Package: MFF (via r-universe)

November 22, 2026

Type Package

Title Meta Fuzzy Functions for Aggregating Prediction Pools

Version 1.0.0

Description Aggregates numeric candidate predictions using Meta Fuzzy Functions and membership-derived weights learned on validation predictions, following Tak (2018) <doi:10.1016/j.asoc.2018.08.009> and Tak (2021) <doi:10.1016/j.ins.2021.01.024>. Provides a common prediction-pool interface, fixed and tuned fits, explicit validation-based selection, and standard methods for inspecting weights, predicting and plotting results. Candidate pools may contain different regression learners, bootstrap fits, alternative hyperparameter configurations, or externally generated regression and time-aligned forecasting predictions. Resampling results of 'caret' and 'tidymodels' can be converted into pools, and forecasting pools are built by rolling-origin evaluation or from 'fable' forecasts. Supports fuzzy c-means, possibilistic fuzzy c-means following Pal et al. (2005) <doi:10.1109/TFUZZ.2004.840099>, Gustafson-Kessel clustering with the improved covariance estimation of Babuska et al. (2002) <doi:10.1109/FUZZ.2002.1006654> and k-means clustering. Optional helpers train built-in or user-defined regression learners on explicit data splits, cross-validation folds or bootstrap samples.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Depends R (>= 3.2.0)

Imports e1071, parallel

Suggests glmnet, randomForest, xgboost, lightgbm, caret, caretEnsemble, tune, workflows, parsnip, rsample, stacks, forecast, fable, fabletools, tsibble, knitr, rmarkdown, testthat (>= 3.1.7)

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation no

Config/roxygen2/version 8.1.1

Author Nihat Tak [aut, cre], Sadık Çoban [ctb]

Maintainer Nihat Tak <nihattak@gmail.com>

Repository https://cran.r-universe.dev

Date/Publication 2026-11-22 00:28:03 UTC

RemoteUrl https://github.com/cran/MFF

RemoteRef HEAD

RemoteSha 5a6e846ca6cf0997f4767bfca3caea55f3cdbe44

Contents

MFF-package . . . 3
as_mff_pool . . . 4
boot_train . . . 6
evaluate . . . 7
fit_mff . . . 8
mff_control . . . 10
mff_cv . . . 11
mff_diamonds . . . 12
mff_example . . . 13
mff_learner . . . 13
mff_pool . . . 14
mff_split . . . 16
model_train . . . 16
plot.mff_fit . . . 18
predict.mff_fit . . . 19
print.mff_fit . . . 20
print.mff_prediction_pool . . . 21
rolling_train . . . 21
summary.mff_fit . . . 23
summary.mff_prediction_pool . . . 23
tune_mff . . . 24
validation_predictions . . . 26
validation_scores . . . 27
weights.mff_fit . . . 28

Index **30**

MFF-package *MFF: Meta fuzzy functions for aggregating prediction pools*

Description

Model-agnostic aggregation of numeric candidate predictions using weights derived from clustering validation prediction profiles.

Main workflow

`mff_pool()` constructs aligned inputs from any prediction source. `model_train()` and `boot_train()` optionally generate regression pools; `mff_cv()` makes `model_train()` use out-of-fold validation predictions, and `as_mff_pool()` converts caret, tidymodels, stacks and fable results. `rolling_train()` builds rolling-origin forecasting pools. `fit_mff()` fits fixed configurations and `tune_mff()` searches a grid. `predict.mff_fit()` applies validation-selected weights to new predictions. `evaluate()` reports MAE, RMSE, MAPE, SMAPE, MSE and median absolute error.

Classes and access

Inputs use `mff_prediction_pool`. Fixed fits use `mff_fit`; tuned fits inherit from `mff_fit` through `mff_tuned`. Use `validation_predictions()`, `weights.mff_fit()`, `selected_configuration()` and `search_results()` to inspect results. All three object types have print and summary methods; fits have plot methods including a weight heatmap. See `mff_example()` for a real-data pool.

External forecasting

Ordered external point forecasts can be supplied with `task = "forecasting"` and validation/test indexes. The package checks index alignment and ordering. External training and information-leakage prevention remain the user's responsibility.

Acknowledgements

Supported by TUBITAK under project 125F138.

Author(s)

Maintainer: Nihat Tak <nihattak@gmail.com>

Authors:

- Nihat Tak <nihattak@gmail.com>

Other contributors:

- Sadık Çoban <sadikc716@gmail.com> [contributor]

References

Tak, N. (2018). Meta fuzzy functions: Application of recurrent type-1 fuzzy functions. Applied Soft Computing, 73, 1–13. doi:10.1016/j.asoc.2018.08.009. Tak, N. (2021). Forecast combination with meta possibilistic fuzzy functions. Information Sciences, 560, 168–182. doi:10.1016/j.ins.2021.01.024. Tak, N. and Ucan, A. (2026). Meta fuzzy feature-selection-based regression functions. Applied Soft Computing, 190, 114592. doi:10.1016/j.asoc.2026.114592.

`as_mff_pool` *Convert resampling results of other modeling frameworks to prediction pools*

Description

`as_mff_pool()` turns out-of-fold predictions stored by another modeling framework into an `mff_pool()`. The out-of-fold predictions and the corresponding training outcomes form the validation part of the pool. When `newdata` is supplied, test predictions are computed by the framework's own prediction methods from models fitted on all training rows. The frameworks are optional; their packages are needed only for the corresponding method.

