Package: transforEmotion (via r-universe)

October 29, 2026

Title Transformer Emotion Analysis for Text, Image, Video, and Speech

Version 0.1.9

Date 2026-10-26

Author Alexander Christensen [aut] (<https://orcid.org/0000-0002-9798-7037>), Hudson Golino [aut] (<https://orcid.org/0000-0002-1601-1447>), Aleksandar Tomašević [aut, cre] (<https://orcid.org/0000-0003-4863-6051>)

Maintainer Aleksandar Tomašević <atomashevic@gmail.com>

Description Zero-shot emotion classification for text, images, video and speech, and tools to find faces and code them by hand. Text is scored with 'Hugging Face' transformer pipelines (default 'cross-encoder/nli-distilroberta-base'). Images and video are scored with 'OpenAI CLIP'; faces can be limited to those found by 'OpenCV YuNet'. Speech segments are scored with 'LAION CLAP'. Frame scores can be calibrated and combined into posteriors, and detected faces or other items can be turned into a local coding task served from R. Any other zero-shot model can be selected by its name on <https://huggingface.co/models>. For details see Tomašević, Golino, and Christensen (2026) <doi:10.5117/CCR2026.2.2.TOMA>.

Depends R (>= 3.5.0)

License GPL (>= 3.0)

Copyright See file inst/COPYRIGHTS

Encoding UTF-8

Imports dplyr, googledrive, Matrix, methods, pbapply, reticulate (>= 1.45.0), textdata, jsonlite, ggplot2, reshape2, httr, httpuv (>= 1.6.0)

Suggests IRdisplay, knitr, LSAfun, markdown, remotes, rmarkdown, testthat (>= 3.0.0), withr

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

Config/pak/sysreqs
cmake make libicu-dev libpng-dev libuv1-dev libssl-dev python3 libx11-dev zlib1g-dev

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

Date/Publication 2026-10-29 19:24:18 UTC

RemoteUrl https://github.com/atomashevic/transforemotion

RemoteRef HEAD

RemoteSha 0b684247fd87dac7dfeed84bcae38f4858ed7014

Contents

transforEmotion-package . . . 3
.init_builtin_models . . . 4
.vision_model_registry . . . 5
add_vision_model . . . 5
annotate_faces . . . 7
annotation_agreement . . . 8
annotation_progress . . . 9
annotation_questions . . . 10
annotation_task . . . 11
as_rag_table . . . 13
calculate_moving_average . . . 14
calibrate_scores . . . 15
check_findingemo_quality . . . 17
consensus_labels . . . 18
delete_transformer . . . 19
detect_faces . . . 19
dlo_dynamics . . . 22
download_findingemo_data . . . 23
draw_faces . . . 25
emotions . . . 26
emoxicon_scores . . . 27
emphasize . . . 28
evaluate_emotions . . . 29
face_scores . . . 31
generate_observables . . . 32
generate_q . . . 33
get_vision_model_config . . . 34
image_scores . . . 34
image_scores_dir . . . 36
is_vision_model_registered . . . 37
list_vision_models . . . 37
load_findingemo_annotations . . . 38
map_discrete_to_vad . . . 40
map_to_emo8 . . . 42
MASS_mvrnorm . . . 43
neo_ipip_extraversion . . . 44

nlp_scores . . . 44
parse_rag_json . . . 47
plot.emotion_evaluation . . . 48
plot_sim_emotions . . . 48
posterior_scores . . . 49
prepare_findingemo_evaluation . . . 51
print.emotion_evaluation . . . 52
punctuate . . . 53
python_requirements . . . 54
rag . . . 55
rag_json_utils . . . 59
rag_sentemo . . . 59
read_annotations . . . 61
read_face_selection . . . 62
register_retriever . . . 63
register_vision_model . . . 63
remove_vision_model . . . 65
select_faces . . . 66
sentence_similarity . . . 67
setup_cache . . . 68
setup_gpu_modules . . . 70
setup_miniconda . . . 71
setup_modules . . . 71
setup_popular_models . . . 72
show_vision_models . . . 73
simulate_video . . . 74
speech_scores . . . 75
stop_words . . . 77
summary.emotion_evaluation . . . 78
te_cleanup_default_venv . . . 78
tinytrolls . . . 79
transformer_scores . . . 79
vad_scores . . . 83
validate_rag_json . . . 86
validate_rag_predictions . . . 87
video_scores . . . 88
view_faces . . . 90

Index **94**

`transforEmotion-package`

transforEmotion–package

Description

Zero-shot emotion classification for text, images, video and speech, and tools to find faces and code them by hand. Text is scored with Hugging Face transformer pipelines (default cross-encoder/nli-distilroberta-base). Images and video are scored with OpenAI CLIP; faces can be limited to those found by OpenCV YuNet. Speech segments are scored with LAION CLAP. Frame scores can be calibrated and combined into posteriors, and detected faces or other items can be turned into a local coding task served from R. Any other zero-shot model can be selected by its name on Hugging Face.

Author(s)

Alexander P. Christensen <alexpaulchristensen@gmail.com>, Hudson Golino <hfg9s@virginia.edu> and Aleksandar Tomasevic <atomashevic@gmail.com>

References

Tomašević, A., Golino, H., & Christensen, A. P. (2026). transforEmotion: An open-source R package for emotion analysis using transformer-based generative AI models. *Computational Communication Research*, *8*(2). doi:10.5117/CCR2026.2.2.TOMA

Yin, W., Hay, J., & Roth, D. (2019). Benchmarking zero-shot text classification: Datasets, evaluation and entailment approach. arXiv preprint arXiv:1909.00161.

See Also

```
citation("transforEmotion")
```

.init_builtin_models *Initialize Built-in Vision Models*

Description

Register the default/built-in vision models that come with transforEmotion. This function is automatically called when the package is loaded.

Usage

```
.init_builtin_models()
```

Value

Invisibly returns TRUE

.vision_model_registry *Vision Model Registry for transforEmotion Package*

Description

Central registry system for managing vision models in transforEmotion. Provides extensible architecture allowing users to register custom vision models beyond the default CLIP-based models.

Usage

```
.vision_model_registry
```

Details

The registry maintains a list of available vision models with their configurations. Each model entry includes the HuggingFace model ID, architecture type, and metadata needed for proper initialization and processing.

Author(s)

Aleksandar Tomasevic <atomashevic@gmail.com>

add_vision_model *User-Friendly Vision Model Management Functions*

Description

High-level functions for managing vision models in transforEmotion, providing an easy interface for extending the package with custom models.

User-friendly wrapper for registering custom vision models with automatic validation and helpful error messages.

Usage

```
add_vision_model(
  name,
  model_id,
  description = NULL,
  architecture = "clip",
  test_labels = NULL,
  force = FALSE
)
```

Arguments

name	A short, memorable name for your model (e.g., "my-emotion-model")
model_id	HuggingFace model identifier or path to local model directory
description	Optional description of the model and its purpose
architecture	Model architecture type. Currently supported: • "clip": Standard CLIP models (most compatible) • "clip-custom": CLIP variants needing special handling • "blip": BLIP models (caption-likelihood scoring) • "align": ALIGN dual-encoder models (direct similarity)
test_labels	Optional character vector to test the model immediately
force	Logical indicating whether to overwrite existing model with same name

Value

Invisibly returns TRUE if successful

Author(s)

Aleksandar Tomasevic <atomashevic@gmail.com> Add a Custom Vision Model

Examples

```
## Not run:
# Add a fine-tuned CLIP model for emotion recognition
add_vision_model(
  name = "emotion-clip",
  model_id = "openai/clip-vit-large-patch14",
  description = "Large CLIP model for better emotion recognition",
  test_labels = c("happy", "sad", "angry"),
  force = TRUE
)

# Add a local model
add_vision_model(
  name = "my-local-model",
  model_id = "/path/to/my/model",
  description = "My custom fine-tuned model"
)

# Add experimental BLIP model
add_vision_model(
  name = "blip-base",
  model_id = "Salesforce/blip-image-captioning-base",
  architecture = "blip",
  description = "BLIP model for image captioning"
)

# Now use any of them in analysis
result <- image_scores("photo.jpg", c("happy", "sad"), model = "emotion-clip")
```

```
batch_results <- image_scores_dir("photos/", c("positive", "negative"),
                                  model = "my-local-model")

## End(Not run)
```

`annotate_faces` *Code faces in the browser*

Description

Starts the coding page of an `annotation_task` from R and opens it in the web browser. Each answer is saved in the task's `results` folder as soon as the coder gives it, so closing the browser or R loses nothing, and a coder who comes back continues where they stopped.

Usage

```
annotate_faces(
  task,
  coder = NULL,
  open = interactive(),
  port = NULL,
  inline = .te_in_notebook(),
  max_side = 640
)
```

Arguments

`task`	An `annotation_task`, or its folder.
`coder`	The coder id to start with. Default: the page asks.
`open`	Logical. Open the page in the web browser. Default is `TRUE` in an interactive session.
`port`	Port of the local server. Default is a free port chosen at random.
`inline`	Logical. Show the coding page inside the notebook, below the cell. Default is `TRUE` in a Jupyter notebook (Google Colab, JupyterLab, VS Code) and `FALSE` elsewhere.
`max_side`	Longest side, in pixels, of the pictures in a notebook page, which stores them in the notebook. Default 640.

Details

The page is served by R (with **httpuv**) on your own computer only (`127.0.0.1`), at an address with a random part, and keeps running while you use R. Coders press the number keys to answer, Enter for the next item, and Backspace to undo.

Coders on other computers can open `page/index.html` of the task from a copy of the folder instead. Their page keeps answers in that browser and ends with **Export annotations**; put the exported files in the task's `results` folder.

In a Jupyter notebook, including Google Colab, the coding page is shown below the cell instead (`inline = TRUE`, the default there), because a page served from R cannot be reached from a notebook running on another computer. Its **Copy R code** button gives a line such as `answers <- read_annotations(task, "te-annotations:...")` to run in a new cell. The line holds every answer given so far, so copy it again after more coding; keep the last one, as the answers are not saved in the task's `results` folder.

Value

Invisibly, the page: a list with its `url` and `stop()`, which stops the server. With `inline = TRUE`, invisibly, the path of the page as one self-contained file.

See Also

`read_annotations`, `annotation_progress`

Examples

```
## Not run:
coding <- annotate_faces("annotation-round-1", coder = "C01")
annotation_progress("annotation-round-1")
coding$stop()

## End(Not run)
```

---

annotation_agreement *Agreement between coders*

Description

Krippendorff's alpha, Fleiss' kappa and percent agreement for each question of a face annotation task.

Usage

```
annotation_agreement(annotations, questions = NULL, level = NULL)
```

Arguments

`annotations`	The result of `read_annotations`.
`questions`	Questions to compute agreement for. Default: all.
`level`	Level of measurement for alpha, as a named character vector (for example `c(intensity = "ordinal")`): `"nominal"`, `"ordinal"`, `"interval"` or `"ratio"`. Default: `"nominal"` for single-choice questions and `"interval"` for Likert questions.

Details

Only items with answers from at least two coders count; a question that was not asked (after "Not a face", say) is missing, not an answer.

`alpha` Krippendorff's alpha, from the coincidence matrix. It allows for missing answers and for items coded by different numbers of coders.

`fleiss_kappa` Fleiss' kappa, when every counted item has answers from the same number of coders; `NA` otherwise. Treats answers as categories.

`percent_agreement` The share of pairs of coders who gave the same answer to an item, averaged over items.

Value

A data frame with one row per question: `question`, `level`, `n_items`, `n_coders`, `n_answers`, `percent_agreement`, `alpha` and `fleiss_kappa`.

References

Krippendorff, K. (2011). Computing Krippendorff's alpha-reliability. https://repository.upenn.edu/items/034a6030-c584-4d14-9d3d-7b7e8d16df20

Fleiss, J. L. (1971). Measuring nominal scale agreement among many raters. *Psychological Bulletin, 76*(5), 378-382.

See Also

`consensus_labels`

---

annotation_progress *Progress of a face annotation task*

Description

Progress of a face annotation task

Usage

```
annotation_progress(task)
```

Arguments

`task`	An `annotation_task`, or its folder.

Value

A data frame with one row per coder: items assigned, answered, skipped and left, and the median seconds per answered item.

See Also

annotate_faces

---

annotation_questions *Questions for a face annotation task*

Description

Ready-made questions for annotation_task. Each question is a list, so you can change them or add your own (see Details).

Usage

```
annotation_questions(which = c("face_check", "emotion", "intensity"))
```

Arguments

which	Names of the ready-made questions, in the order they are asked:
	"face_check" Does the outline contain a face? Yes, not a face, a different person, or can't tell. The other questions are asked only after "Yes", so detector mistakes never get an emotion.
	"emotion" Which emotion the face shows, one of the task's classes.
	"intensity" How intense it is, 1 to 7.
	"valence" How negative or positive the face is, -3 to 3.
	"arousal" How calm or excited the face is, 0 to 6.
	"why" For items with a reference label and a model prediction (annotation_task(items = ...)): what explains the result, compared with the label and the model. The model is wrong, the label is debatable, the wrong face was scored, no face is visible, the model and the label agree, or something else. Asked after the label, the prediction and the scores are shown (reveal = TRUE).

Details

A question is a list with

id A name in lower case: letters, digits and underscores.

type "single" (one of a few options) or "likert" (a whole number from min to max).

prompt The question as coders read it.

options For "single": the answers, or options_from = "classes" for the task's classes. Optional labels: a named list of the text shown for each answer.

min, max For "likert", with optional anchors = list(min = "...", max = "...").

show_if Optional: ask only after another question got one of these answers, as in list(face_check = "ok").

reveal Optional, for items with a reference label and a model prediction: TRUE asks the question only once they are shown, which with blind = TRUE is after the other questions are answered.

required FALSE lets coders leave it unanswered.

Value

A list of questions.

See Also

annotation_task

Examples

```
annotation_questions(c("face_check", "emotion"))

# Add a question of your own
gaze <- list(id = "gaze", type = "single", prompt = "Where does the person look?",
             options = c("camera", "away", "closed"), show_if = list(face_check = "ok"))
questions <- c(annotation_questions(), list(gaze))
```

---

annotation_task *Set up a face annotation task*

Description

Turns detected faces into a coding task: each face is an item that coders see outlined in its picture, with a close-up, and answer questions about, such as whether it is a face at all and which emotion it shows. Start the coding page with annotate_faces, then read the answers with read_annotations.

Usage

```
annotation_task(
  faces = NULL,
  classes = NULL,
  questions = NULL,
  coders = "C01",
  overlap = 1,
  dir,
  seed = 1,
  title = NULL,
  instructions = NULL,
  max_side = 1280,
  task_id = NULL,
  items = NULL,
  frames = 4,
  play = TRUE,
  blind = TRUE
)
```

Arguments

faces	A data frame returned by detect_faces or select_faces: one item per row. Or NULL, with items.
classes	The emotions coders choose from, for example the classes passed to face_scores. Needed by the "emotion" question.
questions	The questions, from annotation_questions. Default: annotation_questions() for faces; annotation_questions(c("emotion", "why")) for items with a label and a prediction, else "emotion".
coders	Coder ids: letters, digits, dots, dashes or underscores.
overlap	Share of the items that every coder codes, for agreement. The rest are split between the coders. Default is 1: every coder codes every item.
dir	A new or empty folder for the task.
seed	Seed for the order of the items. Each coder gets their own order. Your own random numbers are left as they were.
title	Title shown on the page.
instructions	Text shown to coders before the first item.
max_side	Longest side, in pixels, of the picture copies.
task_id	An id for the task. Coders' progress in the browser is kept under it. Default is made from the time.
items	Instead of faces: a data frame with one row per item and the columns media (path of a video or image file) and/or text, and optionally id, label (the reference label, returned as reference), prediction (the model's top class) and one column of model scores per class.
frames	For items: the number of frames shown per video clip, spread over the clip as video_scores(uniform = TRUE) samples them.
play	For items: put the clips themselves in the page, with their sound, when a browser can play them (MP4, WebM, MOV).
blind	For items: show the label, prediction and scores only after the questions without reveal are answered.

Details

With items instead of faces, each item is a video clip, an image or a text, for example the clips a model got wrong. Coders see sampled frames of the clip with its face boxes (the largest face, the one video_scores scores by default, drawn solid), the clip itself with its sound, and the item's text. When the items have a reference label and a model prediction, the page shows them with the model's scores once the coder has answered (blind = TRUE), then asks what explains the result (annotation_questions("why")). Unlike a face task, such a page holds the texts, labels and scores.

The task is a folder:

page/ The coding page: small copies of the images or video frames, under names that say nothing about the files, and the questions. It holds no file names, scores, or other coders' answers.

`items.csv` Which face each item is: its file, frame, face number, track and person. For you, not for coders.

`task.json` The questions, items and each coder's list.

`results/` The answers, one file per coder.

Value

The task, invisibly: a list with its folder (`dir`), `task_id`, `questions`, `coders` and `items` (the contents of `items.csv`).

