Package 'docketscape'

November 13, 2026

Type Package

Title Legal Network and Congestion-Aware Delay Modeling for Court Dockets

Version 0.1.0

Description Splits court delay into two parts: (1) what a case is, represented through the statutory Acts and Sections invoked by the case, and (2) where it sits, represented by time-varying court docket load. Builds a weighted similarity graph of cases from legal provisions using 'igraph' and 'Matrix', estimates censoring-aware expected resolution timelines using legally similar neighbours, calculates relative lateness, and models delay in relation to filing-time court docket load using 'survival'. Designed for case-level judicial records such as the Development Data Lab (DDL) extract of Indian lower courts (<https://www.devdatalab.org/judicial-data>). Observational and non-causal; does not rank judges, evaluate judicial performance, or claim causal delay effects.

License MIT + file LICENSE

URL https://github.com/shraddhaajit/docketscape, https://www.devdatalab.org/judicial-data

BugReports https://github.com/shraddhaajit/docketscape/issues

Encoding UTF-8

Language en-GB

Depends R (>= 4.1.0)

Imports igraph, Matrix, stats, survival

Suggests knitr, rmarkdown, spelling, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Shraddha Ajit [aut, cre], DK Janani [aut]

Maintainer Shraddha Ajit <shra19076@gmail.com>

Repository CRAN

Date/Publication 2026-11-13 16:40:15 UTC

Contents

build_case_graph 2
build_section_graph 4
check_court_overlap 5
check_future_neighbours 6
check_near_duplicates 7
clean_cases 8
court_load 10
decompose_delay 11
diffuse_profiles 12
fit_delay_model 14
neighbourhood_survival 16
print.court_load 17
print.court_overlap_audit 18
print.delay_decomposition 18
print.delay_model 19
print.diffused_profiles 19
print.future_neighbour_audit 20
print.model_validation 20
print.near_duplicate_audit 21
print.neighbourhood_survival 21
print.relative_lateness 22
project_validation_legal_score 22
relative_lateness 24
validate_models 25

Index **28**

`build_case_graph` *Construct a Weighted Case Similarity Graph*

Description

Builds an undirected `igraph` graph of cases where nodes represent cases, edges connect legally similar cases based on their diffused section profiles, and edge weights represent cosine legal similarity values in `[0, 1]` after optional inverse-document-frequency (IDF) section rarity weighting.

Usage

```
build_case_graph(profiles, k = 3, use_idf = TRUE)
```

Arguments

profiles	An object of class diffuse_profiles containing diffused case profiles.
k	Positive integer specifying the maximum number of nearest neighbours each case retains. Defaults to 3.
use_idf	Logical flag indicating whether to weight sections by inverse document frequency (rarity) prior to cosine similarity. Defaults to TRUE.

Details

Graph Construction Pipeline:

1. **IDF Rarity Weighting:** For section s present in df_s cases out of N total cases:

$$w_s = \log\left(\frac{N}{df_s}\right) + 1$$

   Sections appearing in fewer cases receive higher weights.

2. **Cosine Similarity:** For cases i and j with weighted profiles w_i and w_j:

$$\text{sim}(i,j) = \frac{w_i \cdot w_j}{\|w_i\|_2 \|w_j\|_2}$$

3. **k-NN Pruning & Deterministic Symmetrization:** Each case selects up to k neighbours with highest non-zero similarity (ties broken deterministically by case ID). An undirected edge is created if either case selects the other.

Value

An undirected igraph graph object containing:

Vertex attributes name / case_id matching the input case IDs.

Edge attributes weight numeric cosine legal similarity values in [0, 1].

Examples

```
syn_sec <- data.frame(
  case_id = c(1, 1, 2, 2, 3),
  act = c("NI Act", "NI Act", "NI Act", "NI Act", "IPC"),
  section = c("138", "141", "138", "142", "302")
)
sec_g <- build_section_graph(syn_sec)
diff_p <- diffuse_profiles(syn_sec, sec_g)
case_g <- build_case_graph(diff_p, k = 2)
print(case_g)
```

`build_section_graph` *Construct a Legal Section Co-occurrence Graph*

Description

Builds a weighted, undirected `igraph` graph where vertices represent unique legal section identifiers (combination of Act and Section) and edge weights represent the count of distinct cases in which two legal sections co-occur.

Usage

```
build_section_graph(
  sections,
  case = "case_id",
  act = "act",
  section = "section"
)
```

Arguments

`sections`	A `data.frame` containing legal section records for cases.
`case`	Character string specifying the column name for case IDs. Defaults to `"case_id"`.
`act`	Character string specifying the column name for Act names. Defaults to `"act"`.
`section`	Character string specifying the column name for Section numbers/names. Defaults to `"section"`.

Details

Graph Construction Details:

- **Identifiers:** Legal sections are identified by combining Act and Section (e.g., `"NI Act::138"`). This ensures sections from different Acts with identical section numbers remain distinct vertices.
- **Deduplication:** Duplicate occurrences of the same `(case_id, act, section)` triplet are removed prior to counting to prevent artificial inflation of edge weights.
- **Singletons:** Sections that appear without co-occurring with any other section in the same case are retained as isolated vertices with degree 0.
- **Missing Data:** Rows containing `NA`, empty strings, or blank values in `case_id`, `act`, or `section` are excluded from graph construction.

