Package 'OptimalBinningWoE'

September 1, 2026

Type Package

Title Optimal Binning and Weight of Evidence Framework for Modeling

Version 1.13.5

Date 2026-08-31

Description High-performance implementation of 37 optimal binning algorithms (16 categorical, 21 numerical) for Weight of Evidence ('WoE') transformation, credit scoring, and risk modeling. Includes advanced methods such as Mixed Integer Linear Programming ('MILP'), Genetic Algorithms, Simulated Annealing, and Monotonic Regression. Features automatic method selection based on Information Value ('IV') maximization, strict monotonicity enforcement, and efficient handling of large datasets via 'Rcpp'. Provides automated variable screening by Information Value strength and bin ordering, and generation of the equivalent 'SQL' 'CASE' expressions for in-database scoring. Fully integrated with the 'tidymodels' ecosystem for building robust machine learning pipelines.
Based on methods described in Siddiqi (2006) <doi:10.1002/9781119201731> and Navas-Palencia (2020) <doi:10.48550/arXiv.2001.08025>.

License MIT + file LICENSE

URL https://github.com/evandeilton/OptimalBinningWoE, https://evandeilton.github.io/OptimalBinningWoE/

BugReports https://github.com/evandeilton/OptimalBinningWoE/issues

Depends R (>= 4.1.0)

Encoding UTF-8

Language en-US

Imports Rcpp, recipes, rlang, tibble, dials, stats, utils, graphics

LinkingTo Rcpp, RcppEigen, RcppNumerical

Suggests testthat (>= 3.0.0), data.table, openxlsx, glmnet, generics, knitr, rmarkdown, tidymodels, workflows, parsnip, tune

Config/testthat/edition 3

SystemRequirements C++17

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Author José Evandeilton Lopes [aut, cre, cph] (ORCID:
<https://orcid.org/0009-0007-5887-4084>)

Maintainer José Evandeilton Lopes <evandeilton@gmail.com>

Repository CRAN

Date/Publication 2026-08-31 22:00:47 UTC

Contents

.categorical_only_algorithms 4
.numerical_only_algorithms 4
.universal_algorithms 4
.valid_algorithms 5
bake.step_obwoe 5
control.obwoe 6
control.obwoe_scorecard 8
obcorr 10
obwoe 12
obwoe_algorithm 19
obwoe_algorithms 20
obwoe_apply 20
obwoe_bin_cutoff 23
obwoe_gains 24
obwoe_gains_score 28
obwoe_gains_variable 30
obwoe_max_bins 32
obwoe_min_bins 33
obwoe_prune 34
obwoe_psi 35
obwoe_report 36
obwoe_scale 38
obwoe_score 39
obwoe_scorecard 40
obwoe_select 43
obwoe_sql 48
ob_apply_woe_cat 53
ob_apply_woe_num 54
ob_categorical_cm 57
ob_categorical_dmiv 60
ob_categorical_dp 64
ob_categorical_fetb 69
ob_categorical_gmb 74
ob_categorical_ivb 81
ob_categorical_jedi 86

ob_categorical_jedi_mwoe . . . 95
ob_categorical_mba . . . 104
ob_categorical_milp . . . 111
ob_categorical_mob . . . 114
ob_categorical_sab . . . 117
ob_categorical_sblp . . . 120
ob_categorical_sketch . . . 123
ob_categorical_swb . . . 126
ob_categorical_udt . . . 129
ob_cutpoints_cat . . . 132
ob_cutpoints_num . . . 134
ob_numerical_bb . . . 136
ob_numerical_cm . . . 139
ob_numerical_dmiv . . . 141
ob_numerical_dp . . . 144
ob_numerical_ewb . . . 147
ob_numerical_fast_mdlp . . . 149
ob_numerical_fetb . . . 152
ob_numerical_ir . . . 154
ob_numerical_jedi . . . 157
ob_numerical_jedi_mwoe . . . 159
ob_numerical_kmb . . . 161
ob_numerical_ldb . . . 164
ob_numerical_lpdb . . . 168
ob_numerical_mblp . . . 171
ob_numerical_mdlp . . . 176
ob_numerical_mob . . . 183
ob_numerical_mrblp . . . 189
ob_numerical_oslp . . . 194
ob_numerical_sketch . . . 198
ob_numerical_ubsd . . . 202
ob_numerical_udt . . . 206
ob_preprocess . . . 209
plot.obwoe . . . 213
plot.obwoe_gains . . . 215
predict.obwoe_scorecard . . . 215
prep.step_obwoe . . . 216
print.obwoe . . . 217
print.obwoe_sql . . . 218
print.step_obwoe . . . 218
required_pkgs.step_obwoe . . . 219
step_obwoe . . . 219
summary.obwoe . . . 224
tidy.step_obwoe . . . 226
tunable.step_obwoe . . . 227

Index **228**

.categorical_only_algorithms *Categorical-Only Algorithms*

Description

Internal function returning algorithm identifiers that support only categorical features.

Usage

```
.categorical_only_algorithms()
```

Value

A character vector of categorical-only algorithm names.

.numerical_only_algorithms *Numerical-Only Algorithms*

Description

Internal function returning algorithm identifiers that support only numerical features.

Usage

```
.numerical_only_algorithms()
```

Value

A character vector of numerical-only algorithm names.

.universal_algorithms *Universal Algorithms*

Description

Internal function returning algorithm identifiers that support both numerical and categorical features.

Usage

```
.universal_algorithms()
```

Value

A character vector of universal algorithm names.

.valid_algorithms *Valid Binning Algorithms*

Description

Internal function returning the vector of all valid algorithm identifiers supported by the OptimalBinningWoE package. Used for validation and parameter definition.

Usage

```
.valid_algorithms()
```

Value

A character vector of valid algorithm names including "auto".

bake.step_obwoe *Apply the Optimal Binning Transformation*

Description

Applies the learned binning and WoE transformation to new data. This method is called by bake and should not be invoked directly.

Usage

```
## S3 method for class 'step_obwoe'
bake(object, new_data, ...)
```

Arguments

object	A trained step_obwoe object.
new_data	A tibble or data frame to transform.
...	Additional arguments (currently unused).

Value

A tibble with transformed columns according to the output parameter.

`control.obwoe` *Control Parameters for Optimal Binning Algorithms*

Description

Constructs a validated list of control parameters for the obwoe master interface. These parameters govern the behavior of all supported binning algorithms, including convergence criteria, minimum bin sizes, and optimization limits.

Usage

```
control.obwoe(
  bin_cutoff = 0.05,
  max_n_prebins = 20,
  convergence_threshold = 1e-06,
  max_iterations = 1000,
  bin_separator = "%;%",
  verbose = FALSE,
  ...
)
```

Arguments

`bin_cutoff`	Numeric value in $(0, 1)$ specifying the minimum proportion of total observations that a bin must contain. Bins with fewer observations are merged with adjacent bins. Serves as a regularization mechanism to prevent overfitting and ensure statistical stability of WoE estimates. Recommended range: 0.02 to 0.10. Default is 0.05 (5%).
`max_n_prebins`	Integer specifying the maximum number of initial bins created before optimization. For high-cardinality categorical features, categories with similar event rates are pre-merged until this limit is reached. Higher values preserve more granularity but increase computational cost. Typical range: 10 to 50. Default is 20.
`convergence_threshold`	Numeric value specifying the tolerance for algorithm convergence. Iteration stops when the absolute change in Information Value between successive iterations falls below this threshold: $\|IV_t - IV_{t-1}\| < \epsilon$. Smaller values yield more precise solutions at higher computational cost. Typical range: 10^{-4} to 10^{-8}. Default is 10^{-6}.
`max_iterations`	Integer specifying the maximum number of optimization iterations. Prevents infinite loops in degenerate cases. If the algorithm does not converge within this limit, it returns the best solution found. Typical range: 100 to 10000. Default is 1000.
`bin_separator`	Character string used to concatenate category names when multiple categories are merged into a single bin. Should be a string unlikely to appear in actual category names. Default is `"%;%"`.

verbose	Logical indicating whether to print progress messages during feature processing. Useful for debugging or monitoring long-running jobs. Default is FALSE.
...	Additional named parameters reserved for algorithm-specific extensions. Currently unused but included for forward compatibility.

Details

Parameter Impact on Results:

bin_cutoff: Lower values allow smaller bins, which may capture subtle patterns but risk unstable WoE estimates. The variance of WoE estimates increases as $1/n_i$ where n_i is the bin size. For bins with fewer than ~30 observations, consider using Laplace or Bayesian smoothing (applied automatically by most algorithms).

max_n_prebins: Critical for categorical features with many levels. If a feature has 100 categories, setting max_n_prebins = 20 will pre-merge similar categories into 20 groups before optimization.

For *numerical* features this parameter is a modelling decision, not an implementation detail, and the default of 20 is not neutral. Pre-binning runs before any algorithm sees the data, so a heavy-tailed variable whose signal sits in the upper tail can have that tail smeared across a single quantile cell and lost before optimization begins. The effect is silent: the fit reports converged = TRUE and no warning.

Benchmarks on two open datasets (76,020 x 369 and 590,540 x 454, five-fold held-out IV, min_bins and max_bins held fixed) show the size of it. Raising max_n_prebins from 20 to 200 moved the median held-out IV of heavy-tailed numerical predictors by **+129%** and **+39%** respectively, while leaving discrete and categorical predictors untouched. The bin count barely moved, so this is the placement of cut points changing, not extra splits.

It moves results in **both** directions, which is why the default is left alone. On one of the two benchmarks, raising it improved the portfolio but cost 15% of held-out IV on the twenty-five strongest predictors – exactly the ones a scorecard would keep – while inflating IV on weak predictors no model would select. No cheap rule separated the two cases: gating on the tail ratio worked cleanly on one dataset and captured almost nothing on the other, and for roughly half of all predictors the default of 20 was already the best value tested.

The practical advice is therefore per-variable rather than global: leave the default, and tune it against *held-out* data for continuous predictors with a long tail. Tuning on in-sample IV will always favour more resolution and is not evidence of anything.

convergence_threshold: Trade-off between precision and speed. For exploratory analysis, 10^{-4} is sufficient. For production models requiring reproducibility, use 10^{-8} or smaller.

Value

An S3 object of class "obwoe_control" containing all specified parameters. This object is validated and can be passed directly to obwoe.

See Also

obwoe for the main binning interface.

Examples

```
# Default control parameters
```

```
ctrl_default <- control.obwoe()
print(ctrl_default)

# Conservative settings for production
ctrl_production <- control.obwoe(
  bin_cutoff = 0.03,
  max_n_prebins = 30,
  convergence_threshold = 1e-8,
  max_iterations = 5000
)

# Aggressive settings for exploration
ctrl_explore <- control.obwoe(
  bin_cutoff = 0.01,
  max_n_prebins = 50,
  convergence_threshold = 1e-4,
  max_iterations = 500
)
```

`control.obwoe_scorecard` *Control Parameters for the Scorecard Pipeline*

Description

Settings for the stages of `obwoe_scorecard` that have no home in `control.obwoe` or `obwoe_select`.

Usage

```
control.obwoe_scorecard(
  corr_cutoff = 0.7,
  corr_method = "pearson",
  pdo = 20,
  score_ref = 600,
  odds_ref = 50,
  direction = "higher_is_safer",
  n_groups = 10L,
  na_woe = 0,
  drop_negative = TRUE,
  max_abs_coef = 15,
  min_events_per_variable = 20,
  engine_fallback = FALSE,
  overwrite = TRUE,
  digits = 6L
)
```

Arguments

`corr_cutoff`	Numeric in $(0, 1]$. Absolute correlation in the WoE space at or above which two variables are treated as redundant. Default `0.70`. Set to `1` to disable pruning.
`corr_method`	Character string passed to `obcorr`. Default `"pearson"`.
`pdo, score_ref, odds_ref, direction`	Passed to `obwoe_scale`.
`n_groups`	Integer. Number of score bands in the gains and stability tables. Default `10`.
`na_woe`	Numeric. WoE given to a value that falls in no fitted bin — an unseen category, or a missing value where the binner built no missing bin. Default `0`, which is the population-average contribution.
`drop_negative`	Logical. Refit without any variable whose coefficient on WoE comes out negative? Default `TRUE`. A negative coefficient means the variable is fighting the rest of the model, and a scorecard with one cannot be signed off.
`max_abs_coef`	Numeric. Coefficients above this in absolute value are flagged as separation. Default `15`.
`min_events_per_variable`	Numeric. Events per variable in the model below which the fit is flagged as thin. Default `20`, the usual rule of thumb.
`engine_fallback`	Logical. If the requested engine's package is not installed, fall back to `"glm"` with a warning instead of failing? Default `FALSE`.
`overwrite`	Logical. Overwrite `file` if it exists? Default `TRUE`.
`digits`	Integer. Rounding used in the workbook. Default `6`.

Value

An object of class `"obwoe_scorecard_control"`.

See Also

`obwoe_scorecard`

Examples

```
control.obwoe_scorecard(pdo = 25, score_ref = 700, corr_cutoff = 0.8)
```

obcorr *Compute Multiple Robust Correlations Between Numeric Variables*

Description

This function computes various correlation coefficients between all pairs of numeric variables in a data frame. It implements several classical and robust correlation measures, including Pearson, Spearman, Kendall, Hoeffding's D, Distance Correlation, Biweight Midcorrelation, and Percentage Bend correlation.

Usage

```
obcorr(df, method = "all", threads = 0L)
```

Arguments

df	A data frame containing numeric variables. Non-numeric columns will be automatically excluded. At least two numeric variables are required.
method	A character string specifying which correlation method(s) to compute. Possible values are: • "all": Compute all available correlation methods (default). • "pearson": Compute only Pearson correlation. • "spearman": Compute only Spearman correlation. • "kendall": Compute only Kendall correlation. • "hoeffding": Compute only Hoeffding's D. • "distance": Compute only distance correlation. • "biweight": Compute only biweight midcorrelation. • "pbend": Compute only percentage bend correlation. • "robust": Compute robust correlations (biweight and pbend). • "alternative": Compute alternative correlations (hoeffding and distance).
threads	An integer specifying the number of threads to use for parallel computation. If 0 (the default), at most 2 threads are used, honouring any lower limit already set via OMP_NUM_THREADS; this follows CRAN policy, which requires packages not to use more than two cores by default. Pass a positive value to request more explicitly. The number of threads never affects the results or their row order. Ignored if OpenMP is not available.

Details

The function supports multiple correlation methods simultaneously and utilizes OpenMP for parallel computation when available.

Available correlation methods:

- **Pearson**: Standard linear correlation coefficient.
- **Spearman**: Rank-based correlation coefficient.

- **Kendall**: Kendall's tau-b correlation coefficient.
- **Hoeffding**: Hoeffding's D statistic (scaled by 30).
- **Distance**: Distance correlation (Székely et al., 2007).
- **Biweight**: Biweight midcorrelation (robust alternative).
- **Pbend**: Percentage bend correlation (robust alternative).

Value

A data frame with the following columns:

`x`, `y` Names of the variable pairs being correlated.

`pearson` Pearson correlation coefficient.

`spearman` Spearman rank correlation coefficient.

`kendall` Kendall's tau-b correlation coefficient.

`hoeffding` Hoeffding's D statistic (scaled).

`distance` Distance correlation coefficient.

`biweight` Biweight midcorrelation coefficient.

`pbend` Percentage bend correlation coefficient.

The exact columns returned depend on the `method` parameter.

Note

- Missing values (NA) are handled appropriately for each correlation method.
- For robust methods (biweight, pbend), fallback to Pearson correlation occurs when there are insufficient data points or numerical instability.
- Hoeffding's D requires at least 5 complete pairs.
- Distance correlation is computed without forming NxN distance matrices for memory efficiency.
- When OpenMP is available, computations are automatically parallelized across variable pairs.

References

Székely, G.J., Rizzo, M.L., and Bakirov, N.K. (2007). Measuring and testing dependence by correlation of distances. The Annals of Statistics, 35(6), 2769-2794.

Wilcox, R.R. (1994). The percentage bend correlation coefficient. Psychometrika, 59(4), 601-616.

Examples

```
# Create sample data
set.seed(123)
n <- 100
df <- data.frame(
  x1 = rnorm(n),
  x2 = rnorm(n),
  x3 = rt(n, df = 3), # Heavy-tailed distribution
```

```
  x4 = sample(c(0, 1), n, replace = TRUE), # Binary variable
  category = sample(letters[1:3], n, replace = TRUE) # Non-numeric column
)

# Add some relationships
df$x2 <- df$x1 + rnorm(n, 0, 0.5)
df$x3 <- df$x1^2 + rnorm(n, 0, 0.5)

# Compute all correlations
result_all <- obcorr(df)
head(result_all)

# Compute only robust correlations
result_robust <- obcorr(df, method = "robust")

# Compute only Pearson correlation with 2 threads
result_pearson <- obcorr(df, method = "pearson", threads = 2)
```

---

obwoe *Unified Optimal Binning and Weight of Evidence Transformation*

Description

Master interface for optimal discretization and Weight of Evidence (WoE) computation across numerical and categorical predictors. This function serves as the primary entry point for the **OptimalBinningWoE** package, providing automatic feature type detection, intelligent algorithm selection, and unified output structures for seamless integration into credit scoring and predictive modeling workflows.

Usage

```
obwoe(
  data,
  target,
  feature = NULL,
  min_bins = 2,
  max_bins = 7,
  algorithm = "auto",
  control = control.obwoe()
)
```

Arguments

`data`	A `data.frame` containing the predictor variables (features) and the response variable (target). All features to be binned must be present in this data frame. The data frame should not contain list-columns.

`target` Character string specifying the column name of the response variable. Must be a binary outcome encoded as integers `0` (non-event) and `1` (event), or a multinomial outcome encoded as integers `0, 1, 2, ..., K`. Missing values in the target are not permitted.

`feature` Optional character vector specifying which columns to process. If `NULL` (default), all columns except `target` are processed. Features containing only missing values are automatically skipped with a warning.

`min_bins` Integer specifying the minimum number of bins. Must satisfy $2 \leq$ `min_bins` $\leq$ `max_bins`. Algorithms may produce fewer bins if the data has insufficient unique values. Default is 2.

Read as a floor, but it frequently behaves as the binding constraint rather than a safety net. Several algorithms stop merging as soon as their own criterion is satisfied, which can be well before `max_bins`, so `min_bins` ends up setting the partition. On the German Credit `amount` variable with `max_bins = 5`, the default binning returns two bins and an IV of 0.0016; requesting `min_bins = 4` returns an IV of 0.0824 on the same data and the same algorithm. Raising it is a legitimate way to ask for a finer partition, and worth trying when a continuous predictor collapses to two bins under a larger `max_bins`.

`max_bins` Integer specifying the maximum number of bins. Controls the granularity of discretization. Higher values capture more detail but risk overfitting. Typical values range from 5 to 10 for credit scoring applications. Default is 7.

`algorithm` Character string specifying the binning algorithm. Use `"auto"` (default) for automatic selection based on target type: `"jedi"` for binary targets, `"jedi_mwoe"` for multinomial. See Details for the complete algorithm taxonomy.

`control` A list of algorithm-specific control parameters created by `control.obwoe`. Provides fine-grained control over convergence thresholds, bin cutoffs, and other optimization parameters.

Details

Theoretical Foundation:

Weight of Evidence (WoE) transformation is a staple of credit scoring methodology, originating from information theory and the concept of evidential support (Good, 1950; Kullback, 1959). For a bin i, the WoE is defined as:

$$WoE_i = \ln\left(\frac{p_i}{n_i}\right) = \ln\left(\frac{N_{i,1}/N_1}{N_{i,0}/N_0}\right)$$

where:

- $N_{i,1}$ = number of events (target=1) in bin i
- $N_{i,0}$ = number of non-events (target=0) in bin i
- N_1, N_0 = total events and non-events, respectively
- $p_i = N_{i,1}/N_1$ = proportion of events in bin i
- $n_i = N_{i,0}/N_0$ = proportion of non-events in bin i

The Information Value (IV) quantifies the total predictive power of a binning:

$$IV = \sum_{i=1}^{k}(p_i - n_i) \times WoE_i = \sum_{i=1}^{k}(p_i - n_i) \times \ln\left(\frac{p_i}{n_i}\right)$$

where k is the number of bins. IV is equivalent to the Kullback-Leibler divergence between the event and non-event distributions.

Algorithm Taxonomy:

The package provides 28 algorithms organized by supported feature types:

Universal Algorithms (both numerical and categorical):

ID	Full Name	Method
`jedi`	Joint Entropy-Driven Information	Heuristic + IV optimization
`jedi_mwoe`	JEDI Multinomial WoE	Extension for K>2 classes
`cm`	ChiMerge	Bottom-up chi-squared merging
`dp`	Dynamic Programming	Exact optimal IV partitioning
`dmiv`	Decision Tree MIV	Recursive partitioning
`fetb`	Fisher's Exact Test	Statistical significance-based
`mob`	Monotonic Optimal Binning	IV-optimal with monotonicity
`sketch`	Sketching	Probabilistic data structures
`udt`	Unsupervised Decision Tree	Entropy-based without target

Numerical-Only Algorithms:

ID	Description
`bb`	Branch and Bound (exact search)
`ewb`	Equal Width Binning (unsupervised)
`fast_mdlp`	Fast MDLP with pruning
`ir`	Isotonic Regression
`kmb`	K-Means Binning
`ldb`	Local Density Binning
`lpdb`	Local Polynomial Density
`mblp`	Monotonic Binning LP
`mdlp`	Minimum Description Length
`mrblp`	Monotonic Regression LP
`oslp`	Optimal Supervised LP
`ubsd`	Unsupervised Std-Dev Based

Categorical-Only Algorithms:

ID	Description
`gmb`	Greedy Monotonic Binning
`ivb`	Information Value DP (exact)
`mba`	Modified Binning Algorithm
`milp`	Mixed Integer LP
`sab`	Simulated Annealing
`sblp`	Similarity-Based LP
`swb`	Sliding Window Binning

Automatic Type Detection:

Feature types are detected as follows:

- **Numerical**: `numeric` or `integer` vectors not of class `factor`
- **Categorical**: `character`, `factor`, or `logical` vectors

When `algorithm = "auto"`, the function selects:

- `"jedi"` for binary targets (recommended for most use cases)
- `"jedi_mwoe"` for multinomial targets (K > 2 classes)

IV Interpretation Guidelines:

Siddiqi (2006) provides the following IV thresholds for variable selection:

IV Range	Predictive Power
< 0.02	Unpredictive
0.02 - 0.10	Weak
0.10 - 0.30	Medium
0.30 - 0.50	Strong
> 0.50	Suspicious (likely overfitting)

Computational Considerations:

Time complexity varies by algorithm:

- **JEDI, ChiMerge, MOB**: $O(n \log n + k^2 m)$ where n = observations, k = bins, m = iterations
- **Dynamic Programming**: $O(n \cdot k^2)$ for exact solution
- **Equal Width**: $O(n)$ (fastest, but unsupervised)
- **MILP, SBLP**: Potentially exponential (NP-hard problems)

For large datasets ($n > 10^6$), consider:

1. Using `algorithm = "sketch"` for approximate streaming
2. Reducing `max_n_prebins` via `control.obwoe()`
3. Sampling the data before binning

Value

An S3 object of class `"obwoe"` containing:

`results` Named list where each element contains the binning result for a single feature, including:

- `bin` Character vector of bin labels/intervals
- `woe` Numeric vector of Weight of Evidence per bin
- `iv` Numeric vector of Information Value contribution per bin
- `count` Integer vector of observation counts per bin
- `count_pos` Integer vector of positive (event) counts per bin
- `count_neg` Integer vector of negative (non-event) counts per bin
- `cutpoints` Numeric vector of bin boundaries (numerical only)
- `converged` Logical indicating algorithm convergence
- `iterations` Integer count of optimization iterations

`summary` Data frame with one row per feature containing: `feature` (name), `type` (numerical/categorical), `algorithm` (used), `n_bins` (count), `total_iv` (sum), `error` (logical flag)

`target` Name of the target column

`target_type` Detected type: `"binary"` or `"multinomial"`

`n_features` Number of features processed

`call` The matched function call for reproducibility

References

Good, I. J. (1950). Probability and the Weighing of Evidence. *Griffin, London*.

Kullback, S. (1959). Information Theory and Statistics. *Wiley, New York*.

Siddiqi, N. (2006). Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring. *John Wiley & Sons*. doi:10.1002/9781119201731

Thomas, L. C., Edelman, D. B., & Crook, J. N. (2002). Credit Scoring and Its Applications. *SIAM Monographs on Mathematical Modeling and Computation*. doi:10.1137/1.9780898718317

Navas-Palencia, G. (2020). Optimal Binning: Mathematical Programming Formulation and Solution Approach. *Expert Systems with Applications*, 158, 113508. doi:10.1016/j.eswa.2020.113508

Zeng, G. (2014). A Necessary Condition for a Good Binning Algorithm in Credit Scoring. *Applied Mathematical Sciences*, 8(65), 3229-3242.

See Also

`control.obwoe` for algorithm-specific parameters, `obwoe_algorithms` to list all available algorithms with capabilities, `print.obwoe` for display methods, `ob_apply_woe_num` and `ob_apply_woe_cat` to apply WoE transformations to new data.

For individual algorithms with full parameter control: `ob_numerical_jedi`, `ob_categorical_jedi`, `ob_numerical_mdlp`, `ob_categorical_ivb`.

Examples

```
# ============================================================================
# Example 1: Basic Usage with Mixed Feature Types
# ============================================================================
set.seed(42)
n <- 2000

# Simulate credit scoring data
df <- data.frame(
  # Numerical features
  age = pmax(18, pmin(80, rnorm(n, 45, 15))),
  income = exp(rnorm(n, 10, 0.8)),
  debt_ratio = rbeta(n, 2, 5),
  credit_history_months = rpois(n, 60),

  # Categorical features
  education = sample(c("High School", "Bachelor", "Master", "PhD"),
    n,
    replace = TRUE, prob = c(0.35, 0.40, 0.20, 0.05)
  ),
  employment = sample(c("Employed", "Self-Employed", "Unemployed", "Retired"),
```

```
    n,
    replace = TRUE, prob = c(0.60, 0.20, 0.10, 0.10)
  ),

  # Binary target (default probability varies by features)
  target = rbinom(n, 1, 0.15)
)

# Process all features with automatic algorithm selection
result <- obwoe(df, target = "target")
print(result)

# View detailed summary
print(result$summary)

# Access results for a specific feature
age_bins <- result$results$age
print(data.frame(
  bin = age_bins$bin,
  woe = round(age_bins$woe, 3),
  iv = round(age_bins$iv, 4),
  count = age_bins$count
))

# ============================================================================
# Example 2: Using a Specific Algorithm
# ============================================================================

# Use MDLP for numerical features (entropy-based)
result_mdlp <- obwoe(df,
  target = "target",
  feature = c("age", "income"),
  algorithm = "mdlp",
  min_bins = 3,
  max_bins = 6
)

cat("\nMDLP Results:\n")
print(result_mdlp$summary)

# ============================================================================
# Example 3: Custom Control Parameters
# ============================================================================

# Fine-tune algorithm behavior
ctrl <- control.obwoe(
  bin_cutoff = 0.02, # Minimum 2% per bin
  max_n_prebins = 30, # Allow more initial bins
  convergence_threshold = 1e-8
)

result_custom <- obwoe(df,
  target = "target",
```

```
  feature = "debt_ratio",
  algorithm = "jedi",
  control = ctrl
)

cat("\nCustom JEDI Result:\n")
print(result_custom$results$debt_ratio$bin)

# ============================================================================
# Example 4: Comparing Multiple Algorithms
# ============================================================================

algorithms <- c("jedi", "mdlp", "ewb", "mob")
iv_comparison <- sapply(algorithms, function(algo) {
  tryCatch(
    {
      res <- obwoe(df, target = "target", feature = "income", algorithm = algo)
      res$summary$total_iv
    },
    error = function(e) NA_real_
  )
})

cat("\nAlgorithm Comparison (IV for 'income'):\n")
print(sort(iv_comparison, decreasing = TRUE))

# ============================================================================
# Example 5: Feature Selection Based on IV
# ============================================================================

# Process all features and select those with IV > 0.02
result_all <- obwoe(df, target = "target")

strong_features <- result_all$summary[
  result_all$summary$total_iv >= 0.02 & !result_all$summary$error,
  c("feature", "total_iv", "n_bins")
]
strong_features <- strong_features[order(-strong_features$total_iv), ]

cat("\nFeatures with IV >= 0.02 (predictive):\n")
print(strong_features)

# ============================================================================
# Example 6: Handling Algorithm Compatibility
# ============================================================================

# MDLP only works for numerical - will fail for categorical
result_mixed <- obwoe(df,
  target = "target",
  algorithm = "mdlp"
)

# Check for errors
```

```
cat("\nCompatibility check:\n")
print(result_mixed$summary[, c("feature", "type", "error")])
```

---

obwoe_algorithm *Binning Algorithm Parameter*

Description

A qualitative tuning parameter for selecting the optimal binning algorithm in step_obwoe.

Usage

```
obwoe_algorithm(values = NULL)
```

Arguments

values	A character vector of algorithm names to include in the parameter space. If NULL (default), includes all 29 algorithms (28 specific algorithms plus "auto").

Details

The algorithms are organized into three groups:

Universal (support both numerical and categorical features): "auto", "jedi", "jedi_mwoe", "cm", "dp", "dmiv", "fetb", "mob", "sketch", "udt"

Numerical only: "bb", "ewb", "fast_mdlp", "ir", "kmb", "ldb", "lpdb", "mblp", "mdlp", "mrblp", "oslp", "ubsd"

Categorical only: "gmb", "ivb", "mba", "milp", "sab", "sblp", "swb"

When tuning with mixed feature types, consider restricting values to universal algorithms only.

Value

A dials qualitative parameter object.

See Also

step_obwoe, obwoe

Examples

```
# Default: all algorithms
obwoe_algorithm()

# Restrict to universal algorithms for mixed data
obwoe_algorithm(values = c("jedi", "mob", "dp", "cm"))

# Numerical-only algorithms
```

```
obwoe_algorithm(values = c("mdlp", "fast_mdlp", "ewb", "ir"))
```

---

obwoe_algorithms *List Available Algorithms*

Description

Returns a data frame with all available binning algorithms.

Usage

```
obwoe_algorithms()
```

Value

A data frame with algorithm information.

Examples

```
obwoe_algorithms()
```

---

obwoe_apply *Apply Weight of Evidence Transformations to New Data*

Description

Applies the binning and Weight of Evidence (WoE) transformations learned by obwoe to new data. This is the scoring function for deploying WoE-based models in production. For each feature, the function assigns observations to bins and maps them to their corresponding WoE values.

Usage

```
obwoe_apply(
  data,
  obj,
  suffix_bin = "_bin",
  suffix_woe = "_woe",
  keep_original = TRUE,
  na_woe = 0
)
```

Arguments

data	A data.frame containing the features to transform. Must include all features present in the obj results. The target column is optional; if present, it will be included in the output.
obj	An object of class "obwoe" returned by obwoe.
suffix_bin	Character string suffix for bin columns. Default is "_bin".
suffix_woe	Character string suffix for WoE columns. Default is "_woe".
keep_original	Logical. If TRUE (default), include the original feature columns in the output. If FALSE, only bin and WoE columns are returned.
na_woe	Numeric value to assign when an observation cannot be mapped to a bin (e.g., new categories not seen during training), **and** as the fallback for a missing (NA) categorical value when the binner built no dedicated missing-value bin. Default is 0. When a missing-value bin *was* fitted (see Details), NA inputs get that bin's WoE instead of na_woe.

Details

Bin Assignment Logic:

Numerical Features: Observations are assigned to bins based on cutpoints stored in the obwoe object. The cut() function is used with intervals $(a_i, a_{i+1}]$ where $a_0 = -\infty$ and $a_k = +\infty$.

Categorical Features: Categories are matched directly to bin labels. Categories not seen during training are assigned NA for bin and na_woe for WoE.

A missing (NA) categorical value is routed to whichever fitted bin represents missing values, if the binner built one – categorical algorithms (e.g. ob_categorical_jedi) map NA to the token "NA" before fitting, so a bin containing that token (alone or merged with other categories) is treated as the missing-value bin, exactly as obwoe_sql's null_to_na_bin = TRUE default does for IS NULL. na_woe is used for NA only when no such bin exists. **This changed in 1.13.1**: earlier versions always returned na_woe for NA, ignoring any fitted missing-value bin, which disagreed with the generated SQL.

Production Deployment:

For production scoring, it is recommended to:

1. Train the binning model using obwoe() on the training set
2. Save the fitted object with saveRDS()
3. Load and apply using obwoe_apply() on new data

The WoE-transformed features can be used directly as inputs to logistic regression or other linear models, enabling interpretable credit scorecards.

Value

A data.frame containing:

target The target column (if present in data)

<feature> Original feature values (if keep_original = TRUE)

<feature>_bin Assigned bin label for each observation

<feature>_woe Weight of Evidence value for the assigned bin

References

Siddiqi, N. (2006). Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring. *John Wiley & Sons*. doi:10.1002/9781119201731

See Also

`obwoe` for fitting the binning model, `summary.obwoe` for model diagnostics.

Examples

```
# ============================================================================
# Example 1: Basic Usage - Train and Apply
# ============================================================================
set.seed(42)
n <- 1000

# Training data
train_df <- data.frame(
  age = rnorm(n, 40, 15),
  income = exp(rnorm(n, 10, 0.8)),
  education = sample(c("HS", "BA", "MA", "PhD"), n, replace = TRUE),
  target = rbinom(n, 1, 0.15)
)

# Fit binning model
model <- obwoe(train_df, target = "target")

# New data for scoring (could be validation/test set)
new_df <- data.frame(
  age = c(25, 45, 65),
  income = c(20000, 50000, 80000),
  education = c("HS", "MA", "PhD")
)

# Apply transformations
scored <- obwoe_apply(new_df, model)
print(scored)

# Use WoE features for downstream modeling
woe_cols <- grep("_woe$", names(scored), value = TRUE)
print(woe_cols)

# ============================================================================
# Example 2: Without Original Features
# ============================================================================

scored_compact <- obwoe_apply(new_df, model, keep_original = FALSE)
print(scored_compact)
```

obwoe_bin_cutoff *Bin Cutoff Parameter*

Description

A quantitative tuning parameter for the minimum bin support (proportion of observations per bin) in step_obwoe.

Usage

```
obwoe_bin_cutoff(range = c(0.01, 0.1), trans = NULL)
```

Arguments

range	A two-element numeric vector specifying the minimum and maximum values for the parameter. Default is c(0.01, 0.10).
trans	A transformation object from the scales package, or NULL for no transformation. Default is NULL.

Details

The bin cutoff specifies the minimum proportion of observations that each bin must contain. Bins with fewer observations are merged with adjacent bins. This serves as a regularization mechanism:

- Lower values (e.g., 0.01) allow smaller bins, capturing subtle patterns but risking unstable WoE estimates.
- Higher values (e.g., 0.10) enforce larger bins, producing more stable estimates but potentially missing important patterns.

For credit scoring, values between 0.02 and 0.05 are typical. Regulatory guidelines often require minimum bin sizes for model stability.

Value

A dials quantitative parameter object.

See Also

step_obwoe, control.obwoe

Examples

```
obwoe_bin_cutoff()
obwoe_bin_cutoff(range = c(0.02, 0.08))
```

obwoe_gains *Gains Table Statistics for Credit Risk Scorecard Evaluation*

Description

Computes a comprehensive gains table (also known as a lift table or decile analysis) for evaluating the discriminatory power of credit scoring models and optimal binning transformations. The gains table is a fundamental tool in credit risk management for model validation, cutoff selection, and regulatory reporting (Basel II/III, IFRS 9).

This function accepts three types of input:

1. An `"obwoe"` object from `obwoe` (uses stored binning)
2. A `data.frame` from `obwoe_apply` (uses bin/WoE columns)
3. Any `data.frame` with a grouping variable (e.g., score deciles)

Usage

```
obwoe_gains(
  obj,
  target = NULL,
  feature = NULL,
  use_column = c("auto", "bin", "woe", "direct"),
  sort_by = c("id", "woe", "event_rate", "bin"),
  n_groups = NULL
)
```

Arguments

`obj`	Input object: an `"obwoe"` object, a `data.frame` from `obwoe_apply`, or any `data.frame` containing a grouping variable and target values.
`target`	Integer vector of binary target values (0/1) or the name of the target column in `obj`. Required for `data.frame` inputs. For `"obwoe"` objects, the target is extracted automatically.
`feature`	Character string specifying the feature/variable to analyze. For `"obwoe"` objects: defaults to the feature with highest IV. For `data.frame` objects: can be any column name representing groups (e.g., `"age_bin"`, `"age_woe"`, `"score_decile"`).
`use_column`	Character string specifying which column type to use when `obj` is a `data.frame` from `obwoe_apply`: `"bin"` Use the `<feature>_bin` column (default) `"woe"` Use the `<feature>_woe` column (groups by WoE values) `"auto"` Automatically detect: use `_bin` if available `"direct"` Use the `feature` column name directly (for any variable)
`sort_by`	Character string specifying sort order for bins: `"id"` The algorithm's own internal bin order - default

"woe" Descending WoE (highest risk first)

"event_rate" Descending event rate

"bin" The bins' natural (level) order if obj is a factor column, alphabetical order of the bin labels otherwise

Note: the default is "id", not "woe" – kept as-is in 1.13.1 to avoid changing existing callers' output; only this documentation was wrong before.

n_groups Integer. For continuous variables (e.g., scores), the number of groups (deciles) to create. Default is NULL (use existing groups). Set to 10 for standard decile analysis.

Details

Gains Table Construction:

The gains table is constructed by:

1. Sorting observations by risk score or WoE (highest risk first)
2. Grouping into bins (pre-defined or created via quantiles)
3. Computing bin-level and cumulative statistics

The table enables assessment of model rank-ordering ability: a well-calibrated model should show monotonically increasing event rates as risk score increases.

Bin-Level Statistics (18 metrics):

Column	Formula	Description
bin	-	Bin label or interval
count	n_i	Total observations in bin
count_pct	n_i/N	Proportion of total population
pos_count	$n_{i,1}$	Event count (Bad, target=1)
neg_count	$n_{i,0}$	Non-event count (Good, target=0)
pos_rate	$n_{i,1}/n_i$	Event rate (Bad rate) in bin
neg_rate	$n_{i,0}/n_i$	Non-event rate (Good rate)
pos_pct	$n_{i,1}/N_1$	Distribution of events
neg_pct	$n_{i,0}/N_0$	Distribution of non-events
odds	$n_{i,1}/n_{i,0}$	Odds of event
log_odds	$\ln(\text{odds})$	Log-odds (logit)
woe	$\ln(p_i/q_i)$	Weight of Evidence
iv	$(p_i - q_i) \cdot WoE_i$	Information Value contribution
cum_pos_pct	$\sum_{j \le i} p_j$	Cumulative events captured
cum_neg_pct	$\sum_{j \le i} q_j$	Cumulative non-events
ks	$\lvert F_1(i) - F_0(i) \rvert$	KS statistic at bin
lift	pos_rate$/\bar{p}$	Lift over random
capture_rate	cum_pos_pct	Cumulative capture rate

Global Performance Metrics:

Kolmogorov-Smirnov (KS) Statistic: Maximum absolute difference between cumulative distributions of events and non-events. Measures the model's ability to separate populations.

$$KS = \max_i |F_1(i) - F_0(i)|$$

KS Range	Interpretation
< 20%	Poor discrimination
20-40%	Acceptable
40-60%	Good
60-75%	Very good
> 75%	Excellent (verify for data leakage)

Gini Coefficient: Measure of inequality between event and non-event distributions. Equivalent to 2*AUC - 1, representing the area between the Lorenz curve and the line of equality.

$$Gini = 2 \times AUC - 1$$

Area Under ROC Curve (AUC): Probability that a randomly chosen event is ranked higher than a randomly chosen non-event. Computed via the trapezoidal rule.

Total Information Value (IV): Sum of IV contributions across all bins. See obwoe for interpretation guidelines.

Use Cases:

Model Validation: Verify rank-ordering (monotonic event rates) and acceptable KS/Gini.

Cutoff Selection: Identify the bin where the model provides optimal separation for business rules (e.g., auto-approve above score X).

Population Stability: Compare gains tables over time to detect model drift.

Regulatory Reporting: Generate metrics required by Basel II/III and IFRS 9 frameworks.

Value

An S3 object of class `"obwoe_gains"` containing:

`table` Data frame with 18 statistics per bin (see Details)

`metrics` Named list of global performance metrics:

- `ks` Kolmogorov-Smirnov statistic (%)
- `gini` Gini coefficient (%)
- `auc` Area Under ROC Curve
- `total_iv` Total Information Value
- `ks_bin` Bin where maximum KS occurs

`feature` Feature/variable name analyzed

`n_bins` Number of bins/groups

`n_obs` Total observations

`event_rate` Overall event rate

References

Siddiqi, N. (2006). Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring. *John Wiley & Sons*. doi:10.1002/9781119201731

Thomas, L. C., Edelman, D. B., & Crook, J. N. (2002). Credit Scoring and Its Applications. *SIAM Monographs on Mathematical Modeling and Computation*. doi:10.1137/1.9780898718317

Anderson, R. (2007). The Credit Scoring Toolkit: Theory and Practice for Retail Credit Risk Management. *Oxford University Press*.

Hand, D. J., & Henley, W. E. (1997). Statistical Classification Methods in Consumer Credit Scoring: A Review. *Journal of the Royal Statistical Society: Series A*, 160(3), 523-541. doi:10.1111/j.1467-985X.1997.00078.x

See Also

obwoe for optimal binning, obwoe_apply for scoring new data, plot.obwoe_gains for visualization (cumulative gains, KS, lift).

Examples

```
# ============================================================================
# Example 1: From obwoe Object (Standard Usage)
# ============================================================================
set.seed(42)
n <- 1000
df <- data.frame(
  age = rnorm(n, 40, 15),
  income = exp(rnorm(n, 10, 0.8)),
  score = rnorm(n, 600, 100),
  target = rbinom(n, 1, 0.15)
)

model <- obwoe(df, target = "target")
gains <- obwoe_gains(model, feature = "age")
print(gains)

# Access metrics
cat("KS:", gains$metrics$ks, "%\n")
cat("Gini:", gains$metrics$gini, "%\n")

# ============================================================================
# Example 2: From obwoe_apply Output - Using Bin Column
# ============================================================================
scored <- obwoe_apply(df, model)

# Default: uses age_bin column
gains_bin <- obwoe_gains(scored,
  target = df$target, feature = "age",
  use_column = "bin"
)

# ============================================================================
# Example 3: From obwoe_apply Output - Using WoE Column
# ============================================================================
# Group by WoE values (continuous analysis)
gains_woe <- obwoe_gains(scored,
  target = df$target, feature = "age",
  use_column = "woe", n_groups = 5
)
```

```
# ============================================================================
# Example 4: Any Variable - Score Decile Analysis
# ============================================================================
# Create score deciles manually
df$score_decile <- cut(df$score,
  breaks = quantile(df$score, probs = seq(0, 1, 0.1)),
  include.lowest = TRUE, labels = 1:10
)

# Analyze score deciles directly
gains_score <- obwoe_gains(df,
  target = "target", feature = "score_decile",
  use_column = "direct"
)
print(gains_score)

# ============================================================================
# Example 5: Automatic Decile Creation
# ============================================================================
# Use n_groups to automatically create quantile groups
gains_auto <- obwoe_gains(df,
  target = "target", feature = "score",
  use_column = "direct", n_groups = 10
)
```

`obwoe_gains_score` *Compute Comprehensive Gains Table from Binning Results*

Description

This function serves as a high-performance engine (implemented in C++) to calculate a comprehensive set of credit scoring and classification metrics based on pre-aggregated binning results. It takes a list of bin counts and computes metrics such as Information Value (IV), Weight of Evidence (WoE), Kolmogorov-Smirnov (KS), Gini, Lift, and various entropy-based divergence measures.

Usage

```
obwoe_gains_score(binning_result)
```

Arguments

`binning_result` A named `list` or `data.frame` containing the following atomic vectors (all must have the same length):

- `id` Numeric vector of bin identifiers. Determines the sort order for cumulative metrics (e.g., KS, Recall).
- `bin` Character vector of bin labels/intervals.
- `count` Numeric vector of total observations per bin (O_i).

`count_pos` Numeric vector of positive (event) counts per bin (E_i).

`count_neg` Numeric vector of negative (non-event) counts per bin (NE_i).

Details

Mathematical Definitions:

Let E_i and NE_i be the number of events and non-events in bin i, and E_{total}, NE_{total} be the population totals.

Weight of Evidence (WoE) & Information Value (IV):

$$WoE_i = \ln\left(\frac{E_i/E_{total}}{NE_i/NE_{total}}\right)$$

$$IV_i = \left(\frac{E_i}{E_{total}} - \frac{NE_i}{NE_{total}}\right) \times WoE_i$$

Kolmogorov-Smirnov (KS):

$$KS_i = \left|\sum_{j=1}^{i} \frac{E_j}{E_{total}} - \sum_{j=1}^{i} \frac{NE_j}{NE_{total}}\right|$$

Lift:

$$Lift_i = \frac{E_i/(E_i + NE_i)}{E_{total}/(E_{total} + NE_{total})}$$

Kullback-Leibler Divergence (Bernoulli): Measures the divergence between the bin's event rate p_i and the global event rate p_{global}:

$$KL_i = p_i \ln\left(\frac{p_i}{p_{global}}\right) + (1 - p_i) \ln\left(\frac{1 - p_i}{1 - p_{global}}\right)$$

Value

A `data.frame` with the following columns (metrics calculated per bin):

Identifiers `id`, `bin`

Counts & Rates `count`, `pos`, `neg`, `pos_rate` (π_i), `neg_rate` ($1 - \pi_i$), `count_perc` (O_i/O_{total})

Distributions (Shares) `pos_perc` ($D_1(i)$: Share of Bad), `neg_perc` ($D_0(i)$: Share of Good)

Cumulative Statistics `cum_pos`, `cum_neg`, `cum_pos_perc` (CDF_1), `cum_neg_perc` (CDF_0), `cum_count_perc`

Credit Scoring Metrics `woe`, `iv`, `total_iv`, `ks`, `lift`, `odds_pos`, `odds_ratio`

Advanced Metrics `gini_contribution`, `log_likelihood`, `kl_divergence`, `js_divergence`

Classification Metrics `precision`, `recall`, `f1_score`

Examples

```
# Manually constructed binning result
bin_res <- list(
  id = 1:3,
  bin = c("Low", "Medium", "High"),
  count = c(100, 200, 50),
  count_pos = c(5, 30, 20),
  count_neg = c(95, 170, 30)
)

gt <- obwoe_gains_score(bin_res)
print(gt[, c("bin", "woe", "iv", "ks")])
```

---

obwoe_gains_variable *Compute Gains Table for a Binned Feature Vector*

Description

Calculates a full gains table by aggregating a raw binned dataframe against a binary target. Unlike obwoe_gains_score which expects pre-aggregated counts, this function takes observation-level data, aggregates it by the specified group variable (bin, WoE, or ID), and then computes all statistical metrics.

Usage

```
obwoe_gains_variable(binned_df, target, group_var = "bin")
```

Arguments

binned_df	A data.frame resulting from a binning transformation (e.g., via obwoe_apply), containing at least the following columns: feature Original feature values (optional, for reference). bin Character vector of bin labels. woe Numeric vector of Weight of Evidence values. idbin Numeric vector of bin IDs (required for correct sorting).
target	A numeric vector of binary outcomes (0 for non-event, 1 for event). Must have the same length as binned_df. Missing values are not allowed.
group_var	Character string specifying the aggregation key. Options: • "bin": Group by bin label (default). • "woe": Group by WoE value. • "idbin": Group by bin ID.

Details

Aggregation and Sorting: The function first aggregates the binary target by the specified `group_var`. Crucially, it uses the `idbin` column to sort the resulting groups. This ensures that cumulative metrics (like KS and Gini) are calculated based on the logical order of the bins (e.g., low score to high score), not alphabetical order.

Advanced Metrics: In addition to standard credit scoring metrics, this function computes:

- **Jensen-Shannon Divergence**: A symmetrized and smoothed version of KL divergence, useful for measuring stability between the bin distribution and the population distribution.
- **F1-Score, Precision, Recall**: Treating each bin as a potential classification threshold.

Value

A `data.frame` containing the same extensive set of metrics as `obwoe_gains_score`, aggregated by `group_var` and sorted by `idbin`.

References

Siddiqi, N. (2006). *Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring*. Wiley.

Kullback, S., & Leibler, R. A. (1951). On Information and Sufficiency. *The Annals of Mathematical Statistics*.

Examples

```
# Mock data representing a binned feature
df_binned <- data.frame(
  feature = c(10, 20, 30, 10, 20, 50),
  bin = c("Low", "Mid", "High", "Low", "Mid", "High"),
  woe = c(-0.5, 0.2, 1.1, -0.5, 0.2, 1.1),
  idbin = c(1, 2, 3, 1, 2, 3)
)
target <- c(0, 0, 1, 1, 0, 1)

# Calculate gains table grouped by bin ID
gt <- obwoe_gains_variable(df_binned, target, group_var = "idbin")

# Inspect key metrics
print(gt[, c("id", "count", "pos_rate", "lift", "js_divergence")])
```

obwoe_max_bins *Maximum Bins Parameter*

Description

A quantitative tuning parameter for the maximum number of bins in step_obwoe.

Usage

```
obwoe_max_bins(range = c(5L, 20L), trans = NULL)
```

Arguments

range	A two-element integer vector specifying the minimum and maximum values for the parameter. Default is c(5L, 20L).
trans	A transformation object from the scales package, or NULL for no transformation. Default is NULL.

