Package 'bayesiansurpriser'

April 21, 2026

Title Bayesian Surprise for De-Biasing Thematic Maps

Version 0.1.0

Author Dmitry Shkolnik [aut, cre]

Maintainer Dmitry Shkolnik <shkolnikd@gmail.com>

Description Implements Bayesian Surprise methodology for data visualization, based on Correll and Heer (2017) <doi:10.1109/TVCG.2016.2598839> ``Surprise! Bayesian Weighting for De-Biasing Thematic Maps". Provides tools to weight event data relative to spatio-temporal models, highlighting unexpected patterns while de-biasing against known factors like population density or sampling variation. Integrates seamlessly with 'sf' for spatial data and 'ggplot2' for visualization. Supports temporal/streaming data analysis.

License MIT + file LICENSE

URL https://dshkol.github.io/bayesiansurpriser/, https://github.com/dshkol/bayesiansurpriser

BugReports https://github.com/dshkol/bayesiansurpriser/issues

Encoding UTF-8

RoxygenNote 7.3.2

Depends R (>= 4.1.0)

Imports ggplot2 (>= 3.5.0), sf (>= 1.0.0), scales (>= 1.3.0), rlang (>= 1.1.0), cli, stats, MASS, RColorBrewer

Suggests testthat (>= 3.0.0), knitr, rmarkdown, dplyr, tibble, vdiffr, tidycensus, tigris, cancensus, ggrepel

Config/testthat/edition 3

VignetteBuilder knitr

LazyData true

NeedsCompilation no

Repository CRAN

Date/Publication 2026-04-21 20:52:29 UTC

Contents

`add_model` *Add Model to Space*

Description

Adds a new model to an existing model space.

Usage

```
add_model(model_space, model, prior_weight = NULL)
```

Arguments

`model_space`	A `bs_model_space` object
`model`	A `bs_model` object to add
`prior_weight`	Prior probability for the new model. The existing priors are rescaled to accommodate.

Value

Updated `bs_model_space`

Examples

```
space <- model_space(bs_model_uniform())
space <- add_model(space, bs_model_gaussian(), prior_weight = 0.3)
```

auto_surprise *Compute Surprise with Automatic Model Selection*

Description

Simplified interface that automatically selects appropriate models and computes per-observation surprise from the model priors.

Usage

```
auto_surprise(
  observed,
  expected = NULL,
  sample_size = NULL,
  include_gaussian = FALSE,
  include_sampled = FALSE,
  signed = TRUE,
  ...
)
```

Arguments

observed	Numeric vector of observed values
expected	Numeric vector of expected values (optional)
sample_size	Numeric vector of sample sizes (optional)
include_gaussian	Include Gaussian model?
include_sampled	Include KDE model?
signed	Compute signed surprise?
...	Additional arguments

Value

A bs_surprise object

Examples

```
observed <- c(50, 100, 150, 200, 75)
expected <- c(10000, 50000, 100000, 25000, 15000)
result <- auto_surprise(observed, expected)
```

`bayesian_update` *Bayesian Update of Model Space*

Description

Updates the prior probability distribution over models given observed data, using Bayes' rule.

Usage

```
bayesian_update(model_space, observed, region_idx = NULL, ...)
```

Arguments

`model_space`	A `bs_model_space` object
`observed`	Numeric vector of observed values
`region_idx`	Optional integer index for region-specific likelihood
`...`	Additional arguments passed to likelihood functions

Details

Applies Bayes' rule:

$$P(M|D) \propto P(D|M) \cdot P(M)$$

where P(D|M) is the likelihood of data D given model M, and P(M) is the prior.

Value

Updated `bs_model_space` with posterior probabilities

Examples

```
# Create a model space
space <- model_space(
  bs_model_uniform(),
  bs_model_gaussian()
)

# Update with observed data
observed <- c(10, 20, 30, 40, 50)
updated <- bayesian_update(space, observed)
print(updated)
```

`bs_model_baserate` *Create a Base Rate Model*

Description

Creates a model that compares observed events to expected rates based on a known baseline (e.g., population). This addresses "base rate bias" where patterns in visualizations are dominated by underlying factors like population density.

Usage

```
bs_model_baserate(expected, normalize = TRUE, name = NULL)
```

Arguments

`expected`	Numeric vector of expected values or proportions. E.g., population counts, area sizes, or any prior expectation.
`normalize`	Logical; normalize expected to sum to 1?
`name`	Optional name for the model

Details

Under the base rate model, expected proportions are defined by the `expected` vector. The likelihood measures how well observed data matches these expected proportions:

$$P(D|BaseRate) = 1 - \frac{1}{2}\sum_i |O_i - E_i|$$

For example, if region A has 10% of the population, we expect 10% of events. Regions with event rates matching their population share show low surprise; regions with disproportionate rates show high surprise.

This is the primary tool for de-biasing choropleth maps.

Value

A `bs_model_baserate` object

Examples

```
# Population-weighted base rate
population <- c(10000, 50000, 100000, 25000)
model <- bs_model_baserate(population)

# Use in model space
space <- model_space(
  bs_model_uniform(),
  bs_model_baserate(population)
)
```

bs_model_baserate_col *Create Base Rate Model from Column*

Description

Convenience function to create a base rate model from a data frame column.

Usage

```
bs_model_baserate_col(data, column, ...)
```

Arguments

data	Data frame or sf object
column	Name of column containing expected values
...	Additional arguments passed to bs_model_baserate()

Value

A bs_model_baserate object

Examples

```
df <- data.frame(expected = c(100, 200, 300))
model <- bs_model_baserate_col(df, "expected")
model$name
```

bs_model_bootstrap *Create a Bootstrap Sample Model*

Description

Creates a model based on a bootstrap sample of the data. Useful for assessing variability and identifying observations that are surprising relative to resampled distributions.

Usage

```
bs_model_bootstrap(n_bootstrap = 100, seed = NULL, name = NULL)
```

Arguments

n_bootstrap	Number of bootstrap samples to use
seed	Random seed for reproducibility
name	Optional name for the model

Value

A bs_model_bootstrap object

Examples

```
model <- bs_model_bootstrap(n_bootstrap = 100)
```

---

bs_model_funnel *Create a de Moivre Funnel Model*

Description

Creates a model that normalizes observations by their expected standard error, accounting for varying sample sizes. This addresses "sampling error bias" where regions with small sample sizes show artificially high variability.

