Package 'waywiser'

October 23, 2022

Type Package

Title Methods for Assessing Spatial Models

Version 0.2.0

Description Assessing predictive models of spatial data can be challenging, both because these models are typically built for extrapolating outside the original region represented by training data and due to potential spatially structured errors, with ``hot spots'' of higher than expected error clustered geographically due to spatial structure in the underlying data. These functions provide methods for measuring the spatial structure of model errors and evaluating where predictions can be made safely, and are particularly useful for models fit using the 'tidymodels' framework. Methods include Moran's I ('Moran' (1950) <doi:10.2307/2332142>), Geary's C ('Geary' (1954) <doi:10.2307/2986645>), Getis-Ord's G ('Ord' and 'Getis' (1995) <doi:10.1111/j.1538-4632.1995.tb00912.x>), as well as an implementation of the area of applicability methodology from 'Meyer' and 'Pebesma' (2021) (<doi:10.1111/2041-210X.13650>).

License MIT + file LICENSE

URL https://github.com/mikemahoney218/waywiser, https://mikemahoney218.github.io/waywiser/

BugReports https://github.com/mikemahoney218/waywiser/issues

Depends R (>= 3.6)

Imports fields, glue, hardhat, Matrix, purrr, rlang, rsample, sf, spdep, stats, tibble, yardstick

Suggests applicable, covr, dplyr, ggplot2, recipes, sfdep, spatialsample, spelling, testthat (>= 3.0.0), tidymodels, tidyr, vip

Config/testthat/edition 3

Config/testthat/parallel true

Encoding UTF-8

RoxygenNote 7.2.1

Language en-US

NeedsCompilation no

Author Michael Mahoney [aut, cre] (<https://orcid.org/0000-0003-2402-304X>),
Lucas Johnson [ctb] (<https://orcid.org/0000-0002-7953-0260>),
RStudio [cph, fnd]

Maintainer Michael Mahoney <mike.mahoney.218@gmail.com>

Repository CRAN

Date/Publication 2022-10-23 02:20:02 UTC

R topics documented:

`predict.ww_area_of_applicability` *Predict from a* `ww_area_of_applicability`

Description

Predict from a `ww_area_of_applicability`

Usage

```
## S3 method for class 'ww_area_of_applicability'
predict(object, new_data, ...)
```

Arguments

`object`	A `ww_area_of_applicability` object.
`new_data`	A data frame or matrix of new samples.
`...`	Not used.

Details

The function computes the distance indices of the new data and whether or not they are "inside" the area of applicability.

Value

A tibble of predictions, with two columns: `di`, numeric, contains the "dissimilarity index" of each point in `new_data`, while `aoa`, logical, contains whether a row is inside (`TRUE`) or outside (`FALSE`) the area of applicability.

Note that this function is often called using `raster::predict()` or `terra::predict()`, in which case `aoa` will be converted to numeric implicitly; `1` values correspond to cells "inside" the area of applicability and `0` corresponds to cells "outside" the AOA.

The number of rows in the tibble is guaranteed to be the same as the number of rows in `new_data`. Rows with `NA` predictor values will have `NA` `di` and `aoa` values.

See Also

Other area of applicability functions: `ww_area_of_applicability()`

Examples

```
library(vip)
train <- gen_friedman(1000, seed = 101)  # ?vip::gen_friedman
test <- train[701:1000, ]
train <- train[1:700, ]
pp <- stats::ppr(y ~ ., data = train, nterms = 11)
importance <- vi_permute(
  pp,
  target = "y",
  metric = "rsquared",
  pred_wrapper = predict
)

aoa <- ww_area_of_applicability(y ~ ., train, test, importance = importance)
predict(aoa, test)
```

`ww_area_of_applicability` *Find the area of applicability*

Description

This function calculates the "area of applicability" of a model, as introduced by Meyer and Pebesma (2021). While the initial paper introducing this method focused on spatial models, there is nothing inherently spatial about the method; it can be used with any type of data.

