Package 'occupationMeasurement'

July 22, 2025

Title Interactively Measure Occupations in Interviews and Beyond

Version 0.3.2

Description Perform interactive occupation coding during interviews as described in Peycheva, D., Sakshaug, J., Calderwood, L. (2021) <doi:10.2478/jos-2021-0042> and Schierholz, M., Gensicke, M., Tschersich, N., Kreuter, F. (2018) <doi:10.1111/rssa.12297>. Generate suggestions for occupational categories based on free text input, with pre-trained machine learning models in German and a ready-to-use shiny application provided for quick and easy data collection.

License MIT + file LICENSE

URL https://occupationMeasurement.github.io/occupationMeasurement/, https://github.com/occupationMeasurement/occupationMeasurement

BugReports https://github.com/occupationMeasurement/occupationMeasurement/issues

Encoding UTF-8

RoxygenNote 7.2.1

Imports data.table (>= 1.14.2), digest, shiny (>= 1.7.1), stringdist (>= 0.9.8), stringr (>= 1.4.0), text2vec (>= 0.6.1), tm (>= 0.7.8)

Suggests plumber, knitr, mvtnorm, callr, devtools, httr, sessioninfo, shinytest2, testthat, withr

Depends R (>= 4.1)

LazyData true

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Jan Simson [aut, cre] (ORCID: <https://orcid.org/0000-0002-9406-7761>), Olga Kononykhina [aut], Malte Schierholz [aut, cph] (ORCID: <https://orcid.org/0000-0003-4058-1543>)

Maintainer Jan Simson <jan.simson@lmu.de>

Repository CRAN

Date/Publication 2023-09-27 13:40:02 UTC

Contents

`algo_similarity_based_reasoning` *Make suggestions using similarity based reasoning.*

Description

The Algorithm used here corresponds to Algorithm #10 in (Schierholz, 2019). Note: This function should not be used directly, but rather as a step / algorithm in get_job_suggestions.

Usage

```
algo_similarity_based_reasoning(
  text_processed,
  sim_name = "wordwise",
 probabilities = occupationMeasurement::pretrained_models$similarity_based_reasoning,
  ...
)
```

Arguments

text_processed	The processed user input. Will be provided by get_job_suggestions.
sim_name	Which similarity measure to use. Possible values are "wordwise" or "substring".
probabilities	Trained probabilities to be used, defaults to the one bundled with the package. See pretrained_models. This pretrained model always predicts a 5-digit code from the 2010 German Classification of Occupations, with some exceptions: -0004 stands for 'Not precise enough/uncodable', -0006 stands for 'Multiple Jobs', -0012 stands for 'Blue-collar workers', -0019 stands for 'Volunteer/Social Service', and -0030 stands for 'Student assistant'.
...	Additional arguments may be passed from get_job_suggestions(), but will be ignored in this function.

Value

A data.table with suggestions or NULL if no suggestions were found.

References

Schierholz, M. (2019). New Methods for Job and Occupation Classification (Ph.D. Thesis). University of Mannheim.

See Also

get_job_suggestions()

Examples

```
## Not run:
# Use with default settings
if (interactive()) {
  get_job_suggestions(
    "Arzt",
    steps = list(
      simbased_default = list(
        algorithm = algo_similarity_based_reasoning
      )
    )
  )
}
```

```
# Use with substring similarity
if (interactive()) {
 get_job_suggestions(
   "Arzt",
   steps = list(
     simbased_substring = list(
       algorithm = algo_similarity_based_reasoning,
       parameters = list(
         sim_name = "substring"
       )
     )
   )
 )
}

# Comparison of algo_similarity_based_reasoning() with get_job_suggestions()

# Example of using algo_similarity_based_reasoning() directly. Not recommended.
if (interactive()) {
  algo_similarity_based_reasoning(
    preprocess_string("Arzt"),
    sim_name = "wordwise"
  )[order(score, decreasing = TRUE)]
}

# Same output as before, but the function is more adaptable.
if (interactive()) {
  get_job_suggestions(
    "Arzt",
    suggestion_type = "kldb-2010",
    num_suggestions = 1500,
    steps = list(
      simbased_default = list(
        algorithm = algo_similarity_based_reasoning,
        parameters = list(
          sim_name = "wordwise"
        )
      )
    )
  )[, list(kldb_id, score, sim_name, kldb_id_title = title)]
}

## End(Not run)
```

---

api *Start the occupation coding API.*

Description

Start the occupation coding API.

Usage

```
api(
  start = TRUE,
  log_to_file = FALSE,
 file = system.file("plumber", "api", "plumber.R", package = "occupationMeasurement"),
  log_to_console = TRUE,
  log_filepath = file.path("output", "log_api.csv"),
  require_identifier = FALSE,
  allow_origin = NULL
)
```

Arguments

start	Whether to immediately start the api. (Defaults to TRUE)
log_to_file	Whether to requests should be logged in a file. Defaults to FALSE. Note: The file format used here is a CSV file for easier analysis.
file	Path to the `plumber.R` file describing the API. Defaults to `plumber/api/plumber.R` within the installed package. Refer to this file to understand how the API is implemented.
log_to_console	Whether to requests should be logged in the console. Defaults to TRUE.
log_filepath	The path to a CSV file in which to save the structured logs.
require_identifier	Whether an identifier has to be added to api requests in order to match / identify requests afterwards. Defaults to FALSE.
allow_origin	Domain from which to allow cross origin requests (CORS). If the API is running on a different domain / server than the application using it, the website's root has to be provided here e.g. "https://occupationMeasurement.github.io". For more information see the plumber security page, and MDN. Defaults to NULL to not set any header at all.

Value

A Plumber router

See Also

```
vignette("api")
```

Examples

```
if (interactive()) {
  # Get the plumber router
  router <- api(
    start = FALSE,
    # If this is TRUE, the log directory will immediately be created
    log_to_file = FALSE
  )
```

```
  # Start the router
  plumber::pr_run(router)
}

if (interactive()) {
  # Immediately start the API
  api(start = TRUE)
}
```

---

app *Get an instance of the interactive shiny occupation coding app.*

Description

Printing the returned instance or returning it without saving it in a variable will start the app.

Usage

```
app(
  questionnaire = questionnaire_web_survey(),
  app_settings = create_app_settings(save_to_file = TRUE),
  css_file = NULL,
  resource_dir = system.file("www", package = "occupationMeasurement"),
  ...
)
```

Arguments

questionnaire	The questionnaire to load. (Defaults to the questionnaire returned by questionnaire_web_survey().)
app_settings	The app_settings to use. Check the documentation for create_app_settings to learn about the options.
css_file	Path to a CSS file to be included in the app.
resource_dir	From which directory to static files e.g. styles. If you want to load additional resources from outside the package, you should rather do so with shiny::addResourcePath rather than with this parameter.
...	Any additional parameters will be forwarded to shiny::shinyApp().

Value

A shiny app instance.