See Also

`annotate_faces`, `read_annotations`, `annotation_agreement`, `consensus_labels`

Examples

```
## Not run:
faces <- detect_faces(system.file("extdata", "two-people.mp4", package = "transforEmotion"),
                      nframes = 20)
task <- annotation_task(faces, classes = c("happy", "sad", "angry", "neutral"),
                        coders = c("C01", "C02"), dir = "annotation-round-1")
annotate_faces(task)

# A model's errors on video clips, with their descriptions
emotions <- c("angry", "fearful", "happy", "sad")
items <- data.frame(id = errors$clip, media = errors$path, text = errors$description,
                label = errors$label, prediction = errors$predicted, errors[, emotions])
review <- annotation_task(items = items, classes = emotions, dir = "error-review")
annotate_faces(review)
answers <- read_annotations(review)
table(answers$value[answers$question == "why"])

## End(Not run)
```

`as_rag_table` *Convert RAG JSON to a table*

Description

Produces a long-form data.frame with columns: ‘label‘, ‘confidence‘, ‘intensity‘, ‘doc_id‘, ‘span‘, ‘score‘.

Usage

```
as_rag_table(x, validate = TRUE)
```

Arguments

x	JSON string or parsed list.
validate	Logical; validate structure first.

Value

A data.frame suitable for statistical analysis.

Examples

```
j <- '{"labels":["joy","surprise"],"confidences":[0.8,0.5],
  "intensity":0.7,"evidence_chunks":[]}'
as_rag_table(j)
```

---

calculate_moving_average

Calculate the moving average for a time series

Description

This function calculates the moving average for a time series.

Usage

```
calculate_moving_average(data, window_size)
```

Arguments

data	Matrix or Data frame. The time series data
window_size	Numeric integer. The size of the moving average window.

Value

Matrix or Data frame containing the moving average values.

`calibrate_scores` *Calibrate frame scores for posterior_scores()*

Description

Learns how far to trust a model's frame scores, so that `posterior_scores` can treat them as noisy evidence instead of taking them at face value. Without labels it learns the model's average score for each class, its lean towards some labels before it has seen a face. With the true emotion of some videos it also learns the model's confusions: how often it gives each top label when the true emotion is a given one.

Usage

```
calibrate_scores(scores, labels = NULL, classes = NULL, alpha = NULL)
```

Arguments

`scores`	A data frame of scores for one video, one row per frame, as returned by `video_scores` or `face_scores` (one face per frame), or a list of such data frames, one per video. Use scores from the setting you will analyse: the same model, classes and wording, and similar videos.
`labels`	Optional. A character vector with the true emotion of each video, one entry per data frame in `scores`; `NA` for videos without one. Every label must be one of `classes`.
`classes`	Character vector. The score columns. Default: every numeric column except the columns of a faces table (`frame`, `time`, `x`, `confidence` and so on).
`alpha`	Number between 0 and 1: how much one frame's evidence counts. `NULL` (default) estimates it: with labels, as the value that gives the labelled videos the best clip posteriors (log score, with each video left out of the confusion matrix it is scored with, and only videos whose emotion has at least one other labelled video), but never above the value from the correlation of the evidence between neighbouring frames; without labels, or with no such video, that value itself.

Details

A model's scores are not probabilities to be read as they are. CLIP's scores have a lean of their own: averaged over the frames of CREMA-D clips with six bare labels, standard CLIP gives "neutral" 37% of its probability, whichever emotion the actor shows. They make the same mistakes again and again, such as reading disgust as sadness. And frames a fraction of a second apart are nearly copies of each other, so 80 frames are not 80 independent looks at a face.

The calibration turns the score vector p_t of frame t into evidence $L_t(c)$ for each emotion c:

- without labels, $L_t(c) = (p_t(c)/\pi(c))^{\alpha}$, where $\pi(c)$ is the model's mean score for c (`prior`);
- with labels, $L_t(c) = (\sum_k p_t(k) M_{ck})^{\alpha}$, where M_{ck} is the share of frames with true emotion c whose top label is k (`confusion`).

α discounts frames that repeat each other: with $\alpha = 0.1$, ten frames count as one independent look. Without labels it is $(1-\rho)/(1+\rho)$, where ρ is the correlation of the evidence between neighbouring frames, the share of independent observations in a series with that correlation. Videos whose frames all have the same scores are left out of the correlation; if every video is like that, α is 0.01.

Scores are renormalised over `classes` in every row, which is the same as scoring with only those labels.

Value

An object of class `te_calibration`, a list with

classes The score columns, in order.

prior The model's mean score of each class over all frames.

confusion With labels: a matrix whose row `c` gives the share of frames of videos with true emotion `c` that have each top label (add-one smoothed). `NULL` without labels.

likelihood `"prior"` or `"confusion"`: how `posterior_scores` turns scores into evidence.

alpha The weight of one frame's evidence.

n_videos, n_frames, n_labelled The videos, frames with scores and labelled videos it was fitted on.

See Also

`posterior_scores`

Examples

```
# Frame scores of nine short clips (in practice from video_scores())
set.seed(1)
clip <- function(n, emotion) {
  s <- matrix(runif(n * 3), n, 3,
              dimnames = list(NULL, c("neutral", "happy", "sad")))
  s[, "neutral"] <- s[, "neutral"] + 0.6  # the model leans to neutral
  s[, emotion] <- s[, emotion] + 0.5
  as.data.frame(s / rowSums(s))
}
emotions <- rep(c("neutral", "happy", "sad"), each = 3)
scores <- lapply(emotions, function(e) clip(40, e))

# Without labels: correct for the model's lean
calibrate_scores(scores)

# With labels: also learn its confusions
calibration <- calibrate_scores(scores, labels = emotions)
calibration
```

check_findingemo_quality *Check FindingEmo Dataset Quality*

Description

Checks the quality and completeness of a downloaded FindingEmo dataset, reporting on file availability, image accessibility, and potential issues.

Usage

```
check_findingemo_quality(data_dir, check_images = FALSE, sample_size = 10)
```

Arguments

data_dir	Character. Directory containing the FindingEmo dataset.
check_images	Logical. Whether to verify image file accessibility (default: FALSE, as this can be slow).
sample_size	Integer. If check_images is TRUE, number of images to sample for verification (default: 10).

Value

A list containing:

- structure: Dataset structure type ("standard" or "flat")
- files_found: List of available files
- annotations_count: Number of annotations
- urls_count: Number of image URLs
- images_count: Number of downloaded images
- completeness: Percentage of images successfully downloaded
- image_check: Results of image accessibility check (if performed)

Examples

```
## Not run:
# Check dataset quality
quality_report <- check_findingemo_quality("./findingemo_data")
print(quality_report)

# Check with image verification
quality_report <- check_findingemo_quality(
  data_dir = "./findingemo_data",
  check_images = TRUE,
  sample_size = 5
)
```

```
## End(Not run)
```

`consensus_labels` *One label per item from several coders*

Description

Combines the coders' answers to one question into one value per item: the most common answer, or the mean or median of Likert answers. Use it as the truth for `evaluate_emotions`.

Usage

```
consensus_labels(
  annotations,
  question = "emotion",
  method = c("majority", "mean", "median"),
  min_coders = 1
)
```

Arguments

`annotations`	The result of `read_annotations`.
`question`	The question to combine. Default is `"emotion"`.
`method`	`"majority"` (the most common answer), `"mean"` or `"median"`.
`min_coders`	Items with fewer answers than this get `NA`.

Value

A data frame with one row per item: `item_id`, `label` (the combined answer, `NA` for a tie or too few answers), `n_coders`, `agreement` (the share of coders who gave the majority answer; `NA` for the mean or median), `tie`, and the item's face (`source`, `frame`, `frame_index`, `time`, `face`, `track_id`, `person_id`), or, for a task made from `items`, its `id`, `reference`, `prediction` and scores: compare the coders with the reference by `evaluate_emotions(consensus, truth_col = "reference", pred_col = "label")`.

See Also

`annotation_agreement`

delete_transformer *Delete a Transformer Model*

Description

Large language models can be quite large and, when stored locally, can take up a lot of space on your computer. The direct paths to where the models are on your computer is not necessarily intuitive.

This function quickly identifies the models on your computer and informs you which ones can be deleted from it to open up storage space

Usage

```
delete_transformer(model_name, delete = FALSE)
```

Arguments

model_name	Character vector. If no model is provided, then a list of models that are locally stored on the computer are printed
delete	Boolean (length = 1). Should model skip delete question? Defaults to FALSE. Set to TRUE for less interactive deletion

Value

Returns list of models or confirmed deletion

Author(s)

Alexander P. Christensen <alexpaulchristensen@gmail.com>

Examples

```
if(interactive()){
  delete_transformer()
}
```

detect_faces *Detect faces in images or a video*

Description

Finds every face in a set of images or in frames sampled from a video, so you can see which faces the package found, pick the ones to analyse (with select_faces or ordinary subsetting) and score only those with face_scores.

Usage

```
detect_faces(
  x,
  face_detector = c("yunet", "haar"),
  min_confidence = 0.6,
  identify = TRUE,
  identity_threshold = 0.363,
  nframes = 100,
  start = 0,
  end = -1,
  uniform = FALSE,
  ffreq = 15,
  save_frames = FALSE,
  save_video = FALSE,
  save_dir = tempdir(),
  video_name = "temp",
  pattern = "\\.(jpg|jpeg|png|bmp)$",
  recursive = FALSE
)
```

Arguments

x	Image paths or URLs, a directory of images, or one video (a file or a YouTube URL).
face_detector	"yunet" (default), OpenCV's YuNet detector, or "haar", the Haar cascade transforEmotion used before version 0.1.8. YuNet finds small, tilted and partly turned faces and reports a confidence score and five landmarks; Haar finds frontal faces only.
min_confidence	YuNet's confidence threshold, between 0 and 1. Ignored for Haar.
identify	Logical. Give faces of the same person the same person_id, using OpenCV's SFace face recognizer. Needs YuNet. The recognition model (10 MB) is downloaded on first use; see Details.
identity_threshold	Cosine similarity between faces (or video tracks) above which they count as the same person. The default, 0.363, is the threshold SFace's authors recommend; raise it if different people share a person_id, lower it if one person gets several.
nframes, start, end, uniform, ffreq	How frames are sampled from a video; see video_scores. Use the same values in both functions to get the same frames.
save_frames	Logical. Keep the extracted video frames in save_dir. Default is FALSE.
save_video	Logical. Keep a downloaded YouTube video in save_dir. Default is FALSE. The video has no sound when YouTube offers no stream with video and audio in one file.
save_dir	Directory for video frames. Default is a temporary directory.
video_name	Name used for the frame files (<video_name>-frame-<i>.jpg).

pattern, recursive
: When x is a directory: a regular expression for the image files and whether to search subdirectories.

Details

Faces are numbered from left to right within each image or frame. In a video, a face is followed from frame to frame and keeps its track_id. With identify = TRUE, faces of the same person get the same person_id: across all images, or across all tracks of the video, so a person who leaves the picture and comes back, or who appears in several images, is one person.

Face recognition compares faces through SFace embeddings (128 numbers per face). Faces are grouped by average-linkage clustering on cosine similarity; a video track counts as one face, the average of its frames. Two faces in the same image can be the same person (mirrors, screens, posters). The embeddings stay in memory and are never returned or written to disk, and person_ids only label faces within one call: they do not name anyone and do not match ids from other calls.

Grouping faces by person is processing of biometric data under the EU General Data Protection Regulation (Article 9); make sure your ethics approval covers it, or use identify = FALSE.

The YuNet model ships with the package. The SFace model (face_recognition_sface_2021dec_int8.onnx, from OpenCV's model zoo, Apache 2.0 licence) is downloaded into the package cache on first use, or by setup_modules. To use a copy elsewhere, for example on a computer without internet access, set the environment variable TRANSFOREMOTION_SFACE_MODEL to its path. Without the model, person_id is NA.

Value

A data frame with one row per face and the columns:

source The image path or URL, or the video.

frame, frame_index, time For a video: the sample number (as in the frame file name), the frame's index in the video and its time in seconds. NA for images.

face Face number within the image or frame, from left to right.

track_id For a video: the face's track. NA for images.

person_id "p1", "p2", ... numbered by first appearance; NA without recognition.

x, y, width, height The face box in pixels, from the top-left corner of the image.

confidence YuNet's score; NA for Haar.

detector "yunet" or "haar".

image_width, image_height Size of the image or frame in pixels.

right_eye_x, ..., mouth_left_y The five YuNet landmarks (NA for Haar).

The attribute "media" holds one row per image or frame, with the number of faces found (n_faces), so images without faces are not lost. The attribute "settings" records the detector and, for a video, how frames were sampled, which face_scores uses to find the same frames again.

Data Privacy

All processing is done locally; images and frames are never sent to a remote server.

See Also

select_faces, face_scores, draw_faces

Examples

```
## Not run:
image <- system.file("extdata", "two-people.jpg", package = "transforEmotion")
faces <- detect_faces(image)
faces[, c("face", "person_id", "x", "y", "width", "height", "confidence")]
draw_faces(faces)

# Every face of one person in a video, scored once per sampled frame
video <- system.file("extdata", "two-people.mp4", package = "transforEmotion")
faces <- detect_faces(video, nframes = 20)
table(faces$track_id, faces$person_id)
scores <- face_scores(select_faces(faces, persons = "p1"),
                      classes = c("happy", "sad", "angry", "neutral"))

## End(Not run)
```

---

dlo_dynamics *Dynamics function of the DLO model*

Description

This function calculates the dynamics of a system using the DLO (Damped Linear Oscillator) model based on Equation 1 (Ollero et al., 2023). The DLO model is a second-order differential equation that describes the behavior of a damped harmonic oscillator. The function takes in the current state of the system, the derivative of the state, the damping coefficient, the time step, and the values of the eta and zeta parameters. It returns the updated derivative of the state.

Usage

```
dlo_dynamics(x, dxdt, q, dt, eta, zeta)
```

Arguments

x	Numeric. The current state of the system (value of the latent score).
dxdt	Numeric. The derivative of the state (rate of change of the latent score).
q	Numeric. The damping coefficient.
dt	Numeric. The time step.
eta	Numeric. The eta parameter of the DLO model.
zeta	Numeric. The zeta parameter of the DLO model.

Value

A numeric vector containing the updated derivative of the state.

References

Ollero, M. J. F., Estrada, E., Hunter, M. D., & Cancer, P. F. (2023). Characterizing affect dynamics with a damped linear oscillator model: Theoretical considerations and recommendations for individual-level applications. *Psychological Methods*. doi:10.1037/met0000615

`download_findingemo_data` *Download FindingEmo-Light Dataset*

Description

Downloads the FindingEmo-Light dataset using the official PyPI package. This dataset contains 25k images with emotion annotations including valence, arousal, and discrete emotion labels, focusing on complex naturalistic scenes with multiple people in social settings.

Usage

```
download_findingemo_data(
  target_dir,
  max_images = NULL,
  randomize = FALSE,
  skip_existing = TRUE,
  force = FALSE
)
```

Arguments

`target_dir`	Character. Directory to download the dataset to.
`max_images`	Integer. Maximum number of images to download (optional).
`randomize`	Logical. If TRUE and max_images is specified, randomly select images for download. Useful for creating test/benchmark subsets (default: FALSE).
`skip_existing`	Logical. Whether to skip download if dataset already exists (default: TRUE).
`force`	Logical. Force download even if dataset exists (default: FALSE).

Details

This function uses the `findingemo-light` Python package, which is installed automatically on first use (or ahead of time with `setup_modules(extras = "findingemo")`).

The FindingEmo dataset is described in: Mertens, L. et al. (2024). "FindingEmo: An Image Dataset for Emotion Recognition in the Wild". NeurIPS 2024 Datasets and Benchmarks Track.

The dataset uses a flat directory structure with all images stored directly in the images/ subdirectory, annotations.csv and urls.json at the root level.

Note: For copyright reasons, the dataset provides URLs and annotations only. Images are downloaded on-demand from their original sources.

Value

A list containing:

- success: Logical indicating if download was successful
- message: Character string with status message
- target_dir: Path to downloaded data
- annotation_file: Path to annotation file (if successful)
- urls_file: Path to URLs file (if successful)
- image_count: Number of images downloaded (if any)
- annotations: Full annotations data.frame (raw)
- evaluation_data: Data.frame filtered to downloaded images with columns suitable for evaluation workflows (id, truth, image_file, image_path, valence, arousal, emo8_label, emotion)
- evaluation_csv: Path to saved CSV of evaluation_data
- matched_count: Number of annotations matched to downloaded images

See Also

load_findingemo_annotations, setup_modules

Examples

```
## Not run:
# First install required modules
setup_modules(extras = "findingemo")

# Download dataset to local directory
result <- download_findingemo_data("./findingemo_data")

if (result$success) {
  cat("Dataset downloaded to:", result$target_dir)
  cat("Images downloaded:", result$image_count)
}

# Download random subset for testing/benchmarking
result <- download_findingemo_data(
  target_dir = "./findingemo_test",
  max_images = 100,
  randomize = TRUE
)

# Download subset with flat directory structure (always used)
result <- download_findingemo_data(
  target_dir = "./findingemo_subset",
  max_images = 50
)

# Force re-download
result <- download_findingemo_data(
```

```
  target_dir = "./findingemo_data",
  force = TRUE
)

## End(Not run)
```

`draw_faces` *Draw detected faces*

Description

Draws the box and a label for each face on its image or video frame, so you can check what `detect_faces` found: which faces were detected, how they are numbered, and which faces were recognised as the same person. Faces of the same person share a colour. `plot()` on a `detect_faces` result does the same.

