Package: hmatch (via r-universe)

October 24, 2024

Type Package

Title Tools for Cleaning and Matching Hierarchically-Structured Data

Version 0.1.0.9000

Description Tools for matching raw, potentially messy hierarchical data (e.g. province, county, township) against a reference dataset.

License MIT + file LICENSE

URL `https://github.com/epicentre-msf/hmatch`

BugReports `https://github.com/epicentre-msf/hmatch/issues`

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Depends R (>= 2.10)

Imports stringi, stringdist, dplyr, tidyr, rlang

Suggests testthat (>= 2.1.0), covr, sf

Repository https://epicentre-msf.r-universe.dev

RemoteUrl https://github.com/epicentre-msf/hmatch

RemoteRef HEAD

RemoteSha 1a57862a84e18711163fdc11b100ad1e60f2f491

Contents

`count_tokens` *Find frequently occurring tokens within a hierarchical column*

Description

Tokenized matching of hierarchical columns can yield false positives when there are tokens that occur frequently in multiple unique hierarchical values (e.g. "South", "North", "City", etc.).

This is a helper function to find such frequently-occurring tokens, which can then be passed to the `exclude` argument of `hmatch_tokens`. The frequency calculated is the number of unique, string-standardized values in which a given token is found.

Usage

```
count_tokens(
  x,
  split = "[-_[:space:]]+",
  min_freq = 2,
  min_nchar = 3,
  return_values = TRUE,
  std_fn = string_std,
  ...
)
```

Arguments

`x`	a character vector (generally a hierarchical column)
`split`	regex pattern used to split values into tokens. By default splits on any sequence of one or more space characters ("[:space:]"), dashes ("-"), and/or underscores ("_").

`min_freq`	minimum token frequency (i.e. number of unique values in which a given token occurs). Defaults to `2`.
`min_nchar`	minimum token size in number of characters. Defaults to `3`.
`return_values`	logical indicating whether to return the standardized values in which each token is found (`TRUE`), or only the count of the number of unique standardized values (`FALSE`). Defaults to `TRUE`.
`std_fn`	function to standardize strings, as performed within all `hmatch_` functions. Defaults to `string_std`. Set to `NULL` to omit standardization. See also string_standardization.
`...`	additional arguments passed to `std_fn()`

Examples

```
french_departments <- c(
  "Alpes-de-Haute-Provence", "Hautes-Alpes", "Ardennes", "Bouches-du-Rhône",
  "Corse-du-Sud", "Haute-Corse", "Haute-Garonne", "Ille-et-Vilaine",
  "Haute-Loire", "Hautes-Pyrénées", "Pyrénées-Atlantiques", "Hauts-de-Seine"
)

count_tokens(french_departments)
```

`dictionary_recoding` *Dictionary-based recoding of values during hierarchical matching*

Description

During hierarchical matching with the `hmatch_` group of functions, values within `raw` can be temporarily recoded to match values within `ref` based on a dictionary (argument `dict`) that maps raw values to their desired replacement values (optionally limited to a given hierarchical column).

Note that this recoding is done internally, and doesn't actually modify the values of `raw` that are returned (it just enables a match to the proper values of `ref`).

For example, if the raw data contains entries of "USA" for variable "adm0", which we know correspond to the value "United States" within the reference data, we can specify a dictionary as follows:

`dict <- data.frame(value = "USA", replacement = "United States", variable = "adm0")`

The column names in the dictionary don't actually matter, but the column order must be:

1. value in `raw` to temporarily replace
2. replacement value (to match value in `ref`)
3. (optional) name of hierarchical column in `raw` to recode

Specifying column(s) to recode

If the dictionary contains only two columns (values and replacements), then all recoding will be applied to every hierarchical column.

To apply only a portion of the dictionary to all hierarchical columns (and the rest to specified columns), a user can specify a third dictionary column with values of `<NA>` in rows where the recoding should apply to all hierarchical columns. E.g.

```
dict <- data.frame(value = c("USA", "Washingtin" replace = c("United States", "Washington"), variable =
```

For example, the dictionary above specifies that values of "USA" within column "adm0" will be temporarily replaced with "United States", while values of "Washingtin" within any hierarchical column will be replaced with "Washington".

String standardization

Note that string standardization as specifed by argument `std_fn` (see string_standardization) also applies to dictionaries. For example, given the default standardization function which includes case-standardization, a dictionary value of "USA" will match (and therefore recode) `raw` enries "USA" and "usa", but not e.g. "U.S.A.".

`hcodes` *Create codes to identify each unique combination of hierarchical levels in a reference dataset*

Description

Create codes to identify each unique combination of hierarchical levels in a reference dataset. Codes may be integer-based (function `hcodes_int`) or string-based (`hcodes_str`). Integer-based codes reflect the alphabetical ranking of each level within the next-highest level. They are constant-width and may optionally be prefixed with any given string. String-based codes are created by pasting together the values of each hierarchical level with a given separator (with options for string standardization prior to collapsing).

Usage

```
hcodes_str(ref, pattern, by, sep = "__", std_fn = string_std)

hcodes_int(ref, pattern, by, prefix = "")
```

Arguments

`ref`	`data.frame` containing hierarchical columns with reference data
`pattern`	regex pattern to match the names of the hierarchical columns in `ref` (supply either `pattern` *or* `by`)
`by`	vector giving the names of the hierarchical columns in `ref` (supply either `pattern` *or* `by`)
`sep`	(only for `hcodes_str`) desired separator between levels in string-based codes (defaults to "__")

`std_fn`	(only for `hcodes_str`) Function to standardize input strings prior to creating codes. Defaults to `string_std`. Set to `NULL` to omit standardization. See also string_standardization.
`prefix`	(only for `hcodes_int`) character prefix for integer-based codes (defaults to "")

Value

A vector of codes

Examples

```
data(ne_ref)

# string-based codes
hcodes_str(ne_ref, pattern = "^adm")

# integer-based codes
hcodes_int(ne_ref, pattern = "^adm")
```

hmatch — *Match sets of hierarchical variables between a raw and reference dataset*

Description

Match sets of hierarchical values (e.g. province, county, township) in a raw, messy dataset to corresponding values within a reference dataset, optionally accounting for discrepancies between the datasets such as:

- variation in character case, use of accents, or spelling
- variation in hierarchical resolution (e.g. some entries specified to municipality but others only to region)
- missing values at one or more hierarchical levels

Usage

```
hmatch(
  raw,
  ref,
  pattern,
  pattern_ref = pattern,
  by,
  by_ref = by,
  type = "left",
  allow_gaps = TRUE,
  fuzzy = FALSE,
  fuzzy_method = "osa",
    fuzzy_dist = 1L,
    dict = NULL,
    ref_prefix = "ref_",
    std_fn = string_std,
    ...
  )
```

Arguments

`raw`	data frame containing hierarchical columns with raw data
`ref`	data frame containing hierarchical columns with reference data
`pattern`	regex pattern to match the hierarchical columns in `raw` **Note:** hierarchical column names can be matched using either the `pattern` *or* `by` arguments. Or, if neither `pattern` or `by` are specified, the hierarchical columns are assumed to be all column names that are common to both `raw` and `ref`. See specifying_columns.
`pattern_ref`	regex pattern to match the hierarchical columns in `ref`. Defaults to `pattern`, so only need to specify if the hierarchical columns have different names in `raw` and `ref`.
`by`	vector giving the names of the hierarchical columns in `raw`
`by_ref`	vector giving the names of the hierarchical columns in `ref`. Defaults to `by`, so only need to specify if the hierarchical columns have different names in `raw` and `ref`.
`type`	type of join ("left", "inner", "anti", "resolve_left", "resolve_inner", or "resolve_anti"). Defaults to "left". See join_types.
`allow_gaps`	logical indicating whether to allow missing values below the match level, where 'match level' is the highest level with a non-missing value within a given row of `raw`. Defaults to `TRUE`.
`fuzzy`	logical indicating whether to use fuzzy-matching (based on the `stringdist` package). Defaults to FALSE.
`fuzzy_method`	if `fuzzy = TRUE`, the method to use for string distance calculation (see stringdist-metrics). Defaults to "osa".
`fuzzy_dist`	if `fuzzy = TRUE`, the maximum string distance to use to classify matches (i.e. a string distance less than or equal to `fuzzy_dist` will be considered matching). Defaults to `1L`.
`dict`	optional dictionary for recoding values within the hierarchical columns of `raw` (see dictionary_recoding)
`ref_prefix`	prefix to add to names of returned columns from `ref` if they are otherwise identical to names within `raw`. Defaults to "ref_".
`std_fn`	function to standardize strings during matching. Defaults to `string_std`. Set to `NULL` to omit standardization. See also string_standardization.
`...`	additional arguments passed to `std_fn()`

Value

a data frame obtained by matching the hierarchical columns in `raw` and `ref`, using the join type specified by argument `type` (see join_types for more details)

Resolve joins

In `hmatch`, if argument `type` corresponds to a resolve join, rows of `raw` with multiple matches to `ref` are always resolved to 'no match'. This is because `hmatch` does not accept matches below the highest non-missing level within a given row of `raw`. E.g.

```
raw:
1. | United States | <NA>         | Jefferson |
```

Relevant rows from `ref`:

```
1. | United States | New York     | Jefferson |
2. | United States | Pennsylvania | Jefferson |
```

In a regular join with `hmatch`, the single row from `raw` (above) will match both rows of `ref`. However, in a resolve join the multiple conflicting matches (i.e. conflicting values at the 2nd hierarchical level) will result in the row from `raw` being treated as non-matching to `ref`.

Examples

```
data(ne_raw)
data(ne_ref)

hmatch(ne_raw, ne_ref, pattern = "adm", type = "inner")
```

`hmatch_composite` *Implement a variety of hierarchical matching strategies in sequence*

Description

Match a data frame with raw, potentially messy hierarchical data (e.g. province, county, township) against a reference dataset, using a variety of matching strategies implemented in sequence to identify the best-possible match (i.e. highest-resolution) for each row.

The sequence of matching strategies is:

1. (optional) manually-specified matching with `hmatch_manual`
2. complete matching with `hmatch(..., allow_gaps = FALSE)`
3. partial matching with `hmatch(..., allow_gaps = TRUE)`
4. fuzzy partial matching with `hmatch(allow_gaps = TRUE, fuzzy = TRUE)`
5. best-possible matching with `hmatch_settle`

Each approach is implement only on the rows of data for which a single match has not already been identified using the previous approaches.

Usage

```
hmatch_composite(
  raw,
  ref,
  man,
  pattern,
  pattern_ref = pattern,
  by,
  by_ref = by,
  code_col,
  type = "resolve_left",
  allow_gaps = TRUE,
  fuzzy = FALSE,
  fuzzy_method = "osa",
  fuzzy_dist = 1L,
  dict = NULL,
  ref_prefix = "ref_",
  std_fn = string_std,
  ...
)
```

Arguments

`raw`	data frame containing hierarchical columns with raw data
`ref`	data frame containing hierarchical columns with reference data
`man`	(optional) data frame of manually-specified matches, relating a given set of hierarchical values to the code within `ref` to which those values correspond
`pattern`	regex pattern to match the hierarchical columns in `raw` (and `man` if given) (see also specifying_columns)
`pattern_ref`	regex pattern to match the hierarchical columns in `ref`. Defaults to `pattern`, so only need to specify if the hierarchical columns have different names in `raw` and `ref`.
`by`	vector giving the names of the hierarchical columns in `raw` (and `man` if given)
`by_ref`	vector giving the names of the hierarchical columns in `ref`. Defaults to `by`, so only need to specify if the hierarchical columns have different names in `raw` and `ref`.
`code_col`	name of the code column containing codes for matching `ref` and `man` (only required if argument `man` is given)
`type`	type of join ("resolve_left", "resolve_inner", or "resolve_anti"). Defaults to "left". See join_types.
`allow_gaps`	logical indicating whether to allow missing values below the match level, where 'match level' is the highest level with a non-missing value within a given row of `raw`. Defaults to `TRUE`.
`fuzzy`	logical indicating whether to use fuzzy-matching (based on the `stringdist` package). Defaults to FALSE.