Value

An undirected `igraph` graph object containing:

Vertex attributes `name` / `identifier` (unique `"Act::Section"`), `act`, `section`.

Edge attributes `weight` (integer count of distinct cases with co-occurrence).

Examples

```
sections <- data.frame(
  case_id = c(1, 1, 2, 2, 3),
  act = c("NI Act", "NI Act", "NI Act", "NI Act", "IPC"),
  section = c("138", "141", "138", "142", "302")
)
g <- build_section_graph(sections)
print(g)
```

`check_court_overlap` *Audit Court Representation Overlap Across Temporal Split*

Description

Evaluates court representation overlap between training and validation case sets.

Usage

```
check_court_overlap(train_cases, validation_cases, court = "court")
```

Arguments

`train_cases`	A `data.frame` or vector of court identifiers for the training set.
`validation_cases`	A `data.frame` or vector of court identifiers for the validation set.
`court`	Character string specifying the court column name if data frames are provided. Defaults to `"court"`.

Details

Court overlap describes whether validation tests generalisation across time within known courts or tests generalisation to entirely unseen courts. Shared courts are a validation design feature, not data leakage.

Value

An S3 object of class `c("court_overlap_audit", "list")` containing:

training_courts Sorted vector of unique training courts.

validation_courts Sorted vector of unique validation courts.

overlapping_courts Sorted vector of shared courts.

n_overlapping_courts Count of shared courts.

training_only_courts Vector of courts present only in training.

validation_only_courts Vector of courts present only in validation.

overlap_ratio Fraction of validation courts present in training.

interpretation Statistical interpretation of court overlap.

Examples

```
train_df <- data.frame(court = c("C1", "C2"))
val_df <- data.frame(court = c("C1", "C3"))
audit <- check_court_overlap(train_df, val_df)
print(audit)
```

`check_future_neighbours` *Audit Future-Neighbour Leakage in Case Similarity Graph*

Description

Identifies graph edges where a case is connected in `case_graph` to a case filed later than a specified temporal cutoff date (or later than its own filing date).

Usage

```
check_future_neighbours(case_graph, cases_clean, cutoff_date = NULL)
```

Arguments

`case_graph`	An `igraph` graph object produced by `build_case_graph`.
`cases_clean`	A `data.frame` of cleaned cases containing `case_id` and `filing_date`.
`cutoff_date`	Optional `Date` threshold separating training cases from validation cases.

Details

Future-Neighbour Directionality: For an undirected edge between Case A (filed 2012) and Case B (filed 2013), Case A has a future neighbour (Case B), whereas Case B has a past neighbour (Case A). If `cutoff_date` is provided, only edges where a pre-cutoff case is connected to a post-cutoff case are flagged.

Statistical Interpretation: Future-neighbour leakage indicates that a case's legal similarity profile was constructed using cases filed after its temporal cutoff. This is an audit of information leakage, not proof of an invalid model.

Value

An S3 object of class `c("future_neighbour_audit", "list")` containing:

available Logical indicating if the audit was evaluated successfully.

cutoff_date `Date` threshold used, or `NULL` if auditing full graph temporal order.

n_edges_checked Total number of edges in `case_graph`.

n_future_neighbour_edges Count of future-neighbour leakage edges.

n_cases_with_future_neighbours Count of distinct cases with at least one future neighbour.

affected_case_ids Character vector of affected case IDs.

future_neighbour_edges A `data.frame` of individual leakage edges.

max_future_neighbour_filing_date Latest filing date among identified future neighbours.

interpretation Statistical interpretation of audit findings.

Examples

```
cases <- data.frame(
  case_id = c("1", "2"),
  filing_date = as.Date(c("2012-01-10", "2013-05-15"))
)
g <- igraph::make_graph(c("1", "2"), directed = FALSE)
audit <- check_future_neighbours(g, cases, cutoff_date = as.Date("2012-12-31"))
print(audit)
```

---

check_near_duplicates *Audit Same-Date/Court Clusters and Exact Duplicate Records*

Description

Identifies same-date/court case clusters and exact duplicate rows in cleaned case data based on court, filing date, and decision date.

Usage

```
check_near_duplicates(cases_clean, court = "court")
```

Arguments

`cases_clean`	A `data.frame` of cleaned case records.
`court`	Character string specifying the court column name in `cases_clean`. Defaults to `"court"`.

Details

Audit checks for multiple case records sharing identical court, filing date, and decision date. Cases with distinct `case_id`s sharing dates/courts form same-date/court clusters, which are explicitly distinguished from duplicate case records. Fuzzy near-duplicate matching on unstructured text is not performed as case-level text is not present.

Value

An S3 object of class `c("near_duplicate_audit", "list")` containing:

available Logical indicating if audit evaluated successfully.

n_rows_checked Total number of case rows audited.

n_same_date_court_groups Number of distinct same-date/court clusters.

n_rows_in_duplicate_groups Total number of case rows belonging to clusters.

duplicate_groups A data frame detailing clusters and case IDs.

interpretation Statistical interpretation of audit findings.