Details

The maximum number of bins limits algorithm complexity and helps prevent overfitting. Higher values allow more granular discretization but may capture noise rather than signal.

For credit scoring applications, max_bins is typically set between 5 and 10 to balance predictive power with interpretability. Values above 15 are rarely necessary and may indicate overfitting.

Value

A dials quantitative parameter object.

See Also

step_obwoe, obwoe_min_bins

Examples

```
obwoe_max_bins()
obwoe_max_bins(range = c(4L, 12L))
```

obwoe_min_bins *Minimum Bins Parameter*

Description

A quantitative tuning parameter for the minimum number of bins in step_obwoe.

Usage

```
obwoe_min_bins(range = c(2L, 5L), trans = NULL)
```

Arguments

range	A two-element integer vector specifying the minimum and maximum values for the parameter. Default is c(2L, 5L).
trans	A transformation object from the scales package, or NULL for no transformation. Default is NULL.

Details

The minimum number of bins constrains the algorithm to create at least this many bins. Setting min_bins = 2 allows maximum flexibility, while higher values ensure more granular discretization.

For credit scoring applications, min_bins is typically set between 2 and 4 to avoid forcing artificial splits on weakly predictive variables.

Value

A dials quantitative parameter object.

See Also

step_obwoe, obwoe_max_bins

Examples

```
obwoe_min_bins()
obwoe_min_bins(range = c(3L, 7L))
```

obwoe_prune *Prune Redundant Variables by Correlation*

Description

Greedy, iterative removal of variables that carry the same information as a better-ranked one. Information Value ranks variables one at a time; two variables can both be strong and say the same thing, which a model on Weight of Evidence shows as an unstable or sign-flipped coefficient.

Usage

```
obwoe_prune(x, ranking, cutoff = 0.7, method = "pearson")
```

Arguments

x	A data.frame of numeric columns to be compared — normally the WoE-transformed predictors, which is the space the model sees — or a pre-computed pairwise table from obcorr with columns x, y and a correlation column.
ranking	Character vector of variable names, best first. Variables absent from ranking are treated as ranked last.
cutoff	Numeric in $(0, 1]$. Absolute correlation at or above which two variables are considered redundant. Default 0.70.
method	Character string passed to obcorr when x is a data.frame of columns. Default "pearson".

Details

The pass is iterative on purpose. Evaluating every pair independently removes both B and C from a chain $A \sim B \sim C$ even when B and C are unrelated to each other: once B is gone, the pair (B, C) no longer exists. Each iteration therefore drops one variable — the worst-ranked member of the strongest surviving pair — and recomputes what is left.

Value

A list with:

keep Character vector of surviving variables.

dropped data.frame of variable, correlated_with and correlation — one row per removal, in the order the removals happened.

pairs The pairwise table, with an abs_corr column.

cutoff The cutoff used.

See Also

obcorr, obwoe_select

Examples

```
set.seed(1)
n <- 500
a <- rnorm(n)
df <- data.frame(a = a, b = a + rnorm(n, 0, 0.2), c = rnorm(n))

# b duplicates a; a is ranked better, so b goes
obwoe_prune(df, ranking = c("a", "b", "c"), cutoff = 0.7)
```

`obwoe_psi` *Population Stability Index*

Description

Compares how a variable, or a score, is distributed between two samples — typically the development window and a later vintage.

Usage

```
obwoe_psi(base, compare, levels = NULL, breaks = NULL, n_groups = 10L)
```

Arguments

`base`	Vector observed in the reference sample.
`compare`	Vector observed in the sample being monitored.
`levels`	Character vector of the categories or bin labels to compare over. Required when the inputs are not numeric; defaults to the union of the values seen.
`breaks`	Numeric vector of cut points used to band numeric inputs. `NULL` (default) bands them at the deciles of `base`.
`n_groups`	Integer. Number of bands when `breaks` is `NULL`. Default `10`.

Details

$$\mathrm{PSI} = \sum_i (p_i - q_i)\, \ln \frac{p_i}{q_i}$$

which is the same symmetrised Kullback-Leibler divergence that defines Information Value, applied to two vintages of one variable instead of to two classes of one sample. The conventional reading is < 0.10 stable, 0.10–0.25 worth watching, > 0.25 act.

A band that is populated in one sample and empty in the other makes the divergence infinite. That is reported as `Inf` rather than smoothed away, because a segment that has vanished is exactly what monitoring exists to catch; the per-band table shows which one.

Value

A list with `psi` (the total), `table` (the contribution of each band) and `flag`, one of `"stable"`, `"watch"` or `"act"` at the conventional 0.10 and 0.25 thresholds.

See Also

`obwoe_scorecard`

Examples

```
set.seed(1)
obwoe_psi(rnorm(1000), rnorm(1000))
obwoe_psi(rnorm(1000), rnorm(1000, mean = 0.6))
```

`obwoe_report` *Write a Scorecard Workbook*

Description

Turns a fitted `obwoe_scorecard` into a multi-sheet `.xlsx` file: the points table a branch officer reads, the evidence a validation team asks for, and the SQL a deployment engineer runs.

Usage

```
obwoe_report(
  x,
  file,
  control = NULL,
  table = "your_table",
  dialect = "ansi",
  keep_columns = NULL
)
```

Arguments

`x`	An `"obwoe_scorecard"` object.
`file`	Path to the `.xlsx` file to write.
`control`	An `"obwoe_scorecard_control"` object; only `digits` and `overwrite` are used. Defaults to the settings stored on `x` where available.
`table`	Character string naming the source table used in the generated SQL. Default `"your_table"`.
`dialect`	SQL dialect for the deployment sheets, passed to `obwoe_sql`. Default `"ansi"`.
`keep_columns`	Columns the generated SQL carries through unchanged.

Details

The workbook has one sheet per stage, in the order a reviewer reads them:

`01_Model_Summary`	provenance, the funnel, headline metrics
`02_Scorecard`	**the deliverable**: one row per variable and bin, with the integer points
`03_Coefficients`	the fit, with the sign check and standard errors
`04_Bin_Statistics`	the gains table of every binned variable used in training
`05_Screening`	every candidate and why it lived or died
`06_Correlations`	redundancy in the WoE space and what was pruned
`07_Score_Gains`	rank ordering per sample, with observed versus predicted
`08_Stability_PSI`	the score and every variable, per sample
`09_Cutoff_Strategy`	approval rate, bad rate and swap set by cutoff
`10_SQL_WoE`	`obwoe_sql` output: the WoE transform
`11_SQL_Points`	the same bins returning integer points, summed into the score
`12_Reproducibility`	the call, versions, and every warning raised

Splitting the deployment SQL in two is deliberate. The WoE sheet reproduces the model exactly and is what a data scientist re-scores with; the points sheet reproduces the *card*, which is what the business signed and what the decision engine should run.

Value

The path written, invisibly.

See Also

`obwoe_scorecard`, `obwoe_sql`

Examples

```
german <- read.csv(
  gzfile(system.file("extdata", "germancredit.csv.gz",
    package = "OptimalBinningWoE"
  )),
  stringsAsFactors = FALSE
)
german$default <- 1L - german$credit_risk
german$credit_risk <- NULL

sc <- obwoe_scorecard(german, target = "default", seed = 1)
obwoe_report(sc, file = file.path(tempdir(), "scorecard.xlsx"))
```

`obwoe_scale` *Points-to-Double-the-Odds Score Scaling*

Description

Computes the two constants that turn a model's log-odds into scorecard points: the *factor* that fixes how many points double the odds, and the *offset* that anchors a reference score to a reference odds.

Usage

```
obwoe_scale(
  pdo = 20,
  score_ref = 600,
  odds_ref = 50,
  direction = c("higher_is_safer", "higher_is_riskier")
)
```

Arguments

`pdo`	Numeric. Points to Double the Odds. Default `20`.
`score_ref`	Numeric. The reference score. Default `600`.
`odds_ref`	Numeric. The good-to-bad odds at `score_ref`, expressed as a ratio (`50` means 50 non-events per event). Default `50`.
`direction`	Character string fixing which end of the scale is safe: `"higher_is_safer"` (default, the scorecard convention) or `"higher_is_riskier"`.

Details

Scorecards are scaled on the **good-to-bad** odds:

$$\text{Score} = \text{Offset} + \text{Factor}\cdot\ln(\text{odds}_{good:bad}), \qquad \text{Factor} = \frac{\text{PDO}}{\ln 2}, \qquad \text{Offset} = \text{Score}_0 - \text{Factor}\cdot\ln(\text{Odds}_0)$$

Every model in this package predicts the log-odds of the **event** (the bad), which is the other direction, so the deployed form is

$$\text{Score} = \text{Offset} - \text{Factor}\cdot\eta$$

The minus sign is the whole point: a bin with a high Weight of Evidence is a risky bin, and a risky bin must score *fewer* points.

Two identities pin the scaling completely, and `obwoe_score` is tested against both:

- **Anchor.** A case whose odds are `odds_ref` scores exactly `score_ref`.
- **Scale.** Doubling the good-to-bad odds adds exactly `pdo` points, at every point of the scale.

Value

An object of class "obwoe_scaling": a list with pdo, score_ref, odds_ref, direction, factor and offset.

References

Siddiqi, N. (2006). Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring. *John Wiley & Sons*. doi:10.1002/9781119201731

See Also

obwoe_score to apply the scaling, obwoe_scorecard for the full pipeline.

Examples

```
scaling <- obwoe_scale(pdo = 20, score_ref = 600, odds_ref = 50)
scaling

# a case at the reference odds scores the reference score
obwoe_score(-log(50), scaling)

# halving the odds of being good costs exactly one PDO
obwoe_score(-log(25), scaling)
```

---

obwoe_score *Turn Log-Odds into Scorecard Points*

Description

Applies a obwoe_scale scaling to the linear predictor of a model that predicts the log-odds of the event.

Usage

```
obwoe_score(link, scaling, round = FALSE)
```

Arguments

link	Numeric vector of log-odds of the event, as returned by predict(fit, type = "link").
scaling	An "obwoe_scaling" object from obwoe_scale.
round	Logical. Round to whole points? Default FALSE; the unrounded score is what the additivity identity holds for.

Value

Numeric vector of scores, of the same length as link.

See Also

obwoe_scale

Examples

```
scaling <- obwoe_scale()
obwoe_score(c(-log(50), -log(100), 0), scaling)
```

---

obwoe_scorecard *Automated Scorecard Pipeline*

Description

Runs a credit or fraud scorecard end to end in one call: optimal binning, variable screening by Information Value and bin ordering, redundancy pruning in the Weight of Evidence space, model fitting, and score scaling to points. Optionally writes the whole thing to a multi-sheet `.xlsx` workbook that a model committee, a validation team and a deployment engineer can each read.

Usage

```
obwoe_scorecard(
  data,
  target,
  feature = NULL,
  exclude = NULL,
  split = 0.7,
  validation = NULL,
  file = NULL,
  binning = list(),
  screening = list(),
  engine = "glm",
  engine_args = list(),
  control = control.obwoe_scorecard(),
  seed = NULL,
  verbose = FALSE
)
```

Arguments

`data`	A `data.frame` holding the development sample.
`target`	Character string naming the target column. Binary; a factor's **second** level is the event, and which level that was is recorded in the result.
`feature`	Character vector of candidate predictors. `NULL` (default) uses every column except the target, the split column and anything named in `exclude`.
`exclude`	Character vector of columns to keep out of the model — identifiers, dates, and above all fields populated after the outcome was known. Default `NULL`.

`split` How the development sample is divided. One of:

- **a number in** $(0, 1)$ proportion of rows used for training, sampled at random with the event rate preserved (default `0.7`);
- **a column name** an out-of-time split: the earliest value of that column trains, the rest validates;
- **an integer or logical vector** the training rows, given explicitly;
- `NULL` no split — everything trains, and every reported metric is in-sample. Flagged as such throughout.

`validation` Optional named list of extra `data.frames` to score and report on. They never influence any fitted quantity.

`file` Optional path to a `.xlsx` file. `NULL` (default) computes everything and writes nothing; pass the result to `obwoe_report` later to produce the workbook.

`binning` Named list of arguments for `obwoe` — typically `min_bins`, `max_bins`, `algorithm`, `control`.

`screening` Named list of arguments for `obwoe_select` — typically `iv_min`, `iv_max`, `require_monotonic`, `min_bin_pct`, `top_n`.

`engine` Character string naming the model engine — `"glm"` (default), `"obwoe"` (the package's own C++ logistic regression) or `"glmnet"` — or a list supplying `fit`, `link` and `coef` of your own. See Details.

`engine_args` Named list forwarded to the engine's fitting call.

`control` An `"obwoe_scorecard_control"` object from `control.obwoe_scorecard`.

`seed` Optional integer seed, for a reproducible split.

`verbose` Logical. Report progress? Default `FALSE`.

Details

The split comes first:

Binning is **supervised**: the cut points and the WoE are both chosen using the target. Fitting them on rows that are later used to measure performance is leakage, and it is the failure this function is built to make impossible — the split happens before anything is fitted, the binning sees the training rows only, and every other sample is transformed through `obwoe_apply`. Screening statistics are computed on the training rows for the same reason.

The binning is fitted once:

One `"obwoe"` object drives screening, correlation, the model, the points, the stability report and the generated SQL. Refitting the binning for the model — with, say, a different `bin_cutoff` — produces a workbook whose evidence describes a transform the deployed model never saw, and every number in it looks internally consistent.

Points:

With k variables in the model, an intercept α and slopes β_j,

$$\text{points}_{ij} = \frac{\text{Offset}}{k} - \text{Factor}\left(\beta_j \,\text{WoE}_{ij} + \frac{\alpha}{k}\right)$$

so the points of the bins an applicant falls into sum to their score. The table is **fixed**: each bin rounds to one integer, the same for everyone, because a bin whose value depended on who was

being scored would not be a scorecard. Rounding k terms moves the total by at most $k/2$ points against the unrounded model score; that drift is measured and reported rather than hidden, and the card — the sum of the integers — is the deployed definition.

Engines:

An engine is three functions: `fit(x, y, args)`, `link(object, x)` returning the log-odds of the event, and `coef(object)` returning the additive coefficients **or** `NULL`. Returning `NULL` declares the model non-additive: gains, stability and score bands are still produced, but no points table is, because a model that is not a sum of per-variable terms cannot be decomposed into one. Supplying `engine` as a list of those three functions is how a gradient-boosted or **tidymodels** fit is plugged in without the package depending on either.

What it refuses to do:

The pipeline stops, rather than producing a plausible workbook, when the shortlist is empty, when the fit does not converge, or when the requested engine's package is missing. It drops constant predictors before fitting and, by default, refits without any variable whose WoE coefficient comes out negative. Everything it does silently absorb is listed in `$warnings` and written to the workbook.

Value

An object of class `"obwoe_scorecard"`: a list with

`call`, `version`, `built_on`, `seed` provenance

`target`, `event_level`, `candidates`, `selected`, `final` the funnel, as character vectors

`binning` the `"obwoe"` object, fitted on the training rows only

`screening`, `screening_bins` `obwoe_select` at both levels of detail, plus a `stage` column naming the step that rejected each variable: `"in_model"`, `"sign_rejected"` (negative WoE coefficient), `"corr_pruned"`, `"constant_woe"` (one WoE value after the transform) or `"screened_out"` (failed the IV or ordering rules)

`correlation` the `obwoe_prune` result

`control` the resolved `"obwoe_scorecard_control"` this object was actually built with (as of 1.13.1) – `predict()` and `obwoe_report` read `na_woe` from here so every output agrees on the fallback for an unseen category or an unmodeled missing value

`engine` name, requested versus used, and whether additive

`model` the raw fitted object — engine-specific and outside the supported interface

`coefficients`, `diagnostics` the additive fit and its sign, separation and convergence checks

`scaling` the `obwoe_scale` constants

`points` the fixed points table, one row per variable and bin

`samples` per sample: scores, gains table, headline metrics

`stability` PSI for the score and for every variable in the model

`warnings` everything the pipeline wants a human to read

References

Siddiqi, N. (2006). Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring. *John Wiley & Sons*. doi:10.1002/9781119201731

Thomas, L. C., Edelman, D. B., & Crook, J. N. (2002). Credit Scoring and Its Applications. *SIAM*. doi:10.1137/1.9780898718317

Anderson, R. (2007). The Credit Scoring Toolkit. *Oxford University Press*.

See Also

`obwoe_report` to write the workbook, `predict.obwoe_scorecard` to score new data, `obwoe_select` for the screening rules, `obwoe_sql` for the deployment SQL, `obwoe_scale` and `obwoe_psi`.

Examples

```
german <- read.csv(
  gzfile(system.file("extdata", "germancredit.csv.gz",
    package = "OptimalBinningWoE"
  )),
  stringsAsFactors = FALSE
)
german$default <- 1L - german$credit_risk
german$credit_risk <- NULL

sc <- obwoe_scorecard(german, target = "default", seed = 42)
sc

head(sc$points[, c("variable", "bin", "woe", "points")])
sc$samples$train$metrics

# write the workbook
# obwoe_report(sc, file = "scorecard.xlsx")
```

`obwoe_select` *Automated Variable Selection for Weight of Evidence Scorecards*

Description

Screens every binned feature of an `"obwoe"` model against the two criteria that govern variable admission in credit risk practice: **predictive strength**, measured by the Information Value and graded with the Siddiqi (2006) bands, and **guaranteed rank ordering**, measured by monotonicity of the bin event rate. The function never drops rows: a base with 500 candidate variables yields 500 rows, each carrying a selection flag and the exact reason behind the decision, so the analyst can override the automatic verdict at will.

Usage

```
obwoe_select(
  obj,
  detail = c("summary", "full"),
  iv_min = 0.02,
  iv_max = 0.5,
  require_monotonic = c("numeric", "all", "none"),
  monotonicity = c("weak", "strict"),
  min_bins = 2L,
  max_bins = Inf,
  min_bin_pct = 0,
  allow_degenerate = FALSE,
  top_n = NULL,
  sort_by = c("iv", "ks", "gini", "auc", "feature", "none"),
  decreasing = TRUE,
  bin_separator = "%;%"
)
```

Arguments

`obj`	An object of class `"obwoe"` returned by obwoe. The target must be binary; multinomial models are reported as unsupported rather than silently mis-scored.
`detail`	Character string controlling the granularity of the result: `"summary"` One row per feature with its headline metrics and the selection verdict (default). `"full"` One row per feature *and* optimised bin, carrying the complete gains table of every variable alongside the feature-level verdict.
`iv_min`	Numeric. Minimum admissible total Information Value (inclusive). Default `0.02`, the *Unpredictive*/*Weak* boundary.
`iv_max`	Numeric. Exclusive upper bound on the total Information Value. Default `0.50`, the *Suspicious* threshold: variables at or above it are rejected because such strength usually signals target leakage rather than genuine signal. Use `Inf` to disable.
`require_monotonic`	Character string selecting where the ordering constraint applies: `"numeric"` Only numerical features must be monotonic (default). Nominal categories carry no intrinsic order, so their bins can always be sorted by WoE and are trivially rank-orderable. `"all"` Categorical features must also be monotonic in the bin order returned by the algorithm. `"none"` The ordering constraint is reported but not enforced.
`monotonicity`	Character string. `"weak"` (default) accepts non-decreasing or non-increasing profiles; `"strict"` rejects ties between adjacent bins.
`min_bins`	Integer. Minimum number of bins a feature must have to be selected. Default `2`.
`max_bins`	Numeric. Maximum admissible number of bins. Default `Inf`.

`min_bin_pct` Numeric in $[0, 1)$. Minimum population share of the smallest bin. Default `0` (no constraint); `0.05` is the usual scorecard convention.

`allow_degenerate` Logical. If `FALSE` (default), features with a bin containing no events or no non-events are rejected, since their WoE is not finite and the bin cannot be scored reliably out of sample.

`top_n` Integer or `NULL` (default). When supplied, only the `top_n` best features among those passing every rule stay selected; the remainder are flagged `"NOT_IN_TOP_N"`. Ranking follows `sort_by`, falling back to alphabetical order when `sort_by` names no metric.

`sort_by` Character string giving the ordering of the output and the ranking used by `top_n`: `"iv"` (default), `"ks"`, `"gini"`, `"auc"`, `"feature"` or `"none"`.

`decreasing` Logical. Sort the output in decreasing order? Default `TRUE`; ignored when `sort_by` is `"feature"` or `"none"`. It affects presentation only: `rank` and the `top_n` cut always keep the highest values of the ranking metric.

`bin_separator` Character string separating merged categories inside a categorical bin label. Default `"%;%"`, matching `control.obwoe`.

Details

Predictive strength:

The Information Value of a binned variable aggregates the divergence between the event and non-event distributions:

$$IV = \sum_{i=1}^{k} \left(\frac{n_{1,i}}{N_1} - \frac{n_{0,i}}{N_0} \right) \ln\left(\frac{n_{1,i}/N_1}{n_{0,i}/N_0} \right)$$

where $n_{1,i}$ and $n_{0,i}$ are the event and non-event counts of bin i. This is the symmetrised Kullback-Leibler divergence (Jeffreys divergence) between the two conditional distributions. Counts are taken from the fitted binning, so the value is invariant to any smoothing an individual algorithm may apply to its reported WoE.

Bands follow Siddiqi (2006): $IV < 0.02$ unpredictive, $[0.02, 0.10)$ weak, $[0.10, 0.30)$ medium, $[0.30, 0.50)$ strong, ≥ 0.50 suspicious. The upper band is excluded by default: an IV above 0.5 on a single variable is, in practice, far more often a symptom of leakage than of a genuinely dominant predictor.

Guaranteed ordering:

A variable is rank-ordering when its event rate moves in one direction across the bin sequence. Writing the bin odds as $\theta_i = n_{1,i}/n_{0,i}$, both quantities used in scorecards are strictly increasing transformations of θ_i:

$$\pi_i = \frac{n_{1,i}}{n_{1,i} + n_{0,i}} = \frac{\theta_i}{1 + \theta_i}, \qquad WoE_i = \ln \theta_i + \ln \frac{N_0}{N_1}$$

so monotonicity of the event rate π_i and of the empirical WoE are the same statement. The check is therefore performed once, on π_i, which stays finite even when a bin holds a single class; the ordering of the WoE actually applied by the algorithm is reported separately in `woe_monotonic`.

For numerical variables the bin sequence is intrinsic (ordered intervals), making monotonicity a genuine constraint. For nominal categorical variables the sequence is a free relabelling, so a non-monotonic profile can always be repaired by sorting; the default `require_monotonic = "numeric"` reflects this asymmetry.

Discrimination metrics:

`ks`, `auc` and `gini` describe the *deployed* score, that is the WoE the algorithm attaches to each bin — exactly what `obwoe_apply` writes out and what a downstream logistic regression consumes. Bins are ranked by that WoE and merged when tied, then

$$KS = \max_i |F_1(i) - F_0(i)|, \qquad AUC = \frac{1}{N_1 N_0} \sum_i n_{1,i} \left(\sum_{j:s_j<s_i} n_{0,j} + \tfrac{1}{2} n_{0,i} \right), \qquad Gini = 2\,AUC - 1$$

where s_i is the bin WoE and F_1, F_0 are the cumulative event and non-event distributions in that order. The AUC expression is the tie-corrected Mann-Whitney U statistic and reproduces, to machine precision, both the trapezoidal area under the binned ROC curve and the rank-based AUC computed on the WoE-transformed observations. Ranking by WoE rather than by bin id makes `ks` the true KS of the transformed variable even when the binning is not monotonic.

Excellence tiers:

`quality` summarises how cleanly the algorithm categorised a variable:

Tier	Definition
`Excellent`	Selected, strictly monotonic, $IV \geq 0.10$, at least 3 bins, no degenerate bin, smallest bin $\geq 5\%$
`Good`	Selected, monotonic, $IV \geq 0.10$
`Fair`	Selected under every active rule but failing one of the structural refinements above
`Rejected`	Not selected

Reason codes:

`reason` concatenates every violated rule with `";"`, so a variable rejected on two counts reports both:

`OK`	Passed every active rule
`BINNING_ERROR`	The algorithm failed on this variable
`IV_BELOW_MIN`	$IV <$ `iv_min`
`IV_SUSPICIOUS`	$IV \geq$ `iv_max` (leakage risk)
`NOT_MONOTONIC`	Event rate not ordered across bins
`TOO_FEW_BINS`	Fewer bins than `min_bins`
`TOO_MANY_BINS`	More bins than `max_bins`
`SMALL_BIN`	Smallest bin below `min_bin_pct`
`DEGENERATE_BIN`	A bin holds no events or no non-events
`NOT_IN_TOP_N`	Passed the rules but ranked below `top_n`

Value

A `data.table` when the **data.table** package is installed and a `data.frame` otherwise (`data.table` inherits from `data.frame`, so either object supports the usual accessors).

With `detail = "summary"` the table holds one row per feature:

Column	Meaning
`feature`	Variable name
`type`	`"numerical"` or `"categorical"`
`algorithm`	Binning algorithm used
`n_bins`	Number of optimised bins
`n_obs`, `n_pos`, `n_neg`	Population and class counts
`event_rate`	Overall event rate
`total_iv`	Total Information Value
`iv_class`	Siddiqi strength band
`ks`	Kolmogorov-Smirnov statistic in $[0, 1]$
`gini`, `auc`	Discrimination of the binned score
`max_lift`	Largest bin lift over the base rate
`min_bin_pct`, `min_bin_count`	Smallest bin size
`n_degenerate_bins`	Bins with zero events or zero non-events
`woe_min`, `woe_max`, `woe_range`	Spread of the WoE profile
`monotonic`, `monotonic_strict`	Event-rate ordering flags
`monotonic_direction`	`"increasing"`, `"decreasing"`, `"constant"` or `"non-monotonic"`
`n_violations`	Adjacent pairs breaking the dominant trend
`spearman`	Rank correlation between bin order and event rate
`woe_monotonic`	Ordering of the WoE actually applied
`converged`, `iterations`	Optimiser diagnostics
`error`, `error_msg`	Binning failure flag and message
`quality`	Excellence tier (see Details)
`selected`	**Selection flag**
`reason`	**Machine-readable reason codes**
`reason_desc`	**Human-readable justification**
`rank`	Position under `sort_by` among selectable features

With `detail = "full"` the table holds one row per feature *and* optimised bin. Each row carries the complete gains table produced by `obwoe_gains_score` (`count`, `pos`, `neg`, `woe`, `iv`, `total_iv`, `pos_rate`, `lift`, `cum_pos_perc`, `precision`, `recall`, `f1_score`, `kl_divergence`, `js_divergence`, and so on) plus:

`bin_id`, `bin`	Bin position and label
`bin_lower`, `bin_upper`	Interval bounds of a numerical bin, on the half-open convention $(lower, upper]$
`n_categories`, `categories`	Categories merged into a categorical bin
`woe_model`	WoE as reported by the binning algorithm, i.e. the value actually applied by `obwoe_apply`
`ks_bin`	Cumulative KS *at* this bin (the gains-table `ks`), renamed so that `ks` keeps its feature-level mean

The feature-level verdict (`iv_class`, `ks`, `gini`, `auc`, the monotonicity flags, `quality`, `selected`, `reason`, `reason_desc`, `rank`) is repeated on every bin row. Variables whose binning failed have no bins and appear as a single placeholder row, so the row count never hides a candidate variable.

References

Siddiqi, N. (2006). Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring. *John Wiley & Sons*. doi:10.1002/9781119201731

Thomas, L. C., Edelman, D. B., & Crook, J. N. (2002). Credit Scoring and Its Applications. *SIAM Monographs on Mathematical Modeling and Computation*. doi:10.1137/1.9780898718317

Hand, D. J., & Till, R. J. (2001). A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems. *Machine Learning*, 45(2), 171-186. doi:10.1023/A:1010920819831

Kullback, S., & Leibler, R. A. (1951). On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1), 79-86. doi:10.1214/aoms/1177729694

See Also

`obwoe` for fitting the binning, `obwoe_sql` for exporting the selected variables as SQL, `obwoe_gains_score` for the underlying bin-level engine, `summary.obwoe` for the compact model overview.

Examples

```
set.seed(42)
n <- 2000
df <- data.frame(
  age = rnorm(n, 40, 12),
  income = exp(rnorm(n, 10, 0.7)),
  noise = rnorm(n),
  region = sample(c("N", "S", "E", "W"), n, replace = TRUE)
)
df$target <- rbinom(n, 1, plogis(-1.2 + 0.05 * (df$age - 40)))

model <- obwoe(df, target = "target", max_bins = 6)

# One row per variable, every candidate kept
sel <- obwoe_select(model)
sel[, c("feature", "total_iv", "iv_class", "ks", "monotonic", "selected", "reason")]

# Full gains detail for every optimised bin
det <- obwoe_select(model, detail = "full")
head(det[, c("feature", "bin", "count", "pos_rate", "woe", "iv", "selected")])

# Stricter policy: monotonic everywhere, 5% minimum bin, top 10 by KS
strict <- obwoe_select(model,
  require_monotonic = "all", monotonicity = "strict",
  min_bin_pct = 0.05, top_n = 10, sort_by = "ks"
)
```

---

obwoe_sql *Generate SQL for a Fitted Optimal Binning*

Description

Translates a fitted binning into executable SQL so the Weight of Evidence transformation can be applied directly inside a database, with no round trip through R. Every optimised bin becomes one `WHEN` branch of a `CASE` expression, reproducing the interval and category assignments of `obwoe_apply` exactly.

Usage

```
obwoe_sql(
  obj,
  table = "your_table",
  features = NULL,
  output = c("woe", "bin", "index", "both"),
  style = c("select", "case", "cte", "view"),
 dialect = c("ansi", "postgres", "mysql", "mariadb", "sqlserver", "oracle", "spark",
  "hive", "databricks", "bigquery", "snowflake", "redshift", "duckdb", "sqlite"),
  view_name = "woe_transform",
  keep_columns = NULL,
  suffix_woe = "_woe",
  suffix_bin = "_bin",
  na_value = 0,
  null_to_na_bin = TRUE,
  na_categories = c("NA", "Missing", ""),
  explicit_bounds = TRUE,
  digits = NULL,
  quote_identifiers = c("auto", "always", "never"),
  indent = 4L,
  comment = TRUE,
  class_index = NULL,
  bin_separator = "%;%",
  trim_categories = FALSE,
  file = NULL
)
```

Arguments

`obj`	An object of class `"obwoe"` from `obwoe`, a prepped `step_obwoe` step, or a prepped **recipes** recipe containing one.
`table`	Character string naming the source table. May be qualified (`"schema.table"`); each part is quoted independently. Required for every `style` except `"case"`.
`features`	Character vector restricting which variables are exported. `NULL` (default) exports every successfully binned variable. This is the natural place to feed the output of `obwoe_select`, e.g. `features = sel$feature[sel$selected]`.
`output`	Character string choosing what the `CASE` returns: `"woe"` The Weight of Evidence value (default). `"bin"` The bin label as a string literal. `"index"` The 1-based bin id.

"both" Two expressions per feature: bin label and WoE.

style Character string choosing how the expressions are assembled:

"select" A complete SELECT ... FROM table (default).

"case" A named character vector of bare CASE expressions, one per generated column, with no aliases.

"cte" The SELECT wrapped in a WITH ... AS (...) common table expression.

"view" A CREATE VIEW statement built on the SELECT.

dialect Character string naming the target SQL dialect. One of "ansi" (default), "postgres", "mysql", "mariadb", "sqlserver", "oracle", "spark", "hive", "databricks", "bigquery", "snowflake", "redshift", "duckdb", "sqlite".

view_name Character string naming the view or CTE. Defaults to "woe_transform".

keep_columns Character vector of extra columns to carry through unchanged (identifiers, the target, partition keys). Default NULL.

suffix_woe, suffix_bin
Character strings appended to the feature name to build the output column aliases. Defaults "_woe" and "_bin".

na_value Numeric value returned for NULL inputs and for categories unseen during training. Default 0, matching the na_woe default of obwoe_apply.

null_to_na_bin Logical. When TRUE (default) and the training data contained missing values that the binner folded into a bin under one of na_categories, NULL inputs are routed to that bin's value instead of na_value. obwoe_apply applies the same rule to NA for categorical features as of 1.13.1, so the two stay in agreement; keep this TRUE unless you have a specific reason to diverge from the R-side behavior.

na_categories Character vector of tokens the binner uses to represent missing categories. Default c("NA", "Missing", ""), matching ob_apply_woe_cat.

explicit_bounds
Logical. When TRUE (default) each numerical branch states both of its bounds, e.g. WHEN x > 7 AND x <= 10 THEN ..., so every branch is correct in isolation and survives reordering or copy-and-paste. When FALSE the branches cascade with upper bounds only, which is shorter and equally exact given the top-down evaluation order of CASE.

digits Integer or NULL (default). NULL writes cut points and WoE values at full precision, as a decimal string that parses back to the identical double. Supplying a value rounds the literals to that many decimal places for readability, at the cost of exactness on bin boundaries.

quote_identifiers
Character string: "auto" (default) quotes only identifiers that need it, "always" quotes everything, "never" emits bare names.

indent Integer giving the number of spaces prefixed to each WHEN line. Default 4.

comment Logical. Prepend an audit header with the package version, the algorithm and the per-variable Information Value? Default TRUE.

class_index Integer or NULL (default). For multinomial models the WoE is a matrix; this selects which class column to export.

`bin_separator` Character string separating merged categories inside a bin label. Default `"%;%"`, matching `control.obwoe`.

`trim_categories`
Logical. When `FALSE` (default) category names are matched byte for byte, including any leading or trailing whitespace they carry, which is what `obwoe_apply` does in R. Set it to `TRUE` only when the database column holds the unpadded form of a category that reached R padded — a `CHAR(n)` column read through a driver that strips the padding, for instance. Trimming otherwise makes a genuinely padded category unmatchable.

`file` Optional path. When supplied, the generated SQL is also written there with `writeLines`.

Details

Interval semantics:

Numerical bins are half-open on the right, the convention used throughout the package. For cut points $c_1 < c_2 < \cdots < c_k$ the $k + 1$ bins are

$$(-\infty, c_1],\ (c_1, c_2],\ \ldots,\ (c_{k-1}, c_k],\ (c_k, +\infty)$$

which the generated SQL renders as `x <= c1`, `x > c1 AND x <= c2`, and so on. This mirrors `cut(x, breaks = c(-Inf, cutpoints, Inf), right = TRUE)`, the exact call made by `obwoe_apply`, so an observation sitting on a cut point lands in the *lower* bin in R and in SQL alike.

Cut points are read from the fitted `cutpoints` vector, never parsed back from bin labels: label formatting varies between algorithms whereas `cutpoints` is the authoritative numeric boundary. Duplicated cut points are removed exactly as `obwoe_apply` removes them, and a variable whose de-duplicated cut points no longer match its bin count is skipped with a warning rather than exported with a wrong mapping.

Numeric literals:

With `digits = NULL` each cut point is written with the seventeen significant digits that identify an IEEE 754 double uniquely, trailing zeros removed, so a value that is exact in binary still reads short. Rounding a boundary such as `4049.5` to fewer digits would silently move observations between bins, so exactness is the default; scientific notation is avoided because dialects differ on how they type such literals.

NULL handling:

In SQL, `NULL <= 5` evaluates to `NULL`, not to `FALSE`, so a missing value matches no comparison branch and would silently fall through to `ELSE`. Every generated expression therefore opens with an explicit `WHEN <col> IS NULL` branch. Categorical binners represent training missings as the literal category `"NA"`; when `null_to_na_bin = TRUE` the `IS NULL` branch returns that bin's value, so database `NULL`s are scored the way missings were scored during fitting. The `ELSE` branch catches categories never seen in training and returns `na_value`.

Escaping:

Bin labels are the original category strings joined by `bin_separator`, so splitting a label recovers the categories byte for byte. They are matched exactly, whitespace included, unless `trim_categories = TRUE`.

Category labels are emitted as quoted literals with embedded single quotes doubled per ANSI SQL. On MySQL, MariaDB and the Hive family — where a backslash also escapes inside string literals under default settings — backslashes are doubled too. Identifiers are quoted with the dialect's own delimiters, and by default only when the name is not a plain `[A-Za-z_][A-Za-z0-9_]*` token or collides with a reserved word; this keeps generated code readable on case-folding engines such as Oracle and Snowflake, where blanket quoting would force case sensitivity.

Value

An object of class `"obwoe_sql"`, which is a character vector with a `print` method that echoes the statement verbatim. For `style = "case"` the vector is named by output column and holds one `CASE` expression per element; for every other style it is a single string containing the complete statement. Use `as.character()` to strip the class, or `cat()` to display it.

References

Siddiqi, N. (2006). Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring. *John Wiley & Sons*. doi:10.1002/9781119201731

International Organization for Standardization (2016). *ISO/IEC 9075-2:2016 Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation)*.

See Also

`obwoe` for fitting the binning, `obwoe_select` for choosing which variables to export, `obwoe_apply` for the equivalent transformation in R.

Examples

```
set.seed(42)
n <- 1000
df <- data.frame(
  age = rnorm(n, 40, 12),
  region = sample(c("North", "South", "O'Hare"), n, replace = TRUE)
)
df$target <- rbinom(n, 1, plogis(-1 + 0.04 * (df$age - 40)))

model <- obwoe(df, target = "target", max_bins = 5)

# Complete SELECT statement
sql <- obwoe_sql(model, table = "credit.applications", keep_columns = "target")
sql

# Only the variables that survived automatic screening
sel <- obwoe_select(model)
obwoe_sql(model,
  table = "credit.applications",
  features = sel$feature[sel$selected]
)

# Bin labels and WoE side by side, as a Spark view
obwoe_sql(model,
```

```
  table = "credit.applications", output = "both",
  style = "view", dialect = "spark", view_name = "v_woe"
)

# Bare CASE expressions for embedding in an existing query
obwoe_sql(model, style = "case")
```

---

ob_apply_woe_cat *Apply Optimal Weight of Evidence (WoE) to a Categorical Feature*

Description

Transforms a categorical feature into its corresponding Weight of Evidence (WoE) values using pre-computed binning results from an optimal binning algorithm (e.g., ob_categorical_cm).

Usage

```
ob_apply_woe_cat(
  obresults,
  feature,
  bin_separator = "%;%",
  missing_values = c("NA", "Missing", "")
)
```

Arguments

obresults	List output from an optimal binning function for categorical variables. Must contain elements bin (character vector of bin labels) and woe (numeric vector of WoE values). Bins may represent individual categories or merged groups separated by bin_separator.
feature	Character or factor vector of categorical values to be transformed. Automatically coerced to character if provided as factor.
bin_separator	Character string used to separate multiple categories within a single bin label (default: "%;%"). For example, a bin "A%;%B%;%C" contains categories A, B, and C.
missing_values	Character vector specifying which values should be treated as missing (default: c("NA", "Missing", "")). These values are matched against a special bin labeled "NA" or "Missing" in obresults.

Details

This function is typically used in a two-step workflow:

1. Train binning on training data: bins <- ob_categorical_cm(feature_train, target_train)
2. Apply WoE to new data: woe_test <- ob_apply_woe_cat(bins, feature_test)

The function performs exact string matching between categories in `feature` and the bin labels in `obresults$bin`. For merged bins (containing `bin_separator`), the string is split and each component is matched individually.

Value

Numeric vector of WoE values with the same length as `feature`. Categories not found in `obresults` will produce `NA` values with a warning.

Examples

```
# Mock data
train_data <- data.frame(
  category = c("A", "B", "A", "C", "B", "A"),
  default = c(0, 1, 0, 1, 0, 0)
)
test_data <- data.frame(
  category = c("A", "C", "B")
)

# Train binning on training set
train_bins <- ob_categorical_cm(
  feature = train_data$category,
  target = train_data$default
)

# Apply to test set
test_woe <- ob_apply_woe_cat(
  obresults = train_bins,
  feature = test_data$category
)

# Handle custom missing indicators
test_woe <- ob_apply_woe_cat(
  obresults = train_bins,
  feature = test_data$category,
  missing_values = c("NA", "Unknown", "N/A", "")
)
```

---

ob_apply_woe_num *Apply Optimal Weight of Evidence (WoE) to a Numerical Feature*

Description

Transforms a numerical feature into its corresponding Weight of Evidence (WoE) values using pre-computed binning results from an optimal binning algorithm (e.g., `ob_numerical_mdlp`, `ob_numerical_mob`).

Usage

```
ob_apply_woe_num(
  obresults,
  feature,
  include_upper_bound = TRUE,
  missing_values = c(-999)
)
```

Arguments

obresults	List output from an optimal binning function for numerical variables. Must contain elements cutpoints (numeric vector of bin boundaries) and woe (numeric vector of WoE values). The number of WoE values should equal length(cutpoints) + 1.
feature	Numeric vector of values to be transformed. Automatically coerced to numeric if provided in another type.
include_upper_bound	Logical flag controlling interval boundary behavior (default: TRUE): • TRUE: Intervals are (lower, upper] (right-closed). • FALSE: Intervals are [lower, upper) (left-closed). This must match the convention used during binning.
missing_values	Numeric vector of values to be treated as missing (default: c(-999)). These values are assigned the WoE of the special missing bin if it exists in obresults, or NA otherwise.

Details

This function is typically used in a two-step workflow:

1. Train binning on training data: bins <- ob_numerical_mdlp(feature_train, target_train)
2. Apply WoE to new data: woe_test <- ob_apply_woe_num(bins, feature_test)

Bin Assignment Logic: For k cutpoints $c_1 < c_2 < \cdots < c_k$, values are assigned as:

- Bin 1: $x \leq c_1$ (if include_upper_bound = TRUE)
- Bin i: $c_{i-1} < x \leq c_i$ for $i = 2, \ldots, k$
- Bin k+1: $x > c_k$

Handling of Edge Cases:

- Values in missing_values are matched against a bin labeled "NA" or "Missing" in obresults$bin (if available).
- Inf and -Inf are assigned to the last and first bins, respectively.
- Values exactly equal to cutpoints follow the include_upper_bound convention.

Value

Numeric vector of WoE values with the same length as `feature`. Values outside the range of `cutpoints` are assigned to the first or last bin. `NA` values in `feature` are propagated to the output unless explicitly listed in `missing_values`.

See Also

`ob_numerical_mdlp` for MDLP binning, `ob_numerical_mob` for monotonic binning, `ob_apply_woe_cat` for applying WoE to categorical features.

Examples

```
# Mock data
train_data <- data.frame(
  income = c(50000, 75000, 30000, 45000, 80000, 60000),
  default = c(0, 0, 1, 1, 0, 0)
)
test_data <- data.frame(
  income = c(55000, 35000, 90000)
)

# Train binning on training set
train_bins <- ob_numerical_mdlp(
  feature = train_data$income,
  target = train_data$default
)

# Apply to test set
test_woe <- ob_apply_woe_num(
  obresults = train_bins,
  feature = test_data$income
)

# Handle custom missing indicators (e.g., -999, -1)
test_woe <- ob_apply_woe_num(
  obresults = train_bins,
  feature = test_data$income,
  missing_values = c(-999, -1, -9999)
)

# Use left-closed intervals (match scikit-learn convention)
test_woe <- ob_apply_woe_num(
  obresults = train_bins,
  feature = test_data$income,
  include_upper_bound = FALSE
)
```

`ob_categorical_cm` *Optimal Binning for Categorical Variables using Enhanced ChiMerge Algorithm*

Description

Performs supervised discretization of categorical variables using an enhanced implementation of the ChiMerge algorithm (Kerber, 1992) with optional Chi2 extension (Liu & Setiono, 1995). This method optimally groups categorical levels based on their relationship with a binary target variable to maximize predictive power while maintaining statistical significance.

Usage

```
ob_categorical_cm(
  feature,
  target,
  min_bins = 3,
  max_bins = 5,
  bin_cutoff = 0.05,
  max_n_prebins = 20,
  bin_separator = "%;%",
  convergence_threshold = 1e-06,
  max_iterations = 1000,
  chi_merge_threshold = 0.05,
  use_chi2_algorithm = FALSE
)
```

Arguments

`feature`	A character vector or factor representing the categorical predictor variable to be binned.
`target`	An integer vector of binary outcomes (0/1) corresponding to each observation in `feature`.
`min_bins`	Integer. Minimum number of bins to produce. Must be >= 2. Defaults to 3.
`max_bins`	Integer. Maximum number of bins to produce. Must be >= `min_bins`. Defaults to 5.
`bin_cutoff`	Numeric. Threshold for treating categories as rare. Categories with frequency < `bin_cutoff` will be merged with their most similar neighbors. Value must be in (0, 1). Defaults to 0.05.
`max_n_prebins`	Integer. Maximum number of initial pre-bins before merging. Controls computational complexity. Must be >= 2. Defaults to 20.
`bin_separator`	String. Separator used when combining multiple categories into a single bin label. Defaults to "%;%".
`convergence_threshold`	Numeric. Convergence tolerance for iterative merging process. Smaller values require stricter convergence. Must be > 0. Defaults to 1e-6.

`max_iterations` Integer. Maximum iterations for the merging algorithm. Prevents infinite loops. Must be > 0. Defaults to 1000.

`chi_merge_threshold` Numeric. Statistical significance level (p-value) for chi-square tests during merging. Higher values create fewer bins. Value must be in (0, 1). Defaults to 0.05.

`use_chi2_algorithm` Logical. If TRUE, uses the Chi2 variant which performs multi-pass merging with decreasing significance thresholds. Defaults to FALSE.

Details

The algorithm implements two main approaches:

1. Standard ChiMerge: Iteratively merges adjacent bins with lowest chi-square statistics until all remaining pairs are statistically distinguishable at the specified significance level.

2. Chi2 Algorithm (when `use_chi2_algorithm = TRUE`): Performs multiple passes with decreasing significance thresholds ($0.5 \rightarrow 0.001$), creating more robust binning structures particularly for noisy data.

Key features include:

- Rare category handling through pre-merging
- Monotonicity enforcement of Weight of Evidence
- Numerical stability with underflow protection
- Efficient chi-square caching for performance
- Comprehensive input validation and error handling

Information Value interpretation:

- < 0.02: Predictive power not useful
- 0.02-0.1: Weak predictive power
- 0.1-0.3: Medium predictive power
- 0.3-0.5: Strong predictive power
- > 0.5: Suspiciously high (potential overfitting)

Value

A list containing binning results with the following components:

- `id`: Integer vector of bin identifiers (1:n_bins)
- `bin`: Character vector of bin labels (merged category names)
- `woe`: Numeric vector of Weight of Evidence for each bin
- `iv`: Numeric vector of Information Value contribution per bin
- `count`: Integer vector of total observations per bin
- `count_pos`: Integer vector of positive cases per bin
- `count_neg`: Integer vector of negative cases per bin
- `converged`: Logical indicating if algorithm converged

- `iterations`: Integer count of algorithm iterations performed
- `algorithm`: Character string identifying algorithm used
- `warnings`: Character vector of any warnings encountered
- `metadata`: List with additional diagnostic information:
  - `total_iv`: Total Information Value of the binned variable
  - `n_bins`: Final number of bins produced
  - `unique_categories`: Number of unique input categories
  - `total_obs`: Total number of observations processed
  - `execution_time_ms`: Processing time in milliseconds
  - `monotonic`: Direction of WoE monotonicity ("increasing"/"decreasing")

Author(s)

Developed as part of the OptimalBinningWoE package

References

Kerber, R. (1992). ChiMerge: Discretization of numeric attributes. In *Proceedings of the Tenth National Conference on Artificial Intelligence* (pp. 123-128).

Liu, B., & Setiono, R. (1995). Chi2: Feature selection and discretization of numeric attributes. In *Proceedings of the Seventh IEEE International Conference on Tools with Artificial Intelligence* (pp. 372-377).

Examples

```
# Example 1: Basic usage with synthetic data
set.seed(123)
n <- 1000
categories <- c("A", "B", "C", "D", "E", "F", "G", "H")
feature <- sample(categories, n, replace = TRUE, prob = c(
  0.2, 0.15, 0.15,
  0.1, 0.1, 0.1,
  0.1, 0.1
))
# Create target with some association to categories
probs <- c(0.3, 0.4, 0.5, 0.6, 0.7, 0.75, 0.8, 0.85) # increasing probability
target <- sapply(seq_along(feature), function(i) {
  cat_idx <- which(categories == feature[i])
  rbinom(1, 1, probs[cat_idx])
})

result <- ob_categorical_cm(feature, target)
print(result[c("bin", "woe", "iv", "count")])

# View metadata
print(paste("Total IV:", round(result$metadata$total_iv, 3)))
print(paste("Algorithm converged:", result$converged))

# Example 2: Using Chi2 algorithm for more conservative binning
```

```
result_chi2 <- ob_categorical_cm(feature, target,
  use_chi2_algorithm = TRUE,
  max_bins = 6
)

# Compare number of bins
cat("Standard ChiMerge bins:", result$metadata$n_bins, "\n")
cat("Chi2 algorithm bins:", result_chi2$metadata$n_bins, "\n")
```

`ob_categorical_dmiv` *Optimal Binning for Categorical Variables using Divergence Measures*

Description

Performs supervised discretization of categorical variables using a divergence-based hierarchical merging algorithm. This implementation supports multiple information-theoretic and metric divergence measures as described by Zeng (2013), enabling flexible optimization of binning structures for credit scoring and binary classification tasks.

Usage

```
ob_categorical_dmiv(
  feature,
  target,
  min_bins = 3,
  max_bins = 5,
  bin_cutoff = 0.05,
  max_n_prebins = 20,
  bin_separator = "%;%",
  convergence_threshold = 1e-06,
  max_iterations = 1000,
  bin_method = "woe1",
  divergence_method = "l2"
)
```

Arguments

`feature`	A character vector or factor representing the categorical predictor variable to be binned. Missing values are automatically converted to the category `"NA"`.
`target`	An integer vector of binary outcomes (0/1) corresponding to each observation in `feature`. Missing values are not permitted.
`min_bins`	Integer. Minimum number of bins to produce. Must be >= 2. If the final number of bins after merging falls below this threshold, the algorithm will attempt to split bins. Defaults to 3.

