Package 'acR'

November 7, 2026

Type Package

Title Content Analysis in R: Integrated Qualitative (LLMs) and Quantitative Pipeline

Version 0.6.1

Description Provides an integrated pipeline for content analysis combining qualitative coding assisted by large language models (LLMs) with classical quantitative text analysis. Includes modules for pre-processing (tokenization, stopwords for Brazilian Portuguese), descriptive statistics, keyness, co-occurrence networks, word clouds (including comparative and X-ray variants), sentiment analysis via OpLexicon, Latent Dirichlet Allocation (LDA), inter-coder reliability metrics (Krippendorff, Gwet), and modern visualizations based on ggplot2. Special focus on Brazilian corpora and political-institutional codebooks. Inspired by Maerz and Benoit (2025) <https://quallmer.github.io/quallmer/>.

License MIT + file LICENSE

URL https://github.com/andersonheri/acR, https://ahenriquecp.com/acR/

BugReports https://github.com/andersonheri/acR/issues

Depends R (>= 4.1)

Imports cli, dplyr, httr2, irr, jsonlite, purrr, rlang, stats, stringr, tibble, tidyr, tidyselect, utils, withr, yaml

Suggests callr, cluster, covr, ellmer, ggplot2, ggraph, ggwordcloud, igraph, ipeaplot, knitr, shiny, openxlsx, pkgdown, quanteda, readtext, rmarkdown, scales, senatebR, spelling, stopwords, stringi, tesseract, testthat, tidytext, topicmodels, wordcloud, writexl

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Language en-US

RoxygenNote 7.3.2

NeedsCompilation no

Author Anderson Henrique [aut, cre] (ORCID:
<https://orcid.org/0000-0002-1842-2725>)

Maintainer Anderson Henrique <anderson.henrique@usp.br>

Repository CRAN

Date/Publication 2026-11-07 21:30:02 UTC

Contents

ac_build_dictionary . . . 3
ac_ca . . . 4
ac_calibrate_tokens . . . 6
ac_chat_local . . . 7
ac_clean . . . 8
ac_clean_stopwords . . . 11
ac_cluster_documents . . . 12
ac_collocations . . . 14
ac_cooccurrence . . . 16
ac_corpus . . . 18
ac_count . . . 20
ac_dictionary . . . 21
ac_dictionary_import . . . 23
ac_dictionary_save . . . 24
ac_dictionary_validate . . . 25
ac_emotion . . . 26
ac_estimate_tokens . . . 28
ac_export . . . 29
ac_fetch_camara . . . 31
ac_fetch_senado . . . 33
ac_import . . . 36
ac_keyness . . . 38
ac_kwic . . . 40
ac_lda . . . 41
ac_lda_tune . . . 43
ac_lexical_diversity . . . 45
ac_palette . . . 46
ac_plot_ca . . . 47
ac_plot_cluster . . . 48
ac_plot_cooccurrence . . . 49
ac_plot_keyness . . . 51
ac_plot_lda_topics . . . 52
ac_plot_lda_tune . . . 53
ac_plot_lexical_diversity . . . 54
ac_plot_readability . . . 55
ac_plot_sentiment . . . 57
ac_plot_temporal . . . 58

ac_plot_tf_idf . . . 59
ac_plot_top_terms . . . 60
ac_plot_wordcloud_comparative . . . 62
ac_plot_wordscores . . . 63
ac_plot_xray . . . 64
ac_plot_zipf . . . 65
ac_qual_code . . . 66
ac_qual_codebook . . . 71
ac_qual_codebook_add . . . 73
ac_qual_codebook_history . . . 74
ac_qual_codebook_hybrid . . . 75
ac_qual_codebook_merge . . . 76
ac_qual_codebook_remove . . . 78
ac_qual_codebook_translate . . . 79
ac_qual_export_for_review . . . 80
ac_qual_import_human . . . 81
ac_qual_irr . . . 82
ac_qual_list_models . . . 85
ac_qual_load_codebook . . . 86
ac_qual_recommend_model . . . 87
ac_qual_reliability . . . 89
ac_qual_report . . . 91
ac_qual_report_full . . . 93
ac_qual_sample . . . 95
ac_qual_save_codebook . . . 97
ac_qual_search_literature . . . 98
ac_readability . . . 100
ac_sentiment . . . 102
ac_temporal . . . 104
ac_tf_idf . . . 106
ac_tokenize . . . 107
ac_top_terms . . . 109
ac_wordcloud . . . 111
ac_wordscores . . . 112
as_prompt . . . 114
is_ac_corpus . . . 115
theme_ac . . . 116

Index **118**

`ac_build_dictionary` *Construir um dicionário multi-categoria*

Description

`ac_build_dictionary()` monta um dicionário no formato esperado por `ac_dictionary()` a partir de argumentos nomeados: cada argumento é uma categoria, e seu valor é o vetor de palavras daquela categoria.

Usage

```
ac_build_dictionary(...)
```

Arguments

...	Argumentos nomeados, cada um um vetor de caracteres.

Value

Lista nomeada de vetores de caracteres (um dicionário).

See Also

`ac_dictionary()`, `ac_dictionary_import()`, `ac_dictionary_save()`

Examples

```
dict <- ac_build_dictionary(
  positivo = c("bom", "ótimo", "excelente"),
  negativo = c("mau", "péssimo", "horrivel")
)
dict
```

---

ac_ca *Análise de correspondência (CA) entre documentos/grupos e termos*

Description

`ac_ca()` reduz uma tabela de contingência documento/grupo x termo a poucas dimensões, posicionando unidades e termos no mesmo espaço geométrico (Greenacre, 2007). Unidades próximas entre si têm perfis de vocabulário parecidos; termos próximos entre si são usados de forma parecida pelos grupos. É uma alternativa simples e determinística à Alocação de Dirichlet Latente (LDA, já disponível em `ac_lda()`) para visualizar associações entre vocabulário e grupos em duas dimensões: sem assumir um número de tópicos latentes.

Como ler o mapa. As coordenadas de unidades e termos são principais (mapa "simétrico"). Nesse mapa, distâncias **entre unidades** (e entre termos) aproximam distâncias qui-quadrado entre perfis, mas a distância entre uma unidade e um termo **não** tem interpretação direta (Greenacre, 2007, cap. 8-9). Leia a relação unidade-termo pela **direção**: um termo longe da origem e na mesma direção de um grupo é sobrerrepresentado nele; termos perto da origem têm perfil parecido com a média (pouco distintivos).

É útil, por exemplo, para visualizar como o vocabulário difere entre categorias de atendimento ao cliente, entre seções de um conjunto de notícias, ou entre partidos/períodos legislativos: qualquer situação em que se queira um "mapa" conjunto de grupos e das palavras que os caracterizam.

Implementação: SVD (`base::svd()`) sobre os resíduos padronizados de Pearson da tabela de contingência normalizada, no espírito do algoritmo clássico de Análise de Correspondência (Greenacre, 2007). Os sinais de cada dimensão são normalizados (a unidade com maior coordenada em valor absoluto é sempre positiva), para que o resultado seja determinístico entre execuções.

Usage

```
ac_ca(
  corpus,
  by = NULL,
  top_n = 50L,
  min_count = 5L,
  n_dim = 2L,
  drop_punct = TRUE
)
```

Arguments

corpus	Objeto ac_corpus.
by	Nome de uma coluna de metadados em corpus para definir os grupos (linhas da tabela de contingência). Se NULL (padrão), cada documento é uma linha.
top_n	Número máximo de termos (colunas) a manter, pelos mais frequentes no total. Padrão: 50.
min_count	Frequência total mínima de um termo para ele ser mantido. Padrão: 5.
n_dim	Número de dimensões a retornar. Padrão: 2.
drop_punct	Lógico, encaminhado para ac_count(). Padrão: TRUE.

Value

Lista de classe ac_ca com:

- row_coords: tibble com unit, dim1, dim2, ..., mass (massa/peso relativo da linha).
- col_coords: tibble com term, dim1, dim2, ..., mass.
- eigenvalues: autovalores (inércia) de cada dimensão.
- variance_explained: proporção da inércia total explicada por cada dimensão.

References

Greenacre, M. (2007). *Correspondence Analysis in Practice* (2nd ed.). Chapman and Hall/CRC.

See Also

ac_count(), ac_lda(), ac_plot_ca()

Examples

```
df <- data.frame(
  id = paste0("d", 1:6),
  texto = c(
    "entrega atrasada rastreio sumiu",
    "pedido atrasou muito, rastreio não atualiza",
    "produto com defeito, chegou quebrado",
    "veio quebrado, qualidade péssima",
    "atendimento excelente, resolveram rápido",
```

```
    "suporte educado e rápido, resolveu tudo"
  ),
  categoria = c("entrega", "entrega", "produto", "produto",
                "atendimento", "atendimento"),
  stringsAsFactors = FALSE
)
corpus <- ac_corpus(df, text = texto, docid = id, meta = categoria)

ca <- ac_ca(corpus, by = "categoria", min_count = 1)
ca$variance_explained
```

`ac_calibrate_tokens` *Calibrar a razao caracteres/token contra um modelo real*

Description

`ac_calibrate_tokens()` mede, contra um `Chat` real, quantos tokens de entrada um conjunto de textos de amostra realmente consome, e devolve a razao caracteres/token observada – para usar como `chars_per_token` em `ac_estimate_tokens()` (ou como `chars_per_token` em `ac_qual_code()` / futuras funções que dependam da estimativa).

Faz uma chamada real ao modelo por texto de amostra (com uma mensagem mínima, para medir so o consumo de tokens de entrada), então usa poucos textos – o suficiente para calibrar, não uma amostra representativa de todo o corpus.

Usage

```
ac_calibrate_tokens(chat_factory, textos)
```

Arguments

`chat_factory`	Função `function(temperature, system_prompt)` `Chat` que constroi o chat a usar (o mesmo padrao aceito por `model` em `ac_qual_code()`). Permite calibrar contra o provedor/modelo exato que sera usado na rodada real.
`textos`	Vetor de caracteres com textos de amostra (poucos bastam; ideal variar o tamanho).

Value

Número: a mediana de caracteres por token observada na amostra.

Examples

```
## Not run:
chat_factory <- function(temperature, system_prompt) {
  ellmer::chat_openai(model = "gpt-4.1-mini", temperature = temperature,
                      system_prompt = system_prompt, echo = "none")
}
razao <- ac_calibrate_tokens(chat_factory, c("texto curto de teste.",
```

```
                                                "um texto um pouco mais longo de teste."))
coded <- ac_qual_code(corpus, cb, model = chat_factory)

## End(Not run)
```

`ac_chat_local` *Conectar a um servidor LLM local compativel com OpenAI*

Description

`ac_chat_local()` cria um `Chat` do `ellmer` apontando para um servidor local compativel com a API da OpenAI – LM Studio, llama.cpp (`llama-server`), vLLM, Ollama em modo OpenAI, etc. – sem exigir uma chave de API real.

Classificar com um modelo local (via `ac_qual_code()`) elimina o custo por token e mantem os documentos fora de qualquer rede externa, ao preco de gerência propria do servidor: tamanho de contexto, tempo de resposta e throughput passam a depender do hardware local. Use junto com `checkpoint_dir` e `max_retries` em `ac_qual_code()` para rodadas longas, e `ac_estimate_tokens()` para evitar estouro de contexto.

Usage

```
ac_chat_local(
  model,
  base_url = "http://localhost:1234/v1",
  api = c("chat_completions", "responses"),
  temperature = NULL,
  system_prompt = NULL,
  ...
)
```

Arguments

`model`	Nome/identificador do modelo como o servidor local o reconhece (ex.: o nome exibido pelo LM Studio, ou o `--model` passado ao `llama-server`).
`base_url`	URL base do servidor OpenAI-compatible. Padrao: `"http://localhost:1234/v1"` (porta padrao do LM Studio).
`api`	Qual API o servidor expoe: `"chat_completions"` (padrao – a maioria dos servidores locais: LM Studio, llama.cpp, vLLM) ou `"responses"` (servidores que implementam a API Responses da OpenAI).
`temperature`	Temperatura padrao do chat. `NULL` (padrao): usa o padrao do servidor.
`system_prompt`	Prompt de sistema inicial. `NULL` (padrao).
`...`	Argumentos adicionais passados a `ellmer::chat_openai()` ou `ellmer::chat_openai_compatible()`.

Value

Objeto `Chat` do `ellmer`, pronto para uso em `ac_qual_code()` (via `chat = ac_chat_local(...)`).

Examples

```
## Not run:
# LM Studio rodando localmente, servindo um modelo na porta padrao
chat_local <- ac_chat_local(model = "qwen2.5-14b-instruct")

cb <- ac_qual_codebook(
  name = "tom", instructions = "Classifique o tom.",
  categories = list(
    positivo = list(definition = "Tom propositivo."),
    negativo = list(definition = "Tom crítico.")
  )
)
df <- data.frame(id = "d1", texto = "Proponho cooperacao.")
corpus <- ac_corpus(df, text = texto, docid = id)
coded <- ac_qual_code(corpus, cb, chat = chat_local,
                      checkpoint_dir = "checkpoints/")

## End(Not run)
```

`ac_clean` *Limpar e normalizar texto de um corpus*

Description

`ac_clean()` aplica um conjunto configurável de transformações ao texto de um objeto `ac_corpus`, retornando um novo corpus com o texto modificado. As transformações são aplicadas em ordem lógica e registradas como atributo para auditoria.

Usage

```
ac_clean(
  corpus,
  lower = TRUE,
  remove_punct = TRUE,
  remove_numbers = FALSE,
  remove_url = TRUE,
  remove_email = TRUE,
  remove_symbols = FALSE,
  remove_hashtags = FALSE,
  remove_mentions = FALSE,
  remove_stopwords = NULL,
  remove_accents = FALSE,
  normalize_pt = FALSE,
  protect = NULL,
  extra_stopwords = NULL,
  min_char = NULL,
  custom_replacements = NULL,
```

```
  handle_na = c("preserve", "empty", "remove"),
  strip_whitespace = TRUE,
  verbose = FALSE,
  ...
)
```

Arguments

corpus	Objeto de classe ac_corpus, criado por ac_corpus().
lower	Se TRUE (padrão), converte texto para minúsculas.
remove_punct	Se TRUE (padrão), remove pontuação.
remove_numbers	Se TRUE, remove dígitos. Padrão: FALSE.
remove_url	Se TRUE (padrão), remove URLs (http, https, www).
remove_email	Se TRUE (padrão), remove endereços de e-mail.
remove_symbols	Se TRUE, remove símbolos e emojis (@, #, *, etc.). Padrão: FALSE. Cuidado: hashtags e menções podem ser informativas.
remove_hashtags	Se TRUE, remove hashtags (#termo). Padrão: FALSE.
remove_mentions	Se TRUE, remove menções (\@usuário). Padrão: FALSE.
remove_stopwords	Pode ser: • NULL (padrão): não remove stopwords; • uma string com nome de preset ("pt", "pt-br-extended", "pt-legislativo", "en"); • um vetor character com stopwords customizadas.
remove_accents	Se TRUE, remove acentos (ex: "ação" vira "acao"). Padrão: FALSE.
normalize_pt	Se TRUE, aplica normalizações ortográficas do português brasileiro coloquial: "pra" -> "para", "tá" -> "está", etc. Padrão: FALSE.
protect	Vetor character com termos a preservar exatamente como estão. Útil para siglas (c("PT", "PSDB", "CCJ")). Padrão: NULL.
extra_stopwords	Vetor character com stopwords adicionais a remover antes de qualquer análise, combinado ao preset de remove_stopwords. Use ac_clean_stopwords() para inspecionar e editar o objeto padrão. Padrão: NULL.
min_char	Inteiro. Descarta tokens com menos de min_char caracteres após a limpeza. Padrão: NULL (sem filtro).
custom_replacements	Lista nomeada de substituições livres aplicadas antes das demais transformações. Ex: list("pres\\." = "presidente", "dep\\." = "deputado"). Padrão: NULL.
handle_na	Como tratar valores NA no texto: • "preserve" (padrão): mantém NA como NA; • "empty": converte NA para "".

- `"remove"`: remove documentos com `NA` do corpus.

`strip_whitespace`
: Se `TRUE` (padrão), colapsa espaços consecutivos.

`verbose`
: Se `TRUE`, exibe resumo das operações e estatísticas de remoção por etapa. Padrão: `FALSE`.

`...`
: Ignorado com aviso se houver argumentos não reconhecidos.

Details

Ordem de aplicação das transformações:

1. `handle_na`: tratamento de NAs
2. `custom_replacements`: substituições livres
3. `protect`: proteção de termos com placeholders
4. `remove_url` e `remove_email`
5. `remove_hashtags` e `remove_mentions`
6. `remove_symbols`
7. `lower`
8. `remove_accents`
9. `normalize_pt`
10. `remove_numbers`
11. `remove_punct`
12. `remove_stopwords` + `extra_stopwords`
13. `min_char`: remoção de tokens curtos
14. Restauração dos termos protegidos
15. `strip_whitespace` (sempre)

Value

Um novo objeto `ac_corpus` com a coluna `text` transformada e o atributo `cleaning_steps` registrando as operações aplicadas em ordem. Quando `verbose = TRUE`, também imprime um resumo com tokens removidos por etapa e documentos que ficaram vazios após limpeza.

See Also

`ac_corpus()`, `ac_clean_stopwords()`

Examples

```
df <- data.frame(
  id = c("a", "b", "c"),
  texto = c(
    "O deputado do PT disse: 'Defendo a CCJ!' Veja em https://exemplo.org",
    "Sra. presidente, o Sr. senador apresentou o requerimento n\u00ba 123.",
    "T\u00e1 na hora de votar, pra acabar com isso."
```

```
  )
)
corpus <- ac_corpus(df, text = texto, docid = id)

# Limpeza básica
ac_clean(corpus)

# Limpeza completa com todas as opções
ac_clean(
  corpus,
  remove_stopwords  = "pt-legislativo",
  extra_stopwords   = c("isso", "aquilo", "coisa"),
  protect           = c("PT", "CCJ"),
  normalize_pt      = TRUE,
  custom_replacements = list("pres\\." = "presidente"),
  min_char          = 3L,
  verbose           = TRUE
)
```

---

ac_clean_stopwords *Inspecionar e editar stopwords extras do acR*

Description

`ac_clean_stopwords()` retorna (e opcionalmente modifica) o vetor de stopwords adicionais que o pesquisador pode passar a `ac_clean()` via `extra_stopwords`. Funciona como um ponto de partida editável: o pesquisador inspeciona o vetor, adiciona ou remove termos conforme o corpus, e passa o resultado para `extra_stopwords`.

Usage

```
ac_clean_stopwords(
  add = NULL,
  remove = NULL,
  preset = c("empty", "pt", "pt-br-extended", "pt-legislativo")
)
```

Arguments

`add`	Vetor `character` com termos a adicionar ao vetor padrão.
`remove`	Vetor `character` com termos a remover do vetor padrão.
`preset`	String indicando o ponto de partida: `"empty"` (padrão), `"pt"`, `"pt-br-extended"` ou `"pt-legislativo"`.

Value

Vetor `character` de stopwords pronto para passar a `ac_clean(..., extra_stopwords = ...)`.

See Also

ac_clean()

Examples

```
# Ver o vetor padrão vazio e adicionar termos
sw <- ac_clean_stopwords(add = c("nobre", "ilustre", "respeitavel"))
print(sw)

# Partir do preset legislativo e remover termos que interessam ao corpus
sw <- ac_clean_stopwords(
  preset = "pt-legislativo",
  remove = c("lei", "projeto")   # manter: são relevantes para a análise
)

# Usar na limpeza
# ac_clean(corpus, remove_stopwords = "pt", extra_stopwords = sw)
```

---

ac_cluster_documents *Agrupamento não supervisionado de documentos*

Description

ac_cluster_documents() particiona os documentos de um corpus em grupos "duros" (hard clustering) com base na similaridade de vocabulário. Serve para descobrir tipologias latentes, montar amostras estratificadas de revisao humana ou produzir dendrogramas para relatórios metodologicos.

Não substitui ac_lda(): o LDA é um modelo de pertencimento misto (cada documento vira uma mistura de tópicos), enquanto esta função devolve **uma etiqueta por documento**.

Usage

```
ac_cluster_documents(
  corpus,
  method = c("hclust", "kmeans", "pam"),
  features = c("tfidf", "count"),
  k = NULL,
  distance = c("cosine", "euclidean"),
  min_docs = 15L
)
```

Arguments

corpus	Objeto ac_corpus.
method	Algoritmo. "hclust" (padrao) usa agrupamento hierarquico com ligacao Ward.D2; "kmeans" usa k-means com nstart = 25L; "pam" usa Partitioning Around Medoids (requer cluster em Suggests).

`features` Como representar cada documento. `"tfidf"` (padrao) pesa por TF-IDF; `"count"` usa contagens brutas.

`k` Número de grupos. Se `NULL` (padrao) e `method %in% c("hclust","kmeans")`, tenta escolher automaticamente por silhouette entre 2 e 8; requer `cluster`. Se `cluster` não estiver instalado, `k = 3` e o fallback.

`distance` Métrica de dissimilaridade. `"cosine"` (padrão) é o clássico em textos (Salton & McGill, 1983): compara a *composição* do vocabulário, não o tamanho do documento. Internamente ela é calculada como distância de corda, $\sqrt{2(1-\cos\theta)}$, que é a distância euclidiana entre os vetores normalizados para norma 1 (Legendre & Legendre, 2012). A ordem dos pares é a mesma de $1-\cos\theta$, mas a distância de corda é euclidiana, o que o critério de Ward (`hclust`) e o k-means exigem (Murtagh & Legendre, 2014). Com `"cosine"`, o k-means também roda sobre os vetores normalizados. `"euclidean"` usa os vetores brutos, sensíveis ao tamanho do documento.

`min_docs` Mínimo de documentos para não emitir warning de corpus pequeno. Padrao: `15L`.

Value

Objeto de classe `ac_cluster` com:

`assignments` `tibble` com `doc_id` e `cluster` (integer).

`fit` Objeto bruto do algoritmo (`hclust`, `kmeans` ou `pam`).

`k` Número final de grupos.

`method`, `features`, `distance` Parametros usados.

`silhouette` Silhueta média (NA se `cluster` não instalado).

`dtm` Matriz documento-termo usada (para plotagem).

References

Legendre, P.; Legendre, L. (2012). *Numerical Ecology* (3rd ed.). Elsevier.

Murtagh, F.; Legendre, P. (2014). Ward's hierarchical agglomerative clustering method: Which algorithms implement Ward's criterion? *Journal of Classification*, 31(3), 274-295.

Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20, 53-65.

Salton, G.; McGill, M. J. (1983). *Introduction to Modern Information Retrieval*. McGraw-Hill.

See Also

`ac_lda()`, `ac_plot_cluster()`

Examples

```
# Corpus com dois blocos tematicos
df <- data.frame(
  id    = paste0("d", 1:8),
  texto = c("democracia participacao voto liberdade",
```

```
            "cidadania direitos participacao democracia",
            "voto direitos liberdade cidadania",
            "democracia voto participacao popular",
            "mercado economia eficiência privatizacao",
            "privatizacao mercado livre eficiência",
            "economia crescimento investimento mercado",
            "eficiência mercado economia livre"),
  stringsAsFactors = FALSE
)
corpus <- ac_corpus(df, text = texto, docid = id)
clust  <- ac_cluster_documents(corpus, k = 2)
clust$assignments
```

`ac_collocations` *Identificar colocações com medidas de associação estatística*

Description

`ac_collocations()` identifica pares (ou trincas) de palavras que aparecem juntas com mais frequência do que o esperado pelo acaso: colocações como "reforma tributária", "não recomendo" ou "atendimento ao cliente", usando medidas de associação estatística em vez de frequência bruta de bigramas/trigramas.

A diferença importa: bigramas puramente frequentes tendem a ser dominados por combinações de palavras funcionais comuns ("de o", "que a"), que são frequentes mas não informativas. As medidas de association penalizam esse efeito, destacando combinações que ocorrem juntas mais do que a frequência individual de cada palavra sugeriria: o uso típico é alimentar um dicionário customizado (ver `ac_build_dictionary()`) ou simplesmente explorar expressões compostas específicas de um domínio (jurídico, médico, técnico) antes de uma análise de frequência ou nuvem de palavras.

Usage

```
ac_collocations(
  corpus,
  n = 2L,
  measure = c("pmi", "ll", "chi2"),
  min_count = 5L,
  drop_punct = TRUE
)
```

Arguments

`corpus`	Objeto `ac_corpus`.
`n`	Tamanho do n-grama: `2` (padrão, bigramas) ou `3` (trigramas). Para trigramas, a tabela 2x2 compara a primeira palavra com o **bigrama** formado pelas duas seguintes (ex.: "atendimento" × "ao cliente"), e o PMI usa a mesma decomposição: $\log_2[f(w_1 w_2 w_3)N/(f(w_1)f(w_2 w_3))]$.

`measure` Medida de associação:

- `"pmi"` (padrão): *Pointwise Mutual Information* (Church & Hanks, 1990): quanto mais alto, mais a coocorrência surpreende em relação ao esperado pelo acaso. Tende a favorecer pares raros.
- `"ll"`: log-likelihood ratio, G² (Dunning, 1993): mais confiável que o qui-quadrado e que o PMI para eventos raros, porque não supõe normalidade das contagens (Dunning, 1993; Evert, 2005). Atenção: não é a medida do `quanteda.textstats::textstat_collocations()`, que usa o lambda de Blaheta & Johnson (2001) e seu escore z.
- `"chi2"`: qui-quadrado de associação, mesma lógica de `ac_keyness()` aplicada a pares de palavras adjacentes em vez de grupos de documentos.

`min_count` Frequência mínima do n-grama para ser considerado. Padrão: `5`. Medidas de associação são instáveis para n-gramas muito raros.

`drop_punct` Lógico. Se `TRUE` (padrão), ignora n-gramas que incluem tokens de pontuação pura.

Value

Tibble ordenado por associação decrescente, com colunas:

- `collocation`: o n-grama (palavras separadas por espaço);
- `count`: frequência absoluta do n-grama no corpus;
- `association`: valor da medida escolhida em `measure`.

References

Church, K. W.; Hanks, P. (1990). Word association norms, mutual information, and lexicography. *Computational Linguistics*, 16(1), 22-29.

Dunning, T. (1993). Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics*, 19(1), 61-74.

Evert, S. (2005). *The Statistics of Word Cooccurrences: Word Pairs and Collocations*. Tese (Doutorado), Universität Stuttgart.