See Also

vignette("app"), questionnaire_web_survey()

Examples

```
## Not run:
app_instance <- app(
  app_settings = create_app_settings(
    # Important to save results from the app
    save_to_file = TRUE
  )
)

# Start the app
if (interactive()) {
  app_instance
}

## End(Not run)
```

---

auxco *German Auxiliary Classification of Occupations (AuxCO) v1.2.3*

Description

Berufs-Hilfsklassifikation mit Tätigkeitsbeschreibungen.

Usage

```
auxco
```

Format

A list with data.tables:

`categories` data.table. Main list of AuxCO categories including their descriptions etc.

`distinctions` data.table. List of highly similar AuxCO categories that one may want to present to disambiguate between them.

`followup_questions` data.table. Follow-up questions to specify final codings based on AuxCO categories. Includes the questions' answer options as well as information on how to encode more complex occupations which depend on multiple answers.

`mapping_from_isco` data.table. Mapping from ISCO-08 categories to AuxCO categories.

`mapping_from_kldb` data.table. Mapping from KldB 2010 categories to AuxCO categories.

References

Schierholz, Malte; Brenner, Lorraine; Cohausz, Lea; Damminger, Lisa; Fast, Lisa; Hörig, Ann-Kathrin; Huber, Anna-Lena; Ludwig, Theresa; Petry, Annabell; Tschischka, Laura (2018): Vorstellung einer Hilfsklassifikation mit Tätigkeitsbeschreibungen für Zwecke der Berufskodierung. (IAB-Discussion Paper, 2018), Nürnberg, 45 S. https://www.iab.de/183/section.aspx/Publikation/k180509301

See Also

https://github.com/occupationMeasurement/auxiliary-classification, load_auxco()

---

button_next *Go to the next page*

Description

Buttons to navigate between pages.

Usage

```
button_next(label = "Weiter")

button_previous(label = "Zurück")
```

Arguments

label	What label the button should have.

Value

shiny Action Button

Functions

- button_previous(): Go to the previous page

See Also

new_page()

Examples

```
## Not run:
very_simple_page <- new_page(
  page_id = "example",
  render = function(session, run_before_output, input, output, ...) {
    list(
      shiny::tags$h1("My test page"),
      button_previous(),
      button_next()
    )
  }
)

## End(Not run)
```

create_app_settings *Create app_settings.*

Description

This is the primary and most convenient way of configuring the app.

Usage

```
create_app_settings(
  save_to_file,
  suggestion_type = "auxco-1.2.x",
  default_num_suggestions = 5,
  require_respondent_id = FALSE,
  warn_before_leaving = FALSE,
  skip_followup_types = c(),
  response_output_dir = file.path("output", "responses"),
  handle_data = NULL,
  get_job_suggestion_params = NULL,
  display_page_ids = TRUE,
  default_tense = "present",
  default_extra_instructions = "on",
  verbose = TRUE,
  .validate = TRUE
)
```

Arguments

save_to_file	Should responses be saved as files in response_output_dir? Defaults to use the SAVE_TO_FILE environment variable. We recommend setting this to TRUE.
suggestion_type	Which type of suggestion to use / provide. Possible options are "auxco-1.2.x" and "kldb-2010".
default_num_suggestions	The number of suggestions to generate and display to users. Accepts all positive integers. Defaults to 5.
require_respondent_id	Are respondent_ids required? Defaults to FALSE
warn_before_leaving	Should users be warned that their progress will be lost upon leaving the site? Defaults to FALSE.
skip_followup_types	A vector of strings corresponding to the question_type of followup_question that should be skipped. Allowed ‘ values: c("anforderungsniveau", "aufsicht", "spezialisierung", "sonstige")

response_output_dir
: Path to the directory in which to store data from the app. Defaults to `./output/responses/`.

handle_data
: Callback function to handle data from the app. This setting takes a function that get's passed 3 parameters: table_name (A reference name indicating which data to save), data (A dataframe of data to save), session (the user's current session).

get_job_suggestion_params
: List of parameters to pass to get_job_suggestion. Refer to `get_job_suggestions()` for a list of supported parameters. Note that the parameter aggregate_score_threshold needs to be set on `page_first_freetext()` or `page_second_freetext()`.

display_page_ids
: Whether `page_ids` should be displayed within the questionnaires.

default_tense
: We may not always want to ask for the current occupation, but maybe also for the previous occupation in case of pensioners etc. with a value of "past". Possible values are "present" (default), "past". This setting can be overwritten on a session-by-session basis with the URL-Query parameter "tense".

default_extra_instructions
: Display additional instructions for e.g. an interviewer conducting an interview. Possible values are "on" (default), "off". This setting can be overwritten on a session-by-session basis with the URL-Query parameter "extra_instructions".

verbose
: Should additional output be printed when running? Defaults to TRUE.

.validate
: Whether the created app_settings should be validated. Defaults to TRUE.

Value

A list of app_settings.

Examples

```
app_settings <- create_app_settings(
  # Important to save results from the app
  save_to_file = TRUE,
  require_respondent_id = TRUE
)
```

---

get_final_codes *Get the final occupation codes*

Description

The final occupation code will depend on the `suggestion_id` and, possibly, on `followup_answers`, depending on the `suggestion_id` provided. See `occupationMeasurement::auxco$followup_questions` for a list of suggestion_ids (=auxco_id) and their respective recommended follow-up questions.

Usage

```
get_final_codes(
  suggestion_id,
  followup_answers = list(),
  standardized_answer_levels = NULL,
  approximate_standardized_answer_levels = TRUE,
  code_type = c("isco_08", "kldb_10"),
  verbose = TRUE,
  suggestion_type = "auxco-1.2.x",
  suggestion_type_options = list()
)
```

Arguments

`suggestion_id`	Id of the suggestion
`followup_answers`	A named list of the question_ids with their respective answers to the followup_questions. Question ids correspond to list names, answers correspond to list values.
`standardized_answer_levels`	A named list of standardized isco answer levels. Names in the list correspond to the type of isco standard, values correspond to the level itself. Possible standardized answer types are: "isco_skill_level" and "isco_supervisor_manager". These can be used instead of some followup questions if the information is available already from a different source. Please note that standardized answer levels are *not* available for all question types. For a list of options please take a look at the followup questions included in the auxco for example via `occupationMeasurement::auxco$followup_questions`.
`approximate_standardized_answer_levels`	(default TRUE) Follow up questions were designed to provide answer options that are not in conflict with suggestion_id. standardized_answer_levels can be in conflict with suggestion_id, and then no exact matches exist. With approximation, the answer option that is closest to the standardized_answer_levels provided, will be used.
`code_type`	Which type of codes should be returned. Multiple codes can be returned at the same time. Supported types of codes are "isco_08" and "kldb_10". Defaults to "isco_08" and "kldb_10".
`verbose`	(default TRUE) whether to return a `message` or not, detailing potential issues with the input provided.
`suggestion_type`	Which suggestion type is being used. Only auxco-based suggestion_types are supported.
`suggestion_type_options`	A list with options for generating suggestions. Supported options: - `datasets`: Pass specific datasets to be used whenn adding information to predictions e.g. use a specific version of the kldb or auxco. Supported datasets are: "auxco-1.2.x", "kldb-2010". By default the datasets bundled with this package are used.