`max_bins` Integer. Maximum number of bins to produce. Must be >= `min_bins`. The algorithm performs hierarchical merging until this constraint is satisfied. Defaults to 5.

`bin_cutoff` Numeric. Frequency threshold for rare category handling. Categories with relative frequency below this value are candidates for pre-binning. Must be in (0, 1). Defaults to 0.05.

`max_n_prebins` Integer. Maximum number of initial bins before the main merging phase. When unique categories exceed this limit, rare categories are pre-merged into an "other" bin. Must be >= 2. Defaults to 20.

`bin_separator` Character string used to concatenate category names when multiple categories are merged into a single bin. Defaults to "%;%".

`convergence_threshold`
Numeric. Convergence tolerance for the iterative merging process. When the change in minimum divergence between iterations falls below this threshold, the fit is recorded as converged. It does not stop the merging: `max_bins` is a hard constraint, so merging continues by the same criterion until the bin count meets it. Must be > 0. Defaults to 1e-6.

`max_iterations` Integer. Maximum number of merge operations allowed. Prevents infinite loops in edge cases. Must be > 0. Defaults to 1000.

`bin_method` Character string specifying the Weight of Evidence calculation method. Must be one of:

`"woe"` Traditional WoE: $\ln\left(\frac{p_i/P}{n_i/N}\right)$

`"woe1"` Smoothed WoE (Zeng): $\ln\left(\frac{g_i+0.5}{b_i+0.5}\right)$

The smoothed variant provides numerical stability for sparse bins. Defaults to `"woe1"`.

`divergence_method`
Character string specifying the divergence measure used for determining bin similarity. Must be one of:

`"he"` Hellinger Distance: $\sum(\sqrt{p_i} - \sqrt{n_i})^2$

`"kl"` Symmetrized Kullback-Leibler Divergence

`"klj"` Jeffreys J-Divergence: $(p - n)\ln(p/n)$

`"tr"` Triangular Discrimination: $(p - n)^2/(p + n)$

`"sc"` Symmetric Chi-Square: $(p - n)^2(p + n)/(pn)$

`"js"` Jensen-Shannon Divergence

`"l1"` L1 Metric (Manhattan Distance): $|p - n|$

`"l2"` L2 Metric (Euclidean Distance): $\sqrt{\sum(p - n)^2}$

`"ln"` L-infinity Metric (Chebyshev Distance): $\max|p - n|$

Defaults to `"l2"`.

Details

The algorithm implements a hierarchical agglomerative approach where bins are iteratively merged based on minimum pairwise divergence until the `max_bins` constraint is satisfied or convergence is achieved.

Algorithm Workflow:

1. Input validation and frequency computation
2. Pre-binning of rare categories (if unique categories > `max_n_prebins`)
3. Initialization of pairwise divergence matrix
4. Iterative merging of most similar bin pairs
5. Splitting of heterogeneous bins (if bins < `min_bins`)
6. Final metric computation and WoE-based sorting

Divergence Measure Selection: The choice of divergence measure affects the binning structure:

- Information-theoretic measures (`"kl"`, `"js"`, `"klj"`): Emphasize distributional differences; sensitive to rare events
- Metric measures (`"l1"`, `"l2"`, `"ln"`): Provide geometric interpretation; robust to outliers
- Chi-square family (`"sc"`, `"tr"`): Balance between information content and robustness
- Hellinger distance (`"he"`): Bounded measure; suitable for probability distributions

Pre-binning Strategy: When the number of unique categories exceeds `max_n_prebins`, categories with fewer than 5 observations are aggregated into a special "PREBIN_OTHER" bin to control computational complexity.

Value

A list containing the binning results with the following components:

`id` Integer vector of bin identifiers (1-indexed)

`bin` Character vector of bin labels (merged category names)

`woe` Numeric vector of Weight of Evidence values per bin

`divergence` Numeric vector of divergence contribution per bin

`count` Integer vector of total observations per bin

`count_pos` Integer vector of positive cases (target=1) per bin

`count_neg` Integer vector of negative cases (target=0) per bin

`converged` Logical indicating algorithm convergence

`iterations` Integer count of merge operations performed

`total_divergence` Numeric total divergence of the binning solution

`bin_method` Character string of WoE method used

`divergence_method` Character string of divergence measure used

References

Zeng, G. (2013). Metric Divergence Measures and Information Value in Credit Scoring. *Journal of Mathematics*, 2013, Article ID 848271. doi:10.1155/2013/848271

Kullback, S., & Leibler, R. A. (1951). On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1), 79-86.

Lin, J. (1991). Divergence Measures Based on the Shannon Entropy. *IEEE Transactions on Information Theory*, 37(1), 145-151.

See Also

ob_categorical_cm for ChiMerge-based categorical binning

Examples

```
# Example 1: Basic usage with synthetic credit data
set.seed(42)
n <- 1000

# Simulate occupation categories with varying default rates
occupations <- c(
  "Engineer", "Doctor", "Teacher", "Sales",
  "Manager", "Clerk", "Other"
)
default_probs <- c(0.05, 0.03, 0.08, 0.15, 0.07, 0.12, 0.20)

feature <- sample(occupations, n,
  replace = TRUE,
  prob = c(0.15, 0.10, 0.20, 0.18, 0.12, 0.15, 0.10)
)
target <- sapply(feature, function(x) {
  rbinom(1, 1, default_probs[which(occupations == x)])
})

# Apply optimal binning with L2 divergence
result <- ob_categorical_dmiv(feature, target,
  min_bins = 2,
  max_bins = 4,
  divergence_method = "l2"
)

# Examine binning results
print(data.frame(
  bin = result$bin,
  woe = round(result$woe, 3),
  count = result$count,
  event_rate = round(result$count_pos / result$count, 3)
))

# Example 2: Comparing divergence methods
result_js <- ob_categorical_dmiv(feature, target,
  divergence_method = "js",
  max_bins = 4
)
result_kl <- ob_categorical_dmiv(feature, target,
  divergence_method = "kl",
  max_bins = 4
)

cat("Jensen-Shannon bins:", length(result_js$bin), "\n")
cat("Kullback-Leibler bins:", length(result_kl$bin), "\n")
```

```
# Example 3: High cardinality feature with pre-binning
set.seed(123)
postal_codes <- paste0("ZIP_", sprintf("%03d", 1:50))
feature_high_card <- sample(postal_codes, 2000, replace = TRUE)
target_high_card <- rbinom(2000, 1, 0.1)

result_prebin <- ob_categorical_dmiv(
  feature_high_card,
  target_high_card,
  max_n_prebins = 15,
  max_bins = 5
)

cat("Final bins after pre-binning:", length(result_prebin$bin), "\n")
cat("Algorithm converged:", result_prebin$converged, "\n")
```

`ob_categorical_dp` *Optimal Binning for Categorical Variables using Dynamic Programming*

Description

Performs supervised discretization of categorical variables using a dynamic programming algorithm with optional monotonicity constraints. This method maximizes the total Information Value (IV) while ensuring optimal bin formation that respects user-defined constraints on bin count and frequency. The algorithm guarantees global optimality through dynamic programming.

Usage

```
ob_categorical_dp(
  feature,
  target,
  min_bins = 3,
  max_bins = 5,
  bin_cutoff = 0.05,
  max_n_prebins = 20,
  convergence_threshold = 1e-06,
  max_iterations = 1000,
  bin_separator = "%;%",
  monotonic_trend = "auto"
)
```

Arguments

`feature`	A character vector or factor representing the categorical predictor variable to be binned. Missing values are automatically converted to the category `"NA"`.

`target` An integer vector of binary outcomes (0/1) corresponding to each observation in `feature`. Missing values are not permitted.

`min_bins` Integer. Minimum number of bins to produce. Must be >= 2. The algorithm searches for solutions within [`min_bins`, `max_bins`] that maximize total IV. Defaults to 3.

`max_bins` Integer. Maximum number of bins to produce. Must be >= `min_bins`. Defines the upper bound of the search space for the optimal solution. Defaults to 5.

`bin_cutoff` Numeric. Minimum proportion of total observations required for a category to remain separate. Categories below this threshold are merged with similar categories. Must be in (0, 1). Defaults to 0.05.

`max_n_prebins` Integer. Maximum number of initial bins before dynamic programming optimization. Controls computational complexity. Must be >= 2. Defaults to 20.

`convergence_threshold`
Numeric. Convergence tolerance for the iterative dynamic programming updates. Smaller values require stricter convergence. Must be > 0. Defaults to 1e-6.

`max_iterations` Integer. Maximum number of dynamic programming iterations. Prevents excessive computation in edge cases. Must be > 0. Defaults to 1000.

`bin_separator` Character string used to concatenate category names when multiple categories are merged into a single bin. Defaults to `"%;%"`.

`monotonic_trend`
Character string specifying monotonicity constraint for Weight of Evidence. Must be one of:

`"auto"` Automatically determine trend direction (default)

`"ascending"` Enforce increasing WoE across bins

`"descending"` Enforce decreasing WoE across bins

`"none"` No monotonicity constraint

Monotonicity constraints are enforced during the DP optimization phase. Defaults to `"auto"`.

Details

This implementation uses dynamic programming to find the globally optimal binning solution that maximizes total Information Value subject to constraints.

Algorithm Workflow:

1. Input validation and data preprocessing
2. Rare category merging (frequencies below `bin_cutoff`)
3. Pre-binning limitation (if categories exceed `max_n_prebins`)
4. Category sorting by event rate
5. Dynamic programming table initialization
6. Iterative DP optimization with optional monotonicity constraints
7. Backtracking to construct optimal bins

8. Final metric computation

Dynamic Programming Formulation:

Let $DP[i][k]$ represent the maximum total IV achievable using the first i categories partitioned into k bins. The recurrence relation is:

$$DP[i][k] = \max_{j<i}\{DP[j][k-1] + IV(j+1, i)\}$$

where $IV(j+1, i)$ is the Information Value of a bin containing categories from $j+1$ to i. Monotonicity constraints are enforced by restricting transitions that violate WoE ordering.

Computational Complexity:

- Time: $O(n^2 \cdot k \cdot m)$ where n = categories, k = max_bins, m = iterations
- Space: $O(n \cdot k)$ for DP tables

Advantages over Heuristic Methods:

- Guarantees global optimality (within constraint space)
- Explicit monotonicity enforcement
- Deterministic and reproducible results
- Efficient caching mechanism for bin statistics

Value

A list containing the binning results with the following components:

`id` Integer vector of bin identifiers (1-indexed)

`bin` Character vector of bin labels (merged category names)

`woe` Numeric vector of Weight of Evidence values per bin

`iv` Numeric vector of Information Value contribution per bin

`count` Integer vector of total observations per bin

`count_pos` Integer vector of positive cases (target=1) per bin

`count_neg` Integer vector of negative cases (target=0) per bin

`event_rate` Numeric vector of event rates per bin

`total_iv` Numeric total Information Value of the binning solution

`converged` Logical indicating if the DP algorithm converged

`iterations` Integer count of DP iterations performed

`execution_time_ms` Numeric execution time in milliseconds

References

Navas-Palencia, G. (2022). Optimal Binning: Mathematical Programming Formulation. *arXiv preprint arXiv:2001.08025*.

Bellman, R. (1954). The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6), 503-515.

Siddiqi, N. (2017). *Intelligent Credit Scoring: Building and Implementing Better Credit Risk Scorecards* (2nd ed.). Wiley.

Thomas, L. C., Edelman, D. B., & Crook, J. N. (2017). *Credit Scoring and Its Applications* (2nd ed.). SIAM.

See Also

ob_categorical_cm for ChiMerge-based binning, ob_categorical_dmiv for divergence measure-based binning

Examples

```
# Example 1: Basic usage with monotonic WoE enforcement
set.seed(123)
n_obs <- 1000

# Simulate education levels with increasing default risk
education <- c("High School", "Associate", "Bachelor", "Master", "PhD")
default_probs <- c(0.20, 0.15, 0.10, 0.06, 0.03)

cat_feature <- sample(education, n_obs,
  replace = TRUE,
  prob = c(0.30, 0.25, 0.25, 0.15, 0.05)
)
bin_target <- sapply(cat_feature, function(x) {
  rbinom(1, 1, default_probs[which(education == x)])
})

# Apply DP binning with ascending monotonicity
result_dp <- ob_categorical_dp(
  cat_feature,
  bin_target,
  min_bins = 2,
  max_bins = 4,
  monotonic_trend = "ascending"
)

# Display results
print(data.frame(
  Bin = result_dp$bin,
  WoE = round(result_dp$woe, 3),
  IV = round(result_dp$iv, 4),
  Count = result_dp$count,
  EventRate = round(result_dp$event_rate, 3)
))
```

```
cat("Total IV:", round(result_dp$total_iv, 4), "\n")
cat("Converged:", result_dp$converged, "\n")

# Example 2: Comparing monotonicity constraints
result_dp_asc <- ob_categorical_dp(
  cat_feature, bin_target,
  max_bins = 3,
  monotonic_trend = "ascending"
)

result_dp_none <- ob_categorical_dp(
  cat_feature, bin_target,
  max_bins = 3,
  monotonic_trend = "none"
)

cat("\nWith monotonicity:\n")
cat("  Bins:", length(result_dp_asc$bin), "\n")
cat("  Total IV:", round(result_dp_asc$total_iv, 4), "\n")

cat("\nWithout monotonicity:\n")
cat("  Bins:", length(result_dp_none$bin), "\n")
cat("  Total IV:", round(result_dp_none$total_iv, 4), "\n")

# Example 3: High cardinality with pre-binning
set.seed(456)
n_obs_large <- 5000

# Simulate customer segments (high cardinality)
segments <- paste0("Segment_", LETTERS[1:20])
segment_probs <- runif(20, 0.01, 0.20)

cat_feature_hc <- sample(segments, n_obs_large, replace = TRUE)
bin_target_hc <- rbinom(
  n_obs_large, 1,
  segment_probs[match(cat_feature_hc, segments)]
)

result_dp_hc <- ob_categorical_dp(
  cat_feature_hc,
  bin_target_hc,
  min_bins = 3,
  max_bins = 5,
  bin_cutoff = 0.03,
  max_n_prebins = 10
)

cat("\nHigh cardinality example:\n")
cat("  Original categories:", length(unique(cat_feature_hc)), "\n")
cat("  Final bins:", length(result_dp_hc$bin), "\n")
cat("  Execution time:", result_dp_hc$execution_time_ms, "ms\n")
```

```
# Example 4: Handling missing values
set.seed(789)
cat_feature_na <- cat_feature
cat_feature_na[sample(n_obs, 50)] <- NA # Introduce 5% missing

result_dp_na <- ob_categorical_dp(
  cat_feature_na,
  bin_target,
  min_bins = 2,
  max_bins = 4
)

# Check if NA was treated as a category
na_bin <- grep("NA", result_dp_na$bin, value = TRUE)
if (length(na_bin) > 0) {
  cat("\nNA handling:\n")
  cat("  Bin containing NA:", na_bin, "\n")
}
```

---

ob_categorical_fetb *Optimal Binning for Categorical Variables using Fisher's Exact Test*

Description

Performs supervised discretization of categorical variables using Fisher's Exact Test as the similarity criterion for hierarchical bin merging. This method iteratively merges the most statistically similar bins (highest p-value) while enforcing Weight of Evidence monotonicity, providing a statistically rigorous approach to optimal binning.

Usage

```
ob_categorical_fetb(
  feature,
  target,
  min_bins = 3,
  max_bins = 5,
  bin_cutoff = 0.05,
  max_n_prebins = 20,
  convergence_threshold = 1e-06,
  max_iterations = 1000,
  bin_separator = "%;%"
)
```

Arguments

feature	A character vector or factor representing the categorical predictor variable to be binned. Missing values are automatically converted to the category "NA".

`target`	An integer vector of binary outcomes (0/1) corresponding to each observation in `feature`. Missing values are not permitted.
`min_bins`	Integer. Minimum number of bins to produce. Must be >= 2. The algorithm will not merge below this threshold. Defaults to 3.
`max_bins`	Integer. Maximum number of bins to produce. Must be >= `min_bins`. The algorithm merges bins until this constraint is satisfied. Defaults to 5.
`bin_cutoff`	Numeric. Minimum proportion of total observations required for a category to avoid being classified as rare. Rare categories are pre-merged before the main algorithm. Must be in (0, 1). Defaults to 0.05.
`max_n_prebins`	Integer. Maximum number of initial bins before the merging phase. Controls computational complexity for high-cardinality features. Must be >= 2. Defaults to 20.
`convergence_threshold`	Numeric. Convergence tolerance based on Information Value change between iterations. Algorithm stops when $\|\Delta IV\| <$ `convergence_threshold`. Must be > 0. Defaults to 1e-6.
`max_iterations`	Integer. Maximum number of merge operations allowed. Prevents excessive computation. Must be > 0. Defaults to 1000.
`bin_separator`	Character string used to concatenate category names when multiple categories are merged into a single bin. Defaults to "%;%".

Details

This algorithm employs Fisher's Exact Test to quantify statistical similarity between bins based on their 2×2 contingency tables. Unlike chi-square based methods, Fisher's test provides exact p-values without relying on asymptotic approximations, making it particularly suitable for small sample sizes.

Algorithm Workflow:

1. Data preprocessing and frequency computation
2. Rare category identification and pre-merging (frequencies < `bin_cutoff`)
3. Initial bin creation (one category per bin)
4. Iterative merging phase:
   - Compute Fisher's Exact Test p-values for all adjacent bin pairs
   - Merge the pair with the **highest** p-value (most similar)
   - Enforce WoE monotonicity after each merge
   - Check convergence based on IV change
5. Final monotonicity enforcement

Fisher's Exact Test:

For two bins with contingency table:

	Bin 1	Bin 2
Positives	a	c

Negatives	b	d

The exact probability under the null hypothesis of independence is:

$$p = \frac{(a+b)!(c+d)!(a+c)!(b+d)!}{n! \cdot a! \cdot b! \cdot c! \cdot d!}$$

where $n = a + b + c + d$. Higher p-values indicate greater similarity (less evidence against the null hypothesis of identical distributions).

Key Features:

- **Exact inference**: No asymptotic approximations required
- **Small sample robustness**: Valid for any sample size
- **Automatic monotonicity**: WoE ordering enforced after each merge
- **Efficient caching**: Log-factorial and p-value caching for speed
- **Rare category handling**: Pre-merging prevents sparse bins

Computational Complexity:

- Time: $O(k^2 \cdot m)$ where k = initial bins, m = iterations
- Space: $O(k + n_{max})$ for bins and factorial cache

Value

A list containing the binning results with the following components:

`id` Integer vector of bin identifiers (1-indexed)

`bin` Character vector of bin labels (merged category names)

`woe` Numeric vector of Weight of Evidence values per bin

`iv` Numeric vector of Information Value contribution per bin

`count` Integer vector of total observations per bin

`count_pos` Integer vector of positive cases (target=1) per bin

`count_neg` Integer vector of negative cases (target=0) per bin

`converged` Logical indicating algorithm convergence

`iterations` Integer count of merge operations performed

References

Fisher, R. A. (1922). On the interpretation of chi-squared from contingency tables, and the calculation of P. *Journal of the Royal Statistical Society*, 85(1), 87-94. doi:10.2307/2340521

Agresti, A. (2013). *Categorical Data Analysis* (3rd ed.). Wiley.

Mehta, C. R., & Patel, N. R. (1983). A network algorithm for performing Fisher's exact test in r×c contingency tables. *Journal of the American Statistical Association*, 78(382), 427-434.

Zeng, G. (2014). A necessary condition for a good binning algorithm in credit scoring. *Applied Mathematical Sciences*, 8(65), 3229-3242.

See Also

ob_categorical_cm for ChiMerge-based binning, ob_categorical_dp for dynamic programming approach, ob_categorical_dmiv for divergence measure-based binning

Examples

```
# Example 1: Basic usage with Fisher's Exact Test
set.seed(42)
n_obs <- 800

# Simulate customer segments with different risk profiles
segments <- c("Premium", "Standard", "Basic", "Budget", "Economy")
risk_rates <- c(0.05, 0.10, 0.15, 0.22, 0.30)

cat_feature <- sample(segments, n_obs,
  replace = TRUE,
  prob = c(0.15, 0.25, 0.30, 0.20, 0.10)
)
bin_target <- sapply(cat_feature, function(x) {
  rbinom(1, 1, risk_rates[which(segments == x)])
})

# Apply Fisher's Exact Test binning
result_fetb <- ob_categorical_fetb(
  cat_feature,
  bin_target,
  min_bins = 2,
  max_bins = 4
)

# Display results
print(data.frame(
  Bin = result_fetb$bin,
  WoE = round(result_fetb$woe, 3),
  IV = round(result_fetb$iv, 4),
  Count = result_fetb$count,
  EventRate = round(result_fetb$count_pos / result_fetb$count, 3)
))

cat("\nAlgorithm converged:", result_fetb$converged, "\n")
cat("Iterations performed:", result_fetb$iterations, "\n")

# Example 2: Comparing with ChiMerge method
result_cm <- ob_categorical_cm(
  cat_feature,
  bin_target,
  min_bins = 2,
  max_bins = 4
)

cat("\nFisher's Exact Test:\n")
cat("  Final bins:", length(result_fetb$bin), "\n")
```

```
cat("  Total IV:", round(sum(result_fetb$iv), 4), "\n")

cat("\nChiMerge:\n")
cat("  Final bins:", length(result_cm$bin), "\n")
cat("  Total IV:", round(sum(result_cm$iv), 4), "\n")

# Example 3: Small sample size (Fisher's advantage)
set.seed(123)
n_obs_small <- 150

# Small sample with sparse categories
occupation <- c(
  "Doctor", "Lawyer", "Teacher", "Engineer",
  "Sales", "Manager"
)

cat_feature_small <- sample(occupation, n_obs_small,
  replace = TRUE,
  prob = c(0.10, 0.10, 0.20, 0.25, 0.20, 0.15)
)
bin_target_small <- rbinom(n_obs_small, 1, 0.12)

result_fetb_small <- ob_categorical_fetb(
  cat_feature_small,
  bin_target_small,
  min_bins = 2,
  max_bins = 3,
  bin_cutoff = 0.03 # Allow smaller bins for small sample
)

cat("\nSmall sample binning:\n")
cat("  Observations:", n_obs_small, "\n")
cat("  Original categories:", length(unique(cat_feature_small)), "\n")
cat("  Final bins:", length(result_fetb_small$bin), "\n")
cat("  Converged:", result_fetb_small$converged, "\n")

# Example 4: High cardinality with rare categories
set.seed(789)
n_obs_hc <- 2000

# Simulate product codes (high cardinality)
product_codes <- paste0("PROD_", sprintf("%03d", 1:30))

cat_feature_hc <- sample(product_codes, n_obs_hc,
  replace = TRUE,
  prob = c(
    rep(0.05, 10), rep(0.02, 10),
    rep(0.01, 10)
  )
)
bin_target_hc <- rbinom(n_obs_hc, 1, 0.08)

result_fetb_hc <- ob_categorical_fetb(
```

```
  cat_feature_hc,
  bin_target_hc,
  min_bins = 3,
  max_bins = 6,
  bin_cutoff = 0.02,
  max_n_prebins = 15
)

cat("\nHigh cardinality example:\n")
cat("  Original categories:", length(unique(cat_feature_hc)), "\n")
cat("  Final bins:", length(result_fetb_hc$bin), "\n")
cat("  Iterations:", result_fetb_hc$iterations, "\n")

# Check for rare category merging
for (i in seq_along(result_fetb_hc$bin)) {
  n_merged <- length(strsplit(result_fetb_hc$bin[i], "%;%")[[1]])
  if (n_merged > 1) {
    cat("  Bin", i, "contains", n_merged, "merged categories\n")
  }
}

# Example 5: Missing value handling
set.seed(456)
cat_feature_na <- cat_feature
na_indices <- sample(n_obs, 40) # 5% missing
cat_feature_na[na_indices] <- NA

result_fetb_na <- ob_categorical_fetb(
  cat_feature_na,
  bin_target,
  min_bins = 2,
  max_bins = 4
)

# Check NA treatment
na_bin_idx <- grep("NA", result_fetb_na$bin)
if (length(na_bin_idx) > 0) {
  cat("\nMissing value handling:\n")
  cat("  NA bin:", result_fetb_na$bin[na_bin_idx], "\n")
  cat("  NA count:", result_fetb_na$count[na_bin_idx], "\n")
  cat("  NA WoE:", round(result_fetb_na$woe[na_bin_idx], 3), "\n")
}
```

---

ob_categorical_gmb *Optimal Binning for Categorical Variables using Greedy Merge Algorithm*

Description

Performs supervised discretization of categorical variables using a greedy bottom-up merging strategy that iteratively combines bins to maximize total Information Value (IV). This approach uses Bayesian smoothing for numerical stability and employs adaptive monotonicity constraints, providing a fast approximation to optimal binning suitable for high-cardinality features.

Usage

```
ob_categorical_gmb(
  feature,
  target,
  min_bins = 3,
  max_bins = 5,
  bin_cutoff = 0.05,
  max_n_prebins = 20,
  bin_separator = "%;%",
  convergence_threshold = 1e-06,
  max_iterations = 1000
)
```

Arguments

`feature`	A character vector or factor representing the categorical predictor variable to be binned. Missing values are automatically converted to the category `"NA"`.
`target`	An integer vector of binary outcomes (0/1) corresponding to each observation in `feature`. Missing values are not permitted.
`min_bins`	Integer. Minimum number of bins to produce. Must be >= 2. Merging stops when this threshold is reached. Defaults to 3.
`max_bins`	Integer. Maximum number of bins to produce. Must be >= `min_bins`. The algorithm terminates when bins are reduced to this number. Defaults to 5.
`bin_cutoff`	Numeric. Minimum proportion of total observations required for a category to remain separate during initialization. Categories below this threshold are pre-merged. Must be in (0, 1). Defaults to 0.05.
`max_n_prebins`	Integer. Maximum number of initial bins before the greedy merging phase. Controls computational complexity. Must be >= `min_bins`. Defaults to 20.
`bin_separator`	Character string used to concatenate category names when multiple categories are merged into a single bin. Defaults to `"%;%"`.
`convergence_threshold`	Numeric. Convergence tolerance for IV change between iterations. Algorithm stops when $\|\Delta IV\| <$ `convergence_threshold`. Must be > 0. Defaults to 1e-6.
`max_iterations`	Integer. Maximum number of merge operations allowed. Prevents excessive computation. Must be > 0. Defaults to 1000.

Details

The Greedy Merge Binning (GMB) algorithm employs a bottom-up approach where bins are iteratively merged based on maximum IV improvement. Unlike exact optimization methods (e.g.,

dynamic programming), GMB provides approximate solutions with significantly reduced computational cost.

Algorithm Workflow:

1. Input validation and preprocessing
2. Initial bin creation (one category per bin)
3. Rare category merging (frequencies < `bin_cutoff`)
4. Pre-bin limitation (if bins > `max_n_prebins`)
5. Greedy merging phase:
   - Evaluate IV for all possible adjacent bin merges
   - Select merge that maximizes total IV
   - Apply tie-breaking rules for similar merges
   - Update IV cache incrementally
   - Check convergence criteria
6. Adaptive monotonicity enforcement
7. Final metric computation

Bayesian Smoothing:

To prevent numerical instability with sparse bins, WoE is calculated using Bayesian smoothing:

$$WoE_i = \ln\left(\frac{p_i + \alpha_p}{n_i + \alpha_n}\right)$$

where α_p and α_n are prior pseudocounts proportional to the overall event rate. This regularization ensures stable WoE estimates even for bins with zero events.

Greedy Selection Criterion:

At each iteration, the algorithm evaluates the IV gain for merging adjacent bins i and j:

$$\Delta IV_{i,j} = IV_{merged}(i,j) - (IV_i + IV_j)$$

The pair with maximum ΔIV is merged. Early stopping occurs if $\Delta IV > 0.05 \cdot IV_{current}$ (5% improvement threshold).

Tie Handling:

When multiple merges yield similar IV gains (within 10× convergence threshold), the algorithm prefers merges that produce more balanced bins, breaking ties based on size difference:

$$balance_score = |count_i - count_j|$$

Computational Complexity:

- Time: $O(k^2 \cdot m)$ where k = bins, m = iterations
- Space: $O(k^2)$ for IV cache (optional)
- Typical runtime: 10-100× faster than exact methods for $k > 20$

Advantages:

- Fast execution for high-cardinality features
- Incremental IV caching for efficiency
- Bayesian smoothing prevents overfitting
- Adaptive monotonicity with gradient relaxation
- Handles imbalanced datasets robustly

Limitations:

- Approximate solution (not guaranteed global optimum)
- Greedy nature may miss better non-adjacent merges
- Sensitive to initialization order

Value

A list containing the binning results with the following components:

`id` Integer vector of bin identifiers (1-indexed)

`bin` Character vector of bin labels (merged category names)

`woe` Numeric vector of Weight of Evidence values per bin

`iv` Numeric vector of Information Value contribution per bin

`count` Integer vector of total observations per bin

`count_pos` Integer vector of positive cases (target=1) per bin

`count_neg` Integer vector of negative cases (target=0) per bin

`total_iv` Numeric total Information Value of the binning solution

`converged` Logical indicating algorithm convergence

`iterations` Integer count of merge operations performed

References

Navas-Palencia, G. (2020). Optimal binning: mathematical programming formulation and solution approach. *Expert Systems with Applications*, 158, 113508. doi:10.1016/j.eswa.2020.113508

Good, I. J. (1965). The Estimation of Probabilities: An Essay on Modern Bayesian Methods. MIT Press.

Zeng, G. (2014). A necessary condition for a good binning algorithm in credit scoring. *Applied Mathematical Sciences*, 8(65), 3229-3242.

Mironchyk, P., & Tchistiakov, V. (2017). Monotone optimal binning algorithm for credit risk modeling. *SSRN Electronic Journal*. doi:10.2139/ssrn.2978774

See Also

`ob_categorical_dp` for exact optimization via dynamic programming, `ob_categorical_cm` for ChiMerge-based binning, `ob_categorical_fetb` for Fisher's Exact Test binning

Examples

```
# Example 1: Basic greedy merge binning
set.seed(123)
n_obs <- 1500

# Simulate customer types with varying risk
customer_types <- c(
  "Premium", "Gold", "Silver", "Bronze",
  "Basic", "Trial"
)
risk_probs <- c(0.02, 0.05, 0.10, 0.15, 0.22, 0.35)

cat_feature <- sample(customer_types, n_obs,
  replace = TRUE,
  prob = c(0.10, 0.15, 0.25, 0.25, 0.15, 0.10)
)
bin_target <- sapply(cat_feature, function(x) {
  rbinom(1, 1, risk_probs[which(customer_types == x)])
})

# Apply greedy merge binning
result_gmb <- ob_categorical_gmb(
  cat_feature,
  bin_target,
  min_bins = 3,
  max_bins = 4
)

# Display results
print(data.frame(
  Bin = result_gmb$bin,
  WoE = round(result_gmb$woe, 3),
  IV = round(result_gmb$iv, 4),
  Count = result_gmb$count,
  EventRate = round(result_gmb$count_pos / result_gmb$count, 3)
))

cat("\nTotal IV:", round(result_gmb$total_iv, 4), "\n")
cat("Converged:", result_gmb$converged, "\n")
cat("Iterations:", result_gmb$iterations, "\n")

# Example 2: Comparing speed with exact methods
set.seed(456)
n_obs <- 3000

# High cardinality feature
regions <- paste0("Region_", sprintf("%02d", 1:25))
cat_feature_hc <- sample(regions, n_obs, replace = TRUE)
bin_target_hc <- rbinom(n_obs, 1, 0.12)

# Greedy approach (fast)
time_gmb <- system.time({
```

```
  result_gmb_hc <- ob_categorical_gmb(
    cat_feature_hc,
    bin_target_hc,
    min_bins = 3,
    max_bins = 6,
    max_n_prebins = 20
  )
})

# Dynamic programming (exact but slower)
time_dp <- system.time({
  result_dp_hc <- ob_categorical_dp(
    cat_feature_hc,
    bin_target_hc,
    min_bins = 3,
    max_bins = 6,
    max_n_prebins = 20
  )
})

cat("\nPerformance comparison (high cardinality):\n")
cat("  GMB time:", round(time_gmb[3], 3), "seconds\n")
cat("  DP time:", round(time_dp[3], 3), "seconds\n")
cat("  Speedup:", round(time_dp[3] / time_gmb[3], 1), "x\n")
cat("\n  GMB IV:", round(result_gmb_hc$total_iv, 4), "\n")
cat("  DP IV:", round(result_dp_hc$total_iv, 4), "\n")

# Example 3: Convergence behavior
set.seed(789)
n_obs_conv <- 1000

# Feature with natural groupings
education <- c("PhD", "Master", "Bachelor", "HighSchool", "NoHighSchool")
cat_feature_conv <- sample(education, n_obs_conv,
  replace = TRUE,
  prob = c(0.05, 0.15, 0.35, 0.30, 0.15)
)
bin_target_conv <- sapply(cat_feature_conv, function(x) {
  probs <- c(0.02, 0.05, 0.08, 0.15, 0.25)
  rbinom(1, 1, probs[which(education == x)])
})

# Test different convergence thresholds
thresholds <- c(1e-3, 1e-6, 1e-9)

for (thresh in thresholds) {
  result_conv <- ob_categorical_gmb(
    cat_feature_conv,
    bin_target_conv,
    min_bins = 2,
    max_bins = 4,
    convergence_threshold = thresh
  )
```

```
  cat(sprintf(
    "\nThreshold %.0e: %d iterations, converged=%s\n",
    thresh, result_conv$iterations, result_conv$converged
  ))
}

# Example 4: Handling rare categories
set.seed(321)
n_obs_rare <- 2000

# Simulate with many rare categories
products <- c(paste0("Common_", 1:5), paste0("Rare_", 1:15))
product_probs <- c(rep(0.15, 5), rep(0.01, 15))

cat_feature_rare <- sample(products, n_obs_rare,
  replace = TRUE,
  prob = product_probs
)
bin_target_rare <- rbinom(n_obs_rare, 1, 0.10)

result_gmb_rare <- ob_categorical_gmb(
  cat_feature_rare,
  bin_target_rare,
  min_bins = 3,
  max_bins = 5,
  bin_cutoff = 0.03 # Aggressive rare category merging
)

cat("\nRare category handling:\n")
cat("  Original categories:", length(unique(cat_feature_rare)), "\n")
cat("  Final bins:", length(result_gmb_rare$bin), "\n")

# Count merged rare categories
for (i in seq_along(result_gmb_rare$bin)) {
  n_merged <- length(strsplit(result_gmb_rare$bin[i], "%;%")[[1]])
  if (n_merged > 1) {
    cat(sprintf("  Bin %d: %d categories merged\n", i, n_merged))
  }
}

# Example 5: Imbalanced dataset robustness
set.seed(555)
n_obs_imb <- 1200

# Highly imbalanced target (2% event rate)
cat_feature_imb <- sample(c("A", "B", "C", "D", "E"),
  n_obs_imb,
  replace = TRUE
)
bin_target_imb <- rbinom(n_obs_imb, 1, 0.02)

result_gmb_imb <- ob_categorical_gmb(
```

```
  cat_feature_imb,
  bin_target_imb,
  min_bins = 2,
  max_bins = 3
)

cat("\nImbalanced dataset:\n")
cat("  Event rate:", round(mean(bin_target_imb), 4), "\n")
cat("  Total events:", sum(bin_target_imb), "\n")
cat("  Bins created:", length(result_gmb_imb$bin), "\n")
cat("  WoE range:", sprintf(
  "[%.2f, %.2f]\n",
  min(result_gmb_imb$woe),
  max(result_gmb_imb$woe)
))
```

`ob_categorical_ivb` *Optimal Binning for Categorical Variables using Information Value Dynamic Programming*

Description

Performs supervised discretization of categorical variables using a dynamic programming algorithm specifically designed to maximize total Information Value (IV). This implementation employs Bayesian smoothing for numerical stability, maintains monotonic Weight of Evidence constraints, and uses efficient caching strategies for optimal performance with high-cardinality features.

Usage

```
ob_categorical_ivb(
  feature,
  target,
  min_bins = 3,
  max_bins = 5,
  bin_cutoff = 0.05,
  max_n_prebins = 20,
  bin_separator = "%;%",
  convergence_threshold = 1e-06,
  max_iterations = 1000
)
```

Arguments

`feature`	A character vector or factor representing the categorical predictor variable to be binned. Missing values are automatically converted to the category `"NA"`.
`target`	An integer vector of binary outcomes (0/1) corresponding to each observation in `feature`. Missing values are not permitted.

`min_bins` Integer. Minimum number of bins to produce. Must be >= 2. The algorithm searches for solutions within [`min_bins`, `max_bins`] that maximize total IV. Defaults to 3.

`max_bins` Integer. Maximum number of bins to produce. Must be >= `min_bins`. Defines the upper bound of the search space. Defaults to 5.

`bin_cutoff` Numeric. Minimum proportion of total observations required for a category to remain separate. Categories below this threshold are pre-merged before the optimization phase. Must be in (0, 1). Defaults to 0.05.

`max_n_prebins` Integer. Maximum number of initial bins before dynamic programming optimization. Controls computational complexity for high-cardinality features. Must be >= 2. Defaults to 20.

`bin_separator` Character string used to concatenate category names when multiple categories are merged into a single bin. Defaults to "%;%".

`convergence_threshold` Numeric. Convergence tolerance for the iterative optimization process based on IV change. Algorithm stops when $|\Delta IV| <$ `convergence_threshold`. Must be > 0. Defaults to 1e-6.

`max_iterations` Integer. Maximum number of optimization iterations. Prevents excessive computation. Must be > 0. Defaults to 1000.

Details

The Information Value Binning (IVB) algorithm uses dynamic programming to find the globally optimal binning solution that maximizes total IV subject to constraints on bin count and monotonicity.

Algorithm Workflow:

1. Input validation and preprocessing
2. Single-pass category counting and statistics computation
3. Rare category pre-merging (frequencies < `bin_cutoff`)
4. Pre-bin limitation (if categories > `max_n_prebins`)
5. Category sorting by event rate
6. Cumulative statistics cache initialization
7. Dynamic programming table computation:
   - State: $DP[i][k]$ = max IV using first i categories in k bins
   - Transition: $DP[i][k] = \max_j \{DP[j][k-1] + IV(j+1, i)\}$
   - Banded optimization to skip infeasible splits
8. Backtracking to reconstruct optimal bins
9. Adaptive monotonicity enforcement
10. Final metric computation with Bayesian smoothing

Dynamic Programming Formulation:

Let $DP[i][k]$ represent the maximum total IV achievable using the first i categories (sorted by event rate) partitioned into k bins.

Recurrence relation:

$$DP[i][k] = \max_{k-1 \le j < i} \{DP[j][k-1] + IV(j+1, i)\}$$

Base case:

$$DP[i][1] = IV(1, i) \quad \forall i$$

where $IV(j+1, i)$ is the Information Value of a bin containing categories from $j+1$ to i.

Bayesian Smoothing:

To prevent numerical instability and overfitting with sparse bins, WoE and IV are calculated using Bayesian smoothing with pseudocounts:

$$p_i' = \frac{n_{i,pos} + \alpha_p}{N_{pos} + \alpha_{total}}$$

$$n_i' = \frac{n_{i,neg} + \alpha_n}{N_{neg} + \alpha_{total}}$$

where α_p and α_n are prior pseudocounts proportional to the overall event rate, and $\alpha_{total} = 1.0$ (prior strength).

$$WoE_i = \ln\left(\frac{p_i'}{n_i'}\right)$$

$$IV_i = (p_i' - n_i') \times WoE_i$$

Adaptive Monotonicity Enforcement:

After finding the optimal bins, the algorithm enforces WoE monotonicity by:

1. Computing average WoE gap: $\bar{\Delta} = \frac{1}{k-1} \sum_{i=1}^{k-1} |WoE_{i+1} - WoE_i|$
2. Setting adaptive threshold: $\tau = \min(\epsilon, 0.01\bar{\Delta})$
3. Identifying worst violation: $i^* = \arg\max_i \{WoE_i - WoE_{i+1}\}$
4. Evaluating forward and backward merges by IV retention
5. Selecting merge direction that maximizes total IV

Computational Complexity:

- Time: $O(k^2 \cdot n)$ where k = max_bins, n = categories
- Space: $O(k \cdot n)$ for DP tables and cumulative caches
- IV calculations are $O(1)$ due to cumulative statistics caching

Advantages over Alternative Methods:

- **Global optimality**: Guaranteed maximum IV (within constraint space)
- **Bayesian regularization**: Robust to sparse bins and class imbalance
- **Efficient caching**: Cumulative stats and IV memoization
- **Banded optimization**: Reduced search space via feasibility pruning

- **Adaptive monotonicity**: Context-aware threshold for enforcement

Comparison with Related Methods:

- **vs DP (general)**: IVB specifically optimizes IV; general DP more flexible
- **vs GMB**: IVB guarantees optimality; GMB is faster but approximate
- **vs ChiMerge**: IVB uses IV criterion; ChiMerge uses chi-square

Value

A list containing the binning results with the following components:

`id` Integer vector of bin identifiers (1-indexed)

`bin` Character vector of bin labels (merged category names)

`woe` Numeric vector of Weight of Evidence values per bin

`iv` Numeric vector of Information Value contribution per bin

`count` Integer vector of total observations per bin

`count_pos` Integer vector of positive cases (target=1) per bin

`count_neg` Integer vector of negative cases (target=0) per bin

`total_iv` Numeric total Information Value of the binning solution

`converged` Logical indicating algorithm convergence

`iterations` Integer count of optimization iterations performed

References

Navas-Palencia, G. (2020). Optimal binning: mathematical programming formulation and solution approach. *Expert Systems with Applications*, 158, 113508. doi:10.1016/j.eswa.2020.113508

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press.

Siddiqi, N. (2017). *Intelligent Credit Scoring: Building and Implementing Better Credit Risk Scorecards* (2nd ed.). Wiley.

Good, I. J. (1965). *The Estimation of Probabilities: An Essay on Modern Bayesian Methods*. MIT Press.

Anderson, R. (2007). *The Credit Scoring Toolkit: Theory and Practice for Retail Credit Risk Management and Decision Automation*. Oxford University Press.

See Also

`ob_categorical_dp` for general dynamic programming binning, `ob_categorical_gmb` for greedy merge approximation, `ob_categorical_cm` for ChiMerge-based binning

Examples

```
# Example 1: Basic IV optimization with Bayesian smoothing
set.seed(42)
n_obs <- 1200

# Simulate industry sectors with varying default risk
industries <- c(
  "Technology", "Healthcare", "Finance", "Manufacturing",
  "Retail", "Energy"
)
default_rates <- c(0.03, 0.05, 0.08, 0.12, 0.18, 0.25)

cat_feature <- sample(industries, n_obs,
  replace = TRUE,
  prob = c(0.20, 0.18, 0.22, 0.18, 0.12, 0.10)
)
bin_target <- sapply(cat_feature, function(x) {
  rbinom(1, 1, default_rates[which(industries == x)])
})

# Apply IVB optimization
result_ivb <- ob_categorical_ivb(
  cat_feature,
  bin_target,
  min_bins = 3,
  max_bins = 4
)

# Display results
print(data.frame(
  Bin = result_ivb$bin,
  WoE = round(result_ivb$woe, 3),
  IV = round(result_ivb$iv, 4),
  Count = result_ivb$count,
  EventRate = round(result_ivb$count_pos / result_ivb$count, 3)
))

cat("\nTotal IV (maximized):", round(result_ivb$total_iv, 4), "\n")
cat("Converged:", result_ivb$converged, "\n")
cat("Iterations:", result_ivb$iterations, "\n")

# Example 2: Comparing IV optimization with other methods
set.seed(123)
n_obs_comp <- 1500

regions <- c("North", "South", "East", "West", "Central")
cat_feature_comp <- sample(regions, n_obs_comp, replace = TRUE)
bin_target_comp <- rbinom(n_obs_comp, 1, 0.15)

# IVB (IV-optimized)
result_ivb_comp <- ob_categorical_ivb(
  cat_feature_comp, bin_target_comp,
```

```
  min_bins = 2, max_bins = 3
)

# GMB (greedy approximation)
result_gmb_comp <- ob_categorical_gmb(
  cat_feature_comp, bin_target_comp,
  min_bins = 2, max_bins = 3
)

# DP (general optimization)
result_dp_comp <- ob_categorical_dp(
  cat_feature_comp, bin_target_comp,
  min_bins = 2, max_bins = 3
)

cat("\nMethod comparison:\n")
cat("  IVB total IV:", round(result_ivb_comp$total_iv, 4), "\n")
cat("  GMB total IV:", round(result_gmb_comp$total_iv, 4), "\n")
cat("  DP total IV:", round(result_dp_comp$total_iv, 4), "\n")
cat("\nIVB typically achieves highest IV due to explicit optimization\n")
```

---

ob_categorical_jedi *Optimal Binning for Categorical Variables using JEDI Algorithm*

Description

Performs supervised discretization of categorical variables using the Joint Entropy-Driven Information Maximization (JEDI) algorithm. This advanced method combines information-theoretic optimization with intelligent bin merging strategies, employing Bayesian smoothing for numerical stability and adaptive monotonicity enforcement to produce robust, interpretable binning solutions.

Usage

```
ob_categorical_jedi(
  feature,
  target,
  min_bins = 3,
  max_bins = 5,
  bin_cutoff = 0.05,
  max_n_prebins = 20,
  bin_separator = "%;%",
  convergence_threshold = 1e-06,
  max_iterations = 1000
)
```

Arguments

`feature`	A character vector or factor representing the categorical predictor variable to be binned. Missing values are automatically converted to the category `"NA"`.
`target`	An integer vector of binary outcomes (0/1) corresponding to each observation in `feature`. Missing values are not permitted.
`min_bins`	Integer. Minimum number of bins to produce. Must be >= 2. The algorithm will not merge below this threshold. Defaults to 3.
`max_bins`	Integer. Maximum number of bins to produce. Must be >= `min_bins`. The algorithm iteratively merges until this constraint is satisfied. Defaults to 5.
`bin_cutoff`	Numeric. Minimum proportion of total observations required for a category to remain separate during initialization. Categories below this threshold are pre-merged into an "Others" bin. Must be in (0, 1). Defaults to 0.05.
`max_n_prebins`	Integer. Maximum number of initial bins before the main optimization phase. Controls computational complexity for high-cardinality features. Must be >= `min_bins`. Defaults to 20.
`bin_separator`	Character string used to concatenate category names when multiple categories are merged into a single bin. Defaults to "%;%".
`convergence_threshold`	Numeric. Convergence tolerance based on Information Value change between iterations. Algorithm stops when $\|\Delta IV\| <$ `convergence_threshold`. Must be > 0. Defaults to 1e-6.
`max_iterations`	Integer. Maximum number of optimization iterations. Prevents infinite loops in edge cases. Must be > 0. Defaults to 1000.

Details

The JEDI (Joint Entropy-Driven Information Maximization) algorithm represents a sophisticated approach to categorical binning that jointly optimizes Information Value while maintaining monotonic Weight of Evidence constraints through intelligent violation detection and repair strategies.

Algorithm Workflow:

1. Input validation and preprocessing
2. Initial bin creation (one category per bin)
3. Rare category merging (frequencies < `bin_cutoff`)
4. WoE-based monotonic sorting
5. Pre-bin limitation via minimal IV-loss merging
6. Main optimization loop:
    - Monotonicity violation detection (peaks and valleys)
    - Violation severity quantification
    - Intelligent merge selection (minimize IV loss)
    - Convergence monitoring
    - Best solution tracking
7. Final constraint satisfaction (max_bins enforcement)

8. Bayesian-smoothed metric computation

Joint Entropy-Driven Optimization:

Unlike greedy algorithms that optimize locally, JEDI considers the global impact of each merge on total Information Value:

$$IV_{total} = \sum_{i=1}^{k} (p_i - n_i) \times \ln\left(\frac{p_i}{n_i}\right)$$

For each potential merge of bins j and $j+1$, JEDI evaluates:

$$\Delta IV_{j,j+1} = IV_{current} - IV_{merged}(j, j+1)$$

The pair with minimum ΔIV (least information loss) is selected.

Violation Detection and Repair:

JEDI identifies two types of monotonicity violations:

- **Peaks**: $WoE_i > WoE_{i-1}$ and $WoE_i > WoE_{i+1}$
- **Valleys**: $WoE_i < WoE_{i-1}$ and $WoE_i < WoE_{i+1}$

For each violation, severity is quantified as:

$$severity_i = \max\{|WoE_i - WoE_{i-1}|, |WoE_i - WoE_{i+1}|\}$$

The algorithm prioritizes fixing the most severe violation first, evaluating both forward merge $(i, i+1)$ and backward merge $(i-1, i)$ to select the option that minimizes information loss.

Bayesian Smoothing:

To ensure numerical stability with sparse bins, JEDI applies Bayesian smoothing:

$$p_i' = \frac{n_{i,pos} + \alpha_p}{N_{pos} + \alpha_{total}}$$
$$n_i' = \frac{n_{i,neg} + \alpha_n}{N_{neg} + \alpha_{total}}$$

where prior pseudocounts are proportional to overall prevalence:

$$\alpha_p = \alpha_{total} \times \frac{N_{pos}}{N_{pos} + N_{neg}}$$
$$\alpha_n = \alpha_{total} - \alpha_p$$

with $\alpha_{total} = 1.0$ as the prior strength parameter.