Blaheta, D.; Johnson, M. (2001). Unsupervised learning of multi-word verbs. *Proceedings of the ACL Workshop on Collocations*, 54-60.

See Also

`ac_keyness()`, `ac_count()`

Examples

```
# Reviews de um restaurante com expressoes compostas recorrentes
df <- data.frame(
  id = paste0("r", 1:6),
  texto = c(
    "atendimento ao cliente impecavel, super recomendo o lugar",
    "o atendimento ao cliente deixou a desejar dessa vez",
```

```
    "comida boa mas atendimento ao cliente lento",
    "não recomendo, atendimento ao cliente horrivel",
    "ambiente agradável e atendimento ao cliente rápido",
    "preco justo, atendimento ao cliente cordial sempre"
  ),
  stringsAsFactors = FALSE
)
corpus <- ac_corpus(df, text = texto, docid = id)

ac_collocations(corpus, min_count = 2)
```

`ac_cooccurrence` *Calcular co-ocorrências de termos*

Description

`ac_cooccurrence()` calcula pares de termos que co-ocorrem dentro de janelas deslizantes ou dentro do mesmo documento, retornando frequências e medidas de associação (PMI, Dice).

Usage

```
ac_cooccurrence(
  corpus,
  window = 5L,
  unit = c("window", "document"),
  measure = c("count", "pmi", "dice"),
  min_count = 2L,
  ...
)
```

Arguments

`corpus`	Objeto `ac_corpus` ou tibble com colunas `doc_id` e `token` (saída de `ac_tokenize()`).
`window`	Tamanho da janela deslizante (número de tokens de cada lado). Padrão: `5`. Ignorado se `unit = "document"`.
`unit`	Unidade de co-ocorrência: `"window"` (padrão) ou `"document"`.
`measure`	Medidas de associação a calcular. Um ou mais de: `"count"` (frequência conjunta), `"pmi"` (pointwise mutual information), `"dice"` (coeficiente Dice). Padrão: as três.
`min_count`	Frequência mínima de co-ocorrência para incluir o par. Padrão: `2`.
`...`	Ignorado.

Details

PMI (Church & Hanks, 1990) compara a probabilidade conjunta com a esperada sob independência: $\log_2[P(x,y)/(P(x)P(y))]$. Zero indica que o par aparece junto tanto quanto o acaso preveria; valores positivos, mais que o acaso. PMI superestima pares raros (Evert, 2005), por isso use `min_count` para descartá-los.

Dice (Smadja, McKeown & Hatzivassiloglou, 1996) é $2f(x,y)/(f(x)+f(y))$, entre 0 e 1.

As probabilidades usam a mesma unidade da contagem conjunta: com `unit = "window"`, frequências de token e o total de tokens do corpus (aproximação usual de Church & Hanks); com `unit = "document"`, frequência documental (em quantos documentos o termo aparece) e o número de documentos. Pares de um termo com ele mesmo são ignorados.

Value

Tibble com colunas:

- `word1`, `word2`: par de termos (ordenado alfabeticamente);
- `cooc`: frequência de co-ocorrência. Com `unit = "window"`, é o número de vezes em que os dois termos aparecem a até `window` tokens um do outro (cada encontro conta **uma** vez); com `unit = "document"`, é o número de documentos que contêm os dois;
- `pmi` (se solicitado): pointwise mutual information, em bits;
- `dice` (se solicitado): coeficiente Dice.

References

Church, K. W.; Hanks, P. (1990). Word association norms, mutual information, and lexicography. *Computational Linguistics*, 16(1), 22-29.

Evert, S. (2005). *The Statistics of Word Cooccurrences: Word Pairs and Collocations*. Tese (Doutorado), Universität Stuttgart.

Smadja, F.; McKeown, K. R.; Hatzivassiloglou, V. (1996). Translating collocations for bilingual lexicons: A statistical approach. *Computational Linguistics*, 22(1), 1-38.

See Also

`ac_tokenize()`, `ac_plot_cooccurrence()`

Examples

```
# Corpus tematico: coocorrência revela associações conceituais
# (que palavras "andam juntas" no discurso).
df <- data.frame(
  id = paste0("d", 1:6),
  texto = c(
    "democracia participacao voto liberdade cidadania popular",
    "cidadania direitos participacao democracia representacao politica",
    "voto direitos liberdade cidadania soberania popular",
    "mercado economia eficiência privatizacao competicao livre",
    "privatizacao mercado livre eficiência produtividade lucro",
    "economia crescimento investimento mercado capital juros"
```

```
  ),
  stringsAsFactors = FALSE
)

corpus <- ac_corpus(df, text = texto, docid = id) |>
  ac_clean()
tokens <- ac_tokenize(corpus)

# Janela de 3 tokens, sem filtro de frequência mínima
# (em corpora reais use min_count >= 5 para reduzir ruido)
cooc <- ac_cooccurrence(tokens, window = 3L, min_count = 1L)
head(cooc)

# Coocorrência com PMI (associação normalizada por acaso)
cooc_pmi <- ac_cooccurrence(tokens, window = 3L, measure = "pmi",
                            min_count = 1L)
head(cooc_pmi)
```

`ac_corpus` *Construir um corpus para análise de conteúdo*

Description

`ac_corpus()` é a porta de entrada do pipeline do pacote `acR`. Constrói um objeto estruturado de corpus a partir de um `data.frame`, um vetor de caracteres, ou um corpus do `quanteda`, validando a entrada e preservando metadados associados aos documentos.

Usage

```
ac_corpus(data, text = NULL, docid = NULL, meta = NULL, lang = "pt", ...)
```

Arguments

`data`	Entrada do corpus. Pode ser: • um `data.frame` ou `tibble` com pelo menos uma coluna de texto; • um vetor `character` com textos (um documento por elemento); • um objeto `corpus` do pacote `quanteda`.
`text`	Nome da coluna de texto, sem aspas (usa *tidyselect* / *non-standard evaluation*). Apenas relevante quando `data` é um `data.frame`. Se `NULL` (padrão), a função tenta detectar automaticamente uma coluna chamada `text`, `texto`, `doc`, `content` ou `conteúdo`.
`docid`	Nome da coluna de identificador único, sem aspas. Se `NULL` (padrão), gera IDs sequenciais no formato `doc_1`, `doc_2`, ...
`meta`	Colunas de metadados a preservar, via *tidyselect* (ex: `c(partido, data, regiao)` ou `tidyselect::starts_with("var_")`). Se `NULL`, preserva todas as demais colunas do `data.frame`.

lang	Código ISO do idioma do corpus (padrão: "pt"). Utilizado em etapas posteriores (lematização, stopwords, léxicos).
...	Argumentos reservados para extensão futura. No momento, são ignorados com aviso se nomeados de forma desconhecida.

Details

A função realiza cinco validações obrigatórias e falha cedo em caso de problema:

1. data deve ser de um dos tipos suportados;
2. a coluna de texto deve existir (ou ser detectável automaticamente);
3. a coluna de texto não pode ser inteiramente NA ou vazia;
4. doc_id não pode ter valores duplicados;
5. doc_id não pode conter NA.

Documentos com texto vazio ou NA geram aviso (warning) mas são mantidos no corpus com texto "": o usuário decide se filtra depois.

Value

Um objeto de classe ac_corpus (que herda de tbl_df, tbl, data.frame) com as colunas:

- doc_id (character): identificador único do documento.
- text (character): texto do documento.
- Demais colunas de metadados, preservadas de data. O objeto traz também o atributo lang indicando o idioma.

Examples

```
# A partir de um data.frame
df <- data.frame(
  id = c("a", "b", "c"),
  texto = c("Primeiro texto.", "Segundo texto.", "Terceiro."),
  partido = c("PT", "PL", "MDB")
)
corpus <- ac_corpus(df, text = texto, docid = id, meta = partido)
corpus

# A partir de um vetor character (doc_id gerado automaticamente)
ac_corpus(c("Texto um.", "Texto dois."))

# Detecção automática da coluna de texto
df2 <- data.frame(text = c("A", "B"), autor = c("X", "Y"))
ac_corpus(df2)
```

ac_count *Contar frequências de tokens ou n-gramas em um corpus*

Description

`ac_count()` calcula frequências de tokens ou n-gramas a partir de um objeto `ac_corpus()`, usando internamente `ac_tokenize()` seguido de uma agregação com `dplyr::count()`.

É o insumo direto de quase toda análise quantitativa do pacote: `ac_top_terms()` pega o topo, `ac_tf_idf()` recalibra por distintividade, `ac_keyness()` compara grupos, `ac_wordcloud()` visualiza. O argumento by decide se a unidade de análise é o documento ou o grupo agregado.

Pode operar em dois níveis:

- por documento (by = NULL): contagens por doc_id e token;
- por metadados (by = c("partido", ...)): contagens por variáveis de agrupamento e token (agregando vários documentos).

Usage

```
ac_count(corpus, n = 1L, drop_punct = FALSE, by = NULL, sort = TRUE, ...)
```

Arguments

corpus	Objeto de classe `ac_corpus()`.
n	Tamanho do n-grama a ser tokenizado. Encaminhado para `ac_tokenize()`. Veja ?ac_tokenize para detalhes.
drop_punct	Lógico. Se TRUE, remove a pontuação das bordas dos tokens e descarta tokens compostos só de pontuação antes de calcular as frequências (via drop_punct = TRUE em `ac_tokenize()`).
by	Vetor de nomes de colunas de metadados em corpus a serem usados como grupos de agregacao. Se NULL (padrao), as contagens sao feitas por documento (doc_id). Se não for NULL, as colunas indicadas sao usadas como grupos e doc_id não entra no resultado.
sort	Lógico. Se TRUE (padrao), ordena a saida em ordem decrescente de frequência (n).
...	Argumentos adicionais encaminhados para `ac_tokenize()` (por exemplo, keep_empty).

Value

Um `tibble::tibble()`:

- Se by = NULL: colunas doc_id, token, n;
- Se by não nulo: colunas by, token, n.

See Also

`ac_corpus()`, `ac_clean()`, `ac_tokenize()`

Examples

```
df <- data.frame(
  id      = c("d1", "d2", "d3"),
  texto   = c(
    "O deputado do PT falou na CCJ.",
    "O deputado do PL falou novamente.",
    "O senador do PT falou na CCJ."
  ),
  partido = c("PT", "PL", "PT"),
  stringsAsFactors = FALSE
)

corp <- ac_corpus(df, text = texto, docid = id, meta = partido)

# Frequência de palavras (unigramas) por documento
ac_count(corp)

# Frequência de palavras por partido
ac_count(corp, by = "partido")

# Frequência de bigramas por partido, removendo apenas pontuação
ac_count(corp, n = 2, drop_punct = TRUE, by = "partido")
```

`ac_dictionary` *Classificar conteúdo com um dicionário multi-categoria*

Description

`ac_dictionary()` conta, para cada documento de um `ac_corpus`, quantas palavras caem em cada categoria de um dicionário: generalização do método que `ac_sentiment()` já aplica só para polaridade (positivo/negativo). Um dicionário aqui é qualquer lista nomeada de listas de palavras: temas ("economia", "saúde"), enquadramentos ("ameaça", "oportunidade"), categorias de produto, ou qualquer taxonomia definida a priori pelo pesquisador.

Métodos de dicionário são o método mais antigo e mais transparente de análise de conteúdo assistida por computador (Stone et al., 1966, *The General Inquirer*; revisado em Grimmer & Stewart, 2013): toda decisão de classificação é auditável: basta olhar a lista de palavras. O preço é a falta de sensibilidade a contexto: a palavra "não" imediatamente antes de um termo do dicionário não inverte a contagem, e sarcasmo ou negação sofisticada passam batido. Para esses casos, o pipeline de LLM (`ac_qual_code()`) é a alternativa: mais caro, menos transparente, mas sensível a contexto.

Usage

```
ac_dictionary(corpus, dictionary, by = NULL, case_insensitive = TRUE)
```

Arguments

corpus	Objeto ac_corpus.
dictionary	Lista nomeada de vetores de caracteres: uma entrada por categoria, cada uma com as palavras daquela categoria (ver ac_build_dictionary()). Palavras são comparadas a tokens individuais (correspondência exata, não substring).
by	Coluna(s) de agrupamento para agregar além do documento (ex.: "categoria_produto"). Padrão: NULL (uma linha por documento).
case_insensitive	Lógico. Se TRUE (padrão), ignora diferenças de maiúsculas/minúsculas na comparação.

Value

Tibble com uma linha por documento (ou grupo), contendo total_tokens e, para cada categoria do dicionário, duas colunas: <categoria>_n (contagem de palavras daquela categoria) e <categoria>_prop (proporção sobre total_tokens).

References

Stone, P. J.; Dunphy, D. C.; Smith, M. S.; Ogilvie, D. M. (1966). *The General Inquirer: A Computer Approach to Content Analysis*. MIT Press.

Grimmer, J.; Stewart, B. M. (2013). Text as Data: The Promise and Pitfalls of Automatic Content Analysis Methods for Political Texts. *Political Analysis*, 21(3), 267-297.

See Also

ac_build_dictionary(), ac_sentiment(), ac_qual_code()

Examples

```
# Reclamacoes de clientes, categorizadas por tema do problema
df <- data.frame(
  id = paste0("c", 1:4),
  texto = c(
    "a entrega atrasou muito e o rastreio não funcionava",
    "o produto chegou com defeito e quebrado na caixa",
    "cobraram duas vezes no cartao e o reembolso demorou",
    "entrega rápida mas o produto tinha um defeito de fabrica"
  ),
  stringsAsFactors = FALSE
)
corpus <- ac_corpus(df, text = texto, docid = id)

dict <- ac_build_dictionary(
  entrega    = c("entrega", "atrasou", "rastreio", "rápida"),
  produto    = c("defeito", "quebrado", "fabrica"),
  financeiro = c("cobraram", "cartao", "reembolso")
)
```

```
ac_dictionary(corpus, dict)
```

`ac_dictionary_import` *Importar um dicionário de uma tabela em formato longo*

Description

`ac_dictionary_import()` constrói um dicionário (ver `ac_build_dictionary()`) a partir de uma tabela em formato longo: uma linha por par palavra-categoria, o formato em que léxicos costumam ser distribuídos (CSV, planilha). Útil para importar listas de palavras construídas fora do R, ou léxicos de terceiros já traduzidos/adaptados para português (o pacote não distribui nenhum léxico de categorias além do OpLexicon usado em `ac_sentiment()`; veja `vignette("dicionários")` para onde encontrar léxicos de terceiros e como adaptá-los a este formato).

Usage

```
ac_dictionary_import(data, term_col = "termo", category_col = "categoria", ...)
```

Arguments

`data`	Um `data.frame`/tibble em formato longo, ou o caminho de um arquivo `.csv` nesse formato.
`term_col`	Nome da coluna com a palavra. Padrão: `"termo"`.
`category_col`	Nome da coluna com a categoria. Padrão: `"categoria"`.
`...`	Argumentos adicionais passados a `utils::read.csv()` quando `data` é um caminho de arquivo (ex.: `sep`, `fileEncoding`).

Value

Lista nomeada de vetores de caracteres (um dicionário), uma entrada por valor distinto de `category_col`.

See Also

`ac_build_dictionary()`, `ac_dictionary_save()`

Examples

```
lexico <- data.frame(
  termo     = c("bom", "ótimo", "mau", "péssimo"),
  categoria = c("positivo", "positivo", "negativo", "negativo"),
  stringsAsFactors = FALSE
)
ac_dictionary_import(lexico)
```

`ac_dictionary_save` *Salvar ou carregar um dicionário em YAML*

Description

`ac_dictionary_save()` grava um dicionário (ver `ac_build_dictionary()`) em um arquivo YAML legível por humanos, para versionar, compartilhar entre pesquisadores ou editar fora do R. `ac_dictionary_load()` faz o caminho inverso.

Usage

```
ac_dictionary_save(dictionary, path)

ac_dictionary_load(path)
```

Arguments

`dictionary`	Lista nomeada de vetores de caracteres.
`path`	Caminho do arquivo `.yaml`.

Value

`ac_dictionary_save()`: o caminho do arquivo, de forma invisível. `ac_dictionary_load()`: o dicionário (lista nomeada de vetores de caracteres).

See Also

`ac_build_dictionary()`, `ac_dictionary_import()`

Examples

```
dict <- ac_build_dictionary(
  positivo = c("bom", "ótimo"),
  negativo = c("mau", "péssimo")
)
arquivo <- tempfile(fileext = ".yaml")
ac_dictionary_save(dict, arquivo)

dict2 <- ac_dictionary_load(arquivo)
identical(dict, dict2)
```

`ac_dictionary_validate` *Validar um dicionário contra codificação humana*

Description

`ac_dictionary_validate()` compara a classificação por dicionário (saída de `ac_dictionary()`, `ac_emotion()` ou `ac_sentiment()` com categorias binarizadas) contra uma codificação de referência humana, categoria por categoria: a mesma lógica de validação que `ac_qual_reliability()` aplica à classificação por LLM (com `by_variable = TRUE`), agora para o método de dicionário.

Nenhum dicionário deveria ser usado numa análise publicável sem essa etapa (Grimmer & Stewart, 2013): um dicionário construído à mão pode parecer razoável e ainda assim ter baixa precisão ou recall numa categoria específica: só a comparação com uma amostra codificada por humano revela isso.

Usage

```
ac_dictionary_validate(scored, human, cat_col = "categoria", threshold = 0)
```

Arguments

`scored`	Tibble retornado por `ac_dictionary()` ou `ac_emotion()` (com colunas `doc_id` e `<categoria>_n` por categoria).
`human`	Tibble com colunas `doc_id` e uma coluna de categoria (ver `cat_col`): a codificação humana de referência, no mesmo formato aceito por `ac_qual_reliability()` (multilabel como string separada por `"|"`).
`cat_col`	Nome da coluna de categoria em `human`. Padrão: `"categoria"`.
`threshold`	Contagem mínima em `<categoria>_n` para o dicionário considerar a categoria presente no documento. Padrão: `0` (qualquer palavra da categoria presente já conta).

Value

Tibble com uma linha por categoria do dicionário e colunas `categoria`, `n`, `vp`, `fp`, `fn`, `vn`, `concordância`, `precisao`, `recall`, `f1`, `gwet_ac1`, `krippendorff_alfa`: as mesmas colunas de `attr(ac_qual_reliability(...), "by_variable")`. Denominadores indefinidos (ex.: uma categoria que nenhum dos dois lados marcou) geram `NA`, nunca `0` ou `1` artificiais.

References

Grimmer, J.; Stewart, B. M. (2013). Text as Data: The Promise and Pitfalls of Automatic Content Analysis Methods for Political Texts. *Political Analysis*, 21(3), 267-297.

See Also

`ac_dictionary()`, `ac_qual_reliability()`

Examples

```
df <- data.frame(
  id = paste0("d", 1:4),
  texto = c(
    "a entrega atrasou muito",
    "o produto chegou com defeito",
    "cobraram duas vezes no cartao",
    "entrega rápida e produto perfeito"
  ),
  stringsAsFactors = FALSE
)
corpus <- ac_corpus(df, text = texto, docid = id)
dict <- ac_build_dictionary(
  entrega = c("entrega", "atrasou", "rápida"),
  produto = c("defeito", "perfeito")
)
scored <- ac_dictionary(corpus, dict)

# Codificacao humana de referência (simulada para o exemplo)
human <- tibble::tibble(
  doc_id    = paste0("d", 1:4),
  categoria = c("entrega", "produto", NA, "entrega|produto")
)

ac_dictionary_validate(scored, human)
```

`ac_emotion` *Classificar emoções com um léxico de palavra-emoção*

Description

`ac_emotion()` conta, por documento, quantas palavras caem em cada categoria de emoção de um léxico fornecido pelo usuário: indo além da polaridade binária de `ac_sentiment()` para distinguir, por exemplo, se um texto negativo expressa raiva, medo ou tristeza (Mohammad & Turney, 2013, propuseram essa distinção no NRC Emotion Lexicon, hoje o léxico de emoções mais usado em inglês).

O pacote **não distribui** um léxico de emoções em português: o NRC Emotion Lexicon original é em inglês e sua tradução/adaptação para português está sujeita aos termos de licença do léxico de origem, que o usuário deve obter e licenciar por conta própria (o NRC é gratuito para pesquisa, mas não redistribuível livremente dentro de outro pacote). `ac_emotion()` é agnóstica ao léxico: funciona com qualquer tabela palavra→emoção que você tenha: o NRC traduzido, um léxico doméstico, ou uma lista construída à mão para um domínio específico (ex.: emoções em avaliações de atendimento ao cliente).

Internamente, é uma camada sobre `ac_dictionary()`: converte o léxico em formato longo (uma linha por palavra-emoção) para o formato de dicionário multi-categoria e aplica a mesma contagem.

Usage

```
ac_emotion(
  corpus,
  lexicon,
  term_col = "termo",
  emotion_col = "emocao",
  by = NULL
)
```

Arguments

corpus	Objeto ac_corpus.
lexicon	Tibble/data.frame em formato longo com colunas de palavra e emoção (ver term_col/emotion_col). Uma palavra pode aparecer em mais de uma linha (associada a mais de uma emoção).
term_col	Nome da coluna de palavra em lexicon. Padrão: "termo".
emotion_col	Nome da coluna de emoção em lexicon. Padrão: "emocao".
by	Coluna(s) de agrupamento adicional. Padrão: NULL.

Value

Tibble no mesmo formato de ac_dictionary(): total_tokens e, para cada emoção presente em lexicon, colunas <emocao>_n e <emocao>_prop.

References

Mohammad, S. M.; Turney, P. D. (2013). Crowdsourcing a Word-Emotion Association Lexicon. *Computational Intelligence*, 29(3), 436-465.

See Also

ac_dictionary(), ac_sentiment()

Examples

```
# Lexico de emocao mínimo, construido a mao para este exemplo
# (em uso real, substitua por um lexico de emocoes licenciado em PT)
lexico <- data.frame(
  termo  = c("alegria", "feliz", "medo", "assustado", "raiva", "furioso"),
  emocao = c("alegria", "alegria", "medo", "medo", "raiva", "raiva"),
  stringsAsFactors = FALSE
)

df <- data.frame(
  id = paste0("d", 1:3),
  texto = c(
    "fiquei muito feliz com a alegria do momento",
    "fiquei assustado, senti muito medo da situação",
    "fiquei furioso, aquela raiva não passava"
```

```
  ),
  stringsAsFactors = FALSE
)
corpus <- ac_corpus(df, text = texto, docid = id)

ac_emotion(corpus, lexicon = lexico)
```

`ac_estimate_tokens` *Estimar número de tokens de um texto*

Description

`ac_estimate_tokens()` estima o número de tokens de entrada que um texto vai consumir, usando uma heurística de caracteres por token. Util para prever o custo de uma rodada de `ac_qual_code()` ou para detectar, antes de enviar ao LLM, documentos que provavelmente excederao a janela de contexto do modelo.

E uma estimativa deliberadamente barata: não faz nenhuma chamada de rede nem depende do tokenizador exato de um modelo específico (que varia por provedor e as vezes nem e público). Para calibrar a heurística contra um modelo real, use `ac_calibrate_tokens()` com uma amostra pequena de textos.

Usage

```
ac_estimate_tokens(text, chat = NULL, chars_per_token = 3.2)
```

Arguments

`text`	Vetor de caracteres com os textos.
`chat`	Ignorado nesta versao (reservado para uma futura estimativa específica de provedor via `chat$token_count()`). Receber o argumento já na assinatura evita quebrar código que o passar.
`chars_per_token`	Caracteres por token, usado para a estimativa. Padrao `3.2` (razoável para português via BPE; varia por tokenizador e idioma). Calibre com `ac_calibrate_tokens()` se precisar de precisao maior para um modelo específico.

Value

Vetor inteiro com a estimativa de tokens por texto.

Examples

```
ac_estimate_tokens("Este e um texto de exemplo para estimar tokens.")
ac_estimate_tokens(c("texto curto", "um texto um pouco mais longo que o anterior"))
```

ac_export *Exporta resultados de análise de conteúdo em múltiplos formatos*

Description

Exporta data.frames de resultados (corpus codificado, tabelas de frequência, métricas de confiabilidade) para CSV, LaTeX, Excel (.xlsx) ou RDS. Pensada para facilitar a transicao entre o pipeline acR e a escrita acadêmica (tabelas em artigos) ou o compartilhamento de dados replicaveis.

Usage

```
ac_export(
  x,
  path = NULL,
  format = NULL,
  overwrite = TRUE,
  latex_caption = NULL,
  latex_label = NULL,
  latex_digits = 3L,
  excel_sheet = "acR",
  verbose = TRUE
)
```

Arguments

x	data.frame ou objeto ac_irr. Dados a exportar.
path	character. Caminho do arquivo de saida, incluindo extensao (ex.: "resultados.csv", "tabela1.tex", "dados.xlsx"). Se NULL, o formato e inferido pelo argumento format e o arquivo e salvo no diretório de trabalho com nome "acR_export".
format	character. Formato de saida: "csv", "latex", "xlsx", "rds". Se NULL (padrao), inferido pela extensao de path.
overwrite	logical. Se TRUE (padrao), sobrescreve arquivo existente. Se FALSE, lanca erro se o arquivo já existir.
latex_caption	character ou NULL. Legenda da tabela LaTeX.
latex_label	character ou NULL. Label para \\ref{} no LaTeX. Ex.: "tab:resultados".
latex_digits	integer. Casas decimais para colunas numericas na saida LaTeX. Padrao: 3.
excel_sheet	character. Nome da aba no arquivo Excel. Padrao: "acR".
verbose	logical. Se TRUE (padrao), confirma o caminho salvo.

Details

CSV:

Usa utils::write.csv() com row.names = FALSE e encoding UTF-8. Separador: virgula. Adequado para importacao em Stata, SPSS, Python e R.

LaTeX:

Gera código LaTeX via `knitr::kable()` com `format = "latex"`, empacotado em ambiente `table` com `\\centering`, `\\caption` e `\\label`. Adicione `\\usepackage{booktabs}` no preambulo do documento para melhor tipografia.

Excel:

Usa `writexl::write_xlsx()`: sem dependência de Java ou LibreOffice. Cria arquivo `.xlsx` compativel com Excel 2007+, Google Sheets e LibreOffice Calc.

RDS:

Serializa o objeto R completo via `base::saveRDS()`. Preserva tipos, atributos e classes (incluindo objetos `ac_irr`, corpus etc.). Ideal para replicabilidade interna ao projeto.

Value

Invisivel: o caminho do arquivo salvo (`character`).