Examples

```
cases <- data.frame(
  case_id = 1:3,
  court = c("C1", "C1", "C2"),
  filing_date = as.Date(c("2012-01-10", "2012-01-10", "2012-03-05")),
  decision_date = as.Date(c("2012-11-02", "2012-11-02", "2013-02-14"))
)
audit <- check_near_duplicates(cases)
print(audit)
```

---

clean_cases *Standardise and Validate Case Records for Delay and Survival Analysis*

Description

Takes raw case-level data containing filing dates and optional decision dates, validates dates, flags problematic records without deletion, and calculates right-censored survival duration and event indicators relative to an observation end date.

Usage

```
clean_cases(
  cases,
  filing = "filing_date",
  decision = "decision_date",
  observation_end
)
```

Arguments

cases	A data.frame containing case-level records.
filing	Character string specifying the name of the column containing filing dates, or a vector of filing dates matching nrow(cases). Defaults to "filing_date".
decision	Character string specifying the name of the column containing decision dates, or a vector of decision dates matching nrow(cases). Defaults to "decision_date".
observation_end	A single Date object or character string coercible to Date (e.g. "2014-12-31") representing the cutoff date of observation.

Details

Censoring Logic:

- Disposed case: decision_date <= observation_end. duration = decision_date - filing_date, event = 1.
- Pending case: is.na(decision_date). duration = observation_end - filing_date, event = 0.
- Invalid case: Cases with missing filing dates, decision dates before filing dates, or filing dates after observation_end are flagged with is_valid = FALSE and an explicit invalid_reason. They are NOT deleted. Durations for invalid records are set to NA to ensure no negative durations exist in downstream statistical models.

Value

A modified data.frame based on cases with additional or updated columns:

filing_date Standardised Date object for case filing.

decision_date Standardised Date object for case decision (NA if pending).

duration Numeric duration in days. For disposed cases, decision_date - filing_date. For pending cases, observation_end - filing_date. Set to NA_real_ for invalid records.

event Numeric event status indicator. 1 for disposed, 0 for pending, NA_real_ for invalid records.

is_valid Logical flag indicating whether the record is valid (TRUE) or problematic (FALSE).

invalid_reason Character string describing the issue if is_valid == FALSE, otherwise NA_character_.

Examples

```
cases <- data.frame(
  case_id = 1:3,
  filing_date = as.Date(c("2012-01-10", "2012-03-05", "2012-06-20")),
  decision_date = as.Date(c("2012-11-02", "2013-02-14", NA))
)
clean_cases(cases, observation_end = "2014-12-31")
```

court_load *Calculate Time-Varying Court Docket Load*

Description

Computes the number of open cases at each court across specified time points (monthly or daily). A case contributes to court load at time t if it was filed on or before t and has not yet been disposed by t.

Usage

```
court_load(cases_clean, court = "court", by = "month")
```

Arguments

cases_clean	A data.frame of cleaned case records (produced by clean_cases).
court	Character string specifying the column name for court identifiers. Defaults to "court".
by	Character string specifying the time aggregation interval: "month" (default) or "day".

Details

Open-Case Boundary Convention: A case is defined as open at court c at time t if:

$$\text{filing_date} \leq t \quad \text{AND} \quad (\text{event} == 0 \quad \text{OR} \quad \text{decision_date} > t)$$

- **Disposed Cases ($event = 1$):** Contribute to load starting on their filing date up to, but excluding, their decision date. A case disposed on date t is considered no longer open at time point t.
- **Pending Cases ($event = 0$):** Remain open at all time points on or after their filing date through the end of the observation window.
- **Time Points:** For by = "month", time points are set to the 1st of each calendar month (e.g. 2020-01-01, 2020-02-01).

Statistical Interpretation: court_load() measures docket exposure / open case volume over time. It does NOT measure judge efficiency, judicial quality, or causal congestion effects.

Value

An S3 object of class c("court_load", "data.frame") containing:

court Character column containing court identifiers.

date Date object representing the evaluation time point (1st of each month for by = "month").

load Integer count of open cases at that court and date.

Examples

```
cases <- data.frame(
  case_id = 1:3,
  court = c("C1", "C1", "C2"),
  filing_date = as.Date(c("2012-01-10", "2012-03-05", "2012-06-20")),
  decision_date = as.Date(c("2012-11-02", "2013-02-14", NA))
)
cleaned <- clean_cases(cases, observation_end = "2014-12-31")
load_df <- court_load(cleaned, by = "month")
head(load_df)
```

---

decompose_delay *Decompose Delay Model Linear Predictors*

Description

Decomposes the log-hazard linear predictor of a fitted `delay_model` into legal-similarity and filing-time docket-load association components.

Usage

```
decompose_delay(fit)
```

Arguments

`fit`	A `delay_model` object produced by `fit_delay_model`.