Details

The interview situation may not allow to ask these follow-up questions. Some default, but suboptimal occupation code is returned if `followup_answers` is missing.

If `followup_answers` is missing or incomplete, one may wish to insert/infer the missing information by using `standardized_answer_levels`.

Value

A named list corresponding to the code_type(s) specified. Includes a `message` if `verbose = TRUE`

Examples

```
## Not run:
get_final_codes(
  # Führungsaufgaben mit Personalverantwortung  bei der Lebensmittelherstellung
  "9076",
  followup_answers = list(
    # The first answer option in the first followup question
    "Q9076_1" = 2
  )
)

# The same, but using standardized answer levels
get_final_codes(
  # Führungsaufgaben mit Personalverantwortung  bei der Lebensmittelherstellung
  "9076",
  standardized_answer_levels = list(
    # A response corresponding to the standard ISCO Level "supervisor"
    "isco_supervisor_manager" = "isco_supervisor"
  )
)

# Same example with approximate matching, due to conflicting information:
# External data suggest the person is not a supervisor, but the person still
# says she does supervisory tasks (Führungsaufgaben, as encoded in "9076").
# If approximate_standardized_answer_levels = TRUE (the default), the
# selected answer "9076" trumps the external data and we will code this
# person as a supervisor.
get_final_codes(
  # Führungsaufgaben mit Personalverantwortung  bei der Lebensmittelherstellung
  "9076",
  standardized_answer_levels = list(
    # A response corresponding to the standard ISCO Level "not manager nor supervisor"
    "isco_supervisor_manager" = "isco_not_supervising"
  )
)

## End(Not run)
```

get_followup_questions *Get potential follow-up questions for a suggestion.*

Description

Get potential follow-up questions for a suggestion.

Usage

```
get_followup_questions(
  suggestion_id,
  tense = "present",
  suggestion_type = "auxco-1.2.x",
  suggestion_type_options = list(),
  include_answer_codes = FALSE
)
```

Arguments

suggestion_id	Id of the suggestion
tense	Which tense i.e. time to use for questions & answers, this can be "present" or "past". Defaults to "present".
suggestion_type	Which suggestion type is being used. Only auxco-based suggestion_types are supported.
suggestion_type_options	A list with options for generating suggestions. Supported options: - `datasets`: Pass specific datasets to be used whenn adding information to predictions e.g. use a specific version of the kldb or auxco. Supported datasets are: "auxco-1.2.x", "kldb-2010". By default the datasets bundled with this package are used.
include_answer_codes	Whether answer options should contain information on the associated codes. Defaults to FALSE. (Only for internal use, use `get_final_codes()` to get codes)

Value

List of followup questions and their answer options.

Examples

```
## Not run:
# Get followup questions for "Post- und Zustelldienste"
get_followup_questions("1004")

## End(Not run)
```

`get_item_data` *Retrieve data for an item.*

Description

Each page in the questionnaire can have multiple items on it.

Usage

```
get_item_data(
  session,
  page_id,
  item_id = NULL,
  key = c("all", "question_text", "response_text", "response_id"),
  default = NULL
)
```

Arguments

`session`	The shiny session
`page_id`	The page for which to retrieve data.
`item_id`	The item for which to retrieve data. This *has* to be different for different items on the same page. Since most pages contain only a single question/item, `item_id` is set to "default" if missing.
`key`	The key for which to retrieve a value. (Optional) If no key is provided, the items's whole data will be returned.
`default`	A default value to return if the key or page is not present in the questionnaire data.

Value

The items's data.

See Also

`set_item_data()`

Examples

```
## Not run:
# Set up a "fake" shiny session to store data
session <- shiny::MockShinySession$new()
session$userData <- list(
  current_page_id = "other_page",
  questionnaire_data = list(
    example_page = list()
```

```
  )
)

# This code is expected to be run in e.g. run_before or run_after
# It doesn't really make sense to run this code outside
set_item_data(
  session = session,
  page_id = "example_page",
  question_text = "How are you?"
)

# This code is expected to be run in e.g. run_before
get_item_data(
  session = session,
  page_id = "example_page"
)

## End(Not run)
```

`get_job_suggestions` *Make coding suggestions based on a user's open-ended text input.*

Description

Given a `text` input, find up to `num_suggestions` possible occupation categories.

Usage

```
get_job_suggestions(
  text,
  suggestion_type = "auxco-1.2.x",
  num_suggestions = 5,
  suggestion_type_options = list(),
  aggregate_score_threshold = 0.02,
  item_score_threshold = 0,
  distinctions = TRUE,
 steps = list(simbased_wordwise = list(algorithm = algo_similarity_based_reasoning,
  parameters = list(sim_name = "wordwise")), simbased_substring = list(algorithm =
  algo_similarity_based_reasoning, parameters = list(sim_name = "substring"))),
  include_general_id = FALSE
)
```

Arguments

`text`	The raw text input from the user.
`suggestion_type`	Which type of suggestion to use / provide. Possible options are "auxco-1.2.x" and "kldb-2010".

num_suggestions
: The maximum number of suggestions to show. This is an upper bound and less suggestions may be returned. Defaults to 5.

suggestion_type_options
: A list with options for generating suggestions. Supported options: - datasets: Pass specific datasets to be used whenn adding information to predictions e.g. use a specific version of the kldb or auxco. Supported datasets are: "auxco-1.2.x", "kldb-2010". By default the datasets bundled with this package are used.

aggregate_score_threshold
: A single value or named list of thresholds between 0 and 1. If it is a list, each entry should correspond to one of the steps. If it is a single value, it will apply to all steps. Results from that step will only be returned if the sum of their scores is equal to or greater than the specified threshold. With a aggregate_score_threshold of 0 results will always be returned (if there are any).

item_score_threshold
: A threshold between 0 and 1 (usually very small, default 0). Results from any step will only be returned if they are greater than the specified threshold. Allows the removal of highly implausible suggestions.

distinctions
: Whether or not to add additional distinctions to similar occupational categories to the source code. Defaults to TRUE.

steps
: A list with the algorithms to use and their parameters. Each entry of the list should contain a nested list with two entries: algorithm (the algorithm's function itself) and parameters (the parameters to pass onto the algorithm). Each algorithm will also always have access to a default set of three parameters:

  - text_processed: The input text after preprocessing
  - suggestion_type: Which type of suggestion to output
  - num_suggestions: How many suggestions shall be returned These parameters must not be specified manually and will be provided automatically instead. Defaults to:

```
list(
  # try similarity "one word at most 1 letter different" first
  list(
    algorithm = algo_similarity_based_reasoning,
    parameters = list(
      sim_name = "wordwise",
      min_aggregate_prob = 0.535
    )
  ),
  # since everything else failed, try "substring" similarity
  list(
    algorithm = algo_similarity_based_reasoning,
    parameters = list(
      sim_name = "substring",
      min_aggregate_prob = 0.02
    )
  )
)
```

include_general_id
: Whether a general column, called "id" should always be returned. This will automatically contain the appropriate id for different suggestion_types i.e. for "auxco-1-2.x" it will contain the same data as the column "auxco_id".

