Package: irelink (via r-universe)

May 20, 2026

Title Fast Probabilistic Record Linkage

Version 0.0.1

Description Performs fast, scalable probabilistic record linkage and deduplication using the Fellegi-Sunter model. Records lacking a shared unique identifier are compared across configurable dimensions using exact, fuzzy, and distance-based comparisons, with model parameters estimated via unsupervised Expectation-Maximization. Multiple SQL backends are supported through 'DBI', enabling execution from laptop-scale ('DuckDB') through to distributed engines. This package is a translation of the Python 'splink' library by Linacre et al. into idiomatic R.

License MIT + file LICENSE

Depends R (>= 4.1.0)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Imports cli, DBI, duckdb, ggplot2, glue, rlang, stringdist, tibble, tidyselect

Suggests dbplyr, dplyr, igraph, jsonlite, knitr, nanoparquet, rmarkdown, RSQLite, testthat (>= 3.0.0), withr

Config/testthat/edition 3

URL http://christophertkenny.com/irelink/, https://github.com/christopherkenny/irelink

BugReports https://github.com/christopherkenny/irelink/issues

VignetteBuilder knitr

LazyData true

Config/Needs/website christopherkenny/ctktemplate

Config/pak/sysreqs xz-utils

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

Date/Publication 2026-05-20 18:49:52 UTC

RemoteUrl https://github.com/christopherkenny/irelink

RemoteRef HEAD

RemoteSha 51c3d6028221872ca626a454879bd2f15930dad1

Contents

autoplot.il_accuracy *Plot Accuracy Metrics Across Thresholds*

Description

Draws precision, recall, and F1 against the match-probability threshold. The data is produced by `il_accuracy()`.

Usage

```
## S3 method for class 'il_accuracy'
autoplot(object, ...)
```

Arguments

`object`	An `il_accuracy` tibble.
`...`	Additional arguments (currently unused).

Value

A `ggplot2::ggplot()` object.

autoplot.il_comparator_score *Plot Batch Comparator Scores*

Description

Plot Batch Comparator Scores

Usage

```
## S3 method for class 'il_comparator_score'
autoplot(object, ...)
```

Arguments

`object`	An `il_comparator_score` tibble.
`...`	Additional arguments (currently unused).

Value

A `ggplot2::ggplot()` object.

autoplot.il_compared *Quick Plot for Scored Pairs*

Description

Produces a match-weight histogram from scored pairs, or a waterfall chart for a single pair when `which` is provided. This is a convenience wrapper around `ggplot2::ggplot()`. For full control, build a plot directly from the prediction result or from `il_waterfall()`.

Usage

```
## S3 method for class 'il_compared'
autoplot(object, which = NULL, ...)
```

Arguments

`object`	An `il_compared` tibble from `predict.il_model()`.
`which`	An optional integer index. If provided, produces a waterfall chart for that pair. If `NULL` (default), produces a histogram.
`...`	Additional arguments (currently unused).

Value

A `ggplot2::ggplot()` object.

autoplot.il_comparison_vectors *Plot Comparison Vector Distribution*

Description

Plot Comparison Vector Distribution

Usage

```
## S3 method for class 'il_comparison_vectors'
autoplot(object, ...)
```

Arguments

`object`	An `il_comparison_vectors` tibble.
`...`	Additional arguments (currently unused).

Value

A `ggplot2::ggplot()` object showing the top comparison patterns by frequency.

autoplot.il_completeness

Plot Column Completeness

Description

Draws a grouped bar chart of non-null percentages per column, from data produced by `il_completeness()`.

Usage

```
## S3 method for class 'il_completeness'
autoplot(object, ...)
```

Arguments

`object`	An `il_completeness` tibble.
`...`	Additional arguments (currently unused).

Value

A `ggplot2::ggplot()` object.

autoplot.il_count_pairs

Plot Blocking Rule Pair Counts

Description

Draws a horizontal bar chart of candidate pairs generated by each blocking rule, from data produced by `il_count_pairs()`.

Usage

```
## S3 method for class 'il_count_pairs'
autoplot(object, type = c("additional", "raw"), ...)
```

Arguments

`object`	An `il_count_pairs` tibble.
`type`	One of `"additional"` (default) to show the incremental pairs each rule adds beyond those already covered by earlier rules, or `"raw"` to show the total pairs each rule generates independently.
`...`	Additional arguments (currently unused).

Value

A `ggplot2::ggplot()` object.

autoplot.il_model *Quick Match-Weights Plot for a Model*

Description

Produces a ready-made chart from a trained model. By default draws the match-weights bar chart. Set `type = "parameters"` for an m / u probability comparison. For full control, extract data with `il_weights()` or `il_parameters()` and build a custom `ggplot2::ggplot()`.

Usage

```
## S3 method for class 'il_model'
autoplot(object, type = c("weights", "parameters"), ...)
```

Arguments

`object`	A trained `il_model` object.
`type`	One of `"weights"` (default) or `"parameters"`. `"weights"` shows log-2 Bayes factors per comparison level. `"parameters"` shows m and u probabilities side by side.
`...`	Additional arguments (currently unused).

Value

A `ggplot2::ggplot()` object.

autoplot.il_precision_recall *Plot Precision–Recall Curve*

Description

Draws a precision–recall curve from the data produced by `il_precision_recall()`.

Usage

```
## S3 method for class 'il_precision_recall'
autoplot(object, ...)
```

Arguments

`object`	An `il_precision_recall` tibble.
`...`	Additional arguments (currently unused).

Value

A `ggplot2::ggplot()` object.

autoplot.il_profile *Plot Column Value Profiles*

Description

Draws a faceted bar chart of value frequencies per column, from data produced by `il_profile()`.

Usage

```
## S3 method for class 'il_profile'
autoplot(object, ...)
```

Arguments

`object`	An `il_profile` tibble.
`...`	Additional arguments (currently unused).

Value

A `ggplot2::ggplot()` object.

autoplot.il_roc *Plot ROC Curve*

Description

Draws a receiver operating characteristic curve from the data produced by `il_roc()`.

Usage

```
## S3 method for class 'il_roc'
autoplot(object, ...)
```

Arguments

`object`	An `il_roc` tibble.
`...`	Additional arguments (currently unused).

Value

A `ggplot2::ggplot()` object.

`autoplot.il_string_similarity`
Comparator Score Bar Chart

Description

Visualizes the output of `il_string_similarity()` as a horizontal bar chart, making it easy to compare multiple string-distance metrics at a glance.

Usage

```
## S3 method for class 'il_string_similarity'
autoplot(object, ...)
```

Arguments

`object`	An `il_string_similarity` tibble (the return value of `il_string_similarity()`).
`...`	Additional arguments (currently unused).

Value

A `ggplot2::ggplot()` object.

Examples

```
ggplot2::autoplot(il_string_similarity('John', 'Jon'))
```

`autoplot.il_training_history`
Plot EM Training History

Description

Draws parameter estimates across EM iterations, faceted by comparison, from data produced by `il_training_history()`.

Usage

```
## S3 method for class 'il_training_history'
autoplot(object, ...)
```

Arguments

`object`	An `il_training_history` tibble.
`...`	Additional arguments (currently unused).

Value

A ggplot2::ggplot() object.

`autoplot.il_unlinkables` *Plot Unlinkables Curve*

Description

Draws the proportion of records that cannot be linked at each match-probability threshold, from data produced by `il_unlinkables()`.

Usage

```
## S3 method for class 'il_unlinkables'
autoplot(object, ...)
```

Arguments

`object`	An `il_unlinkables` tibble.
`...`	Additional arguments (currently unused).

Value

A ggplot2::ggplot() object.

`block_from_labels` *Derive Blocking Rules from Labeled Pairs*

Description

For each column, computes the fraction of true-match pairs that share the same value (recall). Helps identify which columns make effective blocking keys.

Usage

```
block_from_labels(.data, labels, columns = NULL, con = NULL)
```

Arguments

`.data`	A data frame or character table name.
`labels`	A data frame with `unique_id_l`, `unique_id_r`, and `is_match`.
`columns`	Character vector of column names to evaluate. `NULL` for all non-ID columns.
`con`	A DBI connection from DBI::dbConnect().

Value

A tibble::tibble() with columns column, recall (fraction of true matches caught), and n_matches_caught.

Examples

```
con <- DBI::dbConnect(duckdb::duckdb())
labels <- data.frame(
  unique_id_l = fake_1000_labels$unique_id_l,
  unique_id_r = fake_1000_labels$unique_id_r,
  is_match = as.integer(fake_1000_labels$clerical_match_score >= 0.5)
)
block_from_labels(fake_1000, labels, con = con)
DBI::dbDisconnect(con, shutdown = TRUE)
```

---

block_on *Create a Training-Time Blocking Rule*

Description

Creates a blocking rule for use inside training verbs such as il_estimate_em() and il_estimate_prior(). This is distinct from il_block_on(), which adds prediction-time blocking to a specification. The returned object describes how to partition pairs during training.

Usage

```
block_on(..., .where = NULL, .transform = NULL, .explode = NULL)
```

Arguments

...	Column names (bare or column ~ transform formulas). See il_block_on() for details on the formula syntax. Columns are AND-ed within a single block_on() call.
.where	An optional raw SQL string for non-equality blocking conditions (e.g., "levenshtein(l.dob, r.dob) <= 1"). When supplied alongside column names, the column equalities and the SQL condition are AND-ed together.
.transform	An optional transform applied to every column that does not already have a formula transform. See il_block_on() for details.
.explode	An optional character vector of array column names to unnest before blocking. See il_block_on() for details.

Value

A blocking-rule object for use in training verbs.

Examples

```
block_on(first_name, surname)

# Fuzzy SQL conditions
block_on(first_name, .where = 'levenshtein(l.dob, r.dob) <= 1')

# Phonetic blocking
block_on(first_name, .transform = il_soundex)

# Per-column substring blocking
block_on(first_name ~ il_substr(1, 3), surname ~ il_substr(1, 4))
```

---

cl_and *Combine Comparison Conditions with AND*

Description

Creates a compound level that fires only when all supplied conditions are satisfied.

Usage

```
cl_and(...)
```

Arguments

...	Comparison-level objects to AND together.

Value

A comparison-level object.

Examples

```
cl_and(cl_exact(), cl_jaro_winkler(0.9))
```

---

cl_array_intersect *Array Intersection Comparison*

Description

Creates comparison levels based on the number of shared elements between two array or list columns. Thresholds are integer counts, ordered from strictest (most shared elements required) to most lenient.

Usage

```
cl_array_intersect(...)
```

Arguments

...	Integer count thresholds, ordered from strictest to most lenient (e.g., `2, 1`).

Value

A comparison-level object for use in `il_compare()`.

Examples

```
il_spec() |>
  il_compare(tags, cl_array_intersect(2, 1))
```

---

cl_array_min_distance

Pairwise Array Minimum Distance Comparison

Description

Creates comparison levels based on the best-matching pair of values across two array columns. For each record pair, every element of the left array is compared against every element of the right array. The best score (maximum similarity for `'jaro_winkler'`, minimum distance for `'levenshtein'`) is then tested against each threshold.

Usage

```
cl_array_min_distance(fn = c("jaro_winkler", "levenshtein"), ...)
```

Arguments

fn	Distance function: `'jaro_winkler'` (default) or `'levenshtein'`. For `'jaro_winkler'`, thresholds are similarity scores (0–1, descending, strictest first). For `'levenshtein'`, thresholds are edit distances (non-negative integers, ascending, strictest first).
...	Numeric thresholds, from strictest to most lenient.

Details

On DuckDB the pairwise comparison runs in SQL via an UNNEST cross-join scalar subquery. On SQLite it falls back to an R-side nested apply.

Value

A comparison-level object for use in `il_compare()`.

Examples

```
# Jaro-Winkler: best pairwise similarity >= 0.9 or >= 0.7
il_spec() |>
  il_compare(aliases, cl_array_min_distance('jaro_winkler', 0.9, 0.7))

# Levenshtein: best pairwise edit distance <= 1 or <= 2
il_spec() |>
  il_compare(aliases, cl_array_min_distance('levenshtein', 1, 2))
```

---

cl_array_subset *Array Subset Comparison*

Description

Creates a comparison level that matches when the smaller of two array columns is a complete subset of the larger. In other words, every element of the smaller array appears in the larger one.

Usage

```
cl_array_subset()
```

Details

On DuckDB and PostgreSQL this is computed in SQL using `ARRAY_LENGTH(ARRAY_INTERSECT(...)) = LEAST(ARRAY_LENGTH(...))`. On SQLite it falls back to an R-side set check.

Value

A comparison-level object for use in `il_compare()`.

Examples

```
il_spec() |>
  il_compare(qualifications, cl_array_subset())
```

---

cl_columns_reversed *Swap Detection for Two Columns*

Description

Creates a comparison level that fires when two column values are transposed between the left and right records (e.g., first name and surname accidentally swapped).

Usage

```
cl_columns_reversed(col_name_2, symmetrical = FALSE)
```

Arguments

`col_name_2`	Name of the second column (character). The first column is the one passed to `il_compare()`.
`symmetrical`	Logical. If `TRUE`, checks both directions: `l.col1 = r.col2 AND l.col2 = r.col1`. If `FALSE` (default), checks one direction only: `l.col1 = r.col2`.

Details

Use inside `cl_levels()` to add a swap-detection level to a custom comparison. For a ready-made name comparison that already includes swap detection, see `cl_forename_surname()`.

Value

A comparison-level object for use in `il_compare()` or `cl_levels()`.

Examples

```
# Detect swapped first/last names inside a custom comparison
il_spec() |>
  il_compare(
    first_name,
    cl_levels(
      cl_null(),
      cl_exact(),
      cl_columns_reversed('surname', symmetrical = TRUE),
      cl_else()
    )
  )
```

`cl_cosine` *Cosine Similarity Comparison*

Description

Creates comparison levels based on cosine similarity. Suitable for numeric or vectorised columns. Thresholds are between 0 and 1, ordered from strictest to most lenient.

Usage

```
cl_cosine(...)
```

Arguments

`...`	Numeric thresholds between 0 and 1, ordered from strictest to most lenient.

Value

A comparison-level object for use in `il_compare()`.

Examples

```
il_spec() |>
  il_compare(embedding, cl_cosine(0.8))
```

---

cl_custom *Custom SQL Comparison*

Description

Creates a comparison level from a raw SQL expression. Use this when none of the built-in `cl_*()` helpers fit. The SQL should reference `l.` and `r.` prefixed column names for the left and right records. This is a tagged-string helper with processing semantics.

Usage

```
cl_custom(sql_expr, ...)
```

Arguments

`sql_expr`	A character string containing a valid SQL expression.
`...`	Reserved for future use.

Value

A comparison-level object for use in `il_compare()`.

Examples

```
il_spec() |>
  il_compare(score, cl_custom('l.score + r.score > 10'))
```

---

cl_damerau_levenshtein *Damerau-Levenshtein Edit-Distance Comparison*

Description

Creates comparison levels based on the Damerau-Levenshtein distance, which extends `cl_levenshtein()` by also counting transpositions of two adjacent characters as a single edit.

Usage

```
cl_damerau_levenshtein(..., term_frequency = FALSE)
```

Arguments

`...`	Integer distance thresholds, ordered from strictest to most lenient.
`term_frequency`	Logical. If `TRUE`, adjust match weights by value frequency at the highest comparison level. Defaults to `FALSE`.

Value

A comparison-level object for use in `il_compare()`.

Examples

```
il_spec() |>
  il_compare(name, cl_damerau_levenshtein(1))
```

---

cl_date_diff *Date Difference Comparison*

Description

Creates comparison levels based on the absolute difference between two dates. Thresholds should use the unit helpers `days()`, `months()`, or `years()` for self-documenting, unit-safe specifications. Bare numerics are interpreted as days.

Usage

```
cl_date_diff(...)
```

Arguments

`...`	Duration thresholds created by `days()`, `months()`, or `years()`, ordered from strictest to most lenient.

Value

A comparison-level object for use in `il_compare()`.

Examples

```
il_spec() |>
  il_compare(dob, cl_date_diff(days(30), days(365)))

# Mix units freely
il_spec() |>
  il_compare(dob, cl_date_diff(months(1), years(1)))
```

cl_dob *Date of Birth Comparison*

Description

A pre-built domain comparison for dates of birth. Combines exact matching, a Damerau-Levenshtein string check for transposed digits, and configurable date-difference levels to handle common errors.

Usage

```
cl_dob(
  thresholds = list(months(1), years(1), years(10)),
  term_frequency = FALSE
)
```

Arguments

`thresholds`	A list of unit-tagged threshold values created by `days()`, `months()`, or `years()`, ordered from strictest to most lenient. Defaults to `list(months(1), years(1), years(10))`.
`term_frequency`	Logical. If `TRUE`, adjust match weights by date-of-birth frequency at the highest comparison level. Defaults to `FALSE`.

Value

A comparison-level object for use in `il_compare()`.

Examples

```
il_spec() |>
  il_compare(dob, cl_dob())

# Custom thresholds: within 7 days, 6 months, 2 years
il_spec() |>
  il_compare(dob, cl_dob(thresholds = list(days(7), months(6), years(2))))
```

cl_else *Catch-All Else Level*

Description

Creates a residual level that matches any pair not captured by previous levels. Typically used as the last level inside `cl_levels()`.

Usage

```
cl_else()
```

Value

A comparison-level object.

Examples

```
cl_levels(cl_null(), cl_exact(), cl_else())
```

---

cl_email *Email Address Comparison*

Description

A pre-built domain comparison for email addresses. Provides levels for exact match, username-only match, and domain-only match.

Usage

```
cl_email(term_frequency = FALSE)
```

Arguments

`term_frequency`
: Logical. If `TRUE`, adjust match weights by email frequency at the highest comparison level. Defaults to `FALSE`.

Value

A comparison-level object for use in `il_compare()`.

Examples

```
il_spec() |>
  il_compare(email, cl_email())
```

cl_exact *Exact Equality Comparison*

Description

Creates a comparison level that scores an exact match on a column. Optionally applies term-frequency adjustments so that rare values (e.g., an uncommon surname) receive higher match weights than common ones.

Usage

```
cl_exact(term_frequency = FALSE)
```

Arguments

`term_frequency`
: Logical. If `TRUE`, adjust match weights by value frequency. Defaults to `FALSE`.

Value

A comparison-level object for use in `il_compare()`.

Examples

```
il_spec() |>
  il_compare(city, cl_exact()) |>
  il_compare(county, cl_exact(term_frequency = TRUE))
```

cl_first_last_name *First Name and Last Name Comparison with Swap Detection*

Description

An American-English alias for `cl_forename_surname()`. Compares first and last name columns, including a swap-detection level for accidentally transposed names. Pass this to `il_compare()` on the first-name column and supply the companion last-name column via `last_name`.

Usage

```
cl_first_last_name(last_name = "last_name", term_frequency = FALSE)
```

Arguments

`last_name`	Name of the last name column in the data. Defaults to `'last_name'`.
`term_frequency`	Logical. If `TRUE`, adjust match weights by name frequency at the highest comparison level. Defaults to `FALSE`.

Value

A comparison-level object for use in `il_compare()`.

Examples

```
il_spec() |>
  il_compare(first_name, cl_first_last_name())
```

`cl_forename_surname` *Forename and Surname Comparison with Swap Detection*

Description

A pre-built domain comparison that compares forename and surname columns, including a cross-field swap-detection level (where first name and surname are accidentally transposed). Pass this to `il_compare()` on the forename/first-name column and supply the companion surname/last-name column via `surname`.