`fuzzy_method`	if `fuzzy = TRUE`, the method to use for string distance calculation (see stringdist-metrics). Defaults to "osa".
`fuzzy_dist`	if `fuzzy = TRUE`, the maximum string distance to use to classify matches (i.e. a string distance less than or equal to `fuzzy_dist` will be considered matching). Defaults to `1L`.
`dict`	optional dictionary for recoding values within the hierarchical columns of `raw` (see dictionary_recoding)
`ref_prefix`	prefix to add to names of returned columns from `ref` if they are otherwise identical to names within `raw`. Defaults to "ref_".
`std_fn`	function to standardize strings during matching. Defaults to `string_std`. Set to `NULL` to omit standardization. See also string_standardization.
`...`	additional arguments passed to `std_fn()`

Value

a data frame obtained by matching the hierarchical columns in `raw` and `ref`, using the join type specified by argument `type` (see join_types for more details)

Examples

```
data(ne_raw)
data(ne_ref)

hmatch_composite(ne_raw, ne_ref, fuzzy = TRUE)
```

`hmatch_manual` *Manual hierarchical matching*

Description

Match a data.frame with raw, potentially messy hierarchical data (e.g. province, county, township) against a reference dataset, using a dictionary of manually-specified matches.

Usage

```
hmatch_manual(
  raw,
  ref,
  man,
  pattern,
  pattern_ref = pattern,
  by,
  by_ref = by,
  code_col,
  type = "left",
    ref_prefix = "ref_",
    std_fn = string_std,
    ...
  )
```

Arguments

`raw`	data frame containing hierarchical columns with raw data
`ref`	data frame containing hierarchical columns with reference data
`man`	`data.frame` of manually-specified matches, relating a given set of hierarchical values to the code within `ref` to which those values correspond
`pattern`	regex pattern to match the hierarchical columns in `raw` and `man` (see also specifying_columns)
`pattern_ref`	regex pattern to match the hierarchical columns in `ref`. Defaults to `pattern`, so only need to specify if the hierarchical columns have different names in `raw` and `ref`.
`by`	vector giving the names of the hierarchical columns in `raw` and `man`
`by_ref`	vector giving the names of the hierarchical columns in `ref`. Defaults to `by`, so only need to specify if the hierarchical columns have different names in `raw` and `ref`.
`code_col`	name of the code column containing codes for matching `ref` and `man`
`type`	type of join ("left", "inner", or "anti"). Defaults to "left". See join_types. Note that this function does not allow 'resolve joins', unlike most other `hmatch_` functions.
`ref_prefix`	prefix to add to names of returned columns from `ref` if they are otherwise identical to names within `raw`. Defaults to "ref_".
`std_fn`	function to standardize strings during matching. Defaults to `string_std`. Set to `NULL` to omit standardization. See also string_standardization.
`...`	additional arguments passed to `std_fn()`

Value

a data frame obtained by matching the hierarchical columns in `raw` and `ref` based on sets of matches specified in `man`, using the join type specified by argument `type` (see join_types for more details)

Examples

```
data(ne_raw)
data(ne_ref)

# create df mapping sets of raw hierarchical values to codes within ref
ne_man <- data.frame(
  adm0 = NA_character_,
  adm1 = NA_character_,
  adm2 = "Bergen, N.J.",
  hcode = "211",
  stringsAsFactors = FALSE
)

# find manual matches
hmatch_manual(ne_raw, ne_ref, ne_man, code_col = "hcode", type = "inner")
```

hmatch_parents *Hierarchical matching of parents based on sets of common offspring*

Description

Match a hierarchical column (e.g. region, province, or county) within a raw, potentially messy dataset against a corresponding column within a reference dataset, by searching for similar sets of 'offspring' (i.e. values at the next hierarchical level).

For example, if the raw dataset uses admin1 level "NY" whereas the reference dataset uses "New York", it would be difficult to automatically match these values using only fuzzy-matching. However, we might nonetheless be able to match "NY" to "New York" if they share a common and unique set of 'offspring' (i.e. admin2 values) across both datasets (e.g "Kings", "Queens", "New York", "Suffolk", "Bronx", etc.).

Unlike other hmatch functions, the data frame returned by hmatch_parents only includes *unique* hierarchical combinations and only relevant hierarchical levels (i.e. the parent level and above), along with additional columns giving the number of matching children and total number of children for a given parent.

Usage

```
hmatch_parents(
  raw,
  ref,
  pattern,
  pattern_ref = pattern,
  by,
  by_ref = by,
  level,
  min_matches = 1L,
  type = "left",
  fuzzy = FALSE,
  fuzzy_method = "osa",
  fuzzy_dist = 1L,
  ref_prefix = "ref_",
  std_fn = string_std,
  ...
)
```

Arguments

`raw`	data frame containing hierarchical columns with raw data
`ref`	data frame containing hierarchical columns with reference data
`pattern`	regex pattern to match the hierarchical columns in `raw` **Note:** hierarchical column names can be matched using either the `pattern` *or* `by` arguments. Or, if neither `pattern` or `by` are specified, the hierarchical columns are assumed to be all column names that are common to both `raw` and `ref`. See specifying_columns.
`pattern_ref`	regex pattern to match the hierarchical columns in `ref`. Defaults to `pattern`, so only need to specify if the hierarchical columns have different names in `raw` and `ref`.
`by`	vector giving the names of the hierarchical columns in `raw`
`by_ref`	vector giving the names of the hierarchical columns in `ref`. Defaults to `by`, so only need to specify if the hierarchical columns have different names in `raw` and `ref`.
`level`	name or integer index of the hierarchical level to match at (i.e. the 'parent' level). If a name, must correspond to a hierarchical column within `raw`, not including the very last hierarchical column (which has no hierarchical children). If an integer, must be between 1 and k-1, where k is the number of hierarchical columns.
`min_matches`	minimum number of matching offspring required for parents to be considered a match. Defaults to `1`.
`type`	type of join ("left", "inner" or "anti") (defaults to "left")
`fuzzy`	logical indicating whether to use fuzzy-matching (based on the `stringdist` package). Defaults to FALSE.
`fuzzy_method`	if `fuzzy = TRUE`, the method to use for string distance calculation (see stringdist-metrics). Defaults to "osa".
`fuzzy_dist`	if `fuzzy = TRUE`, the maximum string distance to use to classify matches (i.e. a string distance less than or equal to `fuzzy_dist` will be considered matching). Defaults to `1L`.
`ref_prefix`	prefix to add to names of returned columns from `ref` if they are otherwise identical to names within `raw`. Defaults to "ref_".
`std_fn`	function to standardize strings during matching. Defaults to `string_std`. Set to `NULL` to omit standardization. See also string_standardization.
`...`	additional arguments passed to `std_fn()`

Value

a data frame obtained by matching the hierarchical columns in `raw` and `ref` (at the parent level and above), using the join type specified by argument `type` (see join_types for more details). Note that unlike other `hmatch_` functions, hmatch_parents returns only unique rows and relevant hierarchical columns (i.e. the parent level and above), along with additional columns describing the number of matching children and total number of children for a given parent.

