Package 'themis'

August 2, 2026

Title Extra Recipes Steps for Dealing with Unbalanced Data

Version 1.1.0

Description A dataset with an uneven number of cases in each class is said to be unbalanced. Many models produce a subpar performance on unbalanced datasets. A dataset can be balanced by increasing the number of minority cases using SMOTE 2011 <doi:10.48550/arXiv.1106.1813>, BorderlineSMOTE 2005 <doi:10.1007/11538059_91> and ADASYN 2008. Or by decreasing the number of majority cases using NearMiss 2003 <https://www.site.uottawa.ca/~nat/Workshop2003/jzhang.pdf> or Tomek link removal 1976.

License MIT + file LICENSE

URL https://github.com/tidymodels/themis, https://themis.tidymodels.org

BugReports https://github.com/tidymodels/themis/issues

Depends R (>= 4.1), recipes (>= 1.1.0)

Imports cli, dplyr, generics (>= 0.1.0), glue, gower, hardhat, lifecycle (>= 1.0.3), purrr, RANN, rlang (>= 1.1.0), ROSE, tibble, vctrs, withr

Suggests covr, dials (>= 1.2.0), ggplot2, kernlab, modeldata, philentropy, testthat (>= 3.0.0)

Config/Needs/website tidyverse/tidytemplate

Config/testthat/edition 3

Config/usethis/last-upkeep 2025-04-24

Encoding UTF-8

LazyData true

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Emil Hvitfeldt [aut, cre] (ORCID: <https://orcid.org/0000-0002-0679-1945>), Posit Software, PBC [cph, fnd] (ROR: <https://ror.org/03wc8by49>)

Maintainer Emil Hvitfeldt <emil.hvitfeldt@posit.co>

Repository CRAN

Date/Publication 2026-08-02 05:10:02 UTC

Contents

adasyn *Adaptive Synthetic Algorithm*

Description

Generates synthetic positive instances using ADASYN algorithm.

Usage

```
adasyn(df, var, k = 5, over_ratio = 1, distance = "euclidean")
```

Arguments

df	data.frame or tibble. Must have 1 factor variable and remaining numeric variables.
var	Character, name of variable containing factor variable.
k	An integer. Number of nearest neighbor that are used to generate the new examples of the minority class.
over_ratio	A numeric value for the ratio of the minority-to-majority frequencies. The default value (1) means that all other levels are sampled up to have the same frequency as the most occurring level. A value of 0.5 would mean that the minority levels will have (at most) (approximately) half as many rows as the majority level. A named numeric vector can be used instead to give different levels different targets, for example `c(a = 1, b = 0.5)`. The names must be levels of the outcome and the values are ratios of the majority level, exactly as in the single-number case. Levels that are not named are left untouched, as are rows with a missing outcome. Because a vector of targets is not a single value, supplying one means this argument can no longer be tuned. See `vignette("ratio", package = "themis")` for more details.
distance	A character string specifying the distance metric used for nearest neighbor calculations, defaulting to `"euclidean"`. The available metrics fall into three groups. `"euclidean"`, `"cosine"`, and `"mahalanobis"` use approximate nearest neighbors via the RANN package and scale well to large datasets. `"squared_chord"`, `"matusita"`, `"hellinger"`, and `"bhattacharyya"` are probability-divergence measures that treat each row as a distribution over the predictors, so they require non-negative values. `"hellinger"` and `"bhattacharyya"` further require each row to sum to 1. All four also use the RANN package and scale well to large datasets. `"manhattan"`, `"chebyshev"`, `"canberra"`, `"soergel"`, `"lorentzian"`, `"jeffreys"`, `"topsoe"`, `"jensen-shannon"`, `"jensen_difference"`, `"taneja"`, and `"kumar-johnson"` compute an exact all-pairs distance matrix. This takes time and memory proportional to the square of the number of observations in a class, so these are best

suited to smaller datasets. Everything from "canberra" onwards is a probability divergence requiring non-negative values, is provided by the philentropy package (which must be installed separately), and in the case of "jeffreys", "taneja", and "kumar-johnson" requires strictly positive values, since those divide by individual predictor values.

The probability divergences are meaningful for compositional predictors such as proportions or counts normalized per observation, and are generally not appropriate for standardized predictors.

Details

ADASYN is an extension of SMOTE that adaptively generates synthetic minority class examples based on the local distribution of each minority instance. Instead of generating the same number of synthetic examples for every minority instance, ADASYN generates more synthetic examples for instances that are harder to learn, specifically those surrounded by more majority class neighbors. This focuses synthetic data generation on the difficult boundary regions of the minority class, resulting in a more informative augmentation than standard SMOTE.

All columns used in this function must be numeric with no missing data.

Value

A data.frame or tibble, depending on type of df.

References

He, H., Bai, Y., Garcia, E. and Li, S. 2008. ADASYN: Adaptive synthetic sampling approach for imbalanced learning. Proceedings of IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference. pp.1322-1328.

See Also

step_adasyn() for step function of this method

Other Direct Implementations: bsmote(), cluster_centroids(), cnn(), enn(), instance_hardness(), kmeans_smote(), ncl(), nearmiss(), oss(), rose(), smogn(), smote(), smoten(), smotenc(), svmsmote(), tomek()

Examples

```
circle_numeric <- circle_example[, c("x", "y", "class")]

res <- adasyn(circle_numeric, var = "class")

res <- adasyn(circle_numeric, var = "class", k = 10)

res <- adasyn(circle_numeric, var = "class", over_ratio = 0.8)

res <- adasyn(circle_numeric, var = "class", distance = "manhattan")
```

`bsmote` *Borderline-SMOTE Algorithm*

Description

BSMOTE generates new examples of the minority class using nearest neighbors of these cases in the border region between classes.

Usage

```
bsmote(
  df,
  var,
  k = 5,
  over_ratio = 1,
  all_neighbors = FALSE,
  distance = "euclidean"
)
```

Arguments

`df`	data.frame or tibble. Must have 1 factor variable and remaining numeric variables.
`var`	Character, name of variable containing factor variable.
`k`	An integer. Number of nearest neighbor that are used to generate the new examples of the minority class.
`over_ratio`	A numeric value for the ratio of the minority-to-majority frequencies. The default value (1) means that all other levels are sampled up to have the same frequency as the most occurring level. A value of 0.5 would mean that the minority levels will have (at most) (approximately) half as many rows as the majority level. A named numeric vector can be used instead to give different levels different targets, for example `c(a = 1, b = 0.5)`. The names must be levels of the outcome and the values are ratios of the majority level, exactly as in the single-number case. Levels that are not named are left untouched, as are rows with a missing outcome. Because a vector of targets is not a single value, supplying one means this argument can no longer be tuned. See `vignette("ratio", package = "themis")` for more details.
`all_neighbors`	Type of two borderline-SMOTE method. Defaults to FALSE. See details.
`distance`	A character string specifying the distance metric used for nearest neighbor calculations, defaulting to `"euclidean"`. The available metrics fall into three groups. `"euclidean"`, `"cosine"`, and `"mahalanobis"` use approximate nearest neighbors via the RANN package and scale well to large datasets.

"squared_chord", "matusita", "hellinger", and "bhattacharyya" are probability-divergence measures that treat each row as a distribution over the predictors, so they require non-negative values. "hellinger" and "bhattacharyya" further require each row to sum to 1. All four also use the RANN package and scale well to large datasets.

"manhattan", "chebyshev", "canberra", "soergel", "lorentzian", "jeffreys", "topsoe", "jensen-shannon", "jensen_difference", "taneja", and "kumar-johnson" compute an exact all-pairs distance matrix. This takes time and memory proportional to the square of the number of observations in a class, so these are best suited to smaller datasets. Everything from "canberra" onwards is a probability divergence requiring non-negative values, is provided by the philentropy package (which must be installed separately), and in the case of "jeffreys", "taneja", and "kumar-johnson" requires strictly positive values, since those divide by individual predictor values.

The probability divergences are meaningful for compositional predictors such as proportions or counts normalized per observation, and are generally not appropriate for standardized predictors.

Details

BSMOTE (borderline-SMOTE) works the same way as SMOTE, except that instead of generating points around every point of the minority class each point is first classified into the boxes "danger" and "not". For each point the nearest neighbors are calculated. If all the neighbors come from a different class it is labeled noise and put into the "not" box. If more than half of the neighbors come from a different class it is labeled "danger". Points are generated around points labeled "danger".

If `all_neighbors = FALSE` then points are generated between nearest neighbors in its own class. If `all_neighbors = TRUE` then points are generated between any nearest neighbors. See examples for visualization.

SMOTE generates new examples of the minority class using nearest neighbors of these cases. For each existing minority class example, new examples are created by interpolating between the example and its nearest neighbors. The number of nearest neighbors used is controlled by the number of neighbors argument (`k` in `smote()`, `neighbors` in `step_smote()`), and the number of new examples generated is controlled by `over_ratio`.

All columns used in this function must be numeric with no missing data.

Value

A data.frame or tibble, depending on type of `df`.

References

Hui Han, Wen-Yuan Wang, and Bing-Huan Mao. Borderline-smote: a new over-sampling method in imbalanced data sets learning. In International Conference on Intelligent Computing, pages 878–887. Springer, 2005.

See Also

`step_bsmote()` for step function of this method

Other Direct Implementations: adasyn(), cluster_centroids(), cnn(), enn(), instance_hardness(), kmeans_smote(), ncl(), nearmiss(), oss(), rose(), smogn(), smote(), smoten(), smotenc(), svmsmote(), tomek()

Examples

```
circle_numeric <- circle_example[, c("x", "y", "class")]

res <- bsmote(circle_numeric, var = "class")

res <- bsmote(circle_numeric, var = "class", k = 10)

res <- bsmote(circle_numeric, var = "class", over_ratio = 0.8)

res <- bsmote(circle_numeric, var = "class", all_neighbors = TRUE)

res <- bsmote(circle_numeric, var = "class", distance = "manhattan")
```

---

circle_example *Synthetic Dataset With a Circle*

Description

A random dataset with two classes one of which is inside a circle. Used for examples to show how the different methods handles borders.

Usage

```
circle_example
```

Format

A data frame with 400 rows and 4 variables:

x Numeric.

y Numeric.

class Factor, values "Circle" and "Rest".

id character, ID variable.

Source

Synthetic data, simulated in `data-raw/circle_example.R`.

`cluster_centroids` *ClusterCentroids Algorithm*

Description

Under-samples the majority classes by replacing them with cluster representatives found with k-means.

Usage

```
cluster_centroids(df, var, under_ratio = 1, voting = "soft")
```

Arguments

`df`	data.frame or tibble. Must have 1 factor variable and remaining numeric variables.
`var`	Character, name of variable containing factor variable.
`under_ratio`	A numeric value for the ratio of the majority-to-minority frequencies. The default value (1) means that all other levels are sampled down to have the same frequency as the least occurring level. A value of 2 would mean that the majority levels will have (at most) (approximately) twice as many rows than the minority level. A named numeric vector can be used instead to give different levels different targets, for example `c(a = 2, b = 3)`. The names must be levels of the outcome and the values are ratios of the minority level, exactly as in the single-number case. Levels that are not named are left untouched, as are rows with a missing outcome. Because a vector of targets is not a single value, supplying one means this argument can no longer be tuned. See `vignette("ratio", package = "themis")` for more details.
`voting`	A character string. Either `"soft"` (the default) to use the cluster centroids themselves, or `"hard"` to use the observation closest to each centroid.

Details

Each class larger than the target count is summarized by running k-means on the observations of that class, using as many clusters as the target count. The class is then replaced by one representative per cluster:

`voting = "soft"` the cluster centroids themselves are used, so the returned observations are synthetic points that need not appear in the input.

`voting = "hard"` the observation closest to each centroid is used, so all returned observations are real rows. The representative is always picked from the class being under-sampled.

This makes it the one *prototype generation* under-sampler in this package. The other under-sampling methods perform *prototype selection*, keeping a subset of the original rows.

Because two clusters can share the same closest observation, `voting = "hard"` can return slightly fewer observations than the target count.

All columns used in this function must be numeric with no missing data.

Value

A data.frame or tibble, depending on type of `df`.

See Also

`step_cluster_centroids()` for step function of this method

Other Direct Implementations: `adasyn()`, `bsmote()`, `cnn()`, `enn()`, `instance_hardness()`, `kmeans_smote()`, `ncl()`, `nearmiss()`, `oss()`, `rose()`, `smogn()`, `smote()`, `smoten()`, `smotenc()`, `svmsmote()`, `tomek()`

Examples

```
circle_numeric <- circle_example[, c("x", "y", "class")]

res <- cluster_centroids(circle_numeric, var = "class")

res <- cluster_centroids(circle_numeric, var = "class", voting = "hard")

res <- cluster_centroids(circle_numeric, var = "class", under_ratio = 1.5)
```

---

cnn *Condensed Nearest Neighbors*

Description

Under-samples the majority classes by keeping only a consistent subset of observations that correctly classifies the data using a 1-nearest-neighbor rule.

Usage

```
cnn(df, var, distance = "euclidean")
```

Arguments

`df`	data.frame or tibble. Must have 1 factor variable and remaining numeric variables.
`var`	Character, name of variable containing factor variable.
`distance`	A character string specifying the distance metric used for nearest neighbor calculations, defaulting to `"euclidean"`. The available metrics fall into three groups. `"euclidean"`, `"cosine"`, and `"mahalanobis"` use approximate nearest neighbors via the RANN package and scale well to large datasets.

"squared_chord", "matusita", "hellinger", and "bhattacharyya" are probability-divergence measures that treat each row as a distribution over the predictors, so they require non-negative values. "hellinger" and "bhattacharyya" further require each row to sum to 1. All four also use the RANN package and scale well to large datasets.

"manhattan", "chebyshev", "canberra", "soergel", "lorentzian", "jeffreys", "topsoe", "jensen-shannon", "jensen_difference", "taneja", and "kumar-johnson" compute an exact all-pairs distance matrix. This takes time and memory proportional to the square of the number of observations in a class, so these are best suited to smaller datasets. Everything from "canberra" onwards is a probability divergence requiring non-negative values, is provided by the philentropy package (which must be installed separately), and in the case of "jeffreys", "taneja", and "kumar-johnson" requires strictly positive values, since those divide by individual predictor values.

The probability divergences are meaningful for compositional predictors such as proportions or counts normalized per observation, and are generally not appropriate for standardized predictors.

Details

Condensed Nearest Neighbors (CNN) is an under-sampling method that reduces the majority classes to a consistent subset: a subset that classifies the original data correctly using a 1-nearest-neighbor rule. It starts with a "store" containing all minority class observations and one randomly chosen majority class observation. It then repeatedly scans the remaining majority class observations and moves any that are misclassified by a 1-nearest neighbor fit on the current store into the store. This continues until a full pass adds no new observations. The observations left outside the store are removed.

The smallest class is treated as the minority class and is always kept. CNN tends to keep observations near the decision boundary while discarding redundant interior observations. Because the seed observation and the scan order are random, results depend on the random seed.

All columns used in this function must be numeric with no missing data.

Value

A data.frame or tibble, depending on type of df.

References

Hart, P. (1968). The condensed nearest neighbor rule. IEEE Transactions on Information Theory, 14(3), 515-516.

See Also

step_cnn() for step function of this method

Other Direct Implementations: adasyn(), bsmote(), cluster_centroids(), enn(), instance_hardness(), kmeans_smote(), ncl(), nearmiss(), oss(), rose(), smogn(), smote(), smoten(), smotenc(), svmsmote(), tomek()

Examples

```
circle_numeric <- circle_example[, c("x", "y", "class")]

res <- cnn(circle_numeric, var = "class")

res <- cnn(circle_numeric, var = "class", distance = "manhattan")
```

---

enn *Edited Nearest Neighbors*

Description

Removes observations whose class differs from the majority of their nearest neighbors.

Usage

```
enn(
  df,
  var,
  neighbors = 3,
  distance = "euclidean",
  times = 1,
  all_k = FALSE,
  kind_sel = "mode"
)
```

Arguments

df	data.frame or tibble. Must have 1 factor variable and remaining numeric variables.
var	Character, name of variable containing factor variable.
neighbors	An integer. Number of nearest neighbor that are used to decide whether an observation is removed. Defaults to 3, unlike the over-sampling steps which default to 5.
distance	A character string specifying the distance metric used for nearest neighbor calculations, defaulting to "euclidean". The available metrics fall into three groups. "euclidean", "cosine", and "mahalanobis" use approximate nearest neighbors via the RANN package and scale well to large datasets. "squared_chord", "matusita", "hellinger", and "bhattacharyya" are probability-divergence measures that treat each row as a distribution over the predictors, so they require non-negative values. "hellinger" and "bhattacharyya" further require each row to sum to 1. All four also use the RANN package and scale well to large datasets. "manhattan", "chebyshev", "canberra", "soergel", "lorentzian", "jeffreys", "topsoe", "jensen-shannon", "jensen_difference", "taneja", and "kumar-johnson"

compute an exact all-pairs distance matrix. This takes time and memory proportional to the square of the number of observations in a class, so these are best suited to smaller datasets. Everything from `"canberra"` onwards is a probability divergence requiring non-negative values, is provided by the philentropy package (which must be installed separately), and in the case of `"jeffreys"`, `"taneja"`, and `"kumar-johnson"` requires strictly positive values, since those divide by individual predictor values.

The probability divergences are meaningful for compositional predictors such as proportions or counts normalized per observation, and are generally not appropriate for standardized predictors.

`times` A positive integer for the maximum number of times ENN is applied. Defaults to `1` for a single pass. Values greater than `1` repeat the cleaning, stopping early once a pass removes no observations. Use `Inf` to repeat until convergence (Repeated Edited Nearest Neighbors).

`all_k` A logical. When `TRUE`, ENN is applied with an increasing number of neighbors, from `1` up to `neighbors`, cleaning the data at each step (All k-Nearest Neighbors). Takes precedence over `times`. Defaults to `FALSE`.

`kind_sel` A character string. The rule used to decide whether an observation is removed. `"mode"` (the default) removes an observation when the majority of its neighbors disagree with its class. `"all"` is stricter and removes an observation unless all of its neighbors share its class.

Details

Edited Nearest Neighbors (ENN) is a cleaning method. For each observation it finds the `neighbors` nearest neighbors and, if the class of the observation does not match the majority class among those neighbors, the observation is removed. This tends to remove noisy and borderline observations, which can lead to smoother decision boundaries.

Setting `times` greater than 1 applies ENN repeatedly, removing more noisy and borderline observations on each pass and stopping early once a pass removes nothing. This corresponds to Repeated Edited Nearest Neighbors (RENN).

Setting `all_k = TRUE` applies ENN with increasing numbers of neighbors, from `1` up to `neighbors`, cleaning the data at each step. This corresponds to All k-Nearest Neighbors (AllKNN) and takes precedence over `times`.

Setting `kind_sel = "all"` uses a stricter cleaning rule: instead of removing an observation when the majority of its neighbors disagree, it is removed unless every one of its neighbors shares its class. This removes more observations than the default `kind_sel = "mode"`.

All columns used in this function must be numeric with no missing data.

Use `times = Inf` to repeat ENN until convergence.

Value

A data.frame or tibble, depending on type of `df`.

References

Wilson, D. L. (1972). Asymptotic properties of nearest neighbor rules using edited data. IEEE Transactions on Systems, Man, and Cybernetics, (3), 408-421.

Tomek, I. (1976). An experiment with the edited nearest-neighbor rule. IEEE Transactions on Systems, Man, and Cybernetics, (6), 448-452.

See Also

`step_enn()` for step function of this method

Other Direct Implementations: `adasyn()`, `bsmote()`, `cluster_centroids()`, `cnn()`, `instance_hardness()`, `kmeans_smote()`, `ncl()`, `nearmiss()`, `oss()`, `rose()`, `smogn()`, `smote()`, `smoten()`, `smotenc()`, `svmsmote()`, `tomek()`

Examples

```
circle_numeric <- circle_example[, c("x", "y", "class")]

res <- enn(circle_numeric, var = "class")

res <- enn(circle_numeric, var = "class", neighbors = 5)

res <- enn(circle_numeric, var = "class", distance = "manhattan")

# Repeated Edited Nearest Neighbors (RENN)
res <- enn(circle_numeric, var = "class", times = Inf)

# All k-Nearest Neighbors (AllKNN)
res <- enn(circle_numeric, var = "class", all_k = TRUE)

# Stricter cleaning: remove unless all neighbors agree
res <- enn(circle_numeric, var = "class", kind_sel = "all")
```

`instance_hardness` *Remove hard to classify points*

Description

Under-samples the majority classes by removing the points that are hardest to classify.