Details

The procedure implemented here is, roughly speaking, as follows:

1. Predict categories from KldB 2010, including their scores. The first algorithm mentioned in `steps` is used (default: `algo_similarity_based_reasoning()`).
2. Convert the predicted KldB 2010 categories to `suggestion_type` (default: `auxco-1.2.x`, an n:m mapping, scores are mapped accordingly.). See internal function `convert_suggestions()` for details.
3. Remove predicted categories if their score is below `item_score_threshold` and only keep the `num_suggestions` top-ranked suggestions.
4. Start anew, trying the next algorithm in `steps`, if the the top-ranked suggestions have a low chance to be correct. (Technically, this happens if the summed score of the `num_suggestions` top-ranked suggestions is below `aggregate_score_threshold`.)
5. If `suggestion_type == "auxco-1.2.x"` and `distinctions == TRUE`, insert additional and (highly) similar categories or replace existing ones. See internal function `add_distinctions_auxco()`. Reorder and keep only the `num_suggestions` top-ranked suggestions. Auxco categories which were added during this step can be identified by their scores: It equals 0.05 for categories with high similarity and 0.005 for categories with medium similarity.

Value

A data.table with suggestions or NULL if no suggestions were found.

Examples

```
## Not run:
if (interactive()) {
  get_job_suggestions("Koch")
}

if (interactive()) {
  get_job_suggestions("Schlosser")
}

## End(Not run)
```

get_responses *Convenience function to aggregate all saved results_overview files.*

Description

Expects data to be saved as files.

Usage

```
get_responses(app_settings = create_app_settings(save_to_file = TRUE))
```

Arguments

app_settings	The app_settings configuration, should be the same as used in app().

Value

A combined data.table of user data (based on results_overview) or NULL if there are no files.

Examples

```
## Not run:
app_settings <- create_app_settings(save_to_file = TRUE)
if (interactive()) {
  get_responses(app_settings = app_settings)
}

## End(Not run)
```

isco_08_en *Categories of the The International Standard Classification of Occupations - ISCO-08*

Description

Categories from the International Standard Classification of Occupations - ISCO-08. ISCO-08 is a hierarchical classification, consisting of 10 (1-digit) major groups, 43 (2-digit) sub-major groups, 130 (3-digit) minor groups, and 436 (4-digit) unit groups, all of them included in this data set.

Usage

```
isco_08_en
```

Format

A data frame with 619 rows and 3 variables:

`code` character. Unique ISCO-08 identifier / code.

`label` character. Short label / title for the category.

`description` character. Detailed description of the category.

Details

Source: https://esco.ec.europa.eu This service uses the ESCO classification of the European Commission. The descriptions used here are taken from the ESCO classification (v1.1, Occupations pillar) of the European Commission, which is based on ISCO-08.

More information on the ISCO-08: https://isco-ilo.netlify.app/en/isco-08/, https://www.ilo.org/public/english/bureau/stat/isco

---

`load_auxco` *Load AuxCO from a directory of CSV files*

Description

This function loads the Auxiliary Classification of Occupations (AuxCO) by reading CSVs from the specified directory, while loading e.g. ids in the correct format. Data is loaded into a named list matching the format expected by other functions in this package.

Usage

```
load_auxco(dir, add_explanations = TRUE)
```

Arguments

`dir`	The path to the directory which holds the CSVs.
`add_explanations`	Whether explanations should be added to some of the harder to understand task descriptions. Defaults to TRUE.

Details

This package also includes an already loaded version of the auxco, which can be used straight away *without* calling this function.

Value

A list with multiple data.tables.

See Also

https://github.com/occupationMeasurement/auxiliary-classification, auxco

Examples

```
## Not run:
# This function expects the CSV files from
# https://github.com/occupationMeasurement/auxiliary-classification/releases/
# to be there.
path_to_auxco <- "auxco"
if (dir.exists(path_to_auxco)) {
  load_auxco(path_to_auxco)
}

## End(Not run)
```

`load_kldb_raw` *Clean & Load KldB 2010 dataset.*

Description

Use load_kldb_raw() to load the whole dataset.

Usage

```
load_kldb_raw(
  cache_dir = getOption("occupationMeasurement.cache_dir", tempdir())
)

load_kldb(cache_dir = getOption("occupationMeasurement.cache_dir", tempdir()))
```

Arguments

`cache_dir`	The path to the directory where the downloaded data should be stored. We recommend setting this to "cache" to store data in the working directory. This will prevent reloading the data time and time again. This can be set globally via `options(occupationMeasurement.cache_dir = "cache")`.

Details

Source: https://www.klassifikationsserver.de/klassService/index.jsp?variant=kldb2010

More information on the KldB 2010: https://statistik.arbeitsagentur.de/DE/Navigation/Grundlagen/Klassifikationen/Klassifikder-Berufe/KldB2010-Fassung2020/KldB2010-Fassung2020-Nav.html The KldB 2010 has been revised in 2020. These changes have not been implemented here yet.

Value

A cleaned / slimmed version of the KldB 2010.

Functions

- `load_kldb_raw()`: Load raw KldB 2010 dataset.

Examples

```
## Not run:
# We recommend using a non-temporary directory for caching, so data is
# downloaded only once and not time and time again
cache_dir <- tempdir()
# Note: The dataset will be downloaded from the internet
# Load the cleaned dataset
load_kldb(cache_dir = cache_dir)
# Load the raw dataset
load_kldb_raw(cache_dir = cache_dir)

## End(Not run)
```

`new_page` *Create a new questionnaire page.*

Description

Each page corresponds to a page within the app/questionnaire.

Usage

```
new_page(
  page_id,
  render,
  condition = NULL,
  run_before = NULL,
  render_before = NULL,
  render_after = NULL,
  run_after = NULL
)
```

Arguments

`page_id`	A unique string identifiying this page. (Required) This will be used to store data.
`render`	Function to render the page. (Required) It is expected, that the function returns a list of shiny tags. Its output will be combined with `render_before` and `render_after`. This function has access to the shiny `session` and the `run_before_output`.
`condition`	Function to check whether the page should be shown. When this function returns `TRUE`, the page will be shown upon navigating there, if it returns `FALSE` it will be skipped. Defaults to show the page. This function has access to the shiny `session`.
`run_before`	Function that prepares data to render the page. Called immediately after condition (if `condition` returned `TRUE`). Whatever run_before returns is available in render, render_before and render_after as `run_before_output`. This function has access to the shiny `session`.

render_before	Called exactly like render. Output will be added just *before* the output from render. Mainly used to add additional outputs to existing pages.
render_after	Called exactly like render. Output will be added just *after* the output from render. Mainly used to add additional outputs to existing pages.
run_after	Function that handles the user input when they leave the page. This function has access to the shiny session and shiny input.