...	hierarchical columns from `raw`, parent level and above
...	hierarchical columns from `ref`, parent level and above
`n_child_raw`	total number of unique children belonging to the parent within `raw`
`n_child_ref`	total number of unique children belonging to the parent within `ref`
`n_child_match`	number of children in `raw` with match in `ref`

Examples

```
# e.g. match abbreviated adm1 names to full names based on common offspring
raw <- ne_ref
raw$adm1[raw$adm1 == "Ontario"] <- "ON"
raw$adm1[raw$adm1 == "New York"] <- "NY"
raw$adm1[raw$adm1 == "New Jersey"] <- "NJ"
raw$adm1[raw$adm1 == "Pennsylvania"] <- "PA"

hmatch_parents(
  raw,
  ne_ref,
  pattern = "adm",
  level = "adm1",
  min_matches = 2,
  type = "left"
)
```

`hmatch_permute` *Hierarchical matching with sequential column permutation to allow for values entered at the wrong hierarchical level*

Description

Match a data frame with raw, potentially messy hierarchical data (e.g. province, county, township) against a reference dataset, using sequential permutation of the hierarchical columns to allow for values entered at the wrong hierarchical level.

The function calls `hmatch` on each possible permutation of the hierarchical columns, and then combines the results. Rows of `raw` yielding multiple matches to `ref` can optionally be resolved using a resolve-type join (see section **Resolve joins** below).

Usage

```
hmatch_permute(
  raw,
  ref,
  pattern,
  pattern_ref = pattern,
  by,
  by_ref = by,
    type = "left",
    allow_gaps = TRUE,
    fuzzy = FALSE,
    fuzzy_method = "osa",
    fuzzy_dist = 1L,
    dict = NULL,
    ref_prefix = "ref_",
    std_fn = string_std,
    ...
  )
```

Arguments

`raw`	data frame containing hierarchical columns with raw data
`ref`	data frame containing hierarchical columns with reference data
`pattern`	regex pattern to match the hierarchical columns in `raw` **Note:** hierarchical column names can be matched using either the `pattern` *or* `by` arguments. Or, if neither `pattern` or `by` are specified, the hierarchical columns are assumed to be all column names that are common to both `raw` and `ref`. See specifying_columns.
`pattern_ref`	regex pattern to match the hierarchical columns in `ref`. Defaults to `pattern`, so only need to specify if the hierarchical columns have different names in `raw` and `ref`.
`by`	vector giving the names of the hierarchical columns in `raw`
`by_ref`	vector giving the names of the hierarchical columns in `ref`. Defaults to `by`, so only need to specify if the hierarchical columns have different names in `raw` and `ref`.
`type`	type of join ("left", "inner", "anti", "resolve_left", "resolve_inner", or "resolve_anti"). Defaults to "left". See join_types.
`allow_gaps`	logical indicating whether to allow missing values below the match level, where 'match level' is the highest level with a non-missing value within a given row of `raw`. Defaults to `TRUE`.
`fuzzy`	logical indicating whether to use fuzzy-matching (based on the `stringdist` package). Defaults to FALSE.
`fuzzy_method`	if `fuzzy = TRUE`, the method to use for string distance calculation (see stringdist-metrics). Defaults to "osa".
`fuzzy_dist`	if `fuzzy = TRUE`, the maximum string distance to use to classify matches (i.e. a string distance less than or equal to `fuzzy_dist` will be considered matching). Defaults to `1L`.
`dict`	optional dictionary for recoding values within the hierarchical columns of `raw` (see dictionary_recoding)
`ref_prefix`	prefix to add to names of returned columns from `ref` if they are otherwise identical to names within `raw`. Defaults to "ref_".

std_fn	function to standardize strings during matching. Defaults to string_std. Set to NULL to omit standardization. See also string_standardization.
...	additional arguments passed to std_fn()

Value

a data frame obtained by matching the hierarchical columns in raw and ref, using the join type specified by argument type (see join_types for more details)

Resolve joins

In hmatch_permute, if argument type corresponds to a resolve join, rows of raw with multiple matches to ref are resolved to the highest hierarchical level that is common among all matches (or no match if there is a conflict at the very first level). E.g.

```
raw:
1. | United States | <NA>     | New York |
```

```
Relevant rows from ref:
1. | United States | New York | <NA>     |
2. | United States | New York | New York |
```

In a regular join with hmatch_permute, the single row from raw (above) will match both of the depicted rows from ref. However, in a resolve join the two matches will resolve to the first row from ref, because it reflects the highest hierarchical level that is common to all matches.

Examples

```
data(ne_raw)
data(ne_ref)

hmatch_permute(ne_raw, ne_ref, pattern = "^adm", type = "inner")
```

hmatch_settle

Sequential hierarchical matching at each hierarchical level, settling for the highest resolution match that is possible for each row

Description

Match sets of hierarchical values (e.g. province / county / township) in a raw, messy dataset to corresponding values within a reference dataset, sequentially over each hierarchical level. Specifically, implements hmatch at each successive hierarchical level, starting with only the first level (lowest resolution), then first and second, first second and third, etc.

After the initial matching over all levels, users can optionally use a resolve join to 'settle' for the highest match possible for each row of raw data, even if that match is below the highest-resolution level specified.