Usage

```
ww_area_of_applicability(x, ...)

## S3 method for class 'data.frame'
ww_area_of_applicability(
  x,
  testing = NULL,
  importance,
  ...,
  na_action = na.fail
)

## S3 method for class 'matrix'
ww_area_of_applicability(
  x,
  testing = NULL,
  importance,
  ...,
  na_action = na.fail
)

## S3 method for class 'formula'
ww_area_of_applicability(
  x,
  data,
  testing = NULL,
  importance,
  ...,
  na_action = na.fail
)

## S3 method for class 'recipe'
ww_area_of_applicability(
  x,
  data,
  testing = NULL,
  importance,
  ...,
  na_action = na.fail
)

## S3 method for class 'rset'
ww_area_of_applicability(x, y = NULL, importance, ..., na_action = na.fail)
```

Arguments

x	Either a data frame, matrix, formula (specifying predictor terms on the right-hand side), recipe (from recipes::recipe(), or rset object, produced by re-

sampling functions from rsample or spatialsample.

If `x` is a recipe, it should be the same one used to pre-process the data used in your model. If the recipe used to build the area of applicability doesn't match the one used to build the model, the returned area of applicability won't be correct.

`...` Not currently used.

`testing` A data frame or matrix containing the data used to validate your model. This should be the same data as used to calculate all model accuracy metrics.

If this argument is `NULL`, then this function will use the training data (from `x` or `data`) to calculate within-sample distances. This may result in the area of applicability threshold being set too high, with the result that too many points are classed as "inside" the area of applicability.

`importance` Either:

- A data.frame with two columns: `term`, containing the names of each variable in the training and testing data, and `estimate`, containing the (raw or scaled) feature importance for each variable.
- An object of class `vi` with at least two columns, `Variable` and `Importance`.

All variables in the training data (`x` or `data`, depending on the context) must have a matching importance estimate, and all terms with importance estimates must be in the training data.

`na_action` A function which indicates what should happen when the data (any of `x`, `data`, or `testing`) contain NAs. The default is `na.fail`; you may wish to set it to `na.omit` or any of the functions from the "zoo" package. This function ignores the value of `options("na.action")` in order to make cross-computer (and cross-session) results more stable. Note that this argument only impacts fitting the area of applicability and has no impact on predictions.

`data` The data frame representing your "training" data, when using the formula or recipe methods.

`y` Optional: a recipe (from `recipes::recipe()`) or formula.

If `y` is a recipe, it should be the same one used to pre-process the data used in your model. If the recipe used to build the area of applicability doesn't match the one used to build the model, the returned area of applicability won't be correct.

Details

Predictions made on points "inside" the area of applicability should be as accurate as predictions made on the data provided to `testing`. That means that generally `testing` should be your final hold-out set so that predictions on points inside the area of applicability are accurately described by your reported model metrics. When passing an `rset` object to `x`, predictions made on points "inside" the area of applicability instead should be as accurate as predictions made on the assessment sets during cross-validation.

This method assumes your model was fit using dummy variables in the place of any non-numeric predictor, and that you have one importance score per dummy variable. Having non-numeric predictors will cause this function to fail.

Value

A `ww_area_of_applicability` object, which can be used with `predict()` to calculate the distance of new data to the original training data, and determine if new data is within a model's area of applicability.

Differences from CAST

This implementation differs from Meyer and Pebesma (2021) (and therefore from CAST) when using cross-validated data in order to minimize data leakage. Namely, in order to calculate the dissimilarity index DI_k, CAST:

1. Rescales all data used for cross validation at once, lumping assessment folds in with analysis data.
2. Calculates a single $\bar{d}$ as the mean distance between all points in the rescaled data set, including between points in the same assessment fold.
3. For each point k that's used in an assessment fold, calculates d_k as the minimum distance between k and any point in its corresponding analysis fold.
4. Calculates DI_k by dividing d_k by $\bar{d}$ (which was partially calculated as the distance between k and the rest of the rescaled data).