Adaptive Monotonicity Threshold:

Rather than using a fixed threshold, JEDI computes a context-aware tolerance:

$$\bar{\Delta} = \frac{1}{k-1} \sum_{i=1}^{k-1} |WoE_{i+1} - WoE_i|$$

$$\tau = \min(\epsilon, 0.01\bar{\Delta})$$

This adaptive approach prevents over-merging when natural WoE gaps are small.

Computational Complexity:

- Time: $O(k^2 \cdot m)$ where k = bins, m = iterations
- Space: $O(k^2)$ for IV cache
- Cache hit rate typically > 70% for $k > 10$

Key Innovations:

- **Joint optimization**: Global IV consideration (vs. local greedy)
- **Smart violation repair**: Severity-based prioritization
- **Bidirectional merge evaluation**: Forward vs. backward analysis
- **Best solution tracking**: Retains optimal intermediate states
- **Adaptive thresholds**: Context-aware monotonicity tolerance

Comparison with Related Methods:

Method	Optimization	Monotonicity	Speed
JEDI	Joint/Global	Adaptive	Medium
IVB	DP (Exact)	Enforced	Slow
GMB	Greedy/Local	Enforced	Fast
ChiMerge	Statistical	Optional	Fast

Value

A list containing the binning results with the following components:

`id` Integer vector of bin identifiers (1-indexed)

`bin` Character vector of bin labels (merged category names)

`woe` Numeric vector of Weight of Evidence values per bin

`iv` Numeric vector of Information Value contribution per bin

`count` Integer vector of total observations per bin

`count_pos` Integer vector of positive cases (target=1) per bin

`count_neg` Integer vector of negative cases (target=0) per bin

`total_iv` Numeric total Information Value of the binning solution

`converged` Logical indicating algorithm convergence

`iterations` Integer count of optimization iterations performed

References

Cover, T. M., & Thomas, J. A. (2006). *Elements of Information Theory* (2nd ed.). Wiley-Interscience. doi:10.1002/047174882X

Kullback, S. (1959). *Information Theory and Statistics*. Wiley.

Navas-Palencia, G. (2020). Optimal binning: mathematical programming formulation and solution approach. *Expert Systems with Applications*, 158, 113508. doi:10.1016/j.eswa.2020.113508

Good, I. J. (1965). *The Estimation of Probabilities: An Essay on Modern Bayesian Methods*. MIT Press.

Zeng, G. (2014). A necessary condition for a good binning algorithm in credit scoring. *Applied Mathematical Sciences*, 8(65), 3229-3242.

See Also

ob_categorical_ivb for Information Value DP optimization, ob_categorical_gmb for greedy merge binning, ob_categorical_dp for general dynamic programming, ob_categorical_cm for ChiMerge-based binning

Examples

```
# Example 1: Basic JEDI optimization
set.seed(42)
n_obs <- 1500

# Simulate employment types with risk gradient
employment <- c(
  "Permanent", "Contract", "Temporary", "SelfEmployed",
  "Unemployed", "Student", "Retired"
)
risk_rates <- c(0.03, 0.08, 0.15, 0.12, 0.35, 0.25, 0.10)

cat_feature <- sample(employment, n_obs,
  replace = TRUE,
  prob = c(0.35, 0.20, 0.15, 0.12, 0.08, 0.06, 0.04)
)
bin_target <- sapply(cat_feature, function(x) {
  rbinom(1, 1, risk_rates[which(employment == x)])
})

# Apply JEDI algorithm
result_jedi <- ob_categorical_jedi(
  cat_feature,
  bin_target,
  min_bins = 3,
  max_bins = 5
)

# Display results
print(data.frame(
  Bin = result_jedi$bin,
  WoE = round(result_jedi$woe, 3),
```

```
  IV = round(result_jedi$iv, 4),
  Count = result_jedi$count,
  EventRate = round(result_jedi$count_pos / result_jedi$count, 3)
))

cat("\nTotal IV (jointly optimized):", round(result_jedi$total_iv, 4), "\n")
cat("Converged:", result_jedi$converged, "\n")
cat("Iterations:", result_jedi$iterations, "\n")

# Example 2: Method comparison (JEDI vs alternatives)
set.seed(123)
n_obs_comp <- 2000

departments <- c(
  "Sales", "IT", "HR", "Finance", "Operations",
  "Marketing", "Legal", "R&D"
)
cat_feature_comp <- sample(departments, n_obs_comp, replace = TRUE)
bin_target_comp <- rbinom(n_obs_comp, 1, 0.12)

# JEDI (joint optimization)
result_jedi_comp <- ob_categorical_jedi(
  cat_feature_comp, bin_target_comp,
  min_bins = 3, max_bins = 4
)

# IVB (exact DP)
result_ivb_comp <- ob_categorical_ivb(
  cat_feature_comp, bin_target_comp,
  min_bins = 3, max_bins = 4
)

# GMB (greedy)
result_gmb_comp <- ob_categorical_gmb(
  cat_feature_comp, bin_target_comp,
  min_bins = 3, max_bins = 4
)

cat("\nMethod comparison (Total IV):\n")
cat(
  "  JEDI:", round(result_jedi_comp$total_iv, 4),
  "- converged:", result_jedi_comp$converged, "\n"
)
cat(
  "  IVB:", round(result_ivb_comp$total_iv, 4),
  "- converged:", result_ivb_comp$converged, "\n"
)
cat(
  "  GMB:", round(result_gmb_comp$total_iv, 4),
  "- converged:", result_gmb_comp$converged, "\n"
)

# Example 3: Bayesian smoothing with sparse data
```

```
set.seed(789)
n_obs_sparse <- 400

# Small sample with rare events
categories <- c("A", "B", "C", "D", "E", "F", "G")
cat_probs <- c(0.25, 0.20, 0.18, 0.15, 0.12, 0.07, 0.03)

cat_feature_sparse <- sample(categories, n_obs_sparse,
  replace = TRUE,
  prob = cat_probs
)
bin_target_sparse <- rbinom(n_obs_sparse, 1, 0.05) # 5% event rate

result_jedi_sparse <- ob_categorical_jedi(
  cat_feature_sparse,
  bin_target_sparse,
  min_bins = 2,
  max_bins = 4,
  bin_cutoff = 0.02
)

cat("\nBayesian smoothing (sparse data):\n")
cat("  Sample size:", n_obs_sparse, "\n")
cat("  Total events:", sum(bin_target_sparse), "\n")
cat("  Event rate:", round(mean(bin_target_sparse), 4), "\n")
cat("  Bins created:", length(result_jedi_sparse$bin), "\n\n")

# Show how smoothing prevents extreme WoE values
for (i in seq_along(result_jedi_sparse$bin)) {
  cat(sprintf(
    "  Bin %d: events=%d/%d, WoE=%.3f (smoothed)\n",
    i,
    result_jedi_sparse$count_pos[i],
    result_jedi_sparse$count[i],
    result_jedi_sparse$woe[i]
  ))
}

# Example 4: Violation detection and repair
set.seed(456)
n_obs_viol <- 1200

# Create feature with non-monotonic risk pattern
risk_categories <- c(
  "VeryLow", "Low", "MediumHigh", "Medium", # Intentional non-monotonic
  "High", "VeryHigh"
)
actual_risks <- c(0.02, 0.05, 0.20, 0.12, 0.25, 0.40) # MediumHigh > Medium

cat_feature_viol <- sample(risk_categories, n_obs_viol, replace = TRUE)
bin_target_viol <- sapply(cat_feature_viol, function(x) {
  rbinom(1, 1, actual_risks[which(risk_categories == x)])
})
```

```
result_jedi_viol <- ob_categorical_jedi(
  cat_feature_viol,
  bin_target_viol,
  min_bins = 3,
  max_bins = 5,
  max_iterations = 50
)

cat("\nViolation detection and repair:\n")
cat("  Original categories:", length(unique(cat_feature_viol)), "\n")
cat("  Final bins:", length(result_jedi_viol$bin), "\n")
cat("  Iterations to convergence:", result_jedi_viol$iterations, "\n")
cat("  Monotonicity achieved:", result_jedi_viol$converged, "\n\n")

# Check final WoE monotonicity
woe_diffs <- diff(result_jedi_viol$woe)
cat(
  "  WoE differences between bins:",
  paste(round(woe_diffs, 3), collapse = ", "), "\n"
)
cat("  All positive (monotonic):", all(woe_diffs >= -1e-6), "\n")

# Example 5: High cardinality performance
set.seed(321)
n_obs_hc <- 3000

# Simulate product categories (high cardinality)
products <- paste0("Product_", sprintf("%03d", 1:50))

cat_feature_hc <- sample(products, n_obs_hc, replace = TRUE)
bin_target_hc <- rbinom(n_obs_hc, 1, 0.08)

# Measure JEDI performance
time_jedi_hc <- system.time({
  result_jedi_hc <- ob_categorical_jedi(
    cat_feature_hc,
    bin_target_hc,
    min_bins = 4,
    max_bins = 7,
    max_n_prebins = 20,
    bin_cutoff = 0.02
  )
})

cat("\nHigh cardinality performance:\n")
cat("  Original categories:", length(unique(cat_feature_hc)), "\n")
cat("  Final bins:", length(result_jedi_hc$bin), "\n")
cat("  Execution time:", round(time_jedi_hc[3], 3), "seconds\n")
cat("  Total IV:", round(result_jedi_hc$total_iv, 4), "\n")
cat("  Converged:", result_jedi_hc$converged, "\n")

# Show merged categories
```

```
for (i in seq_along(result_jedi_hc$bin)) {
  n_merged <- length(strsplit(result_jedi_hc$bin[i], "%;%")[[1]])
  if (n_merged > 1) {
    cat(sprintf("  Bin %d: %d categories merged\n", i, n_merged))
  }
}

# Example 6: Convergence behavior
set.seed(555)
n_obs_conv <- 1000

education_levels <- c(
  "Elementary", "HighSchool", "Vocational",
  "Bachelor", "Master", "PhD"
)

cat_feature_conv <- sample(education_levels, n_obs_conv,
  replace = TRUE,
  prob = c(0.10, 0.30, 0.20, 0.25, 0.12, 0.03)
)
bin_target_conv <- rbinom(n_obs_conv, 1, 0.15)

# Test different convergence thresholds
thresholds <- c(1e-3, 1e-6, 1e-9)

for (thresh in thresholds) {
  result_conv <- ob_categorical_jedi(
    cat_feature_conv,
    bin_target_conv,
    min_bins = 2,
    max_bins = 4,
    convergence_threshold = thresh,
    max_iterations = 100
  )

  cat(sprintf("\nThreshold %.0e:\n", thresh))
  cat("  Final bins:", length(result_conv$bin), "\n")
  cat("  Total IV:", round(result_conv$total_iv, 4), "\n")
  cat("  Converged:", result_conv$converged, "\n")
  cat("  Iterations:", result_conv$iterations, "\n")
}

# Example 7: Missing value handling
set.seed(999)
cat_feature_na <- cat_feature
na_indices <- sample(n_obs, 75) # 5% missing
cat_feature_na[na_indices] <- NA

result_jedi_na <- ob_categorical_jedi(
  cat_feature_na,
  bin_target,
  min_bins = 3,
  max_bins = 5
```

```
)

# Locate NA bin
na_bin_idx <- grep("NA", result_jedi_na$bin)
if (length(na_bin_idx) > 0) {
  cat("\nMissing value treatment:\n")
  cat("  NA bin:", result_jedi_na$bin[na_bin_idx], "\n")
  cat("  NA count:", result_jedi_na$count[na_bin_idx], "\n")
  cat(
    "  NA event rate:",
    round(result_jedi_na$count_pos[na_bin_idx] /
      result_jedi_na$count[na_bin_idx], 3), "\n"
  )
  cat("  NA WoE:", round(result_jedi_na$woe[na_bin_idx], 3), "\n")
  cat("  NA IV contribution:", round(result_jedi_na$iv[na_bin_idx], 4), "\n")
}
```

---

ob_categorical_jedi_mwoe *Optimal Binning for Categorical Variables with Multinomial Target using JEDI-MWoE*

Description

Performs supervised discretization of categorical variables for multinomial classification problems using the Joint Entropy-Driven Information Maximization with Multinomial Weight of Evidence (JEDI-MWoE) algorithm. This advanced method extends traditional binning to handle multi-class targets through specialized information-theoretic measures and intelligent optimization strategies.

Usage

```
ob_categorical_jedi_mwoe(
  feature,
  target,
  min_bins = 3,
  max_bins = 5,
  bin_cutoff = 0.05,
  max_n_prebins = 20,
  bin_separator = "%;%",
  convergence_threshold = 1e-06,
  max_iterations = 1000
)
```

Arguments

feature	A character vector or factor representing the categorical predictor variable to be binned. Missing values are automatically converted to the special category "NA".

`target`	An integer vector representing the multinomial outcome variable with consecutive integer classes starting from 0 (e.g., 0, 1, 2, ...). Missing values are not permitted. Must contain at least 2 distinct classes.
`min_bins`	Integer. Minimum number of bins to produce. Must be >= 1. The algorithm will not merge below this threshold. Defaults to 3.
`max_bins`	Integer. Maximum number of bins to produce. Must be >= `min_bins`. The algorithm iteratively merges until this constraint is satisfied. Defaults to 5.
`bin_cutoff`	Numeric. Minimum proportion of total observations required for a category to remain separate during initialization. Categories below this threshold are pre-merged. Must be in (0, 1). Defaults to 0.05.
`max_n_prebins`	Integer. Maximum number of initial bins before the main optimization phase. Controls computational complexity for high-cardinality features. Must be >= `min_bins`. Defaults to 20.
`bin_separator`	Character string used to concatenate category names when multiple categories are merged into a single bin. Defaults to "%;%".
`convergence_threshold`	Numeric. Convergence tolerance based on Information Value change between iterations. Algorithm stops when $\max_c \lvert\Delta IV_c\rvert <$ `convergence_threshold` across all classes. Must be > 0. Defaults to 1e-6.
`max_iterations`	Integer. Maximum number of optimization iterations. Prevents infinite loops in edge cases. Must be > 0. Defaults to 1000.

Details

The JEDI-MWoE (Joint Entropy-Driven Information Maximization with Multinomial Weight of Evidence) algorithm extends traditional optimal binning to handle multinomial classification problems by computing class-specific information measures and optimizing joint information content across all target classes.

Algorithm Workflow:

1. Input validation and preprocessing (multinomial target verification)
2. Initial bin creation (one category per bin)
3. Rare category merging (frequencies < `bin_cutoff`)
4. Pre-bin limitation via statistical similarity merging
5. Main optimization loop with alternating strategies:
    - Jensen-Shannon divergence minimization for similar bin detection
    - Adjacent bin merging with minimal information loss
    - Class-wise monotonicity violation detection and repair
    - Convergence monitoring across all classes
6. Final constraint satisfaction (max_bins enforcement)
7. Laplace-smoothed metric computation

Multinomial Weight of Evidence (M-WoE):

For a bin B and class c, the Multinomial Weight of Evidence is:

$$M\text{-}WoE_{B,c} = \ln\left(\frac{P(c|B) + \alpha}{P(\neg c|B) + \alpha}\right)$$

where:

- $P(c|B) = \frac{n_{B,c}}{n_B}$ is the class probability in bin B
- $P(\neg c|B) = \frac{\sum_{k \neq c} n_{B,k}}{\sum_{k \neq c} n_k}$ is the combined probability of all other classes in bin B
- $\alpha = 0.5$ is the Laplace smoothing parameter

Information Value Extension:

The Information Value for class c in bin B is:

$$IV_{B,c} = (P(c|B) - P(\neg c|B)) \times M\text{-}WoE_{B,c}$$

Total IV for class c across all bins:

$$IV_c = \sum_B |IV_{B,c}|$$

Statistical Similarity Measure:

JEDI-MWoE uses Jensen-Shannon divergence to identify similar bins for merging:

$$JS(P_B, P_{B'}) = \frac{1}{2} \sum_{c=0}^{C-1} \left[P(c|B) \ln \frac{P(c|B)}{M(c)} + P(c|B') \ln \frac{P(c|B')}{M(c)} \right]$$

where $M(c) = \frac{1}{2}[P(c|B) + P(c|B')]$ is the average distribution.

Class-wise Monotonicity Enforcement:

For each class c, the algorithm enforces WoE monotonicity by detecting violations (peaks and valleys) and repairing them through strategic bin merges:

- **Peak**: $M\text{-}WoE_{i-1,c} < M\text{-}WoE_{i,c} > M\text{-}WoE_{i+1,c}$
- **Valley**: $M\text{-}WoE_{i-1,c} > M\text{-}WoE_{i,c} < M\text{-}WoE_{i+1,c}$

Violation severity is measured as:

$$severity_{i,c} = \max\{|M\text{-}WoE_{i,c} - M\text{-}WoE_{i-1,c}|, |M\text{-}WoE_{i,c} - M\text{-}WoE_{i+1,c}|\}$$

Alternating Optimization Strategies:

The algorithm alternates between two merging strategies to balance global similarity and local information preservation:

1. **Divergence-based**: Merge bins with minimum JS divergence
2. **IV-preserving**: Merge adjacent bins with minimum information loss

Laplace Smoothing:

To ensure numerical stability and prevent undefined logarithms, all probability estimates are smoothed with a Laplace prior:

$$P_{smooth}(c|B) = \frac{n_{B,c} + \alpha}{n_B + \alpha \cdot C}$$

where C is the number of classes and $\alpha = 0.5$.

Computational Complexity:

- Time: $O(k^2 \cdot C \cdot m)$ where k = bins, C = classes, m = iterations
- Space: $O(k^2 \cdot C)$ for M-WoE cache
- Cache hit rate typically > 60% for $k > 10$

Key Innovations:

- **Multinomial extension**: Generalizes WoE/IV to multi-class problems
- **Joint optimization**: Simultaneously optimizes across all classes
- **Alternating strategies**: Balances global similarity and local preservation
- **Class-wise monotonicity**: Enforces meaningful ordering for each class
- **Statistical similarity**: Uses Jensen-Shannon divergence for merging

Comparison with Binary Methods:

Aspect	Binary	Multinomial	Extension
Target Classes	2	C >= 2	One-vs-rest
WoE Definition	$\ln(p/n)$	$\ln(P(c)/P(\neg c))$	Class-specific
IV Aggregation	Sum	Per-class	Vector-valued
Similarity	Chi-square	Jensen-Shannon	Distribution-based
Monotonicity	Global	Per-class	Multi-constraint

Value

A list containing the multinomial binning results with the following components:

`id` Integer vector of bin identifiers (1-indexed)

`bin` Character vector of bin labels (merged category names)

`woe` Numeric matrix of Multinomial Weight of Evidence values with dimensions (bins × classes)

`iv` Numeric matrix of Information Value contributions with dimensions (bins × classes)

`count` Integer vector of total observations per bin

`class_counts` Integer matrix of observations per class per bin with dimensions (bins × classes)

`class_rates` Numeric matrix of class proportions per bin with dimensions (bins × classes)

`converged` Logical indicating algorithm convergence

`iterations` Integer count of optimization iterations performed

`n_classes` Integer number of target classes

`total_iv` Numeric vector of total Information Value per class

References

Siddiqi, N. (2006). *Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring*. Wiley.

Lin, J. (1991). Divergence measures based on the Shannon entropy. *IEEE Transactions on Information Theory*, 37(1), 145-151. doi:10.1109/18.61115

Cover, T. M., & Thomas, J. A. (2006). *Elements of Information Theory* (2nd ed.). Wiley-Interscience. doi:10.1002/047174882X

Navas-Palencia, G. (2020). Optimal binning: mathematical programming formulation and solution approach. *Expert Systems with Applications*, 158, 113508. doi:10.1016/j.eswa.2020.113508

Good, I. J. (1965). *The Estimation of Probabilities: An Essay on Modern Bayesian Methods*. MIT Press.

See Also

ob_categorical_jedi for binary target JEDI algorithm, ob_categorical_ivb for binary Information Value DP optimization, ob_categorical_dp for general dynamic programming binning

Examples

```
# Example 1: Basic multinomial JEDI-MWoE optimization
set.seed(42)
n_obs <- 1500

# Simulate customer segments with 3 risk categories
segments <- c("Premium", "Standard", "Basic", "Economy")
# Class probabilities: 0=LowRisk, 1=MediumRisk, 2=HighRisk
risk_probs <- list(
  Premium = c(0.80, 0.15, 0.05), # Mostly LowRisk
  Standard = c(0.40, 0.40, 0.20), # Balanced
  Basic = c(0.15, 0.35, 0.50), # Mostly HighRisk
  Economy = c(0.05, 0.20, 0.75) # Almost all HighRisk
)

cat_feature <- sample(segments, n_obs,
  replace = TRUE,
  prob = c(0.25, 0.35, 0.25, 0.15)
)

# Generate multinomial target (classes 0, 1, 2)
multinom_target <- sapply(cat_feature, function(segment) {
  probs <- risk_probs[[segment]]
  sample(0:2, 1, prob = probs)
})

# Apply JEDI-MWoE algorithm
result_mwoe <- ob_categorical_jedi_mwoe(
  cat_feature,
  multinom_target,
  min_bins = 2,
  max_bins = 3
```

```
)

# Display results
cat("Number of classes:", result_mwoe$n_classes, "\n")
cat("Number of bins:", length(result_mwoe$bin), "\n")
cat("Converged:", result_mwoe$converged, "\n")
cat("Iterations:", result_mwoe$iterations, "\n\n")

# Show bin details
for (i in seq_along(result_mwoe$bin)) {
  cat(sprintf("Bin %d (%s):\n", i, result_mwoe$bin[i]))
  cat("  Total count:", result_mwoe$count[i], "\n")
  cat("  Class counts:", result_mwoe$class_counts[i, ], "\n")
  cat("  Class rates:", round(result_mwoe$class_rates[i, ], 3), "\n")

  # Show WoE and IV for each class
  for (class in 0:(result_mwoe$n_classes - 1)) {
    cat(sprintf(
      "  Class %d: WoE=%.3f, IV=%.4f\n",
      class,
      result_mwoe$woe[i, class + 1], # R is 1-indexed
      result_mwoe$iv[i, class + 1]
    ))
  }
  cat("\n")
}

# Show total IV per class
cat("Total IV per class:\n")
for (class in 0:(result_mwoe$n_classes - 1)) {
  cat(sprintf("  Class %d: %.4f\n", class, result_mwoe$total_iv[class + 1]))
}

# Example 2: High-cardinality multinomial problem
set.seed(123)
n_obs_hc <- 2000

# Simulate product categories with 4 classes
products <- paste0("Product_", LETTERS[1:15])
cat_feature_hc <- sample(products, n_obs_hc, replace = TRUE)

# Generate 4-class target
multinom_target_hc <- sample(0:3, n_obs_hc,
  replace = TRUE,
  prob = c(0.3, 0.25, 0.25, 0.2)
)

result_mwoe_hc <- ob_categorical_jedi_mwoe(
  cat_feature_hc,
  multinom_target_hc,
  min_bins = 3,
  max_bins = 6,
  max_n_prebins = 15,
```

```
  bin_cutoff = 0.03
)

cat("\nHigh-cardinality example:\n")
cat("Original categories:", length(unique(cat_feature_hc)), "\n")
cat("Final bins:", length(result_mwoe_hc$bin), "\n")
cat("Classes:", result_mwoe_hc$n_classes, "\n")
cat("Converged:", result_mwoe_hc$converged, "\n\n")

# Show merged categories
for (i in seq_along(result_mwoe_hc$bin)) {
  n_merged <- length(strsplit(result_mwoe_hc$bin[i], "%;%")[[1]])
  if (n_merged > 1) {
    cat(sprintf("Bin %d: %d categories merged\n", i, n_merged))
  }
}

# Example 3: Laplace smoothing demonstration
set.seed(789)
n_obs_smooth <- 500

# Small sample with sparse categories
categories <- c("A", "B", "C", "D", "E")
cat_feature_smooth <- sample(categories, n_obs_smooth,
  replace = TRUE,
  prob = c(0.3, 0.25, 0.2, 0.15, 0.1)
)

# Generate 3-class target with class imbalance
multinom_target_smooth <- sample(0:2, n_obs_smooth,
  replace = TRUE,
  prob = c(0.6, 0.3, 0.1)
) # Class 0 dominant

result_mwoe_smooth <- ob_categorical_jedi_mwoe(
  cat_feature_smooth,
  multinom_target_smooth,
  min_bins = 2,
  max_bins = 4,
  bin_cutoff = 0.02
)

cat("\nLaplace smoothing demonstration:\n")
cat("Sample size:", n_obs_smooth, "\n")
cat("Classes:", result_mwoe_smooth$n_classes, "\n")
cat("Event distribution:", table(multinom_target_smooth), "\n\n")

# Show how smoothing prevents extreme values
for (i in seq_along(result_mwoe_smooth$bin)) {
  cat(sprintf("Bin %d (%s):\n", i, result_mwoe_smooth$bin[i]))
  cat("  Counts per class:", result_mwoe_smooth$class_counts[i, ], "\n")
  cat("  WoE values:", round(result_mwoe_smooth$woe[i, ], 3), "\n")
  cat("  Note: Extreme WoE values prevented by Laplace smoothing\n\n")
```

```
}

# Example 4: Class-wise monotonicity
set.seed(456)
n_obs_mono <- 1200

# Feature with predictable class patterns
education <- c("PhD", "Master", "Bachelor", "College", "HighSchool")
# Each education level has a preferred class
preferred_classes <- c(2, 1, 0, 1, 2) # PhD→High(2), Bachelor→Low(0), etc.

cat_feature_mono <- sample(education, n_obs_mono, replace = TRUE)

# Generate target with preferred class bias
multinom_target_mono <- sapply(cat_feature_mono, function(edu) {
  pref_class <- preferred_classes[which(education == edu)]
  # Create probability vector with preference
  probs <- rep(0.1, 3) # Base probability
  probs[pref_class + 1] <- 0.8 # Preferred class gets high probability
  sample(0:2, 1, prob = probs / sum(probs))
})

result_mwoe_mono <- ob_categorical_jedi_mwoe(
  cat_feature_mono,
  multinom_target_mono,
  min_bins = 3,
  max_bins = 5
)

cat("Class-wise monotonicity example:\n")
cat("Education levels:", length(education), "\n")
cat("Final bins:", length(result_mwoe_mono$bin), "\n")
cat("Iterations:", result_mwoe_mono$iterations, "\n\n")

# Check monotonicity for each class
for (class in 0:(result_mwoe_mono$n_classes - 1)) {
  woe_series <- result_mwoe_mono$woe[, class + 1]
  diffs <- diff(woe_series)
  is_mono <- all(diffs >= -1e-6) || all(diffs <= 1e-6)
  cat(sprintf("Class %d WoE monotonic: %s\n", class, is_mono))
  cat(sprintf("  WoE series: %s\n", paste(round(woe_series, 3), collapse = ", ")))
}

# Example 5: Missing value handling
set.seed(321)
cat_feature_na <- cat_feature
na_indices <- sample(n_obs, 75) # 5% missing
cat_feature_na[na_indices] <- NA

result_mwoe_na <- ob_categorical_jedi_mwoe(
  cat_feature_na,
  multinom_target,
  min_bins = 2,
```

```
  max_bins = 3
)

# Locate missing value bin
missing_bin_idx <- grep("NA", result_mwoe_na$bin)
if (length(missing_bin_idx) > 0) {
  cat("\nMissing value handling:\n")
  cat("Missing value bin:", result_mwoe_na$bin[missing_bin_idx], "\n")
  cat("Missing value count:", result_mwoe_na$count[missing_bin_idx], "\n")
  cat(
    "Class distribution in missing bin:",
    result_mwoe_na$class_counts[missing_bin_idx, ], "\n"
  )

  # Show class rates for missing bin
  for (class in 0:(result_mwoe_na$n_classes - 1)) {
    cat(sprintf(
      "  Class %d rate: %.3f\n", class,
      result_mwoe_na$class_rates[missing_bin_idx, class + 1]
    ))
  }
}

# Example 6: Convergence behavior
set.seed(555)
n_obs_conv <- 1000

departments <- c("Sales", "IT", "HR", "Finance", "Operations")
cat_feature_conv <- sample(departments, n_obs_conv, replace = TRUE)
multinom_target_conv <- sample(0:2, n_obs_conv, replace = TRUE)

# Test different convergence thresholds
thresholds <- c(1e-3, 1e-6, 1e-9)

for (thresh in thresholds) {
  result_conv <- ob_categorical_jedi_mwoe(
    cat_feature_conv,
    multinom_target_conv,
    min_bins = 2,
    max_bins = 4,
    convergence_threshold = thresh,
    max_iterations = 100
  )

  cat(sprintf("\nThreshold %.0e:\n", thresh))
  cat("  Final bins:", length(result_conv$bin), "\n")
  cat("  Converged:", result_conv$converged, "\n")
  cat("  Iterations:", result_conv$iterations, "\n")

  # Show total IV for each class
  cat("  Total IV per class:")
  for (class in 0:(result_conv$n_classes - 1)) {
    cat(sprintf(" %.4f", result_conv$total_iv[class + 1]))
```

```
  }
  cat("\n")
}
```

`ob_categorical_mba` *Optimal Binning for Categorical Variables using Monotonic Binning Algorithm*

Description

Performs supervised discretization of categorical variables using the Monotonic Binning Algorithm (MBA), which enforces strict Weight of Evidence monotonicity while optimizing Information Value through intelligent bin merging strategies. This implementation includes Bayesian smoothing for numerical stability and adaptive thresholding for robust monotonicity enforcement.

Usage

```
ob_categorical_mba(
  feature,
  target,
  min_bins = 3,
  max_bins = 5,
  bin_cutoff = 0.05,
  max_n_prebins = 20,
  bin_separator = "%;%",
  convergence_threshold = 1e-06,
  max_iterations = 1000
)
```

Arguments

`feature`	A character vector or factor representing the categorical predictor variable to be binned. Missing values are automatically converted to the category `"NA"`.
`target`	An integer vector of binary outcomes (0/1) corresponding to each observation in `feature`. Missing values are not permitted.
`min_bins`	Integer. Minimum number of bins to produce. Must be >= 2. The algorithm will not merge below this threshold. Defaults to 3.
`max_bins`	Integer. Maximum number of bins to produce. Must be >= `min_bins`. The algorithm reduces bins until this constraint is met. Defaults to 5.
`bin_cutoff`	Numeric. Minimum proportion of total observations required for a category to remain separate. Categories below this threshold are pre-merged with similar categories. Must be in (0, 1). Defaults to 0.05.
`max_n_prebins`	Integer. Maximum number of initial bins before the main optimization phase. Controls computational complexity. Must be >= `max_bins`. Defaults to 20.

`bin_separator` Character string used to concatenate category names when multiple categories are merged into a single bin. Defaults to "%;%".

`convergence_threshold` Numeric. Convergence tolerance based on Information Value change between iterations. Algorithm stops when $|\Delta IV| <$ `convergence_threshold`. Must be > 0. Defaults to 1e-6.

`max_iterations` Integer. Maximum number of optimization iterations. Prevents infinite loops. Must be > 0. Defaults to 1000.

Details

The Monotonic Binning Algorithm (MBA) implements a sophisticated approach to categorical binning that guarantees strict Weight of Evidence monotonicity through intelligent violation detection and repair mechanisms.

Algorithm Workflow:

1. Input validation and preprocessing
2. Initial bin creation (one category per bin)
3. Pre-binning limitation to `max_n_prebins`
4. Rare category merging (frequencies < `bin_cutoff`)
5. Bayesian-smoothed WoE calculation
6. Strict monotonicity enforcement with adaptive thresholds
7. IV-optimized bin merging to meet `max_bins` constraint
8. Final consistency verification

Monotonicity Enforcement:

MBA enforces strict monotonicity through an iterative repair process:

1. Sort bins by current WoE values
2. Calculate adaptive threshold: $\tau = \min(\epsilon, 0.01\bar{\Delta})$
3. Identify violations: $sign(WoE_i - WoE_{i-1}) \neq sign(WoE_{i+1} - WoE_i)$
4. Rank violations by severity: $s_i = |WoE_i - WoE_{i-1}| + |WoE_{i+1} - WoE_i|$
5. Repair most severe violations by merging adjacent bins
6. Repeat until no violations remain or `min_bins` reached

Bayesian Smoothing:

To ensure numerical stability and prevent overfitting, MBA applies Bayesian smoothing to WoE and IV calculations:

$$p_i' = \frac{n_{i,pos} + \alpha_p}{N_{pos} + \alpha_{total}}$$

$$n_i' = \frac{n_{i,neg} + \alpha_n}{N_{neg} + \alpha_{total}}$$

where priors are proportional to overall prevalence:

$$\alpha_p = \alpha_{total} \times \frac{N_{pos}}{N_{pos} + N_{neg}}$$

$$\alpha_n = \alpha_{total} - \alpha_p$$

with $\alpha_{total} = 1.0$ as the prior strength parameter.

Intelligent Bin Merging:

When reducing bins to meet the `max_bins` constraint, MBA employs an IV-loss minimization strategy:

$$\Delta IV_{i,j} = IV_i + IV_j - IV_{merged}(i,j)$$

The pair with minimum ΔIV is merged to preserve maximum predictive information.

Computational Complexity:

- Time: $O(k^2 \cdot m)$ where k = bins, m = iterations
- Space: $O(k^2)$ for IV loss cache
- Cache hit rate typically > 75% for $k > 10$

Key Features:

- **Guaranteed monotonicity**: Strict enforcement with adaptive thresholds
- **Bayesian regularization**: Robust to sparse bins and class imbalance
- **Intelligent merging**: Preserves maximum information during reduction
- **Adaptive thresholds**: Context-aware violation detection
- **Consistency verification**: Final integrity checks

Value

A list containing the binning results with the following components:

`id` Integer vector of bin identifiers (1-indexed)

`bin` Character vector of bin labels (merged category names)

`woe` Numeric vector of Weight of Evidence values per bin

`iv` Numeric vector of Information Value contribution per bin

`count` Integer vector of total observations per bin

`count_pos` Integer vector of positive cases (target=1) per bin

`count_neg` Integer vector of negative cases (target=0) per bin

`total_iv` Numeric total Information Value of the binning solution

`converged` Logical indicating algorithm convergence

`iterations` Integer count of optimization iterations performed

References

Mironchyk, P., & Tchistiakov, V. (2017). Monotone optimal binning algorithm for credit risk modeling. *SSRN Electronic Journal*. doi:10.2139/ssrn.2978774

Siddiqi, N. (2017). *Intelligent Credit Scoring: Building and Implementing Better Credit Risk Scorecards* (2nd ed.). Wiley.

Good, I. J. (1965). *The Estimation of Probabilities: An Essay on Modern Bayesian Methods*. MIT Press.

Zeng, G. (2014). A necessary condition for a good binning algorithm in credit scoring. *Applied Mathematical Sciences*, 8(65), 3229-3242.

See Also

ob_categorical_jedi for joint entropy-driven optimization, ob_categorical_dp for dynamic programming approach, ob_categorical_cm for ChiMerge-based binning

Examples

```
# Example 1: Basic monotonic binning with guaranteed WoE ordering
set.seed(42)
n_obs <- 1500

# Simulate risk ratings with natural monotonic relationship
ratings <- c("AAA", "AA", "A", "BBB", "BB", "B", "CCC")
default_probs <- c(0.01, 0.02, 0.05, 0.10, 0.20, 0.35, 0.50)

cat_feature <- sample(ratings, n_obs,
  replace = TRUE,
  prob = c(0.05, 0.10, 0.20, 0.25, 0.20, 0.15, 0.05)
)
bin_target <- sapply(cat_feature, function(x) {
  rbinom(1, 1, default_probs[which(ratings == x)])
})

# Apply MBA algorithm
result_mba <- ob_categorical_mba(
  cat_feature,
  bin_target,
  min_bins = 3,
  max_bins = 5
)

# Display results with guaranteed monotonic WoE
print(data.frame(
  Bin = result_mba$bin,
  WoE = round(result_mba$woe, 3),
  IV = round(result_mba$iv, 4),
  Count = result_mba$count,
  EventRate = round(result_mba$count_pos / result_mba$count, 3)
))
```

```
cat("\nMonotonicity check (WoE differences):\n")
woe_diffs <- diff(result_mba$woe)
cat("  Differences:", paste(round(woe_diffs, 4), collapse = ", "), "\n")
cat("  All positive (increasing):", all(woe_diffs >= -1e-10), "\n")
cat("  Total IV:", round(result_mba$total_iv, 4), "\n")
cat("  Converged:", result_mba$converged, "\n")

# Example 2: Comparison with non-monotonic methods
set.seed(123)
n_obs_comp <- 2000

sectors <- c("Tech", "Health", "Finance", "Manufacturing", "Retail")
cat_feature_comp <- sample(sectors, n_obs_comp, replace = TRUE)
bin_target_comp <- rbinom(n_obs_comp, 1, 0.15)

# MBA (strictly monotonic)
result_mba_comp <- ob_categorical_mba(
  cat_feature_comp, bin_target_comp,
  min_bins = 3, max_bins = 4
)

# Standard binning (may not be monotonic)
result_std_comp <- ob_categorical_cm(
  cat_feature_comp, bin_target_comp,
  min_bins = 3, max_bins = 4
)

cat("\nMonotonicity comparison:\n")
cat(
  "  MBA WoE differences:",
  paste(round(diff(result_mba_comp$woe), 4), collapse = ", "), "\n"
)
cat("  MBA monotonic:", all(diff(result_mba_comp$woe) >= -1e-10), "\n")
cat(
  "  Std WoE differences:",
  paste(round(diff(result_std_comp$woe), 4), collapse = ", "), "\n"
)
cat("  Std monotonic:", all(diff(result_std_comp$woe) >= -1e-10), "\n")

# Example 3: Bayesian smoothing with sparse data
set.seed(789)
n_obs_sparse <- 400

# Small sample with rare categories
categories <- c("A", "B", "C", "D", "E", "F")
cat_probs <- c(0.30, 0.25, 0.20, 0.15, 0.07, 0.03)

cat_feature_sparse <- sample(categories, n_obs_sparse,
  replace = TRUE,
  prob = cat_probs
)
bin_target_sparse <- rbinom(n_obs_sparse, 1, 0.08) # 8% event rate
```

```
result_mba_sparse <- ob_categorical_mba(
  cat_feature_sparse,
  bin_target_sparse,
  min_bins = 2,
  max_bins = 4,
  bin_cutoff = 0.02
)

cat("\nBayesian smoothing (sparse data):\n")
cat("  Sample size:", n_obs_sparse, "\n")
cat("  Events:", sum(bin_target_sparse), "\n")
cat("  Final bins:", length(result_mba_sparse$bin), "\n\n")

# Show how smoothing prevents extreme WoE values
for (i in seq_along(result_mba_sparse$bin)) {
  cat(sprintf(
    "  Bin %d: events=%d/%d, WoE=%.3f (smoothed)\n",
    i,
    result_mba_sparse$count_pos[i],
    result_mba_sparse$count[i],
    result_mba_sparse$woe[i]
  ))
}

# Example 4: High cardinality with pre-binning
set.seed(456)
n_obs_hc <- 3000

# Simulate ZIP codes (high cardinality)
zips <- paste0("ZIP_", sprintf("%04d", 1:50))

cat_feature_hc <- sample(zips, n_obs_hc, replace = TRUE)
bin_target_hc <- rbinom(n_obs_hc, 1, 0.12)

result_mba_hc <- ob_categorical_mba(
  cat_feature_hc,
  bin_target_hc,
  min_bins = 4,
  max_bins = 6,
  max_n_prebins = 20,
  bin_cutoff = 0.01
)

cat("\nHigh cardinality performance:\n")
cat("  Original categories:", length(unique(cat_feature_hc)), "\n")
cat("  Final bins:", length(result_mba_hc$bin), "\n")
cat(
  "  Largest merged bin contains:",
  max(sapply(strsplit(result_mba_hc$bin, "%;%"), length)), "categories\n"
)

# Verify monotonicity in high-cardinality case
woe_monotonic <- all(diff(result_mba_hc$woe) >= -1e-10)
```

```
cat("  WoE monotonic:", woe_monotonic, "\n")

# Example 5: Convergence behavior
set.seed(321)
n_obs_conv <- 1000

business_sizes <- c("Micro", "Small", "Medium", "Large", "Enterprise")
cat_feature_conv <- sample(business_sizes, n_obs_conv, replace = TRUE)
bin_target_conv <- rbinom(n_obs_conv, 1, 0.18)

# Test different convergence thresholds
thresholds <- c(1e-3, 1e-6, 1e-9)

for (thresh in thresholds) {
  result_conv <- ob_categorical_mba(
    cat_feature_conv,
    bin_target_conv,
    min_bins = 2,
    max_bins = 4,
    convergence_threshold = thresh,
    max_iterations = 50
  )

  cat(sprintf("\nThreshold %.0e:\n", thresh))
  cat("  Final bins:", length(result_conv$bin), "\n")
  cat("  Total IV:", round(result_conv$total_iv, 4), "\n")
  cat("  Converged:", result_conv$converged, "\n")
  cat("  Iterations:", result_conv$iterations, "\n")

  # Check monotonicity preservation
  monotonic <- all(diff(result_conv$woe) >= -1e-10)
  cat("  Monotonic:", monotonic, "\n")
}

# Example 6: Missing value handling
set.seed(555)
cat_feature_na <- cat_feature
na_indices <- sample(n_obs, 75) # 5% missing
cat_feature_na[na_indices] <- NA

result_mba_na <- ob_categorical_mba(
  cat_feature_na,
  bin_target,
  min_bins = 3,
  max_bins = 5
)

# Locate NA bin
na_bin_idx <- grep("NA", result_mba_na$bin)
if (length(na_bin_idx) > 0) {
  cat("\nMissing value treatment:\n")
  cat("  NA bin:", result_mba_na$bin[na_bin_idx], "\n")
  cat("  NA count:", result_mba_na$count[na_bin_idx], "\n")
```

```
  cat(
    "  NA event rate:",
    round(result_mba_na$count_pos[na_bin_idx] /
      result_mba_na$count[na_bin_idx], 3), "\n"
  )
  cat("  NA WoE:", round(result_mba_na$woe[na_bin_idx], 3), "\n")
  cat(
    "  Monotonicity preserved:",
    all(diff(result_mba_na$woe) >= -1e-10), "\n"
  )
}
```

---

ob_categorical_milp *Optimal Binning for Categorical Variables using Heuristic Algorithm*

Description

This function performs optimal binning for categorical variables using a heuristic merging approach to maximize Information Value (IV) while maintaining monotonic Weight of Evidence (WoE). Despite its name containing "MILP", it does NOT use Mixed Integer Linear Programming but rather a greedy optimization algorithm.

Usage

```
ob_categorical_milp(
  feature,
  target,
  min_bins = 3L,
  max_bins = 5L,
  bin_cutoff = 0.05,
  max_n_prebins = 20L,
  bin_separator = "%;%",
  convergence_threshold = 1e-06,
  max_iterations = 1000L
)
```

Arguments

feature	A character vector or factor representing the categorical predictor variable. Missing values (NA) will be converted to the string "NA" and treated as a separate category.
target	An integer vector containing binary outcome values (0 or 1). Must be the same length as feature. Cannot contain missing values.
min_bins	Integer. Minimum number of bins to create. Must be at least 2. Default is 3.
max_bins	Integer. Maximum number of bins to create. Must be greater than or equal to min_bins. Default is 5.

`bin_cutoff` Numeric. Minimum relative frequency threshold for individual categories. Categories with frequency below this proportion will be merged with others. Value must be between 0 and 1. Default is 0.05 (5%).

`max_n_prebins` Integer. Maximum number of initial bins before optimization. Used to control computational complexity when dealing with high-cardinality categorical variables. Default is 20.

`bin_separator` Character string used to separate category names when multiple categories are merged into a single bin. Default is "%;%".

`convergence_threshold`
Numeric. Threshold for determining algorithm convergence based on changes in total Information Value. Must be positive. Default is 1e-6.

`max_iterations` Integer. Maximum number of iterations for the optimization process. Must be positive. Default is 1000.

Details

The algorithm follows these steps:

1. Pre-binning: Each unique category becomes an initial bin
2. Rare category handling: Categories below `bin_cutoff` frequency are merged with similar ones
3. Bin reduction: Greedily merge bins to satisfy `min_bins` and `max_bins` constraints
4. Monotonicity enforcement: Ensures WoE is either consistently increasing or decreasing across bins
5. Optimization: Iteratively improves Information Value

Key features include:

- Bayesian smoothing to stabilize WoE estimates for sparse categories
- Automatic handling of missing values (converted to "NA" category)
- Monotonicity constraint enforcement
- Configurable minimum and maximum bin counts
- Rare category pooling based on relative frequency thresholds

Mathematical definitions:

$$WoE_i = \ln\left(\frac{p_i^{(1)}}{p_i^{(0)}}\right)$$

where $p_i^{(1)}$ and $p_i^{(0)}$ are the proportions of positive and negative cases in bin i, respectively, adjusted using Bayesian smoothing.

$$IV = \sum_{i=1}^{n}(p_i^{(1)} - p_i^{(0)}) \times WoE_i$$

Value

A list containing the results of the optimal binning procedure:

`id` Integer vector of bin identifiers (1 to n_bins)

`bin` Character vector of bin labels, which are combinations of original categories separated by `bin_separator`

`woe` Numeric vector of Weight of Evidence values for each bin

`iv` Numeric vector of Information Values for each bin

`count` Integer vector of total observations in each bin

`count_pos` Integer vector of positive outcomes in each bin

`count_neg` Integer vector of negative outcomes in each bin

`total_iv` Numeric scalar. Total Information Value across all bins

`converged` Logical. Whether the algorithm converged within the specified tolerance

`iterations` Integer. Number of iterations performed

Note

- Target variable must contain both 0 and 1 values.
- Empty strings in the feature vector are not allowed and will cause an error.
- For datasets with very few observations in either class (<5), warnings will be issued as results may be unstable.
- The algorithm uses a greedy heuristic approach, not true MILP optimization. For exact solutions, external solvers like Gurobi or CPLEX would be required.

Examples

```
# Generate sample data
set.seed(123)
n <- 1000
feature <- sample(letters[1:8], n, replace = TRUE)
target <- rbinom(n, 1, prob = ifelse(feature %in% c("a", "b"), 0.7, 0.3))

# Perform optimal binning
result <- ob_categorical_milp(feature, target)
print(result[c("bin", "woe", "iv", "count")])

# With custom parameters
result2 <- ob_categorical_milp(
  feature = feature,
  target = target,
  min_bins = 2,
  max_bins = 4,
  bin_cutoff = 0.03
)

# Handling missing values
feature_with_na <- feature
```

```
feature_with_na[sample(length(feature_with_na), 50)] <- NA
result3 <- ob_categorical_milp(feature_with_na, target)
```

`ob_categorical_mob` *Optimal Binning for Categorical Variables using Monotonic Optimal Binning (MOB)*

Description

This function performs optimal binning for categorical variables using the Monotonic Optimal Binning (MOB) algorithm. It creates bins that maintain monotonic Weight of Evidence (WoE) trends while maximizing Information Value.

Usage

```
ob_categorical_mob(
  feature,
  target,
  min_bins = 3L,
  max_bins = 5L,
  bin_cutoff = 0.05,
  max_n_prebins = 20L,
  bin_separator = "%;%",
  convergence_threshold = 1e-06,
  max_iterations = 1000L
)
```

Arguments

`feature`	A character vector or factor representing the categorical predictor variable. Missing values (NA) will be converted to the string "NA" and treated as a separate category.
`target`	An integer vector containing binary outcome values (0 or 1). Must be the same length as `feature`. Cannot contain missing values.
`min_bins`	Integer. Minimum number of bins to create. Must be at least 1. Default is 3.
`max_bins`	Integer. Maximum number of bins to create. Must be greater than or equal to `min_bins`. Default is 5.
`bin_cutoff`	Numeric. Minimum relative frequency threshold for individual categories. Categories with frequency below this proportion will be merged with others. Value must be between 0 and 1. Default is 0.05 (5%).
`max_n_prebins`	Integer. Maximum number of initial bins before optimization. Used to control computational complexity when dealing with high-cardinality categorical variables. Default is 20.
`bin_separator`	Character string used to separate category names when multiple categories are merged into a single bin. Default is "%;%".

`convergence_threshold`
: Numeric. Threshold for determining algorithm convergence based on changes in total Information Value. Must be positive. Default is 1e-6.

`max_iterations`
: Integer. Maximum number of iterations for the optimization process. Must be positive. Default is 1000.

Details

The MOB algorithm follows these steps:

1. Initial sorting: Categories are ordered by their individual WoE values
2. Rare category handling: Categories below `bin_cutoff` frequency are merged with similar ones
3. Pre-binning limitation: Reduces initial bins to `max_n_prebins` using similarity-based merging
4. Monotonicity enforcement: Ensures WoE is either consistently increasing or decreasing across bins
5. Bin count optimization: Adjusts to meet `min_bins`/`max_bins` constraints

Key features include:

- Automatic sorting of categories by WoE for initial structure
- Bayesian smoothing to stabilize WoE estimates for sparse categories
- Guaranteed monotonic WoE trend across final bins
- Configurable minimum and maximum bin counts
- Similarity-based merging for optimal bin combinations

Mathematical definitions:

$$WoE_i = \ln\left(\frac{p_i^{(1)}}{p_i^{(0)}}\right)$$

where $p_i^{(1)}$ and $p_i^{(0)}$ are the proportions of positive and negative cases in bin i, respectively, adjusted using Bayesian smoothing.

$$IV = \sum_{i=1}^{n}(p_i^{(1)} - p_i^{(0)}) \times WoE_i$$

Value

A list containing the results of the optimal binning procedure:

`id` Numeric vector of bin identifiers (1 to n_bins)

`bin` Character vector of bin labels, which are combinations of original categories separated by `bin_separator`

`woe` Numeric vector of Weight of Evidence values for each bin

`iv` Numeric vector of Information Values for each bin

`count` Integer vector of total observations in each bin

`count_pos` Integer vector of positive outcomes in each bin

`count_neg` Integer vector of negative outcomes in each bin

`total_iv` Numeric scalar. Total Information Value across all bins

`converged` Logical. Whether the algorithm converged within the specified tolerance

`iterations` Integer. Number of iterations performed

Note

- Target variable must contain both 0 and 1 values.
- Empty strings in the feature vector are not allowed and will cause an error.
- For datasets with very few observations in either class (<5), warnings will be issued as results may be unstable.
- The algorithm guarantees monotonic WoE across bins.
- When the number of unique categories is less than `max_bins`, each category will form its own bin.

