Package 'AgriFusionR'

August 26, 2026

Title An Integration Framework for Agricultural Analytics

Version 0.1.0

Description Assembles agricultural analyses around a single unit of observation, the management unit within a season, and keeps climate, soil and remote-sensing covariates aligned to it. Covariates are aggregated over phenological windows derived from accumulated growing degree days rather than calendar months, following McMaster and Wilhelm (1997) <doi:10.1016/S0168-1923(97)00027-0>. Models are validated with spatial resampling by default, since random cross-validation inflates apparent skill when observations are spatially autocorrelated, as shown by Roberts and others (2017) <doi:10.1111/ecog.02881>. Prediction intervals use split conformal inference after Lei and others (2018) <doi:10.1080/01621459.2017.1307116>. Data sources and learning algorithms are supplied through registries so that new providers and methods can be added without modifying the package.

License MIT + file LICENSE

Encoding UTF-8

Language en-GB

Depends R (>= 4.1.0)

Imports graphics, grDevices, stats, utils

Suggests testthat (>= 3.0.0), ranger, xgboost, Cubist, glmnet, kernlab, mgcv, treeshap, nasapower, chirps, daymetr, geodata, terra, agridat, knitr, rmarkdown

VignetteBuilder knitr

Config/testthat/edition 3

URL https://github.com/mqfarooqi1/AgriFusionR

BugReports https://github.com/mqfarooqi1/AgriFusionR/issues

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Muhammad Farooqi [aut, cre] (ORCID: <https://orcid.org/0000-0003-4918-9791>)

Maintainer Muhammad Farooqi <mqfarooqi@gmail.com>

Repository CRAN

Date/Publication 2026-08-26 20:20:08 UTC

Contents

`add_layer` *Attach a covariate layer to a project*

Description

Fetches covariates from a registered source and attaches them to the project, keyed to the management unit and season. `add_climate()`, `add_soil()` and `add_satellite()` differ only in the layer they write to and the sources they expect; all three are thin calls to the source registry, so a source added with `register_source()` is usable immediately.

Usage

```
add_layer(p, source, layer, overwrite = FALSE, ...)

add_climate(p, source = "demo", layer = "climate", overwrite = FALSE, ...)

add_soil(p, source = "demo_soil", layer = "soil", overwrite = FALSE, ...)

add_satellite(p, source, layer = "satellite", overwrite = FALSE, ...)
```

Arguments

p	An agri_project().
source	Name of a registered source. See list_sources().
layer	Name to store the layer under.
overwrite	Replace an existing layer of the same name.
...	Passed to the source's fetch function.

Details

Nothing is fetched twice: a layer already present is returned unchanged unless overwrite = TRUE.

Value

The project, with the layer attached and the operation recorded in its provenance.

See Also

register_source(), list_sources(), build_features()

Examples

```
p <- agri_project(demo_agri_data(n_units = 4, n_seasons = 2))
p <- add_climate(p, source = "demo")
names(p$layers)
head(p$layers$climate$data)
```

---

agri_project *Create an agricultural analysis project*

Description

Builds the object every other function in the package operates on. The unit of observation is a **management unit within a season**: a field, plot or administrative area, together with the window over which the crop grew. Each observation is keyed by (unit_id, season_id), and every covariate layer added later must reduce to that key.

Usage

```
agri_project(
  data,
  unit_id = NULL,
  x = NULL,
  y = NULL,
  season = NULL,
  start = NULL,
  end = NULL,
  crop = NULL,
  crs = 4326,
  cache_dir = NULL
)
```

Arguments

data	A data frame with one row per management unit and season.
unit_id, x, y	Column names giving the unit identifier and its coordinates. Detected from names such as unit_id, field, lon, lat when NULL.
season	Column naming the season, typically a year. Detected from season or year.
start, end	Columns giving the start and end of the growing window as dates. Detected from names such as planting/sowing and harvest.
crop	Column giving the crop, if more than one is present.
crs	Coordinate reference system as an EPSG code. Only 4326 is currently treated as geographic; anything else is taken as projected, which changes how distances are computed.
cache_dir	Directory for cached downloads. Defaults to a session temporary directory, so nothing is written outside it unless asked.