Usage

```
as_mff_pool(x, ...)

## Default S3 method:
as_mff_pool(x, ...)

## S3 method for class 'list'
as_mff_pool(x, newdata = NULL, test_observed = NULL, ...)

## S3 method for class 'caretList'
as_mff_pool(x, newdata = NULL, test_observed = NULL, ...)

## S3 method for class 'tune_results'
as_mff_pool(x, newdata = NULL, test_observed = NULL, ...)

## S3 method for class 'data_stack'
as_mff_pool(x, newdata = NULL, test_observed = NULL, ...)

## S3 method for class 'fbl_ts'
as_mff_pool(x, data, horizon = 1L, n_test, ...)
```

Arguments

`x`	An object to convert; see Details.
`...`	Reserved for future use.
`newdata`	Optional data frame of test rows to predict.

test_observed	Optional observed test outcomes, used only for scoring.
data	For a fable, the tsibble holding the observed series.
horizon	For a fable, the forecast step to keep from every origin.
n_test	For a fable, the number of final target times used as test. As in rolling_train(), validation targets end at the origin of the first test target, horizon steps of data before it; targets in between are not used. data must hold the single forecast series.

Details

Supported inputs:

A list of caret models A named list of regression models from caret::train(), or a caretList from caretEnsemble, trained with savePredictions = "final" or "all". Each model contributes the out-of-fold predictions of its selected tuning parameters, averaged over repeated resamples, and becomes one candidate (pool_type = "heterogeneous"). All models must have predicted the same training rows. Test predictions use the final models.

tidymodels tuning results The result of tune::tune_grid() for a regression workflow with control_grid(save_pred = TRUE). Every parameter combination becomes one candidate (pool_type = "hyperparameter"), with out-of-fold predictions averaged over repeated resamples. Test predictions require save_workflow = TRUE: each combination is finalized and refitted on the full training data before predicting newdata.

A stacks data stack A data_stack from stacks::add_candidates(). Its candidate columns become candidates of a validation-only pool; test predictions are not available from a data stack.

Value

An mff_prediction_pool. pool_info(pool)$metadata$source names the framework.

See Also

mff_pool(), model_train()

Examples

```
if (requireNamespace("caret", quietly = TRUE)) {
  set.seed(1)
  dat <- datasets::mtcars
  ctrl <- caret::trainControl(method = "cv", number = 4,
                              savePredictions = "final",
                              index = caret::createFolds(dat$mpg[1:26], k = 4,
                                                         returnTrain = TRUE))
  models <- list(
    linear = caret::train(mpg ~ wt + hp + disp, dat[1:26, ], method = "lm",
                          trControl = ctrl),
    knn = caret::train(mpg ~ wt + hp + disp, dat[1:26, ], method = "knn",
                       trControl = ctrl, tuneLength = 2),
    linear_small = caret::train(mpg ~ wt, dat[1:26, ], method = "lm",
```

```
                                trControl = ctrl)
  )
  pool <- as_mff_pool(models, newdata = dat[27:32, ],
                      test_observed = dat$mpg[27:32])
  fit <- fit_mff(pool, clusters = 2, seed = 1)
  weights(fit)
}
```

`boot_train` *Generate a bootstrap regression prediction pool*

Description

Fit one learner to bootstrap samples of training rows and predict identical validation and test rows for every candidate.

Usage

```
boot_train(
  formula,
  data,
  split,
  times,
  learner = "lm",
  controls = list(),
  control = mff_control()
)
```

Arguments

`formula`	A formula with a single untransformed numeric response. Formula offsets are known additive response-scale contributions: learners fit the training response minus its offset, then the appropriate validation or test offset is added to predictions. Offsets must be finite.
`data`	A data frame. Missing values in used variables are rejected.
`split`	A list created by `mff_split()`.
`times`	Number of bootstrap candidates.
`learner`	A built-in learner name accepted by `model_train()` or an `mff_learner()` object.
`controls`	A named control list for the learner, as for one candidate in `model_train()`.
`control`	An `mff_control()` list. Bootstrap failures stop explicitly.

Value

An `mff_prediction_pool` with uniquely named bootstrap candidates.

Examples

```
pool <- boot_train(Volume ~ Girth + Height, datasets::trees,
                   mff_split(1:15, 16:23, 24:31), times = 6,
                   control = mff_control(seed = 4))
dim(validation_predictions(pool))
fit <- tune_mff(pool, clusters = 2:3, method = "kmeans")
evaluate(predict(fit, pool), test_observed(pool))
```

`evaluate` *Compute error metrics for predicted values or prediction matrices*

Description

Compute error metrics for predicted values or prediction matrices by comparing them with the corresponding true response values.

Usage

```
evaluate(predicted, actual)
```

Arguments

`predicted`	numeric vector or numeric matrix of predictions. If matrix, columns are evaluated separately.
`actual`	numeric vector of true target values.

Details

The evaluate function is used to quantify the predictive performance of meta fuzzy function predictions by comparing them with the corresponding true response values. It supports both vector- and matrix-valued prediction inputs, allowing performance assessment of a single meta fuzzy function as well as simultaneous evaluation of multiple meta fuzzy functions. MAPE is 100 times the mean of abs(error) / (abs(actual) + 1e-8). SMAPE is 100 times the mean of 2 * abs(error) / (abs(predicted) + abs(actual) + 1e-8). MAPE can be unstable when actual values are near zero. When a prediction vector is provided, the function computes a single set of performance metrics; when a prediction matrix is provided, metrics are computed separately for each meta fuzzy function.

Value

A matrix with columns MAE, RMSE, MAPE, SMAPE, MSE, and MedAE, with one row per evaluated prediction vector.

See Also

`fit_mff()` and `tune_mff()`, which use these metrics for validation-based selection, and `predict.mff_fit()` for test predictions.

Examples

```
x <- seq(100)
y <- 2*x + stats::rnorm(100)
m <- stats::lm(y ~ x)
pred <- stats::predict(m)
evaluate(pred, y)
```

`fit_mff` *Fit a fixed MFF configuration*

Description

Cluster candidate validation prediction profiles, normalize their memberships into aggregation weights, and select one meta function using validation loss.