Usage

```
draw_faces(faces, labels = NULL, max_images = 20, dir = tempdir(), plot = TRUE)

## S3 method for class 'te_faces'
plot(x, ...)
```

Arguments

`faces`	A data frame returned by `detect_faces` or `select_faces`, or a subset of its rows.
`labels`	Text drawn above each box: one per row of `faces`, or one for all. By default the face number and person (`"face 1, p1"`), or for a video the track and person (`"track 1, p1"`). Use `""` for boxes only.
`max_images`	Most images or video frames to draw. When there are more, they are picked evenly from the first to the last.
`dir`	Directory for the drawn images (PNG files). Default is a temporary directory.
`plot`	Logical. Show the drawn images in the plot window, or below the cell in a notebook. Default is `TRUE`.
`x`	A data frame returned by `detect_faces`.
`...`	Passed to `draw_faces()`.

Details

Only images and frames with a face in `faces` are drawn. The `"media"` attribute of a `detect_faces` result lists every image or frame with its number of faces, including those without any.

For a video, the frames are extracted again (as in `face_scores`), unless `detect_faces` kept them with `save_frames = TRUE`.

Value

The paths of the drawn images, invisibly.

See Also

detect_faces, select_faces

Examples

```
## Not run:
image <- system.file("extdata", "two-people.jpg", package = "transforEmotion")
faces <- detect_faces(image)
draw_faces(faces)

# A video: a few frames, with each face's track and person
video <- system.file("extdata", "two-people.mp4", package = "transforEmotion")
plot(detect_faces(video, nframes = 20), max_images = 4)

## End(Not run)
```

---

emotions *Emotions Data*

Description

A matrix containing words (n = 175,592) and the emotion category most frequently associated with each word. This dataset is a modified version of the 'DepecheMood++' lexicon developed by Araque, Gatti, Staiano, and Guerini (2018). For proper scoring, text should not be stemmed prior to using this lexicon. This version of the lexicon does not rely on part of speech tagging.

Usage

```
data(emotions)
```

Format

A data frame with 175,592 rows and 9 columns.

word An entry in the lexicon, in English

AFRAID, AMUSED, ANGRY, ANNOYED, DONT_CARE, HAPPY, INSPIRED, SAD The emotional category. All emotions contain either a 0 or 1. If the category is most likely to be associated with the word, it recieves a 1, otherwise, 0. Words are only associated with one category.

References

Araque, O., Gatti, L., Staiano, J., and Guerini, M. (2018). DepecheMood++: A bilingual emotion lexicon built through simple yet powerful techniques. *ArXiv*

Examples

```
data("emotions")
```

`emoxicon_scores` *Emoxicon Scores*

Description

A bag-of-words approach for computing emotions in text data using the lexicon compiled by Araque, Gatti, Staiano, and Guerini (2018).

Usage

```
emoxicon_scores(text, lexicon, exclude)
```

Arguments

`text`	Matrix or data frame. A data frame containing texts to be scored (one text per row)
`lexicon`	The lexicon used to score the words. The default is the `emotions` dataset, a modification of the lexicon developed by Araque, Gatti, Staiano, and Guerini (2018). To use the raw lexicon from Araque et. al (2018) containing the original probability weights, use the `weights` dataset. If another custom lexicon is used, the first column of the lexicon should contain the terms and the subsequent columns contain the scoring categories.
`exclude`	A vector listing terms that should be excluded from the lexicon. Words specified in `exclude` will not influence document scoring. Users should consider excluding 'red herring' words that are more closely related to the topics of the documents, rather than the documents' emotional content. For example, the words "clinton" and "trump" are present in the lexicon and are both associated with the emotion 'AMUSED'. Excluding these words when analyzing political opinions may produce more accurate results.

Author(s)

Tara Valladares <tls8vx at virginia.edu> and Hudson F. Golino <hfg9s at virginia.edu>

References

Araque, O., Gatti, L., Staiano, J., and Guerini, M. (2018). DepecheMood++: A bilingual emotion lexicon built through simple yet powerful techniques. *ArXiv*

See Also

`emotions`, where we describe how we modified the original DepecheMood++ lexicon.

Examples

```
# Obtain "emotions" data
data("emotions")

# Obtain "tinytrolls" data
data("tinytrolls")

## Not run:
# Obtain emoxicon scores for first 10 tweets
emotions_tinytrolls <- emoxicon_scores(text = tinytrolls$content, lexicon = emotions)

## End(Not run)
```

`emphasize` *Generate and emphasize sudden jumps in emotion scores*

Description

This function generates and emphasizes the effect of strong emotions expressions during the period where the derivative of the latent variable is high. The observable value of the strongest emotion from the positive or negative group will spike in the next k time steps. The probability of this happening is p at each time step in which the derivative of the latent variable is greater than 0.2. The jump is proportionate to the derivative of the latent variable and the sum of the observable values of the other emotions.

Usage

```
emphasize(data, num_observables, num_steps, k = 10, p = 0.5)
```

Arguments

`data`	Data frame. The data frame containing the latent and observable variables created by the `simulate_video` function.
`num_observables`	Numeric integer. The number of observable variables per latent factor.
`num_steps`	Numeric integer. The number of time steps used in the simulation.
`k`	Numeric integer. The mumber of time steps to emphasize the effect of strong emotions on future emotions (default is 10). Alternatively: the length of a strong emotional episode.
`p`	Numeric. The probability of the strongest emotion being emphasized in the next k time steps (default is 0.5).

Value

A data frame containing the updated observable variables.

`evaluate_emotions` *Evaluate Emotion Classification Performance*

Description

Comprehensive evaluation function for discrete emotion classification tasks. Computes standard classification metrics including accuracy, F1-scores, AUROC, calibration metrics, and inter-rater reliability measures.

Usage

```
evaluate_emotions(
  data,
  id_col = "id",
  truth_col = "truth",
  pred_col = "pred",
  probs_cols = NULL,
  classes = NULL,
 metrics = c("accuracy", "precision", "recall", "f1_macro", "f1_micro", "auroc", "ece",
    "krippendorff", "confusion_matrix"),
  return_plot = FALSE,
  na_rm = TRUE
)
```

Arguments

`data`	A data frame or file path to CSV containing evaluation data. Must include columns for identifiers, ground truth, predictions, and optionally class probabilities.
`id_col`	Character. Name of column containing unique identifiers (default: "id").
`truth_col`	Character. Name of column containing ground truth labels (default: "truth").
`pred_col`	Character. Name of column containing predicted labels (default: "pred").
`probs_cols`	Character vector. Names of columns containing class probabilities. If NULL, probabilistic metrics will be skipped. AUROC pairs `probs_cols[i]` with `classes[i]`.
`classes`	Character vector. Emotion classes to evaluate. If NULL, will be inferred from the data. With `probs_cols`, the inferred classes take the order of `probs_cols`, matched by column name (without a `"prob_"` prefix if every column has one). If the column names do not match the labels, the classes are sorted and AUROC gives a warning.
`metrics`	Character vector. Metrics to compute. Options include: "accuracy", "precision", "recall", "f1_macro", "f1_micro", "auroc", "ece", "krippendorff", "confusion_matrix" (default: all metrics).
`return_plot`	Logical. Whether to return plotting helpers (default: FALSE).
`na_rm`	Logical. Whether to remove missing values (default: TRUE).

Details

This function implements a comprehensive evaluation pipeline for discrete emotion classification following best practices from the literature.

Metrics computed:

- **Accuracy**: Overall classification accuracy
- **Precision/Recall/F1**: Per-class and macro/micro averages
- **AUROC**: Area under ROC curve (requires probability scores)
- **ECE**: Expected Calibration Error for probability calibration
- **Krippendorff's alpha**: Inter-rater reliability between human and model

Input format: The input data should contain at minimum:

- ID column: Unique identifier for each instance
- Truth column: Ground truth emotion labels
- Prediction column: Model predicted emotion labels
- Probability columns (optional): Class probabilities for each emotion

Value

A list containing:

- `metrics`: Data frame with computed evaluation metrics
- `confusion_matrix`: Confusion matrix (if requested)
- `per_class`: Per-class metrics breakdown
- `summary`: Overall performance summary
- `plot_data`: Data prepared for plotting (if return_plot = TRUE)

References

Grandini, M., Bagli, E., & Visani, G. (2020). Metrics for multi-class classification: an overview. arXiv preprint arXiv:2008.05756.

Krippendorff, K. (2011). Computing Krippendorff's alpha-reliability. Scholarly commons, 25.

Naeini, M. P., Cooper, G., & Hauskrecht, M. (2015). Obtaining well calibrated probabilities using bayesian binning. In AAAI (pp. 2901-2907).

See Also

`transformer_scores`, `nlp_scores`, `emoxicon_scores` for emotion prediction functions.

Examples

```
## Not run:
# Basic evaluation with predicted labels only
results <- evaluate_emotions(
  data = evaluation_data,
  truth_col = "human_label",
  pred_col = "model_prediction"
)

# Full evaluation with probabilities
results <- evaluate_emotions(
  data = evaluation_data,
  truth_col = "ground_truth",
  pred_col = "predicted_class",
  probs_cols = c("prob_anger", "prob_joy", "prob_sadness"),
  return_plot = TRUE
)

# Custom metrics selection
results <- evaluate_emotions(
  data = evaluation_data,
  metrics = c("accuracy", "f1_macro", "confusion_matrix")
)

## End(Not run)
```

---

face_scores *Score selected faces*

Description

Scores each face of a `detect_faces` result with a CLIP model, so you can analyse exactly the faces you picked, for example one person across a whole video.

Usage

```
face_scores(
  faces,
  classes,
  model = "oai-base",
  local_model_path = NULL,
  padding = 0.25
)
```

Arguments

faces	A data frame returned by `detect_faces` (or a subset of its rows).

`classes` A character vector of classes to score.

`model`, `local_model_path`
The vision model; see `image_scores`.

`padding` Share of the face box's width and height added on each side before scoring. Default is 0.25.

Details

For a video, the frames are extracted again with the sampling settings `detect_faces` used (and the video downloaded again from YouTube), unless the frames were kept with `save_frames = TRUE`.

Value

`faces` without the landmark columns, plus one column of scores per class.

Data Privacy

All processing is done locally with the downloaded model.

See Also

`detect_faces`, `select_faces`

Examples

```
## Not run:
image <- system.file("extdata", "boris-1.png", package = "transforEmotion")
faces <- detect_faces(image)
face_scores(faces, classes = c("happy", "sad", "angry", "neutral"))

## End(Not run)
```

---

generate_observables *Generate observable emotion scores data from latent variables*

Description

Function to generate observable data from 2 latent variables (negative and positive affect). The function takes in the latent variable scores, the number of time steps, the number of observable variables per latent factor, and the measurement error variance. It returns a matrix of observable data. The factor loadings are not the same for all observable variables. They have uniform random noise added to them (between -0.15 and 0.15). The loadings are scaled so that the sum of the loadings for each latent factor is 2, to introduce a ceiling effect and to differentiate the dynamics of specific emotions. This is further empahsized by adding small noise to the measurement error variance for each observed variable (between -0.01 and 0.01).

Usage

```
generate_observables(X, num_steps, num_obs, error, loadings = 0.8)
```

Arguments

X	Matrix or Data frame. The (num_steps X 2) matrix of latent variable scores.
num_steps	Numeric integer. Number of time steps.
num_obs	Numeric integer. The number of observable variables per latent factor.
error	Numeric. Measurement error variance.
loadings	Numeric (default = 0.8). The default initial loading of the latent variable on the observable variable.

Value

A (num_steps X num_obs) Matrix or Data frame containing the observable variables.

---

generate_q *Generate a matrix of Dynamic Error values for the DLO simulation*

Description

This function generates a matrix of Dynamic Error values (q) for the DLO simulation.

Usage

```
generate_q(num_steps, sigma_q)
```

Arguments

num_steps	Numeric integer. The number of time steps used in the simulation.
sigma_q	Numeric. Standard deviation of the Dynamic Error/

Value

A (num_steps X 3) matrix of Dynamic Error values for neutral, negative and positive emotion latent score.

get_vision_model_config *Get Vision Model Configuration*

Description

Retrieve the configuration for a specific vision model from the registry.

Usage

```
get_vision_model_config(name)
```

Arguments

name	The name/alias of the registered model

Value

A list with model configuration, or NULL if model not found

image_scores *Calculate image scores using a Hugging Face CLIP model*

Description

This function takes an image file and a vector of classes as input and calculates the scores for each class using a specified Hugging Face CLIP model. Primary use of the function is to calculate FER scores - Facial Expression Detection of emotions based on detected facial expression in images. In case there are more than one face in the image, the function will return the scores of the face selected using the face_selection parameter. If there is no face in the image, the function will return NA for all classes. Function uses reticulate to call the Python functions in the image.py file. If you run this package/function for the first time it will take some time for the package to setup a functioning Python virtual environment in the background. This includes installing Python libraries for facial recognition and emotion detection in text, images and video. Please be patient.

Usage

```
image_scores(
  image,
  classes,
  face_selection = "largest",
  model = "oai-base",
  local_model_path = NULL,
  face_detector = c("yunet", "haar")
)
```

Arguments

image	The path to the image file or URL of the image.
classes	A character vector of classes to classify the image into.
face_selection	The face(s) to score. Can be "largest", "left", "right", "none", "all", face numbers, or a data frame of faces. Default is "largest" and will select the largest face in the image. "left" and "right" will select the face on the far left or the far right side of the image. "none" will use the whole image without cropping. These four return one row, as before. "all" scores every face, face numbers (such as `c(1, 3)`) score those faces, numbered from left to right as `detect_faces` numbers them, and a data frame from `detect_faces` or `select_faces` scores its faces in this image without detecting them again; these return one row per face (see `face_scores`).
model	A string specifying the vision model to use. Options include: • Built-in models: "oai-base" (default), "oai-large", "eva-8B", "jina-v2", "oai-base-fer" and "oai-large-fer" (OpenAI CLIP with the vision encoder fine-tuned on the FER2013 facial-expression dataset) • Any valid HuggingFace model ID, including checkpoints that hold only a fine-tuned CLIP vision encoder; their text encoder and processor come from the base CLIP model named in the checkpoint's config • Custom registered models (see `register_vision_model`) Use `list_vision_models` to see all available models. Note: Using large or untested models may cause memory issues or crashes.
local_model_path	Optional. Path to a local directory containing a pre-downloaded HuggingFace model. If provided, the model will be loaded from this directory instead of being downloaded from HuggingFace. This is useful for offline usage or for using custom fine-tuned models. On Linux/Mac, look in ~/.cache/huggingface/hub/ folder for downloaded models. Navigate to the snapshots folder for the relevant model and point to the directory which contains the config.json file. For example: "/home/username/.cache/huggingface/hub/mode cross-encoder–nli-distilroberta-base/snapshots/b5b020e8117e1ddc6a0c7ed0fd22c0e679edf0fa/" On Windows, the base path is C:\Users\USERNAME\.cache\huggingface\transformers\ Warning: Using very large models from local paths may cause memory issues or crashes depending on your system's resources.
face_detector	`"yunet"` (default), OpenCV's YuNet face detector, or `"haar"`, the Haar cascade used before version 0.1.8. Use `"haar"` to reproduce scores from earlier versions.

Details

Data Privacy: All processing is done locally with the downloaded model, and your images are never sent to any remote server or third-party.

Value

A data frame containing the scores for each class.

Author(s)

Aleksandar Tomasevic <atomashevic@gmail.com>

`image_scores_dir` *Calculate image scores for all images in a directory (fast batch)*

Description

This function scans a directory for image files and computes scores for each image using a Hugging Face CLIP model. It loads the model once and reuses text embeddings for speed, returning one row per image with the filename as image_id and probability columns for each class.

Usage

```
image_scores_dir(
  dir,
  classes,
  face_selection = "largest",
  pattern = "\\.(jpg|jpeg|png|bmp)$",
  recursive = FALSE,
  model = "oai-base",
  local_model_path = NULL,
  face_detector = c("yunet", "haar")
)
```

Arguments

`dir`	Path to a directory containing images.
`classes`	Character vector of labels/classes (length >= 2).
`face_selection`	Face selection strategy: "largest", "left", "right", or "none" (one row per image), "all" (one row per face, with faces of the same person sharing a `person_id`; see `detect_faces`), or a data frame from `detect_faces` (its faces in these images).
`pattern`	Optional regex to filter images (default supports common formats).
`recursive`	Whether to search subdirectories (default FALSE).
`model`	CLIP model alias or HuggingFace model id (see image_scores()).
`local_model_path`	Optional local path to a pre-downloaded model.
`face_detector`	`"yunet"` (default) or `"haar"`; see `image_scores`.

Value

A data.frame with columns: image_id and one column per class. With "all" or a data frame of faces, one row per face, with the columns of `face_scores` after image_id.

is_vision_model_registered *Check if Vision Model is Registered*

Description

Check if a vision model is available in the registry.