Details

Column roles are detected from common names when not given explicitly, so agri_project(data) usually works unchanged. Detection is reported by the print method, and can always be overridden.

Value

An object of class agri_project.

See Also

add_climate(), build_features(), check_project()

Examples

```
d <- demo_agri_data(n_units = 6, n_seasons = 2)
p <- agri_project(d)
p
```

`build_features` *Build the model design matrix*

Description

Reduces every attached layer to one row per management unit and season, which is the key the whole package is organised around.

Usage

```
build_features(
  p,
  aggregation = c("phenology", "monthly", "season"),
  stats = c("mean", "sum", "min", "max"),
  stress = TRUE,
  heat_threshold = 30,
  dry_threshold = 1
)
```

Arguments

`p`	An `agri_project()` with at least one layer attached. For `aggregation = "phenology"`, `phenology_windows()` must have been run.
`aggregation`	How to group days within a season.
`stats`	Statistics to compute for each variable and group.
`stress`	Whether to derive stress-day counters.
`heat_threshold`	Daily maximum temperature, in degrees Celsius, above which a day counts as heat stress.
`dry_threshold`	Daily rainfall, in millimetres, below which a day counts as dry.

Details

Daily layers are aggregated within phenological stage by default, so that a feature such as `prcp_sum_grain_fill` carries the same meaning across sites that sowed weeks apart. Aggregating by calendar month instead is available for comparison, and is the usual practice in the literature; it is offered so the difference can be measured rather than assumed.

Stress counters are derived before aggregation: days above the heat threshold, days below freezing, dry days, and the longest dry spell in the season.

Value

The project, with `features` populated.

See Also

`phenology_windows()`, `check_project()`, `train_model()`

Examples

```
p <- agri_project(demo_agri_data(n_units = 6, n_seasons = 2))
p <- add_climate(p, source = "demo")
p <- phenology_windows(p)
p <- build_features(p)
dim(p$features)
```

`check_project` *Check a project for the faults that invalidate an analysis*

Description

Runs the checks that are cheap to automate and expensive to discover late.

Usage

```
check_project(p, mad_k = 3, max_missing = 0.2)
```

Arguments

`p`	An `agri_project()`.
`mad_k`	Number of median absolute deviations beyond which a value is flagged. Three is conventional.
`max_missing`	Proportion of missing values in a feature above which it is reported.

Details

The most important is the **leakage guard**: a covariate window that extends past the harvest it is supposed to predict produces a model that cannot be deployed and a skill estimate that means nothing. It is easy to introduce by fetching a fixed date range for every site, and hard to see afterwards.

Also checked: duplicated unit-season keys, coordinates outside plausible bounds, missingness by column, and univariate outliers by the median absolute deviation, which is resistant to the outliers it is looking for.

Value

A data frame of issues with columns `severity`, `check` and `detail`, invisibly returned and printed. Zero rows means every check passed.

See Also

`build_features()`, `train_model()`

Examples

```
p <- agri_project(demo_agri_data(n_units = 6, n_seasons = 2))
p <- add_climate(p, source = "demo")
check_project(p)
```

`crop_parameters` *Indicative crop thermal parameters*

Description

Base and upper temperatures, and cumulative growing degree days at the end of each phenological stage, for a small set of crops.

Usage

```
crop_parameters(crop = NULL)
```

Arguments

`crop`	Optional crop name. When `NULL`, all crops are returned.

Value

A data frame with columns `crop`, `t_base`, `t_upper`, `stage` and `gdd_end`, ordered by crop and cumulative thermal time.

