Package: firmmatchr (via r-universe)

October 26, 2026

Title Robust Probabilistic Matching of Company Names

Version 0.2.0

Description A pipeline for matching messy company name strings against a clean dictionary (e.g., 'Orbis'). Implements a cascading strategy: Exact -> Fuzzy ('zoomerjoin') -> 'FTS5' ('SQLite') -> Rarity Weighted. Name normalization covers German, French, Italian and English legal forms and conventions, which suits multilingual registers such as the Swiss one. Normalization discards detail, so several dictionary entries can collapse onto one string; these groups are matched once and a crosswalk back to every original entry is retained. References: Beniamino Green (2025) <https://github.com/beniaminogreen/zoomerjoin>; <https://www.sqlite.org/fts5.html>.

License MIT + file LICENSE

URL https://github.com/swediot/firmmatchr

BugReports https://github.com/swediot/firmmatchr/issues

Encoding UTF-8

Depends R (>= 4.1.0)

Roxygen list(markdown = TRUE)

Imports data.table, stringi, stringdist, zoomerjoin, DBI, RSQLite, cli, progressr, httr, jsonlite, glue, purrr, readr, dplyr, stats

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

Config/pak/sysreqs libicu-dev libssl-dev libx11-dev libclang-dev

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

Date/Publication 2026-07-28 09:29:36 UTC

RemoteUrl https://github.com/swediot/firmmatchr

RemoteRef HEAD

RemoteSha 8ce336262fb5b5fa484762a26daa09b9106b9933

Contents

company_name_stopwords 2
dict_crosswalk 3
expand_matches 4
match_companies 5
normalize_company_name 7
validate_matches_llm 9

Index **11**

`company_name_stopwords` *Stop Words Used by the Normalizer*

Description

Returns the legal-form and noise-word tokens that `normalize_company_name()` strips for a given set of languages. Useful for auditing what the normalizer removes before trusting a match.

Usage

```
company_name_stopwords(lang = c("de", "fr", "it", "en"))
```

Arguments

`lang`	Character vector of language codes. Any of `"de"` (German), `"fr"` (French), `"it"` (Italian), `"en"` (English). Defaults to all four.

Value

A `data.frame` with columns `lang`, `type` (`"legal_form"` or `"noise"`) and `token`.

See Also

`normalize_company_name()`

Examples

```
# Everything stripped for Italian
subset(company_name_stopwords("it"), type == "legal_form")

# How many tokens are removed in total across all four languages?
nrow(company_name_stopwords())
```

dict_crosswalk *Dictionary Crosswalk of a Match Result*

Description

Because normalize_company_name() is lossy, several dictionary rows can collapse onto the same normalized string (e.g. "Meier GmbH" and "Meier AG" both become "meier"). match_companies() matches against one representative row per group and attaches this crosswalk so that every original dictionary row can still be recovered.

Usage

```
dict_crosswalk(x)
```

Arguments

x	A result returned by match_companies().

Value

A data.table with one row per original dictionary row that entered matching, and columns name_clean (the normalized string), dict_id/dict_name (the original id and string), dict_id_rep (the id reported by match_companies() for that group), n_dict_ids (group size) and is_representative.

See Also

expand_matches()

Examples

```
# Wrapped in \donttest{} because the pipeline calls the multi-threaded Rust
# internals of 'zoomerjoin'.

queries <- data.frame(
  query_id = 1:2,
  company_name = c("Meier", "BMW")
)
dictionary <- data.frame(
  orbis_id = c("D001", "D002", "D003"),
  company_name = c("Meier GmbH", "Meier AG", "BMW AG")
)

results <- match_companies(queries, dictionary)

# Both Meier rows survive in the crosswalk, even though only D001 is
# reported as the match.
dict_crosswalk(results)
```

expand_matches *Expand Matches to Every Original Dictionary Row*

Description

`match_companies()` reports one representative dictionary id per match. When normalization collapsed several dictionary rows onto the same string, this function expands the result to one row per (query row, original dictionary row) pair, so the original strings behind a collapsed group are visible.

