Package 'testsampleR'

April 24, 2025

Title Sampling Test Sets and Quantifying Uncertainty for Evaluating Classifiers

Version 0.0.0.9008

Author Francisco Tomas-Valiente <tomasf@ethz.ch> [aut, cre]

Maintainer Francisco Tomas-Valiente <tomasf@ethz.ch>

Description This package allows designing and drawing test sets for classifier evaluation, and quantifying uncertainty in resulting performance metrics. It does so by implementing the methods developed in Tomas-Valiente (2025), which develops sampling distributions for performance metrics based on SRS or stratified test sets. In particular, the package provides functions to design stratified sampling schemes that allow drawing test sets that reduce the variance around performance estimates. In doing so, the package supports various allocations. Additionally, the package provides functions that estimates metrics with standard errors for test set drawn with simple random sampling and stratified sampling. Lastly, the package allows researchers to implement sample size calculations to decide how large a test set to draw to achieve a desired level of statistical precision.

License GPL-3

Encoding UTF-8

RoxygenNote 7.3.2

Date/Publication 2025-03-18 23:00:00 CET

Depends R (>= 2.10)

LazyData true

Suggests knitr, rmarkdown

VignetteBuilder knitr

Contents

`ci_srs_wilson` *Estimate Wilson confidence intervals for simple random sample*

Description

This function estimates CIs with Wilson method from confusion matrix drawn by simple random sampling

Usage

```
ci_srs_wilson(
  tr0_pred0,
  tr1_pred0,
  tr0_pred1,
  tr1_pred1,
  N_sample,
  alpha = 0.05,
  wide = TRUE
)
```

Arguments

`tr0_pred0`	A numeric value capturing the share of true negatives.
`tr1_pred0`	A numeric value capturing the share of false negatives.
`tr0_pred1`	A numeric value capturing the share of false positives.
`tr1_pred1`	A numeric value capturing the share of true positives.
`N_sample`	A numeric value capturing the total number size of the test set.
`alpha`	A numeric value used capturing the type-I error rate for constructing confidence intervals (applies to both bootstrapping and analytic CIs).
`wide`	A logical value capturing whether the output should be in wide format, as opposed to long format. Default is wide.

Details

This function takes a confusion matrix and sample size as input, and produces CIs with Wilson's score method for the F1 score, precision and recall. These CIs are based on assuming the test set was drawn by simple random sampling: the CIs should not be trusted if sampling was conducted in a different way. Note that the sum of the proportion of observations across all four cells must be one. To estimate SEs from a test set data rather than from a confusion matrix, use 'stratified_metrics'.

Value

A data.frame containing the upper and lower endpoints of the Wilson CIs for the F1 score, precision and recall. The dataset is in long or wide format depending on argument 'wide'. These CIs are based on assuming the test set was drawn by simple random sampling: the CIs should not be trusted if sampling was conducted in a different way.

References

Lam, K. F. Y. (2023). Confidence intervals for the f1 score: A comparison of four methods. arXiv preprint arXiv:2309.14621

Tomas-Valiente, F. (2025). Uncertain performance: How to quantify uncertainty and draw test sets when evaluating classifiers.

See Also

stratified_metrics

Examples

```
## enter the estimates for the test set confusion matrix cells and sample size
## get the Wilson CIs analytically under assumption that test set is a SRS
ci_srs_wilson(tr0_pred0 = 0.5, tr1_pred0 = 0.1, tr0_pred1 = 0.2, tr1_pred1 = 0.2,
              N_sample = 1000, alpha = 0.05)
## for long rather than wide format
ci_srs_wilson(tr0_pred0 = 0.5, tr1_pred0 = 0.1, tr0_pred1 = 0.2, tr1_pred1 = 0.2,
              N_sample = 1000, alpha = 0.05, wide=FALSE)
```

---

constant_allocation *Estimate constant allocation*

Description

This function shows how many observations to sample per stratum under constant allocation

Usage

```
constant_allocation(data, N_sample, strata, min_per_bin = 1)
```

Arguments

data	A data.frame from which to sample observations. One column should contain the strata used for sampling.
N_sample	A numeric value capturing the desired sample size.
strata	A character value capturing the column name of the sampling strata. Column should contain factor or character values.
min_per_bin	A numeric value capturing the minimum number of observations per bin. Default is 1.

Details

This function can be used to determine how many observations to sample per stratum under constant allocation. It first divides the desired sample size equally across all strata, and then rounds the allocated observations per bin, and lastly adjusts them so the overall sample size is as close to the target as possible. NAs in the strata variable will be coded as a stratum themselves. Note the strata can be based on any information: either binned predicted probabilities (quantile- or fixed-intervals) or any other categorical information.

Value

A named vector whose values indicate the number of observations to be sampled per each stratum, and whose names correspond to the values in the ‘strata‘ variable of the input dataset.

Examples

```
## say that we want to sample a constant number of observations per stratum
## but with appropriate rounding given one cannot sample fractional observations.
## this function tells us how many obs per stratum to sample, given the strata
data(pop_df)
pop_df$strata <- cut(pop_df$score, 10)
constant_allocation(data=pop_df, N_sample=375, strata="strata", min_per_bin=1)
## note the strata can be based on any information: either predicted probabilities
## (quantile- or fixed-intervals) or any other categorical information
## NAs in the strata variable will be coded as a separate stratum
## for instance, the following code will lead to some NA in strata
## and these get grouped into a separate stratum
pop_df$strata <- cut(pop_df$score, breaks=c(0,0.2,0.4,0.6,0.8,0.9))
proportional_allocation(data=pop_df, N_sample=375, strata="strata", min_per_bin=1)
```

---

do_metrics *Estimate performance metrics from confusion matrix*

Description

This function estimates performance metrics given a test set confusion matrix

Usage

```
do_metrics(tr0_pred0, tr1_pred0, tr0_pred1, tr1_pred1)
```

Arguments

tr0_pred0	A numeric value capturing the share of true negatives.
tr1_pred0	A numeric value capturing the share of false negatives.
tr0_pred1	A numeric value capturing the share of false positives.
tr1_pred1	A numeric value capturing the share of true positives.

Details

This function can be used to estimate metrics from a confusion matrix. The function can be applied regardless of how the test set was sampled, provided the estimated share of observation in each cell of the confusion matrix is unbiased: raw shares are unbiased if test set was SRS, stratified estimators are needed if test set was drawn with stratified sampling. Note that the sum of the proportion of observations across all four cells must be one. The function is called internally within stratified_metrics. The following metrics are supported: precision, recall, F1 score for the positive class, precision of negative class, recall of negative class, F1 score for the negative class, accuracy, MCC (Matthews correlation coefficient), Cohen's kappa, (unweighted) macro-averaged F1 score, weighted F1 score, BM (bookmaker's informedness). The output also shows the proportions of true negatives, false negatives, false positives and true positives.

Value

A data.frame of performance metrics. The following metrics are supported: precision, recall, F1 of positive class, accuracy, precision of negative class, recall of negative class, F1 of negative class, MCC (Matthews correlation coefficient), Cohen's kappa, (unweighted) macro-averaged F1 score, weighted F1 score, BM (bookmaker's informedness). The output also shows the proportions of true negatives, false negatives, false positives and true positives.