Calibrate before trusting

These values are **indicative defaults for getting started, not calibrated constants**. Thermal requirements vary substantially with cultivar, photoperiod and region, and a stage boundary that is wrong by a fortnight will misattribute the weather a model sees. Supply your own thresholds through the `stages` argument of `phenology_windows()` for any analysis you intend to publish.

See Also

`growing_degree_days()`, `phenology_windows()`

Examples

```
crop_parameters("wheat")
unique(crop_parameters()$crop)
```

`demo_agri_data` *A demonstration agricultural data set*

Description

Generates a multi-field, multi-season data set whose yields are produced by a known process, for examples, tests and teaching.

Usage

```
demo_agri_data(n_units = 40, n_seasons = 4, crop = "maize", start_year = 2018)
```

Arguments

`n_units`	Number of management units.
`n_seasons`	Number of seasons per unit.
`crop`	Crop name, used for thermal parameters.
`start_year`	First season.

Details

The data are **simulated, not observed**. Yield is built from rainfall accumulated during grain fill, the count of days above 30 degrees during silking, soil clay content, a smooth spatial trend and a season effect. Because those drivers are known exactly, an analysis of this data set can be checked against the truth rather than against a previous run.

Generation is deterministic: no random number generator is used and the global RNG state is not touched, so repeated calls return identical data.

Value

A data frame with one row per unit and season, containing coordinates, the growing window, soil clay, yield, and the true driver values used to build it.

See Also

`agri_project()`, `add_climate()`

Examples

```
d <- demo_agri_data(n_units = 6, n_seasons = 2)
str(d)
```

`explain` *Explain a fitted model*

Description

Permutation importance and partial dependence, both computed so that the answer means what it appears to mean.

Usage

```
explain(
  object,
  method = c("importance", "pdp", "ale", "ice", "shap"),
  features = NULL,
  n_perm = 5,
  grid = 20,
  ...
)

## S3 method for class 'agri_model'
explain(
  object,
  method = c("importance", "pdp", "ale", "ice", "shap"),
  features = NULL,
  n_perm = 5,
  grid = 20,
  ...
)
```

Arguments

`object`	A model fitted by `train_model()`.
`method`	Which explanation to compute; see the section above.
`features`	Features to examine. Defaults to all for `"importance"` and to the five most important otherwise.
`n_perm`	Number of shuffles per feature and fold. The function uses the ambient random state; call `set.seed()` first for reproducibility.
`grid`	Number of grid points for partial dependence.
`...`	Ignored.

Details

Importance is measured **out of fold**: within each resampling fold, one feature of the held-out rows is shuffled and the increase in that fold's error is recorded. In-sample permutation importance on a flexible learner mostly measures how much the model was able to memorise, which is why it is not offered here.

Note that permutation importance is unreliable when features are strongly correlated, which climate features usually are: shuffling one of a pair of near-duplicates leaves the other to carry the signal, so both look unimportant. Treat the ranking as indicative and read it alongside partial dependence.

Value

A data frame. For `"importance"`, one row per feature with the mean and standard deviation of the error increase. For `"pdp"` and `"ale"`, one row per feature and grid value. For `"ice"`, one row per observation, feature and grid value. For `"shap"`, one row per observation and feature.

Which method answers which question

`"importance"` How much does the model rely on this feature? Measured out of fold, so it reflects generalisation rather than memorisation.

`"pdp"` What shape is the relationship, averaged over everything else? Misleading when features are strongly correlated, because it averages over combinations that never occur.

`"ale"` The same question, but accumulated over local differences within narrow windows of the feature, so it never evaluates the model on impossible combinations. Prefer it to partial dependence whenever the predictors are correlated, which for weather features they always are (Apley and Zhu, 2020).

`"ice"` Partial dependence for each observation separately. Curves that fan out reveal interactions that the averaged curve hides.

`"shap"` Exact tree SHAP values, attributing each prediction among the features. Requires the **treeshap** package and a tree-based learner.