Usage

```
fit_mff(
  pool,
  clusters,
  method = c("fcm", "pfcm", "gk", "kmeans"),
  m = 2,
  eta = 2,
  metric = "RMSE",
  standardize = FALSE,
  reduction = c("none", "pca"),
  nstart = 1L,
  iter_max = NULL,
  seed = NULL,
  reltol = NULL,
  gamma = 0,
  beta = 1e+15
)
```

Arguments

`pool`	An `mff_pool()` prediction pool. At least two candidates are required.
`clusters`	A cluster count, or a vector of counts for tuning; each must be between two and the number of candidates. For k-means and PFCM, it must be smaller than the number of candidates: `stats::kmeans()` does not accept as many centers as candidates, and the PFCM penalties of equation (11) of Pal et al. (2005) are then not all positive.
`method`	Membership method: fuzzy c-means (fcm), possibilistic fuzzy c-means (pfcm), Gustafson-Kessel (gk), or crisp k-means (kmeans). PFCM follows Pal, Pal, Keller and Bezdek (2005) and GK the modified algorithm with improved covariance estimation of Babuska, van der Veen and Kaymak (2002); both are implemented in this package (see Details).

`m` Fuzziness exponent greater than one; ignored for k-means.

`eta` PFCM typicality exponent, greater than 1; ignored for other methods.

`metric` Validation selection metric, using the formulas in `evaluate()`.

`standardize` Standardize each validation-observation coordinate across candidate profiles before clustering. Aggregation uses original predictions.

`reduction` Use original coordinates or all nonzero PCA components. PCA preserves Euclidean distances but need not preserve GK/PFCM solutions.

`nstart` Number of independent random starts per configuration, used by all four methods. The start with the smallest clustering objective is kept; validation error is not used to choose the start.

`iter_max` Iteration limit per start, used by all four methods. Defaults: kmeans 10, fcm 100, pfcm 100, gk 1000.

`seed` Optional local seed; when supplied the caller's RNG state is restored.

`reltol` A finite positive convergence tolerance, or NULL to retain defaults: sqrt(.Machine$double.eps) for FCM, 1e-5 for PFCM and 1e-3 for GK. FCM uses relative objective change (control$reltol), PFCM the mean absolute change of the prototype coordinates, $||V_t - V_{t-1}||_1/(pc)$, and GK the largest absolute change of a membership between iterations. Thus the same value does not mean the same stopping rule across methods. The PFCM tolerance is absolute, as in Pal et al. (2005), and depends on the scale of the predictions: for predictions of very small or very large magnitude, rescale them or set `reltol` accordingly. K-means (`stats::kmeans()`) has no tolerance: it stops when no candidate changes cluster, so `reltol` must be NULL and only `nstart` and `iter_max` apply. K-means does not support reltol; leave it NULL and use iter_max.

`gamma` GK weighting parameter in $[0, 1]$ of the scaled identity matrix added to each fuzzy covariance matrix; 0 (the default) adds none. Following Babuska et al. (2002), the identity matrix is scaled by $\det(\mathrm{cov}(Z))^{1/n}$, where Z holds the clustered profiles and n is their dimension. When the number of validation observations is at least the number of candidates, $\mathrm{cov}(Z)$ is singular and this scale is zero, so that `gamma` has no effect and `gamma = 1` fails. With `reduction = "pca"`, the profiles have fewer dimensions than candidates and `gamma` takes effect. Ignored for other methods.

`beta` GK condition-number threshold for the fuzzy covariance matrices, greater than one; ignored for other methods. When the number of validation observations is at least the number of candidates, the fuzzy covariance matrices are singular, and their zero eigenvalues are set to the largest eigenvalue divided by `beta`; `beta` then enters the cluster volumes $\det(F_i)^{1/n}$. With `reduction = "pca"`, the directions without spread are removed before clustering.

Details

Test matrices and test outcomes do not enter fitting. Meta-function selection is supervised by validation outcomes; clustering candidate prediction profiles does not use the outcomes.

PFCM follows the PFCM-AO algorithm of Pal, Pal, Keller and Bezdek (2005, Table XI) with $a = b = 1$: memberships, typicalities and prototypes are updated by their equations (21)-(23), and

the penalties γ_i are computed by their equation (11) with $K = 1$ from a terminal FCM partition started from the same randomly selected data points. Defaults follow the paper's protocol (tolerance 1e-5, at most 100 iterations). Weights are derived from the memberships u.

GK follows the modified Gustafson-Kessel algorithm of Babuska, van der Veen and Kaymak (2002, Section V and Appendix). Each fuzzy covariance matrix F_i is replaced by $(1-\gamma)F_i+\gamma \det(F_0)^{1/n} I$, where F_0 is the covariance matrix of all clustered profiles and n their dimension; if its condition number exceeds β, every eigenvalue λ with $\beta\lambda < \lambda_{max}$ is set to λ_{max}/β. Distances are $(z-v_i)^T \det(F_i)^{1/n} F_i^{-1}(z-v_i)$ with unit cluster volumes. Starts are initialized as in the authors' implementation, and a start converges when no membership changes by more than `reltol` between iterations. The eigenvalue constraint keeps the matrices invertible when clusters contain few or linearly dependent profiles.

Value

An `mff_fit` list with candidate names, method, parameters, weights, validation scores, selected function, metric, preprocessing and diagnostics. Accessors include `weights.mff_fit()`, `validation_scores()`, `selected_configuration()` and `fit_diagnostics()`.