Usage

```
instance_hardness(df, var, k = 5, under_ratio = 1, distance = "euclidean")
```

Arguments

df	data.frame or tibble. Must have 1 factor variable and remaining numeric variables.
var	Character, name of variable containing factor variable.
k	An integer. Number of nearest neighbors used to estimate the instance hardness of each observation.
under_ratio	A numeric value for the ratio of the majority-to-minority frequencies. The default value (1) means that all other levels are sampled down to have the same frequency as the least occurring level. A value of 2 would mean that the majority levels will have (at most) (approximately) twice as many rows than the minority level. A named numeric vector can be used instead to give different levels different targets, for example `c(a = 2, b = 3)`. The names must be levels of the outcome and the values are ratios of the minority level, exactly as in the single-number case. Levels that are not named are left untouched, as are rows with a missing outcome. Because a vector of targets is not a single value, supplying one means this argument can no longer be tuned. See `vignette("ratio", package = "themis")` for more details.
distance	A character string specifying the distance metric used for nearest neighbor calculations, defaulting to `"euclidean"`. The available metrics fall into three groups. `"euclidean"`, `"cosine"`, and `"mahalanobis"` use approximate nearest neighbors via the RANN package and scale well to large datasets. `"squared_chord"`, `"matusita"`, `"hellinger"`, and `"bhattacharyya"` are probability-divergence measures that treat each row as a distribution over the predictors, so they require non-negative values. `"hellinger"` and `"bhattacharyya"` further require each row to sum to 1. All four also use the RANN package and scale well to large datasets. `"manhattan"`, `"chebyshev"`, `"canberra"`, `"soergel"`, `"lorentzian"`, `"jeffreys"`, `"topsoe"`, `"jensen-shannon"`, `"jensen_difference"`, `"taneja"`, and `"kumar-johnson"` compute an exact all-pairs distance matrix. This takes time and memory proportional to the square of the number of observations in a class, so these are best suited to smaller datasets. Everything from `"canberra"` onwards is a probability divergence requiring non-negative values, is provided by the philentropy package (which must be installed separately), and in the case of `"jeffreys"`, `"taneja"`, and `"kumar-johnson"` requires strictly positive values, since those divide by individual predictor values. The probability divergences are meaningful for compositional predictors such as proportions or counts normalized per observation, and are generally not appropriate for standardized predictors.

Details

The instance hardness of each observation is estimated using the k-Disagreeing Neighbors measure: the proportion of the nearest neighbors that belong to a different class. Observations that are surrounded by points of a different class are considered hard to classify. For each majority class, the hardest observations are removed until the desired `under_ratio` is reached.

All columns used in this function must be numeric with no missing data.

Value

A data.frame or tibble, depending on type of `df`.

References

Smith, M. R., Martinez, T., & Giraud-Carrier, C. (2014). An instance level analysis of data complexity. Machine learning, 95(2), 225-256.

See Also

`step_instance_hardness()` for step function of this method

Other Direct Implementations: `adasyn()`, `bsmote()`, `cluster_centroids()`, `cnn()`, `enn()`, `kmeans_smote()`, `ncl()`, `nearmiss()`, `oss()`, `rose()`, `smogn()`, `smote()`, `smoten()`, `smotenc()`, `svmsmote()`, `tomek()`

Examples

```
circle_numeric <- circle_example[, c("x", "y", "class")]

res <- instance_hardness(circle_numeric, var = "class")

res <- instance_hardness(circle_numeric, var = "class", k = 10)

res <- instance_hardness(circle_numeric, var = "class", under_ratio = 1.5)

res <- instance_hardness(circle_numeric, var = "class", distance = "manhattan")
```

---

kmeans_smote *KMeans-SMOTE Algorithm*

Description

KMeans-SMOTE clusters the predictor space, keeps only the clusters that are dominated by the class being over-sampled, and generates new examples with SMOTE inside those clusters, giving sparser clusters more of the new points.

Usage

```
kmeans_smote(
  df,
  var,
  k = 2,
  over_ratio = 1,
  num_clusters = NULL,
  cluster_balance_threshold = 1,
```

```
  density_exponent = NULL,
  distance = "euclidean"
)
```

Arguments

df	data.frame or tibble. Must have 1 factor variable and remaining numeric variables.
var	Character, name of variable containing factor variable.
k	An integer. Number of nearest neighbor that are used to generate the new examples of the minority class.
over_ratio	A numeric value for the ratio of the minority-to-majority frequencies. The default value (1) means that all other levels are sampled up to have the same frequency as the most occurring level. A value of 0.5 would mean that the minority levels will have (at most) (approximately) half as many rows as the majority level. A named numeric vector can be used instead to give different levels different targets, for example `c(a = 1, b = 0.5)`. The names must be levels of the outcome and the values are ratios of the majority level, exactly as in the single-number case. Levels that are not named are left untouched, as are rows with a missing outcome. Because a vector of targets is not a single value, supplying one means this argument can no longer be tuned. See `vignette("ratio", package = "themis")` for more details.
num_clusters	An integer, the number of clusters to split the predictor space into, or `NULL` (the default) to use `max(2, floor(sqrt(n / 2)))` where `n` is the number of observations.
cluster_balance_threshold	A number. A cluster is used for over-sampling a class only if it contains at least `cluster_balance_threshold` times as many observations of that class as of all other classes combined. Defaults to `1`, meaning that the class must be at least as common as the rest of the data within the cluster. Smaller values keep more clusters.
density_exponent	A number, the exponent applied to the average pairwise distance when measuring how sparse a cluster is, or `NULL` (the default) to use the number of predictors.
distance	A character string specifying the distance metric used for nearest neighbor calculations, defaulting to `"euclidean"`. The available metrics fall into three groups. `"euclidean"`, `"cosine"`, and `"mahalanobis"` use approximate nearest neighbors via the RANN package and scale well to large datasets. `"squared_chord"`, `"matusita"`, `"hellinger"`, and `"bhattacharyya"` are probability-divergence measures that treat each row as a distribution over the predictors, so they require non-negative values. `"hellinger"` and `"bhattacharyya"` further require each row to sum to 1. All four also use the RANN package and scale well to large datasets. `"manhattan"`, `"chebyshev"`, `"canberra"`, `"soergel"`, `"lorentzian"`, `"jeffreys"`, `"topsoe"`, `"jensen-shannon"`, `"jensen_difference"`, `"taneja"`, and `"kumar-johnson"`

compute an exact all-pairs distance matrix. This takes time and memory proportional to the square of the number of observations in a class, so these are best suited to smaller datasets. Everything from `"canberra"` onwards is a probability divergence requiring non-negative values, is provided by the philentropy package (which must be installed separately), and in the case of `"jeffreys"`, `"taneja"`, and `"kumar-johnson"` requires strictly positive values, since those divide by individual predictor values.

The probability divergences are meaningful for compositional predictors such as proportions or counts normalized per observation, and are generally not appropriate for standardized predictors.

Details

KMeans-SMOTE combines k-means clustering with SMOTE to avoid generating synthetic points in regions where the minority class is not actually present. It works in three stages:

1. The predictor space of the whole data set is clustered with `stats::kmeans()` into `num_clusters` clusters.
2. Each cluster is either kept or filtered out. A cluster is kept for a given minority class if the ratio of that class's observations to all other observations in the cluster is at least `cluster_balance_threshold`, and if the cluster contains more than `neighbors` observations of that class so that interpolation is possible.
3. The synthetic points are distributed over the kept clusters according to how sparse each cluster is. The sparsity of a cluster is `mean_pairwise_distance ^ density_exponent / n`, where `n` is the number of minority observations in the cluster. Sparser clusters receive more points. Within each cluster the points are then generated by ordinary SMOTE interpolation, using only that cluster's observations as neighbors.

Filtering on cluster balance keeps synthetic points away from majority-dominated regions, and the density weighting counteracts within-class imbalance rather than only between-class imbalance.

The clustering is done once on all observations, so in a multi-class problem every minority class shares the same clusters; only the filtering and weighting are done per class. Note also that k-means always uses Euclidean distance. The `distance` argument affects the nearest-neighbor search used for interpolation and the sparsity calculation, not the clustering itself.

All columns used in this function must be numeric with no missing data.

Value

A data.frame or tibble, depending on type of `df`.

References

Douzas, G., Bacao, F., and Last, F. (2018). Improving imbalanced learning through a heuristic oversampling method based on k-means and SMOTE. Information Sciences, 465:1-20.

See Also

step_kmeans_smote() for step function of this method

Other Direct Implementations: adasyn(), bsmote(), cluster_centroids(), cnn(), enn(), instance_hardness(), ncl(), nearmiss(), oss(), rose(), smogn(), smote(), smoten(), smotenc(), svmsmote(), tomek()

Examples

```
circle_numeric <- circle_example[, c("x", "y", "class")]

res <- kmeans_smote(circle_numeric, var = "class")

res <- kmeans_smote(circle_numeric, var = "class", num_clusters = 10)

res <- kmeans_smote(circle_numeric, var = "class", over_ratio = 0.8)
```

---

ncl *Neighborhood Cleaning Rule*

Description

Under-samples the majority classes by cleaning noisy observations and observations that pollute the neighborhood of minority class observations.

Usage

```
ncl(df, var, neighbors = 3, distance = "euclidean", threshold_clean = 0.5)
```

Arguments

df	data.frame or tibble. Must have 1 factor variable and remaining numeric variables.
var	Character, name of variable containing factor variable.
neighbors	An integer. Number of nearest neighbor that are used to decide whether an observation is removed. Defaults to 3, unlike the over-sampling steps which default to 5.
distance	A character string specifying the distance metric used for nearest neighbor calculations, defaulting to "euclidean". The available metrics fall into three groups. "euclidean", "cosine", and "mahalanobis" use approximate nearest neighbors via the RANN package and scale well to large datasets. "squared_chord", "matusita", "hellinger", and "bhattacharyya" are probability-divergence measures that treat each row as a distribution over the predictors, so they require non-negative values. "hellinger" and "bhattacharyya" further require each row to sum to 1. All four also use the RANN package and scale well to large datasets.

"manhattan", "chebyshev", "canberra", "soergel", "lorentzian", "jeffreys", "topsoe", "jensen-shannon", "jensen_difference", "taneja", and "kumar-johnson" compute an exact all-pairs distance matrix. This takes time and memory proportional to the square of the number of observations in a class, so these are best suited to smaller datasets. Everything from "canberra" onwards is a probability divergence requiring non-negative values, is provided by the philentropy package (which must be installed separately), and in the case of "jeffreys", "taneja", and "kumar-johnson" requires strictly positive values, since those divide by individual predictor values.

The probability divergences are meaningful for compositional predictors such as proportions or counts normalized per observation, and are generally not appropriate for standardized predictors.

`threshold_clean`

A numeric. Majority classes are only cleaned around minority class observations when their size is greater than `threshold_clean` times the size of the minority class. Defaults to `0.5`.

Details

The Neighborhood Cleaning Rule (NCL) is a cleaning method that combines two passes over the data. First, it applies the Edited Nearest Neighbors rule, removing majority class observations whose class differs from the majority of their `neighbors` nearest neighbors. Second, for each minority class observation that is itself misclassified by its neighbors, the majority class observations among those neighbors are removed. Compared to Edited Nearest Neighbors, this focuses the cleaning on the neighborhoods of minority class observations.

The smallest class is treated as the minority class. Only majority classes larger than `threshold_clean` times the size of the minority class are cleaned in the second pass.

All columns used in this function must be numeric with no missing data.

Value

A data.frame or tibble, depending on type of `df`.

References

Laurikkala, J. (2001). Improving identification of difficult small classes by balancing class distribution. In Conference on Artificial Intelligence in Medicine in Europe (pp. 63-66). Springer.

See Also

`step_ncl()` for step function of this method

Other Direct Implementations: `adasyn()`, `bsmote()`, `cluster_centroids()`, `cnn()`, `enn()`, `instance_hardness()`, `kmeans_smote()`, `nearmiss()`, `oss()`, `rose()`, `smogn()`, `smote()`, `smoten()`, `smotenc()`, `svmsmote()`, `tomek()`

Examples

```
circle_numeric <- circle_example[, c("x", "y", "class")]

res <- ncl(circle_numeric, var = "class")

res <- ncl(circle_numeric, var = "class", neighbors = 5)

res <- ncl(circle_numeric, var = "class", distance = "manhattan")
```

---

nearmiss *Remove Points Near Other Classes*

Description

Generates synthetic positive instances using nearmiss algorithm.

Usage

```
nearmiss(
  df,
  var,
  k = 5,
  under_ratio = 1,
  distance = "euclidean",
  version = 1,
  n_neighbors_ver3 = 3
)
```

Arguments

df	data.frame or tibble. Must have 1 factor variable and remaining numeric variables.
var	Character, name of variable containing factor variable.
k	An integer. Number of nearest neighbor that are used to generate the new examples of the minority class.
under_ratio	A numeric value for the ratio of the majority-to-minority frequencies. The default value (1) means that all other levels are sampled down to have the same frequency as the least occurring level. A value of 2 would mean that the majority levels will have (at most) (approximately) twice as many rows than the minority level. A named numeric vector can be used instead to give different levels different targets, for example `c(a = 2, b = 3)`. The names must be levels of the outcome and the values are ratios of the minority level, exactly as in the single-number case. Levels that are not named are left untouched, as are rows with a missing outcome. Because a vector of targets is not a single value, supplying one means this argument can no longer be tuned. See `vignette("ratio", package = "themis")` for more details.

`distance` A character string specifying the distance metric used for nearest neighbor calculations, defaulting to `"euclidean"`. The available metrics fall into three groups.

`"euclidean"`, `"cosine"`, and `"mahalanobis"` use approximate nearest neighbors via the RANN package and scale well to large datasets.

`"squared_chord"`, `"matusita"`, `"hellinger"`, and `"bhattacharyya"` are probability-divergence measures that treat each row as a distribution over the predictors, so they require non-negative values. `"hellinger"` and `"bhattacharyya"` further require each row to sum to 1. All four also use the RANN package and scale well to large datasets.

`"manhattan"`, `"chebyshev"`, `"canberra"`, `"soergel"`, `"lorentzian"`, `"jeffreys"`, `"topsoe"`, `"jensen-shannon"`, `"jensen_difference"`, `"taneja"`, and `"kumar-johnson"` compute an exact all-pairs distance matrix. This takes time and memory proportional to the square of the number of observations in a class, so these are best suited to smaller datasets. Everything from `"canberra"` onwards is a probability divergence requiring non-negative values, is provided by the philentropy package (which must be installed separately), and in the case of `"jeffreys"`, `"taneja"`, and `"kumar-johnson"` requires strictly positive values, since those divide by individual predictor values.

The probability divergences are meaningful for compositional predictors such as proportions or counts normalized per observation, and are generally not appropriate for standardized predictors.

`version` An integer. Which of the three NearMiss variants to use, `1`, `2`, or `3`. Defaults to `1`. See the details section.

`n_neighbors_ver3`
An integer. The number of nearest neighbors used to build the candidate pool of the NearMiss-3 variant. Only used when `version = 3`. Defaults to `3`.

Details

The `version` argument selects between the three NearMiss variants:

`version = 1` Retains the points from the majority class which have the smallest mean distance to their nearest points in the minority class.

`version = 2` Retains the points from the majority class which have the smallest mean distance to their farthest points in the minority class.

`version = 3` Works in two stages. First, the `n_neighbors_ver3` nearest majority class neighbors of each minority class point form a candidate pool, and all other majority class points are removed. Then the points of that pool which have the largest mean distance to their nearest minority class points are retained.

Since the size of the NearMiss-3 candidate pool is governed by `n_neighbors_ver3` rather than by `under_ratio`, the pool can be smaller than the target set by `under_ratio`. The whole pool is then retained and the target is not reached.

With more than two classes, the mean distance is computed to the nearest points across all other classes, not only the minority class. This differs from imbalanced-learn, which measures distance to the minority class only. The binary case, the primary intended use, is unaffected.

All columns used in this function must be numeric with no missing data.

Value

A data.frame or tibble, depending on type of `df`.

References

Inderjeet Mani and I Zhang. knn approach to unbalanced data distributions: a case study involving information extraction. In Proceedings of workshop on learning from imbalanced datasets, 2003.

See Also

`step_nearmiss()` for step function of this method

Other Direct Implementations: `adasyn()`, `bsmote()`, `cluster_centroids()`, `cnn()`, `enn()`, `instance_hardness()`, `kmeans_smote()`, `ncl()`, `oss()`, `rose()`, `smogn()`, `smote()`, `smoten()`, `smotenc()`, `svmsmote()`, `tomek()`

Examples

```
circle_numeric <- circle_example[, c("x", "y", "class")]

res <- nearmiss(circle_numeric, var = "class")

res <- nearmiss(circle_numeric, var = "class", k = 10)

res <- nearmiss(circle_numeric, var = "class", under_ratio = 1.5)

res <- nearmiss(circle_numeric, var = "class", distance = "manhattan")

res <- nearmiss(circle_numeric, var = "class", version = 2)

res <- nearmiss(circle_numeric, var = "class", version = 3)

res <- nearmiss(
  circle_numeric,
  var = "class",
  version = 3,
  n_neighbors_ver3 = 10
)
```

---

oss *One-Sided Selection*

Description

Under-samples the majority classes by combining Condensed Nearest Neighbors and Tomek's links, first reducing redundant majority class observations and then removing majority class observations that form Tomek links with minority class observations.

Usage

```
oss(df, var, distance = "euclidean")
```

Arguments

df	data.frame or tibble. Must have 1 factor variable and remaining numeric variables.
var	Character, name of variable containing factor variable.
distance	A character string specifying the distance metric used for nearest neighbor calculations, defaulting to "euclidean". The available metrics fall into three groups. "euclidean", "cosine", and "mahalanobis" use approximate nearest neighbors via the RANN package and scale well to large datasets. "squared_chord", "matusita", "hellinger", and "bhattacharyya" are probability-divergence measures that treat each row as a distribution over the predictors, so they require non-negative values. "hellinger" and "bhattacharyya" further require each row to sum to 1. All four also use the RANN package and scale well to large datasets. "manhattan", "chebyshev", "canberra", "soergel", "lorentzian", "jeffreys", "topsoe", "jensen-shannon", "jensen_difference", "taneja", and "kumar-johnson" compute an exact all-pairs distance matrix. This takes time and memory proportional to the square of the number of observations in a class, so these are best suited to smaller datasets. Everything from "canberra" onwards is a probability divergence requiring non-negative values, is provided by the philentropy package (which must be installed separately), and in the case of "jeffreys", "taneja", and "kumar-johnson" requires strictly positive values, since those divide by individual predictor values. The probability divergences are meaningful for compositional predictors such as proportions or counts normalized per observation, and are generally not appropriate for standardized predictors.

Details

One-Sided Selection (OSS) is an under-sampling method that combines two cleaning techniques. It first applies Condensed Nearest Neighbors (CNN) to reduce the majority classes to a consistent subset that correctly classifies the data using a 1-nearest-neighbor rule, discarding redundant interior observations. It then applies Tomek's links to the remaining observations, removing the majority class observations that form Tomek links with minority class observations, cleaning the decision boundary.

The smallest class is treated as the minority class and is always kept. Because the CNN step relies on a random seed observation and a random scan order, results depend on the random seed.

With more than two classes, the Tomek's links step removes both members of a majority-majority link, not only links between a majority and the minority class. The binary case, the primary intended use, is unaffected.

All columns used in this function must be numeric with no missing data.

Value

A data.frame or tibble, depending on type of `df`.

References

Kubat, M., & Matwin, S. (1997). Addressing the curse of imbalanced training sets: one-sided selection. In ICML (Vol. 97, pp. 179-186).

See Also

`step_oss()` for step function of this method

Other Direct Implementations: `adasyn()`, `bsmote()`, `cluster_centroids()`, `cnn()`, `enn()`, `instance_hardness()`, `kmeans_smote()`, `ncl()`, `nearmiss()`, `rose()`, `smogn()`, `smote()`, `smoten()`, `smotenc()`, `svmsmote()`, `tomek()`

Examples

```
circle_numeric <- circle_example[, c("x", "y", "class")]

res <- oss(circle_numeric, var = "class")

res <- oss(circle_numeric, var = "class", distance = "manhattan")
```

---

rose *ROSE Algorithm*

Description

A thin wrapper around `ROSE::ROSE()` that generates a synthetic balanced sample by enlarging the feature space of minority and majority class examples.

Usage

```
rose(
  df,
  var,
  over_ratio = 1,
  minority_prop = 0.5,
  minority_smoothness = 1,
  majority_smoothness = 1
)
```

Arguments

df	A data.frame or tibble. Must have 1 factor variable with exactly 2 levels and remaining numeric variables.
var	Character, name of variable containing the 2-level factor variable.
over_ratio	A numeric value for the total size of the synthetic data relative to twice the size of the majority class. Unlike the other over-sampling steps this is not a per-class target, so a named vector of ratios is not accepted here.
minority_prop	A numeric value between 0 and 1 for the proportion of synthetic observations from the minority class. Defaults to 0.5, which generates an equal split of minority and majority synthetic observations. This parameter controls the class balance *within* the synthetic data, while over_ratio controls the *total size* of the synthetic data.
minority_smoothness	A numeric. Shrink factor to be multiplied by the smoothing parameters to estimate the conditional kernel density of the minority class. Defaults to 1.
majority_smoothness	A numeric. Shrink factor to be multiplied by the smoothing parameters to estimate the conditional kernel density of the majority class. Defaults to 1.

Details

The factor variable used to balance around must only have 2 levels.

The ROSE algorithm works by selecting an observation belonging to class k and generating new examples in its neighborhood, which is determined by a smoothing matrix H_k. Smaller values of minority_smoothness and majority_smoothness shrink the entries of H_k, producing tighter neighborhoods. This is a cautious choice when there is a concern that excessively large neighborhoods could blur the boundaries between classes.

This function is a thin wrapper around ROSE::ROSE(). For full details on the underlying implementation, see that function's documentation.

Value

A data.frame or tibble, depending on type of df.

References

Lunardon, N., Menardi, G., and Torelli, N. (2014). ROSE: a Package for Binary Imbalanced Learning. R Journal, 6:79–89.

Menardi, G. and Torelli, N. (2014). Training and assessing classification rules with imbalanced data. Data Mining and Knowledge Discovery, 28:92–122.