References

Apley, D. W. & Zhu, J. (2020) "Visualizing the effects of predictor variables in black box supervised learning models." Journal of the Royal Statistical Society Series B 82, 1059-1086. doi:10.1111/rssb.12377

See Also

`train_model()`, `uncertainty()`

Examples

```
p <- agri_project(demo_agri_data(n_units = 12, n_seasons = 3))
p <- build_features(phenology_windows(add_climate(p)))
m <- train_model(p, "yield", algorithm = "lm", k = 3)
set.seed(1)
head(explain(m), 5)
head(explain(m, "ale", features = "prcp_sum_grain_fill", grid = 6))
```

`growing_degree_days` *Growing degree days*

Description

Daily thermal time by the capped-average method: the mean of the daily maximum and minimum, each first constrained to the interval between the base and upper temperatures, less the base temperature, with negative values set to zero.

Usage

```
growing_degree_days(tmin, tmax, t_base = 5, t_upper = Inf)
```

Arguments

`tmin`, `tmax`	Numeric vectors of daily minimum and maximum temperature in degrees Celsius.
`t_base`	Base temperature below which development is taken to stop.
`t_upper`	Temperature above which further warmth adds no development.

Details

This is the interpretation McMaster and Wilhelm (1997) label Method 1 with a horizontal cut-off. The choice matters: the two common interpretations of the same equation can differ by hundreds of degree days over a season, so the method is stated rather than left implicit.

Value

A numeric vector of daily growing degree days, never negative.

References

McMaster, G. S. & Wilhelm, W. W. (1997) "Growing degree-days: one equation, two interpretations." Agricultural and Forest Meteorology 87, 291-300. doi:10.1016/S01681923(97)000270

See Also

`crop_parameters()`, `phenology_windows()`

Examples

```
growing_degree_days(tmin = c(4, 8, 12), tmax = c(18, 24, 33), t_base = 5)
```

list_sources *List registered sources and learners*

Description

List registered sources and learners

Usage

```
list_sources()

list_learners()
```

Value

A data frame describing what is currently registered.

See Also

`register_source()`, `register_learner()`

Examples

```
list_sources()
list_learners()
```

phenology_windows *Derive phenological windows from accumulated thermal time*

Description

Labels every day of every growing season with the phenological stage the crop had reached by that date, based on accumulated growing degree days, and summarises the resulting windows.

Usage

```
phenology_windows(
  p,
  crop = NULL,
  stages = NULL,
  t_base = NULL,
  t_upper = NULL,
  layer = "climate"
)
```

Arguments

p	An agri_project() with a climate layer providing tmin and tmax.
crop	Crop name used to look up thermal parameters. When NULL, the project's crop column is used if present, otherwise "generic".
stages	Optional data frame of custom thresholds with columns stage and gdd_end, overriding crop_parameters().
t_base, t_upper	Optional overrides for the base and upper temperatures.
layer	Name of the climate layer to read.

Details

Aligning covariates to phenology rather than to the calendar is the point of the exercise. "Rainfall in September" means different things to two crops sown six weeks apart; "rainfall during grain fill" means the same thing to both. Aggregation in build_features() uses these labels.

Value

The project, with daily stage labels attached to the climate layer and a window summary available as p$windows.

See Also

growing_degree_days(), crop_parameters(), build_features()

Examples

```
p <- agri_project(demo_agri_data(n_units = 4, n_seasons = 2))
p <- add_climate(p, source = "demo")
p <- phenology_windows(p)
head(p$windows)
```

---

plot.agri_model *Plot a fitted model*

Description

Plot a fitted model

Usage

```
## S3 method for class 'agri_model'
plot(
  x,
  type = c("observed", "residuals", "importance"),
  top = 12,
  n_perm = 3,
  ...
)
```

Arguments

x	A model fitted by train_model().
type	"observed" plots out-of-fold predictions against the truth with a one to one line, which is the honest version of the usual fitted-versus-observed figure; "residuals" plots residuals against the prediction, to expose bias that a single skill number hides; "importance" draws out-of-fold permutation importance.
top	Number of features for "importance".
n_perm	Shuffles per feature for "importance".
...	Passed to plot.