Usage

```
hmatch_settle(
  raw,
  ref,
  pattern,
  pattern_ref = pattern,
  by,
  by_ref = by,
  type = "left",
  allow_gaps = TRUE,
  fuzzy = FALSE,
  fuzzy_method = "osa",
  fuzzy_dist = 1L,
  dict = NULL,
  ref_prefix = "ref_",
  std_fn = string_std,
  ...
)
```

Arguments

`raw`	data frame containing hierarchical columns with raw data
`ref`	data frame containing hierarchical columns with reference data
`pattern`	regex pattern to match the hierarchical columns in `raw` **Note:** hierarchical column names can be matched using either the `pattern` *or* `by` arguments. Or, if neither `pattern` or `by` are specified, the hierarchical columns are assumed to be all column names that are common to both `raw` and `ref`. See specifying_columns.
`pattern_ref`	regex pattern to match the hierarchical columns in `ref`. Defaults to `pattern`, so only need to specify if the hierarchical columns have different names in `raw` and `ref`.
`by`	vector giving the names of the hierarchical columns in `raw`
`by_ref`	vector giving the names of the hierarchical columns in `ref`. Defaults to `by`, so only need to specify if the hierarchical columns have different names in `raw` and `ref`.
`type`	type of join ("left", "inner", "anti", "resolve_left", "resolve_inner", or "resolve_anti"). Defaults to "left". See join_types.
`allow_gaps`	logical indicating whether to allow missing values below the match level, where 'match level' is the highest level with a non-missing value within a given row of `raw`. Defaults to `TRUE`.
`fuzzy`	logical indicating whether to use fuzzy-matching (based on the `stringdist` package). Defaults to FALSE.
`fuzzy_method`	if `fuzzy` = `TRUE`, the method to use for string distance calculation (see stringdist-metrics). Defaults to "osa".

`fuzzy_dist`	if `fuzzy = TRUE`, the maximum string distance to use to classify matches (i.e. a string distance less than or equal to `fuzzy_dist` will be considered matching). Defaults to `1L`.
`dict`	optional dictionary for recoding values within the hierarchical columns of `raw` (see dictionary_recoding)
`ref_prefix`	prefix to add to names of returned columns from `ref` if they are otherwise identical to names within `raw`. Defaults to "ref_".
`std_fn`	function to standardize strings during matching. Defaults to `string_std`. Set to `NULL` to omit standardization. See also string_standardization.
`...`	additional arguments passed to `std_fn()`

Value

a data frame obtained by matching the hierarchical columns in `raw` and `ref`, using the join type specified by argument `type` (see join_types for more details)

Resolve joins

In a resolve type join with `hmatch_settle`, rows of `raw` with multiple matches to `ref` are resolved to the highest hierarchical level that is non-conflicting among all matches (or no match if there is a conflict at the very first level). E.g.

`raw`:
```
1. | United States | <NA>         | Jefferson |
```

Relevant rows from `ref`:
```
1. | United States | <NA>         | <NA>      |
2. | United States | New York     | Jefferson |
3. | United States | Pennsylvania | Jefferson |
```

In a regular join, the single row from `raw` (above) will match all three rows from `ref`. However, in a resolve join the multiple matches will be resolved to the first row from `ref`, because only the first hierarchical level ("United States") is non-conflicting among all possible matches.

Note that there's a distinction between "common" values at a given hierarchical level (i.e. a single unique value in each row) and "non-conflicting" values (i.e. a single unique value *or* a missing value). E.g.

`raw`:
```
1. | United States | New York | New York |
```

Relevant rows from `ref`:
```
1. | United States | <NA>     | <NA>     |
2. | United States | New York | <NA>     |
3. | United States | New York | New York |
```

In the example above, only the 1st hierarchical level ("United States") is "common" to all matches, but all hierarchical levels are "non-conflicting" (i.e. because row 2 is a hierarchical child of row 1, and row 3 a child of row 2), and so a resolve-type match will be made to the 3rd row in `ref`.

Examples

```
data(ne_raw)
data(ne_ref)

# return matches at all levels
hmatch_settle(ne_raw, ne_ref, pattern = "^adm", type = "inner")

# use a resolve join to settle for the best possible match for each row
hmatch_settle(ne_raw, ne_ref, pattern = "^adm", type = "resolve_inner")
```

`hmatch_split` *Hierarchical matching, separately at each hierarchical level*

Description

Implements hierarchical matching, separately at each hierarchical level within the data. For a given level, the raw data that is matched includes every unique combination of values at and below the level of interest. E.g.

Level 1:
```
| Canada        |
| United States |
```

Level 2:
```
| Canada        | Ontario      |
| United States | New York     |
| United States | Pennsylvania |
```

Level 3:
```
| Canada        | Ontario      | Ottawa       |
| Canada        | Ontario      | Toronto      |
| United States | New York     | Bronx        |
| United States | New York     | New York     |
| United States | Pennsylvania | Philadelphia |
```

Usage

```
hmatch_split(
  raw,
  ref,
  pattern,
  pattern_ref = pattern,
  by,
  by_ref = by,
  fn = "hmatch",
  type = "left",
  allow_gaps = TRUE,
  fuzzy = FALSE,
  fuzzy_method = "osa",
  fuzzy_dist = 1L,
  dict = NULL,
  ref_prefix = "ref_",
  std_fn = string_std,
  ...,
  levels = NULL,
  always_list = FALSE,
  man,
  code_col,
  always_tokenize = FALSE,
  token_split = "_",
  exclude_freq = 3,
  exclude_nchar = 3,
  exclude_values = NULL
)
```

Arguments

`raw`	data frame containing hierarchical columns with raw data
`ref`	data frame containing hierarchical columns with reference data
`pattern`	regex pattern to match the hierarchical columns in `raw` **Note:** hierarchical column names can be matched using either the `pattern` *or* `by` arguments. Or, if neither `pattern` or `by` are specified, the hierarchical columns are assumed to be all column names that are common to both `raw` and `ref`. See specifying_columns.
`pattern_ref`	regex pattern to match the hierarchical columns in `ref`. Defaults to `pattern`, so only need to specify if the hierarchical columns have different names in `raw` and `ref`.
`by`	vector giving the names of the hierarchical columns in `raw`
`by_ref`	vector giving the names of the hierarchical columns in `ref`. Defaults to `by`, so only need to specify if the hierarchical columns have different names in `raw` and `ref`.
`fn`	which function to use for matching. Current options are `hmatch`, `hmatch_permute`, `hmatch_tokens`, `hmatch_settle`, or `hmatch_composite`. Defaults to "hmatch". Note that some subsequent arguments are only relevant for specific functions (e.g. the `exclude_` arguments are only relevant if `fn = "hmatch_tokens"`).
`type`	type of join ("left", "inner", "anti", "resolve_left", "resolve_inner", or "resolve_anti"). Defaults to "left". See join_types. Note that the details of resolve joins vary somewhat among hmatch functions (see documentation for the relevant function), and that function `hmatch_composite` only allows resolve joins.