Usage

```
is_vision_model_registered(name)
```

Arguments

name	The name/alias of the model to check

Value

Logical indicating if the model is registered

list_vision_models *List Available Vision Models*

Description

List all vision models currently available in the transforEmotion registry.

Usage

```
list_vision_models(
  include_builtin = TRUE,
  architecture_filter = NULL,
  verbose = FALSE
)
```

Arguments

include_builtin	Logical indicating whether to include built-in models (default: TRUE)
architecture_filter	Optional character vector to filter by architecture type
verbose	Logical indicating whether to show detailed information (default: FALSE)

Value

A data.frame with model information, or detailed list if verbose=TRUE

Examples

```
# List all available models
list_vision_models()

# List only CLIP models
list_vision_models(architecture_filter = "clip")

# Get detailed information
list_vision_models(verbose = TRUE)

# See what models are available for image analysis
models <- list_vision_models()
print(paste("Available models:", paste(models$name, collapse = ", ")))
```

---

load_findingemo_annotations

Load FindingEmo-Light Annotations

Description

Loads and preprocesses annotations from a downloaded FindingEmo-Light dataset. Returns a clean R data.frame with emotion annotations, valence/arousal scores, and associated metadata.

Usage

```
load_findingemo_annotations(
  data_dir,
  output_format = c("dataframe", "list"),
  python_path = NULL
)
```

Arguments

data_dir	Character. Directory containing the downloaded FindingEmo data.
output_format	Character. Format for processed data: "dataframe" returns R data.frame, "list" returns full processed data (default: "dataframe").
python_path	Character. Path to Python executable (optional).

Details

This function loads the CSV annotation file and JSON URLs file from a downloaded FindingEmo dataset, performs basic validation and preprocessing, and returns the data in a format suitable for emotion analysis.

The function handles missing values, validates valence/arousal ranges, and provides summary statistics for the loaded data.

Value

If `output_format = "dataframe"`: A data.frame with columns:

- `image_id`: Unique image identifier
- `valence`: Valence score (emotion positivity)
- `arousal`: Arousal score (emotion intensity)
- Additional columns as present in the dataset

If `output_format = "list"`: A list containing:

- `annotations`: Data.frame with annotation data
- `urls`: List with image URL information
- `metadata`: List with dataset metadata

See Also

`download_findingemo_data`, `prepare_findingemo_evaluation`

Examples

```
## Not run:
# Download dataset first
download_result <- download_findingemo_data("./findingemo_data")

if (download_result$success) {
  # Load annotations as data.frame
  annotations <- load_findingemo_annotations("./findingemo_data")

  # Examine the data
  head(annotations)
  summary(annotations)

  # Get full processed data including metadata
  full_data <- load_findingemo_annotations(
    data_dir = "./findingemo_data",
    output_format = "list"
  )

  print(full_data$metadata)
}

## End(Not run)
```

`map_discrete_to_vad` *Map Discrete Emotions to VAD (Valence-Arousal-Dominance) Framework*

Description

Maps discrete emotion classifications from image_scores(), transformer_scores(), or video_scores() functions to the Valence-Arousal-Dominance (VAD) framework using published lexicons. Automatically downloads the NRC VAD lexicon via textdata package on first use.

Usage

```
map_discrete_to_vad(
  results,
  mapping = "nrc_vad",
  weighted = TRUE,
  cache_lexicon = TRUE,
  vad_lexicon = NULL
)
```

Arguments

`results`	Output from image_scores(), transformer_scores(), or video_scores(). Can be a data.frame (from image/video functions) or a list (from transformer functions).
`mapping`	Character. Which VAD mapping to use. Currently supports: • `"nrc_vad"`: Uses NRC VAD lexicon (Mohammad, 2018)
`weighted`	Logical. Whether to compute weighted averages based on confidence scores (default: TRUE). If FALSE, performs simple lookup of the highest-scoring emotion.
`cache_lexicon`	Logical. Whether to cache the VAD lexicon for repeated use (default: TRUE).
`vad_lexicon`	Optional data.frame. Pre-loaded VAD lexicon data to use instead of loading from textdata. Must have columns for word, valence, arousal, dominance (accepts both lowercase and capitalized versions, e.g., Word/word, Valence/valence). If provided, the function will use this data directly.

Details

This function bridges discrete emotion classification outputs with the continuous VAD emotion framework. The VAD model represents emotions in a three-dimensional space where:

- **Valence**: Pleasantness (positive/negative)
- **Arousal**: Activation level (excited/calm)
- **Dominance**: Control (dominant/submissive)

Input Type Detection: The function automatically detects the input type:

- **data.frame**: Assumes output from image_scores() or video_scores()
- **list**: Assumes output from transformer_scores()

Weighting Methods:

- **weighted = TRUE**: Computes weighted average VAD scores based on classification confidence scores
- **weighted = FALSE**: Uses VAD values for the highest-scoring emotion only

VAD Mappings: Currently supports the NRC VAD lexicon which provides VAD ratings for emotion words based on crowdsourced annotations. The lexicon must be downloaded first using 'textdata::lexicon_nrc_vad()' in an interactive session.

Setup Required: Before using this function, download the NRC VAD lexicon by running: 'textdata::lexicon_nrc_vad()' in an interactive R session and accepting the license.

Value

A data.frame with columns:

- `valence`: Valence score (positive vs negative emotion)
- `arousal`: Arousal score (excitement vs calmness)
- `dominance`: Dominance score (control vs submissiveness)

For transformer_scores() input, includes additional identifier columns. For image/video_scores() input, returns one row per input row.

Data Privacy

VAD lexicon is downloaded once and cached locally. No data is sent to external servers.

Author(s)

Aleksandar Tomasevic <atomashevic@gmail.com>

References

Mohammad, S. M. (2018). Obtaining reliable human ratings of valence, arousal, and dominance for 20,000 English words. Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 174-184.

Examples

```
## Not run:
# Method 1: Auto-load from textdata (requires prior download)
textdata::lexicon_nrc_vad()  # Run once to download

# With image scores
image_path <- system.file("extdata", "boris-1.png", package = "transforEmotion")
emotions <- c("joy", "sadness", "anger", "fear", "surprise", "disgust")
img_results <- image_scores(image_path, emotions)
vad_results <- map_discrete_to_vad(img_results)
```

```
# Method 2: Download once and pass as argument (recommended)
nrc_vad <- textdata::lexicon_nrc_vad()  # Download once

# Use with different emotion results
vad_results1 <- map_discrete_to_vad(img_results, vad_lexicon = nrc_vad)

text <- "I am so happy today!"
trans_results <- transformer_scores(text, emotions)
vad_results2 <- map_discrete_to_vad(trans_results, vad_lexicon = nrc_vad)

# Simple lookup (no weighting)
vad_simple <- map_discrete_to_vad(img_results, weighted = FALSE, vad_lexicon = nrc_vad)

## End(Not run)
```

`map_to_emo8` *Map FindingEmo Emotions to Emo8 Labels*

Description

Maps FindingEmo dataset emotion labels to the standard 8 basic emotions (Emo8) from Plutchik's emotion wheel. This function converts complex emotion labels to the 8 fundamental emotions using intensity-based mappings from the circumplex model.

Usage

```
map_to_emo8(findingemo_emotions)
```

Arguments

`findingemo_emotions`
: Character vector of FindingEmo emotion labels to map.

Details

The mapping is based on Plutchik's circumplex model of emotions, where complex emotions are mapped to their corresponding basic emotions:

Basic Emotions (direct mapping): - Joy, Trust, Fear, Surprise, Sadness, Disgust, Anger, Anticipation

Intensity Variations: - High intensity: Ecstasy→Joy, Admiration→Trust, Terror→Fear, etc. - Low intensity: Serenity→Joy, Acceptance→Trust, Apprehension→Fear, etc.

The 8 basic emotions (Emo8) are: joy, trust, fear, surprise, sadness, disgust, anger, anticipation.

Value

Character vector of mapped Emo8 emotion labels. Unmapped emotions return NA.

Examples

```
## Not run:
# Map single emotions
map_to_emo8("Joy")           # "joy"
map_to_emo8("Ecstasy")       # "joy"
map_to_emo8("Serenity")      # "joy"

# Map multiple emotions
findingemo_labels <- c("Joy", "Rage", "Terror", "Interest")
emo8_labels <- map_to_emo8(findingemo_labels)
# Returns: c("joy", "anger", "fear", "anticipation")

# Use in evaluation pipeline
annotations <- load_findingemo_annotations("./data")
annotations$emo8_label <- map_to_emo8(annotations$emotion)

## End(Not run)
```

MASS_mvrnorm *Multivariate Normal (Gaussian) Distribution*

Description

This function generates a random sample from the multivariate normal distribution with mean mu and covariance matrix Sigma.

Usage

```
MASS_mvrnorm(n = 1, mu, Sigma, tol = 1e-06, empirical = FALSE, EISPACK = FALSE)
```

Arguments

n	Numeric integer. The number of observations to generate.
mu	Numeric vector. The mean vector of the multivariate normal distribution.
Sigma	Numeric matrix. The covariance matrix of the multivariate normal distribution.
tol	Numeric. Tolerance for checking the positive definiteness of the covariance matrix.
empirical	Logical. Whether to return the empirical covariance matrix.
EISPACK	Logical. Whether to use the EISPACK routine instead of the LINPACK routine.

Value

A (n X p) matrix of random observations from the multivariate normal distribution. Updated: 26.10.2023.

neo_ipip_extraversion *NEO-PI-R IPIP Extraversion Item Descriptions*

Description

A list (length = 6) of the NEO-PI-R IPIP item descriptions (https://ipip.ori.org/newNEOFacetsKey.htm). Each vector within the 6 list elements contains the item descriptions for the respective Extraversion facets – friendliness, gregariousness, assertiveness, activity_level, excitement_seeking, and cheerfulness

Usage

```
data(neo_ipip_extraversion)
```

Format

A list (length = 6)

Examples

```
data("neo_ipip_extraversion")
```

nlp_scores *Natural Language Processing Scores*

Description

Natural Language Processing using word embeddings to compute semantic similarities (cosine; see `costring`) of text and specified classes. The suggested package **LSAfun** is loaded on use. It is not attached with transforEmotion, because loading it also loads **rgl**, which needs the system OpenGL and GLU libraries.

Usage

```
nlp_scores(
  text,
  classes,
  semantic_space = c("baroni", "cbow", "cbow_ukwac", "en100", "glove", "tasa"),
  preprocess = TRUE,
  remove_stop = TRUE,
  keep_in_env = TRUE,
  envir = 1
)
```

Arguments

`text`	Character vector or list. Text in a vector or list data format
`classes`	Character vector. Classes to score the text
`semantic_space`	Character vector. The semantic space used to compute the distances between words (more than one allowed). Here's a list of the semantic spaces: `"baroni"` Combination of British National Corpus, ukWaC corpus, and a 2009 Wikipedia dump. Space created using continuous bag of words algorithm using a context window size of 11 words (5 left and right) and 400 dimensions. Best word2vec model according to Baroni, Dinu, & Kruszewski (2014) `"cbow"` Combination of British National Corpus, ukWaC corpus, and a 2009 Wikipedia dump. Space created using continuous bag of words algorithm with a context window size of 5 (2 left and right) and 300 dimensions `"cbow_ukwac"` ukWaC corpus with the continuous bag of words algorithm with a context window size of 5 (2 left and right) and 400 dimensions `"en100"` Combination of British National Corpus, ukWaC corpus, and a 2009 Wikipedia dump. 100,000 most frequent words. Uses moving window model with a size of 5 (2 to the left and right). Positive pointwise mutual information and singular value decomposition was used to reduce the space to 300 dimensions `"glove"` Wikipedia 2014 dump and Gigaword 5 with 400,000 words (300 dimensions). Uses co-occurrence of words in text documents (uses cosine similarity) `"tasa"` Latent Semantic Analysis space from TASA corpus all (300 dimensions).Uses co-occurrence of words in text documents (uses cosine similarity)
`preprocess`	Boolean. Should basic preprocessing be applied? Includes making lowercase, keeping only alphanumeric characters, removing escape characters, removing repeated characters, and removing white space. Defaults to `TRUE`
`remove_stop`	Boolean. Should `stop_words` be removed? Defaults to `TRUE`
`keep_in_env`	Boolean. Whether the classifier should be kept in your global environment. Defaults to `TRUE`. By keeping the classifier in your environment, you can skip re-loading the classifier every time you run this function. `TRUE` is recommended
`envir`	Numeric. Environment for the classifier to be saved for repeated use. Defaults to the global environment

Value

Returns semantic distances for the text classes

Author(s)

Alexander P. Christensen <alexpaulchristensen@gmail.com>

References

Baroni, M., Dinu, G., & Kruszewski, G. (2014). Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd annual meting of the association for computational linguistics* (pp. 238-247).

Landauer, T.K., & Dumais, S.T. (1997). A solution to Plato's problem: The Latent Semantic Analysis theory of acquisition, induction and representation of knowledge. *Psychological Review*, *104*, 211-240.

Pennington, J., Socher, R., & Manning, C. D. (2014). GloVe: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing* (pp. 1532-1543).

Examples

```
# Load data
data(neo_ipip_extraversion)

# Example text
text <- neo_ipip_extraversion$friendliness[1:5]

## Not run:
# GloVe
nlp_scores(
 text = text,
 classes = c(
   "friendly", "gregarious", "assertive",
   "active", "excitement", "cheerful"
 )
)

# Baroni
nlp_scores(
 text = text,
 classes = c(
   "friendly", "gregarious", "assertive",
   "active", "excitement", "cheerful"
 ),
 semantic_space = "baroni"
)

# CBOW
nlp_scores(
 text = text,
 classes = c(
   "friendly", "gregarious", "assertive",
   "active", "excitement", "cheerful"
 ),
 semantic_space = "cbow"
)

# CBOW + ukWaC
nlp_scores(
```

```
 text = text,
 classes = c(
   "friendly", "gregarious", "assertive",
   "active", "excitement", "cheerful"
 ),
 semantic_space = "cbow_ukwac"
)

# en100
nlp_scores(
 text = text,
 classes = c(
   "friendly", "gregarious", "assertive",
   "active", "excitement", "cheerful"
 ),
 semantic_space = "en100"
)

# tasa
nlp_scores(
 text = text,
 classes = c(
   "friendly", "gregarious", "assertive",
   "active", "excitement", "cheerful"
 ),
 semantic_space = "tasa"
)

## End(Not run)
```

`parse_rag_json` *Parse RAG JSON*

Description

Parses a JSON string (or list) matching the enforced RAG schema and returns a normalized list: 'list(labels=chr, confidences=num, intensity=num, evidence=data.frame(doc_id, span, score))'.

Usage

```
parse_rag_json(x, validate = TRUE)
```

Arguments

`x`	JSON string or list.
`validate`	Logical; validate structure after parse.

Value

A normalized list with atomic vectors and an ‘evidence‘ data.frame.

Examples

```
j <- '{"labels":["joy"],"confidences":[0.9],
  "intensity":0.8,"evidence_chunks":[]}'
parse_rag_json(j)
```

`plot.emotion_evaluation` *Plot Evaluation Results*

Description

Creates visualizations for emotion evaluation results including confusion matrix heatmaps and per-class metrics bar plots.

Usage

```
## S3 method for class 'emotion_evaluation'
plot(x, type = "both", ...)
```

Arguments

`x`	An emotion_evaluation object from evaluate_emotions()
`type`	Character. Type of plot: "confusion_matrix", "metrics", or "both"
`...`	Additional arguments passed to plotting functions

Value

A ggplot object or list of ggplot objects

`plot_sim_emotions` *Plot the latent or the observable emotion scores.*

Description

Function to plot the latent or the observable emotion scores.

Usage

```
plot_sim_emotions(df, mode = "latent", title = " ")
```

Arguments

df	Data frame. The data frame containing the latent and observable variables created by the simulate_video function.
mode	Character. The mode of the plot. Can be either 'latent', 'positive' or 'negative'.
title	Character. The title of the plot. Default is an empty title, ' '.

Value

A plot of the latent or the observable emotion scores.

---

posterior_scores *Posterior emotions from frame scores*

Description

Combines each frame's scores with what the earlier (and, with method = "smooth", the later) frames showed, and returns the probability of each emotion in every frame and for the whole video. The scores are treated as noisy evidence, as learned by calibrate_scores, and the emotion is assumed to stay the same from one frame to the next with probability stay.

Usage

```
posterior_scores(
  scores,
  calibration,
  stay = 0.99,
  method = c("filter", "smooth")
)
```

Arguments

scores	A data frame of scores for one video, one row per frame, or a list of them, one per video (see calibrate_scores). Rows are put in time order by time or frame when the data frame has them. A row with a missing score (a frame without a face) adds no evidence: its posterior comes from the other frames.
calibration	The result of calibrate_scores.
stay	Probability that the emotion is the same in the next row (default 0.99); otherwise it changes to one of the other emotions, each equally likely. It is per row, so lower it when rows are far apart in time: 0.99 between frames 1/30 s apart means an emotion lasts about 3 s on average. Use 1 for one emotion per video.
method	"filter" (default): each frame's posterior uses that frame and the ones before it, as when following a video as it plays. "smooth": each frame's posterior uses all frames, before and after (the forward-backward algorithm).