Details

Pages are rendered by calling the different life-cycle functions one after another. The order in which they are called is as follows:

1. condition (session) Only if this evaluated to TRUE, continue.
2. run_before (session)
3. render_before (session, run_before_output, input, output)
4. render (session, run_before_output, input, output)
5. render_after (session, run_before_output, input, output) The outputs from render_before, render & render_after are stitched together to produce the final HTML output of the page.
6. run_after (session, input, output) Run when the user leaves the page (=clicks the next button). Any user input has to be handled here. For each question asked, one will typically call set_item_data() to save the collected data internally.

Each of the life-cycle functions above is annotated with the paramaters it has access to. session, input and output are passed directly from shiny and correspond to the objects made available by shiny::shinyServer(), run_before_output is available for convenience and corresponds to whatever is returned by run_before.

Some side-effects occur:

- occupationMeasurement:::init_page_data is called before 1. run_before. It sets up an internal representation of the page data to be saved.
- occupationMeasurement:::finalize_data is called before 6. run_before.
- occupationMeasurement:::save_page_data is called after 6. run_before. It saves the responses on a hard drive, i.e. it appends the responses from this page to table_name == "answers". See the vignette and create_app_settings() for details.

Use of render_before, render_after is discouraged if not necessary, as these two life-cycle functions have only been added to allow for easier modification and extension of existing pages.

Value

A new page object.

Examples

```
## Not run:
very_simple_page <- new_page(
  page_id = "example",
```

```
  render = function(session, run_before_output, input, output, ...) {
    list(
      shiny::tags$h1("My test page"),
      button_previous(),
      button_next()
    )
  }
)

# Example where we also save some data
page_that_saves_two_items <- new_page(
  page_id = "questions_1_and_2",
  render = function(session, run_before_output, page, input, output, ...) {
    list(
      shiny::tags$h1("Questions"),
      shiny::textAreaInput(
        inputId = "day_freetext",
        label = "How was your day? Please give a detailed answer.",
        value = get_item_data(
          session = session, page_id = page$page_id,
          item_id = "day_freetext",
          key = "response_text"
        )
      ),
      shiny::tags$p("How would you rate your day? On a scale of 1 - 4"),
      radioButtons(
        inputId = "day_radio",
        label = NULL,
        width = "100%",
        choices = list(One = 1, Two = 2, Three = 3, Four = 4),
        selected = get_item_data(
          session = session,
          page_id = page$page_id,
          item_id = "day_radio",
          key = "response_id",
          default = character(0)
        )
      ),
      button_previous(),
      button_next()
    )
  },
  run_after = function(session, page, input, ...) {
    set_item_data(
      session = session,
      page_id = page$page_id,
      item_id = "day_freetext",
      response_text = input$day_freetext
    )
    set_item_data(
      session = session,
      page_id = page$page_id,
      item_id = "day_radio",
```

```
      response_id = input$day_radio
    )
  }
)

questionnaire_that_saves_two_items <- list(
  page_that_saves_two_items,
  # So we have a next page to go to
  very_simple_page
)

if (interactive()) {
  app(questionnaire = questionnaire_that_saves_two_items)
}

## End(Not run)
```

---

page_choose_one_option *Show a page with multiple radio button options where once can be picked.*

Description

Show a page with multiple radio button options where once can be picked.

Usage

```
page_choose_one_option(
  page_id,
  question_text = "Please pick one of the following options",
  list_of_choices = list(One = 1, Two = 2, Three = 3),
  choice_labels = NULL,
  next_button = TRUE,
  previous_button = TRUE,
  run_before = NULL,
  run_after = NULL,
  ...
)
```

Arguments

page_id	A unique string identifiying this page. Used to store data.
question_text	The question / text to display. This can be either a string, which will simply be displayed or a function to dynamically determine the question_text.
list_of_choices	A list of answering options. This can either be just a simple list of values or a named list with the names corresponding to what the user sees and the values

corresponding to the actually saved values. e.g. with `list(One = 1, Two = 2, Three = 3)` people will see One, Two, ... and numbers 1, 2, ... will be saved under `response_id`. If you want to use more complex choice names than jsut strings (i.e. HTML), you can also use the choice_labels option for that.

`choice_labels` List or vector of only the choice names to be shown. This has to be matched by an equal-length vector in list_of_choices.

`next_button` Whether to show the button to navigate to the next page? Defaults to TRUE.

`previous_button`
Whether to show the button to navigate to the preivous page? Defaults to TRUE.

`run_before` Similar to `run_before` in `new_page()`, passed explicitly here as this page adds some of its own code to `run_before`.

`run_after` Similar to `run_after` in `new_page()`, passed explicitly here as this page adds some of its own code to `run_after`.

`...` Other parametrs are passed on to `new_page()`

Value

A page object.

See Also

`new_page()`

Examples

```
## Not run:
one_page_questionnaire <- list(
  page_choose_one_option(
    "test_page_radio",
   question_text = "Hello there! Please pick your favorite number from the options below:",
    list_of_choices = list(One = 1, Two = 2, Three = 3)
  ),
  page_final()
)
if (interactive()) {
  app(questionnaire = one_page_questionnaire)
}

## End(Not run)
```

page_feedback *Page to receive feedback on how well the chosen suggestion fits*

Description

Page to receive feedback on how well the chosen suggestion fits

Usage

```
page_feedback(is_interview = FALSE, ...)
```

Arguments

is_interview	Should the page show slightly different / additional instructions and answer options for an interview that is conducted by another person? Defaults to FALSE.
...	All additional parameters are passed first passed on to page_choose_one_option() and then new_page().

Value

A page object.

Examples

```
## Not run:
my_questionnaire <- list(
  page_first_freetext(),
  page_select_suggestion(),
  page_feedback()
)
if (interactive()) {
  app(questionnaire = my_questionnaire)
}

## End(Not run)
```

page_final *A final page, showing instructions to close the window.*

Description

This page saves data in results_overview and marks the questionnaire as complete.

Usage

```
page_final(...)
```

Arguments

...	All additional parameters are passed to new_page()

Value

A page object.

See Also

new_page()

Examples

```
## Not run:
my_questionnaire <- list(
  page_final()
)
if (interactive()) {
  app(questionnaire = my_questionnaire)
}

## End(Not run)
```

---

page_first_freetext *The first freetext question to show.*

Description

Here, the description of the job can be entered in an open freetext field and suggestions will be generated based on the input.