See Also

`ac_qual_irr()`, `ac_qual_code()`

Examples

```
resultados <- data.frame(
  id_discurso   = c("d1", "d2", "d3"),
  nome_deputado = c("Dep. A", "Dep. B", "Dep. C"),
  categoria     = c("progressista", "conservador", "tecnocratico"),
  confianca     = c(0.94, 0.91, 0.87)
)

# Escrevemos em tempdir() para não poluir o filesystem do usuário
out_csv <- file.path(tempdir(), "resultados.csv")
out_tex <- file.path(tempdir(), "resultados.tex")
out_rds <- file.path(tempdir(), "resultados.rds")

# CSV
ac_export(resultados, out_csv)

# LaTeX (para incluir em artigo)
ac_export(
  resultados,
  out_tex,
  latex_caption = "Classificação do tom dos discursos por LLM",
  latex_label   = "tab:tom_discursos"
)

# RDS (replicabilidade)
ac_export(resultados, out_rds)

# Excel (requer openxlsx ou writexl instalado)
if (requireNamespace("writexl", quietly = TRUE) ||
```

```
    requireNamespace("openxlsx", quietly = TRUE)) {
  out_xlsx <- file.path(tempdir(), "resultados.xlsx")
  ac_export(resultados, out_xlsx, excel_sheet = "Classificação")
}

# Objeto ac_irr (comparar codificacao humana vs. LLM)
gold <- data.frame(
  id_discurso = c("d1", "d2", "d3"),
  categoria   = c("progressista", "conservador", "tecnocratico")
)
predicted <- data.frame(
  id_discurso = c("d1", "d2", "d3"),
  categoria   = c("progressista", "conservador", "progressista")
)
irr_result <- ac_qual_irr(gold, predicted, verbose = FALSE)
ac_export(irr_result, file.path(tempdir(), "confiabilidade.csv"))
```

`ac_fetch_camara` *Busca discursos de deputados federais via API da Camara dos Deputados*

Description

Coleta discursos parlamentares diretamente da API pública da Camara dos Deputados (v2), retornando um `data.frame` padronizado e pronto para uso nas funções do `acR`. A coleta e feita em duas etapas: (1) lista deputados conforme os filtros informados; (2) para cada deputado, busca os discursos no periodo solicitado, com paginacao automática.

Usage

```
ac_fetch_camara(
  data_inicio,
  data_fim,
  legislatura = NULL,
  partido = NULL,
  uf = NULL,
  n_max = 100,
  tipo_discurso = "plenario",
  verbose = TRUE,
  sleep = 0.5
)
```

Arguments

`data_inicio`	`character`. Data de inicio no formato `"YYYY-MM-DD"`.
`data_fim`	`character`. Data de fim no formato `"YYYY-MM-DD"`.

legislatura	integer ou NULL. Número da legislatura (ex.: 57 para 2023-2027). Se NULL, usa o periodo definido por data_inicio e data_fim sem filtrar por legislatura.
partido	character ou NULL. Sigla do partido para filtrar deputados (ex.: "PT", "PL", "MDB"). Aceita vetor de siglas. Se NULL, inclui todos os partidos.
uf	character ou NULL. Sigla da UF para filtrar deputados (ex.: "SP", "MG"). Aceita vetor de UFs. Se NULL, inclui todas.
n_max	integer. Número máximo de discursos a retornar. Padrao: 100. Use Inf para coletar todos (atencao: pode ser lento).
tipo_discurso	character. Tipo de evento parlamentar. Opcoes principais: "plenário" (padrao), "comissao", "todos".
verbose	logical. Se TRUE (padrao), exibe mensagens de progresso.
sleep	numeric. Tempo de espera (em segundos) entre chamadas a API para respeitar o rate limit. Padrao: 0.5.

Details

A API Dados Abertos da Camara (v2) não dispoe de endpoint único para discursos por periodo. O fluxo de coleta e:

1. GET /api/v2/deputados - lista deputados com filtros de partido/UF.
2. GET /api/v2/deputados/{id}/discursos - discursos de cada deputado, com paginacao (max. 100 itens por pagina).

O filtro tipo_discurso atua sobre o campo tipoDiscurso retornado pela API. Valores observados: "DISCURSO", "DISCURSO ENCAMINHADO", "BREVE COMUNICAÇÃO", "PELA ORDEM", "COMUNICAÇÃO PARLAMENTAR".

Value

Um data.frame com as colunas:

id_discurso character. Identificador único do discurso.

id_deputado integer. ID do deputado na API da Camara.

nome_deputado character. Nome civil do parlamentar.

partido character. Sigla do partido na data do discurso.

uf character. UF da bancada do parlamentar.

data Date. Data do discurso.

hora_inicio character. Hora de inicio (HH:MM).

tipo_discurso character. Tipo de fase do evento.

sumário character. Sumário do discurso (quando disponivel).

texto character. Texto integral do discurso (quando disponivel).

uri_discurso character. URI do recurso na API.

References

CAMARA DOS DEPUTADOS. Dados Abertos da Camara dos Deputados - API v2. Brasilia, 2024. Disponivel em: https://dadosabertos.camara.leg.br/swagger/api.html. Acesso em: abr. 2026.

See Also

ac_corpus() para transformar o resultado em corpus.

Examples

```
## Not run:
# Discursos do plenário, marco de 2024
disc <- ac_fetch_camara(
  data_inicio = "2024-03-11",
  data_fim    = "2024-03-15",
  n_max       = 50
)

# Apenas PT e PL
disc_partidos <- ac_fetch_camara(
  data_inicio = "2024-01-01",
  data_fim    = "2024-03-31",
  partido     = c("PT", "PL"),
  n_max       = 100
)

# Todos os tipos de discurso
disc_todos <- ac_fetch_camara(
  data_inicio   = "2024-03-01",
  data_fim      = "2024-03-31",
  uf            = "SP",
  tipo_discurso = "todos",
  n_max         = 100
)

## End(Not run)
```

---

ac_fetch_senado *Busca discursos de senadores federais via senatebR*

Description

Coleta discursos parlamentares do Senado Federal usando o pacote **senatebR** como backend, retornando um `data.frame` no mesmo formato padronizado de ac_fetch_camara(). Isso permite combinar corpora das duas casas legislativas sem atrito.

O periodo pode ser especificado por legislatura (atalho conveniente) ou por datas exatas (controle fino). Se ambos forem fornecidos, as datas prevalecem.

Usage

```
ac_fetch_senado(
  data_inicio = NULL,
  data_fim = NULL,
  legislatura_inicio = NULL,
  legislatura_fim = NULL,
  partido = NULL,
  uf = NULL,
  nome_senador = NULL,
  n_max = 100,
  verbose = TRUE,
  sleep = 0.3
)
```

Arguments

data_inicio	character ou NULL. Data de inicio no formato "YYYY-MM-DD". Se NULL, usa o inicio da legislatura_inicio.
data_fim	character ou NULL. Data de fim no formato "YYYY-MM-DD". Se NULL, usa o fim da legislatura_fim.
legislatura_inicio	integer ou NULL. Número da legislatura de inicio (ex.: 56 para 2019-2023, 57 para 2023-2027). Ignorado se data_inicio for fornecido.
legislatura_fim	integer ou NULL. Número da legislatura de fim. Se NULL, usa o mesmo valor de legislatura_inicio.
partido	character ou NULL. Sigla(s) do partido para filtrar senadores (ex.: "PT", c("PT", "PL")). Filtragem pos-coleta. Se NULL, inclui todos os partidos.
uf	character ou NULL. Sigla(s) da UF para filtrar senadores (ex.: "SP", c("SP", "MG")). Filtragem pos-coleta. Se NULL, inclui todas as UFs.
nome_senador	character ou NULL. Padrao de texto para filtrar pelo nome do senador (busca parcial, case-insensitive). Ex.: "Lula", "Pacheco". Se NULL, inclui todos.
n_max	integer. Número máximo de discursos a retornar. Padrao: 100. Use Inf para coletar todos.
verbose	logical. Se TRUE (padrao), exibe mensagens de progresso.
sleep	numeric. Pausa em segundos entre requisicoes. Padrao: 0.3.

Details

Legislaturas do Senado Federal:

Legislatura	Periodo
55	2015-2019
56	2019-2023
57	2023-2027

Compatibilidade com ac_fetch_camara():
O `data.frame` retornado tem as mesmas colunas de `ac_fetch_camara()`, com a adicao da coluna `casa` (`"senado"`). Isso permite combinar os dois corpora com `rbind()` ou `dplyr::bind_rows()`.

Backend:
Esta função usa **senatebR** (Santos, 2026) como backend para acesso a API do Senado Federal. O **senatebR** deve estar instalado: `install.packages("senatebR")`.

Value

Um `data.frame` com as mesmas colunas de `ac_fetch_camara()`:

`id_discurso character`. Identificador único.

`id_deputado character`. Código do senador na API.

`nome_deputado character`. Nome do senador.

`partido character`. Sigla do partido.

`uf character`. UF da representacao.

`data Date`. Data do discurso.

`hora_inicio character`. Hora de inicio (HH:MM).

`tipo_discurso character`. Tipo/fase do discurso.

`sumário character`. Resumo do discurso.

`texto character`. Texto integral (quando disponivel).

`uri_discurso character`. URL do recurso.

`casa character`. Sempre `"senado"`.

References

SANTOS, V. **senatebR**: Collect Data from the Brazilian Federal Senate Open Data API. CRAN, 2026. Disponivel em: https://github.com/vsntos/senatebR.

See Also

`ac_fetch_camara()`, `ac_corpus()`

Examples

```
## Not run:
# Por legislatura
disc_sen <- ac_fetch_senado(
  legislatura_inicio = 57,
  n_max = 50
)

# Por datas + filtro de partido
disc_pt <- ac_fetch_senado(
  data_inicio = "2024-01-01",
  data_fim    = "2024-06-30",
```

```
  partido     = c("PT", "PL"),
  n_max       = 100
)

# Combinar Camara + Senado
disc_camara <- ac_fetch_camara(
  data_inicio = "2024-03-01",
  data_fim    = "2024-03-31",
  n_max       = 50
)
disc_senado <- ac_fetch_senado(
  data_inicio = "2024-03-01",
  data_fim    = "2024-03-31",
  n_max       = 50
)
corpus_bicameral <- dplyr::bind_rows(disc_camara, disc_senado)

## End(Not run)
```

`ac_import` *Importar arquivos para um corpus acR*

Description

Importa arquivos de texto em diferentes formatos (PDF, Word, Excel, CSV, TXT, JSON, imagens com OCR) e retorna diretamente um objeto `ac_corpus`. Suporta caminhos individuais, vetores de arquivos e globs (`dados/*.pdf`).

É a **via de entrada principal** do pipeline quando você trabalha com arquivos em disco (relatórios, transcrições, PDFs escaneados). Faz o que `ac_corpus()` faria a partir de um `data.frame`, mas cobrindo o passo anterior: extração de texto de formatos heterogêneos com detecção automática pelo sufixo do arquivo. Para PDFs escaneados e imagens, aciona OCR via `tesseract` com português como padrão.

Usage

```
ac_import(path, text_field = NULL, lang_ocr = "por", id_from = "filename", ...)
```

Arguments

`path`	Caminho para um arquivo, vetor de arquivos ou glob (ex: `'dados/*.docx'`). Formatos suportados: `.pdf`, `.doc`, `.docx`, `.xlsx`, `.xls`, `.csv`, `.txt`, `.json`, `.png`, `.jpg`, `.jpeg`, `.tiff`.
`text_field`	Nome da coluna que contem o texto, para arquivos tabulares (`.xlsx`, `.xls`, `.csv`). Obrigatório nesses formatos.
`lang_ocr`	Lingua para OCR em imagens e PDFs escaneados. Padrao: `'por'` (português). Use `tesseract::tesseract_info()` para ver linguas instaladas.

id_from	Como gerar os IDs dos documentos. 'filename' (padrao) usa o nome do arquivo sem extensao. 'rownum' usa números sequenciais. Ou um vetor de strings com os IDs desejados.
...	Argumentos adicionais passados para readtext::readtext().

Details

O ac_import() detecta automaticamente o formato pelo extension e aciona o parser correto:

- **PDF com texto selecionavel**: readtext + pdftools
- **PDF escaneado / imagem**: tesseract (OCR)
- **Word** (.doc, .docx): readtext
- **Excel** (.xlsx, .xls): readtext (requer text_field)
- **CSV**: readtext (requer text_field) (em tabelas, **cada linha vira um documento**, com doc_id <arquivo>.<n>, e as demais colunas viram metadados do corpus)
- **TXT / JSON**: readtext
- **Pasta inteira / glob**: todos os arquivos compativeis de uma vez

A ordem dos documentos no corpus resultante segue a ordem de entrada em path (ou a ordem alfabetica retornada por Sys.glob()), independente de o parser ser OCR ou readtext. IDs duplicados (dois arquivos com o mesmo basename) sao desambiguados automaticamente com um sufixo _2, _3, ... e um aviso e emitido.

Dependências opcionais: readtext e tesseract não sao importados automaticamente – o ac_import() verifica se estao instalados e orienta a instalacao caso necessário.

Value

Um objeto ac_corpus pronto para uso com todas as funções do acR.

References

Benoit, K., et al. (2018). readtext: Import and Handling for Plain and Formatted Text Files. R package. https://CRAN.R-project.org/package=readtext

Ooms, J. (2024). tesseract: Open Source OCR Engine. R package. https://CRAN.R-project.org/package=tesseract

See Also

ac_corpus(), ac_clean(), ac_tokenize()

Examples

```
## Not run:
# PDF com texto selecionavel
corpus <- ac_import('relatório.pdf')

# Pasta inteira de Word
corpus <- ac_import('proposicoes/*.docx')
```

```
# Excel -- indicar a coluna de texto
corpus <- ac_import('respostas.xlsx', text_field = 'resposta')

# PDF escaneado (OCR em português)
corpus <- ac_import('ata_manuscrita.pdf', lang_ocr = 'por')

# Imagem (OCR)
corpus <- ac_import('captura.png')

# Pasta mista (PDF + DOCX + TXT)
corpus <- ac_import('dados/*')

## End(Not run)
```

---

ac_keyness *Calcular estatísticas de keyness entre dois grupos*

Description

ac_keyness() calcula estatísticas de "keyness" para comparar a distribuicao de termos entre dois grupos. O criterio que define os grupos e do pesquisador: partido, periodo, tema, regiao, autor, condicao experimental ou qualquer variável categorica no corpus. A função e inspirada em quanteda.textstats::textstat_keyness() e utiliza tabelas 2x2 por termo.

A entrada típica e uma tabela de frequências gerada por ac_count(), agregada por uma coluna de grupo:

- ac_count(corp, by = "grupo") seguido de
- ac_keyness(freq, group = "grupo", target = "A").

Usage

```
ac_keyness(x, group, target, measure = c("chi2", "ll"), sort = TRUE)
```

Arguments

x	Um data.frame ou tibble::tibble() contendo, no mínimo, as colunas token, n e uma coluna de grupo.
group	Nome da coluna em x que identifica os grupos (string). Essa coluna deve possuir exatamente dois valores distintos.
target	Valor de group que sera considerado o grupo alvo (por exemplo, "Governo"). O outro valor sera tratado como grupo de referência.
measure	Estatística de keyness a ser usada. Pode ser "chi2" (padrao) para qui-quadrado com 1 grau de liberdade, ou "ll" para log-likelihood (G^2).
sort	Lógico. Se TRUE (padrao), ordena a saida por keyness em ordem decrescente (termos mais caracteristicos do grupo alvo no topo).

Details

Para cada termo, a função constroi uma tabela 2x2:

- a = n_target (frequência do termo no grupo alvo)
- b = n_reference (frequência do termo no grupo de referência)
- c = total_target - a
- d = total_reference - b

Em seguida, calcula:

- qui-quadrado com 1 g.l. na opcao measure = "chi2";
- log-likelihood ratio (G^2) na opcao measure = "ll".

Em ambos os casos, a estatística e multiplicada pelo sinal da diferenca de frequências relativas (a / total_target - b / total_reference), de forma que valores positivos indiquem termos mais caracteristicos do grupo alvo e valores negativos termos mais caracteristicos do grupo de referência.

Value

Um tibble::tibble() com uma linha por termo, contendo:

- token
- group (nome da coluna de grupo, repetido para referência)
- target, reference
- n_target, n_reference
- total_target, total_reference
- keyness (estatística assinada, positiva quando o termo e mais frequente no grupo alvo, negativa quando e mais frequente no grupo de referência)
- direction (nome do grupo em que o termo e relativamente mais frequente)

See Also

ac_count(), ac_top_terms(), ac_tf_idf()

Examples

```
# Comparar vocabulário entre dois grupos (aqui, dois temas legislativos).
# O criterio de grupo e do pesquisador: partido, periodo, regiao, tema...
df <- data.frame(
  id    = paste0("d", 1:8),
  texto = c(
    "reforma tributária simplifica sistema impostos empresas",
    "IVA dual substitui PIS COFINS ICMS federal",
    "aliquotas excecoes tributárias setores economicos",
    "reforma tributária unifica impostos indiretos",
    "programa habitacional amplia recursos moradia popular",
    "deficit habitacional afeta familias baixa renda",
    "urbanizacao favelas regularizacao fundiária municipios",
```

```
    "moradia digna direito social constituicao"
  ),
  tema  = rep(c("tributário", "habitacao"), each = 4),
  stringsAsFactors = FALSE
)

corp <- ac_corpus(df, text = texto, docid = id)
freq <- ac_count(corp, by = "tema")

# Termos-chave do grupo "tributário" versus o resto
key <- ac_keyness(freq, group = "tema", target = "tributário")
head(key)
```

---

ac_kwic *Concordância (keyword-in-context, KWIC)*

Description

ac_kwic() localiza todas as ocorrências de um termo (ou padrão) nos documentos de um ac_corpus e devolve cada ocorrência com uma janela de palavras antes e depois: a técnica de concordância usada desde os primórdios da linguística de corpus (Sinclair, 1991) para entender *como* uma palavra é usada, não só *com que frequência*.

É o complemento natural de uma análise de frequência ou keyness: saber que "liberdade" é um termo-chave de um grupo não diz nada sobre como o termo é empregado: a concordância mostra se aparece ao lado de "econômica", "de expressão" ou "individual", revelando o enquadramento (framing) real do termo em cada contexto. Funciona em qualquer corpus com linguagem natural: reviews de produto, letras de música, transcrições de atendimento, posts em redes sociais.

Usage

```
ac_kwic(corpus, pattern, window = 5L, regex = FALSE, case_insensitive = TRUE)
```

Arguments

corpus	Objeto ac_corpus.
pattern	Termo ou expressão regular a localizar (string única).
window	Número de palavras de contexto a mostrar antes e depois de cada ocorrência. Padrão: 5.
regex	Lógico. Se FALSE (padrão), pattern é tratado como uma palavra exata (comparação literal, após normalização de caixa). Se TRUE, pattern é tratado como expressão regular (POSIX) aplicada a cada token.
case_insensitive	Lógico. Se TRUE (padrão), ignora diferenças de maiúsculas/minúsculas na busca.

Value

Tibble com uma linha por ocorrência, colunas:

- `doc_id`: documento onde a ocorrência foi encontrada;
- `position`: posição do token correspondente no documento;
- `before`: até `window` palavras imediatamente antes;
- `keyword`: o token que casou com `pattern`, como aparece no texto;
- `after`: até `window` palavras imediatamente depois.

References

Sinclair, J. (1991). *Corpus, Concordance, Collocation*. Oxford University Press.

See Also

`ac_plot_xray()`, `ac_count()`

Examples

```
# Avaliações de atendimento ao cliente
df <- data.frame(
  id = paste0("a", 1:4),
  texto = c(
    "o atendimento foi rápido e a atendente foi muito educada comigo",
    "fiquei esperando muito tempo e o atendimento foi frio e distante",
    "atendimento excelente, resolveram meu problema na hora",
    "o atendimento automático não entendeu minha solicitacao"
  ),
  stringsAsFactors = FALSE
)
corpus <- ac_corpus(df, text = texto, docid = id)

ac_kwic(corpus, pattern = "atendimento", window = 3)
```

`ac_lda` *Ajustar modelo LDA (Latent Dirichlet Allocation)*

Description

`ac_lda()` ajusta um modelo de tópicos LDA sobre um `ac_corpus`, retornando um objeto com os resultados do modelo e tibbles tidy de termos por tópico e prevalência por documento.

Usage

```
ac_lda(corpus, k = 10L, seed = 42L, method = c("VEM", "Gibbs"), ...)
```

Arguments

corpus	Objeto ac_corpus.
k	Número de tópicos. Padrão: 10.
seed	Semente para reprodutibilidade. Padrão: 42.
method	Método de estimação: "VEM" (padrão) ou "Gibbs".
...	Use control = list(...) para repassar parâmetros de controle a topicmodels::LDA() (ex.: control = list(alpha = 0.1, iter = 2000) com method = "Gibbs"). A seed é incluída automaticamente se control não tiver uma.

Value

Lista de classe ac_lda com:

- model: objeto LDA do pacote topicmodels;
- terms: tibble com beta (probabilidade de cada termo por tópico);
- documents: tibble com gamma (prevalência de cada tópico por documento);
- k: número de tópicos;
- params: parâmetros usados.

References

Blei, D. M.; Ng, A. Y.; Jordan, M. I. (2003). Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3, 993-1022.

Vignette: vignette("lda", package = "acR"): tutorial completo com corpus real e visualizações.

See Also

ac_lda_tune(), ac_plot_lda_topics()

Examples

```
# 1. Corpus sintético com 3 temas nitidos (democracia, economia, saude, educacao)
df <- data.frame(
  id = paste0("d", 1:10),
  texto = c(
    "democracia participacao cidadania direitos politica",
    "mercado economia privatizacao crescimento fiscal",
    "saude hospital medico doenca tratamento",
    "educacao escola professor ensino aprendizagem",
    "democracia eleicao voto politica partido",
    "economia inflacao juros fiscal orcamento",
    "saude sus medico hospital remedio",
    "educacao universidade pesquisa ciência",
    "participacao social cidadania direitos igualdade",
    "mercado trabalho emprego salário industria"
  )
)
corpus <- ac_corpus(df, text = texto, docid = id)
```

```
# 2. Ajustar LDA com k=3 tópicos (o corpus tem ~4 temas; k=3 for a experimentacao)
lda <- ac_lda(corpus, k = 3)
lda  # imprime resumo do modelo
```

`ac_lda_tune` *Ajustar múltiplos modelos LDA para selecionar k*

Description

`ac_lda_tune()` ajusta modelos LDA para diferentes valores de k e calcula a **perplexidade fora da amostra** por validação cruzada, para auxiliar na seleção do número de tópicos.

Por que fora da amostra: a perplexidade calculada nos mesmos documentos usados para ajustar o modelo cai quase sempre à medida que k cresce, porque mais tópicos = mais parâmetros para memorizar o corpus. Blei, Ng & Jordan (2003) e Wallach et al. (2009) avaliam a perplexidade em documentos **não vistos** no ajuste; Zhao et al. (2015) usam validação cruzada em 5 partes, como faz esta função por padrão. A versão de treino continua disponível na coluna `perplexity_train`, só para comparação.

Perplexidade é um critério de ajuste preditivo, não de qualidade substantiva: Chang et al. (2009) mostram que modelos com melhor perplexidade podem produzir tópicos **menos** interpretáveis para humanos. Use a curva para delimitar uma faixa de k e decida dentro dela lendo os tópicos.

Usage

```
ac_lda_tune(
  corpus,
  k_range = 5:20,
  seed = 42L,
  method = c("VEM", "Gibbs"),
  folds = 5L,
  ...
)
```

Arguments

`corpus`	Objeto `ac_corpus`.
`k_range`	Vetor de valores de k a testar. Padrão: `5:20`.
`seed`	Semente (ajuste dos modelos e sorteio das partes da validação cruzada). Padrão: `42`.
`method`	Método de estimação. Padrão: `"VEM"`.
`folds`	Número de partes da validação cruzada. Padrão: `5` (limitado ao número de documentos). Use `1` para pular a validação cruzada e calcular só a perplexidade de treino (não recomendado).
`...`	Use `control = list(...)` para repassar parâmetros de controle a `topicmodels::LDA()` (ex.: `control = list(alpha = 0.1, iter = 2000)` com `method = "Gibbs"`). A `seed` é incluída automaticamente se `control` não tiver uma.

Value

Tibble com colunas k, `perplexity` (média da perplexidade nas partes de teste da validação cruzada), `perplexity_sd` (desvio padrão entre as partes) e `perplexity_train` (perplexidade no corpus inteiro usado no ajuste).

References

Blei, D. M.; Ng, A. Y.; Jordan, M. I. (2003). Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3, 993-1022.

Chang, J.; Gerrish, S.; Wang, C.; Boyd-Graber, J.; Blei, D. M. (2009). Reading tea leaves: How humans interpret topic models. *Advances in Neural Information Processing Systems*, 22, 288-296.

Wallach, H. M.; Murray, I.; Salakhutdinov, R.; Mimno, D. (2009). Evaluation methods for topic models. *Proceedings of the 26th International Conference on Machine Learning*, 1105-1112.

Zhao, W. et al. (2015). A heuristic approach to determine an appropriate number of topics in topic modeling. *BMC Bioinformatics*, 16(Suppl 13), S8.

See Also

`ac_lda()`, `ac_plot_lda_tune()`

Examples

```
# 1. Corpus sintético (mesmos temas de ac_lda(), palavras diferentes)
df <- data.frame(
  id = paste0("d", 1:10),
  texto = c(
    "democracia participacao politica voto eleicao",
    "mercado economia fiscal privatizacao",
    "saude hospital medico doenca sus",
    "educacao escola professor universidade",
    "democracia cidadania direitos igualdade",
    "economia inflacao juros orcamento",
    "saude medico remedio tratamento",
    "educacao pesquisa ciência tecnologia",
    "participacao social democracia cidadania",
    "mercado trabalho emprego salário"
  )
)
corpus <- ac_corpus(df, text = texto, docid = id)

# 2. Testar k de 2 a 5: perplexidade fora da amostra (validacao cruzada)
tune <- ac_lda_tune(corpus, k_range = 2:5)
tune
```

`ac_lexical_diversity` *Calcular diversidade lexical (riqueza de vocabulário)*

Description

`ac_lexical_diversity()` mede quão variado é o vocabulário dos documentos de um `ac_corpus`: uma pergunta recorrente em linguística computacional (textos de falantes nativos vs. aprendizes de uma língua), análise de redações escolares, estudos de estilo autoral, ou comparação de roteiros/transcrições por gênero ou locutor.