Usage

```
bs_model_funnel(
  sample_size,
  target_rate = NULL,
  type = c("count", "proportion"),
  formula = c("paper", "poisson"),
  control_limits = c(2, 3),
  name = NULL
)
```

Arguments

sample_size	Numeric vector of sample sizes (e.g., population)
target_rate	Target rate/proportion. If NULL, estimated from data.
type	Type of data: "count" (Poisson) or "proportion" (binomial)
formula	Formula for likelihood computation: • "paper" (default): Uses the funnel score from the paper's unemployment data (dM = Z * sqrt(pop_frac)) and converts it to a two-tailed normal tail probability. • "poisson": Uses Poisson-based standard error and converts the resulting z-score to a two-tailed normal tail probability.
control_limits	Numeric vector of control limits (in SDs) for funnel plot. Default is c(2, 3) for warning and control limits.
name	Optional name for the model

Details

The de Moivre funnel model uses the insight that sampling variability decreases with sample size according to de Moivre's equation:

$$SE = \sigma/\sqrt{n}$$

With formula = "paper": The model uses the formula that matches the paper's unemployment reference data:

$$Z = (rate - mean_rate)/stddev_rate$$

$$dM = Z \times \sqrt{population/total\_population}$$

$$P(D|M) = 2 \times \Phi(-|dM|)$$

With formula = "poisson": For count data (Poisson), the model computes z-scores as:

$$z = (observed - expected)/\sqrt{expected}$$

For proportion data (binomial):

$$z = (observed - expected)/\sqrt{p(1-p)/n}$$

Observations with large z-scores (far from expected after accounting for sample size) are genuinely surprising, while high rates in small regions are discounted as expected variation.

This model is essential for:

- De-biasing per-capita rate maps
- Creating funnel plots
- Identifying genuine outliers vs. sampling noise

Value

A `bs_model_funnel` object

Examples

```
# Population sizes for regions
population <- c(10000, 50000, 100000, 25000)

# Funnel model using the paper's unemployment-reference formula
model <- bs_model_funnel(population, formula = "paper")

# Funnel model with known target rate
model <- bs_model_funnel(population, target_rate = 0.001)

# For proportion data with Poisson-based formula
model <- bs_model_funnel(population, type = "proportion", formula = "poisson")
```

bs_model_funnel_col *Create Funnel Model from Column*

Description

Convenience function to create a funnel model from a data frame column.

Usage

```
bs_model_funnel_col(data, column, ...)
```

Arguments

data	Data frame or sf object
column	Name of column containing sample sizes
...	Additional arguments passed to bs_model_funnel()

Value

A bs_model_funnel object

Examples

```
df <- data.frame(sample_size = c(1000, 2000, 3000))
model <- bs_model_funnel_col(df, "sample_size")
model$name
```

bs_model_gaussian *Create a Gaussian Model*

Description

Creates a model based on a Gaussian (normal) distribution. This parametric model is useful for detecting outliers and identifying multiple modes in data.

Usage

```
bs_model_gaussian(mu = NULL, sigma = NULL, fit_from_data = TRUE, name = NULL)
```

Arguments

mu	Mean of the Gaussian. If NULL, estimated from data.
sigma	Standard deviation. If NULL, estimated from data.
fit_from_data	Logical; estimate parameters from data?
name	Optional name for the model

Details

The Gaussian model assumes data is drawn from a normal distribution. Points far from the mean (in terms of standard deviations) will have low likelihood and thus create high surprise when this model has probability mass.

This model is particularly useful for:

- Detecting spatial outliers
- Identifying multi-modal distributions
- Combating renormalization bias (outliers get suppressed in dynamic visualizations)

Value

A `bs_model_gaussian` object

Examples

```
# Gaussian model with parameters fit from data
model <- bs_model_gaussian()

# Gaussian model with fixed parameters
model <- bs_model_gaussian(mu = 100, sigma = 20)

# Use in model space with other models
population <- c(10000, 50000, 100000, 25000)
space <- model_space(
  bs_model_uniform(),
  bs_model_baserate(population),
  bs_model_gaussian()
)
```

---

bs_model_gaussian_mixture

Create Multi-Modal Gaussian Mixture Model

Description

Creates a model based on a mixture of Gaussians for multi-modal data.

Usage

```
bs_model_gaussian_mixture(means, sds, weights = NULL, name = NULL)
```

Arguments

`means`	Vector of means for each component
`sds`	Vector of standard deviations for each component
`weights`	Vector of mixture weights (must sum to 1)
`name`	Optional name for the model

Value

A bs_model_gaussian_mixture object

Examples

```
# Two-component mixture
model <- bs_model_gaussian_mixture(
  means = c(50, 150),
  sds = c(10, 20),
  weights = c(0.7, 0.3)
)
```

---

bs_model_sampled *Create a Sampled Subset Model (KDE)*

Description

Creates a non-parametric model using kernel density estimation (KDE). This model is built from a sample of the data and can detect when subsequent observations deviate from the pattern established by early data.

Usage

```
bs_model_sampled(
  sample_frac = NULL,
  kernel = "gaussian",
  bandwidth = "nrd0",
  n_grid = 512,
  sample_indices = NULL,
  name = NULL
)
```

Arguments

sample_frac	Fraction of data to use for building the prior (0 < x < 1). If NULL, uses all data for density estimation.
kernel	Kernel type for density estimation. One of: "gaussian", "epanechnikov", "rectangular", "triangular", "biweight", "cosine", "optcosine"
bandwidth	Bandwidth selection method or numeric value. If character, one of: "nrd0", "nrd", "ucv", "bcv", "SJ". If numeric, used directly as bandwidth.
n_grid	Number of points in the density estimation grid (default: 512). Higher values give smoother estimates but use more memory.
sample_indices	Integer vector of specific indices to use for building prior. Overrides sample_frac if provided.
name	Optional name for the model

Details

The sampled model builds a density estimate from a subset of observations (typically early observations in temporal data) and measures surprise as deviation from this learned distribution.

This is useful for:

- Detecting temporal changes in distribution
- Building a "post hoc" model from initial observations
- Detecting emerging patterns in streaming data

The likelihood for each observation is the density at that point under the KDE built from the sample.

Value

A `bs_model_sampled` object

Examples

```
# KDE model using first 10% of data
model <- bs_model_sampled(sample_frac = 0.1)

# KDE with specific bandwidth
model <- bs_model_sampled(bandwidth = 5)

# Use specific observations for training
model <- bs_model_sampled(sample_indices = 1:10)
```

`bs_model_uniform` *Create a Uniform Model*

Description

Creates a model that assumes events are equally likely across all regions. This serves as a "null hypothesis" baseline - regions where events cluster will show high surprise under this model.

Usage

```
bs_model_uniform(n_regions = NULL)
```

Arguments

`n_regions`	Number of regions (optional, inferred from data if NULL)

Details

Under the uniform model, expected probability for each region is 1/n, where n is the total number of regions. The likelihood is computed as:

$$P(D|Uniform) = 1 - \frac{1}{2}\sum_i |O_i - \frac{1}{n}|$$

This is the Total Variation Distance from uniform, transformed to a probability.

The uniform model is useful for detecting spatial clustering - any concentration of events in fewer regions will produce high surprise.

