Package: confcons (via r-universe)

September 12, 2024

Type Package

Title Confidence and Consistency of Predictive Distribution Models

Version 0.3.1.9000

Date 2024-05-15

Description Calculate confidence and consistency that measure the goodness-of-fit and transferability of predictive/potential distribution models (including species distribution models) as described by Somodi & Bede-Fazekas et al. (2024) <doi:10.1016/j.ecolmodel.2024.110667>.

License GPL (>= 3)

Encoding UTF-8

LazyData true

URL https://github.com/bfakos/confcons, https://bfakos.github.io/confcons/

BugReports https://github.com/bfakos/confcons/issues

RoxygenNote 7.3.0

Suggests knitr, rmarkdown, testthat (>= 3.0.0), mockery, vctrs, withr, ROCR, covr, terra, sf, blockCV (>= 3.1-3), ggplot2, ranger, ecospat, ENMeval

VignetteBuilder knitr

Config/testthat/edition 3

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

RemoteUrl https://github.com/bfakos/confcons

RemoteRef HEAD

RemoteSha 7047949479b72e79043ef864233c4b50961eb436

Contents

`confidence` *Confidence of the predictive distribution model*

Description

Calculate the confidence in positive predictions within known presences (CPP, `type = "positive"`) or confidence in predictions within known presences (CP, `type = "neutral"`) based on the occurrence `observations`, the `predictions` of the probability of occurrence, and the two `thresholds` distinguishing certain negatives/positives from uncertain predictions.

Usage

```
confidence(
  observations,
  predictions,
  thresholds = confcons::thresholds(observations = observations, predictions =
    predictions),
  type = "positive"
)
```

Arguments

`observations`	Either an integer or logical vector containing the binary observations where presences are encoded as `1`s/`TRUE`s and absences as `0`s/`FALSE`s.
`predictions`	A numeric vector containing the predicted probabilities of occurrence typically within the `[0, 1]` interval. `length(predictions)` should be equal to `length(observations)` and the order of the elements should match.
`thresholds`	A numeric vector of length two, typically calculated by `thresholds()`. The first element distinguishes certain negatives (certain absences) from uncertain predictions. The second element distinguishes certain positives (certain presences) from uncertain predictions. If missing, `confcons::thresholds(observations = observations, predictions = predictions)` is called, but see section 'Note' about why you should not use the default value.
`type`	A character vector of length one containing the value "positive" (for calculating *confidence in positive predictions* within known presences (CPP)) or "neutral" (for calculating *confidence in predictions* within known presences (CP)). Defaults to "positive".

Value

A numeric vector of length one. It is either NA_real_ or a positive number within the `[0, 1]` interval. Larger value indicates that the model is more confident.

Note

Technically, confidence can be calculated for the training subset, the evaluation subset, or the whole dataset as well. Note, however, that there is not so much sense to calculate confidence in the training subset, except for using the result for `consistency` calculation. If you need only the confidence measure, calculate it on the evaluation subset using `thresholds` previously determined on the whole dataset (i.e., do not use the default value of parameter `thresholds`). See the last example below and the vignette.

See Also

`thresholds` for calculating the two thresholds, `consistency` for calculating consistency