Usage

```
cl_forename_surname(surname = "surname", term_frequency = FALSE)
```

Arguments

`surname`	Name of the surname column in the data. Defaults to `'surname'`.
`term_frequency`	Logical. If `TRUE`, adjust match weights by name frequency at the highest comparison level. Defaults to `FALSE`.

Details

See also `cl_first_last_name()` for an American-English alias.

Value

A comparison-level object for use in `il_compare()`.

Examples

```
il_spec() |>
  il_compare(first_name, cl_forename_surname(surname = 'last_name'))
```

cl_geo_distance *Geographic Distance Comparison*

Description

Creates comparison levels based on the great-circle distance between two latitude/longitude pairs. Thresholds should use the unit helpers `km()` or `mi()` for clarity.

Usage

```
cl_geo_distance(...)
```

Arguments

`...`	Distance thresholds created by `km()` or `mi()`, ordered from strictest to most lenient.

Value

A comparison-level object for use in `il_compare()`.

Examples

```
il_spec() |>
  il_compare(c(lat, lon), cl_geo_distance(km(5), km(50)))

# Use miles instead
il_spec() |>
  il_compare(c(lat, lon), cl_geo_distance(mi(3), mi(30)))
```

cl_jaccard *Jaccard Set Similarity Comparison*

Description

Creates comparison levels based on the Jaccard index, the ratio of the intersection to the union of character n-gram sets. Thresholds are between 0 and 1, ordered from strictest to most lenient.

Usage

```
cl_jaccard(...)
```

Arguments

`...`	Numeric thresholds between 0 and 1, ordered from strictest to most lenient.

Value

A comparison-level object for use in `il_compare()`.

Examples

```
il_spec() |>
  il_compare(name, cl_jaccard(0.9))
```

`cl_jaro` *Jaro String Similarity Comparison*

Description

Creates comparison levels based on the Jaro similarity score (0 to 1). A simpler variant of `cl_jaro_winkler()` without the prefix bonus.

Usage

```
cl_jaro(..., term_frequency = FALSE)
```

Arguments

`...`	Numeric thresholds between 0 and 1, ordered from strictest to most lenient.
`term_frequency`	Logical. If `TRUE`, adjust match weights by value frequency at the highest comparison level. Defaults to `FALSE`.

Value

A comparison-level object for use in `il_compare()`.

Examples

```
il_spec() |>
  il_compare(name, cl_jaro(0.9))
```

cl_jaro_winkler *Jaro-Winkler String Similarity Comparison*

Description

Creates comparison levels based on the Jaro-Winkler similarity score (0 to 1). Thresholds are passed as unnamed arguments ordered from strictest to most lenient.

Usage

```
cl_jaro_winkler(..., term_frequency = FALSE)
```

Arguments

`...`	Numeric thresholds between 0 and 1, ordered from strictest to most lenient (e.g., `0.9, 0.7`).
`term_frequency`	Logical. If `TRUE`, adjust match weights by value frequency at the highest comparison level. Defaults to `FALSE`.

Value

A comparison-level object for use in `il_compare()`.

Examples

```
il_spec() |>
  il_compare(first_name, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(surname, cl_jaro_winkler(0.9, term_frequency = TRUE))
```

cl_levels *Compose Custom Comparison Levels*

Description

Assembles an ordered list of comparison levels from individual level constructors. Use this when the built-in `cl_*()` helpers do not fit and you need full control over the level hierarchy.

Usage

```
cl_levels(..., term_frequency = FALSE)
```

Arguments

`...`	Level objects created by `cl_null()`, `cl_exact()`, `cl_jaro_winkler()`, `cl_else()`, and similar helpers.
`term_frequency`	Logical. If `TRUE`, adjust match weights by value frequency at the highest comparison level. Defaults to `FALSE`.

Value

A comparison-level object for use in `il_compare()`.

Examples

```
il_spec() |>
  il_compare(
    name,
    cl_levels(
      cl_null(),
      cl_exact(),
      cl_jaro_winkler(0.95),
      cl_jaro_winkler(0.88),
      cl_else(),
      term_frequency = TRUE
    )
  )
```

`cl_levenshtein` *Levenshtein Edit-Distance Comparison*

Description

Creates comparison levels based on the Levenshtein edit distance (minimum number of single-character insertions, deletions, or substitutions). Thresholds are integer counts, ordered from strictest (smallest distance) to most lenient.

Usage

```
cl_levenshtein(..., term_frequency = FALSE)
```

Arguments

`...`	Integer distance thresholds, ordered from strictest to most lenient (e.g., `1, 2`).
`term_frequency`	Logical. If `TRUE`, adjust match weights by value frequency at the highest comparison level. Defaults to `FALSE`.

Value

A comparison-level object for use in `il_compare()`.

Examples

```
il_spec() |>
  il_compare(name, cl_levenshtein(1, 2))
```

`cl_literal` *Literal Value Comparison*

Description

Creates a comparison level that checks whether a column equals a fixed literal value on the left record, right record, or both. This is useful as a gate inside `cl_levels()` to restrict a comparison to records with a known value (e.g., only compare names when `country = 'US'`).

Usage

```
cl_literal(value, side = c("both", "left", "right"))
```

Arguments

`value`	A scalar value to compare against. Character values are quoted in the generated SQL. Numerics are not.
`side`	Which record to check: `'both'` (default), `'left'`, or `'right'`.

Value

A comparison-level object for use in `il_compare()` or `cl_levels()`.

Examples

```
il_spec() |>
  il_compare(
    country,
    cl_levels(
      cl_null(),
      cl_literal('US', side = 'both'),
      cl_exact(),
      cl_else()
    )
  )
```

cl_name *Personal Name Comparison*

Description

A pre-built domain comparison for personal names. Combines exact matching and Jaro-Winkler levels with thresholds tuned for typical name variation. Optionally adds a Soundex phonetic level as a final fallback before the else level, which helps catch names that sound similar but are spelled differently (e.g., Smith/Smyth).

Usage

```
cl_name(term_frequency = FALSE, phonetic = FALSE)
```

Arguments

term_frequency	Logical. If TRUE, adjust match weights by name frequency at the highest comparison level. Defaults to FALSE.
phonetic	Logical. If TRUE, add a cl_soundex() level as a fallback before the else level. Defaults to FALSE.

Value

A comparison-level object for use in il_compare().

Examples

```
il_spec() |>
  il_compare(first_name, cl_name()) |>
  il_compare(surname, cl_name(term_frequency = TRUE))

il_spec() |>
  il_compare(first_name, cl_name(phonetic = TRUE))
```

cl_not *Negate a Comparison Condition*

Description

Creates a level that fires when the supplied condition does **not** hold.

Usage

```
cl_not(x)
```

Arguments

x	A comparison-level object to negate.

Value

A comparison-level object.

Examples

```
cl_not(cl_exact())
```

---

cl_null *Null / Missing Value Level*

Description

Creates a level that fires when either or both record values are NULL or NA. Typically used as the first level inside cl_levels().

Usage

```
cl_null()
```

Value

A comparison-level object.

Examples

```
cl_levels(cl_null(), cl_exact(), cl_else())
```

---

cl_numeric_diff *Numeric Absolute Difference Comparison*

Description

Creates comparison levels based on the absolute difference between two numeric values. Thresholds are ordered from strictest (smallest permitted difference) to most lenient.

Usage

```
cl_numeric_diff(...)
```

Arguments

...	Numeric difference thresholds, ordered from strictest to most lenient (e.g., 1, 5).

Value

A comparison-level object for use in `il_compare()`.

Examples

```
il_spec() |>
  il_compare(age, cl_numeric_diff(1, 5))
```

---

cl_or — *Combine Comparison Conditions with OR*

Description

Creates a compound level that fires when any of the supplied conditions are satisfied.

Usage

```
cl_or(...)
```

Arguments

`...`	Comparison-level objects to OR together.

Value

A comparison-level object.

Examples

```
cl_or(cl_jaro_winkler(0.9), cl_levenshtein(1))
```

---

cl_pct_diff — *Numeric Percentage Difference Comparison*

Description

Creates comparison levels based on the relative percentage difference between two numeric values. Thresholds are fractions (e.g., `0.05` for 5%), ordered from strictest to most lenient.

Usage

```
cl_pct_diff(...)
```

Arguments

`...`	Numeric percentage thresholds, ordered from strictest to most lenient (e.g., `0.05, 0.2`).

Value

A comparison-level object for use in `il_compare()`.

Examples

```
il_spec() |>
  il_compare(income, cl_pct_diff(0.05, 0.2))
```

---

cl_postcode *Postcode Comparison*

Description

A pre-built domain comparison for postcodes. Supports exact matching and prefix-based partial matching. Optionally appends geographic distance fallback levels when latitude and longitude columns are available.

Usage

```
cl_postcode(
  term_frequency = FALSE,
  lat_col = NULL,
  long_col = NULL,
  km_thresholds = c(1, 10, 100)
)
```

Arguments

`term_frequency`
: Logical. If `TRUE`, adjust match weights by postcode frequency at the highest comparison level. Defaults to `FALSE`.

`lat_col`, `long_col`
: Character. Names of latitude and longitude columns. Both must be supplied together. When provided, geographic distance levels are appended before `cl_else()`.

`km_thresholds`
: Numeric vector of distance thresholds in kilometres, ordered from strictest to most lenient. Only used when `lat_col` and `long_col` are supplied. Defaults to `c(1, 10, 100)`.

Value

A comparison-level object for use in `il_compare()`.

Examples

```
il_spec() |>
  il_compare(postcode, cl_postcode())

# With geographic fallback (requires lat/lon columns in the data)
il_spec() |>
  il_compare(postcode, cl_postcode(lat_col = 'lat', long_col = 'lon'))
```

---

cl_soundex *Soundex Phonetic Comparison*

Description

Creates a comparison level based on the Soundex phonetic algorithm. Two strings match if their Soundex codes are identical. This can be used as a fallback level within `cl_levels()` or `cl_name()` to catch names that sound similar but are spelled differently (e.g., Smith/Smyth, Robert/Rupert).

Usage

```
cl_soundex()
```

Details

On DuckDB, Soundex runs via a registered SQL MACRO. On PostgreSQL, it uses the native `soundex()` function. On SQLite, it falls back to an R-side implementation.

Value

A comparison-level object for use in `il_compare()` or `cl_levels()`.

Examples

```
il_spec() |>
  il_compare(first_name, cl_soundex())

il_spec() |>
  il_compare(
    first_name,
    cl_levels(
      cl_null(),
      cl_exact(),
      cl_jaro_winkler(0.9),
      cl_soundex(),
      cl_else()
    )
  )
```

cl_time_diff *Time Difference Comparison*

Description

Creates comparison levels based on the absolute difference between two datetime (timestamp) values. Thresholds should use the unit helpers `seconds()`, `minutes()`, `hours()`, `days()`, `months()`, or `years()` for self-documenting, unit-safe specifications. Bare numerics are interpreted as seconds.

Usage

```
cl_time_diff(...)
```

Arguments

`...`	Duration thresholds created by `seconds()`, `minutes()`, `hours()`, `days()`, `months()`, or `years()`, ordered from strictest to most lenient.

Details

This extends `cl_date_diff()` to support sub-day precision for timestamp columns. Use `cl_date_diff()` for date-only columns.

Value

A comparison-level object for use in `il_compare()`.

Examples

```
il_spec() |>
  il_compare(timestamp, cl_time_diff(minutes(5), hours(1)))

# Mix units freely
il_spec() |>
  il_compare(timestamp, cl_time_diff(seconds(30), minutes(10), hours(2)))
```

cl_zip_code *ZIP Code Comparison*

Description

A pre-built domain comparison for US ZIP codes. Provides levels for exact match, 5-digit prefix match (normalizes ZIP+4 against plain 5-digit codes), and 3-digit Sectional Center Facility (SCF) prefix match. Accepts both plain 5-digit (`'90210'`) and ZIP+4 (`'90210-3456'`) formats. Optionally appends geographic distance fallback levels when latitude and longitude columns are available.

Usage

```
cl_zip_code(
  term_frequency = FALSE,
  lat_col = NULL,
  long_col = NULL,
  km_thresholds = c(1, 10, 100)
)
```

Arguments

`term_frequency`	Logical. If `TRUE`, adjust match weights by ZIP code frequency at the highest comparison level. Defaults to `FALSE`.
`lat_col`, `long_col`	Character. Names of latitude and longitude columns. Both must be supplied together. When provided, geographic distance levels are appended before `cl_else()`.
`km_thresholds`	Numeric vector of distance thresholds in kilometres, ordered from strictest to most lenient. Only used when `lat_col` and `long_col` are supplied. Defaults to `c(1, 10, 100)`.

Value

A comparison-level object for use in `il_compare()`.

Examples

```
il_spec() |>
  il_compare(zip, cl_zip_code())

# With geographic fallback (requires lat/lon columns in the data)
il_spec() |>
  il_compare(zip, cl_zip_code(lat_col = 'lat', long_col = 'lon'))
```

`days` *Create a Duration in Days*

Description

A tagged-value constructor that marks a numeric threshold as a number of days. Use inside `cl_date_diff()` for self-documenting, unit-safe thresholds.

Usage

```
days(n)
```

Arguments

`n`	A non-negative numeric value.

Value

A tagged numeric with class `il_days`.

Examples

```
il_spec() |>
  il_compare(dob, cl_date_diff(days(30), days(365)))
```

`fake_1000` *Splink Fake 1000: Deduplication Benchmark*

Description

A dataset of 1,000 synthetic records representing 181 unique people, each with varying numbers of duplicate entries. Duplicates have been corrupted with typographical errors, missing values, and other realistic data-quality issues. This is the primary demo dataset from the Python splink library.

Usage

```
fake_1000
```

Format

A tibble with 1,000 rows and 7 columns:

unique_id Integer. Row identifier (0-indexed).

first_name Character. Given name, sometimes corrupted or missing.

surname Character. Family name, sometimes corrupted or missing.

dob Character. Date of birth in `YYYY-MM-DD` format.

city Character. City of residence, sometimes missing.

email Character. Email address, sometimes corrupted or missing.

cluster Integer. Ground-truth entity label (0-indexed).

Details

The `cluster` column provides ground-truth entity labels: records sharing the same cluster value refer to the same person. The `unique_id` column provides a unique identifier for each row, starting at 0 (matching splink convention).

Source

From the splink datasets repository maintained by the UK Ministry of Justice Analytical Services: https://github.com/moj-analytical-services/splink_datasets. Original data generated by the splink team (Linacre et al.) under the MIT license.

See Also

fake_1000_labels for pairwise clerical labels.

fake_1000_labels *Splink Fake 1000: Clerical Pairwise Labels*

Description

Pairwise clerical labels for the fake_1000 dataset. Each row records whether a pair of records from `fake_1000` is a true match (`clerical_match_score = 1`) or a non-match (`clerical_match_score = 0`). These labels enable evaluation of model accuracy, ROC curves, and precision-recall metrics.

Usage

```
fake_1000_labels
```

Format

A tibble with 3,176 rows and 5 columns:

unique_id_l Integer. `unique_id` of the left record.

source_dataset_l Character. Source dataset name (`"fake_1000"`).

unique_id_r Integer. `unique_id` of the right record.

source_dataset_r Character. Source dataset name (`"fake_1000"`).

clerical_match_score Numeric. 1 for a match, 0 for a non-match.

Source

From the splink datasets repository maintained by the UK Ministry of Justice Analytical Services: https://github.com/moj-analytical-services/splink_datasets. Original data generated by the splink team (Linacre et al.) under the MIT license.

fake_20 *Fake 20: Minimal Deduplication Example*

Description

A small, hand-crafted dataset of 20 records representing 5 unique people. Each person has four records with varying levels of corruption: exact matches, minor typos, and slightly shifted dates of birth. Designed for quick examples and unit tests.

Usage

```
fake_20
```

Format

A tibble with 20 rows and 6 columns:

first_name Character. Given name, sometimes corrupted.

surname Character. Family name, sometimes corrupted.

dob Character. Date of birth in `YYYY-MM-DD` format, sometimes shifted by one day.

city Character. City of residence.

email Character. Email address, sometimes corrupted.

cluster Integer. Ground-truth entity label (1 to 5).

See Also

fake_1000 for a larger benchmark dataset.

`febrl4a` *FEBRL 4a: Record Linkage Original Records*

Description

The FEBRL (Freely Extensible Biomedical Record Linkage) dataset 4a contains 5,000 original records. It is designed to be linked against febrl4b, which contains one duplicate record per original. Ground truth is encoded in `rec_id`: records sharing the same base ID (e.g., `rec-1070-org` and `rec-1070-dup-0`) refer to the same entity.

Usage

```
febrl4a
```

Format

A tibble with 5,000 rows and 11 columns:

rec_id Character. Record identifier encoding entity and origin (`-org` suffix).

given_name Character. Given name, sometimes missing.

surname Character. Family name, sometimes missing.

street_number Integer. Street number, sometimes missing.

address_1 Character. Primary address line, sometimes missing.

address_2 Character. Secondary address line, often missing.

suburb Character. Suburb or neighborhood.

postcode Integer. Postal code.

state Character. Australian state abbreviation.

date_of_birth Integer. Date of birth as `YYYYMMDD` integer, sometimes missing.

soc_sec_id Integer. Social security identifier.

Source

Distributed via the splink datasets repository (https://github.com/moj-analytical-services/splink_datasets) under the MIT license. The FEBRL datasets originate from Christen and Churches (2004) and are widely used as record-linkage benchmarks.

References

Christen, P. and Churches, T. (2004). Febrl – Freely Extensible Biomedical Record Linkage. Australian National University.

See Also

febrl4b for the corresponding duplicate records.

`febrl4b` *FEBRL 4b: Record Linkage Duplicate Records*

Description

The FEBRL (Freely Extensible Biomedical Record Linkage) dataset 4b contains 5,000 duplicate records, one for each original in febrl4a. Duplicates have been corrupted with typographical errors, missing values, and transpositions. Ground truth is encoded in `rec_id`: the base number matches the corresponding original (e.g., `rec-1070-dup-0` matches `rec-1070-org`).

Usage

```
febrl4b
```

Format

A tibble with 5,000 rows and 11 columns:

rec_id Character. Record identifier encoding entity and duplicate status (`-dup-0` suffix).

given_name Character. Given name, sometimes corrupted or missing.

surname Character. Family name, sometimes corrupted or missing.

street_number Integer. Street number, sometimes missing.

address_1 Character. Primary address line, sometimes corrupted.

address_2 Character. Secondary address line, often missing.

suburb Character. Suburb or neighborhood.

postcode Integer. Postal code.

state Character. Australian state abbreviation.

date_of_birth Integer. Date of birth as `YYYYMMDD` integer, sometimes missing.

soc_sec_id Integer. Social security identifier.

Source

Distributed via the splink datasets repository (https://github.com/moj-analytical-services/splink_datasets) under the MIT license. The FEBRL datasets originate from Christen and Churches (2004) and are widely used as record-linkage benchmarks.

References

Christen, P. and Churches, T. (2004). Febrl – Freely Extensible Biomedical Record Linkage. Australian National University.

See Also

febrl4a for the corresponding original records.

`hours` *Create a Duration in Hours*

Description

A tagged-value constructor that marks a numeric threshold as a number of hours. Use inside `cl_time_diff()` for self-documenting, unit-safe thresholds.

Usage

```
hours(n)
```

Arguments

`n`	A non-negative numeric value.

Value

A tagged numeric with class `il_hours`.