Calcula três métricas, cada uma respondendo à mesma pergunta de forma diferente: e cada uma com um ponto fraco diferente:

- **TTR** (*Type-Token Ratio*): proporção simples `tipos / tokens`. Fácil de entender, mas extremamente sensível ao tamanho do texto: textos mais longos quase sempre têm TTR menor, então **não compare TTR entre documentos de tamanhos muito diferentes**.
- **C de Herdan** (Herdan, 1960): `log(tipos) / log(tokens)`. Mais estável que TTR frente a variação de tamanho, mas ainda não é totalmente independente do comprimento do texto.
- **MTLD** (*Measure of Textual Lexical Diversity*; McCarthy & Jarvis, 2010): percorre o texto token a token acumulando um "fator" sempre que a TTR local cai abaixo de 0,72, e usa a média de fatores nos sentidos direto e reverso. É a métrica recomendada quando os documentos comparados têm tamanhos desiguais.

Usage

```
ac_lexical_diversity(corpus, by = NULL, case_sensitive = FALSE)
```

Arguments

`corpus`	Objeto `ac_corpus`.
`by`	Coluna(s) de agrupamento para agregar a diversidade lexical além do documento (ex.: `"autor"`, `"genero"`). Padrão: `NULL` (uma linha por documento).
`case_sensitive`	Lógico. Se `FALSE` (padrão), trata `"Casa"` e `"casa"` como o mesmo tipo lexical.

Value

Tibble com colunas:

- `doc_id` (ou as colunas de `by`, se informado);
- `n_tokens`, `n_types`: contagens de tokens e tipos (palavras únicas);
- `ttr`, `herdan_c`, `mtld`: as três métricas descritas acima.

References

Herdan, G. (1960). *Type-Token Mathematics: A Textbook of Mathematical Linguistics*. Mouton.

McCarthy, P. M.; Jarvis, S. (2010). MTLD, vocd-D, and HD-D: A validation study of sophisticated approaches to lexical diversity assessment. *Behavior Research Methods*, 42(2), 381-392.

See Also

ac_readability()

Examples

```
# Transcricoes de dois narradores com estilos de vocabulário distintos
df <- data.frame(
  id = paste0("t", 1:4),
  texto = c(
    "o gato subiu no muro e desceu depois correu e subiu de novo",
    paste(
      "o felino ascendeu ao parapeito, permaneceu imovel por instantes,",
      "descendeu com cautela e retornou ao ponto de partida original"
    ),
    "ela correu muito rápido e muito rápido e muito rápido ainda",
    paste(
      "a atleta acelerou vertiginosamente, superando marcas anteriores",
      "com uma performance notável e consistente ao longo do percurso"
    )
  ),
  narrador = c("simples", "rebuscado", "simples", "rebuscado"),
  stringsAsFactors = FALSE
)
corpus <- ac_corpus(df, text = texto, docid = id, meta = narrador)

ac_lexical_diversity(corpus)

# Comparar riqueza de vocabulário entre os dois estilos narrativos
ac_lexical_diversity(corpus, by = "narrador")
```

---

ac_palette *Paleta categórica do acR*

Description

Retorna a paleta categórica padrão do pacote: cores compatíveis com contraste WCAG AA e teste de daltonismo (deuteranopia). Deriva da paleta Okabe-Ito, referência para acessibilidade em visualização científica.

Usage

```
ac_palette(n = 8L)
```

Arguments

n	Número de cores a retornar (max 8). Padrão: 8.

Value

Vetor `character` com códigos hex.

See Also

`theme_ac()`

Examples

```
ac_palette()      # todas as 8 cores
ac_palette(3)     # primeiras 3

if (requireNamespace("ggplot2", quietly = TRUE)) {
  ggplot2::ggplot(iris, ggplot2::aes(Sepal.Length, Petal.Length,
                                     color = Species)) +
    ggplot2::geom_point(size = 2, alpha = 0.85) +
    ggplot2::scale_color_manual(values = ac_palette(3)) +
    theme_ac()
}
```

---

ac_plot_ca *Visualizar um biplot de análise de correspondência*

Description

`ac_plot_ca()` plota um biplot com as duas primeiras dimensões da saída de `ac_ca()`: pontos de unidades (documentos/grupos) e de termos no mesmo espaço, com os eixos rotulados pela proporção de inércia explicada.

Usage

```
ac_plot_ca(ca, top_terms = 20L, title = NULL)
```

Arguments

`ca`	Objeto de classe `ac_ca`, saída de `ac_ca()`.
`top_terms`	Número máximo de termos a rotular, pelos de maior distância da origem (mais distintivos). Padrão: `20`.
`title`	Título do gráfico. Padrão: `NULL` (título automático).

Value

Objeto `ggplot`.

See Also

`ac_ca()`

Examples

```
df <- data.frame(
  id = paste0("d", 1:6),
  texto = c(
    "entrega atrasada rastreio sumiu",
    "pedido atrasou muito, rastreio não atualiza",
    "produto com defeito, chegou quebrado",
    "veio quebrado, qualidade péssima",
    "atendimento excelente, resolveram rápido",
    "suporte educado e rápido, resolveu tudo"
  ),
  categoria = c("entrega", "entrega", "produto", "produto",
                "atendimento", "atendimento"),
  stringsAsFactors = FALSE
)
corpus <- ac_corpus(df, text = texto, docid = id, meta = categoria)
ca <- ac_ca(corpus, by = "categoria", min_count = 1)
if (requireNamespace("ggplot2", quietly = TRUE)) {
  ac_plot_ca(ca)
}
```

`ac_plot_cluster` *Visualiza um objeto ac_cluster*

Description

Tres formas complementares de olhar para o mesmo cluster:

- `"dendrogram"` (padrao para `method = "hclust"`): arvore de fusoes com cortes coloridos por grupo.
- `"scatter"`: projecao 2D via PCA sobre a matriz documento-termo, com pontos coloridos por cluster.
- `"heatmap"`: mapa de calor da matriz de dissimilaridade, ordenado pelo dendrograma.

Usage

```
ac_plot_cluster(
  x,
  kind = c("auto", "dendrogram", "scatter", "heatmap"),
  title = NULL,
  palette = NULL
)
```

Arguments

x	Objeto ac_cluster (saida de ac_cluster_documents()).
kind	Tipo de grafico. "auto" (padrao) escolhe conforme o metodo (dendrograma para hclust, scatter para kmeans/pam).
title	Título do grafico.
palette	Vetor de cores para os clusters. Padrao: ac_palette().

Value

Objeto ggplot.

See Also

ac_cluster_documents()

Examples

```
df <- data.frame(
  id = paste0("d", 1:8),
  texto = c("democracia participacao voto",
            "cidadania direitos participacao",
            "voto direitos liberdade",
            "democracia voto participacao",
            "mercado economia eficiência",
            "privatizacao mercado livre",
            "economia crescimento mercado",
            "eficiência mercado livre")
)
corpus <- ac_corpus(df, text = texto, docid = id)
clust  <- ac_cluster_documents(corpus, k = 2)
ac_plot_cluster(clust)
```

---

ac_plot_cooccurrence *Visualizar rede de co-ocorrência de termos*

Description

ac_plot_cooccurrence() gera um gráfico de rede a partir de um tibble de co-ocorrências (saída de ac_cooccurrence()), usando ggplot2 e ggraph.

Usage

```
ac_plot_cooccurrence(
  cooc,
  top_n = 50L,
  weight = c("cooc", "pmi", "dice"),
```

```
  layout = "fr",
  node_color = "#0072B2",
  edge_color = "grey70",
  title = NULL,
  ...
)
```

Arguments

cooc	Tibble com co-ocorrências, saída de ac_cooccurrence().
top_n	Número de pares mais frequentes a exibir. Padrão: 50.
weight	Coluna a usar como peso das arestas: "cooc" (padrão), "pmi" ou "dice".
layout	Layout do grafo. Qualquer layout suportado por ggraph::ggraph(): "fr" (Fruchterman-Reingold, padrão), "kk", "circle", etc.
node_color	Cor dos nós. Padrão: "#0072B2".
edge_color	Cor das arestas. Padrão: "grey70".
title	Título do gráfico. Padrão: NULL.
...	Ignorado.

Value

Objeto ggplot.

See Also

ac_cooccurrence()

Examples

```
# 1. Corpus pequeno de tres documentos
df <- data.frame(
  id = c("d1", "d2", "d3"),
  texto = c(
    "democracia participacao cidadania",
    "participacao politica democracia",
    "cidadania direitos participacao"
  )
)

# 2. Calcular co-ocorrências de termos
corpus <- ac_corpus(df, text = texto, docid = id) |> ac_clean()
cooc <- ac_cooccurrence(ac_tokenize(corpus), min_count = 1)

# 3. Rede visualizada com ggraph (dependência opcional)
if (requireNamespace("ggraph", quietly = TRUE)) {
  ac_plot_cooccurrence(cooc)
}
```

ac_plot_keyness *Plotar estatísticas de keyness*

Description

ac_plot_keyness() cria um grafico de barras com as estatísticas de keyness calculadas por ac_keyness(), destacando os termos mais caracteristicos do grupo alvo e do grupo de referência.

A função usa ggplot2 como base e pode, opcionalmente, aplicar o estilo editorial do pacote ipeaplot.

Usage

```
ac_plot_keyness(
  x,
  n = NULL,
  style = c("default", "ipea"),
  flip = TRUE,
  show_reference = TRUE
)
```

Arguments

x	Um data.frame ou tibble::tibble() contendo, no mínimo, as colunas token, keyness e direction.
n	Número de termos a exibir por direcao. Se NULL (padrao), usa todas as linhas de x. Se informado, seleciona os top n termos com keyness positivo e os top n com keyness negativo. Empates na última posição são desfeitos em ordem alfabética.
style	Estilo grafico. Pode ser "default" (padrao) ou "ipea". Quando "ipea", a função tenta aplicar ipeaplot::theme_ipea().
flip	Lógico. Se TRUE (padrao), usa barras horizontais com ggplot2::coord_flip().
show_reference	Lógico. Se TRUE (padrao), mostra termos caracteristicos do grupo alvo e do grupo de referência. Se FALSE, mostra apenas os termos com keyness positivo.

Value

Um objeto ggplot.

See Also

ac_count(), ac_keyness()

Examples

```
# 1. Corpus fabricado com duas subamostras (Governo vs Oposicao)
df <- data.frame(
  id    = c("d1", "d2", "d3", "d4"),
  texto = c(
    "A A A B",
    "A B",
    "A B B B",
    "B B C"
  ),
  lado  = c("Governo", "Governo", "Oposicao", "Oposicao"),
  stringsAsFactors = FALSE
)

# 2. Calcular termos-chave (sobre-representados no grupo alvo)
corp <- ac_corpus(df, text = texto, docid = id, meta = lado)
freq <- ac_count(corp, by = "lado")
key <- ac_keyness(freq, group = "lado", target = "Governo")

# 3. Grafico dos 5 termos com maior chi-quadrado
ac_plot_keyness(key, n = 5)
```

`ac_plot_lda_topics` *Visualizar top termos por tópico*

Description

`ac_plot_lda_topics()` gera um gráfico de barras com os termos de maior probabilidade (beta) para cada tópico do modelo LDA.

Usage

```
ac_plot_lda_topics(lda_result, top_n = 10L, ncol = NULL, title = NULL, ...)
```

Arguments

`lda_result`	Objeto `ac_lda`, saída de `ac_lda()`.
`top_n`	Número de termos por tópico. Padrão: `10`.
`ncol`	Número de colunas nos facets. Padrão: `NULL` (automático).
`title`	Título. Padrão: `NULL`.
`...`	Ignorado.

Value

Objeto `ggplot`.

See Also

ac_lda()

Examples

```
# Corpus sintético com dois blocos tematicos
df <- data.frame(
  id = paste0("d", 1:8),
  texto = c(
    "democracia participacao voto cidadania",
    "cidadania direitos participacao democracia",
    "voto direitos liberdade cidadania",
    "democracia voto participacao popular",
    "mercado economia eficiência privatizacao",
    "privatizacao mercado livre eficiência",
    "economia crescimento investimento mercado",
    "eficiência mercado economia livre"
  ),
  stringsAsFactors = FALSE
)
corpus <- ac_corpus(df, text = texto, docid = id)
lda    <- ac_lda(corpus, k = 2)

# Grafico com os 5 termos mais probabilisticos por tópico
ac_plot_lda_topics(lda, top_n = 5)
```

---

ac_plot_lda_tune *Visualizar curva de seleção de k (perplexidade)*

Description

ac_plot_lda_tune() gera um gráfico de linha da perplexidade (ou outras métricas) em função do número de tópicos k, auxiliando na escolha do k ideal para o modelo LDA.

Usage

```
ac_plot_lda_tune(tune_result, title = NULL, ...)
```

Arguments

tune_result	Tibble retornado por ac_lda_tune().
title	Título. Padrão: NULL.
...	Ignorado.

Value

Objeto ggplot.

See Also

ac_lda_tune()

Examples

```
# Corpus sintético com dois blocos tematicos
df <- data.frame(
  id = paste0("d", 1:8),
  texto = c(
    "democracia participacao voto cidadania",
    "cidadania direitos participacao democracia",
    "voto direitos liberdade cidadania",
    "democracia voto participacao popular",
    "mercado economia eficiência privatizacao",
    "privatizacao mercado livre eficiência",
    "economia crescimento investimento mercado",
    "eficiência mercado economia livre"
  ),
  stringsAsFactors = FALSE
)
corpus <- ac_corpus(df, text = texto, docid = id)

# Testar k de 2 a 4 e visualizar a curva de perplexidade
# (ponto de inflexao/"cotovelo" sugere um bom k)
tune <- ac_lda_tune(corpus, k_range = 2:4)
ac_plot_lda_tune(tune)
```

---

ac_plot_lexical_diversity

Visualizar diversidade lexical por grupo

Description

ac_plot_lexical_diversity() compara MTLD (ou outra métrica de ac_lexical_diversity()) entre grupos.

Usage

```
ac_plot_lexical_diversity(
  diversity_tbl,
  group_col = NULL,
  metric = c("mtld", "ttr", "herdan_c"),
  type = c("box", "bar"),
  title = NULL
)
```

Arguments

diversity_tbl	Tibble retornado por ac_lexical_diversity(corpus, by = NULL) com uma coluna de grupo adicional já presente, ou por ac_lexical_diversity(corpus, by = ...).
group_col	Nome da coluna de agrupamento no eixo X.
metric	Métrica a visualizar: "mtld" (padrão), "ttr" ou "herdan_c".
type	Tipo de gráfico: "box" (padrão) ou "bar".
title	Título do gráfico. Padrão: NULL.

Value

Objeto ggplot.

See Also

ac_lexical_diversity()

Examples

```
df <- data.frame(
  id = paste0("t", 1:4),
  texto = c(
    "o gato subiu no muro e desceu depois correu e subiu de novo",
    "o felino ascendeu ao parapeito e descendeu com cautela depois",
    "ela correu muito rápido e muito rápido e muito rápido ainda",
    "a atleta acelerou vertiginosamente superando marcas anteriores"
  ),
  narrador = c("simples", "rebuscado", "simples", "rebuscado"),
  stringsAsFactors = FALSE
)
corpus <- ac_corpus(df, text = texto, docid = id, meta = narrador)
div <- ac_lexical_diversity(corpus)
div$narrador <- corpus$narrador
ac_plot_lexical_diversity(div, group_col = "narrador")
```

---

ac_plot_readability *Visualizar legibilidade por grupo*

Description

ac_plot_readability() compara a distribuição do Índice de Flesch (ver ac_readability()) entre grupos, com boxplots: a forma usual de reportar legibilidade comparada na literatura de linguística computacional e comunicação.

Usage

```
ac_plot_readability(
  readability_tbl,
  group_col = NULL,
  type = c("box", "bar"),
  title = NULL
)
```

Arguments

readability_tbl	Tibble retornado por ac_readability(corpus, by = NULL) (uma linha por documento) com uma coluna adicional de grupo já presente (ex.: junte com os metadados do corpus antes de chamar, ou calcule com by e use type = "bar").
group_col	Nome da coluna de agrupamento no eixo X.
type	Tipo de gráfico: "box" (padrão, exige uma linha por documento) ou "bar" (uma barra por grupo; use quando readability_tbl já vier agregado por ac_readability(by = ...)).
title	Título do gráfico. Padrão: NULL.

Value

Objeto ggplot.

See Also

ac_readability()

Examples

```
df <- data.frame(
  id = paste0("r", 1:6),
  texto = c(
    "Bom. Gostei. Rápido.",
    "Fácil de montar e usar no dia a dia.",
    paste(
      "A configuracao inicial demanda familiaridade com conceitos",
      "de rede que extrapolam o público leigo habitual."
    ),
    "Perfeito, superou expectativas.",
    paste(
      "A ausência de documentacao técnica compromete a integracao",
      "com sistemas legados preexistentes na infraestrutura corporativa."
    ),
    "Simples e direto, recomendo."
  ),
  categoria = rep(c("fácil", "difícil"), 3),
  stringsAsFactors = FALSE
)
corpus <- ac_corpus(df, text = texto, docid = id, meta = categoria)
```

```
leg <- ac_readability(corpus)
leg$categoria <- corpus$categoria
ac_plot_readability(leg, group_col = "categoria")
```

`ac_plot_sentiment` *Visualizar sentimento ao longo dos documentos*

Description

`ac_plot_sentiment()` gera visualizações de sentimento: barras por documento, linha temporal, ou distribuição de scores.

Usage

```
ac_plot_sentiment(
  sentiment_tbl,
  type = c("bar", "line", "density"),
  x_col = "doc_id",
  fill_col = "sentiment",
  title = NULL,
  ...
)
```

Arguments

`sentiment_tbl`	Tibble retornado por `ac_sentiment()`.
`type`	Tipo de visualização: `"bar"` (padrão), `"line"`, `"density"`.
`x_col`	Coluna do eixo X. Padrão: `"doc_id"`. Pode ser uma coluna de data para `type = "line"`.
`fill_col`	Coluna para preenchimento/cor. Padrão: `"sentiment"`.
`title`	Título do gráfico. Padrão: `NULL`.
`...`	Ignorado.

Value

Objeto `ggplot`.

See Also

`ac_sentiment()`

Examples

```
# Requer internet na primeira execucao: ac_sentiment() baixa o OpLexicon
# do GitHub e o guarda em cache (tools::R_user_dir("acR", "data")).
# Corpus com quatro documentos de valências distintas
df <- data.frame(
  id = c("a", "b", "c", "d"),
  texto = c(
    "excelente ótimo positivo bom",
    "péssimo terrivel negativo ruim",
    "aprovada proposta reuniao",
    "bom resultado positivo ótimo"
  )
)
corpus <- ac_corpus(df, text = texto, docid = id)

# Calcular scores e visualizar em barras (positivo/neutro/negativo por doc)
sent <- ac_sentiment(corpus)
ac_plot_sentiment(sent)
```

`ac_plot_temporal` *Visualizar uma série temporal de conteúdo*

Description

`ac_plot_temporal()` traça a evolução de uma métrica agregada por período, saída de `ac_temporal()`, com uma linha de tendência suavizada opcional.

Usage

```
ac_plot_temporal(temporal_tbl, smooth = TRUE, title = NULL, ylab = "Valor")
```

Arguments

`temporal_tbl`	Tibble retornado por `ac_temporal()`.
`smooth`	Lógico. Se `TRUE` (padrão), adiciona uma curva de tendência (LOESS) sobre os pontos observados.
`title`	Título do gráfico. Padrão: `NULL`.
`ylab`	Rótulo do eixo Y. Padrão: `"Valor"`.

Value

Objeto `ggplot`.

See Also

`ac_temporal()`

Examples

```
df <- data.frame(
  doc_id = paste0("d", 1:6),
  data   = as.Date(c("2024-01-05", "2024-01-20", "2024-02-02",
                     "2024-02-15", "2024-03-01", "2024-03-10")),
  score  = c(-2, -1, 0, 1, 2, 1)
)
temporal <- ac_temporal(df, date_col = "data", value_col = "score")
ac_plot_temporal(temporal, ylab = "Sentimento médio")
```

`ac_plot_tf_idf` *Plotar termos mais caracteristicos por tf-idf*

Description

`ac_plot_tf_idf()` cria um grafico de barras com os termos mais caracteristicos a partir de uma tabela com a coluna `tf_idf`, tipicamente gerada por `ac_tf_idf()`.

A função usa ggplot2 como base e pode, opcionalmente, aplicar o estilo editorial do pacote `ipeaplot`.

Usage

```
ac_plot_tf_idf(
  x,
  by = NULL,
  n = NULL,
  style = c("default", "ipea"),
  flip = TRUE
)
```

Arguments

`x`	Um `data.frame` ou `tibble::tibble()` contendo, no mínimo, as colunas `token` e `tf_idf`.
`by`	Vetor de nomes de colunas em `x` a serem usados como grupos de facetas. Se `NULL` (padrao), produz um único grafico. Se não for `NULL`, cria facetas por combinacao das colunas informadas.
`n`	Número de termos a exibir. Se `NULL` (padrao), usa todas as linhas de `x`. Se informado, seleciona os top `n` termos por `tf_idf` no geral ou em cada grupo definido por `by`. Empates na última posição são desfeitos em ordem alfabética.
`style`	Estilo grafico. Pode ser `"default"` (padrao) ou `"ipea"`. Quando `"ipea"`, a função tenta aplicar `ipeaplot::theme_ipea()`.
`flip`	Lógico. Se `TRUE` (padrao), usa barras horizontais com `ggplot2::coord_flip()`.

Value

Um objeto `ggplot`.

See Also

ac_count(), ac_tf_idf(), ac_top_terms()

Examples

```
# 1. Corpus de 3 discursos com metadado de partido
df <- data.frame(
  id      = c("d1", "d2", "d3"),
  texto   = c(
    "O deputado do PT falou na CCJ.",
    "O deputado do PL falou novamente.",
    "O senador do PT falou na CCJ."
  ),
  partido = c("PT", "PL", "PT"),
  stringsAsFactors = FALSE
)

# 2. Pipeline padrao: corpus -> frequência -> tf-idf
corp  <- ac_corpus(df, text = texto, docid = id, meta = partido)
freq  <- ac_count(corp)
tfidf <- ac_tf_idf(freq)

# 3. Grafico dos 10 termos com maior tf-idf (mais distintivos)
ac_plot_tf_idf(tfidf, n = 10)

freq_by <- ac_count(corp, by = "partido")
tfidf_by <- ac_tf_idf(freq_by, by = "partido")
ac_plot_tf_idf(tfidf_by, by = "partido", n = 5)
```

---

ac_plot_top_terms *Plotar termos mais frequentes*

Description

ac_plot_top_terms() cria um grafico de barras com os termos mais frequentes a partir de uma tabela de frequências, tipicamente gerada por ac_count() ou filtrada por ac_top_terms().

A função usa ggplot2 como base e pode, opcionalmente, aplicar o estilo editorial do pacote ipeaplot.

Usage

```
ac_plot_top_terms(
  x,
  by = NULL,
  n = NULL,
  style = c("default", "ipea"),
  flip = TRUE
)
```

Arguments

x	Um data.frame ou tibble::tibble() contendo, no mínimo, as colunas token e n.
by	Vetor de nomes de colunas em x a serem usados como grupos de facetas. Se NULL (padrao), produz um único grafico. Se não for NULL, cria facetas por combinacao das colunas informadas.
n	Número de termos a exibir. Se NULL (padrao), usa todas as linhas de x. Se informado, seleciona os top n termos no geral ou em cada grupo definido por by.
style	Estilo grafico. Pode ser "default" (padrao) ou "ipea". Quando "ipea", a função tenta aplicar ipeaplot::theme_ipea().
flip	Lógico. Se TRUE (padrao), usa barras horizontais com ggplot2::coord_flip().

Value

Um objeto ggplot.

See Also

ac_count(), ac_top_terms()

Examples

```
# 1. Corpus curto com metadado de partido
df <- data.frame(
  id      = c("d1", "d2", "d3"),
  texto   = c(
    "O deputado do PT falou na CCJ.",
    "O deputado do PL falou novamente.",
    "O senador do PT falou na CCJ."
  ),
  partido = c("PT", "PL", "PT"),
  stringsAsFactors = FALSE
)
corp <- ac_corpus(df, text = texto, docid = id, meta = partido)

# 2. Top 10 termos do corpus inteiro
freq <- ac_count(corp)
ac_plot_top_terms(freq, n = 10)

# 3. Top 5 termos POR partido (facets)
freq_by <- ac_count(corp, by = "partido")
ac_plot_top_terms(freq_by, by = "partido", n = 5)
```

`ac_plot_wordcloud_comparative` *Nuvem de palavras comparativa entre grupos*

Description

`ac_plot_wordcloud_comparative()` gera nuvens de palavras comparativas entre N grupos de documentos, dispostas em facets lado a lado. Usa TF-IDF (calculado tratando cada grupo como um "documento") para identificar os termos mais distintivos de cada grupo.

Aceita 2, 3, 4+ grupos: cada grupo vira uma faceta. Para dois grupos a leitura fica naturalmente lado a lado; para mais, o layout se organiza em uma linha (ou grade, se muitos grupos).

Usage

```
ac_plot_wordcloud_comparative(
  corpus,
  group,
  max_words = 50L,
  colors = NULL,
  title = NULL,
  seed = 42L,
  backend = c("auto", "ggwordcloud", "ggplot"),
  ...
)
```

Arguments

`corpus`	Objeto `ac_corpus` com coluna de metadado de grupo.
`group`	Coluna de agrupamento (nome sem aspas ou string). Deve ter pelo menos 2 valores únicos.
`max_words`	Número máximo de palavras por grupo. Padrão: `50`.
`colors`	Vetor de cores (uma por grupo, na ordem alfabética dos grupos). Padrão: as primeiras N cores de `ac_palette()` (Okabe-Ito).
`title`	Título do gráfico. Padrão: `NULL`.
`seed`	Semente para o posicionamento aleatório dos termos. Padrao `42L` (garante layout reproduzivel entre chamadas).
`backend`	Motor de renderizacao: `"auto"` (padrao, prefere `ggwordcloud` com facets), `"ggwordcloud"` ou `"ggplot"` (facets com `geom_text` + jitter reproduzivel).
`...`	Ignorado.

Value

Objeto `ggplot`.