Details

The posterior of emotion c in frame t is

$$P_t(c) \propto L_t(c) \sum_{c'} T(c' \to c) P_{t-1}(c')$$

where L_t is the frame's evidence (see `calibrate_scores`) and $T(c \to c) =$ `stay`. The first frame starts from equal probabilities. This is a hidden Markov model over the emotions, with the model's scores as noisy observations.

Value

A list with

frames `scores` (a data frame, or a list of them) in time order, with the class columns holding the posterior probabilities and a `label` column with the most probable emotion (replacing any `label` column already there). No class may be called `label`, `video` or `n_frames`.

clips A data frame with one row per video: `video` (the name in the list, or its number for a video without a name; `frames` is named the same way), `n_frames`, the posterior of each class and `label`. The clip posterior is the posterior at the last frame, which has seen every frame; it is the same for both methods. With `stay` near 1 it is the posterior of one emotion for the whole clip. For long videos in which the emotion changes, summarise `frames` instead, for example with `colMeans()`.

See Also

`calibrate_scores`, `video_scores`

Examples

```
set.seed(1)
clip <- function(n, emotion) {
  s <- matrix(runif(n * 3), n, 3,
              dimnames = list(NULL, c("neutral", "happy", "sad")))
  s[, "neutral"] <- s[, "neutral"] + 0.6
  s[, emotion] <- s[, emotion] + 0.5
  as.data.frame(s / rowSums(s))
}
scores <- list(a = clip(40, "happy"), b = clip(40, "sad"),
               c = clip(40, "neutral"), d = clip(40, "sad"))
calibration <- calibrate_scores(scores)

result <- posterior_scores(scores, calibration)
result$clips
head(result$frames$a)

# Each frame using the whole clip
smoothed <- posterior_scores(scores$a, calibration, method = "smooth")
table(smoothed$frames$label)
```

prepare_findingemo_evaluation
Prepare FindingEmo Data for Evaluation

Description

Prepares FindingEmo dataset annotations for use with `evaluate_emotions()`. Converts the dataset format to match the expected input structure for evaluation functions.

Usage

```
prepare_findingemo_evaluation(
  annotations,
  predictions,
  id_col = "image_id",
  truth_col = "emotion_label",
  pred_col = "predicted_emotion",
  include_va = TRUE
)
```

Arguments

`annotations`	Data.frame. Annotations from `load_findingemo_annotations()`.
`predictions`	Data.frame. Model predictions with same image IDs as annotations.
`id_col`	Character. Column name for image IDs (default: "image_id").
`truth_col`	Character. Column name for ground truth emotions (default: "emotion_label").
`pred_col`	Character. Column name for predicted emotions (default: "predicted_emotion").
`include_va`	Logical. Whether to include valence/arousal columns (default: TRUE).

Details

This function merges FindingEmo annotations with model predictions and formats the result for evaluation. It handles missing values, validates data consistency, and ensures the output matches the expected format for `evaluate_emotions()`.

Value

A data.frame formatted for use with `evaluate_emotions()`, containing:

- `id`: Image identifiers
- `truth`: Ground truth emotion labels
- `pred`: Predicted emotion labels
- `valence`: Valence scores (if available and include_va = TRUE)
- `arousal`: Arousal scores (if available and include_va = TRUE)
- Additional probability columns if present in predictions

See Also

load_findingemo_annotations, evaluate_emotions

Examples

```
## Not run:
# Load annotations
annotations <- load_findingemo_annotations("./findingemo_data")

# Create mock predictions (replace with actual model predictions)
predictions <- data.frame(
  image_id = annotations$image_id[1:100],
  predicted_emotion = sample(c("happy", "sad", "angry"), 100, replace = TRUE),
  prob_happy = runif(100),
  prob_sad = runif(100),
  prob_angry = runif(100)
)

# Prepare for evaluation
eval_data <- prepare_findingemo_evaluation(
  annotations = annotations,
  predictions = predictions
)

# Evaluate model performance
results <- evaluate_emotions(
  data = eval_data,
  probs_cols = c("prob_happy", "prob_sad", "prob_angry")
)

print(results)

## End(Not run)
```

---

print.emotion_evaluation

Print method for emotion evaluation results

Description

Print method for emotion evaluation results

Usage

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

Arguments

x	An emotion_evaluation object
...	Additional arguments (unused)

`punctuate` *Punctuation Removal for Text*

Description

Keeps the punctuations you want and removes the punctuations you don't

Usage

```
punctuate(
  text,
  allowPunctuations = c("-", "?", "'", "\"", ";", ",", ".", "!")
)
```

Arguments

text	Character vector or list. Text in a vector or list data format
allowPunctuations	Character vector. Punctuations that should be allowed in the text. Defaults to common punctuations in English text

Details

Coarsely removes punctuations from text. Keeps general punctuations that are used in most English language text. Apostrophes are much trickier. For example, not allowing "'" will remove apostrophes from contractions like "can't" becoming "cant"

Value

Returns text with only the allowed punctuations

Author(s)

Alexander P. Christensen <alexpaulchristensen@gmail.com>

Examples

```
# Load data
data(neo_ipip_extraversion)

# Example text
text <- neo_ipip_extraversion$friendliness

# Keep only periods
```

```
punctuate(text, allowPunctuations = c("."))
```

`python_requirements` *Python Requirements for transforEmotion*

Description

Returns the Python requirements that transforEmotion installs, in `requirements.txt` format. Use it to build a fixed Python environment, for example in a container image, and point transforEmotion at it with the `TRANSFOREMOTION_PYTHON` environment variable (see Details).

Usage

```
python_requirements(
  extras = character(),
  gpu = FALSE,
  sysname = Sys.info()[["sysname"]],
  machine = Sys.info()[["machine"]]
)
```

Arguments

`extras`	Character vector of optional feature sets to include; see `setup_modules`.
`gpu`	Logical. Use the CUDA build of PyTorch (Linux and Windows) instead of the CPU build.
`sysname`, `machine`	Target operating system and architecture, as returned by `Sys.info()`. Default to the current machine.

Details

The PyTorch requirements are direct wheel URLs built for Python 3.12, so the environment must use Python 3.12.

When `TRANSFOREMOTION_PYTHON` is set to a Python executable, transforEmotion uses that interpreter and does not install anything. This is the route for read-only environments such as Apptainer images, where uv cannot write to its cache.

A function that needs an optional feature set (`extras`), for example `rag()`, stops when the fixed environment does not have the Python modules of that set. The error names the missing modules, a command that installs their packages, and the `EXTRAS` build argument of `docker/Dockerfile`. Use `setup_modules` with the same `extras` to compare the whole environment with the requirements.

A fixed environment does not change when transforEmotion is updated, and a new version can add Python packages. After an update, rebuild the environment or the image from `python_requirements()`, or install the new packages into it. For a read-only image that you cannot rebuild, install a missing package into a writable folder on a machine with network access, for example `uv pip install --python <python> --target <folder> "protobuf>=3.20,<8"`, and start R with the environment variable `PYTHONPATH` set to that folder.

Value

A character vector of requirement specifiers.

Examples

```
# Requirements for a CPU environment with the rag() packages
python_requirements(extras = "rag")

## Not run:
# Write a requirements file and build the environment with uv (shell):
#   uv venv --python 3.12 /opt/te-venv
#   uv pip install --python /opt/te-venv/bin/python -r requirements.txt
writeLines(python_requirements(extras = "rag"), "requirements.txt")

## End(Not run)
```

---

rag *Retrieval-augmented Generation (RAG)*

Description

Performs retrieval-augmented generation {llama-index}

Supports multiple local LLM backends via HuggingFace and llama-index.

Usage

```
rag(
  text = NULL,
  path = NULL,
 transformer = c("TinyLLAMA", "Gemma3-1B", "Gemma3-4B", "Qwen2.5-1.5B", "Qwen2.5-3B",
    "Ministral-3B"),
 prompt = "You are an expert at extracting themes across many texts",
 query,
 response_mode = c("accumulate", "compact", "no_text", "refine", "simple_summarize",
    "tree_summarize"),
 similarity_top_k = 5,
 retriever = c("vector", "bm25"),
 retriever_params = list(),
 output = c("text", "json", "table", "csv"),
 task = c("general", "emotion", "sentiment"),
 labels_set = NULL,
 max_labels = 5,
 global_analysis = FALSE,
 device = c("auto", "cpu", "cuda"),
 temperature = NULL,
 do_sample = NULL,
```

```
  max_new_tokens = NULL,
  top_p = NULL,
  keep_in_env = TRUE,
  envir = 1,
  progress = TRUE
)
```

Arguments

text	Character vector or list. Text in a vector or list data format. path will override input into text Defaults to NULL
path	Character. Path to .pdfs stored locally on your computer. Defaults to NULL
transformer	Character. Large language model to use for RAG. Available models include: **"TinyLLAMA"** Default. TinyLlama 1.1B Chat via HuggingFace. Fast and light local inference. **"Gemma3-1B / Gemma3-4B"** Google's Gemma 3 Instruct via HuggingFace: google/gemma-3-1b-it, google/gemma-3-4b-it. Gemma 3 needs transformers >= 4.50. The Python environment that transforEmotion installs does not have it: it caps transformers below 4.47 for its llama-index 0.10 stack. Gemma 3 therefore works only with a fixed Python environment that has transformers >= 4.50, set with the TRANSFOREMOTION_PYTHON environment variable (see python_requirements). Such an environment also needs huggingface-hub >= 0.26, llama-index-core >= 0.12 and llama-index-llms-huggingface >= 0.4.2. Replace the transformers, huggingface-hub and llama-index-core caps in the python_requirements() output with these versions. transforEmotion is not tested with them. Before it loads a Gemma 3 model, rag() checks the transformers version and stops when it is older than 4.50. **"Qwen2.5-1.5B / Qwen2.5-3B"** Alibaba's Qwen 2.5 Instruct models via HuggingFace: Qwen/Qwen2.5-1.5B-Instruct, Qwen/Qwen2.5-3B-Instruct. Not gated (no Hugging Face token needed) and loadable with the transformers version the package installs. Support structured outputs like Gemma 3. **"Ministral-3B"** A 3B Instruct model with the Mistral architecture from the community repository ministral/Ministral-3b-instruct on HuggingFace. Mistral AI did not publish this model.
prompt	Character (length = 1). Prompt to feed into TinyLLAMA. Defaults to "You are an expert at extracting emotional themes across many texts"
query	Character. The query you'd like to know from the documents. Defaults to prompt if not provided
response_mode	Character (length = 1). Different responses generated from the model. See documentation here. "no_text" only retrieves the texts closest to query (in content) and loads no LLM Defaults to "tree_summarize"
similarity_top_k	Numeric (length = 1). Retrieves most representative texts given the query. Larger values will provide a more comprehensive response but at the cost of

computational efficiency; small values will provide a more focused response at the cost of comprehensiveness. Defaults to 5.

Values will vary based on number of texts but some suggested values might be:

40-60 Comprehensive search across all texts

20-40 Exploratory with good trade-off between comprehensive and speed

5-15 Focused search that should give generally good results

These values depend on the number and quality of texts. Adjust as necessary

`retriever` Character (length = 1). Retrieval backend: one of `"vector"` (default, semantic search using embeddings) or `"bm25"` (lexical BM25 search). BM25 uses llama-index's retriever when available and falls back to the Python `rank_bm25` implementation otherwise. Scores are normalized to [0,1] for consistency.

`retriever_params`
List. Optional parameters passed to the selected retriever handler. Reserved keys include `show_progress`.

`output` Character (length = 1). Output format: one of `"text"`, `"json"`, `"table"`, or `"csv"`.

- `"text"` (default): returns a free-text response with retrieved content.
- Structured outputs (`"json"`/`"table"`/`"csv"`) are supported ONLY for the instruction-tuned models Gemma3-1B, Gemma3-4B, Qwen2.5-1.5B and Qwen2.5-3B. For other models, requests for structured outputs fall back to `"text"`.
- For these models and task = `"sentiment"` or `"emotion"`, returns per-document dominant `label` and `confidence`.
- For these models and task = `"general"`, returns the prior schema with `labels`, `confidences`, `intensity`, and `evidence_chunks`.

`task` Character (length = 1). Task hint for structured extraction: one of `"general"`, `"emotion"`, or `"sentiment"`. When `"emotion"` or `"sentiment"`, the prompt constrains labels to a set (see `labels_set`).

`labels_set` Character vector. Allowed labels for classification when `task != "general"`. If `NULL`, defaults to Emo8 labels for `task = "emotion"` (`c("joy","trust","fear","surprise","sadnes`
`"disgust","anger","anticipation")`) for `task = "emotion"` and `c("positive","neutral","nega`
for `task = "sentiment"`.

`max_labels` Integer (length = 1). Maximum number of labels to return in structured outputs; used to guide the model instruction when `output != "text"`.

`global_analysis`
Boolean (length = 1). Whether to perform analysis across all documents globally (legacy behavior) or per-document (default). When `FALSE` (default), each document is analyzed individually then results are aggregated. When `TRUE`, all documents are processed together for a single global analysis. Defaults to `FALSE`.

`device` Character. Whether to use CPU or GPU for inference. Defaults to `"auto"` which will use GPU over CPU (if CUDA-capable GPU is setup). Set to `"cpu"` to perform over CPU

temperature	Numeric or NULL. Overrides the LLM sampling temperature when using local HF models. Recommended: 0.0–0.2 for structured/classification; 0.3–0.7 for summaries.
do_sample	Logical or NULL. If FALSE, forces greedy decoding for maximum determinism. Defaults are conservative; set explicitly for reproducibility.
max_new_tokens	Integer or NULL. Maximum new tokens to generate. Suggested: 64–128 for label decisions; 256–512 for summaries.
top_p	Numeric or NULL. Nucleus sampling parameter. Typical: 0.7–0.95. Use with do_sample=TRUE.
keep_in_env	Boolean (length = 1). Whether the classifier should be kept in your global environment. Defaults to TRUE. By keeping the classifier in your environment, you can skip re-loading the classifier every time you run this function. TRUE is recommended
envir	Numeric (length = 1). Environment for the classifier to be saved for repeated use. Defaults to the global environment
progress	Boolean (length = 1). Whether progress should be displayed. Defaults to TRUE

Value

For output = "text", returns an object of class "rag" with fields: $response (character), $content (data.frame), and $document_embeddings (matrix). For output = "json", returns a JSON character(1) string matching the enforced schema. For output = "table", returns a data.frame suitable for statistical analysis.

Data Privacy

All processing is done locally with the downloaded model, and your text is never sent to any remote server or third-party.

Author(s)

Alexander P. Christensen <alexpaulchristensen@gmail.com>

Examples

```
# Load data
data(neo_ipip_extraversion)

# Example text
text <- neo_ipip_extraversion$friendliness[1:5]

## Not run:
rag(
 text = text,
 query = "What themes are prevalent across the text?",
 response_mode = "tree_summarize",
 similarity_top_k = 5
)
```

```
# Structured outputs (Gemma 3 or Qwen 2.5)
rag(text = text, query = "Extract emotions", output = "json",
    transformer = "Qwen2.5-3B")
rag(text = text, query = "Extract emotions", output = "table",
    transformer = "Qwen2.5-3B")

## End(Not run)
```

`rag_json_utils` *RAG JSON utilities*

Description

Helpers for validating, parsing, and flattening structured output from ‘rag(output = "json"|"table")‘.

`rag_sentemo` *Structured Emotion/Sentiment via RAG (Small LLMs)*

Description

Convenience wrapper around `rag()` that keeps vector retrieval but simplifies getting structured outputs for emotion or sentiment analysis using small local LLMs (1–4B) with sensible defaults.

Usage

```
rag_sentemo(
  text = NULL,
  path = NULL,
  task = c("emotion", "sentiment"),
  labels_set = NULL,
  max_labels = 5,
 transformer = c("TinyLLAMA", "Gemma3-1B", "Gemma3-4B", "Qwen2.5-1.5B", "Qwen2.5-3B",
    "Ministral-3B"),
  similarity_top_k = 5,
  response_mode = c("compact", "refine", "simple_summarize"),
  output = c("table", "json", "csv"),
  global_analysis = FALSE,
  ...
)
```

Arguments

`text`	Character vector or list. Text to analyze. One entry per document.
`path`	Character. Optional directory with files to index (e.g., PDFs). If provided, overrides `text`.
`task`	Character. One of `"emotion"` or `"sentiment"`.
`labels_set`	Character vector of allowed labels. If `NULL`, defaults to Emo8 for `task = "emotion"` and c("positive","neutral","negative") for `task = "sentiment"`.
`max_labels`	Integer. Max number of labels to return.
`transformer`	Character. Small local LLM to use. One of: • `"TinyLLAMA"` (default) • `"Gemma3-1B"` • `"Gemma3-4B"` • `"Qwen2.5-1.5B"` • `"Qwen2.5-3B"` (not gated, no token needed) • `"Ministral-3B"` Gemma 3 needs transformers >= 4.50, which the Python environment that transforEmotion installs does not have: it caps transformers below 4.47 for its llama-index 0.10 stack. Gemma 3 therefore works only with a fixed environment that has transformers >= 4.50, set with `TRANSFOREMOTION_PYTHON` (see `rag`).
`similarity_top_k`	Integer. Retrieval depth per query. Default 5.
`response_mode`	Character. LlamaIndex response mode. Default `"compact"`.
`output`	Character. `"table"` (default) or `"json"`.
`global_analysis`	Logical. If TRUE, analyze all documents jointly. Default FALSE.
`...`	Additional arguments passed to `rag()` (e.g., `device`, `keep_in_env`).