Examples

```
il_spec() |>
  il_compare(timestamp, cl_time_diff(hours(2), hours(24)))
```

`il_accuracy` *Accuracy Metrics Across Thresholds*

Description

Computes a full suite of classification metrics at a range of match-probability thresholds. Requires labeled pairs.

Usage

```
il_accuracy(model, labels = NULL, labels_col = NULL)
```

Arguments

`model`	A trained `il_model` object.
`labels`	A data frame of labeled pairs with a logical or integer match indicator. Required unless `labels_col` is provided.
`labels_col`	Optional string naming a column in the original data containing ground-truth cluster/entity IDs. When provided, pairwise labels are derived automatically via `labels_from_column()`.

Value

A `tibble::tibble()` with one row per threshold, containing columns `threshold`, `tp`, `fp`, `fn`, `tn`, `fn_blocking_miss`, `precision`, `recall`, `f1`, `f2`, `f0_5`, `specificity`, `npv`, `accuracy`, `p4`, and `phi`.

Examples

```
df <- data.frame(
  unique_id = 1:20,
  first_name = c(
    'John', 'Jon', 'Jane', 'Jane', 'Bob',
    'Bobby', 'Alice', 'Alicia', 'Tom', 'Thomas',
    'John', 'Jon', 'Jane', 'Janet', 'Bob',
    'Robert', 'Alice', 'Alison', 'Tom', 'Tomas'
  ),
  surname = c(
    'Smith', 'Smith', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Brown', 'White', 'White',
    'Smith', 'Smyth', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Browne', 'White', 'White'
  ),
  dob = c(
    '1990-01-01', '1990-01-01', '1985-06-15', '1985-06-15',
    '2000-12-01', '2000-12-01', '1975-03-22', '1975-03-22',
    '1988-07-04', '1988-07-04', '1990-01-01', '1990-01-02',
    '1985-06-15', '1985-06-16', '2000-12-01', '2000-12-02',
    '1975-03-22', '1975-03-23', '1988-07-04', '1988-07-05'
```

```
  ),
  city = c(
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid',
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid'
  ),
  email = c(
    'john@example.com', 'jon@example.com', 'jane@example.com',
    'jane@example.com', 'bob@example.com', 'bobby@example.com',
    'alice@example.com', 'alicia@example.com', 'tom@example.com',
    'thomas@example.com', 'john@example.com', 'jon@example.com',
    'jane@example.com', 'janet@example.com', 'bob@example.com',
    'robert@example.com', 'alice@example.com', 'alison@example.com',
    'tom@example.com', 'tomas@example.com'
  )
)
con <- DBI::dbConnect(duckdb::duckdb())
spec <- il_spec() |>
  il_compare(first_name, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(surname, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(dob, cl_exact()) |>
  il_block_on(surname) |>
  il_block_on(first_name)
model <- il_model(df, spec = spec, con = con)
model <- il_estimate_u(model)
model <- il_estimate_em(model, block_on(surname))
labels <- data.frame(
  unique_id_l = c(1L, 1L),
  unique_id_r = c(11L, 2L),
  is_match = c(1L, 0L)
)

il_accuracy(model, labels = labels)
DBI::dbDisconnect(con, shutdown = TRUE)
```

`il_array_element` *Array Element Column Transform*

Description

Returns a transform that extracts the first or last element of an array-valued column. The result can be passed as the `transform` argument to `il_compare()` or `il_block_on()`, and composed with other transforms via `il_transform()`. On DuckDB and PostgreSQL, maps to SQL array indexing (`col[1]` or `col[-1]`).

Usage

```
il_array_element(position = c("first", "last"))
```

Arguments

position	Either "first" or "last".

Value

An il_column_transform closure.

Examples

```
tf <- il_array_element('first')
tf(list(c('Alice', 'A'), c('Bob'), character(0)))
```

---

il_attach *Attach a Saved Model to Fresh Data*

Description

Takes a loaded (or existing) il_model and binds it to new data and a fresh database connection, producing a model ready for predict() or further training. Accepts in-memory data frames, dbplyr::tbl_lazy table references, or character table names.

Usage

```
il_attach(model, .data, ..., con = NULL, link_type = NULL)
```

Arguments

model	An il_model object, typically from il_load().
.data	A data frame, tibble::tibble(), dbplyr::tbl_lazy, or character table name. The first (or only) input dataset.
...	Additional datasets for multi-table linkage.
con	A DBI connection object from DBI::dbConnect(). Optional when .data is a dbplyr::tbl_lazy, the connection is extracted from the table reference.
link_type	Optionally override the model's link type. If NULL (default), uses the link type stored in the model.

Details

This is the key function for the production workflow: train once with il_model() -> save with il_save() -> later, load with il_load() and attach to new data with il_attach().

The loaded model's trained parameters (m, u, prior) are preserved. You can immediately call predict() on the attached model, or continue training with il_estimate_em() using the existing parameters as a warm start.

Value

The model, now connected to con with data uploaded, ready for predict(), il_find_matches(), or further training.

Examples

```
con <- DBI::dbConnect(duckdb::duckdb())
spec <- il_spec() |>
  il_compare(first_name, cl_jaro_winkler(0.9, 0.7)) |>
  il_block_on(surname)
model <- il_model(fake_1000, spec = spec, con = con)
model <- il_estimate_u(model)
model <- il_estimate_em(model, block_on(surname))
path <- tempfile(fileext = '.rds')
il_save(model, path)
DBI::dbDisconnect(con, shutdown = TRUE)
con2 <- DBI::dbConnect(duckdb::duckdb())
loaded <- il_load(path)
model2 <- il_attach(loaded, fake_1000, con = con2)
DBI::dbDisconnect(con2, shutdown = TRUE)
```

`il_block_on` *Add a Prediction Blocking Rule*

Description

Adds an equality-based blocking rule to a specification. During prediction, only record pairs that agree on the blocking columns are scored. Multiple calls are OR-ed together. Within a single call, columns are AND-ed.

Usage

```
il_block_on(spec, ..., .where = NULL, .transform = NULL, .explode = NULL)
```

Arguments

`spec`	An `il_spec` object (piped in).
`...`	Columns for equality blocking (AND-ed within one call). Each entry is either: • A bare column name, e.g. `surname`. • A `column ~ transform` formula, e.g. `first_name ~ il_substr(1, 3)`, which applies the transform to that column before the equality check. Mix bare names and formulas freely within one call.
`.where`	An optional raw SQL string for non-equality blocking conditions. Defaults to `NULL`.
`.transform`	An optional transform applied to every column that does not already have a formula transform. Can be a single function (e.g. il_soundex) or a named list of functions for per-column transforms. Formula transforms in `...` take precedence over `.transform`.
`.explode`	An optional character vector of column names containing arrays (list columns) to unnest before blocking. Each array element becomes a separate row for the blocking join. Requires a DuckDB or PostgreSQL backend. Defaults to `NULL`.

Value

An updated copy of `spec`.

Examples

```
# Block on state OR first name (two calls = OR)
spec <- il_spec() |>
  il_block_on(state) |>
  il_block_on(first_name)

# Block where state AND year both match (one call = AND)
spec <- il_spec() |>
  il_block_on(state, year)

# Per-column substring blocking with formula syntax
spec <- il_spec() |>
  il_block_on(first_name ~ il_substr(1, 3), surname ~ il_substr(1, 4))

# Mix: substr on one column, plain match on another
spec <- il_spec() |>
  il_block_on(postcode_fake ~ il_substr(1, 3), dob)

# Same transform on all columns
spec <- il_spec() |>
  il_block_on(first_name, .transform = il_soundex)

# Explode array columns before blocking
spec <- il_spec() |>
  il_block_on(email, .explode = 'email')
```

`il_cast_to_string` *Cast to String Column Transform*

Description

Returns a transform that casts any column to a character/VARCHAR type. Useful when a numeric or date column needs to be compared as text. The result can be passed as the `transform` argument to `il_compare()` or `il_block_on()`, and composed with other transforms via `il_transform()`. On DuckDB and PostgreSQL, maps to SQL `CAST(col AS VARCHAR)`.

Usage

```
il_cast_to_string()
```

Value

An `il_column_transform` closure.

Examples

```
tf <- il_cast_to_string()
tf(c(12345L, 67890L))
```

`il_cleanup` *Remove Model-Owned Temporary Tables from Database*

Description

Cleans up the temporary tables owned by a single `il_model`. This is safe to call on a shared DBI connection from `DBI::dbConnect()` containing other live `irelink` models. Use `il_cleanup_all()` only when you explicitly want to remove every `irelink` table from the connection.

Usage

```
il_cleanup(model)
```

Arguments

`model`	An `il_model` object whose connection and tables should be cleaned up.

Value

`model`, invisibly.

Examples

```
df <- data.frame(
  unique_id = 1:20,
  first_name = c(
    'John', 'Jon', 'Jane', 'Jane', 'Bob',
    'Bobby', 'Alice', 'Alicia', 'Tom', 'Thomas',
    'John', 'Jon', 'Jane', 'Janet', 'Bob',
    'Robert', 'Alice', 'Alison', 'Tom', 'Tomas'
  ),
  surname = c(
    'Smith', 'Smith', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Brown', 'White', 'White',
    'Smith', 'Smyth', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Browne', 'White', 'White'
  ),
  dob = c(
    '1990-01-01', '1990-01-01', '1985-06-15', '1985-06-15',
    '2000-12-01', '2000-12-01', '1975-03-22', '1975-03-22',
    '1988-07-04', '1988-07-04', '1990-01-01', '1990-01-02',
    '1985-06-15', '1985-06-16', '2000-12-01', '2000-12-02',
    '1975-03-22', '1975-03-23', '1988-07-04', '1988-07-05'
  ),
```

```
  city = c(
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid',
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid'
  ),
  email = c(
    'john@example.com', 'jon@example.com', 'jane@example.com',
    'jane@example.com', 'bob@example.com', 'bobby@example.com',
    'alice@example.com', 'alicia@example.com', 'tom@example.com',
    'thomas@example.com', 'john@example.com', 'jon@example.com',
    'jane@example.com', 'janet@example.com', 'bob@example.com',
    'robert@example.com', 'alice@example.com', 'alison@example.com',
    'tom@example.com', 'tomas@example.com'
  )
)
con <- DBI::dbConnect(duckdb::duckdb())
spec <- il_spec() |>
  il_compare(first_name, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(surname, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(dob, cl_exact()) |>
  il_block_on(surname) |>
  il_block_on(first_name)
model <- il_model(df, spec = spec, con = con)
model <- il_estimate_u(model)
model <- il_estimate_em(model, block_on(surname))

il_cleanup(model)
DBI::dbDisconnect(con, shutdown = TRUE)
```

`il_cleanup_all` *Remove All irelink Temporary Tables from a Database*

Description

Drops every table or view whose name starts with `__il_` from a DBI connection. This is intended as an explicit interactive escape hatch after failed runs or exploratory sessions. Prefer `il_cleanup()` when cleaning up a specific live model on a shared connection.

Usage

```
il_cleanup_all(con)
```

Arguments

`con`	A DBI connection from `DBI::dbConnect()`.

Value

`con`, invisibly.

Examples

```
df <- data.frame(
  unique_id = 1:4,
  name = c('Ann', 'Anne', 'Bob', 'Rob')
)
con <- DBI::dbConnect(duckdb::duckdb())
spec <- il_spec() |>
  il_compare(name, cl_jaro_winkler(0.9)) |>
  il_block_on(name)
model <- il_model(df, spec = spec, con = con)

il_cleanup_all(con)
DBI::dbDisconnect(con, shutdown = TRUE)
```

---

il_cluster *Cluster Scored Pairs into Entities*

Description

Groups scored record pairs into entity clusters using graph-based methods. The result assigns cluster IDs to records that represent the same real-world entity.

Usage

```
il_cluster(
  pairs,
  threshold = NULL,
  method = c("connected", "best_link"),
  ties_method = c("lowest_id", "drop"),
  source_dataset = NULL
)
```

Arguments

`pairs`	An `il_compared` tibble from `predict.il_model()`.
`threshold`	An optional secondary match-probability threshold. If `NULL` (the default), the threshold from prediction is used.
`method`	One of `"connected"` (default) for connected-components clustering, or `"best_link"` for single-best-link clustering.
`ties_method`	How to handle tied best-link probabilities when `method = "best_link"`. `"lowest_id"` (default) keeps the edge to the record with the smaller `unique_id`. `"drop"` removes all edges where the best-link probability is tied.
`source_dataset`	An optional named character vector or data frame mapping `unique_id` values to their source dataset name. Used with `method = "best_link"` to enforce at-most-one-record per source dataset per cluster. If supplied,

it must cover every `unique_id` present in `pairs`. If a data frame, it must contain columns `unique_id` and `source_dataset`, and `unique_id` values must be unique.

Value

A `tibble::tibble()` with one row per input record, including a `cluster_id` column.

Examples

```
df <- data.frame(
  unique_id = 1:20,
  first_name = c(
    'John', 'Jon', 'Jane', 'Jane', 'Bob',
    'Bobby', 'Alice', 'Alicia', 'Tom', 'Thomas',
    'John', 'Jon', 'Jane', 'Janet', 'Bob',
    'Robert', 'Alice', 'Alison', 'Tom', 'Tomas'
  ),
  surname = c(
    'Smith', 'Smith', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Brown', 'White', 'White',
    'Smith', 'Smyth', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Browne', 'White', 'White'
  ),
  dob = c(
    '1990-01-01', '1990-01-01', '1985-06-15', '1985-06-15',
    '2000-12-01', '2000-12-01', '1975-03-22', '1975-03-22',
    '1988-07-04', '1988-07-04', '1990-01-01', '1990-01-02',
    '1985-06-15', '1985-06-16', '2000-12-01', '2000-12-02',
    '1975-03-22', '1975-03-23', '1988-07-04', '1988-07-05'
  ),
  city = c(
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid',
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid'
  ),
  email = c(
    'john@example.com', 'jon@example.com', 'jane@example.com',
    'jane@example.com', 'bob@example.com', 'bobby@example.com',
    'alice@example.com', 'alicia@example.com', 'tom@example.com',
    'thomas@example.com', 'john@example.com', 'jon@example.com',
    'jane@example.com', 'janet@example.com', 'bob@example.com',
    'robert@example.com', 'alice@example.com', 'alison@example.com',
    'tom@example.com', 'tomas@example.com'
  )
)
con <- DBI::dbConnect(duckdb::duckdb())
spec <- il_spec() |>
  il_compare(first_name, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(surname, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(dob, cl_exact()) |>
  il_block_on(surname) |>
```

```
  il_block_on(first_name)
model <- il_model(df, spec = spec, con = con)
model <- il_estimate_u(model)
model <- il_estimate_em(model, block_on(surname))

pairs <- predict(model, threshold = 0.5)
clusters <- il_cluster(pairs)
DBI::dbDisconnect(con, shutdown = TRUE)
```

`il_cluster_confusion_matrix` *Cluster-Level Confusion Matrix for Deduplication*

Description

Computes a record-level confusion matrix after clustering predicted matches into entities. A record is treated as "duplicated" if it is not the first record in its predicted cluster, and likewise for the ground-truth `labels_col`.

Usage

```
il_cluster_confusion_matrix(
  model,
  labels_col,
  threshold = 0.85,
  method = c("connected", "best_link"),
  ties_method = c("lowest_id", "drop"),
  source_dataset = NULL
)
```

Arguments

`model`	A trained `il_model` object for a deduplication task.
`labels_col`	String naming the ground-truth cluster/entity column in the model's source data.
`threshold`	Match-probability threshold passed to `predict()`. Defaults to `0.85`.
`method`	Clustering method passed to `il_cluster()`.
`ties_method`	Tie handling for `method = "best_link"`, passed to `il_cluster()`.
`source_dataset`	Optional source-dataset mapping passed to `il_cluster()`. If supplied, it must cover every `unique_id` in the predicted pairs, and duplicate `unique_id` mappings are not allowed.

Details

For DuckDB and PostgreSQL backends, pair scoring and clustering are pushed into SQL where possible. The final summary still returns a one-row tibble in R.

Value

A one-row tibble with columns `threshold`, `tp`, `fp`, `fn`, `tn`, `precision`, `recall`, and `f1`.

Examples

```
df <- data.frame(
  unique_id = 1:5,
  first_name = c('John', 'John', 'Mary', 'Bob', 'Bob'),
  surname = c('Smith', 'Smith', 'Jones', 'Brown', 'Brown'),
  cluster = c(1, 1, 2, 3, 4)
)
con <- DBI::dbConnect(duckdb::duckdb())
spec <- il_spec() |>
  il_compare(first_name, cl_exact()) |>
  il_compare(surname, cl_exact()) |>
  il_block_on(surname)
model <- il_model(df, spec = spec, con = con)
model <- il_estimate_u(model)
model <- il_estimate_em(model, block_on(surname))

il_cluster_confusion_matrix(model, labels_col = 'cluster', threshold = 0.85)
DBI::dbDisconnect(con, shutdown = TRUE)
```

`il_comparator_score` *Batch String Similarity Scores*

Description

Computes string-similarity metrics between two columns of a data frame or database table. Useful for profiling data quality and choosing comparison thresholds. Rows where either column is missing are omitted.

Usage

```
il_comparator_score(.data, col_1, col_2, con = NULL)
```

Arguments

`.data`	A data frame or character table name. Table names require `con`.
`col_1`, `col_2`	Column names (unquoted or character).
`con`	A DBI connection from `DBI::dbConnect()`. If `NULL`, uses R-side computation.

Details

With `con = NULL`, all metrics are computed in R with `stringdist::stringdist()`. With a `duckdb::duckdb()` or PostgreSQL connection, computation is pushed to SQL. SQL backends return the same column schema but may leave unsupported metrics as `NA`: DuckDB currently computes `jaro_winkler`, `jaro`, `levenshtein`, and `jaccard`; PostgreSQL computes `levenshtein` and a `jaro_winkler` compatibility column backed by trigram `similarity()`.

Value

A tibble::tibble() with the two input columns and metric columns jaro_winkler, jaro, levenshtein, jaccard, and cosine. Unsupported SQL-backend metrics are present as NA. The result has S3 class il_comparator_score.

Examples

```
df <- data.frame(
  name_l = c('John', 'Jane', 'Bob'),
  name_r = c('Jon', 'Janet', 'Bobby')
)
il_comparator_score(df, name_l, name_r)
```

---

il_comparator_threshold_chart

Comparator Score Threshold Chart

Description

Shows the distribution of pairwise similarity scores and highlights which pairs exceed a given threshold.

Usage

```
il_comparator_threshold_chart(
  .data,
  col_1,
  col_2,
  similarity_threshold = NULL,
  distance_threshold = NULL,
  con = NULL
)
```

Arguments

.data	A data frame or table name.
col_1, col_2	Column names (unquoted or character).
similarity_threshold	Numeric threshold for similarity metrics (>= to include). Applies to jaro_winkler, jaro, jaccard, cosine.
distance_threshold	Integer threshold for distance metrics (<= to include). Applies to levenshtein.
con	A DBI connection from DBI::dbConnect(). If NULL, uses R-side computation.

Value

A ggplot2::ggplot() object.

`il_compare` *Add a Comparison Layer to a Specification*

Description

Declares how one or more columns should be compared when scoring record pairs. Each call adds one comparison to the specification.

