the response (count) variable and `marriage`, `gender`, `race`, `race.ethnicity` and `age` are the sensitive attributes. The remaining variables are used as predictors.

The data contain the following variables:

- `year`, the year of retirement as a numeric variable;
- `age`, the age as a numeric variable;
- `educa`, the number of years in education as a numeric variable;
- `networth`, household net worth as a numeric variable;
- `cognition_catnew` cognistion assessment as a numeric variable;
- `bmi` as a numeric variable;
- `hlthrte`, a numeric health rating;
- `bloodp`, blood pressure diagnosis as a numeric variable;
- `diabetes`, diabetes diagnosis as a numeric variable;
- `cancer`, cancer diagnosis as a numeric variable;
- `lung`, lung disease diagnosis as a numeric variable;
- `heart`, heart condition diagnosis as a numeric variable;
- `stroke`, stroke diagnosis as a numeric variable;
- `pchiat`, psychiatric condition diagnosis as a numeric variable;
- `arthrit`, arthritis diagnosis as a numeric variable;
- `fall`, recently falling as a numeric variable;
- `pain`, pain conditions as a numeric variable;
- `A1c_adj`, biomarker for hemoglobin A1C;
- `CRP_adj`, biomarker for C-reactive protein;
- `CYSC_adj`, biomarker for Cystatin C;
- `HDL_adj`, biomarker for HDL cholesterol;
- `TC_adj`, biomarker for total cholesterol;
- `score`, another numeric health rating;
- `gender`, a factor with levels `"Female"` and `"Male"`;
- `marriage`, a factor with levels `"Married/Partner"` and `"Not Married"`;
- `race`, a factor withe levels `"Black"`, `"Other"` and `"White"`;
- `race.ethnicity`, a factor withe levels `"Hispanic"`, `"NHB"`, `"NHW"` and `"Other"`.

References

`https://hrs.isr.umich.edu/about`

Examples

```
data(health.retirement)

# complete data analysis.
health.retirement = health.retirement[complete.cases(health.retirement), ]
# short-hand variable names.
r = health.retirement[, "score"]
s = health.retirement[, c("marriage", "gender", "race", "age")]
p = health.retirement[, setdiff(names(health.retirement), c(names(r), names(s)))]
# drop the second race variable.
p = p[, colnames(p) != "race.ethnicity"]

## Not run:
# the lambda = 0.1 is very helpful in making model estimation succeed.
m = fgrrm(response = r, sensitive = s, predictors = p, ,
      family = "poisson", unfairness = 0.05, lambda = 0.1)
summary(m)

## End(Not run)
```

`law.school.admissions` *Law School Admission Council data*

Description

Survey among students attending law school in the U.S. in 1991.

Usage

```
data(law.school.admissions)
```

Format

The data contains 20800 observations and the following variables:

- `age`, a continuous variable containing the student's age in years;
- `decile1`, a continuous variable containing the student's decile in the school given his grades in Year 1;
- `decile3`, a continuous variable containing the student's decile in the school given his grades in Year 3;
- `fam_inc`, a continuous variable containing student's family income bracket (from 1 to 5);
- `lsat`, a continuous variable containing the student's LSAT score;
- `ugpa`, a continuous variable containing the student's undergraduate GPA;
- `gender`, a factor with levels `"female"` and `"male"`;
- `race1`, a factor with levels `"asian"`, `"black"`, `"hisp"`, `"other"` and `"white"`;

- `cluster`, a factor with levels `"1"`, `"2"`, `"3"`, `"4"`, `"5"` and `"6"` encoding the tiers of law school prestige;
- `fulltime`, a factor with levels `"FALSE"` and `"TRUE"`, whether the student will work full-time or part-time;
- `bar`, a factor with levels `"FALSE"` and `"TRUE"`, whether the student passed the bar exam on the first try.

Note

The data set has been pre-processed as in Komiyama et al. (2018), with the following exceptions:

- `DOB_yr`, the year of birth, has been dropped because it is (nearly) perfectly collinear with `age`, and thus it is redundant;
- `decile1b` has been dropped because it is (nearly) perfectly collinear with `decile1`, and thus it is redundant.

In that paper, `ugpa` is the response variable, `age` and `race1` are the sensitive attributes and the remaining variables are used as predictors.

References

Sander RH (2004). "A Systemic Analysis of Affirmative Action in American Law Schools". Stanford Law Review, 57:367–483.