References

Tak, N. (2018). Meta fuzzy functions: Application of recurrent type-1 fuzzy functions. Applied Soft Computing, 73, 1–13. doi:10.1016/j.asoc.2018.08.009. Gustafson, D. E. and Kessel, W. C. (1979). Fuzzy clustering with a fuzzy covariance matrix. Proceedings of the IEEE Conference on Decision and Control, 761–766. doi:10.1109/CDC.1978.268028. Babuska, R., van der Veen, P. J. and Kaymak, U. (2002). Improved covariance estimation for Gustafson-Kessel clustering. Proceedings of the IEEE International Conference on Fuzzy Systems, 1081–1085. doi:10.1109/FUZZ.2002.1006654. Pal, N. R., Pal, K., Keller, J. M. and Bezdek, J. C. (2005). A possibilistic fuzzy c-means clustering algorithm. IEEE Transactions on Fuzzy Systems, 13, 517–530. doi:10.1109/TFUZZ.2004.840099.

Examples

```
pool <- mff_example()
fit <- fit_mff(pool, clusters = 2, method = "fcm", seed = 12,
               nstart = 3, reltol = 1e-6)
fit
weights(fit)
evaluate(predict(fit, pool), test_observed(pool))
```

`mff_control` *Configure MFF tuning*

Description

Configure MFF tuning

Usage

```
mff_control(
  seed = 123L,
  parallel = FALSE,
  workers = 1L,
  verbose = FALSE,
  error_action = c("continue", "stop")
)
```

Arguments

seed	An integer seed, or NULL to use the current random state.
parallel	Whether to use a socket cluster for grid evaluation.
workers	A positive integer number of worker processes.
verbose	Whether to report the grid size.
error_action	Continue past failed configurations, or stop.

Value

A validated list of tuning controls.

Examples

```
mff_control(seed = 123, parallel = FALSE)
```

---

mff_cv *Cross-validation for out-of-fold validation predictions*

Description

Describes how model_train() produces validation predictions from the training rows when no separate validation rows are available. Each training row is predicted once by candidates trained on the other folds; these out-of-fold predictions and the training outcomes form the validation part of the pool. Test predictions come from candidates refitted on all training rows.

Usage

```
mff_cv(folds = 5L, fold_id = NULL)
```

Arguments

folds	Number of folds, at least two. Ignored when fold_id is given.
fold_id	Optional fold labels, one per training row in the order of split$train, for example to keep grouped rows in the same fold. At least two distinct labels are required.

Value

An mff_cv object.

Examples

```
dat <- datasets::trees
pool <- model_train(Volume ~ Girth + Height, dat,
                    mff_split(train = 1:24, validation = NULL, test = 25:31),
                    resampling = mff_cv(folds = 4), seed = 1)
dim(validation_predictions(pool))
pool_info(pool)$metadata$resampling$fold_id
```

---

mff_diamonds *Candidate predictions of diamond prices*

Description

Validation and test predictions of seven regression learners for the price of diamonds. The data are 2000 rows sampled at random, with a fixed seed, from the diamonds data of the ggplot2 package. These rows were randomly split into 1000 training, 500 validation and 500 test rows. Each learner was trained by model_train() on the training rows with the formula price ~ carat + cut + color + clarity + depth + table (cut, color and clarity as unordered factors) and its default settings: penalized regressions tuned by train-only cross-validation, random forests with 100 trees, and boosters with 200 rounds and one thread. The predictions are stored so that examples need neither ggplot2 nor the optional learner packages. They were computed with ggplot2 4.0.3, glmnet 5.1, randomForest 4.7.1.2, xgboost 3.2.1.1 and lightgbm 4.7.0. mff_example() builds the data into a prediction pool.

Usage

```
mff_diamonds
```

Format

A data frame with 1000 rows and 10 columns:

row Row of the diamond in ggplot2::diamonds.

set Factor with levels "validation" and "test".

price Observed price in US dollars.

lm, lasso, ridge, elastic_net Predictions of the linear learners.

random_forest, xgboost, lightgbm Predictions of the tree-ensemble learners.

Source

Computed from the diamonds data in the ggplot2 package (H. Wickham, ggplot2: Elegant Graphics for Data Analysis, Springer, 2016).

Examples

```
head(mff_diamonds)
with(mff_diamonds, tapply(price, set, mean))
```

---

mff_example *Prediction-pool example built from diamond prices*

Description

Builds the mff_diamonds predictions into a prediction pool: 500 validation and 500 test rows for seven regression learners of diamond price. Candidate information records the learner family. The example is an interface demonstration, not a benchmark.

Usage

```
mff_example()
```

Value

An `mff_prediction_pool` with seven candidate columns.

Examples

```
pool <- mff_example()
pool
fit <- fit_mff(pool, clusters = 2, method = "fcm", seed = 12)
weights(fit)
evaluate(predict(fit, pool), test_observed(pool))
```

---

mff_learner *Define a user learner*

Description

Wraps any regression method so that `model_train()` and `boot_train()` can train it on explicit splits, cross-validation folds or bootstrap samples. The learner receives the numeric model matrix built from the formula on the training rows (factor levels and contrasts are learned there) and the training response minus any formula offset; the offset is added back to its predictions.

Usage

```
mff_learner(fit, predict, name = "custom", intercept = FALSE)

## S3 method for class 'mff_learner'
print(x, ...)
```

Arguments

fit	A function function(x, y, ...) returning a fitted object. x is the training model matrix and y the numeric response. Per-candidate controls of model_train() are passed as further named arguments.
predict	A function function(object, newx) returning one finite numeric prediction per row of newx.
name	A label recorded as the candidate's algorithm.
intercept	Keep the intercept column of the model matrix in x. The default drops it, as most learners fit their own intercept.
x	An mff_learner.
...	Ignored.

Value

An mff_learner object.