Value

x, invisibly. Called for the plot.

See Also

plot_map(), plot_effect(), plot_uncertainty()

Examples

```
p <- agri_project(demo_agri_data(n_units = 20, n_seasons = 3))
p <- build_features(phenology_windows(add_climate(p)), stats = "sum")
m <- train_model(p, "yield", algorithm = "lm", k = 3)
plot(m)
plot(m, type = "residuals")
```

---

plot.agri_project *Plot a project's management units*

Description

Draws the units in space, sized by how many seasons each was observed in. Worth looking at before modelling: clustered units are the situation that makes random cross-validation misleading, and it is easier to see than to infer.

Usage

```
## S3 method for class 'agri_project'
plot(x, ...)
```

Arguments

x	An agri_project().
...	Passed to plot.

Value

x, invisibly. Called for the plot.

See Also

plot.agri_resample(), plot_map()

Examples

```
p <- agri_project(demo_agri_data(n_units = 20, n_seasons = 3))
plot(p)
```

---

plot.agri_resample *Plot a resampling scheme*

Description

Draws each unit coloured by the fold it is held out in, so that what the blocking actually did can be seen rather than assumed. Blocks that look interleaved are not blocking anything.

Usage

```
## S3 method for class 'agri_resample'
plot(x, ...)
```

Arguments

x	An resample_scheme() object.
...	Passed to plot.

Value

x, invisibly. Called for the plot.

See Also

resample_scheme(), plot.agri_project()

Examples

```
p <- agri_project(demo_agri_data(n_units = 24, n_seasons = 2))
p <- build_features(phenology_windows(add_climate(p)), stats = "sum")
plot(resample_scheme(p, method = "spatial_block", k = 4))
```

`plot_effect` *Plot a marginal effect*

Description

Draws accumulated local effects, partial dependence, or individual conditional expectation curves for one feature.

Usage

```
plot_effect(model, feature, method = c("ale", "pdp", "ice"), grid = 20, ...)
```

Arguments

`model`	A model fitted by `train_model()`.
`feature`	Name of the feature to show.
`method`	`"ale"`, `"pdp"` or `"ice"`.
`grid`	Number of grid points.
`...`	Passed to `plot`.

Details

Accumulated local effects is the default deliberately. Partial dependence averages the model over feature combinations that may never occur, which for correlated weather features it routinely does; ALE accumulates local differences instead and does not.

Value

The computed effect, invisibly.

See Also

`explain()`

Examples

```
p <- agri_project(demo_agri_data(n_units = 20, n_seasons = 3))
p <- build_features(phenology_windows(add_climate(p)), stats = "sum")
m <- train_model(p, "yield", algorithm = "lm", k = 3)
plot_effect(m, "prcp_sum_grain_fill")
```

plot_map *Map predictions, residuals or uncertainty*

Description

Draws each management unit at its coordinates, coloured by what the model produced there. The uncertainty map is the one worth reading: a model can have acceptable average skill and still be useless over part of its area, and only the map shows where.

Usage

```
plot_map(
  model,
  what = c("prediction", "residual", "uncertainty"),
  season = NULL,
  level = 0.9,
  ...
)
```

Arguments

model	A model fitted by train_model().
what	"prediction", "residual" (observed minus out-of-fold prediction, on a diverging scale centred at zero), or "uncertainty" (the conformal interval half-width, wider where the model is less sure).
season	Optional season to show. Defaults to the first, since overplotting several seasons at one location hides all but the last.
level	Coverage level for "uncertainty".
...	Passed to plot.

Value

A data frame of the plotted values, invisibly.

