Package 'ulex'

February 13, 2026

Title Unique Location Extractor

Version 0.1.1

Description Extracts coordinates of an event location from text based on dictionaries of landmarks, roads, and areas. Only returns the location of an event of interest and ignores other location references; for example, if determining the location of a road traffic crash from the text ``crash near [location 1] heading towards [location 2]", only the coordinates of ``location 1" would be returned. Moreover, accounts for differences in spelling between how a user references a location and how a location is captured in location dictionaries. For more information on the algorithm, see Milusheva et al. (2021) <doi:10.1371/journal.pone.0244317>.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Depends R (>= 4.1.0)

Imports dplyr, tidyr, readr, purrr, tidytext, stringr, stringi, ngram, hunspell, stringdist, tm, raster, parallel, sf, quanteda, geodist, spacyr, utils

URL https://dime-worldbank.github.io/ulex/

NeedsCompilation no

Author Robert Marty [aut, cre] (ORCID: <https://orcid.org/0000-0002-3164-3813>)

Maintainer Robert Marty <rmarty@worldbank.org>

Repository CRAN

Date/Publication 2026-02-13 19:10:02 UTC

Contents

augment_gazetteer *Augments Landmark Gazetteer*

Description

Augments Landmark Gazetteer

Usage

```
augment_gazetteer(
  landmarks,
  landmarks.name_var = "name",
  landmarks.type_var = "type",
  grams.min_words = 3,
  grams.max_words = 6,
  grams.skip_gram_first_last_word_match = TRUE,
  grams.add_only_if_name_new = FALSE,
  grams.add_only_if_specific = FALSE,
 types_rm = c("route", "road", "toilet", "political", "locality", "neighborhood",
    "area", "section of populated place"),
  types_rm.except_with_type = c("flyover", "round about", "roundabout"),
  types_rm.except_with_name = c("flyover", "round about", "roundabout"),
  parallel.sep_slash = TRUE,
 parallel.rm_begin = c(tm::stopwords("en"), c("near", "at", "the", "towards", "near")),
 parallel.rm_end = c("bar", "shops", "restaurant", "sports bar", "hotel", "bus station"),
  parallel.word_diff = "default",
 parallel.word_diff_iftype = list(list(words = c("stage", "bus stop", "bus station"),
    type = "transit_station")),
  parallel.rm_begin_iftype = NULL,
 parallel.rm_end_iftype = list(list(words = c("stage", "bus stop", "bus station"), type
    = "transit_station")),
  parallel.word_begin_addtype = NULL,
 parallel.word_end_addtype = list(list(words = c("stage", "bus stop", "bus station"),
    type = "stage")),
  parallel.add_only_if_name_new = FALSE,
  parallel.add_only_if_specific = FALSE,
  rm.contains = c("road", "rd"),
 rm.name_begin = c(tm::stopwords("en"), c("near", "at", "the", "towards", "near")),
 rm.name_end = c("highway", "road", "rd", "way", "ave", "avenue", "street", "st"),
 pos_rm.all = c("ADJ", "ADP", "ADV", "AUX", "CCONJ", "INTJ", "NUM", "PRON", "SCONJ",
    "VERB", "X"),
 pos_rm.except_type = list(pos = c("NOUN", "PROPN"), type = c("bus", "restaurant",
    "bank"), name = ""),
  close_thresh_km = 1,
  quiet = TRUE
)
```

Arguments

landmarks	sf spatial points data.frame of landmarks.
landmarks.name_var	Name of variable indicating name of landmark. (Default: "name").
landmarks.type_var	Name of variable indicating type of landmark. (Default: "type").
grams.min_words	Minimum number of words in name to make n/skip-grams out of name. (Default: 3).
grams.max_words	Maximum number of words in name to make n/skip-grams out of name. Setting a cap helps to reduce spurious landmarks that may come out of really long names. (Default: 6).
grams.skip_gram_first_last_word_match	For skip-grams, should first and last word be the same as the original word? (Default: TRUE).
grams.add_only_if_name_new	When creating new landmarks based on n- and skip-grams, only add an additional landmark if the name of the landmark is new; i.e., the name doesn't already exist in the gazetteer. (Default: FALSE).
grams.add_only_if_specific	When creating new landmarks based on n- and skip-grams, only add an additional landmark if the name of the landmark represents a specific location. A specific location is a location where most landmark entries with the same name are close together (within close_thresh_km kilometers). (Default: FALSE).
types_rm	If landmark has one of these types, remove - unless types_rm.except_with_type or types_rm.except_with_name prevents removing. (Default: c("route", "road", "toilet", "political", "locality", "neighborhood", "area", "section of populated place")).
types_rm.except_with_type	Landmark types to always keep. This parameter only becomes relevant in cases where a landmark has more than one type. If a landmark has both a "types_rm" and a "types_always_keep" landmark, this landmark will be kept. (Default: c("flyover", "round about", "roundabout")).
types_rm.except_with_name	Landmark names to always keep. This parameter only becomes relevant in cases where a landmark is one of "types_rm" Here, we keep the landmark if "names_always_keep" is somewhere in the name. For example, if the landmark is a road but has flyover in the name, we may want to keep the landmark as flyovers are small spatial areas. (Default: c("flyover", "round about", "roundabout")).
parallel.sep_slash	If a landmark contains a slash, create new landmarks before and after the slash. (Default: TRUE).

