

Tópicos em Busca e Aprendizado de Máquina com Textos

com exemplos em Python, Java e KNIME

Geraldo Xexéo

18 de Junho de 2025 15:47:08

Copyright © 2022,2023,2024 Geraldo Xexéo .

Todos os direitos reservados.

Esta é um pré-publicação com o texto parcial em trabalho.

Contato com o autor pelo e-mail xexeo@ufrj.br.

Versão A4

SUMÁRIO

I Entendendo o que é Texto	1
1 Introdução	3
1.1 Objetivo	4
2 Textos e Documentos	5
2.1 O que é um texto?	5
2.2 Conceito geral de documento	6
2.3 Conceito específico de documento	8
2.4 A necessidade da compreensão do texto	8
2.5 Estrutura da língua	9
2.6 Camadas de Estudo das Línguas	9
2.7 A escrita	12
2.8 Múltiplos significados do texto	12
Tanto Mar (1975)	14
Tanto Mar (1978)	15
2.9 O descasamento entre a expressão e a interpretação	15
2.10 A codificação da escrita no computador	17
2.11 Distribuição das palavras	18
2.12 Modelos conceituais para documentos	19
2.12.1 Library Reference Model	19
2.12.2 DoCO	20
2.12.3 Dublin Core	20
2.12.4 Descrições orientadas ao formato	21
2.13 Coleções de documentos	21
2.14 Exercícios	22
2.14.1 Perguntas de Múltipla Escolha	22
2.14.2 Perguntas Dissertativas	24
3 Codificação de Caracteres	27
3.1 Conceitos básicos sobre a textos na forma digital	27
3.2 A representação de informação	28
3.3 Da variedade para o ASCII	29
3.4 <i>Code pages</i>	32

3.5	Unicode	33
3.5.1	UTF	33
3.5.2	UTF-8	33
3.6	Normalização de Unicode	34
3.7	O fim de linha	35
3.8	Representações de Unicode em Python	36
3.9	Desafios apresentados pela codificação	37
3.10	Detectando uma <i>code page</i> automaticamente	39
3.11	Lendo com codificações em Python	40
3.12	Tratando a codificação no KNIME	41
3.13	Codificando strings	45
3.14	Codificações por referência	46
3.15	Codificação de binários em texto	47
3.15.1	Base64	48
3.16	Exercícios	48
3.16.1	Questões Objetivas	48
3.16.2	Questões Discursivas	50
3.16.3	Questões Discursivas	50
3.16.4	Questões de Programação	51
4	Arquivos Textuais	53
4.1	Os formatos mais comuns	53
4.2	Formatos de edição	54
4.3	Linguagens de marcação	55
4.3.1	HTML	56
4.4	HTML 5	58
4.4.1	Limpando HTML de arquivos com Python	58
4.4.2	Limpando HTML de arquivos com KNIME	59
4.5	Outros formatos específicos	62
4.6	Formatos de impressão	62
4.6.1	PostScript	62
4.6.2	PDF	62
4.6.3	Extraindo texto do PDF em Python	63
4.6.4	Extraindo texto do PDF em KNIME	64
4.7	XML	67
4.8	Programas Exemplo	69
4.9	Exercícios	70
4.9.1	Questões Objetivas	70
4.9.2	Questões Discursivas	72
4.9.3	Questões de Programação	72
II	Pré-Processamento e Algoritmos Básicos	77
5	Pré-Processamento	79
5.1	Noções de morfologia da palavra	80
5.1.1	Lematização e Stemming	80
5.2	Tokenização	81
5.3	Normalização de maiúsculas e minúsculas	82

5.4	Stemmers	83
5.4.1	Stemmers por Busca em Tabela	84
5.4.2	<i>Porter Stemmer</i>	85
5.4.3	<i>Snowball Stemmer</i>	85
5.4.4	<i>Lancaster Stemmer</i>	85
5.4.5	<i>Stemmers Criados para o Português</i>	86
5.4.6	Usando os <i>Stemmers</i> em Python	87
5.5	Remoção de <i>stopwords</i>	92
5.5.1	Usando o KNIME	97
5.5.2	Problemas com <i>stopwords</i>	97
5.6	Normalização de texto	99
5.6.1	Estratégias gerais de correção	100
5.6.2	Corrigindo textos com algoritmos genéticos	101
5.7	Uma cadeia de pré-processamento em KNIME	103
5.8	Sentenciação	103
5.9	Codificação de Texto	103
5.10	Codificação <i>one hot</i>	103
5.10.1	Mapeamento dos termos em números	104
5.11	Exercícios	104
6	Estruturas de Dados	107
6.1	Lista Invertida	107
6.1.1	Exemplo de listas invertidas	107
6.2	Melhorias na lista invertida	108
6.3	Listas Invertidas em Python	109
6.3.1	Implementação trivial em Python	109
6.4	Caches	110
6.5	Tabelas <i>Hash</i>	110
6.6	Arquivos de assinatura, impressões Digitais e LSH	113
7	Similaridade	115
7.1	Teoria da similaridade baseada em características de Tverski	116
7.1.1	Axiomas de Tverski	117
7.2	Similaridade em conjuntos	117
7.3	Similaridade em vetores	119
7.3.1	Similaridade como inverso da distância	119
7.3.2	Similaridade do cosseno	121
7.3.3	Funções de erro	121
7.4	Similaridade entre strings	122
7.5	Outras medidas de similaridade e distância	123
7.6	Exercícios	123
III	Recuperação de Texto	125
8	Recuperação da Informação	127
8.1	Motivação	127
8.1.1	Escalas do Problema de Busca	127
8.1.2	Como Atender o Usuário da Busca	128

8.1.3	A Recuperação da Informação na História	128
8.2	Tarefas da Recuperação da Informação	129
8.2.1	A Busca Ad-Hoc	130
8.3	Principais problemas	131
8.4	Aboutness	132
8.5	Relevância	133
8.5.1	Atributos da relevância	134
8.5.2	Manifestações de Relevância	135
8.5.3	O Espaço das Relevâncias	137
8.6	Modelo conceitual da Recuperação da Informação	138
8.7	Modelos dos sistemas de Recuperação da Informação	138
8.8	Conceitos clássicos de Recuperação da Informação	140
8.9	Representação do Documento	141
8.9.1	O mundo externo também representa o documento	143
8.10	Os modelos de Recuperação da Informação	143
8.11	Taxonomia de modelos de Recuperação da Informação	144
9	Medidas de Avaliação	147
9.1	Modelo clássico de avaliação	147
9.2	Medidas clássicas de avaliação	148
9.2.1	Problemas com a acurácia na recuperação da informação	150
9.2.2	Outras medidas clássicas	150
9.3	Fatores que governam a precisão e revocação	151
9.4	Relação entre precisão e revocação	152
9.5	Gráfico de precisão em 11 níveis de revocação	153
9.6	Comparando com apenas um valor	157
9.6.1	<i>Mean Average Precision</i>	157
9.6.2	Teste F	157
9.7	DCG	158
10	Representações de Texto	161
10.1	Representação Booleana	161
10.2	Representações Geométricas ou Vetoriais de Documentos	162
10.3	Modelos Probabilísticos e de Linguagem	162
10.4	Embeddings	163
11	Modelos de Linguagem	165
11.1	Fundamentos Probabilísticos e Modelos de <i>n</i> -gramas	165
11.2	Modelos de Linguagem Neurais: do <i>feed-forward</i> ao <i>recurrent</i>	167
11.3	<i>Sequence-to-Sequence</i> e Mecanismos de Atenção	168
11.4	<i>Transformers</i> : Atenção Multi-cabeças e Pré-treinamento em Larga Escala	169
11.5	Aplicações e Considerações Finais	171
12	Embeddings	173
12.1	Representações Discretas vs. Distribuídas	173
12.2	Embeddings de Palavras Não Contextuais	174
12.3	Embeddings Contextuais: ELMo, BERT, GPT e além	176
12.4	Vantagens, Limitações e Desenvolvimentos Atuais	177

13 Modelo Booleano	181
13.1 Pequena História do Início de Tudo	181
13.2 Modelo Booleano	182
13.3 Usando a Lista Invertida	183
13.4 Consultas booleanas	184
13.5 Simulando o Modelo Booleano	184
13.6 Estratégias para busca em longas listas invertidas	185
13.7 Exemplos com o Modelo Booleano	185
13.8 Similaridade de Documentos no Modelo Booleano	191
13.9 Extensões ao Modelo Booleano	191
13.10 Uso Atual	191
13.11 Exercícios	192
14 Representação Vetorial	193
14.1 Definições do Modelo Vetorial	195
14.2 Medindo o peso de um termo em um documento	196
14.2.1 TF, a frequência do termo	196
14.2.2 IDF, a frequência inversa nos documentos	197
14.2.3 TF-IDF	198
14.3 Medindo a similaridade entre documentos	199
14.3.1 Introduzindo a qualidade	199
14.4 Uso atual do Modelo Vetorial	199
14.5 Extensões ao Modelo Vetorial	199
14.5.1 Modelo vetorial generalizado	200
14.6 Expansão de consulta	200
14.6.1 Feedback de relevância	200
14.6.2 Feedback de Pseudo-Relevância	202
15 Redução de Dimensionalidade e Embeddings	205
15.1 Embeddings	205
15.1.1 Topologia e Embeddings	206
15.1.2 Definição Formal	206
15.1.3 Buscando as Variáveis Latentes	206
15.1.4 Variáveis Latentes e a Estrutura dos Dados	207
15.2 Indexação por Semântica Latente	208
15.2.1 A Decomposição em Valores Singulares	209
15.2.2 Quantas dimensões usar no LSI?	210
15.2.3 Exemplo de uso da LSI em Python	211
15.2.4 Considerações sobre o LSI	217
15.3 Modelo Baseado em Sinais	218
15.4 <i>Embeddings</i>	218
15.5 Histórico e Contexto	218
15.6 Abordagens Baseadas em Predição e Contagem	218
15.6.1 Word2Vec	219
15.6.2 GloVe	220
15.7 Modelos Contextuais	220
15.7.1 Detalhamento do Word2Vec	220
15.8 Seq2Seq	220
15.9 <i>Encoder-Decoder</i>	220

15.10 Atenção	220
15.11 Desafios e Perspectivas	221
16 Modelo Probabilístico	223
16.1 Abordagem Geral do Modelo Probabilístico	224
16.1.1 Discussão do valor a ser otimizado	225
16.1.2 Supondo termos independentes	226
16.2 Estimando as probabilidades	227
16.3 Modelo RSJ	228
16.4 BM25	229
16.4.1 BM25	230
16.5 Modelo Probabilístico hoje	230
17 Modelos de Linguagem	231
17.1 Modelos multinomiais	232
17.2 Smoothing	233
17.3 Modelos de aprendizado profundo	233
18 A Web, Além do Texto	235
18.1 Curta Perspectiva Histórica Pré-Web	235
18.1.1 Uso de Citações na Busca	236
18.2 A Web como um Grafo Direcionado	237
18.3 HITS	238
18.4 Pagerank	243
18.5 A Vida Pós-Pagerank	245
18.6 Exercícios	246
19 Mecanismos de Busca	249
19.1 Funcionamento do Google	250
19.1.1 Alguns algoritmos na história da Google	252
19.1.2 Sistemas de Classificação do Google	254
19.2 Apache Solr	254
19.3 Lucene	255
19.3.1 PyLucene	258
19.4 Whoosh	259
19.5 Outras Ferramentas Úteis	261
19.5.1 Terrier	261
19.5.2 Anserini e Pyserini	261
19.5.3 Ainda outras ferramentas disponíveis	261
20 Crawlers	263
20.1 Tipos de <i>crawler</i>	265
20.2 O que não indexar	266
20.2.1 Sitemaps	268
20.3 Qualidade do <i>crawling</i>	268
20.4 Como decidir o próximo link a visitar?	268
20.5 Arquitetura de um <i>crawler</i>	269
20.6 Armadilhas para <i>crawler</i>	270
20.7 Tópicos relacionados ao <i>crawling</i>	271

20.7.1 Scrapping	271
20.7.2 <i>Deep web</i>	271
20.8 Aplicações de <i>crawling</i>	271
20.9 Exemplo de implementação em Python	272
20.9.1 Testando Crawlers	275
20.9.2 Recursos para Crawling em Python	276
20.10 Exercícios	276
IV Recursos	277
21 Recursos Lexicais	279
21.1 Wordnets	279
21.2 Uso do WordNet	284
21.3 WordNet e Prolog	284
22 Conjuntos de Dados Disponíveis	287
22.1 Coleções Clássicas	287
22.2 Coleções TREC	287
22.3 Acesso as coleções	287
V Aprendizado de Máquina	289
23 Introdução ao Aprendizado de Máquina	291
23.1 Visão Geral do Aprendizado de Máquina	294
23.2 Modelos gerativos vs. discriminativos	295
23.3 Tarefas Básicas do Aprendizado de Máquina	296
23.4 Formas de Aprendizado de Máquina	296
23.5 Modelo do Scikit-Learn	296
24 Classificação de Textos	301
24.1 Análise de Sentimentos	301
25 Algoritmos Clássicos de Classificação	303
25.1 Naïve-Bayes	303
25.2 SVM	303
VI Redes Neurais	305
26 Introdução às Redes Neurais	307
26.1 Inspiração Biológica	307
26.2 Os Primeiros Modelos: O Perceptron	307
26.3 Redes Neurais Feedforward e Backpropagation	307
26.3.1 Principais Funções de Ativação	308
26.4 Deep Learning: Redes Profundas	308
26.5 Redes Recorrentes e LSTM	308
26.6 Bibliotecas Python para Redes Neurais	308
26.7 Uma Ponte para os Transformers	309

27 Embeddings	311
27.1 <i>Embeddings</i> não contextuais	311
27.1.1 Detalhamento do Word2Vec	312
27.2 Seq2Seq	313
27.3 <i>Encoder-Decoder</i>	313
28 Avanços com Redes Neurais	315
28.1 LSTM	315
29 Atenção, Transformadores, BERT e Modelos Grandes de Linguagem	317
29.1 Atenção	317
29.2 Transformadores	317
29.3 BERT	317
29.4 LLMs	317
VII Aplicações	319
30 RAG	321
31 Nuvens de Rótulos	323
31.1 Introdução	323
31.1.1 Um pequena questão de nomenclatura	326
31.2 Visão geral das nuvens de rótulos	327
31.2.1 Características das Nuvens de Rótulos	328
31.3 Um modelo formal para nuvens de rótulo	329
31.3.1 Definições iniciais do modelo	330
31.3.2 Introduzindo recursos e contextos	331
31.3.3 Criando campos semânticos	332
31.3.4 Gerando rótulos	333
31.3.5 Criando nuvens de rótulos	334
31.3.6 Um modelo para nuvens de rótulo de sumário	334
31.3.7 Um modelo para nuvens de rótulo diferenciais	335
31.4 Principais Técnicas Utilizadas	336
31.5 Aplicações	338
31.6 GTAGLIB	342
32 Detecção de Plágio	345
32.1 Propriedade intelectual e as leis brasileiras	345
32.2 Classificações para o plágio	346
Apêndices	349
A Tópicos em Python	351
A.1 Dicionários	351
A.2 Strings	352
A.2.1 Índices de Strings	353
A.2.2 Fatias de Strings	354
A.2.3 O operador <code>in</code>	355
A.2.4 Comparando Strings	355

A.2.5	Funções com String	356
A.2.6	Exercícios	356
B	Tópicos Selecionados de Matemática	357
B.1	Autovetores e Autovalores	357
C	Programas Úteis	359
D	Desafio	361
D.1	Processador de Consultas	363
D.2	Gerador de Lista Invertida	363
D.3	Indexador	364
D.4	Buscador	365
D.5	Avaliação	365
D.6	Stemming	366
D.7	Entrega	366
E	Jogo do Significado	367
E.1	Introdução	367
E.2	Componentes	367
E.3	Modo de jogar	367
E.4	Conclusão	368

 LISTA DE FIGURAS

2.1	Primeira forma de interpretar sintaticamente a sentença “O menino viu a mulher de binóculo”, onde o menino tem o binóculo, baseado em (Pagani, 2022)	11
2.2	Segunda forma de interpretar sintaticamente a sentença “O menino viu a mulher de binóculo”, onde a mulher tem o binóculo, baseado em (Pagani, 2022)	11
2.3	Letra de 1975 da música Tanto Mar, de Chico Buarque.	14
2.4	Letra de 1978 da música Tanto Mar, de Chico Buarque.	15
2.5	Descasamento expressão interpretação	16
2.6	Registro fonético da frase <i>I ate an apple in August</i> segundo o <i>International Phonetic Alphabet, para o inglês americano</i>	17
2.7	Distribuição das 10000 primeiras palavras no corpus Gutenberg, fornecido com o NLTK (Bird, Loper e Klein, 2009).	18
2.8	Modelo de Entidades e Relacionamentos para o FRBR Group 1, baseado em (IFLA, 2009)	19
2.9	Arquitetura do DoCO, obtida sob licença Creative Commons Atribuição 3.0, (Shotton e Peroni, 2015)	20
2.10	Canal de Ruído de Shannon	26
3.1	A Pedra de Roseta, que apresenta um decreto de um conselho de sacerdotes estabelecendo o culto ao Faraó Ptolomeu V. Fonte: Wikipedia.	36
3.2	Um <i>Prolog XML</i> indicando a codificação	38
3.3	Configurando a codificação default do KNIME.	42
3.4	Um workflow KNIME só com um módulo, FileReader.	42
3.5	Lendo um arquivo texto como uma sequência de linhas no KNIME. Módulo File Reader.	43
3.6	Configurando a codificação correta do arquivo.	43
3.7	Configurando a codificação errada, a visualização apresenta caracteres estranhos.	44
4.1	Arquivo texto visualizado em um editor de texto simples compatível com UTF-8.	54
4.2	Arquivo texto visualizado em um editor hexadecimal.	55
4.3	Arquivo de marcação <i>LATEX</i> visualizado pelo software Notepad.	56
4.4	A visualização de um arquivo HTML muito simples.	57
4.5	Imagen parcial do HTML usado em uma página pessoal	58
4.6	Linha do tempo do HTML ao HTML 5.	58
4.7	Visão geral da plataforma Web, segundo WHATWG (2021)	59
4.8	Workflow para ler, via URL, e limpar um arquivo HTML	60

4.9 Recuperando uma página via sua URL	60
4.10 Configurando a remoção de tags de markup	61
4.11 Resultado recuperado pelo nó de Leitura.	61
4.12 Resultado pós remoção do markup.	61
4.13 Workflow para ler e transformar um arquivo PDF em um Document do KNIME	64
4.14 Configuração do nós PDF Parser	65
4.15 Configuração do nó Document Viewer	65
4.16 Vendo a lista de documentos lidos	66
4.17 Vendo o detalhe de um documento, principalmente seu conteúdo textual.	66
4.18 Exemplo de fragmento de arquivo xml.	67
 5.1 Classificação algoritmos de fusão, a partir de Frakes e Baeza-Yates (1992), Galvez, MoyaAnegón e Solana (2005) e Amri e Zenkouar (2018). O caminho em amarelo indica os algoritmos de Porter e o RSLP.	84
5.2 Tempo de execução do stemmer de Porter com diferentes tamanhos de cache, potências de 2 de 0 a 524288 entradas, mostrando um limite prático para essa coleção, onde há 66392 formas únicas. Um cache de 32768 foi suficiente para atingir um desempenho próximo do máximo.	92
5.3 Contagem das 50 palavras mais frequentes do corpus Gutenberg, pelo Programa 5.6 . .	93
5.4 Aba Preprocessing do nó Stop Word Filter no KNIME	98
5.5 Aba Filter Options do nó Stop Word Filter no KNIME	98
5.6 Cadeia de preprocessamento em KNIME	103
 7.1 Formas de medir a distância entre dois pontos em um espaço. Em magenta a distância do cosseno, em azul tracejado a distância euclidiana e em laranja duas medidas (que vão resultar no mesmo valor) da distância de Manhattan	121
 8.1 Modelo estratificado da relevância, baseado em Saracevic (2017).	136
8.2 Tipos de relevância, baseado em Saracevic (2017).	136
8.3 Modelo Conceitual da Recuperação da Informação, segundo (Fuhr, 1992).	138
8.4 Um modelo mínimo para os sistemas de Recuperação da informação. Baseado em van Rijsbergen (1979)	139
8.5 Modelo abstrato de um mecanismo de busca	139
8.6 Um modelo simples para um sistema de busca.	140
8.7 Taxonomia da Recuperação da Informação. Fonte: (Baeza-Yates e Ribeiro-Neto, 2011). .	145
 9.1 Sendo C o conjunto com todos os elementos, A o conjunto dos elementos relevantes, e B o conjunto dos elementos recuperados. Então $A \cap B$ é o conjunto dos verdadeiros positivos, $A - B$ é o conjunto dos falsos negativos, $B - A$ é o conjunto dos falsos positivos e $C - A - B$ é o conjunto dos verdadeiros negativos.	149
9.2 Visão abstrata de um curva média de precisão por revocação média. Fonte:(Salton e McGill, 1983)	153
9.3 Gráfico de precisão por revocação para uma consulta (C_1).	154
9.4 Gráfico de precisão por revocação para as consultas C_1 e C_2	155
9.5 Gráfico de precisão por 11 pontos de revocação para as duas consultas e para o mecanismo. .	156
 14.1 A ideia básica do Modelo Vetorial é que os termos do documento, e da consulta, indicam seu significado, e que a similaridade entre a consulta e os documentos da coleção pode ser calculada em um espaço vetorial.	194
14.2 Exemplo de problema com o uso do tamanho dos vetores	195