Value

A `bs_model_uniform` object

Examples

```
# Create uniform model
model <- bs_model_uniform()

# The model computes likelihood when used in a model space
space <- model_space(model)
```

`canada_mischief` *Canadian Mischief Crime Data by Province*

Description

Crime data for Canadian provinces and territories, showing mischief offenses. This dataset is adapted from the original Bayesian Surprise paper's Canada example and is useful for exploring base-rate effects: Ontario and Quebec dominate raw counts because of population, while model-based surprise scores ask which provinces are unusual relative to the chosen model space.

Usage

```
canada_mischief
```

Format

A data frame with 13 rows and 6 variables:

name Province or territory name

population Population count

mischief_count Number of mischief offenses

rate_per_100k Mischief rate per 100,000 population

pop_proportion Proportion of total Canadian population

mischief_proportion Proportion of total mischief offenses

Source

Correll & Heer (2017). Surprise! Bayesian Weighting for De-Biasing Thematic Maps. IEEE InfoVis.

Examples

```
data(canada_mischief)

# Basic exploration
head(canada_mischief)

# Compute surprise
result <- auto_surprise(
  observed = canada_mischief$mischief_count,
  expected = canada_mischief$population
)

# See which provinces are most surprising under the selected models
canada_mischief$surprise <- result$surprise
canada_mischief$signed_surprise <- result$signed_surprise
canada_mischief[order(-abs(canada_mischief$signed_surprise)), ]
```

---

compute_funnel_data *Compute Funnel Plot Data*

Description

Computes the data needed to create a funnel plot, including control limits.

Usage

```
compute_funnel_data(
  observed,
  sample_size,
  target_rate = NULL,
  type = c("count", "proportion"),
  limits = c(2, 3)
)
```

Arguments

observed	Numeric vector of observed values
sample_size	Numeric vector of sample sizes
target_rate	Target rate. If NULL, computed from data.
type	Type of data: "count" or "proportion"
limits	Vector of SD multiples for control limits (default: c(2, 3))

Value

A data frame with columns: observed, sample_size, expected, z_score, and control limit columns (lower_2sd, upper_2sd, lower_3sd, upper_3sd, etc.)

Examples

```
observed <- c(50, 100, 150, 200)
sample_size <- c(10000, 50000, 100000, 25000)
funnel_data <- compute_funnel_data(observed, sample_size)
```

---

compute_surprise *Compute Per-Region Surprise*

Description

Computes the surprise (KL-divergence) for each observation/region, measuring how much each data point updates beliefs about the model space.

Usage

```
compute_surprise(
  model_space,
  observed,
  expected = NULL,
  return_signed = TRUE,
  return_posteriors = FALSE,
  return_contributions = FALSE,
  normalize_posterior = TRUE,
  ...
)
```

Arguments

model_space	A bs_model_space object
observed	Numeric vector of observed values (one per region)
expected	Numeric vector of expected values (optional, for signed surprise)
return_signed	Logical; compute signed surprise?
return_posteriors	Logical; return per-region posteriors?
return_contributions	Logical; return per-model contributions?
normalize_posterior	Logical; if TRUE (default), normalizes posteriors to sum to 1 before computing KL divergence. This is the standard Bayesian Surprise calculation. If FALSE, uses unnormalized posterior weights (P(D|M) * P(M)) for comparison with the original Correll & Heer JavaScript demo output.
...	Additional arguments passed to likelihood functions

Details

For each region i, computes:

1. The posterior P(M|D_i) given just that region's data
2. The KL-divergence from prior to posterior (surprise)
3. Optionally, the sign based on deviation direction

`normalize_posterior = FALSE` is a legacy replication mode for the original JavaScript demo's per-region map calculation. It is not a proper KL divergence between probability distributions and should not be used as the default method for new analyses.

Value

A `bs_surprise` object

`cumulative_bayesian_update` *Global Bayesian Update Across All Regions*

Description

Performs a cumulative Bayesian update, updating the prior after each observation. This is useful for streaming/temporal data.

Usage

```
cumulative_bayesian_update(model_space, observed, ...)
```

Arguments

`model_space`	A `bs_model_space` object
`observed`	Numeric vector of observed values (in order)
`...`	Additional arguments passed to likelihood functions

Value

A list containing:

- `final_space`: The model space after all updates
- `cumulative_surprise`: Vector of cumulative surprise after each observation
- `posterior_history`: Matrix of posteriors after each update

`default_model_space` *Default Model Space*

Description

Creates a default model space suitable for most choropleth/thematic maps. Includes uniform, base rate, and funnel models.

Usage

```
default_model_space(
  expected,
  sample_size = expected,
  include_gaussian = FALSE,
  prior = NULL
)
```

Arguments

`expected`	Numeric vector of expected values (e.g., population). Used for base rate and funnel models.
`sample_size`	Numeric vector of sample sizes. Defaults to `expected` if not provided.
`include_gaussian`	Logical; include Gaussian model?
`prior`	Prior probabilities for models. Default is uniform.

Value

A `bs_model_space` object

Examples

```
# Default space with population as expected
population <- c(10000, 50000, 100000, 25000)
space <- default_model_space(population)

# Include Gaussian model
space <- default_model_space(population, include_gaussian = TRUE)
```

`example_counties` *Example County Data with Simulated Events*

Description

A simulated dataset of 50 counties with population and event counts. Some counties are designated as "hot spots" (higher than expected rates) and "cold spots" (lower than expected rates) for testing and examples.

Usage

```
example_counties
```

Format

A data frame with 50 rows and 7 variables:

county_id Unique county identifier

name County name

population Population count

events Number of events (e.g., crimes, incidents)

expected Expected number of events based on population

is_hotspot Logical; TRUE if county has elevated rates

is_coldspot Logical; TRUE if county has suppressed rates

Examples

```
data(example_counties)

# Compute surprise
result <- auto_surprise(
  observed = example_counties$events,
  expected = example_counties$population
)

example_counties$surprise <- result$surprise

# Hot spots and cold spots should have higher surprise
with(example_counties, tapply(surprise, is_hotspot, mean))
with(example_counties, tapply(surprise, is_coldspot, mean))
```

funnel_pvalue *Compute P-Value from Funnel Z-Score*

Description

Converts z-scores to two-tailed p-values under the normal distribution.

Usage

```
funnel_pvalue(z)
```

Arguments

z	Numeric vector of z-scores

Value

Numeric vector of p-values

Examples

```
z <- c(-2, -1, 0, 1, 2)
funnel_pvalue(z)
```

funnel_zscore *Funnel Z-Score (de Moivre)*

Description

Computes z-scores accounting for sampling variability using de Moivre's equation. This normalizes observed values by their expected standard error.

