

Package: fathom (via r-universe)

October 30, 2026

Type Package

Title Qualitative Coding and Measurement with Local Language Models

Version 0.1.0

Description Qualitatively code a corpus by asking a local language model a set of questions about each document and reading the answer from the model's next-token probability distribution rather than parsing its text. Extraction never fails to parse, returns a probability per label, and reports the probability mass the model placed outside the label set. Includes the auditing steps that separate usable annotation from plausible-looking output. These inspect the documents that fired on an item and draw blinded samples for human coding. Requires a running Ollama server.

License MIT + file LICENSE

Note The `parlasent` dataset is distributed under CC BY-SA 4.0 rather than MIT. See LICENSE.note.

Encoding UTF-8

SystemRequirements Ollama ($\geq 0.12.11$); ffmpeg (for `prepare_audio()` and `prepare_video()`); whisperx or whispermlx (optional, for `prepare_audio()` and `prepare_video()`)

Depends R ($\geq 4.4.0$)

Imports cli ($\geq 3.0.0$), digest, dplyr, httr2, jsonlite, magrittr, stats, stringi, tibble, tidyr, utils

Suggests ggplot2, testthat ($\geq 3.0.0$), readr, knitr, rmarkdown (≥ 2.13), quarto, withr, ragg, pander

Config/testthat/edition 3

Roxygen list(markdown = TRUE)

URL <https://alcocer-jj.github.io/fathom>,
<https://github.com/alcocer-jj/fathom>

BugReports <https://github.com/alcocer-jj/fathom/issues>

Config/roxygen2/version 8.0.0
LazyData true
VignetteBuilder knitr, quarto
Config/pak/sysreqs libicu-dev libssl-dev
Repository https://alcocer-jj.r-universe.dev
Date/Publication 2026-10-30 04:47:52 UTC
RemoteUrl https://github.com/alcocer-jj/fathom
RemoteRef HEAD
RemoteSha fd309ad473b4431342eb1c5ac17dcfc99ad819e6

Contents

as_wide	2
ati	5
coder_reference	6
coding_sample	7
inspect_cases	9
instrument	12
item_report	17
measure	20
ollama_backend	25
parlasent	27
prepare_audio	29
prepare_image	33
prepare_video	35
preview	39
read_checkpoints	40
template_items	41
threshold_sensitivity	43
validate_sample	45
Index	50

as_wide	<i>Pivot results into an analysis matrix</i>
---------	--

Description

Pivots results into one column per item, holding the probability of the target label.

Usage

```
as_wide(
  results,
  ...,
  target = NULL,
  value = c("prob_renorm", "prob", "expected_score"),
  prefix = "",
  dichotomize = FALSE,
  keep = character(0),
  drop_incomplete = TRUE
)
```

Arguments

results	A fathom_results object from measure() or read_checkpoints() .
...	Not used, and refused if anything reaches it. Every argument after it is name-only, so a misspelling lands here rather than being matched by position.
target	A string naming the label whose probability fills each cell. Defaults to the first label for a multi-item instrument, and to NULL for an exclusive instrument, where every category gets its own column.
value	<p>A string naming the column that supplies the value. Defaults to "prob_renorm", the probability conditioned on the model having answered with one of your labels at all. That is the quantity a human coder given the same categories would have produced, and it is what validate_sample() and the expected score read, so a matrix built here can be compared against either without changing quantity partway through.</p> <p>"prob" is the raw probability, measured against every token the model could have written. Use it when the claim is about whether the model committed rather than about which label it preferred.</p> <p>"expected_score" requires an instrument built with scores, and returns one column per item rather than one per label, since an expected score is one number per document and item.</p>
prefix	A string prepended to column names. Default is "".
dichotomize	A logical value indicating whether to threshold at 0.5 and return integers. Default is FALSE.
keep	A character vector of row-level columns to carry alongside. Each must hold one value per document, since this returns one row per document. A column that varies within a document is refused rather than allowed to widen the pivot. Default is <code>character(0)</code> .
drop_incomplete	A logical value indicating whether to drop documents missing any item. Default is TRUE.

Value

A tibble, one row per document, carrying id, then any keep columns, then the pivoted values. Column names take prefix if given.

What the value columns are depends on the arguments, and there are three shapes.

- Naming `target` gives one column per item, holding that label's probability. The usual shape for an instrument of independent binary items, and the one a measurement model wants.
- Leaving `target` as `NULL` gives one column per label on a single-item instrument, or one per item and label combination on an instrument of several, named `item_label`. Use it when every label matters rather than one.
- `value = "expected_score"` gives one column per item, since an expected score is already one number per document and item. Requires an instrument built with scores, and cannot be combined with `dichotomize`.

Values are probabilities, or integers when `dichotomize = TRUE`. A failed, truncated, or all-absent record becomes `NA`, and `drop_incomplete` decides whether its document leaves the frame. Either way the count is reported rather than left to be noticed.

Normalizing across items

There is no argument to normalize across items. K independent binary items do not form a distribution, as each probability comes from its own softmax over its own two-token distribution. Dividing them by their sum produces a number that looks like a proportion and is not one.

An exclusive instrument is the exception. Its K labels share one softmax at one position, so its rows do sum to one and are comparable across categories. This detects that from the results and defaults `target` to `NULL`, giving one column per category.

Exclusive results

`instrument(..., exclusive = TRUE)` produces a real distribution. Row sums are meaningful, minus whatever leaked outside the label set. On an exclusive instrument scored on discovered first tokens, that remainder is mostly the gap left by the approximation, since the probability of a first token is an upper bound on the probability of the label it opens.

See Also

`measure()` for the results it pivots and `item_report()` for a summary per item.

Examples

```
# Results from a real run ship with the package, so this needs no server.
p <- system.file("extdata", "vignette_results.rds", package = "fathom")
baked <- readRDS(p)
res <- baked$res_six

# An exclusive instrument gives one column per category.
head(as_wide(res, prefix = "p_"))
head(as_wide(res, prefix = "p_", dichotomize = TRUE))
```

`ati`*Mexican access-to-information requests*

Description

Three hundred requests filed through Mexico’s INFOMEX transparency system, drawn from a corpus Erlich et al. hand-coded with six Mexico City research assistants over ten weeks. Their codings are not shipped here. What ships is the request text and the four fields that trace one back to its source, which is what a run with no training data needs. The coded variables are in the replication materials named under @source, but the codebook was never published and the supplemental material names them without saying how they were defined, so what arrives is column names. The requests are in Spanish and run from a single word to 588, median 42.

Usage

`ati`

Format

A tibble with 300 rows and 5 columns.

- `id` is the row identifier from the source data.
- `folio` is the INFOMEX request number, which traces back to the source.
- `date` is the date of the request.
- `dependency` is the agency the request was filed with.
- `text` is the request as filed, in Spanish.

The text

`text` is what the requester typed. Commas were stripped when the source was written to CSV, which is not recoverable and shows as double spaces.

Source

Erlich, A., S. G. Dantas, B. E. Bagozzi, D. Berliner, and B. Palmer-Rubin. 2022. Multi-label Prediction for Political Text-as-Data. *Political Analysis* 30(4), 463–480. doi:10.1017/pan.2021.15

Replication data at doi:10.7910/DVN/SOVPA4, released under CC0 1.0. Sampled with seed 444. The underlying requests are published by Mexico’s federal transparency system.

Examples

`data(ati)`

coder_reference	<i>Collapse several coders into one reference sheet</i>
-----------------	---

Description

Takes the completed sheets and returns one sheet, so `validate_sample(coder_reference(ana, luis, bea), res)` composes without changing a signature. Read `$ceiling` from `validate_sample()` given the same sheets first. It says whether collapsing is honest, since a high alpha means the coders were interchangeable and a low one means the reference is a consensus that never existed.

Usage

```
coder_reference(
  ...,
  method = c("majority", "unanimous", "soft"),
  target = NULL
)
```

Arguments

...	Two or more completed coding sheets, as data frames or paths. Name them to name the coders.
method	A string naming how to collapse a row. "majority" takes the most common label, "unanimous" requires every coder to agree, "soft" keeps the disagreement as a proportion instead of resolving it. Default is "majority".
target	A string naming the label "soft" reports the proportion of. Defaults to the first of the two in sorted order, and the choice is written to the sheet so it is never implicit. Ignored by the other methods.

Value

A tibble, one row per document-item pair that any coder saw. It carries `id`, `item`, `human_label`, and `n_coders` and `n_agree` beside them.

Under "soft" the label column is replaced by `human_prop`, with `human_target` naming the label the proportion is of and `n_target` in place of `n_agree`. `validate_sample()` branches on which of the two arrived.

Collapse methods

"majority" writes the label most coders chose. Where two or more labels tie for the most it writes NA and names those rows, rather than breaking the tie. A rule that broke it would invent a coding no coder produced.

"unanimous" writes the label only where every coder who saw the row chose it, and NA otherwise. It warns, since the rows it discards are the hard ones, so what survives is easier than the draw was.

A row only one coder saw has a majority of one and is trivially unanimous. `n_coders` on the returned sheet is what separates those from a real agreement.

The soft method

"soft" writes no label. It writes `human_prop`, the share of coders who chose target, smoothed as $(n_{\text{target}} + 1) / (n_{\text{coders}} + 2)$ so two coders who both said yes give 0.75 rather than 1. That is the honest reading of two observations, and it keeps the target off the boundaries, which matters because `prob_renorm` reaches exactly zero and a log score would then be infinite. `validate_sample()` scores it with Brier for the same reason.

It needs exactly two labels. Above two the honest soft target is a vector of proportions rather than one number, and reporting one label against the rest would leave a Brier over two categories where a label sheet is scored over all of them. Refused rather than silently rescaled.

See Also

`validate_sample()`, whose `$ceiling` says whether collapsing is honest and which reads the result.

Examples

```
ana <- data.frame(
  id = c("1", "2", "3"), item = "q",
  human_label = c("YES", "NO", "YES")
)
luis <- data.frame(
  id = c("1", "2", "3"), item = "q",
  human_label = c("YES", "YES", "NO")
)
bea <- data.frame(
  id = c("1", "2", "3"), item = "q",
  human_label = c("YES", "NO", "NO")
)
coder_reference(ana = ana, luis = luis, bea = bea)
```

`coding_sample`

Draw a blinded sample for human coding

Description

Draws a sheet of documents and questions with a blank column for a coder, and the model's answer withheld, and writes it as a CSV when `file` is given. A coder shown the model's label agrees with it more often, so withholding it is what makes the resulting agreement figure interpretable.