Usage

```
expand_matches(x, crosswalk = dict_crosswalk(x))
```

Arguments

x	A result returned by `match_companies()`.
crosswalk	The dictionary crosswalk. Defaults to the one attached to `x` by `match_companies()`; supply it explicitly if `x` has been subset or round-tripped through another format.

Value

A `data.table` with columns `query_id`, `query_name`, `dict_id`, `dict_name`, `match_type`, `dict_id_rep`, `n_dict_ids` and `is_representative`. Unambiguous matches contribute exactly one row.

See Also

`dict_crosswalk()`

Examples

```
# Wrapped in \donttest{} because the pipeline calls the multi-threaded Rust
# internals of 'zoomerjoin'.

queries <- data.frame(
  query_id = 1:2,
  company_name = c("Meier", "BMW")
)
dictionary <- data.frame(
  orbis_id = c("D001", "D002", "D003"),
  company_name = c("Meier GmbH", "Meier AG", "BMW AG")
)

results <- match_companies(queries, dictionary)

# One row for BMW, two rows for the collapsed Meier group.
expand_matches(results)
```

`match_companies` *Match Company Names against a Dictionary*

Description

Runs a cascading matching pipeline: Exact -> Fuzzy (Zoomer) -> FTS5 -> Rarity. Matches found in earlier steps are removed from subsequent steps. Names are normalized with German, French, Italian and English conventions by default; see the `lang` argument.

Usage

```
match_companies(
  queries,
  dictionary,
  query_col = "company_name",
  dict_col = "company_name",
  unique_id_col = "query_id",
  dict_id_col = "orbis_id",
  threshold_jw = 0.8,
  threshold_zoomer = 0.4,
  threshold_rarity = 1,
  n_cores = 1,
  dict_norm = TRUE,
  query_norm = TRUE,
  lang = c("de", "fr", "it", "en"),
  extra_stopwords = NULL,
  on_duplicate = c("collapse", "error")
)
```

Arguments

`queries`	Data frame. Must contain columns specified in `query_col` and `unique_id_col`.
`dictionary`	Data frame. Must contain columns specified in `dict_col` and `dict_id_col`.
`query_col`	String. Column name for company names in `queries`.
`dict_col`	String. Column name for company names in `dictionary`.
`unique_id_col`	String. ID column in `queries`.
`dict_id_col`	String. ID column in `dictionary`.
`threshold_jw`	Numeric (0-1). Minimum Jaro-Winkler similarity. Default 0.8.
`threshold_zoomer`	Numeric (0-1). Jaccard threshold for blocking. Default 0.4.
`threshold_rarity`	Numeric. Minimum score for rarity matching. Default 1.0.
`n_cores`	Integer. Number of cores (reserved for future parallel implementation).

`dict_norm` Logical. Apply `normalize_company_name()` to the dictionary names? Set to `FALSE` if the dictionary is already cleaned. Default `TRUE`.

`query_norm` Logical. Apply `normalize_company_name()` to the query names? Set to `FALSE` if the queries are already cleaned. Default `TRUE`. Both sides should normally use the same setting.

`lang` Character vector of language conventions to apply when normalizing. Any of `"de"` (German), `"fr"` (French), `"it"` (Italian), `"en"` (English). Defaults to all four, which suits mixed-language registers such as the Swiss one. Narrow it for single-language data to strip fewer tokens. See `normalize_company_name()`.

`extra_stopwords` Optional character vector of additional whole-word tokens to strip from both sides, on top of the built-in lists. Country names are kept by default; pass them here to remove them.

`on_duplicate` String. What to do when normalization maps several dictionary rows onto the same string: `"collapse"` (default) groups them and keeps a crosswalk, `"error"` aborts.

Value

A `data.table` with one row per matched query row and columns `query_id`, `query_name`, `dict_id`, `dict_name`, `match_type`, `n_dict_ids` and `dict_id_all`. The dictionary crosswalk is attached as the `"dict_crosswalk"` attribute; retrieve it with `dict_crosswalk()`.