Examples

```
# Generate sample data
set.seed(123)
n <- 1000
feature <- sample(letters[1:8], n, replace = TRUE)
target <- rbinom(n, 1, prob = ifelse(feature %in% c("a", "b"), 0.7, 0.3))

# Perform optimal binning
result <- ob_categorical_mob(feature, target)
print(result[c("bin", "woe", "iv", "count")])

# With custom parameters
result2 <- ob_categorical_mob(
  feature = feature,
  target = target,
  min_bins = 2,
  max_bins = 4,
  bin_cutoff = 0.03
)

# Handling missing values
feature_with_na <- feature
feature_with_na[sample(length(feature_with_na), 50)] <- NA
result3 <- ob_categorical_mob(feature_with_na, target)
```

ob_categorical_sab *Optimal Binning for Categorical Variables using Simulated Annealing*

Description

This function performs optimal binning for categorical variables using a Simulated Annealing (SA) optimization algorithm. It maximizes Information Value (IV) while maintaining monotonic Weight of Evidence (WoE) trends.

Usage

```
ob_categorical_sab(
  feature,
  target,
  min_bins = 3L,
  max_bins = 5L,
  bin_cutoff = 0.05,
  max_n_prebins = 20L,
  bin_separator = "%;%",
  initial_temperature = 1,
  cooling_rate = 0.995,
  max_iterations = 1000L,
  convergence_threshold = 1e-06,
  adaptive_cooling = TRUE
)
```

Arguments

feature	A character vector or factor representing the categorical predictor variable. Missing values (NA) will be converted to the string "NA" and treated as a separate category.
target	An integer vector containing binary outcome values (0 or 1). Must be the same length as feature. Cannot contain missing values.
min_bins	Integer. Minimum number of bins to create. Must be at least 2. Default is 3.
max_bins	Integer. Maximum number of bins to create. Must be greater than or equal to min_bins. Default is 5.
bin_cutoff	Numeric. Minimum relative frequency threshold for individual bins. Bins with frequency below this proportion will be penalized. Value must be between 0 and 1. Default is 0.05 (5%).
max_n_prebins	Integer. Maximum number of initial categories before optimization (not directly used in current implementation). Must be greater than or equal to max_bins. Default is 20.
bin_separator	Character string used to separate category names when multiple categories are merged into a single bin. Default is "%;%".

`initial_temperature`
: Numeric. Starting temperature for the simulated annealing algorithm. Higher values allow more exploration. Must be positive. Default is 1.0.

`cooling_rate`
: Numeric. Rate at which temperature decreases during optimization. Value must be between 0 and 1. Lower values lead to faster cooling. Default is 0.995.

`max_iterations`
: Integer. Maximum number of iterations for the optimization process. Must be positive. Default is 1000.

`convergence_threshold`
: Numeric. Threshold for determining algorithm convergence based on changes in Information Value. Must be positive. Default is 1e-6.

`adaptive_cooling`
: Logical. Whether to use adaptive cooling that modifies the cooling rate based on search progress. Default is TRUE.

Details

The SAB (Simulated Annealing Binning) algorithm follows these steps:

1. Initialization: Categories are initially assigned to bins using a k-means-like strategy based on event rates
2. Optimization: Simulated annealing explores different bin assignments to maximize IV
3. Neighborhood generation: Multiple strategies are employed to generate neighboring solutions (swaps, reassignments, event-rate based moves)
4. Acceptance criteria: New solutions are accepted based on the Metropolis criterion with adaptive temperature control
5. Monotonicity enforcement: Final solutions are adjusted to ensure monotonic WoE trends

Key features include:

- Global optimization approach using simulated annealing
- Adaptive cooling schedule to balance exploration and exploitation
- Multiple neighborhood generation strategies for better search
- Bayesian smoothing to stabilize WoE estimates for sparse categories
- Guaranteed monotonic WoE trend across final bins
- Configurable optimization parameters for fine-tuning

Mathematical definitions:

$$WoE_i = \ln\left(\frac{p_i^{(1)}}{p_i^{(0)}}\right)$$

where $p_i^{(1)}$ and $p_i^{(0)}$ are the proportions of positive and negative cases in bin i, respectively, adjusted using Bayesian smoothing.

$$IV = \sum_{i=1}^{n}(p_i^{(1)} - p_i^{(0)}) \times WoE_i$$

The acceptance probability in simulated annealing is:

$$P(accept) = \exp\left(\frac{IV_{new} - IV_{current}}{T}\right)$$

where T is the current temperature.

Value

A list containing the results of the optimal binning procedure:

`id` Numeric vector of bin identifiers (1 to n_bins)

`bin` Character vector of bin labels, which are combinations of original categories separated by `bin_separator`

`woe` Numeric vector of Weight of Evidence values for each bin

`iv` Numeric vector of Information Values for each bin

`count` Integer vector of total observations in each bin

`count_pos` Integer vector of positive outcomes in each bin

`count_neg` Integer vector of negative outcomes in each bin

`total_iv` Numeric scalar. Total Information Value across all bins

`converged` Logical. Whether the algorithm converged within the specified tolerance

`iterations` Integer. Number of iterations performed

Note

- Target variable must contain both 0 and 1 values.
- Empty strings in the feature vector are not allowed and will cause an error.
- For datasets with very few observations in either class (<5), warnings will be issued as results may be unstable.
- The algorithm uses global optimization which may require more computational time compared to heuristic approaches.
- When the number of unique categories is less than `max_bins`, each category will form its own bin.

Examples

```
# Generate sample data
set.seed(123)
n <- 1000
feature <- sample(letters[1:8], n, replace = TRUE)
target <- rbinom(n, 1, prob = ifelse(feature %in% c("a", "b"), 0.7, 0.3))

# Perform optimal binning
result <- ob_categorical_sab(feature, target)
print(result[c("bin", "woe", "iv", "count")])

# With custom parameters
```

```
result2 <- ob_categorical_sab(
  feature = feature,
  target = target,
  min_bins = 2,
  max_bins = 4,
  initial_temperature = 2.0,
  cooling_rate = 0.99
)

# Handling missing values
feature_with_na <- feature
feature_with_na[sample(length(feature_with_na), 50)] <- NA
result3 <- ob_categorical_sab(feature_with_na, target)
```

`ob_categorical_sblp` *Optimal Binning for Categorical Variables using SBLP*

Description

This function performs optimal binning for categorical variables using the Similarity-Based Logistic Partitioning (SBLP) algorithm. This approach combines logistic properties (sorting categories by event rate) with dynamic programming to find the optimal partition that maximizes Information Value (IV).

Usage

```
ob_categorical_sblp(
  feature,
  target,
  min_bins = 3L,
  max_bins = 5L,
  bin_cutoff = 0.05,
  max_n_prebins = 20L,
  convergence_threshold = 1e-06,
  max_iterations = 1000L,
  bin_separator = "%;%",
  alpha = 0.5
)
```

Arguments

`feature`	A character vector or factor representing the categorical predictor variable. Missing values (NA) will be converted to the string "NA" and treated as a separate category.
`target`	An integer vector containing binary outcome values (0 or 1). Must be the same length as `feature`. Cannot contain missing values.
`min_bins`	Integer. Minimum number of bins to create. Must be at least 2. Default is 3.

`max_bins`	Integer. Maximum number of bins to create. Must be greater than or equal to `min_bins`. Default is 5.
`bin_cutoff`	Numeric. Minimum relative frequency threshold for individual categories. Categories with frequency below this proportion will be merged with similar categories before the main optimization. Value must be between 0 and 1. Default is 0.05 (5%).
`max_n_prebins`	Integer. Maximum number of initial bins/groups allowed before the dynamic programming optimization. If the number of unique categories exceeds this, similar adjacent categories are pre-merged. Default is 20.
`convergence_threshold`	Numeric. Threshold for determining algorithm convergence based on changes in total Information Value. Default is 1e-6.
`max_iterations`	Integer. Maximum number of iterations for the optimization process. Default is 1000.
`bin_separator`	Character string used to separate category names when multiple categories are merged into a single bin. Default is "%;%".
`alpha`	Numeric. Laplace smoothing parameter added to counts to avoid division by zero and stabilize WoE calculations for sparse data. Must be non-negative. Default is 0.5.

Details

The SBLP algorithm follows these steps:

1. **Preprocessing**: Handling of missing values and calculation of initial statistics.
2. **Rare Category Consolidation**: Categories with frequency below `bin_cutoff` are merged with statistically similar categories based on their target rates.
3. **Sorting**: Unique categories (or merged groups) are sorted by their empirical event rate (probability of target=1).
4. **Dynamic Programming**: An optimal partitioning algorithm (similar to Jenks Natural Breaks but optimizing IV) is applied to the sorted sequence to determine the cutpoints that maximize the total IV.
5. **Refinement**: Post-processing ensures constraints like monotonicity and minimum bin size are met.

A key feature of this implementation is the use of **Laplace Smoothing** (controlled by the `alpha` parameter) to prevent infinite WoE values and stabilize estimates for categories with small counts.

Mathematical definitions with smoothing:

The smoothed event rate p_i for a bin is calculated as:

$$p_i = \frac{n_{pos} + \alpha}{n_{total} + 2\alpha}$$

The Weight of Evidence (WoE) is computed using smoothed proportions:

$$WoE_i = \ln\left(\frac{p_i^{(1)}}{p_i^{(0)}}\right)$$

where $p_i^{(1)}$ and $p_i^{(0)}$ are the smoothed distributions of positive and negative classes across bins.

Value

A list containing the results of the optimal binning procedure:

`id` Numeric vector of bin identifiers (1 to n_bins)

`bin` Character vector of bin labels, which are combinations of original categories separated by `bin_separator`

`woe` Numeric vector of Weight of Evidence values for each bin

`iv` Numeric vector of Information Values for each bin

`count` Integer vector of total observations in each bin

`count_pos` Integer vector of positive outcomes in each bin

`count_neg` Integer vector of negative outcomes in each bin

`rate` Numeric vector of the observed event rate in each bin

`total_iv` Numeric scalar. Total Information Value across all bins

`converged` Logical. Whether the algorithm converged

`iterations` Integer. Number of iterations performed

Note

- Target variable must contain both 0 and 1 values.
- Unlike heuristic methods, this algorithm uses Dynamic Programming which guarantees an optimal partition given the sorted order of categories.
- Monotonicity is generally enforced by the sorting step, but strictly checked and corrected in the final output.

Examples

```
# Generate sample data
set.seed(123)
n <- 1000
feature <- sample(letters[1:8], n, replace = TRUE)
# Create a relationship where 'a' and 'b' have high probability
target <- rbinom(n, 1, prob = ifelse(feature %in% c("a", "b"), 0.8, 0.2))

# Perform optimal binning
result <- ob_categorical_sblp(feature, target)
print(result[c("bin", "woe", "iv", "count")])

# Using a higher smoothing parameter (alpha)
result_smooth <- ob_categorical_sblp(
  feature = feature,
  target = target,
  alpha = 1.0
)

# Handling missing values
feature_with_na <- feature
feature_with_na[sample(length(feature_with_na), 50)] <- NA
```

```
result_na <- ob_categorical_sblp(feature_with_na, target)
```

---

ob_categorical_sketch *Optimal Binning for Categorical Variables using Sketch-based Algorithm*

Description

This function performs optimal binning for categorical variables using a Sketch-based algorithm designed for large-scale data processing. It employs probabilistic data structures (Count-Min Sketch) to efficiently estimate category frequencies and event rates, enabling near real-time binning on massive datasets.

Usage

```
ob_categorical_sketch(
  feature,
  target,
  min_bins = 3L,
  max_bins = 5L,
  bin_cutoff = 0.05,
  max_n_prebins = 20L,
  bin_separator = "%;%",
  convergence_threshold = 1e-06,
  max_iterations = 1000L,
  sketch_width = 2000L,
  sketch_depth = 5L
)
```

Arguments

feature	A character vector or factor representing the categorical predictor variable. Missing values (NA) will be converted to the string "NA" and treated as a separate category.
target	An integer vector containing binary outcome values (0 or 1). Must be the same length as feature. Cannot contain missing values.
min_bins	Integer. Minimum number of bins to create. Must be at least 2. Default is 3.
max_bins	Integer. Maximum number of bins to create. Must be greater than or equal to min_bins. Default is 5.
bin_cutoff	Numeric. Minimum relative frequency threshold for categories to be considered "heavy hitters". Categories below this proportion will be grouped together. Value must be between 0 and 1. Default is 0.05 (5%).
max_n_prebins	Integer. Maximum number of initial bins created during pre-binning phase. Controls early-stage complexity. Default is 20.

`bin_separator` Character string used to separate category names when multiple categories are merged into a single bin. Default is "%;%".

`convergence_threshold` Numeric. Threshold for determining algorithm convergence based on changes in total Information Value. Default is 1e-6.

`max_iterations` Integer. Maximum number of iterations for the optimization process. Default is 1000.

`sketch_width` Integer. Width of the Count-Min Sketch (number of counters per hash function). Larger values reduce estimation error but increase memory usage. Must be >= 100. Default is 2000.

`sketch_depth` Integer. Depth of the Count-Min Sketch (number of hash functions). Larger values reduce collision probability but increase computational overhead. Must be >= 3. Default is 5.

Details

The Sketch-based algorithm follows these steps:

1. **Frequency Estimation**: Uses Count-Min Sketch to approximate the frequency of each category in a single data pass.
2. **Heavy Hitter Detection**: Identifies frequently occurring categories (above a threshold defined by `bin_cutoff`) using sketch estimates.
3. **Pre-binning**: Creates initial bins from detected heavy categories, grouping rare categories separately.
4. **Optimization**: Applies iterative merging based on statistical divergence measures to optimize Information Value (IV) while respecting bin count constraints (`min_bins`, `max_bins`).
5. **Monotonicity Enforcement**: Ensures the final binning has monotonic Weight of Evidence (WoE).

Key advantages of this approach:

- **Memory Efficiency**: Uses sub-linear space complexity, independent of dataset size.
- **Speed**: Single-pass algorithm with constant-time updates.
- **Scalability**: Suitable for streaming data or datasets too large to fit in memory.
- **Approximation**: Trades perfect accuracy for significant gains in speed and memory usage.

Mathematical concepts:

The Count-Min Sketch uses multiple hash functions to map items to counters:

$$CMS[i][h_i(x)] += 1 \quad \forall i \in \{1, \ldots, d\}$$

where d is the sketch depth and w is the sketch width.

Frequency estimates are obtained by taking the minimum across all counters:

$$\hat{f}(x) = \min_i CMS[i][h_i(x)]$$

Statistical divergence between bins is measured using Jensen-Shannon divergence:

$$JSD(P||Q) = \frac{1}{2}\left[KL(P||M) + KL(Q||M)\right]$$

where $M = \frac{1}{2}(P + Q)$ and KL is the Kullback-Leibler divergence.

Laplace smoothing is applied to WoE and IV calculations:

$$p_{smoothed} = \frac{count + \alpha}{total + 2\alpha}$$

Value

A list containing the results of the optimal binning procedure:

`id` Numeric vector of bin identifiers (1 to n_bins)

`bin` Character vector of bin labels, which are combinations of original categories separated by `bin_separator`

`woe` Numeric vector of Weight of Evidence values for each bin

`iv` Numeric vector of Information Values for each bin

`count` Integer vector of total observations in each bin

`count_pos` Integer vector of positive outcomes in each bin

`count_neg` Integer vector of negative outcomes in each bin

`event_rate` Numeric vector of the observed event rate in each bin

`total_iv` Numeric scalar. Total Information Value across all bins

`converged` Logical. Whether the algorithm converged

`iterations` Integer. Number of iterations performed

Note

- Target variable must contain both 0 and 1 values.
- Due to the probabilistic nature of sketches, results may vary slightly between runs. For deterministic results, consider setting fixed random seeds in the underlying C++ code.
- Accuracy of frequency estimates depends on `sketch_width` and `sketch_depth`. Increase these parameters for higher precision at the cost of memory/computation.
- This algorithm is particularly beneficial when dealing with high-cardinality categorical features or streaming data scenarios.
- For small to medium datasets, deterministic algorithms like SBLP or MOB may provide more accurate results.

References

Cormode, G., & Muthukrishnan, S. (2005). An improved data stream summary: the count-min sketch and its applications. Journal of Algorithms, 55(1), 58-75.

Lin, J., & Keogh, E., Wei, L., & Lonardi, S. (2007). Experiencing SAX: a novel symbolic representation of time series. Data Mining and Knowledge Discovery, 15(2), 107-144.

Examples

```
# Generate sample data
set.seed(123)
n <- 10000
feature <- sample(letters, n, replace = TRUE, prob = c(rep(0.04, 13), rep(0.02, 13)))
# Create a relationship where early letters have higher probability
target_probs <- ifelse(as.numeric(factor(feature)) <= 10, 0.7, 0.3)
target <- rbinom(n, 1, prob = target_probs)

# Perform sketch-based optimal binning
result <- ob_categorical_sketch(feature, target)
print(result[c("bin", "woe", "iv", "count")])

# With custom sketch parameters for higher accuracy
result_high_acc <- ob_categorical_sketch(
  feature = feature,
  target = target,
  min_bins = 3,
  max_bins = 7,
  sketch_width = 4000,
  sketch_depth = 7
)

# Handling missing values
feature_with_na <- feature
feature_with_na[sample(length(feature_with_na), 200)] <- NA
result_na <- ob_categorical_sketch(feature_with_na, target)
```

---

ob_categorical_swb *Optimal Binning for Categorical Variables using Sliding Window Binning (SWB)*

Description

This function performs optimal binning for categorical variables using the Sliding Window Binning (SWB) algorithm. This approach combines initial grouping based on frequency thresholds with iterative optimization to achieve monotonic Weight of Evidence (WoE) while maximizing Information Value (IV).

Usage

```
ob_categorical_swb(
  feature,
  target,
  min_bins = 3L,
  max_bins = 5L,
  bin_cutoff = 0.05,
  max_n_prebins = 20L,
  bin_separator = "%;%",
```

```
  convergence_threshold = 1e-06,
  max_iterations = 1000L
)
```

Arguments

`feature`	A character vector or factor representing the categorical predictor variable. Missing values (NA) will be converted to the string "NA" and treated as a separate category.
`target`	An integer vector containing binary outcome values (0 or 1). Must be the same length as `feature`. Cannot contain missing values.
`min_bins`	Integer. Minimum number of bins to create. Must be at least 1. Default is 3.
`max_bins`	Integer. Maximum number of bins to create. Must be greater than or equal to `min_bins`. Default is 5.
`bin_cutoff`	Numeric. Minimum relative frequency threshold for individual categories. Categories with frequency below this proportion will be grouped together into a single "rare" bin. Value must be between 0 and 1. Default is 0.05 (5%).
`max_n_prebins`	Integer. Maximum number of initial bins created after the frequency-based grouping step. Used to control early-stage complexity. Default is 20.
`bin_separator`	Character string used to separate category names when multiple categories are merged into a single bin. Default is "%;%".
`convergence_threshold`	Numeric. Threshold for determining algorithm convergence based on changes in total Information Value between iterations. Default is 1e-6.
`max_iterations`	Integer. Maximum number of iterations for the optimization process. Default is 1000.

Details

The SWB algorithm follows these steps:

1. **Initialization**: Categories are initially grouped based on frequency thresholds (`bin_cutoff`), separating frequent categories from rare ones.
2. **Preprocessing**: Initial bins are sorted by their WoE values to establish a baseline ordering.
3. **Sliding Window Optimization**: An iterative process evaluates adjacent bin pairs and merges those that contribute least to the overall Information Value or violate monotonicity constraints.
4. **Constraint Enforcement**: The final binning respects the specified `min_bins` and `max_bins` limits while maintaining WoE monotonicity.

Key features of this implementation:

- **Frequency-based Pre-grouping**: Automatically identifies and groups rare categories to reduce dimensionality.
- **Statistical Similarity Measures**: Utilizes Jensen-Shannon divergence to determine optimal merge candidates.

- **Monotonicity Preservation**: Ensures final bins exhibit consistent WoE trends (either increasing or decreasing).
- **Laplace Smoothing**: Employs additive smoothing to prevent numerical instabilities in WoE/IV calculations.

Mathematical concepts:

Weight of Evidence (WoE) with Laplace smoothing:

$$WoE = \ln\left(\frac{(p_{pos} + \alpha)/(N_{pos} + 2\alpha)}{(p_{neg} + \alpha)/(N_{neg} + 2\alpha)}\right)$$

Information Value (IV):

$$IV = \left(\frac{p_{pos} + \alpha}{N_{pos} + 2\alpha} - \frac{p_{neg} + \alpha}{N_{neg} + 2\alpha}\right) \times WoE$$

where p_{pos} and p_{neg} are bin-level counts, N_{pos} and N_{neg} are dataset-level totals, and α is the smoothing parameter (default 0.5).

Jensen-Shannon Divergence between two bins:

$$JSD(P||Q) = \frac{1}{2}\left[KL(P||M) + KL(Q||M)\right]$$

where $M = \frac{1}{2}(P + Q)$ and KL represents Kullback-Leibler divergence.

Value

A list containing the results of the optimal binning procedure:

`id` Numeric vector of bin identifiers (1 to n_bins)

`bin` Character vector of bin labels, which are combinations of original categories separated by `bin_separator`

`woe` Numeric vector of Weight of Evidence values for each bin

`iv` Numeric vector of Information Values for each bin

`count` Integer vector of total observations in each bin

`count_pos` Integer vector of positive outcomes in each bin

`count_neg` Integer vector of negative outcomes in each bin

`event_rate` Numeric vector of the observed event rate in each bin

`total_iv` Numeric scalar. Total Information Value across all bins

`converged` Logical. Whether the algorithm converged within specified tolerances

`iterations` Integer. Number of iterations performed

Note

- Target variable must contain both 0 and 1 values.
- The algorithm prioritizes monotonicity over strict adherence to bin count limits when conflicts arise.
- For datasets with very few unique categories (< 3), each category forms its own bin without optimization.
- Rare category grouping helps stabilize WoE estimates for infrequent values.

Examples

```
# Generate sample data with varying category frequencies
set.seed(456)
n <- 5000
# Create categories with power-law frequency distribution
categories <- c(
  rep("A", 1500), rep("B", 1000), rep("C", 800),
  rep("D", 500), rep("E", 300), rep("F", 200),
  sample(letters[7:26], 700, replace = TRUE)
)
feature <- sample(categories, n, replace = TRUE)
# Create target with dependency on top categories
target_probs <- ifelse(feature %in% c("A", "B"), 0.7,
  ifelse(feature %in% c("C", "D"), 0.5, 0.3)
)
target <- rbinom(n, 1, prob = target_probs)

# Perform sliding window binning
result <- ob_categorical_swb(feature, target)
print(result[c("bin", "woe", "iv", "count")])

# With stricter bin limits
result_strict <- ob_categorical_swb(
  feature = feature,
  target = target,
  min_bins = 4,
  max_bins = 6
)

# Handling missing values
feature_with_na <- feature
feature_with_na[sample(length(feature_with_na), 100)] <- NA
result_na <- ob_categorical_swb(feature_with_na, target)
```

---

ob_categorical_udt *Optimal Binning for Categorical Variables using a User-Defined Technique (UDT)*

Description

This function performs optimal binning for categorical variables using a User-Defined Technique (UDT) that combines frequency-based grouping with statistical similarity measures to create meaningful bins for predictive modeling.

Usage

```
ob_categorical_udt(
  feature,
```

```
  target,
  min_bins = 3L,
  max_bins = 5L,
  bin_cutoff = 0.05,
  max_n_prebins = 20L,
  bin_separator = "%;%",
  convergence_threshold = 1e-06,
  max_iterations = 1000L
)
```

Arguments

`feature`	A character vector or factor representing the categorical predictor variable. Missing values (NA) will be converted to the string "NA" and treated as a separate category.
`target`	An integer vector containing binary outcome values (0 or 1). Must be the same length as `feature`. Cannot contain missing values.
`min_bins`	Integer. Minimum number of bins to create. Must be at least 1. Default is 3.
`max_bins`	Integer. Maximum number of bins to create. Must be greater than or equal to `min_bins`. Default is 5.
`bin_cutoff`	Numeric. Minimum relative frequency threshold for individual categories. Categories with frequency below this proportion will be merged into a collective "rare" bin before optimization. Value must be between 0 and 1. Default is 0.05 (5%).
`max_n_prebins`	Integer. Upper limit on initial bins after frequency filtering. Controls computational complexity in early stages. Default is 20.
`bin_separator`	Character string used to separate category names when multiple categories are combined into a single bin. Default is "%;%".
`convergence_threshold`	Numeric. Threshold for determining algorithm convergence based on relative changes in total Information Value. Default is 1e-6.
`max_iterations`	Integer. Maximum number of iterations permitted for the optimization routine. Default is 1000.

Details

The UDT algorithm follows these steps:

1. **Initialization**: Each unique category is initially placed in its own bin.
2. **Frequency Filtering**: Categories below the `bin_cutoff` frequency threshold are grouped into a single "rare" bin.
3. **Iterative Optimization**: Bins are progressively merged based on statistical similarity (measured by Jensen-Shannon divergence) until the desired number of bins (`max_bins`) is achieved.
4. **Monotonicity Enforcement**: Final bins are sorted by Weight of Evidence to ensure consistent trends.

Key characteristics of this implementation:

- **Flexible Framework**: Designed as a customizable foundation for categorical binning approaches.
- **Statistical Rigor**: Uses information-theoretic measures to guide bin combination decisions.
- **Robust Estimation**: Implements Laplace smoothing to ensure stable WoE/IV calculations even with sparse data.
- **Efficiency Focus**: Employs targeted merging strategies to minimize computational overhead.

Mathematical foundations:

Laplace-smoothed probability estimates:

$$p_{smoothed} = \frac{count + \alpha}{total + 2\alpha}$$

Weight of Evidence calculation:

$$WoE = \ln\left(\frac{p_{pos,smoothed}}{p_{neg,smoothed}}\right)$$

Information Value computation:

$$IV = (p_{pos,smoothed} - p_{neg,smoothed}) \times WoE$$

Jensen-Shannon divergence between bins:

$$JSD(P||Q) = \frac{1}{2}[KL(P||M) + KL(Q||M)]$$

where $M = \frac{1}{2}(P + Q)$ and KL denotes Kullback-Leibler divergence.

Value

A list containing the results of the optimal binning procedure:

`id` Numeric vector of bin identifiers (1 to n_bins)

`bin` Character vector of bin labels, which are combinations of original categories separated by `bin_separator`

`woe` Numeric vector of Weight of Evidence values for each bin

`iv` Numeric vector of Information Values for each bin

`count` Integer vector of total observations in each bin

`count_pos` Integer vector of positive outcomes in each bin

`count_neg` Integer vector of negative outcomes in each bin

`event_rate` Numeric vector of the observed event rate in each bin

`total_iv` Numeric scalar. Total Information Value across all bins

`converged` Logical. Whether the algorithm converged

`iterations` Integer. Number of iterations executed

Note

- Target variable must contain both 0 and 1 values.
- For datasets with 1 or 2 unique categories, no optimization occurs beyond basic WoE/IV calculation.
- The algorithm does not perform bin splitting; it only merges existing bins to respect `max_bins`.
- Rare category pooling improves stability of WoE estimates for infrequent values.

Examples

```
# Generate sample data with skewed category distribution
set.seed(789)
n <- 3000
# Power-law distributed categories
categories <- c(
  rep("X1", 1200), rep("X2", 800), rep("X3", 400),
  sample(LETTERS[4:20], 600, replace = TRUE)
)
feature <- sample(categories, n, replace = TRUE)
# Target probabilities based on category importance
probs <- ifelse(grepl("X", feature), 0.7,
  ifelse(grepl("[A-C]", feature), 0.5, 0.3)
)
target <- rbinom(n, 1, prob = probs)

# Perform user-defined technique binning
result <- ob_categorical_udt(feature, target)
print(result[c("bin", "woe", "iv", "count")])

# Adjust parameters for finer control
result_custom <- ob_categorical_udt(
  feature = feature,
  target = target,
  min_bins = 2,
  max_bins = 7,
  bin_cutoff = 0.03
)

# Handling missing values
feature_with_na <- feature
feature_with_na[sample(length(feature_with_na), 150)] <- NA
result_na <- ob_categorical_udt(feature_with_na, target)
```

---

ob_cutpoints_cat *Binning Categorical Variables using Custom Cutpoints*

Description

This function applies user-defined binning to a categorical variable by grouping specified categories into bins and calculating Weight of Evidence (WoE) and Information Value (IV) for each bin.

Usage

```
ob_cutpoints_cat(feature, target, cutpoints)
```

Arguments

feature	A character vector or factor representing the categorical predictor variable.
target	An integer vector containing binary outcome values (0 or 1). Must be the same length as feature.
cutpoints	A character vector where each element defines a bin by concatenating the original category names with "+" as separator.

Details

The function takes a character vector defining how categories should be grouped. Each element in the cutpoints vector defines one bin by listing the original categories that should be merged, separated by "+" signs.

For example, if you want to create two bins from categories "A", "B", "C", "D":

- Bin 1: "A+B"
- Bin 2: "C+D"

cutpoints still uses "+" as the input separator (simple to type, e.g. "A+B"), but result$woebin is built so it can be handed straight back to ob_apply_woe_cat with its defaults – ob_apply_woe_cat(result$woebin, new_feature) – and get the same WoE this function itself assigned. Before 1.13.1 the emitted bin labels echoed the "+"-joined input verbatim and carried no id column, so a round trip through ob_apply_woe_cat's default bin_separator = "%;%" matched no category and silently fell back to a "Special"/NA bin for every observation.

Value

A list containing:

woefeature Numeric vector of WoE values corresponding to each observation in the input feature

woebin Data frame with one row per bin containing:

- id: Sequential bin identifier
- bin: The bin definition – original categories joined by "%;%" (**not** the "+" used in the cutpoints input; see Details), matching the separator ob_apply_woe_cat defaults to and every ob_categorical_*() algorithm in the main pipeline emits
- count: Total number of observations in the bin
- count_pos: Number of positive outcomes (target=1) in the bin
- count_neg: Number of negative outcomes (target=0) in the bin
- woe: Weight of Evidence for the bin
- iv: Information Value contribution of the bin

Note

- Target variable must contain only 0 and 1 values.
- Every unique category in `feature` must be included in exactly one bin definition in `cutpoints`.
- Categories not mentioned in `cutpoints` will be assigned to bin 0 (which may lead to unexpected results).

Examples

```
# Sample data
feature <- c("A", "B", "C", "D", "A", "B", "C", "D")
target <- c(1, 0, 1, 0, 1, 1, 0, 0)

# Define custom bins: (A,B) and (C,D)
cutpoints <- c("A+B", "C+D")

# Apply binning
result <- ob_cutpoints_cat(feature, target, cutpoints)

# View bin statistics
print(result$woebin)

# View WoE-transformed feature
print(result$woefeature)

# Round-trip through ob_apply_woe_cat() with its defaults
woe_new <- ob_apply_woe_cat(result$woebin, feature)
stopifnot(isTRUE(all.equal(woe_new$woe, result$woefeature)))
```

---

ob_cutpoints_num *Binning Numerical Variables using Custom Cutpoints*

Description

This function applies user-defined binning to a numerical variable by using specified cutpoints to create intervals and calculates Weight of Evidence (WoE) and Information Value (IV) for each interval bin.

Usage

```
ob_cutpoints_num(feature, target, cutpoints)
```

Arguments

`feature`	A numeric vector representing the continuous predictor variable.
`target`	An integer vector containing binary outcome values (0 or 1). Must be the same length as `feature`.
`cutpoints`	A numeric vector of cutpoints that define bin boundaries. These will be automatically sorted in ascending order.

Details

The function takes a numeric vector of cutpoints that define the boundaries between bins. For `n` cutpoints, `n+1` bins are created, each right-closed – $(a, b]$ – matching `ob_apply_woe_num`'s default (`include_upper_bound = TRUE`) and the convention `obwoe`/`obwoe_apply` use throughout the package:

- Bin 1: $(-\infty, cutpoint_1]$
- Bin 2: $(cutpoint_1, cutpoint_2]$
- ...
- Bin n+1: $(cutpoint_n, +\infty)$

`result` does not have the exact shape `ob_apply_woe_num` expects as `obresults` (which needs top-level `cutpoints`, `woe` and `id`), because `woebin` also carries `count`/ `iv` columns. To round-trip, reassemble it: `list(cutpoints = result$cutpoints, woe = result$woebin$woe, id = result$id, bin = result$woebin$bin)`. Before 1.13.1, `result` carried no `id` or top-level `cutpoints` at all (so a round trip through `ob_apply_woe_num` was not even possible without reconstructing them by hand), and the bin boundaries were left-closed $[a, b)$, the opposite of `ob_apply_woe_num`'s `include_upper_bound = TRUE` default – a value that landed exactly on a cutpoint got a different, often sign-flipped, WoE depending on whether it went through the fit or the apply side.

Value

A list containing:

`woefeature` Numeric vector of WoE values corresponding to each observation in the input `feature`

`woebin` Data frame with one row per bin containing:

- `id`: Sequential bin identifier
- `bin`: The bin interval notation (e.g., "(10.00;20.00]")
- `count`: Total number of observations in the bin
- `count_pos`: Number of positive outcomes (target=1) in the bin
- `count_neg`: Number of negative outcomes (target=0) in the bin
- `woe`: Weight of Evidence for the bin
- `iv`: Information Value contribution of the bin

`cutpoints` The sorted numeric cutpoints, exposed at the top level so this result can be reassembled into `ob_apply_woe_num`'s `obresults` argument (see Details)

`id` Sequential bin identifier, same as `woebin$id`; duplicated at the top level for the same reason as `cutpoints`

Note

- Target variable must contain only 0 and 1 values.
- Cutpoints are sorted automatically in ascending order.
- Interval notation uses "(" for exclusive and "]" for inclusive bounds – a value exactly equal to a cutpoint falls in the bin that *ends* at that cutpoint.
- Infinite values in feature are handled appropriately.

Examples

```
# Sample data
feature <- c(5, 15, 25, 35, 45, 55, 65, 75)
target <- c(0, 0, 1, 1, 1, 1, 0, 0)

# Define custom cutpoints
cutpoints <- c(30, 60)

# Apply binning
result <- ob_cutpoints_num(feature, target, cutpoints)

# View bin statistics
print(result$woebin)

# View WoE-transformed feature
print(result$woefeature)

# Round-trip through ob_apply_woe_num() with its defaults
apply_obj <- list(
  cutpoints = result$cutpoints, woe = result$woebin$woe,
  id = result$id, bin = result$woebin$bin
)
woe_new <- ob_apply_woe_num(apply_obj, feature)
stopifnot(isTRUE(all.equal(woe_new$woe, result$woefeature)))
```

---

ob_numerical_bb *Optimal Binning for Numerical Variables using Branch and Bound Algorithm*

Description

Performs supervised discretization of continuous numerical variables using a Branch and Bound-style approach. This algorithm optimally creates bins based on the relationship with a binary target variable, maximizing Information Value (IV) while optionally enforcing monotonicity in Weight of Evidence (WoE).

Usage

```
ob_numerical_bb(
  feature,
  target,
  min_bins = 3,
  max_bins = 5,
  bin_cutoff = 0.05,
  max_n_prebins = 20,
  is_monotonic = TRUE,
  convergence_threshold = 1e-06,
  max_iterations = 1000
)
```

Arguments

feature	A numeric vector representing the continuous predictor variable to be binned. NA values are handled by exclusion during the pre-binning phase.
target	An integer vector of binary outcomes (0/1) corresponding to each observation in feature. Must have the same length as feature.
min_bins	Integer. The minimum number of bins to produce. Must be ≥ 2. Defaults to 3.
max_bins	Integer. The maximum number of bins to produce. Must be $\geq$ min_bins. Defaults to 5.
bin_cutoff	Numeric. The minimum fraction of total observations required for a bin to be considered valid. Bins with frequency < bin_cutoff will be merged with neighbors. Value must be in (0, 1). Defaults to 0.05.
max_n_prebins	Integer. The number of initial quantiles to generate during the pre-binning phase. Higher values provide more granular starting points but increase computation time. Must be $\geq$ min_bins. Defaults to 20.
is_monotonic	Logical. If TRUE, the algorithm enforces a strict monotonic relationship (increasing or decreasing) between the bin indices and their WoE values. This makes the variable more interpretable for linear models. Defaults to TRUE.
convergence_threshold	Numeric. The threshold for the change in total IV to determine convergence during the iterative merging process. Defaults to 1e-6.
max_iterations	Integer. Safety limit for the maximum number of merging iterations. Defaults to 1000.

Details

The algorithm proceeds in several distinct phases to ensure stability and optimality:

1. **Pre-binning:** The numerical feature is initially discretized into max_n_prebins using quantiles. This handles outliers and provides a granular starting point.
2. **Rare Bin Management:** Bins containing fewer observations than the threshold defined by bin_cutoff are iteratively merged with their nearest neighbors to ensure statistical robustness.
3. **Monotonicity Enforcement (Optional):** If is_monotonic = TRUE, the algorithm checks if the WoE trend is strictly increasing or decreasing. If not, it simulates merges in both directions to find the path that preserves the maximum possible Information Value while satisfying the monotonicity constraint.
4. **Optimization Phase:** The algorithm iteratively merges adjacent bins that have the lowest contribution to the total Information Value (IV). This process continues until the number of bins is reduced to max_bins or the change in IV falls below convergence_threshold.

Information Value (IV) Interpretation:

- < 0.02: Not predictive
- 0.02 to 0.1: Weak predictive power
- 0.1 to 0.3: Medium predictive power
- 0.3 to 0.5: Strong predictive power
- > 0.5: Suspiciously high (check for leakage)

Value

A list containing the binning results:

- `id`: Integer vector of bin identifiers (1 to k).
- `bin`: Character vector of bin labels in interval notation (e.g., `"(0.5;1.2]"`).
- `woe`: Numeric vector of Weight of Evidence for each bin.
- `iv`: Numeric vector of Information Value contribution per bin.
- `count`: Integer vector of total observations per bin.
- `count_pos`: Integer vector of positive cases (target=1) per bin.
- `count_neg`: Integer vector of negative cases (target=0) per bin.
- `cutpoints`: Numeric vector of upper boundaries for the bins (excluding Inf).
- `converged`: Logical indicating if the algorithm converged properly.
- `iterations`: Integer count of iterations performed.
- `total_iv`: The total Information Value of the binned variable.

Examples

```
# Example: Binning a variable with a sigmoid relationship to target
set.seed(123)
n <- 1000
# Generate feature
feature <- rnorm(n)

# Generate target based on logistic probability
prob <- 1 / (1 + exp(-2 * feature))
target <- rbinom(n, 1, prob)

# Perform Optimal Binning
result <- ob_numerical_bb(feature, target,
  min_bins = 3,
  max_bins = 5,
  is_monotonic = TRUE
)

# Check results
print(data.frame(
  Bin = result$bin,
  Count = result$count,
  WoE = round(result$woe, 4),
  IV = round(result$iv, 4)
))

cat("Total IV:", result$total_iv, "\n")
```

`ob_numerical_cm` *Optimal Binning for Numerical Variables using Enhanced ChiMerge Algorithm*

Description

Performs supervised discretization of continuous numerical variables using the ChiMerge algorithm (Kerber, 1992) or the Chi2 algorithm (Liu & Setiono, 1995). This function merges adjacent bins based on Chi-square statistics to maximize the discrimination of the binary target variable while ensuring monotonicity and statistical robustness.

Usage

```
ob_numerical_cm(
  feature,
  target,
  min_bins = 3,
  max_bins = 5,
  bin_cutoff = 0.05,
  max_n_prebins = 20,
  convergence_threshold = 1e-06,
  max_iterations = 1000,
  init_method = "equal_frequency",
  chi_merge_threshold = 0.05,
  use_chi2_algorithm = FALSE
)
```

Arguments

`feature`	A numeric vector representing the continuous predictor variable. Missing values (NA) are not supported and should be handled before binning.
`target`	An integer vector of binary outcomes (0/1) corresponding to each observation in `feature`. Must have the same length as `feature`.
`min_bins`	Integer. The minimum number of bins to produce. Must be ≥ 2. Defaults to 3.
`max_bins`	Integer. The maximum number of bins to produce. Must be $\geq$ `min_bins`. Defaults to 5.
`bin_cutoff`	Numeric. The minimum fraction of total observations required for a bin to be considered valid. Bins with frequency < `bin_cutoff` will be merged. Value must be in (0, 1). Defaults to 0.05.
`max_n_prebins`	Integer. The number of initial bins created during the pre-binning phase before the merging process begins. Higher values provide more granular starting points. Must be $\geq$ `max_bins`. Defaults to 20.
`convergence_threshold`	Numeric. The threshold for the change in total IV to determine convergence during the iterative merging process. Defaults to 1e-6.

`max_iterations` Integer. Safety limit for the maximum number of merging iterations. Defaults to 1000.

`init_method` Character string specifying the initialization method. Options are `"equal_frequency"` (quantile-based) or `"equal_width"`. Defaults to `"equal_frequency"`.

`chi_merge_threshold`
Numeric. The significance level (α) for the Chi-square test. Pairs of bins with a p-value > `chi_merge_threshold` are candidates for merging. Defaults to 0.05.

`use_chi2_algorithm`
Logical. If `TRUE`, uses the Chi2 algorithm variant which performs multi-phase merging with decreasing significance levels (0.5, 0.1, 0.05, 0.01, ...). This is often more robust for noisy data. Defaults to `FALSE`.

Details

The function implements two major discretization strategies:

1. **Standard ChiMerge:**
   - Initializes bins using `init_method`.
   - Iteratively merges adjacent bins with the lowest χ^2 statistic.
   - Merging continues until all adjacent pairs have a p-value less than `chi_merge_threshold` or the number of bins reaches `max_bins`.
2. **Chi2 Algorithm:**
   - Activated when `use_chi2_algorithm = TRUE`.
   - Performs multiple passes with decreasing significance levels ($0.5 \rightarrow 0.001$) to automatically select the optimal significance threshold.
   - Checks for inconsistency rates in the data during the process.

Both methods include post-processing steps to enforce:

- **Minimum Bin Size:** Merging rare bins smaller than `bin_cutoff`.
- **Monotonicity:** Ensuring WoE trend is strictly increasing or decreasing to improve model interpretability.

Value

A list containing the binning results:

- `id`: Integer vector of bin identifiers (1 to k).
- `bin`: Character vector of bin labels in interval notation.
- `woe`: Numeric vector of Weight of Evidence for each bin.
- `iv`: Numeric vector of Information Value contribution per bin.
- `count`: Integer vector of total observations per bin.
- `count_pos`: Integer vector of positive cases (target=1).
- `count_neg`: Integer vector of negative cases (target=0).
- `cutpoints`: Numeric vector of upper boundaries (excluding Inf).

- converged: Logical indicating if the algorithm converged.
- iterations: Integer count of iterations performed.
- total_iv: The total Information Value of the binned variable.
- algorithm: String identifying the algorithm used ("ChiMerge" or "Chi2").
- monotonic: Logical indicating if the final WoE trend is monotonic.

References

Kerber, R. (1992). ChiMerge: Discretization of numeric attributes. *Proceedings of the Tenth National Conference on Artificial Intelligence*, 123-128.

Liu, H., & Setiono, R. (1995). Chi2: Feature selection and discretization of numeric attributes. *Tools with Artificial Intelligence*, 388-391.

Examples

```
# Example 1: Standard ChiMerge
set.seed(123)
feature <- rnorm(1000)
# Create a target with a relationship to the feature
target <- rbinom(1000, 1, plogis(2 * feature))

res_cm <- ob_numerical_cm(feature, target,
  min_bins = 3,
  max_bins = 6,
  init_method = "equal_frequency"
)

print(res_cm$bin)
print(res_cm$iv)

# Example 2: Using the Chi2 Algorithm variant
res_chi2 <- ob_numerical_cm(feature, target,
  min_bins = 3,
  max_bins = 6,
  use_chi2_algorithm = TRUE
)

cat("Total IV (ChiMerge):", res_cm$total_iv, "\n")
cat("Total IV (Chi2):", res_chi2$total_iv, "\n")
```

---

ob_numerical_dmiv *Optimal Binning using Metric Divergence Measures (Zeng, 2013)*

Description

Performs supervised discretization of continuous numerical variables using the theoretical framework proposed by Zeng (2013). This method creates bins that maximize a specified divergence measure (e.g., Kullback-Leibler, Hellinger) between the distributions of positive and negative cases, effectively maximizing the Information Value (IV) or other discriminatory statistics.

Usage

```
ob_numerical_dmiv(
  feature,
  target,
  min_bins = 3,
  max_bins = 5,
  bin_cutoff = 0.05,
  max_n_prebins = 20,
  is_monotonic = TRUE,
  convergence_threshold = 1e-06,
  max_iterations = 1000,
  bin_method = c("woe1", "woe"),
  divergence_method = c("l2", "he", "kl", "tr", "klj", "sc", "js", "l1", "ln")
)
```

Arguments

`feature`	A numeric vector representing the continuous predictor variable. Missing values (NA) are excluded during the pre-binning phase.
`target`	An integer vector of binary outcomes (0/1) corresponding to each observation in `feature`. Must have the same length as `feature`.
`min_bins`	Integer. The minimum number of bins to produce. Must be ≥ 2. Defaults to 3.
`max_bins`	Integer. The maximum number of bins to produce. Must be $\geq$ `min_bins`. Defaults to 5.
`bin_cutoff`	Numeric. The minimum fraction of total observations required for a bin to be considered valid. Bins with frequency < `bin_cutoff` will be merged. Value must be in (0, 1). Defaults to 0.05.
`max_n_prebins`	Integer. The number of initial quantiles to generate during the pre-binning phase. Defaults to 20.
`is_monotonic`	Logical. If `TRUE`, the algorithm enforces a strict monotonic relationship (increasing or decreasing) between the bin indices and their WoE values. Defaults to `TRUE`.
`convergence_threshold`	Numeric. The threshold for the change in total divergence to determine convergence during the iterative merging process. Defaults to 1e-6.
`max_iterations`	Integer. Safety limit for the maximum number of merging iterations. Defaults to 1000.
`bin_method`	Character string specifying the formula for Weight of Evidence calculation: • `"woe"`: Standard definition $\ln((p_i/P)/(n_i/N))$. • `"woe1"`: Zeng's definition $\ln(p_i/n_i)$ (direct log odds). Defaults to `"woe1"`.
`divergence_method`	Character string specifying the divergence measure to maximize. Available options: • `"iv"`: Information Value (conceptually similar to KL).

- "he": Hellinger Distance.
- "kl": Kullback-Leibler Divergence.
- "tr": Triangular Discrimination.
- "klj": Jeffrey's Divergence (Symmetric KL).
- "sc": Symmetric Chi-Square Divergence.
- "js": Jensen-Shannon Divergence.
- "l1": Manhattan Distance (L1 Norm).
- "l2": Euclidean Distance (L2 Norm).
- "ln": Chebyshev Distance (L-infinity Norm).

Defaults to "l2".

Details

This algorithm implements the "Metric Divergence Measures" framework. Unlike standard ChiMerge which uses statistical significance, this method uses a branch-and-bound approach to minimize the loss of a specific divergence metric when merging bins.

The Process:

1. **Pre-binning:** Generates granular bins based on quantiles.
2. **Rare Merging:** Merges bins smaller than bin_cutoff.
3. **Monotonicity:** If is_monotonic = TRUE, forces the WoE trend to be monotonic by merging "violating" bins in the direction that maximizes the total divergence.
4. **Optimization:** Iteratively merges the pair of adjacent bins that results in the smallest loss of total divergence, until max_bins is reached.

Value

A list containing the binning results:

- id: Integer vector of bin identifiers.
- bin: Character vector of bin labels in interval notation.
- woe: Numeric vector of Weight of Evidence for each bin.
- divergence: Numeric vector of the chosen divergence contribution per bin.
- count: Integer vector of total observations per bin.
- count_pos: Integer vector of positive cases.
- count_neg: Integer vector of negative cases.
- cutpoints: Numeric vector of upper boundaries (excluding Inf).
- total_divergence: The sum of the divergence measure across all bins.
- bin_method: The WoE calculation method used.
- divergence_method: The divergence measure used.

References

Zeng, G. (2013). Metric Divergence Measures and Information Value in Credit Scoring. *Journal of the Operational Research Society*, 64(5), 712-731.

Examples

```
# Example using the "he" (Hellinger) distance
set.seed(123)
feature <- rnorm(1000)
target <- rbinom(1000, 1, plogis(feature))

result <- ob_numerical_dmiv(feature, target,
  min_bins = 3,
  max_bins = 5,
  divergence_method = "he",
  bin_method = "woe"
)

print(result$bin)
print(result$divergence)
print(paste("Total Hellinger Distance:", round(result$total_divergence, 4)))
```

---

ob_numerical_dp *Optimal Binning for Numerical Variables using Dynamic Programming*

Description

Performs supervised discretization of continuous numerical variables using a greedy heuristic approach that resembles Dynamic Programming. This method is particularly effective at strictly enforcing monotonic trends (ascending or descending) in the Weight of Evidence (WoE), which is critical for the interpretability of logistic regression models in credit scoring.

