Package 'MFF'

May 7, 2026

Type Package

Title Meta Fuzzy Functions

Version 0.2.0

Description Implements Meta Fuzzy Functions (MFFs) for regression Tak and Ucan (2026) <doi:10.1016/j.asoc.2026.114592> by aggregating predictions from multiple base learners using membership weights learned in the prediction space of validation set. The package supports fuzzy and crisp meta-ensemble structures via Fuzzy C-Means (FCM) Tak (2018) <doi:10.1016/j.asoc.2018.08.009>, Possibilistic FCM (PFCM) Tak (2021) <doi:10.1016/j.ins.2021.01.024>, Gustafson–Kessel (GK) clustering, and k-means, and provides a workflow to (i) generate validation/test prediction matrices from common regression learners (linear and penalized regression via 'glmnet', random forests, gradient boosting with 'xgboost' and 'lightgbm'), (ii) fit cluster-wise meta fuzzy functions and compute membership-based weights, (iii) tune clustering-related hyperparameters (number of clusters/functions, fuzziness exponent, possibilistic regularization) via grid search on validation loss, and (iv) predict on new/test prediction matrices and evaluate performance using standard regression metrics (MAE, RMSE, MAPE, SMAPE, MSE, MedAE). This enables flexible, interpretable ensemble regression where different base models contribute to different meta components according to learned memberships.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Imports glmnet, randomForest, xgboost, lightgbm, e1071, ppclust, doParallel, foreach, parallel

Suggests knitr, rmarkdown, MASS

NeedsCompilation no

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

Maintainer Nihat Tak <nihattak@gmail.com>

Repository CRAN

Date/Publication 2026-04-01 19:50:14 UTC

Contents

`MFF-package` *MFF: Meta-Fuzzy Functions*

Description

Implements Meta Fuzzy Functions (MFFs) for regression by aggregating predictions from multiple base models using fuzzy clustering–derived weights. The package allows users to fit MFF models on top of diverse regression learners, including linear and penalized regression models, random forests, and gradient boosting methods. Membership weights are obtained via Fuzzy C-Means, Gustafson–Kessel clustering, Possibilistic FCM, or k-means, enabling both fuzzy and crisp ensemble structures. Clustering-related hyperparameters—such as the number of meta fuzzy functions, fuzziness exponent, and possibilistic regularization parameter—can be systematically tuned using validation data. A dedicated predict method is provided for producing test-set predictions from fitted or tuned MFF objects, along with evaluation tools for performance assessment.

Details

Provides tools for fitting and evaluating Meta Fuzzy Regression Functions by aggregating heterogeneous base regression models through fuzzy membership functions learned in the prediction space, with support for hyperparameter tuning and standard regression performance measures.

Available Functions

`mff()` Fits Meta Fuzzy Regression Functions models by estimating fuzzy membership weights from base-model prediction matrices and constructing cluster-wise meta regression functions.

`tune.mff()` Performs hyperparameter optimization for MFF models via grid search over clustering-related parameters, selecting the configuration that minimizes a chosen validation error metric.

`predict.mff()` S3 prediction method for fitted or tuned MFF objects, generating test-set predictions using membership-weighted aggregation of base-model outputs.

`evaluate()` Computes regression performance metrics (e.g., MAE, RMSE, MAPE, SMAPE, MSE, MedAE) for comparing meta fuzzy functions and base-model predictions.

`model.train()` Convenience function for training multiple regression models and producing validation and test prediction matrices suitable for MFF modeling.

`boot.train()` Convenience function for training bagging for multiple linear regression models and producing validation and test prediction matrices suitable for MFF modeling.

Acknowledgements

This study was supported by Scientific and Technological Research Council of Türkiye (TÜBİTAK) under Grant Number 125F138. The authors thank TÜBİTAK for its support.

Author(s)

Maintainer: Nihat Tak

Authors: Nihat Tak, Sadık Çoban

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., & Uçan, A. (2026). Meta fuzzy feature-selection-based regression functions. *Applied Soft Computing*, 190, 114592. doi:10.1016/j.asoc.2026.114592

`boot.train` *Train Bagging for Multiple Linear Regression and Produce Model Predictions*

Description

It generates prediction matrices for both validation and test sets by aggregating results from all bootstrap iterations.