Details

Linear Predictor Decomposition: For each included case i, the fitted Cox model log-hazard linear predictor η_i is decomposed as:

$$\eta_i = \beta_{\text{legal}} \cdot \text{legal_score}_i + \beta_{\text{load}} \cdot \text{load_at_filing}_i$$

- **Legal Component:** $\text{legal_component}_i = \beta_{\text{legal}} \cdot \text{legal_score}_i$
- **Load Component:** $\text{load_component}_i = \beta_{\text{load}} \cdot \text{load_at_filing}_i$
- **Total Linear Predictor:** $\text{total_linear_predictor}_i = \text{legal_component}_i + \text{load_component}_i$

Absolute Shares: Relative contribution shares are calculated on absolute component magnitudes to prevent denominator distortion or sign cancellation:

$$\text{absolute_legal_share}_i = \frac{|\text{legal_component}_i|}{|\text{legal_component}_i| + |\text{load_component}_i|}$$

$$\text{absolute_load_share}_i = \frac{|\text{load_component}_i|}{|\text{legal_component}_i| + |\text{load_component}_i|}$$

If both components are exactly zero, both absolute shares return `NA`.

Statistical Safeguard: The decomposition partitions the fitted Cox model's linear predictor into observational associations with legal similarity and filing-time docket load. It does NOT estimate causal contributions to delay, judge effects, or court performance/efficiency scores.

Value

An S3 object of class `c("delay_decomposition", "list")` containing:

case_decomposition A `data.frame` of case-level predictor components and absolute shares.

summary A `data.frame` of aggregate summary metrics across all decomposed cases.

coefficients A named vector of the fitted Cox model coefficients (β_{legal}, β_{load}).

model The underlying fitted coxph model object.

interpretation Explicit non-causal statistical safeguard statement.

available Logical indicating whether decomposition was successfully computed.

message Status message describing decomposition availability.

Examples

```
cases <- data.frame(
  case_id = paste0("C", 1:10),
  court = "C1",
  filing_date = as.Date(rep(c("2012-01-10", "2012-02-10"), each = 5)),
  duration = c(50, 80, 110, 140, 70, 90, 130, 160, 120, 100),
  event = c(1, 1, 1, 0, 1, 1, 0, 1, 1, 1),
  stringsAsFactors = FALSE
)
cl <- data.frame(
  court = "C1",
  date = as.Date(c("2012-01-01", "2012-02-01")),
  load = c(10, 20),
  stringsAsFactors = FALSE
)
edges <- data.frame(
  from = paste0("C", 1:9),
  to = paste0("C", 2:10),
  weight = c(0.8, 0.7, 0.6, 0.5, 0.7, 0.8, 0.6, 0.5, 0.9),
  stringsAsFactors = FALSE
)
g <- igraph::graph_from_data_frame(edges, directed = FALSE)
fit <- fit_delay_model(cases, g, cl)
decomp <- decompose_delay(fit)
print(decomp)
```

---

diffuse_profiles *Diffuse Case Legal Profiles Over Section Co-occurrence Graph*

Description

Performs a one-step personalized diffusion of case-level legal section profiles across a weighted section co-occurrence graph. This allows legally related cases to share similarity even when they do not invoke identical sections.

Usage

```
diffuse_profiles(
  sections,
  section_graph,
  case = "case_id",
  act = "act",
  section = "section",
  alpha = 0.7
)
```

Arguments

sections	A data.frame containing case section records (must include case ID, Act, and Section columns).
section_graph	An igraph graph object representing legal section co-occurrence (e.g. created by build_section_graph).
case	Character string specifying the column name for case IDs. Defaults to "case_id".
act	Character string specifying the column name for Act names. Defaults to "act".
section	Character string specifying the column name for Section numbers/names. Defaults to "section".
alpha	Numeric value in [0, 1] controlling the weight retained by the original observed section profile. Defaults to 0.7.

Details

Mathematical Formulation: Let A be the weighted adjacency matrix of the section graph, and D be the diagonal degree matrix. The transition matrix is defined as $P = D^{-1}A$ (with rows of isolated nodes set to 0).

For a case with normalized initial section indicator vector x_{init} (where $\|x_{init}\|_1 = 1$):

$$x_{raw} = \alpha \cdot x_{init} + (1 - \alpha) \cdot (x_{init} P)$$

The final profile is normalized to unit L_1 norm:

$$x_{final} = \frac{x_{raw}}{\|x_{raw}\|_1}$$

If $\alpha = 1$, the original section profile is returned without diffusion. If $\alpha = 0$, the profile is derived entirely from neighboring sections in the co-occurrence graph.

Value

An S3 object of class "diffused_profiles" containing:

profiles A sparse matrix of class dgCMatrix (from Matrix) of dimension N_cases x M_sections, with row sums equal to 1.

case_ids Character vector of unique case IDs corresponding to matrix rows.

section_ids Character vector of section identifiers ("Act::Section") corresponding to matrix columns.

alpha The numeric alpha value used for diffusion.

Examples

```
syn_sec <- data.frame(
  case_id = c(1, 1, 2, 2, 3),
  act = c("NI Act", "NI Act", "NI Act", "NI Act", "IPC"),
  section = c("138", "141", "138", "142", "302")
)
g <- build_section_graph(syn_sec)
diff_res <- diffuse_profiles(syn_sec, g, alpha = 0.7)
print(diff_res)
```

`fit_delay_model` *Fit Censoring-Aware Delay Model*

Description

Fits a Cox proportional hazards model relating case resolution speed to the case's legal similarity profile score and its court's docket load at filing time.