Examples

```
data(law.school.admissions)

# short-hand variable names.
ll = law.school.admissions
r = ll[, "ugpa"]
s = ll[, c("age", "race1")]
p = ll[, setdiff(names(ll), c("ugpa", "age", "race1"))]

m = nclm(response = r, sensitive = s, predictors = p, unfairness = 0.05)
summary(m)

m = frrm(response = r, sensitive = s, predictors = p, unfairness = 0.05)
summary(m)
```

`methods for fair.model objects`

Extract Information from fair.model Objects

Description

Extract various quantities of interest from an object of class `fair.model`.

Usage

```
# methods for all fair.model objects.
## S3 method for class 'fair.model'
coef(object, ...)
## S3 method for class 'fair.model'
residuals(object, ...)
## S3 method for class 'fair.model'
fitted(object, type = "response", ...)
## S3 method for class 'fair.model'
sigma(object, ...)
## S3 method for class 'fair.model'
deviance(object, ...)
## S3 method for class 'fair.model'
logLik(object, ...)
## S3 method for class 'fair.model'
nobs(object, ...)
## S3 method for class 'fair.model'
print(x, digits, ...)
## S3 method for class 'fair.model'
summary(object, ...)
## S3 method for class 'fair.model'
all.equal(target, current, ...)
## S3 method for class 'fair.model'
plot(x, support = FALSE, regression = FALSE, ncol = 2, ...)

# predict() methods.
## S3 method for class 'nclm'
predict(object, new.predictors, new.sensitive, type = "response", ...)
## S3 method for class 'zlm'
predict(object, new.predictors, type = "response", ...)
## S3 method for class 'zlrm'
predict(object, new.predictors, type = "response", ...)
## S3 method for class 'frrm'
predict(object, new.predictors, new.sensitive, type = "response", ...)
## S3 method for class 'fgrrm'
predict(object, new.predictors, new.sensitive, type = "response", ...)
```

Arguments

`object`, `x`, `target`, `current`
: an object of class `fair.model` or `nclm`.

`type`
: a character string, the type of fitted value. If `"response"`, `fitted()` and `predict()` will return the fitted values (if the response in the model is continuous) or the classification probabilities (if it was discrete). If `"class"` and `object` is a classifier, `fitted()` and `predict()` will return the class labels as a factor. If `"link"` and `object` is a classifier, `fitted()` and `predict()` will return the linear component of the fitted or predicted value, on the scale of the link function.

`digits`
: a non-negative integer, the number of significant digits.

`new.predictors`	a numeric matrix or a data frame containing numeric and factor columns; the predictors for the new observations.
`new.sensitive`	a numeric matrix or a data frame containing numeric and factor columns; the sensitive attributes for the new observations.
`support`	a logical value, whether to draw support lines (diagonal of the first quadrant, horizontal line at zero, etc.) in `plot()`.
`regression`	a logical value, whether to draw the regression line of the observed values on the fitted values from the model in `plot()`.
`ncol`	a positive integer, the number of columns the plots will be arranged into.
`...`	additional arguments, currently ignored.

Author(s)

Marco Scutari

`national.longitudinal.survey` *Income and Labour Market Activities*

Description

Survey results from the U.S. Bureau of Labor Statistics to gather information on the labour market activities and other life events of several groups.

Usage

```
data(national.longitudinal.survey)
```

Format

The data contains 4908 observations and the following variables:

- `age`, a numeric variable containing the interviewee's age in years;
- `race`, a factor with 20 levels denoting various racial/ethnic origins;
- `gender`, a factor with levels `"Male"` and `"Female"`.
- `grade90`, a factor containing the highest completed school grade from "3RD GRADE" to "8TH YR COL OR MORE", with 18 levels;
- `income06`, a numeric variable, income in 2006 in 10000-USD units;
- `income96`, a numeric variable, income in 1996 in 10000-USD units;
- `income90`, a numeric variable, income in 1990 in 10000-USD units;
- `partner`, a factor encoding whether the interviewee has a partner, with levels `"No"` and `"Yes"`;
- `height`, a numeric variable, the height of the interviewee;
- `weight`, a numeric variable, the weight of the interviewee;

- famsize, a numeric variable, the number of family members;
- genhealth, a factor with levels "Excellent", "Very Good", "Good", "Fair", "Poor" encoding the general health status of the interviewee;
- illegalact, a numeric variable containing the number of illegal acts committed by the interviewee;
- charged, a numeric variable containing the number of illegal acts for which the interviewee has been charged;
- jobsnum90, a numeric value, the number of different jobs ever reported;
- afqt89, a numeric value, the percentile score of the "Profiles, Armed Forces Qualification Test" (AFQT);
- typejob90, a factor with 13 levels encoding different job types;
- jobtrain90, a factor with levels "No" and "Yes" encoding whether the job was classified as training.