Value

For Gemma 3 and Qwen 2.5 and `output = "table"/"csv"`, a data.frame with columns `doc_id, text, label, confidence`.
For Gemma 3 and Qwen 2.5 and `output = "json"`, a JSON array of per-doc objects with those fields.
For other models, structured outputs are not supported; the function falls back to `output = "text"` and returns a free-text `"rag"` object.

Examples

```
## Not run:
texts <- c(
  "I feel so happy and grateful today!",
  "This is frustrating and makes me angry."
)
rag_sentemo(texts, task = "emotion", output = "table")
rag_sentemo(texts, task = "sentiment", output = "json")
```

```
rag_sentemo(texts, task = "emotion", output = "table", transformer = "Qwen2.5-3B")

## End(Not run)
```

`read_annotations` *Read the answers of a face annotation task*

Description

Reads what coders answered in `annotate_faces`: the files in the task's `results` folder, and any exported files you add.

Usage

```
read_annotations(task, files = NULL, revisions = c("last", "all"))
```

Arguments

`task`	An `annotation_task`, or its folder.
`files`	Answer files to read, or `"te-annotations:..."` codes from the **Copy R code** button of a coding page in a notebook. Default: every `.jsonl` and `.json` file in the task's `results` folder.
`revisions`	`"last"` (default) keeps each coder's last answer to each item; `"all"` keeps every change, for checking.

Value

A data frame with one row per item, coder and question: `item_id`, `coder_id`, `question`, `value` (the answer as text, `NA` when the question was not asked or the item was skipped), `value_num` (the answer as a number, for Likert questions), `status` (`"answered"` or `"skipped"`), `revision`, `rt_ms` (time on the item), `shown_at`, `submitted_at` and `session_id`, followed by the item's face: `source`, `frame`, `frame_index`, `time`, `face`, `track_id` and `person_id`. For a task made from `items`: the item's `id`, `kind`, `media`, `reference` (its `label`), `prediction` and score columns.

See Also

`annotation_agreement`, `consensus_labels`

`read_face_selection` *Read faces chosen in the browser*

Description

Reads the file that **Export selection** in `view_faces(select = TRUE)` downloads, or the code in the line that **Copy R code** gives in a notebook, and returns the chosen faces.

Usage

```
read_face_selection(file, faces)
```

Arguments

`file`	Path of the exported `selection-<task>.json` file, its text (as copied from the page), or the `"te-selection:..."` code in the line that **Copy R code** gives in a notebook.
`faces`	The same data frame that was passed to `view_faces`, or the full `detect_faces` result it was taken from.

Value

The chosen rows of `faces`, which `face_scores` scores and `face_selection` in `image_scores`, `image_scores_dir` and `video_scores` accepts.

See Also

`view_faces`

Examples

```
## Not run:
faces <- detect_faces(system.file("extdata", "two-people.jpg", package = "transforEmotion"))
view_faces(faces, select = TRUE)
chosen <- read_face_selection("~/Downloads/selection-faces-20261011-101500.json", faces)

## End(Not run)
```

register_retriever *Register a custom retriever*

Description

Registers a retriever under a name. The handler should construct and return a query engine compatible with llama-index or a fallback with a ‘query_fn‘.

Usage

```
register_retriever(name, handler)
```

Arguments

name	Character scalar; retriever name (e.g., "my_retriever").
handler	Function with signature: function(llama_index, documents, similarity_top_k, response_mode, params) -> engine_or_list where the return value is either a Python query engine with ‘$query()‘ or a list with element ‘query_fn‘ taking a single ‘query‘ argument and returning a list with ‘response‘ and ‘source_nodes‘. Note: Settings are configured globally via llama_index.core.Settings.

register_vision_model *Register a Vision Model*

Description

Register a new vision model in the transforEmotion registry, making it available for use with image_scores(), video_scores(), and related functions.

Usage

```
register_vision_model(
  name,
  model_id,
  architecture = "clip",
  description = NULL,
  preprocessing_config = NULL,
  requires_special_handling = FALSE
)
```

Arguments

`name`	A short name/alias for the model (e.g., "my-custom-clip")
`model_id`	The HuggingFace model identifier or path to local model
`architecture`	The model architecture type. Currently supported: • `"clip"`: Standard CLIP dual-encoder models (default). Also accepts checkpoints that hold only a fine-tuned CLIP vision encoder (`model_type` "clip_vision_model"); the text encoder, projections and processor are then loaded from the base CLIP model named in the checkpoint's `config.json` ("_name_or_path") • `"clip-custom"`: CLIP variants requiring special handling • `"blip"`: BLIP captioning/VQA models (supported via BLIP adapter) • `"align"`: ALIGN dual-encoder models (supported via ALIGN adapter)
`description`	Optional description of the model
`preprocessing_config`	Optional list of preprocessing parameters
`requires_special_handling`	Logical indicating if the model needs custom processing beyond standard CLIP pipeline

Value

Invisibly returns TRUE if registration successful

Examples

```
## Not run:
# Register a custom CLIP model
register_vision_model(
  name = "my-emotion-clip",
  model_id = "j-hartmann/emotion-english-distilroberta-base",
  architecture = "clip",
  description = "Custom CLIP fine-tuned on emotion datasets"
)

# Register a local model
register_vision_model(
  name = "local-clip",
  model_id = "/path/to/local/model",
  architecture = "clip",
  description = "Locally stored fine-tuned model"
)

# Register experimental BLIP model
register_vision_model(
  name = "blip-caption",
  model_id = "Salesforce/blip-image-captioning-base",
  architecture = "blip",
  description = "BLIP model for image captioning"
)
```

```
## End(Not run)
```

`remove_vision_model` *Remove a Vision Model*

Description

Remove a custom vision model from the registry. Built-in models cannot be removed.

Remove a vision model from the transforEmotion registry.

Usage

```
remove_vision_model(name, confirm = TRUE)
```

```
remove_vision_model(name, confirm = TRUE)
```

Arguments

`name`	The name/alias of the model to remove
`confirm`	Logical indicating whether to show confirmation prompt (default: TRUE)

Value

Invisibly returns TRUE if successful

Invisibly returns TRUE if removal successful

Examples

```
## Not run:
# Remove a custom model
remove_vision_model("my-custom-model")

# Remove without confirmation prompt
remove_vision_model("my-custom-model", confirm = FALSE)

## End(Not run)
```

`select_faces` *Select faces to analyse*

Description

Keeps the faces of a `detect_faces` result that a rule, face numbers, tracks or people pick. The result can be passed to `face_scores`, or as `face_selection` to `image_scores` or `video_scores`. Ordinary subsetting (for example `faces[faces$person_id == "p1", ]`) works as well.

Usage

```
select_faces(
  faces,
  rule = c("all", "largest", "left", "right"),
  ids = NULL,
  tracks = NULL,
  persons = NULL
)
```

Arguments

`faces`	A data frame returned by `detect_faces`.
`rule`	`"all"` (default), or `"largest"`, `"left"` or `"right"` to keep one face per image or frame, with the rules of `face_selection` in `image_scores`.
`ids`	Face numbers to keep (the `face` column).
`tracks`	Video track ids to keep (the `track_id` column).
`persons`	Person ids to keep (the `person_id` column), for example `"p1"`.

Value

The selected rows of `faces`, with its attributes.

Examples

```
faces <- data.frame(
  source = "group.jpg", frame = NA, face = 1:3, person_id = c("p1", "p2", "p3"),
  x = c(10, 200, 400), y = 50, width = c(80, 120, 60), height = c(80, 120, 60)
)
select_faces(faces, rule = "largest")
select_faces(faces, persons = c("p1", "p3"))
```

sentence_similarity *Sentence Similarity*

Description

Uses sentence similarity models from huggingface to compute embeddings of `text` and `comparison_text` and returns the cosine similarity of each pair

Usage

```
sentence_similarity(
  text,
  comparison_text,
  transformer = c("all_minilm_l6"),
  device = c("auto", "cpu", "cuda"),
  preprocess = FALSE,
  keep_in_env = TRUE,
  envir = 1
)
```

Arguments

`text`	Character vector or list. Text in a vector or list data format
`comparison_text`	Character vector or list. Text in a vector or list data format
`transformer`	Character. Specific sentence similarity transformer to be used. Defaults to `"all_minilm_l6"` (see huggingface) Also allows any sentence similarity models with a pipeline from huggingface to be used by using the specified name (e.g., `"sentence-transformers/all-mpnet-base-v2"`; see Examples)
`device`	Character. Whether to use CPU or GPU for inference. Defaults to `"auto"` which will use GPU over CPU (if CUDA-capable GPU is setup). Set to `"cpu"` to perform over CPU. A classifier kept in `envir` moves to the device set by `"cpu"` or `"cuda"`. With `"auto"`, it stays on its current device
`preprocess`	Boolean. Should basic preprocessing be applied? Includes making lowercase, keeping only alphanumeric characters, removing escape characters, removing repeated characters, and removing white space. Defaults to `FALSE`. Transformers generally are OK without preprocessing and handle many of these functions internally, so setting to `TRUE` will not change performance much
`keep_in_env`	Boolean. Whether the classifier should be kept in your global environment. Defaults to `TRUE`. By keeping the classifier in your environment, you can skip re-loading the classifier every time you run this function. `TRUE` is recommended
`envir`	Numeric. Environment for the classifier to be saved for repeated use. Defaults to the global environment

Value

Returns a *n* x *m* similarity matrix where *n* is length of `text` and *m* is the length of `comparison_text`

Author(s)

Alexander P. Christensen <alexpaulchristensen@gmail.com>

Examples

```
# Load data
data(neo_ipip_extraversion)

# Example text
text <- neo_ipip_extraversion$friendliness[1:5]

## Not run:
# Example with defaults
sentence_similarity(
 text = text, comparison_text = text
)

# Example with model from 'sentence-transformers'
sentence_similarity(
 text = text, comparison_text = text,
 transformer = "sentence-transformers/all-mpnet-base-v2"
)


## End(Not run)
```

`setup_cache` *Use a Shared or Offline Cache*

Description

Stores the Python environment, Python itself and the Hugging Face models in one folder, and optionally runs without network access. This is the route for HPC clusters: prepare the folder on a login node with internet access, then use it offline on compute nodes.

Usage

```
setup_cache(path, offline = FALSE, gpu = NULL)
```

Arguments

path	Folder for the caches, for example on project or scratch storage shared by login and compute nodes. It needs several GB and must be writable.
offline	Logical. TRUE uses only what is already in the cache, with no network access (sets UV_OFFLINE and HF_HUB_OFFLINE). FALSE, the default, unsets both variables, so that setup_modules() can fill the cache, also when the environment turns offline mode on (the Docker image sets HF_HUB_OFFLINE, or a .Renviron file can set either).
gpu	Logical or NULL. Choose the CUDA (TRUE) or CPU (FALSE) build of PyTorch instead of detecting the GPU. Login nodes usually have no GPU, so when the compute nodes do, pass gpu = TRUE both when preparing the cache and in jobs.

Details

Call setup_cache() in a new R session, after library(transforEmotion) and before any analysis. It sets these environment variables for the session:

UV_CACHE_DIR uv package cache and Python environments

UV_PYTHON_INSTALL_DIR, UV_PYTHON_PREFERENCE a Python interpreter kept in the cache, so compute nodes do not need one

HF_HOME Hugging Face models

HF_HUB_CACHE, SENTENCE_TRANSFORMERS_HOME the model folder in HF_HOME, <path>/huggingface/hub. These variables take precedence over HF_HOME, so setup_cache() sets them too, for example over the values that the Docker image sets.

R_USER_CACHE_DIR reticulate's own copy of uv and the NLTK data used by rag(). This also moves the cache of any other R package that uses tools::R_user_dir() in this session.

It also unsets TRANSFORMERS_CACHE, PYTORCH_TRANSFORMERS_CACHE and PYTORCH_PRETRAINED_BERT_CACHE. transformers reads these older variables before HF_HUB_CACHE, and it warns in every session when one of them is set.

Offline sessions can use any combination of the feature sets prepared with setup_modules(), and any model it downloaded.

After you update transforEmotion, repeat the preparation once with network access: setup_cache() with the same path and gpu value but without offline = TRUE, then the same setup_modules() call. A new version can change the Python requirements, and an offline session cannot install them. Before Python starts, the first analysis function of an offline session checks with uv that the cache has every declared Python package. If one is missing, the function stops with uv's reason, for example that a package "was not found in the cache", and the setup_cache() and setup_modules() calls to run.

The cache must stay writable, because uv locks it even when reusing an environment. For read-only environments such as Apptainer images, build a fixed Python environment instead (see python_requirements).

Value

Invisibly, a named character vector of the environment variables set.

Examples

```
## Not run:
# On a login node, with internet access
library(transforEmotion)
setup_cache("/project/mylab/transforEmotion-cache", gpu = TRUE)
setup_modules(extras = "rag", models = "facebook/bart-large-mnli")

# In a job on a compute node, without internet access
library(transforEmotion)
setup_cache("/project/mylab/transforEmotion-cache", offline = TRUE, gpu = TRUE)
transformer_scores(text, classes, transformer = "facebook-bart")

## End(Not run)
```

---

setup_gpu_modules *Install GPU Python Modules*

Description

Adds `bitsandbytes`, used for 4-bit quantization of the EVA-CLIP-8B vision model, to the Python environment. Equivalent to `setup_modules(extras = "gpu", download_models = FALSE)`.

Usage

```
setup_gpu_modules()
```

Details

The CUDA build of PyTorch does not need this function: it is selected automatically when an NVIDIA GPU is detected (see `setup_modules`). `image_scores()` and `video_scores()` add `bitsandbytes` themselves when EVA-CLIP-8B is used on a GPU.

Note

This function requires NVIDIA GPU and drivers to be properly installed.

Author(s)

Alexander P. Christensen <alexpaulchristensen@gmail.com>

setup_miniconda *Deprecated: Miniconda setup (use uv instead)*

Description

setup_miniconda() is deprecated. The transforEmotion package now uses reticulate's uv-based ephemeral environments managed via 'py_require()'. No conda or Miniconda installation is required.

Usage

```
setup_miniconda()
```

setup_modules *Setup Required Python Modules*

Description

Builds the Python environment for transforEmotion ahead of first use and downloads the default models, so the first analysis call does not wait on either. Calling it is optional: every function sets up what it needs on first use.

Usage

```
setup_modules(
  extras = character(),
  download_models = TRUE,
  models = character()
)
```

Arguments

extras	Character vector of optional feature sets to install together with the core packages: "rag" LlamaIndex packages used by rag() "youtube" pytubefix, used by video_scores() for YouTube URLs "audio" av (PyAV, with FFmpeg), used by speech_scores() to read audio "findingemo" findingemo-light, used by the FindingEmo dataset functions "gpu" bitsandbytes, used for 4-bit quantization of EVA-CLIP-8B
download_models	Logical. Download the default text, image and sentence-similarity models into the Hugging Face cache (about 1 GB), the face recognition model detect_faces uses (10 MB), and, when extras includes "rag", the default embedding model and TinyLLAMA used by rag() (about 2.3 GB), and, when extras includes "audio", the LAION CLAP model used by speech_scores (about 800 MB). Default TRUE. Skipped when HF_HUB_OFFLINE is set.

`models` Character vector of additional Hugging Face model IDs to download, such as `"facebook/bart-large-mnli"`. Use it to prepare models for offline use (see `setup_cache`).

Details

Python requirements are declared with `py_require`. reticulate resolves them with uv into a cached environment, so after the first run later sessions start in under a second. No conda installation or manual uv installation is needed; reticulate downloads uv when it is not on the PATH.

PyTorch is installed as a CPU-only build unless an NVIDIA GPU is detected. Set `TE_FORCE_CPU=1` to always use the CPU build, or `TRANSFOREMOTION_USE_GPU=1` to always use the CUDA build. The choice is made when the package is loaded, so set either variable in `.Renviron` or before `library(transforEmotion)`, in a new R session.

When `TRANSFOREMOTION_PYTHON` is set, `setup_modules()` installs nothing. It compares the packages in that environment with the requirements of the core set and of `extras` (see `python_requirements`), and gives a warning that names each package that is missing or has a version outside its requirement, with a command that installs them.

Value

Invisibly returns `NULL`.

Author(s)

Alexander P. Christensen <alexpaulchristensen@gmail.com>

Examples

```
## Not run:
# Core packages and default models
setup_modules()

# Also install the packages rag() needs
setup_modules(extras = "rag")

# Prepare an additional model for offline use
setup_modules(models = "facebook/bart-large-mnli")

## End(Not run)
```

---

setup_popular_models *Quick Setup for Popular Models*

Description

Convenience function to quickly add popular vision models with pre-configured settings.