See Also

stratified_metrics

Examples

```
## enter the estimates for the test set confusion matrix cells
## get estimates of key performance metriccs
do_metrics(tr0_pred0 = 0.5, tr1_pred0 = 0.1, tr0_pred1 = 0.1, tr1_pred1 = 0.3)
```

---

optimal_allocation *Estimate optimal allocation*

Description

This function shows how many observations to sample per stratum under efficient stratified sampling

Usage

```
optimal_allocation(
  data,
  N_sample,
  strata,
  stratifying,
  min_per_bin = 1,
  threshold = 0.5,
  pi1 = NULL,
  pi0 = NULL,
  recall = NULL,
  external_k = NULL,
```

```
  external_positive_share = NULL,
  weight_se_f1 = 1,
  weight_se_rec = 0,
  weight_se_prec = 0
)
```

Arguments

data	A data.frame from which to sample observations. One column should contain the strata used for sampling.
N_sample	A numeric value capturing the desired sample size.
strata	A character value capturing the column name of the sampling strata. Column should contain factor or character values. The strata should be based on the same variable entered in 'stratifying'.
stratifying	A character value capturing the column name of the stratifying variable. This column should contain continuous numeric values between 0 and 1, and its values should reflect the predicted probability of exhibiting the outcome.
min_per_bin	A numeric value capturing the minimum number of observations per bin. Default is 1.
threshold	A numeric value capturing the threshold for binary classification.
pi1	A numeric value capturing the expected precision (share of cases exhibiting the outcome out of predicted positives). This value must be entered for the function to produce an output.
pi0	A numeric value capturing the inverse of the expected precision for the negative class (share of cases exhibiting the outcome out of predicted negatives). If left blank, the function computes 'pi0' from recall and imbalance.
recall	A numeric value capturing the expected value of recall (share of positive cases among those exhibiting the outcome). This value only needs to be entered if 'pi0' is left blank.
external_k	A numeric value capturing the imbalance ratio (number of predicted positives over predicted negatives) in the population on which the recall estimate is based. This only needs to be entered if 'recall' is based on a population that has a different imbalance than the test's population. The argument is only required if 'pi0' is not entered directly. If left blank, it will be calculated internally using 'external_positive_share', if this is entered, or assumed to be equal to the test population's imbalance.
external_positive_share	A numeric value capturing the positive share (number of predicted positives out of all observations) in the population on which the recall estimate is based. This only needs to be entered if 'recall' is based on a population that has a different imbalance than the test's population. The argument is only required if 'pi0' is not entered directly. If left blank, it will be calculated internally using 'external_positive_share', if this is entered, or assumed to be equal to the test population's imbalance.
weight_se_f1	A positive numeric value capturing how much to weigh the SE for the F1 score in the objective function to be minimized. The default for 'weight_se_f1' is '1'.
weight_se_rec	A positive numeric value capturing how much to weigh the SE for recall in the objective function to be minimized. The default for 'weight_se_rec' is '0'.
weight_se_prec	A positive numeric value capturing how much to weigh the SE for precision in the objective function to be minimized. The default for 'weight_se_prec' is '0'.

Details

This function can be used to determine how many observations to sample per stratum under efficient stratified sampling. To do so, you should stratify by the predicted probability of exhibiting the outcome, as produced by the classifier to be evaluated: this should be the ‘stratifying‘ variable. The strata in ‘strata‘ should be based on this variable, and neither the ‘stratifying‘ variable nor the ‘strata‘ variable should contain NAs. At minimum, the ‘strata‘ variable should have one stratum for the positive and one for the negatives. You can create additional strata by binning the predicted probability, but importantly each stratum should contain only positive or only negative observations. The function first determines how many positive observations should be sampled in the test set, given parameter values and objective function, calling `optimal_n_positives` internally. This is the value that minimizes an objective function subject to a sample size constraint (see documentation of function `optimal_n_positives` for details). Then, the function performs proportional allocation within the positive and negative subsamples separately, adjusting the number of sampled observations per bin so that the overall sample size is as close to the target as possible. The following parameters need to be entered to receive an output: (i) pi1 (precision, TP/(TP+FP)), (ii) pi0 (FN/(FN+TN)) or recall (FN/(FN+TN)). These parameters should be guessed or estimated on other data (pi0, pi1, recall). Note that if recall is entered instead of pi0, then it is advised to also enter the imbalance ratio or positive share of the dataset on which the recall guess is based (external_k or external_positive_share): if this is not done, the imbalance in the test data is estimated by the function and used instead.

Value

A named vector whose values indicate the number of observations to be sampled per each stratum, and whose names correspond to the values in the ‘strata‘ variable of the input dataset.

References

Tomas-Valiente, F. (2025). Uncertain performance: How to quantify uncertainty and draw test sets when evaluating classifiers.

Examples

```
## example 1: say that we want to sample observations using two-bin stratification
## of positives and negatives (dichotomized at 0.5 threshold), picking the number
## of positives and negatives optimally, subject to a sample size constraint that
## N_positive + N_negative = 500. here, assume we're minimizing the objective
## function SE(F1) + 0.5 SE(precision) + 0.5 SE(recall). as assumptions, we have
## guesses for pi1 (TP/TP+FP) and pi0 (FN/FN+TN) of 0.4 and 0.02.
data(pop_df)
pop_df$strata <- ifelse(pop_df$score>=0.5, "yes", "no")
optimal_allocation(data = pop_df, N_sample = 500, strata = 'strata',
                   stratifying = 'score', min_per_bin=1, threshold=0.5, pi1=0.4,
                   pi0=0.2, weight_se_f1=1, weight_se_rec=0.5, weight_se_prec=0.5)
## example 2: same but we now want to further implement proportional stratification
## within the positives and negatives separately, after having sampled the optimal
## number of positives and negatives subject to the sample size constraint. also,
## instead of making a guess about pi0, we make a guess about recall (0.6) based on
## previous literature, and specify the imbalance in the data were said recall was
## estimated (20% positives). objective function and constraints are as above.
data(pop_df)
pop_df$strata <- cut(pop_df$score, breaks=seq(0,1,0.1))
optimal_allocation(data = pop_df, N_sample = 500, strata = 'strata',
                   stratifying = 'score', min_per_bin=1, threshold=0.5, pi1=0.4,
```

```
                  recall=0.6, external_positive_share=0.2,
                  weight_se_f1=1, weight_se_rec=0.5, weight_se_prec=0.5)
## example 3: like example 2 but assuming only care about F1, and assuming we
## specify the imbalance ratio in the dataset where recall was estimated: we
## know it has 1 positive for every 2 negatives so external_k is 1/2=0.5
data(pop_df)
pop_df$strata <- cut(pop_df$score, breaks=seq(0,1,0.1))
optimal_allocation(data = pop_df, N_sample = 500, strata = 'strata',
                  stratifying = 'score', min_per_bin=1, threshold=0.5,
                  pi1=0.4, recall=0.6, external_k=0.5,
                  weight_se_f1=1, weight_se_rec=0, weight_se_prec=0)
```

`optimal_n_positives` *Calculator of optimal number of positives*

Description

This function estimates how many positives should optimally be sampled

Usage

```
optimal_n_positives(
  N_sample = NULL,
  pi1 = NULL,
  pi0 = NULL,
  recall = NULL,
  k = NULL,
  positive_share = NULL,
  external_k = NULL,
  external_positive_share = NULL,
  weight_se_f1 = 1,
  weight_se_rec = 0,
  weight_se_prec = 0
)
```

Arguments

`N_sample`	A numeric value capturing the total test set sample size considered as binding constraint (includes both positives and negatives).
`pi1`	A numeric value capturing the expected precision (share of cases exhibiting the outcome out of predicted positives). This value must be entered for the function to produce an output.
`pi0`	A numeric value capturing the inverse of the expected precision for the negative class (share of cases exhibiting the outcome out of predicted negatives). If left blank, the function computes ‘pi0‘ from recall and imbalance.
`recall`	A numeric value capturing the expected value of recall (share of positive cases among those exhibiting the outcome). This value only needs to be entered if ‘pi0‘ is left blank.
`k`	A numeric value capturing the imbalance ratio (number of predicted positives over predicted negatives) in the population from which the test set should be drawn. Argument ‘k‘ can be left blank, but then ‘positive_share‘ must be specified. Typically, it should be below 1, as positives are defined as the rare class.