Usage

```
funnel_zscore(observed, expected, sample_size, type = c("count", "proportion"))
```

Arguments

observed	Numeric vector of observed counts or rates
expected	Numeric vector of expected values (e.g., from base rate)
sample_size	Numeric vector of sample sizes (e.g., population)
type	Type of data: "count" (Poisson) or "proportion" (binomial)

Details

For count data (Poisson), SE = sqrt(expected). For proportion data (binomial), SE = sqrt(p * (1-p) / n).

Larger sample sizes result in smaller standard errors, so the same deviation is more "surprising" for larger samples.

Value

Numeric vector of z-scores

Examples

```
# Observed crimes vs expected (from overall rate)
observed <- c(50, 100, 150)
expected <- c(45, 95, 160)
sample_size <- c(10000, 50000, 100000)
funnel_zscore(observed, expected, sample_size, type = "count")
```

---

geom_surprise *Surprise Map Geom*

Description

A convenience geom that combines `stat_surprise` with `geom_sf` for easy creation of surprise maps.

Usage

```
geom_surprise(
  mapping = NULL,
  data = NULL,
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  fill_type = c("surprise", "signed"),
  models = c("uniform", "baserate", "funnel"),
  color = NA,
  colour = color,
  linewidth = 0.1,
  ...
)
```

Arguments

mapping	Aesthetic mapping created by ggplot2::aes(). Required aesthetics are geometry (from sf) and observed. Optional aesthetics include expected and sample_size.
data	Data (typically an sf object)
position	Position adjustment
na.rm	Remove NA values?
show.legend	Show legend?
inherit.aes	Inherit aesthetics from ggplot?
fill_type	Type of surprise for fill aesthetic: • "surprise": Unsigned surprise (always positive) • "signed": Signed surprise (positive = higher than expected, negative = lower than expected)
models	Character vector of model types to use. Options: "uniform", "baserate", "gaussian", "sampled", "funnel"
color, colour	Border color for polygons
linewidth	Border line width
...	Additional arguments passed to the layer

Value

A list of ggplot2 layers

Aesthetics

geom_surprise understands the following aesthetics:

geometry sf geometry column (required)

observed Observed values (required)

expected Expected values (optional, for base rate model)

sample_size Sample sizes (optional, for funnel model)

fill Mapped to surprise by default

color/colour Border color

Examples

```
library(ggplot2)
library(sf)

nc <- st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)

# Basic surprise map - geometry must be mapped explicitly
ggplot(nc) +
  geom_surprise(aes(geometry = geometry, observed = SID74, expected = BIR74)) +
  scale_fill_surprise()
```

```
# Signed surprise with diverging scale
ggplot(nc) +
  geom_surprise(
    aes(geometry = geometry, observed = SID74, expected = BIR74),
    fill_type = "signed"
  ) +
  scale_fill_surprise_diverging()

# Customize appearance
ggplot(nc) +
  geom_surprise(
    aes(geometry = geometry, observed = SID74, expected = BIR74),
    color = "white",
    linewidth = 0.2
  ) +
  scale_fill_surprise() +
  theme_minimal()
```

`geom_surprise_density` *Surprise Density Plot*

Description

Creates a density plot of surprise values.

Usage

```
geom_surprise_density(
  mapping = NULL,
  data = NULL,
  which = c("surprise", "signed_surprise"),
  fill = "#4575b4",
  color = "black",
  ...
)
```

Arguments

`mapping`	Aesthetic mapping
`data`	Data with surprise values
`which`	Which surprise to plot: "surprise" or "signed_surprise"
`fill`	Fill color
`color`	Border color
`...`	Additional arguments passed to `ggplot2::geom_histogram()`

Value

A ggplot2 layer

`geom_surprise_histogram` *Surprise Histogram*

Description

Creates a histogram of surprise values.

Usage

```
geom_surprise_histogram(
  mapping = NULL,
  data = NULL,
  which = c("surprise", "signed_surprise"),
  bins = 30,
  fill = "#4575b4",
  color = "white",
  ...
)
```

Arguments

`mapping`	Aesthetic mapping
`data`	Data with surprise values
`which`	Which surprise to plot: "surprise" or "signed_surprise"
`bins`	Number of bins
`fill`	Fill color
`color`	Border color
`...`	Additional arguments passed to `ggplot2::geom_histogram()`

Value

A ggplot2 layer

Examples

```
library(ggplot2)
library(sf)

nc <- st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
nc_surprise <- surprise(nc, observed = SID74, expected = BIR74)

ggplot(nc_surprise) +
  geom_surprise_histogram()
```

get_model_space *Get the model space from a surprise result*

Description

Get the model space from a surprise result

Usage

```
get_model_space(x, ...)
```

Arguments

x	A bs_surprise or bs_surprise_sf object
...	Additional arguments (unused)

Value

A bs_model_space object

get_surprise *Extract surprise values from result objects*

Description

Extract surprise values from result objects

Usage

```
get_surprise(x, type = c("surprise", "signed"), ...)
```

Arguments

x	A bs_surprise, bs_surprise_sf, or bs_surprise_temporal object
type	Which surprise to extract: "surprise" or "signed"
...	Additional arguments (unused)

Value

Numeric vector of surprise values

get_surprise_at_time *Get Surprise at Specific Time*

Description

Extracts surprise values for a specific time period from temporal results.

Usage

```
get_surprise_at_time(temporal_result, time)
```

Arguments

temporal_result	A bs_surprise_temporal object
time	Time value to extract

Value

A bs_surprise object for that time period

kl_divergence *Kullback-Leibler Divergence*

Description

Computes the KL-divergence from prior to posterior distribution, which measures "surprise" in the Bayesian framework.

Usage

```
kl_divergence(posterior, prior, base = 2)
```

Arguments

posterior	Numeric vector of posterior probabilities
prior	Numeric vector of prior probabilities (same length as posterior)
base	Base of logarithm (default: 2 for bits)

Details

KL-divergence is defined as:

$$D_{KL}(P||Q) = \sum_i P_i \log(P_i/Q_i)$$

where P is the posterior and Q is the prior. The divergence is 0 when posterior equals prior (no surprise), and increases as they differ.

Zero probabilities are handled by excluding those terms (convention that 0 * log(0) = 0).

Value

Numeric scalar: the KL-divergence value (always non-negative)

Examples

```
# No surprise when prior equals posterior
kl_divergence(c(0.5, 0.5), c(0.5, 0.5))

# High surprise when distributions differ
kl_divergence(c(0.9, 0.1), c(0.5, 0.5))

# Maximum surprise when posterior is certain
kl_divergence(c(1.0, 0.0), c(0.5, 0.5))
```

---

log_sum_exp *Log-Sum-Exp (Numerically Stable)*

Description

Computes log(sum(exp(x))) in a numerically stable way.

Usage

```
log_sum_exp(x)
```

Arguments

x	Numeric vector of log values

Details

Uses the identity: log(sum(exp(x))) = max(x) + log(sum(exp(x - max(x)))) This avoids overflow when x contains large positive values.