See Also

step_rose() for step function of this method

Other Direct Implementations: adasyn(), bsmote(), cluster_centroids(), cnn(), enn(), instance_hardness(), kmeans_smote(), ncl(), nearmiss(), oss(), smogn(), smote(), smoten(), smotenc(), svmsmote(), tomek()

Examples

```
rose(circle_example[, c("x", "y", "class")], var = "class")

rose(circle_example[, c("x", "y", "class")], var = "class", over_ratio = 0.8)
```

---

smogn *SMOGN Algorithm*

Description

SMOGN generates new examples for imbalanced regression problems. It identifies rare regions of the continuous outcome using a relevance function, over-samples those regions with a combination of SMOTE-style interpolation and the introduction of Gaussian noise, and under-samples the common regions.

Usage

```
smogn(
  df,
  var,
  threshold = 0.5,
  relevance = NULL,
  neighbors = 5,
  perturbation = 0.02,
  distance = "euclidean"
)
```

Arguments

df	data.frame or tibble. Must have 1 numeric outcome variable and remaining numeric variables.
var	Character, name of the numeric outcome variable.
threshold	A number between 0 and 1. Outcome values with a relevance at or above this value are treated as rare and over-sampled. Defaults to `0.5`.
relevance	A matrix of relevance control points, or `NULL` (default). When `NULL`, relevance is derived automatically from the boxplot extremes of the outcome. When supplied, the first column gives outcome values and the second column their relevance in `[0, 1]`.
neighbors	An integer. Number of nearest neighbor that are used to generate the new examples of the rare values.
perturbation	A number. The magnitude of the Gaussian noise added when generating synthetic examples in unsafe regions. Defaults to `0.02`.

`distance` A character string specifying the distance metric used for nearest neighbor calculations, defaulting to `"euclidean"`. The available metrics fall into three groups.

`"euclidean"`, `"cosine"`, and `"mahalanobis"` use approximate nearest neighbors via the RANN package and scale well to large datasets.

`"squared_chord"`, `"matusita"`, `"hellinger"`, and `"bhattacharyya"` are probability-divergence measures that treat each row as a distribution over the predictors, so they require non-negative values. `"hellinger"` and `"bhattacharyya"` further require each row to sum to 1. All four also use the RANN package and scale well to large datasets.

`"manhattan"`, `"chebyshev"`, `"canberra"`, `"soergel"`, `"lorentzian"`, `"jeffreys"`, `"topsoe"`, `"jensen-shannon"`, `"jensen_difference"`, `"taneja"`, and `"kumar-johnson"` compute an exact all-pairs distance matrix. This takes time and memory proportional to the square of the number of observations in a class, so these are best suited to smaller datasets. Everything from `"canberra"` onwards is a probability divergence requiring non-negative values, is provided by the philentropy package (which must be installed separately), and in the case of `"jeffreys"`, `"taneja"`, and `"kumar-johnson"` requires strictly positive values, since those divide by individual predictor values.

The probability divergences are meaningful for compositional predictors such as proportions or counts normalized per observation, and are generally not appropriate for standardized predictors.

Details

SMOGN is a pre-processing approach for imbalanced regression. A relevance function assigns each outcome value a relevance score, and values with a relevance at or above `threshold` are treated as rare. The data is split into contiguous bins of rare and common outcome values. Common bins are under-sampled and rare bins are over-sampled toward a balanced size. New rare examples are generated either by interpolating between an example and a nearby neighbor (when they are close enough to be considered safe) or by perturbing the example with Gaussian noise (when they are not), where the amount of noise is controlled by `perturbation`.

By default relevance is derived automatically from the boxplot extremes of the outcome, giving the median a relevance of 0 and the extreme values a relevance of 1. A matrix of relevance control points can instead be supplied through `relevance`, with the first column giving outcome values and the second column their relevance.

All columns used in this function must be numeric with no missing data.

Value

A data.frame or tibble, depending on type of `df`.

References

Branco, P., Torgo, L., and Ribeiro, R. P. (2017). SMOGN: a pre-processing approach for imbalanced regression. Proceedings of Machine Learning Research, 74:36-50.

See Also

step_smogn() for step function of this method

Other Direct Implementations: adasyn(), bsmote(), cluster_centroids(), cnn(), enn(), instance_hardness(), kmeans_smote(), ncl(), nearmiss(), oss(), rose(), smote(), smoten(), smotenc(), svmsmote(), tomek()

Examples

```
circle_numeric <- circle_example[, c("x", "y")]

res <- smogn(circle_numeric, var = "x")

res <- smogn(circle_numeric, var = "x", neighbors = 10)
```

---

smote *SMOTE Algorithm*

Description

SMOTE generates new examples of the minority class using nearest neighbors of these cases.

Usage

```
smote(df, var, k = 5, over_ratio = 1, distance = "euclidean")
```

Arguments

df	data.frame or tibble. Must have 1 factor variable and remaining numeric variables.
var	Character, name of variable containing factor variable.
k	An integer. Number of nearest neighbor that are used to generate the new examples of the minority class.
over_ratio	A numeric value for the ratio of the minority-to-majority frequencies. The default value (1) means that all other levels are sampled up to have the same frequency as the most occurring level. A value of 0.5 would mean that the minority levels will have (at most) (approximately) half as many rows as the majority level. A named numeric vector can be used instead to give different levels different targets, for example `c(a = 1, b = 0.5)`. The names must be levels of the outcome and the values are ratios of the majority level, exactly as in the single-number case. Levels that are not named are left untouched, as are rows with a missing outcome. Because a vector of targets is not a single value, supplying one means this argument can no longer be tuned. See `vignette("ratio", package = "themis")` for more details.

distance
: A character string specifying the distance metric used for nearest neighbor calculations, defaulting to "euclidean". The available metrics fall into three groups.

  "euclidean", "cosine", and "mahalanobis" use approximate nearest neighbors via the RANN package and scale well to large datasets.

  "squared_chord", "matusita", "hellinger", and "bhattacharyya" are probability-divergence measures that treat each row as a distribution over the predictors, so they require non-negative values. "hellinger" and "bhattacharyya" further require each row to sum to 1. All four also use the RANN package and scale well to large datasets.

  "manhattan", "chebyshev", "canberra", "soergel", "lorentzian", "jeffreys", "topsoe", "jensen-shannon", "jensen_difference", "taneja", and "kumar-johnson" compute an exact all-pairs distance matrix. This takes time and memory proportional to the square of the number of observations in a class, so these are best suited to smaller datasets. Everything from "canberra" onwards is a probability divergence requiring non-negative values, is provided by the philentropy package (which must be installed separately), and in the case of "jeffreys", "taneja", and "kumar-johnson" requires strictly positive values, since those divide by individual predictor values.

  The probability divergences are meaningful for compositional predictors such as proportions or counts normalized per observation, and are generally not appropriate for standardized predictors.

Details

SMOTE generates new examples of the minority class using nearest neighbors of these cases. For each existing minority class example, new examples are created by interpolating between the example and its nearest neighbors. The number of nearest neighbors used is controlled by the number of neighbors argument (k in `smote()`, `neighbors` in `step_smote()`), and the number of new examples generated is controlled by `over_ratio`.

All columns used in this function must be numeric with no missing data.

Value

A data.frame or tibble, depending on type of `df`.

References

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). Smote: Synthetic minority over-sampling technique. Journal of Artificial Intelligence Research, 16:321-357.

See Also

`step_smote()` for step function of this method

Other Direct Implementations: `adasyn()`, `bsmote()`, `cluster_centroids()`, `cnn()`, `enn()`, `instance_hardness()`, `kmeans_smote()`, `ncl()`, `nearmiss()`, `oss()`, `rose()`, `smogn()`, `smoten()`, `smotenc()`, `svmsmote()`, `tomek()`

Examples

```
circle_numeric <- circle_example[, c("x", "y", "class")]

res <- smote(circle_numeric, var = "class")

res <- smote(circle_numeric, var = "class", k = 10)

res <- smote(circle_numeric, var = "class", over_ratio = 0.8)

res <- smote(circle_numeric, var = "class", distance = "manhattan")
```

`smoten` *SMOTEN Algorithm*

Description

SMOTEN generates new examples of the minority class using nearest neighbors of these cases, for data sets where all predictors are categorical.

Usage

```
smoten(df, var, k = 5, over_ratio = 1)
```

Arguments

`df`	data.frame or tibble. Must have 1 factor variable used as the outcome and remaining categorical (factor or character) variables.
`var`	Character, name of variable containing factor variable.
`k`	An integer. Number of nearest neighbor that are used to generate the new examples of the minority class.
`over_ratio`	A numeric value for the ratio of the minority-to-majority frequencies. The default value (1) means that all other levels are sampled up to have the same frequency as the most occurring level. A value of 0.5 would mean that the minority levels will have (at most) (approximately) half as many rows as the majority level. A named numeric vector can be used instead to give different levels different targets, for example `c(a = 1, b = 0.5)`. The names must be levels of the outcome and the values are ratios of the majority level, exactly as in the single-number case. Levels that are not named are left untouched, as are rows with a missing outcome. Because a vector of targets is not a single value, supplying one means this argument can no longer be tuned. See `vignette("ratio", package = "themis")` for more details.

Details

SMOTEN is a variant of SMOTE designed for data sets where all predictors are categorical (nominal). Since standard SMOTE relies on interpolation in continuous space, it cannot be applied to categorical features directly. SMOTEN instead uses the Value Difference Metric (VDM) to measure the distance between observations. For each minority class example, new synthetic examples are generated by taking the most common value of each predictor among its nearest neighbors. The number of new examples generated is controlled by `over_ratio`.

All columns other than `var` must be categorical (factor or character) with no missing data.

Value

A data.frame or tibble, depending on type of `df`.

References

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). Smote: Synthetic minority over-sampling technique. Journal of Artificial Intelligence Research, 16:321-357.

See Also

`step_smoten()` for step function of this method

Other Direct Implementations: `adasyn()`, `bsmote()`, `cluster_centroids()`, `cnn()`, `enn()`, `instance_hardness()`, `kmeans_smote()`, `ncl()`, `nearmiss()`, `oss()`, `rose()`, `smogn()`, `smote()`, `smotenc()`, `svmsmote()`, `tomek()`

Examples

```
df <- data.frame(
  x = factor(sample(letters[1:3], 100, replace = TRUE)),
  y = factor(sample(letters[1:2], 100, replace = TRUE)),
  class = factor(c(rep("minority", 20), rep("majority", 80)))
)

res <- smoten(df, var = "class")

res <- smoten(df, var = "class", k = 10)

res <- smoten(df, var = "class", over_ratio = 0.8)
```

---

smotenc *SMOTENC Algorithm*

Description

SMOTENC generates new examples of the minority class using nearest neighbors of these cases, and can handle categorical variables

Usage

```
smotenc(df, var, k = 5, over_ratio = 1)
```

Arguments

df	data.frame or tibble. Must have 1 factor variable and remaining numeric variables.
var	Character, name of variable containing factor variable.
k	An integer. Number of nearest neighbor that are used to generate the new examples of the minority class.
over_ratio	A numeric value for the ratio of the minority-to-majority frequencies. The default value (1) means that all other levels are sampled up to have the same frequency as the most occurring level. A value of 0.5 would mean that the minority levels will have (at most) (approximately) half as many rows as the majority level. A named numeric vector can be used instead to give different levels different targets, for example `c(a = 1, b = 0.5)`. The names must be levels of the outcome and the values are ratios of the majority level, exactly as in the single-number case. Levels that are not named are left untouched, as are rows with a missing outcome. Because a vector of targets is not a single value, supplying one means this argument can no longer be tuned. See `vignette("ratio", package = "themis")` for more details.

Details

SMOTENC extends SMOTE to handle data sets with a mix of numeric and categorical predictors. For each minority class example, new synthetic examples are generated by interpolating between the example and its nearest neighbors using Gower's distance. Numeric features are interpolated continuously; categorical features take the most common value among the neighbors. The number of new examples generated is controlled by `over_ratio`.

Columns can be numeric and categorical with no missing data.

Value

A data.frame or tibble, depending on type of `df`.

References

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). Smote: Synthetic minority over-sampling technique. Journal of Artificial Intelligence Research, 16:321-357.

Gower, J. C. (1971). A general coefficient of similarity and some of its properties. Biometrics 27(4):857-871. (For the distance metric used)

See Also

`step_smotenc()` for step function of this method

Other Direct Implementations: adasyn(), bsmote(), cluster_centroids(), cnn(), enn(), instance_hardness(), kmeans_smote(), ncl(), nearmiss(), oss(), rose(), smogn(), smote(), smoten(), svmsmote(), tomek()

Examples

```
circle_numeric <- circle_example[, c("x", "y", "class")]

res <- smotenc(circle_numeric, var = "class")

res <- smotenc(circle_numeric, var = "class", k = 10)

res <- smotenc(circle_numeric, var = "class", over_ratio = 0.8)
```

---

step_adasyn *Apply Adaptive Synthetic Algorithm*

Description

step_adasyn() creates a *specification* of a recipe step that generates synthetic positive instances using ADASYN algorithm.

Usage

```
step_adasyn(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  column = NULL,
  over_ratio = 1,
  neighbors = 5,
  distance = "euclidean",
  indicator_column = NULL,
  skip = TRUE,
  seed = sample.int(10^5, 1),
  id = rand_id("adasyn")
)
```

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 which variable is used to sample the data. See recipes::selections for more details. The selection should result in *single factor variable*. For the tidy method, these are not currently used.
role	Not used by this step since no new variables are created.

`trained` A logical to indicate if the quantities for preprocessing have been estimated.

`column` A character string of the variable name that will be populated (eventually) by the `...` selectors.

`over_ratio` A numeric value for the ratio of the minority-to-majority frequencies. The default value (1) means that all other levels are sampled up to have the same frequency as the most occurring level. A value of 0.5 would mean that the minority levels will have (at most) (approximately) half as many rows as the majority level.

A named numeric vector can be used instead to give different levels different targets, for example `c(a = 1, b = 0.5)`. The names must be levels of the outcome and the values are ratios of the majority level, exactly as in the single-number case. Levels that are not named are left untouched, as are rows with a missing outcome. Because a vector of targets is not a single value, supplying one means this argument can no longer be tuned. See `vignette("ratio", package = "themis")` for more details.

`neighbors` An integer. Number of nearest neighbor that are used to generate the new examples of the minority class.

`distance` A character string specifying the distance metric used for nearest neighbor calculations, defaulting to `"euclidean"`. The available metrics fall into three groups.

`"euclidean"`, `"cosine"`, and `"mahalanobis"` use approximate nearest neighbors via the RANN package and scale well to large datasets.

`"squared_chord"`, `"matusita"`, `"hellinger"`, and `"bhattacharyya"` are probability-divergence measures that treat each row as a distribution over the predictors, so they require non-negative values. `"hellinger"` and `"bhattacharyya"` further require each row to sum to 1. All four also use the RANN package and scale well to large datasets.

`"manhattan"`, `"chebyshev"`, `"canberra"`, `"soergel"`, `"lorentzian"`, `"jeffreys"`, `"topsoe"`, `"jensen-shannon"`, `"jensen_difference"`, `"taneja"`, and `"kumar-johnson"` compute an exact all-pairs distance matrix. This takes time and memory proportional to the square of the number of observations in a class, so these are best suited to smaller datasets. Everything from `"canberra"` onwards is a probability divergence requiring non-negative values, is provided by the philentropy package (which must be installed separately), and in the case of `"jeffreys"`, `"taneja"`, and `"kumar-johnson"` requires strictly positive values, since those divide by individual predictor values.

The probability divergences are meaningful for compositional predictors such as proportions or counts normalized per observation, and are generally not appropriate for standardized predictors.

`indicator_column`

A single string or `NULL` (the default). If a string is given, a logical column with that name is added to the output, marking rows added by the step (`TRUE`) vs rows from the original data (`FALSE`).

`skip` A logical. Should the step be skipped when the recipe is baked by `bake()`? While all operations are baked when `prep()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)).

	Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.
`seed`	An integer that will be used as the seed when applied.
`id`	A character string that is unique to this step to identify it.

Details

ADASYN is an extension of SMOTE that adaptively generates synthetic minority class examples based on the local distribution of each minority instance. Instead of generating the same number of synthetic examples for every minority instance, ADASYN generates more synthetic examples for instances that are harder to learn, specifically those surrounded by more majority class neighbors. This focuses synthetic data generation on the difficult boundary regions of the minority class, resulting in a more informative augmentation than standard SMOTE.

All columns in the data are sampled and returned by `recipes::juice()` and `recipes::bake()`.

All columns used in this step must be numeric with no missing data.

When used in modeling, users should strongly consider using the option `skip = TRUE` so that the extra sampling is *not* conducted outside of the training set.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` which is the variable used to sample.

Minimum observations

Each minority class must have at least `neighbors + 1` observations to perform the ADASYN algorithm.

Tidying

When you `tidy()` this step, a tibble is returned with columns `terms` and `id`:

terms character, the selectors or variables selected

id character, id of this step

Tuning Parameters

This step has 2 tuning parameters:

- `over_ratio`: Over-Sampling Ratio (type: double, default: 1)
- `neighbors`: # Nearest Neighbors (type: integer, default: 5)

Case weights

The underlying operation does not allow for case weights. Supplying data with a case weights column to this step results in an error.

References

He, H., Bai, Y., Garcia, E. and Li, S. 2008. ADASYN: Adaptive synthetic sampling approach for imbalanced learning. Proceedings of IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference. pp.1322-1328.

See Also

adasyn() for direct implementation

Other Steps for over-sampling: step_bsmote(), step_kmeans_smote(), step_rose(), step_smogn(), step_smote(), step_smoten(), step_smotenc(), step_svmsmote(), step_upsample()

Examples

```
library(recipes)
library(modeldata)
data(hpc_data)

hpc_data0 <- hpc_data |>
  select(-protocol, -day)

orig <- count(hpc_data0, class, name = "orig")
orig

up_rec <- recipe(class ~ ., data = hpc_data0) |>
  # Bring the minority levels up to about 1000 each
  # 1000/2211 is approx 0.4523
  step_adasyn(class, over_ratio = 0.4523) |>
  prep()

training <- up_rec |>
  bake(new_data = NULL) |>
  count(class, name = "training")
training

# Since `skip` defaults to TRUE, baking the step has no effect
baked <- up_rec |>
  bake(new_data = hpc_data0) |>
  count(class, name = "baked")
baked

# Note that if the original data contained more rows than the
# target n (= ratio * majority_n), the data are left alone:
orig |>
  left_join(training, by = "class") |>
  left_join(baked, by = "class")

library(ggplot2)

ggplot(circle_example, aes(x, y, color = class)) +
  geom_point() +
  labs(title = "Without ADASYN")
```

```
recipe(class ~ x + y, data = circle_example) |>
  step_adasyn(class) |>
  prep() |>
  bake(new_data = NULL) |>
  ggplot(aes(x, y, color = class)) +
  geom_point() +
  labs(title = "With ADASYN")
```

`step_bsmote` *Apply Borderline-SMOTE Algorithm*

Description

`step_bsmote()` creates a *specification* of a recipe step that generate new examples of the minority class using nearest neighbors of these cases in the border region between classes.

Usage

```
step_bsmote(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  column = NULL,
  over_ratio = 1,
  neighbors = 5,
  all_neighbors = FALSE,
  distance = "euclidean",
  indicator_column = NULL,
  skip = TRUE,
  seed = sample.int(10^5, 1),
  id = rand_id("bsmote")
)
```

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 which variable is used to sample the data. See recipes::selections for more details. The selection should result in *single factor variable*. For the `tidy` method, these are not currently used.
`role`	Not used by this step since no new variables are created.
`trained`	A logical to indicate if the quantities for preprocessing have been estimated.
`column`	A character string of the variable name that will be populated (eventually) by the `...` selectors.

over_ratio
: A numeric value for the ratio of the minority-to-majority frequencies. The default value (1) means that all other levels are sampled up to have the same frequency as the most occurring level. A value of 0.5 would mean that the minority levels will have (at most) (approximately) half as many rows as the majority level.

  A named numeric vector can be used instead to give different levels different targets, for example `c(a = 1, b = 0.5)`. The names must be levels of the outcome and the values are ratios of the majority level, exactly as in the single-number case. Levels that are not named are left untouched, as are rows with a missing outcome. Because a vector of targets is not a single value, supplying one means this argument can no longer be tuned. See `vignette("ratio", package = "themis")` for more details.

neighbors
: An integer. Number of nearest neighbor that are used to generate the new examples of the minority class.

all_neighbors
: Type of two borderline-SMOTE method. Defaults to FALSE. See details.

distance
: A character string specifying the distance metric used for nearest neighbor calculations, defaulting to `"euclidean"`. The available metrics fall into three groups.

  `"euclidean"`, `"cosine"`, and `"mahalanobis"` use approximate nearest neighbors via the RANN package and scale well to large datasets.

  `"squared_chord"`, `"matusita"`, `"hellinger"`, and `"bhattacharyya"` are probability-divergence measures that treat each row as a distribution over the predictors, so they require non-negative values. `"hellinger"` and `"bhattacharyya"` further require each row to sum to 1. All four also use the RANN package and scale well to large datasets.

  `"manhattan"`, `"chebyshev"`, `"canberra"`, `"soergel"`, `"lorentzian"`, `"jeffreys"`, `"topsoe"`, `"jensen-shannon"`, `"jensen_difference"`, `"taneja"`, and `"kumar-johnson"` compute an exact all-pairs distance matrix. This takes time and memory proportional to the square of the number of observations in a class, so these are best suited to smaller datasets. Everything from `"canberra"` onwards is a probability divergence requiring non-negative values, is provided by the philentropy package (which must be installed separately), and in the case of `"jeffreys"`, `"taneja"`, and `"kumar-johnson"` requires strictly positive values, since those divide by individual predictor values.