Usage

```
fit_delay_model(cases_clean, case_graph, court_load_df, court = "court")
```

Arguments

`cases_clean`	A `data.frame` of cleaned case records (produced by `clean_cases`).
`case_graph`	An `igraph` graph of case similarities (produced by `build_case_graph`).
`court_load_df`	A `court_load` data frame (produced by `court_load`) containing columns `court`, `date`, and `load`.
`court`	Character string specifying the column name for court identifiers in `cases_clean`. Defaults to `"court"`.

Details

Legal Score Definition: For case i, `legal_score` is calculated as the mean weight of all incident edges in `case_graph`:

$$\text{legal_score}_i = \frac{1}{|N(i)|} \sum_{j \in N(i)} w_{ij}$$

If a case has no legal graph neighbours, `legal_score` is assigned `NA` (not 0).

Filing-Time Court Load Alignment: Each case's filing date is mapped to the first of its calendar month (`format(filing_date, "%Y-%m-01")`), and matched against the court's docket load from `court_load_df`. `load_at_filing` represents baseline docket exposure at filing, not time-varying cumulative load across case lifetime.

Statistical Interpretation: Model estimates quantify observational associations. A hazard ratio HR > 1 for `load_at_filing` indicates that higher baseline docket load is associated with a higher

hazard of resolution (faster resolution), conditional on legal score. Model coefficients are non-causal associations and must NOT be interpreted as causal delay impacts or judge performance measures.

Value

An S3 object of class `c("delay_model", "list")` containing:

model The fitted `coxph` model object (or `NULL` if fitting failed).

model_data The subset `data.frame` used for model fitting.

coefficients A data frame of estimated coefficients, standard errors, z-scores, and p-values.

hazard_ratios A data frame of hazard ratios ($\exp(\beta)$) and 95% confidence intervals.

n_obs Total number of valid observations included in the model.

n_events Number of resolution events (`event == 1`) in the model data.

n_excluded Total number of cases excluded due to missing data or unmapped predictors.

exclusions Named integer vector showing breakdown of exclusion reasons.

legal_score_def Character string describing the legal score calculation method.

load_def Character string describing filing-time court docket load alignment.

formula The `formula` used in fitting.

ph_test Output of `cox.zph` diagnostic test if feasible, otherwise `NULL`.

fit_success Logical indicating whether the Cox regression model fitted successfully.

fit_message Character description of model status or error message if fitting failed.

Examples

```
cases <- data.frame(
  case_id = paste0("C", 1:10),
  court = "C1",
  filing_date = as.Date(rep(c("2012-01-10", "2012-02-10"), each = 5)),
  duration = c(50, 80, 110, 140, 70, 90, 130, 160, 120, 100),
  event = c(1, 1, 1, 0, 1, 1, 0, 1, 1, 1),
  stringsAsFactors = FALSE
)
cl <- data.frame(
  court = "C1",
  date = as.Date(c("2012-01-01", "2012-02-01")),
  load = c(10, 20),
  stringsAsFactors = FALSE
)
edges <- data.frame(
  from = paste0("C", 1:9),
  to = paste0("C", 2:10),
  weight = c(0.8, 0.7, 0.6, 0.5, 0.7, 0.8, 0.6, 0.5, 0.9),
  stringsAsFactors = FALSE
)
g <- igraph::graph_from_data_frame(edges, directed = FALSE)
fit <- fit_delay_model(cases, g, cl)
print(fit)
```

`neighbourhood_survival` *Censoring-Aware Neighbourhood Survival Curve Estimator*

Description

Estimates a case-specific weighted Kaplan-Meier survival curve using the legally similar neighbouring cases of a target case in the case similarity graph.

Usage

```
neighbourhood_survival(case_graph, cases_clean, case_id)
```

Arguments

`case_graph`	An `igraph` graph of cases (e.g. produced by `build_case_graph`).
`cases_clean`	A `data.frame` of cleaned cases (e.g. produced by `clean_cases`).
`case_id`	Character string or numeric value specifying the target case ID.

Details

Weighted Kaplan-Meier Estimator: For a target case i, let $j \in \text{Neighbours}(i)$ be its legally similar neighbours in `case_graph` with similarity weights w_j. Target case i is explicitly excluded from its own neighbourhood to prevent outcome leakage.

At each distinct disposal event time $t_{(l)}$:

$$Y(t_{(l)}) = \sum_{j:T_j \geq t_{(l)}} w_j \quad \text{(Weighted Risk Set)}$$

$$d(t_{(l)}) = \sum_{j:T_j = t_{(l)}, \delta_j = 1} w_j \quad \text{(Weighted Disposals)}$$

$$S(t) = \prod_{t_{(l)} \leq t} \left(1 - \frac{d(t_{(l)})}{Y(t_{(l)})}\right)$$

Right-censored neighbours ($\delta_j = 0$) remain in the risk set $Y(t)$ until their censoring duration but do not contribute to event counts $d(t)$.