Value

Numeric scalar: log(sum(exp(x)))

Examples

```
# Direct computation would overflow
x <- c(1000, 1001, 1002)
log_sum_exp(x)  # Returns ~1002.41
```

model_names *Get Model Names*

Description

Get Model Names

Usage

```
model_names(model_space)
```

Arguments

model_space	A bs_model_space object

Value

Character vector of model names

model_space *Create a Model Space*

Description

Combines multiple models into a model space with prior probabilities. The model space represents the set of hypotheses about how data is generated.

Usage

```
model_space(..., prior = NULL, names = NULL)
```

Arguments

...	bs_model objects or a list of models
prior	Numeric vector of prior probabilities (must sum to 1). If NULL (default), uses uniform prior.
names	Optional character vector of names for models

Value

A bs_model_space object

Examples

```
# Create model space with uniform prior
space <- model_space(
  bs_model_uniform(),
  bs_model_gaussian()
)

# Create with custom prior
space <- model_space(
  bs_model_uniform(),
  bs_model_baserate(c(0.2, 0.3, 0.5)),
  prior = c(0.3, 0.7)
)

# Create from list
models <- list(bs_model_uniform(), bs_model_gaussian())
space <- model_space(models)
```

---

normalize_minmax *Min-Max Normalization*

Description

Scales values to the range 0 to 1 using min-max normalization.

Usage

```
normalize_minmax(x, min_val = NULL, max_val = NULL, na.rm = TRUE)
```

Arguments

x	Numeric vector
min_val	Minimum value for scaling (default: min of x)
max_val	Maximum value for scaling (default: max of x)
na.rm	Logical; remove NA values when computing range?

Value

Numeric vector scaled to the range 0 to 1

Examples

```
normalize_minmax(c(10, 20, 30, 40, 50))
normalize_minmax(c(-5, 0, 5), min_val = -10, max_val = 10)
```

normalize_prob *Normalize to Probability Distribution*

Description

Normalizes a numeric vector to sum to 1, creating a valid probability distribution.

Usage

```
normalize_prob(x, na.rm = FALSE)
```

Arguments

x	Numeric vector (non-negative values expected)
na.rm	Logical; remove NA values before normalizing?

Details

If all values are zero or if sum is zero, returns uniform distribution. Negative values are set to zero with a warning.

Value

Numeric vector summing to 1 (or containing NAs if input had NAs and na.rm = FALSE)

Examples

```
normalize_prob(c(1, 2, 3, 4))
normalize_prob(c(10, 20, 30))
normalize_prob(c(0, 0, 0))  # Returns uniform
```

normalize_rate *Normalize to Rate (Per Capita)*

Description

Computes rates by dividing counts by a base value (e.g., population).

Usage

```
normalize_rate(count, base, per = 1, na_for_zero = TRUE)
```

Arguments

count	Numeric vector of counts (e.g., number of events)
base	Numeric vector of base values (e.g., population)
per	Multiplier for scaling (e.g., 100000 for "per 100k")
na_for_zero	Logical; return NA when base is zero?

Value

Numeric vector of rates

Examples

```
# Crime rate per 100,000 population
crimes <- c(50, 100, 200)
population <- c(10000, 50000, 100000)
normalize_rate(crimes, population, per = 100000)
```

---

normalize_robust *Robust Normalization (using quantiles)*

Description

Scales values using quantiles instead of min/max for robustness to outliers.

Usage

```
normalize_robust(x, lower_quantile = 0.05, upper_quantile = 0.95, na.rm = TRUE)
```

Arguments

x	Numeric vector
lower_quantile	Lower quantile for scaling (default: 0.05)
upper_quantile	Upper quantile for scaling (default: 0.95)
na.rm	Logical; remove NA values when computing quantiles?

Value

Numeric vector scaled approximately to the range 0 to 1, with outliers potentially outside this range

Examples

```
# With outliers
x <- c(1, 2, 3, 4, 5, 100)  # 100 is an outlier
normalize_robust(x)
```

normalize_zscore *Z-Score Normalization*

Description

Normalizes values to z-scores (standard deviations from mean).

Usage

```
normalize_zscore(x, center = NULL, scale = NULL, na.rm = TRUE)
```

Arguments

x	Numeric vector
center	Value to center around (default: mean of x)
scale	Scale factor (default: standard deviation of x)
na.rm	Logical; remove NA values when computing center/scale?

Value

Numeric vector of z-scores

Examples

```
x <- c(10, 20, 30, 40, 50)
normalize_zscore(x)
```

n_models *Get Number of Models*

Description

Get Number of Models

Usage

```
n_models(model_space)
```

Arguments

model_space	A bs_model_space object

Value

Integer number of models

plot.bs_model_space *Plot Model Space*

Description

Visualizes the prior and posterior probabilities of a model space.

Usage

```
## S3 method for class 'bs_model_space'
plot(
  x,
  y = NULL,
  main = "Model Probabilities",
  col = c("#4575b4", "#d73027"),
  ...
)
```

Arguments

x	A bs_model_space object
y	Unused
main	Plot title
col	Colors for prior and posterior bars
...	Additional arguments passed to barplot()

Value

Invisibly returns the input object

plot.bs_surprise *Plot Surprise Result*

Description

Creates a simple histogram or density plot of surprise values.

Usage

```
## S3 method for class 'bs_surprise'
plot(
  x,
  y = NULL,
  which = c("surprise", "signed_surprise"),
  type = c("histogram", "density"),
  main = NULL,
  xlab = NULL,
  col = "#4575b4",
  border = "white",
  ...
)
```

Arguments

x	A bs_surprise object
y	Unused
which	Which to plot: "surprise" or "signed_surprise"
type	Plot type: "histogram" or "density"
main	Plot title
xlab	X-axis label
col	Fill color
border	Border color
...	Additional arguments passed to hist() or density()

Value

Invisibly returns the input object

---

plot.bs_surprise_sf *Plot Surprise Map (sf)*

Description

Plots a surprise map using sf's plot method. For ggplot2 integration, see geom_surprise() and scale_fill_surprise().

Usage

```
## S3 method for class 'bs_surprise_sf'
plot(
  x,
  y = NULL,
  which = c("surprise", "signed_surprise"),
  pal = NULL,
  breaks = NULL,
  nbreaks = 9,
  main = NULL,
  border = "grey50",
  lwd = 0.2,
  key.pos = 4,
  ...
)
```

Arguments

x	A bs_surprise_sf object
y	Unused (for S3 method compatibility)
which	Which variable to plot: "surprise" or "signed_surprise"
pal	Color palette. If NULL, uses default diverging palette for signed surprise and sequential for unsigned.
breaks	Breaks for color scale. If NULL, uses pretty breaks.
nbreaks	Number of breaks if breaks is NULL
main	Plot title. If NULL, auto-generated.
border	Border color for polygons
lwd	Line width for borders
key.pos	Position of legend (1=below, 2=left, 3=above, 4=right, NULL=none)
...	Additional arguments passed to sf::plot()

Value

Invisibly returns the input object

Examples

```
library(sf)
nc <- st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
result <- surprise(nc, observed = SID74, expected = BIR74)

# Default plot
plot(result)

# Plot signed surprise
plot(result, which = "signed_surprise")
```

```
# Custom palette
plot(result, pal = heat.colors(9))
```

`plot.bs_surprise_temporal` *Plot Temporal Surprise*

Description

Creates time series plots of surprise evolution.