Usage

```
ob_numerical_dp(
  feature,
  target,
  min_bins = 3,
  max_bins = 5,
  bin_cutoff = 0.05,
  max_n_prebins = 20,
  convergence_threshold = 1e-06,
  max_iterations = 1000,
  monotonic_trend = c("auto", "ascending", "descending", "none")
)
```

Arguments

feature	A numeric vector representing the continuous predictor variable. Missing values (NA) should be handled prior to binning, as they are not supported by this algorithm.

`target` An integer vector of binary outcomes (0/1) corresponding to each observation in `feature`. Must have the same length as `feature`.

`min_bins` Integer. The minimum number of bins to produce. Must be ≥ 2. Defaults to 3.

`max_bins` Integer. The maximum number of bins to produce. Must be $\geq$ `min_bins`. Defaults to 5.

`bin_cutoff` Numeric. The minimum fraction of total observations required for a bin to be considered valid. Bins with frequency < `bin_cutoff` will be merged. Value must be in (0, 1). Defaults to 0.05.

`max_n_prebins` Integer. The number of initial quantiles to generate during the pre-binning phase. Defaults to 20.

`convergence_threshold`
Numeric. The threshold for the change in metrics to determine convergence during the iterative merging process. Defaults to 1e-6.

`max_iterations` Integer. Safety limit for the maximum number of merging iterations. Defaults to 1000.

`monotonic_trend`
Character string specifying the desired direction of the Weight of Evidence (WoE) trend.

- `"auto"`: Automatically determines the most likely trend (ascending or descending) based on the correlation between the feature and the target.
- `"ascending"`: Forces the WoE to increase as the feature value increases.
- `"descending"`: Forces the WoE to decrease as the feature value increases.
- `"none"`: Does not enforce any monotonic constraint (allows peaks and valleys).

Defaults to `"auto"`.

Details

Although named "DP" (Dynamic Programming) in some contexts, this implementation primarily uses a **greedy heuristic** to optimize the Information Value (IV) while satisfying constraints.

Algorithm Steps:

1. **Pre-binning:** Generates initial granular bins based on quantiles.
2. **Trend Determination:** If `monotonic_trend = "auto"`, calculates the Pearson correlation between the feature and target to decide if the WoE should increase or decrease.
3. **Monotonicity Enforcement:** Iteratively merges adjacent bins that violate the determined or requested trend.
4. **Constraint Satisfaction:** Merges rare bins (below `bin_cutoff`) and ensures the number of bins is within `[min_bins, max_bins]`.
5. **Optimization:** Greedily merges similar bins (based on WoE difference) to reduce complexity while attempting to preserve information.

This method is often preferred when strict business logic dictates a specific relationship direction (e.g., "higher income must imply lower risk").

Value

A list containing the binning results:

- `id`: Integer vector of bin identifiers.
- `bin`: Character vector of bin labels in interval notation.
- `woe`: Numeric vector of Weight of Evidence for each bin.
- `iv`: Numeric vector of Information Value contribution per bin.
- `count`: Integer vector of total observations per bin.
- `count_pos`: Integer vector of positive cases.
- `count_neg`: Integer vector of negative cases.
- `event_rate`: Numeric vector of the target event rate in each bin.
- `cutpoints`: Numeric vector of upper boundaries (excluding Inf).
- `total_iv`: The total Information Value of the binned variable.
- `monotonic_trend`: The actual trend enforced ("ascending", "descending", or "none").
- `execution_time_ms`: Execution time in milliseconds.

See Also

ob_numerical_cm, ob_numerical_bb

Examples

```
# Example: forcing a descending trend
set.seed(123)
feature <- runif(1000, 0, 100)
# Target has a complex relationship, but we want to force a linear view
target <- rbinom(1000, 1, 0.5 + 0.003 * feature) # slightly positive trend

# Force "descending" (even if data suggests ascending) to see enforcement
result <- ob_numerical_dp(feature, target,
  min_bins = 3,
  max_bins = 5,
  monotonic_trend = "descending"
)

print(result$bin)
print(result$woe) # Should be strictly decreasing
```

ob_numerical_ewb *Hybrid Optimal Binning using Equal-Width Initialization and IV Optimization*

Description

Performs supervised discretization of continuous numerical variables using a hybrid approach. The algorithm initializes with an Equal-Width Binning (EWB) strategy to capture the scale of the variable, followed by an iterative, supervised optimization phase that merges bins to maximize Information Value (IV) and enforce monotonicity.

Usage

```
ob_numerical_ewb(
  feature,
  target,
  min_bins = 3,
  max_bins = 5,
  bin_cutoff = 0.05,
  max_n_prebins = 20,
  is_monotonic = TRUE,
  convergence_threshold = 1e-06,
  max_iterations = 1000
)
```

Arguments

feature	A numeric vector representing the continuous predictor variable. Missing values (NA) are excluded during the pre-binning phase but should ideally be handled prior to binning.
target	An integer vector of binary outcomes (0/1) corresponding to each observation in feature. Must have the same length as feature.
min_bins	Integer. The minimum number of bins to produce. Must be ≥ 2. Defaults to 3.
max_bins	Integer. The maximum number of bins to produce. Must be $\geq$ min_bins. Defaults to 5.
bin_cutoff	Numeric. The minimum fraction of total observations required for a bin to be considered valid. Bins with frequency < bin_cutoff are merged with their most similar neighbor (based on event rate). Value must be in (0, 1). Defaults to 0.05.
max_n_prebins	Integer. The number of initial equal-width intervals to generate during the pre-binning phase. This parameter defines the initial granularity/search space. Defaults to 20.
is_monotonic	Logical. If TRUE, the algorithm enforces a strict monotonic relationship (increasing or decreasing) between the bin indices and their Weight of Evidence (WoE). Defaults to TRUE.

`convergence_threshold`
: Numeric. The threshold for determining convergence during the iterative merging process. Defaults to 1e-6.

`max_iterations`
: Integer. Safety limit for the maximum number of merging iterations. Defaults to 1000.

Details

Unlike standard Equal-Width binning which is purely unsupervised, this function implements a **Hybrid Discretization Pipeline**:

1. **Phase 1: Unsupervised Initialization (Scale Preservation)** The range of the feature $[min(x), max(x)]$ is divided into `max_n_prebins` intervals of equal width $w = (max(x) - min(x))/N$. This step preserves the cardinal magnitude of the data but is sensitive to outliers.
2. **Phase 2: Statistical Stabilization** Bins falling below the `bin_cutoff` threshold are merged. Unlike naive approaches, this implementation merges rare bins with the neighbor that has the most similar class distribution (event rate), minimizing the distortion of the predictive relationship.
3. **Phase 3: Monotonicity Enforcement** If `is_monotonic = TRUE`, the algorithm checks for non-monotonic trends in the Weight of Evidence (WoE). Violating adjacent bins are iteratively merged to ensure a strictly increasing or decreasing relationship, which is a key requirement for interpretable Logistic Regression scorecards.
4. **Phase 4: IV-Based Optimization** If the number of bins exceeds `max_bins`, the algorithm applies a hierarchical bottom-up merging strategy. It calculates the *Information Value Loss* for every possible pair of adjacent bins:

$$\Delta IV = (IV_i + IV_{i+1}) - IV_{merged}$$

   The pair minimizing this loss is merged, ensuring that the final coarse classes retain the maximum possible predictive power of the original variable.

Technical Note on Outliers: Because the initialization is based on the range, extreme outliers can compress the majority of the data into a single initial bin. If your data is highly skewed or contains outliers, consider using `ob_numerical_cm` (Quantile/ChiMerge) or winsorizing the data before using this function.

Value

A list containing the binning results:

- `id`: Integer vector of bin identifiers.
- `bin`: Character vector of bin labels in interval notation.
- `woe`: Numeric vector of Weight of Evidence for each bin.
- `iv`: Numeric vector of Information Value contribution per bin.
- `count`: Integer vector of total observations per bin.
- `count_pos`: Integer vector of positive cases.
- `count_neg`: Integer vector of negative cases.

- `cutpoints`: Numeric vector of upper boundaries (excluding Inf).
- `total_iv`: The total Information Value of the binned variable.
- `converged`: Logical indicating if the algorithm converged.

References

Dougherty, J., Kohavi, R., & Sahami, M. (1995). Supervised and unsupervised discretization of continuous features. *Machine Learning Proceedings*, 194-202.

Siddiqi, N. (2012). *Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring*. John Wiley & Sons.

Catlett, J. (1991). On changing continuous attributes into ordered discrete attributes. *Proceedings of the European Working Session on Learning on Machine Learning*, 164-178.

See Also

`ob_numerical_cm` for Quantile/Chi-Square binning, `ob_numerical_dp` for Dynamic Programming approaches.

Examples

```
# Example 1: Uniform distribution (Ideal for Equal-Width)
set.seed(123)
feature <- runif(1000, 0, 100)
target <- rbinom(1000, 1, plogis(0.05 * feature - 2))

res_ewb <- ob_numerical_ewb(feature, target, max_bins = 5)
print(res_ewb$bin)
print(paste("Total IV:", round(res_ewb$total_iv, 4)))

# Example 2: Effect of Outliers (The weakness of Equal-Width)
feature_outlier <- c(feature, 10000) # One extreme outlier
target_outlier <- c(target, 0)

# Note: The algorithm tries to recover, but the initial split is distorted
res_outlier <- ob_numerical_ewb(feature_outlier, target_outlier, max_bins = 5)
print(res_outlier$bin)
```

---

ob_numerical_fast_mdlp

Optimal Binning using MDLP with Monotonicity Constraints

Description

Performs supervised discretization of continuous numerical variables using the Minimum Description Length Principle (MDLP) algorithm, enhanced with optional monotonicity constraints on the Weight of Evidence (WoE). This method is particularly suitable for creating interpretable bins for logistic regression models in domains like credit scoring.

Usage

```
ob_numerical_fast_mdlp(
  feature,
  target,
  min_bins = 2L,
  max_bins = 5L,
  bin_cutoff = 0.05,
  max_n_prebins = 100L,
  convergence_threshold = 1e-06,
  max_iterations = 1000L,
  force_monotonicity = TRUE
)
```

Arguments

`feature`	A numeric vector representing the continuous predictor variable. Missing values (NA) are excluded during the binning process.
`target`	An integer vector of binary outcomes (0/1) corresponding to each observation in `feature`. Must have the same length as `feature`.
`min_bins`	Integer. The minimum number of bins to produce. Must be ≥ 2. Defaults to 2.
`max_bins`	Integer. The maximum number of bins to produce. Must be $\geq$ `min_bins`. Defaults to 5.
`bin_cutoff`	Numeric. Currently unused in this implementation (reserved for future versions). Defaults to 0.05.
`max_n_prebins`	Integer. Currently unused in this implementation (reserved for future versions). Defaults to 100.
`convergence_threshold`	Numeric. The threshold for determining convergence during the iterative monotonicity enforcement process. Defaults to 1e-6.
`max_iterations`	Integer. Safety limit for the maximum number of iterations in the monotonicity enforcement phase. Defaults to 1000.
`force_monotonicity`	Logical. If `TRUE`, the algorithm enforces a strict monotonic relationship (increasing or decreasing) between the bin indices and their Weight of Evidence (WoE) values. Defaults to `TRUE`.

Details

This function implements a sophisticated hybrid approach combining the classic MDLP algorithm with modern monotonicity constraints.

Algorithm Pipeline:

1. **Data Preparation:** Removes NA values and sorts the data by feature value.
2. **MDLP Discretization (Fayyad & Irani, 1993):**
   - Recursively evaluates all possible binary splits of the sorted data.
   - For each potential split, calculates the Information Gain (IG).

- Applies the MDLP stopping criterion:

$$IG > \frac{\log_2(N-1) + \Delta}{N}$$

  where N is the total number of samples and $\Delta = \log_2(3^k - 2) - k \cdot E(S)$ (for binary classification, $k = 2$).
- Only accepts splits that significantly reduce entropy beyond what would be expected by chance, balancing model fit with complexity.

3. **Constraint Enforcement:**
   - **Min/Max Bins:** Adjusts the number of bins to meet `[min_bins, max_bins]` requirements through intelligent splitting or merging.
   - **Monotonicity (if enabled):** Iteratively merges adjacent bins with the most similar WoE values until a strictly increasing or decreasing trend is achieved across all bins.

Technical Notes:

- The algorithm uses Laplace smoothing ($\alpha = 0.5$) when calculating WoE to prevent $\log(0)$ errors for bins with pure class distributions.
- When all feature values are identical, the algorithm creates artificial bins.
- The monotonicity enforcement phase is iterative and uses the `convergence_threshold` to determine when changes in WoE become negligible.

Value

A list containing the binning results:

- `id`: Integer vector of bin identifiers.
- `bin`: Character vector of bin labels in interval notation.
- `woe`: Numeric vector of Weight of Evidence for each bin.
- `iv`: Numeric vector of Information Value contribution per bin.
- `count`: Integer vector of total observations per bin.
- `count_pos`: Integer vector of positive cases.
- `count_neg`: Integer vector of negative cases.
- `cutpoints`: Numeric vector of upper boundaries (excluding Inf).
- `converged`: Logical indicating if the monotonicity enforcement converged.
- `iterations`: Integer count of iterations in monotonicity phase.

References

Fayyad, U. M., & Irani, K. B. (1993). Multi-interval discretization of continuous-valued attributes for classification learning. *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, 1022-1029.

Kurgan, L. A., & Musilek, P. (2006). A survey of techniques. *IEEE Transactions on Knowledge and Data Engineering*, 18(5), 673-689.

Garcia, S., Luengo, J., & Herrera, F. (2013). Data preprocessing in data mining. *Springer Science & Business Media*.

See Also

ob_numerical_cm for ChiMerge-based approaches, ob_numerical_dp for dynamic programming methods.

Examples

```
# Example: Standard usage with monotonicity
set.seed(123)
feature <- rnorm(1000)
target <- rbinom(1000, 1, plogis(2 * feature)) # Positive relationship

result <- ob_numerical_fast_mdlp(feature, target,
  min_bins = 3,
  max_bins = 6,
  force_monotonicity = TRUE
)

print(result$bin)
print(result$woe) # Should show a monotonic trend

# Example: Disabling monotonicity for exploratory analysis
result_no_mono <- ob_numerical_fast_mdlp(feature, target,
  min_bins = 3,
  max_bins = 6,
  force_monotonicity = FALSE
)

print(result_no_mono$woe) # May show non-monotonic patterns
```

---

ob_numerical_fetb *Optimal Binning using Fisher's Exact Test*

Description

Performs supervised discretization of continuous numerical variables using Fisher's Exact Test. This method iteratively merges adjacent bins that are statistically similar (highest p-value) while strictly enforcing a monotonic Weight of Evidence (WoE) trend.

Usage

```
ob_numerical_fetb(
  feature,
  target,
  min_bins = 3,
  max_bins = 5,
  max_n_prebins = 20,
  convergence_threshold = 1e-06,
  max_iterations = 1000
)
```

Arguments

`feature`	A numeric vector representing the continuous predictor variable. Missing values (NA) should be handled prior to binning.
`target`	An integer vector of binary outcomes (0/1) corresponding to each observation in `feature`. Must have the same length as `feature`.
`min_bins`	Integer. The minimum number of bins to produce. Must be ≥ 2. Defaults to 3.
`max_bins`	Integer. The maximum number of bins to produce. Must be $\geq$ `min_bins`. Defaults to 5.
`max_n_prebins`	Integer. The number of initial quantiles to generate during the pre-binning phase. Defaults to 20.
`convergence_threshold`	Numeric. The threshold for the change in Information Value (IV) to determine convergence during the iterative merging process. Defaults to 1e-6.
`max_iterations`	Integer. Safety limit for the maximum number of merging iterations. Defaults to 1000.

Details

The **Fisher's Exact Test Binning (FETB)** algorithm provides a robust statistical alternative to ChiMerge.

Key Differences from ChiMerge:

- **Exact Probability:** Instead of relying on the Chi-Square asymptotic approximation (which can be unreliable for small bin counts), FETB calculates the exact hypergeometric probability of independence between the bin index and the target.
- **Merge Criterion:** In each step, the algorithm identifies the pair of adjacent bins with the *highest* p-value (indicating they are the most statistically indistinguishable) and merges them.
- **Monotonicity:** The algorithm incorporates a check after every merge to ensure the WoE trend remains monotonic, merging strictly violating bins immediately.

This method is particularly recommended when working with smaller datasets or highly imbalanced target classes, where the assumptions of the Chi-Square test might be violated.

Value

A list containing the binning results:

- `id`: Integer vector of bin identifiers.
- `bin`: Character vector of bin labels in interval notation.
- `woe`: Numeric vector of Weight of Evidence for each bin.
- `iv`: Numeric vector of Information Value contribution per bin.
- `count`: Integer vector of total observations per bin.
- `count_pos`: Integer vector of positive cases.
- `count_neg`: Integer vector of negative cases.
- `cutpoints`: Numeric vector of upper boundaries (excluding Inf).
- `converged`: Logical indicating if the algorithm converged.
- `iterations`: Integer count of iterations performed.

See Also

ob_numerical_cm

Examples

```
# Example: Binning a small dataset where Fisher's Exact Test excels
set.seed(123)
feature <- rnorm(100)
target <- rbinom(100, 1, 0.2)

result <- ob_numerical_fetb(feature, target,
  min_bins = 2,
  max_bins = 4,
  max_n_prebins = 10
)

print(result$bin)
print(result$woe)
```

---

ob_numerical_ir *Optimal Binning using Isotonic Regression (PAVA)*

Description

Performs supervised discretization of continuous numerical variables using Isotonic Regression (specifically the Pool Adjacent Violators Algorithm - PAVA). This method ensures a strictly monotonic relationship between bin indices and the empirical event rate, making it ideal for applications requiring shape constraints like credit scoring.

Usage

```
ob_numerical_ir(
  feature,
  target,
  min_bins = 3,
  max_bins = 5,
  bin_cutoff = 0.05,
  max_n_prebins = 20,
  auto_monotonicity = TRUE,
  convergence_threshold = 1e-06,
  max_iterations = 1000
)
```

Arguments

`feature`	A numeric vector representing the continuous predictor variable. Missing values (NA) are excluded from the binning process.
`target`	An integer vector of binary outcomes (0/1) corresponding to each observation in `feature`. Must have the same length as `feature`.
`min_bins`	Integer. The target minimum number of bins. Must be ≥ 2. Defaults to 3. Enforcing monotonicity can require pooling bins together, so the result may hold fewer bins than requested; see Details.
`max_bins`	Integer. The maximum number of bins to produce. Must be $\geq$ `min_bins`. Defaults to 5.
`bin_cutoff`	Numeric. The minimum fraction of total observations required for a bin to be considered valid. Bins with frequency < `bin_cutoff` will be merged with neighbors. Value must be in (0, 1). Defaults to 0.05.
`max_n_prebins`	Integer. The number of initial quantiles to generate during the pre-binning phase. Defaults to 20.
`auto_monotonicity`	Logical. If `TRUE`, the algorithm automatically determines the optimal monotonicity direction (increasing or decreasing) based on the Pearson correlation between feature values and target. If `FALSE`, defaults to increasing monotonicity. Defaults to `TRUE`.
`convergence_threshold`	Numeric. Reserved for future use. Currently not actively used by the PAVA algorithm, which has guaranteed convergence. Defaults to 1e-6.
`max_iterations`	Integer. Safety limit for iterative merging operations during pre-processing steps (e.g., rare bin merging). Defaults to 1000.

Details

This function implements a **shape-constrained** binning approach using **Isotonic Regression**. Unlike heuristic merging strategies (ChiMerge, DP), this method finds the optimal monotonic fit in a single pass.

Core Algorithm (PAVA): The Pool Adjacent Violators Algorithm (Best & Chakravarti, 1990) is used to transform the empirical event rates of initial bins into a sequence that is either monotonically increasing or decreasing. It works by scanning the sequence and merging ("pooling") any adjacent pairs that violate the desired trend until a perfect fit is achieved. This guarantees an optimal solution in $O(n)$ time.

Process Flow:

1. **Pre-binning:** Creates initial bins using quantiles.
2. **Stabilization:** Merges bins below `bin_cutoff`.
3. **Trend Detection:** If `auto_monotonicity = TRUE`, calculates the correlation between feature midpoints and bin event rates to determine if the relationship should be increasing or decreasing.
4. **Shape Enforcement:** Applies PAVA to the sequence of bin event rates and *merges* the bins that the algorithm pools into each block, so the surviving bins are monotonic in their own observed event rate.

5. **Metric Calculation:** Derives WoE and IV from the observed counts of those bins.

Pooling merges bins: with the bin counts as weights, the fitted rate PAVA assigns to a block is exactly $\sum count_pos / \sum count$ over the block's members. Merging them therefore reproduces the isotonic fit exactly while `count`, `count_pos` and `count_neg` remain the counts actually observed between the reported `cutpoints`. A consequence is that `min_bins` is a *target* rather than a guarantee for this algorithm: when the empirical event rate violates the trend, monotonicity can only be restored by pooling, and the result may hold fewer bins than requested. Use a non-shape-constrained method such as `ob_numerical_mdlp` when the bin count matters more than the shape constraint.

Advantages:

- **Global Optimality:** PAVA finds the best fit under the monotonicity constraint.
- **No Hyperparameters:** Unlike ChiMerge's p-value threshold, PAVA requires no significance level tuning for the core regression step.
- **Robustness:** Less sensitive to arbitrary thresholds compared to greedy merging.

Value

A list containing the binning results:

- `id`: Integer vector of bin identifiers.
- `bin`: Character vector of bin labels in interval notation.
- `woe`: Numeric vector of Weight of Evidence for each bin.
- `iv`: Numeric vector of Information Value contribution per bin.
- `count`: Integer vector of total observations per bin.
- `count_pos`: Integer vector of positive cases observed in the bin.
- `count_neg`: Integer vector of negative cases observed in the bin.
- `cutpoints`: Numeric vector of upper boundaries (excluding Inf).
- `total_iv`: The total Information Value of the binned variable.
- `monotone_increasing`: Logical indicating if the final WoE trend is increasing.
- `converged`: Logical indicating successful completion.

References

Barlow, R. E., Bartholomew, D. J., Bremner, J. M., & Brunk, H. D. (1972). *Statistical inference under order restrictions*. John Wiley & Sons.

Best, M. J., & Chakravarti, N. (1990). Active set algorithms for isotonic regression; A unifying framework. *Mathematical Programming*, 47(1-3), 425-439.

See Also

`ob_numerical_dp` for greedy dynamic programming approaches.

Examples

```
# Example: Forcing a monotonic WoE trend
set.seed(123)
feature <- rnorm(500)
# Create a slightly noisy but generally increasing relationship
prob <- plogis(0.5 * feature + rnorm(500, 0, 0.3))
target <- rbinom(500, 1, prob)

result <- ob_numerical_ir(feature, target,
  min_bins = 4,
  max_bins = 6,
  auto_monotonicity = TRUE
)

print(result$bin)
print(round(result$woe, 3))
print(paste("Monotonic Increasing:", result$monotone_increasing))
```

---

ob_numerical_jedi *Optimal Binning using Joint Entropy-Driven Interval Discretization (JEDI)*

Description

Performs supervised discretization of continuous numerical variables using a holistic approach that balances entropy reduction (information gain) with statistical stability. The JEDI algorithm combines quantile-based initialization with an iterative optimization process that enforces monotonicity and minimizes Information Value (IV) loss.

Usage

```
ob_numerical_jedi(
  feature,
  target,
  min_bins = 3,
  max_bins = 5,
  bin_cutoff = 0.05,
  max_n_prebins = 20,
  convergence_threshold = 1e-06,
  max_iterations = 1000
)
```

Arguments

feature	A numeric vector representing the continuous predictor variable. Missing values (NA) should be handled prior to binning.

`target`	An integer vector of binary outcomes (0/1) corresponding to each observation in `feature`. Must have the same length as `feature`.
`min_bins`	Integer. The minimum number of bins to produce. Must be ≥ 2. Defaults to 3.
`max_bins`	Integer. The maximum number of bins to produce. Must be $\geq$ `min_bins`. Defaults to 5.
`bin_cutoff`	Numeric. The minimum fraction of total observations required for a bin to be considered valid. Bins smaller than this threshold are merged. Value must be in (0, 1). Defaults to 0.05.
`max_n_prebins`	Integer. The number of initial quantiles to generate during the initialization phase. Defaults to 20.
`convergence_threshold`	Numeric. The threshold for the change in total IV to determine convergence during the iterative optimization. Defaults to 1e-6.
`max_iterations`	Integer. Safety limit for the maximum number of iterations. Defaults to 1000.

Details

The JEDI algorithm is designed to be a robust "all-rounder" for credit scoring and risk modeling. Its methodology proceeds in four distinct stages:

1. **Initialization (Quantile Pre-binning):** The feature space is divided into `max_n_prebins` segments containing approximately equal numbers of observations. This ensures the algorithm starts with a statistically balanced view of the data.
2. **Stabilization (Rare Bin Merging):** Adjacent bins with frequencies below `bin_cutoff` are merged. The merge direction is chosen to minimize the distortion of the event rate (similar to ChiMerge).
3. **Monotonicity Enforcement:** The algorithm heuristically determines the dominant trend (increasing or decreasing) of the Weight of Evidence (WoE) and iteratively merges adjacent bins that violate this trend. This step effectively reduces the conditional entropy of the binning sequence with respect to the target.
4. **IV Optimization:** If the number of bins exceeds `max_bins`, the algorithm merges the pair of adjacent bins that results in the smallest decrease in total Information Value. This greedy approach ensures that the final discretization retains the maximum possible predictive power given the constraints.

This joint approach (Entropy/IV + Stability constraints) makes JEDI particularly effective for datasets with noise or non-monotonic initial distributions that require smoothing.

Value

A list containing the binning results:

- `id`: Integer vector of bin identifiers.
- `bin`: Character vector of bin labels in interval notation.
- `woe`: Numeric vector of Weight of Evidence for each bin.
- `iv`: Numeric vector of Information Value contribution per bin.

- `count`: Integer vector of total observations per bin.
- `count_pos`: Integer vector of positive cases.
- `count_neg`: Integer vector of negative cases.
- `cutpoints`: Numeric vector of upper boundaries (excluding Inf).
- `converged`: Logical indicating if the algorithm converged.
- `iterations`: Integer count of iterations performed.

See Also

ob_numerical_cm, ob_numerical_ir

Examples

```
# Example: Binning a variable with a complex relationship
set.seed(123)
feature <- rnorm(1000)
# Target probability has a quadratic component (non-monotonic)
# JEDI will try to force a monotonic approximation that maximizes IV
target <- rbinom(1000, 1, plogis(0.5 * feature + 0.1 * feature^2))

result <- ob_numerical_jedi(feature, target,
  min_bins = 3,
  max_bins = 6,
  max_n_prebins = 20
)

print(result$bin)
```

---

ob_numerical_jedi_mwoe

Optimal Binning for Multiclass Targets using JEDI M-WOE

Description

Performs supervised discretization of continuous numerical variables for **multiclass** target variables (e.g., 0, 1, 2). It extends the Joint Entropy-Driven Interval (JEDI) discretization framework to calculate and optimize the Multinomial Weight of Evidence (M-WOE) for each class simultaneously.

Usage

```
ob_numerical_jedi_mwoe(
  feature,
  target,
  min_bins = 3,
  max_bins = 5,
  bin_cutoff = 0.05,
```

```
  max_n_prebins = 20,
  convergence_threshold = 1e-06,
  max_iterations = 1000
)
```

Arguments

`feature`	A numeric vector representing the continuous predictor variable. Missing values (NA) should be excluded prior to execution.
`target`	An integer vector of multiclass outcomes (0, 1, ..., K-1) corresponding to each observation in `feature`. Must have at least 2 distinct classes.
`min_bins`	Integer. The minimum number of bins to produce. Must be ≥ 2. Defaults to 3.
`max_bins`	Integer. The maximum number of bins to produce. Must be $\geq$ `min_bins`. Defaults to 5.
`bin_cutoff`	Numeric. The minimum fraction of total observations required for a bin to be considered valid. Bins smaller than this threshold are merged. Defaults to 0.05.
`max_n_prebins`	Integer. The number of initial quantiles to generate during the pre-binning phase. Defaults to 20.
`convergence_threshold`	Numeric. The threshold for the change in total Multinomial IV to determine convergence. Defaults to 1e-6.
`max_iterations`	Integer. Safety limit for the maximum number of iterations. Defaults to 1000.

Details

Multinomial Weight of Evidence (M-WOE): For a target with K classes, the WoE for class k in bin i is defined using a "One-vs-Rest" approach:

$$WOE_{i,k} = \ln\left(\frac{P(X \in bin_i | Y = k)}{P(X \in bin_i | Y \neq k)}\right)$$

Algorithm Workflow:

1. **Multiclass Initialization:** The algorithm starts with quantile-based bins and computes the initial event rates for all K classes.
2. **Joint Monotonicity:** The algorithm attempts to enforce monotonicity for *all* classes. If bin i violates the trend for Class 1 OR Class 2, it may be merged. This ensures the variable is predictive across the entire spectrum of outcomes.
3. **Global IV Optimization:** When reducing the number of bins to `max_bins`, the algorithm merges the pair of bins that minimizes the loss of the *Sum of IVs* across all classes:

$$Loss = \sum_{k=0}^{K-1} \Delta IV_k$$

This method is ideal for use cases like:

- predicting loan status (Current, Late, Default)
- customer churn levels (Active, Dormant, Churned)
- ordinal survey responses.

Value

A list containing the binning results:

- `id`: Integer vector of bin identifiers.
- `bin`: Character vector of bin labels in interval notation.
- `woe`: A numeric matrix where each column represents the WoE for a specific class (One-vs-Rest).
- `iv`: A numeric matrix where each column represents the IV contribution for a specific class.
- `count`: Integer vector of total observations per bin.
- `class_counts`: A matrix of observation counts per class per bin.
- `cutpoints`: Numeric vector of upper boundaries (excluding Inf).
- `n_classes`: The number of distinct target classes found.

See Also

`ob_numerical_jedi` for the binary version.

Examples

```
# Example: Multiclass target (0, 1, 2)
set.seed(123)
feature <- rnorm(1000)
# Class 0: low feature, Class 1: medium, Class 2: high
target <- cut(feature + rnorm(1000, 0, 0.5),
  breaks = c(-Inf, -0.5, 0.5, Inf),
  labels = FALSE
) - 1

result <- ob_numerical_jedi_mwoe(feature, target,
  min_bins = 3,
  max_bins = 5
)

# Check WoE for Class 2 (High values)
print(result$woe[, 3]) # Column 3 corresponds to Class 2
```

---

ob_numerical_kmb *Optimal Binning using K-means Inspired Initialization (KMB)*

Description

Performs supervised discretization of continuous numerical variables using a K-means inspired binning strategy. Initial bin boundaries are determined by placing centroids uniformly across the feature range and defining cuts at midpoints. The algorithm then optimizes these bins using statistical constraints.

Usage

```
ob_numerical_kmb(
  feature,
  target,
  min_bins = 3,
  max_bins = 5,
  bin_cutoff = 0.05,
  max_n_prebins = 20,
  enforce_monotonic = TRUE,
  convergence_threshold = 1e-06,
  max_iterations = 1000
)
```

Arguments

`feature`	A numeric vector representing the continuous predictor variable. Missing values (NA) should be handled prior to binning.
`target`	An integer vector of binary outcomes (0/1) corresponding to each observation in `feature`. Must have the same length as `feature`.
`min_bins`	Integer. The minimum number of bins to produce. Must be ≥ 2. Defaults to 3.
`max_bins`	Integer. The maximum number of bins to produce. Must be $\geq$ `min_bins`. Defaults to 5.
`bin_cutoff`	Numeric. The minimum fraction of total observations required for a bin to be considered valid. Bins smaller than this threshold are merged. Value must be in (0, 1). Defaults to 0.05.
`max_n_prebins`	Integer. The number of initial centroids/bins to generate during the initialization phase. Defaults to 20.
`enforce_monotonic`	Logical. If `TRUE`, the algorithm enforces a monotonic relationship in the Weight of Evidence (WoE) across bins. Defaults to `TRUE`.
`convergence_threshold`	Numeric. The threshold for determining convergence during the iterative optimization process. Defaults to 1e-6.
`max_iterations`	Integer. Safety limit for the maximum number of iterations. Defaults to 1000.

Details

The KMB algorithm offers a unique initialization strategy compared to standard binning methods:

1. **Initialization (K-means Style):** Instead of using quantiles, `max_n_prebins` centroids are placed uniformly across the range $[min(x), max(x)]$. Bin boundaries are then defined as the midpoints between adjacent centroids. This can lead to more evenly distributed initial bin widths in terms of the feature's scale.
2. **Optimization:** The initialized bins undergo standard post-processing:
   - **Rare Bin Merging:** Bins below `bin_cutoff` are merged with their most similar neighbor (by event rate).

- **Monotonicity:** If `enforce_monotonic = TRUE`, adjacent bins violating the dominant WoE trend are merged.
- **Bin Count Adjustment:** If the number of bins exceeds `max_bins`, the algorithm greedily merges adjacent bins with the smallest absolute difference in Information Value.

This method can be advantageous when the underlying distribution of the feature is relatively uniform, as it avoids creating overly granular bins in dense regions from the start.

Value

A list containing the binning results:

- `id`: Integer vector of bin identifiers.
- `bin`: Character vector of bin labels in interval notation.
- `woe`: Numeric vector of Weight of Evidence for each bin.
- `iv`: Numeric vector of Information Value contribution per bin.
- `count`: Integer vector of total observations per bin.
- `count_pos`: Integer vector of positive cases.
- `count_neg`: Integer vector of negative cases.
- `centroids`: Numeric vector of bin centroids (mean feature value per bin).
- `cutpoints`: Numeric vector of upper boundaries (excluding Inf).
- `total_iv`: The total Information Value of the binned variable.
- `converged`: Logical indicating if the algorithm converged.

See Also

`ob_numerical_ewb`, `ob_numerical_cm`

Examples

```
# Example: Comparing KMB with EWB on uniform data
set.seed(123)
feature <- runif(1000, 0, 100)
target <- rbinom(1000, 1, plogis(0.02 * feature))

result_kmb <- ob_numerical_kmb(feature, target, max_bins = 5)
print(result_kmb$bin)
print(paste("KMB Total IV:", round(result_kmb$total_iv, 4)))
```

ob_numerical_ldb *Optimal Binning for Numerical Variables using Local Density Binning*

Description

Implements supervised discretization via Local Density Binning (LDB), a method that leverages kernel density estimation to identify natural transition regions in the feature space while optimizing the Weight of Evidence (WoE) monotonicity and Information Value (IV) for binary classification tasks.

Usage

```
ob_numerical_ldb(
  feature,
  target,
  min_bins = 3,
  max_bins = 5,
  bin_cutoff = 0.05,
  max_n_prebins = 20,
  enforce_monotonic = TRUE,
  convergence_threshold = 1e-06,
  max_iterations = 1000
)
```

Arguments

feature	Numeric vector of feature values to be binned. Missing values (NA) and infinite values are automatically filtered out during preprocessing.
target	Integer vector of binary target values (must contain only 0 and 1). Must have the same length as feature.
min_bins	Minimum number of bins to generate (default: 3). Must be at least 2.
max_bins	Maximum number of bins to generate (default: 5). Must be greater than or equal to min_bins.
bin_cutoff	Minimum fraction of total observations in each bin (default: 0.05). Bins with frequency below this threshold are merged with adjacent bins. Must be in the range [0, 1].
max_n_prebins	Maximum number of pre-bins before optimization (default: 20). Controls granularity of initial density-based discretization.
enforce_monotonic	Logical flag to enforce monotonicity in WoE values across bins (default: TRUE). When enabled, bins violating monotonicity are iteratively merged until global monotonicity is achieved.
convergence_threshold	Convergence threshold for iterative optimization (default: 1e-6). Currently used for future extensions.

`max_iterations` Maximum number of iterations for merging operations (default: 1000). Prevents infinite loops in edge cases.

Details

Algorithm Overview

The Local Density Binning (LDB) algorithm operates in four sequential phases:

Phase 1: Density-Based Pre-binning

The algorithm employs kernel density estimation (KDE) with a Gaussian kernel to identify the local density structure of the feature:

$$\hat{f}(x) = \frac{1}{nh\sqrt{2\pi}} \sum_{i=1}^{n} \exp\left[-\frac{(x - x_i)^2}{2h^2}\right]$$

where h is the bandwidth computed via Silverman's rule of thumb:

$$h = 0.9 \times \min(\hat{\sigma}, \text{IQR}/1.34) \times n^{-1/5}$$

Bin boundaries are placed at local minima of $\hat{f}(x)$, which correspond to natural transition regions where density is lowest (analogous to valleys in the density landscape). This strategy ensures bins capture homogeneous subpopulations.

Phase 2: Weight of Evidence Computation

For each bin i, the WoE quantifies the log-ratio of positive to negative class distributions, adjusted with Laplace smoothing ($\alpha = 0.5$) to prevent division by zero:

$$\text{WoE}_i = \ln\left(\frac{\text{DistGood}_i}{\text{DistBad}_i}\right)$$

where:

$$\text{DistGood}_i = \frac{n_i^+ + \alpha}{n^+ + K\alpha}, \quad \text{DistBad}_i = \frac{n_i^- + \alpha}{n^- + K\alpha}$$

and K is the total number of bins. The Information Value for bin i is:

$$\text{IV}_i = (\text{DistGood}_i - \text{DistBad}_i) \times \text{WoE}_i$$

Total IV aggregates discriminatory power: $\text{IV}_{\text{total}} = \sum_{i=1}^{K} \text{IV}_i$.

Phase 3: Monotonicity Enforcement

When `enforce_monotonic = TRUE`, the algorithm ensures WoE values are monotonic with respect to bin order. The direction (increasing/decreasing) is determined via Pearson correlation between bin indices and WoE values. Bins violating monotonicity are iteratively merged using the merge strategy described in Phase 4, continuing until global monotonicity is achieved or `min_bins` is reached.

This approach is rooted in isotonic regression principles (Robertson et al., 1988), ensuring the scorecard maintains a consistent logical relationship between feature values and credit risk.

Phase 4: Adaptive Bin Merging

Two merging criteria are applied sequentially:

1. **Frequency-based merging**: Bins with total count below `bin_cutoff` $\times n$ are merged with the adjacent bin having the most similar event rate (minimizing heterogeneity). If event rates are equivalent, the merge that preserves higher IV is preferred.
2. **Cardinality reduction**: If the number of bins exceeds `max_bins`, the pair of adjacent bins minimizing IV loss when merged is identified via:

$$\Delta\text{IV}_{i,i+1} = \text{IV}_i + \text{IV}_{i+1} - \text{IV}_{\text{merged}}$$

This greedy optimization continues until $K \leq$ `max_bins`.

Theoretical Foundations

- **Kernel Density Estimation**: The bandwidth selection follows Silverman (1986, Chapter 3), balancing bias-variance tradeoff for univariate density estimation.
- **Weight of Evidence**: Siddiqi (2006) formalizes WoE/IV as measures of predictive strength in credit scoring, with IV thresholds: < 0.02 (unpredictive), 0.02-0.1 (weak), 0.1-0.3 (medium), 0.3-0.5 (strong), > 0.5 (suspect overfitting).
- **Supervised Discretization**: García et al. (2013) categorize LDB within "static" supervised methods that do not require iterative feedback from the model, unlike dynamic methods (e.g., ChiMerge).

Computational Complexity

- KDE computation: $O(n^2)$ for naive implementation (each of n points evaluates n kernel terms).
- Binary search for bin assignment: $O(n \log K)$ where K is the number of bins.
- Merge iterations: $O(K^2 \times \text{max_iterations})$ in worst case.

For large datasets ($n > 10^5$), the KDE phase dominates runtime.

Value

A list containing:

id Integer vector of bin identifiers (1-based indexing).

bin Character vector of bin intervals in the format `"(lower;upper]"`.

woe Numeric vector of Weight of Evidence values for each bin.

iv Numeric vector of Information Value contributions for each bin.

count Integer vector of total observations in each bin.

count_pos Integer vector of positive class (target = 1) counts per bin.

count_neg Integer vector of negative class (target = 0) counts per bin.

event_rate Numeric vector of event rates (proportion of positives) per bin.

cutpoints Numeric vector of cutpoints defining bin boundaries (excluding -Inf and +Inf).

converged Logical flag indicating whether the algorithm converged within `max_iterations`.

iterations Integer count of iterations performed during optimization.

total_iv Numeric scalar representing the total Information Value (sum of all bin IVs).

monotonicity Character string indicating monotonicity status: `"increasing"`, `"decreasing"`, or `"none"`.

Author(s)

Lopes, J. E. (implemented algorithm)

References

- Silverman, B. W. (1986). *Density Estimation for Statistics and Data Analysis*. Chapman and Hall/CRC.
- Siddiqi, N. (2006). *Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring*. Wiley.
- Dougherty, J., Kohavi, R., & Sahami, M. (1995). "Supervised and Unsupervised Discretization of Continuous Features". *Proceedings of the 12th International Conference on Machine Learning*, pp. 194-202.
- Robertson, T., Wright, F. T., & Dykstra, R. L. (1988). *Order Restricted Statistical Inference*. Wiley.
- García, S., Luengo, J., Sáez, J. A., López, V., & Herrera, F. (2013). "A Survey of Discretization Techniques: Taxonomy and Empirical Analysis in Supervised Learning". *IEEE Transactions on Knowledge and Data Engineering*, 25(4), 734-750.

See Also

`ob_numerical_mdlp` for Minimum Description Length Principle binning, `ob_numerical_mob` for monotonic binning with similar constraints.

Examples

```
# Simulate credit scoring data
set.seed(42)
n <- 10000
feature <- c(
  rnorm(3000, mean = 600, sd = 50), # Low-risk segment
  rnorm(4000, mean = 700, sd = 40), # Medium-risk segment
  rnorm(3000, mean = 750, sd = 30) # High-risk segment
)
target <- c(
  rbinom(3000, 1, 0.15), # 15% default rate
  rbinom(4000, 1, 0.08), # 8% default rate
  rbinom(3000, 1, 0.03) # 3% default rate
)

# Apply LDB with monotonicity enforcement
result <- ob_numerical_ldb(
  feature = feature,
```

```
  target = target,
  min_bins = 3,
  max_bins = 5,
  bin_cutoff = 0.05,
  max_n_prebins = 20,
  enforce_monotonic = TRUE
)

# Inspect binning quality
print(result$total_iv) # Should be > 0.1 for predictive features
print(result$monotonicity) # Should indicate direction

# Visualize WoE pattern
plot(result$woe,
  type = "b", xlab = "Bin", ylab = "WoE",
  main = "Monotonic WoE Trend"
)

# Generate scorecard transformation
bin_mapping <- data.frame(
  bin = result$bin,
  woe = result$woe,
  iv = result$iv
)
print(bin_mapping)
```

---

ob_numerical_lpdb *Optimal Binning using Local Polynomial Density Binning (LPDB)*

Description

Performs supervised discretization of continuous numerical variables using a novel approach that combines non-parametric density estimation with information-theoretic optimization. The algorithm first identifies natural clusters and boundaries in the feature distribution using local polynomial density estimation, then refines the bins to maximize predictive power.

Usage

```
ob_numerical_lpdb(
  feature,
  target,
  min_bins = 3,
  max_bins = 5,
  bin_cutoff = 0.05,
  max_n_prebins = 20,
  polynomial_degree = 3,
  enforce_monotonic = TRUE,
```

```
  convergence_threshold = 1e-06,
  max_iterations = 1000
)
```

Arguments

`feature`	A numeric vector representing the continuous predictor variable. Missing values (NA) should be handled prior to binning.
`target`	An integer vector of binary outcomes (0/1) corresponding to each observation in `feature`. Must have the same length as `feature`.
`min_bins`	Integer. The minimum number of bins to produce. Must be ≥ 2. Defaults to 3.
`max_bins`	Integer. The maximum number of bins to produce. Must be $\geq$ `min_bins`. Defaults to 5.
`bin_cutoff`	Numeric. The minimum fraction of total observations required for a bin to be considered valid. Bins smaller than this threshold are merged. Value must be in (0, 1). Defaults to 0.05.
`max_n_prebins`	Integer. The maximum number of initial candidate cut points to generate during the density estimation phase. Defaults to 20.
`polynomial_degree`	Integer. The degree of the local polynomial used for density estimation (note: currently approximated via KDE). Defaults to 3.
`enforce_monotonic`	Logical. If `TRUE`, the algorithm forces the Weight of Evidence (WoE) trend to be strictly monotonic. Defaults to `TRUE`.
`convergence_threshold`	Numeric. The threshold for determining convergence during the iterative merging process. Defaults to 1e-6.
`max_iterations`	Integer. Safety limit for the maximum number of merging iterations. Defaults to 1000.

Details

The **Local Polynomial Density Binning (LPDB)** algorithm is a two-stage process:

1. **Density-Based Initialization:**
   - Estimates the probability density function $f(x)$ of the `feature` using Kernel Density Estimation (KDE), which approximates local polynomial regression.
   - Identifies *critical points* on the density curve, such as local minima and inflection points. These points often correspond to natural boundaries between clusters or modes in the data.
   - Uses these critical points as initial candidate cut points to form pre-bins.
2. **Supervised Refinement:**
   - Calculates WoE and IV for each pre-bin.
   - Enforces monotonicity by merging bins that violate the trend (determined by the correlation between bin centroids and WoE values).

- Merges bins with frequencies below `bin_cutoff`.
- Iteratively merges bins to meet the `max_bins` constraint, choosing merges that minimize the loss of total Information Value.

This method is particularly powerful for complex, multi-modal distributions where standard quantile or equal-width binning might obscure important structural breaks.

Value

A list containing the binning results:

- `id`: Integer vector of bin identifiers.
- `bin`: Character vector of bin labels in interval notation.
- `woe`: Numeric vector of Weight of Evidence for each bin.
- `iv`: Numeric vector of Information Value contribution per bin.
- `count`: Integer vector of total observations per bin.
- `count_pos`: Integer vector of positive cases.
- `count_neg`: Integer vector of negative cases.
- `event_rate`: Numeric vector of the target event rate in each bin.
- `centroids`: Numeric vector of the geometric centroids of the final bins.
- `cutpoints`: Numeric vector of upper boundaries (excluding Inf).
- `total_iv`: The total Information Value of the binned variable.
- `monotonicity`: Character string indicating the final WoE trend ("increasing", "decreasing", or "none").

See Also

`ob_numerical_kmb`, `ob_numerical_jedi`

Examples

```
# Example: Binning a tri-modal distribution
set.seed(123)
# Feature with three distinct clusters
feature <- c(rnorm(300, mean = -3), rnorm(400, mean = 0), rnorm(300, mean = 3))
# Target depends on these clusters
target <- rbinom(1000, 1, plogis(feature))

result <- ob_numerical_lpdb(feature, target,
  min_bins = 3,
  max_bins = 5
)

print(result$bin) # Should ideally find cuts near -1.5 and 1.5
print(result$monotonicity)
```

ob_numerical_mblp *Optimal Binning for Numerical Features Using Monotonic Binning via Linear Programming*

Description

Implements a greedy optimization algorithm for supervised discretization of numerical features with **guaranteed monotonicity** in Weight of Evidence (WoE). Despite the "Linear Programming" designation, this method employs an iterative heuristic based on quantile pre-binning, Information Value (IV) optimization, and monotonicity enforcement through adaptive bin merging.

Important Note: This algorithm does not use formal Linear Programming solvers (e.g., simplex method). The name reflects the conceptual formulation of binning as a constrained optimization problem, but the implementation uses a deterministic greedy heuristic for computational efficiency.

Usage

```
ob_numerical_mblp(
  feature,
  target,
  min_bins = 3,
  max_bins = 5,
  bin_cutoff = 0.05,
  max_n_prebins = 20,
  force_monotonic_direction = 0,
  convergence_threshold = 1e-06,
  max_iterations = 1000
)
```

Arguments

feature	Numeric vector of feature values to be binned. Missing values (NA) and infinite values are automatically removed during preprocessing.
target	Integer vector of binary target values (must contain only 0 and 1). Must have the same length as feature.
min_bins	Minimum number of bins to generate (default: 3). Must be at least 2.
max_bins	Maximum number of bins to generate (default: 5). Must be greater than or equal to min_bins.
bin_cutoff	Minimum fraction of total observations in each bin (default: 0.05). Bins with frequency below this threshold are merged with adjacent bins. Must be in the range (0, 1).
max_n_prebins	Maximum number of pre-bins before optimization (default: 20). Controls granularity of initial quantile-based discretization.
force_monotonic_direction	Integer flag to force a specific monotonicity direction (default: 0). Valid values:

- `0`: Automatically determine direction via correlation between bin indices and WoE values.
- `1`: Force increasing monotonicity (WoE increases with feature value).
- `-1`: Force decreasing monotonicity (WoE decreases with feature value).

`convergence_threshold`
: Convergence threshold for iterative optimization (default: 1e-6). Iteration stops when the absolute change in total IV between consecutive iterations falls below this value.

`max_iterations`
: Maximum number of iterations for the optimization loop (default: 1000). Prevents infinite loops in pathological cases.

Details

Algorithm Overview

The Monotonic Binning via Linear Programming (MBLP) algorithm operates in four sequential phases designed to balance predictive power (IV maximization) and interpretability (monotonic WoE):

Phase 1: Quantile-Based Pre-binning

Initial bin boundaries are determined using empirical quantiles of the feature distribution. For k pre-bins, cutpoints are computed as:

$$q_i = x_{(\lceil p_i \times (N-1) \rceil)}, \quad p_i = \frac{i}{k}, \quad i = 1, 2, \ldots, k-1$$

where $x_{(j)}$ denotes the j-th order statistic. This approach ensures equal-frequency bins under the assumption of continuous data, though ties may cause deviations in practice. The first and last boundaries are set to $-\infty$ and $+\infty$, respectively.