Because assessment data is used to calculate constants for rescaling analysis data and $\bar{d}$, the assessment data may appear too "similar" to the analysis data when calculating DI_k. As such, waywiser treats each fold in an `rset` independently:

1. Each analysis set is rescaled independently.
2. Separate $\bar{d}$ are calculated for each fold, as the mean distance between all points in the analysis set for that fold.
3. Identically to CAST, d_k is the minimum distance between a point k in the assessment fold and any point in the corresponding analysis fold.
4. DI_k is then found by dividing d_k by $\bar{d}$, which was calculated independently from k.

Predictions are made using the full training data set, rescaled once (in the same way as CAST), and the mean $\bar{d}$ across folds, under the assumption that the "final" model in use will be retrained using the entire data set.

In practice, this means waywiser produces very slightly higher $\bar{d}$ values than CAST and a slightly higher area of applicability threshold than CAST when using `rset` objects.

References

H. Meyer and E. Pebesma. 2021. "Predicting into unknown space? Estimating the area of applicability of spatial prediction models," Methods in Ecology and Evolution 12(9), pp 1620 - 1633, doi: 10.1111/2041-210X.13650.

See Also

Other area of applicability functions: `predict.ww_area_of_applicability()`

Examples

```
train <- vip::gen_friedman(1000, seed = 101)  # ?vip::gen_friedman
test <- train[701:1000, ]
train <- train[1:700, ]
pp <- stats::ppr(y ~ ., data = train, nterms = 11)
importance <- vip::vi_permute(
  pp,
  target = "y",
  metric = "rsquared",
  pred_wrapper = predict
)

aoa <- ww_area_of_applicability(y ~ ., train, test, importance = importance)
predict(aoa, test)
```

`ww_build_neighbors` *Make 'nb' objects from sf objects*

Description

Make 'nb' objects from sf objects

Usage

```
ww_build_neighbors(data, nb = NULL, ..., call = rlang::caller_env())
```

Arguments

`data`	An sf object (of class "sf" or "sfc").
`nb`	An object of class "nb" (in which case it will be returned unchanged), or a function to create an object of class "nb" from `data` and `...`, or `NULL`. See details.
`...`	Arguments passed to the neighbor-creating function.
`call`	The execution environment of a currently running function, e.g. `call = caller_env()`. The corresponding function call is retrieved and mentioned in error messages as the source of the error. You only need to supply `call` when throwing a condition from a helper function which wouldn't be relevant to mention in the message. Can also be `NULL` or a defused function call to respectively not display any call or hard-code a code to display. For more information about error calls, see Including function calls in error messages.

Details

When nb = NULL, the method used to create neighbors from data is dependent on what geometry type data is:

- If nb = NULL and data is a point geometry (classes "sfc_POINT" or "sfc_MULTIPOINT") the "nb" object will be created using ww_make_point_neighbors().
- If nb = NULL and data is a polygon geometry (classes "sfc_POLYGON" or "sfc_MULTIPOLYGON") the "nb" object will be created using ww_make_polygon_neighbors().
- If nb = NULL and data is any other geometry type, the "nb" object will be created using the centroids of the data as points, with a warning.

Value

An object of class "nb".

---

ww_build_weights *Build "listw" objects of spatial weights*

Description

Build "listw" objects of spatial weights

Usage

```
ww_build_weights(x, wt = NULL, include_self = FALSE, ...)
```

Arguments

x	Either an sf object or a "nb" neighbors list object. If an sf object, will be converted into a neighbors list via ww_build_neighbors().
wt	Either a "listw" object (which will be returned unchanged), a function for creating a "listw" object from x, or NULL, in which case weights will be constructed via spdep::nb2listw().
include_self	Include each region itself in its own list of neighbors?
...	Arguments passed to the weight constructing function.