Usage

```
coding_sample(
  results,
  docs = NULL,
  ...,
  text = NULL,
  id = NULL,
```

```

    image = NULL,
    n = 200L,
    items = NULL,
    labels_allowed = NULL,
    copy_media = TRUE,
    file = NULL,
    seed = 1234L
  )

```

Arguments

<code>results</code>	A <code>fathom_results</code> object from <code>measure()</code> or <code>read_checkpoints()</code> .
<code>docs</code>	A character vector, or the data frame passed to <code>measure()</code> . Default is <code>NULL</code> , which uses the documents a character-vector run stored on the results.
<code>...</code>	Not used, and refused if anything reaches it. Every argument after it is name-only, so a misspelling lands here rather than being matched by position.
<code>text, id</code>	Strings naming the text and identifier columns, if <code>docs</code> is a data frame. <code>text</code> is then required, and <code>id</code> defaults to the row position.
<code>image</code>	A string naming a column in <code>docs</code> holding the media for each document, so the sheet carries it as well as any text. A character column is one file per document. A list column holds several, such as the frames <code>prepare_video()</code> returns, and they share one cell joined with <code> </code> . Default is <code>NULL</code> .
<code>n</code>	An integer giving the total document-item pairs to draw, across all questions. Default is 200.
<code>items</code>	A character vector restricting the draw to specific items, or <code>NULL</code> (the default) for all of them.
<code>labels_allowed</code>	A character vector of permitted values for the <code>human_label</code> column, written into the sheet so a coder knows the answer scheme. Defaults to the instrument label names.
<code>copy_media</code>	A logical value indicating whether to copy the files <code>image</code> points at into a folder beside the CSV and rewrite the column relative to it. Acts only when both <code>file</code> and <code>image</code> are given. <code>FALSE</code> leaves the original absolute paths, which is right when the coders share the filesystem the sheet was written on and wrong otherwise. Default is <code>TRUE</code> .
<code>file</code>	A string giving the path to write the CSV to, or <code>NULL</code> (the default) to write nothing.
<code>seed</code>	An integer giving the seed for the draw, so the same call draws the same rows. The session's random stream is put back afterwards, so this seed reaches the draw and nothing after it. <code>NULL</code> draws from the session's stream instead. Default is 1234.

Value

A tibble with one row per document-item pair drawn, in random order so the order leaks nothing about the draw. The same document appears once per item it was drawn on.

The columns are for the coder, the document last so the others stay on screen beside it.

- `id` is the document identifier, matching the results.
- `item` names which question this row is about.
- `question` is the question text as the model received it, so a coder answers the same question rather than a paraphrase of it. NA when the results carry no instrument.
- `allowed` holds the permitted answers, pipe-separated, so the sheet states its own answer scheme.
- `human_label` is blank, for the coder to fill with one value from `allowed`. The model's answer is deliberately absent.
- `note` is blank, for the coder. Nothing reads it, and it exists so a coder has somewhere to record a case they were unsure about.
- `text` is the document, in full.
- `image` is the file `image` named, or NA when none was. Several files, a video window's frames, are joined in order with `|`, the way `allowed` joins the answers. Rewritten relative to the CSV when `copy_media` acted.

The draw

Every document-item pair has the same chance of being drawn, and the budget is split across questions in proportion to how many pairs each one has, so a figure computed on the sheet is already a figure about the corpus.

Item is a stratification variable rather than a filter, so the draw runs within each question and `n` stays the total across them. The draw is systematic, so the sample is exactly `n` rows on every call. An `n` at or above the number of pairs is a census.

See Also

[validate_sample\(\)](#), which reads the coded sheet, and [coder_reference\(\)](#) for collapsing several coders into one.

Examples

```
# Results from a real run ship with the package, so this needs no server.
p <- system.file("extdata", "vignette_results.rds", package = "fathom")
baked <- readRDS(p)
sheet <- coding_sample(baked$res_six, parlasent, text = "sentence",
  id = "id", n = 20, seed = 1234)
head(sheet)
```

inspect_cases

Read the documents that fired on an item

Description

Pulls the documents that endorsed an item, with their full text, for checking whether the model was right.

Usage

```
inspect_cases(
  results,
  item,
  docs = NULL,
  ...,
  text = NULL,
  id = NULL,
  which = c("endorsed", "uncertain"),
  target = NULL,
  n = 6L,
  chars = NULL,
  file = NULL,
  seed = NULL
)
```

Arguments

<code>results</code>	A fathom_results object from measure() or read_checkpoints() .
<code>item</code>	A string naming the item to inspect.
<code>docs</code>	A character vector of documents, or the data frame passed to measure() . Optional when measure() was given a character vector, since the documents are stored on the results in that case. Default is NULL.
<code>...</code>	Not used, and refused if anything reaches it. Every argument after it is name-only, so a misspelling lands here rather than being matched by position.
<code>text, id</code>	Strings naming the text and identifier columns, if docs is a data frame. text is then required, and id defaults to the row position.
<code>which</code>	A string choosing the documents, "endorsed" (the default) for those where target won or "uncertain" for the closest calls regardless of answer. "uncertain" has nothing to sort on when every document came back at the same extreme, and says so rather than returning corpus order under that label. To read the documents an item did not fire on, ask for the other label as target rather than for the complement of this one.
<code>target</code>	A string naming the label counted as endorsement. Defaults to the first.
<code>n</code>	An integer giving the number of documents to show. Default is 6.
<code>chars</code>	An integer giving the characters each document is truncated to, or NULL (the default) for full text. Warns.
<code>file</code>	A string giving a path to write markdown to instead of printing, or NULL (the default) to print.
<code>seed</code>	An integer giving a seed for the sample, or NULL (the default). It reaches this draw only, and the session's random stream is put back afterwards.

Value

Invisibly, a tibble of the selected cases, one row per document, in the order they are printed. Printing is the point of this function and the returned object is a convenience, so it carries the columns the printout drew on rather than a curated set.

- `id` is the document identifier.
- `item` is the item inspected, the same value on every row.
- `label` is `target`, the same value on every row, since the rows for other labels are dropped before selection.
- `prob_renorm` is the probability on `target` for that document, rescaled over the labels. This is what which filtered on and what the printout shows.
- `prob` is the raw probability on `target`, before rescaling.
- `prob_other` is the probability the model placed on tokens that were none of the labels, for that document.
- `text` is the document, joined from docs. Truncated if `chars` was set. Named `text` even when the source column was called something else.

Any other columns the results carry travel through untouched, including anything passed to keep in `measure()` and the remaining diagnostic columns. A run that found no qualifying cases returns a tibble with no rows.

The header line

Three numbers, in this order. How many cases are printed, how many qualified before sampling down to `n`, and how large the corpus was. The middle one is the one to read, since it says whether the printed cases are all of them or a sample. Reporting the corpus size alone would read as though every document not shown had failed the filter, which is the opposite of what it means.

Endorsement by label count

On two labels it is the 0.5 cut on `prob_renorm`, which is what `item_report()` counts as `endorse_renorm`. Above two the labels compete for one pool, a winner at 0.3 clears no cut, and endorsement is instead the label that took the most mass, which is what `item_report()` counts as `n_won`.

"uncertain" is a rank rather than a threshold in both cases, so it commits to no cut point. On two labels it sorts by distance from 0.5. Above two it sorts by margin, the gap between the winning label and the runner-up, so the first rows are the closest calls.

Excerpts versus full text

Full text is the default. The passage that triggered an item is often outside a keyword-centered window, so an excerpt can read as a false negative when the item was working. Setting `chars` truncates and warns.

See Also

`item_report()` for the summary these documents sit behind. `coding_sample()` for drawing cases to hand to a coder rather than reading a few.

Examples

```
# Results from a real run ship with the package, so this needs no server.
p <- system.file("extdata", "vignette_results.rds", package = "fathom")
baked <- readRDS(p)
res <- baked$res_six

# measure() was given a data frame, so the documents are passed again.
inspect_cases(res, "category", parlasent, text = "sentence", id = "id",
  target = "1", n = 2, seed = 1)
```

instrument

Construct a validated instrument of items

Description

Bundles the questions to be answered, the labels the model may answer with, and the instruction that elicits them.

Usage

```
instrument(
  items,
  labels = c("YES", "NO"),
  ...,
  definitions = NULL,
  scores = NULL,
  exclusive = FALSE,
  exact = FALSE,
  probe_texts = NULL,
  cue = NULL,
  lang = c("en", "es"),
  medium = NULL,
  text_label = NULL,
  answer_label = NULL,
  name = NULL,
  backend = NULL
)
```

Arguments

items	<p>A named list or named character vector of question strings. An unnamed single string is accepted and named Q1. When <code>exclusive = TRUE</code>, this may instead be a stem name ("primary" or "best") or a custom question, with the categories supplied through labels.</p> <p>A question is a free string and it is yours. It may run to a paragraph, reproduce a codebook definition verbatim, carry an exclusion clause, or include worked examples. Nothing about the scoring changes, since the question sits in the prompt and the answer is read from one token.</p>
-------	--

labels	<p>A character vector of answer words, at least two. Each element is its own label. Matching is case folded and reads by prefix, so a token counts when it opens a label and a tokenizer splitting PRESUPUESTO into P still scores. Text explaining what a label means goes in definitions, not here.</p> <p>One word each, and whitespace is refused. NOT_SURE and NOT-SURE build, since joining leaves a single answer word that a tokenizer may split into a prefix of itself. NOT SURE does not, since one token cannot span the space and the label would be matched by NOT alone. Default is c("YES", "NO").</p>
...	Not used, and refused if anything reaches it. Every argument after it is name-only, so a misspelling lands here rather than being matched by position.
definitions	A named character vector explaining what each label means, or NULL (the default). An exclusive instrument lists these in the question, so the model reads the explanation and types only the name. Any instrument can carry them, where they appear in print() and improve the generated probe document.
scores	A named numeric vector giving each label a value, or NULL (the default). Names must match the label names exactly. Supplying them asserts that the spacing between the numbers is meaningful, which is what makes an average of them defined and is a stronger claim than that the labels are ordered. Nothing checks it, since unequal spacing is legitimate on a real scale.
exclusive	A logical value indicating whether to build an exclusive-choice instrument. One question lists all K categories, the model picks one, and all K share a softmax at position 1. The result is a distribution, unlike an instrument of independent binary items. The category names become answer tokens here, so they must differ at their first character. Default is FALSE.
exact	<p>A logical value indicating whether to score by exact match against discovered tokens rather than by prefix. Requires backend, since those tokens come from probing.</p> <p>This is the one case where a label carries more than one string. Prefix matching needs the labels to differ at their first character, because a split label often emits only that character and E cannot be attributed between <i>economia</i> and <i>educacion</i>. Probing sends one throwaway document per label, reads back what the model actually writes, ECON and EDUC, and records it. Scoring then consults that record instead of guessing from a prefix, and a bare E counts for nothing and shows up in prob_other rather than being assigned to whichever label sorted first.</p> <p>Nobody types those entries. Use exact when the labels cannot be made to differ at their first character, and rename them if they can. Default is FALSE.</p>
probe_texts	A named character vector of example documents, one per label, used in place of the generated probe document, or NULL (the default). Supply these when the labels describe a property of the text rather than a topic, since the generated document is <i>about</i> the label rather than one that exhibits it.
cue	A string giving the instruction appended after each question. Generated from labels and lang if NULL. Supply your own for a language without a preset.
lang	A string giving the language preset for the prompt scaffolding, "en" (the default) or "es". Sets cue, text_label, and answer_label together, and any of the three can still be overridden individually. For any other language, pass all three

explicitly rather than expecting a preset. See the section below for what each preset contains.

This is the scaffolding, not the question. A question written in Spanish under the default "en" preset produces a Spanish item wrapped in an English cue, which is a real prompt and rarely the intended one.

medium	A string naming the stem set a preset name is looked up in, one of "text", "audio", "image" or "video". Read only when items names a preset, since a literal question carries its own wording. There is no default, because the sets are not translations of one another. A picture mentions nothing, so image has show and depict where text has topic and mention, and video has both plus discuss, since a row holds frames and a transcript at once and the verb is what says which one the question is about.
text_label, answer_label	Strings giving the prompt scaffolding. Default to the lang preset.
name	A string giving a label for the instrument, carried into provenance, or NULL (the default).
backend	An <code>ollama_backend()</code> object, or NULL (the default). When supplied, each label is probed against a production-shaped prompt and its observed first token is recorded. Probing checks that the labels are distinguishable at position 1 and prints what each one emits. It changes scoring only under <code>exact = TRUE</code> , since a discovered token already opens its own label and so is matched by prefix regardless.