  The probability divergences are meaningful for compositional predictors such as proportions or counts normalized per observation, and are generally not appropriate for standardized predictors.

indicator_column
: A single string or `NULL` (the default). If a string is given, a logical column with that name is added to the output, marking rows added by the step (`TRUE`) vs rows from the original data (`FALSE`).

skip
: A logical. Should the step be skipped when the recipe is baked by `bake()`? While all operations are baked when `prep()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

seed	An integer that will be used as the seed when applied.
id	A character string that is unique to this step to identify it.

Details

BSMOTE (borderline-SMOTE) works the same way as SMOTE, except that instead of generating points around every point of the minority class each point is first classified into the boxes "danger" and "not". For each point the nearest neighbors are calculated. If all the neighbors come from a different class it is labeled noise and put into the "not" box. If more than half of the neighbors come from a different class it is labeled "danger". Points are generated around points labeled "danger".

If `all_neighbors = FALSE` then points are generated between nearest neighbors in its own class. If `all_neighbors = TRUE` then points are generated between any nearest neighbors. See examples for visualization.

SMOTE generates new examples of the minority class using nearest neighbors of these cases. For each existing minority class example, new examples are created by interpolating between the example and its nearest neighbors. The number of nearest neighbors used is controlled by the number of neighbors argument (k in `smote()`, `neighbors` in `step_smote()`), and the number of new examples generated is controlled by `over_ratio`.

All columns in the data are sampled and returned by `recipes::juice()` and `recipes::bake()`.

All columns used in this step must be numeric with no missing data.

When used in modeling, users should strongly consider using the option `skip = TRUE` so that the extra sampling is *not* conducted outside of the training set.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` which is the variable used to sample.

Minimum observations

Each minority class must have at least `neighbors + 1` observations to perform the BSMOTE algorithm.

Tidying

When you `tidy()` this step, a tibble is returned with columns `terms` and `id`:

terms character, the selectors or variables selected

id character, id of this step

Tuning Parameters

This step has 3 tuning parameters:

- `over_ratio`: Over-Sampling Ratio (type: double, default: 1)
- `neighbors`: # Nearest Neighbors (type: integer, default: 5)
- `all_neighbors`: All Neighbors (type: logical, default: FALSE)

Case weights

The underlying operation does not allow for case weights. Supplying data with a case weights column to this step results in an error.

References

Hui Han, Wen-Yuan Wang, and Bing-Huan Mao. Borderline-smote: a new over-sampling method in imbalanced data sets learning. In International Conference on Intelligent Computing, pages 878–887. Springer, 2005.

See Also

`bsmote()` for direct implementation

Other Steps for over-sampling: `step_adasyn()`, `step_kmeans_smote()`, `step_rose()`, `step_smogn()`, `step_smote()`, `step_smoten()`, `step_smotenc()`, `step_svmsmote()`, `step_upsample()`

Examples

```
library(recipes)
library(modeldata)
data(hpc_data)

hpc_data0 <- hpc_data |>
  select(-protocol, -day)

orig <- count(hpc_data0, class, name = "orig")
orig

up_rec <- recipe(class ~ ., data = hpc_data0) |>
  # Bring the minority levels up to about 1000 each
  # 1000/2211 is approx 0.4523
  step_bsmote(class, over_ratio = 0.4523) |>
  prep()

training <- up_rec |>
  bake(new_data = NULL) |>
  count(class, name = "training")
training

# Since `skip` defaults to TRUE, baking the step has no effect
baked <- up_rec |>
  bake(new_data = hpc_data0) |>
  count(class, name = "baked")
baked

# Note that if the original data contained more rows than the
# target n (= ratio * majority_n), the data are left alone:
orig |>
  left_join(training, by = "class") |>
  left_join(baked, by = "class")
```

```
library(ggplot2)

ggplot(circle_example, aes(x, y, color = class)) +
  geom_point() +
  labs(title = "Without SMOTE")

recipe(class ~ x + y, data = circle_example) |>
  step_bsmote(class, all_neighbors = FALSE) |>
  prep() |>
  bake(new_data = NULL) |>
  ggplot(aes(x, y, color = class)) +
  geom_point() +
  labs(title = "With borderline-SMOTE, all_neighbors = FALSE")

recipe(class ~ x + y, data = circle_example) |>
  step_bsmote(class, all_neighbors = TRUE) |>
  prep() |>
  bake(new_data = NULL) |>
  ggplot(aes(x, y, color = class)) +
  geom_point() +
  labs(title = "With borderline-SMOTE, all_neighbors = TRUE")
```

---

step_cluster_centroids

Under-Sampling by Cluster Centroids

Description

step_cluster_centroids() creates a *specification* of a recipe step that removes majority class instances by replacing each majority class with cluster representatives found with k-means.

Usage

```
step_cluster_centroids(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  column = NULL,
  under_ratio = 1,
  voting = "soft",
  skip = TRUE,
  seed = sample.int(10^5, 1),
  distance_with = recipes::all_predictors(),
  id = rand_id("cluster_centroids")
)
```

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 which variable is used to sample the data. See recipes::selections for more details. The selection should result in *single factor variable*. For the tidy method, these are not currently used.
role	Not used by this step since no new variables are created.
trained	A logical to indicate if the quantities for preprocessing have been estimated.
column	A character string of the variable name that will be populated (eventually) by the ... selectors.
under_ratio	A numeric value for the ratio of the majority-to-minority frequencies. The default value (1) means that all other levels are sampled down to have the same frequency as the least occurring level. A value of 2 would mean that the majority levels will have (at most) (approximately) twice as many rows than the minority level. A named numeric vector can be used instead to give different levels different targets, for example c(a = 2, b = 3). The names must be levels of the outcome and the values are ratios of the minority level, exactly as in the single-number case. Levels that are not named are left untouched, as are rows with a missing outcome. Because a vector of targets is not a single value, supplying one means this argument can no longer be tuned. See vignette("ratio", package = "themis") for more details.
voting	A character string. Either "soft" (the default) to use the cluster centroids themselves, or "hard" to use the observation closest to each centroid.
skip	A logical. Should the step be skipped when the recipe is baked by bake()? While all operations are baked when prep() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations.
seed	An integer that will be used as the seed when applied.
distance_with	A call to a selector function to choose which variables are used to compute the clusters. Defaults to recipes::all_predictors(). The variable selected by ... is always excluded from the clustering.
id	A character string that is unique to this step to identify it.

Details

Each class larger than the target count is summarized by running k-means on the observations of that class, using as many clusters as the target count. The class is then replaced by one representative per cluster:

voting = "soft" the cluster centroids themselves are used, so the returned observations are synthetic points that need not appear in the input.

voting = "hard" the observation closest to each centroid is used, so all returned observations are real rows. The representative is always picked from the class being under-sampled.

This makes it the one *prototype generation* under-sampler in this package. The other under-sampling methods perform *prototype selection*, keeping a subset of the original rows.

Because two clusters can share the same closest observation, `voting = "hard"` can return slightly fewer observations than the target count.

All columns in the data are sampled and returned by `recipes::juice()` and `recipes::bake()`.

All columns selected by `distance_with` must be numeric with no missing data.

When used in modeling, users should strongly consider using the option `skip = TRUE` so that the extra sampling is *not* conducted outside of the training set.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` which is the variable used to sample.

Non-predictor columns

With `voting = "soft"` the observations of an under-sampled class are replaced by synthetic points, and columns that are not selected by `distance_with` have no value for those points. Such columns are set to `NA`, as they are for the over-sampling steps that synthesize observations. Use `voting = "hard"` if these columns must be kept intact.

Tidying

When you `tidy()` this step, a tibble is returned with columns `terms` and `id`:

terms character, the selectors or variables selected

id character, id of this step

Tuning Parameters

This step has 1 tuning parameters:

- `under_ratio`: Under-Sampling Ratio (type: double, default: 1)

Case weights

The underlying operation does not allow for case weights. Supplying data with a case weights column to this step results in an error.

See Also

`cluster_centroids()` for direct implementation

Other Steps for under-sampling: `step_cnn()`, `step_downsample()`, `step_enn()`, `step_instance_hardness()`, `step_ncl()`, `step_nearmiss()`, `step_oss()`, `step_tomek()`

Examples

```
library(recipes)
library(modeldata)
data(hpc_data)

hpc_data0 <- hpc_data |>
  select(-protocol, -day)

orig <- count(hpc_data0, class, name = "orig")
orig

up_rec <- recipe(class ~ ., data = hpc_data0) |>
  # Bring the majority levels down to about 1000 each
  # 1000/259 is approx 3.862
  step_cluster_centroids(class, under_ratio = 3.862) |>
  prep()

training <- up_rec |>
  bake(new_data = NULL) |>
  count(class, name = "training")
training

# Since `skip` defaults to TRUE, baking the step has no effect
baked <- up_rec |>
  bake(new_data = hpc_data0) |>
  count(class, name = "baked")
baked

library(ggplot2)

ggplot(circle_example, aes(x, y, color = class)) +
  geom_point() +
  labs(title = "Without ClusterCentroids") +
  xlim(c(1, 15)) +
  ylim(c(1, 15))

recipe(class ~ x + y, data = circle_example) |>
  step_cluster_centroids(class) |>
  prep() |>
  bake(new_data = NULL) |>
  ggplot(aes(x, y, color = class)) +
  geom_point() +
  labs(title = "With ClusterCentroids") +
  xlim(c(1, 15)) +
  ylim(c(1, 15))
```

---

step_cnn *Condensed Nearest Neighbors*

Description

`step_cnn()` creates a *specification* of a recipe step that removes redundant majority class observations, keeping only a consistent subset that correctly classifies the data using a 1-nearest-neighbor rule.

Usage

```
step_cnn(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  column = NULL,
  distance = "euclidean",
  skip = TRUE,
  seed = sample.int(10^5, 1),
  distance_with = recipes::all_predictors(),
  id = rand_id("cnn")
)
```

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 which variable is used to sample the data. See recipes::selections for more details. The selection should result in *single factor variable*. For the `tidy` method, these are not currently used.
`role`	Not used by this step since no new variables are created.
`trained`	A logical to indicate if the quantities for preprocessing have been estimated.
`column`	A character string of the variable name that will be populated (eventually) by the `...` selectors.
`distance`	A character string specifying the distance metric used for nearest neighbor calculations, defaulting to `"euclidean"`. The available metrics fall into three groups. `"euclidean"`, `"cosine"`, and `"mahalanobis"` use approximate nearest neighbors via the RANN package and scale well to large datasets. `"squared_chord"`, `"matusita"`, `"hellinger"`, and `"bhattacharyya"` are probability-divergence measures that treat each row as a distribution over the predictors, so they require non-negative values. `"hellinger"` and `"bhattacharyya"` further require each row to sum to 1. All four also use the RANN package and scale well to large datasets. `"manhattan"`, `"chebyshev"`, `"canberra"`, `"soergel"`, `"lorentzian"`, `"jeffreys"`, `"topsoe"`, `"jensen-shannon"`, `"jensen_difference"`, `"taneja"`, and `"kumar-johnson"` compute an exact all-pairs distance matrix. This takes time and memory proportional to the square of the number of observations in a class, so these are best suited to smaller datasets. Everything from `"canberra"` onwards is a probability divergence requiring non-negative values, is provided by the philentropy

package (which must be installed separately), and in the case of "jeffreys", "taneja", and "kumar-johnson" requires strictly positive values, since those divide by individual predictor values.

The probability divergences are meaningful for compositional predictors such as proportions or counts normalized per observation, and are generally not appropriate for standardized predictors.

skip — A logical. Should the step be skipped when the recipe is baked by bake()? While all operations are baked when prep() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations.

seed — An integer that will be used as the seed when applied.

distance_with — A call to a selector function to choose which variables are used for distance calculations. Defaults to recipes::all_predictors(). The variable selected by ... is always excluded from the distance calculations.

id — A character string that is unique to this step to identify it.

Details

Condensed Nearest Neighbors (CNN) is an under-sampling method that reduces the majority classes to a consistent subset: a subset that classifies the original data correctly using a 1-nearest-neighbor rule. It starts with a "store" containing all minority class observations and one randomly chosen majority class observation. It then repeatedly scans the remaining majority class observations and moves any that are misclassified by a 1-nearest neighbor fit on the current store into the store. This continues until a full pass adds no new observations. The observations left outside the store are removed.

The smallest class is treated as the minority class and is always kept. CNN tends to keep observations near the decision boundary while discarding redundant interior observations. Because the seed observation and the scan order are random, results depend on the random seed.

All variables selected by distance_with must be numeric with no missing data.

All columns in the data are sampled and returned by recipes::juice() and recipes::bake().

When used in modeling, users should strongly consider using the option skip = TRUE so that the extra sampling is *not* conducted outside of the training set.

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns terms which is the variable used to sample.

Tidying

When you tidy() this step, a tibble is returned with columns terms and id:

terms character, the selectors or variables selected

id character, id of this step

Case weights

The underlying operation does not allow for case weights. Supplying data with a case weights column to this step results in an error.

References

Hart, P. (1968). The condensed nearest neighbor rule. IEEE Transactions on Information Theory, 14(3), 515-516.

See Also

`cnn()` for direct implementation

Other Steps for under-sampling: `step_cluster_centroids()`, `step_downsample()`, `step_enn()`, `step_instance_hardness()`, `step_ncl()`, `step_nearmiss()`, `step_oss()`, `step_tomek()`

Examples

```
library(recipes)
library(modeldata)
data(hpc_data)

hpc_data0 <- hpc_data |>
  select(-protocol, -day)

orig <- count(hpc_data0, class, name = "orig")
orig

up_rec <- recipe(class ~ ., data = hpc_data0) |>
  step_cnn(class) |>
  prep()

training <- up_rec |>
  bake(new_data = NULL) |>
  count(class, name = "training")
training

# Since `skip` defaults to TRUE, baking the step has no effect
baked <- up_rec |>
  bake(new_data = hpc_data0) |>
  count(class, name = "baked")
baked

orig |>
  left_join(training, by = "class") |>
  left_join(baked, by = "class")

library(ggplot2)

ggplot(circle_example, aes(x, y, color = class)) +
  geom_point() +
  labs(title = "Without CNN") +
```

```
  xlim(c(1, 15)) +
  ylim(c(1, 15))

recipe(class ~ x + y, data = circle_example) |>
  step_cnn(class) |>
  prep() |>
  bake(new_data = NULL) |>
  ggplot(aes(x, y, color = class)) +
  geom_point() +
  labs(title = "With CNN") +
  xlim(c(1, 15)) +
  ylim(c(1, 15))
```

`step_downsample` *Down-Sample a Data Set Based on a Factor Variable*

Description

`step_downsample()` creates a *specification* of a recipe step that will remove rows of a data set to make the occurrence of levels in a specific factor level equal.

Usage

```
step_downsample(
  recipe,
  ...,
  under_ratio = 1,
  ratio = deprecated(),
  replacement = FALSE,
  role = NA,
  trained = FALSE,
  column = NULL,
  target = NA,
  skip = TRUE,
  seed = sample.int(10^5, 1),
  id = rand_id("downsample")
)
```

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 which variable is used to sample the data. See recipes::selections for more details. The selection should result in *single factor variable*. For the `tidy` method, these are not currently used.

under_ratio
: A numeric value for the ratio of the majority-to-minority frequencies. The default value (1) means that all other levels are sampled down to have the same frequency as the least occurring level. A value of 2 would mean that the majority levels will have (at most) (approximately) twice as many rows than the minority level.

  A named numeric vector can be used instead to give different levels different targets, for example `c(a = 2, b = 3)`. The names must be levels of the outcome and the values are ratios of the minority level, exactly as in the single-number case. Levels that are not named are left untouched, as are rows with a missing outcome. Because a vector of targets is not a single value, supplying one means this argument can no longer be tuned. See `vignette("ratio", package = "themis")` for more details.

ratio
: Deprecated argument; same as `under_ratio`

replacement
: A logical value indicating whether the under-sample should be drawn with replacement. Defaults to `FALSE`, in which case each retained row is distinct. When `TRUE` the same row can be selected more than once, giving a bootstrapped under-sample. The number of rows retained for each level is unaffected, so no level is sampled up beyond its original size.

role
: Not used by this step since no new variables are created.

trained
: A logical to indicate if the quantities for preprocessing have been estimated.

column
: A character string of the variable name that will be populated (eventually) by the ... selectors.

target
: A named numeric vector giving the number of rows to sample each level down to. This should not be set by the user and will be populated by `prep`.

skip
: A logical. Should the step be skipped when the recipe is baked by `bake()`? While all operations are baked when `prep()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

seed
: An integer that will be used as the seed when applied.

id
: A character string that is unique to this step to identify it.

Details

Down-sampling is intended to be performed on the *training* set alone. For this reason, the default is `skip = TRUE`.

If there are missing values in the factor variable that is used to define the sampling, missing data are selected at random in the same way that the other factor levels are sampled. Missing values are not used to determine the amount of data in the minority level

For any data with factor levels occurring with the same frequency as the minority level, all data will be retained.

All columns in the data are sampled and returned by `recipes::juice()` and `recipes::bake()`.

Keep in mind that the location of down-sampling in the step may have effects. For example, if centering and scaling, it is not clear whether those operations should be conducted *before* or *after* rows are removed.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` which is the variable used to sample.

Tidying

When you `tidy()` this step, a tibble is returned with columns `terms` and `id`:

terms character, the selectors or variables selected

id character, id of this step

Tuning Parameters

This step has 1 tuning parameters:

- `under_ratio`: Under-Sampling Ratio (type: double, default: 1)

Case weights

This step performs an unsupervised operation that can utilize case weights. To use them, see the documentation in recipes::case_weights and the examples on `tidymodels.org`.

See Also

Other Steps for under-sampling: `step_cluster_centroids()`, `step_cnn()`, `step_enn()`, `step_instance_hardness()`, `step_ncl()`, `step_nearmiss()`, `step_oss()`, `step_tomek()`

Examples

```
library(recipes)
library(modeldata)
data(hpc_data)

hpc_data0 <- hpc_data |>
  select(-protocol, -day)

orig <- count(hpc_data0, class, name = "orig")
orig

up_rec <- recipe(class ~ ., data = hpc_data0) |>
  # Bring the majority levels down to about 1000 each
  # 1000/259 is approx 3.862
  step_downsample(class, under_ratio = 3.862) |>
  prep()

training <- up_rec |>
  bake(new_data = NULL) |>
  count(class, name = "training")
training

# Since `skip` defaults to TRUE, baking the step has no effect
```

```
baked <- up_rec |>
  bake(new_data = hpc_data0) |>
  count(class, name = "baked")
baked

# Note that if the original data contained more rows than the
# target n (= ratio * majority_n), the data are left alone:
orig |>
  left_join(training, by = "class") |>
  left_join(baked, by = "class")

# A named vector gives each level its own target. Here only "VF" is
# sampled down, to about twice the size of the minority level.
recipe(class ~ ., data = hpc_data0) |>
  step_downsample(class, under_ratio = c(VF = 2)) |>
  prep() |>
  bake(new_data = NULL) |>
  count(class)

library(ggplot2)

ggplot(circle_example, aes(x, y, color = class)) +
  geom_point() +
  labs(title = "Without downsample")

recipe(class ~ x + y, data = circle_example) |>
  step_downsample(class) |>
  prep() |>
  bake(new_data = NULL) |>
  ggplot(aes(x, y, color = class)) +
  geom_point() +
  labs(title = "With downsample")
```

`step_enn` *Edited Nearest Neighbors*

Description

`step_enn()` creates a *specification* of a recipe step that removes observations whose class differs from the majority of their nearest neighbors.

Usage

```
step_enn(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  column = NULL,
```

```
  neighbors = 3,
  distance = "euclidean",
  times = 1,
  all_k = FALSE,
  kind_sel = "mode",
  skip = TRUE,
  seed = sample.int(10^5, 1),
  distance_with = recipes::all_predictors(),
  id = rand_id("enn")
)
```

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 which variable is used to sample the data. See recipes::selections for more details. The selection should result in *single factor variable*. For the tidy method, these are not currently used.
role	Not used by this step since no new variables are created.
trained	A logical to indicate if the quantities for preprocessing have been estimated.
column	A character string of the variable name that will be populated (eventually) by the ... selectors.
neighbors	An integer. Number of nearest neighbor that are used to decide whether an observation is removed. Defaults to 3, unlike the over-sampling steps which default to 5.
distance	A character string specifying the distance metric used for nearest neighbor calculations, defaulting to "euclidean". The available metrics fall into three groups. "euclidean", "cosine", and "mahalanobis" use approximate nearest neighbors via the RANN package and scale well to large datasets. "squared_chord", "matusita", "hellinger", and "bhattacharyya" are probability-divergence measures that treat each row as a distribution over the predictors, so they require non-negative values. "hellinger" and "bhattacharyya" further require each row to sum to 1. All four also use the RANN package and scale well to large datasets. "manhattan", "chebyshev", "canberra", "soergel", "lorentzian", "jeffreys", "topsoe", "jensen-shannon", "jensen_difference", "taneja", and "kumar-johnson" compute an exact all-pairs distance matrix. This takes time and memory proportional to the square of the number of observations in a class, so these are best suited to smaller datasets. Everything from "canberra" onwards is a probability divergence requiring non-negative values, is provided by the philentropy package (which must be installed separately), and in the case of "jeffreys", "taneja", and "kumar-johnson" requires strictly positive values, since those divide by individual predictor values. The probability divergences are meaningful for compositional predictors such as proportions or counts normalized per observation, and are generally not appropriate for standardized predictors.

times	A positive integer for the maximum number of times ENN is applied. Defaults to 1 for a single pass. Values greater than 1 repeat the cleaning, stopping early once a pass removes no observations. Use Inf to repeat until convergence (Repeated Edited Nearest Neighbors).
all_k	A logical. When TRUE, ENN is applied with an increasing number of neighbors, from 1 up to neighbors, cleaning the data at each step (All k-Nearest Neighbors). Takes precedence over times. Defaults to FALSE.
kind_sel	A character string. The rule used to decide whether an observation is removed. "mode" (the default) removes an observation when the majority of its neighbors disagree with its class. "all" is stricter and removes an observation unless all of its neighbors share its class.
skip	A logical. Should the step be skipped when the recipe is baked by bake()? While all operations are baked when prep() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations.
seed	An integer that will be used as the seed when applied.
distance_with	A call to a selector function to choose which variables are used for distance calculations. Defaults to recipes::all_predictors(). The variable selected by ... is always excluded from the distance calculations.
id	A character string that is unique to this step to identify it.