Usage

```
page_first_freetext(
  is_interview = FALSE,
  aggregate_score_threshold = 0.535,
  ...
)
```

Arguments

is_interview	Should the page show slightly different / additional instructions and answer options for an interview that is conducted by another person? Defaults to FALSE.
aggregate_score_threshold	The total sum of the scores of the suggestions has to be higher than this threshold for suggestions to be shown. The parameter is passed on to get_job_suggestions().
...	All additional parameters are passed to new_page()

Value

A page object.

See Also

new_page()

Examples

```
## Not run:
my_questionnaire <- list(
  page_first_freetext(),
  page_second_freetext(),
  page_select_suggestion(),
  page_none_selected_freetext(),
  page_followup(1),
  page_followup(2)
)
if (interactive()) {
  app(questionnaire = my_questionnaire)
}

## End(Not run)
```

---

page_followup *Show potential followup questions to the user.*

Description

To disambiguate between similar occupations. Depending on the suggestion, multiple followup questions can be shown.

Usage

```
page_followup(index, is_interview = FALSE, ...)
```

Arguments

index	The index of the followup question (1-based). To show the first followup question (if there are any) use page_followup(index = 1), to show a potential second followup question use page_followup(index = 2). For example questionnaire_web_survey() uses ..., page_followup(index = 1), page_followup(index = 2), ...
is_interview	Should the page show slightly different / additional instructions and answer options for an interview that is conducted by another person? Defaults to FALSE.
...	All additional parameters are passed to new_page()

Value

A page object.

See Also

new_page()

Examples

```
## Not run:
my_questionnaire <- list(
  page_first_freetext(),
  page_second_freetext(),
  page_select_suggestion(),
  page_none_selected_freetext(),
  page_followup(1),
  page_followup(2)
)
if (interactive()) {
  app(questionnaire = my_questionnaire)
}

## End(Not run)
```

---

page_freetext *Show a page with a text field where free text can be entered.*

Description

Show a page with a text field where free text can be entered.

Usage

```
page_freetext(
  page_id,
  question_text = "Please enter your answer in the box below",
  is_interview = FALSE,
  no_answer_checkbox = TRUE,
  next_button = TRUE,
  previous_button = TRUE,
  trigger_next_on_enter = TRUE,
  render_question_text = TRUE,
  run_before = NULL,
  run_after = NULL,
  ...
)
```

Arguments

page_id	A unique string identifiying this page. Used to store data.
question_text	The question / text to display. This can be either a string, which will simply be displayed or a function to dynamically determine the question_text.
is_interview	Should the page show slightly different / additional instructions and answer options for an interview that is conducted by another person? Defaults to FALSE.
no_answer_checkbox	Whether to provide a checkbox to denote that no answer has been provided.
next_button	Whether to show the button to navigate to the next page? Defaults to TRUE.
previous_button	Whether to show the button to navigate to the preivous page? Defaults to TRUE.
trigger_next_on_enter	Whether the next button is triggered when one presses enter. Defaults to TRUE. There are known issues with IE11.
render_question_text	Whether the question text should be displayed? Only set this to FALSE, if you wish to change the rendering of the question_text by e.g. using render_before. Defaults to TRUE.
run_before	Similar to run_before in new_page(), passed explicitly here as this page adds some of its own code to run_before.
run_after	Similar to run_after in new_page(), passed explicitly here as this page adds some of its own code to run_after.
...	Other parametrs are passed on to new_page()

Value

A page object.

See Also

new_page()

Examples

```
## Not run:
page_freetext(
  "test_page_freetext",
  question_text = "Hello there! Please fill in your name below:",
  no_answer_checkbox = TRUE
)

## End(Not run)
```

page_none_selected_freetext *An additional freetext page to show when no suggestion has been selected.*

Description

An additional freetext page to show when no suggestion has been selected.

Usage

```
page_none_selected_freetext(is_interview = FALSE, ...)
```

Arguments

is_interview	Should the page show slightly different / additional instructions and answer options for an interview that is conducted by another person? Defaults to FALSE.
...	All additional parameters are passed to new_page()

Value

A page object.

See Also

new_page()

Examples

```
## Not run:
my_questionnaire <- list(
  page_first_freetext(),
  page_second_freetext(),
  page_select_suggestion(),
  page_none_selected_freetext(),
  page_followup(1),
  page_followup(2)
)
if (interactive()) {
  app(questionnaire = my_questionnaire)
}

## End(Not run)
```

page_results *Page showing the user's results*

Description

This page is only meant for demonstration purposes. Users can see what they entered and which code was being saved. The page is only included in the questionnaire_demo(), but not in the other questionnaire templates.

Usage

```
page_results(...)
```

Arguments

...	All additional parameters are passed to new_page()

Value

A page object.

See Also

new_page()

Examples

```
## Not run:
my_questionnaire <- list(
  page_first_freetext(),
  page_second_freetext(),
  page_select_suggestion(),
  page_none_selected_freetext(),
  page_followup(1),
  page_followup(2),
  page_results()
)
if (interactive()) {
  app(questionnaire = my_questionnaire)
}

## End(Not run)
```

page_second_freetext *An optional, second free text question if the first didn't yield suggestions.*

Description

If the first freetext question didn't provide satisfactory results, ask for more details and try again.

Usage

```
page_second_freetext(
  combine_input_with_first = TRUE,
  is_interview = FALSE,
  aggregate_score_threshold = 0.02,
  ...
)
```

Arguments

combine_input_with_first	Should input be combined with the previous question?
is_interview	Should the page show slightly different / additional instructions and answer options for an interview that is conducted by another person? Defaults to FALSE.
aggregate_score_threshold	The total sum of the scores of the suggestions has to be higher than this threshold for suggestions to be shown. The parameter is passed on to get_job_suggestions().
...	All additional parameters are passed to new_page()

Value

A page object.

See Also

new_page()

Examples

```
## Not run:
my_questionnaire <- list(
  page_first_freetext(),
  page_second_freetext(),
  page_select_suggestion(),
  page_none_selected_freetext(),
  page_followup(1),
  page_followup(2)
)
```

```
if (interactive()) {
  app(questionnaire = my_questionnaire)
}

## End(Not run)
```

`page_select_suggestion` *Display the generated suggestions for the user to pick one.*

Description

Display the generated suggestions for the user to pick one.

Usage

```
page_select_suggestion(is_interview = FALSE, ...)
```

Arguments

`is_interview`	Should the page show slightly different / additional instructions and answer options for an interview that is conducted by another person? Defaults to FALSE.
`...`	All additional parameters are passed to `new_page()`

Value

A page object.

See Also

`new_page()`

Examples

```
## Not run:
my_questionnaire <- list(
  page_first_freetext(),
  page_second_freetext(),
  page_select_suggestion(),
  page_none_selected_freetext(),
  page_followup(1),
  page_followup(2)
)
if (interactive()) {
  app(questionnaire = my_questionnaire)
}

## End(Not run)
```

page_welcome *Welcome Page (optional)*

Description

Providing an introduction and greeting participants.