14.3 Usando o cosseno dos vetores.	195
14.4 Comparação das curvas para \log_{10} e variações de K para TF_K	198
14.5 Interpretação geométrica do feedback de relevância.	202
 15.1 O Rolo Suiço é um exemplo clássico usado para ilustrar desafios em técnicas de embedding e redução de dimensionalidade. Ele representa uma superfície bidimensional que foi enrolada em uma terceira dimensão de maneira que não pode ser desdobrada em um plano sem distorção. O objetivo de usar o Rolo Suiço em técnicas de embedding é demonstrar a capacidade de um algoritmo de desenrolar superfícies curvas enquanto preserva a estrutura local dos dados. Fonte: elaboração do autor, baseada em (J. A. Lee e Verleysen, 2007)	207
15.2 Exemplificando o uso de uma transformação de um objeto representado em um espaço tridimensional para um espaço bidimensional.	207
15.3 Visão abstrata do LSI.	208
15.4 Esquema do SVD. Fonte: (Cohen, 2022)	209
15.5 Esquema do SVD após a redução de <i>rank</i> . Fonte: (Cohen, 2022)	209
15.6 Representação de um elipsóide em 3 dimensões mostrando a diferença de tamanho do objeto entre os eixos.	210
15.7 Vectors reduzidos do exemplo.	214
15.8 Representação dos modos CBOW e Skip-Gram. Fonte: (Bengio et al., 2003)	219
 17.1 Um autômata simples só com dois estados	231
17.2 Um modelo de linguagem de unígrafo, parcialmente descrito como um autômata. Fonte: (Christopher D. Manning, Raghavan e Schütze, 2009a)	232
 18.1 Grafo para um conjunto de páginas web descrito no texto.	238
18.2 Um grafo de co-citação para o mesmo conjunto de páginas (Figura 18.1). Dois nós estão ligados se um mesmo nó apontar para os dois.	238
18.3 A web pode ser vista como um grafo direcionado. Nessa imagens estão marcados os <i>hubs</i> em magenta e autoridades em laranja. Os valores calculados estão na Tabela 18.1.	239
18.4 Ilustração do primeiro passo do algoritmo HITS, aa construção do <i>base set</i> . No início um mecanismo de busca retorna os nós marcados com <i>r</i> , em laranja, o <i>root set</i> . O conjunto é então extendido com todos os nós apontados por eles, marcados com <i>o</i> , em roxo, e por no máximo <i>k</i> nós que apontam para eles, <i>i</i> , em azul.	240
18.5 Grafo com os nós anotados com seu <i>PageRank</i>	245
 19.1 Modelo genérico de um mecanismo de busca moderno, baseado em diversas descrições do funcionamento do Google.	250
19.2 Arquitetura original do Google (redesenho). Fonte: (Brin e Page, 1998)	251
19.3 Visão geral da arquitetura do Solr. Fonte: (Sun, 2019)	255
 20.1 Arquitetura abstrata de um crawler. Fonte: Castillo (2004)	264
20.2 Exemplo de arquivo robot.txt. Fonte: Koster et al. (2022)	267
20.3 Arquitetura do Nutch	270
 21.1 Tela de consulta no site da WordNet	280
21.2 Resposta a consulta "talk", fornecida no site da WordNet.	281
21.3 Visualização da WordNet pelo Visual Wordnet	282
21.4 Visualização da WordNet pelo site Visual Words.	282
21.5 Exemplo de relação de hiperonímia presente no WordNet.	282

21.6 Exemplo da relação de meronímia no WordNet	283
21.7 Visualização, no VisualWord da palavra talk, ligada a vários conceitos, sendo um o verbo talk no sentido de falar, que é "a kind of" Communicate.	283
23.1 Quatro objetivos comuns de aprendizado de máquina.	292
23.2 Workflow KNIME para criação de um tag cloud.	292
23.3 Tag Cloud criado com o modelo da Figura 23.2.	293
23.4 Modelo genérico do aprendizado de máquina, adaptado de (Cherkassky e Mulier, 2007)	294
23.5 Processo básico do aprendizado de máquina usando o Scikit-Learn, modelo em BPMN	297
23.6 Árvore de decisão gerada pelo Programa 23.1.	300
27.1 Representação dos modos CBOW e Skip-Gram. Fonte: (Bengio et al., 2003)	312
31.1 Nuvem de palavras geradas a partir de uma versão do capítulo 2 deste livro, usando o site https://www.wordclouds.com/ , em 9 de abril de 2022. Algumas stop words foram removidas manualmente.	323
31.2 Primeira imagem impressa conhecida de uma nuvem de palavras. Fonte: Ombre Blanches (2012)	324
31.3 Nuvem de rótulos sumário para o conjunto de documentos gerados por uma busca sobre a palavra Jaguar. Fonte: F. F. Morgado (2010)	325
31.4 Nuvem de rótulos diferencial para um documento sobre o animal jaguar, em referência ao conjunto de documentos gerado por uma busca pela palavra jaguar. Fonte: F. F. Morgado (2010)	326
31.5 Duas maneiras de construir uma nuvem de rótulos.	329
31.6 Nuvem ordenada por importância do rótulo.	329
31.7 Modelo UML que descreve como são construídos os atributos de um objeto	330
31.8 Campos semânticos e recursos em UML.	333
31.9 Uma instância de um processo de geração de nuvens de rótulo. Adaptado de: F. F. Morgado (2010).	338
31.10 Relevância de termos para o período 2000-2019. Fonte: B. D. d. Paiva e Pinho (2022)	339
31.11 Relevância de termos para o período 2020-2022. Fonte: B. D. d. Paiva e Pinho (2022)	340
31.12 Nuvem de rótulos de sumário para os títulos das publicações entre 2002 e 2022. Fonte: Boechat e Kuchpil (2022)	341
31.13 Nuvem de rótulos de diferencial para os títulos das publicações entre 2002 a 2004. Boechat e Kuchpil (2022)	341
31.14 Nuvem de rótulos de diferencial para os títulos das publicações entre 2020 a 2022. Boechat e Kuchpil (2022)	341
A.1 As posições de indexação das strings estão na verdade entre as letras	354
D.1 Modelo de processo para o exercício.	362

LISTA DE TABELAS

3.1	Tabela ASCII, ou Unicode com a representação UTF-8	31
3.2	Alguns caracteres Unicode e suas representações UTF-8	34
3.3	Exemplo de como é feita uma codificação Base64	48
5.1	Lista de <i>stopwords</i> do sistema <i>Smart</i> (Salton e McGill, 1983).	94
5.2	Lista de <i>stopwords</i> em português brasileiro obtida em http://www.ranks.nl/stopwords/ . .	95
6.1	Representação de um lista invertida.	108
6.2	Exemplo de inserções em uma tabela de endereçamento direto com endereçamento aberto. Palavras que colidem estão em marcadas.	112
8.1	Fragmento de uma Matriz Termo-Documento para os quatro primeiros parágrafos dessa seção.	141
9.1	Matriz de Confusão	149
9.2	Matriz de Confusão Desbalanceada	150
9.3	Algumas medidas muito usadas, seus sinônimos e fórmulas.	151
9.4	Evolução das medidas de revocação e precisão a cada item analisado da resposta à consulta.	154
9.5	Tabela para construção do gráfico de 11 pontos	156
13.1	Matriz Termo-Documento, ou Matriz de Incidência, simples.	184
14.1	Exemplo de criação passo a passo dos vetores de documentos de uma coleção.	196
18.1	Valor de autoridade e hub para o grafo da Figura 18.3.	239
20.1	Alguns identificadores de agentes	267
A.1	Alguns formas de usar dicionários em Python	352

LISTA DE PROGRAMAS

2.1	Interpretação para o diagrama da Figura 2.1	10
2.2	Interpretação para o diagrama da Figura 2.2	10
3.1	Exemplo de leitura com normalização	35
3.2	Exemplo de leitura com codificação	37
3.3	Exemplo simples de uso do chardet	39
3.4	Exemplo de programa que usa o chardet	39
3.5	Uso do charset-normalizer.	40
3.6	Uso do charset-normalizer com normalização automática.	40
3.7	Comandos para abrir um arquivo escolhendo a codificação	40
3.8	Exemplo de programa que usa o chardet e lê o arquivo	41
3.9	Obtendo a sequência correta da codificação de uma string	45
3.10	Salvando um mesmo arquivo em várias codificações.	45
3.11	Processando entidades HTML	46
4.1	Programa simples para obter todo o texto dentro de um arquivo HTML	59
4.2	Usando o pdfminer.six para extrair texto de um arquivo PDF	63
4.3	Usando uma forma mais complexa de extrair texto de um arquivo PDF	63
4.4	Exemplo de arquivo XML usando um DTD.	68
4.5	Exemplo de arquivo DTD.	68
4.6	Exemplo de arquivo XSD.	68
4.7	Funções de apoio, arquivo birtutils	69
4.8	Lendo arquivos do corpus Folha, resulta em um dicionário com id dos artigos e indicando o texto sem tratamento.	69
5.1	Tokenização com o NLTK	82
5.2	Tokenização com o Gensim	82
5.3	Uso dos stemmers para inglês no NLTK	87
5.4	Uso dos stemmers para português no NLTK	88
5.5	Usando stemmer com cache em Python	90
5.6	Contagem das 50 primeiras palavras do corpus Gutenberg no NLTK	92
5.7	Removendo as <i>stopwords</i> com NLTK	93
5.8	Programa que usa <i>go-words</i> compostas	98
5.9	Corretor simples segundo Norvig (2007)	100
6.1	Criação simples de uma lista invertida	109
7.1	Função para calcular a distância Levenshtein segundo sua definição	122

13.1 Programa simples para fazer consultas do tipo E e OU a uma lista-invertida	184
13.2 Preparação do Exemplo.	185
13.3 Cálculo da uma matriz termo documento.	186
13.4 Leitura de um documento palavra a palavra e registro dos documentos que possuem os termos índices em uma “lista invertida” construída por meio de um dicionário.	187
13.5 Calcula operadores E, OU e NÃO para duas listas.	188
13.6 Transforma um expressão lógica infixa em pós-fixa.	189
13.7 Processa uma busca booleana na coleção em memória usando o documento original. .	190
13.8 Processa uma busca booleana na coleção em memória usando índice.	190
15.1 Declarações	211
18.1 Cálculo do Pagerank	244
19.1 Exemplo do uso do Lucene.	256
19.2 Exemplo do uso do Woosh.	259
20.1 Exemplo de como evitar robots no HTML.	267
20.2 Um crawler simples em Python que respeita um intervalo de 1 a 3 segundos.	272
20.3 Crawler com tratamento da exclusão pelos robot.txt	274
21.1 Programa Simples em Prolog	284
23.1 Programa para árvore de decisão que busca uma maneira de determinar a codificação de um arquivo em português	297
31.1 Fragmento de código com exemplo de uso da GTAGLIB	342
A.1 Obtendo um caractere indexado em uma string	353
A.2 Obtendo um caractere indexado negativo em uma string	353
A.3 Fatiando uma string	354
A.4 Fatias que pulam letras.	354
A.5 Marcando só a posição final.	355
A.6 Cada letra equivale a um número.	355
C.1 Programa que gera os arquivos para o aprendizado da detecção de codificações	359

LISTA DE ALGORITMOS

5.1 Algoritmo geral de programação genética. Fonte (Poli, Langdon e McPhee, 2008)	102
18.1 Algoritmo 1 do HITS, construção do sub-grafo focado	240
18.2 Algoritmo 2 do HITS - atenção a direção de p e q	242
18.3 Algoritmo do PageRank	243
18.4 Algoritmo do BrowseRank. Fonte: (Y. Liu et al., 2008)	246
18.5 Algoritmo Trustrank	246
20.1 Algoritmo básico de um crawler	265

LISTA DE ACRÔNIMOS

ASCII	American Standard Code for Information Interchange
BCD	Binary Coded Decimal
CSS	Cascade Style Sheets
DCG	Discounted Cumulative Gain
DCMI	Dublin Core Metadata Initiative
DoCO	Document Components Ontology
EBCDIC	Extended Binary Coded Interchange Code
FRBR	Functional Requirements for Bibliographic Records
HITS	Hypertext Induced Topic Search
HTML	HyperText Markup Language
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Standard Organizations
LDA	Latent Dirichlet Allocation
LSA/LSI	Latent Semantic Analysis/Indexing
MAP	Mean Average Precision
MARC	Machine Readable Cataloging Record
PRF	Probabilistic Relevance Framework
PRP	Probabilistic Ranking Principle
RFC	Request For Comments
SGML	Standard Generalized Markup Language
SVM	Support Vector Machine
UTF	Unicode Transformation Format
W3C	World Wide Web Consortium
WWW	World Wide Web
XML	eXtensible Markup Language

Parte I

Entendendo o que é Texto

CAPÍTULO 1

INTRODUÇÃO

Este livro foi composto a partir das notas de aula da cadeira de **Busca e Mineração de Texto**, parte do curso de mestrado e doutorado do Programa de Engenharia de Sistemas e Computação da COPPE, o Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa em Engenharia, da Universidade Federal do Rio de Janeiro.

Sua origem é um curso de Busca e Recuperação da Informação, que foi dado por mais de 10 anos, que também olhava outras áreas, como busca de imagens. Porém, com o desenvolvimento crescente em todas áreas, passou a ter o foco em texto, mas não só em busca, já que o interesse da comunidade se abriu para várias outras questões. Entre elas, o autor, e seu orientados, se envolveram principalmente com análise de sentimentos, plágio, várias tarefas de classificação e, mais recentemente, com a aplicação de LLMs.

Tendo em vista essa origem, o livro é escrito para quem tem uma boa formação em Computação, em especial em programação, sendo Python a linguagem de escolha, e com conhecimento básico de tópicos como HTML, XML, internet, etc.

O assunto desse livro é uma interseção entre várias áreas de pesquisa, incluindo, mas não limitada a:

- Processamento de Linguagem Natural;
- Linguística e Linguística Computacional;
- Recuperação da Informação;
- Ciência da Informação;
- Biblioteconomia;
- Aprendizado de Máquina;
- Inteligência Artificial e Computacional;
- Representação do Conhecimento;
- Estatística, e
- Processamento de Texto.

Uma maneira de entender o processamento de texto, ou o processamento de linguagem natural, é por meio das tarefas incluídas nessa área. Em especial, este livro trata das seguintes tarefas:

- Busca ad-hoc, isto é, buscar um conjunto de documentos que responde a uma consulta, a tarefa básica do Google;

- Classificação de documentos, isto é, classificar um conjunto de documentos de acordo com um conjunto de rótulos pré-estabelecidos, por meio do aprendizado de máquina (em construção), e
- Detecção de Plágio, uma tarefa complexa que busca candidatos a plágio em uma primeira etapa e a detecção específica do plágio em uma segunda etapa (em construção).

Outras tarefas possíveis ainda não tratadas diretamente neste livro, mas que podem utilizar as mesmas representações e algoritmos aqui tratados:

- Filtragem, a escolha de documentos ao longo do tempo para uma certa consulta fixa;
- Roteamento, filtragem com ranking;
- Recomendação, seleção de documentos para um usuário a partir de informações sobre o uso do sistema por outros usuários;
- Agrupamento, a organização de um conjunto de documentos em grupos por similaridade;
- Extração da Informação, a extração de informações específicas ou genéricas de um documento ou de um conjunto de documentos, e
- o Reconhecimento de Entidades Nomeadas.

Todas essas tarefas dependem de outros assuntos, que são citados e tratados no livro em diferentes profundidades, como:

- Álgebra Linear e Álgebra Linear Computacional;
- Estruturas de Dados em Memória;
- Estruturas de Dados em Disco;
- Bancos de Dados Relacionais e Não Relacionais;
- Algoritmos de Aprendizado de Máquina;
- Redes Neurais e Redes Neurais Profundas;
- Programação (em Python), e
- Desenvolvimento No-Code e Low-Code.

1.1 Objetivo

O principal motivo deste livro é habilitar o aluno para iniciar uma dissertação de mestrado e doutorado que trate ou utilize de processamento de texto para algumas aplicações:

- Busca e Recuperação da Informação, em especial na tarefa de busca *ad-hoc*;
- Mineração de Texto, em especial nas tarefas de classificação, inclusive análise de sentimento, e
- Identificação de Plágio.

Os objetivos de ensino-aprendizado são capacitar os leitores a:

- conhecer o que é texto e o que é texto em sua forma digital e as formas como é encontrado;
- conhecer as tarefas de busca e mineração de texto;
- entender a dificuldade de cada tarefa;
- conhecer e aplicar os algoritmos básicas para as tarefas principais;
- conhecer e aplicar ferramentas de software típicas para as tarefas principais;
- realizar experimento na área, e
- conhecer e aplicar as medidas de avaliação para as tarefas.

CAPÍTULO 2

TEXTOS E DOCUMENTOS

Este capítulo trata do que é um texto e um documento. As duas primeiras seções tratam de perguntas que parecem ter uma resposta simples, mas na verdade geraram discussões e redefinições ao longo do tempo. “O que é um texto?”, perguntam os linguistas, principalmente, preocupados com sua capacidade de analisá-los e estudá-los. “O que é um documento?”, por sua vez, é uma pergunta clássica na Ciência da Informação, discutida a partir do século XIX. Neste capítulo mostramos algumas das respostas para ambas perguntas, escolhendo aquelas mais adequadas à abordagem deste livro.

2.1 O que é um texto?

Um **texto** pode ser definido de forma muito objetiva, e com uma visão computacional, como uma **sequência estruturada de símbolos convencionados que registra uma informação**, de acordo com uma língua, por meio da escrita. Um texto seria, então, basicamente o que é conhecido, nas linguagens de programação, como uma *string*, mas não qualquer *string*: uma *string* que registra uma informação em uma língua, usando para isso uma forma escrita da mesma.

Este primeiro parágrafo já trata de alguns temas que são importantes no contexto deste livro: um texto, para conter informação, tem significado, que é expresso em uma língua, e as construções da língua são representadas por meio de sequências de caracteres, possivelmente provenientes de mais de um conjunto de símbolos, que podem, por sua vez, assumir várias formas visuais. Este livro é escrito em português, usando caracteres latinos e sua forma é indicada pela fonte “Latin Modern Roman”. Ao longo do texto são usados letras de outros alfabetos, como o α , do grego. Também são usados símbolos adicionais, como vírgulas e operadores matemáticos.

Já para a Linguística, texto se refere a qualquer passagem, falada ou escrita, de qualquer tamanho, que forma um todo unificado(Haliday e Hasan, 1976, pg. 1). Como visto antes, espera-se que um texto tenha um significado. Este livro não trata de registros sonoros de enunciados falados em uma língua, tais registros precisam estar transcritos para serem tratadas pelas técnicas aqui descritas.

Essas duas definições bastam para o trabalho apresentado neste livro, porém é importante citar que, para outros trabalhos, como a análise crítica, definições mais complexas podem ser usadas. Haliday e Hasan (1976, pg. 23), ainda no mesmo capítulo da última definição, redefinem texto como “uma passagem de discurso que é coerente em duas visões: é coerente em respeito ao contexto da situação,

e portanto consistente no registro, e é coerente em respeito a si mesma, e portanto coesiva.”. Como consistência no registro deve ser entendido uma continuidade de significado em relação a situação. Essa definição exige bem mais do texto: que ele tenha um significado e que o significado faça sentido no contexto. Atkins, Clear e Ostler (1992) mostram que uma definição similar, “uma série de sentenças e parágrafos coerentes” falha com uma quantidade de “unidades de linguagem” sobre as quais é preciso trabalhar: pequenos anúncios, artigos de jornais, poemas. Nessa lista é possível adicionar mensagens em redes sociais, por exemplo.

Também é importante chamar atenção que há alguma expectativa, em todo o trabalho com textos, que eles sejam internamente coerentes de alguma forma, porém a importância da coerência, e consequentemente do significado, varia de acordo com a técnica usada e com a expectativa do usuário.