Details

Edited Nearest Neighbors (ENN) is a cleaning method. For each observation it finds the neighbors nearest neighbors and, if the class of the observation does not match the majority class among those neighbors, the observation is removed. This tends to remove noisy and borderline observations, which can lead to smoother decision boundaries.

Setting times greater than 1 applies ENN repeatedly, removing more noisy and borderline observations on each pass and stopping early once a pass removes nothing. This corresponds to Repeated Edited Nearest Neighbors (RENN).

Setting all_k = TRUE applies ENN with increasing numbers of neighbors, from 1 up to neighbors, cleaning the data at each step. This corresponds to All k-Nearest Neighbors (AllKNN) and takes precedence over times.

Setting kind_sel = "all" uses a stricter cleaning rule: instead of removing an observation when the majority of its neighbors disagree, it is removed unless every one of its neighbors shares its class. This removes more observations than the default kind_sel = "mode".

All variables selected by distance_with must be numeric with no missing data.

All columns in the data are sampled and returned by recipes::juice() and recipes::bake().

When used in modeling, users should strongly consider using the option skip = TRUE so that the extra sampling is *not* conducted outside of the training set.

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns terms which is the variable used to sample.

Minimum observations

The data must have at least `neighbors + 1` observations for the nearest neighbors to be computed.

Tidying

When you `tidy()` this step, a tibble is returned with columns `terms` and `id`:

terms character, the selectors or variables selected

id character, id of this step

Tuning Parameters

This step has 2 tuning parameters:

- `neighbors`: # Nearest Neighbors (type: integer, default: 3)
- `all_k`: Pruning (type: logical, default: FALSE)

Case weights

The underlying operation does not allow for case weights. Supplying data with a case weights column to this step results in an error.

References

Wilson, D. L. (1972). Asymptotic properties of nearest neighbor rules using edited data. IEEE Transactions on Systems, Man, and Cybernetics, (3), 408-421.

Tomek, I. (1976). An experiment with the edited nearest-neighbor rule. IEEE Transactions on Systems, Man, and Cybernetics, (6), 448-452.

See Also

`enn()` for direct implementation

`step_smote()`, which is commonly composed before `step_enn()` to clean the ambiguous points that over-sampling creates near the class boundary (the equivalent of imbalanced-learn's `SMOTEENN`).

Other Steps for under-sampling: `step_cluster_centroids()`, `step_cnn()`, `step_downsample()`, `step_instance_hardness()`, `step_ncl()`, `step_nearmiss()`, `step_oss()`, `step_tomek()`

Examples

```
library(recipes)
library(modeldata)
data(hpc_data)

hpc_data0 <- hpc_data |>
  select(-protocol, -day)

orig <- count(hpc_data0, class, name = "orig")
orig
```

```
up_rec <- recipe(class ~ ., data = hpc_data0) |>
  step_enn(class) |>
  prep()

training <- up_rec |>
  bake(new_data = NULL) |>
  count(class, name = "training")
training

# Since `skip` defaults to TRUE, baking the step has no effect
baked <- up_rec |>
  bake(new_data = hpc_data0) |>
  count(class, name = "baked")
baked

orig |>
  left_join(training, by = "class") |>
  left_join(baked, by = "class")

library(ggplot2)

ggplot(circle_example, aes(x, y, color = class)) +
  geom_point() +
  labs(title = "Without ENN") +
  xlim(c(1, 15)) +
  ylim(c(1, 15))

recipe(class ~ x + y, data = circle_example) |>
  step_enn(class) |>
  prep() |>
  bake(new_data = NULL) |>
  ggplot(aes(x, y, color = class)) +
  geom_point() +
  labs(title = "With ENN") +
  xlim(c(1, 15)) +
  ylim(c(1, 15))
```

`step_instance_hardness` *Remove hard to classify points*

Description

`step_instance_hardness()` creates a *specification* of a recipe step that removes majority class instances by under-sampling the points that are hardest to classify.

Usage

```
step_instance_hardness(
```

```
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  column = NULL,
  under_ratio = 1,
  neighbors = 5,
  distance = "euclidean",
  skip = TRUE,
  seed = sample.int(10^5, 1),
  distance_with = recipes::all_predictors(),
  id = rand_id("instance_hardness")
)
```

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 which variable is used to sample the data. See recipes::selections for more details. The selection should result in *single factor variable*. For the tidy method, these are not currently used.
role	Not used by this step since no new variables are created.
trained	A logical to indicate if the quantities for preprocessing have been estimated.
column	A character string of the variable name that will be populated (eventually) by the ... selectors.
under_ratio	A numeric value for the ratio of the majority-to-minority frequencies. The default value (1) means that all other levels are sampled down to have the same frequency as the least occurring level. A value of 2 would mean that the majority levels will have (at most) (approximately) twice as many rows than the minority level. A named numeric vector can be used instead to give different levels different targets, for example `c(a = 2, b = 3)`. The names must be levels of the outcome and the values are ratios of the minority level, exactly as in the single-number case. Levels that are not named are left untouched, as are rows with a missing outcome. Because a vector of targets is not a single value, supplying one means this argument can no longer be tuned. See `vignette("ratio", package = "themis")` for more details.
neighbors	An integer. Number of nearest neighbor that are used to generate the new examples of the minority class.
distance	A character string specifying the distance metric used for nearest neighbor calculations, defaulting to `"euclidean"`. The available metrics fall into three groups. `"euclidean"`, `"cosine"`, and `"mahalanobis"` use approximate nearest neighbors via the RANN package and scale well to large datasets. `"squared_chord"`, `"matusita"`, `"hellinger"`, and `"bhattacharyya"` are probability-divergence measures that treat each row as a distribution over the predictors, so

they require non-negative values. "hellinger" and "bhattacharyya" further require each row to sum to 1. All four also use the RANN package and scale well to large datasets.

"manhattan", "chebyshev", "canberra", "soergel", "lorentzian", "jeffreys", "topsoe", "jensen-shannon", "jensen_difference", "taneja", and "kumar-johnson" compute an exact all-pairs distance matrix. This takes time and memory proportional to the square of the number of observations in a class, so these are best suited to smaller datasets. Everything from "canberra" onwards is a probability divergence requiring non-negative values, is provided by the philentropy package (which must be installed separately), and in the case of "jeffreys", "taneja", and "kumar-johnson" requires strictly positive values, since those divide by individual predictor values.

The probability divergences are meaningful for compositional predictors such as proportions or counts normalized per observation, and are generally not appropriate for standardized predictors.

`skip` A logical. Should the step be skipped when the recipe is baked by `bake()`? While all operations are baked when `prep()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

`seed` An integer that will be used as the seed when applied.

`distance_with` A call to a selector function to choose which variables are used for distance calculations. Defaults to `recipes::all_predictors()`. The variable selected by `...` is always excluded from the distance calculations.

`id` A character string that is unique to this step to identify it.

Details

The instance hardness of each observation is estimated using the k-Disagreeing Neighbors measure: the proportion of the nearest neighbors that belong to a different class. Observations that are surrounded by points of a different class are considered hard to classify. For each majority class, the hardest observations are removed until the desired `under_ratio` is reached.

All columns in the data are sampled and returned by `recipes::juice()` and `recipes::bake()`.

All columns selected by `distance_with` must be numeric with no missing data.

When used in modeling, users should strongly consider using the option `skip = TRUE` so that the extra sampling is *not* conducted outside of the training set.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` which is the variable used to sample.

Minimum observations

The data must have at least `neighbors + 1` observations for the nearest neighbors to be computed.

Tidying

When you `tidy()` this step, a tibble is returned with columns `terms` and `id`:

terms character, the selectors or variables selected

id character, id of this step

Tuning Parameters

This step has 2 tuning parameters:

- `under_ratio`: Under-Sampling Ratio (type: double, default: 1)
- `neighbors`: # Nearest Neighbors (type: integer, default: 5)

Case weights

The underlying operation does not allow for case weights. Supplying data with a case weights column to this step results in an error.

References

Smith, M. R., Martinez, T., & Giraud-Carrier, C. (2014). An instance level analysis of data complexity. Machine learning, 95(2), 225-256.

See Also

`instance_hardness()` for direct implementation

Other Steps for under-sampling: `step_cluster_centroids()`, `step_cnn()`, `step_downsample()`, `step_enn()`, `step_ncl()`, `step_nearmiss()`, `step_oss()`, `step_tomek()`

Examples

```
library(recipes)
library(modeldata)
data(hpc_data)

hpc_data0 <- hpc_data |>
  select(-protocol, -day)

orig <- count(hpc_data0, class, name = "orig")
orig

up_rec <- recipe(class ~ ., data = hpc_data0) |>
  # Bring the majority levels down to about 1000 each
  # 1000/259 is approx 3.862
  step_instance_hardness(class, under_ratio = 3.862) |>
  prep()

training <- up_rec |>
  bake(new_data = NULL) |>
  count(class, name = "training")
```

```
training

# Since `skip` defaults to TRUE, baking the step has no effect
baked <- up_rec |>
  bake(new_data = hpc_data0) |>
  count(class, name = "baked")
baked

# Note that if the original data contained fewer rows than the
# target n (= ratio * minority_n), the data are left alone:
orig |>
  left_join(training, by = "class") |>
  left_join(baked, by = "class")

library(ggplot2)

ggplot(circle_example, aes(x, y, color = class)) +
  geom_point() +
  labs(title = "Without instance hardness") +
  xlim(c(1, 15)) +
  ylim(c(1, 15))

recipe(class ~ x + y, data = circle_example) |>
  step_instance_hardness(class) |>
  prep() |>
  bake(new_data = NULL) |>
  ggplot(aes(x, y, color = class)) +
  geom_point() +
  labs(title = "With instance hardness") +
  xlim(c(1, 15)) +
  ylim(c(1, 15))
```

`step_kmeans_smote` *Apply KMeans-SMOTE Algorithm*

Description

`step_kmeans_smote()` creates a *specification* of a recipe step that generates new examples of the minority class, restricting them to the regions of the predictor space where that class is dominant.

Usage

```
step_kmeans_smote(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  column = NULL,
  over_ratio = 1,
```

```
  neighbors = 2,
  num_clusters = NULL,
  cluster_balance_threshold = 1,
  density_exponent = NULL,
  distance = "euclidean",
  indicator_column = NULL,
  skip = TRUE,
  seed = sample.int(10^5, 1),
  id = rand_id("kmeans_smote")
)
```

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 which variable is used to sample the data. See recipes::selections for more details. The selection should result in *single factor variable*. For the tidy method, these are not currently used.
role	Not used by this step since no new variables are created.
trained	A logical to indicate if the quantities for preprocessing have been estimated.
column	A character string of the variable name that will be populated (eventually) by the ... selectors.
over_ratio	A numeric value for the ratio of the minority-to-majority frequencies. The default value (1) means that all other levels are sampled up to have the same frequency as the most occurring level. A value of 0.5 would mean that the minority levels will have (at most) (approximately) half as many rows as the majority level. A named numeric vector can be used instead to give different levels different targets, for example `c(a = 1, b = 0.5)`. The names must be levels of the outcome and the values are ratios of the majority level, exactly as in the single-number case. Levels that are not named are left untouched, as are rows with a missing outcome. Because a vector of targets is not a single value, supplying one means this argument can no longer be tuned. See `vignette("ratio", package = "themis")` for more details.
neighbors	An integer. Number of nearest neighbor that are used to generate the new examples of the minority class. Only observations within the same cluster are considered as neighbors.
num_clusters	An integer, the number of clusters to split the predictor space into, or `NULL` (the default) to use `max(2, floor(sqrt(n / 2)))` where `n` is the number of observations.
cluster_balance_threshold	A number. A cluster is used for over-sampling a class only if it contains at least `cluster_balance_threshold` times as many observations of that class as of all other classes combined. Defaults to `1`, meaning that the class must be at least as common as the rest of the data within the cluster. Smaller values keep more clusters.

`density_exponent`
: A number, the exponent applied to the average pairwise distance when measuring how sparse a cluster is, or `NULL` (the default) to use the number of predictors.

`distance`
: A character string specifying the distance metric used for nearest neighbor calculations, defaulting to `"euclidean"`. The available metrics fall into three groups.

`"euclidean"`, `"cosine"`, and `"mahalanobis"` use approximate nearest neighbors via the RANN package and scale well to large datasets.

`"squared_chord"`, `"matusita"`, `"hellinger"`, and `"bhattacharyya"` are probability-divergence measures that treat each row as a distribution over the predictors, so they require non-negative values. `"hellinger"` and `"bhattacharyya"` further require each row to sum to 1. All four also use the RANN package and scale well to large datasets.

`"manhattan"`, `"chebyshev"`, `"canberra"`, `"soergel"`, `"lorentzian"`, `"jeffreys"`, `"topsoe"`, `"jensen-shannon"`, `"jensen_difference"`, `"taneja"`, and `"kumar-johnson"` compute an exact all-pairs distance matrix. This takes time and memory proportional to the square of the number of observations in a class, so these are best suited to smaller datasets. Everything from `"canberra"` onwards is a probability divergence requiring non-negative values, is provided by the philentropy package (which must be installed separately), and in the case of `"jeffreys"`, `"taneja"`, and `"kumar-johnson"` requires strictly positive values, since those divide by individual predictor values.

The probability divergences are meaningful for compositional predictors such as proportions or counts normalized per observation, and are generally not appropriate for standardized predictors.

`indicator_column`
: A single string or `NULL` (the default). If a string is given, a logical column with that name is added to the output, marking rows added by the step (`TRUE`) vs rows from the original data (`FALSE`).

`skip`
: A logical. Should the step be skipped when the recipe is baked by `bake()`? While all operations are baked when `prep()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

`seed`
: An integer that will be used as the seed when applied.

`id`
: A character string that is unique to this step to identify it.

Details

KMeans-SMOTE combines k-means clustering with SMOTE to avoid generating synthetic points in regions where the minority class is not actually present. It works in three stages:

1. The predictor space of the whole data set is clustered with `stats::kmeans()` into `num_clusters` clusters.
2. Each cluster is either kept or filtered out. A cluster is kept for a given minority class if the ratio of that class's observations to all other observations in the cluster is at least `cluster_balance_threshold`, and if the cluster contains more than `neighbors` observations of that class so that interpolation is possible.

3. The synthetic points are distributed over the kept clusters according to how sparse each cluster is. The sparsity of a cluster is `mean_pairwise_distance ^ density_exponent / n`, where `n` is the number of minority observations in the cluster. Sparser clusters receive more points. Within each cluster the points are then generated by ordinary SMOTE interpolation, using only that cluster's observations as neighbors.

Filtering on cluster balance keeps synthetic points away from majority-dominated regions, and the density weighting counteracts within-class imbalance rather than only between-class imbalance.

The clustering is done once on all observations, so in a multi-class problem every minority class shares the same clusters; only the filtering and weighting are done per class. Note also that k-means always uses Euclidean distance. The `distance` argument affects the nearest-neighbor search used for interpolation and the sparsity calculation, not the clustering itself.

All columns in the data are sampled and returned by `recipes::juice()` and `recipes::bake()`.

All columns used in this step must be numeric with no missing data.

When used in modeling, users should strongly consider using the option `skip = TRUE` so that the extra sampling is *not* conducted outside of the training set.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` which is the variable used to sample.

Minimum observations

Each minority class must have more than `neighbors` observations in at least one kept cluster. If no cluster qualifies, an error is thrown suggesting which arguments to loosen.

Tidying

When you `tidy()` this step, a tibble is returned with columns `terms` and `id`:

terms character, the selectors or variables selected

id character, id of this step

Tuning Parameters

This step has 3 tuning parameters:

- `over_ratio`: Over-Sampling Ratio (type: double, default: 1)
- `neighbors`: # Nearest Neighbors (type: integer, default: 2)
- `num_clusters`: # Clusters (type: integer, default: NULL)

Case weights

The underlying operation does not allow for case weights. Supplying data with a case weights column to this step results in an error.

References

Douzas, G., Bacao, F., and Last, F. (2018). Improving imbalanced learning through a heuristic oversampling method based on k-means and SMOTE. Information Sciences, 465:1-20.

See Also

kmeans_smote() for direct implementation

step_smote() for the same interpolation without the clustering step

Other Steps for over-sampling: step_adasyn(), step_bsmote(), step_rose(), step_smogn(), step_smote(), step_smoten(), step_smotenc(), step_svmsmote(), step_upsample()

Examples

```
library(recipes)
library(modeldata)
data(hpc_data)

hpc_data0 <- hpc_data |>
  select(-protocol, -day)

orig <- count(hpc_data0, class, name = "orig")
orig

up_rec <- recipe(class ~ ., data = hpc_data0) |>
  # Bring the minority levels up to about 1000 each
  # 1000/2211 is approx 0.4523
  step_kmeans_smote(class, over_ratio = 0.4523) |>
  prep()

training <- up_rec |>
  bake(new_data = NULL) |>
  count(class, name = "training")
training

# Since `skip` defaults to TRUE, baking the step has no effect
baked <- up_rec |>
  bake(new_data = hpc_data0) |>
  count(class, name = "baked")
baked

library(ggplot2)

ggplot(circle_example, aes(x, y, color = class)) +
  geom_point() +
  labs(title = "Without KMeans-SMOTE")

recipe(class ~ x + y, data = circle_example) |>
  step_kmeans_smote(class) |>
  prep() |>
  bake(new_data = NULL) |>
  ggplot(aes(x, y, color = class)) +
```

```
  geom_point() +
  labs(title = "With KMeans-SMOTE")
```

`step_ncl` *Neighborhood Cleaning Rule*

Description

`step_ncl()` creates a *specification* of a recipe step that removes majority class observations that are noisy or that pollute the neighborhood of minority class observations.

Usage

```
step_ncl(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  column = NULL,
  neighbors = 3,
  distance = "euclidean",
  threshold_clean = 0.5,
  skip = TRUE,
  seed = sample.int(10^5, 1),
  distance_with = recipes::all_predictors(),
  id = rand_id("ncl")
)
```

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 which variable is used to sample the data. See recipes::selections for more details. The selection should result in *single factor variable*. For the `tidy` method, these are not currently used.
`role`	Not used by this step since no new variables are created.
`trained`	A logical to indicate if the quantities for preprocessing have been estimated.
`column`	A character string of the variable name that will be populated (eventually) by the `...` selectors.
`neighbors`	An integer. Number of nearest neighbor that are used to decide whether an observation is removed. Defaults to `3`, unlike the over-sampling steps which default to `5`.

distance
: A character string specifying the distance metric used for nearest neighbor calculations, defaulting to "euclidean". The available metrics fall into three groups.

  "euclidean", "cosine", and "mahalanobis" use approximate nearest neighbors via the RANN package and scale well to large datasets.

  "squared_chord", "matusita", "hellinger", and "bhattacharyya" are probability-divergence measures that treat each row as a distribution over the predictors, so they require non-negative values. "hellinger" and "bhattacharyya" further require each row to sum to 1. All four also use the RANN package and scale well to large datasets.

  "manhattan", "chebyshev", "canberra", "soergel", "lorentzian", "jeffreys", "topsoe", "jensen-shannon", "jensen_difference", "taneja", and "kumar-johnson" compute an exact all-pairs distance matrix. This takes time and memory proportional to the square of the number of observations in a class, so these are best suited to smaller datasets. Everything from "canberra" onwards is a probability divergence requiring non-negative values, is provided by the philentropy package (which must be installed separately), and in the case of "jeffreys", "taneja", and "kumar-johnson" requires strictly positive values, since those divide by individual predictor values.

  The probability divergences are meaningful for compositional predictors such as proportions or counts normalized per observation, and are generally not appropriate for standardized predictors.

threshold_clean
: A numeric. Majority classes are only cleaned around minority class observations when their size is greater than threshold_clean times the size of the minority class. Defaults to 0.5.

skip
: A logical. Should the step be skipped when the recipe is baked by bake()? While all operations are baked when prep() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations.

seed
: An integer that will be used as the seed when applied.

distance_with
: A call to a selector function to choose which variables are used for distance calculations. Defaults to recipes::all_predictors(). The variable selected by ... is always excluded from the distance calculations.

id
: A character string that is unique to this step to identify it.

Details

The Neighborhood Cleaning Rule (NCL) is a cleaning method that combines two passes over the data. First, it applies the Edited Nearest Neighbors rule, removing majority class observations whose class differs from the majority of their neighbors nearest neighbors. Second, for each minority class observation that is itself misclassified by its neighbors, the majority class observations among those neighbors are removed. Compared to Edited Nearest Neighbors, this focuses the cleaning on the neighborhoods of minority class observations.

The smallest class is treated as the minority class. Only majority classes larger than threshold_clean times the size of the minority class are cleaned in the second pass.

All variables selected by `distance_with` must be numeric with no missing data.