Phase 2: Frequency-Based Bin Merging

Bins with total count below `bin_cutoff` $\times N$ are iteratively merged with adjacent bins to ensure statistical reliability. The merge strategy selects the neighbor with the smallest count (greedy heuristic), continuing until all bins meet the frequency threshold or `min_bins` is reached.

Phase 3: Monotonicity Direction Determination

If `force_monotonic_direction = 0`, the algorithm computes the Pearson correlation between bin indices and WoE values:

$$\rho = \frac{\sum_{i=1}^{k} (i - \bar{i})(\text{WoE}_i - \overline{\text{WoE}})}{\sqrt{\sum_{i=1}^{k} (i - \bar{i})^2 \sum_{i=1}^{k} (\text{WoE}_i - \overline{\text{WoE}})^2}}$$

The monotonicity direction is set as:

$$\text{direction} = \begin{cases} 1 & \text{if } \rho \geq 0 \text{ (increasing)} \\ -1 & \text{if } \rho < 0 \text{ (decreasing)} \end{cases}$$

If `force_monotonic_direction` is explicitly set to 1 or -1, that value overrides the correlation-based determination.

Phase 4: Iterative Optimization Loop

The core optimization alternates between two enforcement steps until convergence:

1. **Cardinality Constraint**: If the number of bins k exceeds `max_bins`, the algorithm identifies the pair of adjacent bins $(i, i+1)$ that minimizes the IV loss when merged:

$$\Delta\text{IV}_{i,i+1} = \text{IV}_i + \text{IV}_{i+1} - \text{IV}_{\text{merged}}$$

   where $\text{IV}_{\text{merged}}$ is recalculated using combined counts. The merge is performed only if it preserves monotonicity (checked via WoE comparison with neighboring bins).

2. **Monotonicity Enforcement**: For each pair of consecutive bins, violations are detected as:
   - **Increasing**: $\text{WoE}_i < \text{WoE}_{i-1} - \epsilon$
   - **Decreasing**: $\text{WoE}_i > \text{WoE}_{i-1} + \epsilon$

   where $\epsilon = 10^{-10}$ (numerical tolerance). Violating bins are immediately merged.

3. **Convergence Test**: After each iteration, the total IV is compared to the previous iteration. If $|\text{IV}^{(t)} - \text{IV}^{(t-1)}| <$ convergence_threshold or monotonicity is achieved, the loop terminates.

Weight of Evidence Computation

WoE for bin i uses Laplace smoothing ($\alpha = 0.5$) to handle zero counts:

$$\text{WoE}_i = \ln\left(\frac{\text{DistGood}_i}{\text{DistBad}_i}\right)$$

where:

$$\text{DistGood}_i = \frac{n_i^+ + \alpha}{n^+ + k\alpha}, \quad \text{DistBad}_i = \frac{n_i^- + \alpha}{n^- + k\alpha}$$

and k is the current number of bins. The Information Value contribution is:

$$\text{IV}_i = (\text{DistGood}_i - \text{DistBad}_i) \times \text{WoE}_i$$

Theoretical Foundations

- **Monotonicity Requirement**: Zeng (2014) proves that monotonic WoE is a necessary condition for stable scorecards under data drift. Non-monotonic patterns often indicate overfitting to noise.
- **Greedy Optimization**: Unlike global optimizers (MILP), greedy heuristics provide no optimality guarantees but achieve O(k²) complexity per iteration versus exponential for exact methods.
- **Quantile Binning**: Ensures initial bins have approximately equal sample sizes, reducing variance in WoE estimates (especially critical for minority classes).

Comparison with True Linear Programming

Formal LP formulations for binning (Belotti et al., 2016) express the problem as:

$$\max_{\mathbf{z},\mathbf{b}} \sum_{i=1}^{k} \text{IV}_i(\mathbf{b})$$

subject to:

$$\text{WoE}_i \leq \text{WoE}_{i+1} \quad \forall i \quad \text{(monotonicity)}$$

$$\sum_{j=1}^{N} z_{ij} = 1 \quad \forall j \quad \text{(assignment)}$$

$$z_{ij} \in \{0, 1\}, \quad b_i \in \mathbb{R}$$

where z_{ij} indicates if observation j is in bin i, and b_i are bin boundaries. Such formulations require MILP solvers (CPLEX, Gurobi) and scale poorly beyond $N > 10^4$. MBLP sacrifices global optimality for **scalability** and **determinism**.

Computational Complexity

- Initial sorting: $O(N \log N)$
- Quantile computation: $O(k)$
- Per-iteration operations: $O(k^2)$ (pairwise comparisons for merging)
- Total: $O(N \log N + k^2 \times \text{max_iterations})$

For typical credit scoring datasets ($N \sim 10^5$, $k \sim 5$), runtime is dominated by sorting. Pathological cases (highly non-monotonic data) may require many iterations to enforce monotonicity.

Value

A list containing:

id Integer vector of bin identifiers (1-based indexing).

bin Character vector of bin intervals in the format `"(lower;upper]"`.

woe Numeric vector of Weight of Evidence values for each bin.

iv Numeric vector of Information Value contributions for each bin.

count Integer vector of total observations in each bin.

count_pos Integer vector of positive class (target = 1) counts per bin.

count_neg Integer vector of negative class (target = 0) counts per bin.

event_rate Numeric vector of event rates (proportion of positives) per bin.

cutpoints Numeric vector of cutpoints defining bin boundaries (excluding -Inf and +Inf).

converged Logical flag indicating whether the algorithm converged within `max_iterations`.

iterations Integer count of iterations performed during optimization.

total_iv Numeric scalar representing the total Information Value (sum of all bin IVs).

monotonicity Character string indicating monotonicity status: `"increasing"`, `"decreasing"`, or `"none"`.

Author(s)

Lopes, J. E. (implemented algorithm based on Mironchyk & Tchistiakov, 2017)

References

- Zeng, G. (2014). "A Necessary Condition for a Good Binning Algorithm in Credit Scoring". *Applied Mathematical Sciences*, 8(65), 3229-3242.
- Mironchyk, P., & Tchistiakov, V. (2017). "Monotone optimal binning algorithm for credit risk modeling". *Frontiers in Applied Mathematics and Statistics*, 3, 2.
- Belotti, P., Bonami, P., Fischetti, M., Lodi, A., Monaci, M., Nogales-Gómez, A., & Salvagnin, D. (2016). "On handling indicator constraints in mixed integer programming". *Computational Optimization and Applications*, 65(3), 545-566.
- Thomas, L. C., Edelman, D. B., & Crook, J. N. (2002). *Credit Scoring and Its Applications*. SIAM.
- Louzada, F., Ara, A., & Fernandes, G. B. (2016). "Classification methods applied to credit scoring: Systematic review and overall comparison". *Surveys in Operations Research and Management Science*, 21(2), 117-134.
- Naeem, B., Huda, N., & Aziz, A. (2013). "Developing Scorecards with Constrained Logistic Regression". *Proceedings of the International Workshop on Data Mining Applications*.

See Also

ob_numerical_ldb for density-based binning, ob_numerical_mdlp for entropy-based discretization with MDLP criterion.

Examples

```
# Simulate non-monotonic credit scoring data
set.seed(123)
n <- 8000
feature <- c(
  rnorm(2000, mean = 550, sd = 60), # High-risk segment (low scores)
  rnorm(3000, mean = 680, sd = 50), # Medium-risk segment
  rnorm(2000, mean = 720, sd = 40), # Low-risk segment
  rnorm(1000, mean = 620, sd = 55) # Mixed segment (creates non-monotonicity)
)
target <- c(
  rbinom(2000, 1, 0.25), # 25% default rate
  rbinom(3000, 1, 0.10), # 10% default rate
  rbinom(2000, 1, 0.03), # 3% default rate
  rbinom(1000, 1, 0.15) # 15% default rate (violates monotonicity)
)

# Apply MBLP with automatic monotonicity detection
result_auto <- ob_numerical_mblp(
  feature = feature,
  target = target,
  min_bins = 3,
  max_bins = 5,
  bin_cutoff = 0.05,
  max_n_prebins = 20,
  force_monotonic_direction = 0 # Auto-detect
)
```

```
print(result_auto$monotonicity) # Check detected direction
print(result_auto$total_iv) # Should be > 0.1 for predictive features

# Force decreasing monotonicity (higher score = lower WoE = lower risk)
result_forced <- ob_numerical_mblp(
  feature = feature,
  target = target,
  min_bins = 4,
  max_bins = 6,
  force_monotonic_direction = -1 # Force decreasing
)

# Verify monotonicity enforcement
stopifnot(all(diff(result_forced$woe) <= 1e-9)) # Should be non-increasing

# Compare convergence
cat(sprintf(
  "Auto mode: %d iterations, IV = %.4f\n",
  result_auto$iterations, result_auto$total_iv
))
cat(sprintf(
  "Forced mode: %d iterations, IV = %.4f\n",
  result_forced$iterations, result_forced$total_iv
))

# Visualize binning quality
oldpar <- par(mfrow = c(1, 2))
plot(result_auto$woe,
  type = "b", col = "blue", pch = 19,
  xlab = "Bin", ylab = "WoE", main = "Auto-Detected Monotonicity"
)
plot(result_forced$woe,
  type = "b", col = "red", pch = 19,
  xlab = "Bin", ylab = "WoE", main = "Forced Decreasing"
)
par(oldpar)
```

`ob_numerical_mdlp` *Optimal Binning for Numerical Features using Minimum Description Length Principle*

Description

Implements the Minimum Description Length Principle (MDLP) for supervised discretization of numerical features. MDLP balances model complexity (number of bins) and data fit (information gain) through a rigorous information-theoretic framework, automatically determining the optimal number of bins without arbitrary thresholds.

Unlike heuristic methods, MDLP provides a **theoretically grounded stopping criterion** based on the trade-off between encoding the binning structure and encoding the data given that structure. This makes it particularly robust against overfitting in noisy datasets.

Usage

```
ob_numerical_mdlp(
  feature,
  target,
  min_bins = 3,
  max_bins = 5,
  bin_cutoff = 0.05,
  max_n_prebins = 20,
  convergence_threshold = 1e-06,
  max_iterations = 1000,
  laplace_smoothing = 0.5
)
```

Arguments

`feature`	Numeric vector of feature values to be binned. Missing values (NA) are automatically removed during preprocessing. Infinite values trigger a warning but are handled internally.
`target`	Integer vector of binary target values (must contain only 0 and 1). Must have the same length as `feature`.
`min_bins`	Minimum number of bins to generate (default: 3). Must be at least 2. If the number of unique feature values is less than `min_bins`, the algorithm adjusts automatically.
`max_bins`	Maximum number of bins to generate (default: 5). Must be greater than or equal to `min_bins`. Acts as a hard constraint after MDLP optimization. When the MDL criterion would stop at more bins than `max_bins` allows, merging continues past the MDL optimum, each step taking the pair with the smallest increase in MDL cost, until the cap is met. `min_bins` is never violated to satisfy it.
`bin_cutoff`	Minimum fraction of total observations required in each bin (default: 0.05). Bins with frequency below this threshold are merged with adjacent bins to ensure statistical reliability. Must be in the range (0, 1).
`max_n_prebins`	Maximum number of pre-bins before MDLP optimization (default: 20). Higher values allow finer granularity but increase computational cost. Must be at least 2.
`convergence_threshold`	Convergence threshold for iterative optimization (default: 1e-6). Currently used internally for future extensions; MDLP convergence is primarily determined by the MDL cost function.
`max_iterations`	Maximum number of iterations for bin merging operations (default: 1000). Prevents infinite loops in pathological cases. A warning is issued if this limit is reached.

`laplace_smoothing`
: Laplace smoothing parameter for WoE calculation (default: 0.5). Prevents division by zero and stabilizes WoE estimates in bins with zero counts for one class. Must be non-negative. Higher values increase regularization but may dilute signal in small bins.

Details

Algorithm Overview

The MDLP algorithm executes in five sequential phases:

Phase 1: Data Preparation and Validation

Input data is validated for:

- Binary target (only 0 and 1 values)
- Parameter consistency (`min_bins` <= `max_bins`, valid ranges)
- Missing value detection (NaN/Inf are filtered out with a warning)

Feature-target pairs are sorted by feature value in ascending order, enabling efficient bin assignment via linear scan.

Phase 2: Equal-Frequency Pre-binning

Initial bins are created by dividing the sorted data into approximately equal-sized groups:

$$n_{\text{records/bin}} = \max\left(1, \left\lfloor \frac{N}{\text{max_n_prebins}} \right\rfloor\right)$$

This ensures each pre-bin has sufficient observations for stable entropy estimation. Bin boundaries are set to feature values at split points, with first and last boundaries at $-\infty$ and $+\infty$.

For each bin i, Shannon entropy is computed:

$$H(S_i) = -p_i \log_2(p_i) - q_i \log_2(q_i)$$

where $p_i = n_i^+ / n_i$ (proportion of positives) and $q_i = 1 - p_i$. Pure bins ($p_i = 0$ or $p_i = 1$) have $H(S_i) = 0$.

Performance Note: Entropy calculation uses a precomputed lookup table for bin counts 0-100, achieving 30-50% speedup compared to runtime computation.

Phase 3: MDL-Based Greedy Merging

The core optimization minimizes the Minimum Description Length, defined as:

$$\text{MDL}(k) = L_{\text{model}}(k) + L_{\text{data}}(k)$$

where:

- **Model Cost**: $L_{\text{model}}(k) = \log_2(k - 1)$
  Encodes the number of bins. Increases logarithmically with bin count, penalizing complex models.

- **Data Cost**: $L_{\text{data}}(k) = N \cdot H(S_{\text{total}}) - \sum_{i=1}^{k} n_i \cdot H(S_i)$
  Measures unexplained uncertainty after binning. Lower values indicate better class separation.

The algorithm iteratively evaluates all $k-1$ adjacent bin pairs, computing MDL$(k-1)$ for each potential merge. The pair minimizing MDL cost is merged, continuing until:

1. $k =$ min_bins, or
2. No merge reduces MDL cost (local optimum), or
3. `max_iterations` is reached

Theoretical Guarantee (Fayyad & Irani, 1993): The MDL criterion provides a **consistent estimator** of the true discretization complexity under mild regularity conditions, unlike ad-hoc stopping rules.

Phase 4: Rare Bin Handling

Bins with frequency $n_i/N <$ bin_cutoff are merged with adjacent bins. The merge direction (left or right) is chosen by minimizing post-merge entropy:

$$\text{direction} = \arg\min_{d \in \{\text{left,right}\}} H(S_i \cup S_{i+d})$$

This preserves class homogeneity while ensuring statistical reliability.

Phase 5: Monotonicity Enforcement (Optional)

If WoE values violate monotonicity ($\text{WoE}_i < \text{WoE}_{i-1}$), bins are iteratively merged until:

$$\text{WoE}_1 \leq \text{WoE}_2 \leq \cdots \leq \text{WoE}_k$$

Merge decisions prioritize preserving Information Value:

$$\Delta\text{IV} = \text{IV}_i + \text{IV}_{i+1} - \text{IV}_{\text{merged}}$$

Merges proceed only if $\text{IV}_{\text{merged}} \geq 0.5 \times (\text{IV}_i + \text{IV}_{i+1})$.

Weight of Evidence Computation

WoE for bin i includes Laplace smoothing to handle zero counts:

$$\text{WoE}_i = \ln\left(\frac{n_i^+ + \alpha}{n^+ + k\alpha} \Big/ \frac{n_i^- + \alpha}{n^- + k\alpha}\right)$$

where $\alpha =$ laplace_smoothing and k is the number of bins.

Edge cases:

- If $n_i^+ + \alpha = n_i^- + \alpha = 0$: $\text{WoE}_i = 0$
- If $n_i^+ + \alpha = 0$: $\text{WoE}_i = -20$ (capped)
- If $n_i^- + \alpha = 0$: $\text{WoE}_i = +20$ (capped)

Information Value is computed as:

$$\text{IV}_i = \left(\frac{n_i^+}{n^+} - \frac{n_i^-}{n^-} \right) \times \text{WoE}_i$$

Comparison with Other Methods

Method	Stopping Criterion	Optimality
MDLP	Information-theoretic (MDL cost)	Local optimum with theoretical guarantees
LDB	Heuristic (density minima)	No formal optimality
MBLP	Heuristic (IV loss threshold)	Greedy approximation
ChiMerge	Statistical (χ^2 test)	Dependent on significance level

Computational Complexity

- Sorting: $O(N \log N)$
- Pre-binning: $O(N)$
- MDL optimization: $O(k^3 \times I)$ where I is the number of merge iterations (typically $I \approx k$)
- Total: $O(N \log N + k^3 \times I)$

For typical credit scoring datasets ($N \sim 10^5$, $k \sim 5$), runtime is dominated by sorting.

Value

A list containing:

id Integer vector of bin identifiers (1-based indexing).

bin Character vector of bin intervals in the format `"[lower;upper)"`. The first bin starts with `-Inf` and the last bin ends with `+Inf`.

woe Numeric vector of Weight of Evidence values for each bin, computed with Laplace smoothing.

iv Numeric vector of Information Value contributions for each bin.

count Integer vector of total observations in each bin.

count_pos Integer vector of positive class (target = 1) counts per bin.

count_neg Integer vector of negative class (target = 0) counts per bin.

cutpoints Numeric vector of cutpoints defining bin boundaries (excluding -Inf and +Inf). These are the upper bounds of bins 1 to k-1.

total_iv Numeric scalar representing the total Information Value (sum of all bin IVs).

converged Logical flag indicating whether the algorithm converged. Set to `FALSE` if `max_iterations` was reached during any merging phase.

iterations Integer count of iterations performed across all optimization phases (MDL merging, rare bin merging, monotonicity enforcement).

Author(s)

Lopes, J. E. (algorithm implementation based on Fayyad & Irani, 1993)

References

- Fayyad, U. M., & Irani, K. B. (1993). "Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning". *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1022-1027.
- Rissanen, J. (1978). "Modeling by shortest data description". *Automatica*, 14(5), 465-471.
- Shannon, C. E. (1948). "A Mathematical Theory of Communication". *Bell System Technical Journal*, 27(3), 379-423.
- Dougherty, J., Kohavi, R., & Sahami, M. (1995). "Supervised and Unsupervised Discretization of Continuous Features". *Proceedings of the 12th International Conference on Machine Learning (ICML)*, pp. 194-202.
- Witten, I. H., Frank, E., & Hall, M. A. (2011). *Data Mining: Practical Machine Learning Tools and Techniques* (3rd ed.). Morgan Kaufmann.
- Cerqueira, V., & Torgo, L. (2019). "Automatic Feature Engineering for Predictive Modeling of Multivariate Time Series". arXiv:1910.01344.

See Also

ob_numerical_ldb for density-based binning, ob_numerical_mblp for monotonicity-constrained binning.

Examples

```
# Simulate overdispersed credit scoring data with noise
set.seed(2024)
n <- 10000

# Create feature with multiple regimes and noise
feature <- c(
  rnorm(3000, mean = 580, sd = 70), # High-risk cluster
  rnorm(4000, mean = 680, sd = 50), # Medium-risk cluster
  rnorm(2000, mean = 740, sd = 40), # Low-risk cluster
  runif(1000, min = 500, max = 800) # Noise (uniform distribution)
)

target <- c(
  rbinom(3000, 1, 0.30), # 30% default rate
  rbinom(4000, 1, 0.12), # 12% default rate
  rbinom(2000, 1, 0.04), # 4% default rate
  rbinom(1000, 1, 0.15) # Noisy segment
)

# Apply MDLP with default parameters
result <- ob_numerical_mdlp(
  feature = feature,
  target = target,
  min_bins = 3,
  max_bins = 5,
  bin_cutoff = 0.05,
  max_n_prebins = 20
```

```
)

# Inspect results
print(result$bin)
print(data.frame(
  Bin = result$bin,
  WoE = round(result$woe, 4),
  IV = round(result$iv, 4),
  Count = result$count
))

cat(sprintf("\nTotal IV: %.4f\n", result$total_iv))
cat(sprintf("Converged: %s\n", result$converged))
cat(sprintf("Iterations: %d\n", result$iterations))

# Verify monotonicity
is_monotonic <- all(diff(result$woe) >= -1e-10)
cat(sprintf("WoE Monotonic: %s\n", is_monotonic))

# Compare with different Laplace smoothing
result_nosmooth <- ob_numerical_mdlp(
  feature = feature,
  target = target,
  laplace_smoothing = 0.0 # No smoothing (risky for rare bins)
)

result_highsmooth <- ob_numerical_mdlp(
  feature = feature,
  target = target,
  laplace_smoothing = 2.0 # Higher regularization
)

# Compare WoE stability
data.frame(
  Bin = seq_along(result$woe),
  WoE_default = result$woe,
  WoE_no_smooth = result_nosmooth$woe,
  WoE_high_smooth = result_highsmooth$woe
)

# Visualize binning structure
oldpar <- par(mfrow = c(1, 2))

# WoE plot
plot(result$woe,
  type = "b", col = "blue", pch = 19,
  xlab = "Bin", ylab = "WoE",
  main = "Weight of Evidence by Bin"
)
grid()

# IV contribution plot
barplot(result$iv,
```

```
  names.arg = seq_along(result$iv),
  col = "steelblue", border = "white",
  xlab = "Bin", ylab = "IV Contribution",
  main = sprintf("Total IV = %.4f", result$total_iv)
)
grid()
par(oldpar)
```

`ob_numerical_mob` *Optimal Binning for Numerical Features using Monotonic Optimal Binning*

Description

Implements Monotonic Optimal Binning (MOB), a supervised discretization algorithm that enforces strict monotonicity in Weight of Evidence (WoE) values. MOB is designed for credit scoring and risk modeling applications where monotonicity is a **regulatory requirement** or essential for model interpretability and stakeholder acceptance.

Unlike heuristic methods that treat monotonicity as a post-processing step, MOB integrates monotonicity constraints into the core optimization loop, ensuring that the final binning satisfies: $\text{WoE}_1 \leq \text{WoE}_2 \leq \cdots \leq \text{WoE}_k$ (or the reverse for decreasing patterns).

Usage

```
ob_numerical_mob(
  feature,
  target,
  min_bins = 3,
  max_bins = 5,
  bin_cutoff = 0.05,
  max_n_prebins = 20,
  convergence_threshold = 1e-06,
  max_iterations = 1000,
  laplace_smoothing = 0.5
)
```

Arguments

`feature`	Numeric vector of feature values to be binned. Missing values (NA) are automatically removed during preprocessing. Infinite values trigger a warning but are handled internally.
`target`	Integer vector of binary target values (must contain only 0 and 1). Must have the same length as `feature`.
`min_bins`	Minimum number of bins to generate (default: 3). Must be at least 2. Acts as a hard constraint during monotonicity enforcement; the algorithm will not merge below this threshold even if violations persist.

`max_bins` Maximum number of bins to generate (default: 5). Must be greater than or equal to `min_bins`. The algorithm reduces bins via greedy merging if the initial count exceeds this limit.

`bin_cutoff` Minimum fraction of total observations required in each bin (default: 0.05). Bins with frequency below this threshold are merged with adjacent bins. Must be in the range (0, 1).

`max_n_prebins` Maximum number of pre-bins before optimization (default: 20). Controls granularity of initial equal-frequency discretization. Must be at least equal to `min_bins`.

`convergence_threshold`
Convergence threshold for iterative optimization (default: 1e-6). Reserved for future extensions; current implementation uses `max_iterations` as the primary stopping criterion.

`max_iterations` Maximum number of iterations for bin merging and monotonicity enforcement (default: 1000). Prevents infinite loops in pathological cases. A warning is issued if this limit is reached without achieving convergence.

`laplace_smoothing`
Laplace smoothing parameter for WoE calculation (default: 0.5). Prevents division by zero and stabilizes WoE estimates in bins with zero counts for one class. Must be non-negative. Standard values: 0.5 (Laplace), 1.0 (Jeffreys prior).

Details

Algorithm Overview

The MOB algorithm executes in five sequential phases with strict monotonicity enforcement integrated throughout:

Phase 1: Equal-Frequency Pre-binning

Initial bins are created by dividing sorted data into approximately equal-sized groups:

$$n_{\text{bin}} = \left\lfloor \frac{N}{\min(\text{max_n_prebins}, n_{\text{unique}})} \right\rfloor$$

Bin boundaries are set to feature values at split points, ensuring no gaps between consecutive bins. First and last boundaries are set to $-\infty$ and $+\infty$.

This approach balances statistical stability (sufficient observations per bin) with granularity (ability to detect local patterns).

Phase 2: Rare Bin Merging

Bins with total count below `bin_cutoff` $\times N$ are iteratively merged. The merge direction (left or right) is chosen to minimize Information Value loss:

$$\text{direction} = \arg\min_{d \in \{\text{left,right}\}} (\text{IV}_{\text{before}} - \text{IV}_{\text{after merge}})$$

where:

$$\text{IV}_{\text{before}} = \text{IV}_i + \text{IV}_{i+d}$$

$$\text{IV}_{\text{after}} = (\text{DistGood}_{\text{merged}} - \text{DistBad}_{\text{merged}}) \times \text{WoE}_{\text{merged}}$$

Merging continues until all bins meet the frequency threshold or `min_bins` is reached.

Phase 3: Initial WoE/IV Calculation

Weight of Evidence for each bin i is computed with Laplace smoothing:

$$\text{WoE}_i = \ln\left(\frac{n_i^+ + \alpha}{n^+ + k\alpha} \Big/ \frac{n_i^- + \alpha}{n^- + k\alpha}\right)$$

where $\alpha =$ laplace_smoothing and k is the current number of bins. Information Value is:

$$\text{IV}_i = \left(\frac{n_i^+ + \alpha}{n^+ + k\alpha} - \frac{n_i^- + \alpha}{n^- + k\alpha}\right) \times \text{WoE}_i$$

Edge case handling:

- If both distributions approach zero: $\text{WoE}_i = 0$
- If only positive distribution is zero: $\text{WoE}_i = -20$ (capped)
- If only negative distribution is zero: $\text{WoE}_i = +20$ (capped)

Phase 4: Monotonicity Enforcement

The algorithm first determines the desired monotonicity direction by examining the relationship between the first two bins:

$$\text{should_increase} = \begin{cases} \text{TRUE} & \text{if } \text{WoE}_1 \geq \text{WoE}_0 \\ \text{FALSE} & \text{otherwise} \end{cases}$$

For each bin i from 1 to $k-1$, violations are detected as:

$$\text{violation} = \begin{cases} \text{WoE}_i < \text{WoE}_{i-1} & \text{if should_increase} \\ \text{WoE}_i > \text{WoE}_{i-1} & \text{if } \neg\text{should_increase} \end{cases}$$

When a violation is found at index i, the algorithm attempts two merge strategies:

1. **Merge with previous bin**: Combine bins $i-1$ and i, then verify the merged bin's WoE is compatible with neighbors:

$$\text{WoE}_{i-2} \leq \text{WoE}_{\text{merged}} \leq \text{WoE}_{i+1} \quad \text{(if should_increase)}$$

2. **Merge with next bin**: If strategy 1 fails, merge bins i and $i+1$.

Merging continues iteratively until either:

- All WoE values satisfy monotonicity constraints
- The number of bins reaches `min_bins`
- `max_iterations` is exceeded (triggers warning)

After each merge, WoE and IV are recalculated for all bins to reflect updated distributions.

Phase 5: Bin Count Reduction

If the number of bins exceeds `max_bins` after monotonicity enforcement, additional merges are performed. The algorithm identifies the pair of adjacent bins that minimizes IV loss when merged:

$$\text{merge_idx} = \arg\min_{i=0}^{k-2} (\text{IV}_i + \text{IV}_{i+1} - \text{IV}_{\text{merged}})$$

This greedy approach continues until $k \leq$ max_bins.

Theoretical Foundations

- **Monotonicity as Stability Criterion**: Zeng (2014) proves that non-monotonic WoE patterns are unstable under population drift, leading to unreliable predictions when the data distribution shifts.
- **Regulatory Compliance**: Basel II/III validation requirements (BCBS, 2005) explicitly require monotonic relationships between risk drivers and probability of default for IRB models.
- **Information Preservation**: While enforcing monotonicity reduces model flexibility, Mironchyk & Tchistiakov (2017) demonstrate that the IV loss is typically < 5% compared to unconstrained binning for real credit portfolios.

Comparison with Related Methods

Method	Monotonicity	Enforcement	Use Case
MOB	Guaranteed	During optimization	Regulatory scorecards
MBLP	Target	Iterative post-process	General credit models
MDLP	Optional	Post-hoc merging	Exploratory analysis
LDB	Optional	Post-hoc merging	Research/prototyping

Computational Complexity

- Sorting: $O(N \log N)$
- Pre-binning: $O(N)$
- Rare bin merging: $O(k^2 \times I_{\text{rare}})$ where I_{rare} is the number of rare bins
- Monotonicity enforcement: $O(k^2 \times I_{\text{mono}})$ where I_{mono} is the number of violations (worst case: $O(k)$)
- Bin reduction: $O(k \times (k_{\text{initial}} - \text{max_bins}))$
- Total: $O(N \log N + k^2 \times \text{max_iterations})$

For typical credit scoring datasets ($N \sim 10^5$, $k \sim 5$), runtime is dominated by sorting. Pathological cases (e.g., perfectly alternating WoE values) may require $O(k^2)$ merges.

Value

A list containing:

id Integer vector of bin identifiers (1-based indexing).

bin Character vector of bin intervals in the format `"[lower;upper)"`. The first bin starts with `-Inf` and the last bin ends with `+Inf`.

woe Numeric vector of Weight of Evidence values for each bin. Guaranteed to be monotonic (either non-decreasing or non-increasing).

iv Numeric vector of Information Value contributions for each bin.

count Integer vector of total observations in each bin.

count_pos Integer vector of positive class (target = 1) counts per bin.

count_neg Integer vector of negative class (target = 0) counts per bin.

event_rate Numeric vector of event rates (proportion of positives) per bin.

cutpoints Numeric vector of cutpoints defining bin boundaries (excluding -Inf and +Inf). These are the upper bounds of bins 1 to k-1.

total_iv Numeric scalar representing the total Information Value (sum of all bin IVs).

converged Logical flag indicating whether the algorithm converged within `max_iterations`. `FALSE` indicates the iteration limit was reached during rare bin merging or monotonicity enforcement.

iterations Integer count of iterations performed across all optimization phases (rare bin merging + monotonicity enforcement + bin reduction).

Author(s)

Lopes, J. E. (algorithm implementation based on Mironchyk & Tchistiakov, 2017)

References

- Mironchyk, P., & Tchistiakov, V. (2017). "Monotone optimal binning algorithm for credit risk modeling". *Frontiers in Applied Mathematics and Statistics*, 3, 2.
- Zeng, G. (2014). "A Necessary Condition for a Good Binning Algorithm in Credit Scoring". *Applied Mathematical Sciences*, 8(65), 3229-3242.
- Thomas, L. C., Edelman, D. B., & Crook, J. N. (2002). *Credit Scoring and Its Applications*. SIAM.
- Siddiqi, N. (2006). *Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring*. Wiley.
- Basel Committee on Banking Supervision (2005). "Studies on the Validation of Internal Rating Systems". *Bank for International Settlements Working Paper No. 14*.
- Naeem, B., Huda, N., & Aziz, A. (2013). "Developing Scorecards with Constrained Logistic Regression". *Proceedings of the International Workshop on Data Mining Applications*.

See Also

`ob_numerical_mblp` for monotonicity-targeted binning with correlation-based direction detection, `ob_numerical_mdlp` for information-theoretic binning without monotonicity constraints.

Examples

```
# Simulate non-monotonic credit scoring data
set.seed(42)
n <- 12000

# Create feature with inherent monotonic relationship + noise
feature <- c(
  rnorm(4000, mean = 600, sd = 50), # Low scores (high risk)
  rnorm(5000, mean = 680, sd = 45), # Medium scores
  rnorm(3000, mean = 740, sd = 35) # High scores (low risk)
)

target <- c(
  rbinom(4000, 1, 0.25), # 25% default
  rbinom(5000, 1, 0.10), # 10% default
  rbinom(3000, 1, 0.03) # 3% default
)

# Apply MOB
result <- ob_numerical_mob(
  feature = feature,
  target = target,
  min_bins = 2,
  max_bins = 5,
  bin_cutoff = 0.05,
  max_n_prebins = 20
)

# Verify monotonicity
print(result$woe)
stopifnot(all(diff(result$woe) <= 1e-10)) # Non-increasing WoE

# Inspect binning quality
binning_table <- data.frame(
  Bin = result$bin,
  WoE = round(result$woe, 4),
  IV = round(result$iv, 4),
  Count = result$count,
  EventRate = round(result$event_rate, 4)
)
print(binning_table)

cat(sprintf("\nTotal IV: %.4f\n", result$total_iv))
cat(sprintf(
  "Converged: %s (iterations: %d)\n",
  result$converged, result$iterations
))

# Visualize monotonic pattern
oldpar <- par(mfrow = c(1, 2))

# WoE monotonicity
```

```
plot(result$woe,
  type = "b", col = "darkgreen", pch = 19, lwd = 2,
  xlab = "Bin", ylab = "WoE",
  main = "Guaranteed Monotonic WoE"
)
grid()

# Event rate vs WoE relationship
plot(result$event_rate, result$woe,
  pch = 19, col = "steelblue",
  xlab = "Event Rate", ylab = "WoE",
  main = "WoE vs Event Rate"
)
abline(lm(result$woe ~ result$event_rate), col = "red", lwd = 2)
grid()
par(oldpar)
```

`ob_numerical_mrblp` *Optimal Binning for Numerical Features using Monotonic Risk Binning with Likelihood Ratio Pre-binning*

Description

Implements a greedy binning algorithm with monotonicity enforcement and **majority-vote direction detection**. *Important Note*: Despite the "Likelihood Ratio Pre-binning" designation in the name, the current implementation uses **equal-frequency pre-binning** without likelihood ratio statistics. The algorithm is functionally a variant of Monotonic Optimal Binning (MOB) with minor differences in merge strategies.

This method is suitable for credit scoring applications requiring monotonic WoE patterns, but users should be aware that it does not employ the statistical rigor implied by "Likelihood Ratio" in the name.

Usage

```
ob_numerical_mrblp(
  feature,
  target,
  min_bins = 3,
  max_bins = 5,
  bin_cutoff = 0.05,
  max_n_prebins = 20,
  convergence_threshold = 1e-06,
  max_iterations = 1000,
  laplace_smoothing = 0.5
)
```

Arguments

`feature`	Numeric vector of feature values to be binned. Missing values (NA) and infinite values are **not permitted** and will trigger an error (unlike other binning methods that issue warnings).
`target`	Integer vector of binary target values (must contain only 0 and 1). Must have the same length as `feature`.
`min_bins`	Minimum number of bins to generate (default: 3). Must be at least 2. Acts as a hard constraint during monotonicity enforcement.
`max_bins`	Maximum number of bins to generate (default: 5). Must be greater than or equal to `min_bins`.
`bin_cutoff`	Minimum fraction of total observations required in each bin (default: 0.05). Bins with frequency below this threshold are merged. Must be in the range (0, 1).
`max_n_prebins`	Maximum number of pre-bins before optimization (default: 20). Must be at least equal to `min_bins`.
`convergence_threshold`	Convergence threshold (default: 1e-6). Currently used to check if WoE range is below threshold; primary stopping criterion is `max_iterations`.
`max_iterations`	Maximum number of iterations for bin merging and monotonicity enforcement (default: 1000). Prevents infinite loops.
`laplace_smoothing`	Laplace smoothing parameter for WoE calculation (default: 0.5). Must be non-negative.

Details

Algorithm Overview

The MRBLP algorithm executes in five phases:

Phase 1: Equal-Frequency Pre-binning

Initial bins are created by dividing sorted data into approximately equal-sized groups:

$$n_{\text{bin}} = \max\left(1, \left\lfloor \frac{N}{\text{max_n_prebins}} \right\rfloor\right)$$

Note: Despite "Likelihood Ratio Pre-binning" in the name, no likelihood ratio statistics are computed. A true likelihood ratio approach would compute:

$$\text{LR}(c) = \prod_{x \le c} \frac{P(x|y=1)}{P(x|y=0)} \times \prod_{x > c} \frac{P(x|y=1)}{P(x|y=0)}$$

and select cutpoints c that maximize $|\log \text{LR}(c)|$. This is **not implemented** in the current version.

Phase 2: Rare Bin Merging

Bins with total count below `bin_cutoff` $\times N$ are merged. The merge direction (left or right) is chosen to minimize IV loss:

$$\text{direction} = \arg\min_{d \in \{\text{left,right}\}} (\text{IV}_i + \text{IV}_{i+d} - \text{IV}_{\text{merged}})$$

Phase 3: Initial WoE/IV Calculation

Weight of Evidence for bin i:

$$\text{WoE}_i = \ln\left(\frac{n_i^+ + \alpha}{n^+ + k\alpha} \Big/ \frac{n_i^- + \alpha}{n^- + k\alpha}\right)$$

where $\alpha =$ laplace_smoothing and k is the number of bins.

Phase 4: Monotonicity Enforcement

The algorithm determines the desired monotonicity direction via **majority vote**:

$$\text{increasing} = \begin{cases} \text{TRUE} & \text{if } \#\{\text{WoE}_i > \text{WoE}_{i-1}\} \geq \#\{\text{WoE}_i < \text{WoE}_{i-1}\} \\ \text{FALSE} & \text{otherwise} \end{cases}$$

This differs from:

- **MOB**: Uses first two bins only (`WoE[1] >= WoE[0]`)
- **MBLP**: Uses Pearson correlation between bin indices and WoE

Violations are detected as:

$$\text{violation} = \begin{cases} \text{WoE}_i < \text{WoE}_{i-1} & \text{if increasing} \\ \text{WoE}_i > \text{WoE}_{i-1} & \text{if decreasing} \end{cases}$$

Violating bins are merged iteratively until monotonicity is achieved or `min_bins` is reached.

Phase 5: Bin Count Reduction

If the number of bins exceeds `max_bins`, the algorithm merges bins with the **smallest absolute IV difference**:

$$\text{merge_idx} = \arg\min_{i=0}^{k-2} |\text{IV}_i - \text{IV}_{i+1}|$$

Critique: This criterion assumes bins with similar IVs are redundant, which is not theoretically justified. A more rigorous approach (used in MBLP) minimizes IV loss **after merge**:

$$\Delta\text{IV} = \text{IV}_i + \text{IV}_{i+1} - \text{IV}_{\text{merged}}$$

Theoretical Foundations

- **Monotonicity Enforcement**: Based on Zeng (2014), ensuring stability under data distribution shifts.
- **Likelihood Ratio (Theoretical)**: Neyman-Pearson lemma establishes likelihood ratio as the optimal test statistic for hypothesis testing. For binning, cutpoints maximizing LR would theoretically yield optimal class separation. **However, this is not implemented**.

- **Practical Equivalence**: The algorithm is functionally equivalent to MOB with minor differences in direction detection and merge strategies.

Comparison with Related Methods

Method	Pre-binning	Direction Detection	Merge Criterion
MRBLP	Equal-frequency	Majority vote	Min IV difference
MOB	Equal-frequency	First two bins	Min IV loss
MBLP	Quantile-based	Pearson correlation	Min IV loss
MDLP	Equal-frequency	N/A (optional)	MDL cost

Computational Complexity

Identical to MOB: $O(N \log N + k^2 \times \text{max_iterations})$

When to Use MRBLP vs Alternatives

- **Use MRBLP**: If you specifically need majority-vote direction detection and can tolerate the non-standard merge criterion.
- **Use MOB**: For simplicity and slightly faster direction detection.
- **Use MBLP**: For more robust direction detection via correlation.
- **Use MDLP**: For information-theoretic optimality without mandatory monotonicity.

Value

A list containing:

id Integer vector of bin identifiers (1-based indexing).

bin Character vector of bin intervals in the format `"[lower;upper)"`.

woe Numeric vector of Weight of Evidence values. Guaranteed to be monotonic.

iv Numeric vector of Information Value contributions per bin.

count Integer vector of total observations per bin.

count_pos Integer vector of positive class counts per bin.

count_neg Integer vector of negative class counts per bin.

event_rate Numeric vector of event rates per bin.

cutpoints Numeric vector of bin boundaries (excluding -Inf and +Inf).

total_iv Total Information Value (sum of bin IVs).

converged Logical flag indicating convergence within `max_iterations`.

iterations Integer count of iterations performed.

Author(s)

Lopes, J. E.

References

- Neyman, J., & Pearson, E. S. (1933). "On the Problem of the Most Efficient Tests of Statistical Hypotheses". *Philosophical Transactions of the Royal Society A*, 231(694-706), 289-337. [Theoretical foundation for likelihood ratio, not implemented in code]
- Mironchyk, P., & Tchistiakov, V. (2017). "Monotone optimal binning algorithm for credit risk modeling". *Frontiers in Applied Mathematics and Statistics*, 3, 2.
- Zeng, G. (2014). "A Necessary Condition for a Good Binning Algorithm in Credit Scoring". *Applied Mathematical Sciences*, 8(65), 3229-3242.
- Siddiqi, N. (2006). *Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring*. Wiley.
- Anderson, R. (2007). *The Credit Scoring Toolkit: Theory and Practice for Retail Credit Risk Management and Decision Automation*. Oxford University Press.
- Hosmer, D. W., Lemeshow, S., & Sturdivant, R. X. (2013). *Applied Logistic Regression* (3rd ed.). Wiley.

See Also

ob_numerical_mob for the base monotonic binning algorithm, ob_numerical_mblp for correlation-based direction detection, ob_numerical_mdlp for information-theoretic binning.

Examples

```
# Simulate credit scoring data
set.seed(2024)
n <- 10000
feature <- c(
  rnorm(4000, mean = 620, sd = 50),
  rnorm(4000, mean = 690, sd = 45),
  rnorm(2000, mean = 740, sd = 35)
)
target <- c(
  rbinom(4000, 1, 0.20),
  rbinom(4000, 1, 0.10),
  rbinom(2000, 1, 0.04)
)

# Apply MRBLP
result <- ob_numerical_mrblp(
  feature = feature,
  target = target,
  min_bins = 3,
  max_bins = 5
)

# Compare with MOB (should be very similar)
result_mob <- ob_numerical_mob(
  feature = feature,
  target = target,
  min_bins = 3,
```

```
  max_bins = 5
)

# Compare results
data.frame(
  Method = c("MRBLP", "MOB"),
  N_Bins = c(length(result$woe), length(result_mob$woe)),
  Total_IV = c(result$total_iv, result_mob$total_iv),
  Iterations = c(result$iterations, result_mob$iterations)
)
```

`ob_numerical_oslp` *Optimal Binning for Numerical Variables using Optimal Supervised Learning Partitioning*

Description

Implements a greedy binning algorithm with quantile-based pre-binning and monotonicity enforcement. **Important Note**: Despite "Optimal Supervised Learning Partitioning" and "LP" in the name, the algorithm uses **greedy heuristics** without formal Linear Programming or convex optimization. The method is functionally equivalent to `ob_numerical_mrblp` with minor differences in pre-binning strategy and bin reduction criteria.

Users seeking true optimization-based binning should consider Mixed-Integer Programming (MIP) implementations (e.g., via `ompr` or `lpSolve` packages), though these scale poorly beyond N > 10,000 observations.

Usage

```
ob_numerical_oslp(
  feature,
  target,
  min_bins = 3,
  max_bins = 5,
  bin_cutoff = 0.05,
  max_n_prebins = 20,
  convergence_threshold = 1e-06,
  max_iterations = 1000,
  laplace_smoothing = 0.5
)
```

Arguments

`feature`	Numeric vector of feature values. Missing values (NA) and infinite values are **not permitted** and will trigger an error.

`target`	Integer or numeric vector of binary target values (must contain only 0 and 1). Must have the same length as `feature`. Unlike other binning methods, OSLP internally uses `double` for target, allowing implicit conversion from integer.
`min_bins`	Minimum number of bins (default: 3). Must be at least 2.
`max_bins`	Maximum number of bins (default: 5). Must be greater than or equal to `min_bins`.
`bin_cutoff`	Minimum fraction of total observations per bin (default: 0.05). Must be in (0, 1).
`max_n_prebins`	Maximum number of pre-bins (default: 20). Must be at least equal to `min_bins`.
`convergence_threshold`	Convergence threshold for IV change (default: 1e-6).
`max_iterations`	Maximum iterations (default: 1000).
`laplace_smoothing`	Laplace smoothing parameter (default: 0.5). Must be non-negative.

Details

Algorithm Overview

OSLP executes in five phases:

Phase 1: Quantile-Based Pre-binning

Unlike equal-frequency methods that ensure balanced bin sizes, OSLP places cutpoints at quantiles of **unique feature values**:

$$\text{edge}_i = \text{unique_vals}\left[\lfloor p_i \times (n_{\text{unique}} - 1) \rfloor\right]$$

where $p_i = i/\text{max_n_prebins}$.

Critique: If unique values are clustered (e.g., many observations at specific values), bins may have vastly different sizes, violating the equal-frequency principle that ensures statistical stability.

Phase 2: Rare Bin Merging

Bins with $n_i/N <$ bin_cutoff are merged. The merge direction minimizes IV loss:

$$\Delta\text{IV} = \text{IV}_i + \text{IV}_{i+d} - \text{IV}_{\text{merged}}$$

where $d \in \{-1, +1\}$ (left or right neighbor).

Phase 3: Initial WoE/IV Calculation

Standard WoE with Laplace smoothing:

$$\text{WoE}_i = \ln\left(\frac{n_i^+ + \alpha}{n^+ + k\alpha} \Big/ \frac{n_i^- + \alpha}{n^- + k\alpha}\right)$$

Phase 4: Monotonicity Enforcement

Direction determined via majority vote (identical to MRBLP):

$$\text{increasing} = \begin{cases} \text{TRUE} & \text{if } \sum_i \mathbb{1}_{\{\text{WoE}_i > \text{WoE}_{i-1}\}} \geq \sum_i \mathbb{1}_{\{\text{WoE}_i < \text{WoE}_{i-1}\}} \\ \text{FALSE} & \text{otherwise} \end{cases}$$

Violations are merged iteratively.

Phase 5: Bin Count Reduction

If $k >$ max_bins, merge bins with the **smallest combined IV**:

$$\text{merge_idx} = \arg\min_{i=0}^{k-2} (\text{IV}_i + \text{IV}_{i+1})$$

Rationale: Assumes bins with low total IV contribute least to predictive power. However, this ignores the interaction between bins; a low-IV bin may be essential for monotonicity or preventing gaps.

Theoretical Foundations

Despite the name "Optimal Supervised Learning Partitioning", the algorithm lacks:

- **Global optimality guarantees**: Greedy merging is myopic
- **Formal loss function**: No explicit objective being minimized
- **LP formulation**: No constraint matrix, simplex solver, or dual variables

A true optimal partitioning approach would formulate the problem as:

$$\min_{\mathbf{z},\mathbf{b}} \left\{ -\sum_{i=1}^{k} \text{IV}_i(\mathbf{b}) + \lambda k \right\}$$

subject to:

$$\sum_{j=1}^{k} z_{ij} = 1 \quad \forall i \in \{1, \ldots, N\}$$

$$\text{WoE}_j \leq \text{WoE}_{j+1} \quad \forall j$$

$$z_{ij} \in \{0, 1\}, \quad b_j \in \mathbb{R}$$

where z_{ij} indicates observation i assigned to bin j, and λ is a complexity penalty. This requires MILP solvers (CPLEX, Gurobi) and is intractable for $N > 10^4$.

Comparison with Related Methods

Method	Pre-binning	Direction	Merge (max_bins)	Target Type
OSLP	Quantile (unique vals)	Majority vote	Min (IV(i) + IV(i+1))	double
MRBLP	Equal-frequency	Majority vote	Min |IV(i) - IV(i+1)|	int
MOB	Equal-frequency	First two bins	Min IV loss	int
MBLP	Quantile (data)	Correlation	Min IV loss	int

When to Use OSLP

- **Use OSLP**: Never. Use MBLP or MOB instead for better pre-binning and merge strategies.
- **Use MBLP**: For robust direction detection via correlation.
- **Use MDLP**: For information-theoretic stopping criteria.
- **Use True LP**: For small datasets (N < 1000) where global optimality is critical and computational cost is acceptable.

Value

A list containing:

id Integer bin identifiers (1-based).

bin Character bin intervals `"[lower;upper)"`.

woe Numeric WoE values (guaranteed monotonic).

iv Numeric IV contributions per bin.

count Integer total observations per bin.

count_pos Integer positive class counts.

count_neg Integer negative class counts.

event_rate Numeric event rates.

cutpoints Numeric bin boundaries (excluding ±Inf).

total_iv Total Information Value.

converged Logical convergence flag.

iterations Integer iteration count.

Author(s)

Lopes, J. E.

References

- Mironchyk, P., & Tchistiakov, V. (2017). "Monotone optimal binning algorithm for credit risk modeling". *Frontiers in Applied Mathematics and Statistics*, 3, 2.
- Zeng, G. (2014). "A Necessary Condition for a Good Binning Algorithm in Credit Scoring". *Applied Mathematical Sciences*, 8(65), 3229-3242.
- Fayyad, U. M., & Irani, K. B. (1993). "Multi-Interval Discretization of Continuous-Valued Attributes". *IJCAI*, pp. 1022-1027.
- Good, I. J. (1952). "Rational Decisions". *Journal of the Royal Statistical Society B*, 14(1), 107-114.
- Siddiqi, N. (2006). *Credit Risk Scorecards*. Wiley.