See Also

plot.agri_model(), uncertainty()

Examples

```
p <- agri_project(demo_agri_data(n_units = 24, n_seasons = 2))
p <- build_features(phenology_windows(add_climate(p)), stats = "sum")
m <- train_model(p, "yield", algorithm = "lm", k = 4)
plot_map(m, "residual")
```

plot_uncertainty *Plot prediction intervals and their coverage*

Description

Observations are sorted by prediction and drawn with their conformal interval, with the ones the interval missed picked out. A calibrated interval should miss about `1 - level` of them, scattered rather than concentrated at one end.

Usage

```
plot_uncertainty(model, level = 0.9, ...)
```

Arguments

`model`	A model fitted by `train_model()`.
`level`	Coverage level.
`...`	Passed to `plot`.

Value

A data frame of predictions, bounds and whether each was covered, invisibly.

See Also

`uncertainty()`, `predict.agri_model()`

Examples

```
p <- agri_project(demo_agri_data(n_units = 20, n_seasons = 3))
p <- build_features(phenology_windows(add_climate(p)), stats = "sum")
m <- train_model(p, "yield", algorithm = "lm", k = 3)
plot_uncertainty(m)
```

predict.agri_model *Predict from a fitted model*

Description

Predict from a fitted model

Usage

```
## S3 method for class 'agri_model'
predict(object, newdata = NULL, interval = FALSE, level = 0.9, ...)
```

Arguments

object	A model fitted by train_model().
newdata	Optional data frame of features. When omitted, predictions for the training rows are returned, alongside the out-of-fold predictions, which are the honest ones.
interval	Attach conformal prediction intervals.
level	Coverage level for the interval.
...	Ignored.

Value

A data frame with the keys, .pred, and when newdata is omitted .pred_oof; plus .lower and .upper when interval = TRUE.

See Also

uncertainty()

Examples

```
p <- agri_project(demo_agri_data(n_units = 12, n_seasons = 3))
p <- build_features(phenology_windows(add_climate(p)))
m <- train_model(p, "yield", algorithm = "lm", k = 3)
head(predict(m, interval = TRUE))
```

---

register_learner *Register a learning algorithm*

Description

Adds an algorithm to the learner registry so that train_model() can select it by name. Keeping learners behind a registry is what allows the package to delegate to established implementations rather than reimplement them.

Usage

```
register_learner(name, fit, predict, requires = character(), description = "")
```

Arguments

name	Name used to select the learner.
fit	A function function(x, y, ...) returning a fitted model, where x is a numeric data frame of predictors and y a numeric response.
predict	A function function(object, newx, ...) returning a numeric vector of predictions.
requires	Character vector of packages the learner needs.
description	A one-line human description.

Value

Invisibly, the registered name.

See Also

list_learners(), register_source()

Examples

```
register_learner("mean_only",
                 fit = function(x, y, ...) mean(y),
                 predict = function(object, newx, ...) {
                     rep(object, nrow(newx))
                 })
"mean_only" %in% list_learners()$name
```

---

register_source *Register a covariate source*

Description

Adds a data provider to the source registry, making it available to add_climate(), add_soil() and add_satellite(). This is the extension point for new providers: no change to the package is needed.

Usage

```
register_source(
  name,
  fetch,
  provides,
  kind = c("series", "static"),
  requires_network = TRUE,
  description = ""
)
```

Arguments

name	Name used to select the source, for example "power".
fetch	A function with signature function(units, seasons, ...) returning a data frame. Series sources must return one row per unit and date, with columns unit_id and date; static sources one row per unit, with column unit_id.
provides	Character vector of the variables the source returns.
kind	Either "series" for time-varying data or "static" for values fixed within a unit.

`requires_network` Whether the source needs internet access. Sources that do are skipped in tests and examples.

`description` A one-line human description.

Value

Invisibly, the registered name.