`positive_share` A numeric value capturing the positive share (number of predicted positives out of all observations) in the population from which the test set should be drawn. Argument 'positive_share' can be left blank, but then 'k' must specified. Typically, it should be below 0.5, as positives are defined as the rare class.

`external_k` A numeric value capturing the imbalance ratio (number of predicted positives over predicted negatives) in the population on which the recall estimate is based. This only needs to be entered if 'recall' is based on a population that has a different imbalance than the test's population. The argument is only required if 'pi0' is not entered directly. If left blank, it will be calculated internally using 'external_positive_share', if this is entered, or assumed to be equal to the test population's imbalance. Typically, it should be below 1, as positives are defined as the rare class.

`external_positive_share`
A numeric value capturing the positive share (number of predicted positives out of all observations) in the population on which the recall estimate is based. This only needs to be entered if 'recall' is based on a population that has a different imbalance than the test's population. The argument is only required if 'pi0' is not entered directly. If left blank, it will be calculated internally using 'external_positive_share', if this is entered, or assumed to be equal to the test population's imbalance. Typically, it should be below 0.5, as positives are defined as the rare class.

`weight_se_f1` A positive numeric value capturing how much to weigh the SE for the F1 score in the objective function to be minimized by the calculator. The default for 'weight_se_f1' is '1'.

`weight_se_rec` A positive numeric value capturing how much to weigh the SE for recall in the objective function to be minimized by the calculator. The default for 'weight_se_rec' is '0'.

`weight_se_prec` A positive numeric value capturing how much to weigh the SE for precision in the objective function to be minimized by the calculator. The default for 'weight_se_prec' is '0'.

Details

This function can be used to estimate how many positive observations should be included in the test set under efficient stratified random sampling (SRS). This is the value that minimizes an objective function subject to a sample size constraint. The objective function is a linear combination of the standard errors (SEs) of precision, recall and F1 score, where weights can be specified by the user. These standard errors (SEs) are based on analytic formulas for the variance of the sampling distribution of the test set estimates of F1, precision and recall under two-bin stratified sampling. Such variances are based on expected parameter values, which need to be entered by the user. The value returns a unique integer value that minimizes the expression subject to the constraint. The following parameters need to be entered to receive an output: (i) pi1 (precision, TP/(TP+FP)), (ii) pi0 (FN/(FN+TN)) or recall (FN/(FN+TN)), (iii) imbalance ratio k ((TP+FP)/(TN+FN)) or positive share ((TP+FP)/(TN+FN+TP+FP)). These parameters should be guessed or estimated on other data (pi0, pi1, recall), or based on the predicted labels in the population (k, positive share). Note that if recall is entered instead of pi0, then it is advised to also enter the imbalance ratio or positive share of the dataset on which the recall guess is based (external_k or external_positive_share): if this is not done, the imbalance in the test data (k or positive_share) is used instead.

Value

A numeric value capturing how many positive observations should be sampled given the parameter values and the objective function specified by the user.

Examples

```
## example 1: how many out of 500 sampled obs should be positive if our goal is
## minimizing the SE of F1? The answer here assumes that we expect precision of
## 0.4, pi0 of 0.05, and 20% of positives
optimal_n_positives(N_sample = 500, pi1=0.4, pi0=0.05, positive_share=0.2,
                    weight_se_f1=1, weight_se_rec=0, weight_se_prec=0)
## example 2: same but now we assume recall of 0.6, and that this was estimated on a
## sample with 0.4 of positives
optimal_n_positives(N_sample = 500, pi1=0.4, recall=0.6, positive_share=0.2,
              external_positive_share=0.2, weight_se_f1=1, weight_se_rec=0, weight_se_prec=0)
## example 3: same as in example 1, but in the sample there is 1 positive for
## every 2 negatives (k=1/2)
optimal_n_positives(N_sample = 500, pi1=0.4, recall=0.6, k=0.5, external_positive_share=0.2,
                    weight_se_f1=1, weight_se_rec=0, weight_se_prec=0)
## example 4: same as in example 1, but we aim to minimize SE(F1)+0.5 SE(recall)+0.5 precision
optimal_n_positives(N_sample = 500, pi1=0.4, pi0=0.05, positive_share=0.2,
                    external_positive_share=0.2, weight_se_f1=1,
                    weight_se_rec=0.5, weight_se_prec=0.5)
```

---

pop_df *Example data of a population from which to draw a test set*

Description

Fake data where each row corresponds to some observation in the population from which the test set is to be sampled, and for which we know its predicted probability of exhibiting the outcome of interest according to some classifier. This dataset is meant to be used for examples and illustrations of the functions in the package.

Usage

```
data(pop_df)
```

Format

A data frame with 20000 rows and 2 variables:

id Identifier of each sampled observation

score Predicted probaility of exhibiting the outcome

Examples

```
data(pop_df)
```

proportional_allocation *Estimate proportional allocation*

Description

This function shows how many observations to sample per stratum under proportional allocation

Usage

```
proportional_allocation(data, N_sample, strata, min_per_bin = 1)
```

Arguments

data	A data.frame from which to sample observations. One column should contain the strata used for sampling.
N_sample	A numeric value capturing the desired sample size.
strata	A character value capturing the column name of the sampling strata. Column should contain factor or character values.
min_per_bin	A numeric value capturing the minimum number of observations per bin. Default is 1.

Details

This function can be used to determine how many observations to sample per stratum under proportional allocation. It first divides the desired sample size proportionally to each stratum's size, then rounds the allocated observations per bin, and lastly adjusts them so the overall sample size is as close to the target as possible. NAs in the strata variable will be coded as a stratum themselves. Note the strata can be based on any information: either binned predicted probabilities (quantile- or fixed-intervals) or any other categorical information.

Value

A named vector whose values indicate the number of observations to be sampled per each stratum, and whose names correspond to the values in the 'strata' variable of the input dataset.