parallel.rm_begin
: If a landmark name begins with one of these words, add a landmark that excludes the word. (Default: `c(tm::stopwords("en"), c("near","at","the", "towards", "near"))`).

parallel.rm_end
: If a landmark name ends with one of these words, add a landmark that excludes the word. (Default: `c("bar", "shops", "restaurant","sports bar","hotel", "bus station")`).

parallel.word_diff
: If the landmark includes one of these words, add a landmark that swaps the word for the other word (e.g., "center" with "centre"). By default, uses a set collection of words. Users can also manually specify different word versions. Input should be a `data.frame` with the following variables: `version_1` (for one spelling of the word) and `version_2` (for a second spelling of the word).

parallel.word_diff_iftype
: If the landmark includes one of these words, add a landmark that swaps the word for the other word (e.g., "bus stop" with "bus station"). Enter a named list of words, with `words = c()` and `type = c()`. (Default: `list(list(words = c("stage", "bus stop", "bus station"), type = "transit_station"))`).

parallel.rm_begin_iftype
: If a landmark name begins with one of these words, add a landmark that excludes the word if the landmark is a certain type. (Default: `NULL`).

parallel.rm_end_iftype
: If a landmark name ends with one of these words, add a landmark that excludes the word if the landmark is a certain type. (Default: `list(list(words = c("stage", "bus stop", "bus station"), type = "transit_station"))`).

parallel.word_begin_addtype
: If the landmark begins with one of these words, add the type. For example, if landmark is "restaurant", this indicates the landmark is a restaurant. Adding the "restaurant" to landmark ensures that the type is reflected. (Default: `NULL`).

parallel.word_end_addtype
: If the landmark ends with one of these words, add the type. For example, if landmark is "X stage", this indicates the landmark is a bus stage. Adding the "stage" to landmark ensures that the type is reflected. (Default: `list(list(words = c("stage", "bus stop", "bus station"), type = "stage"))`).

parallel.add_only_if_name_new
: When creating parallel landmarks using the above parameters, only add an additional landmark if the name of the landmark is new; i.e., the name doesn't already exist in the gazetteer. (Default: `FALSE`).

parallel.add_only_if_specific
: When creating parallel landmarks using the above parameters, only add an additional landmark if the name of the landmark represents a specific location. A specific location is a location where most landmark entries with the same name are close together (within `close_thresh_km` kilometers). (Default: `FALSE`).

rm.contains
: Remove the landmark if it contains one of these words. Implemented after N/skip-grams and parallel landmarks are added. (Default: `c("road", "rd")`).

`rm.name_begin` Remove the landmark if it begins with one of these words. Implemented after N/skip-grams and parallel landmarks are added. (Default: `c(tm::stopwords("en"), c("near","at","the", "towards", "near"))`).

`rm.name_end` Remove the landmark if it ends with one of these words. Implemented after N/skip-grams and parallel landmarks are added. (Default: `c("highway", "road", "rd", "way", "ave", "avenue", "street", "st")`).

`pos_rm.all` Part-of-speech categories to remove. Part-of-speech determined by Spacy. (Default: `c("ADJ", "ADP", "ADV", "AUX", "CCONJ", "INTJ", "NUM", "PRON", "SCONJ", "VERB", "X")`).

`pos_rm.except_type` When specify part-of-speech categories to remove in `pos_rm.all`, when to override `pos_rm.all` and keep the word. Names list with: (1) `pos` (if the word is also another type of part-of-speech); (2) `type` (if the word is also a certain type of place); and (3) `name` (if the word includes certain text). Example: `list(pos = c("NOUN", "PROPN"), type = c("bus", "restaurant", "bank"), name = c("parliament"))`. (Default: `list(pos = c("NOUN", "PROPN"), type = c("bus", "restaurant", "bank"), name = "")`).

`close_thresh_km` When to consider locations close together. Used when determining if a landmark name with multiple locations are specific (close together) or general (far apart). (Default: 1).

`quiet` Print progress of function. (Default: `TRUE`).

Value

`sf` spatial point data.frame of landmarks.