Value

An S3 object of class `"neighbourhood_survival"` containing:

target_case_id Character string of the target case ID.

target_status Character status (`"pending"`, `"disposed"`, or `"invalid"`).

target_duration Observed age/duration of the target case.

target_event Event indicator of the target case.

neighbours `data.frame` containing neighbour case IDs, weights, durations, and event indicators.

n_neighbours Integer count of valid neighbours used.

survival_table `data.frame` with columns `time`, `n_at_risk`, `n_events`, `surv`.

median_survival Estimated median resolution time (days), or `NA_real_` if not reached.

Examples

```
cases <- data.frame(
  case_id = 1:3,
  filing_date = as.Date(c("2012-01-10", "2012-03-05", "2012-06-20")),
  decision_date = as.Date(c("2012-11-02", "2013-02-14", NA))
)
cleaned <- clean_cases(cases, observation_end = "2014-12-31")
sec <- data.frame(
  case_id = c(1, 1, 2, 2, 3),
  act = c("NI Act", "NI Act", "NI Act", "NI Act", "IPC"),
  section = c("138", "141", "138", "142", "302")
)
sg <- build_section_graph(sec)
dp <- diffuse_profiles(sec, sg)
cg <- build_case_graph(dp, k = 2)
km_res <- neighbourhood_survival(cg, cleaned, case_id = 1)
print(km_res)
```

`print.court_load` *Print method for court_load*

Description

Print method for court_load

Usage

```
## S3 method for class 'court_load'
print(x, ...)
```

Arguments

`x`	A `court_load` object.
`...`	Additional arguments (unused).

Value

Invisibly returns the input `court_load` object of class `c("court_load", "data.frame")`. Called primarily for its side effect of printing a formatted docket load summary to the console.

`print.court_overlap_audit` *Print method for court_overlap_audit*

Description

Print method for court_overlap_audit

Usage

```
## S3 method for class 'court_overlap_audit'
print(x, ...)
```

Arguments

`x`	A `court_overlap_audit` object.
`...`	Additional arguments (unused).

Value

Invisibly returns the input `court_overlap_audit` object of class `c("court_overlap_audit", "list")`. Called primarily for its side effect of printing the court overlap audit summary to the console.

`print.delay_decomposition` *Print method for delay_decomposition*

Description

Print method for delay_decomposition

Usage

```
## S3 method for class 'delay_decomposition'
print(x, ...)
```

Arguments

`x`	A `delay_decomposition` object.
`...`	Additional arguments (unused).

Value

Invisibly returns the input `delay_decomposition` object of class `c("delay_decomposition", "list")`. Called primarily for its side effect of printing the linear predictor decomposition summary to the console.

print.delay_model *Print method for delay_model*

Description

Print method for delay_model

Usage

```
## S3 method for class 'delay_model'
print(x, ...)
```

Arguments

x	A delay_model object.
...	Additional arguments (unused).

Value

Invisibly returns the input delay_model object of class c("delay_model", "list"). Called primarily for its side effect of printing the fitted delay model summary and hazard ratios to the console.

print.diffused_profiles *Print method for diffused_profiles*

Description

Print method for diffused_profiles

Usage

```
## S3 method for class 'diffused_profiles'
print(x, ...)
```

Arguments

x	A diffused_profiles object.
...	Additional arguments (unused).

Value

Invisibly returns the input diffused_profiles object of class "diffused_profiles". Called primarily for its side effect of printing profile dimensions and metadata to the console.

print.future_neighbour_audit

Print method for future_neighbour_audit

Description

Print method for future_neighbour_audit

Usage

```
## S3 method for class 'future_neighbour_audit'
print(x, ...)
```

Arguments

x	A future_neighbour_audit object.
...	Additional arguments (unused).

Value

Invisibly returns the input future_neighbour_audit object of class c("future_neighbour_audit", "list"). Called primarily for its side effect of printing the temporal leakage audit summary to the console.

print.model_validation

Print method for model_validation

Description

Print method for model_validation

Usage

```
## S3 method for class 'model_validation'
print(x, ...)
```

Arguments

x	A model_validation object.
...	Additional arguments (unused).

Value

Invisibly returns the input model_validation object of class c("model_validation", "list"). Called primarily for its side effect of printing temporal holdout validation diagnostics and C-index metrics to the console.

print.near_duplicate_audit *Print method for near_duplicate_audit*

Description

Print method for near_duplicate_audit

Usage

```
## S3 method for class 'near_duplicate_audit'
print(x, ...)
```

Arguments

x	A near_duplicate_audit object.
...	Additional arguments (unused).

Value

Invisibly returns the input near_duplicate_audit object of class c("near_duplicate_audit", "list"). Called primarily for its side effect of printing same-date/court duplicate cluster audit findings to the console.

print.neighbourhood_survival *Print method for neighbourhood_survival*

Description

Print method for neighbourhood_survival

Usage

```
## S3 method for class 'neighbourhood_survival'
print(x, ...)
```

Arguments

x	A neighbourhood_survival object.
...	Additional arguments (unused).

Value

Invisibly returns the input neighbourhood_survival object of class "neighbourhood_survival". Called primarily for its side effect of printing neighbourhood Kaplan-Meier estimates and median survival duration to the console.

print.relative_lateness *Print method for relative_lateness*

Description

Print method for relative_lateness

Usage

```
## S3 method for class 'relative_lateness'
print(x, ...)
```

Arguments

x	A relative_lateness object.
...	Additional arguments (unused).