Examples

```
## say that we want to sample a number of observations proportional to the
## stratum's prevalence in the population, but with appropriate rounding given
## one cannot sample fractional observations. this function tells us how many
## obs per stratum to sample, given the strata
data(pop_df)
pop_df$strata <- cut(pop_df$score, breaks=c(0,0.2,0.4,0.6,0.8,1))
proportional_allocation(data=pop_df, N_sample=375, strata="strata", min_per_bin=1)
## note the strata can be based on any information: either predicted probabilities
## (quantile- or fixed-intervals) or any other categorical information
## NAs in the strata variable will be coded as a separate stratum
## for instance, the following code will lead to some NA in strata
## and these get grouped into a separate stratum
pop_df$strata <- cut(pop_df$score, breaks=c(0,0.2,0.4,0.6,0.8,0.9))
proportional_allocation(data=pop_df, N_sample=375, strata="strata", min_per_bin=1)
```

sample_df *Example data of a test set*

Description

Fake data with where each row corresponds to some observation in the annotated test set, for which we know its stratum, sampling probability and true label. This dataset is meant to be used for examples and illustrations of the functions in the package.

Usage

```
data(sample_df)
```

Format

A data frame with 1394 rows and 5 variables:

id Identifier of each sampled observation

score Predicted probaility of exhibiting the outcome

strata Stratum of the sampled observation (based on score)

Prob Sampling probability

truth Annotated labels: whether the observation actually exhibits the outcome

Examples

```
data(sample_df)
```

se_srs *Estimate standard errors for simple random sample*

Description

This function analytically estimates SEs from confusion matrix drawn by simple random sampling

Usage

```
se_srs(tr0_pred0, tr1_pred0, tr0_pred1, tr1_pred1, N_sample)
```

Arguments

tr0_pred0	A numeric value capturing the share of true negatives.
tr1_pred0	A numeric value capturing the share of false negatives.
tr0_pred1	A numeric value capturing the share of false positives.
tr1_pred1	A numeric value capturing the share of true positives.
N_sample	A numeric value capturing the total number size of the test set.

Details

This function takes a confusion matrix and sample size as input, and produces SEs with Wilson method for the F1 score, precision and recall. These SEs are based on assuming the test set was drawn by simple random sampling: the SEs should not be trusted if sampling was conducted in a different way. Note that the sum of the proportion of observations across all four cells must be one. If you want to get CIs rather than just SEs, use 'ci_srs_wilson' or bootstrapping (e.g. with 'stratified_metrics'): under SRS, these are better than Wald-type CIs. To estimate SEs from a test set data rather than from a confusion matrix, use 'stratified_metrics'.

Value

A data.frame containing the analytic SEs for the F1 score, precision and recall. These SEs are based on assuming the test set was drawn by simple random sampling: the SEs should not be trusted if sampling was conducted in a different way.

References

Flores, P., M. Salicru, A. Sanchez-Pla, and J. Ocaña (2022). An equivalence test between features lists, based on the sorensen–dice index and the joint frequencies of go term enrichment. BMC bioinformatics 23 (1), 207.

Lam, K. F. Y. (2023). Confidence intervals for the f1 score: A comparison of four methods. arXiv preprint arXiv:2309.14621

Takahashi, K., K. Yamamoto, A. Kuchiba, and T. Koyama (2022). Confidence interval for micro-averaged f 1 and macro-averaged f 1 scores. Applied Intelligence 52 (5), 4961–4972.

Tomas-Valiente, F. (2025). Uncertain performance: How to quantify uncertainty and draw test sets when evaluating classifiers.

See Also

`ci_srs_wilson` `stratified_metrics`

Examples

```
## enter the estimates for the confusion matrix cells and sample size
## get the SE analytically under assumption that test set is a SRS
se_srs(tr0_pred0 = 0.5, tr1_pred0 = 0.1, tr0_pred1 = 0.1, tr1_pred1 = 0.3, N_sample = 1000)
```

---

se_strat *Estimate standard errors for stratified random sample*

Description

This function analytically estimates SEs from confusion matrix drawn by simple random sampling

Usage

```
se_strat(tr0_pred0, tr1_pred0, tr0_pred1, tr1_pred1, N_positive, N_negative)
```

Arguments

tr0_pred0	A numeric value capturing the share of true negatives.
tr1_pred0	A numeric value capturing the share of false negatives.
tr0_pred1	A numeric value capturing the share of false positives.
tr1_pred1	A numeric value capturing the share of true positives.
N_positive	A numeric value capturing the number of positive observations sampled into the test set.
N_negative	A numeric value capturing the number of negative observations sampled into the test set.

Details

This function takes a confusion matrix and sample size as input, and produces SEs with Wilson method for the F1 score, precision and recall. These SEs are based on assuming two-bin stratified random sampling, where the two bins are positive and negative observations: the SEs should not be trusted if sampling was conducted in a different way (either not by two-bin positives vs negatives stratification, or by stratifying on some other variable). Note that the sum of the proportion of observations across all four cells must be one. If you want to get CIs rather than just SEs and your sample is stratified, it is advised to use bootstrapping (e.g. with 'stratified_metrics') rather than Wald-type CIs. To estimate SEs from a test set data rather than from a confusion matrix, use 'stratified_metrics'.

Value

A data.frame containing the analytic SEs for the F1 score, precision and recall under two-bin stratification of positives and negatives. These SEs are not valid if the test set was not drawn through stratified sampling (e.g SRS) or if it was drawn through some other stratification regime (either not by two-bin positives vs negatives stratification, or by stratifying on some other variable).

References

Shang, H., J.-M. Langlois, K. Tsioutsiouliklis, and C. Kang (2023). Precision/recall on imbalanced test data. International Conference on Artificial Intelligence and Statistics PMLR: 9879–9891.

Tomas-Valiente, F. (2025). Uncertain performance: How to quantify uncertainty and draw test sets when evaluating classifiers.

See Also

stratified_metrics

Examples

```
## enter the estimates for the test set confusion matrix cells and sample sizes
## get the SE analytically under assumption that test set is a two-bin
## stratified sample (positives and negatives)
se_strat(tr0_pred0 = 0.5, tr1_pred0 = 0.1, tr0_pred1 = 0.1, tr1_pred1 = 0.3,
         N_positive = 600, N_negative = 400)
```

`stratified_metrics` *Estimate performance metrics from test set*

Description

This function estimates performance metrics given an annotated test set

Usage

```
stratified_metrics(
  data,
  truth,
  pred,
  probs = NULL,
  threshold,
  se = TRUE,
  bs = NULL,
  strata,
  seed = 1234,
  alpha = 0.05
)
```

Arguments

`data`	A data.frame containing the annotated test set.
`truth`	A character value capturing the column name of the annotated labels. Column should capture whether each observation exhibits the outcome (1/TRUE) or not (0/FALSE).
`pred`	A character value capturing the column name of the classifier's predictions. Column should capture each observation's predicted label (either numeric or logical), or the predicted probability of exhibiting the outcome (numeric). Default is NULL, which assumes equal sampling probability across observations.
`probs`	A character value capturing the column name of the sampling probabilities.
`threshold`	A numeric value capturing the threshold for binary classification.
`se`	A logical value whether SEs should be calculated. If all observations have the same sampling probability, the function calls 'se_srs' to estimate SEs using the analytical formula. In this case, Wilson CI lower and upper bounds for precision, recall and F1 score are also outputted. If observations do not have the same sampling probability, then the function calls 'se_strat' to compute stratified SEs: this assumes that the stratification was done on the basis of positives and negatives (if proportional stratification was used within the positives and negatives separately, the SEs are conservative). If stratification was done on some other variable, then 'se' should be set to FALSE, and bootstrapping should be used instead.
`bs`	A numeric value indicating the number of bootstrapping iterations used to compute confidence intervals. If 'NULL', then bootstrapping is not implemented. For stratified sampling, bootstrapping is stratified by stratum too.

strata	A character value capturing the column name of the sampling strata. Column should contain factor or character values. This only needs to be entered if 'bs' is not 'NULL' and the sampling was stratified (sampling probabilities are not constant across observations).
seed	A numeric value used as seed for the bootstrapping.
alpha	A numeric value used capturing the type-I error rate for constructing confidence intervals (applies to both bootstrapping and analytic CIs).