Examples

```
# Projection pursuit regression from base R.
ppr_learner <- mff_learner(
  fit = function(x, y, nterms = 2) stats::ppr(x, y, nterms = nterms),
  predict = function(object, newx) stats::predict(object, newx),
  name = "ppr"
)
dat <- datasets::airquality[stats::complete.cases(datasets::airquality), ]
set.seed(1)
rows <- sample(nrow(dat))
pool <- model_train(Ozone ~ Solar.R + Wind + Temp, dat,
                    mff_split(rows[1:60], rows[61:85], rows[86:111]),
                    learners = list(lm = "lm", ppr2 = ppr_learner,
                                    ppr3 = ppr_learner),
                    controls = list(ppr3 = list(nterms = 3)), seed = 1)
candidate_info(pool)
```

---

mff_pool *Create an aligned prediction pool*

Description

Construct the shared input to fixed and tuned MFF aggregation. Rows are observations and columns are candidate predictions, irrespective of the software or algorithm that generated them.

Usage

```
mff_pool(
  validation_predictions,
  validation_observed,
  test_predictions = NULL,
  test_observed = NULL,
  candidate_info = NULL,
  task = c("regression", "forecasting"),
  pool_type = c("external", "heterogeneous", "bootstrap", "hyperparameter"),
  validation_index = NULL,
  test_index = NULL,
  metadata = list()
)
```

Arguments

`validation_predictions`
: A finite numeric matrix with unique column names.

`validation_observed`
: A finite numeric validation response vector.

`test_predictions`
: An optional numeric matrix with the same candidate names and order as the validation matrix.

`test_observed`
: Optional observed test responses; used only for scoring.

`candidate_info`
: Optional data frame with a `candidate` column matching the prediction columns, and arbitrary algorithm or hyperparameter columns.

`task`
: Either `"regression"` or `"forecasting"`.

`pool_type`
: A descriptive pool type: external, heterogeneous, bootstrap, or hyperparameter. This label does not change the aggregation algorithm.

`validation_index`, `test_index`
: Optional observation indexes. Forecasting requires strictly increasing numeric, Date, or POSIXct indexes; validation must precede test. This checks alignment, not how external models were trained. For h-step-ahead forecasts, validation targets should end at the origin of the first test target (h steps before it); later validation outcomes would be observed after a test origin. `rolling_train()` builds pools this way.

`metadata`
: A list of provenance or split information.

Value

An `mff_prediction_pool` list. Its matrices, outcomes, candidate information and provenance are accessed using `validation_predictions()`, `test_predictions()`, `candidate_info()` and `pool_info()`.

See Also

`fit_mff()`, `tune_mff()`, `model_train()`, `boot_train()`

Examples

```
pool <- mff_example()
pool
dim(validation_predictions(pool))
candidate_info(pool)
```

`mff_split` *Define training, validation and test rows*

Description

Define training, validation and test rows

Usage

```
mff_split(train, validation, test)
```

Arguments

`train`, `test`	Nonempty vectors of positive integer row indexes.
`validation`	A nonempty vector of positive integer row indexes, or NULL when validation predictions are produced by cross-validation on the training rows (see `mff_cv()`).

Value

A list with three disjoint index vectors, used by `model_train()` and `boot_train()`; `validation` has length zero when it was NULL. Omitted data rows are not used.

Examples

```
split <- mff_split(1:15, 16:23, 24:31)
cv_split <- mff_split(train = 1:23, validation = NULL, test = 24:31)
```

`model_train` *Train regression candidates on explicit data splits*

Description

Generate aligned predictions for a common pool. Candidate training is optional: external predictions can be supplied directly to `mff_pool()`.

Usage

```
model_train(
  formula,
  data,
  split,
  learners = "lm",
  controls = list(),
  seed = 123L,
  resampling = NULL
)
```

Arguments

formula	A formula with a single untransformed numeric response. Formula offsets are known additive response-scale contributions: learners fit the training response minus its offset, then the appropriate validation or test offset is added to predictions. Offsets must be finite.
data	A data frame. Missing values in used variables are rejected.
split	A list created by mff_split().
learners	Unique learner names: lm, lasso, ridge, elastic_net, random_forest, xgboost, or lightgbm; or a named list whose elements are such names or mff_learner() objects. List names become candidate names, so one algorithm can enter several times with different controls. The default lm needs no optional learner.
controls	A named list of named control lists, one per candidate name. For an mff_learner() the controls are passed to its fit function. Penalized models accept cv.glmnet arguments (alpha is fixed by the name; intercept is determined by the formula and conflicting controls fail). Random forests accept randomForest arguments. Boosters accept nrounds and a params list. glmnet defaults to at most five train-only folds; RF uses 100 trees; boosters use 200 rounds and one thread. Single-predictor penalized fits use an excluded zero column internally because glmnet requires at least two input columns. This does not introduce an additional fitted predictor. Penalized cross-validation requires at least three training rows; the chosen folds must also permit fitting the requested model.
seed	A local seed, or NULL.
resampling	NULL to predict the validation rows of split, or an mff_cv() object to use out-of-fold predictions of the training rows as validation predictions; split must then have no validation rows. Within each fold, model-matrix encoding, factor levels and learner tuning use only that fold's training rows. Test predictions come from candidates refitted on all training rows; test rows never enter fold fitting.

Value

An mff_prediction_pool, including candidate settings and split provenance. One candidate is allowed as a baseline, but fitting MFF needs at least two. Unseen factor levels fail; encoding is learned on train rows. With cross-validation, a factor level that occurs only in a held-out fold also fails, and the fold assignment is stored in pool_info(pool)$metadata$resampling.