Examples

```
set.seed(12345)

# Using logical observations, default 'thresholds' and 'type' parameter:
observations_1000_logical <- c(rep(x = FALSE, times = 500),
                               rep(x = TRUE, times = 500))
predictions_1000 <- c(runif(n = 500, min = 0, max = 0.7),
                      runif(n = 500, min = 0.3, max = 1))
confidence(observations = observations_1000_logical,
           predictions = predictions_1000) # 0.561

# Using integer observations, default 'thresholds' parameter,
# both 'positive' and 'neutral' confidence type:
observations_4000_integer <- c(rep(x = 0L, times = 3000),
                               rep(x = 1L, times = 1000))
predictions_4000 <- c(runif(n = 3000, min = 0, max = 0.8),
                      runif(n = 1000, min = 0.2, max = 0.9))
confidence(observations = observations_4000_integer,
           predictions = predictions_4000, type = "positive") # 0.691
confidence(observations = observations_4000_integer,
           predictions = predictions_4000, type = "neutral") # 0.778

# Using some previously selected thresholds:
strict_thresholds <- c(0.1, 0.9)
permissive_thresholds <- c(0.4, 0.5)
percentile_thresholds <- quantile(x = predictions_4000[observations_4000_integer == 1],
                                  probs = c(0.1, 0.9)) # 10th and 90th percentile
confidence(observations = observations_4000_integer,
           predictions = predictions_4000,
           thresholds = strict_thresholds,
           type = "neutral") # 0
confidence(observations = observations_4000_integer,
           predictions = predictions_4000,
```

```
           thresholds = permissive_thresholds,
           type = "neutral") # 0.836
confidence(observations = observations_4000_integer,
           predictions = predictions_4000,
           thresholds = percentile_thresholds,
           type = "neutral") # 0.2

# Real-life case
# (thresholds calculated from the whole dataset, confidence from the evaluation subset):
dataset <- data.frame(
  observations = observations_4000_integer,
  predictions = predictions_4000,
  evaluation_mask = c(rep(x = FALSE, times = 250),
                      rep(x = TRUE, times = 250),
                      rep(x = FALSE, times = 250),
                      rep(x = TRUE, times = 250))
)
thresholds_whole <- thresholds(observations = dataset$observations,
                               predictions = dataset$predictions)
(confidence_evaluation <- confidence(observations = dataset$observations[dataset$evaluation_mask],
                             predictions = dataset$predictions[dataset$evaluation_mask],
                                    thresholds = thresholds_whole)) # 0.671

# Wrong parameterization:
try(confidence(observations = observations_1000_logical,
               predictions = predictions_1000,
               type = "pos")) # error
try(confidence(observations = observations_1000_logical,
               predictions = predictions_1000,
               thresholds = c(0.2, NA_real_))) # warning
try(confidence(observations = observations_1000_logical,
               predictions = predictions_1000,
               thresholds = c(-0.4, 0.85))) # warning
try(confidence(observations = observations_1000_logical,
               predictions = predictions_1000,
               thresholds = c(0.6, 0.3))) # warning
try(confidence(observations = observations_1000_logical,
               predictions = predictions_4000)) # error
set.seed(12345)
observations_4000_numeric <- c(rep(x = 0, times = 3000),
                               rep(x = 1, times = 1000))
predictions_4000_strange <- c(runif(n = 3000, min = -0.3, max = 0.4),
                              runif(n = 1000, min = 0.6, max = 1.5))
try(confidence(observations = observations_4000_numeric,
               predictions = predictions_4000_strange,
               thresholds = c(0.2, 0.7))) # multiple warnings
mask_of_normal_predictions <- predictions_4000_strange >= 0 & predictions_4000_strange <= 1
confidence(observations = as.integer(observations_4000_numeric)[mask_of_normal_predictions],
           predictions = predictions_4000_strange[mask_of_normal_predictions],
           thresholds = c(0.2, 0.7)) # OK
```

`consistency` *Consistency of the predictive distribution model*

Description

Calculate consistency (DCPP, DCP) of the model as the difference of the confidence calculated on the evaluation and the confidence calculated on the training subset. Consistency serves as a proxy for model's transferability.

Usage

```
consistency(conf_train, conf_eval)
```

Arguments

`conf_train`	**Conf**idence calculated on the **train**ing subset: a numeric vector of length one, containing a number within the `[0, 1]` interval. Typically calculated by function `confidence()` using the training subset.
`conf_eval`	**Conf**idence calculated on the **eval**uation subset: a numeric vector of length one, containing a number within the `[0, 1]` interval. Typically calculated by function `confidence()` using the evaluation subset.

Value

A numeric vector of length one. It is either NA_real_ or a number within the `[-1, 1]` interval. Typically, it falls within the `[-1, 0]` interval. Greater value indicates more consistent/transferable model. I.e, the closer the returned value is to -1, the less consistence/transferable the model is. Value above 0 might be an artifact or might indicate that the training and evaluation subsets were accidentally swapped.