Usage

```
setup_popular_models(models)
```

Arguments

models	Character vector of model shortcuts to add. Available options: • "blip-base": BLIP base model for image captioning and VQA • "blip-large": BLIP large model for better performance • "align-base": ALIGN base model for image-text alignment

Value

Invisibly returns TRUE if all models added successfully

Examples

```
## Not run:
# Add BLIP models for image captioning
setup_popular_models("blip-base")

# Add multiple experimental models at once
setup_popular_models(c("blip-base", "blip-large", "align-base"))

# Then use them in your analysis
list_vision_models()  # See all available models
result <- image_scores("image.jpg", c("happy", "sad"), model = "blip-base")

## End(Not run)
```

---

show_vision_models *Show Available Vision Models*

Description

Display all available vision models in a user-friendly format with additional details and usage hints.

Usage

```
show_vision_models(show_details = FALSE, filter_by = NULL)
```

Arguments

show_details	Logical indicating whether to show detailed information
filter_by	Optional character vector to filter by architecture type

Value

Invisibly returns the models data.frame

Examples

```
# Show all models
show_vision_models()

# Show only CLIP models
show_vision_models(filter_by = "clip")

# Show detailed information
show_vision_models(show_details = TRUE)
```

`simulate_video` *Simulate latent and observed emotion scores for a single "video"*

Description

This function simulates emotions in a video using the DLO model implemented as continuous time state space model. The function takes in several parameters, including the time step, number of steps, number of observables, and various model parameters. It returns a data frame containing the simulated emotions and their derivatives, as well as smoothed versions of the observables. The initial state of the video is always the same. Neutral score is 0.5 and both positive and negative emotion score is 0.25. To simulate more realistic time series, there is an option of including a sudden jump in the emotion scores. This is done by emphasizing the effect of the dominant emotion during the period where the derivative of the latent variable is high. The observable value of the strongest emotion from the positive or negative group will spike in the next k time step (emph.dur). The probability of this happening is p at each time step in which the derivative of the latent variable is greater than 0.2. The jump is proportionate to the derivative of the latent variable and the sum of the observable values of the other emotions.

Usage

```
simulate_video(
  dt,
  num_steps,
  num_observables,
  eta_n,
  zeta_n,
  eta,
  zeta,
  sigma_q,
  sd_observable,
  loadings,
  window_size,
  emph = FALSE,
  emph.dur = 10,
  emph.prob = 0.5
)
```

Arguments

dt	Numeric real. The time step for the simulation (in minutes).
num_steps	Numeric real. Total length of the video (in minutes).
num_observables	Numeric integer. The number of observables to generate per factor. Total number of observables generated is 2 x num_observables.
eta_n	Numeric. The eta parameter for the neutral state.
zeta_n	Numeric. The zeta parameter for the neutral state.
eta	Numeric. The eta parameter for the positive and negative emotions.
zeta	Numeric. The zeta parameter for the positive and negative emotions.
sigma_q	Numeric. The standard deviation of Dynamic Error of the q(t) function.
sd_observable	Numeric. The standard deviation of the measurement error.
loadings	Numeric (default = 0.8). The default initial loading of the latent variable on the observable variable.
window_size	Numeric integer. The window size for smoothing the observables.
emph	Logical. Whether to emphasize the effect of dominant emotion (default is FALSE).
emph.dur	Numeric integer. The duration of the emphasis (default is 10).
emph.prob	Numeric. The probability of the dominant emotion being emphasized (default is 0.5).

Value

A data frame (num_steps X (6 + num_observables)) containing the latent scores for neutral score, positive emotions, negative emotions and their derivatives, as well as smoothed versions of the observables.

Examples

```
simulate_video(dt = 0.01, num_steps = 50, num_observables = 4,
               eta_n = 0.5, zeta_n = 0.5,
               eta = 0.5, zeta = 0.5,
               sigma_q = 0.1, sd_observable = 0.1,
               loadings = 0.8, window_size = 10)
```

---

speech_scores *Speech Emotion Scores for Segments of an Audio or Video File*

Description

Splits the audio track of an audio file, a video file or a YouTube video into segments and scores each segment against user-defined classes with a CLAP audio-text model (zero-shot). Each segment's scores are the softmax over the classes of the similarity between the segment's audio embedding and each class's text embedding, as `image_scores` does with CLIP.

Usage

```
speech_scores(
  audio,
  classes,
  nsegments = 100,
  segment_length = 5,
  start = 0,
  end = -1,
  uniform = FALSE,
  model = "laion-clap",
  local_model_path = NULL
)
```

Arguments

audio	Path to an audio or video file (any format FFmpeg reads, such as WAV, MP3, FLAC, OGG, M4A or MP4), or the URL of a YouTube video (only its audio is downloaded, to a temporary file that is removed afterwards).
classes	A character vector of at least two classes. CLAP compares the audio with each class as text, so short descriptions of the voice (for example "a person speaking angrily") can work better than single words.
nsegments	The maximum number of segments to score. Default is 100.
segment_length	Segment length in seconds, at most 10 (CLAP reads 10 seconds of audio). Default is 5.
start	The start time, in seconds, of the audio range to analyze. Default is 0.
end	The end time, in seconds, of the audio range to analyze. Default is -1: the end of the audio.
uniform	Logical. FALSE (default) scores consecutive segments from start, at most nsegments. TRUE spreads at most nsegments segments evenly over the range, without overlap.
model	The CLAP model: "laion-clap" (default, laion/larger_clap_music_and_speech, trained on general audio, music and speech) or the Hugging Face ID of another CLAP model, such as "laion/larger_clap_general".
local_model_path	Optional. Path to a local directory that holds a downloaded CLAP model (the folder with config.json); it is used instead of model.

Details

The audio is converted to mono at the model's sampling rate (48 kHz). With uniform = FALSE, a last segment shorter than one second is not scored, unless it is the only segment. A range no longer than segment_length is scored as one segment. Segments shorter than 10 seconds are repeated to fill CLAP's 10-second input.

General-purpose CLAP models separate speech emotions much less well than CLIP separates facial expressions. On RAVDESS (1440 acted English clips, eight emotions), "laion-clap" had an unweighted average recall of 0.13 with one-word classes such as "angry" and 0.17 with classes

such as "a person speaking angrily", where chance is 0.125. Check the scores against labelled examples from your own data before you use them.

Value

A data frame with one row per segment: segment (1, 2, ...), start and end (the segment's times in the file, in seconds) and one column of scores per class. The scores of a row sum to 1.

Data Privacy

All processing is done locally with the downloaded model, and your audio is never sent to any remote server or third-party.

Author(s)

Aleksandar Tomasevic <atomashevic@gmail.com>

See Also

video_scores for the frames of the same video.

Examples

```
## Not run:
speech_scores("interview.wav",
              classes = c("a person speaking angrily", "a person speaking happily",
                          "a person speaking sadly", "a person speaking calmly"),
              nsegments = 20, segment_length = 5)

## End(Not run)
```

---

stop_words *Stop Words from the* tm *Package*

Description

174 English stop words in the *tm* package

Usage

```
data(stop_words)
```

Format

A vector (length = 174)

Examples

```
data("stop_words")
```

summary.emotion_evaluation *Summary method for emotion evaluation results*

Description

Summary method for emotion evaluation results

Usage

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

Arguments

object	An emotion_evaluation object
...	Additional arguments (unused)

te_cleanup_default_venv *Remove reticulate's default virtualenv (r-reticulate)*

Description

Removes the default 'reticulate' virtual environment at '~/.virtualenvs/r-reticulate' to avoid conflicts with uv-managed environments used by transforEmotion. This is optional and only needed if you want to ensure uv's environment is preferred.

Usage

```
te_cleanup_default_venv(confirm = TRUE)
```

Arguments

confirm	Logical. Ask for confirmation before removal. Default TRUE.

Value

Invisibly returns TRUE on success, FALSE otherwise.

Examples

```
## Not run:
te_cleanup_default_venv()
te_cleanup_default_venv(confirm = FALSE)

## End(Not run)
```

tinytrolls *Russian Trolls Data - Small Version*

Description

A matrix containing a smaller subset of tweets from the `trolls` dataset, useful for test purposes. There are approximately 20,000 tweets from 50 authors. This dataset includes only authored tweets by each account; retweets, reposts, and repeated tweets have been removed. The original data was provided by FiveThirtyEight and Clemson University researchers Darren Linvill and Patrick Warren. For more information, visit https://github.com/fivethirtyeight/russian-troll-tweets

Usage

```
data(tinytrolls)
```

Format

A data frame with 22,143 rows and 6 columns.

content A tweet.

author The name of the handle that authored the tweet.

publish_date The date the tweet was published on.

followers How many followers the handle had at the time of posting.

updates How many interactions (including likes, tweets, retweets) the post garnered.

account_type Left or Right

Examples

```
data(tinytrolls)
```

transformer_scores *Sentiment Analysis Scores*

Description

Uses sentiment analysis pipelines from huggingface to compute probabilities that the text corresponds to the specified classes

Usage

```
transformer_scores(
  text,
  classes,
  multiple_classes = FALSE,
  transformer = c("cross-encoder-roberta", "cross-encoder-distilroberta",
    "facebook-bart"),
  device = c("auto", "cpu", "cuda"),
  preprocess = FALSE,
  keep_in_env = TRUE,
  envir = 1,
  local_model_path = NULL
)
```

Arguments

`text`	Character vector or list. Text in a vector or list data format
`classes`	Character vector. Classes to score the text
`multiple_classes`	Boolean. Whether the text can belong to multiple true classes. Defaults to `FALSE`. Set to `TRUE` to get scores with multiple classes
`transformer`	Character. Specific zero-shot sentiment analysis transformer to be used. Default options: `"cross-encoder-roberta"` Uses Cross-Encoder's Natural Language Interface RoBERTa Base zero-shot classification model trained on the Stanford Natural Language Inference (SNLI) corpus and MultiNLI datasets `"cross-encoder-distilroberta"` Uses Cross-Encoder's Natural Language Interface DistilRoBERTa Base zero-shot classification model trained on the Stanford Natural Language Inference (SNLI) corpus and MultiNLI datasets. The DistilRoBERTa is intended to be a smaller, more lightweight version of `"cross-encoder-roberta"`, that sacrifices some accuracy for much faster speed (see https://www.sbert.net/docs/cross_encoder/pretrained_models.html#nli) `"facebook-bart"` Uses Facebook's BART Large zero-shot classification model trained on the Multi-Genre Natural Language Inference (MultiNLI) dataset Defaults to `"cross-encoder-distilroberta"` Also allows any other model on the Hugging Face Hub with the zero-shot-classification pipeline tag. Give its full model ID, `"owner/name"`, exactly as the model page shows it (for example, `"typeform/distilbert-base-uncased-mnli"`; see Examples). The model is downloaded on first use. The three default models are trained on English text. For text in other languages, use a multilingual model, for example `"joeddav/xlm-roberta-large-xnli"` (XLM-RoBERTa large fine-tuned on MultiNLI and XNLI, 15 languages, a download of about 2.2 GB). The classes can be in the language of the text. There is no `"facebook/xlm-roberta-large-xnli"`: a model ID that does not exist fails with "is not a valid model identifier". For offline use, download the model first with `setup_modules(models = "joeddav/xlm-roberta-large-xnli")`.

Note: Using custom HuggingFace model IDs beyond the recommended models is done at your own risk. Large models may cause memory issues or crashes, especially on systems with limited resources. The package has been optimized and tested with the recommended models listed above.

`device` Character. Whether to use CPU or GPU for inference. Defaults to `"auto"` which will use GPU over CPU (if CUDA-capable GPU is setup). Set to `"cpu"` to perform over CPU

`preprocess` Boolean. Should basic preprocessing be applied? Includes making lowercase, keeping only alphanumeric characters, removing escape characters, removing repeated characters, and removing white space. Defaults to `FALSE`. Transformers generally are OK without preprocessing and handle many of these functions internally, so setting to `TRUE` will not change performance much

`keep_in_env` Boolean. Whether the classifier should be kept in your global environment. Defaults to `TRUE`. By keeping the classifier in your environment, you can skip re-loading the classifier every time you run this function. `TRUE` is recommended. The classifier is kept under the name in `transformer`. A later call reuses it only if both calls give the same `local_model_path`, or neither gives one

`envir` Numeric. Environment for the classifier to be saved for repeated use. Defaults to the global environment

`local_model_path`

Optional. Path to a local directory containing a pre-downloaded HuggingFace model. If provided, the model will be loaded from this directory instead of being downloaded from HuggingFace. This is useful for offline usage or for using custom fine-tuned models.

On Linux/Mac, look in ~/.cache/huggingface/hub/ folder for downloaded models. Navigate to the snapshots folder for the relevant model and point to the directory which contains the config.json file. For example: "/home/username/.cache/huggingface/hub/mode cross-encoder–nli-distilroberta-base/snapshots/b5b020e8117e1ddc6a0c7ed0fd22c0e679edf0fa/"

On Windows, the base path is C:\Users\USERNAME\.cache\huggingface\transformers\

Warning: Using very large models from local paths may cause memory issues or crashes depending on your system's resources.

Value

Returns probabilities for the text classes

Data Privacy

All processing is done locally with the downloaded model, and your text is never sent to any remote server or third-party.

Author(s)

Alexander P. Christensen <alexpaulchristensen@gmail.com>

References

BART
Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., ... & Zettlemoyer, L. (2019). Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

RoBERTa
Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., ... & Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Zero-shot classification
Yin, W., Hay, J., & Roth, D. (2019). Benchmarking zero-shot text classification: Datasets, evaluation and entailment approach. *arXiv preprint arXiv:1909.00161*.

MultiNLI dataset
Williams, A., Nangia, N., & Bowman, S. R. (2017). A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*.

Examples

```
# Load data
data(neo_ipip_extraversion)

# Example text
text <- neo_ipip_extraversion$friendliness[1:5]

## Not run:
# Cross-Encoder DistilRoBERTa
transformer_scores(
 text = text,
 classes = c(
   "friendly", "gregarious", "assertive",
   "active", "excitement", "cheerful"
 )
)

# Facebook BART Large
transformer_scores(
 text = text,
 classes = c(
   "friendly", "gregarious", "assertive",
   "active", "excitement", "cheerful"
 ),
 transformer = "facebook-bart"
)

# Directly from huggingface: typeform/distilbert-base-uncased-mnli
transformer_scores(
 text = text,
 classes = c(
   "friendly", "gregarious", "assertive",
   "active", "excitement", "cheerful"
 ),
```

```
 transformer = "typeform/distilbert-base-uncased-mnli"
)

# Text in another language: a multilingual model (about 2.2 GB)
transformer_scores(
 text = "Estoy muy contento de verte hoy",
 classes = c("alegría", "ira", "miedo"),
 transformer = "joeddav/xlm-roberta-large-xnli"
)

## End(Not run)
```

---

vad_scores *Direct VAD (Valence-Arousal-Dominance) Prediction*

Description

Directly predicts VAD dimensions using classification with definitional labels, bypassing the intermediate step of discrete emotion classification. This approach uses rich, educational descriptions of each VAD pole to help transformer models understand the psychological concepts and make more accurate predictions.

Usage

```
vad_scores(
  input,
  input_type = "auto",
  dimensions = c("valence", "arousal", "dominance"),
  label_type = "definitional",
  custom_labels = NULL,
  model = "auto",
  ...
)
```

Arguments

input	Input data. Can be: • Character: Text string, image file path, or video URL • Character vector: Multiple texts or image paths • List: Multiple text strings
input_type	Character. Type of input data: • "auto": Automatically detect based on input (default) • "text": Text input for transformer classification • "image": Image file path(s) for visual classification • "video": Video URL(s) for video analysis

`dimensions` Character vector. Which VAD dimensions to predict:

- `"valence"`: Positive vs negative emotional experience
- `"arousal"`: High vs low activation/energy
- `"dominance"`: Control vs powerlessness

Default: all three dimensions

`label_type` Character. Type of labels to use:

- `"definitional"`: Rich descriptive labels with definitions (default)
- `"simple"`: Basic polar labels (positive/negative, etc.)
- `"custom"`: User-provided custom labels

`custom_labels` Optional list. Custom labels when label_type = "custom". Must follow structure: list(valence = list(positive = "...", negative = "..."), ...). Give an element for each dimension in `dimensions`. The poles are `positive` and `negative` for valence, and `high` and `low` for arousal and dominance.

`model` Character. The model that scores each dimension. `"auto"` (default) uses `"cross-encoder-distilrobert` for text and `"oai-base"` for images and video. Any other value goes unchanged to the scoring function for the input type, so the same values work:

- Text: the `transformer` argument of `transformer_scores`: a built-in name or the full ID of a Hugging Face model with the zero-shot-classification pipeline tag. For text in other languages, use a multilingual model, for example `"joeddav/xlm-roberta-large-xnli"`
- Image: the `model` argument of `image_scores`
- Video: the `model` argument of `video_scores`

Give the model here, not as `transformer` in `....` The definitional and simple labels are in English; to give labels in the language of the text, use `label_type = "custom"`.

`...` Additional arguments passed to underlying classification functions (`transformer_scores`, `image_scores`, or `video_scores`), for example `local_model_path`. `vad_scores()` sets the classes and the model itself, so `classes` and `transformer` cannot be given here.