Details

This function can be used on a test set drawn from simple random sampling, or from stratified sampling if sampling probabilities are provided. The function unbiasedly estimates the proportion of observations in each cell of the test set confusion matrix, using stratified estimators if needed, and then produces key metrics based on them by internally calling do_metrics. The input data should contain a column capturing whether each observation exhibits the outcome (1/TRUE) or not (0/FALSE), and a column capturing the predicted label (either predicted probability, or binary label in either numeric or logical format). The following metrics are supported: precision, recall, F1 of positive class, accuracy, precision of negative class, recall of negative class, F1 of negative class, MCC (Matthews correlation coefficient), Cohen's kappa, (unweighted) macro-averaged F1 score, weighted F1 score, BM (bookmaker's informedness). The output also shows the proportions of true negatives, false negatives, false positives and true positives.

Value

A data.frame of performance metrics. The following metrics are supported: precision, recall, F1 of positive class, accuracy, precision of negative class, recall of negative class, F1 of negative class, MCC (Matthews correlation coefficient), Cohen's kappa, (unweighted) macro-averaged F1 score, weighted F1 score, BM (bookmaker's informedness). The output also shows the proportions of true negatives, false negatives, false positives and true positives. If 'bs' was specified, then the resulting data.frame also produces the bootstrapped SEs and CIs (columns ending in 'boot_se','boot_lwr', 'boot_upr'): these are based on stratified bootstrapping if the test set was drawn through stratified sampling. If 'se' was set to 'TRUE', analytic SEs are included in the output data.frame too for precision, recall and F1 score (columns 'f1_se', 'recall_se', 'precision_se'). These SEs are valid for SRS and two-bin efficient stratification by positives vs negatives. If proportional stratification was used when samplingpositives and negatives, the analytic SEs are conservative. If other stratification regime was used, these SEs should not be trusted and bootstrapped SEs/CIs should be considered instead.

See Also

do_metrics, se_srs, se_strat

Examples

```
## example 1: compute metrics after stratified sampling using probability weights
data(sample_df)
stratified_metrics(data = sample_df, truth = "truth", pred = "score", probs="Prob",
                   threshold = 0.5, strata = "strata", se=FALSE)
## example 2: same as example 1 but with bootstrapped SEs/CIs
data(sample_df)
stratified_metrics(data = sample_df, truth = "truth", pred = "score", probs="Prob",
                   threshold = 0.5, strata = "strata", se=FALSE, bs=200)
## example 3: same as example 1 but assuming the test set is a SRS from the
## population. since it's a SRS, analytic SEs and CIs are valid and computed
```

```
data(sample_df)
sample_df$Prob <- 1
stratified_metrics(data = sample_df, truth = "truth", pred = "score", probs="Prob",
                   threshold = 0.5, strata = "strata", se=TRUE)
## alternatively, can achieve the same with
stratified_metrics(data = sample_df, truth = "truth", pred = "score", probs=NULL,
                   threshold = 0.5, strata = "strata", se=TRUE)
## can additionally get bootstrapped SEs and CIs with
stratified_metrics(data = sample_df, truth = "truth", pred = "score", probs=NULL,
                   threshold = 0.5, strata = "strata", se=TRUE, bs=200)
## example 4: same as example 1 but computing SEs analytically.
## this assumes that the test set is based on positive-negative two-bin
## stratification. this provides conservative inference if the test set is drawn
## by optimal stratification (first picking the number of positives and negatives
## as in two-bin stratification, then proportional stratification within positive
## and negative subsamples). the SEs are invalid if the stratification regime does
## not stratify on the predicted probability of the outcome.
stratified_metrics(data = sample_df, truth = "truth", pred = "score", probs="Prob",
                   threshold = 0.5, strata = "strata", se=TRUE)
```

`testsampler` *Implement stratified sampling*

Description

This function implements stratified sampling under constant, proportional or optimal allocation

Usage

```
testsampler(
  data,
  stratifying,
  N_sample = NA,
  na.rm = "stop",
  allocation = "optimal",
  threshold = 0.5,
  N_bins_left = 5,
  N_bins_right = 5,
  seed = 1234,
  min_per_bin = 1,
  manual_allocation = NULL,
  pi1 = NULL,
  pi0 = NULL,
  trained_model = NULL,
  recall = NULL,
  external_k = NULL,
  external_positive_share = NULL,
  n_positive = NULL,
  weight_se_f1 = 1,
  weight_se_rec = 0,
  weight_se_prec = 0
)
```

Arguments

data	A data.frame from which to sample observations. One column should contain the stratifying variable used to construct the strata for sampling.
stratifying	A character value capturing the column name of the stratifying variable. This column in 'data' should contain continuous numeric values between 0 and 1 (predicted probability of exhibiting the outcome), or else character labels corresponding to the strata. If the variable captures predicted probabilities, the positive class should typically be defined as the rare class. If user wants to rely on custom strata, enter the name of character variable containing the strata. If 'allocation' is 'optimal', the variable indicated in 'stratifying' should contain continuous outcome probability: it should not be categorical.
N_sample	A numeric value capturing the desired sample size. Not needed if allocation is set to 'manual'.
na.rm	A character value indicating how to deal with NA values in the stratifying variable (indicated by 'stratifying'). Options include: 'drop' (this drops observations with NA in the stratifying variable), 'stop' (this ensures the function stops running if there are any NAs in the stratifying variable), 'impute' (this median-imputes the NAs in the stratifying variable) and encode (this turns the NAs into a stratum of their own; this option is not supported when using optimal). Default is 'stop'. When 'manual' allocation is selected, users should create a stratum for NAs manually before calling the function.
allocation	A character value capturing the type of allocation regime used for stratified sampling. Options include: 'constant', 'proportional', 'optimal' and 'manual'. If set to 'constant', the function calls constant_allocation, and implements constant allocation, aiming for all strata to get the same number of observations in the test set. If set to 'proportional', the function calls proportional_allocation, and implements proportional allocation, aiming for each stratum to get a proportional number of observations in the test set relative to the stratum's number of observations in the population. If set to 'optimal', the function attempts to call optimal_allocation (only if parameter values support this), and implements efficient two-bin allocation, determining how many positives and negatives to optimally sample (or taking this from the user-provided 'n_positive') and then performing proportional allocation in each stratum. If set to 'manual', the function samples as many observations per bin as indicated in the 'manual_allocation' argument.
threshold	A numeric value capturing the threshold for binary classification. Not needed if allocation is set to 'manual'.
N_bins_left	A numeric value capturing how many bins there should between zero and 'threshold' (closed and open respectively). Not needed if allocation is set to 'manual'.
N_bins_right	A numeric value capturing how many bins there should between 'threshold' and 1 (both closed). Not needed if allocation is set to 'manual'.
seed	A numeric value that sets seed internally for sampling.
min_per_bin	A numeric value capturing the minimum number of observations per bin. Default is 1. Not needed if allocation is set to 'manual'.
manual_allocation	A named vector, whose elements are numeric, and whose names correspond to unique strata. This vector indicates how many observations to sample, allowing the user to specify a specific custom stratified allocation. This argument should only be provided when argument 'allocation' is set to 'manual'.