Examples

```
pool <- model_train(Volume ~ Girth + Height, datasets::trees,
                    mff_split(1:15, 16:23, 24:31), learners = "lm")
summary(pool)

# Several candidates with learner settings given in controls; the settings
# are stored with the pool and printed by pool_info().
if (requireNamespace("glmnet", quietly = TRUE) &&
    requireNamespace("randomForest", quietly = TRUE)) {
  pool <- model_train(Volume ~ Girth + Height, datasets::trees,
                      mff_split(1:15, 16:23, 24:31),
                      learners = list(lm = "lm", ridge = "ridge",
                                      forest_small = "random_forest",
                                      forest_large = "random_forest"),
                      controls = list(ridge = list(nfolds = 3),
                                      forest_small = list(ntree = 25),
                                      forest_large = list(ntree = 500,
                                                          nodesize = 3)),
                      seed = 1)
  pool_info(pool)
}
```

`plot.mff_fit` *Plot an MFF fit*

Description

Draw candidate weights, validation scores, tuning-search losses, or test diagnostics for a fixed or tuned MFF fit.

Usage

```
## S3 method for class 'mff_fit'
plot(
  x,
  type = c("weights", "weight_heatmap", "validation_scores", "test_scores",
    "observed_predicted", "series", "search"),
  metric = NULL,
  newdata = NULL,
  observed = NULL,
  show_all = FALSE,
  heatmap_values = TRUE,
  heatmap_digits = 3L,
  ...
)
```

Arguments

x	An mff_fit or mff_tuned.
type	Plot weights (grouped bars), a weight heatmap, validation scores, test scores, observed versus predicted test values, a test series, or tuning-search losses ("search", tuned fits only). Score plots mark the function selected on validation data, and the search plot labels every configuration and marks the selected one.
metric	A metric supported by evaluate(); defaults to the fit metric.
newdata	A candidate prediction matrix or an mff_prediction_pool for test plots. Candidate names and order must match the fit.
observed	Observed test outcomes for test plots. For a pool, stored test outcomes are used when observed is NULL.
show_all	Whether test plots show all functions of the fitted configuration. The default shows only the function selected on validation data; test plots never reselect a function.
heatmap_values	Whether to print weights in heatmap cells.
heatmap_digits	Nonnegative integer number of decimal places.
...	Additional graphics arguments passed to the base plotting function; they replace the defaults of the plot, such as the axis labels.

Value

The fitted object, invisibly.

Examples

```
pool <- mff_example()
fit <- tune_mff(pool, clusters = 2:3, method = "kmeans")
plot(fit, type = "weight_heatmap")
plot(fit, type = "observed_predicted", newdata = pool)
plot(fit, type = "search")
```

---

predict.mff_fit *Predict from an MFF fit*

Description

Apply validation-selected weights to aligned candidate predictions.

Usage

```
## S3 method for class 'mff_fit'
predict(object, newdata, type = c("selected", "all"), ...)
```

Arguments

object	An mff_fit or mff_tuned.
newdata	A finite numeric prediction matrix, or an mff_prediction_pool with a test matrix. Candidate names and their order must match the fit.
type	Use the validation-selected function or all functions in the fitted configuration. "all" does not mean all tuning-grid configurations.
...	Reserved for future use.

Value

A numeric vector for "selected" or a numeric matrix for "all". Observed test outcomes are never used to select a function.

Examples

```
pool <- mff_example()
fit <- fit_mff(pool, clusters = 2, method = "kmeans", seed = 3)
predict(fit, pool)
predict(fit, test_predictions(pool), type = "all")
```

---

print.mff_fit *Display an MFF fit*

Description

Displays the selected engine's convergence status, iteration count and messages. Iteration-limit and backend-warning results are identified explicitly; see fit_diagnostics() for the status values.

Usage

```
## S3 method for class 'mff_fit'
print(x, ...)
```

Arguments

x	An mff_fit or mff_tuned.
...	Reserved for future use.

Value

The input object, invisibly.

print.mff_prediction_pool *Display a prediction pool*

Description

Display a prediction pool

Usage

```
## S3 method for class 'mff_prediction_pool'
print(x, ...)
```

Arguments

x	An mff_prediction_pool.
...	Reserved for future use.

Value

The input, invisibly.

rolling_train *Build a forecasting pool by rolling-origin evaluation*

Description

Produces aligned validation and test forecasts for a univariate series. For every target time t of the validation and test blocks, each model is fitted to the observations up to the origin $t - h$, where h is horizon, and its h-step-ahead point forecast becomes the candidate prediction for t. A model therefore never receives an observation later than its origin. The test block consists of the last n_test observations. The validation block ends at the origin of the first test target, h steps before it, so that no validation outcome used to learn and select the aggregation is observed after any test origin; for $h > 1$ the $h - 1$ target times between the blocks are not used. The earliest origin must leave at least min_train observations.

Usage

```
rolling_train(
  y,
  models,
  horizon = 1L,
  n_validation,
  n_test,
  window = c("expanding", "sliding"),
```

```
  window_size = NULL,
  min_train = 2L,
  index = NULL,
  control = mff_control()
)
```

Arguments

y	A numeric vector or univariate ts without missing values. Models receive ts windows with the original frequency.
models	A named list of functions function(y, h) returning h point forecasts, either as a numeric vector or as an object with a numeric mean component, such as a forecast package forecast object.
horizon	Forecast horizon h, a positive integer. Pools for different horizons are built separately.
n_validation, n_test	Numbers of validation and test target times.
window	"expanding" uses all observations up to the origin; "sliding" uses the last window_size of them.
window_size	Window length for "sliding".
min_train	Minimum number of observations a model may receive.
index	Optional strictly increasing numeric, Date or POSIXct index of y. Defaults to time(y) for ts input and to positions otherwise.
control	An mff_control() list. seed gives each origin its own reproducible seed, and parallel/workers distribute origins over socket workers.

Value

An mff_prediction_pool with task = "forecasting". The origins of the validation and test forecasts and the unused target times between the blocks (all in the units of index) and the settings are stored in pool_info(pool)$metadata.