Details

This function implements direct VAD prediction using the approach: Input → VAD Classification → VAD Scores

Instead of mapping from discrete emotions, each VAD dimension is treated as a separate binary classification task using definitional labels that explain the psychological concepts.

Definitional Labels (default): The function uses rich descriptions that educate the model about each dimension:

- **Valence**: "Positive emotional experience: pleasant, joyful, ..."
- **Arousal**: "High energy emotional state: intense, excited, ..."
- **Dominance**: "Feeling in control: powerful, confident, ..."

Input Type Detection: When input_type = "auto", the function detects input type based on:

- URLs starting with "http": Video
- File paths with image extensions: Image
- Everything else: Text

Score Interpretation: Scores represent the probability that the input exhibits the "high" pole:

- **Valence**: 1.0 = very positive, 0.0 = very negative
- **Arousal**: 1.0 = high energy, 0.0 = very calm
- **Dominance**: 1.0 = very controlling, 0.0 = very powerless

Value

A data.frame with columns:

- `input_id`: Identifier for each input (text content, filename, or index)
- `valence`: Valence score (0-1, where 1 = positive)
- `arousal`: Arousal score (0-1, where 1 = high arousal)
- `dominance`: Dominance score (0-1, where 1 = high dominance)

Only requested dimensions are included in output.

Data Privacy

All processing is done locally with downloaded models. Data is never sent to external servers.

Author(s)

Aleksandar Tomasevic <atomashevic@gmail.com>

References

Russell, J. A. (1980). A circumplex model of affect. Journal of Personality and Social Psychology, 39(6), 1161-1178.

Bradley, M. M., & Lang, P. J. (1994). Measuring emotion: the self-assessment manikin and the semantic differential. Journal of Behavior Therapy and Experimental Psychiatry, 25(1), 49-59.

Examples

```
## Not run:
# Text VAD analysis
texts <- c("I'm absolutely thrilled!", "I feel so helpless and sad", "This is boring")
text_vad <- vad_scores(texts, input_type = "text")
print(text_vad)

# Image VAD analysis
image_path <- system.file("extdata", "boris-1.png", package = "transforEmotion")
image_vad <- vad_scores(image_path, input_type = "image")
print(image_vad)
```

```
# Single dimension prediction
valence_only <- vad_scores(texts, dimensions = "valence")

# Using simple labels for speed
simple_vad <- vad_scores(texts, label_type = "simple")

# Custom labels for domain-specific applications
custom_labels <- list(
  valence = list(
    positive = "Customer satisfaction and positive brand sentiment",
    negative = "Customer complaints and negative brand sentiment"
  )
)
brand_vad <- vad_scores(texts, dimensions = "valence",
                        label_type = "custom", custom_labels = custom_labels)

## End(Not run)
```

`validate_rag_json` *Validate a RAG JSON structure*

Description

Ensures the object has the expected fields and types: ‘labels‘, ‘confidences‘, ‘intensity‘, and ‘evidence_chunks‘.

Usage

```
validate_rag_json(x, error = TRUE)
```

Arguments

`x`	A list (parsed JSON) to validate.
`error`	Logical; if TRUE, throws an error on invalid input.

Value

Invisibly returns TRUE when valid; otherwise FALSE or error.

validate_rag_predictions *Validate RAG Emotion/Sentiment Predictions*

Description

Evaluates emotion/sentiment predictions from rag() or rag_sentemo() against ground truth labels using the same metrics pipeline as evaluate_emotions(). Supports table or JSON structured outputs.

Usage

```
validate_rag_predictions(
  rag_output,
  ground_truth,
  id_col = NULL,
  task = c("emotion", "sentiment"),
  labels_set = NULL,
  metrics = c("accuracy", "f1_macro", "confusion_matrix"),
  return_plot = FALSE
)
```

Arguments

rag_output	Output from rag() or rag_sentemo() with structured outputs (data.frame with columns like ‘doc_id‘, ‘label‘, ‘confidence‘; or JSON string with these fields). Global schema outputs with ‘labels‘/‘confidences‘ are also handled by reducing to the top label.
ground_truth	Character vector of ground truth labels matching the number of predictions (or length of provided ids).
id_col	Optional identifier. If ‘rag_output‘ is a data.frame and ‘id_col‘ is a character scalar naming a column present in it, that column is used as the prediction id. Alternatively, ‘id_col‘ can be a vector of ids (same length as ‘ground_truth‘) used to align ground truth to the predictions by merge.
task	Task type: one of ‘"emotion"‘ or ‘"sentiment"‘ (used for metadata and optional label set enforcement).
labels_set	Optional character vector of allowed labels for validation. If provided, predictions will be lowercased and filtered to this set where possible.
metrics	Metrics to compute, forwarded to evaluate_emotions() (e.g., ‘c("accuracy","f1_macro","confusion_matrix
return_plot	Logical; whether to include plotting helpers.

Value

A list of evaluation results in the same format as evaluate_emotions(), augmented with ‘$rag_metadata‘ summarizing RAG-specific context (documents, transformer, task).

Examples

```
## Not run:
texts <- c(
  "I feel so happy and grateful today!",
  "This is frustrating and makes me angry.",
  "I'm not sure how I feel about this."
)

# Get predictions (structured per-document output)
rag_results <- rag_sentemo(
  texts,
  task = "emotion",
  output = "table",
  transformer = "Qwen2.5-3B"
)

# Ground truth labels
ground_truth <- c("joy", "anger", "neutral")

# Validate predictions
validation_results <- validate_rag_predictions(
  rag_output = rag_results,
  ground_truth = ground_truth,
  task = "emotion",
  metrics = c("accuracy", "f1_macro", "confusion_matrix"),
  return_plot = TRUE
)

## End(Not run)
```

---

video_scores *Run FER on a YouTube video using a Hugging Face CLIP model*

Description

This function retrieves facial expression recognition (FER) scores from a specific number of frames extracted from a YouTube video using a specified Hugging Face CLIP model. It utilizes Python libraries for facial recognition and emotion detection in text, images, and video.

Usage

```
video_scores(
  video,
  classes,
  nframes = 100,
  face_selection = "largest",
  start = 0,
  end = -1,
```

```
  uniform = FALSE,
  ffreq = 15,
  save_video = FALSE,
  save_frames = FALSE,
  save_dir = "temp/",
  video_name = "temp",
  model = "oai-base",
  local_model_path = NULL,
  face_detector = c("yunet", "haar")
)
```

Arguments

video	The URL of the YouTube video to analyze.
classes	A character vector specifying the classes to analyze.
nframes	The number of frames to analyze in the video. Default is 100.
face_selection	The method for selecting faces in the video. Options are "largest", "left", "right", "none", "all", or a data frame of faces. Default is "largest". Use "none" to classify the whole frame without face cropping. These four pick one face per frame and return one row per frame, as before. "all" scores every face in every frame, and a data frame from detect_faces or select_faces (for example the faces of one person) scores those faces; both return one row per face with its frame, time, track and person (see face_scores). A data frame of faces brings its own frame sampling settings, so nframes, start, end, uniform and ffreq are ignored.
start	The start time of the video range to analyze. Default is 0.
end	The end time of the video range to analyze. Default is -1 and this means that video won't be cut. If end is a positive number greater than start, the video will be cut from start to end.
uniform	Logical indicating whether to uniformly sample frames from the video. Default is FALSE.
ffreq	The frame frequency for sampling frames from the video. Default is 15.
save_video	Logical indicating whether to save the analyzed video. Default is FALSE. A saved YouTube video has no sound when YouTube offers no stream with video and audio in one file, which is now the usual case; use speech_scores for the audio.
save_frames	Logical indicating whether to save the analyzed frames. Default is FALSE.
save_dir	The directory to save the analyzed frames. Default is "temp/".
video_name	The name of the analyzed video. Default is "temp".
model	A string specifying the vision model to use. Options include: • Built-in models: "oai-base" (default), "oai-large", "eva-8B", "jina-v2", "oai-base-fer" and "oai-large-fer" (OpenAI CLIP with the vision encoder fine-tuned on the FER2013 facial-expression dataset) • Any valid HuggingFace model ID, including checkpoints that hold only a fine-tuned CLIP vision encoder; their text encoder and processor come from the base CLIP model named in the checkpoint's config

- Custom registered models (see register_vision_model)

Use list_vision_models to see all available models. Note: Video processing is memory-intensive, so use caution with large models.

local_model_path

Optional. Path to a local directory containing a pre-downloaded HuggingFace model. If provided, the model will be loaded from this directory instead of being downloaded from HuggingFace. This is useful for offline usage or for using custom fine-tuned models.

On Linux/Mac, look in ~/.cache/huggingface/hub/ folder for downloaded models. Navigate to the snapshots folder for the relevant model and point to the directory which contains the config.json file. For example: "/home/username/.cache/huggingface/hub/mode cross-encoder–nli-distilroberta-base/snapshots/b5b020e8117e1ddc6a0c7ed0fd22c0e679edf0fa/"

On Windows, the base path is C:\Users\USERNAME\.cache\huggingface\transformers\

Warning: Using very large models from local paths may cause memory issues or crashes depending on your system's resources, especially when processing videos with many frames.

face_detector "yunet" (default), OpenCV's YuNet face detector, or "haar", the Haar cascade used before version 0.1.8. Use "haar" to reproduce scores from earlier versions.

Value

A data frame of scores, one row per frame that could be read (or one row per face; see face_selection).

Data Privacy

All processing is done locally with the downloaded model, and your video frames are never sent to any remote server or third-party.

Author(s)

Aleksandar Tomasevic <atomashevic@gmail.com>

---

view_faces *Look at detected faces in the browser, and choose faces to analyse*

Description

Starts a small web page from R that shows every image, or every sampled video frame, with its face boxes, and opens it in your web browser. Faces of the same person share a colour, and a People panel lists each person with the images, frames and tracks they appear in. With select = TRUE, you click faces, video tracks or people, then **Send to R**, and view_faces() returns the chosen faces.

Usage

```
view_faces(
  faces,
  select = FALSE,
  selected = NULL,
  live = TRUE,
  open = interactive(),
  dir = NULL,
  title = NULL,
  max_side = NULL,
  port = NULL,
  inline = .te_in_notebook()
)
```

Arguments

faces	A data frame returned by detect_faces or select_faces.
select	Logical. Let the viewer choose faces and return them. Default is FALSE.
selected	Faces to start with selected, as rows of faces (for example select_faces(faces, "largest")). Only with select = TRUE.
live	Logical. Serve the page from R (the default). FALSE only writes the folder.
open	Logical. Open the page in the web browser. Default is TRUE in an interactive session.
dir	A new or empty folder for the page. Default is a new folder in the temporary directory, which R deletes when the session ends.
title	Title shown at the top of the page.
max_side	Longest side, in pixels, of the image and frame copies in the page. Default is 1280, or 640 inside a notebook, where the copies are stored in the notebook itself. Boxes are drawn in the original pixels, whatever the size of the copies.
port	Port of the local server. Default is a free port chosen at random; set one to forward it over SSH when R runs on a remote computer.
inline	Logical. Show the page inside the notebook, below the cell, instead of in a browser tab. Default is TRUE in a Jupyter notebook (Google Colab, JupyterLab, VS Code) and FALSE elsewhere. live and open are then ignored.

Details

The page is served by R itself (with **httpuv**) on your own computer only (127.0.0.1), at an address with a random part that only you get. It makes no outside requests. While R waits for a selection, the console is busy; press Esc (Ctrl+C in a terminal) to stop waiting.

With live = FALSE, no server is started: the page is written as a folder that also works when opened from disk, for example after copying it from a remote computer to yours. Its **Export selection** button then downloads a file that read_face_selection() reads.

In a Jupyter notebook, including Google Colab, the page is shown inside the notebook, below the cell (inline = TRUE, the default there). A page served from R cannot be reached from a notebook

running on another computer, so in select mode the page gives a line of R code instead: click **Copy R code**, paste the line into a new cell and run it, and `read_face_selection()` returns the chosen faces.

Images without a face are shown too when `faces` is a complete `detect_faces` result; for a subset, only images with a face in it are shown. For a video, every sampled frame is shown, and the frames are extracted again (as in `face_scores`), unless `detect_faces` kept them with `save_frames = TRUE`.

In select mode, clicking a face in a still image selects that face; clicking a face in a video selects its whole track; clicking a person selects every face of that person.

When R runs on a remote computer over SSH, forward the port first (the message printed by `view_faces()` gives the command), or use `live = FALSE` and copy the folder.

The page lists file names and shows the images, so share it only with people who may see them. It contains no face embeddings and no scores.

Value

With `select = TRUE`: the chosen rows of `faces`, which `face_scores` scores and `face_selection` in `image_scores`, `image_scores_dir` and `video_scores` accepts, or `NULL` if you click **Cancel**. Otherwise, invisibly, the page: a list with its `url`, its folder `dir`, and `stop()`, which stops the server. With `live = FALSE`, the path of the page's `index.html`, invisibly. With `inline = TRUE`, invisibly, the path of the page as one self-contained file, `faces.html` in `dir`; in select mode the chosen faces come from the copied line of code.

See Also

`draw_faces` for the same boxes as plots in R, `select_faces` to choose faces by rule, number or person.

Examples

```
## Not run:
images <- system.file("extdata", c("two-people.jpg", "boris-2.png", "trump2.jpg"),
                      package = "transforEmotion")
faces <- detect_faces(images)
view_faces(faces)

# Click the faces to keep, then Send to R
chosen <- view_faces(faces, select = TRUE)
face_scores(chosen, classes = c("happy", "sad", "angry", "neutral"))

# Without a server: a folder to copy, and a file to read back
view_faces(faces, select = TRUE, live = FALSE, dir = "faces-page")
chosen <- read_face_selection("~/Downloads/selection-faces-20261011-101500.json", faces)

# In a notebook (Colab): the page appears below the cell. Click the faces,
# then Copy R code, and run the copied line in a new cell, for example
view_faces(faces, select = TRUE)
chosen <- read_face_selection("te-selection:eyJzY2hlbWEiOi...", faces)

## End(Not run)
```

Index

∗ **datasets**
    emotions, 26
    neo_ipip_extraversion, 44
    stop_words, 77
    tinytrolls, 79
.init_builtin_models, 4
.vision_model_registry, 5

add_vision_model, 5
annotate_faces, 7, *10, 11*, *13*, *61*
annotation_agreement, 8, *13*, *18*, *61*
annotation_progress, *8*, 9
annotation_questions, 10, *12*
annotation_task, *7*, *9, 10*, 11, *11*, *61*
as_rag_table, 13

calculate_moving_average, 14
calibrate_scores, 15, *49, 50*
check_findingemo_quality, 17
consensus_labels, *9*, *13*, 18, *61*
costring, *44*

delete_transformer, 19
detect_faces, *12*, 19, *25, 26*, *31, 32*, *35, 36*, *62*, *66*, *71*, *89*, *91, 92*
dlo_dynamics, 22
download_findingemo_data, 23, *39*
draw_faces, *22*, 25, *92*

emotions, 26, *27*
emoxicon_scores, 27, *30*
emphasize, 28
evaluate_emotions, *18*, 29, *52*

face_scores, *12*, *15*, *19*, *21, 22*, *25*, 31, *35, 36*, *62*, *66*, *89*, *92*

generate_observables, 32
generate_q, 33
get_vision_model_config, 34

image_scores, *32*, 34, *36*, *62*, *66*, *75*, *84*, *92*
image_scores_dir, 36, *62*, *92*
is_vision_model_registered, 37

list_vision_models, *35*, 37, *90*
load_findingemo_annotations, *24*, 38, *52*

map_discrete_to_vad, 40
map_to_emo8, 42
MASS_mvrnorm, 43

neo_ipip_extraversion, 44
nlp_scores, *30*, 44

parse_rag_json, 47
plot.emotion_evaluation, 48
plot.te_faces *(*draw_faces*)*, 25
plot_sim_emotions, 48
posterior_scores, *15, 16*, 49
prepare_findingemo_evaluation, *39*, 51
print.emotion_evaluation, 52
punctuate, 53
py_require, *72*
python_requirements, 54, *56*, *69*, *72*

rag, 55, *60*
rag_json_utils, 59
rag_sentemo, 59
read_annotations, *8*, *11*, *13*, *18*, 61
read_face_selection, 62, *91, 92*
register_retriever, 63
register_vision_model, *35*, 63, *90*
remove_vision_model, 65

select_faces, *12*, *19*, *22*, *25, 26*, *32*, *35*, 66, *89*, *91, 92*
sentence_similarity, 67
setup_cache, 68, *72*
setup_gpu_modules, 70
setup_miniconda, 71
setup_modules, *21*, *24*, *54*, *70*, 71

setup_popular_models, 72
show_vision_models, 73
simulate_video, 74
speech_scores, *71*, 75, *89*
stop_words, *45*, 77
summary.emotion_evaluation, 78

te_cleanup_default_venv, 78
tinytrolls, 79
transforEmotion
        *(*transforEmotion-package*)*, 3
transforEmotion-package, 3
transformer_scores, *30*, 79, *84*

vad_scores, 83
validate_rag_json, 86
validate_rag_predictions, 87
video_scores, *12*, *15*, *20*, *50*, *62*, *66*, *77*, *84*,
        88, *92*
view_faces, *62*, 90

weights, *27*