Usage

```
page_welcome(
  title = "Herzlich Willkommen zum Modul zur automatischen Berufskodierung!",
  ...
)
```

Arguments

title	The heading with which to greet participants.
...	All additional parameters are passed to new_page()

Value

A page object.

See Also

new_page()

Examples

```
## Not run:
my_questionnaire <- list(page_welcome)
if (interactive()) {
  app(questionnaire = my_questionnaire)
}

## End(Not run)
```

preprocess_string *Preprocess a string, removing special characters and handling abbreviations.*

Description

Replace some common characters / character sequences (e.g., Ä, Ü, "DIPL.-ING.") with their uppercase equivalents and removes punctuation, empty spaces and the word "Diplom".

Usage

```
preprocess_string(verbatim, lang = "de")
```

Arguments

verbatim	The character vector to process.
lang	The language the text is in. Currently only German is supported. Defaults to "de" (German).

Details

charToRaw() helps to find UTF-8 characters.

Value

The same character vector after processing

Examples

```
## Not run:
preprocess_string(c(
  "Verkauf von B\u00fcchern, Schreibwaren",
 "Fach\u00e4rztin f\u00fcr Kinder- und Jugendmedizin im \u00f6ffentlichen Gesundheitswesen",
  "Industriemechaniker",
  "Dipl.-Ing. - Agrarwirtschaft (Landwirtschaft)"
))

## End(Not run)
```

`pretrained_models` *Pretrained ML models to be used with the package.*

Description

Pretrained ML models to be used with the package.

Usage

```
pretrained_models
```

Format

A nested list with pretrained machine learning models:

`similarity_based_reasoning` list. Contains pretrained models to be used with `algo_similarity_based_reasoning()`.

`similarity_based_reasoning$wordwise` list. Contains the pretrained model to be used for providing suggestions using full wordwise matching.

`similarity_based_reasoning$substring` list. Contains the pretrained model to be used for providing suggestions using substring matching.

This training data always predicts a 5-digit code from the 2010 German Classification of Occupations, with some exceptions: -0004 stands for 'Not precise enough/uncodable', -0006 stands for 'Multiple Jobs', -0012 stands for 'Blue-collar workers', -0019 stands for 'Volunteer/Social Service', and -0030 stands for 'Student assistant'.

Source

Data from the following surveys were pooled:

Antoni, M., Drasch, K., Kleinert, C., Matthes, B., Ruland, M. and Trahms, A. (2010): Arbeiten und Lernen im Wandel * Teil 1: Überblick über die Studie, FDZ-Methodenreport 05/2010, Forschungsdatenzentrum der Bundesagentur für Arbeit im Institut für Arbeitsmarkt- und Berufsforschung, Nuremberg.

Rohrbach-Schmidt, D., Hall, A. (2013): BIBB/BAuA Employment Survey 2012, BIBB-FDZ Data and Methodological Reports Nr. 1/2013. Version 4.1, Federal Institute for Vocational Education and Training (Research Data Centre), Bonn.

Lange, C., Finger, J., Allen, J., Born, S., Hoebel, J., Kuhnert, R., Müters, S., Thelen, J., Schmich, P., Varga, M., von der Lippe, E., Wetzstein, M., Ziese, T. (2017): Implementation of the European Health Interview Survey (EHIS) into the German Health Update (GEDA), Archives of Public Health, 75, 1–14.

Hoffmann, R., Lange, M., Butschalowsky, H., Houben, R., Schmich, P., Allen, J., Kuhnert, R., Schaffrath Rosario, A., Gößwald, A. (2018): KiGGS Wave 2 Cross-Sectional Study—Participant Acquisition, Response Rates and Representativeness, Journal of Health Monitoring, 3, 78–91. (only wave 2)

Trappmann, M., Beste, J., Bethmann, A., Müller, G. (2013): The PASS Panel Survey after Six Waves, Journal for Labour Market Research, 46, 275–281. (only wave 10)

Job titles were taken from the following publication:

Bundesagentur für Arbeit (2019). Gesamtberufsliste der Bundesagentur für Arbeit. Stand: 03.01.2019. https://download-portal.arbeitsagentur.de/files/.

Basically, leaving some data anonymization steps aside, we count for each job title from the Gesamtberufsliste (and some additional titles/texts) how many responses from all surveys are similar to this job title, separately for each coded category. Similarity is calculated in two ways, implying that we obtain two different counts: SubstringSimilarity refers to situations where the job title from the Gesamtberufsliste is a substring of the verbal answer; WordwiseSimilarity refers to situations where a word from the verbal answer is identical to a job title from the Gesamtberufsliste, except that one character from this word is allowed to change (Levenshtein distance). These counts are available as two separate files in the data-raw/training-data/ directory of this package. The algorithm to create these counts is available inside an R-package at https://github.com/malsch/occupationCoding, along with further documentation.

`train_similarity_based_reasoning()` is then used to train the ML models. See data-raw/pretrained_models.R for the raw counts and further details.

See Also

`algo_similarity_based_reasoning()`, `train_similarity_based_reasoning()`, https://github.com/malsch/occupation

`questionnaire_demo` *A demo questionnaire with additional explanations*

Description

View the function's code itself to see the used pages.

Usage

```
questionnaire_demo(show_feedback_page = TRUE)
```

Arguments

`show_feedback_page`
: Show the `page_feedback()` to evaluate the fit of the chosen suggestion.

Details

Note, that this function has more complex code to create the additional pages.

Value

A questionnaire for `app()` i.e. a list of pages.

Examples

```
## Not run:
# Inspect the code to create the questionnaire_demo
print(questionnaire_demo)

if (interactive()) {
  # Run the app with the questionnaire_demo
  app(questionnaire = questionnaire_demo())
}

## End(Not run)
```

`questionnaire_interviewer_administered` *A questionnaire for interviewer-administered surveys*

Description

A questionnaire for Computer-assisted Interviewing (CAI), i.e. telephone interviewing or personal interviewing. In both modes, interviewer asks questions to an interviewee.

Usage

```
questionnaire_interviewer_administered(show_feedback_page = TRUE)
```

Arguments

`show_feedback_page`
: Show the `page_feedback()` to evaluate the fit of the chosen suggestion.

Details

View the function's code to see the used pages. This function is meant as a template that can be changed to meet your requirements.

Value

A questionnaire for `app()` i.e. a list of pages.

Examples

```
## Not run:
# Inspect the code to create the questionnaire_interviewer_administered
print(questionnaire_interviewer_administered)

if (interactive()) {
  # Run the app with the questionnaire_interviewer_administered
```

```
  app(questionnaire = questionnaire_interviewer_administered())
}

## End(Not run)
```

`questionnaire_web_survey` *A web survey which participants can navigate themselves.*

Description

The basic default questionnaire. View the function's code to see the used pages. This function is meant as a template that can be changed to meet your requirements.