`pi1` A numeric value capturing the expected precision (share of cases exhibiting the outcome out of predicted positives). This value is then used to determine what the optimal allocation is. This argument only must be entered if 'allocation' is set to 'optimal' and no 'caret' model object is provided in 'trained_model'.

`pi0` A numeric value capturing the inverse of the expected precision for the negative class (share of cases exhibiting the outcome out of predicted negatives). This value is then used to determine what the optimal allocation is. This argument only must be entered if 'allocation' is set to 'optimal' and no 'caret' model object is provided in 'trained_model' and 'recall' is not entered.

`trained_model` A 'caret' model object, used to estimate 'pi1' and 'pi0' if these are not manually provided, and if simultaneously 'allocation' is set to 'manual'. The model can only be used if cross validation was used during training: then, excluded folds are used to guess 'pi1' and 'pi0'.

`recall` A numeric value capturing the expected value of recall (share of positive cases among those exhibiting the outcome). This argument only needs to be entered if 'allocation' is set to 'optimal' and 'pi0' is not entered and no 'caret' model object is provided in 'trained_model'

`external_k` A numeric value capturing the imbalance ratio (number of predicted positives over predicted negatives) in the population on which the recall estimate is based. This only needs to be entered if 'recall' is based on a population that has a different imbalance than the test's population. The argument is only required if 'pi0' is not entered directly and no 'caret' model object is provided in 'trained_model' and 'external_positive_share' is not entered, while allocation is set to 'optimal'. If defined, it should typically be below 1, as positives are defined as the rare class.

`external_positive_share` A numeric value capturing the positive share (number of predicted positives out of all observations) in the population on which the recall estimate is based. This only needs to be entered if 'recall' is based on a population that has a different imbalance than the test's population. The argument is only required if 'pi0' is not entered directly and no 'caret' model object is provided in 'trained_model' and 'external_k' is not entered, while allocation is set to 'optimal'. If defined, it should typically be below 0.5, as positives are defined as the rare class.

`n_positive` A numeric value indicating how many positive observations to include in the stratified sample. Positives and negatives are then sampled separately using stratified sampling with proportional allocation in the positive and negative subsamples respectively. Note 'n_positive' should be an integer, and it should only be entered if 'allocation' is set to 'optimal'.

`weight_se_f1` A positive numeric value capturing how much to weigh the SE for the F1 score in the objective function to be minimized for optimal allocation (see details in `optimal_allocation`). The default for 'weight_se_f1' is '1'. This should only be entered if 'allocation' is set to 'optimal'.

`weight_se_rec` A positive numeric value capturing how much to weigh the SE for recall in the objective function to be minimized for optimal allocation (see details in `optimal_allocation`). The default for 'weight_se_rec' is '0'. This should only be entered if 'allocation' is set to 'optimal'.

`weight_se_prec` A positive numeric value capturing how much to weigh the SE for precision in the objective function to be minimized for optimal allocation (see details in `optimal_allocation`). The default for 'weight_se_prec' is '0'. This should only be entered if 'allocation' is set to 'optimal'.

Details

This function can be used to implement the sampling process. It take as input the dataset from which to sample and produces a smaller sampled dataset as output, i.e. the test set. The function supports stratified sampling with constant allocation across bins, proportional allocation, optimal two-bin allocation (with the optimal number of positives for minimizing some linear combination of the SEs of precision, recall and F1 score), and custom allocation to be manually inputed. The function also allows users to flexibly select binning strategy, by specifying how many bins they want to each side of the predicted probability threshold. If users want to stratify on the basis of some variable other than the predicted probability, they can construct the bins manually before using the function and then enter a character variable as 'stratifying' (this supports all types of allocation except optimal). The output dataset contains the same columns as the input plus: sampling indices, strata bins, and sampling probabilities which will be used after annotation for constructing stratified estimates of performance metrics. For optimal allocation, the following parameters need to be entered: (i) pi1 (precision, TP/(TP+FP)), (ii) pi0 (FN/(FN+TN)) or recall (FN/(FN+TN)). These parameters should be guessed or estimated on other data (pi0, pi1, recall). Note that if recall is entered instead of pi0, then it is advised to also enter the imbalance ratio or positive share of the dataset on which the recall guess is based (external_k or external_positive_share): if this is not done, the imbalance in the test data is estimated by the function and used instead.

Value

A data.frame object that contains the test set sample to be annotated. This data.frame contains same columns as the original data.frame inputted in the 'data' argument, but contains fewer rows since it is just a sample. Additionally it contains the following column: 'stratifying' (numeric variable with the values of the stratifying variable if the input corresponded to a numeric stratifying variable; character variable indicating the strata if this was the input), 'id_sampling' (variable indicating which rows were included in the sample), 'strata' (character variable indicating what stratum each sampled observation belongs to), and 'Prob' (numeric variable indicating the sampling probability for any given observation in the test sample).

References

Tomas-Valiente, F. (2025). Uncertain performance: How to quantify uncertainty and draw test sets when evaluating classifiers.