Note

The data set has been pre-processed differently from Komiyama et al. (2018). In particular:

- the variables income96 and income06 have been retained as alternative responses;
- the variables height, weight, race, partner and famsize have been retained;
- the variables grade90 and genhealth are coded as ordered factors because they do not make sense on a numeric scale.

In that paper, income90 is the response variable, gender and age are the sensitive attributes.

References

U.S. Bureau of Labor Statistics.
https://www.bls.gov/nls/

Examples

```
data(national.longitudinal.survey)

# short-hand variable names.
nn = national.longitudinal.survey
# remove alternative response variables.
nn = nn[, setdiff(names(nn), c("income96", "income06"))]
# short-hand variable names.
r = nn[, "income90"]
s = nn[, c("gender", "age")]
p = nn[, setdiff(names(nn), c("income90", "gender", "age"))]

m = nclm(response = r, sensitive = s, predictors = p, unfairness = 0.05)
summary(m)

m = frrm(response = r, sensitive = s, predictors = p, unfairness = 0.05)
summary(m)
```

nclm *Nonconvex Optimization for Regression with Fairness Constraints*

Description

Fair regression model based on nonconvex optimization from Komiyama et al. (2018).

Usage

```
nclm(response, predictors, sensitive, unfairness, covfun, lambda = 0,
  save.auxiliary = FALSE)
```

Arguments

`response` a numeric vector, the response variable.

`predictors` a numeric matrix or a data frame containing numeric and factor columns; the predictors.

`sensitive` a numeric matrix or a data frame containing numeric and factor columns; the sensitive attributes.

`unfairness` a positive number in [0, 1], how unfair is the model allowed to be. A value of `0` means the model is completely fair, while a value of `1` means the model is not constrained to be fair at all.

`covfun` a function computing covariance matrices. It defaults to the `cov()` function from the **stats** package.

`lambda` a non-negative number, a ridge-regression penalty coefficient. It defaults to zero.

`save.auxiliary` a logical value, whether to save the fitted values and the residuals of the auxiliary model that constructs the decorrelated predictors. The default value is `FALSE`.

Details

`nclm()` defines fairness as statistical parity. The model bounds the proportion of the variance that is explained by the sensitive attributes over the total explained variance.

The algorithm proposed by Komiyama et al. (2018) works like this:

1. regresses the predictors against the sensitive attributes;
2. constructs a new set of predictors that are decorrelated from the sensitive attributes using the residuals of this regression;
3. regresses the response against the decorrelated predictors and the sensitive attributes, while
4. bounding the proportion of variance the sensitive attributes can explain with respect to the overall explained variance of the model.

Both `sensitive` and `predictors` are standardized internally before estimating the regression coefficients, which are then rescaled back to match the original scales of the variables. `response` is only standardized if it has a variance smaller than `1`, as that seems to improve the stability of the solutions provided by the optimizer (as far as the data included in **fairml** are concerned).

The `covfun` argument makes it possible to specify a custom function to compute the covariance matrices used in the constrained optimization. Some examples are the kernel estimators described in Komiyama et al. (2018) and the shrinkage estimators in the **corpcor** package.

Value

`nclm()` returns an object of class `c("nclm", "fair.model")`.

Author(s)

Marco Scutari

References

Komiyama J, Takeda A, Honda J, Shimao H (2018). "Nonconvex Optimization for Regression with Fairness Constraints". Proceedints of the 35th International Conference on Machine Learning (ICML), PMLR **80**:2737–2746.
`http://proceedings.mlr.press/v80/komiyama18a/komiyama18a.pdf`

See Also

frrm, zlm

`obesity.levels` *Obesity Levels*

Description

Predict obesity levels based on eating habits and physical condition.

Usage

```
data(obesity.levels)
```

Format

The data contains 2111 observations and 17 variables. See the UCI Machine Learning Repository for details.

Note

The data set has been minimally pre-processed: the only change is that the only observation for which the `CALC` variable was equal to `"Always"` has been changed to `"Frequently"` to merge the two levels.

The obesity level `NObeyesdad` is the response variable (with 7 different levels) and `Age` and `Gender` are the sensitive attributes. The remaining variables are used as predictors.