`allow_gaps`	logical indicating whether to allow missing values below the match level, where 'match level' is the highest level with a non-missing value within a given row of `raw`. Defaults to `TRUE`.
`fuzzy`	logical indicating whether to use fuzzy-matching (based on the `stringdist` package). Defaults to FALSE.
`fuzzy_method`	if `fuzzy = TRUE`, the method to use for string distance calculation (see stringdist-metrics). Defaults to "osa".
`fuzzy_dist`	if `fuzzy = TRUE`, the maximum string distance to use to classify matches (i.e. a string distance less than or equal to `fuzzy_dist` will be considered matching). Defaults to `1L`.
`dict`	optional dictionary for recoding values within the hierarchical columns of `raw` (see dictionary_recoding)
`ref_prefix`	prefix to add to names of returned columns from `ref` if they are otherwise identical to names within `raw`. Defaults to "ref_".
`std_fn`	function to standardize strings during matching. Defaults to `string_std`. Set to `NULL` to omit standardization. See also string_standardization.
`...`	additional arguments passed to `std_fn()`
`levels`	a vector of names or integer indices corresponding to one or more of the hierarchical columns in `raw` to match at. Defaults to `NULL` in which case matches are made at each hierarchical level.
`always_list`	logical indicating whether to always return a list, even when argument `levels` specifies a single match level. Defaults to `FALSE`.
`man`	(optional) data frame of manually-specified matches, relating a given set of hierarchical values to the code within `ref` to which those values correspond
`code_col`	name of the code column containing codes for matching `ref` and `man` (only required if argument `man` is given)
`always_tokenize`	logical indicating whether to tokenize all values prior to matching (`TRUE`), or to first attempt non-tokenized matching with `hmatch` and only tokenize values within `raw` (and corresponding putative matches within `ref`) that don't have a non-tokenized match (`FALSE`). Defaults to `FALSE`.
`token_split`	regex pattern to split strings into tokens. Currently tokenization is implemented *after* string-standardizatipn with argument `std_fn` (this may change in a future version), so the regex pattern should split *standardized* strings rather than the original strings. Defaults to "_".
`exclude_freq`	exclude tokens from matching if they have a frequency greater than or equal to this value. Refers to the number of unique, string-standardized values at a given hierarchical level in which a given token occurs, as calculated by `count_tokens` (separately for `raw` and `ref`). Defaults to `3`.
`exclude_nchar`	exclude tokens from matching if they have nchar less than or equal to this value. Defaults to `3`.
`exclude_values`	character vector of additional tokens to exclude from matching. Subject to string-standardizatipn with argument `std_fn`.

Value

A list of data frames, each returned by a call to `fn` on the unique combination of hierarchical values at the given hierarchical level. The number of elements in the list corresponds to the number of hierarchical columns in `raw`, or, if specified, the number of elements in argument `levels`.

However, if `always_list = FALSE` and `length(levels) == 1`, a single data frame is returned (i.e. not wrapped in a list).

Examples

```
data(ne_raw)
data(ne_ref)

# by default calls fn `hmatch` separately for each hierarchical level
hmatch_split(ne_raw, ne_ref)

# can also specify other hmatch functions, and subsets of hierarchical levels
hmatch_split(ne_raw, ne_ref, fn = "hmatch_tokens", levels = 2:3)
```

`hmatch_tokens` *Hierarchical matching with tokenization of multi-term values*

Description

Match sets of hierarchical values (e.g. province / county / township) in a raw, messy dataset to corresponding values within a reference dataset, using tokenization to help match multi-term values that might otherwise be difficult to match (e.g. "New York City" vs. "New York").

Includes options for ignoring matches from frequently-occurring tokens (e.g. "North", "South", "City"), small tokens (e.g. "El", "San", "New"), or any other set of tokens specified by the user.

Usage

```
hmatch_tokens(
  raw,
  ref,
  pattern,
  pattern_ref = pattern,
  by,
  by_ref = by,
  type = "left",
  allow_gaps = TRUE,
  always_tokenize = FALSE,
  token_split = "_",
  token_min = 1,
  exclude_freq = 3,
  exclude_nchar = 3,
  exclude_values = NULL,
    fuzzy = FALSE,
    fuzzy_method = "osa",
    fuzzy_dist = 1L,
    dict = NULL,
    ref_prefix = "ref_",
    std_fn = string_std,
    ...
)
```

Arguments

`raw`	data frame containing hierarchical columns with raw data
`ref`	data frame containing hierarchical columns with reference data
`pattern`	regex pattern to match the hierarchical columns in `raw` **Note:** hierarchical column names can be matched using either the `pattern` *or* `by` arguments. Or, if neither `pattern` or `by` are specified, the hierarchical columns are assumed to be all column names that are common to both `raw` and `ref`. See specifying_columns.
`pattern_ref`	regex pattern to match the hierarchical columns in `ref`. Defaults to `pattern`, so only need to specify if the hierarchical columns have different names in `raw` and `ref`.
`by`	vector giving the names of the hierarchical columns in `raw`
`by_ref`	vector giving the names of the hierarchical columns in `ref`. Defaults to `by`, so only need to specify if the hierarchical columns have different names in `raw` and `ref`.
`type`	type of join ("left", "inner", "anti", "resolve_left", "resolve_inner", or "resolve_anti"). Defaults to "left". See join_types.
`allow_gaps`	logical indicating whether to allow missing values below the match level, where 'match level' is the highest level with a non-missing value within a given row of `raw`. Defaults to `TRUE`.
`always_tokenize`	logical indicating whether to tokenize all values prior to matching (`TRUE`), or to first attempt non-tokenized matching with `hmatch` and only tokenize values within `raw` (and corresponding putative matches within `ref`) that don't have a non-tokenized match (`FALSE`). Defaults to `FALSE`.
`token_split`	regex pattern to split strings into tokens. Currently tokenization is implemented *after* string-standardizatipn with argument `std_fn` (this may change in a future version), so the regex pattern should split *standardized* strings rather than the original strings. Defaults to "_".
`token_min`	minimum number of tokens that must match for a term to be considered matching overall. Defaults to 1.
`exclude_freq`	exclude tokens from matching if they have a frequency greater than or equal to this value. Refers to the number of unique, string-standardized values at a given hierarchical level in which a given token occurs, as calculated by `count_tokens` (separately for `raw` and `ref`). Defaults to `3`.