Duplicate handling

Normalization is lossy on purpose: `"Meier GmbH"` and `"Meier AG"` both become `"meier"`. Distinct dictionary rows can therefore collapse onto the same normalized string. Rather than failing, `match_companies()` groups such rows, matches against one representative per group, and keeps a full crosswalk so no original row is ever lost:

- `dict_id` in the result is the representative (first) id of the group.
- `dict_id_all` lists every id in the group, separated by `"|"`.
- `n_dict_ids` is the size of the group (`1` when there was no collapse).
- `dict_crosswalk()` returns the mapping from every original dictionary row to its representative, and `expand_matches()` expands a result table to one row per original dictionary row.

Set `on_duplicate = "error"` to restore the older, strict behaviour of aborting when duplicates are present.

Queries are handled the same way: identical normalized query names are matched once and the result is fanned back out, so duplicated query names (and even duplicated query ids) are safe.

Rows whose name is empty or `NA` after normalization are dropped with a warning, since an empty string would otherwise match every other empty string.

See Also

`dict_crosswalk()`, `expand_matches()`

Examples

```
# Wrapped in \donttest{} because the pipeline calls the multi-threaded Rust
# internals of 'zoomerjoin'.

# Create sample query data
queries <- data.frame(
  query_id = 1:3,
  company_name = c("BMW", "Siemens AG", "Deutsche Bank")
)

# Dictionary with two rows that collapse onto the same normalized name
# ("siemens"): the pipeline groups them instead of failing.
dictionary <- data.frame(
  orbis_id = c("D001", "D002", "D003", "D004"),
  company_name = c("BMW AG", "Siemens AG", "Siemens GmbH", "Commerzbank AG")
)

results <- match_companies(
  queries = queries,
  dictionary = dictionary,
  query_col = "company_name",
  dict_col = "company_name",
  unique_id_col = "query_id",
  dict_id_col = "orbis_id"
)

print(results)

# Recover every original dictionary row behind a collapsed match
print(expand_matches(results))
```

`normalize_company_name` *Normalize Company Names*

Description

Standardizes company names by lowercasing, transliterating to ASCII, removing legal forms and noise words, and collapsing punctuation. Supports German, French, Italian and English conventions, which covers the four languages of Swiss and neighbouring company registers.

Usage

```
normalize_company_name(
  x,
  lang = c("de", "fr", "it", "en"),
  extra_stopwords = NULL
)
```

Arguments

x	A character vector of company names.
lang	Character vector of language codes whose conventions should be applied. Any of "de" (German), "fr" (French), "it" (Italian), "en" (English). Defaults to all four; narrow it if your data is single-language and you want to avoid over-stripping.
extra_stopwords	Optional character vector of additional whole-word tokens to strip, applied after the built-in lists. Use it for corpus-specific filler, or to remove geography (c("deutschland", "schweiz")), which is kept by default.

Details

Processing order matters and is:

1. Lowercase, trim, and turn &, /, + into spaces.
2. German digraph expansion (a-umlaut to ae, eszett to ss, ...) when "de" is among lang. This runs *before* transliteration, which would otherwise reduce u-umlaut to u rather than ue.
3. Transliteration of remaining accents to ASCII, so that accented legal forms (Societe, Sarl, Societa as actually written) are recognised.
4. Apostrophes are deleted rather than split on, so Romance elisions and English possessives join up: L'Oreal and Loreal both give loreal, and McDonald's gives mcdonalds.
5. Dotted acronyms are joined: S.A.R.L. becomes sarl, so it is removed as a legal form rather than left behind as four stray letters.
6. Legal forms and noise words are removed, then punctuation and repeated whitespace are collapsed.

Country names are deliberately **not** removed. They identify a firm as often as they pad it: stripping them turns Credit Suisse into credit and Air France into air, which can then exact-match an unrelated company. Because a spurious exact match is trusted and never reaches the fuzzy stages or the LLM validator, the safer default is to keep them. Structural branch markers (Niederlassung, Succursale, Filiale) *are* removed. Pass extra_stopwords = c("deutschland", "schweiz") to strip geography anyway.