Smiraglia (2001, pg. 3) diz que “Um texto é um conjunto de palavras que constitui uma escrita. Um texto não é o mesmo que um documento, que é o recipiente físico (um item) onde o texto está registrado. Um documento pode só ter um texto, mas um texto pode aparecer em vários documentos”¹. O mesmo autor ainda diz que “Uma obra é um conjunto de ideias criadas provavelmente por um autor ou talvez um compositor, ou outro artista, posta em um documento usando texto, com a intenção de ser comunicada a um receptor (provavelmente um leitor ou ouvinte). Uma obra pode ter muitos textos, e pode aparecer em muitos documentos” (Smiraglia, 2001)².

2.2 Conceito geral de documento

A questão de definir o que é um **documento** parece ser ainda mais complicada que a de definir o que é um texto. Inicialmente após a invenção da impressão por tipos móveis, por Gutemberg, depois com a sistematização da pesquisa científica, e mais atualmente após a internet, os documentos se tornaram cada vez mais importantes e pervasivos na sociedade, servindo de base de pesquisas até campanhas políticas. Organizá-los se tornou uma profissão, que se transformou em uma área de pesquisa (Briet, 1951; Buckland, 1997; Document Academy, 2022).

O que era inicialmente chamado de “Documentação”, como prática ou Ciência, se transformou na “Ciência da Informação” em torno de 1950. Essa área se preocupa, entre outras coisas, com permitir o acesso sistemático a documentos, seu registro e sua descrição. A questão seminal levantada pelos pesquisadores foi a mesma que se faz nessa seção: o que é um documento? Uma explicação frequente foi: “Qualquer expressão do pensamento humano”, já outra forma de pensar é “Qualquer coisa a qual podemos aplicar as técnicas de documentação” (Buckland, 1997).

Originalmente, documentos eram vistos apenas nos registros textuais, como cartas, contratos e livros, porém, com a evolução da Ciência da Informação, e de outras formas de mídia, como fotografias e filmes, houve uma ampliação do conceito. Já no século XIX, pioneiros como Paul Otlet, um dos precursores do hipertexto em rede, discutiam documentos de forma funcional, o que levaria a incluir esculturas, objetos de museu, e até mesmo animais, sendo que documento poderia ser visto como uma “evidência física organizada” (Buckland, 1997).

Em 1937, o Instituto Internacional para Cooperação Intelectual definiu tecnicamente um documento como “Qualquer fonte de informação, em forma material, capaz de ser usada para referência, ou estudo, ou como uma autoridade. Exemplos: manuscritos, material impresso, ilustrações, diagramas, espécimes de museu, etc.”. Essa abordagem, mais generalista, dominou por algum tempo o pensamento dos “documentalistas”, que hoje são “cientistas da informação” (Buckland, 1997).

¹E nós chamamos a atenção que um documento por ter vários textos, como uma coletânea

²Por exemplo, uma mesma obra pode estar em um livro, em um jornal, em um *web site*

A pesquisadora Briet (1951) propôs, por meio de exemplos, o que seria um documento. Segundo ela “Uma estrela é um documento? Um seixo rolado por um rio é um documento? Um animal vivo é um documento? Não. Mas são documentos as fotografias e os catálogos de estrela, as pedras de um museu de mineralogia, os animais catalogados e expostos em um zoológico” (Briet, 1951). Em seu texto, seminal na área, ela explica que a própria criação dos documentos forçou a tarefa de organizá-los, e com isso aparecem os “documentalistas”.

Meyriat (1981) fala de **documentos por intenção**, objetos como livros e jornais, que são criados como documentos, e **documentos por atribuição**, como o antílope exposto de Brie. Sua definição, “O documento pode ser definido como um objeto que suporta a informação, que serve para comunicar e que é durável (a comunicação pode, assim, ser repetida).”(Meyriat, 1981), parece bem adequada. Ainda, segundo Meyriat (1981), “Cada mensagem tem um significado e não se pode definir um documento independentemente do significado da mensagem que ele tem a função de transmitir”.

Como todo objeto pode ser encarregado de suportar informação, a noção de documento é muito mais ampla do que a de escrito (ou texto), e, do mesmo modo, se pode escrever em objetos diferentes, como estátuas (Meyriat, 1981). Isso dá um protagonismo ao receptor da mensagem: é ele que vê a informação no objeto, ao interpretá-lo. Ainda Meyriat (1981) dá o exemplo do jornal usado para enrolar um peixe, que perde, pelo menos temporariamente, sua função de documento, mas que pode recuperá-la mais tarde se alguém o usa para ler uma notícia. Mais ainda, o “documento sempre pode receber novas questões, com a esperança de se obter informações novas em resposta”(Meyriat, 1981). Por exemplo, um texto que indique o horário de saída de um trem antes da invenção dos automóveis teve uma utilidade fugaz, no dia em que o trem partiu, e depois não teve utilidade nenhuma por anos. Muito mais tarde, pode se tornar útil, se perguntas forem feitas sobre o impacto da introdução dos automóveis no uso de trens. Finalmente, Meyriat (1981) conclui que “O objeto próprio da abordagem documentária, ..., é, portanto, a informação”.

Isso leva a concluir que o importante no documento não é o registro físico, como um livro, mas sim a informação associada a eles, e que essa informação depende do desejo e da interpretação tanto do emissor quanto do receptor, sendo transformada pelo tempo, pelo espaço e pelos agentes que agem sobre ela (Meyriat, 1981).

Se tudo que contém informação, ou que pode ser visto como um suporte de informação pode ser um documento, o que diferencia os documentos? A Document Academy (2022) apresenta alguns aspectos que fazem algo ser um documento:

- **Indexalidade**, pois são representações que apontam para ou discutem outras coisas, conceito também chamado de *aboutness*;
- **Pluralidade**, possuindo vários aspectos, como mental, material e social;
- **Estabilidade** (*Fixity*), em sua forma física, sendo um *móvel imutável*, porém a Web trouxe também um grau de fluidez;
- **Documentalidade**, a capacidade de produzir, suportar, permitir, encorajar, permitir, influenciar, tornar possível, proibir... em seus arranjos com outras coisas, e
- **Produtividade**, que permite a construção de outros documentos a partir dele (Document Academy, 2022).

(Document Academy, 2022)

Documentos, porém, como chama atenção Briet (1951), não são apenas textos, mas textos tratados de alguma forma, de maneira que possam ser organizados e catalogados. Isso é feito por meio de dados sobre o documento, ou dados sobre os dados, o que é chamado de **metadado**. Uma mensagem de micromídia social, um livro, notícias de jornal, são documentos não só porque contém texto, mas também porque é possível identificá-los de várias formas, como autor, fonte, data e hora de produção,

etc. Textos, porém, ao contrário de outros objetos, já contém dados que permitem identificá-los: o próprio texto.

2.3 Conceito específico de documento

Especificamente neste livro, o **documento** é a **unidade com que o texto é trabalhado de forma que seu significado é considerado coerente o suficiente para poder ser o resultado de uma busca ou um objeto a ser classificado**. Então, dependendo do sistema, um documento pode ser um livro, um capítulo, um parágrafo, uma linha, um relatório, uma mandado judicial, um *tweet*, a transcrição de uma fala, etc.

Nas tarefas que este livro trata especificamente, é normalmente importante usar um conjunto de documentos, que é chamado de **corpus**³ ou *dataset*, o último sendo um termo típico do aprendizado de máquina. Esse conjunto de documentos pode ser usado como base de dados de um sistema de recuperação de informação, na prática servindo de ruído no processo, atrapalhando a busca⁴, de informação usada para uma tarefa de aprendizado de máquina, etc.

2.4 A necessidade da compreensão do texto

A partir do que foi visto anteriormente, é possível afirmar que todas as tarefas descritas neste livro processam texto, normalmente na forma de linguagem natural. Esses textos estão normalmente organizados em um corpus, que não é nada mais que um conjunto de documentos que passou por uma curadoria que tinha um objetivo. As técnicas aqui descritas podem ser adaptadas para tratar outras formas mais específicas de texto, como programas de computador.

Para esse processamento, o que se busca é alguma representação, possivelmente para comparar representações ou extrair informações, do significado, ou dos múltiplos significados, de um ou mais textos, ou de um texto e uma consulta, ou requisito de informação.

Por exemplo: na busca ad-hoc o objetivo é encontrar um texto que seja relevante a uma necessidade de informação do usuário, na classificação os textos são rotulados como pertencentes a alguma classe, no agrupamento os textos são organizados em grupos por similaridade, na identificação de entidades nomeadas se busca extrair nomes de pessoas, lugares, ou datas que estão em um texto, etc.

Supostamente, todas essas operações devem obter resultados relativos aos vários significados dos documentos, provavelmente dentro de um contexto ou de uma tarefa. Na prática, são usados algoritmos, alguns simples, outros extremamente complexos, sobre representações e descrições dos textos. Mesmo quando complexos, esses algoritmos são muitas vezes usados como caixas-pretas, e a complexidade passa despercebida ao usuário ou mesmo ao desenvolvedor.

Se seres humanos realizassem todas os problemas que serão tratados computacionalmente neste livro, isso implicaria que tivessem a capacidade de processar imensas quantidades de texto com a mesma facilidade que podem processar, ao ler e pensar sobre eles, uma sentença ou um livro. O que os algoritmos aqui descritos tentam é substituir certas capacidades humanas, da mesma forma que programas que realizam cálculo complexos o fazem. A grande diferença entre os problemas aqui tratados e os cálculos matemáticos é que mesmo os seres humanos não são capazes de solucionar a maioria dos problemas aqui descrito de forma perfeita, porque há problemas de compreensão de texto

³Cujjo plural é corpora

⁴A tarefa de busca é muito fácil em uma base com um só documento

até para humanos, ligados não só ao poder cognitivo, mas também ao contexto específico e geral em que o texto é processado, além de algumas tarefas realizadas não terem uma resposta definitiva ou fixa. Com o computador, apesar de termos um grande aumento de velocidade, os problemas de buscar essas respostas aumentam, principalmente pela dificuldade de determinar, de alguma forma, os vários significados e informações fornecidas por um texto ao longo do tempo.

Essa dificuldade já se inicia na própria língua. Por exemplo, a sentença “A manga é amarela” não pode ser compreendida totalmente se não há uma contextualização: ela trata de uma manga como parte de peças de vestuário ou uma fruta?

2.5 Estrutura da língua

A abordagem de “entender o texto” para algumas das tarefas típicas que fazemos com um computador é complicada. Por um tempo, muito do trabalho da Inteligência Artificial tentou criar representações do conhecimento que pudesse ser entendidas, no que é chamado de abordagem simbólica ou simbolista. Um exemplo dessas representações seria criar uma expressão em Prolog como `ver(raposa,uva)` para a frase “a raposa vê a uva”.

Hoje em dia, muito desse trabalho é feito sem que o texto seja realmente compreendido, no sentido em que não é criada uma explicação explícita do significado do texto de forma comprehensível por um ser humano, mas, no lugar disso, são criadas representações que possuem mais ou menos significado, ou são apenas abstrações matemáticas, na forma de redes neurais, no que é chamado de abordagem conexionista. Mesmo sistemas de sucesso, como o Chat-GPT, são baseados na ideia de gerar um texto a partir de outro, e não tem uma representação explícita, que possa ser entendida, do conhecimento que guardam.

Algumas dessas representações levam em conta que grande parte do significado do texto está nas palavras ou nas sequências de palavras, conhecidas como **n-gramas**, mas como será visto mais adiante neste capítulo, isto pode não ser verdade, por causa do contexto e das interpretações que vão além do que está escrito de forma literal e mesmo além da intenção do autor.

Mesmo traduções, cuja abordagem inicial dos pesquisadores era criar um modelo do significado da sentença em uma língua e reescrever esse significado em outra língua, hoje são feitas sem a criação de uma representação de um conhecimento real das línguas, ou seja, sem conhecimento linguístico e do significado das sentenças, mas apenas por aprendizado de máquina, na forma de redes neurais. Se antes havia uma representação intermediária, gerada a partir da interpretação do texto e que permitia a síntese de um novo texto em outra língua, hoje ela não existe, e as regras de tradução estão escondidas dentro da representação matemática distribuída pelos valores dos pesos em uma rede neural.

2.6 Camadas de Estudo das Línguas

A complexidade das línguas, ou das linguagens em geral, pode ser melhor compreendida quando é conhecida a forma como as estudamos e descrevemos: por meio de uma hierarquia formada por camadas que podem precisar interagir entre si. Essas camadas de estudo são Daniel Jurafsky e J. H. Martin, 2008

- o **fonético**, o estudo dos sons da língua;
- o **morfológico**, que estuda partes significativas das palavras;
- o **léxico**, que trata das palavras como um todo;

- o **gramatical** ou **sintático**, que trata das estruturas das sentenças;
- o **semântico**, que trata do significado dessas estruturas;
- o **pragmático**, como a língua é usada para alcançar objetivos, e
- e o de **discurso**, que estuda unidades de texto.

Todos esses níveis podem possuir ambiguidade, tanto na língua falada quanto na língua escrita. Por exemplo, em chinês é necessário descobrir as fronteiras entre as palavras, pois a escrita não possui espaço, já em alemão, mesmo possuindo espaços, algumas palavras precisam ser divididas em suas partes. Outro exemplo é a necessidade de corrigir palavras escritas de forma errada, como em uma mensagem de mídia social, usando para isso o conhecimento da similaridade dos sons representados pela escrita. Um erro comum é a troca de uma letra devido a forma de falar a palavra, como escrever “minino” em vez de “menino”. Há também interações entre vários desses níveis. Em hebraico, como não há vogais, para compreender o significado de uma palavra pode ser necessário primeiro entender como está sendo usada, o que está pelo menos no nível semântico, para então decidir o verdadeiro significado da mesma. Mesmo em português, para identificar o significado da palavra “manga”, é possível analisar as palavras ao redor e procurar dicas em outras palavras, como “camisa”, “árvore” ou “comeu”.

O problema de compreender o texto, para um computador e para um humano, é resolver essas e outras ambiguidades. Como surgiu e como existe no ser humano a capacidade de fazer isso de forma tão sofisticada ainda é um segredo a ser descoberto.

A verdade é que isso não só é difícil, como pode ser impossível até para um humano, principalmente se não há um contexto ao redor. Além disso, as técnicas usadas na Computação, por vezes, aumentam essa ambiguidade. Uma representação comum para indexação de texto, conhecida como **Bag of Words** (**BoW**) considera apenas a presença da palavra no documento, não sua ordem. Assim “Brasil vence Alemanha” tem a mesma representação que “Alemanha vence Brasil”, enquanto seu significado é totalmente diferente.

A questão do significado de “manga”, citada anteriormente, inclui um problema semântico sobre um objeto léxico, mas mesmo que saibamos todos os significados exatos das palavras, podemos ainda ter um problema de entender a sentença.

Por exemplo, na sentença “O menino viu a mulher de binóculo”, quem estava com o binóculo? É fácil entender todos os léxicos, supondo que cada um tenha um significado único, ou escolhendo o significado mais comum, porém podem ser criadas, no mínimo, duas estruturas gramaticais distintas (sintaxe), cada uma com um significado (semântica) para o seu todo.

A primeira representação indica que o menino usava o binóculo para ver a mulher, como na Figura 2.1, representada semanticamente no Programa 2.1.

Programa 2.1: Interpretação para o diagrama da Figura 2.1

```

1 /* ----- Quem tem o binóculo é o menino ----- */
2 existe(menino).
3 existe(mulher).
4 existe(binóculo).
5 possui(menino,binóculo).
6 acao(menino,ver,mulher).
7 acao(menino,usar,binóculo).

```

Outra interpretação é que o menino viu uma mulher que tinha um binóculo, como na Figura 2.2, representado semanticamente no Programa 2.2.

Programa 2.2: Interpretação para o diagrama da Figura 2.2

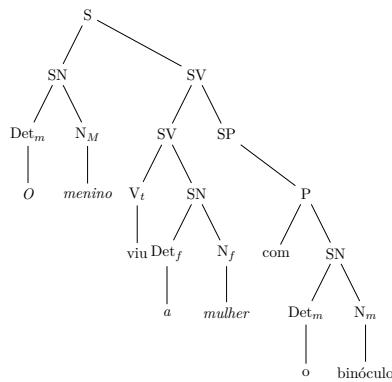


Figura 2.1: Primeira forma de interpretar sintaticamente a sentença “O menino viu a mulher de binóculo”, onde o menino tem o binóculo, baseado em (Pagani, 2022)

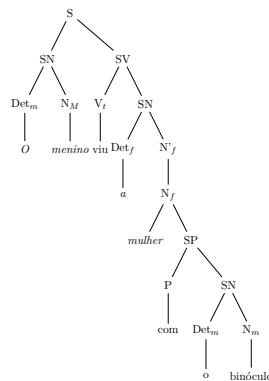


Figura 2.2: Segunda forma de interpretar sintaticamente a sentença “O menino viu a mulher de binóculo”, onde a mulher tem o binóculo, baseado em (Pagani, 2022)

```

1 /* ----- Quem tem o binóculo é a mulher ----- */
2 existe(menino).
3 existe(mulher).
4 existe(binóculo).
5 acao(menino,ver,mulher).
6 acao(mulher,carregar,binóculo).
  
```

Um exemplo de pragmática interessante é a prática de “pedir oferecendo”. Nessa caso, se um convidado fala para o dono de uma casa “Você não gostaria de um café?”, pode ser que seu desejo seja que o outro faça um café para ele. Toda a interpretação da frase leva a um leitor desavisado a interpretar que o emissor está oferecendo o café, porém no contexto específico, por exemplo, saber que a pessoa que oferece não sabe fazer café, ou não seria adequado fazê-lo já que é um visitante, ao nível da pragmática, levaria a deduzir que é uma frase que tem como objetivo que o receptor faça o café, possivelmente para ambos.

Assim, antes de ser complicada quando escrita, a língua é complicada *per si*. Neste livro, várias complicações do nível fonético, que exigem bom poder computacional, como a da identificação dos fonemas, a separação das palavras faladas, as variações regionais, os erros de pronúncia e outras, são evitadas, já que tratamos apenas do texto escrito, que oferece os seus próprios desafios.

2.7 A escrita

Segundo Rogers (2005, pg. 2), a escrita é “o uso de marcas gráficas para representar enunciados linguísticos específicos”. Ainda o mesmo autor, logo a seguir, diz que “A língua é um sistema complexo que reside em nosso cérebro e permite produzir e interpretar enunciados” (Rogers, 2005, pg. 2). Logo, o escrita, e por consequência o texto, não é a língua, mas uma forma de representar mensagens na língua.

Chomsky (2002) afirmou que uma língua natural, como o português ou o inglês⁵ é uma linguagem no sentido de ser “um conjunto (finito ou infinito) de sentenças, cada uma finita em comprimento e construída a partir de um conjunto finito de elementos”. Como disse Rogers (2005), uma pessoa, para se comunicar, possui, de forma intuitiva, as regras de formação dessa linguagem, ou seja, de sua gramática (Chomsky, 2002). A escrita torna esse enunciado, na forma de um texto, visível (ou invés de audível) e permite que seja registrado de certas maneiras (Rogers, 2005).

Assim, a escrita representa a língua, e apenas a língua, como forma de comunicação (Rogers, 2005). Na escrita ocidental moderna, esses símbolos (o **alfabeto**) são letras que se compõe em palavras, que formam sentenças, que formam documentos. Estágios intermediários, como estruturas gramaticais da sentença e organização dos documento em parágrafos, seções e capítulos também são possíveis. Para isso são necessários sistemas de escrita.

Um **sistema de escrita** é uma forma de comunicação por símbolos, que registram uma língua falada por **caracteres** ou **grafemas**. Para entender melhor o que é um sistema de escrita, é possível adotar uma classificação proposta por Daniels (2003):

- **logográficos**, onde os grafemas são ideogramas ou pictogramas que denotam **morfemas**, as unidades indivisíveis da língua, como o chinês;
- **silábicos**, onde os grafemas representam sílabas, como o Katakana para japonês;
- **abjads**, onde os grafemas representam consoantes com uma vogal inerente e variações são indicadas por diacríticos, que são secundários e não são dominantes, muitas vezes sendo abandonados na escrita culta, como o Hebraico;
- **abugidas**, ou semi-alfabéticos, onde os grafemas são consoantes e as vogais são representadas por diacríticos, como o Devanagari,;
- **alfabéticos**, onde os grafemas representam consoantes e vogais, como os alfabetos Latino, Cirílico e Grego;
- **trálicos (featural)**, onde cada parte de um símbolo representa um traço fonético, como o Coreano.
- **sistemas mistos**.