Examples

```
## example 1: draw a test set of 1000 observations, by stratified sampling on
## predicted probability bins, with 5 bins left of 0.5 threshold and 5 bins right
## of this threshold with constant allocation. The function also supports different
## thresholds and different number of bins to either side of cutoff (including
## asymmetric bin numbers), but 5-5 is a reasonable default for small samples:
## if bigger sample, can have more bins. threshold should be the value at which
## one starts to deem an observation positive (here 0.5, but alternative values
## are possible).
data(pop_df)
out <- testsampler(data=pop_df, stratifying='score', N_sample=1000, allocation='constant',
                   threshold=0.5, N_bins_left=5, N_bins_right=5)
## example 2: draw a test set of 1000 observations, by stratified sampling on
## predicted probability bins, with 5 bins left of 0.5 threshold and 5 bins right
## of this threshold with proportional allocation.
data(pop_df)
out <- testsampler(data=pop_df, stratifying='score', N_sample=1000, allocation='proportional',
                   threshold=0.5, N_bins_left=5, N_bins_right=5)
```

```
## example 3: draw a test set of 1000 observations, by stratified sampling on
## predicted probability bins, with 5 bins left of 0.5 threshold and 5 bins right
## of this threshold with optimal allocation. first, optimal number of positives and negatives
## is determined, and then proportional allocation within positives/negatives. assume we're
## minimizing the objective function SE(F1) + 0.5 SE(precision) + 0.5 SE(recall).
## as assumptions, we have guesses for pi1 (TP/TP+FP) and pi0 (FN/FN+TN) of 0.4 and 0.02.
data(pop_df)
out <- testsampler(data=pop_df, stratifying='score', N_sample=1000, threshold=0.5,
                   N_bins_left=5, N_bins_right=5, pi1=0.4, pi0=0.2, weight_se_f1=1,
                   weight_se_rec=0.5, weight_se_prec=0.5)
## example 4: like example 3, but instead we only care about the SE of the f1.
## Plus, we want simple two-bin positive-negative stratification (0.5 threshold).
data(pop_df)
out <- testsampler(data=pop_df, stratifying='score', N_sample=1000, threshold=0.5,
                   N_bins_left=1, N_bins_right=1, pi1=0.4, pi0=0.2, weight_se_f1=1,
                   weight_se_rec=0, weight_se_prec=0)
## example 5: like example 3, but assuming we know recall was 0.6 in some other
## dataset where the imbalance ratio had 1 positive for every 2 negatives so
## external_k is 1/2=0.5
data(pop_df)
out <- testsampler(data=pop_df, stratifying='score', N_sample=1000, threshold=0.5,
                   N_bins_left=5, N_bins_right=5, pi1=0.4, recall=0.6, external_k=0.5,
                   weight_se_f1=1, weight_se_rec=0.5, weight_se_prec=0.5)
## example 6: we decide manually how many positives and negatives to sample (500
## each) and then sample with proportional allocation with the positive and negative
## strata (5 bins in each).
data(pop_df)
out <- testsampler(data=pop_df, stratifying='score', N_sample=1000, threshold=0.5,
                   N_bins_left=5, N_bins_right=5, n_positive=500)
## example 7: like example 1, we use constant allocation stratification, but on
## some non-numeric categorical variable. therefore, no threshold needed
data(pop_df)
pop_df$somevar <- as.character(rbinom(nrow(pop_df), 2, 0.4)) #create random stratifying variable
out <- testsampler(data=pop_df, stratifying='somevar', allocation='constant',
                   N_sample=1000, N_bins_left=5, N_bins_right=5)
## example 8: like example 7, we stratify the sampling on some custom categorical
## variable, but specifying a custom allocation
data(pop_df)
pop_df$somevar <- as.character(rbinom(nrow(pop_df), 2, 0.4)) #create stratifying variable
custom <- c(200, 200, 600)
names(custom) <- c("0", "1", "2") # these names need to match the values of somevar
out <- testsampler(data=pop_df, stratifying='somevar', allocation='manual',
                   manual_allocation=custom)
## example 9: optimal stratification with categorical labels is only possible if
## each stratum contains only positives or only negatives. This might happen when
## you want to create custom bins, e.g. bins that are not to a constant distance
## to each other. You can implement this using manual allocation in the following
## way, assuming pi1=0.4 and pi0=0.2.
data(pop_df)
# stratify in custom bins
pop_df$custom_bins <- cut(pop_df$score, breaks=c(0, 0.1, 0.4, 0.5, 0.6, 0.8, 0.9, 1))
# determine optimal allocation
custom <- optimal_allocation(data = pop_df, N_sample = 500, strata = 'custom_bins',
                             stratifying = 'score', min_per_bin=1, threshold=0.5,
                             pi1=0.4, pi0=0.2, weight_se_f1=1, weight_se_rec=0,
                             weight_se_prec=0)
# draw test set with manual allocation
```

```
out <- testsampler(data=pop_df, stratifying='custom_bins', allocation='manual',
                   manual_allocation=custom)
```

`test_samplesize` *Sample size calculator*

Description

This function estimates how many observations need to be sampled to achieve a desired standard error.

Usage

```
test_samplesize(
  se_f1 = NULL,
  se_precision = NULL,
  se_recall = NULL,
  pi1 = NULL,
  pi0 = NULL,
  recall = NULL,
  k = NULL,
  positive_share = NULL,
  external_k = NULL,
  external_positive_share = NULL,
  max_N = 10000,
  min_N = 1,
  by_N = 1,
  weight_se_f1 = 1,
  weight_se_prec = 1,
  weight_se_rec = 1
)
```

Arguments

`se_f1`	A numeric value capturing the desired standard error for the F1 score of the positive class. If non-numeric, then the sample size calculation ignores the SE of F1. The sample size calculator selects a sample size such that the SE of the F1 score is below 'se_f1'.
`se_precision`	A numeric value capturing the desired standard error for precision of the positive class. If non-numeric, then the sample size calculation ignores the SE of precision. The sample size calculator selects a sample size such that the SE of precision is below 'se_precision'.
`se_recall`	A numeric value capturing the desired standard error for recall of the positive class. If non-numeric, then the sample size calculation ignores the SE of recall. The sample size calculator selects a sample size such that the SE of recall is below 'se_recall'.
`pi1`	A numeric value capturing the expected precision (share of cases exhibiting the outcome out of predicted positives). This value must be entered for the function to produce an output.

pi0 A numeric value capturing the inverse of the expected precision for the negative class (share of cases exhibiting the outcome out of predicted negatives). If left blank, the function computes 'pi0' from recall and imbalance. Note this value need not be entered if 'se_recall' and 'se_f1' are left blank.

recall A numeric value capturing the expected value of recall (share of positive cases among those exhibiting the outcome). This value only needs to be entered if 'se_recall' or 'se_f1' are specified but 'pi0' is left blank.

k A numeric value capturing the imbalance ratio (number of predicted positives over predicted negatives) in the population from which the test set should be drawn. Argument 'k' can be left blank, but then 'positive_share' must be specified. Typically, it should be below 1, as positives are defined as the rare class.

positive_share A numeric value capturing the positive share (number of predicted positives out of all observations) in the population from which the test set should be drawn. Argument 'positive_share' can be left blank, but then 'k' must specified. Typically, it should be below 0.5, as positives are defined as the rare class.

external_k A numeric value capturing the imbalance ratio (number of predicted positives over predicted negatives) in the population on which the recall estimate is based. This only needs to be entered if 'recall' is based on a population that has a different imbalance than the test's population. The argument is only required if 'pi0' is not entered directly. If left blank, it will be calculated internally using 'external_positive_share', if this is entered, or assumed to be equal to the test population's imbalance 'k'. Typically, it should be below 1, as positives are defined as the rare class.

external_positive_share A numeric value capturing the positive share (number of predicted positives out of all observations) in the population on which the recall estimate is based. This only needs to be entered if 'recall' is based on a population that has a different imbalance than the test's population. The argument is only required if 'pi0' is not entered directly. If left blank, it will be calculated internally using 'external_positive_share', if this is entered, or assumed to be equal to the test population's imbalance. Typically, it should be below 0.5, as positives are defined as the rare class.

max_N A numeric value capturing the maximum sample size to be considered. Default of 10000, should be increased if the desired SE could not be achieved with under 10000 observations.

min_N A numeric value capturing the minimum sample size to be considered. Default is '1'.

by_N A numeric value capturing the size of the intervals between 'min_N' and 'max_N' to be considered as possible sample sizes. Default is '1'.

weight_se_f1 A positive numeric value capturing how much to weigh the SE for the F1 score. This only applies when there are multiple values of 'n_positive' that verify the desired SE conditions for a given sample size under stratified sampling. When this happens, the value of 'n_positive' is selected by picking the number that minimizes the sum of the SE for F1 score, precision and recall, weighted by 'weight_se_f1', 'weight_se_precision' and , 'weight_se_recall'. The default for 'weight_se_f1' is '1'.

weight_se_prec A positive numeric value capturing how much to weigh the SE for precision. This only applies when there are multiple values of 'n_positive' that verify the desired SE conditions for a given sample size under stratified sampling. When this happens, the value of 'n_positive' is selected by picking the number

that minimizes the sum of the SE for F1 score, precision and recall, weighted by ‘weight_se_f1‘, ‘weight_se_prec‘ and , ‘weight_se_rec‘. The default for ‘weight_se_prec‘ is ‘1‘.