Value

Invisibly returns the input relative_lateness object of class "relative_lateness". Called primarily for its side effect of printing relative lateness percentile and diagnostic interpretation to the console.

project_validation_legal_score *Project Validation Case Legal Profiles Against Training Representation Space*

Description

Projects validation cases onto a training-derived legal section representation space using training section graph diffusion, training IDF weights, and similarity against training cases only. This ensures validation case legal scores are calculated strictly from training information, preventing future-neighbour leakage and validation-to-validation contamination.

Usage

```
project_validation_legal_score(
  val_sections,
  train_sections,
  k = 3,
  use_idf = TRUE,
  alpha = 0.7
)
```

Arguments

`val_sections`	A `data.frame` of legal section records for validation cases (must contain `case_id`, `act`, `section`).
`train_sections`	A `data.frame` of legal section records for training cases (must contain `case_id`, `act`, `section`).
`k`	Positive integer specifying the maximum number of training neighbours each validation case retains. Defaults to `3`.
`use_idf`	Logical flag indicating whether to apply training inverse document frequency (IDF) weighting. Defaults to `TRUE`.
`alpha`	Numeric diffusion parameter in `[0, 1]`. Defaults to `0.7`.

Details

Information Boundary & Projection Mechanics:

1. **Training Representation:** The section graph, section transition matrix P_{train}, and IDF weights w_{train} are constructed exclusively from `train_sections`.
2. **Unseen Sections:** Any section present in a validation case that was absent from `train_sections` is assigned zero contribution (ignored in representation space) without modifying training vocabulary or matrices.
3. **Independent Projection:** Each validation case is projected independently against training cases. Validation cases cannot form edges to each other (`val_cases_using_val_neighbours = 0`).

Value

A `list` containing:

validation_legal_scores Named numeric vector of projected legal scores for validation cases.

unseen_section_cases Character vector of validation case IDs containing sections absent from the training section vocabulary.

val_cases_using_train_neighbours Count of validation cases with at least 1 training neighbour.

val_cases_using_val_neighbours Count of validation cases using validation neighbours (always `0L` by design).

neighbour_map List mapping each validation case ID to a data frame of its top training neighbours and similarity weights.

Examples

```
train_sec <- data.frame(
  case_id = c("1", "2", "2"),
  act = c("IPC", "IPC", "CrPC"),
  section = c("420", "420", "154"),
  stringsAsFactors = FALSE
)
val_sec <- data.frame(
  case_id = c("3", "3"),
```

```
  act = c("IPC", "NI"),
  section = c("420", "138"),
  stringsAsFactors = FALSE
)
proj <- project_validation_legal_score(val_sec, train_sec, k = 2)
print(proj$validation_legal_scores)
```

`relative_lateness` *Compute Relative Lateness of a Case*

Description

Evaluates a target case's current age or resolution duration relative to the censoring-aware survival timeline of its legally similar neighbouring cases.

Usage

```
relative_lateness(neighbourhood)
```

Arguments

`neighbourhood`	An object of class `neighbourhood_survival` produced by `neighbourhood_survival()`.

Details

Estimability Rules: Relative position is considered `estimable = TRUE` if either:

1. The survival curve has reached `0.0` at or before the target duration T_i.
2. At least one resolution event ($\delta = 1$) occurred in the neighbourhood AND the target duration T_i does not exceed the maximum follow-up duration of the available neighbours.

Otherwise, `estimable = FALSE` and `relative_position` is set to `NA_real_` (e.g. when all neighbours are censored, or when the target duration exceeds the maximum follow-up of all neighbours).

Step-Function Convention: When estimable, $S_i(T_i)$ is evaluated using Kaplan-Meier right-continuous step function logic:

- If T_i is before the first event time: $S_i(T_i) = 1.0$.
- If T_i lies between event times: $S_i(T_i)$ equals the survival probability of the most recent event time $\leq T_i$.

Value

An S3 object of class `"relative_lateness"` containing:

target_case_id Character string of the target case ID.

target_status Character status (`"pending"` or `"disposed"`).

target_duration Observed duration/age of the target case (days).

survival_at_duration Survival probability $S_i(T_i)$ from the neighbourhood curve at duration T_i, or `NA_real_` if not estimable.

relative_position Relative position percentile $1 - S_i(T_i)$ in `[0, 1]`, or `NA_real_` if not estimable.

estimable Logical flag indicating whether the relative position is statistically estimable (`TRUE`) or not (`FALSE`).

interpretation Human-readable non-causal summary interpretation.

n_neighbours Count of valid neighbouring cases used.

median_neighbourhood_survival Median resolution time of neighbours (days), or `NA_real_`.

neighbourhood The input `neighbourhood_survival` object.