Value

An object of class `fathom_instrument`, a list. Its print method shows the parts usually wanted, and `preview()` shows what the model will actually receive.

The instrument:

- `items` is the named list of questions, one per item, in the order declared. That order is the order `item_report()` and the results use, so an instrument reads as the questionnaire it was written as.
- `labels` is the named list of answer labels.
- `normalized` holds what each label is actually matched against. Under prefix matching this is the label itself, and under `exact = TRUE` it is the variants probing discovered. This is the field to read to find out why a token did or did not count.
- `definitions` is the category text shown to the model beneath the question, or NULL. Where long category wording goes, since it has no first-character constraint on it.
- `scores` is the numeric value per label, or NULL. Sits outside the prompt hash on purpose, so rescaling re-extracts nothing.

How it gets asked:

- `cue` is the assembled instruction naming the permitted answers.
- `text_label` and `answer_label` are the words introducing the document and the answer slot.
- `lang` names which scaffolding preset supplied the three above.

Shape and mode:

- `exclusive` says whether the labels share one distribution. `FALSE` gives independent binary items, so a document may match several.
- `exact_match` says whether scoring compares against probed tokens rather than by prefix. The escape for a label set that has to collide.
- `k` and `n_items` are the counts of labels and of items.
- `name` is the short label-set name used in printing.

Implementation:

`probe_texts` and `firsts` are implementation. They hold what probing was run against and the token each label emitted when probed, and neither is a stable interface.

Label matching

Scoring reads one token and counts it as a label when it opens that label, so `BU` counts as `BUDGET`. Nothing has to be discovered first, because a model writing `BUDGET` must emit a prefix of it, and that follows from every tokenizer in use detokenizing by concatenation rather than from any one model's vocabulary.

The price is that labels must differ at their first character. Under prefix matching this is checked at construction from the strings alone, with no backend. A split label frequently emits only its first character, and two labels sharing one cannot then be told apart. With distinct first characters no two labels share any prefix, so every match resolves to one label.

A label carrying whitespace is refused at construction, because scoring reads one token. One token cannot span a space, so a two-word label is matched by its opening word and the rest is never read, which scores the opening word rather than scoring zero. Join the words if the answer really is compound, since `NOT_SURE` and `NOT-SURE` are both accepted.

`exact = TRUE` is the way out when they have to collide. Each label is probed once, the emitted token is recorded, and scoring consults that record instead of reading a prefix. Passing a backend alone runs the probe without changing how scoring works, which is worth doing once on a new model to see the token map.

Labels and definitions

Labels and definitions are different things. `labels` names the answer words and the surface forms they may appear in. `definitions` holds the text a reader or a model needs to know what a label means. Keeping them apart matters because scoring matches the token at position 1 against labels, so a definition sitting in `labels` registers a token the model never emits and that label scores zero on every document with no error raised.

An exclusive instrument lists the definitions under its question, which is what lets a category be a long phrase while the answer stays one word. Other instruments store them without putting them in the prompt, since their question is yours to write.

Ordinal labels

`scores` attaches a numeric value to each label. The values are metadata. They never appear in the prompt and the model never types them, so the labels stay semantic words.

Scoring then returns an `expected_score` column, the sum of each label's value weighted by its renormalized probability. The labels of any one call share a softmax, since they are read at a single position from a single distribution, so this works on any instrument rather than only exclusive ones.

Nothing checks that the labels are ordered. `scores` asserts an order the package cannot verify, and reversing the values reverses every score without complaint.

Language presets

Three strings the prompt is built from, plus the word used to join the labels inside the cue.

part	en	es
<code>text_label</code>	Text	Texto
<code>answer_label</code>	Answer	Respuesta
<code>label joiner</code>	or	o
<code>cue</code>	Answer with exactly one word: ...	Responde con exactamente una palabra: ...

So a two-label instrument under `lang = "es"` closes with *Responde con exactamente una palabra: sí o no.* and labels its parts *Texto* and *Respuesta*.

Two stems ship for `exclusive = TRUE`, in both languages, and unlike the item stems in `template_items()` neither takes a `{label}`, since the categories arrive through definitions instead.

stem	en	es
"primary"	What is the PRIMARY subject of this text?	¿Cuál es el tema PRINCIPAL de este texto?
"best"	Which category best describes this text?	¿Qué categoría describe mejor este texto?

They are not interchangeable. "primary" asks what the text is about, which assumes the categories are subjects. "best" asks which category describes it, which assumes only that they are a scheme worth sorting into. Any other string is used as the question verbatim, in whatever language it is written.

Case folding and accented labels

A generated token is folded to upper case before it is compared to a label, so *sí*, *Sí*, and *Śí* all count as one answer.

`preview()` and `check_backend()` fold the same way, which is what makes the preview table a reliable picture of what scoring will do.

The first-character rule is therefore about characters and not bytes. Two labels differing only in an accent still collide once folded, and this function refuses them. Where a coding scheme forces that collision, `exact = TRUE` is the escape.

See Also

`template_items()` for items built from one stem, `preview()` for the prompt it assembles, and `measure()` for running it.

Examples

```
b <- instrument(
  items = list(
    econ = "Does the text discuss economic conditions?",
    immig = "Does the text discuss immigration?"
  ),
  labels = c("YES", "NO")
)
b
```

item_report	<i>Summarize how each item behaved</i>
-------------	--

Description

Produces a diagnostic data frame where each row is one item, or each row is one item and label, summarizing how each item behaved across the corpus. Each column answers a different question about whether an item is usable.

It is purely descriptive, meaning no column here reports whether an item is correct. A confident, low-leakage item can still be measuring something other than the construct. [inspect_cases\(\)](#) is what answers that, by showing the documents.

Usage

```
item_report(results, target = NULL, order = c("declared", "size"))
```

Arguments

results	A fathom_results object from measure() or read_checkpoints() .
target	A string naming the label counted as endorsement. Defaults to the first (e.g., "yes" in an instrument where "yes" was written first before "no"). Used by the endorsement shape only.
order	A string setting how labels are ordered within a question. "declared", the default, is the order the instrument wrote them, which for an ordered scale is the scale and which every other part of the package already uses. "size" puts the most frequent winner first, which is the reading for unordered categories. No effect on the endorsement shape, which has no label rows.

Value

A tibble in one of three shapes, chosen as described below. Items come back in the order the instrument declares them rather than alphabetically, and an item present in the results but absent from the instrument sorts after the declared ones.

Endorsement shape:

Exactly two labels and no scores. One row per item.

- `item` is the question name.
- `n` counts the documents scored on this item. Failed calls are excluded, so this can sit below the corpus size.
- `endorse_renorm` is the share of those documents where `target` reached 0.5 on `prob_renorm`, the probability after the labels have been rescaled to sum to 1. Reaching it means the label beat the other one.
- `n_endorse_renorm` is the count behind `endorse_renorm`. Its denominator is `n` minus the documents where no label appeared, which have no renormalized probability. That count is `pct_both_absent` times `n`, and is 0 whenever `pct_both_absent` is.
- `endorse` is the same cut on `prob`, the probability against everything the model could have written. Reaching it means the model committed to the label.
- `n_endorse` is the count behind `endorse`. Its denominator is `n`.
- `mean_prob` is the mean `prob` on `target`, so it belongs to `endorse` rather than to `endorse_renorm`. It separates from `endorse` only when the probabilities cluster near the 0.5 cut. When they all sit near 0 or 1 the two agree almost exactly, so the gap between them says how far the endorsements were from being coin flips.
- `pct_extreme_renorm` is the share of renormalized probabilities at or beyond 0.01 and 0.99, so it asks how the on-label mass was shaped rather than how much of it there was. This is the substantive one, for the same reason `endorse_renorm` is. It is not unconditionally the safer one. A document that answered almost entirely outside the label set can leave this denominator, read as graded, or read as certain, depending on how its residue fell.
- `pct_extreme` is the share of raw probabilities at or beyond 0.01 and 0.99. Reads `prob`, so it cannot separate a decisive answer from one that never entered the label set, since both put every label at the floor or the ceiling. Read it against `pct_extreme_renorm` and `max_prob_other` rather than alone. Near 1 means almost every answer came back at one end or the other, so these are labels in numeric clothing. Reporting them as degrees of confidence would claim more than they carry.
- `pct_both_absent` is the share of documents where no label appeared at position 1. That is a template or prompt failure rather than a low score. A broken chat template registers in `n_off_label` rather than here, because the labels stay in the distribution at trace mass underneath whatever the template wrote first, which leaves `pct_both_absent` at 0 while every answer is unusable.
- `max_prob_other` is the most probability the model ever placed on tokens that were none of your labels, anywhere in the corpus. Leaked mass, in the term used elsewhere in these docs.
- `min_top1` is the least confident single answer in the corpus. For each document, take the probability of whichever token the model was most likely to write, label or not. This is the smallest of those. High values do not mean the item is correct.
- `n_off_label` counts the documents where the emitted token matched no label. It is judged by the same rule scoring uses, so a token that opens a label counts as that label and a split label is not reported as off-label.
- `n_failed` counts the documents whose call failed. Divided by the label count, so it counts documents rather than rows.
- `other_vs_median` is `max_prob_other` divided by the middle value of that column across the instrument, so 2 means this item placed twice as much probability outside the labels as the typical item did. Present only for an instrument of three or more items, since a middle value across one or two carries nothing. This is the one column the item count decides. Absolute

leaked mass varies by model, by language, and by how close to 0 and 1 the instrument's probabilities sit, so values are not comparable across instruments.

- `mean_expected` is the average expected score across documents, which on two labels is the mean of `prob_renorm` on target rescaled onto the scores. Present only when the instrument carries scores. Note that `mean_prob` is the raw probability, so the two differ by whatever mass fell outside the labels.

Choice shape:

More than two labels and no scores. One row per item and label.

- `item` and `label` are the question and category.
- `n_won` counts the documents where this label took the largest probability. A document whose distribution held none of the labels wins nothing.
- `share_won` is `n_won` over the documents scored on that item, so the shares of an item fall short of 1 by the share that answered outside the labels.

Scored shape:

Whenever the instrument carries scores and there are more than two labels. One row per item and label, the same winner distribution plus the scale.

- `item` and `label` are the question and category.
- `score` is the numeric value the instrument assigns that label.
- `n_won` and `share_won` are as in the choice shape.
- `mean_expected` is the average expected score across documents, per question. For a single document the expected score multiplies each label's value by its probability and adds them up, after those probabilities have been rescaled to sum to 1 across the labels. Because it uses every label rather than only the winner, a document the model split between two neighboring levels lands between them.
- `sd_expected` is the standard deviation of the same, across documents.

Two endorsement rates

`endorse` and `endorse_renorm` count the same documents at the same cut of 0.5, and differ only in which probability they cut. Do not read the gap between them as measurement error or as disagreement between two estimates of one quantity. It is two quantities.

Neither cut is configurable. 0.5 on `prob_renorm` is the `argmax` on a two-label instrument, and `threshold_sensitivity()` exists to show how little the cut point matters on `prob`. Both are reported so the choice does not have to be made in advance.

Output shapes

Three shapes come back, and which one is decided by the **labels** rather than by the items. The number of items changes how many rows come back, not which columns. The test runs in this order.