`exclude_nchar`	exclude tokens from matching if they have nchar less than or equal to this value. Defaults to `3`.
`exclude_values`	character vector of additional tokens to exclude from matching. Subject to string-standardizatipn with argument `std_fn`.
`fuzzy`	logical indicating whether to use fuzzy-matching (based on the `stringdist` package). Defaults to FALSE.
`fuzzy_method`	if `fuzzy = TRUE`, the method to use for string distance calculation (see stringdist-metrics). Defaults to "osa".
`fuzzy_dist`	if `fuzzy = TRUE`, the maximum string distance to use to classify matches (i.e. a string distance less than or equal to `fuzzy_dist` will be considered matching). Defaults to `1L`.
`dict`	optional dictionary for recoding values within the hierarchical columns of `raw` (see dictionary_recoding)
`ref_prefix`	prefix to add to names of returned columns from `ref` if they are otherwise identical to names within `raw`. Defaults to "ref_".
`std_fn`	function to standardize strings during matching. Defaults to `string_std`. Set to `NULL` to omit standardization. See also string_standardization.
`...`	additional arguments passed to `std_fn()`

Value

a data frame obtained by matching the hierarchical columns in `raw` and `ref`, using the join type specified by argument `type` (see join_types for more details)

Resolve joins

Uses the same approach to resolve joins as `hmatch`.

Examples

```
data(ne_raw)
data(ne_ref)

# add tokens to some values within ref to illustrate tokenized matching
ne_ref$adm0[ne_ref$adm0 == "United States"] <- "United States of America"
ne_ref$adm1[ne_ref$adm1 == "New York"] <- "New York State"

hmatch_tokens(ne_raw, ne_ref, type = "inner", token_min = 1)
```

join_types *Types of hierarchical joins*

Description

The basic join types used in the `hmatch` package ("left", "inner", "anti") are conceptually equivalent to dplyr's `join` types.

For each of the three join types there is also a counterpart prefixed by "resolve_" ("resolve_left", "resolve_inner", "resolve_anti"). In a resolve join rows of `raw` with matches to multiple rows of `ref` are resolved either to a single best match or no match before the subsequent join type is implemented. In a resolve join, rows of `raw` are never duplicated.

The exact details of match resolution vary somewhat among functions, and are explained within each function's documentation.

Value

`left`	return all rows from `raw`, and all columns from `raw` and `ref`. Rows in `raw` with no match in `ref` will have NA values in the new columns taken from `ref`. If there are multiple matches between `raw` and `ref`, all combinations of the matches are returned.
`inner`	return only the rows of `raw` that have matches in `ref`, and all columns from `raw` and `ref`. If there are multiple matches between `raw` and `ref`, all combinations of the matches are returned.
`anti`	return all rows from `raw` where there are not matches in `ref`, keeping just columns from `raw`
`resolve_left`	similar to "left", except that any row of `raw` that initially has multiple matches to `ref` is resolved to either a single 'best' match or no match. All rows of `raw` are returned, and rows of `raw` are never duplicated.
`resolve_inner`	similar to "inner", except that any row of `raw` that initially has multiple matches to `ref` is resolved to either a single 'best' match or no match. Only the rows of `raw` that can be resolved to a single best match are returned, and rows of `raw` are never duplicated.
`resolve_anti`	similar to "anti", except that any row of `raw` that initially has multiple matches to `ref` is considered non-matching (along with rows of `raw` that initially have no matches to `ref`), and returned as a single row. Rows of `raw` are never duplicated.

max_levels *Maximum hierarchical levels*

Description

Given a data frame with columns specifying hierarchically-nested levels, find the maximum non-missing hierarchical level for each row.

Usage

```
max_levels(x, pattern, by, type = c("index", "name"))
```

Arguments

`x`	a data frame containing hierarchical columns
`pattern`	regex pattern to match the names of the hierarchical columns in `ref` (supply either `pattern` *or* `by`)
`by`	vector giving the names of the hierarchical columns in `ref` (supply either `pattern` *or* `by`)
`type`	type of return, either "index" to return integer indices (starting at 1) or "name" to return column names (as matched by `pattern` or `by`)

Value

Vector of indices or names corresponding to the maximum non-missing hierarchical level for each row

Examples

```
data(ne_ref)

# return integer indices (starting at 1)
max_levels(ne_raw, pattern = "^adm")

# return column names
max_levels(ne_raw, pattern = "^adm", type = "name")
```

`ne_raw` *Raw dataset*

Description

Raw entries of select administrative districts from the northeastern portion of North America.

Usage

```
ne_raw
```

Format

A data.frame with 15 rows and 4 variables:

id Identifier

adm0 Name of administrative 0 level (country)

adm1 Name of administrative 1 level (state/province)

adm2 Name of administrative 2 level (county/census division)

ne_ref — *Reference dataset*

Description

Reference table of select administrative districts in the northeastern portion of North America.

Usage

```
ne_ref
```

Format

A data.frame with 31 rows and 4 variables, all of class character:

level Administrative level

adm0 Name of administrative 0 level (country)

adm1 Name of administrative 1 level (state/province)

adm2 Name of administrative 2 level (county/census division)

hcode Hierarchical code

ref_expand — *Expand a reference data.frame containing N hierarchical columns to an N-level reference data.frame*

Description

For example, a municipality-level reference data.frame might contain three hierarchical columns — country, state, and municipality — but nonetheless only reflect the municipality level in that all rows represent a unique municipality. The lower-resolution levels (state, country) are implied but not explicitly represented as unique rows. If we wish to allow matches to the lower-resolution levels, we need additional rows specific to these levels.

This function takes a reference data.frame with N hierarchical columns, and adds rows for each unique combination of each level that is not currently explicitly represented.