The data contain the following variables:

- Gender;
- Age;
- Height;
- Weight;
- family_history_with_overweight;
- FAVC, frequent consumption of high caloric food as a factor with levels "no" and "yes";
- FCVC, frequency of consumption of vegetables as a numeric variable;
- NCP, number of main meals;
- CAEC, consumption of food between meals as a factor with levels "no", "Sometimes", "Frequently" and "Always";
- SMOKE, smoking status as a factor with levels "no" and "yes";
- CH2O, consumption of water daily as a numeric variable;
- SCC, calories consumption monitoring as a factor with level "no" and "yes";
- FAF, physical activity frequency as a numeric variable;
- TUE, time using technology devices as a numeric variable;
- CALC, consumption of alcohol as a dfactor with levels "no", "Sometimes", "Frequently" and "Always";
- MTRANS, transportation used as a factor with levels "Automobile", "Bike", "Motorbike", "Public_Transportation" and "Walking";
- NObeyesdad, the obesity level as a factor with levels "Insufficient_Weight", "Normal_Weight", "Overweight_Level_I", "Overweight_Level_II", "Obesity_Type_I", "Obesity_Type_II", "Obesity_Type_III".

References

UCI Machine Learning Repository.
https://archive-beta.ics.uci.edu/dataset/544

Examples

```
data(obesity.levels)

# short-hand variable names.
r = obesity.levels[, "NObeyesdad"]
s = obesity.levels[, c("Gender", "Age")]
p = obesity.levels[, setdiff(names(obesity.levels), c("NObeyesdad", "Gender", "Age"))]

## Not run:
# the lambda = 0.1 is very helpful in making model estimation succeed.
m = fgrrm(response = r, sensitive = s, predictors = p, ,
      family = "multinomial", unfairness = 0.05, lambda = 0.1)
summary(m)

## End(Not run)
```

synthetic data sets *Synthetic Data Set to Test Fair Models*

Description

Synthetic data set used as test cases in the **fairml** package.

Usage

```
data(vu.test)
```

Format

The data are stored a list with following three elements:

- gaussian, binomial, poisson, coxph and multinomial are response variables for the different families;
- X, a numeric matrix containing 3 predictors called X1, X2 and X3;
- S, a numeric matrix containing 3 sensitive attributes called S1, S2 and S3.

Note

This data set is called vu.test because it is generated from *v*ery *u*nfair models in which sensitive attributes explain the lion's share of the overall explained variance or deviance.

The code used to generate the predictors and the sensitive attributes is as follows.

```
library(mvtnorm)
sigma = matrix(0.3, nrow = 6, ncol = 6)
diag(sigma) = 1
n = 1000
X = rmvnorm(n, mean = rep(0, 6), sigma = sigma)
S = X[, 4:6]
X = X[, 1:3]
colnames(X) = c("X1", "X2", "X3")
colnames(S) = c("S1", "S2", "S3")
```

The continuous response in gaussian is produced as follows.

```
gaussian = 2 + 2 * X[, 1] + 3 * X[, 2] + 4 * X[, 3] + 5 * S[, 1] +
               6 * S[, 2] + 7 * S[, 3] + rnorm(n, sd = 10)
```

The discrete response in binomial is produced as follows.

```
nu = 1 + 0.5 * X[, 1] + 0.6 * X[, 2] + 0.7 * X[, 3] + 0.8 * S[, 1] +
         0.9 * S[, 2] + 1.0 * S[, 3]
binomial = rbinom(n = nrow(X), size = 1, prob = exp(nu) / (1 + exp(nu)))
binomial = as.factor(binomial)
```

The log-linear response in poisson is produced as follows.

```
nu = 1 + 0.5 * X[, 1] + 0.6 * X[, 2] + 0.7 * X[, 3] + 0.8 * S[, 1] +
         0.9 * S[, 2] + 1.0 * S[, 3]
poisson = rpois(n = nrow(X), lambda = exp(nu))
```

The response for the Cox proportional hazards coxph is produced as follows.

```
fx = 1 + 0.5 * X[, 1] + 0.6 * X[, 2] + 0.7 * X[, 3] + 0.8 * S[, 1] +
         0.9 * S[, 2] + 1.0 * S[, 3]
hx = exp(fx)
ty = rexp(length(fx), hx)
tcens = rbinom(n = length(fx), prob = 0.3, size = 1)
coxph = cbind(time = ty, status = 1 - tcens)
```