1. Exactly two labels. **Endorsement shape.**
2. Otherwise, the instrument carries scores. **Scored shape.**
3. Otherwise, more than two labels. **Choice shape.**

The label count is tested first because on two labels the scored shape reports one number, the average expected score, and drops a dozen diagnostics to do it. There is no reason to choose between them. A scored instrument of two labels reports as endorsement and gains `mean_expected` as a column, so the number arrives without the diagnostics leaving.

`exclusive = TRUE` is not itself the test. An exclusive instrument lands in the choice or scored shape because it usually has more than two labels or a scale, and an exclusive instrument of two labels reports as endorsement like any other. Equally, K separate questions sharing one scale reach the scored shape without being exclusive at all.

Nothing here decides whether your labels are ordered, and nothing can. What scores says is that an average of those numbers is defined, which is why the shape is named for the argument rather than for a measurement level. `validate_sample()` is where ordering is declared, by its level.

The shapes differ in row granularity as well as in columns. Endorsement gives one row per item. Choice and scored give one row per item **and label**, because a shared distribution has no single number per question to report. Choice and scored report a winner distribution, and leaked mass reaches them as a message rather than as a column.

`target` applies to the endorsement shape only, and is ignored by the other two. There is no per-label endorsement rate for a forced choice, since the labels all compete for one pool of probability, and a rate for whichever label sorted first is not a meaningful quantity.

See Also

`inspect_cases()` for the documents behind an item, and `threshold_sensitivity()` for how the rates move with the cut point.

Examples

```
# Results from a real run ship with the package, so this needs no server.
p <- system.file("extdata", "vignette_results.rds", package = "fathom")
baked <- readRDS(p)
item_report(baked$res_indep)
```

measure

Measure documents against an instrument

Description

Runs every item in the instrument against every document and returns the probabilities in `$probs`, one row per document, item and label, beside `$calls`, one row per document and item.

Usage

```
measure(x, ...)

## S3 method for class 'character'
measure(x, backend, instrument, ...)
```

```
## S3 method for class 'data.frame'
measure(
  x,
  backend,
  instrument,
  ...,
  text = NULL,
  id = NULL,
  image = NULL,
  keep = character(0),
  num_ctx = "auto",
  items = NULL,
  checkpoint = NULL,
  verbosity = c("normal", "quiet", "verbose")
)
```

Arguments

x	A character vector of documents, or a data frame.
...	Passed to methods by the generic. On a method it is not used, and is refused if anything reaches it, since every argument after it is name-only and a misspelling would otherwise be swallowed.
backend	An ollama_backend() object.
instrument	An instrument() .
text	A string naming the column holding document text. Optional when image names one, so a picture with no caption is a corpus. Default is NULL.
id	A string naming an identifier column. Defaults to the integer row index.
image	A string naming a column holding media sent alongside the prompt rather than inside it. A character column is one picture per document. A list column is several, one element per document holding its paths in order, which is how a video window arrives. Nothing is shared between calls, so a run costs documents times items at close to full price, and a window of K frames costs K encodes on every call. prepare_image() builds a corpus in the character shape. Default is NULL.
keep	A character vector of additional columns to carry into the results. Names must be unique, and a column whose name the results already use is refused, since both copies would come back suffixed and the name would then read as NULL. Rename it in x first.
num_ctx	An integer giving the context window ceiling, or "auto". Left unset, Ollama allocates the model's full window and the cache that goes with it whether the prompts need it or not, which on a local machine is memory taken from elsewhere. Setting the ceiling to what the corpus needs returns it. Setting it too low truncates silently, since llama.cpp keeps the tail and the answer looks ordinary, and the size needed is not knowable by inspection because documents and questions both vary and the answer needs room after both. So "auto" (the default) measures it. The longest document by word count is sent once with no ceiling and its token count read back, then rounded up with

	headroom. An integer is used as given, for a corpus already known. NULL or NA sends no ceiling and lets Ollama choose, in which case the truncated column is NA for every row. A resumed run uses whatever its checkpoints were written under, overriding all of these.
items	A character vector of item names to restrict the run to, or NULL (the default) for every item.
checkpoint	A string giving the directory for per-document .rds files, or NULL (the default) to hold everything in memory.
verbosity	A string setting how much to report. Each level is the one before it plus more. <ul style="list-style-type: none"> • "quiet" reports progress and the run summary only. • "normal" is the default. It adds the header, meaning a title over one line each for the model, the context ceiling, the first token the backend check read, and the size of the job. • "verbose" adds one line per item, with its winning label, probability, leaked mass, and wall time.

Every check runs at every level. Only the reporting changes. "quiet" is for loops over several instruments against one corpus, where the caller states the configuration once rather than once per call. For no output at all, wrap the call in `suppressMessages()`. Progress is a bar redrawn in place in an interactive session, and a line every ten documents anywhere else, such as a knitted document or a log.

Value

A `fathom_results` object. A list of two frames, with the run's provenance on its attributes. `$probs` holds one row per document, item and label. `$calls` holds one row per document and item, carrying everything that describes the answer rather than any one label, so it is stored once instead of repeating down the label rows.

Keys, in both frames, and what a join is on:

- `id` is the document identifier, from the `id` column or from the names of a character vector.
- `item` names which instrument question produced the row.

`$probs`, differing down the labels of one call:

- `label` is the label this row scores, in the order the instrument declares them.
- `prob_renorm` is prob rescaled so the labels sum to 1. The measure. NA when no label appeared at all, since there is nothing to condition on.
- `prob` is the raw probability, against every token the model could have written. Does not sum to 1 down the labels.
- `present` says whether any token counting as this label appeared in the returned distribution. A prob of zero means the label was measured at zero when this is TRUE, and fell below `top_logprobs` when it is FALSE.
- `call_ok` is FALSE when the call failed. One fact per call, kept here rather than only in `$calls` because it guards almost every filter over these rows.

\$calls, one row per document and item:

Anything named in keep rides here too, since a carried column describes the document rather than any one label.

- `error` is the message when the call failed, otherwise NA. Non-NA exactly when `call_ok` is FALSE.
- `truncated` is TRUE when the prompt came within 64 tokens of `num_ctx`, so the document may have lost its opening.
- `top1_token` is the token the model actually generated, whether or not it counts as a label.
- `top1_prob` is the probability of that token. When it is not a label, this is the only record of how much mass the winning non-label held.
- `prob_other` is the mass on tokens counting as no label. The diagnostic that says whether the question landed.
- `all_absent` is TRUE when no label appeared anywhere in the returned distribution.
- `margin` is the gap between the top two `prob_renorm` values. What `inspect_cases()` sorts on to find the closest calls above two labels.
- `expected_score` is `prob_renorm` weighted by the instrument's scores. NA for an instrument built without them.
- `prompt_hash` is the hash of the assembled prompt. What checkpoint invalidation matches on, and not for reading.

On the attributes, constant for the run:

`instrument` and `backend` as given, `model`, `num_ctx` as resolved, `exclusive`, and `run_at`.

Prompt ordering

The document goes before the question, and items for a document run contiguously, so Ollama's prefix cache prefills the document once per document rather than once per item.

Checkpointing

Checkpointing is off by default. Supply `checkpoint` to enable it. One `.rds` per document holds the full record plus provenance, and a document whose items are all current is skipped without a model call. There is no separate resume function. Calling this twice against the same directory does only what is outstanding.

Invalidation is per item, keyed on a hash of the question, cue, labels, scaffolding, and model. Re-wording one item re-extracts that item across the corpus at question-only cost, since the first call still warms the document cache. This is not proportionally cheap. A partial rerun still pays the full document prefill for every row.

scores are the exception, and they are outside the hash on purpose. Rescaling an ordinal instrument changes no token the model sees, so re-extracting would be waste. The expected score is instead recomputed from the stored probabilities against whatever instrument is in hand, on this path and in `read_checkpoints()`, so a rescaled resume returns the new arithmetic without spending a call.

There is no schema tag. A change to scoring code that leaves prompts unchanged will not invalidate anything. Delete the directory by hand.

Cost of `num_ctx = "auto"`

One extra call per `measure()` invocation, not per document. Running several instruments over the same corpus measures once each, because a different instrument means a different prompt and therefore a different token count. Passing an integer once the corpus is known skips it.

Retrying failures

A failed item is stored with `call_ok = FALSE` and is therefore stale on the next pass, so re-running retries exactly those. There is no separate retry sweep, which would run against a cold cache.

Character input

A character vector stores the documents on the results in a `docs` attribute, so the audit functions find them without a `docs` argument. Ids are the vector's names when it has them, and its positions otherwise.

A data frame does not, and `measure()` never writes the document into a column. It reads the one you name and keeps a hash of the assembled prompt. Carrying the text instead would put a copy of the corpus in `$calls`, once per item, and the results are the object that gets saved.

Naming that column in `keep` carries it deliberately, and then it is a kept column like any other rather than something the schema promises.

See Also

`ollama_backend()` and `instrument()` for its two inputs, `preview()` for the prompt it sends, `read_checkpoints()` for a run cut short, and `item_report()` and `inspect_cases()` for reading what comes back.

Examples

```
## Not run:
# Needs a running Ollama server, so it is not run here.
oll <- ollama_backend("gemma4:31b")
b <- instrument(
  items = "Where does this sentence fall on the six-point sentiment scale?",
  labels = c("1", "2", "3", "4", "5", "6"),
  exclusive = TRUE
)
res <- measure(parlasent, instrument = b, backend = oll, text = "sentence",
  id = "id")
res$probs

## End(Not run)
```

ollama_backend	<i>Configure an Ollama backend</i>
----------------	------------------------------------

Description

Bundles connection and sampling settings into one object, passed down whole so parameters cannot drift between call sites. Drift can create context size differences between calls, defeating the prefix cache and forcing Ollama to reload the model.

Scoring reads log probabilities, which need an Ollama server at version 0.12.11 or later. [measure\(\)](#) checks for them before a run and stops when none come back.

Usage

```
ollama_backend(  
  model = NULL,  
  ...,  
  keep_alive = "24h",  
  timeout = 600,  
  top_logprobs = 20L,  
  host = NULL,  
  modalities = c("text", "image"),  
  validate = TRUE  
)
```

Arguments

model	A string giving the model name as Ollama knows it, e.g. "gemma3:27b". The full tag is required. Shorthand is refused. Omit it entirely to list what the server has installed.
...	Not used, and refused if anything reaches it. Every argument after it is name-only, so a misspelling lands here rather than being matched by position.
keep_alive	A string setting how long Ollama holds the model after a call. The default, "24h", is long because a corpus run over one model should not pay a reload between documents. A loop over several models on one machine wants a short value instead, since each one called stays resident for the whole window and the cost arrives as memory pressure rather than as an error.
timeout	A number giving the request timeout in seconds for generation calls. A cold prefill on a very long document can run minutes. Discovery and validation use their own shorter timeout. Default is 600.
top_logprobs	An integer giving the number of alternatives to return at each position. Ollama caps this at 20. Lower it only if every label reliably appears in the top-k. Default is 20.
host	A string giving the server URL. Defaults to OLLAMA_HOST or localhost. A host given with no scheme and no port takes port 11434, the way Ollama's own client reads OLLAMA_HOST.

modalities	A character vector of what the server accepts, which is text, image or both. A fact about the server rather than the model. <code>measure()</code> refuses a column naming something absent here, locally and before any call, because Ollama returns HTTP 200 for a field it does not recognize. Audio and video are not among them. <code>prepare_audio()</code> turns a recording into a transcript, and <code>prepare_video()</code> turns a video into frames plus a transcript when it has sound, which reach the model as text and image. Default is <code>c("text", "image")</code> .
validate	A logical value indicating whether to contact the server and confirm model is installed. FALSE constructs blind with no network call, which is what examples, documentation builds, and constructor tests use. Default is TRUE.