The result is **idempotent**: normalize_company_name() applied to its own output is a no-op, so a single pass is always enough. (Before 0.2.0 it was not, because stop words were matched against the still-accented string: Nestle Sarl (accented) became nestle sarl on the first pass and only lost sarl on the second. Code that looped until the output stabilized can now call this once.)

Normalization is deliberately lossy, so distinct firms can end up sharing a normalized name. match_companies() handles that by grouping rather than failing; see dict_crosswalk(). Names consisting only of stop words normalize to "".

Use company_name_stopwords() to inspect exactly which tokens are removed.

Value

A character vector of normalized names.

See Also

company_name_stopwords(), match_companies()

Examples

```
# German
normalize_company_name("BMW AG")
normalize_company_name("Siemens GmbH & Co. KG")
normalize_company_name("Gebr. Mueller Grosshandel AG")

# French, including an accented legal form and an elision
normalize_company_name("Societe Generale S.A.")
normalize_company_name("L'Oreal France S.A.")

# Italian
normalize_company_name("Ditta Rossi S.p.A.")

# English
normalize_company_name("The Boring Company LLC")

# Restrict to one language to strip less
normalize_company_name("Sa Casa di Roma", lang = "de")
normalize_company_name("Sa Casa di Roma", lang = c("fr", "it"))

# Country names are kept by default, because they are often identifying
normalize_company_name("Credit Suisse AG")
# ... strip them explicitly if your data needs it
normalize_company_name("Toyota Deutschland GmbH", extra_stopwords = "deutschland")
```

---

validate_matches_llm *Validate Matches using LLM (Azure OpenAI)*

Description

Sends doubtful matches (not "Perfect" or "Unmatched") to an LLM for verification. Supports resuming from interruptions via chunk files.

Usage

```
validate_matches_llm(
  data,
  query_name_col,
  dict_name_col,
  output_dir = tempdir(),
  filename_stem = "match_validation",
  batch_size = 20,
  api_key = NULL,
  endpoint = NULL,
  deployment = NULL,
```

```
  engine = c("azure", "openai", "local")
)
```

Arguments

data	Data frame. Must contain the columns specified by query_name_col and dict_name_col.
query_name_col	String. Column containing the user's query name (Employer).
dict_name_col	String. Column containing the dictionary match name (Registry).
output_dir	String. Directory to save temporary chunks and final results. Defaults to tempdir().
filename_stem	String. Base name for output files.
batch_size	Integer. Number of rows to process before saving a chunk.
api_key	String. API Key. Defaults to Sys.getenv("AZURE_API_KEY") or Sys.getenv("OPENAI_API_KEY").
endpoint	String. API Endpoint. Defaults to Sys.getenv("AZURE_ENDPOINT") or Sys.getenv("OPENAI_ENDPOIN
deployment	String. Deployment or model name. Defaults to Sys.getenv("AZURE_DEPLOYMENT") or Sys.getenv("OPENAI_MODEL").
engine	String. Either "azure", "openai", or "local". Defaults to "azure". Use "local" (or "openai") for local LLMs like Ollama.

Value

A data frame with added LLM_decision and LLM_reason columns.

Examples

```
## Not run:
# Sample matched data
matched_data <- data.frame(
  employer_name = c("BMW", "Siemens"),
  registry_name = c("BMW AG", "SAP SE"),
  dict_id = c("D001", "D002"),
  match_type = c("Fuzzy", "Fuzzy")
)

# Validate using LLM (requires Azure credentials)
validated <- validate_matches_llm(
  data = matched_data,
  query_name_col = "employer_name",
  dict_name_col = "registry_name"
)

print(validated)

## End(Not run)
```

Index

company_name_stopwords, 2
company_name_stopwords(), *8, 9*

dict_crosswalk, 3
dict_crosswalk(), *4*, *6*, *8*

expand_matches, 4
expand_matches(), *3*, *6*

match_companies, 5
match_companies(), *3, 4*, *8, 9*

normalize_company_name, 7
normalize_company_name(), *2, 3*, *6*

validate_matches_llm, 9