All columns in the data are sampled and returned by `recipes::juice()` and `recipes::bake()`.

When used in modeling, users should strongly consider using the option `skip = TRUE` so that the extra sampling is *not* conducted outside of the training set.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` which is the variable used to sample.

Minimum observations

The data must have at least `neighbors + 1` observations for the nearest neighbors to be computed.

Tidying

When you `tidy()` this step, a tibble is returned with columns `terms` and `id`:

terms character, the selectors or variables selected

id character, id of this step

Tuning Parameters

This step has 2 tuning parameters:

- `neighbors`: # Nearest Neighbors (type: integer, default: 3)
- `threshold_clean`: Threshold (type: double, default: 0.5)

Case weights

The underlying operation does not allow for case weights. Supplying data with a case weights column to this step results in an error.

References

Laurikkala, J. (2001). Improving identification of difficult small classes by balancing class distribution. In Conference on Artificial Intelligence in Medicine in Europe (pp. 63-66). Springer.

See Also

`ncl()` for direct implementation

Other Steps for under-sampling: `step_cluster_centroids()`, `step_cnn()`, `step_downsample()`, `step_enn()`, `step_instance_hardness()`, `step_nearmiss()`, `step_oss()`, `step_tomek()`

Examples

```
library(recipes)
library(modeldata)
data(hpc_data)

hpc_data0 <- hpc_data |>
  select(-protocol, -day)

orig <- count(hpc_data0, class, name = "orig")
orig

up_rec <- recipe(class ~ ., data = hpc_data0) |>
  step_ncl(class) |>
  prep()

training <- up_rec |>
  bake(new_data = NULL) |>
  count(class, name = "training")
training

# Since `skip` defaults to TRUE, baking the step has no effect
baked <- up_rec |>
  bake(new_data = hpc_data0) |>
  count(class, name = "baked")
baked

orig |>
  left_join(training, by = "class") |>
  left_join(baked, by = "class")

library(ggplot2)

ggplot(circle_example, aes(x, y, color = class)) +
  geom_point() +
  labs(title = "Without NCL") +
  xlim(c(1, 15)) +
  ylim(c(1, 15))

recipe(class ~ x + y, data = circle_example) |>
  step_ncl(class) |>
  prep() |>
  bake(new_data = NULL) |>
  ggplot(aes(x, y, color = class)) +
  geom_point() +
  labs(title = "With NCL") +
  xlim(c(1, 15)) +
  ylim(c(1, 15))
```

`step_nearmiss` *Remove Points Near Other Classes*

Description

`step_nearmiss()` creates a *specification* of a recipe step that removes majority class instances by undersampling points in the majority class based on their distance to points in the minority class.

Usage

```
step_nearmiss(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  column = NULL,
  under_ratio = 1,
  neighbors = 5,
  distance = "euclidean",
  version = 1,
  n_neighbors_ver3 = 3,
  skip = TRUE,
  seed = sample.int(10^5, 1),
  distance_with = recipes::all_predictors(),
  id = rand_id("nearmiss")
)
```

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 which variable is used to sample the data. See recipes::selections for more details. The selection should result in *single factor variable*. For the `tidy` method, these are not currently used.
`role`	Not used by this step since no new variables are created.
`trained`	A logical to indicate if the quantities for preprocessing have been estimated.
`column`	A character string of the variable name that will be populated (eventually) by the `...` selectors.
`under_ratio`	A numeric value for the ratio of the majority-to-minority frequencies. The default value (1) means that all other levels are sampled down to have the same frequency as the least occurring level. A value of 2 would mean that the majority levels will have (at most) (approximately) twice as many rows than the minority level. A named numeric vector can be used instead to give different levels different targets, for example `c(a = 2, b = 3)`. The names must be levels of the outcome and the values are ratios of the minority level, exactly as in the single-number case. Levels that are not named are left untouched, as are rows with a missing outcome. Because a vector of targets is not a single value, supplying one means this argument can no longer be tuned. See `vignette("ratio", package = "themis")` for more details.

`neighbors` An integer. Number of nearest neighbor that are used to generate the new examples of the minority class.

`distance` A character string specifying the distance metric used for nearest neighbor calculations, defaulting to `"euclidean"`. The available metrics fall into three groups.

`"euclidean"`, `"cosine"`, and `"mahalanobis"` use approximate nearest neighbors via the RANN package and scale well to large datasets.

`"squared_chord"`, `"matusita"`, `"hellinger"`, and `"bhattacharyya"` are probability-divergence measures that treat each row as a distribution over the predictors, so they require non-negative values. `"hellinger"` and `"bhattacharyya"` further require each row to sum to 1. All four also use the RANN package and scale well to large datasets.

`"manhattan"`, `"chebyshev"`, `"canberra"`, `"soergel"`, `"lorentzian"`, `"jeffreys"`, `"topsoe"`, `"jensen-shannon"`, `"jensen_difference"`, `"taneja"`, and `"kumar-johnson"` compute an exact all-pairs distance matrix. This takes time and memory proportional to the square of the number of observations in a class, so these are best suited to smaller datasets. Everything from `"canberra"` onwards is a probability divergence requiring non-negative values, is provided by the philentropy package (which must be installed separately), and in the case of `"jeffreys"`, `"taneja"`, and `"kumar-johnson"` requires strictly positive values, since those divide by individual predictor values.

The probability divergences are meaningful for compositional predictors such as proportions or counts normalized per observation, and are generally not appropriate for standardized predictors.

`version` An integer. Which of the three NearMiss variants to use, `1`, `2`, or `3`. Defaults to `1`. See the details section.

`n_neighbors_ver3`
An integer. The number of nearest neighbors used to build the candidate pool of the NearMiss-3 variant. Only used when `version = 3`. Defaults to `3`.

`skip` A logical. Should the step be skipped when the recipe is baked by `bake()`? While all operations are baked when `prep()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

`seed` An integer that will be used as the seed when applied.

`distance_with` A call to a selector function to choose which variables are used for distance calculations. Defaults to `recipes::all_predictors()`. The variable selected by `...` is always excluded from the distance calculations.

`id` A character string that is unique to this step to identify it.

Details

The `version` argument selects between the three NearMiss variants:

`version = 1` Retains the points from the majority class which have the smallest mean distance to their nearest points in the minority class.

`version = 2` Retains the points from the majority class which have the smallest mean distance to their farthest points in the minority class.

`version = 3` Works in two stages. First, the `n_neighbors_ver3` nearest majority class neighbors of each minority class point form a candidate pool, and all other majority class points are removed. Then the points of that pool which have the largest mean distance to their nearest minority class points are retained.

Since the size of the NearMiss-3 candidate pool is governed by `n_neighbors_ver3` rather than by `under_ratio`, the pool can be smaller than the target set by `under_ratio`. The whole pool is then retained and the target is not reached.

With more than two classes, the mean distance is computed to the nearest points across all other classes, not only the minority class. This differs from imbalanced-learn, which measures distance to the minority class only. The binary case, the primary intended use, is unaffected.

All columns in the data are sampled and returned by `recipes::juice()` and `recipes::bake()`.

All columns selected by `distance_with` must be numeric with no missing data.

When used in modeling, users should strongly consider using the option `skip = TRUE` so that the extra sampling is *not* conducted outside of the training set.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` which is the variable used to sample.

Minimum observations

Each minority class must have at least `neighbors + 1` observations to perform the NearMiss algorithm.

Tidying

When you `tidy()` this step, a tibble is returned with columns `terms` and `id`:

terms character, the selectors or variables selected

id character, id of this step

Tuning Parameters

This step has 2 tuning parameters:

- `under_ratio`: Under-Sampling Ratio (type: double, default: 1)
- `neighbors`: # Nearest Neighbors (type: integer, default: 5)

Case weights

The underlying operation does not allow for case weights. Supplying data with a case weights column to this step results in an error.

References

Inderjeet Mani and I Zhang. knn approach to unbalanced data distributions: a case study involving information extraction. In Proceedings of workshop on learning from imbalanced datasets, 2003.

See Also

`nearmiss()` for direct implementation

Other Steps for under-sampling: `step_cluster_centroids()`, `step_cnn()`, `step_downsample()`, `step_enn()`, `step_instance_hardness()`, `step_ncl()`, `step_oss()`, `step_tomek()`

Examples

```
library(recipes)
library(modeldata)
data(hpc_data)

hpc_data0 <- hpc_data |>
  select(-protocol, -day)

orig <- count(hpc_data0, class, name = "orig")
orig

up_rec <- recipe(class ~ ., data = hpc_data0) |>
  # Bring the majority levels down to about 1000 each
  # 1000/259 is approx 3.862
  step_nearmiss(class, under_ratio = 3.862) |>
  prep()

training <- up_rec |>
  bake(new_data = NULL) |>
  count(class, name = "training")
training

# Since `skip` defaults to TRUE, baking the step has no effect
baked <- up_rec |>
  bake(new_data = hpc_data0) |>
  count(class, name = "baked")
baked

# Note that if the original data contained fewer rows than the
# target n (= ratio * minority_n), the data are left alone:
orig |>
  left_join(training, by = "class") |>
  left_join(baked, by = "class")

library(ggplot2)

ggplot(circle_example, aes(x, y, color = class)) +
  geom_point() +
  labs(title = "Without NEARMISS") +
  xlim(c(1, 15)) +
```

```
  ylim(c(1, 15))

recipe(class ~ x + y, data = circle_example) |>
  step_nearmiss(class) |>
  prep() |>
  bake(new_data = NULL) |>
  ggplot(aes(x, y, color = class)) +
  geom_point() +
  labs(title = "With NEARMISS") +
  xlim(c(1, 15)) +
  ylim(c(1, 15))
```

`step_oss` *One-Sided Selection*

Description

`step_oss()` creates a *specification* of a recipe step that removes majority class observations using One-Sided Selection, combining Condensed Nearest Neighbors and Tomek's links.

Usage

```
step_oss(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  column = NULL,
  distance = "euclidean",
  skip = TRUE,
  seed = sample.int(10^5, 1),
  distance_with = recipes::all_predictors(),
  id = rand_id("oss")
)
```

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 which variable is used to sample the data. See recipes::selections for more details. The selection should result in *single factor variable*. For the `tidy` method, these are not currently used.
`role`	Not used by this step since no new variables are created.
`trained`	A logical to indicate if the quantities for preprocessing have been estimated.
`column`	A character string of the variable name that will be populated (eventually) by the `...` selectors.

distance
: A character string specifying the distance metric used for nearest neighbor calculations, defaulting to "euclidean". The available metrics fall into three groups.

  "euclidean", "cosine", and "mahalanobis" use approximate nearest neighbors via the RANN package and scale well to large datasets.

  "squared_chord", "matusita", "hellinger", and "bhattacharyya" are probability-divergence measures that treat each row as a distribution over the predictors, so they require non-negative values. "hellinger" and "bhattacharyya" further require each row to sum to 1. All four also use the RANN package and scale well to large datasets.

  "manhattan", "chebyshev", "canberra", "soergel", "lorentzian", "jeffreys", "topsoe", "jensen-shannon", "jensen_difference", "taneja", and "kumar-johnson" compute an exact all-pairs distance matrix. This takes time and memory proportional to the square of the number of observations in a class, so these are best suited to smaller datasets. Everything from "canberra" onwards is a probability divergence requiring non-negative values, is provided by the philentropy package (which must be installed separately), and in the case of "jeffreys", "taneja", and "kumar-johnson" requires strictly positive values, since those divide by individual predictor values.

  The probability divergences are meaningful for compositional predictors such as proportions or counts normalized per observation, and are generally not appropriate for standardized predictors.

skip
: A logical. Should the step be skipped when the recipe is baked by bake()? While all operations are baked when prep() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations.

seed
: An integer that will be used as the seed when applied.

distance_with
: A call to a selector function to choose which variables are used for distance calculations. Defaults to recipes::all_predictors(). The variable selected by ... is always excluded from the distance calculations.

id
: A character string that is unique to this step to identify it.

Details

One-Sided Selection (OSS) is an under-sampling method that combines two cleaning techniques. It first applies Condensed Nearest Neighbors (CNN) to reduce the majority classes to a consistent subset that correctly classifies the data using a 1-nearest-neighbor rule, discarding redundant interior observations. It then applies Tomek's links to the remaining observations, removing the majority class observations that form Tomek links with minority class observations, cleaning the decision boundary.

The smallest class is treated as the minority class and is always kept. Because the CNN step relies on a random seed observation and a random scan order, results depend on the random seed.

With more than two classes, the Tomek's links step removes both members of a majority-majority link, not only links between a majority and the minority class. The binary case, the primary intended use, is unaffected.

All variables selected by `distance_with` must be numeric with no missing data.

All columns in the data are sampled and returned by `recipes::juice()` and `recipes::bake()`.

When used in modeling, users should strongly consider using the option `skip = TRUE` so that the extra sampling is *not* conducted outside of the training set.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` which is the variable used to sample.

Tidying

When you `tidy()` this step, a tibble is returned with columns `terms` and `id`:

terms character, the selectors or variables selected

id character, id of this step

Case weights

The underlying operation does not allow for case weights. Supplying data with a case weights column to this step results in an error.

References

Kubat, M., & Matwin, S. (1997). Addressing the curse of imbalanced training sets: one-sided selection. In ICML (Vol. 97, pp. 179-186).

See Also

`oss()` for direct implementation

Other Steps for under-sampling: `step_cluster_centroids()`, `step_cnn()`, `step_downsample()`, `step_enn()`, `step_instance_hardness()`, `step_ncl()`, `step_nearmiss()`, `step_tomek()`

Examples

```
library(recipes)
library(modeldata)
data(hpc_data)

hpc_data0 <- hpc_data |>
  select(-protocol, -day)

orig <- count(hpc_data0, class, name = "orig")
orig

up_rec <- recipe(class ~ ., data = hpc_data0) |>
  step_oss(class) |>
  prep()

training <- up_rec |>
```

```
  bake(new_data = NULL) |>
  count(class, name = "training")
training

# Since `skip` defaults to TRUE, baking the step has no effect
baked <- up_rec |>
  bake(new_data = hpc_data0) |>
  count(class, name = "baked")
baked

orig |>
  left_join(training, by = "class") |>
  left_join(baked, by = "class")

library(ggplot2)

ggplot(circle_example, aes(x, y, color = class)) +
  geom_point() +
  labs(title = "Without OSS") +
  xlim(c(1, 15)) +
  ylim(c(1, 15))

recipe(class ~ x + y, data = circle_example) |>
  step_oss(class) |>
  prep() |>
  bake(new_data = NULL) |>
  ggplot(aes(x, y, color = class)) +
  geom_point() +
  labs(title = "With OSS") +
  xlim(c(1, 15)) +
  ylim(c(1, 15))
```

`step_rose` *Apply ROSE Algorithm*

Description

`step_rose()` creates a *specification* of a recipe step that generates samples of synthetic data by enlarging the feature space of minority and majority class examples. Using `ROSE::ROSE()`.

Usage

```
step_rose(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  column = NULL,
  over_ratio = 1,
```

```
  minority_prop = 0.5,
  minority_smoothness = 1,
  majority_smoothness = 1,
  indicator_column = NULL,
  skip = TRUE,
  seed = sample.int(10^5, 1),
  id = rand_id("rose")
)
```

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 which variable is used to sample the data. See recipes::selections for more details. The selection should result in *single factor variable*. For the tidy method, these are not currently used.
role	Not used by this step since no new variables are created.
trained	A logical to indicate if the quantities for preprocessing have been estimated.
column	A character string of the variable name that will be populated (eventually) by the ... selectors.
over_ratio	A numeric value for the total size of the synthetic data relative to twice the size of the majority class. Unlike the other over-sampling steps this is not a per-class target, so a named vector of ratios is not accepted here.
minority_prop	A numeric value between 0 and 1 for the proportion of synthetic observations from the minority class. Defaults to 0.5, which generates an equal split of minority and majority synthetic observations. This parameter controls the class balance *within* the synthetic data, while over_ratio controls the *total size* of the synthetic data.
minority_smoothness	A numeric. Shrink factor to be multiplied by the smoothing parameters to estimate the conditional kernel density of the minority class. Defaults to 1.
majority_smoothness	A numeric. Shrink factor to be multiplied by the smoothing parameters to estimate the conditional kernel density of the majority class. Defaults to 1.
indicator_column	A single string or NULL (the default). If a string is given, a logical column with that name is added to the output. Because ROSE generates a fully synthetic dataset, all rows are marked TRUE.
skip	A logical. Should the step be skipped when the recipe is baked by bake()? While all operations are baked when prep() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations.
seed	An integer that will be used as the seed when applied.
id	A character string that is unique to this step to identify it.

Details

The factor variable used to balance around must only have 2 levels.

The ROSE algorithm works by selecting an observation belonging to class k and generating new examples in its neighborhood, which is determined by a smoothing matrix H_k. Smaller values of `minority_smoothness` and `majority_smoothness` shrink the entries of H_k, producing tighter neighborhoods. This is a cautious choice when there is a concern that excessively large neighborhoods could blur the boundaries between classes.

All columns in the data are sampled and returned by `recipes::juice()` and `recipes::bake()`.

When used in modeling, users should strongly consider using the option `skip = TRUE` so that the extra sampling is *not* conducted outside of the training set.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` which is the variable used to sample.

Tidying

When you `tidy()` this step, a tibble is returned with columns `terms` and `id`:

terms character, the selectors or variables selected

id character, id of this step

Tuning Parameters

This step has 1 tuning parameters:

- `over_ratio`: Over-Sampling Ratio (type: double, default: 1)

Case weights

The underlying operation does not allow for case weights. Supplying data with a case weights column to this step results in an error.

References

Lunardon, N., Menardi, G., and Torelli, N. (2014). ROSE: a Package for Binary Imbalanced Learning. R Journal, 6:79–89.

Menardi, G. and Torelli, N. (2014). Training and assessing classification rules with imbalanced data. Data Mining and Knowledge Discovery, 28:92–122.

See Also

`rose()` for direct implementation

Other Steps for over-sampling: `step_adasyn()`, `step_bsmote()`, `step_kmeans_smote()`, `step_smogn()`, `step_smote()`, `step_smoten()`, `step_smotenc()`, `step_svmsmote()`, `step_upsample()`

Examples

```
library(recipes)
library(modeldata)
data(hpc_data)

hpc_data0 <- hpc_data |>
  mutate(class = factor(class == "VF", labels = c("not VF", "VF"))) |>
  select(-protocol, -day)

orig <- count(hpc_data0, class, name = "orig")
orig

up_rec <- recipe(class ~ ., data = hpc_data0) |>
  step_rose(class) |>
  prep()

training <- up_rec |>
  bake(new_data = NULL) |>
  count(class, name = "training")
training

# Since `skip` defaults to TRUE, baking the step has no effect
baked <- up_rec |>
  bake(new_data = hpc_data0) |>
  count(class, name = "baked")
baked

orig |>
  left_join(training, by = "class") |>
  left_join(baked, by = "class")

library(ggplot2)

ggplot(circle_example, aes(x, y, color = class)) +
  geom_point() +
  labs(title = "Without ROSE")

recipe(class ~ x + y, data = circle_example) |>
  step_rose(class) |>
  prep() |>
  bake(new_data = NULL) |>
  ggplot(aes(x, y, color = class)) +
  geom_point() +
  labs(title = "With ROSE")
```

`step_smogn` *Apply SMOGN Algorithm*

Description

`step_smogn()` creates a *specification* of a recipe step that generates new examples for imbalanced regression problems using SMOGN.

Usage

```
step_smogn(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  column = NULL,
  threshold = 0.5,
  relevance = NULL,
  neighbors = 5,
  perturbation = 0.02,
  distance = "euclidean",
  indicator_column = NULL,
  skip = TRUE,
  seed = sample.int(10^5, 1),
  id = rand_id("smogn")
)
```

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 which variable is used to sample the data. See recipes::selections for more details. The selection should result in *single numeric variable*. For the `tidy` method, these are not currently used.
`role`	Not used by this step since no new variables are created.
`trained`	A logical to indicate if the quantities for preprocessing have been estimated.
`column`	A character string of the variable name that will be populated (eventually) by the `...` selectors.
`threshold`	A number between 0 and 1. Outcome values with a relevance at or above this value are treated as rare and over-sampled. Defaults to `0.5`.
`relevance`	A matrix of relevance control points, or `NULL` (default). When `NULL`, relevance is derived automatically from the boxplot extremes of the outcome. When supplied, the first column gives outcome values and the second column their relevance in `[0, 1]`.
`neighbors`	An integer. Number of nearest neighbor that are used to generate the new examples of the rare values.
`perturbation`	A number. The magnitude of the Gaussian noise added when generating synthetic examples in unsafe regions. Defaults to `0.02`.

distance
: A character string specifying the distance metric used for nearest neighbor calculations, defaulting to "euclidean". The available metrics fall into three groups.

  "euclidean", "cosine", and "mahalanobis" use approximate nearest neighbors via the RANN package and scale well to large datasets.

  "squared_chord", "matusita", "hellinger", and "bhattacharyya" are probability-divergence measures that treat each row as a distribution over the predictors, so they require non-negative values. "hellinger" and "bhattacharyya" further require each row to sum to 1. All four also use the RANN package and scale well to large datasets.

  "manhattan", "chebyshev", "canberra", "soergel", "lorentzian", "jeffreys", "topsoe", "jensen-shannon", "jensen_difference", "taneja", and "kumar-johnson" compute an exact all-pairs distance matrix. This takes time and memory proportional to the square of the number of observations in a class, so these are best suited to smaller datasets. Everything from "canberra" onwards is a probability divergence requiring non-negative values, is provided by the philentropy package (which must be installed separately), and in the case of "jeffreys", "taneja", and "kumar-johnson" requires strictly positive values, since those divide by individual predictor values.