Value

An object of class `fathom_backend`, a list of eight settings. It holds no connection, so it is cheap to build and safe to keep.

- `model` is the tag calls are sent to. Recorded on the results as the `model` attribute, so a corpus says which model produced it.
- `host` is the server address, normalized to include a scheme.
- `modalities` holds what this server accepts. `measure()` refuses a column naming something absent here, before any call, because Ollama answers HTTP 200 to a field it does not recognize and an undeclared column would otherwise be discarded in silence.
- `top_logprobs` says how many alternatives are requested at the answer position. Ollama caps this at 20. A label whose token falls outside the returned set is scored as absent, so `prob` is a lower bound and `prob_other` absorbs the difference.
- `think` is always FALSE, and not an argument. Reasoning is disabled on every call, because scoring reads generated position 1 and a model that reasons first puts something else there.
- `keep_alive` says how long the server holds the model in memory between calls. Long enough that a corpus run does not pay reload cost.
- `timeout` is the number of seconds before a single call is abandoned. A call that times out returns a failed row rather than ending the run.
- `validated` says whether the constructor confirmed the model is installed. FALSE when built with `validate = FALSE`, which skips the check rather than failing it.

Called with no model name it prints what is installed and stops, which doubles as a way to confirm the server is running.

Omitting `model`

The call contacts the server, prints the models installed there, and errors. It does not return NULL or a partial object, so `instrument()` and `measure()` never receive a backend with no model attached.

Temperature and seed

Neither is exposed. Temperature is fixed at 0, where the returned probabilities are the model's unmodified posterior. Scoring reads the distribution at generated position 1 and discards the token that would have been drawn from it, so temperature rescales the logits being measured without changing what is read.

At temperature 0 there is no sampling, and identical calls return byte-identical token sequences. A seed has nothing to control.

See Also

`measure()`, which takes the backend, and `instrument()`, which can probe labels against it.

Examples

```
ollama_backend("gemma3:27b", validate = FALSE)
```

parlasent

UK parliamentary sentences, coded for sentiment on a six-level scale

Description

A hundred sentences from ParlaSent 1.0 (Mochtak et al. 2023), the English training file, drawn from UK parliamentary proceedings and annotated by two coders with a reconciliation pass. It is the package's exclusive-choice corpus and the only one whose labels collide.

Usage

```
parlasent
```

Format

A tibble with 100 rows and 19 columns.

- `id` is the document identifier, `uk_001` to `uk_100`.
- `sentence` is the document, 11 to 32 words, median 22.
- `label` is the three-level reconciled sentiment. Factor, Negative, Neutral, Positive.
- `reconciliation` is the six-level reconciled code. Factor, ordered from Negative through Positive.
- `annotator1` and `annotator2` are the two independent six-level codings.
- `name` is the member speaking. 93 distinct across the hundred.
- `party` is the party affiliation. Eight parties.
- `gender` is the recorded gender of the member.
- `ruling` says whether the member sat with the government or the opposition.
- `date` is the date of the sitting, 2015-01-08 to 2021-02-23.
- `term` is the parliamentary term.
- `document_id` and `sentence_id` are ParlaSent internal identifiers.
- `split` is the authors' own train, dev, and test partition. This sample crosses all three, since nothing here is trained.
- `stratum_N` is the size of the six-level stratum in the source file.
- `stratum_n` is the number drawn from it.
- `weight` is `stratum_N / stratum_n`, for weighting back at the six-level stratum.
- `weight3` is the same pooled to the three-level class.

Details

Normalized for scoring, Negative and Neutral both begin with N. Prefix matching cannot tell them apart, since a split label often emits only its first character, so `instrument()` refuses to build this label set that way and it can only be scored with `exact = TRUE` and a live backend. The collision is a property of a published coding scheme rather than something constructed to demonstrate one.

The annotators worked six levels and the three-level label is their collapse. Both mixed levels fold to their own polarity, so Neutral means no sentiment content and nothing else. Ambiguity lives in M_Positive and M_Negative, which are inside the positive and negative classes rather than beside them.

level	meaning
Positive	entirely or predominantly positive
M_Positive	ambiguous or mixed, leaning positive
P_Neutral	no sentiment content, leaning positive
N_Neutral	no sentiment content, leaning negative
M_Negative	ambiguous or mixed, leaning negative
Negative	entirely or predominantly negative

Sampling

Stratified on the six levels rather than on the three, seed 444. Seventeen each of Positive, Negative, P_Neutral, and N_Neutral, and sixteen each of the two mixed levels, giving 33 negative, 34 neutral, and 33 positive.

Sampling on the collapse would have let the draw decide the mix inside each class, and nothing would record which mix arrived. The neutral levels matter most. P_Neutral outnumbered N_Neutral 413 to 267 in the source, so a proportional draw would give a neutral class that leans positive, and a model answering that lean would look biased when the sample was.

The mixed levels are kept rather than dropped. They are the only sentences that give margin and prob_other anything to separate. A corpus of unambiguous cases returns saturated on every row and every uncertainty diagnostic has nothing to show.

Class balance

The source file is 48.8 percent negative, 26.2 percent neutral, and 25.0 percent positive, against 33, 34, and 33 here. Recall, specificity, and AUC are computed within a class and carry over. Accuracy, precision, F1, and kappa move with the class mix and do not. `weight3` reweights to the source proportions, and `weight` does the same at the six-level stratum, though at a higher cost in precision. The effective sample size is about 90 using `weight3` and about 69 using `weight`.

The sample is also deliberately harder than the file it came from. Mixed levels are 32 percent here against 14.6 percent there.

The human ceiling

`annotator1` and `annotator2` are the two independent codings before reconciliation, so the corpus carries its own benchmark. On these hundred sentences the two coders agree with each other on 0.720, which is 0.765 on the four unambiguous levels, 0.625 on the two mixed ones, and 0.570 on the raw six. Across the full 2,600-sentence file they agree on 0.783, and on the raw six levels 0.660.

A model is worth reading against 0.720 rather than against 1. The gap between the pure and mixed figures is what the uncertainty diagnostics should track.

Source

Mochtak, M., Rupnik, P., Meden, K., and Ljubescic, N. (2023). The multilingual sentiment dataset of parliamentary debates ParlaSent 1.0. Slovenian language resource repository CLARIN.SI. <http://hdl.handle.net/11356/1868>

Distributed under CC BY-SA 4.0, unlike the rest of this package. See `LICENSE.note`. Drawn from the English training file with seed 444.

References

Kuzman Pungerssek, T., Ljubescic, N., Erjavec, T., et al. (2025). Linguistically annotated multilingual comparable corpora of parliamentary debates ParlaMint-en.ana 5.0. CLARIN.SI. <http://hdl.handle.net/11356/2006>

Mochtak, M., Rupnik, P., and Ljubescic, N. (2024). The ParlaSent multilingual training dataset for sentiment identification in parliamentary proceedings. *Proceedings of LREC-COLING 2024*, 16024-16036.

Examples

```
data(parlasent)
table(parlasent$reconciliation, parlasent$label)

# The two coders before reconciliation, on the raw six levels.
mean(parlasent$annotator1 == parlasent$annotator2)
```

```
prepare_audio
```

Turn recordings into a corpus of annotated transcripts

Description

Transcribes a recording and writes its pauses and speakers back into the text, so `measure(text = )` reads a transcript that carries them. A plain transcript records what was said and discards how. Pauses, hesitation and who was speaking leave with it, and for a construct that lives in delivery rather than in wording that is the absence of the thing being measured.

Usage

```
prepare_audio(
  x,
  ...,
  diarize_names = FALSE,
  annotate_timing = FALSE,
  tool = c("whisperx", "whispermlx"),
  asr_model = "large-v3",
```

```

min_speakers = NULL,
max_speakers = NULL,
language = NULL,
hf_token = NULL,
extra = NULL,
min_pause = 0.2,
speakers = NULL,
id = NULL,
cache_dir = NULL,
force = FALSE,
verbosity = c("normal", "quiet", "verbose")
)

```

Arguments

<code>x</code>	A character vector of paths to audio files, one row each. A path that does not resolve comes back with <code>exists = FALSE</code> rather than being dropped.
<code>...</code>	Not used, and refused if anything reaches it. Every argument after it is name-only, so a misspelling lands here rather than being matched by position.
<code>diarize_names</code>	A logical value indicating whether to label who is speaking. This is a transcription setting rather than a rendering one. It runs the diarizer, needs <code>hf_token</code> , and is in the cache key, so switching it re-transcribes. It also decides whether the text has turns in it at all, since a turn is a run of words by one speaker. Off, every silence is a [pause] and the transcript is one run of text. Default is <code>FALSE</code> .
<code>annotate_timing</code>	A logical value indicating whether silences are written into the text, as [pause 0.4s] inside a turn and [gap 0.8s] between two. Default is <code>FALSE</code> .
<code>tool</code>	A string naming the recognizer to run, "whisperx" (the default) or "whispermlx". Both emit the same JSON.
<code>asr_model</code>	A string naming the Whisper checkpoint, which decides the words. Both tools take the same short names. The command-line default is <code>small</code> , which is not the default here. Default is "large-v3".
<code>min_speakers, max_speakers</code>	Integers passed to the diarizer when known, or <code>NULL</code> (the default). An interview has two speakers and the researcher knows it beforehand, which is the cheapest reduction in diarization error available.
<code>language</code>	A string giving the spoken language, detected per file when <code>NULL</code> (the default).
<code>hf_token</code>	A string giving a HuggingFace read token, required when <code>diarize_names</code> is on. Default is <code>NULL</code> .
<code>extra</code>	A character vector of further flags for the recognizer, passed through verbatim and recorded on the result. Each element is its own argument, so <code>c("--beam_size", "10")</code> rather than one string. Default is <code>NULL</code> .
<code>min_pause</code>	A number giving the seconds below which a silence is not marked or counted. Jefferson leaves anything under 0.2 as an untimed micropause and gives a figure above it, which is where the default comes from.

speakers	A named character vector renaming the speakers, such as <code>c(SPEAKER_00 = "Interviewer")</code> . A name not in the timings is refused rather than ignored. Default is <code>NULL</code> .
id	A character vector of identifiers, one per file. Defaults to the file names without their extensions, and must be unique.
cache_dir	A string giving the directory where transcripts are written and looked for. Defaults to beside each recording.
force	A logical value indicating whether to transcribe again even where a usable transcript is cached. Default is <code>FALSE</code> .
verbosity	A string setting how much to report. <ul style="list-style-type: none"> • "quiet" reports nothing. • "normal" is the default. One line per recording as it is transcribed, or in an interactive session a bar redrawn in place when there is more than one. The recognizer's own output is held rather than printed, and appears only if the call fails. • "verbose" lets the recognizer write to the console as it runs. These tools report their dependencies at length on every invocation, so this is for one long file rather than for a corpus.

Value

A tibble of one row per file.