See Also

`ob_numerical_mrblp` for nearly identical algorithm with better pre-binning, `ob_numerical_mblp` for correlation-based direction detection, `ob_numerical_mdlp` for information-theoretic optimality.

Examples

```
set.seed(123)
n <- 5000
feature <- c(
  rnorm(2000, 600, 50),
  rnorm(2000, 680, 40),
  rnorm(1000, 740, 30)
```

```
)
target <- c(
  rbinom(2000, 1, 0.25),
  rbinom(2000, 1, 0.10),
  rbinom(1000, 1, 0.03)
)

result <- ob_numerical_oslp(
  feature = feature,
  target = target,
  min_bins = 3,
  max_bins = 5
)

print(result$woe)
print(result$total_iv)

# Compare with MRBLP (should be nearly identical)
result_mrblp <- ob_numerical_mrblp(
  feature = feature,
  target = target,
  min_bins = 3,
  max_bins = 5
)

data.frame(
  Method = c("OSLP", "MRBLP"),
  Total_IV = c(result$total_iv, result_mrblp$total_iv),
  N_Bins = c(length(result$woe), length(result_mrblp$woe))
)
```

`ob_numerical_sketch` *Optimal Binning for Numerical Variables using Sketch-based Algorithm*

Description

Implements optimal binning using the **KLL Sketch** (Karnin, Lang, Liberty, 2016), a probabilistic data structure for quantile approximation in data streams. This is the **only method in the package** that uses a fundamentally different algorithmic approach (streaming algorithms) compared to batch processing methods (MOB, MDLP, etc.).

The sketch-based approach enables:

- **Sublinear space complexity**: O(k log N) vs O(N) for batch methods
- **Single-pass processing**: Suitable for streaming data
- **Provable approximation guarantees**: Quantile error $\epsilon \approx O(1/k)$

The method combines KLL Sketch for candidate generation with either Dynamic Programming (for small N <= 50) or greedy IV-based selection (for larger datasets), followed by monotonicity enforcement via the Pool Adjacent Violators Algorithm (PAVA).

Usage

```
ob_numerical_sketch(
  feature,
  target,
  min_bins = 3,
  max_bins = 5,
  bin_cutoff = 0.05,
  max_n_prebins = 20,
  monotonic = TRUE,
  convergence_threshold = 1e-06,
  max_iterations = 1000,
  sketch_k = 200
)
```

Arguments

`feature`	Numeric vector of feature values. Missing values (NA) are **not permitted** and will trigger an error. Infinite values (Inf, -Inf) and NaN are also not allowed.
`target`	Integer vector of binary target values (must contain only 0 and 1). Must have the same length as `feature`. Missing values are not permitted.
`min_bins`	Minimum number of bins (default: 3). Must be at least 2.
`max_bins`	Maximum number of bins (default: 5). Must be >= `min_bins`.
`bin_cutoff`	Minimum fraction of total observations per bin (default: 0.05). Must be in (0, 1). Bins with fewer observations will be merged with neighbors.
`max_n_prebins`	Maximum number of pre-bins to generate from quantiles (default: 20). This parameter controls the initial granularity of binning candidates. Higher values provide more flexibility but increase computational cost.
`monotonic`	Logical flag to enforce WoE monotonicity (default: TRUE). Uses PAVA (Pool Adjacent Violators Algorithm) for enforcement. Direction (increasing/ decreasing) is automatically detected from the data.
`convergence_threshold`	Convergence threshold for IV change (default: 1e-6). Optimization stops when the change in total IV between iterations falls below this value.
`max_iterations`	Maximum iterations for bin optimization (default: 1000). Prevents infinite loops in the optimization process.
`sketch_k`	Integer parameter controlling sketch accuracy (default: 200). Larger values improve quantile precision but increase memory usage. **Approximation error**: $\epsilon \approx 1/k$ (200 $\rightarrow$ 0.5% error). **Valid range**: [10, 1000]. Typical values: 50 (fast), 200 (balanced), 500 (precise).

Details

Algorithm Overview

The sketch-based binning algorithm executes in four phases:

Phase 1: KLL Sketch Construction

The KLL Sketch maintains a compressed, multi-level representation of the data distribution:

$$\text{Sketch} = \{\text{Compactor}_0, \text{Compactor}_1, \ldots, \text{Compactor}_L\}$$

where each Compactor_ℓ stores items with weight 2^ℓ. When a compactor exceeds capacity k (controlled by `sketch_k`), it is compacted.

Theoretical Guarantees (Karnin et al., 2016):

For a quantile q with estimated value $\hat{q}$:

$$|\text{rank}(\hat{q}) - q \cdot N| \leq \epsilon \cdot N$$

where $\epsilon \approx O(1/k)$ and space complexity is $O(k \log(N/k))$.

Phase 2: Candidate Extraction

Approximately 40 quantiles are extracted from the sketch using a non-uniform grid with higher resolution in distribution tails.

Phase 3: Optimal Cutpoint Selection

For small datasets (N <= 50), Dynamic Programming maximizes total IV. For larger datasets, a greedy IV-based selection is used.

Phase 4: Bin Refinement

Bins are refined through frequency constraint enforcement, monotonicity enforcement (if requested), and bin count optimization to minimize IV loss.

Computational Complexity

- **Time**: $O(N \log k + N \times C + k^2 \times I)$
- **Space**: $O(k \log N)$ for large N

When to Use Sketch-based Binning

- **Use**: Large datasets (N > 10^6) with memory constraints or streaming data
- **Avoid**: Small datasets (N < 1000) where approximation error may dominate

Value

A list of class `c("OptimalBinningSketch", "OptimalBinning")` containing:

id Numeric vector of bin identifiers (1-based indexing).

bin_lower Numeric vector of lower bin boundaries (inclusive).

bin_upper Numeric vector of upper bin boundaries (inclusive for last bin, exclusive for others).

woe Numeric vector of Weight of Evidence values. Monotonic if `monotonic = TRUE`.

iv Numeric vector of Information Value contributions per bin.

count Integer vector of total observations per bin.

count_pos Integer vector of positive class (target = 1) counts per bin.

count_neg Integer vector of negative class (target = 0) counts per bin.

cutpoints Numeric vector of bin split points (length = number of bins - 1). These are the internal boundaries between bins.

converged Logical flag indicating whether optimization converged.

iterations Integer number of optimization iterations performed.

Author(s)

Lopes, J. E.

References

- Karnin, Z., Lang, K., & Liberty, E. (2016). "Optimal Quantile Approximation in Streams". *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 71-78. doi:10.1109/FOCS.2016.20
- Greenwald, M., & Khanna, S. (2001). "Space-efficient online computation of quantile summaries". *ACM SIGMOD Record*, 30(2), 58-66. doi:10.1145/376284.375670
- Barlow, R. E., Bartholomew, D. J., Bremner, J. M., & Brunk, H. D. (1972). *Statistical Inference Under Order Restrictions*. Wiley.
- Siddiqi, N. (2006). *Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring*. Wiley. doi:10.1002/9781119201731

See Also

ob_numerical_mdlp, ob_numerical_mblp

Examples

```
# Example 1: Basic usage with simulated data
set.seed(123)
feature <- rnorm(500, mean = 100, sd = 20)
target <- rbinom(500, 1, prob = plogis((feature - 100) / 20))

result <- ob_numerical_sketch(
  feature = feature,
  target = target,
  min_bins = 3,
  max_bins = 5
)

# Display results
print(data.frame(
  Bin = result$id,
  Count = result$count,
  WoE = round(result$woe, 4),
```

```
  IV = round(result$iv, 4)
))

# Example 2: Comparing different sketch_k values
set.seed(456)
x <- rnorm(1000, 50, 15)
y <- rbinom(1000, 1, prob = 0.3)

result_k50 <- ob_numerical_sketch(x, y, sketch_k = 50)
result_k200 <- ob_numerical_sketch(x, y, sketch_k = 200)

cat("K=50 IV:", sum(result_k50$iv), "\n")
cat("K=200 IV:", sum(result_k200$iv), "\n")
```

`ob_numerical_ubsd` *Optimal Binning for Numerical Variables using Unsupervised Binning with Standard Deviation*

Description

Implements a **hybrid binning algorithm** that initializes bins using **unsupervised statistical properties** (mean and standard deviation of the feature) and refines them through **supervised optimization** using Weight of Evidence (WoE) and Information Value (IV).

Important Clarification: Despite "Unsupervised" in the name, this method is **predominantly supervised**. The unsupervised component is limited to the initial bin creation step (~1% of the algorithm). All subsequent refinement (merge, monotonicity enforcement, bin count adjustment) uses the target variable extensively.

The statistical initialization via $\mu \pm k\sigma$ provides a data-driven starting point that may be advantageous for approximately normal distributions, but offers no guarantees for skewed or multimodal data.

Usage

```
ob_numerical_ubsd(
  feature,
  target,
  min_bins = 3,
  max_bins = 5,
  bin_cutoff = 0.05,
  max_n_prebins = 20,
  convergence_threshold = 1e-06,
  max_iterations = 1000,
  laplace_smoothing = 0.5
)
```

Arguments

`feature`	Numeric vector of feature values. Missing values (NA) and infinite values are **not permitted** and will trigger an error.
`target`	Integer or numeric vector of binary target values (must contain only 0 and 1). Must have the same length as `feature`.
`min_bins`	Minimum number of bins (default: 3). Must be at least 2.
`max_bins`	Maximum number of bins (default: 5). Must be $\geq$ `min_bins`.
`bin_cutoff`	Minimum fraction of total observations per bin (default: 0.05). Must be in (0, 1).
`max_n_prebins`	Maximum number of pre-bins before optimization (default: 20). Must be at least equal to `min_bins`.
`convergence_threshold`	Convergence threshold for IV change (default: 1e-6).
`max_iterations`	Maximum iterations for optimization (default: 1000).
`laplace_smoothing`	Laplace smoothing parameter (default: 0.5). Must be non-negative.

Details

Algorithm Overview

UBSD executes in six phases:

Phase 1: Statistical Initialization (UNSUPERVISED)

Initial bin edges are created by combining two approaches:

1. **Standard deviation-based cutpoints**:

$$\{\mu - 2\sigma, \mu - \sigma, \mu, \mu + \sigma, \mu + 2\sigma\}$$

   where μ is the sample mean and σ is the sample standard deviation (with Bessel correction: $N - 1$ divisor).

2. **Equal-width cutpoints**:

$$\left\{ x_{\min} + i \times \frac{x_{\max} - x_{\min}}{\text{max_n_prebins}} \right\}_{i=1}^{\text{max_n_prebins}-1}$$

The union of these two sets is taken, sorted, and limited to `max_n_prebins` edges (plus $-\infty$ and $+\infty$ boundaries).

Rationale: For approximately normal distributions, $\mu \pm k\sigma$ cutpoints align with natural quantiles:

- $\mu - 2\sigma$ to $\mu + 2\sigma$ captures ~95% of data (68-95-99.7 rule)
- Equal-width ensures coverage of entire range

Limitation: For skewed distributions (e.g., log-normal), $\mu - 2\sigma$ may fall outside the data range, creating empty bins.

Special Case: If $\sigma < \epsilon$ (feature is nearly constant), fallback to pure equal-width binning.

Phase 2: Observation Assignment

Each observation is assigned to a bin via linear search:

$$\text{bin}(x_i) = \min\{j : x_i > \text{lower}_j \wedge x_i \leq \text{upper}_j\}$$

Counts are accumulated: `count`, `count_pos`, `count_neg`.

Phase 3: Rare Bin Merging (SUPERVISED)

Bins with count $<$ bin_cutoff $\times N$ are merged with adjacent bins. Merge direction is chosen to minimize IV loss:

$$\text{direction} = \arg\min_{d \in \{\text{left,right}\}} (\text{IV}_i + \text{IV}_{i+d})$$

This is a **supervised** step (uses IV computed from target).

Phase 4: WoE/IV Calculation (SUPERVISED)

Weight of Evidence with Laplace smoothing:

$$\text{WoE}_i = \ln\left(\frac{n_i^+ + \alpha}{n^+ + k\alpha} \Big/ \frac{n_i^- + \alpha}{n^- + k\alpha}\right)$$

Information Value:

$$\text{IV}_i = \left(\frac{n_i^+ + \alpha}{n^+ + k\alpha} - \frac{n_i^- + \alpha}{n^- + k\alpha}\right) \times \text{WoE}_i$$

Phase 5: Monotonicity Enforcement (SUPERVISED)

Direction is auto-detected via majority vote:

$$\text{increasing} = \begin{cases} \text{TRUE} & \text{if } \sum_i \mathbb{1}_{\{\text{WoE}_i > \text{WoE}_{i-1}\}} \geq \sum_i \mathbb{1}_{\{\text{WoE}_i < \text{WoE}_{i-1}\}} \\ \text{FALSE} & \text{otherwise} \end{cases}$$

Violations are resolved via PAVA (Pool Adjacent Violators Algorithm).

Phase 6: Bin Count Adjustment (SUPERVISED)

If $k >$ max_bins, bins are merged to minimize IV loss:

$$\text{merge_idx} = \arg\min_{i=0}^{k-2} (\text{IV}_i + \text{IV}_{i+1})$$

Convergence Criterion:

$$|\text{IV}_{\text{total}}^{(t)} - \text{IV}_{\text{total}}^{(t-1)}| < \text{convergence_threshold}$$

Comparison with Related Methods

Method	Initialization	Truly Unsupervised?	Best For
UBSD	$\mu \pm k\sigma$ + equal-width	No (1 pct unsup)	Normal distributions
MOB/MRBLP	Equal-frequency	No (0 pct unsup)	General use

MDLP	Equal-frequency	No (0 pct unsup)	Information theory
Sketch	KLL Sketch quantiles	No (0 pct unsup)	Streaming data

When to Use UBSD

- **Use UBSD**: If you have prior knowledge that the feature is approximately normally distributed and want bins aligned with standard deviations (e.g., for interpretability: "2 standard deviations below mean").
- **Avoid UBSD**: For skewed distributions (use MDLP or MOB), for multimodal distributions (use LDB), or when you need provable optimality (use Sketch for quantile guarantees).
- **Alternative**: For true unsupervised binning (no target), use `cut()` with `breaks = "Sturges"` or `"FD"` (Freedman-Diaconis).

Computational Complexity

Identical to MOB/MRBLP: $O(N + k^2 \times \text{max_iterations})$

Value

A list containing:

id Integer bin identifiers (1-based).

bin Character bin intervals `"[lower;upper)"`.

woe Numeric WoE values (monotonic after enforcement).

iv Numeric IV contributions per bin.

count Integer total observations per bin.

count_pos Integer positive class counts.

count_neg Integer negative class counts.

event_rate Numeric event rates per bin.

cutpoints Numeric bin boundaries (excluding $\pm\infty$).

total_iv Total Information Value.

converged Logical convergence flag.

iterations Integer iteration count.

Author(s)

Lopes, J. E.

References

- Sturges, H. A. (1926). "The Choice of a Class Interval". *Journal of the American Statistical Association*, 21(153), 65-66.
- Scott, D. W. (1979). "On optimal and data-based histograms". *Biometrika*, 66(3), 605-610.
- Freedman, D., & Diaconis, P. (1981). "On the histogram as a density estimator: L2 theory". *Zeitschrift fuer Wahrscheinlichkeitstheorie*, 57(4), 453-476.

- Thomas, L. C. (2009). *Consumer Credit Models: Pricing, Profit, and Portfolios*. Oxford University Press.
- Zeng, G. (2014). "A Necessary Condition for a Good Binning Algorithm in Credit Scoring". *Applied Mathematical Sciences*, 8(65), 3229-3242.
- Siddiqi, N. (2006). *Credit Risk Scorecards*. Wiley.

See Also

ob_numerical_mdlp for information-theoretic binning, ob_numerical_mob for pure supervised binning, cut for true unsupervised binning.

Examples

```
# Simulate normally distributed credit scores
set.seed(123)
n <- 5000

# Feature: Normally distributed FICO scores
feature <- rnorm(n, mean = 680, sd = 60)

# Target: Logistic relationship with score
prob_default <- 1 / (1 + exp((feature - 680) / 30))
target <- rbinom(n, 1, prob_default)

# Apply UBSD
result <- ob_numerical_ubsd(
  feature = feature,
  target = target,
  min_bins = 3,
  max_bins = 5
)

# Compare with MDLP (should be similar for normal data)
result_mdlp <- ob_numerical_mdlp(feature, target)

data.frame(
  Method = c("UBSD", "MDLP"),
  N_Bins = c(length(result$woe), length(result_mdlp$woe)),
  Total_IV = c(result$total_iv, result_mdlp$total_iv)
)
```

---

ob_numerical_udt *Optimal Binning for Numerical Variables using Entropy-Based Partitioning*

Description

Implements a supervised binning algorithm that uses Information Gain (Entropy) to identify the most informative initial split points, followed by a bottom-up merging process to satisfy constraints (minimum frequency, monotonicity, max bins).

Although historically referred to as "Unsupervised Decision Trees" in some contexts, this method is strictly **supervised** (uses target variable) and operates **bottom-up** after an initial entropy-based selection of cutpoints. It is particularly effective when the relationship between feature and target is non-linear but highly informative in specific regions.

Usage

```
ob_numerical_udt(
  feature,
  target,
  min_bins = 3,
  max_bins = 5,
  bin_cutoff = 0.05,
  max_n_prebins = 20,
  laplace_smoothing = 0.5,
  monotonicity_direction = "none",
  convergence_threshold = 1e-06,
  max_iterations = 1000
)
```

Arguments

`feature`	Numeric vector of feature values. Missing values (NA) are handled by placing them in a separate bin. Infinite values are treated as valid numeric extremes or placed in the missing bin if they represent errors.
`target`	Integer vector of binary target values (must contain only 0 and 1). Must have the same length as `feature`.
`min_bins`	Minimum number of bins (default: 3). Must be at least 2.
`max_bins`	Maximum number of bins (default: 5). Must be greater than or equal to `min_bins`.
`bin_cutoff`	Minimum fraction of total observations per bin (default: 0.05). Bins below this threshold are merged based on Event Rate similarity.
`max_n_prebins`	Maximum number of pre-bins (default: 20). The algorithm will select the top `max_n_prebins` cutpoints with highest Information Gain. **Performance Note**: High values (>50) may significantly slow down processing for large datasets due to the O(N^2) nature of cutpoint selection.
`laplace_smoothing`	Laplace smoothing parameter (default: 0.5) for WoE calculation.
`monotonicity_direction`	String specifying monotonicity constraint: • `"none"` (default): No monotonicity enforcement. • `"increasing"`: WoE must be non-decreasing. • `"decreasing"`: WoE must be non-increasing.

- `"auto"`: Automatically determined by Pearson correlation.

`convergence_threshold`
: Convergence threshold for IV optimization (default: 1e-6).

`max_iterations`
: Maximum iterations for optimization loop (default: 1000).

Details

Algorithm Overview

The UDT algorithm executes in four phases:

Phase 1: Entropy-Based Pre-binning

The algorithm evaluates every possible cutpoint c (midpoints between sorted unique values) using Information Gain (IG):

$$IG(S, c) = H(S) - \left(\frac{|S_L|}{|S|} H(S_L) + \frac{|S_R|}{|S|} H(S_R) \right)$$

The top `max_n_prebins` cutpoints with the highest IG are selected to form the initial bins. This ensures that the starting bins capture the most discriminative regions of the feature space.

Phase 2: Rare Bin Merging

Bins with frequency $<$ bin_cutoff are merged. The merge partner is chosen to minimize the difference in Event Rates:

$$\text{merge_idx} = \arg \min_{j \in \{i-1, i+1\}} |ER_i - ER_j|$$

This differs from IV-based methods and aims to preserve local risk probability smoothness.

Phase 3: Monotonicity Enforcement

If requested, monotonicity is enforced by iteratively merging bins that violate the specified direction (`"increasing"`, `"decreasing"`, or `"auto"`). Auto-direction is determined by the sign of the Pearson correlation between feature and target.

Phase 4: Constraint Satisfaction

If $k >$ max_bins, bins are merged minimizing IV loss until the constraint is met.

Warning on Complexity

The pre-binning phase evaluates Information Gain for *all* unique values. For continuous features with many unique values (e.g., $N > 10,000$), this step can be computationally intensive ($O(N^2)$). Consider rounding or using `ob_numerical_sketch` for very large datasets.

Value

A list containing:

id Integer bin identifiers (1-based).

bin Character bin intervals `"(lower;upper]"`.

woe Numeric WoE values.

iv Numeric IV contributions.

event_rate Numeric event rates.

count Integer total observations.

count_pos Integer positive class counts.

count_neg Integer negative class counts.

cutpoints Numeric bin boundaries.

total_iv Total Information Value.

gini Gini index (2*AUC - 1) calculated on the binned data.

ks Kolmogorov-Smirnov statistic calculated on the binned data.

converged Logical convergence flag.

iterations Integer iteration count.

Author(s)

Lopes, J. E.

References

- Quinlan, J. R. (1986). "Induction of Decision Trees". *Machine Learning*, 1(1), 81-106.
- Fayyad, U. M., & Irani, K. B. (1992). "On the Handling of Continuous-Valued Attributes in Decision Tree Generation". *Machine Learning*, 8, 87-102.
- Liu, H., et al. (2002). "Discretization: An Enabling Technique". *Data Mining and Knowledge Discovery*, 6(4), 393-423.

See Also

`ob_numerical_mdlp` for a pure MDL-based approach, `ob_numerical_sketch` for fast approximation on large data.

---

ob_preprocess *Data Preprocessor for Optimal Binning*

Description

Prepares features for optimal binning by handling missing values and optionally detecting/treating outliers. Supports both numerical and categorical variables with configurable preprocessing strategies.

Usage

```
ob_preprocess(
  feature,
  target,
  num_miss_value = -999,
  char_miss_value = "N/A",
  outlier_method = "iqr",
```

```
  outlier_process = FALSE,
  preprocess = "both",
  iqr_k = 1.5,
  zscore_threshold = 3,
  grubbs_alpha = 0.05
)
```

Arguments

feature	Vector (numeric, character, or factor) to be preprocessed. Type is automatically detected.
target	Numeric or integer vector of binary target values (0/1). Must have the same length as feature. Used for validation but not directly in preprocessing.
num_miss_value	Numeric value to replace missing (NA) values in numerical features (default: -999.0). Choose a value outside the expected range of the feature.
char_miss_value	Character string to replace missing (NA) values in categorical features (default: "N/A").
outlier_method	Character string specifying the outlier detection method for numerical features (default: "iqr"). Options: • "iqr": Interquartile Range method. Outliers are values $< Q_1 - k \times IQR$ or $> Q_3 + k \times IQR$. • "zscore": Z-score method. Outliers are values with $\|z\| >$ threshold where $z = (x - \mu)/\sigma$. • "grubbs": Grubbs' test for outliers (iterative). Removes the most extreme value if it exceeds the critical G-statistic at significance level grubbs_alpha.
outlier_process	Logical flag to enable outlier detection and treatment (default: FALSE). Only applies to numerical features.
preprocess	Character vector specifying output components (default: "both"): • "feature": Return preprocessed feature data only. • "report": Return preprocessing report only (summary statistics, counts). • "both": Return both preprocessed data and report.
iqr_k	Multiplier for the IQR method (default: 1.5). Larger values are more conservative (fewer outliers). Common values: 1.5 (standard), 3.0 (extreme).
zscore_threshold	Z-score threshold for outlier detection (default: 3.0). Values with $\|z\| >$ threshold are considered outliers.
grubbs_alpha	Significance level for Grubbs' test (default: 0.05). Lower values are more conservative (fewer outliers detected).

Details

Preprocessing Pipeline:

1. **Type Detection**: Automatically classifies feature as numeric or categorical based on R type.

2. **Missing Value Handling**: Replaces NA with num_miss_value (numeric) or char_miss_value (categorical).
3. **Outlier Detection** (if outlier_process = TRUE for numeric):
   - **IQR Method**: Caps outliers at boundaries $[Q_1 - k \times IQR, Q_3 + k \times IQR]$.
   - **Z-score Method**: Caps outliers at $[\mu - t \times \sigma, \mu + t \times \sigma]$.
   - **Grubbs' Test**: Iteratively removes the most extreme value if $G = \frac{\max |x_i - \bar{x}|}{s} > G_{\text{critical}}$.
4. **Summary Calculation**: Computes statistics before and after preprocessing for validation.

Outlier Treatment Strategies:

- IQR and Z-score: **Winsorization** (capping at boundaries).
- Grubbs: **Removal** (replaced with num_miss_value).

Use Cases:

- **Before binning**: Stabilize binning algorithms by removing extreme values that could create singleton bins.
- **Data quality audit**: Identify features with excessive missingness or outliers.
- **Model deployment**: Ensure test data undergoes identical preprocessing as training data.

Value

A list with up to two elements (depending on preprocess):

preprocess Data frame with columns:

- feature: Original feature values.
- feature_preprocessed: Preprocessed feature values (NAs replaced, outliers capped or removed).

report Data frame with one row containing:

- variable_type: "numeric" or "categorical".
- missing_count: Number of NA values replaced.
- outlier_count: Number of outliers detected (numeric only, NA for categorical).
- original_stats: String representation of summary statistics before preprocessing (min, Q1, median, mean, Q3, max for numeric).
- preprocessed_stats: Summary statistics after preprocessing.

References

- Grubbs, F. E. (1950). "Sample Criteria for Testing Outlying Observations". *Annals of Mathematical Statistics*, 21(1), 27-58.
- Tukey, J. W. (1977). *Exploratory Data Analysis*. Addison-Wesley. [IQR method]

Examples

```
# Numerical feature with outliers
set.seed(123)
feature_num <- c(rnorm(95, 50, 10), NA, NA, 200, -100, 250)
target <- sample(0:1, 100, replace = TRUE)

# Preprocess with IQR outlier detection
result_iqr <- ob_preprocess(
  feature = feature_num,
  target = target,
  outlier_process = TRUE,
  outlier_method = "iqr",
  iqr_k = 1.5
)

print(result_iqr$report)
# Shows: missing_count = 2, outlier_count = 3

# Categorical feature
feature_cat <- c(rep("A", 30), rep("B", 40), rep("C", 28), NA, NA)
target_cat <- sample(0:1, 100, replace = TRUE)

result_cat <- ob_preprocess(
  feature = feature_cat,
  target = target_cat,
  char_miss_value = "Missing"
)

# Compare original vs preprocessed
head(result_cat$preprocess)
# Shows NA replaced with "Missing"

# Return only report (no data)
result_report <- ob_preprocess(
  feature = feature_num,
  target = target,
  preprocess = "report",
  outlier_process = TRUE
)

# Grubbs' test (most conservative)
result_grubbs <- ob_preprocess(
  feature = feature_num,
  target = target,
  outlier_process = TRUE,
  outlier_method = "grubbs",
  grubbs_alpha = 0.01 # Very strict
)
```

plot.obwoe *Plot Method for obwoe Objects*

Description

Creates publication-quality visualizations of optimal binning results. Supports multiple plot types including IV ranking charts, WoE profiles, and bin distribution plots. All plots follow credit scoring visualization conventions.

Usage

```
## S3 method for class 'obwoe'
plot(
  x,
  type = c("iv", "woe", "bins"),
  feature = NULL,
  top_n = 15,
  show_threshold = TRUE,
  ...
)
```

Arguments

x	An object of class "obwoe".
type	Character string specifying the plot type: "iv" Information Value ranking bar chart (default) "woe" Weight of Evidence profile for selected features "bins" Bin distribution (count and event rate)
feature	Character vector of feature names to plot (for "woe" and "bins" types). If NULL, uses top 6 features by IV.
top_n	Integer. For "iv" type, number of top features to display. Default is 15. Set to NULL to display all.
show_threshold	Logical. For "iv" type, draw horizontal lines at IV thresholds (0.02, 0.10, 0.30)? Default is TRUE.
...	Additional arguments passed to base plotting functions.

Details

Plot Types:

IV Ranking (type = "iv"): Horizontal bar chart showing features ranked by Information Value. Colors indicate predictive power classification:

- Gray: IV < 0.02 (Unpredictive)
- Yellow: 0.02 <= IV < 0.10 (Weak)
- Orange: 0.10 <= IV < 0.30 (Medium)

- Green: 0.30 <= IV < 0.50 (Strong)
- Red: IV >= 0.50 (Suspicious)

WoE Profile (`type = "woe"`): Bar chart showing Weight of Evidence values for each bin. Positive WoE indicates higher-than-average event rate; negative WoE indicates lower-than-average event rate. Monotonic WoE patterns are generally preferred for interpretability.

Bin Distribution (`type = "bins"`): Dual-axis plot showing observation counts (bars) and event rates (line). Useful for diagnosing bin quality and class imbalance.

Value

Invisibly returns `NULL`. Called for side effect (plotting).

References

Thomas, L. C., Edelman, D. B., & Crook, J. N. (2002). Credit Scoring and Its Applications. *SIAM Monographs on Mathematical Modeling and Computation*. doi:10.1137/1.9780898718317

See Also

`obwoe`, `summary.obwoe`.

Examples

```
set.seed(42)
df <- data.frame(
  age = rnorm(500, 40, 15),
  income = rgamma(500, 2, 0.0001),
  score = rnorm(500, 600, 100),
  target = rbinom(500, 1, 0.2)
)
result <- obwoe(df, target = "target")

# IV ranking chart
plot(result, type = "iv")

# WoE profile for specific feature
plot(result, type = "woe", feature = "age")

# Bin distribution
plot(result, type = "bins", feature = "income")
```

plot.obwoe_gains *Plot Gains Table*

Description

Visualizes gains table metrics including cumulative capture curves, KS plot, and lift chart.

Usage

```
## S3 method for class 'obwoe_gains'
plot(x, type = c("cumulative", "ks", "lift", "woe_iv"), ...)
```

Arguments

x	An object of class "obwoe_gains".
type	Character string: "cumulative" (default), "ks", "lift", or "woe_iv".
...	Additional arguments passed to plotting functions.

Value

Invisibly returns NULL.

predict.obwoe_scorecard *Score New Data With a Fitted Scorecard*

Description

Applies a fitted scorecard to new data. Reads only the binning, the coefficients and the scaling — never the raw fitted model — so a scorecard saved with saveRDS() keeps scoring after the engine's package is gone.

Usage

```
## S3 method for class 'obwoe_scorecard'
predict(
  object,
  new_data,
  type = c("score", "card", "link", "prob", "woe"),
  ...
)
```

Arguments

object	An "obwoe_scorecard" object.
new_data	A data.frame carrying the model's variables.
type	One of "score" (default), "card" (the sum of the integer points, i.e. what the deployed table gives), "link" (log-odds of the event), "prob" or "woe".
...	Ignored.

Value

A numeric vector, or a data.frame when type = "woe".

See Also

obwoe_scorecard

Examples

```
german <- read.csv(
  gzfile(system.file("extdata", "germancredit.csv.gz",
    package = "OptimalBinningWoE"
  )),
  stringsAsFactors = FALSE
)
german$default <- 1L - german$credit_risk
german$credit_risk <- NULL

sc <- obwoe_scorecard(german, target = "default", seed = 1)
summary(predict(sc, german))
```

---

prep.step_obwoe *Prepare the Optimal Binning Step*

Description

Fits the optimal binning models on training data. This method is called by prep and should not be invoked directly.

Usage

```
## S3 method for class 'step_obwoe'
prep(x, training, info = NULL, ...)
```

Arguments

x	A step_obwoe object.
training	A tibble or data frame containing the training data.
info	A tibble with column metadata from the recipe.
...	Additional arguments (currently unused).

Value

A trained step_obwoe object with binning_results populated.

---

print.obwoe *Print Method for obwoe Objects*

Description

Displays a concise summary of optimal binning results, including the number of successfully processed features and top predictors ranked by Information Value.

Usage

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

Arguments

x	An object of class "obwoe".
...	Additional arguments (currently ignored).

Value

Invisibly returns x.

See Also

summary.obwoe for detailed statistics, plot.obwoe for visualization.

print.obwoe_sql *Print Method for Generated SQL*

Description

Echoes the generated SQL verbatim instead of the quoted, escaped form base R would print for a character vector.

Usage

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

Arguments

x	An object of class "obwoe_sql".
...	Additional arguments (currently ignored).

Value

Invisibly returns x.

See Also

obwoe_sql

Examples

```
set.seed(1)
df <- data.frame(x = rnorm(500))
df$target <- rbinom(500, 1, plogis(df$x))
print(obwoe_sql(obwoe(df, target = "target"), table = "t"))
```

print.step_obwoe *Print Method for step_obwoe*

Description

Prints a concise summary of the step_obwoe object.

Usage

```
## S3 method for class 'step_obwoe'
print(x, width = max(20L, options()$width - 30L), ...)
```

Arguments

x	A step_obwoe object.
width	Maximum width for printing term names.
...	Additional arguments (currently unused).

Value

Invisibly returns x.

---

required_pkgs.step_obwoe *Required Packages for step_obwoe*

Description

Lists the packages required to execute the step_obwoe transformation.

Usage

```
## S3 method for class 'step_obwoe'
required_pkgs(x, ...)
```

Arguments

x	A step_obwoe object.
...	Additional arguments (currently unused).

Value

A character vector of package names.

---

step_obwoe *Optimal Binning and WoE Transformation Step*

Description

step_obwoe() creates a *specification* of a recipe step that discretizes predictor variables using one of 28 state-of-the-art optimal binning algorithms and transforms them into Weight of Evidence (WoE) values. This step fully integrates the **OptimalBinningWoE** package into the tidymodels framework, supporting supervised discretization for both binary and multinomial classification targets with extensive hyperparameter tuning capabilities.

Usage

```
step_obwoe(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  outcome = NULL,
  algorithm = "auto",
  min_bins = 2L,
  max_bins = 10L,
  bin_cutoff = 0.05,
  output = c("woe", "bin", "both"),
  suffix_woe = "_woe",
  suffix_bin = "_bin",
  na_woe = 0,
  control = list(),
  binning_results = NULL,
  skip = FALSE,
  id = recipes::rand_id("obwoe")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose variables for this step. See selections for available selectors. Common choices include all_predictors(), all_numeric_predictors(), or all_nominal_predictors(). Ensure the selected variables are compatible with the chosen algorithm (e.g., do not apply "mdlp" to categorical data).
role	For variables created by this step, what role should they have? Default is "predictor".
trained	A logical indicating whether the step has been trained (fitted). This should not be set manually.
outcome	A character string specifying the name of the binary or multinomial response variable. This argument is **required** as all binning algorithms are supervised. The outcome must exist in the training data provided to prep(). The outcome should be encoded as a factor (standard for tidymodels classification) or as integers 0/1 for binary, 0/1/2/... for multinomial.
algorithm	Character string specifying the binning algorithm to use. Use "auto" (default) for automatic selection based on target type: "jedi" for binary targets, "jedi_mwoe" for multinomial. Available algorithms are organized by supported feature types: **Universal (numerical and categorical):** "auto", "jedi", "jedi_mwoe", "cm", "dp", "dmiv", "fetb", "mob", "sketch", "udt" **Numerical only:** "bb", "ewb", "fast_mdlp", "ir", "kmb", "ldb", "lpdb", "mblp", "mdlp", "mrblp", "oslp", "ubsd"

Categorical only: `"gmb"`, `"ivb"`, `"mba"`, `"milp"`, `"sab"`, `"sblp"`, `"swb"`

This parameter is tunable with `tune()`.

`min_bins` Integer specifying the minimum number of bins to create. Must be at least 2. Default is 2. This parameter is tunable with `tune()`.

`max_bins` Integer specifying the maximum number of bins to create. Must be greater than or equal to `min_bins`. Default is 10. This parameter is tunable with `tune()`.

`bin_cutoff` Numeric value between 0 and 1 (exclusive) specifying the minimum proportion of total observations that each bin must contain. Bins with fewer observations are merged with adjacent bins. This serves as a regularization mechanism to prevent overfitting and ensure statistical stability of WoE estimates. Default is 0.05 (5%). This parameter is tunable with `tune()`.

`output` Character string specifying the transformation output format:

- `"woe"` Replaces the original variable with WoE values (default). This is the standard choice for logistic regression scorecards.
- `"bin"` Replaces the original variable with bin labels (character). Useful for tree-based models or exploratory analysis.
- `"both"` Keeps the original column and adds two new columns with suffixes `_woe` and `_bin`. Useful for model comparison or audit trails.

`suffix_woe` Character string suffix appended to create WoE column names when `output = "both"`. Default is `"_woe"`.

`suffix_bin` Character string suffix appended to create bin column names when `output = "both"`. Default is `"_bin"`.

`na_woe` Numeric value to assign to observations that cannot be mapped to a bin during `bake()`. This includes missing values (`NA`) and unseen categories not present in the training data. Default is 0, which represents neutral evidence (neither good nor bad).

`control` A named list of additional control parameters passed to `control.obwoe`. These provide fine-grained control over algorithm behavior such as convergence thresholds and maximum pre-bins. Parameters specified directly in `step_obwoe()` (e.g., `bin_cutoff`) take precedence over values in this list.

`binning_results`

Internal storage for fitted binning models after `prep()`. Do not set manually.

`skip` Logical. Should this step be skipped when `bake()` is called on new data? Default is `FALSE`. Setting to `TRUE` is rarely needed for WoE transformations but may be useful in specialized workflows.

`id` A unique character string to identify this step. If not provided, a random identifier is generated.

Details

Weight of Evidence Transformation:

Weight of Evidence (WoE) is a supervised encoding technique that transforms categorical and continuous variables into a scale that measures the predictive strength of each value or bin relative to the target variable. For a bin i, the WoE is defined as:

$$WoE_i = \ln\left(\frac{\text{Distribution of Events}_i}{\text{Distribution of Non-Events}_i}\right)$$

Positive WoE values indicate the bin has a higher proportion of events (e.g., defaults) than the overall population, while negative values indicate lower risk.

Algorithm Selection Strategy:

The `algorithm` parameter provides access to 28 binning algorithms:

- Use `algorithm = "auto"` (default) for automatic selection: `"jedi"` for binary targets, `"jedi_mwoe"` for multinomial.
- Use `algorithm = "mob"` (Monotonic Optimal Binning) when monotonic WoE trends are required for regulatory compliance (Basel/IFRS 9).
- Use `algorithm = "mdlp"` for entropy-based discretization of numerical variables (requires `all_numeric_predictors()`).
- Use `algorithm = "dp"` (Dynamic Programming) for exact optimal solutions when computational cost is acceptable.

If an incompatible algorithm is applied to a variable (e.g., `"mdlp"` on a factor), the step will issue a warning during `prep()` and skip that variable, leaving it untransformed.

Handling New Data:

During `bake()`, observations are mapped to bins learned during `prep()`:

- **Numerical variables**: Values are assigned to bins based on the learned cutpoints using interval notation.
- **Categorical variables**: Categories are matched to their corresponding bins. Categories not seen during training receive the `na_woe` value.
- **Missing values**: Always receive the `na_woe` value.

Tuning with tune:

This step is fully compatible with the `tune` package. The following parameters support `tune()`:

- `algorithm`: See `obwoe_algorithm`.
- `min_bins`: See `obwoe_min_bins`.
- `max_bins`: See `obwoe_max_bins`.
- `bin_cutoff`: See `obwoe_bin_cutoff`.

Case Weights:

This step does not currently support case weights. All observations are treated with equal weight during binning optimization.

Value

An updated `recipe` object with the new step appended.

References

Siddiqi, N. (2006). Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring. *John Wiley & Sons*. doi:10.1002/9781119201731

Thomas, L. C., Edelman, D. B., & Crook, J. N. (2002). Credit Scoring and Its Applications. *SIAM Monographs on Mathematical Modeling and Computation*. doi:10.1137/1.9780898718317

Navas-Palencia, G. (2020). Optimal Binning: Mathematical Programming Formulation and Solution Approach. *Expert Systems with Applications*, 158, 113508. doi:10.1016/j.eswa.2020.113508

See Also

obwoe for the underlying binning engine, control.obwoe for advanced control parameters, obwoe_algorithm, obwoe_min_bins, obwoe_max_bins, obwoe_bin_cutoff for tuning parameter definitions, recipe, prep, bake for the tidymodels recipe framework.

Examples

```
library(recipes)

# Simulated credit data
set.seed(123)
credit_data <- data.frame(
  age = rnorm(500, 45, 12),
  income = exp(rnorm(500, 10, 0.6)),
  employment = sample(c("Employed", "Self-Employed", "Unemployed"),
    500,
    replace = TRUE, prob = c(0.7, 0.2, 0.1)
  ),
  education = factor(c("HighSchool", "Bachelor", "Master", "PhD")[
    sample(1:4, 500, replace = TRUE, prob = c(0.3, 0.4, 0.2, 0.1))
  ]),
  default = factor(rbinom(500, 1, 0.15),
    levels = c(0, 1),
    labels = c("No", "Yes")
  )
)

# Example 1: Basic usage with automatic algorithm selection
rec_basic <- recipe(default ~ ., data = credit_data) %>%
  step_obwoe(all_predictors(), outcome = "default")

rec_prepped <- prep(rec_basic)
baked_data <- bake(rec_prepped, new_data = NULL)
head(baked_data)

# View binning details
tidy(rec_prepped, number = 1)

# Example 2: Numerical-only algorithm on numeric predictors
rec_mdlp <- recipe(default ~ age + income, data = credit_data) %>%
  step_obwoe(all_numeric_predictors(),
    outcome = "default",
```

```
    algorithm = "mdlp",
    min_bins = 3,
    max_bins = 6
  )

# Example 3: Output both bins and WoE
rec_both <- recipe(default ~ age, data = credit_data) %>%
  step_obwoe(age,
    outcome = "default",
    output = "both"
  )

baked_both <- bake(prep(rec_both), new_data = NULL)
names(baked_both)
# Contains: default, age, age_woe, age_bin

# Example 4: Custom control parameters
rec_custom <- recipe(default ~ ., data = credit_data) %>%
  step_obwoe(all_predictors(),
    outcome = "default",
    algorithm = "mob",
    bin_cutoff = 0.03,
    control = list(
      max_n_prebins = 30,
      convergence_threshold = 1e-8
    )
  )

# Example 5: Tuning specification (for use with tune package)
# rec_tune <- recipe(default ~ ., data = credit_data) %>%
#   step_obwoe(all_predictors(),
#              outcome = "default",
#              algorithm = tune(),
#              min_bins = tune(),
#              max_bins = tune())
```

---

summary.obwoe *Summary Method for obwoe Objects*

Description

Generates comprehensive summary statistics for optimal binning results, including predictive power classification based on established IV thresholds (Siddiqi, 2006), aggregate metrics, and feature-level diagnostics.

Usage

```
## S3 method for class 'obwoe'
summary(object, sort_by = "iv", decreasing = TRUE, ...)
```

Arguments

`object`	An object of class `"obwoe"`.
`sort_by`	Character string specifying the column to sort by. Options: `"iv"` (default), `"n_bins"`, `"feature"`.
`decreasing`	Logical. Sort in decreasing order? Default is `TRUE` for IV, `FALSE` for feature names.
`...`	Additional arguments (currently ignored).

Details

IV Classification Thresholds:

Following Siddiqi (2006), features are classified by predictive power:

Classification	IV Range
Unpredictive	< 0.02
Weak	0.02 - 0.10
Medium	0.10 - 0.30
Strong	0.30 - 0.50
Suspicious	> 0.50

Features with IV > 0.50 should be examined for data leakage or overfitting, as such high values are rarely observed in practice.

Value

An S3 object of class `"summary.obwoe"` containing:

`feature_summary` Data frame with per-feature statistics including IV classification (Unpredictive/Weak/Medium/Strong/Suspicious)

`aggregate` Named list of aggregate statistics:

- `n_features` Total features processed
- `n_successful` Features without errors
- `n_errors` Features with errors
- `total_iv_sum` Sum of all feature IVs
- `mean_iv` Mean IV across features
- `median_iv` Median IV across features
- `mean_bins` Mean number of bins
- `iv_range` Min and max IV values

`iv_distribution` Table of IV classification counts

`target` Target column name

`target_type` Target type (binary/multinomial)

References

Siddiqi, N. (2006). Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring. *John Wiley & Sons*. doi:10.1002/9781119201731

See Also

obwoe for the main binning function, print.obwoe, plot.obwoe.

Examples

```
set.seed(42)
df <- data.frame(
  x1 = rnorm(500), x2 = rnorm(500), x3 = rnorm(500),
  target = rbinom(500, 1, 0.2)
)
result <- obwoe(df, target = "target")
summary(result)
```

---

tidy.step_obwoe *Tidy Method for step_obwoe*

Description

Returns a tibble with information about the binning transformation. For trained steps, returns one row per bin per feature, including bin labels, WoE values, and IV contributions. For untrained steps, returns a placeholder tibble.

Usage

```
## S3 method for class 'step_obwoe'
tidy(x, ...)
```

Arguments

x	A step_obwoe object.
...	Additional arguments (currently unused).

Value

A tibble with columns:

terms Character. Feature name.

bin Character. Bin label or interval.

woe Numeric. Weight of Evidence value for the bin.

iv Numeric. Information Value contribution of the bin.

id Character. Step identifier.

`tunable.step_obwoe` *Tunable Parameters for step_obwoe*

Description

Returns information about which parameters of step_obwoe can be tuned using the `tune` package.

Usage

```
## S3 method for class 'step_obwoe'
tunable(x, ...)
```

Arguments

`x`	A step_obwoe object.
`...`	Additional arguments (currently unused).

Value

A tibble describing tunable parameters.

Index

.categorical_only_algorithms, 4
.numerical_only_algorithms, 4
.universal_algorithms, 4
.valid_algorithms, 5

bake, *5*, *223*
bake.step_obwoe, 5

control.obwoe, 6, *8*, *13*, *16*, *23*, *45*, *51*, *221*, *223*
control.obwoe_scorecard, 8, *41*
cut, *206*

ob_apply_woe_cat, *16*, *50*, 53, *56*, *133*
ob_apply_woe_num, *16*, 54, *135*
ob_categorical_cm, *53*, 57, *63*, *67*, *72*, *77*, *84*, *90*, *107*
ob_categorical_dmiv, 60, *67*, *72*
ob_categorical_dp, 64, *72*, *77*, *84*, *90*, *99*, *107*
ob_categorical_fetb, 69, *77*
ob_categorical_gmb, 74, *84*, *90*
ob_categorical_ivb, *16*, 81, *90*, *99*
ob_categorical_jedi, *16*, *21*, 86, *99*, *107*
ob_categorical_jedi_mwoe, 95
ob_categorical_mba, 104
ob_categorical_milp, 111
ob_categorical_mob, 114
ob_categorical_sab, 117
ob_categorical_sblp, 120
ob_categorical_sketch, 123
ob_categorical_swb, 126
ob_categorical_udt, 129
ob_cutpoints_cat, 132
ob_cutpoints_num, 134
ob_numerical_bb, 136, *146*
ob_numerical_cm, 139, *146*, *148, 149*, *152*, *154*, *159*, *163*
ob_numerical_dmiv, 141
ob_numerical_dp, 144, *149*, *152*, *156*
ob_numerical_ewb, 147, *163*
ob_numerical_fast_mdlp, 149
ob_numerical_fetb, 152
ob_numerical_ir, 154, *159*
ob_numerical_jedi, *16*, 157, *161*, *170*
ob_numerical_jedi_mwoe, 159
ob_numerical_kmb, 161, *170*
ob_numerical_ldb, 164, *175*, *181*
ob_numerical_lpdb, 168
ob_numerical_mblp, 171, *181*, *187*, *193*, *197*, *201*
ob_numerical_mdlp, *16*, *54*, *56*, *156*, *167*, *175*, 176, *187*, *193*, *197*, *201*, *206*, *209*
ob_numerical_mob, *54*, *56*, *167*, 183, *193*, *206*
ob_numerical_mrblp, 189, *194*, *197*
ob_numerical_oslp, 194
ob_numerical_sketch, 198, *208, 209*
ob_numerical_ubsd, 202
ob_numerical_udt, 206
ob_preprocess, 209
obcorr, *9*, 10, *34*
obwoe, *6, 7*, 12, *19–22*, *24*, *26, 27*, *41*, *44*, *48, 49*, *52*, *135*, *214*, *223*, *226*
obwoe_algorithm, 19, *222, 223*
obwoe_algorithms, *16*, 20
obwoe_apply, 20, *24*, *27*, *41*, *46, 47*, *49–52*, *135*
obwoe_bin_cutoff, 23, *222, 223*
obwoe_gains, 24
obwoe_gains_score, 28, *30, 31*, *47, 48*
obwoe_gains_variable, 30
obwoe_max_bins, 32, *33*, *222, 223*
obwoe_min_bins, *32*, 33, *222, 223*
obwoe_prune, 34, *42*
obwoe_psi, 35, *43*
obwoe_report, 36, *41–43*
obwoe_scale, *9*, 38, *39, 40*, *42, 43*
obwoe_score, *38, 39*, 39

obwoe_scorecard, *8, 9*, *36, 37*, *39*, 40, *216*
obwoe_select, *8*, *34*, *41–43*, 43, *49*, *52*
obwoe_sql, *21*, *36, 37*, *43*, *48*, 48, *218*

plot.obwoe, 213, *217*, *226*
plot.obwoe_gains, *27*, 215
predict.obwoe_scorecard, *43*, 215
prep, *216*, *223*
prep.step_obwoe, 216
print.obwoe, *16*, 217, *226*
print.obwoe_sql, 218
print.step_obwoe, 218

recipe, *223*
required_pkgs.step_obwoe, 219

selections, *220*
step_obwoe, *19*, *23*, *32, 33*, *49*, 219
summary.obwoe, *22*, *48*, *214*, *217*, 224

tidy.step_obwoe, 226
tunable.step_obwoe, 227

writeLines, *51*