(Daniels, 2003)

2.8 Múltiplos significados do texto

Um texto pode ter muitos significados, pois sua compreensão depende do autor, do leitor, de seus contextos e de um contexto compartilhado por ambos. Já vimos que isso acontece por meio de interpretações diferenciadas de sentenças, devido a questões lexicais, gramaticais, etc. Porém podem existir mais significados, tanto para questões específicas, como o significado de uma palavra, como para o texto em geral.

Ditados populares, por exemplo, possuem um sentido denotativo, literal, e um conotativo, não literal. Quando falamos “Água mole em pedra dura, tanto bate até que fura”, estamos falando que a água desgasta as pedras, porém o que queremos dizer é que a persistência faz com que alcancemos o

⁵Em inglês, uma *natural language*, mas acredito que a melhor tradução é língua natural, pois em português língua e linguagem são conceitos diferentes que são traduzidos para a mesma palavra em inglês: language.

que parece impossível (Louro, 2007). Usado em um contexto específico, porém, o significado conotativo pode ser mais específico, como “pediu tanto que conseguiu”. Além disso, na geologia, a sentença pode ser tomada pelo que diz, literalmente.

O autor e o leitor podem ser influenciados por várias estruturas psico-cognitivas, como sua intenção, sua capacidade de tradução dessa intenção em texto, sua capacidade de interpretação, seu conhecimento da língua, etc. O autor, por exemplo, pode querer passar sua mensagem de uma forma não literal, por meio de uma linguagem figurada. E o leitor pode, também, procurar interpretar outras coisas além do texto, como o estado mental do autor.

Ainda quanto ao leitor, sua interpretação vai levar em conta a tarefa que está realizando ao ler o texto, seus conhecimentos, seu estado cognitivo e mesmo suas emoções. Por exemplo, um mesmo texto jornalístico sobre o governo pode ser entendido por diferentes leitores como uma descrição de fatos, um libelo de oposição, uma provocação aos governantes, ou, mais tarde na história, um exemplo de palavras usadas em uma época específica.

Os significados se multiplicam entre o autor e o leitor porque pode existir uma linguagem figurada, porque a leitura pode ser feita com outra intenção, porque o leitor não entende a ironia ou outra figura de linguagem, porque referências temporais se perderam, etc.

A letra da música Tanto Mar de Chico Buarque de Holanda. (Figura 2.3), em sua primeira versão, de 1975, oferece uma oportunidade de interpretação. Seu texto aparente fala sobre uma festa que ocorreu quando o Eu-lírico estava ausente e doente em outro lugar.

Um leitor atual sem o conhecimento histórico, ou ingênuo na época, pode imaginar que Chico Buarque realmente fala de uma festa de um amigo distante que foi decorada com cravos, a qual talvez não tenha ido por estar doente. Estudando o contexto da época, fica claro que é uma homenagem a “Revolução dos Cravos”, que derrubou, em Portugal, o ditador Marcelo Caetano, herdeiro político de Salazar, e que ao mesmo tempo declara que as coisas não vão bem no Brasil, em pleno governo Médici, da ditadura militar (Neto, 2017).

Há uma mensagem original do autor? Com o conhecimento que temos do contexto em que esse texto foi criado, ela pode ser descrita como o Eu-lírico afirmando que “Fico feliz porque o povo derrubou o ditador em Portugal, mas aqui no Brasil ainda temos uma ditadura e precisamos de alguma inspiração ou apoio para derrubá-la”. Com nosso conhecimento histórico, o contexto, podemos dizer que essa era a opinião do autor, o que pode nem sempre ser verdade em relação ao Eu-lírico. Sabendo que a música foi censurada, também entendemos que ficou claro para os censores esse significado. Mas tarde, em 1978, Chico Buarque reescreve a letra para atualizá-la de acordo com o sentimento que tinha na época, três anos depois da Revolução dos Cravos e com certa desesperança de que pudesse acontecer algo semelhante no Brasil⁶(Neto, 2017).

Um outro leitor pode não estar interessado no significado da letra como uma declaração de apoio à revolução, mas sim na capacidade dos censores em perceber mensagens quase que disfarçadas dentro das músicas. Um terceiro leitor pode querer entender quais são as referências à cultura portuguesa. Outro leitor pode estar buscando inspiração para rimas ou tentando entender a poética do autor.

Certos detalhes que dão mais significado dependem de conhecimentos prévios, como o uso de “pá”, típico de Portugal, o uso da segunda pessoa do singular, que não é comum no Rio de Janeiro onde vive Chico Buarque, e a citação à cultura portuguesa que acontece em “Sei também quanto é preciso,

⁶A música pode ser escutada, também na versão de 1978, Figura 2.4, em <https://www.youtube.com/watch?v=Pj5VuYSmd4k>

Tanto Mar (1975)

Chico Buarque de Holanda

Sei que estás em festa, pá
 Fico contente
 E enquanto estou ausente
 Guarda um cravo para mim

Eu queria estar na festa, pá
 Com a tua gente
 E colher pessoalmente
 Uma flor do teu jardim

Sei que há léguas a nos separar
 Tanto mar, tanto mar
 Sei também quanto é preciso, pá
 Navegar, navegar

Lá faz primavera, pá
 Cá estou doente
 Manda urgentemente
 Algum cheirinho de alecrim!

Figura 2.3: Letra de 1975 da música Tanto Mar, de Chico Buarque.

pá, Navegar, navegar”⁷. Todas esses significados estão além do texto em si, e ligam o verdadeiro significado a Portugal. Há um detalhe a mais na música que se refere a Portugal, e que não aparece no texto: ela é um fado, mesmo que feito por um brasileiro⁸.

Vamos supor agora que um pesquisador queira saber se houve influência da Revolução dos Cravos na política ou na cultura brasileira. Certamente essa música é uma prova que alguma coisa aconteceu. Porém, como a mensagem não é passada de forma direta, uma busca pelos termos Salazarismo e Revolução dos Cravos não chegará a ela. Uma busca feita no Google, em 30 de junho 2020, por cravos encontrou 95 artigos seguidos sobre o problema dermatológico até a primeira entrada sobre outro assunto, que é sobre um livro infantil. Apenas a entrada 101 apontou um livro sobre a Revolução dos Cravos, que começa citando a música de Chico Buarque. Se tivesse a paciência para chegar nessa entrada, o pesquisador poderia então buscar a música completa usando palavras chave retiradas desse texto.

⁷Referência a frase “navegar é preciso, viver não é preciso”, famosa em língua portuguesa por estar no quase-fado “Os Argonautas” (Veloso, 1969), do artista brasileiro Caetano Veloso, como referência a texto do “O Livro do Desassossego”, de Bernardo Soares, heterônimo de Fernando Pessoa, mas que na verdade é original do general romano Pompeu, conforme Plutarco. No caso, a frase foi dita por Pompeu para convencer os tripulantes a levar suprimento a Roma, que vivia forte crise (navegar é preciso), em face aos riscos da navegação da época (viver não é preciso). Essa frase, então, pode ser associada à necessidade de mudar o Brasil da época, a custa da própria vida. Isso é um exemplo da enorme quantidade de conhecimento que pode estar associada a 8 palavras em uma música (Neto, 2017).

⁸O leitor pode procurar a versão final da musica, lançada apenas em 1978, onde a revolução dos cravos já tinha “murchado”

Tanto Mar (1978)

Chico Buarque de Holanda

Foi bonita a festa, pá
 Fiquei contente
 Ainda guardo renitente
 Um velho cravo para mim

Já murcharam tua festa, pá
 Mas certamente
 Esqueceram uma semente
 Em algum canto de jardim

Sei que há léguas a nos separar
 Tanto mar, tanto mar
 Sei também quanto é preciso, pá
 Navegar, navegar

Canta a primavera, pá
 Cá estou carente
 Manda novamente
 Algum cheirinho de alecrim

Figura 2.4: Letra de 1978 da música *Tanto Mar*, de Chico Buarque.

Um exemplo clássico de dificuldade de interpretação é o livro *Flatland: A Romance of Many Dimensions*, cuja primeira edição é de 1884, que é reconhecido pela maioria dos seus leitores como uma sátira social à posição da mulher na sociedade, mas por vezes acusado de ser um livro misógino, que defende uma posição inferior, ao ponto do autor ter sentido necessidade de se explicar em sua segunda edição. Seu texto, porém, fala sobre um mundo habitado por quadrados, círculos e linhas. Sua classificação, direta do texto (*full-text*), dificilmente apresentará relação com o tema “relações sociais no século XIX”, já que é, na superfície do texto, uma fantasia.

É claro que textos técnicos de Ciências Exatas têm menos interpretações possíveis que textos narrativos ou líricos, porém ainda assim há complicações. Um mesmo problema, ou técnica de solução, por exemplo, pode ser conhecido por vários nomes⁹.

2.9 O descasamento entre a expressão e a interpretação

A figura 2.5 busca mostrar que existe um descasamento entre a ideia original de um autor e a compreensão, ou interpretação, que o leitor tem sobre o documento produzido pelo primeiro, principalmente quando esse caminho é mediado por um sistema de busca.

Esse descasamento é aumentado pelo ruído inserido pelo processo de registro, indexação e busca, apesar de já existir pelo próprio processamento cognitivo do emissor e do receptor da mensagem.

⁹Já passei bom tempo tentando achar um nome para um tipo de problema que atingi, até que descobri que o problema já tinha sido publicado com seis nomes diferentes.

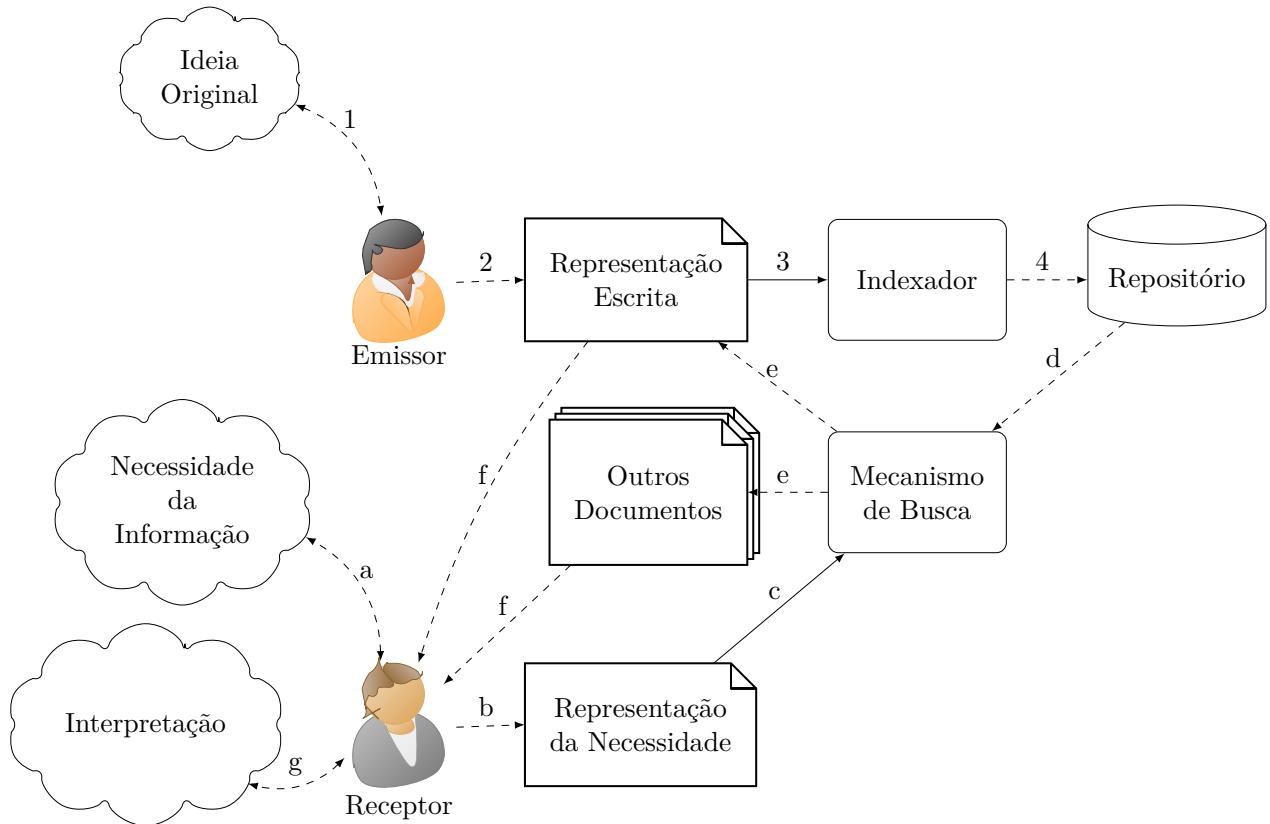


Figura 2.5: O descasamento entre a expressão e a interpretação de uma ideia. As linhas tracejadas indicam traduções onde há ruído.

Na figura há um autor ou emissor, que possui uma ideia original. Ele precisa contar essa ideia, e o faz por meio de uma língua. Nesse ponto já existe o que podemos chamar de ruídos no processo, pois nem toda língua possui palavras para todas as ideias. Por exemplo, um escritor brasileiro terá dificuldade de escrever em inglês exatamente o que entendemos por saudade, ou teria dificuldade de descrever em português o conceito alemão de *schadenfreude*, que significa a experiência de prazer causada pela má fortuna de outra pessoa, como rir de alguém que escorrega na casca de banana. Este texto, escrito, é representado de alguma forma que permite sua indexação ou classificação uma base de dados. É possível também imaginar que o próprio autor não sabe muito bem como expressar sua ideia, ou também não sabe ou não quer escrever na norma culta, o que é comum nas redes sociais.

Na parte inferior da figura há outra pessoa, que tem um desejo por alguma informação. Esse desejo pode ser vago, ou mesmo um total desconhecimento do que precisa realmente saber. Ela tem então que expressar esse desconhecimento ou desejo na forma de uma consulta a um sistema de busca e recuperação, que é uma interface com a base de dados. Nessa expressão ela faz suposições sobre o comportamento do sistema e tenta representar, da melhor forma possível, seu desejo de informação. Hoje em dia na maioria das vezes esse desejo é representado como uma ou poucas palavras enviadas a um mecanismo geral de busca como o Google.

O sistema retorna (outras) representações, como o título ou um pequeno resumo, das representações dos documentos que encontrou na base. O usuário, a seguir, deve selecionar entre as possivelmente várias sugestões do sistema, para poder ler o texto original do autor, e dar sua interpretação. Tanto na determinação da consulta, na escolha entre as representações apresentadas do documento, quanto na interpretação do documento propriamente dito, existe a interferência de vários fatores, contextuais,

cognitivos ou outros, que podem de uma forma geral serem considerados ruídos na comunicação entre o autor (emissor) e o leitor (receptor).

2.10 A codificação da escrita no computador

A escrita é um processo de codificação. Em relação a língua falada, há uma correspondência não muito precisa entre letras, modificadas ou não, e sequência de letras, aos sons que proferimos. Na prática isso hoje obriga os linguistas a ter um conjunto próprio de símbolos para representar os fonemas¹⁰, sendo o mais usado o Alfabeto Fonético Internacional (International Phonetic Association, 1999). Um exemplo disso pode ser visto na frase em inglês *I ate an apple in August*, onde as vogais representam fonemas diferentes, sendo que às vezes uma vogal representa dois fonemas e outras vezes duas vogais representam um fonema. A figura 2.6 exemplifica a variação de sons de vogais na língua inglesa falada nos EUA.

I ate an apple in August.
 aɪ eɪt ən 'æpəl ɪn 'aɡəst .

Figura 2.6: Registro fonético da frase *I ate an apple in August* segundo o *International Phonetic Alphabet*, para o inglês americano

As letras podem ser escritas de várias formas no papel. Em geral diferenciamos amplamente em letra cursiva e letra de forma, mas existe uma variedade de formas pessoais de escrever as letras cursivas e de forma, além de fontes criadas por designers que seguem essas características, antes mesmo da invenção do computador.

Quando um texto é colocado no computador ele deve ser codificado de alguma maneira. Uma alternativa seria tirar um retrato do texto escrito (*scan*) e guardar o texto na forma de uma imagem. O problema é que processar essa imagem para que o computador possa tratar o texto propriamente dito é um processo muito difícil. Além disso, imagens gastam muita memória e não tornariam a comunicação entre computador e pessoas fácil.

Assim, desde o começo da Computação, letras, e espaços, foram codificadas em bits, que são guardados na memória do computador. Essa codificação é inspirada em antigas codificações para comunicação, como o código Morse, que usa traços, pontos e espaços, o Baudot, que usa 5 bits, ou mesmo pelo código Braille.

No momento, nos computadores e na maior parte das aplicações, vivemos uma fase final de transição entre o ASCII, um padrão fortíssimo no passado e ainda existente em documentos, antigos ou novos, na forma original ou em suas extensões, e um novo padrão, o Unicode, que engloba o ASCII, sendo seu formato UTF-8 o mais comum.

Além disso, os documentos mais sofisticados, como os visualizados em processadores de texto como o Word, ou em arquivos PDF, são codificados de forma adicional para indicar sua representação. Por exemplo, um documento com os caracteres “ABCD” pode ter um código dizendo que eles devem ser apresentados ao usuário, ou impressos, com a fonte Arial, corpo 12. Essa informação é normalmente adicionada ao mesmo arquivo (ou seja, documento), na forma de marcações especiais.

O importante é, então, notar que existe um texto escrito, que é **representado** no computador na forma de uma sequência de bits por um processo de codificação. De acordo com o tipo de arquivo,

¹⁰Menores unidades sonoras que formam as palavras de uma língua (Araújo, 2022)

esse texto pode ser entremeado com informações para a sua visualização, impressão, indexação (na forma de metadados), ou outras. Isso é um fator complicador para as máquinas, que para tratar o texto precisam limpar tudo que não é o texto. Além disso, algumas dessas marcações podem gerar algum contexto adicional implícito, como o uso de negrito e itálico¹¹.

A representação de letras e o formato de documentos serão tratados de forma mais detalhada em outros capítulos.

2.11 Distribuição das palavras

Como qualquer um pode reconhecer, as palavras são usadas com frequência diferente na língua e nos documentos de uma língua. Algumas palavras são muito comuns, como “não” e outras raras, como “filaucioso”, um sinônimo de presunçoso. Em inglês, no Brown Corpus, a palavra “the” sozinha representa 7% de todas as palavras.

George Kingsley Zipf, um professor de linguística de Harvard, propôs o que é hoje conhecido como a Lei de Zipf: a frequência de ocorrência de algum evento P em função do seu rank n , quando o rank é determinado pela frequência de ocorrência, segue uma lei de potência $P_n \approx \frac{1}{n^a}$ com a perto da unidade.

Essa propriedade pode ser verificada observando a distribuição das 10.000 palavras mais frequentes presentes no corpus Gutenberg, fornecido com o NLTK (Bird, Loper e Klein, 2009), apresentada na Figura 2.7.

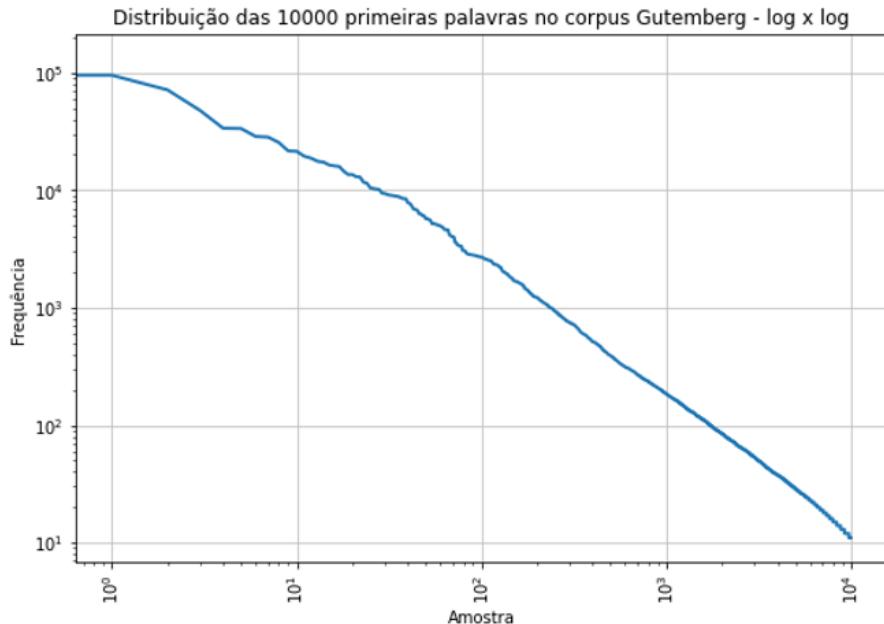


Figura 2.7: Distribuição das 10000 primeiras palavras no corpus Gutenberg, fornecido com o NLTK (Bird, Loper e Klein, 2009).