Usage

```
## S3 method for class 'bs_surprise_temporal'
plot(
  x,
  y = NULL,
  type = c("time_series", "heatmap", "cumulative"),
  regions = NULL,
  main = NULL,
  xlab = "Time",
  ylab = NULL,
  col = NULL,
  ...
)
```

Arguments

`x`	A `bs_surprise_temporal` object
`y`	Unused
`type`	Plot type: "time_series", "heatmap", or "cumulative"
`regions`	Which regions to plot (for time_series). If NULL, plots all.
`main`	Plot title
`xlab`	X-axis label
`ylab`	Y-axis label
`col`	Colors
`...`	Additional arguments

Value

Invisibly returns the input object

remove_model *Remove Model from Space*

Description

Removes a model from an existing model space.

Usage

```
remove_model(model_space, which)
```

Arguments

model_space	A bs_model_space object
which	Integer index or character name of model to remove

Value

Updated bs_model_space

scale_fill_surprise *Surprise Color Scale (Sequential)*

Description

Sequential color scale for unsigned surprise values. Uses viridis palettes for perceptually uniform color mapping.

Usage

```
scale_fill_surprise(
  ...,
  option = "inferno",
  direction = 1,
  name = "Surprise",
  begin = 0,
  end = 1
)

scale_colour_surprise(
  ...,
  option = "inferno",
  direction = 1,
  name = "Surprise",
  begin = 0,
```

```
  end = 1
)

scale_color_surprise(
  ...,
  option = "inferno",
  direction = 1,
  name = "Surprise",
  begin = 0,
  end = 1
)
```

Arguments

`...`	Arguments passed to `ggplot2::scale_fill_viridis_c()`
`option`	Viridis palette option: "magma" (A), "inferno" (B), "plasma" (C), "viridis" (D), "cividis" (E), "rocket" (F), "mako" (G), "turbo" (H)
`direction`	Direction of palette (1 = low-to-high, -1 = reversed)
`name`	Legend title
`begin, end`	Range of palette to use (0 to 1)

Value

A ggplot2 scale object

Examples

```
library(ggplot2)
library(sf)

nc <- st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)

# Basic usage - geometry must be mapped explicitly
ggplot(nc) +
  geom_surprise(aes(geometry = geometry, observed = SID74, expected = BIR74)) +
  scale_fill_surprise()
```

---

scale_fill_surprise_binned

Binned Surprise Scale

Description

Discrete binned color scale for surprise values. Useful for creating choropleth maps with clearly distinguishable categories.

Usage

```
scale_fill_surprise_binned(
  n.breaks = 5,
  palette = "YlOrRd",
  direction = 1,
  name = "Surprise",
  ...
)

scale_colour_surprise_binned(
  n.breaks = 5,
  palette = "YlOrRd",
  direction = 1,
  name = "Surprise",
  ...
)

scale_color_surprise_binned(
  n.breaks = 5,
  palette = "YlOrRd",
  direction = 1,
  name = "Surprise",
  ...
)
```

Arguments

n.breaks	Number of breaks/bins
palette	ColorBrewer palette name. For unsigned surprise, sequential palettes like "YlOrRd", "Oranges", "Reds" work well.
direction	Direction of palette (1 = normal, -1 = reversed)
name	Legend title
...	Additional arguments passed to ggplot2::scale_fill_fermenter()

Value

A ggplot2 scale object

Examples

```
library(ggplot2)
library(sf)

nc <- st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)

# Binned surprise scale - geometry must be mapped explicitly
ggplot(nc) +
  geom_surprise(aes(geometry = geometry, observed = SID74, expected = BIR74)) +
  scale_fill_surprise_binned(n.breaks = 5)
```

`scale_fill_surprise_diverging` *Signed Surprise Color Scale (Diverging)*

Description

Diverging color scale for signed surprise values. Negative values (lower than expected) are shown in one color, positive values (higher than expected) in another, with neutral (zero) in the middle.

Usage

```
scale_fill_surprise_diverging(
  low = scales::muted("blue"),
  mid = "white",
  high = scales::muted("red"),
  midpoint = 0,
  name = "Signed Surprise",
  ...
)

scale_colour_surprise_diverging(
  low = scales::muted("blue"),
  mid = "white",
  high = scales::muted("red"),
  midpoint = 0,
  name = "Signed Surprise",
  ...
)

scale_color_surprise_diverging(
  low = scales::muted("blue"),
  mid = "white",
  high = scales::muted("red"),
  midpoint = 0,
  name = "Signed Surprise",
  ...
)
```

Arguments

`low`	Color for negative surprise (lower than expected). Default is blue.
`mid`	Color for zero surprise (as expected). Default is white.
`high`	Color for positive surprise (higher than expected). Default is red.
`midpoint`	Value of the midpoint. Default is 0.
`name`	Legend title
`...`	Additional arguments passed to `ggplot2::scale_fill_gradient2()`

Value

A ggplot2 scale object

Examples

```
library(ggplot2)
library(sf)

nc <- st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)

# Signed surprise with diverging scale - geometry must be mapped explicitly
ggplot(nc) +
  geom_surprise(
    aes(geometry = geometry, observed = SID74, expected = BIR74),
    fill_type = "signed"
  ) +
  scale_fill_surprise_diverging()
```

---

scale_fill_surprise_diverging_binned

Binned Diverging Surprise Scale

Description

Binned diverging scale for signed surprise values.

Usage

```
scale_fill_surprise_diverging_binned(
  n.breaks = 7,
  palette = "RdBu",
  direction = -1,
  name = "Signed Surprise",
  ...
)
```

Arguments

n.breaks	Number of breaks/bins
palette	ColorBrewer diverging palette name. Options: "RdBu", "RdYlBu", "BrBG", "PiYG", "PRGn", "PuOr", "RdGy", "Spectral"
direction	Direction of palette (1 = normal, -1 = reversed)
name	Legend title
...	Additional arguments

Value

A ggplot2 scale object

scale_fill_surprise_manual

Manual Surprise Breaks Scale

Description

Create a scale with manually specified breaks for surprise values.