`weight_se_rec` A positive numeric value capturing how much to weigh the SE for recall. This only applies when there are multiple values of ‘n_positive‘ that verify the desired SE conditions for a given sample size under stratified sampling. When this happens, the value of ‘n_positive‘ is selected by picking the number that minimizes the sum of the SE for F1 score, precision and recall, weighted by ‘weight_se_f1‘, ‘weight_se_prec‘ and , ‘weight_se_rec‘. The default for ‘weight_se_rec‘ is ‘1‘.

Details

This function can be used to estimate how many observations need to be included in the test set under simple random sampling (SRS) or under efficient stratified sampling. The standard errors (SEs) are based on analytic formulas for the variance of the sampling distribution of the test set estimates of F1, precision and recall. Such variances are based on expected parameter values, which need to be entered by the user. Given parameter values, the function looks at what sample sizes up to ‘max_N‘ verify that the SEs for F1, precision and recall are below the maximum desired SE, which need to be entered by the user. Under SRS, the function returns the minimum sample size that verifies the desired SEs conditions, as well as the SEs under such sample size. Under stratified sampling, the function returns the minimum sample size such that the desired SE conditions can be verified for some allocation (i.e. some number of positives to be sampled): this is based on two-bin stratified sampling (sampling positives and negatives separately), and the resulting sample calculation is not valid if stratifying on some other variable. Under stratified sampling, the function also returns how to allocate the sample size between positives and negatives to verify the desired SEs conditions for the given sample size. When there are multiple values of ‘n_positive‘ that verify the desired SE conditions for a given sample size, the value of ‘n_positive‘ is selected by picking the number that minimizes the sum of the SE for F1 score, precision and recall, weighted by some user-provided weights. The following parameters need to be entered to receive an output: (i) pi1 (precision, TP/(TP+FP)), (ii) pi0 (FN/(FN+TN)) or recall (FN/(FN+TN)), (iii) imbalance ratio k ((TP+FP)/(TN+FN)) or positive share ((TP+FP)/(TN+FN+TP+FP)). These parameters should be guessed or estimated on other data (pi0, pi1, recall), or based on the predicted labels in the population (k, positive share). Note that if recall is entered instead of pi0, then it is advised to also enter the imbalance ratio or positive share of the dataset on which the recall guess is based (external_k or external_positive_share): if this is not done, the imbalance in the test data (k or positive_share) is used instead.

Value

A list of two data.frame objects (‘srs‘ and ‘stratified‘), which contain the result of the power calculation under SRS and stratified sampling respectively. These data.frames contain the minimum number of observations (‘sample_size_srs‘ and ‘sample_size_strat‘) needed to verify the desired SEs conditions (i.e. smaller than ‘se_f1‘, ‘se_precision‘ and ‘se_recall‘). For stratified sampling, the output also contains how many positive observations (‘n_positives‘) need to be sampled to achieve the desired SEs. Under either sampling, the output also indicates what the SE would be for each metric and sampling stratgy, given the selected sample size. When the maximum sample size (‘max_N‘) is too small to achieve the desired SEs, the output indicates this instead.

References

Tomas-Valiente, F. (2025). Uncertain performance: How to quantify uncertainty and draw test sets when evaluating classifiers.

Examples

```
## example 1: required sample size to get SE of F1 below 3pp
## assuming we have guesses for pi1 (TP/TP+FP) and pi0 (FN/FN+TN) of 0.4 and 0.02
## assuming we know in the test set 10% of observations are positive
## further assuming that when multiple allocations exist, we only care about F1
test_samplesize(se_f1=0.03, pi1=0.4, pi0=0.02, positive_share=0.1,
                weight_se_f1=1, weight_se_prec=0, weight_se_rec=0)
## example 2: same but assuming we know test set has 1 positive for every 2
## negatives so k is 1/2=0.5
test_samplesize(se_f1=0.03, pi1=0.4, pi0=0.02, k=0.5,
                weight_se_f1=1, weight_se_prec=0, weight_se_rec=0)
## example 3: now we want SE for F1 to be up to 3pp and SE for recall to be up to 5pp
## and precision's SE is below 5pp. plus, assume in case multiple allocations verify this,
## we want the allocation minimizing SE(F1)+0.5 SE(recall)+0.5 SE(precision).
## assumptions about pi1, pi0 and k are as in example 2.
test_samplesize(se_f1=0.03, se_precision=0.05, se_recall=0.05,
                pi1=0.4, pi0=0.02, k=0.5,
                weight_se_f1=1, weight_se_prec=0.5, weight_se_rec=0.5)
## example 4: same as example 2 but we do not have a guess for pi0. however, we
## know the classifier had recall of 0.6 in a dataset where 20% of observations
## were positive
test_samplesize(se_f1=0.03, pi1=0.4, k=0.5,
                recall=0.6, external_positive_share=0.2,
                weight_se_f1=1, weight_se_prec=0, weight_se_rec=0)
## example 5: same as example 4 but we do not know the imbalance on the dataset
## where recall was estimated, so we assume it to be same as in our evaluation
## dataset
test_samplesize(se_f1=0.03, pi1=0.4, k=0.5, recall=0.6,
                weight_se_f1=1, weight_se_prec=0, weight_se_rec=0)
## example 6: same as example 1 but we only care about precision and want
## to get its SE below 3pp. then we do not need to specify pi0, recall,
## external_k or external_positive_share
test_samplesize(se_precision=0.03, pi1=0.4, k=0.5,
                weight_se_f1=0, weight_se_prec=1, weight_se_rec=0)
```

`weight_fixer` *Fix weights after incomplete annotations*

Description

This function estimates probability weights given a partially annotated test set

Usage

```
weight_fixer(data, truth = "truth", strata = "strata", probs = NULL)
```

Arguments

`data`	A data.frame containing the annotated test set.
`truth`	A character value capturing the column name of the annotated labels. NAs should be entered for observations that were not annotated.
`strata`	A character value capturing the column name of the sampling strata. Column should contain factor or character values.

probs — A character value capturing the column name of the sampling probabilities. Default assumes equal sampling probability for all observations in the original sampling.

Details

This function can be used on a test set drawn by stratified sampling that has not been fully annotated, e.g. if human annotators stopped annotating halfway through the annotation task. The function re-computes sampling probabilities given the annotation was incomplete. This assumes that the comments that were indeed annotated from each stratum are a SRS of the stratum.

Value

A data.frame just like the input 'data', but with recomputed sampling probabilities in column 'Prob_new'.

Examples

```
## assume that some observations were not annotated, such that their value in
## the column with annotated labels is NA. function will update the weights so
## estimates remain unbiased. Note this assumes that the annotated observations
## are a SRS of their respective stratum if the test set was drawn by stratified
## sampling, or a SRS of the entire population if the test set was drawn by SRS.
data(sample_df)
sample_df$truth[30:45] <- NA #code some labels as missing because not annotated
fixed_df <- weight_fixer(data = sample_df, truth = "truth", strata = "strata", probs="Prob")
```

Index