¹¹Neste livro o negrito é usado para marcar palavras que estão sendo definidas, formal ou informalmente, naquele parágrafo.

2.12 Modelos conceituais para documentos

Nesta seção são apresentados modelos conceituais, isto é, formas de entender um documento. Esses modelos são descritos de várias maneiras, modelos de entidades e relacionamentos, ontologias, coleções de metadados, sendo que algumas vezes o foco é o modelo em si, em uma descrição ontológica do que é o documento, outras vezes é uma forma de trocar informação, onde a definição do formato é mais importante.

2.12.1 Library Reference Model

Originalmente chamado de “Requisitos Funcionais para Registros Bibliográficos”, **FRBR**, é um modelo de entidades e relacionamentos que “fornece um arcabouço estruturado, claramente definido, para relacionar dados que são registrados em registros bibliográficos e para as necessidades dos usuários desses registros” (IFLA, 2009). É, na prática, uma definição formal de termos usados de forma arbitrária anteriormente (Tillett, 2004).

Hoje o FRBR atualizado, é parte do *IFLA Library Reference Model*, LRM, que inclui também atualizações de outros dois modelos, o FRAD, *Functional Requirements for Authority Data* e o FRSAD, o *Functional Requirements for Subject Authority Data*(Riva, Buf e umer, 2017).

As entidades do LRM estão divididas em grupos. O Grupo 1, exposto na Figura 2.8 esclarece os termos Obra, Expressão, Manifestação e Item. A ideia é que Obras são realizadas por Expressões que estão incorporadas em Manifestações, que estão exemplificadas em Itens. Um conto, por exemplo, é uma obra, que pode ser expressa na forma de um texto, ou um áudio, que podem ser publicados sozinhos, ou incorporados em outra manifestação, como uma coletânea, e que são manipulados com unidades específicas, como exemplares de livros ou CDs.

Deve ficar claro que sempre trabalhamos com itens, porém o que miramos é normalmente o significado da Obra.

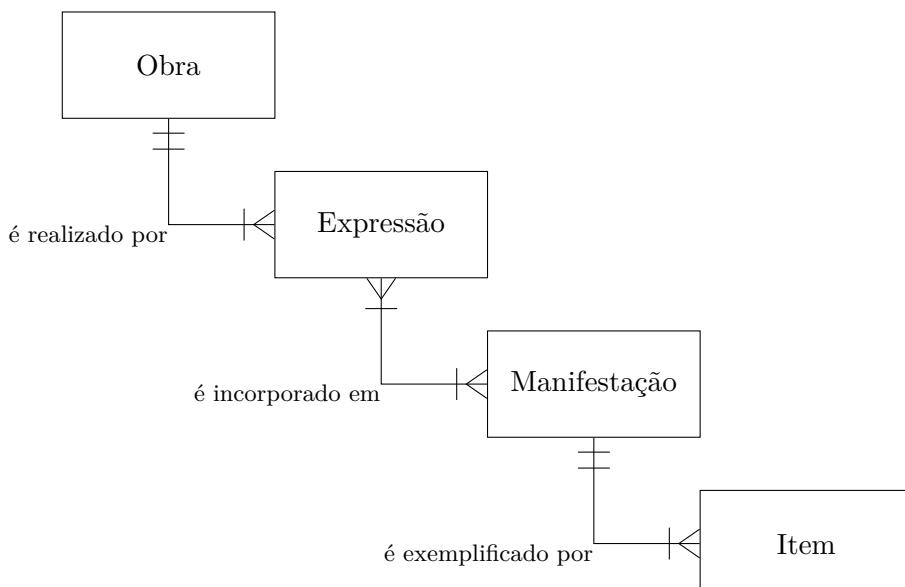


Figura 2.8: Modelo de Entidades e Relacionamentos para o FRBR Group 1, baseado em (IFLA, 2009)

O Grupo 2 discute responsabilidades, e apresenta duas entidades: Pessoa e Corpo Corporativo. A ideia é que uma Obra é criada por uma ou mais entidades do Grupo 2, que por sua vez pode realizar expressões, produzir manifestações e possuir itens(IFLA, 2009).

O Grupo 3 também apresenta novas entidades: Conceito, Objeto, Evento ou Lugar. Ele discute relacionamentos da Obra. Uma Obra, então, tem como assunto entidades dos três grupos(IFLA, 2009).

O modelo também propõe atributos. Uma obra, por exemplo, possui os atributos: título, forma, data, outras características de distinção, contexto, ... (IFLA, 2009). Não cabe a este texto desctrinhar o LRM, mas algumas observações são importantes.

A primeira é que todo o trabalho de processamento é feito sobre itens, na verdade, representações de itens na forma digital. Porém, toda o resultado visa tratar das obras e de seus múltiplos significados. A segunda, de ordem mais prática, é que o LRM vê o documento, ou obra, como um todo, mas outras formas de representação, como o DoCO, podem ver partes menores.

2.12.2 DoCO

DoCO é a **Document Components Ontology**, uma ontologia de que “fornece um vocabulário escrito estruturado sobre components de documentos, sendo eles estruturais (como blocos, parágrafos, seções) ou retóricos (introdução, discussão, figura)”(Constantin et al., 2016). É composto de três partes: *Document Components*, *Discourse Elements Ontology* e *Pattern Ontology*(Constantin et al., 2016). Sua arquitetura é descrita na Figura 2.9.

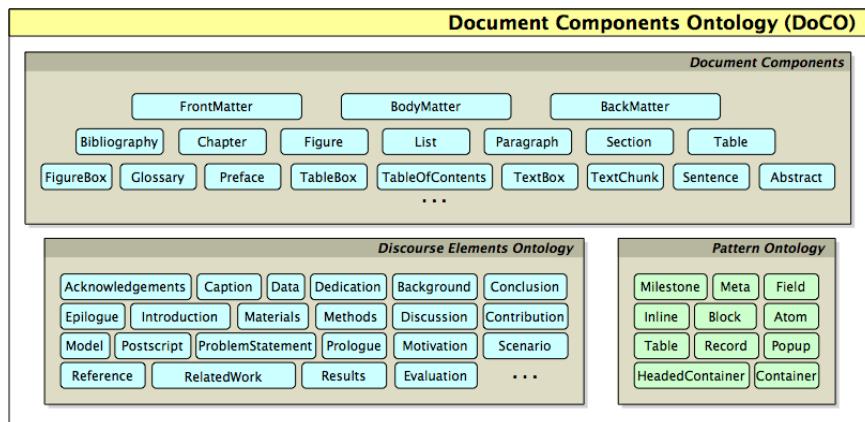


Figura 2.9: Arquitetura do DoCO, obtida sob licença Creative Commons Atribuição 3.0, (Shotton e Peroni, 2015)

Fica claro que DoCO, ao contrário do LRM, permite discutir a estrutura do documento de pelo menos duas formas: pelos componentes do documento e pelos elementos de discurso.

2.12.3 Dublin Core

DCMI é o acrônimo do **Dublin Core Metadata Initiative**, uma “organização que suporta a inovação em *design* de metadados e melhores práticas na ecologia de metadados”(DCMI, 2022a). Sua origem, e missão atual, é manter o Dublin Core, um padrão de termos de meta-dados criado para a Web em 1995 (DCMI, 2022c), e que hoje é parte de diferentes padrões, como o ANSI/NISO Z39.85-2021 (NISO, 2013). Essa organização mantém diferentes formas de metadados, isto é, dados que

descrevem documentos, entre elas o *Dublin Core* (DCMI, 2022c), que hoje é composto, entre outros documentos, do *DCMI Metadata Terms* (DCMI, 2022b).

Os principais termos definidos no *Dublin Core* (NISO, 2013) são: *title, creator, subject, description, publisher, contributor, date, type, format, identifier, source, language, relation, coverage e rights*.

2.12.4 Descrições orientadas ao formato

Um conjunto de metadados define um modelo conceitual do que é o dado, porém, nos formatos descritos nessa seção, o foco principal é o formato.

MARC, ou *Machine Readable Cataloging Record* é um padrão, hoje na versão MARC 21, mantido pela Biblioteca do Congresso Americano e utilizado mundialmente, para registros bibliográficos. Ele é composto de vários formatos de registros: bibliográfico, autoridades, posses, classificação e informação sobre comunidades (Library of Congress, 2020). Os formatos são expressos em sequências de bytes. Existe uma versão MARC XML, porém ela não foi definida sobre a semântica, mas sobre os campos do MARC. BIBFRAME é um padrão que se propõe a substituir o MARC (Library of Congress, 2022).

Outros formatos conhecidos para descrição de metadados de documentos são os usados por sistemas gerenciadores de referências bibliográficas, entre eles: .bib, .RIS, refer, e o formato de softwares muito usados como EndNote, Papyrus, Zotero, Mendeley, etc. Na prática são versões reduzidas das representações usadas pelas bibliotecas, como o MARC.

Atualmente, muitos metadados o descritos em XML ou JSON. Por exemplo, o Twitter descreve seus tweets em JSON quando eles são recuperados.

2.13 Coleções de documentos

Atkins, Clear e Ostler (1992) definem quatro tipos de coleções de textos:

- **Arquivo:** um repositório de textos eletrônicos legíveis que não são ligados de uma forma coordenada;
- **Biblioteca Eletrônica de Textos:** uma coleção de textos eletrônicos em um formato padronizado dentro de certas convenções relacionadas a conteúdo, etc,mas sem restrições rigorosas de seleção;
- **Corpus:** uma parte da biblioteca eletrônica, construído a partir de critérios explícito, para um objetivo específico, e
- **Subcorpus:** uma parte de um corpus, podendo ser um componente estático de um corpus complexo ou uma seleção dinâmica de um corpus feita durante uma análise *on-line*.

(Atkins, Clear e Ostler, 1992)

Várias áreas de pesquisa têm interesse em corpus, sendo seu uso muito comum para validação de algoritmos e aprendizado de máquina, na Computação, mas também para estudos de linguística. Alguns corpus são muito usados, como:

- Em inglês
 - *Brown University Standard Corpus of Present-Day American English*, conhecido como **Brown Corpus**, criado em 1961, com mais de 1 milhão de palavras e 500 documentos, disponível no NLTK.
 - Vários corpus fornecidos pela agência de notícias *Reuters*:Reuters, Reuters-21578, RCV1, RCV2, TRC2.
 - Corpus com valor histórico, considerados pequenos atualmente, mas que podem ser utilizados para testes, como o Cystic Fibrosis e a Cranfield Collection.

- Movie Reviews, usado para análise de sentimentos.
- GOV2 Test Collection.
- 20 Newsgroup dataset.
- TIPSTER.
- Em português, disponibilizados pela Linguateca¹²
 - CETEM Público, um corpus de aproximadamente 180 milhões de palavras em português de Portugal formado por notícias do jornal Público.
 - CETEM Folha, um corpus de cerca de 24 milhões de palavras em português do brasil formado por notícias da Folha de São Paulo.
 - CHAVE, contendo textos do Público e da Folha de São Paulo, em 1994 e 1995, com um total de 210.734 notícias.
 - Floresta Treebank, de português brasileiro, disponível no NLTK.
 - MacMorpho Corpus, de português brasileiro, disponível no NLTK.
 - ReLi, para análise de sentimentos.

Alguns corpora podem ser livremente acessados, outros são oferecidos gratuitamente sob licença para pesquisadores, e ainda existem os pagos. Por exemplo, o corpus English Gigaword Fifth Edition¹³ custa US\$6,000, para não sócios do LDC¹⁴. Já os corpus Reuters são gratuitos, mas entregues sob a assinatura de um licença, e todos os corpus do NLTK são gratuitos e usados praticamente de forma livre.

2.14 Exercícios

2.14.1 Perguntas de Múltipla Escolha

Exercício 2.1:

Qual das seguintes opções define melhor um texto na visão computacional?

1. Uma sequência aleatória de caracteres.
2. Uma coleção de imagens digitalizadas.
3. Uma sequência estruturada de símbolos convencionados que regista uma informação.
4. Um conjunto de dados numéricos.
5. Uma lista de comandos em uma linguagem de programação.

Exercício 2.2:

O que é considerado um documento na Ciência da Informação?

1. Apenas textos escritos.
2. Qualquer expressão do pensamento humano.
3. Somente publicações digitais.
4. Apenas objetos físicos como livros e revistas.
5. Somente arquivos armazenados eletronicamente.

Exercício 2.3:

O que caracteriza a Lei de Zipf na distribuição de palavras em um texto?

¹²<https://www.linguateca.pt/>

¹³<https://catalog.ldc.upenn.edu/LDC2011T07>

¹⁴A subscrição padrão do LDC custa US\$2,400

1. Distribuição uniforme das palavras.
2. Frequência de palavras não relacionada ao seu tamanho.
3. Frequência de palavras inversamente proporcional à sua ordem de ocorrência.
4. Todas as palavras têm a mesma frequência de ocorrência.
5. A frequência de palavras é diretamente proporcional ao seu comprimento.

Exercício 2.4:

Qual destes não é um nível de compreensão da língua mencionado no texto?

1. Fonético.
2. Morfológico.
3. Léxico.
4. Digital.
5. Semântico.

Exercício 2.5:

Qual dos seguintes itens não é uma característica associada aos documentos?

1. Indexalidade.
2. Pluralidade.
3. Estabilidade.
4. Inalterabilidade.
5. Produtividade.

Exercício 2.6:

O que diferencia um documento de um texto, segundo Smiraglia?

1. Um documento é sempre digital, enquanto um texto pode ser físico.
2. Um documento pode conter múltiplos textos.
3. Um texto é uma forma de documento.
4. Documentos são exclusivamente para comunicação escrita.
5. Textos não possuem significado sem um documento.

Exercício 2.7:

Segundo o texto, qual das seguintes opções não é uma forma de escrita?

1. Logográfica.
2. Silábica.
3. Alfabética.
4. Cromática.
5. Trácica.

Exercício 2.8:

Qual é a principal preocupação ao definir o que é um texto?

1. Seu comprimento.
2. A presença de metadados.
3. Sua capacidade de ser analisado e estudado.
4. A plataforma em que é publicado.

5. O número de autores.

Exercício 2.9:

Qual é a função principal de um corpus?

1. Armazenar grandes volumes de dados não textuais.
2. Servir como base de dados para sistemas de recuperação de informação.
3. Manter um registro histórico de uma língua.
4. Armazenar exclusivamente documentos legais.
5. Prover uma plataforma para a publicação de novos textos.

Exercício 2.10:

O que significa a termo "string" no contexto de um texto computacional?

1. Uma sequência de números.
2. Um tipo especial de variável booleana.
3. Uma sequência de caracteres.
4. Um método de compressão de dados.
5. Uma operação matemática.

2.14.2 Perguntas Dissertativas

Exercício 2.11:

Explique a importância da coerência interna em um texto. Discuta como a expectativa de coerência varia de acordo com a técnica usada e a expectativa do usuário, utilizando exemplos do texto para ilustrar diferentes abordagens à definição de texto.

Exercício 2.12:

Descreva a evolução do conceito de documento ao longo do tempo. Reflita sobre como as definições de documento se expandiram desde o século XIX até a era da internet, destacando como essa evolução reflete mudanças na sociedade e na tecnologia.

Exercício 2.13:

Explique como a abordagem simbólica e a abordagem conexionista diferem na tentativa de processamento de textos pela inteligência artificial, destacando o papel das redes neurais na abordagem conexionista.

Exercício 2.14:

Analise o papel da língua e da escrita na representação de informações em textos, considerando os desafios associados à codificação de textos em computadores e a relação entre a língua falada e a escrita.

Exercício 2.15:

Discuta a relevância dos modelos conceituais para documentos, como o FRBR e o DoCO, na organização e descrição de informações em bibliotecas e bases de dados eletrônicas.

Exercício 2.16:

Considere o artigo “Fellipe Duarte, Danielle Caled, Geraldo Xexéo: Minmax Circular Sector Arc for External Plagiarism’s Heuristic Retrieval stage. Knowl. Based Syst. 137:1-18 (2017)”¹⁵. Faça as seguintes descrições do artigo:

- Com o Dublin Core, usando XML.
- Com DoCO, usando a notação oferecida em <http://www.sparontologies.net/examples>.
- Com o formato do bibtex, ou biblatex.

Exercício 2.17:

Discuta as similaridades e diferenças entre os formatos formatos Dublin Core e DoCO, e interpreta como as diferenças aparecem em função do objetivo dos formatos.

Exercício 2.18:

Que formato parece mais adequado, e por que, para descrever cada documento em uma coleção destinada a guardar:

- livros,
- arquivos científicos,
- páginas web recuperadas sobre um assunto,
- mensagens de tweeter, e
- processos legais,
- leis e decretos,
- patentes,
- mensagens de correio eletrônico,
- notícias de jornal.

Exercício 2.19:

Investigue na internet, e relate, se existem formatos, internacionais, de outras nações, e especialmente brasileiros, específicos de metadados para:

- leis e decretos,
- processos burocráticos do governo,
- processos jurídicos em um tribunal,
- patentes, e
- notícias de jornal.

Exercício 2.20:

Usando o site <https://visl.sdu.dk/visl/pt/parsing/automatic/complex.php>, faça a análise sintática da frase “O menino envergonhado viu ontem o homem sozinho com o binóculo novo”. Comente a solução, única, apresentada pelo analisador e qual o seu significado.

Exercício 2.21:

Explique por que a sentença “A mãe vestiu a menina com o vestido vermelho”, que aparentemente tem a mesma estrutura de “o menino viu o homem com o binóculo”, não oferece ambiguidade para um leitor, mas pode ser complexa para um programa de computador.

¹⁵<https://doi.org/10.1016/j.knosys.2017.08.013>

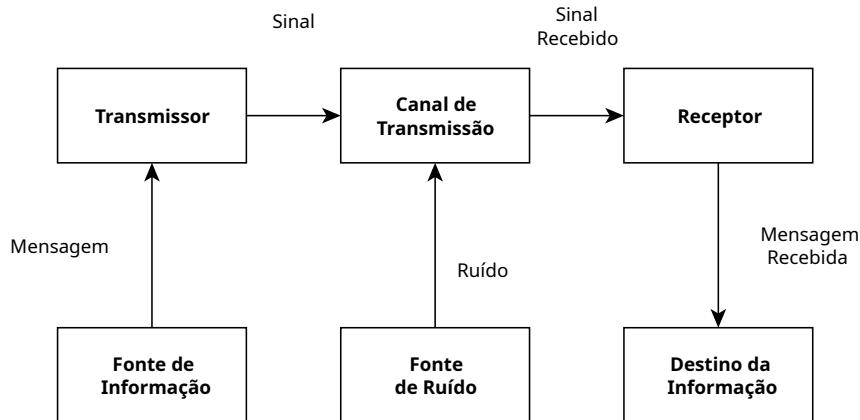


Figura 2.10: Canal de Ruído de Shannon

Exercício 2.22:

Analise o formato BIBFRAME, principalmente em relação ao formato MARC 21 e ao padrão FRBR. Discuta como os tags do formato BIBFRAME foram criados em relação a expectativa que temos de uma arquivo XML.

Exercício 2.23:

A Figura 2.10 mostra o modelo clássico de canal de ruído de Shannon. Discuta como podemos modelar o processo de comunicação entre duas pessoas por meio de um texto usando esse conceito.

Exercício 2.24:

Escolha um modelo conceitual para documentos e o descreva usando entre 1.200 e 1.500 palavras.

Exercício 2.25:

Escolha um corpus e o descreva usando entre 500 e 750 palavras.

Exercício 2.26:

Faça um programa que execute os quatro algoritmos de *stemming* em inglês do NLTK que foram expostos aqui com caches e teste tamanhos de cache que sejam potência de 2, de 2 a 2048. Compare velocidades e o tamanho ideal do cache.

Exercício 2.27:

Analise a comunicação face a face em comparação com a comunicação apenas por textos, como em e-mail e Whatsapp, e faça uma interpretação usando o canal de ruído de Shannon para explicar porque enganos e desacordos podem ocorrer mais frequentemente na forma textual pura.

CAPÍTULO 3

CODIFICAÇÃO DE CARACTERES

Um texto, no computador, é uma sequência de caracteres disponibilizado de diversas formas: como arquivos, mensagens em redes sociais, guardados em bancos de dados, gerados por meio de programas, etc.

Para processar um texto, o primeiro passo é entender que caracteres foram usados para construí-lo. Este capítulo trata da representação de caracteres em computadores.