See Also

as_mff_pool() for forecasts already produced with fable.

Examples

```
y <- datasets::AirPassengers
models <- list(
  naive = function(y, h) rep(utils::tail(y, 1), h),
  seasonal_naive = function(y, h) utils::tail(y, 12)[seq_len(h)],
  drift = function(y, h) {
    n <- length(y)
    utils::tail(y, 1) + seq_len(h) * (y[n] - y[1]) / (n - 1)
  }
)
pool <- rolling_train(y, models, horizon = 1, n_validation = 24, n_test = 12)
```

```
pool
fit <- fit_mff(pool, clusters = 2, seed = 1)
evaluate(predict(fit, pool), test_observed(pool))
```

`summary.mff_fit` *Summarize an MFF fit*

Description

The printed summary includes convergence status, iterations and engine messages, including an explicit note when the iteration limit was reached or the backend emitted warnings.

Usage

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

## S3 method for class 'mff_fit_summary'
print(x, ...)
```

Arguments

`object`	An `mff_fit` or `mff_tuned`.
`...`	Reserved for future use.
`x`	An `mff_fit_summary`.

Value

An `mff_fit_summary` list. Fixed and tuned summaries share the same fields; search is NULL for fixed fits and a data frame for tuned fits.

`summary.mff_prediction_pool` *Summarize a prediction pool*

Description

Summarize a prediction pool

Usage

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

## S3 method for class 'mff_pool_summary'
print(x, ...)
```

Arguments

object	An mff_prediction_pool.
...	Reserved for future use.
x	A pool summary.

Value

A classed mff_pool_summary list with dimensions and candidate metadata.

---

tune_mff *Tune MFF configurations on validation predictions*

Description

Search an explicit grid and return the selected fit with search diagnostics.

Usage

```
tune_mff(
  pool,
  clusters,
  method = c("fcm", "pfcm", "gk", "kmeans"),
  m = seq(1.5, 2.5, by = 0.1),
  eta = 2,
  metric = "RMSE",
  standardize = FALSE,
  reduction = c("none", "pca"),
  nstart = 1L,
  iter_max = NULL,
  control = mff_control(),
  reltol = NULL,
  gamma = 0,
  beta = 1e+15
)
```

Arguments

pool	An mff_pool() prediction pool. At least two candidates are required.
clusters	A cluster count, or a vector of counts for tuning; each must be between two and the number of candidates. For k-means and PFCM, it must be smaller than the number of candidates: stats::kmeans() does not accept as many centers as candidates, and the PFCM penalties of equation (11) of Pal et al. (2005) are then not all positive.

`method` Membership method: fuzzy c-means (fcm), possibilistic fuzzy c-means (pfcm), Gustafson-Kessel (gk), or crisp k-means (kmeans). PFCM follows Pal, Pal, Keller and Bezdek (2005) and GK the modified algorithm with improved covariance estimation of Babuska, van der Veen and Kaymak (2002); both are implemented in this package (see Details).

`m` Fuzziness exponent greater than one; ignored for k-means.

`eta` PFCM typicality exponent, greater than 1; ignored for other methods.

`metric` Validation selection metric, using the formulas in `evaluate()`.

`standardize` Standardize each validation-observation coordinate across candidate profiles before clustering. Aggregation uses original predictions.

`reduction` Use original coordinates or all nonzero PCA components. PCA preserves Euclidean distances but need not preserve GK/PFCM solutions.

`nstart` Number of independent random starts per configuration, used by all four methods. The start with the smallest clustering objective is kept; validation error is not used to choose the start.

`iter_max` Iteration limit per start, used by all four methods. Defaults: kmeans 10, fcm 100, pfcm 100, gk 1000.

`control` A control list created by `mff_control()`.

`reltol` A finite positive convergence tolerance, or NULL to retain defaults: sqrt(.Machine$double.eps) for FCM, 1e-5 for PFCM and 1e-3 for GK. FCM uses relative objective change (control$reltol), PFCM the mean absolute change of the prototype coordinates, $||V_t - V_{t-1}||_1/(pc)$, and GK the largest absolute change of a membership between iterations. Thus the same value does not mean the same stopping rule across methods. The PFCM tolerance is absolute, as in Pal et al. (2005), and depends on the scale of the predictions: for predictions of very small or very large magnitude, rescale them or set `reltol` accordingly. K-means (`stats::kmeans()`) has no tolerance: it stops when no candidate changes cluster, so `reltol` must be NULL and only `nstart` and `iter_max` apply. K-means does not support reltol; leave it NULL and use iter_max.

`gamma` One or more GK weighting parameters in $[0, 1]$, searched as a grid dimension; ignored for other methods. See `fit_mff()` for the case in which `gamma` has no effect.

`beta` GK condition-number threshold for the fuzzy covariance matrices, greater than one; ignored for other methods. When the number of validation observations is at least the number of candidates, the fuzzy covariance matrices are singular, and their zero eigenvalues are set to the largest eigenvalue divided by `beta`; `beta` then enters the cluster volumes $\det(F_i)^{1/n}$. With `reduction = "pca"`, the directions without spread are removed before clustering.

Details

Grid order follows `expand.grid(m, clusters, eta)` for PFCM and `expand.grid(m, clusters, gamma)` for GK, with the method-irrelevant dimensions removed. Exact ties select the first grid row, then the first meta function. Failed rows are recorded; all-failed searches stop. Only validation data are sent to workers.

Value

An object of class `c("mff_tuned", "mff_fit")`. All `mff_fit` fields retain their meaning. Additional fields are a search data frame (grid, seed, success, loss, elapsed time and messages) and tuning controls.