See Also

ac_tf_idf()

Examples

```
# Corpus dividido em dois grupos com vocabulário contrastante
df <- data.frame(
  id     = paste0("d", 1:6),
  texto  = c(
    "democracia participacao popular voto",
    "direitos cidadania liberdade democracia",
    "participacao popular igualdade direitos",
    "mercado economia privatizacao eficiência",
    "privatizacao mercado livre eficiência",
    "economia crescimento mercado investimento"
  ),
  grupo = c("A","A","A","B","B","B")
)
corpus <- ac_corpus(df, text = texto, docid = id)

# Nuvem comparativa: termos distintivos de cada grupo
ac_plot_wordcloud_comparative(corpus, group = grupo)
```

---

ac_plot_wordscores *Visualizar escores Wordscores por documento*

Description

ac_plot_wordscores() plota os escores estimados por ac_wordscores() num eixo único, destacando quais documentos foram usados como referência.

Usage

```
ac_plot_wordscores(wordscores_tbl, title = NULL)
```

Arguments

wordscores_tbl	Tibble retornado por ac_wordscores().
title	Título do grafico. Padrao: NULL (título automático).

Value

Objeto ggplot.

See Also

ac_wordscores()

Examples

```
df <- data.frame(
  id = paste0("r", 1:4),
  texto = c(
    "excelente, recomendo muito",
    "bom, atende bem",
    "ruim, não gostei",
    "péssimo, decepcao"
  ),
  stringsAsFactors = FALSE
)
corpus <- ac_corpus(df, text = texto, docid = id)
notas <- c(r1 = 5, r3 = 2, r4 = 1)
out <- ac_wordscores(corpus, notas)
if (requireNamespace("ggplot2", quietly = TRUE)) {
  ac_plot_wordscores(out)
}
```

`ac_plot_xray` *Gráfico X-ray: dispersão lexical de termos no corpus*

Description

`ac_plot_xray()` exibe a posição de ocorrência de um ou mais termos ao longo do texto de cada documento, como marcações verticais numa linha horizontal. Útil para visualizar padrões de uso ao longo de discursos, capítulos ou documentos longos.

Usage

```
ac_plot_xray(
  corpus,
  terms,
  ignore_case = TRUE,
  colors = NULL,
  title = NULL,
  ...
)
```

Arguments

`corpus`	Objeto `ac_corpus`.
`terms`	Vetor de termos a rastrear (após limpeza e tokenização).
`ignore_case`	Se `TRUE` (padrão), ignora diferenças de capitalização.
`colors`	Vetor de cores para os termos. Se `NULL`, usa paleta padrão.
`title`	Título do gráfico. Padrão: `NULL`.
`...`	Ignorado.

Value

Objeto ggplot.

See Also

ac_corpus()

Examples

```
# Dois documentos com repeticao de termos-alvo em posicoes diferentes
df <- data.frame(
  id = c("d1", "d2"),
  texto = c(
    "democracia liberdade igualdade democracia direitos democracia",
    "mercado liberdade privatizacao mercado eficiência mercado"
  )
)
corpus <- ac_corpus(df, text = texto, docid = id)

# X-ray: pontos marcam a posicao relativa de cada termo dentro do texto
ac_plot_xray(corpus, terms = c("democracia", "mercado", "liberdade"))
```

---

ac_plot_zipf *Visualizar a Lei de Zipf (frequência × posto)*

Description

ac_plot_zipf() produz o gráfico diagnóstico clássico de Zipf (1949): frequência de cada palavra contra sua posição no ranking de frequência (1º, 2º, 3º... termo mais comum), em escala log-log. Em praticamente qualquer corpus de linguagem natural: avaliações de produto, letras de música, posts de redes sociais, discursos: essa relação é aproximadamente linear com inclinação próxima de -1: a palavra mais frequente aparece cerca de duas vezes mais que a segunda, três vezes mais que a terceira, e assim por diante.

É útil como **diagnóstico de qualidade do corpus** antes de qualquer análise mais sofisticada: desvios fortes da reta esperada podem indicar problemas de tokenização (pontuação não removida dominando o ranking), um corpus pequeno demais, ou texto fortemente repetitivo (boilerplate, templates).

Usage

```
ac_plot_zipf(freq_tbl, by = NULL, title = NULL)
```

Arguments

freq_tbl	Tibble com colunas token e n, tipicamente a saída de ac_count(). Se vier com uma coluna de grupo (de ac_count(corp, by = ...)), use by para facetar.
by	Nome da coluna de grupo em freq_tbl para facetar o gráfico (um painel por grupo). Padrão: NULL (corpus inteiro).
title	Título do gráfico. Padrão: NULL.

Value

Objeto ggplot.

References

Zipf, G. K. (1949). *Human Behavior and the Principle of Least Effort*. Addison-Wesley.

See Also

ac_count(), ac_top_terms()

Examples

```
# Avaliações de produtos de duas categorias
df <- data.frame(
  id = paste0("r", 1:6),
  texto = c(
    "bom produto bom preco bom atendimento",
    "produto bom mas entrega lenta",
    "preco justo produto funcional",
    "péssimo atendimento produto com defeito",
    "excelente produto rápido e eficiente",
    "produto ok nada de especial"
  ),
  categoria = rep(c("eletronico", "casa"), 3),
  stringsAsFactors = FALSE
)
corpus <- ac_corpus(df, text = texto, docid = id)
freq <- ac_count(corpus, drop_punct = TRUE, by = NULL)

ac_plot_zipf(freq)
```

---

ac_qual_code *Classificar textos com LLM usando um codebook*

Description

ac_qual_code() classifica os textos de um ac_corpus de acordo com um ac_codebook, usando um modelo de linguagem via ellmer. Retorna um tibble com a classificação, grau de certeza (via self-consistency) e raciocinio da LLM para cada documento.

E o **motor de classificação** do pipeline qualitativo do acR. Assume que o codebook já foi construido com ac_qual_codebook() e (idealmente) testado numa amostra piloto. A saida e sempre validada com ac_qual_reliability() contra uma amostra codificada por humano – nenhuma análise categorial publicável dispensa essa etapa.

Tres parametros determinam qualidade e custo:

- k_consistency: número de rodadas de *self-consistency* (Wang et al., 2023). O mesmo texto e classificado k vezes com pequena variacao de temperatura, e a categoria final e a moda. O confidence_score sai dessa concordância – 1,0 significa que todas as k rodadas concordaram. Padrao k = 3 custa 3x mais tokens que uma rodada única, e e a configuracao mínima para reportar confianca de forma defensavel.
- reasoning: pede raciocinio estruturado. Melhora a qualidade em casos ambiguos mas praticamente dobra o custo por documento. Use "short" como padrao e "detailed" so quando planeja auditar decisoes individuais.
- live: tira a classificação do modo "caixa-preta" mostrando cada documento em tempo real. Numa rodada de 500 discursos, você percebe nos primeiros 10 se o codebook esta funcionando – antes de gastar todo o orcamento de tokens.

Usage

```
ac_qual_code(
  corpus,
  codebook,
  model = "anthropic/claude-sonnet-4-5",
  chat = NULL,
  confidence = c("total", "by_variable", "both", "none"),
  k_consistency = 3L,
  temperature = 0.3,
  reasoning = TRUE,
  reasoning_length = c("short", "medium", "detailed"),
  main_temperature = 0,
  reasoning_extras = FALSE,
  live = c("off", "terminal", "progress", "shiny"),
  max_retries = 2L,
  checkpoint_dir = NULL,
  ...
)
```

Arguments

corpus	Objeto ac_corpus.
codebook	Objeto ac_codebook, saida de ac_qual_codebook().

`model` Modelo LLM a usar. Aceita string no formato `"provedor/modelo"` (ex: `"anthropic/claude-sonnet-4`
`"openai/gpt-4.1"`), objeto `Chat` do pacote `ellmer` pre-configurado, ou função fabrica `function(temperature, system_prompt) Chat` para controle total sobre a construcao do chat a cada rodada. Quando `chat` e fornecido, `model` e ignorado.

`chat` Objeto `Chat` do pacote `ellmer` (ex: `chat_google_gemini()`, `chat_openai()`, `chat_ollama()`). Quando fornecido, tem prioridade sobre `model`. Permite usar qualquer provedor suportado pelo `ellmer`.

`confidence` Como calcular certeza:

- `"total"` (padrao): uma coluna `confidence_score` com a proporção de rodadas de self-consistency que concordaram com a categoria vencedora (moda);
- `"by_variable"`: uma coluna `<categoria>_confidence` por categoria do codebook, com a proporção de rodadas que escolheram *aquela* categoria especificamente (não so a vencedora) – util para montar uma matriz de confianca por tema/categoria;
- `"both"`: as colunas por categoria + `confidence_score` (a da vencedora);
- `"none"`: não calcula certeza (mais rápido, menor custo).

`k_consistency` Número de rodadas para self-consistency. Padrao: `3`. Ignorado se `confidence = "none"`.

`temperature` Temperatura das rodadas de consistency. Padrao: `0.3`.

`reasoning` Lógico. Se `TRUE` (padrao), inclui coluna `raciocinio` com justificativa da classificação.

`reasoning_length`
Tamanho do raciocinio: `"short"` (1 frase, padrao), `"medium"` (3 frases), `"detailed"` (paragrafo).

`main_temperature`
Temperatura da rodada principal (a que gera `categoria`, `categoria_label` e `raciocinio` no resultado final). Padrao `0` (deterministica). Separado de `temperature` (usado so nas rodadas extras de self-consistency) porque os dois cumprem papeis diferentes: a rodada principal deve ser reprodutível, as extras devem variar para medir concordância.

`reasoning_extras`
Lógico. Se `TRUE`, pede raciocinio também nas rodadas extras de self-consistency (mais caro, não usado no resultado final). Padrao `FALSE`.

`live` Visualizacao em tempo real da classificação:

- `"off"` (padrao): sem live view;
- `"terminal"`: barra de progresso com doc atual, categoria, confianca e inicio do raciocinio a cada iteracao;
- `"progress"`: barra de progresso enxuta (doc atual e status), sem ecoar categoria/raciocinio – util para rodadas grandes onde o terminal ficaria poluido;
- `"shiny"`: abre janela Shiny em background com tabela atualizando conforme documentos sao classificados (requer `shiny` e `callr`).

`max_retries` Número máximo de novas tentativas por rodada quando a chamada ao LLM falha (erro de rede, timeout, resposta invalida). Padrao 2. Espera exponencial entre tentativas (até 30s). Quando o erro e `"limite_taxa"` (HTTP 429), a espera segue a dica do provedor ("try again in 12s"), até 90s: em planos gratuitos com `k_consistency` alto, aumente `max_retries`. Quando o erro e classificado como `"estouro_contexto"`, não ha nova tentativa (repetir não resolve). Se mais da metade dos últimos 5+ documentos falhar, a execucao e abortada com sugestao de usar `checkpoint_dir`.

`checkpoint_dir` Diretório para salvar o resultado de cada documento assim que classificado. Se já existir um checkpoint valido para um `doc_id` (mesmos `codebook`, `model`, `k_consistency` e `temperature`), o documento e pulado e o resultado salvo e reutilizado – permite retomar uma rodada interrompida sem reclassificar tudo. `NULL` (padrao): sem checkpoint.

`...` Argumentos adicionais passados a `ellmer::chat()`. Permite uso de APIs OpenAI-compatible self-hosted via `base_url`.

Value

Tibble com colunas:

- `doc_id`: identificador do documento;
- Metadados originais do corpus;
- `categoria`: slug interno da categoria (chave da lista `categories` no codebook). Em `multilabel = TRUE`, string pipe-separada sem espaco (`"técnica|politica"`).
- `categoria_label`: rotulo humano correspondente, vindo do campo `label` de cada categoria (se ausente, repete o slug). Em multilabel, unidos por `" | "` (espaco dos dois lados).
- `confidence_score`: grau de certeza da categoria vencedora (0-1). Presente quando `confidence` e `"total"` ou `"both"`.
- `confidence_level`: `"alta"` (`>= 0,80`), `"média"` (`>= 0,61`) ou `"baixa"` – faixas **convencionadas pelo pacote** para resumir `confidence_score`. Não vêm da literatura de confiabilidade: as faixas de Landis & Koch (1977) são para kappa, não para proporção de rodadas concordantes. A ideia de usar a concordância entre amostras como sinal de incerteza vem de Wang et al. (2023) e Xiong et al. (2024), que mostram que ela se correlaciona com acerto, mas não propõem cortes universais. Com `k_consistency = 3`, os valores possíveis são só 1/3, 2/3 e 1. Acompanha `confidence_score`.
- `<categoria>_confidence`: uma coluna por categoria do codebook (ex.: `positivo_confidence`, `negativo_confidence`), com a proporção de rodadas de self-consistency que escolheram aquela categoria. Presentes quando `confidence` e `"by_variable"` ou `"both"`.
- `raciocinio`: justificativa da classificação (se `reasoning = TRUE`).
- `status`: resultado da rodada principal: `"ok"`, `"resposta_invalida"` (JSON não parseavel), `"estouro_contexto"`, `"limite_taxa"` (limite de requisições ou de tokens por minuto/dia do provedor, HTTP 429), `"timeout"` ou `"erro_servidor"`.
- `n_tentativas`: número de tentativas até obter `status = "ok"` (ou até esgotar `max_retries`).
- `tokens_entrada`, `tokens_saida`: tokens de entrada/saida da rodada principal, via `chat$get_tokens()` (`NA` se o provedor não expoe).

- segundos: tempo total da rodada principal, incluindo novas tentativas.
- tokens_por_segundo: tokens_saida / segundos.

O resultado carrega também um atributo run_info (ver attributes()) com metadados da execucao: versoes de acR e ellmer, modelo, base_url, parametros efetivos aplicados ao provedor, temperaturas pedidas, k_consistency, hash do codebook e timestamps de inicio/fim.

References

Wang, X. et al. (2023). Self-Consistency Improves Chain of Thought Reasoning in Language Models. *International Conference on Learning Representations (ICLR)*.

Xiong, M. et al. (2024). Can LLMs Express Their Uncertainty? An Empirical Evaluation of Confidence Elicitation in LLMs. *International Conference on Learning Representations (ICLR)*.

Landis, J. R.; Koch, G. G. (1977). The Measurement of Observer Agreement for Categorical Data. *Biometrics*, 33(1), 159-174.

Gilardi, F.; Alizadeh, M.; Kubli, M. (2023). ChatGPT Outperforms Crowd Workers for Text-Annotation Tasks. *PNAS*, 120(30).

Examples

```
## Not run:
cb <- ac_qual_codebook(
  name         = "tom",
  instructions = "Classifique o tom do discurso.",
  categories   = list(
    positivo = list(definition = "Tom propositivo e colaborativo."),
    negativo = list(definition = "Tom crítico e confrontacional.")
  )
)

df <- data.frame(
  id    = c("d1", "d2"),
  texto = c("Proponho cooperacao.", "Este governo e um fracasso.")
)
corpus <- ac_corpus(df, text = texto, docid = id)

# Usando string de modelo (comportamento padrao)
coded <- ac_qual_code(corpus, cb, model = "anthropic/claude-sonnet-4-5")

# Usando objeto Chat do ellmer (recomendado para controle fino)
chat_obj <- ellmer::chat_google_gemini(model = "gemini-2.5-flash", echo = "none")
coded <- ac_qual_code(corpus, cb, chat = chat_obj)

# Groq (inferência rápida, plano gratuito)
chat_groq <- ellmer::chat_groq(model = "openai/gpt-oss-120b", echo = "none")
coded <- ac_qual_code(corpus, cb, chat = chat_groq)

# Ollama (modelos locais, sem envio de dados externos)
chat_local <- ellmer::chat_ollama(model = "llama3.2", echo = "none")
coded <- ac_qual_code(corpus, cb, chat = chat_local)
```

```
# confidence = "both": além de confidence_score (categoria vencedora),
# ganha uma coluna <categoria>_confidence por categoria do codebook --
# util para montar uma matriz de confianca por tema entre documentos
# ou comparar modelos na mesma tarefa.
coded_both <- ac_qual_code(corpus, cb, model = "anthropic/claude-sonnet-4-5",
                           confidence = "both")
coded_both[, c("doc_id", "categoria", "confidence_score",
               "positivo_confidence", "negativo_confidence")]

## End(Not run)
```

---

ac_qual_codebook *Criar um codebook para análise de conteúdo qualitativa*

Description

`ac_qual_codebook()` cria um livro de códigos estruturado para classificação de textos via LLM. É o **instrumento central** da análise de conteúdo assistida por IA: nenhum resultado publicável de codificação automática dispensa um codebook explícito, versionável e passível de revisão por pares (Krippendorff, 2018).

Um codebook do `acR` operacionaliza cinco elementos por categoria: definição (o que é), exemplos positivos (o que é, concretamente), exemplos negativos (o que não é, para desambiguar categorias vizinhas), referências (ancorar a categoria em literatura publicada) e peso (indicação relativa de prioridade no prompt). Um codebook bem construído é a diferença entre a LLM adivinhar (com prior próprio, não replicável) e a LLM **aplicar** uma operacionalização reproduzível (Gilardi et al., 2023).

Suporta três modos de construção:

- `"manual"` (padrão): o pesquisador fornece definições, exemplos e referências diretamente.
- `"induced"`: a LLM induz categorias automaticamente a partir de uma amostra do corpus, sugerindo nomes, definições e exemplos. Útil como ponto de partida quando você não tem um esquema teórico a priori.
- `"literature"`: a LLM busca definições na literatura acadêmica, gerando um banco estruturado com trecho original, tradução, autor, ano, revista e link. O pesquisador revisa e aprova interativamente.

Todo codebook mantém um `history` das modificações: quem alterou, quando e o quê. Essencial para auditoria metodológica quando o instrumento evolui entre a versão piloto e a versão final publicada.

Usage

```
ac_qual_codebook(
  name,
  instructions,
  categories = list(),
```

```
  corpus = NULL,
  n_categories = 5L,
  mode = c("manual", "induced", "literature"),
  multilabel = FALSE,
  lang = "pt",
  chat = NULL,
  model = "anthropic/claude-sonnet-4-5",
  journals = "default",
  n_refs = 5L,
  check_overlap = FALSE,
  ...
)
```

Arguments

`name`	Nome identificador do codebook (string).
`instructions`	Instrução geral para a LLM.
`categories`	Lista nomeada de categorias. Cada elemento pode conter: • `definition`: definição operacional da categoria (obrigatório). • `label`: rótulo humano para exibição em relatórios e tabelas (opcional). Se ausente, o `acR` usa o próprio nome da categoria (chave da lista). Exemplo: `label = "Gestão fiscal-administrativa geral"` para uma categoria de nome interno `"gestao_fiscal_administrativa_geral"`. A saída de `ac_qual_code()` ganha uma coluna `categoria_label` traduzida. • `examples_pos`: vetor de exemplos positivos (recomendado). • `examples_neg`: vetor de exemplos negativos (recomendado). • `references`: vetor de referências bibliográficas (opcional). • `weight`: número entre 0 e 1 indicando a importância relativa da categoria para a LLM (padrão: `1`). Categorias raras ou difíceis podem receber peso maior para instrução extra.
`corpus`	Objeto `ac_corpus`. Obrigatório no modo `"induced"`.
`n_categories`	Inteiro. Número de categorias a induzir. Padrão: `5L`.
`mode`	`"manual"` (padrão), `"induced"` ou `"literature"`.
`multilabel`	Lógico. Se `TRUE`, um documento pode pertencer a mais de uma categoria. Padrão: `FALSE`.
`lang`	Idioma do corpus: `"pt"` (padrão) ou `"en"`.
`chat`	Objeto `Chat` do pacote `ellmer`. Tem prioridade sobre `model`.
`model`	Modelo LLM. Padrão: `"anthropic/claude-sonnet-4-5"`.
`journals`	Periódicos para busca de literatura.
`n_refs`	Número de referências por categoria. Padrão: `5`.
`check_overlap`	Se `TRUE`, verifica sobreposição semântica entre definições e avisa o pesquisador. Requer `chat` ou `model`. Padrão: `FALSE`.
`...`	Ignorado.

Value

Objeto de classe ac_codebook.

References

Gilardi, F., Alizadeh, M., & Kubli, M. (2023). ChatGPT outperforms crowd workers for text-annotation tasks. *PNAS*, 120(30).

Krippendorff, K. (2018). *Content Analysis: An Introduction to Its Methodology* (4th ed.). SAGE.

Sampaio, R. C., & Lycarião, D. (2021). *Análise de conteúdo categorial: manual de aplicação*. Brasília: ENAP.

Examples

```
# Codebook manual com duas categorias, cada uma com exemplos positivos
# (o que E) e negativos (o que NÃO e, para desambiguar categorias vizinhas)
cb <- ac_qual_codebook(
  name         = "tom_discurso",
  instructions = "Classifique o tom geral do discurso.",
  categories   = list(
    positivo = list(
      definition   = "Discurso com tom propositivo e colaborativo.",
      examples_pos = c("Proponho que trabalhemos juntos nesta agenda."),
      examples_neg = c("Este governo e um desastre completo."),
      weight       = 1  # peso relativo no prompt; use >1 para priorizar
    ),
    negativo = list(
      definition   = "Discurso com tom crítico ou confrontacional.",
      examples_pos = c("Esta proposta vai arruinar o pais."),
      examples_neg = c("Apresento esta emenda para melhorar o texto."),
      weight       = 1
    )
  )
)
cb  # imprime resumo do codebook
```

---

ac_qual_codebook_add *Adicionar categoria a um codebook existente*

Description

ac_qual_codebook_add() adiciona uma ou mais categorias a um ac_codebook já criado, sem precisar recriar o objeto do zero. Útil para refinamento iterativo do codebook durante a análise.

Usage

```
ac_qual_codebook_add(codebook, ...)
```

Arguments

codebook	Objeto ac_codebook.
...	Categorias a adicionar, nomeadas. Cada elemento deve ser uma lista com definition e, opcionalmente, examples_pos, examples_neg, weight e references.

Value

Objeto ac_codebook atualizado.

See Also

ac_qual_codebook(), ac_qual_codebook_remove()

Examples

```
# Codebook inicial com 2 categorias
cb <- ac_qual_codebook(
  name         = "tom",
  instructions = "Classifique o tom.",
  categories   = list(
    positivo = list(definition = "Tom propositivo."),
    negativo = list(definition = "Tom crítico.")
  )
)

# Adicionar uma terceira categoria (nome do argumento vira nome da categoria)
cb <- ac_qual_codebook_add(cb,
  neutro = list(
    definition   = "Tom neutro, sem posicionamento claro.",
    examples_pos = c("O projeto foi apresentado na sessão de hoje.")
  )
)
names(cb$categories)  # "positivo" "negativo" "neutro"
```

---

ac_qual_codebook_history

Exibir histórico de modificações de um codebook

Description

ac_qual_codebook_history() retorna e imprime o histórico de ações registradas em um ac_codebook (adições, remoções, merges, traduções etc.).

Usage

```
ac_qual_codebook_history(codebook, n = Inf)
```

Arguments

codebook	Objeto ac_codebook.
n	Número máximo de entradas a exibir. Padrão: Inf (todas).

Value

Tibble com colunas timestamp, action e detail (invisível).

Examples

```
# Criar codebook e aplicar duas modificacoes
cb <- ac_qual_codebook(
  name         = "sentimento",
  instructions = "Classifique o sentimento.",
  categories   = list(
    positivo = list(definition = "Sentimento positivo."),
    negativo = list(definition = "Sentimento negativo.")
  )
)
cb <- ac_qual_codebook_add(cb,
  neutro = list(definition = "Sem valência clara.")
)
cb <- ac_qual_codebook_remove(cb, "neutro")

# Ver histórico completo de acoes registradas
ac_qual_codebook_history(cb)
```

---

ac_qual_codebook_hybrid

Enriquecer codebook com literatura via LLM (modo híbrido)

Description

ac_qual_codebook_hybrid() re-ancora as definições de um ac_codebook existente em referências bibliográficas buscadas via LLM, combinando definições manuais com fundamento teórico induzido da literatura.

Usage

```
ac_qual_codebook_hybrid(
  codebook,
  chat = NULL,
  model = "anthropic/claude-sonnet-4-5",
  concepts = NULL,
  journals = "default",
  n_refs = 3L,
  lang = "pt"
)
```

Arguments

codebook	Objeto ac_codebook.
chat	Objeto Chat do pacote ellmer. Tem prioridade sobre model.
model	Modelo LLM. Padrão: "anthropic/claude-sonnet-4-5".
concepts	Lista nomeada com conceitos por categoria (opcional).
journals	Periódicos para busca. Padrão: "default".
n_refs	Número de referências por categoria. Padrão: 3L.
lang	Idioma: "pt" (padrão) ou "en".

Value

Objeto ac_codebook com definições atualizadas e literatura anexada.

Examples

```
## Not run:
# Requer conexao com a internet + credenciais da LLM (ANTHROPIC_API_KEY).

# 1. Codebook manual como ponto de partida
cb <- ac_qual_codebook(
  name         = "populismo",
  instructions = "Identifique tom populista no discurso.",
  categories   = list(
    populista     = list(definition = "Apela ao povo contra a elite."),
    nao_populista = list(definition = "Discurso técnico ou institucional.")
  )
)

# 2. Enriquecer com literatura: busca 3 refs por categoria e reescreve
#    as definicoes com base na literatura recuperada
cb_ancorado <- ac_qual_codebook_hybrid(
  codebook = cb,
  n_refs   = 3L,
  lang     = "pt"
)

# 3. Inspecionar as referências anexadas a cada categoria
cb_ancorado$categories$populista$references

## End(Not run)
```

---

ac_qual_codebook_merge

Fundir dois codebooks em um

Description

ac_qual_codebook_merge() combina as categorias de dois objetos ac_codebook em um único codebook, com controle de conflitos de nomes.

Usage

```
ac_qual_codebook_merge(
  cb1,
  cb2,
  name = NULL,
  on_conflict = c("error", "keep_first", "keep_second", "rename_second"),
  instructions = NULL
)
```

Arguments

cb1	Objeto ac_codebook (base).
cb2	Objeto ac_codebook (a fundir).
name	Nome do codebook resultante. Padrão: "cb1_cb2".
on_conflict	Estratégia em caso de categorias com o mesmo nome: "error" (padrão), "keep_first", "keep_second" ou "rename_second".
instructions	Instrução geral do novo codebook. Se NULL, usa a de cb1.

Value

Objeto ac_codebook fundido.