Usage

```
boot.train(
  target,
  data,
  ntest,
  nvalid,
  B,
  seed = NULL,
  parallel = FALSE,
  ncores = NULL
)
```

Arguments

`target`	A character string specifying the name of the response variable.
`data`	A `data.frame` containing the target and predictor variables.
`ntest`	An integer specifying the number of observations to be assigned to the test set.

nvalid	An integer specifying the number of observations to be assigned to the validation set.
B	An integer specifying the number of bootstrap samples to generate.
seed	An optional numeric value for reproducibility.
parallel	Logical; if TRUE, bootstrap iterations are performed in parallel. Requires doParallel and foreach packages.
ncores	An integer specifying the number of CPU cores to use for parallel processing. Defaults to parallel::detectCores() - 1.

Details

Splits the input dataset into training, validation, and test sets, then performs bootstrap sampling on the training set to fit multiple linear models. Predictions are returned as matrices with dimension $N_{valid} \times B$ and $N_{test} \times B$. These matrices are the standard input x for mff() and tune.mff().

Value

A list containing the following components:

- pred_matrix_valid: A matrix of dimensions $N_{valid} \times B$ containing validation set predictions.
- pred_matrix_test: A matrix of dimensions $N_{test} \times B$ containing test set predictions.
- y_valid: A numeric vector of actual target values for the validation set.
- y_test: A numeric vector of actual target values for the test set.
- metadata: A list containing the number of training samples (ntrain), the number of bootstrap iterations (B), and parallel processing status.

Examples

```
results <- boot.train(target = "medv", data = MASS::Boston,
                          ntest = 50, nvalid = 50, B = 10,
                          seed = 123, parallel = FALSE)

# Accessing validation prediction matrix
head(results$pred_matrix_valid)
```

---

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. 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

mff for generating predictions, model.train for preparing base model prediction matrices, tune.mff for hyperparameter tuning performance.

Examples

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

---

mff *Generate Meta-Fuzzy Function*

Description

Construct meta-fuzzy functions by computing membership weights and cluster-wise predictions.

Usage

```
mff(
  x,
  y,
  c,
  m = 2,
  eta = 2,
  iter.max = NULL,
  nstart = 1,
  stand = FALSE,
  method = c("fcm", "pfcm", "kmeans", "gk")
)
```

Arguments

x	A numeric matrix of base-model predictions.
y	A numeric vector of true response values.
c	An integer specifying the number of clusters (functions).
m	A numeric fuzziness exponent (typically m = 2) used in FCM-type membership estimation. Larger values increase fuzziness (more diffuse memberships), while values closer to 1 yield sharper assignments.
eta	numeric regularization parameter used by the possibilistic FCM method (method = "pfcm").
iter.max	An integer specifying the maximum number of iterations allowed for the clustering algorithm.
nstart	n integer controlling the number of random initializations used when method = "kmeans" to improve robustness of the final clustering solution
stand	Logical; if TRUE, the transposed data is standardized (mean=0, sd=1) before clustering to ensure scale-invariance between models.
method	A character string selecting the membership-generation method. Available options are "fcm", "gk", "pfcm", and "kmeans".

Details

The *mff* function is the core constructor of the Meta Fuzzy Function (MFF) framework. It takes a matrix of base-model predictions and derives a membership-weight structure that defines multiple meta fuzzy functions using a selected membership-generation method. In the MFF setting, each base learner is represented by its prediction vector across samples; therefore, *mff* internally transposes the prediction matrix so that base models are treated as observations in the meta-clustering space. The resulting membership matrix is then used to form meta fuzzy function predictions through weighted aggregation of base-model outputs.

The function supports four membership-generation methods: classical Fuzzy C-Means (FCM), Gustafson-Kessel (GK) clustering for non-spherical structures, Possibilistic FCM (PFCM) producing softmax-like weights, and deterministic k-means for a situation with no uncertainty. After membership estimation, meta fuzzy function predictions are computed via linear combinations of base-model predictions and the learned membership- based weights, and the predictive performance of

each meta fuzzy function is assessed using *evaluate*. Membership weights are standardized column-wise to ensure that the total contribution of base models within each meta fuzzy function sums to one, facilitating interpretation and comparison across meta fuzzy functions.