Examples

```
library(ulex)
library(spacyr)
spacy_install()

lm_sf <- data.frame(name = c("white house",
                             "the world bank group",
                             "the george washington university"),
                    lat = c(38.897778,
                            38.89935,
                            38.9007),
                    lon = c(-77.036389,
                            -77.04275,
                            -77.0508),
                    type = c("building", "building", "building")) |>
sf::st_as_sf(coords = c("lon", "lat"),
         crs = 4326)

lm_aug_sf <- augment_gazetteer(lm_sf)
```

locate_event *Locate Event*

Description

Locate Event

Usage

```
locate_event(
  text,
  landmark_gazetteer,
  landmark_gazetteer.name_var = "name",
  landmark_gazetteer.type_var = "type",
  roads,
  roads.name_var = "name",
  areas,
  areas.name_var = "name",
  event_words,
 prepositions_list = list(c("at", "next to", "around", "just after", "opposite", "opp",
  "apa", "hapa", "happened at", "just before", "at the", "outside", "right before"),
  c("near", "after", "toward", "along", "towards", "approach"), c("past", "from",
    "on")),
  junction_words = c("intersection", "junction"),
  false_positive_phrases = "",
  type_list = NULL,
  clost_dist_thresh = 500,
  fuzzy_match = TRUE,
  fuzzy_match.min_word_length = c(5, 11),
  fuzzy_match.dist = c(1, 2),
  fuzzy_match.ngram_max = 3,
  fuzzy_match.first_letters_same = TRUE,
  fuzzy_match.last_letters_same = TRUE,
  quiet = TRUE,
  mc_cores = 1
)
```

Arguments

text	Vector of texts to be geolocated.
landmark_gazetteer	sf spatial data.frame representing landmarks.
landmark_gazetteer.name_var	Name of variable indicating name of landmark.
landmark_gazetteer.type_var	Name of variable indicating type of landmark.

roads — sf spatial data.frame representing roads.

roads.name_var — Name of variable indicating name of road.

areas — sf spatial data.frame representing areas, such as administrative areas or neighborhoods.

areas.name_var — Name of variable indicating name of area.

event_words — Vector of event words, representing events to be geocoded.

prepositions_list — List of vectors of prepositions. Order of list determines order of preposition precedence. (Default: `list(c("at", "next to","around", "just after", "opposite","opp", "apa", "hapa","happened at", "just before","at the","outside", "right before"), c("near", "after", "toward", "along", "towards", "approach"), c("past","from","on"))`).

junction_words — Vector of junction words to check for when determining intersection of roads. (Default: `c("intersection", "junction")`).

false_positive_phrases — Common words found in text that include spurious location references (eg, **githurai bus** is the name of a bus, but **githurai** is also a place). These may be common phrases that should be checked and ignored in the text. (Default: `""`).

type_list — List of vectors of types. Order of list determines order or type precedence. (Default: NULL).

clost_dist_thresh — Distance (meters) as to what is considered "close"; for example, when considering whether a landmark is close to a road. (Default: 500).

fuzzy_match — Whether to implement fuzzy matching of landmarks using levenstein distance. (Default: TRUE).

fuzzy_match.min_word_length — Minimum word length to use for fuzzy matching; vector length must be the same as fuzzy_match.dist. (Default: c(5,11)).

fuzzy_match.dist — Allowable levenstein distances for fuzzy matching; vector length must be same as fuzzy_match.min_word_length. (Default: c(1,2)).

fuzzy_match.ngram_max — The number of n-grams that should be extracted from text to calculate a levensteing distance against landmarks. For example, if the text is composed of 5 words: w1 w2 w3 w4 and fuzzy_match.ngram_max = 3, the function extracts w1 w2 w3 and compares the levenstein distance to all landmarks. Then in checks w2 w3 w4, etc. (Default: 3).

fuzzy_match.first_letters_same — When implementing a fuzzy match, should the first letter of the original and found word be the same? (Default: TRUE).

fuzzy_match.last_letters_same — When implementing a fuzzy match, should the last letter of the original and found word be the same? (Default: TRUE).

quiet — If FALSE, prints text that is being geocoded. (Default: TRUE).

mc_cores — If > 1, uses geolocates events in parallel across multiple cores relying on the parallel package. (Default: 1).

Value

sf spatial dataframe of geolocated events.

Examples

```
library(ulex)
library(sf)

## Landmarks
landmarks_sf <- data.frame(lat = runif(3),
                           lon = runif(3),
                           name = c("restaurant", "bank", "hotel"),
                           type = c("poi", "poi", "poi")) |>
  st_as_sf(coords = c("lon", "lat"),
           crs = 4326)

## Road
coords <- matrix(runif(4), ncol = 2)
road_sf <- coords |>
  st_linestring() |>
  st_sfc(crs = 4326)
road_sf <- st_sf(geometry = road_sf)
road_sf$name <- "main st"

## Area
n <- 5
coords <- matrix(runif(2 * n, min = 0, max = 10), ncol = 2)
coords <- rbind(coords, coords[1,])
polygon <- st_polygon(list(coords))
area_sf <- st_sfc(polygon, crs = 4326)
area_sf <- st_sf(geometry = area_sf)
area_sf$name <- "place"

## Locate Event
event_sf <- locate_event(text = "accident near hotel",
                         landmark_gazetteer = landmarks_sf,
                         roads = road_sf,
                         areas = area_sf,
                         event_words = c("accident", "crash"))
```

Index