Examples

```
# Dois codebooks pequenos que cobrem dimensoes distintas
cb_tom <- ac_qual_codebook(
  name         = "tom",
  instructions = "Classifique o tom.",
  categories   = list(
    positivo = list(definition = "Tom positivo."),
    negativo = list(definition = "Tom negativo.")
  )
)
cb_estilo <- ac_qual_codebook(
  name         = "estilo",
  instructions = "Classifique o estilo retorico.",
  categories   = list(
    pathos = list(definition = "Apelo emocional."),
    logos  = list(definition = "Apelo racional.")
  )
)

# Fundir em um único codebook com 4 categorias
cb <- ac_qual_codebook_merge(cb_tom, cb_estilo, name = "tom_estilo")
```

```
names(cb$categories)  # "positivo" "negativo" "pathos" "logos"
```

---

ac_qual_codebook_remove *Remover categoria de um codebook existente*

Description

ac_qual_codebook_remove() remove uma ou mais categorias de um ac_codebook existente.

Usage

```
ac_qual_codebook_remove(codebook, categories)
```

Arguments

codebook	Objeto ac_codebook.
categories	Vetor character com os nomes das categorias a remover.

Value

Objeto ac_codebook atualizado.

See Also

ac_qual_codebook(), ac_qual_codebook_add()

Examples

```
# Codebook com 3 categorias
cb <- ac_qual_codebook(
  name         = "tom",
  instructions = "Classifique o tom.",
  categories   = list(
    positivo = list(definition = "Tom propositivo."),
    negativo = list(definition = "Tom crítico."),
    neutro   = list(definition = "Tom neutro.")
  )
)

# Remover a categoria "neutro" (aceita também um vetor de nomes)
cb <- ac_qual_codebook_remove(cb, "neutro")
names(cb$categories)  # "positivo" "negativo"
```

ac_qual_codebook_translate *Traduzir codebook para outro idioma via LLM*

Description

ac_qual_codebook_translate() traduz as instruções, definições e exemplos de um ac_codebook para o idioma alvo usando uma LLM, preservando a estrutura e os metadados do objeto.

Usage

```
ac_qual_codebook_translate(
  codebook,
  to = c("en", "pt"),
  chat = NULL,
  model = "anthropic/claude-sonnet-4-5",
  translate_examples = TRUE
)
```

Arguments

codebook	Objeto ac_codebook.
to	Idioma alvo: "en" (padrão) ou "pt".
chat	Objeto Chat do pacote ellmer. Tem prioridade sobre model.
model	Modelo LLM. Padrão: "anthropic/claude-sonnet-4-5".
translate_examples	Se TRUE (padrão), traduz também os exemplos.

Value

Objeto ac_codebook traduzido.

Examples

```
## Not run:
# Requer credenciais da LLM (ANTHROPIC_API_KEY ou GROQ_API_KEY).

cb_pt <- ac_qual_codebook(
  name         = "polaridade",
  instructions = "Classifique a polaridade do texto.",
  categories   = list(
    favor  = list(definition = "Apoia a proposta.",
                  examples_pos = "Sou totalmente a favor desta reforma."),
    contra = list(definition = "Opoe-se a proposta.",
                  examples_pos = "Esta proposta e um retrocesso.")
  )
)
```

```
# Traduzir para ingles preservando estrutura e exemplos
cb_en <- ac_qual_codebook_translate(cb_pt, to = "en")
cb_en$lang  # "en"
cb_en$categories$favor$definition

## End(Not run)
```

`ac_qual_export_for_review` *Exportar amostra para revisão humana em Excel*

Description

`ac_qual_export_for_review()` exporta uma amostra de documentos classificados para um arquivo Excel, com colunas para o codificador humano preencher.

Usage

```
ac_qual_export_for_review(
  sample,
  path = "validacao_humana.xlsx",
  corpus = NULL,
  ...
)
```

Arguments

`sample`	Tibble, saída de `ac_qual_sample()`.
`path`	Caminho do arquivo `.xlsx`. Padrão: `"validacao_humana.xlsx"`.
`corpus`	Objeto `ac_corpus` original (opcional). Se fornecido, inclui o texto completo de cada documento na planilha.
`...`	Ignorado.

Value

Invisível: caminho do arquivo gerado.

Examples

```
if (requireNamespace("openxlsx", quietly = TRUE)) {
  # Amostra de documentos já classificados pela LLM
  coded <- tibble::tibble(
    doc_id           = paste0("doc_", 1:5),
    categoria        = c("favor", "contra", "favor", "contra", "favor"),
    confidence_score = c(0.6, 0.9, 0.8, 0.7, 0.55)
  )
```

```
  amostra <- ac_qual_sample(coded, n = 3, strategy = "uncertainty")

  # Exportar para revisao humana em arquivo temporário
  arquivo <- tempfile(fileext = ".xlsx")
  ac_qual_export_for_review(amostra, path = arquivo)
  file.exists(arquivo)
}
```

`ac_qual_import_human` *Importar classificação humana de Excel*

Description

`ac_qual_import_human()` importa um arquivo Excel preenchido por um codificador humano, retornando um tibble compatível com `ac_qual_reliability()`.

Usage

```
ac_qual_import_human(
  path,
  cat_col = "categoria_humano",
  id_col = "doc_id",
  ...
)
```

Arguments

`path`	Caminho do arquivo `.xlsx`.
`cat_col`	Nome da coluna com a classificação humana. Padrão: `"categoria_humano"`.
`id_col`	Nome da coluna de identificador. Padrão: `"doc_id"`.
`...`	Ignorado.

Value

Tibble com colunas `doc_id` e `categoria`.

Examples

```
if (requireNamespace("openxlsx", quietly = TRUE)) {
  # Simular uma planilha já preenchida pelo codificador humano
  arquivo <- tempfile(fileext = ".xlsx")
  openxlsx::write.xlsx(
    data.frame(
      doc_id          = paste0("doc_", 1:3),
      categoria_humano = c("favor", "contra", "favor")
    ),
    arquivo
```

```
  )

  # Importar de volta para o R para uso com ac_qual_reliability()
  humano <- ac_qual_import_human(arquivo)
  humano
}
```

`ac_qual_irr` *Calcular métricas de confiabilidade inter-anotador*

Description

Compara as classificações de dois ou mais anotadores (humanos ou LLMs) e retorna métricas padronizadas de concordância. Suporta Cohen's Kappa (dois anotadores), Fleiss' Kappa (multi-anotador), Krippendorff's Alpha e percentual de concordância simples.

Complementa `ac_qual_reliability()`: aquela compara **um par** LLM × humano com bootstrap; esta é o motor genérico para **qualquer par ou painel** de anotadores. Use-a diretamente quando você tem duas rodadas humanas para calibrar entre codificadores antes de acender a LLM, ou quando quer comparar LLMs entre si sobre a mesma amostra.

A entrada são dois data.frames (anotador 1 e anotador 2), cada um com uma coluna de identificador e uma de categoria, alinhados pelo identificador.

Usage

```
ac_qual_irr(
  gold,
  predicted,
  method = "all",
  id_col = "id_discurso",
  cat_col = "categoria",
  weight = "unweighted",
  conf_level = 0.95,
  verbose = TRUE,
  bootstrap = 1000L
)
```

Arguments

`gold`	`data.frame`. Anotacoes de referência (anotador humano ou gold standard). Deve conter colunas `id_discurso` (ou `id`) e `categoria`.
`predicted`	`data.frame`. Anotacoes a comparar (ex.: saida do LLM via `ac_qual_code()`). Mesmas colunas exigidas.
`method`	`character`. Métrica(s) a calcular. Opcoes: `"all"` (padrao), `"cohen_kappa"`, `"fleiss_kappa"`, `"krippendorff"`, `"percent_agreement"`. Aceita vetor de múltiplas opcoes.

id_col	character. Nome da coluna de identificador nos data.frames. Padrao: "id_discurso".
cat_col	character. Nome da coluna de categoria nos data.frames. Padrao: "categoria".
weight	character. Tipo de ponderação para Cohen's Kappa: "unweighted" (padrão), "linear" ou "squared" (Cohen, 1968). Os pesos só fazem sentido para categorias **ordinais**: a ordem usada é a dos níveis do fator em gold[[cat_col]] (se for fator) ou a ordem alfabética (caso contrário). Para categorias nominais, use "unweighted".
conf_level	numeric. Nivel de confianca para intervalos (0-1). Padrao: 0.95.
verbose	logical. Se TRUE (padrao), imprime resumo formatado.
bootstrap	integer. Número de réplicas *bootstrap* (reamostragem de documentos) para os intervalos de confiança de Kappa, Fleiss e Alpha. Padrão: 1000. Use 0 para não calcular IC.

Details

Interpretação do Kappa (Landis & Koch, 1977):

Kappa	Concordância
< 0.00	Pobre
0.00 - 0.20	Leve
0.21 - 0.40	Razoável
0.41 - 0.60	Moderada
0.61 - 0.80	Substancial
0.81 - 1.00	Quase perfeita

Os próprios autores chamam essas faixas de arbitrárias (Landis & Koch, 1977, p. 165): são rótulos de referência, não testes.

Fleiss' Kappa:
Generalização do kappa para vários anotadores (Fleiss, 1971). Com dois anotadores, como aqui, ela supõe uma distribuição marginal comum aos dois e por isso equivale ao *pi* de Scott, não ao kappa de Cohen; os valores ficam próximos, mas não idênticos.

Krippendorff's Alpha:
Métrica mais geral: funciona com qualquer número de anotadores e lida com dados faltantes (Krippendorff, 2018). Esta função calcula a versão **nominal**. A interpretação segue Krippendorff (2018, sec. 12.4), não Landis & Koch: alfa >= 0,800 confiável; 0,667-0,800 só para conclusões provisórias; < 0,667 insuficiente.

Percentual de concordância:
Não corrige por acaso. Rótulos segundo Neuendorf (2017): >= 0,90 alta; >= 0,80 aceitável na maioria dos casos; >= 0,70 só em estudos exploratórios; abaixo disso, insuficiente.

Intervalos de confiança:
Calculados por *bootstrap* percentil, reamostrando documentos com reposição (Efron & Tibshirani, 1993). Uma versão anterior usava kappa / z como erro-padrão, mas o z do irr é calculado **sob a hipótese nula** (kappa = 0); esse erro-padrão não serve para IC em torno do kappa estimado (Fleiss, Cohen & Everitt, 1969) e gerava limites fora de [-1, 1].

Value

Um objeto de classe `ac_irr` (lista) com os elementos:

`metrics` `data.frame` com colunas `metric`, `estimate`, `ci_lower`, `ci_upper`, `interpretation`.

`confusion` `table`. Matriz de confusao entre os anotadores.

`n_docs` `integer`. Número de documentos comparados.

`n_annotators` `integer`. Número de anotadores.

`categories` `character`. Categorias encontradas.

`method` `character`. Métrica(s) calculadas.

References

KRIPPENDORFF, K. **Content Analysis: An Introduction to Its Methodology**. 4. ed. Thousand Oaks: SAGE, 2018.

LANDIS, J. R.; KOCH, G. G. The measurement of observer agreement for categorical data. **Biometrics**, v. 33, n. 1, p. 159-174, 1977.

COHEN, J. Weighted kappa: nominal scale agreement with provision for scaled disagreement or partial credit. **Psychological Bulletin**, v. 70, n. 4, p. 213-220, 1968.

EFRON, B.; TIBSHIRANI, R. J. **An Introduction to the Bootstrap**. New York: Chapman & Hall, 1993.

FLEISS, J. L. Measuring nominal scale agreement among many raters. **Psychological Bulletin**, v. 76, n. 5, p. 378-382, 1971.

FLEISS, J. L.; COHEN, J.; EVERITT, B. S. Large sample standard errors of kappa and weighted kappa. **Psychological Bulletin**, v. 72, n. 5, p. 323-327, 1969.

NEUENDORF, K. A. **The Content Analysis Guidebook**. 2. ed. Thousand Oaks: SAGE, 2017.

See Also

`ac_qual_code()`, `ac_qual_sample()`

Examples

```
# Comparar LLM vs. anotador humano
humano <- data.frame(
  id_discurso = c("d1", "d2", "d3", "d4", "d5"),
  categoria   = c("progressista", "conservador", "tecnocratico",
                  "progressista", "conservador")
)

llm <- data.frame(
  id_discurso = c("d1", "d2", "d3", "d4", "d5"),
  categoria   = c("progressista", "conservador", "progressista",
                  "progressista", "conservador")
)

resultado <- ac_qual_irr(gold = humano, predicted = llm)
print(resultado)
```

```
# So Cohen's Kappa
ac_qual_irr(humano, llm, method = "cohen_kappa")
```

`ac_qual_list_models` *Listar modelos LLM disponíveis para análise de conteúdo*

Description

`ac_qual_list_models()` retorna um tibble com os modelos LLM disponíveis para uso com `ac_qual_code()`, incluindo informações de custo, janela de contexto e compatibilidade com análise de conteúdo qualitativa em Ciências Sociais.

Dois modos de operação:

- `live = FALSE` (padrão): usa banco interno curado, funciona offline.
- `live = TRUE`: consulta a API do provedor via `ellmer::models_*()` para obter a lista mais atualizada. Requer chave de API configurada.

Usage

```
ac_qual_list_models(
  provider = "all",
  filter = NULL,
  sort_by = c("cost", "name", "context"),
  live = FALSE,
  ...
)
```

Arguments

`provider`	Provedor(es) a listar. Pode ser `"all"` (padrão) ou um ou mais de: `"anthropic"`, `"openai"`, `"google"`, `"groq"`, `"deepseek"`, `"mistral"`, `"ollama"`.
`filter`	String para filtrar modelos por nome ou ID (ex: `"claude"`, `"gpt-4"`). Padrão: `NULL` (sem filtro).
`sort_by`	Como ordenar os resultados: `"cost"` (padrão, menor custo primeiro), `"name"`, `"context"` (maior janela de contexto primeiro).
`live`	Lógico. Se `TRUE`, consulta a API do provedor ao vivo via `ellmer::models_*()`. Requer chave de API. Padrão: `FALSE`.
`...`	Ignorado.

Value

Tibble com colunas:

- `provider`: nome do provedor;
- `model_id`: identificador do modelo para uso em `ac_qual_code()`;
- `name`: nome legível;
- `context_k`: janela de contexto em milhares de tokens;
- `cost_input`: custo por 1M tokens de entrada (USD), `NA` se gratuito/local;
- `cost_output`: custo por 1M tokens de saída (USD);
- `tier`: categoria (`"frontier"`, `"balanced"`, `"fast"`, `"free"`, `"local"`);
- `pt_support`: suporte estimado ao português (`"alto"`, `"médio"`, `"baixo"`);
- `acr_string`: string pronta para uso em `model = ...` no `acR`.

See Also

`ac_qual_recommend_model()`, `ac_qual_code()`

Examples

```
# Listar todos os modelos do banco interno
ac_qual_list_models()

# Só modelos Anthropic
ac_qual_list_models(provider = "anthropic")

# Modelos baratos com suporte a PT
ac_qual_list_models(sort_by = "cost") |>
  dplyr::filter(pt_support == "alto", cost_input < 1)
```

`ac_qual_load_codebook` *Carregar codebook de arquivo YAML*

Description

Le um arquivo YAML gerado por `ac_qual_save_codebook()` e reconstroi o objeto `ac_codebook` na sessão atual. Uso típico: retomar uma análise iniciada em outra sessão ou por outro pesquisador.

Usage

```
ac_qual_load_codebook(path, ...)
```

Arguments

`path`	Caminho do arquivo `.yaml`.
`...`	Ignorado.

Value

Objeto ac_codebook.

Examples

```
# Preparar um codebook e salvar em arquivo temporário
cb <- ac_qual_codebook(
  name         = "polaridade",
  instructions = "Classifique a polaridade do texto.",
  categories   = list(
    favor   = list(definition = "Apoia a proposta."),
    contra  = list(definition = "Opoe-se a proposta.")
  )
)
arquivo <- tempfile(fileext = ".yaml")
ac_qual_save_codebook(cb, path = arquivo)

# Recarregar em outra sessão
cb_novo <- ac_qual_load_codebook(arquivo)
names(cb_novo$categories)  # "favor" "contra"
```

---

ac_qual_recommend_model

Recomendar modelo LLM para análise de conteúdo qualitativa

Description

ac_qual_recommend_model() sugere o(s) modelo(s) mais adequado(s) para uma tarefa específica de análise de conteúdo qualitativa, considerando custo, desempenho em português e tipo de tarefa.

As recomendações são baseadas em benchmarks de classificação de texto em Ciências Sociais (Gilardi et al., 2023; Törnberg, 2023; Alizadeh et al., 2023) e na experiência prática com corpora em português brasileiro.

Usage

```
ac_qual_recommend_model(
  task = c("coding", "literature", "both"),
  budget = c("medium", "low", "high", "free"),
  lang = "pt",
  local = FALSE,
  n = 3L,
  ...
)
```

Arguments

`task`	Tipo de tarefa: • `"coding"` (padrão): classificação de textos com codebook existente; • `"literature"`: geração de definições e busca de referências; • `"both"`: ambas as tarefas.
`budget`	Orçamento disponível: • `"free"`: apenas modelos gratuitos ou locais; • `"low"`: até USD 1/1M tokens de entrada; • `"medium"`: até USD 5/1M tokens (padrão); • `"high"`: sem restrição de custo.
`lang`	Idioma predominante do corpus: `"pt"` (padrão) ou `"en"`.
`local`	Lógico. Se `TRUE`, prioriza modelos locais (Ollama). Padrão: `FALSE`.
`n`	Número de recomendações a retornar. Padrão: `3`.
`...`	Ignorado.

Value

Tibble com as colunas de `ac_qual_list_models()` mais:

- `rank`: posição na recomendação;
- `score`: pontuação composta (0-100);
- `justificativa`: texto explicando por que o modelo foi recomendado.

References

Gilardi, F.; Alizadeh, M.; Kubli, M. (2023). ChatGPT Outperforms Crowd Workers for Text-Annotation Tasks. *PNAS*, 120(30).

Tornberg, P. (2023). ChatGPT-4 Outperforms Experts and Crowd Workers in Annotating Political Twitter Messages with Zero-Shot Learning. *PLOS ONE*, 18(4).

Alizadeh, M. et al. (2023). Open-Source LLMs for Text Annotation: A Practical Guide for Model Setting and Fine-Tuning. *arXiv*, 2307.02179.

See Also

`ac_qual_list_models()`, `ac_qual_code()`

Examples

```
# Recomendação padrão para classificação em PT com orcamento médio
ac_qual_recommend_model()

# Opcao gratuita para explorar
ac_qual_recommend_model(budget = "free")

# Local (Ollama) para dados sigilosos
ac_qual_recommend_model(local = TRUE)
```

`ac_qual_reliability` *Calcular confiabilidade entre codificação LLM e humana*

Description

`ac_qual_reliability()` calcula métricas de concordância entre a classificação feita pela LLM e uma classificação humana de referência, com intervalos de confiança via bootstrap.

É a **última porta antes de publicar** um estudo de análise de conteúdo assistida por LLM. Nenhuma referência metodológica atual (Krippendorff, 2018; Gwet, 2014; Gilardi et al., 2023) aceita rotulagem automática sem um subconjunto codificado por humano e concordância documentada.

Quatro métricas são computadas por padrão, cada uma respondendo a uma pergunta diferente:

- **Percent agreement** (`percent_agreement`): fração de documentos em que LLM e humano concordam. Fácil de reportar, mas ignora concordância por acaso: infla para categorias binárias com distribuição desigual.
- **Alpha de Krippendorff** (`krippendorff`): padrão-ouro em análise de conteúdo. Corrige por acaso, aceita missings, funciona com qualquer número de categorias. Krippendorff (2018, sec. 12.4) recomenda alfa >= 0,800 para conclusões sem ressalvas e 0,667 <= alfa < 0,800 apenas para conclusões provisórias; abaixo de 0,667 os dados não devem sustentar conclusões.
- **AC1 de Gwet** (`gwet_ac1`): alternativa mais estável quando as categorias são muito desbalanceadas (o "paradoxo do kappa" de Feinstein & Cicchetti, 1990: kappa despenca mesmo com alta concordância observada se uma categoria domina). Gwet (2014) propõe o AC1 justamente para esse caso.
- **F1 macro** (`f1_macro`): média não ponderada dos F1 por categoria (Sokolova & Lapalme, 2009). Útil quando você quer garantir que a LLM está indo bem também nas categorias raras, não só nas frequentes. **Não** é corrigido por acaso, então não deve ser lido com as faixas de Landis & Koch (1977), que foram propostas para o kappa.

Como a coluna `interpretation` é preenchida:

Cada métrica usa a referência própria dela, para não misturar escalas:

- **Alpha de Krippendorff**: `>= 0,800` confiável; `0,667-0,800` só conclusões provisórias; `< 0,667` insuficiente (Krippendorff, 2018).
- **AC1 de Gwet**: faixas de Landis & Koch (1977): `< 0` pobre; `0-0,20` leve; `0,21-0,40` razoável; `0,41-0,60` moderada; `0,61-0,80` substancial; `0,81-1` quase perfeita. Gwet (2014, cap. 6) usa essas faixas como referência de *benchmarking*.
- **Percent agreement**: `>= 0,90` alta; `0,80-0,90` aceitável na maioria dos casos; `0,70-0,80` só em estudos exploratórios; `< 0,70` insuficiente (Neuendorf, 2017; Lombard, Snyder-Duch & Bracken, 2002).
- **F1 macro**: não há faixa canônica na literatura; o valor é reportado sem rótulo qualitativo.

Essas faixas são convenções, não testes estatísticos: o próprio Landis & Koch (1977, p. 165) as chamam de arbitrárias. Por isso o IC *bootstrap* importa: um alfa de 0,82 com IC de 0,61 a 0,95 **não** permite afirmar que a confiabilidade passa de 0,800 (Krippendorff, 2018, sec. 12.5, decide justamente pela probabilidade de o alfa ficar abaixo do mínimo exigido).

Intervalos de confiança de 95% vêm de *bootstrap* não paramétrico (padrão: 1000 réplicas). Reporte pelo menos Krippendorff e F1 macro no artigo, com os ICs; discuta divergências humano × LLM em amostra de casos representativos.

Usage

```
ac_qual_reliability(
  llm,
  human,
  cat_col = "categoria",
  metrics = c("krippendorff", "gwet_ac1", "f1_macro", "percent_agreement"),
  bootstrap = 1000L,
  ci_level = 0.95,
  by_variable = FALSE,
 exclude_status = c("estouro_contexto", "limite_taxa", "timeout", "erro_servidor",
    "parcial"),
  ...
)
```

Arguments

`llm`	Tibble com classificação LLM, saída de `ac_qual_code()`.
`human`	Tibble com classificação humana, saída de `ac_qual_import_human()`.
`cat_col`	Nome da coluna de categoria. Padrão: `"categoria"`.
`metrics`	Vetor de métricas a calcular. Padrão: `c("krippendorff", "gwet_ac1", "f1_macro", "percent_agreement")`.
`bootstrap`	Número de amostras bootstrap para IC. Padrão: `1000`.
`ci_level`	Nível de confiança do IC. Padrão: `0.95`.
`by_variable`	Lógico. Se `TRUE`, calcula também métricas binárias (concordância, precisão, recall, F1, Gwet AC1, alfa de Krippendorff) para cada categoria do codebook individualmente, tratando a presença de cada categoria como um problema sim/não: essencial quando `multilabel = TRUE`, onde um único `categoria` (`"a|b"`) mistura várias decisões binárias independentes que as métricas agregadas (acima) não conseguem diagnosticar separadamente. Denominadores zero (ex.: uma categoria nunca escolhida por nenhum dos dois lados) geram `NA`, nunca `0` ou `1` artificiais. Padrão `FALSE`.
`exclude_status`	Vetor de valores da coluna `status` (ver `ac_qual_code()`) a excluir da comparação antes de calcular as métricas por variável: documentos que falharam na classificação não têm uma categoria confiável para comparar. Ignorado se `llm` não tiver coluna `status`. Padrão: `c("estouro_contexto", "limite_taxa", "timeout", "erro_servidor","parcial")`.
`...`	Ignorado.

Value

Tibble com colunas: `metric`, `estimate`, `ci_lower`, `ci_upper`, `interpretation`. Quando `by_variable = TRUE`, o resultado carrega também um atributo `by_variable`: um tibble com uma linha por categoria do codebook e colunas `categoria`, `n`, `vp`, `fp`, `fn`, `vn`, `concordância`, `precisao`, `recall`, `f1`, `gwet_ac1`, `krippendorff_alfa`.

References

Krippendorff, K. (2018). *Content Analysis: An Introduction to Its Methodology* (4th ed.). SAGE.

Gwet, K. L. (2014). *Handbook of Inter-Rater Reliability* (4th ed.). Advanced Analytics.

Landis, J. R.; Koch, G. G. (1977). The Measurement of Observer Agreement for Categorical Data. *Biometrics*, 33(1), 159-174.

Feinstein, A. R.; Cicchetti, D. V. (1990). High agreement but low kappa: I. The problems of two paradoxes. *Journal of Clinical Epidemiology*, 43(6), 543-549.

Lombard, M.; Snyder-Duch, J.; Bracken, C. C. (2002). Content analysis in mass communication: Assessment and reporting of intercoder reliability. *Human Communication Research*, 28(4), 587-604.

Neuendorf, K. A. (2017). *The Content Analysis Guidebook* (2nd ed.). SAGE.

Sokolova, M.; Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4), 427-437.

Examples

```
# Simular saidas: LLM (coded) e revisao humana (humano_df)
# ambos com as colunas doc_id + categoria
coded <- tibble::tibble(
  doc_id    = paste0("d", 1:5),
  categoria = c("favor", "contra", "favor", "contra", "favor")
)
humano_df <- tibble::tibble(
  doc_id    = paste0("d", 1:5),
  categoria = c("favor", "contra", "contra", "contra", "favor")
)

# Calcular métricas de confiabilidade (bootstrap curto so para demonstrar;
# em uso real, deixe o padrao de 1000 replicas)
rel <- ac_qual_reliability(
  llm       = coded,
  human     = humano_df,
  bootstrap = 50
)
print(rel)
```

`ac_qual_report` *Gerar relatório de replicabilidade da análise qualitativa*

Description

`ac_qual_report()` gera um documento estruturado, pronto para artigo ou relatório, com todas as decisões metodológicas da rodada de codificação qualitativa: codebook completo, histórico de modificações, configuração da LLM, distribuição de resultados, métricas de confiabilidade e referências bibliográficas.