  The probability divergences are meaningful for compositional predictors such as proportions or counts normalized per observation, and are generally not appropriate for standardized predictors.

indicator_column
: A single string or NULL (the default). If a string is given, a logical column with that name is added to the output, marking rows added by the step (TRUE) vs rows from the original data (FALSE).

skip
: A logical. Should the step be skipped when the recipe is baked by bake()? While all operations are baked when prep() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations.

seed
: An integer that will be used as the seed when applied.

id
: A character string that is unique to this step to identify it.

Details

SMOGN is a pre-processing approach for imbalanced regression. A relevance function assigns each outcome value a relevance score, and values with a relevance at or above threshold are treated as rare. The data is split into contiguous bins of rare and common outcome values. Common bins are under-sampled and rare bins are over-sampled toward a balanced size. New rare examples are generated either by interpolating between an example and a nearby neighbor (when they are close enough to be considered safe) or by perturbing the example with Gaussian noise (when they are not), where the amount of noise is controlled by perturbation.

By default relevance is derived automatically from the boxplot extremes of the outcome, giving the median a relevance of 0 and the extreme values a relevance of 1. A matrix of relevance control points can instead be supplied through relevance, with the first column giving outcome values and the second column their relevance.

All columns in the data are sampled and returned by `recipes::juice()` and `recipes::bake()`.

All columns used in this step must be numeric with no missing data.

When used in modeling, users should strongly consider using the option `skip = TRUE` so that the extra sampling is *not* conducted outside of the training set.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` which is the variable used to sample.

Tidying

When you `tidy()` this step, a tibble is returned with columns `terms` and `id`:

terms character, the selectors or variables selected

id character, id of this step

Tuning Parameters

This step has 2 tuning parameters:

- `neighbors`: # Nearest Neighbors (type: integer, default: 5)
- `threshold`: Threshold (type: double, default: 0.5)

Case weights

The underlying operation does not allow for case weights. Supplying data with a case weights column to this step results in an error.

References

Branco, P., Torgo, L., and Ribeiro, R. P. (2017). SMOGN: a pre-processing approach for imbalanced regression. Proceedings of Machine Learning Research, 74:36-50.

See Also

`smogn()` for direct implementation

Other Steps for over-sampling: `step_adasyn()`, `step_bsmote()`, `step_kmeans_smote()`, `step_rose()`, `step_smote()`, `step_smoten()`, `step_smotenc()`, `step_svmsmote()`, `step_upsample()`

Examples

```
library(recipes)
library(ggplot2)

ggplot(circle_example, aes(x)) +
  geom_histogram(bins = 30) +
  labs(title = "Without SMOGN")

recipe(y ~ x, data = circle_example) |>
```

```
  step_smogn(y) |>
  prep() |>
  bake(new_data = NULL) |>
  ggplot(aes(y)) +
  geom_histogram(bins = 30) +
  labs(title = "With SMOGN")
```

`step_smote` *Apply SMOTE Algorithm*

Description

`step_smote()` creates a *specification* of a recipe step that generate new examples of the minority class using nearest neighbors of these cases.

Usage

```
step_smote(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  column = NULL,
  over_ratio = 1,
  neighbors = 5,
  distance = "euclidean",
  indicator_column = NULL,
  skip = TRUE,
  seed = sample.int(10^5, 1),
  id = rand_id("smote")
)
```

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 which variable is used to sample the data. See recipes::selections for more details. The selection should result in *single factor variable*. For the `tidy` method, these are not currently used.
`role`	Not used by this step since no new variables are created.
`trained`	A logical to indicate if the quantities for preprocessing have been estimated.
`column`	A character string of the variable name that will be populated (eventually) by the `...` selectors.

over_ratio
: A numeric value for the ratio of the minority-to-majority frequencies. The default value (1) means that all other levels are sampled up to have the same frequency as the most occurring level. A value of 0.5 would mean that the minority levels will have (at most) (approximately) half as many rows as the majority level.

  A named numeric vector can be used instead to give different levels different targets, for example `c(a = 1, b = 0.5)`. The names must be levels of the outcome and the values are ratios of the majority level, exactly as in the single-number case. Levels that are not named are left untouched, as are rows with a missing outcome. Because a vector of targets is not a single value, supplying one means this argument can no longer be tuned. See `vignette("ratio", package = "themis")` for more details.

neighbors
: An integer. Number of nearest neighbor that are used to generate the new examples of the minority class.

distance
: A character string specifying the distance metric used for nearest neighbor calculations, defaulting to `"euclidean"`. The available metrics fall into three groups.

  `"euclidean"`, `"cosine"`, and `"mahalanobis"` use approximate nearest neighbors via the RANN package and scale well to large datasets.

  `"squared_chord"`, `"matusita"`, `"hellinger"`, and `"bhattacharyya"` are probability-divergence measures that treat each row as a distribution over the predictors, so they require non-negative values. `"hellinger"` and `"bhattacharyya"` further require each row to sum to 1. All four also use the RANN package and scale well to large datasets.

  `"manhattan"`, `"chebyshev"`, `"canberra"`, `"soergel"`, `"lorentzian"`, `"jeffreys"`, `"topsoe"`, `"jensen-shannon"`, `"jensen_difference"`, `"taneja"`, and `"kumar-johnson"` compute an exact all-pairs distance matrix. This takes time and memory proportional to the square of the number of observations in a class, so these are best suited to smaller datasets. Everything from `"canberra"` onwards is a probability divergence requiring non-negative values, is provided by the philentropy package (which must be installed separately), and in the case of `"jeffreys"`, `"taneja"`, and `"kumar-johnson"` requires strictly positive values, since those divide by individual predictor values.

  The probability divergences are meaningful for compositional predictors such as proportions or counts normalized per observation, and are generally not appropriate for standardized predictors.

indicator_column
: A single string or `NULL` (the default). If a string is given, a logical column with that name is added to the output, marking rows added by the step (`TRUE`) vs rows from the original data (`FALSE`).

skip
: A logical. Should the step be skipped when the recipe is baked by `bake()`? While all operations are baked when `prep()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

seed
: An integer that will be used as the seed when applied.

id
: A character string that is unique to this step to identify it.

Details

SMOTE generates new examples of the minority class using nearest neighbors of these cases. For each existing minority class example, new examples are created by interpolating between the example and its nearest neighbors. The number of nearest neighbors used is controlled by the number of neighbors argument (k in `smote()`, `neighbors` in `step_smote()`), and the number of new examples generated is controlled by `over_ratio`.

All columns in the data are sampled and returned by `recipes::juice()` and `recipes::bake()`.

All columns used in this step must be numeric with no missing data.

When used in modeling, users should strongly consider using the option `skip = TRUE` so that the extra sampling is *not* conducted outside of the training set.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` which is the variable used to sample.

Minimum observations

Each minority class must have at least `neighbors + 1` observations to perform the SMOTE algorithm.

Tidying

When you `tidy()` this step, a tibble is returned with columns `terms` and `id`:

terms character, the selectors or variables selected

id character, id of this step

Tuning Parameters

This step has 2 tuning parameters:

- `over_ratio`: Over-Sampling Ratio (type: double, default: 1)
- `neighbors`: # Nearest Neighbors (type: integer, default: 5)

Case weights

The underlying operation does not allow for case weights. Supplying data with a case weights column to this step results in an error.

References

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). Smote: Synthetic minority over-sampling technique. Journal of Artificial Intelligence Research, 16:321-357.

See Also

`smote()` for direct implementation

`step_enn()` and `step_tomek()`, which are commonly composed after `step_smote()` to clean the ambiguous points that over-sampling creates near the class boundary (the equivalent of imbalanced-learn's `SMOTEENN` and `SMOTETomek`).

Other Steps for over-sampling: `step_adasyn()`, `step_bsmote()`, `step_kmeans_smote()`, `step_rose()`, `step_smogn()`, `step_smoten()`, `step_smotenc()`, `step_svmsmote()`, `step_upsample()`

Examples

```
library(recipes)
library(modeldata)
data(hpc_data)

hpc_data0 <- hpc_data |>
  select(-protocol, -day)

orig <- count(hpc_data0, class, name = "orig")
orig

up_rec <- recipe(class ~ ., data = hpc_data0) |>
  # Bring the minority levels up to about 1000 each
  # 1000/2211 is approx 0.4523
  step_smote(class, over_ratio = 0.4523) |>
  prep()

training <- up_rec |>
  bake(new_data = NULL) |>
  count(class, name = "training")
training

# Since `skip` defaults to TRUE, baking the step has no effect
baked <- up_rec |>
  bake(new_data = hpc_data0) |>
  count(class, name = "baked")
baked

# Note that if the original data contained more rows than the
# target n (= ratio * majority_n), the data are left alone:
orig |>
  left_join(training, by = "class") |>
  left_join(baked, by = "class")

# A named vector gives each level its own target. Here "VF" is left
# untouched and only "L" is brought up to the size of the majority level.
recipe(class ~ ., data = hpc_data0) |>
  step_smote(class, over_ratio = c(L = 1)) |>
  prep() |>
  bake(new_data = NULL) |>
  count(class)
```

```
library(ggplot2)

ggplot(circle_example, aes(x, y, color = class)) +
  geom_point() +
  labs(title = "Without SMOTE")

recipe(class ~ x + y, data = circle_example) |>
  step_smote(class) |>
  prep() |>
  bake(new_data = NULL) |>
  ggplot(aes(x, y, color = class)) +
  geom_point() +
  labs(title = "With SMOTE")
```

`step_smoten` *Apply SMOTEN Algorithm*

Description

`step_smoten()` creates a *specification* of a recipe step that generate new examples of the minority class using nearest neighbors of these cases, for data sets where all predictors are categorical (nominal). The Value Difference Metric (VDM) is used to measure the distance between observations. For each predictor, the most common category among neighbors is chosen.

Usage

```
step_smoten(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  column = NULL,
  over_ratio = 1,
  neighbors = 5,
  indicator_column = NULL,
  skip = TRUE,
  seed = sample.int(10^5, 1),
  id = rand_id("smoten")
)
```

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 which variable is used to sample the data. See recipes::selections for more details. The selection should result in *single factor variable*. For the `tidy` method, these are not currently used.

role	Not used by this step since no new variables are created.
trained	A logical to indicate if the quantities for preprocessing have been estimated.
column	A character string of the variable name that will be populated (eventually) by the ... selectors.
over_ratio	A numeric value for the ratio of the minority-to-majority frequencies. The default value (1) means that all other levels are sampled up to have the same frequency as the most occurring level. A value of 0.5 would mean that the minority levels will have (at most) (approximately) half as many rows as the majority level. A named numeric vector can be used instead to give different levels different targets, for example `c(a = 1, b = 0.5)`. The names must be levels of the outcome and the values are ratios of the majority level, exactly as in the single-number case. Levels that are not named are left untouched, as are rows with a missing outcome. Because a vector of targets is not a single value, supplying one means this argument can no longer be tuned. See `vignette("ratio", package = "themis")` for more details.
neighbors	An integer. Number of nearest neighbor that are used to generate the new examples of the minority class.
indicator_column	A single string or `NULL` (the default). If a string is given, a logical column with that name is added to the output, marking rows added by the step (`TRUE`) vs rows from the original data (`FALSE`).
skip	A logical. Should the step be skipped when the recipe is baked by `bake()`? While all operations are baked when `prep()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.
seed	An integer that will be used as the seed when applied.
id	A character string that is unique to this step to identify it.

Details

SMOTEN is a variant of SMOTE designed for data sets where all predictors are categorical (nominal). Since standard SMOTE relies on interpolation in continuous space, it cannot be applied to categorical features directly. SMOTEN instead uses the Value Difference Metric (VDM) to measure the distance between observations. For each minority class example, new synthetic examples are generated by taking the most common value of each predictor among its nearest neighbors. The number of new examples generated is controlled by `over_ratio`.

All columns in the data are sampled and returned by `recipes::juice()` and `recipes::bake()`.

All predictor columns must be categorical (factor or character) with no missing data.

When used in modeling, users should strongly consider using the option `skip = TRUE` so that the extra sampling is *not* conducted outside of the training set.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` which is the variable used to sample.

Minimum observations

Each minority class must have at least `neighbors + 1` observations to perform the SMOTEN algorithm.

Value Difference Metric

The Value Difference Metric (VDM) used here deviates from Chawla's stated form in two ways. The per-feature deltas are aggregated by summing them (r = 1) rather than by taking their Euclidean norm (r = 2), and when a synthetic value is chosen by majority vote of the nearest neighbors the seed observation itself is excluded from the vote. The metric is internally consistent and is a valid VDM variant, but be aware of these choices when comparing results with other implementations.

Tidying

When you `tidy()` this step, a tibble is returned with columns `terms` and `id`:

terms character, the selectors or variables selected

id character, id of this step

Tuning Parameters

This step has 2 tuning parameters:

- `over_ratio`: Over-Sampling Ratio (type: double, default: 1)
- `neighbors`: # Nearest Neighbors (type: integer, default: 5)

Case weights

The underlying operation does not allow for case weights. Supplying data with a case weights column to this step results in an error.

References

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). Smote: Synthetic minority over-sampling technique. Journal of Artificial Intelligence Research, 16:321-357.

See Also

`smoten()` for direct implementation

Other Steps for over-sampling: `step_adasyn()`, `step_bsmote()`, `step_kmeans_smote()`, `step_rose()`, `step_smogn()`, `step_smote()`, `step_smotenc()`, `step_svmsmote()`, `step_upsample()`

Examples

```
library(recipes)
library(modeldata)
data(hpc_data)

hpc_cat <- hpc_data[, c("class", "protocol", "day")]
```

```
orig <- count(hpc_cat, class, name = "orig")
orig

up_rec <- recipe(class ~ ., data = hpc_cat) |>
  step_smoten(class) |>
  prep()

training <- up_rec |>
  bake(new_data = NULL) |>
  count(class, name = "training")
training

# Since `skip` defaults to TRUE, baking the step has no effect
baked <- up_rec |>
  bake(new_data = hpc_cat) |>
  count(class, name = "baked")
baked
```

---

step_smotenc *Apply SMOTENC Algorithm*

Description

step_smotenc() creates a *specification* of a recipe step that generate new examples of the minority class using nearest neighbors of these cases. Gower's distance is used to handle mixed data types. For categorical variables, the most common category along neighbors is chosen.

Usage

```
step_smotenc(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  column = NULL,
  over_ratio = 1,
  neighbors = 5,
  indicator_column = NULL,
  skip = TRUE,
  seed = sample.int(10^5, 1),
  id = rand_id("smotenc")
)
```

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 which variable is used to sample the data. See recipes::selections for more details. The selection should result in *single factor variable*. For the `tidy` method, these are not currently used.

`role` Not used by this step since no new variables are created.

`trained` A logical to indicate if the quantities for preprocessing have been estimated.

`column` A character string of the variable name that will be populated (eventually) by the `...` selectors.

`over_ratio` A numeric value for the ratio of the minority-to-majority frequencies. The default value (1) means that all other levels are sampled up to have the same frequency as the most occurring level. A value of 0.5 would mean that the minority levels will have (at most) (approximately) half as many rows as the majority level.

A named numeric vector can be used instead to give different levels different targets, for example `c(a = 1, b = 0.5)`. The names must be levels of the outcome and the values are ratios of the majority level, exactly as in the single-number case. Levels that are not named are left untouched, as are rows with a missing outcome. Because a vector of targets is not a single value, supplying one means this argument can no longer be tuned. See `vignette("ratio", package = "themis")` for more details.

`neighbors` An integer. Number of nearest neighbor that are used to generate the new examples of the minority class.

`indicator_column`

A single string or `NULL` (the default). If a string is given, a logical column with that name is added to the output, marking rows added by the step (`TRUE`) vs rows from the original data (`FALSE`).

`skip` A logical. Should the step be skipped when the recipe is baked by `bake()`? While all operations are baked when `prep()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

`seed` An integer that will be used as the seed when applied.

`id` A character string that is unique to this step to identify it.

Details

SMOTENC extends SMOTE to handle data sets with a mix of numeric and categorical predictors. For each minority class example, new synthetic examples are generated by interpolating between the example and its nearest neighbors using Gower's distance. Numeric features are interpolated continuously; categorical features take the most common value among the neighbors. The number of new examples generated is controlled by `over_ratio`.

All columns in the data are sampled and returned by `recipes::juice()` and `recipes::bake()`.

Columns can be numeric and categorical with no missing data.

When used in modeling, users should strongly consider using the option `skip = TRUE` so that the extra sampling is *not* conducted outside of the training set.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` which is the variable used to sample.

Minimum observations

Each minority class must have at least `neighbors + 1` observations to perform the SMOTENC algorithm.

Tidying

When you `tidy()` this step, a tibble is returned with columns `terms` and `id`:

terms character, the selectors or variables selected

id character, id of this step

Tuning Parameters

This step has 2 tuning parameters:

- `over_ratio`: Over-Sampling Ratio (type: double, default: 1)
- `neighbors`: # Nearest Neighbors (type: integer, default: 5)

Case weights

The underlying operation does not allow for case weights. Supplying data with a case weights column to this step results in an error.

References

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). Smote: Synthetic minority over-sampling technique. Journal of Artificial Intelligence Research, 16:321-357.

Gower, J. C. (1971). A general coefficient of similarity and some of its properties. Biometrics 27(4):857-871. (For the distance metric used)

See Also

`smotenc()` for direct implementation

Other Steps for over-sampling: `step_adasyn()`, `step_bsmote()`, `step_kmeans_smote()`, `step_rose()`, `step_smogn()`, `step_smote()`, `step_smoten()`, `step_svmsmote()`, `step_upsample()`

Examples

```
library(recipes)
library(modeldata)
data(hpc_data)

orig <- count(hpc_data, class, name = "orig")
orig
```

```
up_rec <- recipe(class ~ ., data = hpc_data) |>
  # Bring the minority levels up to about 1000 each
  # 1000/2211 is approx 0.4523
  step_smotenc(class, over_ratio = 0.4523) |>
  prep()

training <- up_rec |>
  bake(new_data = NULL) |>
  count(class, name = "training")
training

# Since `skip` defaults to TRUE, baking the step has no effect
baked <- up_rec |>
  bake(new_data = hpc_data) |>
  count(class, name = "baked")
baked

# Note that if the original data contained more rows than the
# target n (= ratio * majority_n), the data are left alone:
orig |>
  left_join(training, by = "class") |>
  left_join(baked, by = "class")
```

`step_svmsmote` *Apply SVM-SMOTE Algorithm*

Description

`step_svmsmote()` creates a *specification* of a recipe step that generate new examples of the minority class near the decision boundary using the support vectors of a fitted support vector machine.

Usage

```
step_svmsmote(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  column = NULL,
  over_ratio = 1,
  neighbors = 5,
  distance = "euclidean",
  m_neighbors = NULL,
  out_step = 0.5,
  indicator_column = NULL,
  skip = TRUE,
  seed = sample.int(10^5, 1),
```

```
  id = rand_id("svmsmote")
)
```

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 which variable is used to sample the data. See recipes::selections for more details. The selection should result in *single factor variable*. For the tidy method, these are not currently used.
role	Not used by this step since no new variables are created.
trained	A logical to indicate if the quantities for preprocessing have been estimated.
column	A character string of the variable name that will be populated (eventually) by the ... selectors.
over_ratio	A numeric value for the ratio of the minority-to-majority frequencies. The default value (1) means that all other levels are sampled up to have the same frequency as the most occurring level. A value of 0.5 would mean that the minority levels will have (at most) (approximately) half as many rows as the majority level. A named numeric vector can be used instead to give different levels different targets, for example `c(a = 1, b = 0.5)`. The names must be levels of the outcome and the values are ratios of the majority level, exactly as in the single-number case. Levels that are not named are left untouched, as are rows with a missing outcome. Because a vector of targets is not a single value, supplying one means this argument can no longer be tuned. See `vignette("ratio", package = "themis")` for more details.
neighbors	An integer. Number of nearest neighbor that are used to generate the new examples of the minority class.
distance	A character string specifying the distance metric used for nearest neighbor calculations, defaulting to `"euclidean"`. The available metrics fall into three groups. `"euclidean"`, `"cosine"`, and `"mahalanobis"` use approximate nearest neighbors via the RANN package and scale well to large datasets. `"squared_chord"`, `"matusita"`, `"hellinger"`, and `"bhattacharyya"` are probability-divergence measures that treat each row as a distribution over the predictors, so they require non-negative values. `"hellinger"` and `"bhattacharyya"` further require each row to sum to 1. All four also use the RANN package and scale well to large datasets. `"manhattan"`, `"chebyshev"`, `"canberra"`, `"soergel"`, `"lorentzian"`, `"jeffreys"`, `"topsoe"`, `"jensen-shannon"`, `"jensen_difference"`, `"taneja"`, and `"kumar-johnson"` compute an exact all-pairs distance matrix. This takes time and memory proportional to the square of the number of observations in a class, so these are best suited to smaller datasets. Everything from `"canberra"` onwards is a probability divergence requiring non-negative values, is provided by the philentropy package (which must be installed separately), and in the case of `"jeffreys"`,

"taneja", and "kumar-johnson" requires strictly positive values, since those divide by individual predictor values.

The probability divergences are meaningful for compositional predictors such as proportions or counts normalized per observation, and are generally not appropriate for standardized predictors.

`m_neighbors` An integer or NULL. Number of nearest neighbors, among all classes, that are used to label each minority support vector as noise, danger, or safe. Defaults to NULL, which means `2 * neighbors`.

`out_step` A number. Step size used when extrapolating new examples away from safe support vectors. Defaults to 0.5.