Value

A listw object.

ww_global_geary_c *Global Geary's C statistic*

Description

Calculate the global Geary's C statistic for model residuals. `ww_global_geary_c()` returns the statistic itself, while `ww_global_geary_pvalue()` returns the associated p value. `ww_global_geary()` returns both.

Usage

```
ww_global_geary_c(data, ...)

ww_global_geary_c_vec(
  truth,
  estimate,
  wt = NULL,
  alternative = "greater",
  randomization = TRUE,
  na_rm = TRUE,
  ...
)

ww_global_geary_pvalue(data, ...)

ww_global_geary_pvalue_vec(
  truth,
  estimate,
  wt = NULL,
  alternative = "greater",
  randomization = TRUE,
  na_rm = TRUE,
  ...
)

ww_global_geary(
  data,
  truth,
  estimate,
  wt = NULL,
  alternative = "greater",
  randomization = TRUE,
  na_rm = TRUE,
  ...
)
```

Arguments

data	A data.frame containing the columns specified by the truth and estimate arguments.
...	Additional arguments passed to spdep::geary.test().
truth	The column identifier for the true results (that is numeric). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a numeric vector.
estimate	The column identifier for the predicted results (that is also numeric). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a numeric vector.
wt	A "listw" object, for instance as created with ww_build_weights().
alternative	a character string specifying the alternative hypothesis, must be one of "greater" (default), "less" or "two.sided".
randomization	variance of I calculated under the assumption of randomisation, if FALSE normality
na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds.

Value

A tibble with columns .metric, .estimator, and .estimate and nrow(data) rows of values. For grouped data frames, the number of rows returned will be the same as the number of groups. For _vec() functions, a single value (or NA).

Examples

```
data(guerry, package = "sfdep")

guerry_modeled <- guerry
guerry_lm <- lm(crime_pers ~ literacy, guerry_modeled)
guerry_modeled$predictions <- predict(guerry_lm, guerry_modeled)

## Not run:
ww_global_geary(guerry_modeled, crime_pers, predictions)

## End(Not run)
```

---

ww_global_moran_i *Global Moran's I statistic*

Description

Calculate the global Moran's I statistic for model residuals. `ww_global_moran_i()` returns the statistic itself, while `ww_global_moran_pvalue()` returns the associated p value. `ww_global_moran()` returns both.

Usage

```
ww_global_moran_i(data, ...)

ww_global_moran_i_vec(
  truth,
  estimate,
  wt = NULL,
  alternative = "greater",
  randomization = TRUE,
  na_rm = TRUE,
  ...
)

ww_global_moran_pvalue(data, ...)

ww_global_moran_pvalue_vec(
  truth,
  estimate,
  wt = NULL,
  alternative = "greater",
  randomization = TRUE,
  na_rm = TRUE,
  ...
)

ww_global_moran(
  data,
  truth,
  estimate,
  wt = NULL,
  alternative = "greater",
  randomization = TRUE,
  na_rm = TRUE,
  ...
)
```

Arguments

`data`	A `data.frame` containing the columns specified by the `truth` and `estimate` arguments.
`...`	Additional arguments passed to `spdep::moran.test()`.

truth	The column identifier for the true results (that is numeric). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a numeric vector.
estimate	The column identifier for the predicted results (that is also numeric). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a numeric vector.
wt	A "listw" object, for instance as created with ww_build_weights().
alternative	a character string specifying the alternative hypothesis, must be one of greater (default), less or two.sided.
randomization	variance of I calculated under the assumption of randomisation, if FALSE normality
na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds.

Value

A tibble with columns .metric, .estimator, and .estimate and nrow(data) rows of values. For grouped data frames, the number of rows returned will be the same as the number of groups. For _vec() functions, a single value (or NA).