Usage

```
il_compare(
  spec,
  col,
  method,
  ...,
  transform = NULL,
  tf_adjustment_weight = 1,
  tf_minimum_u_value = 0
)
```

Arguments

`spec`	An `il_spec` object (piped in).
`col`	<tidy-select> Column(s) to compare. Accepts bare names, `c()`, and tidyselect helpers.
`method`	A comparison helper object created by a `cl_*()` function such as `cl_exact()` or `cl_jaro_winkler()`.
`...`	Reserved for future use.
`transform`	An optional transformation function applied to both left and right column values *before* comparison. Common choices include `tolower`, `toupper`, and `trimws`, which are automatically translated to SQL when a database backend is available. Custom functions work on the R-side path only.
`tf_adjustment_weight`	Numeric power to raise the term-frequency Bayes factor to. A value of `1.0` (the default) applies the full adjustment. Use `0` to disable it entirely. Only relevant when the comparison method has `term_frequency = TRUE`.
`tf_minimum_u_value`	Numeric floor for the term-frequency denominator. When both TF values are below this threshold, it is used instead, preventing unrealistically large match weights for very rare terms. Defaults to `0.0` (no floor).

Details

`col` accepts tidyselect expressions: a bare column name, `c(col_a, col_b)`, or helpers such as `tidyselect::starts_with()`. When multiple columns are targeted, each receives its own comparison layer with the same method.

Value

An updated copy of `spec`.

Examples

```
spec <- il_spec() |>
  il_compare(first_name, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(dob, cl_date_diff(days(30), days(365)))

# Apply a transform before comparing
spec <- il_spec() |>
  il_compare(first_name, cl_jaro_winkler(0.9, 0.7), transform = tolower)

# Scale TF adjustment weight
spec <- il_spec() |>
  il_compare(first_name, cl_jaro_winkler(0.9, term_frequency = TRUE),
    tf_adjustment_weight = 0.5, tf_minimum_u_value = 0.001
  )
```

`il_compare_records` *Compare Two Individual Records*

Description

Scores a single pair of records against a specification without requiring a full training pipeline. Useful for quick one-off comparisons or debugging.

Usage

```
il_compare_records(record_a, record_b, spec, con = NULL)
```

Arguments

`record_a`	A named list or single-row data frame representing the first record.
`record_b`	A named list or single-row data frame representing the second record.
`spec`	An `il_spec` object describing the comparisons to perform.
`con`	A DBI connection object from `DBI::dbConnect()`. If `NULL` (default), a temporary DuckDB connection is created and closed on exit.

Value

A single-row tibble of per-comparison gamma values.

Examples

```
df <- data.frame(
  unique_id = 1:20,
  first_name = c(
    'John', 'Jon', 'Jane', 'Jane', 'Bob',
    'Bobby', 'Alice', 'Alicia', 'Tom', 'Thomas',
    'John', 'Jon', 'Jane', 'Janet', 'Bob',
    'Robert', 'Alice', 'Alison', 'Tom', 'Tomas'
  ),
  surname = c(
    'Smith', 'Smith', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Brown', 'White', 'White',
    'Smith', 'Smyth', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Browne', 'White', 'White'
  ),
  dob = c(
    '1990-01-01', '1990-01-01', '1985-06-15', '1985-06-15',
    '2000-12-01', '2000-12-01', '1975-03-22', '1975-03-22',
    '1988-07-04', '1988-07-04', '1990-01-01', '1990-01-02',
    '1985-06-15', '1985-06-16', '2000-12-01', '2000-12-02',
    '1975-03-22', '1975-03-23', '1988-07-04', '1988-07-05'
  ),
  city = c(
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid',
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid'
  ),
  email = c(
    'john@example.com', 'jon@example.com', 'jane@example.com',
    'jane@example.com', 'bob@example.com', 'bobby@example.com',
    'alice@example.com', 'alicia@example.com', 'tom@example.com',
    'thomas@example.com', 'john@example.com', 'jon@example.com',
    'jane@example.com', 'janet@example.com', 'bob@example.com',
    'robert@example.com', 'alice@example.com', 'alison@example.com',
    'tom@example.com', 'tomas@example.com'
  )
)
con <- DBI::dbConnect(duckdb::duckdb())
spec <- il_spec() |>
  il_compare(first_name, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(surname, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(dob, cl_exact()) |>
  il_block_on(surname) |>
  il_block_on(first_name)
record_a <- df[1, ]
record_b <- df[2, ]

il_compare_records(record_a, record_b, spec = spec, con = con)
DBI::dbDisconnect(con, shutdown = TRUE)
```

`il_comparison_vectors` *Comparison Vector Distribution*

Description

Computes the distribution of gamma patterns (agreement vectors) across record pairs. Each unique combination of gamma values across comparisons is a "comparison vector". This function counts how often each pattern occurs.

Usage

```
il_comparison_vectors(model, blocking = NULL, limit = NULL)
```

Arguments

`model`	A trained `il_model`.
`blocking`	A blocking rule created by `block_on()`. If `NULL`, uses all blocking rules from the model spec.
`limit`	Maximum number of pairs to sample. Defaults to `NULL` (all pairs).

Details

On DuckDB/PostgreSQL, the computation runs entirely in SQL.

Value

A `tibble::tibble()` with one row per unique comparison vector and columns `gamma_<col>` for each comparison plus `count` (number of pairs with that pattern) and `proportion`. Class `il_comparison_vectors`.

Examples

```
con <- DBI::dbConnect(duckdb::duckdb())
spec <- il_spec() |>
  il_compare(first_name, cl_exact()) |>
  il_compare(surname, cl_exact())
model <- il_model(fake_20, spec = spec, con = con)
vectors <- il_comparison_vectors(model)
ggplot2::autoplot(vectors)
il_cleanup(model)
DBI::dbDisconnect(con, shutdown = TRUE)
```

`il_completeness` *Column Completeness Across Datasets*

Description

Computes the percentage of non-null values for each column across one or more datasets.

Usage

```
il_completeness(..., con = NULL)
```

Arguments

`...`	One or more data frames, dbplyr::tbl_lazy references, or character table names to profile.
`con`	A DBI connection object from `DBI::dbConnect()`. Optional when all inputs are dbplyr::tbl_lazy references.

Value

A `tibble::tibble()` with columns `table`, `column`, `n_total`, `n_non_null`, and `pct_non_null`.

Examples

```
df <- data.frame(
  unique_id = 1:20,
  first_name = c(
    'John', 'Jon', 'Jane', 'Jane', 'Bob',
    'Bobby', 'Alice', 'Alicia', 'Tom', 'Thomas',
    'John', 'Jon', 'Jane', 'Janet', 'Bob',
    'Robert', 'Alice', 'Alison', 'Tom', 'Tomas'
  ),
  surname = c(
    'Smith', 'Smith', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Brown', 'White', 'White',
    'Smith', 'Smyth', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Browne', 'White', 'White'
  ),
  dob = c(
    '1990-01-01', '1990-01-01', '1985-06-15', '1985-06-15',
    '2000-12-01', '2000-12-01', '1975-03-22', '1975-03-22',
    '1988-07-04', '1988-07-04', '1990-01-01', '1990-01-02',
    '1985-06-15', '1985-06-16', '2000-12-01', '2000-12-02',
    '1975-03-22', '1975-03-23', '1988-07-04', '1988-07-05'
  ),
  city = c(
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid',
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid'
```

```
  ),
  email = c(
    'john@example.com', 'jon@example.com', 'jane@example.com',
    'jane@example.com', 'bob@example.com', 'bobby@example.com',
    'alice@example.com', 'alicia@example.com', 'tom@example.com',
    'thomas@example.com', 'john@example.com', 'jon@example.com',
    'jane@example.com', 'janet@example.com', 'bob@example.com',
    'robert@example.com', 'alice@example.com', 'alison@example.com',
    'tom@example.com', 'tomas@example.com'
  )
)
con <- DBI::dbConnect(duckdb::duckdb())
il_completeness(df, con = con)
DBI::dbDisconnect(con, shutdown = TRUE)
```

`il_confusion_matrix` *Confusion Matrix at a Threshold*

Description

Computes the thresholded confusion-matrix counts for labeled pairs. Uses the same SQL-first scoring path as `il_accuracy()` on supported backends, so labeled pairs do not need to be predicted and collected in full before evaluation.

Usage

```
il_confusion_matrix(model, labels = NULL, threshold = 0.85, labels_col = NULL)
```

Arguments

`model`	A trained `il_model` object.
`labels`	A data frame of labeled pairs with a logical or integer match indicator. Required unless `labels_col` is provided.
`threshold`	A numeric value between 0 and 1 for classifying pairs as matches. Defaults to `0.85`.
`labels_col`	Optional string naming a column in the original data containing ground-truth cluster/entity IDs. When provided, pairwise labels are derived automatically via `labels_from_column()`.

Value

A one-row tibble containing `threshold`, `tp`, `fp`, `fn`, `tn`, `fn_blocking_miss`, `precision`, `recall`, and `f1`.

Examples

```
df <- data.frame(
  unique_id = 1:20,
  first_name = c(
    'John', 'Jon', 'Jane', 'Jane', 'Bob',
    'Bobby', 'Alice', 'Alicia', 'Tom', 'Thomas',
    'John', 'Jon', 'Jane', 'Janet', 'Bob',
    'Robert', 'Alice', 'Alison', 'Tom', 'Tomas'
  ),
  surname = c(
    'Smith', 'Smith', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Brown', 'White', 'White',
    'Smith', 'Smyth', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Browne', 'White', 'White'
  ),
  dob = c(
    '1990-01-01', '1990-01-01', '1985-06-15', '1985-06-15',
    '2000-12-01', '2000-12-01', '1975-03-22', '1975-03-22',
    '1988-07-04', '1988-07-04', '1990-01-01', '1990-01-02',
    '1985-06-15', '1985-06-16', '2000-12-01', '2000-12-02',
    '1975-03-22', '1975-03-23', '1988-07-04', '1988-07-05'
  )
)
con <- DBI::dbConnect(duckdb::duckdb())
spec <- il_spec() |>
  il_compare(first_name, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(surname, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(dob, cl_exact()) |>
  il_block_on(surname) |>
  il_block_on(first_name)
model <- il_model(df, spec = spec, con = con)
model <- il_estimate_u(model)
model <- il_estimate_em(model, block_on(surname))
labels <- data.frame(
  unique_id_l = c(1L, 1L),
  unique_id_r = c(11L, 2L),
  is_match = c(1L, 0L)
)

il_confusion_matrix(model, labels = labels, threshold = 0.85)
DBI::dbDisconnect(con, shutdown = TRUE)
```

---

il_constrain_m *Add a Fixed Matched-Class Constraint*

Description

Fixes one comparison's matched-class `m` probabilities during EM. This is a hard constraint, stored separately from regularizing priors.

Usage

```
il_constrain_m(model, col, exact = NULL, levels = NULL)
```

Arguments

`model`	An `il_model` object.
`col`	Comparison column, supplied as a bare name or string.
`exact`	Probability for the strongest gamma level.
`levels`	Complete named probability vector, with names such as `"0"`, `"1"`, and `"2"`.

Value

The model with constraint metadata in `model$params$constraints`.

`il_constraints` *Inspect Model Constraints*

Description

Inspect Model Constraints

Usage

```
il_constraints(model)
```

Arguments

`model`	An `il_model` object.

Value

A `tibble::tibble()` of stored fixed-constraint metadata.

`il_count_pairs` *Count Candidate Pairs Under Blocking Rules*

Description

Estimates how many record pairs each blocking rule generates without performing full comparisons. Useful for tuning blocking strategies before training. Too many pairs is slow, while too few misses matches.

Usage

```
il_count_pairs(.data, ..., con = NULL, link_type = c("dedupe", "link"))
```

Arguments

`.data`	A data frame, dbplyr::tbl_lazy, or character table name (first or only dataset).
`...`	Blocking rules created by `block_on()`, and optionally additional datasets for linkage.
`con`	A DBI connection object from `DBI::dbConnect()`. Optional when `.data` is a dbplyr::tbl_lazy.
`link_type`	One of `"dedupe"` (default) or `"link"`.

Value

A `tibble::tibble()` with columns `rule` and `n_pairs`. When blocking rules are supplied, it also includes `cumulative_pairs` and `pct_of_cartesian`.

Examples

```
df <- data.frame(
  unique_id = 1:20,
  first_name = c(
    'John', 'Jon', 'Jane', 'Jane', 'Bob',
    'Bobby', 'Alice', 'Alicia', 'Tom', 'Thomas',
    'John', 'Jon', 'Jane', 'Janet', 'Bob',
    'Robert', 'Alice', 'Alison', 'Tom', 'Tomas'
  ),
  surname = c(
    'Smith', 'Smith', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Brown', 'White', 'White',
    'Smith', 'Smyth', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Browne', 'White', 'White'
  ),
  dob = c(
    '1990-01-01', '1990-01-01', '1985-06-15', '1985-06-15',
    '2000-12-01', '2000-12-01', '1975-03-22', '1975-03-22',
    '1988-07-04', '1988-07-04', '1990-01-01', '1990-01-02',
```

```
    '1985-06-15', '1985-06-16', '2000-12-01', '2000-12-02',
    '1975-03-22', '1975-03-23', '1988-07-04', '1988-07-05'
  ),
  city = c(
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid',
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid'
  ),
  email = c(
    'john@example.com', 'jon@example.com', 'jane@example.com',
    'jane@example.com', 'bob@example.com', 'bobby@example.com',
    'alice@example.com', 'alicia@example.com', 'tom@example.com',
    'thomas@example.com', 'john@example.com', 'jon@example.com',
    'jane@example.com', 'janet@example.com', 'bob@example.com',
    'robert@example.com', 'alice@example.com', 'alison@example.com',
    'tom@example.com', 'tomas@example.com'
  )
)
con <- DBI::dbConnect(duckdb::duckdb())
il_count_pairs(
  df,
  block_on(surname),
  block_on(first_name),
  con = con
)
DBI::dbDisconnect(con, shutdown = TRUE)
```

`il_deterministic_link` *Deterministic Record Linkage*

Description

Finds exact-match record pairs using the blocking rules in the specification, without requiring probabilistic training. This is a common first step before probabilistic linkage. Pairs that match on all blocking columns are returned directly.

Usage

```
il_deterministic_link(
  .data,
  ...,
  spec,
  con = NULL,
  link_type = c("dedupe", "link", "link_and_dedupe")
)
```

Arguments

`.data`	A data frame, dbplyr::tbl_lazy, or character table name (first or only dataset).
`...`	Additional datasets for multi-table linkage.
`spec`	An `il_spec` object with blocking rules defined via `il_block_on()`.
`con`	A DBI connection object from `DBI::dbConnect()`. Optional when `.data` is a dbplyr::tbl_lazy.
`link_type`	One of `"dedupe"` (default), `"link"`, or `"link_and_dedupe"`.

Value

A `tibble::tibble()` of exact-match record pairs.

Examples

```
df <- data.frame(
  unique_id = 1:20,
  first_name = c(
    'John', 'Jon', 'Jane', 'Jane', 'Bob',
    'Bobby', 'Alice', 'Alicia', 'Tom', 'Thomas',
    'John', 'Jon', 'Jane', 'Janet', 'Bob',
    'Robert', 'Alice', 'Alison', 'Tom', 'Tomas'
  ),
  surname = c(
    'Smith', 'Smith', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Brown', 'White', 'White',
    'Smith', 'Smyth', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Browne', 'White', 'White'
  ),
  dob = c(
    '1990-01-01', '1990-01-01', '1985-06-15', '1985-06-15',
    '2000-12-01', '2000-12-01', '1975-03-22', '1975-03-22',
    '1988-07-04', '1988-07-04', '1990-01-01', '1990-01-02',
    '1985-06-15', '1985-06-16', '2000-12-01', '2000-12-02',
    '1975-03-22', '1975-03-23', '1988-07-04', '1988-07-05'
  ),
  city = c(
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid',
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid'
  ),
  email = c(
    'john@example.com', 'jon@example.com', 'jane@example.com',
    'jane@example.com', 'bob@example.com', 'bobby@example.com',
    'alice@example.com', 'alicia@example.com', 'tom@example.com',
    'thomas@example.com', 'john@example.com', 'jon@example.com',
    'jane@example.com', 'janet@example.com', 'bob@example.com',
    'robert@example.com', 'alice@example.com', 'alison@example.com',
    'tom@example.com', 'tomas@example.com'
```

```
  )
)
con <- DBI::dbConnect(duckdb::duckdb())
spec <- il_spec() |>
  il_compare(first_name, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(surname, cl_jaro_winkler(0.9, 0.7)) |>
  il_block_on(first_name, surname, dob)

exact_matches <- il_deterministic_link(df, spec = spec, con = con)
DBI::dbDisconnect(con, shutdown = TRUE)
```

`il_errors` *Identify Prediction Errors*

Description

Compares model predictions against labeled pairs and returns all false-positive and false-negative errors at a given threshold. Useful for understanding which record pairs the model gets wrong.

Usage

```
il_errors(model, labels = NULL, threshold = 0.85, labels_col = NULL)
```

Arguments

`model`	A trained `il_model` object.
`labels`	A data frame of labeled pairs with a logical or integer match indicator. Required unless `labels_col` is provided.
`threshold`	A numeric value between 0 and 1 for classifying pairs as matches. Defaults to `0.85`.
`labels_col`	Optional string naming a column in the original data containing ground-truth cluster/entity IDs.

Value

A `tibble::tibble()` of misclassified pairs with columns `unique_id_l`, `unique_id_r`, `match_weight`, `match_probability`, `true_label`, and `error_type`.

Examples

```
df <- data.frame(
  unique_id = 1:20,
  first_name = c(
    'John', 'Jon', 'Jane', 'Jane', 'Bob',
    'Bobby', 'Alice', 'Alicia', 'Tom', 'Thomas',
    'John', 'Jon', 'Jane', 'Janet', 'Bob',
    'Robert', 'Alice', 'Alison', 'Tom', 'Tomas'
```

```
  ),
  surname = c(
    'Smith', 'Smith', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Brown', 'White', 'White',
    'Smith', 'Smyth', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Browne', 'White', 'White'
  ),
  dob = c(
    '1990-01-01', '1990-01-01', '1985-06-15', '1985-06-15',
    '2000-12-01', '2000-12-01', '1975-03-22', '1975-03-22',
    '1988-07-04', '1988-07-04', '1990-01-01', '1990-01-02',
    '1985-06-15', '1985-06-16', '2000-12-01', '2000-12-02',
    '1975-03-22', '1975-03-23', '1988-07-04', '1988-07-05'
  ),
  city = c(
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid',
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid'
  ),
  email = c(
    'john@example.com', 'jon@example.com', 'jane@example.com',
    'jane@example.com', 'bob@example.com', 'bobby@example.com',
    'alice@example.com', 'alicia@example.com', 'tom@example.com',
    'thomas@example.com', 'john@example.com', 'jon@example.com',
    'jane@example.com', 'janet@example.com', 'bob@example.com',
    'robert@example.com', 'alice@example.com', 'alison@example.com',
    'tom@example.com', 'tomas@example.com'
  )
)
con <- DBI::dbConnect(duckdb::duckdb())
spec <- il_spec() |>
  il_compare(first_name, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(surname, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(dob, cl_exact()) |>
  il_block_on(surname) |>
  il_block_on(first_name)
model <- il_model(df, spec = spec, con = con)
model <- il_estimate_u(model)
model <- il_estimate_em(model, block_on(surname))
labels <- data.frame(
  unique_id_l = c(1L, 1L),
  unique_id_r = c(11L, 2L),
  is_match = c(1L, 0L)
)

il_errors(model, labels = labels, threshold = 0.85)
DBI::dbDisconnect(con, shutdown = TRUE)
```

`il_estimate_em` *Train Parameters via Expectation-Maximization*

Description

Runs the EM algorithm under a blocking rule to learn m and u parameters from unlabeled data. Multiple calls with different blocking rules can be chained to train on complementary subsets of record pairs. Each call updates the model cumulatively.