É a resposta do acR a um risco real de comunicação em análises assistidas por LLM: **o revisor ou leitor não consegue reproduzir a rodada** sem saber exatamente qual modelo, qual codebook, quais parâmetros e qual amostra foram usados. Sem esse relatório, o leitor tem que confiar no autor: algo que a tradição de análise de conteúdo (Krippendorff, 2018) sempre rejeitou. O documento gerado é autocontido e pode ser anexado como material suplementar do artigo, publicado como apêndice ou depositado num repositório de dados junto com o codebook em YAML.

Suporta saída em Markdown (.md) ou HTML autocontido (.html), em português ou inglês (para submissões internacionais).

Usage

```
ac_qual_report(
  coded,
  codebook,
  reliability = NULL,
  chat = NULL,
  title = NULL,
  author = NULL,
  method = NULL,
  format = c("md", "html"),
  path = NULL,
  lang = c("pt", "en")
)
```

Arguments

coded	Tibble com resultado de ac_qual_code().
codebook	Objeto ac_codebook usado na classificação.
reliability	Opcional. Saida de ac_qual_reliability(); se fornecido, adiciona secao de confiabilidade inter-codificador.
chat	Opcional. Objeto Chat do ellmer (ou string de modelo); se fornecido, extrai provedor/modelo/parametros para o relatório.
title	Título do relatório. Padrao: gerado a partir do nome do codebook.
author	Autor(es) do estudo (opcional).
method	Descrição livre do metodo de coleta do corpus (opcional).
format	Formato de saida: "md" (padrao) ou "html".
path	Caminho do arquivo destino. Se NULL, usa tempfile().
lang	Idioma: "pt" (padrao) ou "en".

Value

Invisivel: caminho do arquivo gerado.

Examples

```
# Simular resultado de ac_qual_code para o exemplo
cb <- ac_qual_codebook(
  name         = "polaridade",
  instructions = "Classifique a polaridade do texto.",
  categories   = list(
    favor  = list(definition = "Apoio a proposta."),
    contra = list(definition = "Oposicao a proposta.")
  )
)

coded <- tibble::tibble(
  doc_id           = paste0("d", 1:5),
  categoria        = c("favor", "contra", "favor", "contra", "favor"),
  confidence_score = c(1.00, 0.67, 1.00, 1.00, 0.67),
  reasoning        = rep("...", 5)
)

# Gerar relatório em markdown temporário
arquivo <- tempfile(fileext = ".md")
ac_qual_report(coded, cb, path = arquivo, author = "Fulano de Tal")
# readLines(arquivo, n = 20)
```

`ac_qual_report_full` *Gerar relatório consolidado de múltiplas variáveis de conteúdo*

Description

`ac_qual_report_full()` gera **um único** documento (Markdown ou HTML) consolidando resultados de N variáveis de codificação qualitativa: cada uma com seu próprio codebook, rodada de `ac_qual_code()` e, opcionalmente, métricas de `ac_qual_reliability()` ou `ac_qual_irr()`.

É a resposta natural ao caso de uso mais comum de análise de conteúdo categorial (Krippendorff, 2018): um estudo tem várias variáveis de conteúdo (tom, posicionamento, tema, framing...). Antes desta função, o pesquisador precisava chamar `ac_qual_report()` N vezes e emendar os relatórios manualmente, ou escrever um consolidador ad hoc por projeto.

Aceita variáveis `multilabel = TRUE` e `multilabel = FALSE` no mesmo relatório: cada seção herda a apresentação apropriada para o tipo.

Usage

```
ac_qual_report_full(
  variables,
  chat = NULL,
  title = NULL,
  author = NULL,
  method = NULL,
  format = c("md", "html"),
```

```
  path = NULL,
  lang = c("pt", "en")
)
```

Arguments

variables	Lista **nomeada** onde cada elemento é uma lista com: • coded: tibble com resultado de ac_qual_code() (obrigatório). • codebook: objeto ac_codebook (obrigatório). • reliability: opcional, saída de ac_qual_reliability() ou ac_qual_irr(). Os nomes da lista viram os títulos das seções do relatório.
chat	Opcional. Objeto Chat do ellmer ou string de modelo "provider/model". Aplica-se a todas as variáveis (assumindo que todas foram classificadas com o mesmo modelo: se não, deixe NULL).
title	Título do relatório consolidado. Padrão: "Relatório consolidado de análise de conteúdo".
author	Autor(es) do estudo.
method	Descrição do método de coleta do corpus.
format	"md" (padrão) ou "html".
path	Caminho do arquivo destino. Se NULL, usa tempfile().
lang	"pt" (padrão) ou "en".

Value

Invisível: caminho do arquivo gerado.

See Also

ac_qual_report() para relatório de uma única variável.

Examples

```
# Duas variáveis de conteúdo, cada uma com seu codebook
cb_tom <- ac_qual_codebook(
  name = "tom",
  instructions = "Classifique o tom.",
  categories = list(
    pos = list(definition = "Positivo.", label = "Positivo"),
    neg = list(definition = "Negativo.", label = "Negativo")
  )
)
cb_pos <- ac_qual_codebook(
  name = "posicao",
  instructions = "Classifique a posicao.",
  categories = list(
    favor  = list(definition = "A favor.",  label = "A favor"),
    contra = list(definition = "Contra.",   label = "Contra")
  )
```

```
)

coded_tom <- tibble::tibble(
  doc_id = paste0("d", 1:3),
  categoria = c("pos", "neg", "pos"),
  confidence_score = c(1, 0.67, 1)
)
coded_pos <- tibble::tibble(
  doc_id = paste0("d", 1:3),
  categoria = c("favor", "contra", "favor"),
  confidence_score = c(1, 1, 0.67)
)

arquivo <- tempfile(fileext = ".md")
ac_qual_report_full(
  variables = list(
    "Tom do discurso" = list(coded = coded_tom, codebook = cb_tom),
    "Posicao politica" = list(coded = coded_pos, codebook = cb_pos)
  ),
  path = arquivo,
  author = "Fulano de Tal"
)
```

`ac_qual_sample` *Amostrar documentos para validação humana*

Description

`ac_qual_sample()` seleciona uma amostra de documentos classificados pela LLM para validação por um codificador humano.

Duas finalidades diferentes, duas estratégias diferentes.

1. *Estimar a confiabilidade* que vai para o artigo (`ac_qual_reliability()`): a amostra precisa ser **representativa** do corpus. Krippendorff (2018, cap. 12) exige que os dados de confiabilidade sejam amostrados da mesma população que os dados analisados; uma amostra enviesada para casos difíceis subestima a concordância real, e uma enviesada para casos fáceis a superestima. Use `"random"` (padrão) ou `"proportional"`.
2. *Diagnosticar erros* e refinar o codebook: aí vale concentrar o esforço humano onde a LLM mais erra, como em amostragem por incerteza de *active learning* (Settles, 2012). Use `"uncertainty"` ou `"disagreement"`, mas **não** reporte a concordância calculada nessa amostra como confiabilidade do corpus.

Sobre o tamanho: Neuendorf (2017) sugere uma subamostra de confiabilidade de 10% a 20% do corpus, raramente menos de 50 unidades; Krippendorff (2018, sec. 12.3) oferece um cálculo formal a partir do alfa mínimo aceitável e da frequência da categoria mais rara.

Usage

```
ac_qual_sample(
  coded,
  n = 50L,
 strategy = c("random", "uncertainty", "stratified", "disagreement", "proportional"),
  strata = NULL,
  min_per_stratum = 1L,
  min_stratum_size = 10L,
  exclude_ids = NULL,
  seed = 42L,
  ...
)
```

Arguments

`coded`	Tibble com classificação LLM, saída de `ac_qual_code()`.
`n`	Número de documentos a amostrar. Padrão: `50`.
`strategy`	Estratégia de amostragem: • `"random"` (padrão): amostra aleatória simples. Representativa: use para estimar a confiabilidade que vai para o artigo; • `"uncertainty"`: prioriza documentos com menor `confidence_score` (maior incerteza da LLM). Serve para diagnóstico de erros, **não** para estimar confiabilidade (amostra enviesada para casos difíceis); • `"stratified"`: divide `n` igualmente entre as categorias atribuídas pela LLM (sobre-representa categorias raras; não é representativa); • `"disagreement"`: prioriza documentos onde rodadas de self-consistency divergiram (requer `confidence_score` < 1); • `"proportional"`: amostragem estratificada com alocação proporcional ao tamanho de cada estrato (método do maior resto), respeitando um mínimo por estrato. Diferente de `"stratified"` (que divide `n` igualmente entre categorias), preserva a distribuição original de `strata` na amostra: a escolha correta quando a amostra alimenta uma estimativa populacional (ex.: `ac_qual_reliability()` ponderada), não só uma checagem qualitativa de cada categoria.
`strata`	Nome da coluna usada para estratificar quando `strategy = "proportional"`. Obrigatório nesse caso.
`min_per_stratum`	Mínimo de documentos por estrato na amostra proporcional, mesmo que a alocação exata dê menos. Padrão `1`.
`min_stratum_size`	Estratos da população com menos documentos que isso são agrupados em `"Outras"` antes de alocar: evita alocar uma fração ínfima da amostra a uma categoria com 2 documentos na população inteira. Padrão `10`.
`exclude_ids`	Vetor de `doc_id` a excluir do universo amostrado (ex.: documentos já validados em uma rodada anterior). Padrão `NULL`.

seed	Semente para reprodutibilidade. Padrão: 42.
...	Ignorado.

Value

Tibble com os documentos selecionados, incluindo uma coluna sample_reason indicando por que cada documento foi selecionado. Com strategy = "proportional", ganha também peso_amostral (n_populacao_estrato / n_amostra_estrato) para uso em estimativas ponderadas.

References

Krippendorff, K. (2018). *Content Analysis: An Introduction to Its Methodology* (4th ed.). SAGE.

Neuendorf, K. A. (2017). *The Content Analysis Guidebook* (2nd ed.). SAGE.

Settles, B. (2012). *Active Learning*. Morgan & Claypool.

Examples

```
# Simular saida de ac_qual_code() com 20 documentos já classificados
set.seed(1)
coded <- tibble::tibble(
  doc_id           = paste0("doc_", 1:20),
  categoria        = sample(c("favor", "contra"), 20, replace = TRUE),
  confidence_score = runif(20, 0.5, 1.0)
)

# Amostra representativa (padrao): para estimar a confiabilidade
ac_qual_sample(coded, n = 5)

# Priorizar casos mais incertos: diagnostico de erros, nao confiabilidade
ac_qual_sample(coded, n = 5, strategy = "uncertainty")

# Garantir cobertura proporcional das duas categorias
ac_qual_sample(coded, n = 6, strategy = "stratified")
```

---

ac_qual_save_codebook *Salvar codebook em arquivo YAML*

Description

Serializa um objeto ac_codebook em disco no formato YAML, preservando instrucoes, categorias, exemplos, referências e histórico de modificacoes. YAML foi escolhido por ser legivel por humanos e versionavel em Git.

Usage

```
ac_qual_save_codebook(codebook, path = NULL, ...)
```

Arguments

codebook	Objeto ac_codebook.
path	Caminho do arquivo .yaml. Se NULL, deriva do nome do codebook.
...	Ignorado.

Value

Invisivel: caminho do arquivo gerado.

Examples

```
# Criar um codebook simples
cb <- ac_qual_codebook(
  name         = "tom_discurso",
  instructions = "Classifique o tom do discurso.",
  categories   = list(
    positivo = list(definition = "Tom propositivo e colaborativo."),
    negativo = list(definition = "Tom crítico e confrontacional.")
  )
)

# Salvar em arquivo temporário (usar caminho real fora de exemplos)
arquivo <- tempfile(fileext = ".yaml")
ac_qual_save_codebook(cb, path = arquivo)

# Reabrir para conferir
cb_recarregado <- ac_qual_load_codebook(arquivo)
identical(cb$categories, cb_recarregado$categories)
```

---

ac_qual_search_literature

Buscar referências bibliograficas sobre um conceito via OpenAlex e LLM

Description

ac_qual_search_literature() busca referências acadêmicas reais na API do OpenAlex e usa um modelo de linguagem via ellmer para sintetizar os abstracts em português. Retorna um tibble com metadados bibliograficos verificados e definicoes sintetizadas pela LLM.

A arquitetura e: OpenAlex recupera registros reais (autor, ano, DOI, abstract, revista, número de citacoes); a LLM sintetiza o abstract em português e extrai o trecho mais relevante. Isso evita alucinacoes bibliograficas comuns quando a LLM opera sem fonte externa.

Usage

```
ac_qual_search_literature(
  concept,
  chat = NULL,
  model = "anthropic/claude-sonnet-4-5",
  n_refs = 5L,
  journals = "default",
  lang = "pt",
  min_citations = 0L,
  ...
)
```

Arguments

concept	String. Conceito ou termo teórico a buscar (ex: "democratic backsliding", "state capacity").
chat	Objeto Chat do pacote ellmer (ex: chat_google_gemini(), chat_openai(), chat_ollama()). Quando fornecido, tem prioridade sobre model.
model	String no formato "provedor/modelo". Ignorado quando chat e fornecido.
n_refs	Inteiro. Número de referências a retornar. Padrao: 5.
journals	Periodicos a considerar. Opcoes: • "default": lista curada de periodicos de referência em CP/CS/AP; • "all": sem restricao de periodico; • Vetor de strings: lista customizada (ex: c("default", "RBCS")).
lang	Idioma das definicoes sintetizadas. Padrao: "pt".
min_citations	Inteiro. Número mínimo de citacoes. Padrao: 0.
...	Argumentos adicionais passados a ellmer::chat().

Value

Tibble com colunas: conceito, autor, ano, revista, n_citacoes, trecho_original, definicao_pt, abstract_original, link.

References

Priem, J. et al. (2022). OpenAlex: A fully-open index of the global research system. *arXiv*, 2205.01833.

Gilardi, F.; Alizadeh, M.; Kubli, M. (2023). ChatGPT Outperforms Crowd Workers for Text-Annotation Tasks. *PNAS*, 120(30).

Examples

```
## Not run:
# Requer internet (busca no OpenAlex) e credenciais da LLM (para sintese)

# Chat via Groq (plano gratuito com gpt-oss-120b)
```

```
chat_obj <- ellmer::chat_groq(model = "openai/gpt-oss-120b", echo = "none")

# Buscar 5 referências mais citadas sobre o conceito e sintetizar cada uma
lit <- ac_qual_search_literature(
  concept       = "democratic backsliding",
  n_refs        = 5,
  min_citations = 50,  # filtra papers com pouca reverberacao
  chat          = chat_obj
)

# Colunas principais do resultado
print(lit[, c("autor", "ano", "revista", "n_citacoes", "definicao_pt")])

## End(Not run)
```

`ac_readability` *Calcular legibilidade textual (Índice de Flesch adaptado ao português)*

Description

`ac_readability()` estima a legibilidade (facilidade de leitura) dos documentos de um `ac_corpus`, usando o Índice de Legibilidade de Flesch adaptado ao português por Martins et al. (1996) a partir da fórmula original de Flesch (1948).

É uma ferramenta de diagnóstico, não de julgamento de qualidade: um texto "difícil" pelo índice pode ser exatamente o esperado: uma decisão judicial, um artigo técnico, uma bula de remédio. O uso típico é **comparativo**: saber se o manual de instruções do produto A é mais acessível que o do produto B, se a imprensa simplificou a linguagem de comunicados oficiais ao longo do tempo, ou se avaliações de um produto em um site de e-commerce variam em complexidade por categoria.

A fórmula depende da contagem de sílabas por palavra, que a função estima contando grupos de vogais (heurística comum em legibilidade automática, já que não há dicionário de sílabas open-source completo para português). Isso introduz algum erro em palavras com hiatos, ditongos ou dígrafos vocálicos, mas a estimativa costuma ficar próxima da contagem manual para fins comparativos.

Usage

```
ac_readability(corpus, by = NULL)
```

Arguments

`corpus`	Objeto `ac_corpus`.
`by`	Coluna(s) de agrupamento para agregar a legibilidade além do documento (ex.: `"categoria"`, `"fonte"`, `"ano"`). Padrão: `NULL` (uma linha por documento).

Details

A fórmula (Martins et al., 1996) é:

$$ILF = 248{,}835 - 1{,}015 \times \frac{palavras}{frases} - 84{,}6 \times \frac{slabas}{palavras}$$

Faixas de interpretação de Martins et al. (1996), que calibraram a fórmula em livros didáticos brasileiros:

- `>= 75`: muito fácil (1º ao 5º ano do ensino fundamental);
- `50-74,9`: fácil (6º ao 9º ano);
- `25-49,9`: difícil (ensino médio e superior);
- `< 25`: muito difícil (textos acadêmicos/especializados).

Contagens aproximadas. As sílabas são estimadas por grupos de vogais contíguas, heurística usual em legibilidade automática: acerta ditongos ("cai-xa"), mas conta hiatos como uma sílaba só ("sa-ú-de" vira 2), o que tende a subestimar sílabas e superestimar o índice. As frases são separadas em `.`, `!` e `?`, então abreviações ("Sr.", "art.") e números decimais ("3.5") inflam o número de frases. Ao comparar grupos de textos do mesmo gênero, o viés é parecido nos dois lados; para afirmar o nível absoluto de um texto, confira uma amostra à mão.

Value

Tibble com colunas:

- `doc_id` (ou as colunas de `by`, se informado);
- `n_palavras`, `n_frases`, `n_silabas`: contagens usadas na fórmula;
- `flesch`: Índice de Legibilidade de Flesch adaptado ao português (tipicamente 0-100, mas sem teto nem piso matemático: textos muito densos podem produzir valores negativos; valores mais altos = texto mais fácil de ler);
- `nivel`: interpretação categórica do índice (ver `@details`).

References

Flesch, R. (1948). A New Readability Yardstick. *Journal of Applied Psychology*, 32(3), 221-233.

Martins, T. B. F., Ghiraldelo, C. M., Nunes, M. G. V., Oliveira Jr., O. N. (1996). Readability formulas applied to textbooks in Brazilian Portuguese. *ICMSC Technical Report*, 28.

See Also

`ac_lexical_diversity()`

Examples

```
# Avaliações de produtos em um e-commerce, com nivel de complexidade variado
df <- data.frame(
  id = paste0("r", 1:4),
  texto = c(
    "Bom. Gostei. Chegou rápido.",
```

```
    paste(
      "A ergonomia do dispositivo privilegia uma distribuicao",
      "assimetrica de peso, o que compromete substancialmente a",
      "usabilidade prolongada em cenários de utilizacao intensiva."
    ),
    "Fácil de usar e bonito.",
    "O manual e confuso e a instalacao exige conhecimento técnico avancado."
  ),
  categoria = c("eletronico", "eletronico", "casa", "eletronico"),
  stringsAsFactors = FALSE
)
corpus <- ac_corpus(df, text = texto, docid = id)

ac_readability(corpus)

# Comparando legibilidade média por categoria de produto
ac_readability(corpus, by = "categoria")
```

`ac_sentiment` *Análise de sentimento com OpLexicon*

Description

`ac_sentiment()` calcula a polaridade de sentimento dos documentos de um `ac_corpus` usando o OpLexicon (Souza & Vieira, 2012), retornando pontuações por documento e, opcionalmente, por grupo ou janela temporal.

Usage

```
ac_sentiment(
  corpus,
  by = NULL,
  lexicon = c("oplexicon"),
  method = c("sum", "mean", "ratio"),
  ...
)
```

Arguments

`corpus`	Objeto `ac_corpus`.
`by`	Coluna(s) de agrupamento para agregar o sentimento além do documento (ex: `"partido"`, `"data"`): todos os tokens dos documentos do grupo entram juntos no cálculo, e a saída tem uma linha por grupo (sem `doc_id`). Padrão: `NULL` (uma linha por documento).
`lexicon`	Léxico a usar. Atualmente apenas `"oplexicon"` (padrão).
`method`	Método de agregação por documento (ou grupo):

- `"sum"` (padrão): soma das polaridades (+1/-1). Cresce com o tamanho do texto: não compare documentos de tamanhos muito diferentes por esse escore.
- `"mean"`: soma das polaridades dividida pelo **total de tokens** (inclusive os que não estão no léxico), ou seja, saldo de sentimento por palavra.
- `"ratio"`: $(pos - neg)/(pos + neg)$, entre -1 e 1; usa só as palavras com polaridade e por isso independe do tamanho do texto (é a forma "relativa" de Lowe et al., 2011, sem a suavização logarítmica).

`...` Ignorado.

Details

Download e cache. O OpLexicon não vem dentro do pacote: na primeira execução ele é baixado do repositório público do léxico no GitHub e salvo em `tools::R_user_dir("acR", "data")`; as execuções seguintes usam esse arquivo e funcionam sem internet. Se o download falhar (sem conexão, por exemplo), a função para com uma mensagem explicando o motivo.

Como o léxico é aplicado. O texto é limpo (minúsculas, sem pontuação), tokenizado em palavras e cada palavra recebe a polaridade do OpLexicon (-1, 0, +1); palavras fora do léxico valem 0. Alguns termos aparecem no OpLexicon mais de uma vez (uma por classe gramatical); nesse caso usa-se o sinal da soma das polaridades do termo (polaridades conflitantes, como +1 e -1, viram 0), para que cada token conte uma única vez.

Limites do método de dicionário. A contagem é palavra a palavra: não trata negação ("não é bom" soma +1), intensificadores ("muito ruim") nem ironia, e a polaridade de uma palavra depende do domínio (Taboada et al., 2011; Grimmer & Stewart, 2013). Por isso valide numa amostra codificada por humano (ver `ac_dictionary_validate()`) antes de interpretar os escores.

Value

Tibble com colunas:

- `doc_id`: identificador do documento;
- Colunas de metadado (se `by` especificado);
- `n_pos`: número de tokens positivos;
- `n_neg`: número de tokens negativos;
- `n_neu`: número de tokens sem polaridade (neutros no léxico **ou** ausentes dele);
- `score`: pontuação de sentimento (método escolhido);
- `sentiment`: classificação (`"positivo"`, `"negativo"`, `"neutro"`).

References

Souza, M.; Vieira, R. (2012). Sentiment Analysis on Twitter Data for Portuguese Language. *PROPOR*.

Souza, M.; Vieira, R.; Busetti, D.; Chishman, R.; Alves, I. M. (2011). Construction of a Portuguese Opinion Lexicon from multiple resources. *STIL/SBC*.

Grimmer, J.; Stewart, B. M. (2013). Text as Data: The Promise and Pitfalls of Automatic Content Analysis Methods for Political Texts. *Political Analysis*, 21(3), 267-297.

Lowe, W.; Benoit, K.; Mikhaylov, S.; Laver, M. (2011). Scaling Policy Preferences from Coded Political Texts. *Legislative Studies Quarterly*, 36(1), 123-155.

Taboada, M.; Brooke, J.; Tofiloski, M.; Voll, K.; Stede, M. (2011). Lexicon-Based Methods for Sentiment Analysis. *Computational Linguistics*, 37(2), 267-307.

See Also

`ac_plot_sentiment()`

Examples

```
# Requer internet na primeira execucao: ac_sentiment() baixa o OpLexicon
# do GitHub e o guarda em cache (tools::R_user_dir("acR", "data")).
# Discursos com valência afetiva variada
df <- data.frame(
  id = paste0("d", 1:6),
  texto = c(
    "A reforma tributária e um passo excelente para o pais",
    "Esta proposta e um desastre, um retrocesso terrivel",
    "O relator apresentou o parecer na sessão ordinária",
    "Comemoramos essa vitória histórica com muita alegria",
    "Rejeitamos com veemência essa medida injusta e ilegal",
    "A comissao encerra os trabalhos as 18h"
  ),
  partido = rep(c("A", "B"), 3),
  stringsAsFactors = FALSE
)
corpus <- ac_corpus(df, text = texto, docid = id)

# Polaridade agregada por documento (soma de scores OpLexicon)
ac_sentiment(corpus)

# Agregado por grupo (aqui, por partido)
ac_sentiment(corpus, by = "partido")

# Metodo alternativo: razao positivos/negativos
ac_sentiment(corpus, method = "ratio")
```

`ac_temporal` *Agregar uma métrica por período de tempo*

Description

`ac_temporal()` agrega qualquer métrica por documento (sentimento, frequência de um termo, legibilidade, confiança de classificação...) em janelas de tempo, para análise diacrônica: a pergunta "isso mudou ao longo do tempo?" aparece em quase todo estudo de conteúdo com timestamp: cobertura de imprensa sobre um tema, reclamações de clientes por mês, evolução do tom de comunicados de uma empresa, posts de redes sociais durante uma crise.

A função é deliberadamente genérica: não importa se o valor vem de `ac_sentiment()`, `ac_readability()`, `ac_count()` filtrado para um termo específico, ou da saída de `ac_qual_code()`: qualquer tibble com uma coluna de data e uma coluna numérica serve de entrada.

Usage

```
ac_temporal(
  data,
  date_col,
  value_col,
  unit = c("month", "day", "week", "year"),
  stat = c("mean", "sum", "n")
)
```

Arguments

`data`	Tibble (ou `data.frame`) com, no mínimo, uma coluna de data e uma coluna numérica a agregar: tipicamente o resultado de uma das funções de análise do pacote, unido aos metadados do corpus (ex.: `dplyr::left_join()` de `ac_sentiment()` com a data de cada documento).
`date_col`	Nome da coluna de data em `data`. Convertida com `as.Date()` se ainda não for `Date`.
`value_col`	Nome da coluna numérica a agregar por período.
`unit`	Unidade temporal da agregação: `"day"`, `"week"`, `"month"` (padrão) ou `"year"`.
`stat`	Estatística de agregação dentro de cada período: `"mean"` (padrão), `"sum"` ou `"n"` (contagem de documentos, ignorando `value_col`).

Value

Tibble com colunas `period` (`Date`, início do período), `value` (estatística agregada) e `n_docs` (documentos naquele período).

References

Welbers, K.; Van Atteveldt, W.; Benoit, K. (2017). Text Analysis in R. *Communication Methods and Measures*, 11(4), 245-265.

See Also

`ac_plot_temporal()`, `ac_sentiment()`

Examples

```
# Reclamacoes de clientes ao longo de alguns meses, com score de
# sentimento já calculado por documento (ex.: saida de ac_sentiment())
df <- data.frame(
  doc_id = paste0("d", 1:6),
  data   = as.Date(c("2024-01-05", "2024-01-20", "2024-02-02",
                     "2024-02-15", "2024-03-01", "2024-03-10")),
```

```
  score  = c(-2, -1, 0, 1, 2, 1)
)

ac_temporal(df, date_col = "data", value_col = "score", unit = "month")
```

---

ac_tf_idf *Calcular tf-idf para termos em documentos ou grupos*

Description

`ac_tf_idf()` calcula a frequência de termos (`tf`), a frequência inversa de documentos (`idf`) e o produto `tf_idf` a partir de uma tabela de frequências de termos (tipicamente o resultado de `ac_count()`).