3.1 Conceitos básicos sobre a textos na forma digital

Uma **língua**, ou **lingua natural**, é um sistema estruturado de sinais, sons, gestos ou símbolos usados para comunicação visual, verbal ou escrita entre as pessoas. A língua é um conjunto de regras que regem como as palavras são formadas e organizadas para criar significado (Macbeth, 2011). Exemplos de línguas incluem inglês, espanhol, francês, chinês, etc. Não deve ser confundida com a fala ou a escrita. Libras, a Linguagem Brasileira de Sinais, é uma língua com sinais. Este livro é sobre o uso da língua escrita, e outras formas, como sons, devem ser primeiramente traduzidas para a forma escrita.

Não devemos confundir, em português, língua e **linguagem**, que inclui sistemas mais amplos e abstratos e não apenas a língua, mas também outras formas de comunicação, como gestos, expressões faciais, postura corporal, entonação e tom de voz. A linguagem é usada para expressar pensamentos, emoções, desejos e necessidades, e é fundamental para a comunicação humana. Existe as linguagens da dança, da pintura. Porém a forma mais importante no estudo da comunicação entre os homens é a comunicação que fazemos por meio das línguas.

Um **script**, ou uma **escrita**, é um conjunto de caracteres escritos usados para representar um **idioma**, uma língua específica falada por uma comunidade de falantes. Crystal (2019) define um *script* como “um método para a escrita de sons da fala em um modo sistemático e consistente, de um ponto de vista específico”. É, portanto, um conjunto de regras para escrever um idioma usando um conjunto específico de símbolos. Por exemplo, o *script latino*, “abcd...” é usado para escrever muitos idiomas, incluindo inglês, francês e espanhol, enquanto o *script grego*, “αβγ...” é usado para escrever grego. Em português, um *script* pode ser chamado também de sistema de escrita, porém um **sistema de escrita** pode significar também o *script* e as regras para usá-lo. Existem muitos sistemas de escrita

diferentes, incluindo alfabetos, abjads, abugidas, silabários e sistemas logográficos. Um exemplo de um alfabeto é o *script* latino, enquanto o sistema de escrita chinês é um sistema logográfico. Crystal (2019) também indica como sinônimos de *script* as palavras **transcrição** e **notação**.

Uma **fonte** é um conjunto de caracteres digitais que representam um tipo de letra específico em um tamanho, estilo e peso específicos. Inclui o design das letras, números e símbolos usados em um idioma escrito. Por exemplo, Times New Roman e Arial são ambas fontes usadas no script latino.

Um **caractere** é um símbolo único usado para representar uma letra, número ou outro símbolo em um idioma escrito. Por exemplo, a letra “a” é um caractere usado no *script*, e alfabeto, latino, a letra α no alfabeto grego e a letra נ no sistema de escrita hebraico, que chamamos informalmente de alfabeto, mas é um sistema de outro tipo. Este livro trabalha com a codificação de caracteres.

Um **glifo**, ou **glyph**, é uma representação específica de um caractere em uma fonte ou sistema de escrita específico. É a imagem visual de um personagem que aparece em uma tela ou página impressa. Por exemplo, uma letra pode ter glifos diferentes em diferentes fontes, como um “A” sem ou um “A” com serifa, ou ainda um “A” cursivo.

Fontes e glifos não são importantes para este livro. A construção do significado de textos em meio digital se faz apenas pelos caracteres. Sua visualização, porém, depende de fontes e seus glifos. Arquivos com imagens de texto também não serão tratados neste livro, se for o caso de um problema real a que devem ser aplicadas as técnicas aqui descritas, essas imagens devem passar por um processo de reconhecimento de texto.

A discussão entre o que é um glifo e um caractere suscita discussões de vários níveis, inclusive filosóficos, porém uma definição prática é considerar o glifo a imagem ou símbolo usado no sistema de escrita, ou em uma notação, como em Música ou Matemática, e um caractere uma classe de equivalência de glifos (Haralambous, 2007).

Unicode, a maneira de codificar caracteres e *scripts* mais moderna, caracteres são um conceito abstrato das menores unidades atômicas do texto escrito que tem valor semântico. Eles representam letras, pontuações e outros símbolos. É possível perceber que pontuações, por exemplo, não são normalmente incluídas em alfabetos, mas pertencem a sistemas de escrita. Em Unicode, letras semelhantes, ou mesmo iguais, em sistemas de escrita diferentes tem representações diferentes, mesmo que seus glifos sejam iguais. Também é possível que letrar totalmente diferentes usem o mesmo glifo por coincidência ou motivos históricos. Cada caractere em Unicode é representado por um **code point**, ou seja, um número (Allen et al., 2007), que permite sua identificação única.

3.2 A representação de informação

Alguns autores dizem que os computadores processam números, e que é preciso representar caracteres como números para que os computadores os entendam, mas isso é uma simplificação falsa.

Na verdade, computadores processam sinais binários, os bits, e esses sinais são agrupados, de forma arbitrária e criada pelos fabricantes, em palavras, ou bytes¹

A escolha de 8 bits, e seus múltiplos, como tamanho da CPU, foi tomada ao longo do tempo, tanto por necessidades ou facilidades técnicas, como por tendências de mercado e necessidades de retrocompatibilidade. Deve ficar claro que, apesar dos computadores terem instruções especializadas para operações matemáticas, essas instruções supõem também uma representação, seja ela binário com

¹A palavra *byte*, hoje associada a exatamente 8-bits, já significou apenas uma unidade completa que podia ser processada pelo computador.

complemento-de-2, Binary Coded Decimal (BCD)², IEEE 470 para números de ponto flutuante ou outra. O conjunto de instruções do Intel i7 possui pelo menos 14 tipos diferentes de adição, de acordo com a representação de um número.

O uso de 4 bits em um dos primeiros microprocessadores produzidos, o Intel 4004, garantia que era possível guardar pelo menos 1 algarismo decimal descrito no formato BCD, que seria usado para um cliente fazer calculadoras (Leibson, 2021).

Os 8-bits são usados atualmente, entre outros motivos, pelo fato de terem sido lançados microprocessadores de 8-bits que causaram a revolução da microcomputação, como o Intel 8080, o Zilog Z-80 e o MOS Technology 6502. A partir daí foram usados múltiplos de 8, como 16, 32, 64 e 128, para manter uma compatibilidade. Os 8-bits, inicialmente, era apenas um dos tamanhos possíveis da “palavra” do computador. Por exemplo, os CDC Cyber 170, no final dos anos 1970, usavam caracteres de 6-bits, endereços de 18 bits, instruções de 15 ou 30 bits e palavras de 60 bits, e para os quais um “byte”, tinha 12 bits (Control Data Corporation, 1975).

Assim, fica estabelecido que tanto a representação típica em 8-bits, quanto o uso de números para indicar como um caracter é representado, são construções culturais e tecnológicas que não significam que caracteres são números, ou que 8 bits sejam realmente necessários.

Em todo caso, para evitar escrever caracteres como sequência de bits, normalmente é usada uma a representação como inteiro decimal (ou hexadecimal) da sequência de bits usadas no computador para descrever seu código.

Assim, um caracterer ‘A’ do alfabeto latino, em código ASCII e em UTF-8, é representado como a sequência de 8 bits ‘01000001’, ou com o valor inteiro decimal 65, ou ainda pelo valor inteiro hexadecimal 0x41³. Já em código EBCDIC, hoje de uso muito restrito, é representado o valor 193 ou 0xC1.

Computadores têm que ser binários?

Não há nenhuma obrigação real ou lei da física que obrigue os computadores a serem baseados na lógica binária. Porém é mais fácil trabalhar com a unidade mínima de informação, o bit, e juntá-los em grupos para otimizar o processamento.

Tanto Knuth (1997, p. 208) quanto Claude Shannon expressaram opinião que um computador baseado em lógica ternária seria mais interessante que um binário.

Para exemplificar é possível citar o Setun um computador que usava lógica ternária, criado sob liderança de Sergei Sobolev e Nikolay Brusentsov, na Moscow State University, e fabricado pela empresa Kazan (Brousentsov et al., 2021; Wikipedia, 2020).

3.3 Da variedade para o ASCII

No início da computação a preocupação com padrões de representações de textos era pequena, e cada fabricante foi criando uma regra própria. Linguagens de programação, como APL, podiam exigir caracteres específicos, e o uso do computador era fortemente ligado a operações matemáticas.

²Na verdade existem vários formatos de BCD, que podem ser vistos na página da Wikipedia https://en.wikipedia.org/wiki/Binary-coded_decimal

³A partir de agora, quando necessário, principalmente para diferenciar de números em decimal, hexadecimais serão representados por números começando com os caracteres “0x”. Em outros momentos, quando estiver claro o contexto, serão escritos em grupos de dois caracteres.

Nas comunicações, porém, a necessidade de padrões para troca de comunicação já era estabelecida. O código Morse, baseado em “pontos e traços”, possibilitou a telegrafia. Mackenzie (1980) descreve várias codificações para transmissão de informações, como o código CCITT⁴ # 2, para telégrafos, com 58 caracteres e 6 bits, estabelecido em 1931 e ainda códigos específicos para cartões perfurados.

O Código Morse

O código Morse pode ser visto tanto como um código binário, formado pelo sinal ligado ou desligado, quanto como um código com cinco símbolos: o ponto (*dit*), o traço (*dahs*) e os intervalos pequeno, médio e longo.

A unidade básica de medida de tempo é o tempo usado para o ponto. O traço deve medir três vezes o tempo do ponto, e o espaço entre pontos e pontos ou pontos e traços também deve durar o mesmo que um ponto. Letras são separadas por um silêncio equivalente a três pontos, palavras por um silêncio equivalente a sete pontos.

Assim, para enviar as palavras “MORSE CODE”, a sequência seria:

Que, usando 1 para os tempos onde o circuito está ligado e zero para os tempos em que está desligado, resulta no código binário:

111011100011101110001011101010100010000000111010111010001110111011100011101010001

Essa mensagem duraria 87 vezes o tempo usado para um ponto.

Mais uma curiosidade. Apesar de Morse não diferenciar entre maiúsculas e minúsculas, é comum considerar que as letras são maiúsculas.

Já no final do anos 1950, foi iniciada a discussão de um novo código, que se tornaria o *American Standard Code for Information Interchange* (**ASCII**)⁵, apresentado na Tabela 3.1). Durante a criação apareceram como requisitos fornecer: letras, numerais, pontuações, símbolos comerciais e matemáticos e ainda caracteres de controle. Isso levou à necessidade de representar mais de 64 caracteres, ultrapassando os 6-bits usados até então, logo pelo menos 7 bits deveriam ser usados. Existiam fortes argumentos para o uso de 8-bits, mas eles foram vencidos pelos defensores dos 7-bits, que economiza bits na transmissão, e decorrentemente, o tamanho de uma mensagem enviada de forma serial. Entre os argumentos dos defensores dos 7 bits estava: “As 128 combinações disponíveis em um conjunto de caracteres de 7-bits satisfazem os requisitos de troca de informação e controle da grande maioria dos usuários” (Mackenzie, 1980). As discussões sobre o tema foram bastante técnicas, e seus efeitos perduraram até hoje, porém mostram a ingenuidade do início da Computação, além de uma preocupação imediata com o alfabeto da língua inglesa, indicada pela ausência de letras acentuadas. Por causa dessa deficiência, passaram a existir vários extensões ao ASCII.

Mas como o **ASCII** assumiu um papel preponderante e venceu todos as outras codificações, resistindo até hoje dentro do código Unicode? Isso pode ser parcialmente atribuído ao “memorandum 127”, assinado em 11 de março de 1968, pelo presidente americano Lyndon B. Johnson, que não só adotava o ASCII com padrão federal, mas também obrigava que qualquer computador comprado tivesse a capacidade de usar não só o ASCII, mas também outros formatos prescritos para fita magnética e de papel (Johnson, 1968)⁶. Outro fator importante foi sua adoção pelos microcomputadores. Mesmo assim, a IBM, usava e ainda usa, em alguns de seus sistemas, como o z/OS, um formato criado por

⁴Comité Consultatif International Telegraphic et Telephonique

⁵Muitos pensam ser o número 2 em algorismos romano, mas a pronúncia correta é *as-kii*

⁶Uma forma de entrada e saída comum na época e hoje abandonada, como os cartões perfurados.

Tabela 3.1: Tabela ASCII, ou Unicode com a representação UTF-8

Dec	Controle	Dec	Char	Dec	Char	Dec	Char
0	NUL (null)	32	SPACE	64	@	96	'
1	SOH (start of heading)	33	!	65	A	97	a
2	STX (start of text)	34	"	66	B	98	b
3	ETX (end of text)	35	#	67	C	99	c
4	EOT (end of transmission)	36	\$	68	D	100	d
5	ENQ (enquiry)	37	%	69	E	101	e
6	ACK (acknowledge)	38	&	70	F	102	f
7	BEL (bell)	39	,	71	G	103	g
8	BS (backspace)	40	(72	H	104	h
9	TAB (horizontal tab)	41)	73	I	105	i
10	LF (NL line feed, new line)	42	*	74	J	106	j
11	VT (vertical tab)	43	+	75	K	107	k
12	FF (NP form feed, new page)	44	,	76	L	108	l
13	CR (carriage return)	45	-	77	M	109	m
14	SO (shift out)	46	.	78	N	110	n
15	SI (shift in)	47	/	79	O	111	o
16	DLE (data link escape)	48	0	80	P	112	p
17	DC1 (device control 1)	49	1	81	Q	113	q
18	DC2 (device control 2)	50	2	82	R	114	r
19	DC3 (device control 3)	51	3	83	S	115	s
20	DC4 (device control 4)	52	4	84	T	116	t
21	NAK (negative acknowledge)	53	5	85	U	117	u
22	SYN (synchronous idle)	54	6	86	V	118	v
23	ETB (end of trans. block)	55	7	87	W	119	w
24	CAN (cancel)	56	8	88	X	120	x
25	EM (end of medium)	57	9	89	Y	121	y
26	SUB (substitute)	58	:	90	Z	122	z
27	ESC (escape)	59	;	91	[123	{
28	FS (file separator)	60	<	92	\	124	
29	GS (group separator)	61	=	93]	125	}
30	RS (record separator)	62	>	94	^	126	~
31	US (unit separator)	63	?	95	_	127	DEL

ela como padrão em seus sistemas, conhecido como EBCDIC, *Extended Binary Coded Interchage Code* (IBM, 2010). Esse código foi criado simultaneamente ao ASCII para o lançamento da linha OS/360, um dos produtos de maior sucesso da empresa (Haralambous, 2007). Outros países, como o Japão (JIS X) e a Rússia (KOI-7) possuíam códigos semelhantes ao ASCII, mantendo compatibilidade na questão de sinalização, mas usando seus próprios caracteres.

A maioria dos programadores, mesmo hoje em dia, têm a necessidade, em algum momento, de se referir a Tabela ASCII, principalmente quando trata de letras. Isso significa trabalhar com os números usados para representar os caracteres de acordo com o “Código Padrão Americano para Intercâmbio de Informações”, que pode ser visto na Tabela 3.1. Não só ela é importante historicamente, aparece em livros de programação, como é mantida pelo código mais novo, o Unicode UTF-8.

Por que esses códigos

Os códigos ASCII não foram escolhidos de forma totalmente arbitrária.

Os dígitos, por exemplo, vão do número 0x30 até 0x39. Isso significa que se seus bits iniciais forem jogados fora, os últimos representam o dígito em código decimal binário.

Já as letras foram escolhidas de forma que somando ou subtraindo 32 a elas, ou seja, ligando apenas um bit, fosse possível trocá-las de letras maiúsculas para minúsculas e vice-versa.

3.4 *Code pages*

Como ficou demonstrado mais tarde, 7 bits foram insuficientes para as reais necessidades dos usuários. Eles não cobriam, por exemplo, caracteres acentuados, ou outros alfabetos.

Enquanto participava das discussões que criariam o ASCII, a IBM criou em paralelo, em 1963, o EBCDIC, de 8 bits, para permitir usar um conjunto mais amplo de caracteres que o seu código original, o BCD, permitia. Além disso, a ideia também permitia várias codificações, por exemplo para atender o japonês, e dava um número de 16 bits para identificar cada codificação. Inicialmente, esse número era correspondente a página do manual da IBM. Por isso no nome *code page*, que basicamente dizia a página em que estava o código específico (Haralambous, 2007).

Outras empresas adotaram a prática, e com isso foram criadas centenas de variações que usavam os 8 bits disponíveis em um byte para codificar caracteres, e mesmo não havendo mais relação com páginas reais, foi mantido o nome. Ele foi adotado também no sistema operacional dos microcomputadores, o PC-DOS, e migrou para a Microsoft por causa da versão MS-DOS. Porém, as empresas usavam códigos diferentes para páginas iguais. São mais de 200 code pages só no Microsoft Windows, outras para sistemas operacionais diferentes (Haralambous, 2007).

Porém cada fabricante possuía suas próprias code pages, muitas vezes com outros nomes, o que dificulta o compartilhamento de informações. Para facilitar esse compartilhamento, foram criados comitês de padronização na ISO, que acabaram criando várias extensões de 8 bits para o ASCII, nos padrões **ISO-8859**, tentando organizar a área. Entre eles, os mais conhecidos no Ocidente são o ISO 8859-1, ou **ISO Latin 1**, que atendia a maioria das línguas da Europa Ocidental, o ISO 8850-2, para as línguas do Leste Europeu, o ISO 8859-5 para o alfabeto Cirílico (Haralambous, 2007).

Os padrões ISO são levemente diferentes das *code pages* da Microsoft. Devido ao poder da Microsoft, a *code page 1252*, um super-conjunto da ISO 8850-1, acabou sendo o padrão de fato para as linguagens ocidentais, incluindo o próprio inglês. Até hoje muitos arquivos digitais são encontrados usando essa *code page*.

A expansão de padrões e *code pages* levou a uma nova tentativa de padronização. Essa tentativa, que atualmente deu certo, se consolidou na padronização dos caracteres no Unicode, que fornece um código para cada caractere, chamado de *code point*, e uma forma de representação dos *code points* em uma sequência de bits (Haralambous, 2007).

Hoje o Unicode é o formato preferido para a criação de novos documentos ou bases de dados.

3.5 Unicode

Unicode é um padrão internacional que “fornecce um número único para cada caracter, não importa que plataforma, dispositivo, aplicação ou linguagem”(Unicode, Inc., 2017). Unicode apresenta um conjunto de restrições adicionais ao padrão **ISO/IEC 10646**, porém possui os mesmos códigos de caracteres e formas de codificação que o padrão, sendo ambos sincronizados.

Os códigos são representados por números inteiros decimais, sendo facilmente mapeáveis nos bytes do computador moderno. Os primeiros 127 caracteres do Unicode são compatíveis com os 127 códigos do ASCII. Em sua versão 14.0 o padrão Unicode continha 144.697 caracteres de várias linguagens ou, mais corretamente, *scripts*, incluindo gráficos (Unicode, Inc., 2021).

O Unicode não representa linguagens, mas sim coleções de letras e símbolos usadas para representar informação em um sistema de escrita. Um *script* pode representar um sistema de escrita atual, antigo ou histórico. *Scripts* não são iguais a sistemas de escrita. Por exemplo, japonês pode ser escrito nos *scripts* Han, Hiragana ou Katakana, seguindo o mesmo sistema de escrita.

Unicode também não indica como as letras, ou outros símbolos, são desenhados (os glifos). Assim, todas as linguagens que usam um “a” minúsculo, usam a mesma codificação para esse “a” minúsculo, e a forma de representá-lo é escolhida por meio de uma fonte, como Arial, que indica como o “a” minúsculo é desenhado.

3.5.1 UTF

O Unicode define um conjunto universal de pontos de código para representar caracteres de diversos sistemas de escrita, mas não especifica como esses pontos de código são representados em um computador ou transmitidos entre dispositivos. Para isso, o mesmo padrão define as formas de codificação, os *Unicode Transformation Formats*(UTF). Os UTF são métodos de codificação que transformam os pontos de código abstratos do Unicode em sequências de bytes concretas(Unicode, Inc., 2019). Os formatos UTF foram criados para não permitir a sobreposição⁷ de códigos. A sobreposição ocorre quando um código, pode ser encontrado dentro de outro código(Unicode, Inc., 2019).

A forma mais direta é o **UTF-32** (Unicode Transformation Format), que é simplesmente a representação do número do caracter Unicode em 32 bits, o que permite 4.294.967.295 caracteres. Isso significa, porém, que cada caracter usa 4 bytes, o que é um desperdício.

O **UTF-16** é uma codificação que usa unidades de 16 bits para representar os caracteres Unicode. Isso permite que ele codifique 65.536 caracteres diretamente, conhecidos como o Plano Multilíngue Básico (BMP). Para representar caracteres fora desse intervalo, o UTF-16 utiliza pares substitutos, combinando duas unidades de 16 bits para codificar caracteres adicionais, estendendo assim sua capacidade além do BMP.