- `id`, `audio` and `exists` identify the recording.
- `text` is the transcript, with turns where `diarize_names` was on and silences where `annotate_timing` was.
- `duration_s` and `n_words` hold what the recording holds.
- `n_turns` and `n_speakers` are `NA` when the timings carry no speaker.
- `n pauses` counts the silences inside a turn at or above `min_pause`.
- `n_gaps` counts the silences at a turn boundary, `NA` when not diarized.
- `median_gap_s` is the middle silence between two words.
- `speech_rate` and `articulation_rate` are words per second over the whole recording, and over voiced time only. The first counts the silences and the second does not, so the ratio between them is how much of the elapsed time was pausing rather than talking.
- `median_turn_s` is the middle turn, `NA` without diarization.
- `median_score` is the middle alignment confidence, `NA` where the transcript carries none. Every silence figure rests on the word boundaries this describes, so it is what says how far to trust them.
- `diarized` says whether the transcript carried speakers, which is also whether the text has turns in it.
- `annotate_timing` says whether the silences were written in.
- `transcribed` says whether this call produced the transcript.
- `tool`, `asr_model` and `extra` hold what made the words, or `NA` where a transcript was found with no record beside it. `transcribed` and these together say whether a row was made now, read from a verified transcript, or read from one of unknown origin.
- `language` and `min_pause` are read off the transcript or applied by this call, so both hold however the transcript was obtained.

Caching

A transcript already on disk is read instead of transcribing again, so an interrupted run resumes and a hand-corrected transcript survives. `force = TRUE` returns to the recognizer.

What counts as already done is the question `measure()` asks of its checkpoints, which is whether the work was done under the settings now being requested rather than merely done. `tool`, `asr_model`, `diarize_names`, `language` and `extra` are recorded in a `.fathom.json` beside each transcript, and a run whose settings differ transcribes again. Without that, a call with `diarize_names` off in a directory that already holds a diarized transcript would read the diarized one and quietly ignore the argument.

`annotate_timing`, `min_pause` and `speakers` are not in that record, since none of them reaches the recognizer. Changing a rendering costs nothing and re-reads the same transcript.

A transcript with no record beside it is used rather than discarded, because that is what a hand-corrected one looks like. Its `tool`, `asr_model` and `extra` come back NA, since naming settings that did not produce those words would be worse than saying nothing.

Transcript format

The row a model is asked about is the whole recording, and the structure inside it is written into text rather than spread across rows.

Inside a row, a line is a turn, meaning a maximal run of words by one speaker. Nothing splits a turn further. A silence inside one person's talk is written where it happened, as `[pause 0.4s]`, which says how long it was rather than only that it passed a threshold.

Silence at a turn boundary is written `[gap 0.8s]` and belongs to neither speaker, so it goes on a line of its own. Whether a silence is a hesitation or a turn transition is a fact about the recording, so the two markers are told apart by whether the timings carry a speaker, not by whether labels were asked for.

Lines are newline separated with the speaker in brackets ahead of them.

Speakers

Speakers reach the output only from the recognizer. Absent, `n_speakers` and `n_turns` are NA and `diarized` is FALSE, so a transcript with no speaker in it is never read as a transcript with one speaker in it.

`speakers` renames what the recognizer emitted, since `SPEAKER_00` carries nothing a model can use. It is also the fastest way to put the answer in the prompt. A speaker renamed Angry Protester has been coded before the model saw the text, and nothing in the results would show it.

Pause and gap counts

`n_pauses` and `n_gaps` count what `min_pause` found in the timings, so they hold with `annotate_timing` off, where the text shows nothing. Only `text` and `annotate_timing` change with the rendering.

None of them reaches the model, since `measure()` reads `id` and `text` and nothing else.

`n_pauses` also moves with diarization on the same words, because without turns a turn transition is just another silence.

References

Jefferson, G. (1989). Preliminary notes on a possible metric which provides for a standard maximum silence of approximately one second in conversation. In D. Roger and P. Bull (eds), *Conversation. An Interdisciplinary Perspective*, 166–196. Multilingual Matters.

Stivers, T., Enfield, N. J., Brown, P., Englert, C., Hayashi, M., Heinemann, T., Hoymann, G., Rossano, F., de Ruiter, J. P., Yoon, K.-E., and Levinson, S. C. (2009). Universals and cultural variation in turn-taking in conversation. *Proceedings of the National Academy of Sciences* 106(26), 10587–10592. doi:[10.1073/pnas.0903616106](https://doi.org/10.1073/pnas.0903616106)

See Also

[prepare_video\(\)](#) for recordings with pictures, and [measure\(\)](#), which reads the text column. `vignette("quick_start", package = "fathom")` for installing the recognizer and the diarization token.

Examples

```
## Not run:
# Needs ffmpeg and a recognizer, so it is not run here.

# Everything off, which is a plain transcript and needs no token and no
# optional package.
plain <- prepare_audio("hearing_04.wav")

# Two speakers known in advance, labeled turns with the silences marked.
docs <- prepare_audio(
  c("hearing_04.wav", "hearing_11.wav"),
  diarize_names = TRUE,
  annotate_timing = TRUE,
  min_speakers = 2,
  max_speakers = 2,
  hf_token = Sys.getenv("HF_TOKEN")
)

# Anything else the recognizer takes.
tuned <- prepare_audio(
  "hearing_04.wav",
  extra = c("--no_speech_threshold", "0.4", "--beam_size", "10")
)

## End(Not run)
```

Description

Builds a tibble of one row per file, with the columns the rest of the package expects. `id` keys the results and the audit joins, `image` is the path `measure()` reads, and `bytes` is a size covariate.

Nothing is decoded. `bytes` comes from `file.size()`, which needs no image reader and no system library.

Usage

```
prepare_image(paths, id = NULL)
```

Arguments

<code>paths</code>	A character vector of file paths.
<code>id</code>	A character vector of identifiers, one per path. Defaults to the file name without its extension, which is what a corpus of scanned pages usually keys on.

Value

A tibble with `id`, `image`, `bytes`, `exists` and `ext`.

Missing files

A path that does not resolve comes back with `exists = FALSE` and `bytes` as `NA` rather than being silently removed, because a corpus that quietly lost four documents is the failure this exists to prevent. Filter on `exists` after reading the count.

See Also

[measure\(\)](#), which reads the image column.

Examples

```
f <- tempfile(fileext = ".png")
grDevices::png(f, width = 8, height = 8)
graphics::par(mar = c(0, 0, 0, 0))
graphics::plot.new()
grDevices::dev.off()

prepare_image(f)
```

prepare_video	<i>Turn videos into a corpus carrying both what was shown and what was said</i>
---------------	---

Description

Turns each recording into one row, with frames in a list column and a transcript beside them. `measure(text = , image = )` then reads both on the same call, since they are two columns of one row rather than two objects to align.

The audio half is `prepare_audio()` unchanged, running on a wav ffmpeg extracts.

Usage

```
prepare_video(
  x,
  ...,
  frames = 8,
  select = c("uniform", "change"),
  include_first = FALSE,
  min_gap = NULL,
  quality = 3,
  diarize_names = FALSE,
  annotate_timing = FALSE,
  tool = c("whisperx", "whispermlx"),
  asr_model = "large-v3",
  min_speakers = NULL,
  max_speakers = NULL,
  language = NULL,
  hf_token = NULL,
  extra = NULL,
  min_pause = 0.2,
  speakers = NULL,
  id = NULL,
  cache_dir = NULL,
  force = FALSE,
  verbosity = c("normal", "quiet", "verbose")
)
```

Arguments

<code>x</code>	A character vector of paths to video files, one row each. A vector rather than a directory, so <code>list.files()</code> is where the pattern lives.
<code>...</code>	Not used, and refused if anything reaches it. Every argument after it is name-only, so a misspelling lands here rather than being matched by position.
<code>frames</code>	An integer giving the number of frames per recording. Fixed across the corpus. Default is 8.

select	A string naming the rule that chooses the frames. "uniform" spreads them evenly and is the default. "change" takes the highest scene scores, which finds cuts and concentrates on visually busy stretches.
include_first	A logical value indicating whether the opening frame is always taken. It spends one of frames rather than adding to it, so n_frames does not vary with a flag. Frame zero has no predecessor and scores exactly zero, so change can never reach it otherwise, and an establishing shot is sometimes the most informative picture in a recording. Default is FALSE.
min_gap	A number giving the seconds below which two chosen frames may not sit. "change" only, and ignored rather than refused under "uniform", where even spacing cannot bunch. Defaults to duration / frames, which says no two frames sit closer than an even sample would put them, so "change" moves frames toward the action without being able to pack them into one stretch. Where that leaves fewer candidates than the budget, the remainder go in the middle of the widest space left.
quality	An integer giving the JPEG quality passed to ffmpeg as -q:v, 2 best to 31 worst. Default is 3.
diarize_names	A logical value indicating whether to label who is speaking. This is a transcription setting rather than a rendering one. It runs the diarizer, needs hf_token, and is in the cache key, so switching it re-transcribes. It also decides whether the text has turns in it at all, since a turn is a run of words by one speaker. Off, every silence is a [pause] and the transcript is one run of text. Default is FALSE.
annotate_timing	A logical value indicating whether silences are written into the text, as [pause 0.4s] inside a turn and [gap 0.8s] between two. Default is FALSE.
tool	A string naming the recognizer to run, "whisperx" (the default) or "whispermlx". Both emit the same JSON.
asr_model	A string naming the Whisper checkpoint, which decides the words. Both tools take the same short names. The command-line default is small, which is not the default here. Default is "large-v3".
min_speakers, max_speakers	Passed to prepare_audio() on the extracted audio, behaving there exactly as on a recording handed to it directly.
language	A string giving the spoken language, detected per file when NULL (the default).
hf_token	A string giving a HuggingFace read token, required when diarize_names is on. Default is NULL.
extra	A character vector of further flags for the recognizer, passed through verbatim and recorded on the result. Each element is its own argument, so c("--beam_size", "10") rather than one string. Default is NULL.
min_pause	A number giving the seconds below which a silence is not marked or counted. Jefferson leaves anything under 0.2 as an untimed micropause and gives a figure above it, which is where the default comes from.
speakers	A named character vector renaming the speakers, such as c(SPEAKER_00 = "Interviewer"). A name not in the timings is refused rather than ignored. Default is NULL.

id	A character vector of identifiers, one per file. Defaults to the file names without their extensions, and must be unique.
cache_dir	A string giving the directory where frames, the wav and the transcript are written and looked for. Defaults to beside each recording.
force	A logical value indicating whether to extract and transcribe again even where usable output is cached. Default is FALSE.
verbosity	A string setting how much to report. <ul style="list-style-type: none"> • "quiet" reports nothing. • "normal" is the default. One line per recording as it is transcribed, or in an interactive session a bar redrawn in place when there is more than one. The recognizer's own output is held rather than printed, and appears only if the call fails. • "verbose" lets the recognizer write to the console as it runs. These tools report their dependencies at length on every invocation, so this is for one long file rather than for a corpus.

Value

A tibble of one row per file, carrying every column `prepare_audio()` returns plus these.

- video is the recording, and has_audio says whether it had a sound track at all.
- image is a list column of frame paths in time order, which is the shape `measure(image = )` takes for several pictures per document.
- n_frames and frame_times say how many were taken and when.
- select and include_first name which rule chose them.
- scene_median and scene_max are the middle and largest scene score across every frame, which say whether the recording had cuts in it or whether the spacing did the work. Read before trusting "change".

Choosing frames

`select = "uniform"` divides the recording into frames equal segments and takes the middle of each. Midpoints rather than endpoints, since a frame at exactly zero is often a title card and one at exactly the end is often black.