Usage

```
scale_fill_surprise_manual(
  breaks = c(0.5, 1, 1.5, 2),
  colors = c("#ffffb2", "#fecc5c", "#fd8d3c", "#f03b20", "#bd0026"),
  name = "Surprise",
  labels = waiver(),
  na.value = "grey50",
  ...
)
```

Arguments

breaks	Numeric vector of break points
colors	Character vector of colors (length should be length(breaks) + 1 for binned, or length(breaks) for continuous)
name	Legend title
labels	Labels for legend
na.value	Color for NA values
...	Additional arguments

Value

A ggplot2 scale object

scale_fill_surprise_thresholds

Signed Surprise Scale with Meaningful Thresholds

Description

Diverging color scale for signed surprise with breaks at meaningful thresholds based on the interpretation of surprise values.

Usage

```
scale_fill_surprise_thresholds(
  breaks = c(-1, -0.5, -0.1, 0.1, 0.5, 1),
  palette = "RdBu",
  direction = -1,
  name = "Signed\nSurprise",
  na.value = "grey80",
  ...
)

scale_colour_surprise_thresholds(
  breaks = c(-1, -0.5, -0.1, 0.1, 0.5, 1),
  palette = "RdBu",
  direction = -1,
  name = "Signed\nSurprise",
  na.value = "grey80",
  ...
)

scale_color_surprise_thresholds(
  breaks = c(-1, -0.5, -0.1, 0.1, 0.5, 1),
  palette = "RdBu",
  direction = -1,
  name = "Signed\nSurprise",
  na.value = "grey80",
  ...
)
```

Arguments

`breaks`	Numeric vector of break points. Default uses meaningful thresholds: -1, -0.5, -0.1, 0.1, 0.5, 1 (symmetric around 0)
`palette`	ColorBrewer diverging palette. Default "RdBu".
`direction`	Palette direction. Default -1 (red = positive/higher).
`name`	Legend title
`na.value`	Color for NA values
`...`	Additional arguments passed to scale

Details

Meaningful thresholds for interpreting signed surprise:

- `|surprise|` < `0.1`: Trivial - essentially as expected
- `|surprise|` `0.1-0.5`: Minor to moderate deviation
- `|surprise|` `0.5-1.0`: Substantial - genuinely surprising
- `|surprise|` > `1.0`: High - very surprising

Positive values indicate higher than expected; negative indicate lower.

Value

A ggplot2 scale object

Examples

```
library(ggplot2)
library(sf)

nc <- st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
result <- surprise(nc, observed = SID74, expected = BIR74)
nc$signed_surprise <- get_surprise(result, "signed")

ggplot(nc) +
  geom_sf(aes(fill = signed_surprise)) +
  scale_fill_surprise_thresholds()
```

---

set_prior *Set Prior Probabilities*

Description

Updates the prior probabilities of a model space.

Usage

```
set_prior(model_space, prior)
```

Arguments

model_space	A bs_model_space object
prior	Numeric vector of new prior probabilities (must sum to 1)

Value

Updated bs_model_space

`stat_surprise` *Compute Surprise as ggplot2 Stat*

Description

This stat computes Bayesian surprise for sf geometries, allowing you to visualize surprise directly in ggplot2.

Usage

```
stat_surprise(
  mapping = NULL,
  data = NULL,
  geom = "sf",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  models = c("uniform", "baserate", "funnel"),
  signed = TRUE,
  ...
)
```

Arguments

`mapping`	Aesthetic mapping created by `ggplot2::aes()`. Required aesthetics are `geometry` (from sf) and `observed`. Optional aesthetics include `expected` and `sample_size`.
`data`	Data (typically an sf object)
`geom`	Geometry to use (default: "sf")
`position`	Position adjustment
`na.rm`	Remove NA values?
`show.legend`	Show legend?
`inherit.aes`	Inherit aesthetics from ggplot?
`models`	Character vector of model types to use. Options: "uniform", "baserate", "gaussian", "sampled", "funnel"
`signed`	Logical; compute signed surprise?
`...`	Additional arguments passed to the layer

Value

A ggplot2 layer

Computed Variables

surprise Bayesian surprise (KL-divergence)

signed_surprise Signed surprise (if signed = TRUE)

Examples

```
library(ggplot2)
library(sf)

nc <- st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)

# Basic surprise map - geometry must be mapped explicitly
ggplot(nc) +
  stat_surprise(aes(geometry = geometry, observed = SID74, expected = BIR74)) +
  scale_fill_surprise()
```

`stat_surprise_sf` *Stat for Surprise with sf Geometries*

Description

Stat for Surprise with sf Geometries

Usage

```
stat_surprise_sf(
  mapping = NULL,
  data = NULL,
  geom = ggplot2::GeomSf,
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  models = c("uniform", "baserate", "funnel"),
  signed = TRUE,
  ...
)
```

Arguments

`mapping`	Aesthetic mapping created by `ggplot2::aes()`. Required aesthetics are `geometry` (from sf) and `observed`. Optional aesthetics include `expected` and `sample_size`.
`data`	Data (typically an sf object)
`geom`	Geometry to use (default: "sf")
`position`	Position adjustment
`na.rm`	Remove NA values?
`show.legend`	Show legend?
`inherit.aes`	Inherit aesthetics from ggplot?
`models`	Character vector of model types to use. Options: "uniform", "baserate", "gaussian", "sampled", "funnel"
`signed`	Logical; compute signed surprise?
`...`	Additional arguments passed to the layer

Value

A list containing a ggplot2 layer using StatSurpriseSf and `ggplot2::coord_sf()`. The stat computes `surprise` and, when `signed = TRUE`, `signed_surprise`, which are available to downstream geoms via `after_stat()`.

`st_density` *Spatial Density Estimation for sf Objects*

Description

Computes kernel density estimates for point or polygon sf objects.

Usage

```
st_density(
  x,
  method = c("kde", "kriging"),
  bandwidth = NULL,
  n = 100,
  weights = NULL,
  ...
)
```

Arguments

`x`	An sf object with point or polygon geometries
`method`	Density estimation method: "kde" (kernel density) or "kriging" (requires additional packages)
`bandwidth`	Bandwidth for KDE. If NULL, estimated automatically.
`n`	Grid size for density estimation
`weights`	Optional weights for each feature
`...`	Additional arguments passed to density estimation functions

Value

A list with density estimates and grid information

Examples

```
library(sf)
# Create random points
pts <- st_as_sf(data.frame(
  x = rnorm(100),
  y = rnorm(100)
), coords = c("x", "y"))

# Compute density
dens <- st_density(pts)
```

st_density_at *Evaluate Density at sf Feature Locations*

Description

Evaluates a density estimate at the locations of features in an sf object.

Usage

```
st_density_at(density, x)
```

Arguments

density	Density estimate from st_density()
x	sf object with features to evaluate at

Value

Numeric vector of density values

st_surprise *Compute Surprise for sf Object*

Description

Convenience wrapper for computing surprise on sf objects.