Usage

```
ref_expand(ref, pattern, by, lowest_level = 1L)
```

Arguments

`ref`	`data.frame` containing hierarchical columns with reference data
`pattern`	regex pattern to match the names of the hierarchical columns in `ref` (supply either `pattern` *or* `by`)
`by`	vector giving the names of the hierarchical columns in `ref` (supply either `pattern` *or* `by`)
`lowest_level`	integer representing the lowest-resolution level (defaults to `1`)

Value

A `data.frame` created by expanding `ref` to all implied hierarchical levels

Examples

```
# subset example reference df to the admin-2 level
ne_ref_adm2 <- ne_ref[!is.na(ne_ref$adm2),]

# expand back to all levels
ref_expand(ne_ref_adm2, pattern = "adm", lowest_level = 0)
```

`separate_hcode` *Separate a hierarchical code reflecting multiple levels into its constituent parts, with one column for each level*

Description

Separate a data frame column containing hierarchical codes into multiple columns, one for each level within the hierarchical code.

Like `tidyr::separate` except that successive levels are cumulative rather then independent. E.g. the code "canada__ontario__toronto" would be split into three levels:

1. "canada"
2. "canada__ontario"
3. "canada__ontario__toronto"

Usage

```
separate_hcode(
  x,
  col,
  into,
  sep = "__",
  extra = c("warn", "drop"),
  remove = FALSE
)
```

Arguments

`x`	`data.frame` containing a column with hierarchical codes
`col`	Name of the column within `x` containing hierarchical codes.
`into`	Vector of column names to separate `col` into
`sep`	Separator between levels in the hierarchical codes. Defaults to "__".
`extra`	What to do if a hierarchical code contains more levels than are implied by argument `into`. • "warn" (the default): emit a warning and drop extra values • "drop": drop any extra values without a warning
`remove`	Logical indicating whether to remove `col` from the output. Defaults to `FALSE`.

Value

The original data.frame `x` with additional columns for each level of the hierarchical code

Examples

```
data(ne_ref)

# generate pcode
ne_ref$pcode <- hcodes_str(ne_ref, pattern = "^adm\\d")

# separate pcode into constituent levels
separate_hcode(
  ne_ref,
  col = "pcode",
  into = c("adm0_pcode", "adm1_pcode", "adm2_pcode")
)
```

`specifying_columns` *Specifying hierarchical columns with arguments* `pattern` *or* `by`

Description

Within the `hmatch_` group of functions, there are three ways to specify the hierarchical columns to be matched.

In all cases, it is assumed that matched columns are already correctly ordered, with the first matched column reflecting the broadest hierarchical level (lowest-resolution, e.g. country) and the last column reflecting the finest level (highest-resolution, e.g. township).

(1) All column names common to `raw` and `ref`

If neither `pattern` nor `by` are specified (the default), then the hierarchical columns are assumed to be all column names that are common to both `raw` and `ref`.

(2) Regex pattern

Arguments `pattern` and `pattern_ref` take regex patterns to match the hierarchical columns in `raw` and `ref`, respectively. Argument `pattern_ref` only needs to be specified if it's different from `pattern` (i.e. if the hierarchical columns have different names in `raw` vs. `ref`).

For example, if the hierarchical columns in `raw` are "ADM_1", "ADM_2", and "ADM_3", which correspond respectively to columns within `ref` named "REF_ADM_1", "REF_ADM_2", and "REF_ADM_3", then the pattern arguments can be specified as:

- `pattern = "^ADM_[[:digit:]]"`
- `pattern_ref = "^REF_ADM_[[:digit:]]"`

Alternatively, because `pattern_ref` defaults to the same value as `pattern` (unless otherwise specified), one could specify a single regex pattern that matches the hierarchical columns in both `raw` and `ref`, e.g.

- `pattern = "ADM_[[:digit:]]"`

However, the user should exercise care to ensure that there are no non-hierarchical columns within `raw` or `ref` that may inadvertently be matched by the given pattern.

(3) Vector of column names

If the hierarchical columns cannot easily be matched with a regex pattern, one can specify the relevant column names in vector form using arguments `by` and `by_ref`. As with `pattern_ref`, argument `by_ref` only needs to be specified if it's different from `by` (i.e. if the hierarchical columns have different names in `raw` vs. `ref`).

For example, if the hierarchical columns in `raw` are "state", "county", and "township", which correspond respectively to columns within `ref` named "admin1", "admin2", and "admin3", then the`by` arguments can be specified with:

- `by = c("state", "county", "township")`
- `by_ref = c("admin1", "admin2", "admin3")`

`string_standardization` *String Standardization*

Description

Prior to matching raw and reference datasets, one might wish to standardize the strings within the match columns to account for differences in case, punctuation, etc.

By default, this standardization is performed with function `string_std`, which implements four transformations:

1. standardize case (`base::tolower`)
2. remove sequences of non-alphanumeric characters at start or end of string

3. replace remaining sequences of non-alphanumeric characters with "_"
4. remove diacritics (`stringi::stri_trans_general`)
5. (optional) convert roman numerals (I, II, ..., XLIX) to arabic (1, 2, ..., 49)

Alternatively, the user may provide any function that takes a vector of strings and returns a vector of transformed strings. To omit any transformation, set argument `std_fn = NULL`.

Note that the standardized versions of the match columns are never returned. They are used only during matching, and then removed prior to the return.

`string_std` *String standardization prior to matching*

Description

Standardizes strings prior to performing a match, using the following transformations:

1. standardize case (`base::tolower`)
2. remove sequences of non-alphanumeric characters at start or end of string
3. replace remaining sequences of non-alphanumeric characters with "_"
4. remove diacritics (`stringi::stri_trans_general`)
5. (optional) convert roman numerals (I, II, ..., XLIX) to arabic (1, 2, ..., 49)

Usage

```
string_std(x, convert_roman = FALSE)
```

Arguments

`x`	a string
`convert_roman`	logical indiciating whether to convert roman numerals (I, II, ..., XLIX) to arabic (1, 2, ..., 49)

Value

The standardized version of `x`

See Also

string_standardization

Examples

```
string_std("United STATES")
string_std("R\u00e9publique  d\u00e9mocratique du  Congo")

# convert roman numerals to arabic
string_std("Mungindu-II (Sud)")
string_std("Mungindu-II (Sud)", convert_roman = TRUE)

# note the conversion only works if the numeral is separated from other
# alphanumeric characters by punctuation or space characters
string_std("MunginduII", convert_roman = TRUE) # roman numeral not recognized
```

Index