`select = "change"` takes the frames highest scene scores instead, which `ffmpeg` computes for every frame as how much this picture differs from the one before it. It finds cuts that even spacing misses, and it concentrates. `min_gap` is what stops that, and it exists only for this mode.

Recordings without audio

`ffprobe` is asked whether an audio stream exists before anything is extracted. Where there is none, the row comes back with its frames and with `has_audio = FALSE`, `text` and every audio count NA, and the count is reported. A batch of campaign advertisements does not stop because three of them are silent.

`exists` and `has_audio` answer different questions and both are needed.

case	exists	has_audio	text
file not found	FALSE	NA	NA
present, silent	TRUE	FALSE	NA
present, with audio	TRUE	TRUE	the transcript

NA on has_audio is doing work. The file was never opened, so FALSE would assert something that was not checked.

n_words is NA for a silent recording rather than 0, because nothing was transcribed. A recording whose audio track is silence is a different thing, gets transcribed, and comes back with n_words = 0.

Every other audio column is NA there too, diarized and tool and asr_model included. Nothing ran, so there is nothing to report and naming a model would assert settings that produced no words.

Frame storage

On disk beside the recording, in <stem>_frames/, because measure(image =) reads paths.

The way lands beside it, which is what lets [prepare_audio\(\)](#)'s cache work unchanged. The frames keep a record of their own, <stem>_frames.fathom.json in the frames directory, holding the times they were cut at and the JPEG quality. Changing frames, select, include_first, min_gap or quality changes that record, so every frame is cut again and the old pictures are removed, while the transcript is read back from its own cache.

See Also

[prepare_audio\(\)](#), which transcribes the sound track, and [measure\(\)](#), which reads both columns.

Examples

```
## Not run:
# Needs ffmpeg, a recognizer and an Ollama server, so it is not run here.

ads <- list.files("campaign_ads", "[.]mp4$", full.names = TRUE)
ins <- instrument(c(attack = "Does this advertisement attack an opponent?"))

# Frames spread evenly, and whatever was said alongside them.
d <- prepare_video(ads, frames = 8)
measure(
  d,
  backend = ollama_backend("gemma3:27b"),
  instrument = ins,
  text = "text",
  image = "image"
)

# More frames, chosen where the picture moved, with the speakers labeled.
d <- prepare_video(
  ads,
  frames = 16,
  select = "change",
```

```

    diarize_names = TRUE,
    annotate_timing = TRUE,
    hf_token = Sys.getenv("HF_TOKEN")
  )

  ## End(Not run)

```

```
preview
```

```
Print the assembled prompt
```

Description

Prints the exact string the model will receive. Makes no model calls. Prompt failures do not appear in the output of `measure()`.

Usage

```

preview(
  instrument,
  text = "[document text goes here]",
  item = NULL,
  image = NULL,
  truncate = 300L
)

```

Arguments

<code>instrument</code>	An <code>instrument()</code> .
<code>text</code>	An example document. A short placeholder by default.
<code>item</code>	A string naming the item to show. Defaults to the first.
<code>image</code>	A character vector of paths, one or several, to the media the run would send, or NULL (the default) to send none. A video window is a vector of frame paths. Their presence removes the document block from the prompt, since an image corpus has no text to quote, and adds a bracketed line above it. That line is shown and never sent. Anything in square brackets here describes what travels beside the prompt rather than quoting what is inside it.
<code>truncate</code>	An integer giving the number of characters of document text to show. NULL shows all. Default is 300.

Value

Invisibly, the assembled prompt string for the item shown, exactly as the model would receive it. A single character scalar.

The printed output is larger than the return value and the difference matters. Printing shows the prompt and, beneath it, the table of what each label will be matched against. That table is where most extraction failures are visible, and it is not in the returned string.

So capture the value to diff two prompts or to paste one into a paper, and read the printed output to check the instrument. No call is made either way.

The matching table

The table underneath has two columns. On the left the label, on the right every token that will be counted as that label. They read the same when a label is its own only variant, which is the usual arrangement. They diverge once accented forms are supplied, and again after probing registers whatever token a split label actually emits.

Under prefix matching the right column is not the whole story, since a token also counts when it merely opens one of those forms. BU counts as BUDGET without appearing there. The instrument header says which rule is in force.

See Also

`instrument()` for the prompt's parts and `measure()` for sending it.

Examples

```
b <- instrument(
  template_items("topic", c(econ = "the economy", immig = "immigration"),
    medium = "text"),
  labels = c("YES", "NO")
)
preview(b)
```

read_checkpoints	<i>Read a checkpoint directory back into results</i>
------------------	--

Description

Reconstructs the return value of `measure()` from disk without making any model calls. Use when a run completed but the return value was lost, or to inspect how far a killed run progressed.

Usage

```
read_checkpoints(dir)
```

Arguments

`dir` A string giving the checkpoint directory.

Value

A `fathom_results` object, the two frames `measure()` returns and documents there. Rows arrive in whatever order the directory listing gives, so sort if order matters.

The attributes are read back off the stored payloads rather than stamped from the current session, so a corpus reports the run that produced it.

- `instrument` is the instrument as it was when the run happened.

- num_ctx is the ceiling that run used, not the one a run would get now. A resumed run keeps this, which is why mixing ceilings inside one corpus cannot happen silently.
- exclusive says whether the instrument shared one distribution.
- run_at says when the run happened, not when it was read.

A file that cannot be read is treated as absent rather than as an error, since a killed run can leave one half-written and re-extracting that document is the only recoverable reading. Errors when the directory holds no readable checkpoints at all.

Provenance

The context ceiling, the exclusivity flag and the run timestamp are read from disk rather than inferred from the current session. A run written under one ceiling reports that ceiling even if the current session would resolve a different one.

See Also

[measure\(\)](#), whose checkpoint argument writes what this reads.

Examples

```
## Not run:
# Needs a running Ollama server, so it is not run here.
oll <- ollama_backend("gemma4:31b")
b <- instrument(c(tone = "Is this sentence positive?"),
  labels = c("YES", "NO"))
res <- measure(parlasent, instrument = b, backend = oll, text = "sentence",
  id = "id", checkpoint = "ckpt")

# A run cut short leaves its finished documents in the directory.
res <- read_checkpoints("ckpt")

## End(Not run)
```

template_items

Build items by substituting categories into one question stem

Description

Turns a list of categories into an instrument of independent binary items. The category goes into the question, not the answer, so categories may be any length and the tokenization constraint never applies.

Usage

```
template_items(stem, categories, lang = c("en", "es"), medium = NULL)
```

Arguments

stem	A named stem, or a custom string containing {label}.
categories	A character vector of categories. Names, if present, become item names.
lang	A string giving the language for the named stems, "en" (the default) or "es". Ignored for a custom string, which is used exactly as written. This sets the question only. The scaffolding around it is <code>instrument()</code> 's own lang, and leaving that at its default gives a Spanish question under an English cue.
medium	A string naming the stem set a named stem is looked up in, one of "text", "audio", "image" or "video". Read only for a named stem, since a custom string carries its own wording. There is no default, because the sets differ by medium rather than translating, so a default would answer a question about the document with a guess.

Value

A named list of questions, one per category, ready to pass to `instrument()` as its first argument.

Names are the names of categories, so they become the item names in the results and in `item_report()`. Pass a named vector to control them. An unnamed vector uses the values themselves, which rarely suits a category that runs to a phrase.

Values are the stem with {label} replaced by each category's text. Order is preserved, and an instrument reads in the order its items were declared rather than alphabetically.

The result is an ordinary list, so an item can be edited before it reaches `instrument()` when one category needs wording the stem cannot give it.

The named stems

Four sets, one per medium, each in English and in Spanish. `medium` picks the set and `lang` picks the language.

medium	stems
"text"	topic, mention, express, about, frame
"audio"	topic, mention, express, about, frame
"image"	show, depict, express, about, frame
"video"	show, depict, discuss, express, about, frame

The sets are not translations of one another, which is why `medium` has no default.

The text set in full. The other three take the same shape with their own noun and their own verbs.

stem	en	es
"topic"	Does this text discuss {label}?	¿Este texto trata sobre {label}?
"mention"	Does this text mention {label}?	¿Este texto menciona {label}?
"express"	Does this text express {label}?	¿Este texto expresa {label}?
"about"	Is this text primarily about {label}?	¿Este texto trata principalmente de {label}?
"frame"	Does this text frame the issue in terms of {label}?	¿Este texto enmarca el tema en términos de {label}?

These differ in what they assert, not in topic. "mention" fires on a passing reference. "about" requires the text to be centrally concerned with the category. The choice is a measurement decision, and it is the same decision in either language, since the two halves of a row are translations of one another rather than separate instruments.

A custom string ignores lang entirely, so a Spanish question written by hand is passed through as typed. The scaffolding around it is a separate setting, on `instrument()`, and the two can disagree. See its lang argument.

See Also

`instrument()`, which takes the items.

Examples

```
template_items("topic", c(econ = "the economy", immig = "immigration"),
               medium = "text")

# The same stem name asks a different question of a different document.
template_items("about", c(econ = "the economy"), medium = "image")

# A custom string carries its own wording, so it needs no medium.
template_items("Does the author appear to endorse {label}?", c("austerity"))
```

`threshold_sensitivity` *Compute endorsement rates across a range of thresholds*

Description

Computes the endorsement rate per item at each cut point, which shows how much of an item's rate is a property of the documents and how much is a property of where the cut falls.

Usage

```
threshold_sensitivity(
  results,
  target = NULL,
  thresholds = seq(0.1, 0.9, by = 0.1),
  value = c("prob_renorm", "prob")
)
```

Arguments

<code>results</code>	A fathom_results object from <code>measure()</code> or <code>read_checkpoints()</code> .
<code>target</code>	A string naming the label counted as endorsement. Defaults to the first.
<code>thresholds</code>	A numeric vector of cut points to evaluate. Default is <code>seq(0.1, 0.9, by = 0.1)</code> .

value A string naming the probability column to threshold. Defaults to "prob_renorm", the figure after the label probabilities have been rescaled to sum to 1. That is the question a cut point is usually being asked to answer, which is whether documents pile up near the boundary between your labels, and it is the quantity everything downstream compares against.

"prob" is the raw probability, so a sweep over it also moves with mass the model spent outside your labels. That confounds two things, and a rate that shifts under it may be reporting compliance rather than the corpus. Use it when the claim is about whether the model committed.

Value

A tibble with one row per item and threshold, items in instrument order and thresholds ascending within an item. The message printed alongside is a ranking rather than a table and stays widest swing first.

- `item` is the question name.
- `threshold` is the cut point, one row per value in thresholds.
- `n` counts the documents scored on this item. The same on every row of an item, since the cut point changes which documents count as endorsing and not how many were asked.
- `n_endorse` counts the documents whose value for target reached the cut.
- `endorse` is `n_endorse` over the documents with a value to cut. A document where no label appeared has no `prob_renorm`, so under the default it leaves this denominator, and `endorse` is `n_endorse` over `n` only when no such document exists. That is the split `item_report()` makes between `endorse_renorm` and `endorse`. This is the column to read down, and the spread from its smallest to largest value within an item is what the message ranks on.

More than two labels

This assumes a two-label instrument. Thresholding one label's probability asks whether that label is present, which is a question only a two-label instrument answers. On an instrument of three or more labels the labels compete for one pool of probability, so a label can win at 0.3 and lose at 0.6, and its rate at a series of cut points describes the arithmetic rather than the corpus. On a scored instrument the labels are levels of a scale, and a cut point on one level is not a measurement of anything.