Examples

```
data(guerry, package = "sfdep")

guerry_modeled <- guerry
guerry_lm <- lm(crime_pers ~ literacy, guerry_modeled)
guerry_modeled$predictions <- predict(guerry_lm, guerry_modeled)

ww_global_moran_i(guerry_modeled, crime_pers, predictions)
ww_global_moran(guerry_modeled, crime_pers, predictions)
```

---

ww_local_geary_c *Local Geary's C statistic*

Description

Calculate the local Geary's C statistic for model residuals. ww_local_geary_c() returns the statistic itself, while ww_local_geary_pvalue() returns the associated p value. ww_local_geary() returns both.

Usage

```
ww_local_geary_c(data, ...)

ww_local_geary_c_vec(truth, estimate, wt, na_rm = TRUE, ...)

ww_local_geary_pvalue(data, ...)

ww_local_geary_pvalue_vec(
  truth,
  estimate,
  wt = NULL,
  alternative = "two.sided",
  nsim = 499,
  na_rm = TRUE,
  ...
)

ww_local_geary(
  data,
  truth,
  estimate,
  wt = NULL,
  alternative = "two.sided",
  nsim = 499,
  na_rm = TRUE,
  ...
)
```

Arguments

data	A data.frame containing the columns specified by the truth and estimate arguments.
...	Additional arguments passed to spdep::localC_perm().
truth	The column identifier for the true results (that is numeric). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a numeric vector.
estimate	The column identifier for the predicted results (that is also numeric). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a numeric vector.
wt	A "listw" object, for instance as created with ww_build_weights().
na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds.
alternative	A character defining the alternative hypothesis. Must be one of "two.sided", "less" or "greater".
nsim	The number of simulations to be used for permutation test.

Value

A tibble with columns .metric, .estimator, and .estimate and `nrow(data)` rows of values. For grouped data frames, the number of rows returned will be the same as the number of groups. For `_vec()` functions, a numeric vector of `length(truth)` (or NA).

Examples

```
data(guerry, package = "sfdep")

guerry_modeled <- guerry
guerry_lm <- lm(crime_pers ~ literacy, guerry_modeled)
guerry_modeled$predictions <- predict(guerry_lm, guerry_modeled)

ww_local_geary_c(guerry_modeled, crime_pers, predictions)
ww_local_geary(guerry_modeled, crime_pers, predictions)
```

---

ww_local_getis_ord_g *Local Getis-Ord G and G* statistic*

Description

Calculate the local Getis-Ord G and G* statistic for model residuals. `ww_local_getis_ord_g()` returns the statistic itself, while `ww_local_getis_ord_pvalue()` returns the associated p value. `ww_local_getis_ord()` returns both.

Usage

```
ww_local_getis_ord_g(data, ...)

ww_local_getis_ord_g_vec(
  truth,
  estimate,
  wt = NULL,
  alternative = "two.sided",
  nsim = 499,
  na_rm = TRUE,
  ...,
  include_self = FALSE
)

ww_local_getis_ord_pvalue(data, ...)

ww_local_getis_ord_pvalue_vec(
  truth,
  estimate,
```

```
  wt = NULL,
  alternative = "two.sided",
  nsim = 499,
  na_rm = TRUE,
  ...,
  include_self = FALSE
)

ww_local_getis_ord(
  data,
  truth,
  estimate,
  wt = NULL,
  alternative = "two.sided",
  nsim = 499,
  na_rm = TRUE,
  ...,
  include_self = FALSE
)
```

Arguments

data	A data.frame containing the columns specified by the truth and estimate arguments.
...	Arguments passed to spdep::localG_perm()
truth	The column identifier for the true results (that is numeric). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a numeric vector.
estimate	The column identifier for the predicted results (that is also numeric). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a numeric vector.
wt	A "listw" object, for instance as created with ww_build_weights().
alternative	a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less".
nsim	default 499, number of conditonal permutation simulations
na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds.
include_self	Include each region itself in its own list of neighbors? Only used when wt is NULL, and if TRUE means this function calculates G* instead of G.