See Also

`thresholds` for calculating the two thresholds, `confidence` for calculating confidence

Examples

```
# Simple examples:
consistency(conf_train = 0.93,
            conf_eval = 0.21) # -0.72 - hardly consistent/transferable model
consistency(conf_train = 0.43,
        conf_eval = 0.35) # -0.08 - consistent/transferable model, although not so confident
consistency(conf_train = 0.87,
        conf_eval = 0.71) # -0.16 - a consistent/transferable model that is confident as well
consistency(conf_train = 0.67,
            conf_eval = 0.78) # 0.11 - positive value might be an artifact
consistency(conf_train = 0.67,
            conf_eval = NA_real_) # NA
```

```
# Real-life case:
set.seed(12345)
observations <- c(rep(x = FALSE, times = 500),
                  rep(x = TRUE, times = 500))
predictions <- c(runif(n = 500, min = 0, max = 0.7),
                 runif(n = 500, min = 0.3, max = 1))
dataset <- data.frame(
 observations = observations,
 predictions = predictions,
 evaluation_mask = c(rep(x = FALSE, times = 250),
                     rep(x = TRUE, times = 250),
                     rep(x = FALSE, times = 250),
                     rep(x = TRUE, times = 250))
)
thresholds_whole <- thresholds(observations = dataset$observations,
                               predictions = dataset$predictions)
confidence_training <- confidence(observations = dataset$observations[!dataset$evaluation_mask],
                            predictions = dataset$predictions[!dataset$evaluation_mask],
                                 thresholds = thresholds_whole) # 0.602
confidence_evaluation <- confidence(observations = dataset$observations[dataset$evaluation_mask],
                              predictions = dataset$predictions[dataset$evaluation_mask],
                                    thresholds = thresholds_whole) # 0.520
consistency(conf_train = confidence_training,
            conf_eval = confidence_evaluation) # -0.083 - consistent/transferable model

# Wrong parameterization:
try(consistency(conf_train = 1.3,
                conf_eval = 0.5)) # warning
try(consistency(conf_train = 0.6,
                conf_eval = c(0.4, 0.5))) # warning
```

`measures` *Goodness-of-fit, confidence and consistency measures*

Description

Wrapper function for calculating the predictive distribution model's `confidence`, `consistency`, and optionally some well-known goodness-of-fit measures as well. The calculated measures are as follows:

- confidence in predictions (CP) and confidence in positive predictions (CPP) within known presences for the training and evaluation subsets
- consistency of predictions (difference of CPs; DCP) and positive predictions (difference of CPPs; DCPP)
- Area Under the ROC Curve (AUC) - optional (see parameter `goodness`)
- maximum of the True Skill Statistic (maxTSS) - optional (see parameter `goodness`)

Usage

```
measures(
  observations,
  predictions,
  evaluation_mask,
  goodness = FALSE,
  df = FALSE
)
```

Arguments

observations	Either an integer or logical vector containing the binary observations where presences are encoded as 1s/TRUEs and absences as 0s/FALSEs.
predictions	A numeric vector containing the predicted probabilities of occurrence typically within the [0, 1] interval. length(predictions) should be equal to length(observations) and the order of the elements should match.
evaluation_mask	A logical vector (mask) of the evaluation subset. Its ith element indicates whether the ith element of observations was used for evaluation (TRUE) or for training (FALSE). length(evaluation_mask) should be equal to length(observations) and the order of the elements should match, i.e. observations[evaluation_mask] were the evaluation subset and observations[!evaluation_mask] were the training subset.
goodness	Logical vector of length one, defaults to FALSE. Indicates, whether goodness-of-fit measures (AUC and maxTSS) should be calculated. If set to TRUE, external package **ROCR** (Sing et al. 2005) is needed for the calculation (see section 'Note').
df	Logical vector of length one, defaults to FALSE. Indicates, whether the returned value should be a one-row data.frame that is rbind()able if measures() is called on multiple models in a for loop or a lapply(). See section 'Value' and 'Examples' for details.