`indicator_column`
A single string or NULL (the default). If a string is given, a logical column with that name is added to the output, marking rows added by the step (TRUE) vs rows from the original data (FALSE).

`skip` A logical. Should the step be skipped when the recipe is baked by bake()? While all operations are baked when prep() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

`seed` An integer that will be used as the seed when applied.

`id` A character string that is unique to this step to identify it.

Details

SVM-SMOTE (Support Vector Machine SMOTE) works the same way as SMOTE, except that instead of generating points around every point of the minority class, it focuses generation near the decision boundary. A support vector machine is fitted to the data and the support vectors that belong to the minority class are used as the base points for generating new examples.

For each minority support vector its nearest neighbors among all classes are calculated. If all of the neighbors come from a different class the support vector is labeled noise and is discarded. If more than half of the neighbors come from a different class the support vector is labeled "danger" and new points are interpolated between it and its minority-class neighbors. The remaining support vectors are considered to be in a safe region and new points are extrapolated away from their minority-class neighbors.

The number of neighbors used for this labeling is controlled by `m_neighbors`, and how far the extrapolated points are placed is controlled by `out_step`.

The support vector machine is always fitted with `kernlab::ksvm()` using a radial basis function kernel (`kernel = "rbfdot"`) and a cost of `C = 1`, matching the reference implementations of the method. These are not exposed as arguments because the fitted model is only used to identify which minority observations are support vectors, and not for prediction, so the sampling results are relatively insensitive to them.

SMOTE generates new examples of the minority class using nearest neighbors of these cases. For each existing minority class example, new examples are created by interpolating between the example and its nearest neighbors. The number of nearest neighbors used is controlled by the number of neighbors argument (k in `smote()`, `neighbors` in `step_smote()`), and the number of new examples generated is controlled by `over_ratio`.

All columns in the data are sampled and returned by `recipes::juice()` and `recipes::bake()`.

All columns used in this step must be numeric with no missing data.

When used in modeling, users should strongly consider using the option `skip = TRUE` so that the extra sampling is *not* conducted outside of the training set.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` which is the variable used to sample.

Minimum observations

Each minority class must have at least `neighbors + 1` observations to perform the SVM-SMOTE algorithm.

Tidying

When you `tidy()` this step, a tibble is returned with columns `terms` and `id`:

terms character, the selectors or variables selected

id character, id of this step

Tuning Parameters

This step has 3 tuning parameters:

- `over_ratio`: Over-Sampling Ratio (type: double, default: 1)
- `neighbors`: # Nearest Neighbors (type: integer, default: 5)
- `m_neighbors`: # Nearest Neighbors (type: integer, default: NULL)

Case weights

The underlying operation does not allow for case weights. Supplying data with a case weights column to this step results in an error.

References

Nguyen, H. M., Cooper, E. W., and Kamei, K. (2011). Borderline over-sampling for imbalanced data classification. International Journal of Knowledge Engineering and Soft Data Paradigms, 3(1), 4-21.

See Also

`svmsmote()` for direct implementation

Other Steps for over-sampling: `step_adasyn()`, `step_bsmote()`, `step_kmeans_smote()`, `step_rose()`, `step_smogn()`, `step_smote()`, `step_smoten()`, `step_smotenc()`, `step_upsample()`

Examples

```
library(recipes)
library(modeldata)
data(hpc_data)

hpc_data0 <- hpc_data |>
  select(-protocol, -day)

orig <- count(hpc_data0, class, name = "orig")
orig

up_rec <- recipe(class ~ ., data = hpc_data0) |>
  # Bring the minority levels up to about 1000 each
  # 1000/2211 is approx 0.4523
  step_svmsmote(class, over_ratio = 0.4523) |>
  prep()

training <- up_rec |>
  bake(new_data = NULL) |>
  count(class, name = "training")
training

# Since `skip` defaults to TRUE, baking the step has no effect
baked <- up_rec |>
  bake(new_data = hpc_data0) |>
  count(class, name = "baked")
baked

library(ggplot2)

ggplot(circle_example, aes(x, y, color = class)) +
  geom_point() +
  labs(title = "Without SMOTE")

recipe(class ~ x + y, data = circle_example) |>
  step_svmsmote(class) |>
  prep() |>
  bake(new_data = NULL) |>
  ggplot(aes(x, y, color = class)) +
  geom_point() +
  labs(title = "With SVM-SMOTE")
```

---

step_tomek *Remove Tomek's Links*

Description

`step_tomek()` creates a *specification* of a recipe step that removes majority class instances of tomek links.

Usage

```
step_tomek(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  column = NULL,
  distance = "euclidean",
  skip = TRUE,
  seed = sample.int(10^5, 1),
  distance_with = recipes::all_predictors(),
  id = rand_id("tomek")
)
```

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 which variable is used to sample the data. See recipes::selections for more details. The selection should result in *single factor variable*. For the tidy method, these are not currently used.
role	Not used by this step since no new variables are created.
trained	A logical to indicate if the quantities for preprocessing have been estimated.
column	A character string of the variable name that will be populated (eventually) by the ... selectors.
distance	A character string specifying the distance metric used for nearest neighbor calculations, defaulting to "euclidean". The available metrics fall into three groups. "euclidean", "cosine", and "mahalanobis" use approximate nearest neighbors via the RANN package and scale well to large datasets. "squared_chord", "matusita", "hellinger", and "bhattacharyya" are probability-divergence measures that treat each row as a distribution over the predictors, so they require non-negative values. "hellinger" and "bhattacharyya" further require each row to sum to 1. All four also use the RANN package and scale well to large datasets. "manhattan", "chebyshev", "canberra", "soergel", "lorentzian", "jeffreys", "topsoe", "jensen-shannon", "jensen_difference", "taneja", and "kumar-johnson" compute an exact all-pairs distance matrix. This takes time and memory proportional to the square of the number of observations in a class, so these are best suited to smaller datasets. Everything from "canberra" onwards is a probability divergence requiring non-negative values, is provided by the philentropy package (which must be installed separately), and in the case of "jeffreys", "taneja", and "kumar-johnson" requires strictly positive values, since those divide by individual predictor values. The probability divergences are meaningful for compositional predictors such as proportions or counts normalized per observation, and are generally not appropriate for standardized predictors.

skip	A logical. Should the step be skipped when the recipe is baked by bake()? While all operations are baked when prep() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations.
seed	An integer that will be used as the seed when applied.
distance_with	A call to a selector function to choose which variables are used for distance calculations. Defaults to recipes::all_predictors(). The variable selected by ... is always excluded from the distance calculations.
id	A character string that is unique to this step to identify it.

Details

A Tomek link is a pair of points from different classes that are each other's nearest neighbors. Such pairs sit on or very near the decision boundary and are considered noise or borderline cases. The algorithm identifies all Tomek links and removes the majority class instance from each pair, cleaning the class boundary without discarding non-boundary majority examples. Because only boundary points are removed, this typically discards far fewer observations than other under-sampling methods.

All variables selected by distance_with must be numeric with no missing data.

All columns in the data are sampled and returned by recipes::juice() and recipes::bake().

When used in modeling, users should strongly consider using the option skip = TRUE so that the extra sampling is *not* conducted outside of the training set.

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns terms which is the variable used to sample.

Tidying

When you tidy() this step, a tibble is returned with columns terms and id:

terms character, the selectors or variables selected

id character, id of this step

Case weights

The underlying operation does not allow for case weights. Supplying data with a case weights column to this step results in an error.

References

Tomek. Two modifications of cnn. IEEE Trans. Syst. Man Cybern., 6:769-772, 1976.

See Also

tomek() for direct implementation

step_smote(), which is commonly composed before step_tomek() to clean the ambiguous points that over-sampling creates near the class boundary (the equivalent of imbalanced-learn's SMOTETomek).

Other Steps for under-sampling: step_cluster_centroids(), step_cnn(), step_downsample(), step_enn(), step_instance_hardness(), step_ncl(), step_nearmiss(), step_oss()

Examples

```
library(recipes)
library(modeldata)
data(hpc_data)

hpc_data0 <- hpc_data |>
  select(-protocol, -day)

orig <- count(hpc_data0, class, name = "orig")
orig

up_rec <- recipe(class ~ ., data = hpc_data0) |>
  step_tomek(class) |>
  prep()

training <- up_rec |>
  bake(new_data = NULL) |>
  count(class, name = "training")
training

# Since `skip` defaults to TRUE, baking the step has no effect
baked <- up_rec |>
  bake(new_data = hpc_data0) |>
  count(class, name = "baked")
baked

orig |>
  left_join(training, by = "class") |>
  left_join(baked, by = "class")

library(ggplot2)

ggplot(circle_example, aes(x, y, color = class)) +
  geom_point() +
  labs(title = "Without Tomek") +
  xlim(c(1, 15)) +
  ylim(c(1, 15))

recipe(class ~ x + y, data = circle_example) |>
  step_tomek(class) |>
  prep() |>
  bake(new_data = NULL) |>
  ggplot(aes(x, y, color = class)) +
```

```
  geom_point() +
  labs(title = "With Tomek") +
  xlim(c(1, 15)) +
  ylim(c(1, 15))
```

`step_upsample` *Up-Sample a Data Set Based on a Factor Variable*

Description

`step_upsample()` creates a *specification* of a recipe step that will replicate rows of a data set to make the occurrence of levels in a specific factor level equal.

Usage

```
step_upsample(
  recipe,
  ...,
  over_ratio = 1,
  ratio = deprecated(),
  role = NA,
  trained = FALSE,
  column = NULL,
  target = NA,
  indicator_column = NULL,
  skip = TRUE,
  seed = sample.int(10^5, 1),
  id = rand_id("upsample")
)
```

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 which variable is used to sample the data. See recipes::selections for more details. The selection should result in *single factor variable*. For the `tidy` method, these are not currently used.
`over_ratio`	A numeric value for the ratio of the minority-to-majority frequencies. The default value (1) means that all other levels are sampled up to have the same frequency as the most occurring level. A value of 0.5 would mean that the minority levels will have (at most) (approximately) half as many rows as the majority level. A named numeric vector can be used instead to give different levels different targets, for example `c(a = 1, b = 0.5)`. The names must be levels of the outcome and the values are ratios of the majority level, exactly as in the single-number

case. Levels that are not named are left untouched, as are rows with a missing outcome. Because a vector of targets is not a single value, supplying one means this argument can no longer be tuned. See `vignette("ratio", package = "themis")` for more details.

`ratio` Deprecated argument; same as `over_ratio`.

`role` For new variables created by this step, what analysis role should they be assigned? Only used when `indicator_column` is not `NULL`.

`trained` A logical to indicate if the quantities for preprocessing have been estimated.

`column` A character string of the variable name that will be populated (eventually) by the ... selectors.

`target` A named numeric vector giving the number of rows to sample each level up to. This should not be set by the user and will be populated by `prep`.

`indicator_column`
A single string or `NULL` (the default). If a string is given, a logical column with that name is added to the output, marking rows added by the step (`TRUE`) vs rows from the original data (`FALSE`).

`skip` A logical. Should the step be skipped when the recipe is baked by `bake()`? While all operations are baked when `prep()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

`seed` An integer that will be used as the seed when applied.

`id` A character string that is unique to this step to identify it.

Details

Up-sampling is intended to be performed on the *training* set alone. For this reason, the default is `skip = TRUE`.

If there are missing values in the factor variable that is used to define the sampling, missing data are selected at random in the same way that the other factor levels are sampled. Missing values are not used to determine the amount of data in the majority level (see example below).

For any data with factor levels occurring with the same frequency as the majority level, all data will be retained.

All columns in the data are sampled and returned by `recipes::juice()` and `recipes::bake()`.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` which is the variable used to sample.

Tidying

When you `tidy()` this step, a tibble is returned with columns `terms` and `id`:

terms character, the selectors or variables selected

id character, id of this step

Tuning Parameters

This step has 1 tuning parameters:

- `over_ratio`: Over-Sampling Ratio (type: double, default: 1)

Case weights

This step performs an unsupervised operation that can utilize case weights. To use them, see the documentation in recipes::case_weights and the examples on `tidymodels.org`.

See Also

Other Steps for over-sampling: `step_adasyn()`, `step_bsmote()`, `step_kmeans_smote()`, `step_rose()`, `step_smogn()`, `step_smote()`, `step_smoten()`, `step_smotenc()`, `step_svmsmote()`

Examples

```
library(recipes)
library(modeldata)
data(hpc_data)

hpc_data0 <- hpc_data |>
  select(-protocol, -day)

orig <- count(hpc_data0, class, name = "orig")
orig

up_rec <- recipe(class ~ ., data = hpc_data0) |>
  # Bring the minority levels up to about 1000 each
  # 1000/2211 is approx 0.4523
  step_upsample(class, over_ratio = 0.4523) |>
  prep()

training <- up_rec |>
  bake(new_data = NULL) |>
  count(class, name = "training")
training

# Since `skip` defaults to TRUE, baking the step has no effect
baked <- up_rec |>
  bake(new_data = hpc_data0) |>
  count(class, name = "baked")
baked

# Note that if the original data contained more rows than the
# target n (= ratio * majority_n), the data are left alone:
orig |>
  left_join(training, by = "class") |>
  left_join(baked, by = "class")

library(ggplot2)
```

```
ggplot(circle_example, aes(x, y, color = class)) +
  geom_point() +
  labs(title = "Without upsample")

recipe(class ~ x + y, data = circle_example) |>
  step_upsample(class) |>
  prep() |>
  bake(new_data = NULL) |>
  ggplot(aes(x, y, color = class)) +
  geom_jitter(width = 0.1, height = 0.1) +
  labs(title = "With upsample (with jittering)")
```

`svmsmote` *SVM-SMOTE Algorithm*

Description

SVM-SMOTE generates new examples of the minority class near the decision boundary, using the support vectors of a fitted SVM to decide where to place synthetic examples.

Usage

```
svmsmote(
  df,
  var,
  k = 5,
  over_ratio = 1,
  distance = "euclidean",
  m_neighbors = NULL,
  out_step = 0.5
)
```

Arguments

`df`	data.frame or tibble. Must have 1 factor variable and remaining numeric variables.
`var`	Character, name of variable containing factor variable.
`k`	An integer. Number of nearest neighbor that are used to generate the new examples of the minority class.
`over_ratio`	A numeric value for the ratio of the minority-to-majority frequencies. The default value (1) means that all other levels are sampled up to have the same frequency as the most occurring level. A value of 0.5 would mean that the minority levels will have (at most) (approximately) half as many rows as the majority level. A named numeric vector can be used instead to give different levels different targets, for example `c(a = 1, b = 0.5)`. The names must be levels of the outcome

and the values are ratios of the majority level, exactly as in the single-number case. Levels that are not named are left untouched, as are rows with a missing outcome. Because a vector of targets is not a single value, supplying one means this argument can no longer be tuned. See `vignette("ratio", package = "themis")` for more details.

`distance` A character string specifying the distance metric used for nearest neighbor calculations, defaulting to `"euclidean"`. The available metrics fall into three groups.

`"euclidean"`, `"cosine"`, and `"mahalanobis"` use approximate nearest neighbors via the RANN package and scale well to large datasets.

`"squared_chord"`, `"matusita"`, `"hellinger"`, and `"bhattacharyya"` are probability-divergence measures that treat each row as a distribution over the predictors, so they require non-negative values. `"hellinger"` and `"bhattacharyya"` further require each row to sum to 1. All four also use the RANN package and scale well to large datasets.

`"manhattan"`, `"chebyshev"`, `"canberra"`, `"soergel"`, `"lorentzian"`, `"jeffreys"`, `"topsoe"`, `"jensen-shannon"`, `"jensen_difference"`, `"taneja"`, and `"kumar-johnson"` compute an exact all-pairs distance matrix. This takes time and memory proportional to the square of the number of observations in a class, so these are best suited to smaller datasets. Everything from `"canberra"` onwards is a probability divergence requiring non-negative values, is provided by the philentropy package (which must be installed separately), and in the case of `"jeffreys"`, `"taneja"`, and `"kumar-johnson"` requires strictly positive values, since those divide by individual predictor values.

The probability divergences are meaningful for compositional predictors such as proportions or counts normalized per observation, and are generally not appropriate for standardized predictors.

`m_neighbors` An integer or `NULL`. Number of nearest neighbors, among all classes, that are used to label each minority support vector as noise, danger, or safe. Defaults to `NULL`, which means `2 * k`.

`out_step` A number. Step size used when extrapolating new examples away from safe support vectors. Defaults to 0.5.

Details

SVM-SMOTE (Support Vector Machine SMOTE) works the same way as SMOTE, except that instead of generating points around every point of the minority class, it focuses generation near the decision boundary. A support vector machine is fitted to the data and the support vectors that belong to the minority class are used as the base points for generating new examples.

For each minority support vector its nearest neighbors among all classes are calculated. If all of the neighbors come from a different class the support vector is labeled noise and is discarded. If more than half of the neighbors come from a different class the support vector is labeled "danger" and new points are interpolated between it and its minority-class neighbors. The remaining support vectors are considered to be in a safe region and new points are extrapolated away from their minority-class neighbors.

The number of neighbors used for this labeling is controlled by `m_neighbors`, and how far the extrapolated points are placed is controlled by `out_step`.

The support vector machine is always fitted with `kernlab::ksvm()` using a radial basis function kernel (`kernel = "rbfdot"`) and a cost of `C = 1`, matching the reference implementations of the method. These are not exposed as arguments because the fitted model is only used to identify which minority observations are support vectors, and not for prediction, so the sampling results are relatively insensitive to them.

SMOTE generates new examples of the minority class using nearest neighbors of these cases. For each existing minority class example, new examples are created by interpolating between the example and its nearest neighbors. The number of nearest neighbors used is controlled by the number of neighbors argument (`k` in `smote()`, `neighbors` in `step_smote()`), and the number of new examples generated is controlled by `over_ratio`.

All columns used in this function must be numeric with no missing data.

Value

A data.frame or tibble, depending on type of `df`.

References

Nguyen, H. M., Cooper, E. W., and Kamei, K. (2011). Borderline over-sampling for imbalanced data classification. International Journal of Knowledge Engineering and Soft Data Paradigms, 3(1), 4-21.

See Also

`step_svmsmote()` for step function of this method

Other Direct Implementations: `adasyn()`, `bsmote()`, `cluster_centroids()`, `cnn()`, `enn()`, `instance_hardness()`, `kmeans_smote()`, `ncl()`, `nearmiss()`, `oss()`, `rose()`, `smogn()`, `smote()`, `smoten()`, `smotenc()`, `tomek()`

Examples

```
circle_numeric <- circle_example[, c("x", "y", "class")]

res <- svmsmote(circle_numeric, var = "class")

res <- svmsmote(circle_numeric, var = "class", k = 10)

res <- svmsmote(circle_numeric, var = "class", over_ratio = 0.8)

res <- svmsmote(circle_numeric, var = "class", distance = "manhattan")

res <- svmsmote(circle_numeric, var = "class", m_neighbors = 20)
```

tomek *Remove Tomek's Links*

Description

Removes the majority class member of each pair of observations that form a Tomek link.

Usage

```
tomek(df, var, distance = "euclidean")
```

Arguments

df	data.frame or tibble. Must have 1 factor variable and remaining numeric variables.
var	Character, name of variable containing factor variable.
distance	A character string specifying the distance metric used for nearest neighbor calculations, defaulting to "euclidean". The available metrics fall into three groups. "euclidean", "cosine", and "mahalanobis" use approximate nearest neighbors via the RANN package and scale well to large datasets. "squared_chord", "matusita", "hellinger", and "bhattacharyya" are probability-divergence measures that treat each row as a distribution over the predictors, so they require non-negative values. "hellinger" and "bhattacharyya" further require each row to sum to 1. All four also use the RANN package and scale well to large datasets. "manhattan", "chebyshev", "canberra", "soergel", "lorentzian", "jeffreys", "topsoe", "jensen-shannon", "jensen_difference", "taneja", and "kumar-johnson" compute an exact all-pairs distance matrix. This takes time and memory proportional to the square of the number of observations in a class, so these are best suited to smaller datasets. Everything from "canberra" onwards is a probability divergence requiring non-negative values, is provided by the philentropy package (which must be installed separately), and in the case of "jeffreys", "taneja", and "kumar-johnson" requires strictly positive values, since those divide by individual predictor values. The probability divergences are meaningful for compositional predictors such as proportions or counts normalized per observation, and are generally not appropriate for standardized predictors.

Details

A Tomek link is a pair of points from different classes that are each other's nearest neighbors. Such pairs sit on or very near the decision boundary and are considered noise or borderline cases. The algorithm identifies all Tomek links and removes the majority class instance from each pair, cleaning the class boundary without discarding non-boundary majority examples. Because only boundary

points are removed, this typically discards far fewer observations than other under-sampling methods.

All columns used in this function must be numeric with no missing data.

Value

A data.frame or tibble, depending on type of `df`.

References

Tomek. Two modifications of cnn. IEEE Trans. Syst. Man Cybern., 6:769-772, 1976.

See Also

`step_tomek()` for step function of this method

Other Direct Implementations: `adasyn()`, `bsmote()`, `cluster_centroids()`, `cnn()`, `enn()`, `instance_hardness()`, `kmeans_smote()`, `ncl()`, `nearmiss()`, `oss()`, `rose()`, `smogn()`, `smote()`, `smoten()`, `smotenc()`, `svmsmote()`

Examples

```
circle_numeric <- circle_example[, c("x", "y", "class")]

res <- tomek(circle_numeric, var = "class")

res <- tomek(circle_numeric, var = "class", distance = "manhattan")
```

Index