The function does not refuse, because a cut point is the whole subject here and there is no other quantity for it to report. It runs on whatever you pass and the figures will be arithmetically correct.

See Also

`item_report()` for the winner distribution, and for `mean_expected` on a scored instrument, which needs no cut.

Examples

```
# Results from a real run ship with the package, so this needs no server.
p <- system.file("extdata", "vignette_results.rds", package = "fathom")
baked <- readRDS(p)
threshold_sensitivity(baked$res_indep, target = "YES")
```

validate_sample

Validate a completed coding sheet against the model

Description

Takes the sheet `coding_sample()` wrote, with `human_label` filled in, and reports how well the model agreed with the coder.

The sheet carries no probabilities, which is what makes it blind. This joins back to the results on `id` and `item` to recover them, so pass the same results object the sample came from.

`vignette("complete_run")` works a run end to end.

Usage

```
validate_sample(
  sheet,
  results,
  ...,
  level = c("nominal", "ordinal"),
  labels = NULL,
  instrument = NULL,
  calibration = FALSE,
  coded = FALSE,
  ci = c("none", "bootstrap"),
  conf_level = 0.95,
  bootstrap_n = 2000L,
  seed = 1234L
)
```

Arguments

<code>sheet</code>	A data frame holding the completed coding sheet, or a string giving a path to the CSV. A named list of sheets scores the model against every coder in it and adds the ceiling and the gap. See the section below.
<code>results</code>	A <code>fathom_results</code> object, the one the sheet was drawn from.
<code>...</code>	Not used, and refused if anything reaches it. Every argument after it is name-only, so a misspelling lands here rather than being matched by position.
<code>level</code>	A string, "nominal" (the default) or "ordinal". It decides how a disagreement is charged, so it selects the proper scoring rule in <code>\$calibration</code> as well as the <code>alpha</code> .
<code>labels</code>	A character vector giving, under "ordinal", every label in scale order. Ignored under "nominal". Default is <code>NULL</code> .
<code>instrument</code>	A <code>fathom_instrument</code> , the one the run used. Under "ordinal" its scores supply the scale, so labels need not be retyped. An explicit labels wins over it. Default is <code>NULL</code> .

calibration	A logical value indicating whether to return \$calibration, for a run keeping prob_renorm and feeding it onward. Default is FALSE.
coded	A logical value indicating whether to return \$coded, the sheet joined to the model's answer. Default is FALSE.
ci	A string choosing whether to report intervals. "none", the default, returns none and the interval columns are absent rather than present and full of NA. "bootstrap" adds one bracketed column beside each figure.
conf_level	A number giving the coverage of every interval reported, 0.95 by default. It is stamped on the result as the conf_level attribute, and the print method states it under the first table that carries an interval.
bootstrap_n	An integer giving the replicates for each interval. The default is a round two thousand. Below a hundred a percentile interval is reading a tail made of a handful of draws, and is refused.
seed	An integer giving the seed for the bootstrap, so the intervals come out the same on every run with the same inputs. The session's random stream is put back on exit, so nothing drawn before this call reaches the intervals and this seed reaches nothing after it. NULL draws from the session's stream instead. Default is 1234.

Value

An object of class `fathom_validation`, a list of tables. The model tables carry `item` and, on the several-sheet path, `coder`. Under `ci = "bootstrap"` each figure gains a bracketed `_ci` column beside it.

component	holds	when
\$discrimination	how often the model chose the coder's label	a label sheet
\$categories	precision, recall and AUC for each label	a label sheet
\$calibration	how honest the probabilities were	calibration = TRUE, or a soft sheet
\$ceiling	how well the coders agreed with each other	several sheets
\$gap	the model against that ceiling	several sheets
\$coded	the sheet joined to the model's answer	coded = TRUE

Discrimination:

One row per question, plus a pooled (all items) row where the instrument has more than one.

- `n` counts the coded rows.
- `agreement` is the share of rows where the model chose the coder's label.
- `alpha` is Krippendorff's alpha (Hayes and Krippendorff 2007) between the model and the coder, under the level asked for.

Categories:

One row per label per question, with `label` and `n`, the rows the coder gave that label.

- `precision` is the share of the rows the model gave the label where the coder chose it too.
- `recall` is the share of the rows the coder gave the label where the model chose it too.
- `auc` is the chance that a row the coder gave the label gets a higher probability for it than a row the coder did not.

All three count exact matches, so under "ordinal" a model one level off is charged as a full miss. alpha and rps are the figures that respect the order. On two labels the two auc rows are equal, since one label's probability is one minus the other's.

Calibration:

One row per question, plus the pooled row, with n and one score column named for the rule level selects.

- brier is squared error summed across categories, so on two labels a confident correct answer costs 0, a confident wrong one costs 2, and never committing costs 0.5.
- rps is the same sum on the cumulative probabilities and runs from 0 to k - 1. Lower is better for both. A soft reference stays on brier.

Ceiling:

\$ceiling\$alpha is one row per question with n_units, the rows two or more coders reached, and alpha over all coders at once. \$ceiling\$pairs is one row per pair of coders per question, with coder_a, coder_b, n, the rows both coded, agreement and alpha. This is what the coders managed on these rows, and not a target.

Gap:

One row per question.

- n counts the rows the model and the coders share.
- model_alpha is the model's average alpha with the coders.
- coder_alpha is the coders' average alpha with each other on the same rows.
- gap is the first minus the second, positive where the model agreed with the coders more closely than they agreed with each other, and gap_ci is its interval.

Coded:

The sheet joined to the model's label and its probabilities, one row per coded document-item, or per document-item per coder on the several-sheet path. It is returned because a design-based correction takes it as input (Egami et al. 2023, Angelopoulos et al. 2023).

The attributes carry level, conf_level, the number of coded rows as n_coded, and on the several-sheet path the coder names as coders.

Several coders

sheet takes a named list as well as one sheet, so validate_sample(list(ana = ana, luis = luis), res) scores the model against each coder in one call and adds \$ceiling and \$gap. Each coder is scored through the one-sheet path, so no figure is computed twice or computed differently, and the model tables gain a coder column after item.

Coders need not have worked every row. \$ceiling\$alpha uses every row two or more coders reached, and each pair in \$ceiling\$pairs falls back to the rows both of them saw, so the two tables can rest on different counts.

\$gap resamples whatever ci is set to, since the gap is read through its interval. Every replicate redraws the documents once and recomputes both sides on that draw, because the two share the documents they rest on.

A list of one is refused. A sheet `coder_reference()` consolidated warns when it is placed in a list, since agreeing with a consensus is easier than agreeing with one person.

Ordered labels

Ordered labels need `level = "ordinal"`. The default charges every disagreement the same, which is right for unordered labels and wrong for a scale, where two coders one step apart have agreed more than two coders at opposite ends. Pass labels in scale order, least to greatest, or an instrument carrying scores. Nothing is inferred, because no order can be recovered from the sheets.

A soft reference

A sheet carrying `human_prop` instead of `human_label` is scored as a measurement and not as a classification. There is no second categorical variable, so `$discrimination` and `$categories` are not returned. What comes back is `$calibration`, with brier against the proportion, whatever calibration is set to.

References

- Angelopoulos, A. N., Bates, S., Fannjiang, C., Jordan, M. I., and Zrnic, T. (2023). Prediction-powered inference. *Science* 382(6671), 669–674. doi:[10.1126/science.adi6000](https://doi.org/10.1126/science.adi6000)
- Brier, G. W. (1950). Verification of forecasts expressed in terms of probability. *Monthly Weather Review* 78(1), 1–3. doi:[10.1175/15200493\(1950\)078<0001:VOFEIT>2.0.CO;2](https://doi.org/10.1175/15200493(1950)078<0001:VOFEIT>2.0.CO;2)
- Egami, N., Hinck, M., Stewart, B., and Wei, H. (2023). Using imperfect surrogates for downstream inference. Design-based supervised learning for social science applications of large language models. *Advances in Neural Information Processing Systems* 36, 68589–68601. doi:[10.52202/075280-3000](https://doi.org/10.52202/075280-3000)
- Hayes, A. F., and Krippendorff, K. (2007). Answering the call for a standard reliability measure for coding data. *Communication Methods and Measures* 1(1), 77–89. doi:[10.1080/19312450709336664](https://doi.org/10.1080/19312450709336664)

See Also

[coding_sample\(\)](#) for drawing the sheet, and [coder_reference\(\)](#) for consolidating several coders into one sheet.

Examples

```
# Results from a real run ship with the package, so this needs no server.
p <- system.file("extdata", "vignette_results.rds", package = "fathom")
baked <- readRDS(p)
res <- baked$res_six
sheet <- coding_sample(res, parlasent, text = "sentence", id = "id", n = 20,
  seed = 1234)

# The parlasent coders, standing in for someone filling the sheet in.
six <- c("Negative", "M_Negative", "N_Neutral", "P_Neutral", "M_Positive",
  "Positive")
coded <- function(who) {
  out <- sheet
  out$human_label <- as.character(
    match(parlasent[[who]][match(sheet$id, parlasent$id)], six)
  )
  out
}
```

```
}  
a1 <- coded("annotator1")  
  
v <- validate_sample(a1, res, level = "ordinal", labels = as.character(1:6))  
v$discrimination  
v$categories  
  
# A second coder adds the ceiling the two set between themselves, and the  
# model against it.  
a2 <- coded("annotator2")  
v2 <- validate_sample(list(annotator1 = a1, annotator2 = a2), res,  
  level = "ordinal", labels = as.character(1:6), bootstrap_n = 200)  
v2$ceiling$pairs  
v2$gap
```

Index

- * **datasets**
 - ati, [5](#)
 - parlasent, [27](#)
- as_wide, [2](#)
- ati, [5](#)
- coder_reference, [6](#)
- coder_reference(), [9](#), [47](#), [48](#)
- coding_sample, [7](#)
- coding_sample(), [11](#), [45](#), [48](#)
- inspect_cases, [9](#)
- inspect_cases(), [17](#), [20](#), [23](#), [24](#)
- instrument, [12](#)
- instrument(), [21](#), [24](#), [26–28](#), [39](#), [40](#), [42](#), [43](#)
- item_report, [17](#)
- item_report(), [4](#), [11](#), [14](#), [24](#), [42](#), [44](#)
- measure, [20](#)
- measure(), [3](#), [4](#), [8](#), [10](#), [11](#), [16](#), [17](#), [25–27](#),
[32–34](#), [38–41](#), [43](#)
- ollama_backend, [25](#)
- ollama_backend(), [14](#), [21](#), [24](#)
- parlasent, [27](#)
- prepare_audio, [29](#)
- prepare_audio(), [26](#), [35–38](#)
- prepare_image, [33](#)
- prepare_video, [35](#)
- prepare_video(), [8](#), [26](#), [33](#)
- preview, [39](#)
- preview(), [14](#), [16](#), [24](#)
- read_checkpoints, [40](#)
- read_checkpoints(), [3](#), [8](#), [10](#), [17](#), [23](#), [24](#), [43](#)
- suppressMessages(), [22](#)
- template_items, [41](#)
- template_items(), [16](#)
- threshold_sensitivity, [43](#)
- threshold_sensitivity(), [19](#), [20](#)
- validate_sample, [45](#)
- validate_sample(), [3](#), [6](#), [7](#), [9](#), [20](#)