Value

A list containing:

- `method`: The clustering method used for membership estimation.
- `weights`: A column-standardized membership (weight) matrix.
- `cluster_scores`: A data frame of evaluation metrics computed for each cluster(function).

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

Bezdek, J. C., Ehrlich, R., & Full, W. (1984). FCM: The fuzzy c-means clustering algorithm. *Computers & Geosciences*, 10(2), 191-203. doi:10.1016/00983004(84)900207

Cebeci, Z. (2019). Comparison of internal validity indices for fuzzy clustering. *Journal of Agricultural Informatics*, 10(2), 1-14. doi:10.17700/jai.2019.10.2.537

Gustafson, D. E., & Kessel, W. C. (1978). Fuzzy clustering with a fuzzy covariance matrix. In *1978 IEEE Conference on Decision and Control including the 17th Symposium on Adaptive Processes*, 761-766. doi:10.1109/CDC.1978.268028

Meyer, D., Dimitriadou, E., Hornik, K., Weingessel, A., & Leisch, F. (2024). *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*. R package version 1.7-16. https://CRAN.R-project.org/package=e1071

Gustafson, D. E., & Kessel, W. C. (1978). Fuzzy clustering with a fuzzy covariance matrix. In *1978 IEEE Conference on Decision and Control including the 17th Symposium on Adaptive Processes*, 761-766. doi:10.1109/CDC.1978.268028

Pal, N. R., Pal, K., Keller, J. M., & Bezdek, J. C. (2005). A possibilistic fuzzy c-means clustering algorithm. *IEEE Transactions on Fuzzy Systems*, 13(4), 517-530. doi:10.1109/TFUZZ.2004.840099

See Also

`model.train` for preparing input matrices, `predict.mff` for test set predictions, `tune.mff` for hyperparameter optimization, `evaluate` for performance metrics.

Examples

```
result_train <- model.train(
   target = "medv",
   data = MASS::Boston,
   ntest = 50,
   nvalid = 50,
   seed = 123
)

mff_model <- mff(result_train$pred_matrix_valid, result_train$y_valid, c = 4,
```

```
iter.max=100,nstart = 100,method = "kmeans")
mff_model
```

`model.train` *Train Multiple Regression and Produce Model Predictions*

Description

Train multiple base learners and generate prediction matrices for use in the Meta Fuzzy Function framework.

Usage

```
model.train(target, data, ntest, nvalid, seed = 123)
```

Arguments

`target`	character string specifying the name of the response variable in the data frame. This variable is excluded from the predictor set and used as the ground truth for training and evaluation.
`data`	A data frame containing the predictor variables and the target variable. All columns except target are treated as predictors.
`ntest`	An integer indicating the number of observations allocated to the test set. This sub- set is completely held out from model training and validation and is used for final performance assessment.
`nvalid`	An integer specifying the number of observations assigned to the validation set. Predic- tions on this subset are used to construct Meta Fuzzy Functions and to tune clustering-related hyperparameters.
`seed`	An integer used to set the random seed for reproducibility.

Details

Splits data into train/validation/test, then fits a suite of base learners and generates predictions for validation and test. Predictions are returned as matrices with dimension $N_{test} \times M$. These matrices are the standard input x for `mff()` and `tune.mff()`.

Base learners include linear regression, Lasso, Ridge, Elastic Net, Random Forest, XGBoost, and LightGBM, as implemented by the package dependencies.

If a selected method requires hyperparameter optimization, this optimization is not performed within the `model.train` function. Instead, all hyperparameters are fixed a priori using commonly accepted default values.

Training base models is not a mandatory step to use the MFF framework. The `model.train` function is provided as a convenience utility only. Users may independently train any number of prediction methods using external workflows or software and directly supply their predictions as inputs to the MFF.

Accordingly, the `model.train` function can be completely skipped while still fully utilizing the MFF framework with precomputed model outputs.

Value

A list containing:

- `pred_matrix_valid`: A numeric matrix of validation-set predictions, where each column corresponds to a base model.
- `pred_matrix_test`: A numeric matrix of test-set predictions generated by the same base models.
- `y_valid`: A numeric vector of true response values for the validation set.
- `y_test`: A numeric vector of true response values for the test set.