Usage

```
il_estimate_em(
  model,
  blocking,
  convergence = 1e-05,
  fix_u = TRUE,
  fix_m = FALSE,
  max_iterations = 100L,
  fix_prior = FALSE,
  estimate_without_tf = TRUE,
  derive_prior = FALSE,
  estimator_mode = c("independent", "dependency-aware"),
  ...
)
```

Arguments

`model`	An `il_model` object (piped in).
`blocking`	A blocking rule created by `block_on()`.
`convergence`	A numeric convergence tolerance. The EM loop stops when the largest change in any updated parameter is below this value. Defaults to `1e-5`.
`fix_u`	Logical. If `TRUE` (the default), hold u parameters fixed during EM, so only m is updated. Set to `FALSE` to also estimate u. Only supported with `estimator_mode = "independent"`.
`fix_m`	Logical. If `TRUE`, hold m parameters fixed during EM. Defaults to `FALSE`. At least one of `fix_u` and `fix_m` must be `FALSE`, otherwise the algorithm cannot learn anything. Only supported with `estimator_mode = "independent"`.
`max_iterations`	Maximum number of EM iterations. Defaults to `100L`. The loop stops early when convergence is reached.
`fix_prior`	Logical. If `TRUE`, hold the prior (probability that two random records match) fixed during EM iterations. Defaults to `FALSE`.
`estimate_without_tf`	Logical. If `TRUE` (the default), EM runs on aggregated gamma-pattern counts (fast, but ignores per-pair term frequency variation). If `FALSE`, EM runs on individual pairs and incorporates per-pair TF adjustments in the E-step. Only matters when at least one comparison has `term_frequency = TRUE`. Only supported with `estimator_mode = "independent"`.

`derive_prior` Logical. If `TRUE`, derive the prior from the trained parameter values after EM completes and store it in the model. Defaults to `FALSE`. Only supported with `estimator_mode = "independent"`.

`estimator_mode` Estimator to use. `"independent"` keeps the conditionally independent Fellegi-Sunter EM estimator. `"dependency-aware"` fits log-linear matched and unmatched comparison-pattern distributions.

`...` Reserved for future options.

Value

An updated `il_model` with trained m and u parameters.

Examples

```
df <- data.frame(
  unique_id = 1:20,
  first_name = c(
    'John', 'Jon', 'Jane', 'Jane', 'Bob',
    'Bobby', 'Alice', 'Alicia', 'Tom', 'Thomas',
    'John', 'Jon', 'Jane', 'Janet', 'Bob',
    'Robert', 'Alice', 'Alison', 'Tom', 'Tomas'
  ),
  surname = c(
    'Smith', 'Smith', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Brown', 'White', 'White',
    'Smith', 'Smyth', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Browne', 'White', 'White'
  ),
  dob = c(
    '1990-01-01', '1990-01-01', '1985-06-15', '1985-06-15',
    '2000-12-01', '2000-12-01', '1975-03-22', '1975-03-22',
    '1988-07-04', '1988-07-04', '1990-01-01', '1990-01-02',
    '1985-06-15', '1985-06-16', '2000-12-01', '2000-12-02',
    '1975-03-22', '1975-03-23', '1988-07-04', '1988-07-05'
  ),
  city = c(
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid',
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid'
  ),
  email = c(
    'john@example.com', 'jon@example.com', 'jane@example.com',
    'jane@example.com', 'bob@example.com', 'bobby@example.com',
    'alice@example.com', 'alicia@example.com', 'tom@example.com',
    'thomas@example.com', 'john@example.com', 'jon@example.com',
    'jane@example.com', 'janet@example.com', 'bob@example.com',
    'robert@example.com', 'alice@example.com', 'alison@example.com',
    'tom@example.com', 'tomas@example.com'
  )
)
```

```
con <- DBI::dbConnect(duckdb::duckdb())
spec <- il_spec() |>
  il_compare(first_name, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(surname, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(dob, cl_exact()) |>
  il_block_on(surname) |>
  il_block_on(first_name)
model <- il_model(df, spec = spec, con = con)
model <- il_estimate_u(model)

model <- il_estimate_em(model, block_on(surname))
DBI::dbDisconnect(con, shutdown = TRUE)
```

`il_estimate_m_from_column` *Estimate Match (m) Parameters from a Label Column*

Description

Learns the m probabilities from a ground-truth identifier column (e.g., Social Security Number) present in the input data. Records sharing the same label value are treated as true matches. This is an alternative to `il_estimate_m_from_labels()`, which requires a separate table of pairwise labels.

Usage

```
il_estimate_m_from_column(model, label_col)
```

Arguments

`model`	An `il_model` object (piped in).
`label_col`	The unquoted name of a column in the input data containing ground-truth entity identifiers.

Value

An updated `il_model` with estimated m parameters.

Examples

```
con <- DBI::dbConnect(duckdb::duckdb())
spec <- il_spec() |>
  il_compare(first_name, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(surname, cl_exact()) |>
  il_compare(dob, cl_exact()) |>
  il_block_on(surname)
model <- il_model(fake_20, spec = spec, con = con)
model <- il_estimate_u(model)

model <- il_estimate_m_from_column(model, city)
DBI::dbDisconnect(con, shutdown = TRUE)
```

`il_estimate_m_from_labels` *Estimate Match (m) Parameters from Labeled Data*

Description

Learns the m probabilities (the probability of observing each comparison level given that the records **do** match) from a set of pre-labeled record pairs. Use this instead of `il_estimate_em()` when ground-truth labels are available.

Usage

```
il_estimate_m_from_labels(model, labels)
```

Arguments

`model`	An `il_model` object (piped in).
`labels`	A data frame of labeled pairs with columns identifying the left record, right record, and a logical or integer match indicator.

Value

An updated `il_model` with estimated m parameters.

Examples

```
df <- data.frame(
  unique_id = 1:20,
  first_name = c(
    'John', 'Jon', 'Jane', 'Jane', 'Bob',
    'Bobby', 'Alice', 'Alicia', 'Tom', 'Thomas',
    'John', 'Jon', 'Jane', 'Janet', 'Bob',
    'Robert', 'Alice', 'Alison', 'Tom', 'Tomas'
  ),
  surname = c(
    'Smith', 'Smith', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Brown', 'White', 'White',
    'Smith', 'Smyth', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Browne', 'White', 'White'
  ),
  dob = c(
    '1990-01-01', '1990-01-01', '1985-06-15', '1985-06-15',
    '2000-12-01', '2000-12-01', '1975-03-22', '1975-03-22',
    '1988-07-04', '1988-07-04', '1990-01-01', '1990-01-02',
    '1985-06-15', '1985-06-16', '2000-12-01', '2000-12-02',
    '1975-03-22', '1975-03-23', '1988-07-04', '1988-07-05'
  ),
  city = c(
    'London', 'London', 'Paris', 'Paris', 'Berlin',
```

```
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid',
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid'
  ),
  email = c(
    'john@example.com', 'jon@example.com', 'jane@example.com',
    'jane@example.com', 'bob@example.com', 'bobby@example.com',
    'alice@example.com', 'alicia@example.com', 'tom@example.com',
    'thomas@example.com', 'john@example.com', 'jon@example.com',
    'jane@example.com', 'janet@example.com', 'bob@example.com',
    'robert@example.com', 'alice@example.com', 'alison@example.com',
    'tom@example.com', 'tomas@example.com'
  )
)
con <- DBI::dbConnect(duckdb::duckdb())
spec <- il_spec() |>
  il_compare(first_name, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(surname, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(dob, cl_exact()) |>
  il_block_on(surname) |>
  il_block_on(first_name)
model <- il_model(df, spec = spec, con = con)
model <- il_estimate_u(model)
labels <- data.frame(
  unique_id_l = c(1L, 1L),
  unique_id_r = c(11L, 2L),
  is_match = c(1L, 0L)
)

model <- il_estimate_m_from_labels(model, labels)
DBI::dbDisconnect(con, shutdown = TRUE)
```

`il_estimate_prior` *Estimate the Prior Match Probability*

Description

Estimates the probability that two randomly selected records from the dataset are a match, using deterministic rules and a recall assumption. This prior anchors the Fellegi-Sunter model before more detailed parameter estimation.

Usage

```
il_estimate_prior(model, ..., recall = 0.7, profile_sql = FALSE)
```

Arguments

`model`	An `il_model` object (piped in).
`...`	Blocking rules created by `block_on()` that define deterministic matching criteria.

`recall`	A numeric value between 0 and 1 representing the assumed recall of the deterministic rules. Defaults to `0.7`.
`profile_sql`	Logical. If `TRUE`, store lightweight SQL timing metadata in `model$params$sql_profile`.

Value

An updated `il_model` with the estimated prior.

Examples

```
df <- data.frame(
  unique_id = 1:20,
  first_name = c(
    'John', 'Jon', 'Jane', 'Jane', 'Bob',
    'Bobby', 'Alice', 'Alicia', 'Tom', 'Thomas',
    'John', 'Jon', 'Jane', 'Janet', 'Bob',
    'Robert', 'Alice', 'Alison', 'Tom', 'Tomas'
  ),
  surname = c(
    'Smith', 'Smith', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Brown', 'White', 'White',
    'Smith', 'Smyth', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Browne', 'White', 'White'
  ),
  dob = c(
    '1990-01-01', '1990-01-01', '1985-06-15', '1985-06-15',
    '2000-12-01', '2000-12-01', '1975-03-22', '1975-03-22',
    '1988-07-04', '1988-07-04', '1990-01-01', '1990-01-02',
    '1985-06-15', '1985-06-16', '2000-12-01', '2000-12-02',
    '1975-03-22', '1975-03-23', '1988-07-04', '1988-07-05'
  ),
  city = c(
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid',
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid'
  ),
  email = c(
    'john@example.com', 'jon@example.com', 'jane@example.com',
    'jane@example.com', 'bob@example.com', 'bobby@example.com',
    'alice@example.com', 'alicia@example.com', 'tom@example.com',
    'thomas@example.com', 'john@example.com', 'jon@example.com',
    'jane@example.com', 'janet@example.com', 'bob@example.com',
    'robert@example.com', 'alice@example.com', 'alison@example.com',
    'tom@example.com', 'tomas@example.com'
  )
)
con <- DBI::dbConnect(duckdb::duckdb())
spec <- il_spec() |>
  il_compare(first_name, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(surname, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(dob, cl_exact()) |>
```

```
  il_block_on(surname) |>
  il_block_on(first_name)
model <- il_model(df, spec = spec, con = con)

model <- il_estimate_prior(model, block_on(first_name, surname, dob))
DBI::dbDisconnect(con, shutdown = TRUE)
```

`il_estimate_u` *Estimate Non-Match (u) Parameters*

Description

Estimates the u probabilities (the probability of observing each comparison level given that the records do **not** match) by randomly sampling record pairs. Most random pairs are non-matches, so the observed level frequencies approximate the u distribution.

Usage

```
il_estimate_u(
  model,
  max_pairs = 1e+06,
  min_count_per_level = NULL,
  chunk_size = NULL,
  profile_sql = FALSE
)
```

Arguments

`model`	An `il_model` object (piped in).
`max_pairs`	Maximum number of random pairs to sample. Defaults to `1e6`.
`min_count_per_level`	Optional integer. When set, chunked estimation stops once every comparison level has been observed at least this many times, or once `max_pairs` has been sampled.
`chunk_size`	Optional integer number of pairs to score per chunk. When set, u estimation accumulates gamma counts across chunks instead of using one aggregate query.
`profile_sql`	Logical. If `TRUE`, store lightweight SQL timing metadata in `model$params$sql_profile`.

Value

An updated `il_model` with estimated u parameters.

Examples

```
df <- data.frame(
  unique_id = 1:20,
  first_name = c(
    'John', 'Jon', 'Jane', 'Jane', 'Bob',
    'Bobby', 'Alice', 'Alicia', 'Tom', 'Thomas',
    'John', 'Jon', 'Jane', 'Janet', 'Bob',
    'Robert', 'Alice', 'Alison', 'Tom', 'Tomas'
  ),
  surname = c(
    'Smith', 'Smith', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Brown', 'White', 'White',
    'Smith', 'Smyth', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Browne', 'White', 'White'
  ),
  dob = c(
    '1990-01-01', '1990-01-01', '1985-06-15', '1985-06-15',
    '2000-12-01', '2000-12-01', '1975-03-22', '1975-03-22',
    '1988-07-04', '1988-07-04', '1990-01-01', '1990-01-02',
    '1985-06-15', '1985-06-16', '2000-12-01', '2000-12-02',
    '1975-03-22', '1975-03-23', '1988-07-04', '1988-07-05'
  ),
  city = c(
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid',
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid'
  ),
  email = c(
    'john@example.com', 'jon@example.com', 'jane@example.com',
    'jane@example.com', 'bob@example.com', 'bobby@example.com',
    'alice@example.com', 'alicia@example.com', 'tom@example.com',
    'thomas@example.com', 'john@example.com', 'jon@example.com',
    'jane@example.com', 'janet@example.com', 'bob@example.com',
    'robert@example.com', 'alice@example.com', 'alison@example.com',
    'tom@example.com', 'tomas@example.com'
  )
)
con <- DBI::dbConnect(duckdb::duckdb())
spec <- il_spec() |>
  il_compare(first_name, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(surname, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(dob, cl_exact()) |>
  il_block_on(surname) |>
  il_block_on(first_name)
model <- il_model(df, spec = spec, con = con)

model <- il_estimate_u(model)
DBI::dbDisconnect(con, shutdown = TRUE)
```

`il_find_blocking_below` *Find Blocking Rules Below a Pair-Count Threshold*

Description

Searches for single-column (and optionally two-column) blocking rules that keep the total number of candidate pairs below a given ceiling.

Usage

```
il_find_blocking_below(
  .data,
  max_pairs,
  columns = NULL,
  con = NULL,
  link_type = c("dedupe", "link"),
  max_depth = 2L
)
```

Arguments

`.data`	A data frame, dbplyr::tbl_lazy, or character table name.
`max_pairs`	Maximum number of pairs allowed.
`columns`	Character vector of column names. `NULL` for all.
`con`	A DBI connection object from `DBI::dbConnect()`.
`link_type`	One of `"dedupe"` (default) or `"link"`.
`max_depth`	Maximum depth of column combinations (default `2`).

Value

A `tibble::tibble()` of qualifying blocking rules, sorted by `n_pairs` ascending. Empty tibble if no rules qualify.

Examples

```
con <- DBI::dbConnect(duckdb::duckdb())
il_find_blocking_below(fake_1000, max_pairs = 100000, con = con)
DBI::dbDisconnect(con, shutdown = TRUE)
```

`il_find_matches` *Find Matches for New Records*

Description

Scores new records against the data already loaded into a trained model. Useful for real-time or incremental matching where new records arrive after the model has been trained.

Usage

```
il_find_matches(model, new_records, threshold = 0.85)
```

Arguments

`model`	A trained `il_model` object.
`new_records`	A data frame, dbplyr::tbl_lazy, or character table name of new records to match against the model's existing data.
`threshold`	A numeric value between 0 and 1. Only matches at or above this probability are returned. Defaults to `0.85`.

Value

An `il_compared` tibble of scored pairs between new records and existing data.

Examples

```
df <- data.frame(
  unique_id = 1:20,
  first_name = c(
    'John', 'Jon', 'Jane', 'Jane', 'Bob',
    'Bobby', 'Alice', 'Alicia', 'Tom', 'Thomas',
    'John', 'Jon', 'Jane', 'Janet', 'Bob',
    'Robert', 'Alice', 'Alison', 'Tom', 'Tomas'
  ),
  surname = c(
    'Smith', 'Smith', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Brown', 'White', 'White',
    'Smith', 'Smyth', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Browne', 'White', 'White'
  ),
  dob = c(
    '1990-01-01', '1990-01-01', '1985-06-15', '1985-06-15',
    '2000-12-01', '2000-12-01', '1975-03-22', '1975-03-22',
    '1988-07-04', '1988-07-04', '1990-01-01', '1990-01-02',
    '1985-06-15', '1985-06-16', '2000-12-01', '2000-12-02',
    '1975-03-22', '1975-03-23', '1988-07-04', '1988-07-05'
  ),
  city = c(
    'London', 'London', 'Paris', 'Paris', 'Berlin',
```

```
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid',
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid'
  ),
  email = c(
    'john@example.com', 'jon@example.com', 'jane@example.com',
    'jane@example.com', 'bob@example.com', 'bobby@example.com',
    'alice@example.com', 'alicia@example.com', 'tom@example.com',
    'thomas@example.com', 'john@example.com', 'jon@example.com',
    'jane@example.com', 'janet@example.com', 'bob@example.com',
    'robert@example.com', 'alice@example.com', 'alison@example.com',
    'tom@example.com', 'tomas@example.com'
  )
)
con <- DBI::dbConnect(duckdb::duckdb())
spec <- il_spec() |>
  il_compare(first_name, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(surname, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(dob, cl_exact()) |>
  il_block_on(surname) |>
  il_block_on(first_name)
model <- il_model(df, spec = spec, con = con)
model <- il_estimate_u(model)
model <- il_estimate_em(model, block_on(surname))
new_df <- data.frame(
  first_name = 'Jhon', surname = 'Smith',
  dob = '1990-01-15', city = 'London'
)

il_find_matches(model, new_df, threshold = 0.5)
DBI::dbDisconnect(con, shutdown = TRUE)
```

`il_graph_metrics` *Compute Graph Metrics for Clusters*

Description

Returns node-, edge-, and cluster-level metrics from the linkage graph. Useful for diagnosing cluster quality and identifying bridge edges or weakly connected components.

Usage

```
il_graph_metrics(pairs, clusters)
```

Arguments

`pairs`	An `il_compared` tibble from `predict.il_model()`.
`clusters`	A tibble from `il_cluster()` with `unique_id` and `cluster_id` columns.

Value

A named list of three tibbles:

`nodes` Record-level metrics (degree, centrality).

`edges` Edge-level metrics (match probability, bridge flag).

`clusters` Cluster-level metrics (size, density).

Examples

```
df <- data.frame(
  unique_id = 1:20,
  first_name = c(
    'John', 'Jon', 'Jane', 'Jane', 'Bob',
    'Bobby', 'Alice', 'Alicia', 'Tom', 'Thomas',
    'John', 'Jon', 'Jane', 'Janet', 'Bob',
    'Robert', 'Alice', 'Alison', 'Tom', 'Tomas'
  ),
  surname = c(
    'Smith', 'Smith', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Brown', 'White', 'White',
    'Smith', 'Smyth', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Browne', 'White', 'White'
  ),
  dob = c(
    '1990-01-01', '1990-01-01', '1985-06-15', '1985-06-15',
    '2000-12-01', '2000-12-01', '1975-03-22', '1975-03-22',
    '1988-07-04', '1988-07-04', '1990-01-01', '1990-01-02',
    '1985-06-15', '1985-06-16', '2000-12-01', '2000-12-02',
    '1975-03-22', '1975-03-23', '1988-07-04', '1988-07-05'
  ),
  city = c(
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid',
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid'
  ),
  email = c(
    'john@example.com', 'jon@example.com', 'jane@example.com',
    'jane@example.com', 'bob@example.com', 'bobby@example.com',
    'alice@example.com', 'alicia@example.com', 'tom@example.com',
    'thomas@example.com', 'john@example.com', 'jon@example.com',
    'jane@example.com', 'janet@example.com', 'bob@example.com',
    'robert@example.com', 'alice@example.com', 'alison@example.com',
    'tom@example.com', 'tomas@example.com'
  )
)
con <- DBI::dbConnect(duckdb::duckdb())
spec <- il_spec() |>
  il_compare(first_name, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(surname, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(dob, cl_exact()) |>
  il_block_on(surname) |>
```

```
  il_block_on(first_name)
model <- il_model(df, spec = spec, con = con)
model <- il_estimate_u(model)
model <- il_estimate_em(model, block_on(surname))
pairs <- predict(model, threshold = 0.5)
clusters <- il_cluster(pairs)

metrics <- il_graph_metrics(pairs, clusters)
metrics$clusters
DBI::dbDisconnect(con, shutdown = TRUE)
```

`il_largest_blocks` *Identify the Largest Blocking Bins*

Description

For a given blocking rule, returns the `n` blocking-key combinations that produce the most record pairs. This helps diagnose skew, where a single dominant key can create a quadratic explosion of pairs.