Value

A named numeric vector (if df is FALSE; the default) or a data.frame (if df is TRUE) of one row. length() of the vector or ncol() of the data.frame is 6 (if goodness is FALSE; the default) or 8 (if goodness is TRUE). The name of the elements/columns are as follows:

CP_train confidence in predictions within known presences (CP) for the training subset

CP_eval confidence in predictions within known presences (CP) for the evaluation subset

DCP consistency of predictions (difference of CPs)

CPP_train confidence in positive predictions within known presences (CPP) for the training subset

CPP_eval confidence in positive predictions within known presences (CPP) for the evaluation subset

DCPP consistency of positive predictions (difference of CPPs)

AUC Area Under the ROC Curve (Hanley and McNeil 1982; calculated by `ROCR::performance()`). This element/column is available only if parameter 'goodness' is set to TRUE. If package **ROCR** is not available but parameter 'goodness' is set to TRUE, the value of AUC is NA_real_ and a warning is raised.

maxTSS Maximum of the True Skill Statistic (Allouche et al. 2006; calculated by `ROCR::performance()`). This element/column is available only if parameter 'goodness' is set to TRUE. If package **ROCR** is not available but parameter 'goodness' is set to TRUE, the value of maxTSS is NA_real_ and a warning is raised.

Note

Since **confcons** is a light-weight, stand-alone packages, it does not import package **ROCR** (Sing et al. 2005), i.e. installing **confcons** does not mean installing **ROCR** automatically. If you need AUC and maxTSS (i.e., parameter 'goodness' is set to TRUE), you should install **ROCR** or install **confcons** along with its dependencies (i.e., `devtools::install_github(repo = "bfakos/confcons", dependencies = TRUE)`).

References

- Allouche O, Tsoar A, Kadmon R (2006): Assessing the accuracy of species distribution models: prevalence, kappa and the true skill statistic (TSS). Journal of Applied Ecology 43(6): 1223-1232. doi:10.1111/j.13652664.2006.01214.x.
- Hanley JA, McNeil BJ (1982): The meaning and use of the area under a receiver operating characteristic (ROC) curve. Radiology 143(1): 29-36. doi:10.1148/radiology.143.1.7063747.
- Sing T, Sander O, Beerenwinkel N, Lengauer T. (2005): ROCR: visualizing classifier performance in R. Bioinformatics 21(20): 3940-3941. doi:10.1093/bioinformatics/bti623.

See Also

`confidence` for calculating confidence, `consistency` for calculating consistency, `ROCR::performance()` for calculating AUC and TSS

Examples

```
set.seed(12345)
dataset <- data.frame(
 observations = c(rep(x = FALSE, times = 500),
                  rep(x = TRUE, times = 500)),
 predictions_model1 = c(runif(n = 250, min = 0, max = 0.6),
                        runif(n = 250, min = 0.1, max = 0.7),
                        runif(n = 250, min = 0.4, max = 1),
                        runif(n = 250, min = 0.3, max = 0.9)),
 predictions_model2 = c(runif(n = 250, min = 0.1, max = 0.55),
                        runif(n = 250, min = 0.15, max = 0.6),
                        runif(n = 250, min = 0.3, max = 0.9),
                        runif(n = 250, min = 0.25, max = 0.8)),
 evaluation_mask = c(rep(x = FALSE, times = 250),
                     rep(x = TRUE, times = 250),
                     rep(x = FALSE, times = 250),
                     rep(x = TRUE, times = 250))
```

```
)

# Default parameterization, return a vector without AUC and maxTSS:
conf_and_cons <- measures(observations = dataset$observations,
                          predictions = dataset$predictions_model1,
                          evaluation_mask = dataset$evaluation_mask)
print(conf_and_cons)
names(conf_and_cons)
conf_and_cons[c("CPP_eval", "DCPP")]

# Calculate AUC and maxTSS as well if package ROCR is installed:
if (requireNamespace(package = "ROCR", quietly = TRUE)) {
  conf_and_cons_and_goodness <- measures(observations = dataset$observations,
                                         predictions = dataset$predictions_model1,
                                         evaluation_mask = dataset$evaluation_mask,
                                         goodness = TRUE)
}

# Calculate the measures for multiple models in a for loop:
model_IDs <- as.character(1:2)
for (model_ID in model_IDs) {
  column_name <- paste0("predictions_model", model_ID)
  conf_and_cons <- measures(observations = dataset$observations,
                            predictions = dataset[, column_name, drop = TRUE],
                            evaluation_mask = dataset$evaluation_mask,
                            df = TRUE)
  if (model_ID == model_IDs[1]) {
    conf_and_cons_df <- conf_and_cons
  } else {
    conf_and_cons_df <- rbind(conf_and_cons_df, conf_and_cons)
  }
}
conf_and_cons_df

# Calculate the measures for multiple models in a lapply():
conf_and_cons_list <- lapply(X = model_IDs,
                             FUN = function(model_ID) {
                               column_name <- paste0("predictions_model", model_ID)
                               measures(observations = dataset$observations,
                                        predictions = dataset[, column_name, drop = TRUE],
                                          evaluation_mask = dataset$evaluation_mask,
                                          df = TRUE)
                             })
conf_and_cons_df <- do.call(what = rbind,
                            args = conf_and_cons_list)
conf_and_cons_df
```