The discrete response in multinomial is produced as follows.

```
nu1 = 1 + 0.5 * X[, 1] + 0.6 * X[, 2] + 0.7 * X[, 3] + 0.8 * S[, 1] +
          0.9 * S[, 2] + 1.0 * S[, 3]
nu2 = 1 + 0.2 * X[, 1] + 0.2 * X[, 2] + 0.2 * X[, 3] + 0.6 * S[, 1] +
          0.6 * S[, 2] + 0.6 * S[, 3]
nu3 = 1 + 0.7 * X[, 1] + 0.6 * X[, 2] + 0.5 * X[, 3] + 0.1 * S[, 1] +
          0.1 * S[, 2] + 0.1 * S[, 3]
nu4 = 1 + 0.4 * X[, 1] + 0.4 * X[, 2] + 0.4 * X[, 3] + 0.4 * S[, 1] +
          0.4 * S[, 2] + 0.4 * S[, 3]
norm = exp(nu1) + exp(nu2) + exp(nu3) + exp(nu4)
probs = matrix(c(exp(nu1) / norm, exp(nu2) / norm,
                 exp(nu3) / norm, exp(nu4) / norm),
        ncol = 4, byrow = FALSE)
multinomial = apply(probs, MARGIN = 1,
                function(x) sample(letters[1:4], size = 1, prob = x))
multinomial = factor(multinomial, labels = letters[1:4])
```

Author(s)

Marco Scutari

Examples

```
summary(fgrrm(response = vu.test$gaussian, predictors = vu.test$X,
  sensitive = vu.test$S, unfairness = 1, family = "gaussian"))
summary(fgrrm(response = vu.test$binomial, predictors = vu.test$X,
  sensitive = vu.test$S, unfairness = 1, family = "binomial"))
summary(fgrrm(response = vu.test$poisson, predictors = vu.test$X,
  sensitive = vu.test$S, unfairness = 1, family = "poisson"))
summary(fgrrm(response = vu.test$coxph, predictors = vu.test$X,
  sensitive = vu.test$S, unfairness = 1, family = "cox"))
summary(fgrrm(response = vu.test$multinomial, predictors = vu.test$X,
  sensitive = vu.test$S, unfairness = 1, family = "multinomial"))
```

zlm *Zafar's Linear and Logistic Regressions*

Description

Linear and logistic regression models enforcing fairness by bounding the covariance between sensitive attributes and predictors.

Usage

```
# a fair linear regression model.
zlm(response, predictors, sensitive, unfairness)
zlm.orig(response, predictors, sensitive, max.abs.cov)
# a fair logistic regression model.
zlrm(response, predictors, sensitive, unfairness)
zlrm.orig(response, predictors, sensitive, max.abs.cov)
```

Arguments

`response`	a numeric vector, the response variable.
`predictors`	a numeric matrix or a data frame containing numeric and factor columns; the predictors.
`sensitive`	a numeric matrix or a data frame containing numeric and factor columns; the sensitive attributes.
`unfairness`	a positive number in [0, 1], how unfair is the model allowed to be. A value of `0` means the model is completely fair, while a value of `1` means the model is not constrained to be fair at all.
`max.abs.cov`	a non-negative number, the original bound on the maximum absolute covariance from Zafar et al. (2019).

Details

`zlm()` and `zlrm()` define fairness as statistical parity.

Estimation minimizes the log-likelihood of the regression models under the constraint that the correlation between each sensitive attribute and the fitted values (on the linear predictor scale, in the case of logistic regression) is smaller than `unfairness` in absolute value. Both models include `predictors` as explanatory variables; the variables `sensitive` only appear in the constraints.

The only difference between `zlm()` and `zlm.orig()`, and between `zlrm()` and `zlrm.orig()`, is that the latter uses the original constraint on the covariances of the individual sensitive attributes from Zafar et al. (2019).

Value

`zlm()` and `zlm.orig()` return an object of class `c("zlm", "fair.model")`. `zlrm()` and `zlrm.orig()` return an object of class `c("zlrm", "fair.model")`.

Author(s)

Marco Scutari

References

Zafar BJ, Valera I, Gomez-Rodriguez M, Gummadi KP (2019). "Fairness Constraints: a Flexible Approach for Fair Classification". Journal of Machine Learning Research, 30:1–42.
`https://www.jmlr.org/papers/volume20/18-262/18-262.pdf`

See Also

nclm, frrm, fgrrm

Index