Examples

```
cases <- data.frame(
  case_id = 1:3,
  filing_date = as.Date(c("2012-01-10", "2012-03-05", "2012-06-20")),
  decision_date = as.Date(c("2012-11-02", "2013-02-14", NA))
)
cleaned <- clean_cases(cases, observation_end = "2014-12-31")
sec <- data.frame(
  case_id = c(1, 1, 2, 2, 3),
  act = c("NI Act", "NI Act", "NI Act", "NI Act", "IPC"),
  section = c("138", "141", "138", "142", "302")
)
sg <- build_section_graph(sec)
dp <- diffuse_profiles(sec, sg)
cg <- build_case_graph(dp, k = 2)
km_res <- neighbourhood_survival(cg, cleaned, case_id = 1)
lateness <- relative_lateness(km_res)
print(lateness)
```

`validate_models` *Temporal Validation of Delay Models*

Description

Evaluates out-of-sample statistical discrimination (Harrell's C-index) and calibration of a fitted `delay_model` using a chronological temporal holdout split based on filing dates. Audits for potential future-neighbour leakage when predictors are constructed over a full-sample case graph, and evaluates a strictly leakage-aware validation pipeline using training-only graph construction and validation projection.

Usage

```
validate_models(
  fit,
  data = NULL,
  case_graph = NULL,
  time_horizon = NULL,
  temporal_split = 0.7,
  seed = 123,
  sections = NULL,
  k = 3,
  alpha = 0.7,
  use_idf = TRUE
)
```

Arguments

fit	A delay_model object produced by fit_delay_model.
data	Optional data.frame of cases for validation. Defaults to fit$model_data.
case_graph	Optional igraph graph of case similarities. Defaults to fit$case_graph.
time_horizon	Optional numeric time horizon (duration) for calibration evaluation. Defaults to median duration.
temporal_split	Numeric fraction specifying the proportion of earliest filing date cases assigned to training. Defaults to 0.7.
seed	Optional integer random seed for reproducibility. Defaults to 123.
sections	Optional data.frame of legal section records used for leakage-aware validation projection.
k	Positive integer specifying top-k neighbours for case graph. Defaults to 3.
alpha	Numeric diffusion weight in [0, 1]. Defaults to 0.7.
use_idf	Logical flag indicating whether to use IDF weighting. Defaults to TRUE.

Value

An S3 object of class c("model_validation", "list") containing:

available Logical indicating whether temporal validation was successfully evaluated.

temporal_split Numeric fraction used for splitting (e.g. 0.7).

split_date Date cutoff threshold separating training and validation cases.

n_train Number of training observations.

n_val Number of validation observations.

events_train Number of resolution events in the training subset.

events_val Number of resolution events in the validation subset.

train_date_range Vector of minimum and maximum filing dates in training.

val_date_range Vector of minimum and maximum filing dates in validation.

training_c_index Harrell's concordance index on the training set.

validation_c_index Harrell's concordance index on the held-out validation set.

naive_validation_c_index C-index computed using full-sample graph legal scores.

leakage_aware_validation_c_index C-index computed using training-only graph construction and validation projection.

future_neighbour_leakage Logical indicating whether full-sample training cases have future neighbours.

val_cases_using_val_neighbours Count of validation cases using validation neighbours (0 by design).

val_cases_using_train_neighbours Count of validation cases using training neighbours.

unseen_section_val_cases Count of validation cases containing sections unseen in training.

full_model_diagnostics List containing hazard ratios and `cox.zph` proportional hazards test.

interpretation Explicit non-causal statistical safeguard statement.

message Character description of validation status.

Examples

```
cases <- data.frame(
  case_id = paste0("C", 1:20),
  court = "C1",
  filing_date = as.Date(rep(c("2012-01-10", "2012-02-10"), each = 10)),
  duration = c(50, 80, 110, 140, 70, 90, 130, 160, 120, 100,
               60, 85, 115, 145, 75, 95, 135, 165, 125, 105),
  event = rep(c(1, 1, 1, 0, 1), 4),
  stringsAsFactors = FALSE
)
cl <- data.frame(
  court = "C1",
  date = as.Date(c("2012-01-01", "2012-02-01")),
  load = c(10, 20),
  stringsAsFactors = FALSE
)
edges <- data.frame(
  from = paste0("C", 1:19),
  to = paste0("C", 2:20),
  weight = rep(c(0.8, 0.7, 0.6, 0.9), length.out = 19),
  stringsAsFactors = FALSE
)
g <- igraph::graph_from_data_frame(edges, directed = FALSE)
fit <- fit_delay_model(cases, g, cl)
val <- validate_models(fit, temporal_split = 0.7)
print(val)
```

Index

build_case_graph, 2, *6*, *14*, *16*
build_section_graph, 4, *13*

check_court_overlap, 5
check_future_neighbours, 6
check_near_duplicates, 7
clean_cases, 8, *10*, *14*, *16*
court_load, 10, *14*
cox.zph, *15*
coxph, *12*, *15*

decompose_delay, 11
diffuse_profiles, *3*, 12

fit_delay_model, *11*, 14, *26*
formula, *15*

neighbourhood_survival, 16, *24, 25*

print.court_load, 17
print.court_overlap_audit, 18
print.delay_decomposition, 18
print.delay_model, 19
print.diffused_profiles, 19
print.future_neighbour_audit, 20
print.model_validation, 20
print.near_duplicate_audit, 21
print.neighbourhood_survival, 21
print.relative_lateness, 22
project_validation_legal_score, 22

relative_lateness, 24

validate_models, 25