Usage

```
il_largest_blocks(
  .data,
  rule,
  n = 5L,
  con = NULL,
  link_type = c("dedupe", "link")
)
```

Arguments

`.data`	A data frame, dbplyr::tbl_lazy, or character table name (first or only dataset).
`rule`	A blocking rule created by `block_on()`.
`n`	Integer. Number of largest bins to return. Defaults to `5`.
`con`	A DBI connection object from `DBI::dbConnect()`. Optional when `.data` is a dbplyr::tbl_lazy.
`link_type`	One of `"dedupe"` (default) or `"link"`.

Value

A `tibble::tibble()` with one row per blocking-key combination, sorted by descending pair count. Columns are the blocking-key values plus `n_records` and `n_pairs`.

Examples

```
df <- data.frame(
  unique_id = 1:20,
  first_name = c(
    'John', 'Jon', 'Jane', 'Jane', 'Bob',
    'Bobby', 'Alice', 'Alicia', 'Tom', 'Thomas',
    'John', 'Jon', 'Jane', 'Janet', 'Bob',
    'Robert', 'Alice', 'Alison', 'Tom', 'Tomas'
  ),
  surname = c(
    'Smith', 'Smith', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Brown', 'White', 'White',
    'Smith', 'Smyth', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Browne', 'White', 'White'
  ),
  city = c(
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid',
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid'
  )
)
con <- DBI::dbConnect(duckdb::duckdb())
il_largest_blocks(df, block_on(city), n = 3, con = con)
DBI::dbDisconnect(con, shutdown = TRUE)
```

---

il_load *Load a Saved Model*

Description

Reads a saved `il_model` object from `.json` or `.rds`.

Usage

```
il_load(path)
```

Arguments

`path`	A file path (character string) to a saved model.

Details

Settings JSON is loaded into an `il_model` that can be used with `il_attach()` and `predict()`. The database connection and any in-database tables are not loaded.

JSON imports reconstruct equivalent SQL-backed comparison and blocking behavior for use after `il_attach()`. They do not necessarily recreate the original irelink helper objects or transform functions stored in the original spec.

Value

An `il_model` object.

Examples

```
df <- data.frame(
  unique_id = 1:20,
  first_name = c(
    'John', 'Jon', 'Jane', 'Jane', 'Bob',
    'Bobby', 'Alice', 'Alicia', 'Tom', 'Thomas',
    'John', 'Jon', 'Jane', 'Janet', 'Bob',
    'Robert', 'Alice', 'Alison', 'Tom', 'Tomas'
  ),
  surname = c(
    'Smith', 'Smith', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Brown', 'White', 'White',
    'Smith', 'Smyth', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Browne', 'White', 'White'
  ),
  dob = c(
    '1990-01-01', '1990-01-01', '1985-06-15', '1985-06-15',
    '2000-12-01', '2000-12-01', '1975-03-22', '1975-03-22',
    '1988-07-04', '1988-07-04', '1990-01-01', '1990-01-02',
    '1985-06-15', '1985-06-16', '2000-12-01', '2000-12-02',
    '1975-03-22', '1975-03-23', '1988-07-04', '1988-07-05'
  )
)
con <- DBI::dbConnect(duckdb::duckdb())
spec <- il_spec() |>
  il_compare(first_name, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(surname, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(dob, cl_exact()) |>
  il_block_on(surname) |>
  il_block_on(first_name)
model <- il_model(df, spec = spec, con = con)
model <- il_estimate_u(model)
model <- il_estimate_em(model, block_on(surname))
tmp <- tempfile(fileext = '.rds')
il_save(model, tmp)

loaded <- il_load(tmp)
DBI::dbDisconnect(con, shutdown = TRUE)
```

`il_model` *Create a Linkage Model*

Description

Binds one or more datasets to a specification and a database connection, producing an untrained model. Accepts in-memory data frames, dbplyr::tbl_lazy table references, or character table names for data that already lives in a database.

Usage

```
il_model(
  .data,
  ...,
  spec,
  con = NULL,
  link_type = c("dedupe", "link", "link_and_dedupe")
)
```

Arguments

`.data`	A data frame, `tibble::tibble()`, dbplyr::tbl_lazy, or character table name. The first (or only) input dataset. If no `unique_id` column is present, one is generated automatically.
`...`	Additional datasets for multi-table linkage (same types as `.data`).
`spec`	An `il_spec` object built with `il_spec()`, `il_compare()`, and `il_block_on()`.
`con`	A DBI connection object from `DBI::dbConnect()` (e.g., from `DBI::dbConnect(duckdb::duckdb(` Optional when `.data` is a dbplyr::tbl_lazy, the connection is extracted from the table reference.
`link_type`	One of `"dedupe"` (default), `"link"`, or `"link_and_dedupe"`.

Details

When `.data` is a dbplyr::tbl_lazy (from `dplyr::tbl()`), the connection is extracted automatically and data stays in-database with zero copying. A `unique_id` column is injected automatically if not already present.

Value

An untrained `il_model` object, ready for training verbs.

Examples

```
con <- DBI::dbConnect(duckdb::duckdb())
spec <- il_spec() |>
  il_compare(first_name, cl_jaro_winkler(0.9, 0.7)) |>
  il_block_on(surname)

model <- il_model(fake_20, spec = spec, con = con)

# Database-backed: pass a dbplyr reference directly
DBI::dbWriteTable(con, 'my_data', fake_20, overwrite = TRUE)
tbl_ref <- dplyr::tbl(con, 'my_data')
model2 <- il_model(tbl_ref, spec = spec)
DBI::dbDisconnect(con, shutdown = TRUE)
```

il_nullif *Replace a Value with NA Column Transform*

Description

Returns a transform that replaces a specific value with `NA`. Commonly used to convert empty strings to `NA` before comparison so that missing-data levels are triggered correctly. The result can be passed as the `transform` argument to `il_compare()` or `il_block_on()`, and composed with other transforms via `il_transform()`. On DuckDB and PostgreSQL, maps to SQL `NULLIF`.

Usage

```
il_nullif(value)
```

Arguments

`value`	The value to treat as missing.

Value

An `il_column_transform` closure.

Examples

```
tf <- il_nullif('')
tf(c('New York', '', 'Chicago'))

# Use before comparison to treat blank city as missing
spec <- il_spec() |>
  il_compare(city, cl_exact(), transform = il_nullif(''))
```

il_parameters *Extract Model Parameters*

Description

Returns a tidy tibble of m and u probabilities for every comparison level in the model. Designed for use with `ggplot2::geom_point()`.

Usage

```
il_parameters(model)
```

Arguments

`model`	A trained `il_model` object.

Value

For independent models, a tibble with columns `comparison`, `gamma_level`, `m`, and `u`. For dependency-aware models, the fitted training-pattern table used for scoring.

Examples

```
df <- data.frame(
  unique_id = 1:20,
  first_name = c(
    'John', 'Jon', 'Jane', 'Jane', 'Bob',
    'Bobby', 'Alice', 'Alicia', 'Tom', 'Thomas',
    'John', 'Jon', 'Jane', 'Janet', 'Bob',
    'Robert', 'Alice', 'Alison', 'Tom', 'Tomas'
  ),
  surname = c(
    'Smith', 'Smith', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Brown', 'White', 'White',
    'Smith', 'Smyth', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Browne', 'White', 'White'
  ),
  dob = c(
    '1990-01-01', '1990-01-01', '1985-06-15', '1985-06-15',
    '2000-12-01', '2000-12-01', '1975-03-22', '1975-03-22',
    '1988-07-04', '1988-07-04', '1990-01-01', '1990-01-02',
    '1985-06-15', '1985-06-16', '2000-12-01', '2000-12-02',
    '1975-03-22', '1975-03-23', '1988-07-04', '1988-07-05'
  ),
  city = c(
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid',
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid'
  ),
  email = c(
    'john@example.com', 'jon@example.com', 'jane@example.com',
    'jane@example.com', 'bob@example.com', 'bobby@example.com',
    'alice@example.com', 'alicia@example.com', 'tom@example.com',
    'thomas@example.com', 'john@example.com', 'jon@example.com',
    'jane@example.com', 'janet@example.com', 'bob@example.com',
    'robert@example.com', 'alice@example.com', 'alison@example.com',
    'tom@example.com', 'tomas@example.com'
  )
)
con <- DBI::dbConnect(duckdb::duckdb())
spec <- il_spec() |>
  il_compare(first_name, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(surname, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(dob, cl_exact()) |>
  il_block_on(surname) |>
  il_block_on(first_name)
model <- il_model(df, spec = spec, con = con)
model <- il_estimate_u(model)
```

```
model <- il_estimate_em(model, block_on(surname))

il_parameters(model)
DBI::dbDisconnect(con, shutdown = TRUE)
```

---

il_phonetic_chart *Phonetic Match Chart*

Description

Visualizes phonetic coding agreement between two columns. Shows how Soundex groupings match across pairs.

Usage

```
il_phonetic_chart(.data, col_1, col_2, con = NULL)
```

Arguments

`.data`	A data frame or character table name.
`col_1`, `col_2`	Column names (unquoted or character).
`con`	A DBI connection from `DBI::dbConnect()`. If provided and `duckdb::duckdb()` or PostgreSQL, computes Soundex in SQL.

Value

A `ggplot2::ggplot()` object.

---

il_precision_recall *Compute Precision-Recall Curve Data*

Description

Returns a tidy tibble of precision and recall values at each match-probability threshold. Requires labeled pairs. Designed for use with `ggplot2::geom_line()`.

Usage

```
il_precision_recall(model, labels = NULL, labels_col = NULL)
```

Arguments

`model`	A trained `il_model` object.
`labels`	A data frame of labeled pairs with a logical or integer match indicator. Required unless `labels_col` is provided.
`labels_col`	Optional string naming a column in the original data containing ground-truth cluster/entity IDs.

Value

A tibble::tibble() with columns threshold, precision, and recall.

Examples

```
df <- data.frame(
  unique_id = 1:20,
  first_name = c(
    'John', 'Jon', 'Jane', 'Jane', 'Bob',
    'Bobby', 'Alice', 'Alicia', 'Tom', 'Thomas',
    'John', 'Jon', 'Jane', 'Janet', 'Bob',
    'Robert', 'Alice', 'Alison', 'Tom', 'Tomas'
  ),
  surname = c(
    'Smith', 'Smith', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Brown', 'White', 'White',
    'Smith', 'Smyth', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Browne', 'White', 'White'
  ),
  dob = c(
    '1990-01-01', '1990-01-01', '1985-06-15', '1985-06-15',
    '2000-12-01', '2000-12-01', '1975-03-22', '1975-03-22',
    '1988-07-04', '1988-07-04', '1990-01-01', '1990-01-02',
    '1985-06-15', '1985-06-16', '2000-12-01', '2000-12-02',
    '1975-03-22', '1975-03-23', '1988-07-04', '1988-07-05'
  ),
  city = c(
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid',
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid'
  ),
  email = c(
    'john@example.com', 'jon@example.com', 'jane@example.com',
    'jane@example.com', 'bob@example.com', 'bobby@example.com',
    'alice@example.com', 'alicia@example.com', 'tom@example.com',
    'thomas@example.com', 'john@example.com', 'jon@example.com',
    'jane@example.com', 'janet@example.com', 'bob@example.com',
    'robert@example.com', 'alice@example.com', 'alison@example.com',
    'tom@example.com', 'tomas@example.com'
  )
)
con <- DBI::dbConnect(duckdb::duckdb())
spec <- il_spec() |>
  il_compare(first_name, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(surname, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(dob, cl_exact()) |>
  il_block_on(surname) |>
  il_block_on(first_name)
model <- il_model(df, spec = spec, con = con)
model <- il_estimate_u(model)
model <- il_estimate_em(model, block_on(surname))
```

```
labels <- data.frame(
  unique_id_l = c(1L, 1L),
  unique_id_r = c(11L, 2L),
  is_match = c(1L, 0L)
)

il_precision_recall(model, labels = labels)
DBI::dbDisconnect(con, shutdown = TRUE)
```

`il_prior_m` *Add a Matched-Class Comparison Prior*

Description

Adds a Dirichlet regularizing prior for one comparison's matched-class `m` probabilities. Use `exact` for the strongest agreement level, or `levels` for a complete named gamma-level distribution.

Usage

```
il_prior_m(
  model,
  col,
  exact = NULL,
  levels = NULL,
  strength,
  remainder = c("current")
)
```

Arguments

`model`	An `il_model` object.
`col`	Comparison column, supplied as a bare name or string.
`exact`	Probability for the strongest gamma level.
`levels`	Complete named probability vector, with names such as `"0"`, `"1"`, and `"2"`.
`strength`	Non-negative effective sample size.
`remainder`	How to distribute non-exact probability. Currently `"current"` uses trained/current `m` probabilities when available and otherwise falls back to uniform.

Value

The model with prior metadata stored in `model$params$priors`.

il_prior_prevalence *Add a Prevalence Prior*

Description

Sets a target for the global match prevalence. With `strength = NULL`, the target is used only as the model's starting prior. With a finite strength, EM also uses the target as Beta pseudo-counts.

Usage

```
il_prior_prevalence(model, probability, strength = NULL)
```

Arguments

`model`	An `il_model` object.
`probability`	Target match probability, strictly between 0 and 1.
`strength`	Optional non-negative effective sample size.

Value

The model with prior metadata stored in `model$params$priors`.

il_priors *Inspect Model Priors*

Description

Inspect Model Priors

Usage

```
il_priors(model)
```

Arguments

`model`	An `il_model` object.

Value

A `tibble::tibble()` of stored prior metadata.

`il_profile` *Profile Column Value Distributions*

Description

Computes summary statistics and value-frequency distributions for selected columns of a dataset. Useful for understanding data quality before defining comparison rules. Accepts data frames, dbplyr::tbl_lazy table references, or character table names.

Usage

```
il_profile(.data, ..., con = NULL, top_n = NULL, bottom_n = NULL)
```

Arguments

`.data`	A data frame, dbplyr::tbl_lazy, or character table name.
`...`	Columns to profile, specified as unquoted names or as character strings containing raw SQL expressions (e.g., `"city || ' ' || first_name"`). If empty, all columns are profiled.
`con`	A DBI connection object from `DBI::dbConnect()`. Optional when `.data` is a dbplyr::tbl_lazy.
`top_n`	Integer. Number of most-frequent values to return per column. Defaults to `NULL` (return all values).
`bottom_n`	Integer. Number of least-frequent values to return per column. Defaults to `NULL` (return all values).

Value

A `tibble::tibble()` of per-column summary statistics.

Examples

```
df <- data.frame(
  unique_id = 1:20,
  first_name = c(
    'John', 'Jon', 'Jane', 'Jane', 'Bob',
    'Bobby', 'Alice', 'Alicia', 'Tom', 'Thomas',
    'John', 'Jon', 'Jane', 'Janet', 'Bob',
    'Robert', 'Alice', 'Alison', 'Tom', 'Tomas'
  ),
  surname = c(
    'Smith', 'Smith', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Brown', 'White', 'White',
    'Smith', 'Smyth', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Browne', 'White', 'White'
  ),
  dob = c(
    '1990-01-01', '1990-01-01', '1985-06-15', '1985-06-15',
```

```
    '2000-12-01', '2000-12-01', '1975-03-22', '1975-03-22',
    '1988-07-04', '1988-07-04', '1990-01-01', '1990-01-02',
    '1985-06-15', '1985-06-16', '2000-12-01', '2000-12-02',
    '1975-03-22', '1975-03-23', '1988-07-04', '1988-07-05'
  ),
  city = c(
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid',
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid'
  ),
  email = c(
    'john@example.com', 'jon@example.com', 'jane@example.com',
    'jane@example.com', 'bob@example.com', 'bobby@example.com',
    'alice@example.com', 'alicia@example.com', 'tom@example.com',
    'thomas@example.com', 'john@example.com', 'jon@example.com',
    'jane@example.com', 'janet@example.com', 'bob@example.com',
    'robert@example.com', 'alice@example.com', 'alison@example.com',
    'tom@example.com', 'tomas@example.com'
  )
)
con <- DBI::dbConnect(duckdb::duckdb())
il_profile(df, first_name, surname, con = con, top_n = 5)
DBI::dbDisconnect(con, shutdown = TRUE)
```

`il_regex_extract` *Regex Extraction Column Transform*

Description

Returns a transform that extracts a regex match from a string column. Returns `NA` when no match is found. The result can be passed as the `transform` argument to `il_compare()` or `il_block_on()`, and composed with other transforms via `il_transform()`. On DuckDB and PostgreSQL, the computation is pushed into SQL.

Usage

```
il_regex_extract(pattern, group = 0L)
```

Arguments

`pattern`	A regular expression.
`group`	Integer capture group to extract. Use `0` for the whole match, or a positive integer for a numbered capture group.

Value

An `il_column_transform` closure.

Examples

```
# Extract a 5-digit ZIP code from a freeform address string
tf <- il_regex_extract('\\d{5}')
tf(c('Apt 4, 90210', '10001-1234', 'no zip'))
```

---

il_register_tf *Register Pre-Computed Term Frequency Tables*

Description

Allows you to supply pre-computed term frequency lookup tables instead of having them computed automatically from the data. This is useful when you have production TF tables from a larger dataset or want to reuse TF values across multiple linkage runs.

Usage

```
il_register_tf(model, col, tf_data, overwrite = FALSE)
```

Arguments

`model`	An `il_model` object.
`col`	Character name of the comparison column.
`tf_data`	A data frame with columns `<col>` and `tf_<col>`.
`overwrite`	Logical. If `TRUE`, overwrite an existing TF table for this column. Defaults to `FALSE`.

Details

The supplied data must have exactly two columns: the value column (named the same as the comparison column) and the frequency column (named `tf_<col>`).

Value

The updated model, with the TF table registered in the database.

Examples

```
con <- DBI::dbConnect(duckdb::duckdb())
spec <- il_spec() |>
  il_compare(first_name, cl_exact()) |>
  il_block_on(surname)
model <- il_model(fake_20, spec = spec, con = con)
tf <- data.frame(
  first_name = c('John', 'Jane', 'Bob', 'Alice', 'Tom'),
  tf_first_name = rep(0.2, 5)
)
model <- il_register_tf(model, 'first_name', tf)
il_cleanup(model)
DBI::dbDisconnect(con, shutdown = TRUE)
```

`il_roc` *Compute ROC Curve Data*

Description

Returns a tidy tibble of false-positive rates and true-positive rates at each match-probability threshold, for plotting an ROC curve. Requires labeled pairs. Designed for use with `ggplot2::geom_line()`.

