Package 'transforEmotion'

May 8, 2026

Title Sentiment Analysis for Text, Image and Video using Transformer Models

Version 0.1.7

Date 2026-01-05

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

Maintainer Aleksandar Tomašević <atomashevic@gmail.com>

Description Implements sentiment analysis using huggingface <https://huggingface.co> transformer zero-shot classification model pipelines for text and image data. The default text pipeline is Cross-Encoder's DistilRoBERTa <https://huggingface.co/cross-encoder/nli-distilroberta-base> and default image/video pipeline is Open AI's CLIP <https://huggingface.co/openai/clip-vit-base-patch32>. All other zero-shot classification model pipelines can be implemented using their model name from <https://huggingface.co/models?pipeline_tag=zero-shot-classification>.

Depends R (>= 3.5.0)

License GPL (>= 3.0)

Encoding UTF-8

Imports dplyr, googledrive, LSAfun, Matrix, methods, pbapply, remotes, reticulate, textdata, jsonlite, ggplot2, reshape2, httr

Suggests knitr, markdown, rmarkdown, rstudioapi, testthat (>= 3.0.0)

VignetteBuilder knitr

RoxygenNote 7.3.3

Config/testthat/edition 3

NeedsCompilation no

Repository CRAN

Date/Publication 2026-01-08 14:10:02 UTC

Contents

`transforEmotion-package` *transforEmotion–package*

Description

Implements sentiment and emotion analysis using huggingface transformer zero-shot classification model pipelines on text and image data. The default text pipeline is Cross-Encoder's Distil-RoBERTa and default image/video pipeline is Open AI's CLIP. All other zero-shot classification model pipelines can be implemented using their model name from https://huggingface.co/models?pipeline_tag=zero-shot-classification.

Author(s)

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

References

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

`.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
```

Format

An object of class `environment` of length 4.

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)
```

---

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.

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)
```

---

check_nvidia_gpu *Install Necessary Python Modules*

Description

Installs required Python modules for the {transforEmotion} package, using uv for fast, reproducible environments. Optionally detects GPU and can add GPU-oriented packages.

Usage

```
check_nvidia_gpu()
```

Details

This function performs the following steps:

- Detects NVIDIA GPU availability automatically
- Installs core modules including transformers, torch, tensorflow, and other dependencies
- For GPU systems, adds GPU-specific packages (and optional extras via `setup_gpu_modules()`)

The function declares Python requirements via `py_require`, which uses uv to resolve and cache an ephemeral environment on first use. No conda/Miniconda is required.

Note

For GPU support, NVIDIA drivers must be properly installed on your system. If you need vendor-specific wheels (e.g., for CUDA), configure package indexes prior to calling this function (see Notes in documentation).

Author(s)

Alexander P. Christensen <alexpaulchristensen@gmail.com>

`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()
}
```

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 requires the `findingemo-light` Python package to be installed. Use `setup_modules()` to install required dependencies before calling this function.

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()

# 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)
```

---

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.
classes	Character vector. Emotion classes to evaluate. If NULL, will be inferred from the data.
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)
```

---

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
)
```

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 method to select the face in the image. Can be "largest", "left", "right", or "none". 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. Face_selection method is irrelevant if there is only one face in the image.
`model`	A string specifying the vision model to use. Options include: • Built-in models: "oai-base" (default), "oai-large", "eva-8B", "jina-v2" • Any valid HuggingFace model ID • 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.

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
)
```

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".
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.

Value

A data.frame with columns: image_id and one column per class.

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

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.

---

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("."))
```

---

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", "Qwen3-1.7B", "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. **"Qwen3-0.6B / Qwen3-1.7B"** Qwen 3 small Instruct models via HuggingFace: Qwen/Qwen3-0.6B-Instruct, Qwen/Qwen3-1.7B-Instruct. **"Ministral-3B"** Mistral's compact 3B Instruct via HuggingFace: ministral/Ministral-3b-instruct
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 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 Gemma3-1B and Gemma3-4B. For other models, requests for structured outputs fall back to "text".
- For Gemma3-1B/4B and task = "sentiment" or "emotion", returns per-document dominant label and confidence.
- For Gemma3-1B/4B 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
rag(text = text, query = "Extract emotions", output = "json")
rag(text = text, query = "Extract emotions", output = "table")

## 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", "Qwen3-1.7B", "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"` • `"Qwen3-0.6B"` • `"Qwen3-1.7B"` • `"Ministral-3B"`
`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 Gemma3-1B/4B and `output = "table"/"csv"`, a data.frame with columns `doc_id, text, label, confidence`.
For Gemma3-1B/4B 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")

## 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) • `"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)
```

`sentence_similarity` *Sentiment Analysis Scores*

Description

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

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., `"typeform/distilbert-base-uncased-mnli"`; 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
`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_gpu_modules *Install GPU Python Modules*

Description

Installs GPU-specific Python modules using uv-managed environments.

Usage

```
setup_gpu_modules()
```

Details

This function installs additional GPU-specific modules including:

- AutoAWQ for weight quantization
- Auto-GPTQ for GPU quantization
- Optimum for transformer optimization
- llama-cpp-python (Linux only) for CPU/GPU inference

The function is typically called by `setup_modules()` when GPU installation is selected, but can also be run independently to add GPU-related packages.

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

Installs and configures required Python modules for transforEmotion, optionally enabling GPU-accelerated variants when a compatible NVIDIA GPU is detected. Uses reticulate's uv-backed ephemeral environment.

Usage

```
setup_modules()
```

Details

This function ensures required modules are available and can add additional GPU-specific packages when requested. See also `setup_gpu_modules()` for GPU add-ons.

Value

Invisibly returns NULL.

`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)
```

---

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 zero-shot classification models with a pipeline from huggingface to be used by using the specified name (e.g., `"typeform/distilbert-base-uncased-mnli"`; see Examples)

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

`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"
)

## 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 = "..."), ...)
`model`	Character. Model to use for classification. Depends on input_type: • Text: transformer model (see transformer_scores documentation) • Image: CLIP model (see image_scores documentation) • Video: CLIP model (see video_scores documentation)
`...`	Additional arguments passed to underlying classification functions (transformer_scores, image_scores, or video_scores)

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 valence, which refers to pleasant, enjoyable..."
- **Arousal**: "High arousal, which refers to intense, energetic..."
- **Dominance**: "High dominance, which refers to feeling in control..."

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 = "Gemma3-1B"
)

# 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
)
```

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", or "none". Default is "largest". Use "none" to classify the whole frame without face cropping.
`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.

`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"
- Any valid HuggingFace model ID
- 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.

Value

A result object containing the analyzed video scores.

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>

Index