Usage

```
st_surprise(x, observed, expected = NULL, ...)
```

Arguments

x	An sf object
observed	Column name (unquoted or string) or numeric vector of observed values
expected	Column name or vector of expected values (for base rate model). If NULL and models include base rate, computed from observed.
...	Additional arguments passed to model likelihood functions

Value

A bs_surprise_sf object

Examples

```
library(sf)
nc <- st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
result <- st_surprise(nc, observed = SID74, expected = BIR74)
plot(result)
```

---

surprise.sf *Compute Bayesian Surprise*

Description

Main function to compute Bayesian surprise for spatial or tabular data. This measures how much each observation updates beliefs about a set of models, highlighting unexpected patterns while debiasing against known factors.

Usage

```
## S3 method for class 'sf'
surprise(
  data,
  observed,
  expected = NULL,
  sample_size = NULL,
  models = c("uniform", "baserate", "funnel"),
  prior = NULL,
  signed = TRUE,
  ...
)

surprise(
  data,
  observed,
  expected = NULL,
  sample_size = NULL,
  models = c("uniform", "baserate", "funnel"),
  prior = NULL,
  signed = TRUE,
  normalize_posterior = TRUE,
  ...
)

## S3 method for class 'data.frame'
surprise(
  data,
  observed,
  expected = NULL,
  sample_size = NULL,
```

```
  models = c("uniform", "baserate", "funnel"),
  prior = NULL,
  signed = TRUE,
  normalize_posterior = TRUE,
  ...
)

## S3 method for class 'tbl_df'
surprise(data, ...)
```

Arguments

`data`	Data frame, tibble, or sf object
`observed`	Column name (unquoted or string) or numeric vector of observed values
`expected`	Column name or vector of expected values (for base rate model). If NULL and models include base rate, computed from observed.
`sample_size`	Column name or vector of sample sizes (for funnel model). Defaults to `expected` if not provided.
`models`	Model specification. Can be: • A `bs_model_space` object • A character vector of model types: "uniform", "baserate", "gaussian", "sampled", "funnel" • A list of `bs_model` objects
`prior`	Numeric vector of prior probabilities for models. Only used when `models` is a character vector or list.
`signed`	Logical; compute signed surprise?
`...`	Additional arguments passed to model likelihood functions
`normalize_posterior`	Logical; if TRUE (default), normalizes posteriors before computing KL divergence. This is the standard Bayesian Surprise calculation. If FALSE, uses the unnormalized per-region posterior weights used by the original Correll & Heer JavaScript demo; this option is retained only for legacy comparison.

Value

For data frames: the input with `surprise` (and optionally `signed_surprise`) columns added, plus a `surprise_result` attribute. For sf objects: a `bs_surprise_sf` object.

Examples

```
# Using sf package's NC data
library(sf)
nc <- st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)

# Basic usage with default models
result <- surprise(nc, observed = SID74, expected = BIR74)
```

```
# With specific model types
result <- surprise(nc,
  observed = "SID74",
  expected = "BIR74",
  models = c("uniform", "baserate", "funnel")
)

# With custom model space
space <- model_space(
  bs_model_uniform(),
  bs_model_baserate(nc$BIR74)
)
result <- surprise(nc, observed = SID74, models = space)

# View results
plot(result, which = "signed_surprise")
```

`surprise_animate` *Create Animation-Ready Data from Temporal Results*

Description

Converts temporal surprise results into a format suitable for animation (e.g., with gganimate).

Usage

```
surprise_animate(temporal_result, include_posterior = FALSE)
```

Arguments

`temporal_result`
: A `bs_surprise_temporal` object

`include_posterior`
: Include posterior probabilities in output?

Value

A data frame with columns: time, region (if applicable), surprise, signed_surprise, and optionally model posteriors

surprise_rolling *Rolling Window Surprise*

Description

Computes surprise using a rolling window of observations.

Usage

```
surprise_rolling(
  observed,
  expected = NULL,
  window_size = 10,
  step = 1,
  models = c("uniform", "baserate", "funnel"),
  ...
)
```

Arguments

observed	Numeric vector of observed values (time-ordered)
expected	Numeric vector of expected values
window_size	Number of observations in the window
step	Step size for moving the window
models	Model specification
...	Additional arguments passed to compute_surprise()

Value

A list with surprise values for each window position

surprise_temporal *Compute Temporal Surprise*

Description

Computes surprise over time, allowing beliefs to update as new data arrives. This is useful for detecting changes in patterns over time and for streaming data applications.

Usage

```
surprise_temporal(
  data,
  time_col,
  observed,
  expected = NULL,
  region_col = NULL,
  models = c("uniform", "baserate", "funnel"),
  update_prior = TRUE,
  window_size = NULL,
  signed = TRUE,
  ...
)
```

Arguments

data	Data frame with time-indexed observations
time_col	Column name for time variable (unquoted or string)
observed	Column name for observed values
expected	Column name for expected values (optional)
region_col	Column name for region/spatial identifier (optional). If provided, computes per-region surprise at each time point.
models	Model specification (see surprise())
update_prior	Logical; should prior be updated after each time step?
window_size	For rolling window analysis: number of time periods to include. If NULL, uses all prior data.
signed	Compute signed surprise?
...	Additional arguments passed to compute_surprise()

Value

A bs_surprise_temporal object containing:

- surprise_by_time: List of surprise results for each time period
- time_values: Vector of time values
- cumulative_surprise: Matrix of cumulative surprise
- model_space: The model space used

Examples

```
# Create temporal data
df <- data.frame(
  year = rep(2010:2020, each = 5),
  region = rep(letters[1:5], 11),
  events = rpois(55, lambda = 50),
  population = rep(c(10000, 50000, 100000, 25000, 15000), 11)
```

```
)

# Compute temporal surprise
result <- surprise_temporal(df,
  time_col = year,
  observed = events,
  expected = population,
  region_col = region
)

# View results
print(result)
```

`update_surprise` *Update Surprise with New Data (Streaming)*

Description

Updates an existing surprise result with new observations. Useful for real-time or streaming data applications.

Usage

```
update_surprise(
  surprise_result,
  new_observed,
  new_expected = NULL,
  time_value = NULL,
  update_prior = TRUE
)
```

Arguments

`surprise_result`	An existing `bs_surprise` or `bs_surprise_temporal` object
`new_observed`	New observed values
`new_expected`	New expected values (optional)
`time_value`	Time value for the new observation (for temporal)
`update_prior`	Update prior with current posterior?

Value

Updated surprise result

Examples

```
# Initial computation
observed <- c(50, 100, 150)
expected <- c(10000, 50000, 100000)
result <- auto_surprise(observed, expected)

# Update with new data
new_obs <- c(200, 75)
new_exp <- c(80000, 20000)
result <- update_surprise(result, new_obs, new_exp)
```

Index