Examples

```
pool <- mff_example()
fit <- tune_mff(pool, clusters = 2:3, method = "fcm", m = c(1.5, 2),
                nstart = 3, reltol = 1e-6, control = mff_control(seed = 12))
selected_configuration(fit)
search_results(fit)
```

`validation_predictions` *Access prediction-pool components*

Description

Retrieve aligned predictions, observed responses and candidate metadata without depending on the internal list layout.

Usage

```
validation_predictions(x)

validation_observed(x)

test_predictions(x)

test_observed(x)

candidate_names(x)

candidate_info(x)

pool_info(x)

## S3 method for class 'mff_pool_info'
print(x, ...)
```

Arguments

`x`	An `mff_prediction_pool`; for `print()`, an `mff_pool_info`.
`...`	Ignored.

Value

Predictions are numeric matrices; responses are numeric vectors. Missing test components return NULL. Candidate names are a character vector, candidate information a data frame, and pool information a list of class mff_pool_info containing task, type, indexes, and provenance. Its print method shows a compact summary; the full components are accessed with $ or [[.

Examples

```
pool <- mff_example()
head(validation_predictions(pool))
head(test_observed(pool))
pool_info(pool)
```

---

validation_scores *Access fitted MFF results*

Description

Retrieve validation scores, the selected function or configuration, and diagnostics without accessing internal list components.

Usage

```
validation_scores(object)

selected_function(object)

selected_configuration(object)

## S3 method for class 'mff_configuration'
print(x, ...)

search_results(object)

fit_diagnostics(object)
```

Arguments

object	An mff_fit or mff_tuned object.
x	An mff_configuration.
...	Ignored.

Value

validation_scores() returns a numeric matrix with one row per meta function; selected_function() returns its selected integer index; selected_configuration() returns a named list (class mff_configuration) of the method, the metric and the parameters that apply to the method; fit_diagnostics() returns engine diagnostics and preprocessing metadata. Its convergence element is "converged" when the selected start met the backend stopping rule (the tolerance for FCM, PFCM and GK; no reassignment for kmeans) before iter_max without warnings, "iteration_limit" when iter_max was reached first, and "backend_warning" when the rule was met but the backend emitted warnings, so convergence is not confirmed. search_results() returns the tuning data frame and requires mff_tuned.

Examples

```
fit <- tune_mff(mff_example(), clusters = 2:3, method = "kmeans")
validation_scores(fit)
selected_function(fit)
selected_configuration(fit)
search_results(fit)
```

---

weights.mff_fit *Extract MFF aggregation weights*

Description

Extract MFF aggregation weights

Usage

```
## S3 method for class 'mff_fit'
weights(object, type = c("selected", "all"), ...)

## S3 method for class 'mff_fit'
coef(object, ...)
```

Arguments

object	An mff_fit or mff_tuned.
type	Extract selected-function weights or all function weights.
...	Reserved for future use.

Value

A named numeric vector for "selected", or a candidate-by-function numeric matrix for "all". coef() returns selected aggregation coefficients, with no intercept.

Examples

```
fit <- fit_mff(mff_example(), clusters = 2, method = "kmeans", seed = 3)
weights(fit)
weights(fit, type = "all")
coef(fit)
```

Index

∗ **datasets**
    mff_diamonds, 12

as_mff_pool, 4
as_mff_pool(), *3*, *22*

boot_train, 6
boot_train(), *3*, *13*, *15, 16*

candidate_info
        *(*validation_predictions*)*, 26
candidate_info(), *15*
candidate_names
        *(*validation_predictions*)*, 26
coef.mff_fit *(*weights.mff_fit*)*, 28

evaluate, 7
evaluate(), *3*, *9*, *19*, *25*

fit_diagnostics *(*validation_scores*)*, 27
fit_diagnostics(), *10*, *20*
fit_mff, 8
fit_mff(), *3*, *7*, *15*, *25*

MFF *(*MFF-package*)*, 3
MFF-package, 3
mff_control, 10
mff_control(), *6*, *22*, *25*
mff_cv, 11
mff_cv(), *3*, *16, 17*
mff_diamonds, 12, *13*
mff_example, 13
mff_example(), *3*, *12*
mff_learner, 13
mff_learner(), *6*, *17*
mff_pool, 14
mff_pool(), *3–5*, *8*, *16*, *24*
mff_split, 16
mff_split(), *6*, *17*
model_train, 16
model_train(), *3*, *5*, *6*, *11–16*

plot.mff_fit, 18
pool_info *(*validation_predictions*)*, 26
pool_info(), *15*
predict.mff_fit, 19
predict.mff_fit(), *3*, *7*
print.mff_configuration
        *(*validation_scores*)*, 27
print.mff_fit, 20
print.mff_fit_summary
        *(*summary.mff_fit*)*, 23
print.mff_learner *(*mff_learner*)*, 13
print.mff_pool_info
        *(*validation_predictions*)*, 26
print.mff_pool_summary
        *(*summary.mff_prediction_pool*)*,
        23
print.mff_prediction_pool, 21

rolling_train, 21
rolling_train(), *3*, *5*, *15*

search_results *(*validation_scores*)*, 27
search_results(), *3*
selected_configuration
        *(*validation_scores*)*, 27
selected_configuration(), *3*, *10*
selected_function *(*validation_scores*)*,
        27
stats::kmeans(), *8, 9*, *24, 25*
summary.mff_fit, 23
summary.mff_prediction_pool, 23

test_observed *(*validation_predictions*)*,
        26
test_predictions
        *(*validation_predictions*)*, 26
test_predictions(), *15*
tune_mff, 24
tune_mff(), *3*, *7*, *15*

validation_observed
(validation_predictions), 26
validation_predictions, 26
validation_predictions(), 3, 15
validation_scores, 27
validation_scores(), 10

weights.mff_fit, 28
weights.mff_fit(), 3, 10