See Also

`list_sources()`, `register_learner()`

Examples

```
register_source("flat",
                fetch = function(units, seasons, ...) {
                    data.frame(unit_id = units$unit_id, elevation = 100)
                },
                provides = "elevation", kind = "static",
                requires_network = FALSE)
"flat" %in% list_sources()$name
```

---

report *Write a model card*

Description

Turns the provenance ledger and the validation results into a model card in Markdown: what was fitted, on what data, validated how, with which caveats.

Usage

```
report(object, file = NULL, top = 10, ...)

## S3 method for class 'agri_model'
report(object, file = NULL, top = 10, ...)
```

Arguments

`object` A model fitted by `train_model()`.

`file` Optional path to write to. The text is always returned.

`top` Number of features to list.

`...` Ignored.

Details

The limitations section is generated from the model's own diagnostics rather than written by hand, so it cannot fall out of step with the results. If random cross-validation would have overstated the model, the card says so.

Value

A character vector of Markdown lines, invisibly when written to file.

See Also

`train_model()`, `provenance()`

Examples

```
p <- agri_project(demo_agri_data(n_units = 12, n_seasons = 3))
p <- build_features(phenology_windows(add_climate(p)))
m <- train_model(p, "yield", algorithm = "lm", k = 3)
cat(head(report(m), 20), sep = "\n")
```

---

resample_scheme *Build a resampling scheme*

Description

Constructs the train and test splits used to estimate predictive skill.

Usage

```
resample_scheme(
  p,
  method = c("spatial_block", "leave_location_out", "forward_season", "random"),
  k = 5,
  buffer = 0
)
```

Arguments

p	An `agri_project()` with features built.
method	`"spatial_block"` groups units into compact blocks by k-means on their coordinates and holds out whole blocks; `"leave_location_out"` holds out whole units; `"forward_season"` trains on past seasons and tests on the next, never using the future to predict the past; `"random"` ignores structure entirely.
k	Number of folds.
buffer	Exclude training rows whose unit lies within this distance of any test unit. Kilometres when the project is geographic, otherwise coordinate units. Buffering removes the residual optimism that blocking alone leaves at block edges.

Value

An object of class `agri_resample`.

Why the default is spatial

Agricultural observations near one another share weather, soil and management. Under random k-fold cross-validation a test point almost always has a near-duplicate in the training set, so the estimate answers "how well does this interpolate between my own plots" when the question asked is usually "how well does this predict somewhere new". The gap between the two is often large. Roberts et al. (2017) set out the problem and the blocking remedies; Meyer and Pebesma (2021) show how far a model can be trusted outside the space it was trained in.

`"random"` remains available, and `train_model()` reports both so the difference can be quantified rather than argued about.

References

Roberts, D. R. et al. (2017) "Cross-validation strategies for data with temporal, spatial, hierarchical, or phylogenetic structure." Ecography 40, 913-929. doi:10.1111/ecog.02881

Meyer, H. & Pebesma, E. (2021) "Predicting into unknown space? Estimating the area of applicability of spatial prediction models." Methods in Ecology and Evolution 12, 1620-1633. doi:10.1111/2041210X.13650

See Also

`train_model()`

Examples

```
p <- agri_project(demo_agri_data(n_units = 12, n_seasons = 2))
p <- add_climate(p, source = "demo")
p <- phenology_windows(p)
p <- build_features(p)
resample_scheme(p, method = "spatial_block", k = 3)
```

`stack_weights` *Stack weights from a fitted ensemble*

Description

Reports how much each base learner contributed to a `"stack"` model, which is usually more informative than the ensemble's accuracy alone: a stack that puts all its weight on one learner is telling you the others added nothing.

Usage

```
stack_weights(model)
```

Arguments

model	A model fitted by train_model() with algorithm = "stack".

Value

A named numeric vector of weights summing to one.