Usage

```
il_roc(model, labels = NULL, labels_col = NULL)
```

Arguments

`model`	A trained `il_model` object.
`labels`	A data frame of labeled pairs with a logical or integer match indicator. Required unless `labels_col` is provided.
`labels_col`	Optional string naming a column in the original data containing ground-truth cluster/entity IDs.

Value

A `tibble::tibble()` with columns `threshold`, `fpr`, and `tpr`.

Examples

```
df <- data.frame(
  unique_id = 1:20,
  first_name = c(
    'John', 'Jon', 'Jane', 'Jane', 'Bob',
    'Bobby', 'Alice', 'Alicia', 'Tom', 'Thomas',
    'John', 'Jon', 'Jane', 'Janet', 'Bob',
    'Robert', 'Alice', 'Alison', 'Tom', 'Tomas'
  ),
  surname = c(
    'Smith', 'Smith', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Brown', 'White', 'White',
    'Smith', 'Smyth', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Browne', 'White', 'White'
  ),
  dob = c(
    '1990-01-01', '1990-01-01', '1985-06-15', '1985-06-15',
    '2000-12-01', '2000-12-01', '1975-03-22', '1975-03-22',
    '1988-07-04', '1988-07-04', '1990-01-01', '1990-01-02',
    '1985-06-15', '1985-06-16', '2000-12-01', '2000-12-02',
    '1975-03-22', '1975-03-23', '1988-07-04', '1988-07-05'
  ),
  city = c(
```

```
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid',
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid'
  ),
  email = c(
    'john@example.com', 'jon@example.com', 'jane@example.com',
    'jane@example.com', 'bob@example.com', 'bobby@example.com',
    'alice@example.com', 'alicia@example.com', 'tom@example.com',
    'thomas@example.com', 'john@example.com', 'jon@example.com',
    'jane@example.com', 'janet@example.com', 'bob@example.com',
    'robert@example.com', 'alice@example.com', 'alison@example.com',
    'tom@example.com', 'tomas@example.com'
  )
)
con <- DBI::dbConnect(duckdb::duckdb())
spec <- il_spec() |>
  il_compare(first_name, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(surname, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(dob, cl_exact()) |>
  il_block_on(surname) |>
  il_block_on(first_name)
model <- il_model(df, spec = spec, con = con)
model <- il_estimate_u(model)
model <- il_estimate_em(model, block_on(surname))
labels <- data.frame(
  unique_id_l = c(1L, 1L),
  unique_id_r = c(11L, 2L),
  is_match = c(1L, 0L)
)

il_roc(model, labels = labels)
DBI::dbDisconnect(con, shutdown = TRUE)
```

`il_save` *Save a Model to Disk*

Description

Serializes a trained `il_model` object to `.json` or `.rds`, chosen from `path`.

Usage

```
il_save(model, path, overwrite = FALSE)
```

Arguments

`model`	A trained `il_model` object.
`path`	A file path (character string) where the model will be saved.
`overwrite`	If `TRUE`, overwrite an existing file at `path`. Defaults to `FALSE`.

Details

`.json` writes Splink-style settings JSON. Other extensions write RDS. The database connection and any in-database tables are not stored. Supply a fresh connection with `il_attach()` after loading.

JSON export preserves scoring and prediction behavior by lowering comparisons and blocking rules to SQL. It does not guarantee exact round-tripping of irelink helper structure such as transform functions or structured blocking-rule fields.

Value

`model`, invisibly.

Examples

```
df <- data.frame(
  unique_id = 1:20,
  first_name = c(
    'John', 'Jon', 'Jane', 'Jane', 'Bob',
    'Bobby', 'Alice', 'Alicia', 'Tom', 'Thomas',
    'John', 'Jon', 'Jane', 'Janet', 'Bob',
    'Robert', 'Alice', 'Alison', 'Tom', 'Tomas'
  ),
  surname = c(
    'Smith', 'Smith', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Brown', 'White', 'White',
    'Smith', 'Smyth', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Browne', 'White', 'White'
  ),
  dob = c(
    '1990-01-01', '1990-01-01', '1985-06-15', '1985-06-15',
    '2000-12-01', '2000-12-01', '1975-03-22', '1975-03-22',
    '1988-07-04', '1988-07-04', '1990-01-01', '1990-01-02',
    '1985-06-15', '1985-06-16', '2000-12-01', '2000-12-02',
    '1975-03-22', '1975-03-23', '1988-07-04', '1988-07-05'
  )
)
con <- DBI::dbConnect(duckdb::duckdb())
spec <- il_spec() |>
  il_compare(first_name, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(surname, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(dob, cl_exact()) |>
  il_block_on(surname) |>
  il_block_on(first_name)
model <- il_model(df, spec = spec, con = con)
model <- il_estimate_u(model)
model <- il_estimate_em(model, block_on(surname))
tmp <- tempfile(fileext = '.rds')

il_save(model, tmp)
DBI::dbDisconnect(con, shutdown = TRUE)
```

`il_score_missing_edges` *Score Missing Edges Within Clusters*

Description

Identifies pairs of records within the same cluster that were not already scored during prediction (e.g. because they were in different blocking groups), and scores them using the model. This can reveal low-confidence links that bridge otherwise separate sub-clusters.

Usage

```
il_score_missing_edges(model, pairs, clusters, threshold = 0)
```

Arguments

`model`	A trained `il_model` object.
`pairs`	An `il_compared` tibble from `predict.il_model()`.
`clusters`	A tibble from `il_cluster()` with columns `unique_id` and `cluster_id`.
`threshold`	Numeric match-probability threshold for returned pairs. Defaults to `0`.

Value

An `il_compared` tibble of newly scored pairs (those not already in `pairs`).

Examples

```
df <- data.frame(
  unique_id = c(1, 2, 3),
  first_name = c('John', 'John', 'Jon'),
  surname = c('Smith', 'Smyth', 'Smith')
)
con <- DBI::dbConnect(duckdb::duckdb())
spec <- il_spec() |>
  il_compare(first_name, cl_exact()) |>
  il_compare(surname, cl_exact()) |>
  il_block_on(surname)
model <- il_model(df, spec = spec, con = con)
model <- il_estimate_u(model)
model <- il_estimate_em(model, block_on(surname))
pairs <- predict(model, threshold = 0.01)
clusters <- tibble::tibble(
  unique_id = c('1', '2', '3'),
  cluster_id = 'cluster_1'
)
missing <- il_score_missing_edges(model, pairs, clusters)
il_cleanup(model)
DBI::dbDisconnect(con, shutdown = TRUE)
```

il_score_patterns *Score Comparison Patterns*

Description

Scores an aggregated or row-level comparison-pattern table using a trained model. Dependency-aware models use their fitted log-linear pattern state. independent models use the fieldwise m/u parameters.

Usage

```
il_score_patterns(model, patterns)
```

Arguments

`model`	A trained `il_model`.
`patterns`	A data frame containing either comparison columns named like the model comparisons or gamma columns named `gamma_<comparison>`.

Value

A `tibble::tibble()` containing the input columns plus `match_weight` and `total_match_weight`, and `match_probability`.

il_spec *Create an Empty Linkage Specification*

Description

Initializes a blank `il_spec` object onto which comparison layers and blocking rules are added with `il_compare()` and `il_block_on()`.

Usage

```
il_spec()
```

Value

An `il_spec` object with no comparisons or blocking rules.

Examples

```
spec <- il_spec() |>
  il_compare(first_name, cl_jaro_winkler(0.9, 0.7)) |>
  il_block_on(surname)
```

il_string_similarity *Compute String Similarity Scores*

Description

Computes multiple string-similarity metrics between two strings in a single call. No database connection is required. Useful for quick exploration of how names or other text fields compare.

Usage

```
il_string_similarity(a, b)
```

Arguments

a	A character string.
b	A character string.

Value

A single-row tibble with columns `jaro_winkler`, `jaro`, `levenshtein`, `jaccard`, and `cosine`.

Examples

```
il_string_similarity('John', 'Jon')
```

il_substr *Extract a Substring Column Transform*

Description

Returns a transform that extracts a fixed-width substring from a string column. The result can be passed as the `transform` argument to `il_compare()` or `il_block_on()`, and composed with other transforms via `il_transform()`. On DuckDB and PostgreSQL, the computation is pushed into SQL.

Usage

```
il_substr(start, length)
```

Arguments

start	Integer start position (1-indexed).
length	Integer number of characters to extract.

Value

An `il_column_transform` closure.

Examples

```
tf <- il_substr(1, 3)
tf(c('Johnson', 'Smith', 'Lee'))

# Use for blocking on the first 3 characters of a name
spec <- il_spec() |>
  il_block_on(last_name, .transform = il_substr(1, 3))
```

`il_suggest_blocking` *Suggest Blocking Rules*

Description

Enumerates single-column blocking rules and ranks them by a heuristic that balances pair reduction against field coverage. Useful for choosing initial blocking rules before training.

Usage

```
il_suggest_blocking(
  .data,
  columns = NULL,
  con = NULL,
  link_type = c("dedupe", "link"),
  max_depth = 1L
)
```

Arguments

`.data`	A data frame, dbplyr::tbl_lazy, or character table name.
`columns`	Character vector of column names to evaluate. When `NULL` (the default), all non-ID columns are tried.
`con`	A DBI connection object from `DBI::dbConnect()`. Optional when `.data` is already registered in the database.
`link_type`	One of `"dedupe"` (default) or `"link"`.
`max_depth`	Maximum number of columns to combine in a single blocking rule. Defaults to `1` (single-column rules only). Set to `2` to also evaluate two-column combinations.

Value

A `tibble::tibble()` with columns `rule`, `n_distinct`, `coverage`, `n_pairs`, `pct_of_cartesian`, and `score`, sorted by `score` descending. Higher scores indicate better blocking rules.

Examples

```
df <- data.frame(
  unique_id = 1:20,
  first_name = c(
    'John', 'Jon', 'Jane', 'Jane', 'Bob',
    'Bobby', 'Alice', 'Alicia', 'Tom', 'Thomas',
    'John', 'Jon', 'Jane', 'Janet', 'Bob',
    'Robert', 'Alice', 'Alison', 'Tom', 'Tomas'
  ),
  surname = c(
    'Smith', 'Smith', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Brown', 'White', 'White',
    'Smith', 'Smyth', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Browne', 'White', 'White'
  )
)
con <- DBI::dbConnect(duckdb::duckdb())
il_suggest_blocking(df, con = con)
DBI::dbDisconnect(con, shutdown = TRUE)
```

---

il_tf_chart *Term Frequency Adjustment Chart*

Description

Visualizes the distribution of term frequencies for a column in the model. Shows how individual values shift the match weight via the TF adjustment. Rare values boost the weight, while common values penalize it.

Usage

```
il_tf_chart(model, col, n_most_freq = 10L, n_least_freq = 5L)
```

Arguments

`model`	An `il_model` object with `term_frequency = TRUE` enabled for at least one comparison column.
`col`	A character string naming the column to plot.
`n_most_freq`	Number of most-frequent values to label. Default 10.
`n_least_freq`	Number of least-frequent values to label. Default 5.

Value

A `ggplot2::ggplot()` object.

Examples

```
con <- DBI::dbConnect(duckdb::duckdb())
spec <- il_spec() |>
  il_compare(first_name, cl_exact(term_frequency = TRUE))
model <- il_model(fake_20, spec = spec, con = con)
il_tf_chart(model, 'first_name')
il_cleanup(model)
DBI::dbDisconnect(con, shutdown = TRUE)
```

---

il_training_history *Extract EM Training History*

Description

Returns a tidy tibble of parameter estimates at each EM iteration, useful for diagnosing convergence. Designed for use with `ggplot2::geom_line()` and `ggplot2::facet_wrap()`.

Usage

```
il_training_history(model)
```

Arguments

`model`	A trained `il_model` object.

Value

A `tibble::tibble()` with columns `session`, `iteration`, `comparison`, `gamma_level`, and `value`.

Examples

```
df <- data.frame(
  unique_id = 1:20,
  first_name = c(
    'John', 'Jon', 'Jane', 'Jane', 'Bob',
    'Bobby', 'Alice', 'Alicia', 'Tom', 'Thomas',
    'John', 'Jon', 'Jane', 'Janet', 'Bob',
    'Robert', 'Alice', 'Alison', 'Tom', 'Tomas'
  ),
  surname = c(
    'Smith', 'Smith', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Brown', 'White', 'White',
    'Smith', 'Smyth', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Browne', 'White', 'White'
  ),
  dob = c(
    '1990-01-01', '1990-01-01', '1985-06-15', '1985-06-15',
    '2000-12-01', '2000-12-01', '1975-03-22', '1975-03-22',
```

```
    '1988-07-04', '1988-07-04', '1990-01-01', '1990-01-02',
    '1985-06-15', '1985-06-16', '2000-12-01', '2000-12-02',
    '1975-03-22', '1975-03-23', '1988-07-04', '1988-07-05'
  ),
  city = c(
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid',
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid'
  ),
  email = c(
    'john@example.com', 'jon@example.com', 'jane@example.com',
    'jane@example.com', 'bob@example.com', 'bobby@example.com',
    'alice@example.com', 'alicia@example.com', 'tom@example.com',
    'thomas@example.com', 'john@example.com', 'jon@example.com',
    'jane@example.com', 'janet@example.com', 'bob@example.com',
    'robert@example.com', 'alice@example.com', 'alison@example.com',
    'tom@example.com', 'tomas@example.com'
  )
)
con <- DBI::dbConnect(duckdb::duckdb())
spec <- il_spec() |>
  il_compare(first_name, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(surname, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(dob, cl_exact()) |>
  il_block_on(surname) |>
  il_block_on(first_name)
model <- il_model(df, spec = spec, con = con)
model <- il_estimate_u(model)
model <- il_estimate_em(model, block_on(surname))

il_training_history(model)
DBI::dbDisconnect(con, shutdown = TRUE)
```

`il_transform` *Compose Multiple Transforms into a Chain*

Description

Creates a single transform function that applies multiple transformations in sequence. The first function is applied first and the last function is applied last. The result is itself a function that can be passed as the `transform` argument to `il_compare()` or `il_block_on()`.

Usage

```
il_transform(...)
```

Arguments

...	Two or more functions to compose, in application order. Each must be a recognized transform (e.g. `tolower`, `toupper`, `trimws`, il_soundex, il_metaphone, il_dmetaphone).

Details

On the SQL side, the transforms are nested inside-out: `il_transform(tolower, trimws)` becomes `TRIM(LOWER(col))`.

Value

A function of class `il_transform_chain` that applies all transforms in order. The individual steps are stored in the `"transforms"` attribute.

Examples

```
# Lower-case then trim whitespace
tf <- il_transform(tolower, trimws)
tf('  Hello  ')

# Use in a specification
spec <- il_spec() |>
  il_compare(name, cl_exact(), transform = il_transform(tolower, trimws))
```

---

il_try_parse_date *Try-Parse Date Column Transform*

Description

Returns a transform that attempts to parse a string column as a date. Unlike `as.Date()`, failures return `NA`/`NULL` rather than raising an error. On DuckDB this uses `try_strptime()`, and on PostgreSQL it uses `TO_DATE()`. The result can be passed as the `transform` argument to `il_compare()` or `il_block_on()`, and composed with other transforms via `il_transform()`.

Usage

```
il_try_parse_date(format = "%Y-%m-%d")
```

Arguments

format	A `strptime`-style format string. Defaults to `"%Y-%m-%d"`.

Value

An `il_column_transform` closure.

Examples

```
tf <- il_try_parse_date()
tf(c('2020-01-15', 'not-a-date', '1985-06-30'))

# Non-ISO format
tf2 <- il_try_parse_date('%m/%d/%Y')
tf2(c('01/15/2020', 'bad'))
```

---

il_try_parse_timestamp

Try-Parse Timestamp Column Transform

Description

Returns a transform that attempts to parse a string column as a timestamp. Unlike `as.POSIXct()`, failures return `NA`/`NULL` rather than raising an error. On DuckDB this uses `try_strptime()`, and on PostgreSQL it uses `TO_TIMESTAMP()`. The result can be passed as the `transform` argument to `il_compare()` or `il_block_on()`, and composed with other transforms via `il_transform()`.

Usage

```
il_try_parse_timestamp(format = "%Y-%m-%d %H:%M:%S")
```

Arguments

`format`	A `strptime`-style format string. Defaults to `"%Y-%m-%d %H:%M:%S"`.

Value

An `il_column_transform` closure.

Examples

```
tf <- il_try_parse_timestamp()
tf(c('2020-01-15 08:30:00', 'not-a-timestamp', '1985-06-30 12:00:00'))

# Custom format
tf2 <- il_try_parse_timestamp('%m/%d/%Y %I:%M %p')
tf2(c('01/15/2020 08:30 AM', 'bad'))
```

`il_unlinkables` *Compute Unlinkable Records*

Description

Calculates the proportion of records that cannot be linked at each match-probability threshold. Returns a tidy tibble for plotting the "unlinkables curve". This helps show how restrictive each threshold is.

Usage

```
il_unlinkables(model)
```

Arguments

`model`	A trained `il_model` object.

Value

A `tibble::tibble()` with columns `threshold` and `pct_unlinkable`.

Examples

```
df <- data.frame(
  unique_id = 1:20,
  first_name = c(
    'John', 'Jon', 'Jane', 'Jane', 'Bob',
    'Bobby', 'Alice', 'Alicia', 'Tom', 'Thomas',
    'John', 'Jon', 'Jane', 'Janet', 'Bob',
    'Robert', 'Alice', 'Alison', 'Tom', 'Tomas'
  ),
  surname = c(
    'Smith', 'Smith', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Brown', 'White', 'White',
    'Smith', 'Smyth', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Browne', 'White', 'White'
  ),
  dob = c(
    '1990-01-01', '1990-01-01', '1985-06-15', '1985-06-15',
    '2000-12-01', '2000-12-01', '1975-03-22', '1975-03-22',
    '1988-07-04', '1988-07-04', '1990-01-01', '1990-01-02',
    '1985-06-15', '1985-06-16', '2000-12-01', '2000-12-02',
    '1975-03-22', '1975-03-23', '1988-07-04', '1988-07-05'
  ),
  city = c(
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid',
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid'
  ),
```

```
  email = c(
    'john@example.com', 'jon@example.com', 'jane@example.com',
    'jane@example.com', 'bob@example.com', 'bobby@example.com',
    'alice@example.com', 'alicia@example.com', 'tom@example.com',
    'thomas@example.com', 'john@example.com', 'jon@example.com',
    'jane@example.com', 'janet@example.com', 'bob@example.com',
    'robert@example.com', 'alice@example.com', 'alison@example.com',
    'tom@example.com', 'tomas@example.com'
  )
)
con <- DBI::dbConnect(duckdb::duckdb())
spec <- il_spec() |>
  il_compare(first_name, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(surname, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(dob, cl_exact()) |>
  il_block_on(surname) |>
  il_block_on(first_name)
model <- il_model(df, spec = spec, con = con)
model <- il_estimate_u(model)
model <- il_estimate_em(model, block_on(surname))

il_unlinkables(model)
DBI::dbDisconnect(con, shutdown = TRUE)
```

`il_waterfall` *Extract Waterfall Data for a Single Pair*

Description

Returns a tidy tibble showing how each comparison contributed to the total match weight for a specific record pair. Designed for use with `ggplot2::geom_col()` and `ggplot2::coord_flip()`.

Usage

```
il_waterfall(pairs, which = 1L)
```

Arguments

`pairs`	An `il_compared` tibble from `predict.il_model()`.
`which`	An integer index identifying which row (pair) to decompose. Defaults to `1L`.