A função segue a mesma lógica de `tidytext::bind_tf_idf()`, mas adaptada para o fluxo de trabalho do pacote:

- sem grupos (`by = NULL`): cada `doc_id` e tratado como um documento;
- com grupos (`by = c("partido", ...)`): cada combinacao de metadados e tratada como um "documento" no calculo de `idf`.

Usage

```
ac_tf_idf(x, by = NULL)
```

Arguments

`x`	Um `data.frame` ou `tibble::tibble()` contendo, no mínimo, as colunas `token` e `n`. Em geral, o resultado de `ac_count()`.
`by`	Vetor de nomes de colunas em `x` que identificam documentos ou grupos. Se `NULL` (padrao), usa `doc_id` (que deve existir em `x`). Se não for `NULL`, cada combinacao de `by` e tratada como um documento no calculo de `idf`.

Details

A definicao de `tf`, `idf` e `tf_idf` segue a literatura padrao de tf-idf em mineracao de texto:

- `tf` e a frequência relativa do termo no documento;
- `idf` e log(N / df), em que N e o número de documentos e df e o número de documentos que contem o termo;
- `tf_idf` e o produto `tf * idf`.

A tabela de entrada deve ter exatamente uma linha por combinacao de documento/grupo e termo (isto e, uma linha por termo-em-documento).

Value

Um tibble::tibble() com as colunas originais de x mais tres colunas numeric as:

- tf: frequência do termo no documento/grupo;
- idf: frequência inversa de documentos;
- tf_idf: produto tf * idf.

See Also

ac_count(), ac_top_terms(), tidytext::bind_tf_idf()

Examples

```
# Corpus com discursos sobre tres agendas distintas.
# TF-IDF vai destacar o vocabulário proprio de cada texto,
# não os termos comuns a todos (que a frequência bruta mostraria).
df <- data.frame(
  id    = paste0("d", 1:6),
  texto = c(
    "reforma tributária simplifica sistema impostos empresas",
    "IVA dual substitui PIS COFINS ICMS federal",
    "programa habitacional amplia recursos moradia popular urbana",
    "deficit habitacional afeta familias baixa renda cidades",
    "educacao básica recebe recursos Fundeb Congresso",
    "alfabetizacao idade certa meta Plano Nacional Educacao"
  ),
  tema  = rep(c("tributário", "habitacao", "educacao"), each = 2),
  stringsAsFactors = FALSE
)

corp <- ac_corpus(df, text = texto, docid = id)

# TF-IDF por documento
freq  <- ac_count(corp)
tfidf <- ac_tf_idf(freq)
head(tfidf)

# TF-IDF por tema (cada tema tratado como "documento agregado")
freq_by  <- ac_count(corp, by = "tema")
tfidf_by <- ac_tf_idf(freq_by, by = "tema")
head(tfidf_by)
```

---

ac_tokenize *Tokenizar textos de um corpus acR*

Description

ac_tokenize() recebe um objeto ac_corpus() e retorna um tibble em formato *tidy*, com um token por linha, no estilo usado em análises de texto no ecossistema tidy.

A função implementa tokenização em palavras (n = 1) ou n-gramas de tamanho arbitrário (n > 1), usando janelas contíguas de tokens dentro de cada documento.

É o segundo passo canônico do pipeline quantitativo, geralmente entre ac_clean() e ac_count(). Para análises que dependem de expressões compostas (\"reforma tributária\", \"desenvolvimento sustentável\"), use n = 2L para bigramas: cada n-grama vira uma unidade de contagem independente.

Usage

```
ac_tokenize(
  corpus,
  token = c("word"),
  n = 1L,
  keep_empty = FALSE,
  drop_punct = FALSE,
  ...
)
```

Arguments

corpus	Objeto de classe ac_corpus().
token	Tipo de tokenizacao desejada. Atualmente apenas "word" e suportado (padrao), reservado para futura expansao.
n	Tamanho do n-grama. Deve ser um inteiro maior ou igual a 1. Para n = 1, o resultado sao tokens individuais; para n = 2, bigramas ("A B"), para n = 3, trigramas etc.
keep_empty	Lógico. Se FALSE (padrao), documentos que resultarem em texto vazio após a limpeza não geram linhas na saida. Se TRUE, cada documento vazio gera uma linha com token = NA quando n = 1.
drop_punct	Lógico. Se TRUE, remove a pontuação das **bordas** de cada token ("excelente," vira "excelente", "(péssimo)" vira "péssimo") e descarta tokens que ficam vazios ("!", "..."), antes de construir n-gramas. Pontuação **interna** é preservada, para não quebrar formas como "e-mail", "guarda-chuva" ou "3,5". Sem isso, "excelente," e "excelente" seriam contados como dois tipos diferentes, o que distorce frequências, TTR, dicionários e Wordscores.
...	Ignorado, reservado para argumentos futuros.

Value

Um tibble::tibble() com colunas:

- doc_id: identificador do documento (herdado de ac_corpus);
- token_id: posicao (1, 2, 3, ...) do token ou n-grama no documento;
- token: texto do token ou n-grama.

See Also

ac_corpus(), ac_clean()

Examples

```
df <- data.frame(
  id    = c("d1", "d2"),
  texto = c(
    "O deputado do PT falou na CCJ.",
    "Votar pra valer, agora!"
  )
)

corp <- ac_corpus(df, text = texto, docid = id)

# Tokenizacao simples em palavras
tokens <- ac_tokenize(corp)
tokens

# Removendo tokens que sao apenas pontuação
df2 <- data.frame(
  id    = "d1",
  texto = "Ola, mundo ! ..."
)
corp2   <- ac_corpus(df2, text = texto, docid = id)
tokens2 <- ac_tokenize(corp2, drop_punct = TRUE)
tokens2

# Bigramas
ac_tokenize(corp, n = 2)
```

---

ac_top_terms *Selecionar os termos mais frequentes*

Description

ac_top_terms() seleciona os n termos mais frequentes a partir de uma tabela de frequências (tipicamente o resultado de ac_count()).

Pode operar em dois modos:

- sem grupos (by = NULL): retorna os n termos mais frequentes no geral;
- com grupos (by = c("partido", ...)): retorna os n termos mais frequentes em cada combinacao de metadados.

Usage

```
ac_top_terms(x, n = 20L, by = NULL, sort = TRUE)
```

Arguments

x	Um data.frame ou tibble::tibble() contendo, no mínimo, as colunas token e n. Em geral, o resultado de ac_count().
n	Número de termos a selecionar. Valor inteiro >= 1.
by	Vetor de nomes de colunas em x a serem usados como grupos de agregacao. Se NULL (padrao), a selecao e feita no conjunto total. Se não for NULL, os n termos mais frequentes sao selecionados dentro de cada combinacao de by.
sort	Lógico. Se TRUE (padrão), ordena a saída em ordem decrescente de frequência (n). Se FALSE, ordena os termos selecionados alfabeticamente (dentro de cada grupo, quando houver).

Value

Um tibble::tibble() com as colunas token e n (mais as colunas de by, se houver), com exatamente n termos (ou menos, se o vocabulário for menor) no geral ou por grupo. A frequência de cada termo é **somada** sobre os documentos (ou dentro de cada grupo), de modo que uma tabela por documento, saída padrão de ac_count(), dá o total do corpus. Empates na última posição são desfeitos em ordem alfabética.

See Also

ac_count(), ac_tokenize()

Examples

```
df <- data.frame(
  id      = c("d1", "d2", "d3"),
  texto   = c(
    "O deputado do PT falou na CCJ.",
    "O deputado do PL falou novamente.",
    "O senador do PT falou na CCJ."
  ),
  partido = c("PT", "PL", "PT"),
  stringsAsFactors = FALSE
)

corp <- ac_corpus(df, text = texto, docid = id, meta = partido)

# Top 10 termos no corpus inteiro
freq <- ac_count(corp)
ac_top_terms(freq, n = 10)

# Top 5 termos por partido
freq_by <- ac_count(corp, by = "partido")
ac_top_terms(freq_by, n = 5, by = "partido")
```

ac_wordcloud *Criar nuvem de palavras*

Description

ac_wordcloud() cria uma nuvem de palavras a partir de uma tabela de frequências, tipicamente gerada por ac_count().

Por padrão prefere ggwordcloud (retorna ggplot, layout mais agradável, tipografia melhor); cai para wordcloud clássico se o primeiro não estiver instalado.

Usage

```
ac_wordcloud(
  x,
  max_words = 100,
  min_n = 1,
  colors = NULL,
  backend = c("auto", "ggwordcloud", "wordcloud"),
  title = NULL,
  seed = 42L,
  ...
)
```

Arguments

x	Um data.frame ou tibble contendo, no mínimo, as colunas token e n.
max_words	Número máximo de palavras a desenhar. Padrão: 100.
min_n	Frequência mínima para incluir um termo. Padrão: 1.
colors	Vetor de cores usado no gráfico. Padrão: paleta ac_palette().
backend	Motor a usar: "auto" (padrão, prefere ggwordcloud), "ggwordcloud" ou "wordcloud".
title	Título opcional (apenas em modo ggwordcloud).
seed	Semente para reprodutibilidade do layout. Padrão: 42L. Use NULL para usar o RNG corrente da sessão. A semente é escopada via withr::with_seed() e não altera o .Random.seed global.
...	Argumentos adicionais encaminhados para o motor escolhido (ggwordcloud::geom_text_wordcloud ou wordcloud::wordcloud).

Value

Um objeto ggplot (backend ggwordcloud) ou, invisivelmente, o data.frame filtrado (backend wordcloud).

See Also

ac_count(), ac_top_terms(), ac_palette()

Examples

```
# Corpus pequeno para demonstrar
df <- data.frame(
  id    = paste0("d", 1:8),
  texto = c(
    "reforma tributária simplifica sistema empresas",
    "reforma reduz distorcoes fiscais brasileiras",
    "sistema tributário complexo prejudica empresas",
    "reforma modernizacao arrecadacao federal",
    "IVA substitui impostos indiretos federais",
    "reforma tributária arrecadacao IVA aliquotas",
    "simplificacao impostos aliquotas empresas",
    "reforma federal moderniza sistema tributário"
  ),
  stringsAsFactors = FALSE
)
corp <- ac_corpus(df, text = texto, docid = id) |>
  ac_clean(remove_stopwords = "pt")
freq <- ac_count(corp)

# Motor ggplot moderno (recomendado)
if (requireNamespace("ggwordcloud", quietly = TRUE)) {
  ac_wordcloud(freq, max_words = 30, title = "Termos frequentes")
}

# Motor classico (fallback)
if (requireNamespace("wordcloud", quietly = TRUE)) {
  ac_wordcloud(freq, max_words = 30, backend = "wordcloud")
}
```

`ac_wordscores` *Escalonamento ideológico de documentos pelo método Wordscores*

Description

`ac_wordscores()` posiciona documentos num espectro contínuo (ex.: esquerda-direita, pró/contra uma reforma) a partir só do vocabulário que usam, pelo algoritmo Wordscores de Laver, Benoit & Garry (2003). É um método supervisionado: parte de um pequeno conjunto de documentos de referência com posição conhecida (ex.: manifestos de partidos com posição já estimada por especialistas, ou avaliações de produto com nota conhecida) e estima a posição de todos os outros documentos ("documentos virgens") comparando seu vocabulário ao vocabulário dos documentos de referência.

A lógica: cada palavra recebe um "escore de palavra", a média ponderada das posições dos documentos de referência em que ela aparece (ponderada pela frequência relativa da palavra em cada referência). Um documento virgem recebe a média dos escores de suas palavras, ponderada pela frequência relativa delas no próprio documento virgem: só palavras que também aparecem nos documentos de referência contam.

Fora da ciência política (onde o método nasceu, aplicado a manifestos partidários), a mesma lógica serve para qualquer escala contínua latente: por exemplo, estimar o quão favorável um conjunto de avaliações de produto é a partir de um pequeno conjunto de avaliações com nota de estrelas conhecida, ou posicionar transcrições de atendimento num espectro de satisfação do cliente.

Usage

```
ac_wordscores(
  corpus,
  scores,
  rescale = c("none", "lbg", "mv"),
  drop_punct = TRUE
)
```

Arguments

`corpus`	Objeto `ac_corpus`.
`scores`	Vetor numérico nomeado (nomes = `doc_id`) ou tibble com colunas `doc_id` e `score`, com a posição conhecida dos documentos de referência. Documentos de `corpus` ausentes de `scores` (ou com `score` = `NA`) são tratados como documentos virgens, a serem estimados.
`rescale`	Tipo de reescalonamento aplicado aos escores brutos, para corrigir a compressão em torno da média que o método produz (palavras comuns a todos os textos puxam todo escore para o centro da escala): • `"none"` (padrão): sem reescalonamento, escore bruto. • `"lbg"`: transformação de Laver, Benoit & Garry (2003, p. 316), $S_v^* = (S_v - \bar{S}_v)(DP_r/DP_v) + \bar{S}_v$, em que $\bar{S}_v$ e DP_v são a média e o desvio-padrão dos escores brutos dos **documentos virgens** e DP_r o desvio-padrão dos escores **conhecidos** das referências. Dá aos virgens a mesma dispersão das referências, mantendo a média deles. Exige pelo menos 2 virgens e depende de quais virgens entram: com poucos, fica instável (com exatamente 2, eles caem sempre a cerca de 0,71 desvio-padrão da média, seja qual for o texto). • `"mv"`: transformação de Martin & Vanberg (2008), $S_v^* = (S_v - P_1)/(P_2 - P_1) \cdot (A_2 - A_1) + A_1$, em que A_1 e A_2 são os escores conhecidos das referências extremas (menor e maior) e P_1, P_2 os escores brutos que o método atribui a essas mesmas referências. Não depende do conjunto de virgens: é a opção mais robusta quando há poucos documentos a estimar. A mesma transformação é aplicada também às referências (como se fossem virgens), o que serve de checagem: uma referência bem recuperada fica perto do seu escore conhecido.
`drop_punct`	Lógico, encaminhado para `ac_count()`. Padrão: `TRUE`.

Value

Tibble com colunas `doc_id`, `score` (escore estimado, já reescalonado conforme `rescale`), `score_raw` (escore bruto, antes do reescalonamento) e `is_reference` (`TRUE` para os documentos usados como referência).

References

Laver, M.; Benoit, K.; Garry, J. (2003). Extracting Policy Positions from Political Texts Using Words as Data. *American Political Science Review*, 97(2), 311-331.

Martin, L. W.; Vanberg, G. (2008). A Robust Transformation Procedure for Interpreting Political Text. *Political Analysis*, 16(1), 93-100.

See Also

ac_count(), ac_ca()

Examples

```
# Avaliações de produto com nota conhecida (referência) e sem nota
# (a estimar), usadas como escala continua de satisfacao
df <- data.frame(
  id = paste0("r", 1:6),
  texto = c(
    "produto excelente, superou expectativas, recomendo muito",
    "muito bom, chegou rápido e funciona perfeitamente",
    "produto regular, cumpre o básico",
    "não funcionou bem, qualidade ruim, não recomendo",
    "péssimo, quebrado e com defeito, decepcao total",
    "produto ok, nada de especial mas atende"
  ),
  stringsAsFactors = FALSE
)
corpus <- ac_corpus(df, text = texto, docid = id)

# r1, r2, r4, r5 com nota conhecida (1-5); r3 e r6 sao "virgens"
notas <- c(r1 = 5, r2 = 4, r4 = 2, r5 = 1)

ac_wordscores(corpus, notas)
```

---

as_prompt *Converter codebook em system prompt para LLM*

Description

as_prompt() é um genérico S3 que converte um objeto em system prompt formatado para uso com LLMs. O método as_prompt.ac_codebook() gera o prompt a partir de um ac_codebook, incluindo instruções, categorias, exemplos, pesos e, opcionalmente, raciocínio estruturado.

Usage

```
as_prompt(x, ...)

## Default S3 method:
```

```
as_prompt(x, ...)

## S3 method for class 'ac_codebook'
as_prompt(
  x,
  reasoning = TRUE,
  reasoning_length = c("short", "medium", "detailed"),
  ...
)
```

Arguments

x	Objeto a converter (para as_prompt.ac_codebook: um ac_codebook).
...	Argumentos adicionais passados ao método.
reasoning	Lógico. Se TRUE, inclui campo de raciocínio no JSON de saída. Padrão: TRUE.
reasoning_length	Extensão do raciocínio: "short", "medium" ou "detailed".

Value

String com o system prompt (invisível).

Examples

```
# Codebook base
cb <- ac_qual_codebook(
  name         = "tom",
  instructions = "Classifique o tom do texto.",
  categories   = list(
    formal   = list(definition = "Linguagem técnica e impessoal."),
    informal = list(definition = "Linguagem coloquial ou emotiva.")
  )
)

# Gerar o system prompt para uso direto com objetos Chat do ellmer
prompt <- as_prompt(cb, reasoning = TRUE, reasoning_length = "short")
substr(prompt, 1, 200)  # inspecionar o comeco do prompt
```

---

is_ac_corpus *Verificar se um objeto é um corpus do* acR

Description

Função auxiliar para testar se um objeto pertence à classe ac_corpus. Útil para validação de argumentos em funções do pipeline.

Usage

```
is_ac_corpus(x)
```

Arguments

x	Objeto qualquer a ser testado.

Value

TRUE se x é um objeto de classe ac_corpus, FALSE caso contrário.

Examples

```
# Objeto criado via ac_corpus() sempre tem a classe
corpus <- ac_corpus(c("Texto um.", "Texto dois."))
is_ac_corpus(corpus)         # TRUE

# Estruturas comuns não contam
is_ac_corpus(data.frame())   # FALSE
is_ac_corpus("texto solto")  # FALSE
```

---

theme_ac *Tema visual consistente do acR*

Description

theme_ac() retorna um tema ggplot2 minimalista e consistente, usado por todos os ac_plot_*() do pacote. Deriva de ggplot2::theme_minimal() com ajustes editoriais: tipografia mais compacta, gridlines suaves, títulos em negrito com espaçamento negativo (visual editorial).

Também expõe ac_palette() para uma paleta categórica coerente com o tema (compatível com acessibilidade AA).

Usage

```
theme_ac(base_size = 12, base_family = "")
```

Arguments

base_size	Tamanho base da fonte. Padrão: 12.
base_family	Família tipográfica. Padrão: "" (usa sistema).

Value

Um objeto ggplot2::theme.

See Also

ac_palette()

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  ggplot2::ggplot(mtcars, ggplot2::aes(mpg, wt)) +
    ggplot2::geom_point(color = ac_palette()[1]) +
    ggplot2::labs(title = "MPG vs. peso", subtitle = "Tema editorial acR") +
    theme_ac()
}
```

Index

∗ **corpus**
ac_clean, 8
ac_corpus, 18
ac_count, 20
ac_keyness, 38
ac_plot_keyness, 51
ac_plot_tf_idf, 59
ac_plot_top_terms, 60
ac_tf_idf, 106
ac_tokenize, 107
ac_top_terms, 109
is_ac_corpus, 115
∗ **qualitative**
ac_calibrate_tokens, 6
ac_chat_local, 7
ac_estimate_tokens, 28
ac_qual_code, 66
ac_qual_codebook, 71
ac_qual_codebook_add, 73
ac_qual_codebook_history, 74
ac_qual_codebook_hybrid, 75
ac_qual_codebook_merge, 76
ac_qual_codebook_remove, 78
ac_qual_codebook_translate, 79
ac_qual_export_for_review, 80
ac_qual_import_human, 81
ac_qual_list_models, 85
ac_qual_load_codebook, 86
ac_qual_recommend_model, 87
ac_qual_reliability, 89
ac_qual_report, 91
ac_qual_report_full, 93
ac_qual_sample, 95
ac_qual_save_codebook, 97
ac_qual_search_literature, 98
as_prompt, 114
∗ **quantitative**
ac_build_dictionary, 3
ac_ca, 4
ac_cluster_documents, 12
ac_collocations, 14
ac_cooccurrence, 16
ac_dictionary, 21
ac_dictionary_import, 23
ac_dictionary_save, 24
ac_dictionary_validate, 25
ac_emotion, 26
ac_kwic, 40
ac_lda, 41
ac_lda_tune, 43
ac_lexical_diversity, 45
ac_plot_ca, 47
ac_plot_wordscores, 63
ac_readability, 100
ac_sentiment, 102
ac_temporal, 104
ac_wordscores, 112
∗ **visualization**
ac_palette, 46
ac_plot_cluster, 48
ac_plot_cooccurrence, 49
ac_plot_lda_topics, 52
ac_plot_lda_tune, 53
ac_plot_lexical_diversity, 54
ac_plot_readability, 55
ac_plot_sentiment, 57
ac_plot_temporal, 58
ac_plot_wordcloud_comparative, 62
ac_plot_xray, 64
ac_plot_zipf, 65
ac_wordcloud, 111
theme_ac, 116

ac_build_dictionary, 3
ac_build_dictionary(), *14*, *22–24*
ac_ca, 4
ac_ca(), *47*, *114*
ac_calibrate_tokens, 6
ac_calibrate_tokens(), *28*

ac_chat_local, 7
ac_clean, 8
ac_clean(), *11, 12*, *20*, *37*, *108, 109*
ac_clean_stopwords, 11
ac_clean_stopwords(), *9, 10*
ac_cluster_documents, 12
ac_cluster_documents(), *49*
ac_collocations, 14
ac_cooccurrence, 16
ac_cooccurrence(), *49, 50*
ac_corpus, 18, *35*
ac_corpus(), *9, 10*, *20*, *33*, *37*, *65*, *108, 109*
ac_count, 20
ac_count(), *5*, *15*, *38, 39*, *41*, *51*, *60, 61*, *66*, *105–111*, *113, 114*
ac_dictionary, 21
ac_dictionary(), *3, 4*, *25–27*
ac_dictionary_import, 23
ac_dictionary_import(), *4*, *24*
ac_dictionary_load *(*ac_dictionary_save*)*, 24
ac_dictionary_save, 24
ac_dictionary_save(), *4*, *23*
ac_dictionary_validate, 25
ac_dictionary_validate(), *103*
ac_emotion, 26
ac_emotion(), *25*
ac_estimate_tokens, 28
ac_estimate_tokens(), *6, 7*
ac_export, 29
ac_fetch_camara, 31, *33*, *35*
ac_fetch_senado, 33
ac_import, 36
ac_keyness, 38
ac_keyness(), *15*, *20*, *51*
ac_kwic, 40
ac_lda, 41
ac_lda(), *4, 5*, *12, 13*, *44*, *52, 53*
ac_lda_tune, 43
ac_lda_tune(), *42*, *53, 54*
ac_lexical_diversity, 45
ac_lexical_diversity(), *54, 55*, *101*
ac_palette, 46
ac_palette(), *49*, *62*, *111*, *116, 117*
ac_plot_ca, 47
ac_plot_ca(), *5*
ac_plot_cluster, 48
ac_plot_cluster(), *13*
ac_plot_cooccurrence, 49
ac_plot_cooccurrence(), *17*
ac_plot_keyness, 51
ac_plot_lda_topics, 52
ac_plot_lda_topics(), *42*
ac_plot_lda_tune, 53
ac_plot_lda_tune(), *44*
ac_plot_lexical_diversity, 54
ac_plot_readability, 55
ac_plot_sentiment, 57
ac_plot_sentiment(), *104*
ac_plot_temporal, 58
ac_plot_temporal(), *105*
ac_plot_tf_idf, 59
ac_plot_top_terms, 60
ac_plot_wordcloud_comparative, 62
ac_plot_wordscores, 63
ac_plot_xray, 64
ac_plot_xray(), *41*
ac_plot_zipf, 65
ac_qual_code, 66
ac_qual_code(), *6, 7*, *21, 22*, *28*, *30*, *72*, *82*, *84–86*, *88*, *90*, *92–94*, *96*, *105*
ac_qual_codebook, 71
ac_qual_codebook(), *67*, *74*, *78*
ac_qual_codebook_add, 73
ac_qual_codebook_add(), *78*
ac_qual_codebook_history, 74
ac_qual_codebook_hybrid, 75
ac_qual_codebook_merge, 76
ac_qual_codebook_remove, 78
ac_qual_codebook_remove(), *74*
ac_qual_codebook_translate, 79
ac_qual_export_for_review, 80
ac_qual_import_human, 81
ac_qual_import_human(), *90*
ac_qual_irr, 82
ac_qual_irr(), *30*, *93, 94*
ac_qual_list_models, 85
ac_qual_list_models(), *88*
ac_qual_load_codebook, 86
ac_qual_recommend_model, 87
ac_qual_recommend_model(), *86*
ac_qual_reliability, 89
ac_qual_reliability(), *25*, *67*, *81, 82*, *92–95*
ac_qual_report, 91
ac_qual_report(), *93, 94*

ac_qual_report_full, 93
ac_qual_sample, 95
ac_qual_sample(), *80*, *84*
ac_qual_save_codebook, 97
ac_qual_save_codebook(), *86*
ac_qual_search_literature, 98
ac_readability, 100
ac_readability(), *46*, *55, 56*, *105*
ac_sentiment, 102
ac_sentiment(), *21–23*, *25–27*, *57*, *105*
ac_temporal, 104
ac_temporal(), *58*
ac_tf_idf, 106
ac_tf_idf(), *20*, *39*, *59, 60*, *63*
ac_tokenize, 107
ac_tokenize(), *16, 17*, *20*, *37*, *110*
ac_top_terms, 109
ac_top_terms(), *20*, *39*, *60, 61*, *66*, *107*, *111*
ac_wordcloud, 111
ac_wordcloud(), *20*
ac_wordscores, 112
ac_wordscores(), *63*
as_prompt, 114
attributes(), *70*

base::saveRDS(), *30*

dplyr::count(), *20*

ggplot2, *51*, *59, 60*
ggplot2::coord_flip(), *51*, *59*, *61*

is_ac_corpus, 115

knitr::kable(), *30*

quanteda.textstats::textstat_keyness(), *38*

theme_ac, 116
theme_ac(), *47*
tibble::tibble(), *20*, *38, 39*, *51*, *59*, *61*, *106–108*, *110*
tidytext::bind_tf_idf(), *106*
topicmodels::LDA(), *42, 43*

utils::read.csv(), *23*
utils::write.csv(), *29*

writexl::write_xlsx(), *30*