See Also

train_model(), list_learners()

Examples

```
p <- agri_project(demo_agri_data(n_units = 14, n_seasons = 3))
p <- build_features(phenology_windows(add_climate(p)), stats = "sum")
set.seed(1)
m <- train_model(p, "yield", algorithm = "stack", k = 3,
                 compare_random = FALSE)
stack_weights(m)
```

---

train_model *Train and honestly validate a model*

Description

Fits a model to predict target from the features built for the project, estimating skill by spatial resampling rather than random folds.

Usage

```
train_model(
  p,
  target,
  algorithm = "auto",
  resampling = NULL,
  method = "spatial_block",
  k = 5,
  buffer = 0,
  compare_random = TRUE,
  ...
)
```

Arguments

p	An agri_project() with features built.
target	Name of the response column in the project's observations.
algorithm	A registered learner, or "auto" to use "ranger" when available and "lm" otherwise. See list_learners().
resampling	A resample_scheme() object. Built automatically when NULL.
method, k, buffer	Passed to resample_scheme() when it builds the scheme itself.
compare_random	Also evaluate with random folds, to quantify the optimism that random cross-validation would have introduced.
...	Passed to the learner's fit function.

Details

By default the same model is also evaluated with random folds and both results are reported. The difference between them is the amount by which random cross-validation would have overstated the model, and it is worth knowing before any figure is published.

Missing predictor values are filled with the median of the training rows of each fold, computed inside the fold so that no information crosses the split.

Value

An object of class agri_model.

See Also

resample_scheme(), explain(), uncertainty(), report()

Examples

```
p <- agri_project(demo_agri_data(n_units = 12, n_seasons = 3))
p <- add_climate(p, source = "demo")
p <- phenology_windows(p)
p <- build_features(p)
m <- train_model(p, target = "yield", algorithm = "lm", k = 3)
m
```

---

uncertainty *Prediction intervals by split conformal inference*

Description

Turns the out-of-fold residuals into prediction intervals with a distribution-free finite-sample coverage guarantee.

Usage

```
uncertainty(object, level = 0.9, ...)

## S3 method for class 'agri_model'
uncertainty(object, level = 0.9, ...)
```

Arguments

`object`	A model fitted by `train_model()`.
`level`	Target coverage, between 0 and 1.
`...`	Ignored.

Details

Conformal inference is used because it is the only interval method that works uniformly across a registry of arbitrary learners: it assumes nothing about the model or the error distribution, only that the calibration and prediction data are exchangeable. Because the residuals come from **spatial** resampling, the resulting intervals inherit that honesty; intervals calibrated on random folds would be too narrow for the same reason random cross-validation is too optimistic.

Value

An object of class `agri_uncertainty` giving the interval half-width, the target level and the estimated empirical coverage.

Reported coverage

The half-width and the coverage check cannot come from the same residuals without being circular. Coverage is therefore estimated by two-fold splitting of the residual vector: calibrate on one half, measure on the other, and average the two directions.

References

Lei, J., G'Sell, M., Rinaldo, A., Tibshirani, R. J. & Wasserman, L. (2018) "Distribution-free predictive inference for regression." Journal of the American Statistical Association 113, 1094-1111. doi:10.1080/01621459.2017.1307116

See Also

`train_model()`, `predict.agri_model()`

Examples

```
p <- agri_project(demo_agri_data(n_units = 12, n_seasons = 3))
p <- build_features(phenology_windows(add_climate(p)))
m <- train_model(p, "yield", algorithm = "lm", k = 3)
uncertainty(m)
```

`units_of` *Accessors for project components*

Description

Accessors for project components

Usage

```
units_of(p)

seasons_of(p)

provenance(p)
```

Arguments

`p`	An `agri_project()`.

Value

`units_of()` returns one row per management unit with its coordinates; `seasons_of()` one row per unit and season with the growing window; `provenance()` the ledger of operations applied so far.

Examples

```
p <- agri_project(demo_agri_data(n_units = 4, n_seasons = 2))
units_of(p)
head(seasons_of(p))
```

Index