Usage

```
questionnaire_web_survey(show_feedback_page = TRUE)
```

Arguments

`show_feedback_page` Show the `page_feedback()` to evaluate the fit of the chosen suggestion.

Value

A questionnaire for `app()`, i.e. a list of pages.

Examples

```
## Not run:
# Inspect the code to create the questionnaire_web_survey
print(questionnaire_web_survey)

if (interactive()) {
  # Run the app with the questionnaire_web_survey
  app(questionnaire = questionnaire_web_survey())
}

if (interactive()) {
  # This is used by default within app
  app()
}

## End(Not run)
```

set_item_data *Set / save data for an item.*

Description

There can be multiple items on any given page. Items can be different questions, or multiple variables that need to be saved from a single question. The `question_text` is typically saved in `run_before` and the reply (`response_text` and/or `response_id`) is typically saved in `run_after`.

Usage

```
set_item_data(
  session,
  page_id,
  item_id = NULL,
  question_text = NULL,
  response_text = NULL,
  response_id = NULL
)
```

Arguments

`session`	The shiny session
`page_id`	The page for which to retrieve data.
`item_id`	The item for which to set/update data. This *has* to be different for different items on the same page. Since most pages contain only a single question/item, `item_id` is set to "default" if missing.
`question_text`	The question's text. (optional)
`response_text`	The user's response in text form. (optional)
`response_id`	The user's response as an id from a set of choices. (optional)

Value

nothing

See Also

`get_item_data()`

Examples

```
## Not run:
# Set up a "fake" shiny session to store data
session <- shiny::MockShinySession$new()
session$userData <- list(
```

```
  current_page_id = "other_page",
  questionnaire_data = list(
    example_page = list()
  )
)

# This code is expected to be run in e.g. run_before or run_after
# It doesn't really make sense to run this code outside
set_item_data(
  session = session,
  page_id = "example_page",
  question_text = "How are you?"
)

set_item_data(
  session = session,
  page_id = "example_page",
  response_id = 3,
  response_text = "I'm doing great! (response_id = 3)"
)

## End(Not run)
```

`train_similarity_based_reasoning` *Train Similarity Based Probability Model with anonymized training data*

Description

This function requires the mvtnorm package.

Usage

```
train_similarity_based_reasoning(
  anonymized_data,
  num_allowed_codes = 1291,
  coding_index_w_codes,
  coding_index_without_codes = NULL,
  preprocessing = list(stopwords = NULL, stemming = NULL, strPreprocessing = TRUE,
    removePunct = FALSE),
  dist_type = c("wordwise", "substring", "fulltext"),
  dist_control = list(method = "osa", weight = c(d = 1, i = 1, s = 1, t = 1)),
  threshold = c(max = 3, use = 1),
  simulation_control = list(n.draws = 250, check_normality = FALSE)
)
```

Arguments

`anonymized_data`
: `surveyCountsSubstringSimilarity` or `surveyCountsWordwiseSimilarity`

`num_allowed_codes`
: the number of allowed codes in the target classification. There are 1286 categories in the KldB 2010 plus 5 special codes in both anonymized training data sets, so the default value is 1291.

`coding_index_w_codes`
: a data.table with columns

  bezMale a character vector, contains masculine job titles from the coding index.

  bezFemale a character vector, contains feminine job titles from the coding index.

  Code a character vector with associated classification codes.

`coding_index_without_codes`
: (not used, but automatically determined) Any words from `anonymized_data$dictString` that are not found within `coding_index_w_codes` belong into this character vector.

`preprocessing`
: a list with elements

  stopwords a character vector, use `tm::stopwords("de")` for German stopwords. Only used if `dist_type = "wordwise"`.

  stemming `NULL` for no stemming and `"de"` for stemming using the German porter stemmer. Do not use unless the job titles in `coding_index_w_codes` were stemmed.

  strPreprocessing `TRUE` if `preprocess_string` shall be used.

  removePunct `TRUE` if `removePunctuation` shall be used.

`dist_type`
: How to calculate similarity between entries from both coding_indices and verbal answers from the survey? Three options are currently supported. Since we use the `stringdist`-function excessively, one could easily extend the functionality of this procedure to other distance metrics.

  dist_type = "fulltext" Uses the `stringdist`-function directly after preprocessing to calculate distances. (the simplest approach but least useful.)

  dist_type = "substring" An entry from the coding index and a verbal answer are similar if the entry from the coding index is a substring of the verbal answer.

  dist_type = "wordwise" After preprocessing, split the verbal answer into words. Then calculate for each word separately the the similarity with entries from the coding index, using `stringdist`. Not the complete verbal answer but only the words (0 or more) that have highest similarity are then used to determine similarity with entries from the coding index.

`dist_control`
: If `dist_type = "fulltext"` or `dist_type = "wordwise"` the entries from this list will be passed to `stringdist`. Currently only two possible entries are supported (method = "osa", weight = c(d = 1, i = 1, s = 1, t = 1) is recommended), but one could easily extend the functionality.

`threshold`
: A numeric vector with two elements. If `dist_type = "fulltext"` or `dist_type = "wordwise"`, the threshold determines up to which distance a verbal answer

and an entry from the coding index are similar. The second number actually gets used. The first number is only used to speed up similarity calculations. It should be identical or larger than the second number.

`simulation_control`

a list with two components,

n.draws Number of draws from the posterior distribution to determine posterior predictive probabilities. The larger, the more precise the results will be.

check_normality We would like that the hyperprior distribution is normal. Set check_normality to TRUE to do some diagnostics about this.

Value

a list with components

prediction.datasets$modelProb Contains all entries from the coding index. dist = "official" if the entry stems from coding_index_w_codes and dist = selfcreated if the entry stems from coding_index_without_codes. `string.prob` is used for weighting purposes (model averaging) if a new verbal answer is similar to multiple strings. `unobserved.mean.theta` gives a probability (usually very low) for any category that was not observed in the training data together with this string.

prediction.datasets$categoryProb `mean.theta` is the probability for `code` given that an incoming verbal answer is similar to `string`. Only available if this code was at least a single time observed with this string (Use `unobserved.mean.theta` otherwise).

num_allowed_codes Number of categories in the classification.

preprocessing The input parameter stored to replicate preprocessing with incoming data.

dist_type The input parameter stored to replicate distance calculations with incoming data.

dist_control The input parameter stored to replicate distance calculations with incoming data.

threshold The input parameter stored to replicate distance calculations with incoming data.

simulation_control The input parameters controlling the Monte Carlo simulation.

References

Schierholz, Malte (2019): New methods for job and occupation classification. Dissertation, Mannheim. https://madoc.bib.uni-mannheim.de/50617/, pp. 206-208 and p. 268, pp. 308-320

https://github.com/malsch/occupationCoding (function trainSimilarityBasedReasoning2 is implemented here)

See Also

pretrained_models, which were created using this function.

Index