3.5.2 UTF-8

A forma mais usada para representar os caracteres é o **UTF-8**, um código de tamanho variado, que representa os caracteres Unicode com 8, 16, 24 ou 32 bits(Unicode, Inc., 2019). No Windows, UTF-8 corresponde a *code page* 65001. A Tabela 3.2 exemplifica caracteres unicodes com sua representação em UTF-8 com vários tamanhos. UTF-8 pode codificar 1.112.064 *code points* do Unicode (Unicode, Inc., 2019).

⁷Em inglês, *overlap*

Tabela 3.2: Alguns caracteres Unicode e suas representações UTF-8

Unicode		Exemplo	UTF-8	
Nome	Código		Decimal	Hexadecimal
LATIN CAPITAL LETTER A	U+0041	A	65	41
GREEK SMALL LETTER ALPHA	U+03B1	α	206 177	CE B1
EURO SIGN	U+20AC	€	226 130 172	E2 82 AC
GRINNING FACE EMOJI	U+1F600	☺	240 159 152 128	F0 9F 90 80

3.6 Normalização de Unicode

Normalização do Unicode é o processo de converter texto em uma forma consistente e padronizada. Isso é necessário porque há uma ambiguidade importante em Unicode, causada pela possibilidade de combinar caracteres para formar caracteres compostos. Isso acontece porque ao construir a lista de caracteres, e com objetivo de manter uma compatibilidade com outros esquemas de caracteres, foram ao mesmo tempo criados caracteres presentando as formas acentuadas de algumas letras e acentos que podem ser combinados com as formas sem acento⁸. Dessa forma, alguns caracteres, chamados de *combining characters* foram criados especialmente para serem usados em combinações com caracteres comuns, como o acento agudo combinante⁹, e que são combinados com as vogais no português, mas podem, por serem caracteres combinantes, serem combinados com qualquer outra letra, e inclusive em combinações múltiplas (Miranda, 2023).

Assim, as letras com acento têm pelo menos duas representações: as suas representações próprias, ou pré-compostas, e as representação com combinação entre a letra sem acento e o acento agudo. Essa representação dupla é referenciada como uma **equivalência canônica** entre a representação pré-composta e a combinado. As equivalências canônicas acontece quando existe um caracter pré-composto que pode ser usado no lugar de uma combinação. Um conceito similar é o de equivalência por compatibilidade, que se refere a representação visual similar de um mesmo conceito (Whistler, 2023).

Por exemplo, a letra “á” (a com acento agudo) pode ser representada tanto pelo *code point* “U+00E1”, que é o seu *code point* pré-composto, quanto pela sequência “U+0061”+“U+0301”, que é o caracter ‘a’ seguido do caracter que representa o acento agudo combinante. Logo, quando escrito por um programa, esse “á” pode ser representado por um ou dois bytes (de acordo com a representação UTF-8). Dessa forma, pode acontecer que um arquivo venha com um formato, com o outro, ou mesmo com ambos. Isso pode ser tratado por meio de técnicas de **normalização Unicode** (Miranda, 2023).

Em um programa de computador é importante reconhecer as formas de representação, o que gera formas normalizadas conhecidas como **NFC**, **NFD**, **NFKC** e **NFKD**. As formas têm essas características (Miranda, 2023; Whistler, 2023):

- NFC: *Normalization Form Canonical Composition*: onde todos os caracteres devem ser usados na forma pré-composta;
- NFD: *Normalization Form Canonical Decomposition*: onde todos os caracteres deve ser usados na forma combinada;
- NFKC: *Normalization Form Compatibility Composition*: compõe além dos símbolos que têm forma pré-composta, outros símbolos que não tem equivalência canônica, mas que têm uma compatibilidade.
- NFKD: *Normalization Form Compatibility Decomposition*: decompõe além dos símbolos que têm forma pré-composta, outros símbolos que não tem equivalência canônica, mas que têm uma compatibilidade.

⁸Na verdade, diacríticos

⁹O nome oficial em inglês é *combining acute accent*, que é um caracter diferente do *acute accent*

Os casos NFC e NFD são mais simples porque tratam apenas da equivalência canônica, que são todas conhecidas. Já o caso NFKD e NFKC tratam de compatibilidade, um conceito estendido que inclui símbolos como o de marca registrada (™) e as letras TM. Nesse caso, porém, como é uma compatibilidade, a conversão de ™ para TM pode facilitar um mecanismo de busca, mas não podemos fazer a inversa automaticamente e salvar o arquivo.

A normalização pode ser feita em Python usando o pacote `unicodedata` e a função `normalize`. O programa 3.1 mostra um exemplo de seu uso.

Programa 3.1: Exemplo de leitura com normalização

```

1 import unicodedata
2
3 def normalizar(texto):
4     """
5     Aplica a normalização Unicode NFKD a uma string.
6
7     A normalização NFKD (Normalization Form KD) decompõe os caracteres compostos em
8     ↪ seus
9     componentes básicos. Isso é especialmente útil para caracteres acentuados,
10    onde o caractere acentuado é decomposto em um caractere base e o acento separado
11    ↪ .
12
13 Parâmetros:
14 texto (str): A string que será normalizada.
15
16 Retorna:
17 str: A string normalizada em forma NFKD.
18 """
19
20 # Aplica a normalização NFKD usando unicodedata.normalize
21 return unicodedata.normalize('NFKD', texto)
22
23 # Exemplo de uso da função
24 texto_original = "Olá, mundo!"
25 texto_normalizado = normalizar(texto_original)
26 print("Texto Original:", texto_original)
27 print("Texto Normalizado:", texto_normalizado)

```

3.7 O fim de linha

Arquivos ou mensagens em texto normalmente são divididas em linhas. Lamentavelmente, motivos diferentes levaram os criadores dos sistemas operacionais a usar formas diferentes de indicar um fim de linha.

Seguindo a tradição dos teletipos, o MSDOS e MS Windows adotam dois caracteres para indicar o fim de linha, o CR (Carriage Return) e o LF (Line Feed), nessa ordem. Como teletipos usavam folhas de papel, para mudar de linha era necessário voltar o “carro”, isto é, a cabeça de impressão, e pular uma linha na folha.

Os sistemas baseados em Unix usam um só caracter, o LF. Isso inclui o Mac OSX em diante. Curiosamente, o sistema do Mac pré-OSX usava apenas o CR.

Em EBCDIC existe o caracter EOL (*end of line*).

3.8 Representações de Unicode em Python

Podemos representar o *code point* Unicode em um programa de várias maneiras. Uma delas é usar o caracter diretamente, como em á. Outra é usar seu código, como em \u00e1, ou como mostramos no texto anterior, \u0061\u0301. Essa representação é de 16 bits. Pode ser usada uma de 32 bits também, como em U000000e1. Outra forma é usar o nome da letra em Unicode, com a notação \N{\textit{<nome>}}, como em \N{LATIN_SMALL LETTER_A_WITH_ACUTE}. Também é possível usar a função `chr` tanto com decimais quanto com hexadecimais, como em `chr(0xe1)` ou `chr(225)`.

A Pedra de Roseta

A Pedra de Roseta, cuja a foto da frente está na Figura 3.1, foi o documento que permitiu a compreensão dos significados dos hieróglifos. Ela apresenta o mesmo texto em três versões: o superior em hieróglifos egípcios, o segundo em demótico e o terceiro em grego. Ela foi a chave que permitiu decifrar a escrita egípcia.

A tradução foi feita a partir do texto em grego antigo. Uma das chaves para a solução final foi o uso de caracteres fonéticos para representar os nomes estrangeiros em demótico e os cartuchos. Na prática isso significa dizer que o mesmo nome estava codificado foneticamente em caracteres de sistemas de escrita diferentes.

Uma característica interessante: ao decifrar Pedra de Roseta por uma dica fonética, também se aprendeu como falar a língua antiga do Egito.

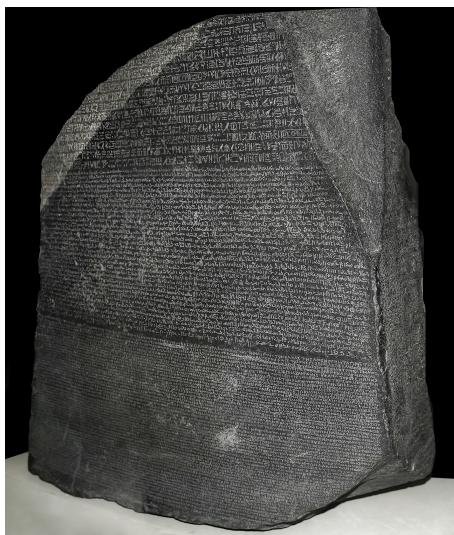


Figura 3.1: A Pedra de Roseta, que apresenta um decreto de um conselho de sacerdotes estabelecendo o culto ao Faraó Ptolomeu V. Fonte: Wikipedia.

3.9 Desafios apresentados pela codificação

O principal desafio apresentado pela codificação é o fato de ser impossível conhecer qual é a codificação antes de processar um texto sem que essa informação esteja em outro lugar, seja na forma de arquivo ou mensagem, e, ao mesmo tempo, é essencial conhecê-la para poder fazer o processamento correto.

Por exemplo, seja um arquivo que aberto com um editor de texto simples que apresenta os seguintes 12 caracteres:

aáàâæéèêëçÇ

Se salvo com a codificação UTF-8, ele apresentará os seguintes bytes (em hexadecimal):

61 C3 A1 C3 A0 C3 A2 C3 A4 65 C3 A9 C3 A8 C3 AA C3 AB C3 A6 C3 87

Já se codificado com a code page 1252 (Western European) do MS Windows, apresentará os seguintes bytes:

61 E1 E0 E2 E4 65 E9 E8 EA EB E7 C7

Ao programar, isso exige algum cuidado. Por exemplo, em Python (Rossum e Drake, 2009) existem várias variáveis do sistema que indicam a codificação. A leitura de arquivos é feita, por default, usando a que está em `locale.getpreferredencoding()`.

O programa Programa 3.2 lê dois arquivos apresentados anteriormente, um em UTF-8 e outro na *code page* 1252. Ele mostra vários valores de codificação do sistemas indicando UTF-8, mas o único importante indica CP1252. Se o arquivo em CP1252 for lido em UTF-8 há um erro, pois alguns códigos não são permitidos em UTF-8.

Programa 3.2: Exemplo de leitura com codificação

```

1 import sys
2 import locale
3
4 print("Default encoding: {}".format(sys.getdefaultencoding()))
5 print("File system encoding: {}".format(sys.getfilesystemencoding()))
6 print("Local preferred encoding: {}".format(locale.getpreferredencoding()))
7
8 f = open("Exemploutf8.txt", "r")
9 print("Defaut open encoding: {}".format(f.encoding))
10 for i in f:
11     print(i)
12 f.close()
13
14 f = open("Exemploutf8.txt", "r", encoding="utf-8")
15 print("Forcing utf-8, we get: {}".format(f.encoding))
16 for i in f:
17     print(i)
18 f.close()
19

```

```

20 # File in cp1252
21 f = open("ExemploWindowsWestern1251.txt", "r")
22 print("Defaut open encoding: {}".format(f.encoding))
23 for i in f:
24     print(i)
25 f.close()
26
27
28 f = open("ExemploWindowsWestern1251.txt", "r", encoding='cp1252')
29 print("Forcing cp-1252, we get: {}".format(f.encoding))
30 for i in f:
31     print(i)
32 f.close()

```

O resultado de executar esse programa é¹⁰:

<pre> 1 Default encoding: utf-8 2 File system encoding: utf-8 3 Local preferred encoding: cp1252 4 Defaut open encoding: cp1252 5 aÃ¡ÃÃÃÃÃÃÃ 6 Forcing utf-8, we get: utf-8 7 aÃÃÃÃÃÃÃ 8 Defaut open encoding: cp1252 9 aÃÃÃÃÃÃÃ 10 Forcing cp-1252, we get: cp1252 11 aÃÃÃÃÃÃÃ </pre>	Saída do Programa 3.2
--	------------------------------

Recomenda-se fortemente que toda abertura de arquivo especifique a codificação, principalmente porque não há garantia que a codificação preferida seja UTF-8 em todas as plataformas que o Python executa.

Atualmente, a grande maioria dos textos usados na internet está sendo produzida em UTF-8, porém isso não é uma verdade absoluta. Além disso, muito texto hoje contido em banco de dados está em um formato diferente, baseado no ASCII, mas usando uma *code page* que precisa ser identificada.

Alguns formatos, como o XML, podem indicar a codificação. Por exemplo, um *Prolog XML*, como o da Figura 3.2, é um elemento opcional no arquivo XML, mas se existe, pode indicar a codificação. Além disso, o padrão XML define UTF-8 como a codificação padrão.

```
<?xml version="1.0" encoding="UTF-8"?>
```

Figura 3.2: Um *Prolog XML* indicando a codificação

Já uma página obtida pelo protocolo HTTP possui um cabeçalho “Content-type”, que supostamente forneceria a codificação correta. Isso também deveria ser padrão nos arquivos HTML por meio da marcação `<meta http-equiv="content-type">`.

¹⁰ Esperando que todos os caracteres apareçam aqui como aparecem no console IPython.

UTF-8 é a codificação default para os formatos: HTML5, CSS, JavaScript, PHP e SQL. Também Python 3 adota Unicode, isso significa que a codificação padrão de leitura de programas Python supõe arquivos em Unicode. Isso também significa que se uma string Python é convertida em uma lista de bytes, a string e a lista podem ter tamanhos diferentes.

Uma *stream* qualquer UTF (como um arquivo) pode começar indicando sua codificação com uma sequência específica conhecida como *Byte Order Mark* (BOM), que é um caracter U+FEFF. Em UTF-8 isso é a sequência 0xEF,0xBB,0xBF. Esse uso não é uma recomendação e é, na prática, incomum.

3.10 Detectando uma *code page* automaticamente

Não existe uma forma padrão de se detectar a codificação de um texto que é desconhecido. Existem porém heurísticas que podem ser usadas.

O pacote `chardet` 4.0.0 é um “Universal encoding detector for Python 2 and 3”, i.e., se propõe a resolver esse problema em Python, por meio de modelos treinados. Ele pode ser encontrado em <https://pypi.org/project/chardet/>.

Em sua documentação (Pilgrim, Blanchard e Cordasco, 2015) o pacote apresenta um exemplo simples de uso, via o interpretador:

Programa 3.3: Exemplo simples de uso do chardet

```

1 >>> import urllib.request
2 >>> rawdata = urllib.request.urlopen('http://yahoo.co.jp/').read()
3 >>> import chardet
4 >>> chardet.detect(rawdata)
5 { 'encoding': 'EUC-JP', 'confidence': 0.99}

```

No mesmo documento, também apresentam uma solução para descoberta da codificação em um programa:

Programa 3.4: Exemplo de programa que usa o chardet

```

1 import urllib.request
2 from chardet.universaldetector import UniversalDetector
3
4 usock = urllib.request.urlopen('http://yahoo.co.jp/')
5 detector = UniversalDetector()
6 for line in usock.readlines():
7     detector.feed(line)
8     if detector.done:
9         break
10 detector.close()
11 usock.close()
12
13 print(detector.result)

```

Também é possível usar o pacote `Charset-Normalizer`¹¹. Esse pacote é capaz de trabalhar de forma semelhante ao `chardet`, como mostra o Programa 3.5.

¹¹<https://pypi.org/project/charset-normalizer/>

Programa 3.5: Uso do charset-normalizer.

```

1 import os
2 from charset_normalizer import detect
3
4 path = 'F:\Github\Dados—Exercicios—Texto\Capitulo Codificacao de Caracteres'
5
6 os.chdir(path)
7 files = os.listdir(os.getcwd())
8
9
10 for fname in files:
11     f = open(fname, "rb")
12     result = detect(f.read())
13     print(fname+": "+result["encoding"])

```

O pacote `charset-normalizer` pode já fornecer o texto normalizado, usando o `best()`, como no Programa 3.6.

Programa 3.6: Uso do charset-normalizer com normalização automática.

```

1 import os
2 from charset_normalizer import from_path
3
4 LINES = 5
5 COLUMNS = 72
6
7 path = 'F:\Github\Dados—Exercicios—Texto\Capitulo Codificacao de Caracteres'
8
9 os.chdir(path)
10 files = os.listdir(os.getcwd())
11
12 text = str(from_path("base.txt").best())
13
14 for i in range(LINES):
15     print(text[i*COLUMNS:i*COLUMNS+COLUMNS])

```

3.11 Lendo com codificações em Python

É simples abrir um arquivo para leitura com uma codificação específica, bastando para isso declarar o argumento `encoding` no comando `open`, como no Programa 3.7.

Programa 3.7: Comandos para abrir um arquivo escolhendo a codificação

```

1 f = open(fname, encoding="utf-8")
2 g = open(gname, encoding="iso-8859-1")

```

Assim para ler um arquivo tentando prever sua codificação, com o pacote `chardet` são necessárias duas leituras, sendo que a primeira deve fazer a leitura de forma binária (em Python), como é feito

no Programa 3.8. Já para o pacote `charset-normalizer`, o Programa 3.6 mostra como isso pode ser feito em uma só leitura.

Programa 3.8: Exemplo de programa que usa o chardet e lê o arquivo

```

1 from chardet.universaldetector import UniversalDetector
2
3 # lê para prever
4 f = open("Exemploutf8.txt", "rb")
5 detector = UniversalDetector()
6 for line in f:
7     detector.feed(line)
8     if detector.done: break
9 detector.close()
10 f.close()
11
12 print(detector.result)
13
14 # lê para obter as linhas de forma correta
15
16 f = open("Exemploutf8.txt", "r", encoding=detector.result["encoding"])
17 for line in f:
18     print(line)
19 f.close()

```

3.12 Tratando a codificação no KNIME

Como o KNIME possui nós com comportamentos distintos, duas estratégias são possíveis para tratar a codificação de textos (strings).

A primeira é usar a codificação default do KNIME. Ela pode ser alterada no arquivo knime.ini ou nas preferências em General:Workspace:Text File Encoding. O valor default para o KNIME é CP1252. Isto é exemplificado na Figura 3.3.

A outra opção é usar um módulo “File Reader”, exemplificado na Figura 3.4. Ele é feito para ler arquivos de dados, mas pode ser usado também para ler um texto linha a linha. Para isso é possível marcar o delimitador de coluna como “\r”, isto é, o caractere LF, como na Figura 3.5, e usar o tab Encoding para definir a forma de codificação, como na Figura 3.6. Caso a codificação esteja errada, é normalmente possível identificar no *preview*, como na Figura 3.7.

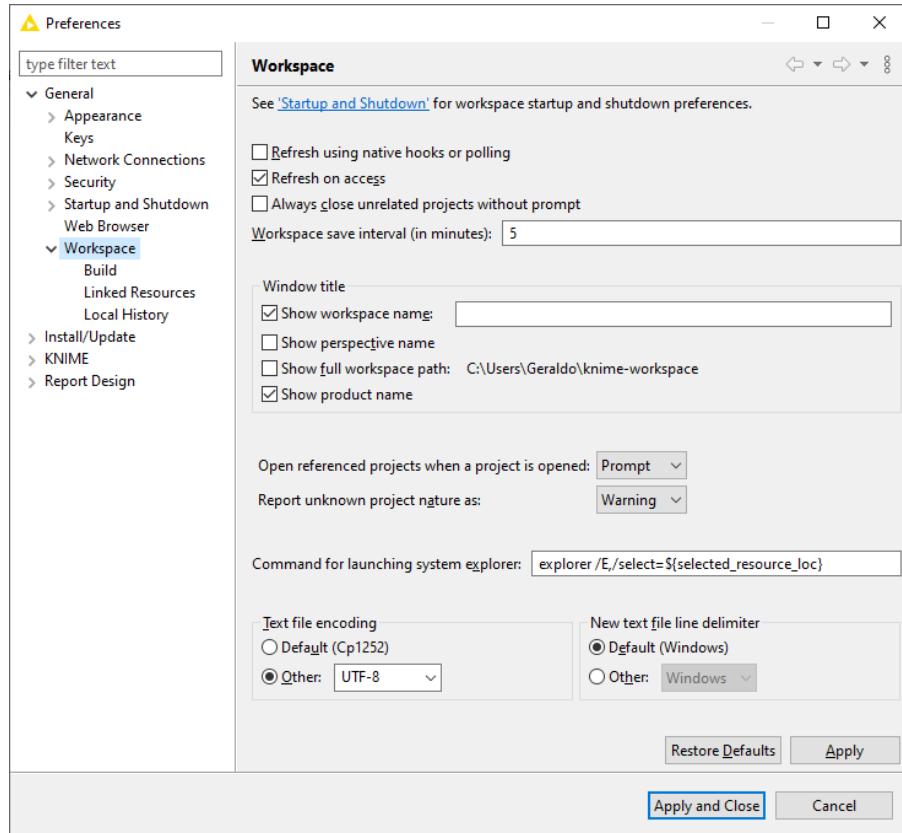


Figura 3.3: Configurando a codificação default do KNIME.