References

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5-32. doi:10.1023/A:1010933404324

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785-794. doi:10.1145/2939672.2939785

Chen, T., He, T., Benesty, M., et al. (2025). *xgboost: Extreme Gradient Boosting*. R package version 3.1.2.1. https://CRAN.R-project.org/package=xgboost

Ke, G., Meng, Q., Finley, T., et al. (2017). LightGBM: A highly efficient gradient boosting decision tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 3149-3157.

Liaw, A., & Wiener, M. (2002). Classification and Regression by randomForest. *R News*, 2(3), 18-22. https://CRAN.R-project.org/doc/Rnews/

Shi, Y., Ke, G., Soukhavong, D., et al. (2025). *lightgbm: Light Gradient Boosting Machine*. R package version 4.6.0. https://CRAN.R-project.org/package=lightgbm

Tay, J. K., Narasimhan, B., & Hastie, T. (2023). Elastic Net Regularization Paths for All Generalized Linear Models. *Journal of Statistical Software*, 106(1), 1-31. doi:10.18637/jss.v106.i01

See Also

`mff` for the main framework application, `tune.mff` for hyperparameter optimization,

Examples

```
boston <- MASS::Boston
 result <- model.train(
   target = "medv",
   data = boston,
   ntest = 50,
   nvalid = 50,
   seed = 123
 )

 head(result$pred_matrix_valid)
 head(result$pred_matrix_test)
```

`predict.mff` *Predict method for objects of class mff*

Description

The *predict* method for objects of class *mff* is used to generate predictions from the Meta Fuzzy Function framework on the test dataset.

Usage

```
## S3 method for class 'mff'
predict(object, pred_matrix, type = c("best", "all"), ...)
```

Arguments

`object`	an object of class mff (returned by mff() or tune.mff()).
`pred_matrix`	numeric matrix ($N_{test} \times M$) of base predictions for the new data (e.g., test set).
`type`	'best' uses the optimal configuration/cluster selected by tune.mff(); 'all' returns outputs for all tuned configurations.
`...`	further arguments passed to or from other methods.

Details

In this setting, the input to *predict* is the matrix of test-set predictions produced by the base models that were trained in the earlier stage using *model.train*.

Let $X_{test} \in \mathbb{R}^{N_{test} \times M}$ denote the prediction matrix of M trained base models on the test dataset, where N_{test} is the number of test observations. Each column of X_{test} corresponds to the predictions generated by a single base learner. Using the membership-based weight matrix $W = [w_{mc}]$ obtained during validation, meta fuzzy function predictions on the test set are computed as

$$\hat{y}^{(c)}_{MFF,test} = \sum_{m=1}^{M} w^{(m)(c)} \hat{y}^{(m)}_{test} \quad (11)$$

where $\hat{y}^{(m)}_{test}$ denotes the test-set prediction vector produced by base model m, and $c = 1, \ldots, C$ indexes the meta fuzzy functions.

The predict function supports two prediction modes, controlled by the type argument:

- type = "best" returns predictions obtained from the single meta fuzzy function that achieved the optimal validation performance during training or tuning (see Eq. 9).
- type = "all" returns predictions from all meta fuzzy functions, allowing users to examine alternative predictive components within the MFF model.

When type = "best" is selected, the final prediction for test observation i is given by

$$\hat{y}_{i,final} = \sum_{m=1}^{M} w^{(m)(c^*)} \hat{y}_{i,test}^{(m)} \quad (12)$$

where c^* denotes the index of the best-performing meta fuzzy function selected based on validation performance.

Computes cluster-wise predictions via membership-weighted aggregation of base-model predictions. For inspection and reporting purposes, a rounded copy of the membership weights (four decimal places) is returned.

If a single best-performing cluster is defined via tune.mff, the corresponding weight vector is also returned.

Value

- `mff_preds`: A numeric matrix of meta fuzzy function predictions on the test set. When "best", this matrix reduces to a single column.
- `mff_weights`: The membership based weight matrix. If a single meta fuzzy function is selected, the corresponding weight vector is returned.