---

thresholds *Thresholds needed to create the extended confusion matrix*

Description

Calculate the two thresholds distinguishing certain negatives/positives from uncertain predictions. The thresholds are needed to create the extended confusion matrix and are further used for confidence calculation.

Usage

```
thresholds(observations, predictions = NULL, type = "mean", range = 0.5)
```

Arguments

observations	Either an integer or logical vector containing the binary observations where presences are encoded as 1s/TRUEs and absences as 0s/FALSEs.
predictions	A numeric vector containing the predicted probabilities of occurrence typically within the [0, 1] interval. length(predictions) should be equal to length(observations) and the order of the elements should match. predictions is optional: needed and used only if type is 'mean' and ignored otherwise.
type	A character vector of length one containing the value 'mean' (for calculating mean of the predictions within known presences and absences) or 'information' (for calculating thresholds based on relative information gain) . Defaults to 'mean'.
range	A numeric vector of length one containing a value from the]0, 0.5] interval. It is the parameter of the information-based method and is used only if type is 'information'. The larger the range is, the more predictions are treated as uncertain. Defaults to 0.5.

Value

A named numeric vector of length 2. The first element ('threshold1') is the mean of probabilities predicted to the absence locations distinguishing certain negatives (certain absences) from uncertain predictions. The second element ('threshold2') is the mean of probabilities predicted to the presence locations distinguishing certain positives (certain presences) from uncertain predictions. For a typical model better than the random guess, the first element is smaller than the second one. The returned value might contain NaN(s) if the number of observed presences and/or absences is 0.

Note

thresholds() should be called using the whole dataset containing both training and evaluation locations.

See Also

confidence for calculating confidence, consistency for calculating consistency

Examples

```
set.seed(12345)

# Using logical observations:
observations_1000_logical <- c(rep(x = FALSE, times = 500),
                               rep(x = TRUE, times = 500))
predictions_1000 <- c(runif(n = 500, min = 0, max = 0.7),
                      runif(n = 500, min = 0.3, max = 1))
thresholds(observations = observations_1000_logical,
           predictions = predictions_1000) # 0.370 0.650

# Using integer observations:
observations_4000_integer <- c(rep(x = 0L, times = 3000),
                               rep(x = 1L, times = 1000))
predictions_4000 <- c(runif(n = 3000, min = 0, max = 0.8),
                      runif(n = 1000, min = 0.2, max = 0.9))
thresholds(observations = observations_4000_integer,
           predictions = predictions_4000) # 0.399 0.545

# Wrong parameterization:
try(thresholds(observations = observations_1000_logical,
               predictions = predictions_4000)) # error
set.seed(12345)
observations_4000_numeric <- c(rep(x = 0, times = 3000),
                               rep(x = 1, times = 1000))
predictions_4000_strange <- c(runif(n = 3000, min = -0.3, max = 0.4),
                              runif(n = 1000, min = 0.6, max = 1.5))
try(thresholds(observations = observations_4000_numeric,
               predictions = predictions_4000_strange)) # multiple warnings
mask_of_normal_predictions <- predictions_4000_strange >= 0 & predictions_4000_strange <= 1
thresholds(observations = as.integer(observations_4000_numeric)[mask_of_normal_predictions],
           predictions = predictions_4000_strange[mask_of_normal_predictions]) # OK
```

Index