Value

A `tibble::tibble()` with columns `step`, `order`, `contribution`, `direction`, `start`, and `end`. The rows include the prior odds, one row per comparison contribution, and a final total.

Examples

```
df <- data.frame(
  unique_id = 1:20,
  first_name = c(
    'John', 'Jon', 'Jane', 'Jane', 'Bob',
    'Bobby', 'Alice', 'Alicia', 'Tom', 'Thomas',
    'John', 'Jon', 'Jane', 'Janet', 'Bob',
    'Robert', 'Alice', 'Alison', 'Tom', 'Tomas'
  ),
  surname = c(
    'Smith', 'Smith', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Brown', 'White', 'White',
    'Smith', 'Smyth', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Browne', 'White', 'White'
  ),
  dob = c(
    '1990-01-01', '1990-01-01', '1985-06-15', '1985-06-15',
    '2000-12-01', '2000-12-01', '1975-03-22', '1975-03-22',
    '1988-07-04', '1988-07-04', '1990-01-01', '1990-01-02',
    '1985-06-15', '1985-06-16', '2000-12-01', '2000-12-02',
    '1975-03-22', '1975-03-23', '1988-07-04', '1988-07-05'
  ),
  city = c(
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid',
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid'
  ),
  email = c(
    'john@example.com', 'jon@example.com', 'jane@example.com',
    'jane@example.com', 'bob@example.com', 'bobby@example.com',
    'alice@example.com', 'alicia@example.com', 'tom@example.com',
    'thomas@example.com', 'john@example.com', 'jon@example.com',
    'jane@example.com', 'janet@example.com', 'bob@example.com',
    'robert@example.com', 'alice@example.com', 'alison@example.com',
    'tom@example.com', 'tomas@example.com'
  )
)
con <- DBI::dbConnect(duckdb::duckdb())
spec <- il_spec() |>
  il_compare(first_name, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(surname, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(dob, cl_exact()) |>
  il_block_on(surname) |>
  il_block_on(first_name)
model <- il_model(df, spec = spec, con = con)
model <- il_estimate_u(model)
model <- il_estimate_em(model, block_on(surname))
pairs <- predict(model, threshold = 0.5)

il_waterfall(pairs, which = 1)
DBI::dbDisconnect(con, shutdown = TRUE)
```

`il_weights` *Extract Match Weights by Comparison Level*

Description

Returns a tidy tibble of comparison levels with their m probabilities, u probabilities, and log-2 Bayes factors (match weights). Designed for use with `ggplot2::geom_col()` and `ggplot2::facet_wrap()`.

Usage

```
il_weights(model)
```

Arguments

`model`	A trained `il_model` object.

Value

A `tibble::tibble()` with columns `comparison`, `gamma_level`, `m_prob`, `u_prob`, and `weight`.

Examples

```
df <- data.frame(
  unique_id = 1:20,
  first_name = c(
    'John', 'Jon', 'Jane', 'Jane', 'Bob',
    'Bobby', 'Alice', 'Alicia', 'Tom', 'Thomas',
    'John', 'Jon', 'Jane', 'Janet', 'Bob',
    'Robert', 'Alice', 'Alison', 'Tom', 'Tomas'
  ),
  surname = c(
    'Smith', 'Smith', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Brown', 'White', 'White',
    'Smith', 'Smyth', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Browne', 'White', 'White'
  ),
  dob = c(
    '1990-01-01', '1990-01-01', '1985-06-15', '1985-06-15',
    '2000-12-01', '2000-12-01', '1975-03-22', '1975-03-22',
    '1988-07-04', '1988-07-04', '1990-01-01', '1990-01-02',
    '1985-06-15', '1985-06-16', '2000-12-01', '2000-12-02',
    '1975-03-22', '1975-03-23', '1988-07-04', '1988-07-05'
  ),
  city = c(
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid',
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid'
  ),
```

```
  email = c(
    'john@example.com', 'jon@example.com', 'jane@example.com',
    'jane@example.com', 'bob@example.com', 'bobby@example.com',
    'alice@example.com', 'alicia@example.com', 'tom@example.com',
    'thomas@example.com', 'john@example.com', 'jon@example.com',
    'jane@example.com', 'janet@example.com', 'bob@example.com',
    'robert@example.com', 'alice@example.com', 'alison@example.com',
    'tom@example.com', 'tomas@example.com'
  )
)
con <- DBI::dbConnect(duckdb::duckdb())
spec <- il_spec() |>
  il_compare(first_name, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(surname, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(dob, cl_exact()) |>
  il_block_on(surname) |>
  il_block_on(first_name)
model <- il_model(df, spec = spec, con = con)
model <- il_estimate_u(model)
model <- il_estimate_em(model, block_on(surname))

il_weights(model)
DBI::dbDisconnect(con, shutdown = TRUE)
```

`is_il_model` *Test if an Object is an irelink Model*

Description

Returns `TRUE` if `x` inherits from class `il_model`.

Usage

```
is_il_model(x)
```

Arguments

`x`	An object to test.

Value

A single logical value.

Examples

```
is_il_model(il_spec())
```

is_il_spec — *Test if an Object is an irelink Specification*

Description

Returns `TRUE` if `x` inherits from class `il_spec`.

Usage

```
is_il_spec(x)
```

Arguments

`x`	An object to test.

Value

A single logical value.

Examples

```
is_il_spec(il_spec())
```

km — *Create a Distance in Kilometres*

Description

A tagged-value constructor that marks a numeric threshold as a distance in kilometres. Use inside `cl_geo_distance()` for self-documenting thresholds.

Usage

```
km(n)
```

Arguments

`n`	A non-negative numeric value.

Value

A tagged numeric with class `il_km`.

Examples

```
il_spec() |>
  il_compare(c(lat, lon), cl_geo_distance(km(5), km(50)))
```

`labels_from_column` *Derive Pairwise Labels from a Ground-Truth Column*

Description

Given a model and a column name containing cluster or entity IDs, generates pairwise labels for all predicted pairs. Two records sharing the same value in `labels_col` are labeled as matches.

Usage

```
labels_from_column(model, labels_col, threshold = 0)
```

Arguments

`model`	A trained `il_model` object.
`labels_col`	A string naming the column in the original data that contains the ground-truth cluster or entity identifier.
`threshold`	Match-probability threshold for selecting predicted pairs. Defaults to `0` to include all candidate pairs.

Details

This is a convenience wrapper: instead of manually building a labels data frame with `unique_id_l`, `unique_id_r`, and `is_match`, you supply the column name and let irelink derive everything.

Value

A data frame with columns `unique_id_l`, `unique_id_r`, and `is_match` (integer 0/1).

Examples

```
con <- DBI::dbConnect(duckdb::duckdb())
spec <- il_spec() |>
  il_compare(first_name, cl_jaro_winkler(0.9, 0.7)) |>
  il_block_on(surname)
model <- il_model(fake_1000, spec = spec, con = con)
model <- il_estimate_u(model)
model <- il_estimate_em(model, block_on(surname))
labels_from_column(model, 'cluster')
DBI::dbDisconnect(con, shutdown = TRUE)
```

mi — *Create a Distance in Miles*

Description

A tagged-value constructor that marks a numeric threshold as a distance in miles. Converted to kilometres internally by `cl_geo_distance()`.

Usage

```
mi(n)
```

Arguments

`n`	A non-negative numeric value.

Value

A tagged numeric with class `il_mi`.

Examples

```
il_spec() |>
  il_compare(c(lat, lon), cl_geo_distance(mi(3), mi(30)))
```

minutes — *Create a Duration in Minutes*

Description

A tagged-value constructor that marks a numeric threshold as a number of minutes. Use inside `cl_time_diff()` for self-documenting thresholds.

Usage

```
minutes(n)
```

Arguments

`n`	A non-negative numeric value.

Value

A tagged numeric with class `il_minutes`.

Examples

```
il_spec() |>
  il_compare(timestamp, cl_time_diff(minutes(5), minutes(60)))
```

months *Create a Duration in Months*

Description

A tagged-value constructor that marks a numeric threshold as a number of months. Use inside `cl_date_diff()` for self-documenting thresholds.

Usage

```
months(n)
```

Arguments

`n`	A non-negative numeric value.

Value

A tagged numeric with class `il_months`.

Examples

```
il_spec() |>
  il_compare(dob, cl_date_diff(months(1), months(12)))
```

phonetic *Phonetic Transform Functions*

Description

Phonetic algorithms for blocking and comparison transforms. These functions compute phonetic encodings that group similar-sounding names together. They are useful for blocking rules that tolerate spelling variation.

Usage

```
il_soundex(x)

il_metaphone(x)

il_dmetaphone(x)
```

Arguments

`x`	A character vector to encode.

Details

When used as a `transform` argument in `il_block_on()`, `block_on()`, or `il_compare()`, the computation is pushed into SQL so data is never materialized into R.

SQL availability:

Function	DuckDB	PostgreSQL	SQLite
`il_soundex`	(macro)	(native)	comparisons only (R-side)
`il_metaphone`		(native)	
`il_dmetaphone`		(native)	

SQLite does not expose a way to register scalar R functions as SQL UDFs, so phonetic transforms cannot be used in **blocking rules** on SQLite. They continue to work in **comparisons** on SQLite via the R-side gamma computation path.

Value

A character vector of phonetic codes (same length as `x`).

Examples

```
il_soundex(c('Smith', 'Smyth'))
il_soundex(c('Robert', 'Rupert'))
```

`predict.il_model` *Score Record Pairs from a Trained Model*

Description

Generates and scores all candidate record pairs that pass the blocking rules, returning those above the match-probability threshold. This is an S3 method for `stats::predict()`.

Usage

```
## S3 method for class 'il_model'
predict(
  object,
  threshold = 0.85,
  threshold_match_weight = NULL,
  type = c("pairs", "weights"),
  collect = TRUE,
  include_fields = FALSE,
  greedy = FALSE,
  profile_sql = FALSE,
  ...
)
```

Arguments

`object`	A trained `il_model` object.
`threshold`	A numeric value between 0 and 1. Only pairs with a match probability at or above this threshold are returned. Defaults to `0.85`. Ignored when `threshold_match_weight` is set.
`threshold_match_weight`	Optional numeric value. When set, pairs are filtered on evidence-only match weight (log2 Bayes factor) instead of probability. Typical values range from about -5 to +30. Overrides `threshold`.
`type`	One of `"pairs"` (default) to return scored pairs, or `"weights"` to return match weights on a log-2 Bayes-factor scale.
`collect`	If `TRUE` (the default), scored pairs are collected into an in-memory tibble. If `FALSE`, scoring is performed entirely in-database and the result is a lightweight `il_compared_lazy` reference that `il_cluster()` can consume directly, avoiding the round-trip of collecting millions of rows into R and re-uploading them. Requires a DuckDB or PostgreSQL backend.
`include_fields`	If `TRUE`, the original column values from both records in each pair are included in the output (suffixed `_l` and `_r`). Defaults to `FALSE` for performance. When `collect = FALSE` the join is performed in-database before the table is created.
`greedy`	If `TRUE`, keep a deterministic one-to-one greedy matching for link models. Defaults to `FALSE`, returning all above-threshold candidate pairs. Greedy matching sorts pairs by descending posterior match probability, then by left and right row order.
`profile_sql`	Logical. If `TRUE`, attach lightweight SQL timing metadata to collected predictions or include it on lazy predictions.
`...`	Additional arguments passed to the generic.

Value

When `collect = TRUE`: an `il_compared` tibble with one row per candidate pair, including columns for record IDs, match weight, total match weight, match probability, and per-comparison gamma values. `match_weight` is the evidence-only log2 Bayes factor. The additive prior term is exposed separately through `total_match_weight`, whose value is `match_weight + log2(prior / (1 - prior))`. When `collect = FALSE`: an `il_compared_lazy` object referencing the scored pairs table in the database.

Examples

```
df <- data.frame(
  unique_id = 1:20,
  first_name = c(
    'John', 'Jon', 'Jane', 'Jane', 'Bob',
    'Bobby', 'Alice', 'Alicia', 'Tom', 'Thomas',
    'John', 'Jon', 'Jane', 'Janet', 'Bob',
    'Robert', 'Alice', 'Alison', 'Tom', 'Tomas'
```

```
  ),
  surname = c(
    'Smith', 'Smith', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Brown', 'White', 'White',
    'Smith', 'Smyth', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Browne', 'White', 'White'
  ),
  dob = c(
    '1990-01-01', '1990-01-01', '1985-06-15', '1985-06-15',
    '2000-12-01', '2000-12-01', '1975-03-22', '1975-03-22',
    '1988-07-04', '1988-07-04', '1990-01-01', '1990-01-02',
    '1985-06-15', '1985-06-16', '2000-12-01', '2000-12-02',
    '1975-03-22', '1975-03-23', '1988-07-04', '1988-07-05'
  ),
  city = c(
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid',
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid'
  ),
  email = c(
    'john@example.com', 'jon@example.com', 'jane@example.com',
    'jane@example.com', 'bob@example.com', 'bobby@example.com',
    'alice@example.com', 'alicia@example.com', 'tom@example.com',
    'thomas@example.com', 'john@example.com', 'jon@example.com',
    'jane@example.com', 'janet@example.com', 'bob@example.com',
    'robert@example.com', 'alice@example.com', 'alison@example.com',
    'tom@example.com', 'tomas@example.com'
  )
)
con <- DBI::dbConnect(duckdb::duckdb())
spec <- il_spec() |>
  il_compare(first_name, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(surname, cl_jaro_winkler(0.9, 0.7)) |>
  il_compare(dob, cl_exact()) |>
  il_block_on(surname) |>
  il_block_on(first_name)
model <- il_model(df, spec = spec, con = con)
model <- il_estimate_u(model)
model <- il_estimate_em(model, block_on(surname))

pairs <- predict(model, threshold = 0.5)
DBI::dbDisconnect(con, shutdown = TRUE)
```

`print.il_model` *Print an irelink Model*

Description

Displays a human-readable summary of the model's type, data, training status, comparisons, and blocking rules.

Usage

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

Arguments

x	An `il_model` object.
...	Additional arguments passed to `print()`.

Value

`x`, invisibly.

Examples

```
df <- data.frame(
  unique_id = 1:20,
  first_name = c(
    'John', 'Jon', 'Jane', 'Jane', 'Bob',
    'Bobby', 'Alice', 'Alicia', 'Tom', 'Thomas',
    'John', 'Jon', 'Jane', 'Janet', 'Bob',
    'Robert', 'Alice', 'Alison', 'Tom', 'Tomas'
  ),
  surname = c(
    'Smith', 'Smith', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Brown', 'White', 'White',
    'Smith', 'Smyth', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Browne', 'White', 'White'
  ),
  dob = c(
    '1990-01-01', '1990-01-01', '1985-06-15', '1985-06-15',
    '2000-12-01', '2000-12-01', '1975-03-22', '1975-03-22',
    '1988-07-04', '1988-07-04', '1990-01-01', '1990-01-02',
    '1985-06-15', '1985-06-16', '2000-12-01', '2000-12-02',
    '1975-03-22', '1975-03-23', '1988-07-04', '1988-07-05'
  ),
  city = c(
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid',
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid'
  ),
  email = c(
    'john@example.com', 'jon@example.com', 'jane@example.com',
    'jane@example.com', 'bob@example.com', 'bobby@example.com',
    'alice@example.com', 'alicia@example.com', 'tom@example.com',
    'thomas@example.com', 'john@example.com', 'jon@example.com',
    'jane@example.com', 'janet@example.com', 'bob@example.com',
    'robert@example.com', 'alice@example.com', 'alison@example.com',
    'tom@example.com', 'tomas@example.com'
  )
)
```

```
con <- DBI::dbConnect(duckdb::duckdb())
spec <- il_spec() |>
  il_compare(first_name, cl_jaro_winkler(0.9, 0.7)) |>
  il_block_on(surname)
model <- il_model(df, spec = spec, con = con)
print(model)
DBI::dbDisconnect(con, shutdown = TRUE)
```

`print.il_spec` *Print an irelink Specification*

Description

Displays a human-readable summary of the comparisons and blocking rules stored in an `il_spec` object.

Usage

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

Arguments

`x`	An `il_spec` object.
`...`	Additional arguments passed to `print()`.

Value

`x`, invisibly.

Examples

```
spec <- il_spec() |>
  il_compare(first_name, cl_exact())
print(spec)
```

`seconds` *Create a Duration in Seconds*

Description

A tagged-value constructor that marks a numeric threshold as a number of seconds. Use inside `cl_time_diff()` for self-documenting thresholds.

Usage

```
seconds(n)
```

Arguments

`n`	A non-negative numeric value.

Value

A tagged numeric with class `il_seconds`.

Examples

```
il_spec() |>
  il_compare(timestamp, cl_time_diff(seconds(30), seconds(300)))
```

`summary.il_model` *Summarize an irelink Model*

Description

Prints a detailed table of trained parameters including m and u probabilities, match weights, and the prior match probability for each comparison level.

Usage

```
## S3 method for class 'il_model'
summary(object, ...)
```

Arguments

`object`	An `il_model` object.
`...`	Additional arguments passed to `summary()`.

Value

A summary object, invisibly.

Examples

```
df <- data.frame(
  unique_id = 1:20,
  first_name = c(
    'John', 'Jon', 'Jane', 'Jane', 'Bob',
    'Bobby', 'Alice', 'Alicia', 'Tom', 'Thomas',
    'John', 'Jon', 'Jane', 'Janet', 'Bob',
    'Robert', 'Alice', 'Alison', 'Tom', 'Tomas'
  ),
  surname = c(
    'Smith', 'Smith', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Brown', 'White', 'White',
    'Smith', 'Smyth', 'Doe', 'Doe', 'Jones',
    'Jones', 'Brown', 'Browne', 'White', 'White'
```

```
  ),
  dob = c(
    '1990-01-01', '1990-01-01', '1985-06-15', '1985-06-15',
    '2000-12-01', '2000-12-01', '1975-03-22', '1975-03-22',
    '1988-07-04', '1988-07-04', '1990-01-01', '1990-01-02',
    '1985-06-15', '1985-06-16', '2000-12-01', '2000-12-02',
    '1975-03-22', '1975-03-23', '1988-07-04', '1988-07-05'
  ),
  city = c(
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid',
    'London', 'London', 'Paris', 'Paris', 'Berlin',
    'Berlin', 'Rome', 'Rome', 'Madrid', 'Madrid'
  ),
  email = c(
    'john@example.com', 'jon@example.com', 'jane@example.com',
    'jane@example.com', 'bob@example.com', 'bobby@example.com',
    'alice@example.com', 'alicia@example.com', 'tom@example.com',
    'thomas@example.com', 'john@example.com', 'jon@example.com',
    'jane@example.com', 'janet@example.com', 'bob@example.com',
    'robert@example.com', 'alice@example.com', 'alison@example.com',
    'tom@example.com', 'tomas@example.com'
  )
)
con <- DBI::dbConnect(duckdb::duckdb())
spec <- il_spec() |>
  il_compare(first_name, cl_jaro_winkler(0.9, 0.7)) |>
  il_block_on(surname)
model <- il_model(df, spec = spec, con = con)
summary(model)
DBI::dbDisconnect(con, shutdown = TRUE)
```

`years` *Create a Duration in Years*

Description

A tagged-value constructor that marks a numeric threshold as a number of years. Use inside `cl_date_diff()` for self-documenting thresholds.

Usage

```
years(n)
```

Arguments

`n`	A non-negative numeric value.

Value

A tagged numeric with class `il_years`.

Examples

```
il_spec() |>
  il_compare(dob, cl_date_diff(years(1)))
```

Index