See Also

`mff`, `tune.mff`, `evaluate`

Examples

```
res <- model.train(target="medv", data=MASS::Boston, ntest=50, nvalid=50, seed = 123)
fit <- tune.mff(res$pred_matrix_valid, res$y_valid, max_c=6, mff.method="kmeans")
out <- predict(fit, pred_matrix=res$pred_matrix_test, type="best")
head(out$mff_preds)
out$mff_weights
```

`tune.mff` *Hyperparameter Search for Meta-Fuzzy Function*

Description

The *tune.mff* function performs hyperparameter optimization via grid search for Meta Fuzzy Functions (MFFs) by searching over clustering-related parameter combinations and selecting the configuration that yields the lowest validation error.

Usage

```
tune.mff(
  x,
  y,
  max_c,
  m_seq = seq(1.1, 3, by = 0.1),
  eta_seq = seq(1.1, 3, by = 0.4),
  iter.max = NULL,
  nstart = 1,
  seed = 123,
  mff.method = c("fcm", "pfcm", "kmeans", "gk"),
  eval.method = c("MAE", "RMSE", "MAPE", "SMAPE", "MSE", "MedAE"),
  stand = FALSE,
  parallel = FALSE,
  num_cores = NULL,
  logging = TRUE
)
```

Arguments

x	A numeric matrix of base-model predictions with dimensions $N_{test} \times M$. Each column corresponds to a base learner.
y	numeric vector of validation targets. This vector is used to evaluate meta fuzzy function predictions.
max_c	An integer specifying the maximum number of clusters to be considered in the search.
m_seq	A numeric vector of candidate values for the fuzziness exponent m used in FCM-type methods.
eta_seq	A numeric vector of candidate values for the probabilistic regularization parameter η used when mff.method = "pfcm".
iter.max	An integer specifying the maximum number of iterations allowed for the clustering algorithm within each grid evaluation..
nstart	integer; An integer controlling the number of random initializations for k-means when mff.method = "kmeans".
seed	An integer used to set the random seed for reproducibility during weight computation and parameter search.
mff.method	A character string selecting the membership-generation method.
eval.method	A character string specifying the metric used to select the best-performing meta fuzzy function.
stand	Logical; if TRUE, the transposed data is standardized (mean=0, sd=1) before clustering to ensure scale-invariance between models.
parallel	Logical; if TRUE, the grid search process is executed in parallel to accelerate hyperparameter optimization.

num_cores	An integer specifying the number of CPU cores to utilize for parallel processing. If NULL (default), the function automatically detects and uses all available cores minus one (parallel::detectCores() - 1).
logging	A logical flag indicating whether progress information is printed during the search.

Details

Given a matrix of base-model predictions and the corresponding validation targets, *tune.mff* repeatedly calls *mff* to compute membership weights, generate meta fuzzy function predictions, and evaluate these predictions using a user-specified metric. The best configuration is determined by the minimum value of the selected evaluation metric among the scores obtained from the meta fuzzy function predictions produced under each candidate setting.

The search space depends on the selected membership-generation method. For classical Fuzzy C-Means ("fcm"), the function explores combinations of the number of clusters c and the fuzziness index m. For possibilistic FCM ("pfcm"), the grid additionally includes the possibilistic regularization parameter η. For k-means ("kmeans"), the search is performed only over the number of clusters(c). The function returns the best-performing configuration together with the corresponding weight structure, the index of the best-performing meta fuzzy function, and the full set of evaluation results, enabling transparent reporting and reproducible model selection.

Value

- algorithm: The selected membership-generation method.
- eval.method: The evaluation metric used in model selection.
- weights: The membership (weight) matrix associated with the best-performing configuration.
- best_params: A list containing the hyperparameters that achieved the best score.
- best_cluster: The index of the meta fuzzy function yielding the minimum validation error.
- best_weight: The weight vector corresponding to the best-performing meta fuzzy function.
- best_scores: The full set of evaluation scores for all meta fuzzy function predictions under the best configuration.

See Also

mff, model.train, predict.mff

Examples

```
res <- model.train(target="medv", data=MASS::Boston, ntest=50, nvalid=50, seed = 123)
fit <- tune.mff(res$pred_matrix_valid, res$y_valid, max_c=6, mff.method="kmeans")
out <- predict(fit, pred_matrix=res$pred_matrix_test, type="best")
head(out$mff_preds)
out$mff_weights
```

Index