Value

A tibble with columns .metric, .estimator, and .estimate and nrow(data) rows of values. For grouped data frames, the number of rows returned will be the same as the number of groups. For _vec() functions, a numeric vector of length(truth) (or NA).

Examples

```
data(guerry, package = "sfdep")

guerry_modeled <- guerry
guerry_lm <- lm(crime_pers ~ literacy, guerry_modeled)
guerry_modeled$predictions <- predict(guerry_lm, guerry_modeled)

ww_local_getis_ord_g(guerry_modeled, crime_pers, predictions)
ww_local_getis_ord(guerry_modeled, crime_pers, predictions)
ww_local_getis_ord(guerry_modeled, crime_pers, predictions, include_self = TRUE)
```

---

ww_local_moran_i *Local Moran's I statistic*

Description

Calculate the local Moran's I statistic for model residuals. `ww_local_moran_i()` returns the statistic itself, while `ww_local_moran_pvalue()` returns the associated p value. `ww_local_moran()` returns both.

Usage

```
ww_local_moran_i(data, ...)

ww_local_moran_i_vec(
  truth,
  estimate,
  wt = NULL,
  alternative = "two.sided",
  na_rm = TRUE,
  ...
)

ww_local_moran_pvalue(data, ...)

ww_local_moran_pvalue_vec(
  truth,
  estimate,
  wt = NULL,
  alternative = "two.sided",
  na_rm = TRUE,
  ...
)

ww_local_moran(
```

```
  data,
  truth,
  estimate,
  wt = NULL,
  alternative = "two.sided",
  na_rm = TRUE,
  ...
)
```

Arguments

data	A data.frame containing the columns specified by the truth and estimate arguments.
...	Additional arguments passed to spdep::localmoran().
truth	The column identifier for the true results (that is numeric). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a numeric vector.
estimate	The column identifier for the predicted results (that is also numeric). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a numeric vector.
wt	A "listw" object, for instance as created with ww_build_weights().
alternative	a character string specifying the alternative hypothesis, must be one of greater, less or two.sided (default).
na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds.

Value

A tibble with columns .metric, .estimator, and .estimate and nrow(data) rows of values. For grouped data frames, the number of rows returned will be the same as the number of groups. For _vec() functions, a numeric vector of length(truth) (or NA).

Examples

```
data(guerry, package = "sfdep")

guerry_modeled <- guerry
guerry_lm <- lm(crime_pers ~ literacy, guerry_modeled)
guerry_modeled$predictions <- predict(guerry_lm, guerry_modeled)

ww_local_moran_i(guerry_modeled, crime_pers, predictions)
ww_local_moran(guerry_modeled, crime_pers, predictions)
```

`ww_make_point_neighbors`

Make 'nb' objects from point geometries

Description

This function uses `spdep::knearneigh()` and `spdep::knn2nb()` to create a "nb" neighbors list.

Usage

```
ww_make_point_neighbors(data, k = 1, sym = FALSE, ...)
```

Arguments

`data`	An `sfc_POINT` or `sfc_MULTIPOINT` object.
`k`	How many nearest neighbors to use in `spdep::knearneigh()`.
`sym`	Force the output neighbors list (from `spdep::knn2nb()`) to symmetry.
`...`	Other arguments passed to `spdep::knearneigh()`.

Value

An object of class "nb"

`ww_make_polygon_neighbors`

Make 'nb' objects from polygon geometries

Description

This function is an extremely thin wrapper around `spdep::poly2nb()`, renamed to use the way-wiser "ww" prefix.

Usage

```
ww_make_polygon_neighbors(data, ...)
```

Arguments

`data`	An `sfc_POLYGON` or `sfc_MULTIPOLYGON` object.
`...`	Additional arguments passed to `spdep::poly2nb()`.

Value

An object of class "nb"

Index