Figura 3.4: Um workflow KNIME só com um módulo, FileReader.

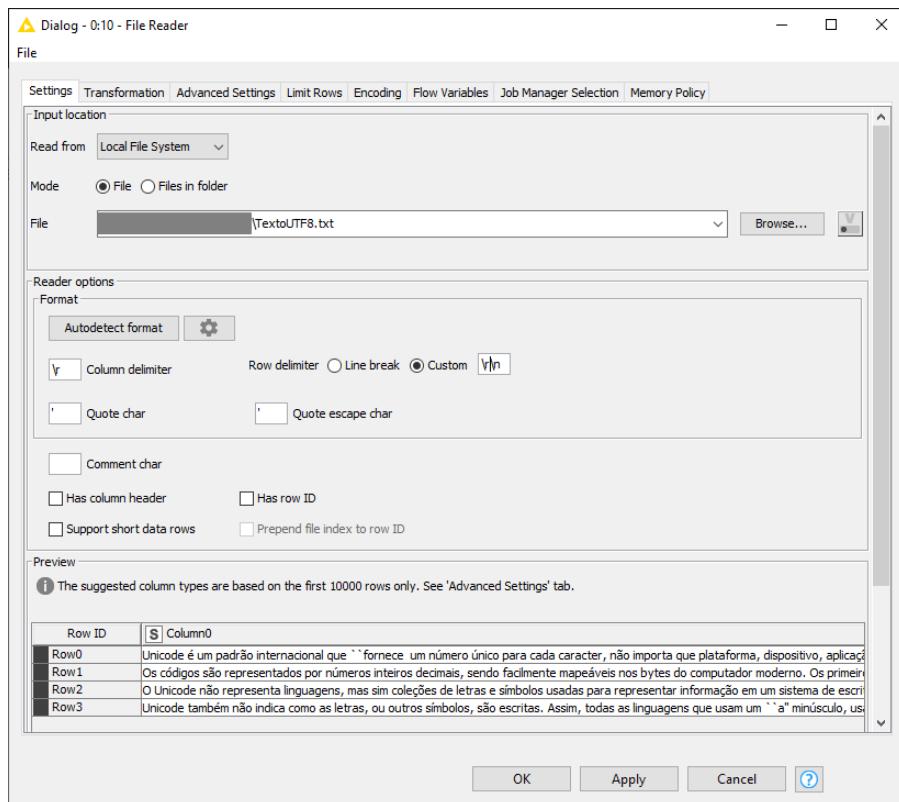


Figura 3.5: Lendo um arquivo texto como uma sequência de linhas no KNIME. Módulo File Reader.

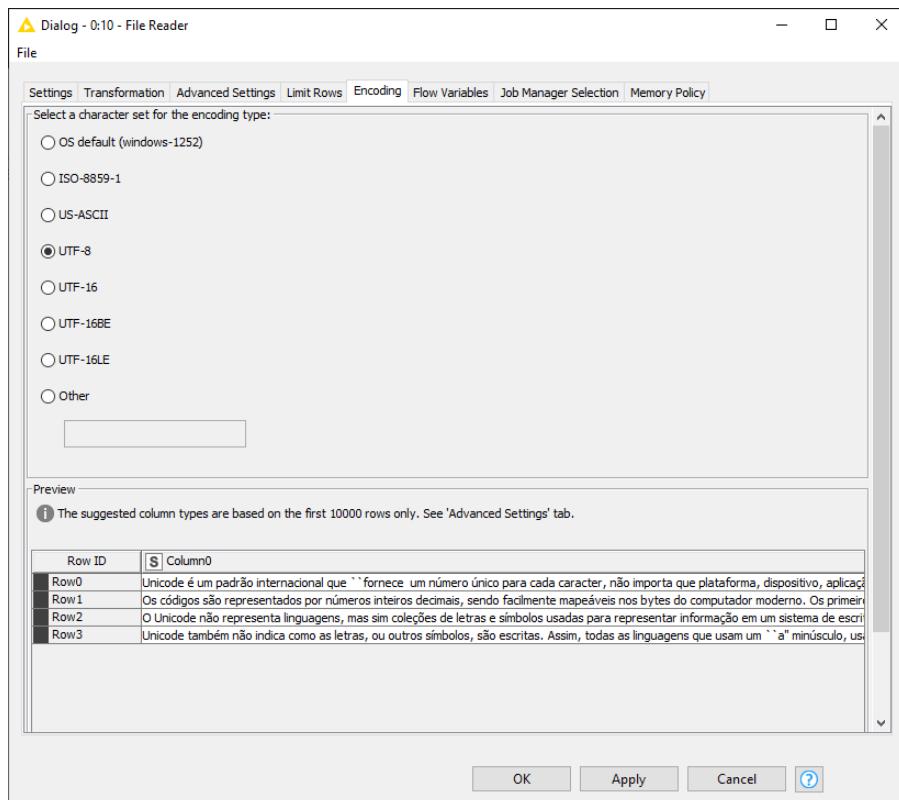


Figura 3.6: Configurando a codificação correta do arquivo.

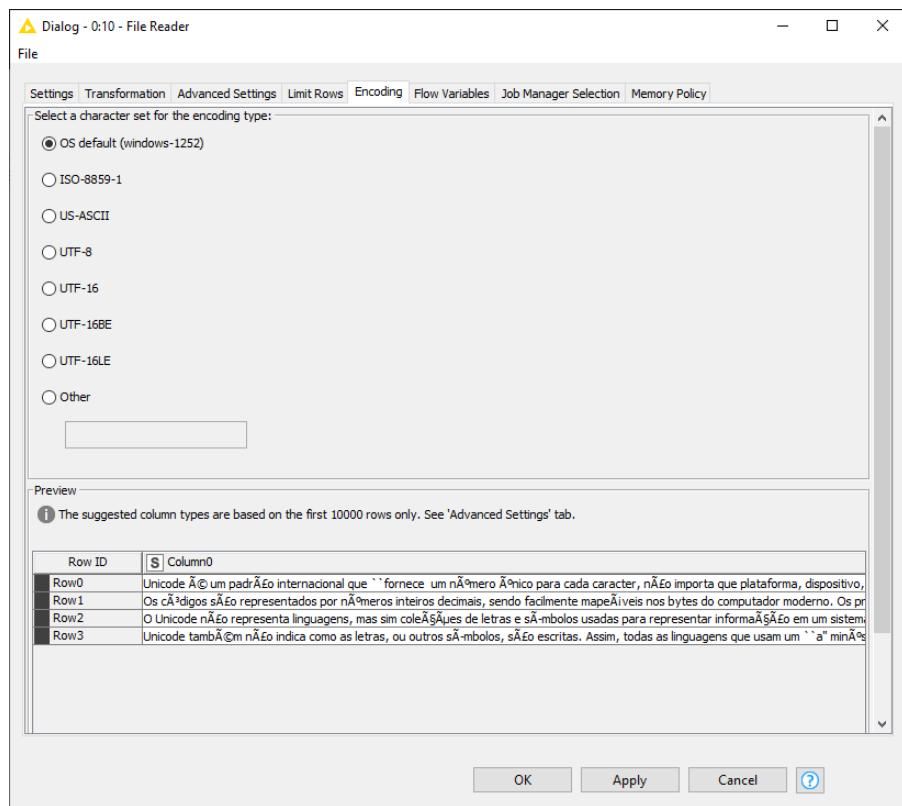


Figura 3.7: Configurando a codificação errada, a visualização apresenta caracteres estranhos.

3.13 Codificando strings

Em Python, pode ser necessário em algum momento gerar a codificação de uma string, em algum código específico. Isso é feito com a função `byte`, como no Programa 3.9:

Programa 3.9: Obtendo a sequência correta da codificação de uma string

```

1 import locale
2
3 print("Locale é: {}".format(locale.getpreferredencoding()))
4
5 string = "coração brasileiro"
6 print("A string '{0}' com tamanho {1}".format(string, len(string)))
7
8 encodings = ['utf-8', 'ascii', 'cp1252']
9
10 for enc in encodings:
11     try:
12         sb = bytes(string, enc)
13         print("Encoding:{} - '{0}' - len {1}".format(sb, len(sb), enc))
14     except Exception as e:
15         print(str(e)[0:70])

```

A saída desse programa é:

Saída do Programa 3.9

```

1 Locale é: cp1252
2 A string 'coração brasileiro' com tamanho 18
3 Encoding:utf-8 - 'b'cora\xc3\xb5a7\xc3\xb3o brasileiro' - len 20
4 'ascii' codec can't encode characters in position 4-5: ordinal not in
5 Encoding:cp1252 - 'b'cora\xe7\xe3o brasileiro' - len 18

```

Também é possível simplesmente escolher a codificação quando se escreve o arquivo, como mostra o Programa 3.10.

Programa 3.10: Salvando um mesmo arquivo em várias codificações.

```

1 import os
2
3 path = 'F:\Github\Dados—Exercicios—Texto\Capitulo Codificacao de Caracteres'
4
5 os.chdir(path)
6
7 f = open("base.txt", "r", encoding="utf-8")
8 text = f.read()
9
10 encoders = ['utf-32', 'cp1252']
11
12 for i in range(len(encoders)):
13     g = open("texto "+str(i)+".txt", "w", encoding=encoders[i])

```

```
14     g.write(text)
15     g.close()
```

3.14 Codificações por referência

Alguns padrões definem como representar em um arquivo símbolos desejados sem que eles estejam na codificação utilizada. Nesse caso, os símbolos que não podem ser representados são representados de acordo com alguma regra, usando apenas símbolos existentes na codificação.

Um exemplo disso são as entidades HTML (*HTML entities*). Elas são definidas de duas formas:

`&entity_name;`

ou

`&entity_decimal_number;`

ou ainda

`ntity_hexadecimal_number;`

As referências numéricas são equivalentes aos *code points* do Unicode.

Por exemplo, em HTML, `&nbs1;` identifica um *non-breaking space*, já `´` identifica um “a” com acento agudo, “á”. O símbolo de copyright, “©”, pode ser gerado a partir de `©`, ou, usando a forma numérica, `©`.

O W3C definiu alguns conjuntos de entidades, para HTML, MathML, XML e HTML 5.

Outras linguagens, como L^AT_EX apresentam também códigos similares para alguns ou todos os caracteres, como `\copyright` para o “©”.

A vantagem de usar essa forma de representação é que qualquer caractere pode ser representado apenas com os caracteres ASCII ou com um conjunto menor ainda de números.

Ao processar, porém, é necessário tratar esses códigos adicionais de alguma forma. Em Python isso pode ser feito com o pacote `html` e a função `unescape`, como no Programa 3.11.

Programa 3.11: Processando entidades HTML

```
1 import html
2
3 import html
4
5
6 T = ["Teste&copy;\u0141\tá", u"Teste&copy;\u0141\tá",
7      b"Teste&copy;\u0141\t", r"Teste&copy;\u0141\tá"]
8
9 for s in T:
10    try:
11        print(s)
12        print(html.unescape(s))
```

```
13     except Exception as e:
14         print("Erro:"+str(e))
```

Cujo resultado é:

Resultado do Programa 3.11	
1	Teste© á
2	Teste\l á
3	Teste© á
4	Teste\l á
5	b'Teste©\\u0141\t'
6	Erro:a bytes-like object is required, not 'str'
7	Teste©\\u0141\tá
8	Teste\l\\u0141\tá

Interpretando o resultado é possível entender o que acontece, sabendo o significado das letras “u”, unicode, “r” raw, e “b”, bytes.

A primeira string, "Teste©\\u0141\tá" é comum. Nesse caso ela é processada normalmente como string, o que faz com que tanto o tab quanto o código unicode inserido. Ao ser processada pelo unescape do html, o faz corretamente. A segunda string é uma string unicode, e funciona exatamente como a normal, pois em Python 3 as string normais são unicode¹².

A terceira string é uma sequência de bytes na verdade. Ela não aceita, já na compilação, o “á”, que foi retirado. Além disso ela dá erro no processamento `html.unescape`.

A quarta string é do tipo raw, e não aceita “\” como *escape*, porém aceita também o processamento `html.unescape`.

i Representação interna de strings em Python 3

Em Python 3 as strings tem uma representação interna múltipla, que tem como objetivo otimizar o funcionamento. Assim, a partir da versão 3.3, a representação interna de uma string depende da necessidade criada pelos caracteres da própria string e pode ser Latin-1 (utf-8), UCS-2 (predecessora do UTF-16, de 2 bytes e de tamanho fixo), ou UCS-4 (predecessora do UTF-32).

A PEP 393 descreve a *Flexible String Representation* para Python(Löwies, 2010).

3.15 Codificação de binários em texto

Uma estratégia usada em alguns protocolos de transmissão é a transformação de binários em texto, normalmente ASCII. Isso evita que algumas configurações de 8-bits sejam interpretadas de alguma forma indesejável pelos protocolos, como os caracteres de comando. Isso é muito usado, por exemplo, para a transmissão de imagens, ou qualquer outra tipo de arquivo binário, por meio de correio eletrônico, mesmo havendo extensões 8-bits do protocolo original (Postel, 1982).

¹²Em Python 2 elas são bytestring.

Existem vários formatos que seguem esse princípio, e a Wikipedia fornece uma boa lista¹³: Ascii85, Base32, Base36, Base45, Base58, Base62, Base64, Base85, BinHex, uuencode, ... Nesse texto vamos exemplificar todos usando o Base64, um dos formatos usado no MIME (Freed e Borenstein, 1996).

3.15.1 Base64

Base64 é um conjunto de esquemas de codificação que permite transmitir dados binários na forma de texto. É especialmente usado no protocolo SMTP(Klensin, 2008; Postel, 1982) de transferência de e-mail, para transportar imagens ou documentos, por meio do MIME (Freed e Borenstein, 1996), o que causa um aumento tamanho nos arquivos. Claramente, não é comum representar texto como Base64, porém documentos completos, por exemplo em PDF, podem estar “escondidos” dentro desse formato.

Em Base64 são usados 64 caracteres ASCII, para indicar 64 grupos de 6 bits. Na implementação MIME de Base64 são usados, nessa ordem e começando por zero, os caracteres A-Z, a-z, 0-9 e os caracteres “+” e “/”. Qualquer necessidade de completar os bits, *padding*, é feita com o caractere “=”. Assim, por exemplo, o ‘A’ é o número 0, o ‘b’ é o número 27, o ‘+’ é o número 62. A Tabela 3.3 exemplifica uma codificação da palavra “Man” em ASCII para Base64, cujo resultado seria “TWFu”. Como Base64 codifica 8 bits em 6 bits, cada 3 bytes codificados gera exatamente 4 caracteres na string.

Tabela 3.3: Exemplo de como é feita uma codificação Base64

Entrada	M	a	n
ASCII	0x4d	0x61	0x6e
bits	0 1 0 0 1 1 0 1 0 1 1 0 0 0 0 1 0 1 1 0 1 1 1 0		
Sexteto	19	22	5
Saída	T	W	F
			u

3.16 Exercícios

3.16.1 Questões Objetivas

Exercício 3.1:

O que é uma língua natural?

1. Um sistema de comunicação que utiliza apenas sons.
2. Um conjunto de regras para programação de computadores.
3. Um sistema estruturado de sinais, sons, gestos ou símbolos usados para comunicação.
4. Uma forma de escrita baseada unicamente em símbolos matemáticos.
5. Um conjunto de palavras em um dicionário.

Exercício 3.2:

Qual a diferença principal entre língua e linguagem?

1. Língua é um tipo de linguagem musical.
2. Linguagem inclui sistemas mais amplos e abstratos de comunicação, além da língua.
3. Linguagem é um subconjunto da língua.
4. Língua é usada apenas para comunicação escrita.
5. Não existe diferença entre os dois termos.

¹³https://en.wikipedia.org/wiki/Binary-to-text_encoding

Exercício 3.3:

O que define um script ou uma escrita?

1. Uma coleção de obras literárias.
2. Uma técnica para desenho artístico.
3. Um conjunto de caracteres usados para representar uma língua.
4. Uma forma específica de codificação de computador.
5. Um método para gravar vídeos.

Exercício 3.4:

O que é um glifo?

1. Uma unidade básica de medida de som.
2. Uma representação específica de um caractere em uma fonte ou sistema de escrita.
3. Um antigo método de comunicação.
4. Um tipo de código binário.
5. Uma ferramenta de software para design gráfico.

Exercício 3.5:

Qual a função principal do Unicode?

1. Converter texto em imagens.
2. Fornecer um número único para cada caractere, independente da plataforma ou aplicação.
3. Comprimir arquivos de texto para economizar espaço.
4. Criar novas fontes para sistemas operacionais.
5. Traduzir automaticamente entre diferentes línguas.

Exercício 3.6:

Como é representado um caractere em Unicode?

1. Por um conjunto de símbolos gráficos.
2. Através de uma combinação de letras e números.
3. Por um code point, que é um número único.
4. Utilizando uma sequência específica de bits.
5. Através de uma representação gráfica única.

Exercício 3.7:

O que é UTF-8?

1. Uma versão antiga do Unicode.
2. Um tipo de fonte para sistemas operacionais.
3. Uma codificação para transferência de arquivos.
4. Um formato de codificação que representa caracteres Unicode com 8, 16, 24 ou 32 bits.
5. Um software de tradução de línguas.

Exercício 3.8:

O que diferencia o ASCII do Unicode?

1. ASCII pode representar mais caracteres que o Unicode.

2. Unicode pode representar uma gama muito mais ampla de caracteres que o ASCII.
3. ASCII é usado apenas em sistemas Windows, enquanto Unicode é para Mac.
4. Unicode é uma versão mais antiga do ASCII.
5. Não há diferença; são termos intercambiáveis.

Exercício 3.9:

Por que é importante normalizar textos em Unicode?

1. Para garantir que todos os textos sejam escritos em inglês.
2. Para aumentar a velocidade de carregamento das páginas da web.
3. Para tratar diferentes formas de representar o mesmo caractere de maneira consistente.
4. Para converter todos os textos em ASCII.
5. Para criar novos caracteres que não existem no Unicode.

Exercício 3.10:

O que significa a codificação UTF-32?

1. Um formato que representa todos os caracteres Unicode com 8 bits.
2. Um formato que representa todos os caracteres Unicode com 32 bits.
3. Uma codificação obsoleta que foi substituída pelo UTF-8.
4. Um método para criptografar textos.
5. Um novo padrão que substituirá o Unicode

3.16.2 Questões Discursivas

Atenção: os arquivos a serem usados para os exercícios desse capítulo podem ser encontrados em:
<https://github.com/LINE-PESC/Dados-Exercicios-Texto>

Exercício 3.11:

No diretório “Capitulo Codificacao de Caracteres” estão disponíveis alguns arquivos “.txt”, criados a partir do arquivo “base.txt”, porém cada um com uma codificação. Analise esses arquivos visualmente e tente adivinhar que formato usam. É possível usar editores que mostram os bytes, como o disponível em <https://hexed.it/>. Depois use o Python e tente detectar automaticamente os formatos com o módulo chardet.

3.16.3 Questões Discursivas

Exercício 3.12:

Use o formato MARC 21 para registrar o livro “Abbott, Edwin (1991). Flatland: A Romance of Many Dimensions. 2¹ ed. Princeton University Press”. Crie o record MARC 21 duas vezes, uma considerando o formato MARC-8 e outra o formato UCS/Unicode. Salve ambos em arquivos e leia os arquivos com um editor de bytes, como o disponível em <https://hexed.it/>. Qual diferença entre eles? Imprima também, com e sem codificação para utf-8. O que significam os caracteres especiais que aparecem no registro?

Exercício 3.13:

Explique como a codificação de caracteres influencia a globalização de software e conteúdo digital. Discuta os desafios enfrentados pelos desenvolvedores de software ao criar aplicativos que devem funcionar em várias línguas e culturas, considerando diferentes sistemas de escrita.

Exercício 3.14:

Descreva o processo de normalização Unicode e sua importância para a comparação de strings em diferentes linguagens de programação. Aborde os tipos de normalização Unicode (NFC, NFD, NFKC, NFKD) e como eles afetam o processamento de texto.

Exercício 3.15:

Discuta o papel das fontes e glifos na visualização de caracteres de diferentes línguas. Explique como o mesmo caractere Unicode pode ser exibido de maneira diferente em diversas fontes e o impacto disso na legibilidade do texto em contextos multilíngues.

Exercício 3.16:

Explique a diferença entre os termos "script", "sistema de escrita" e "alfabeto" no contexto da codificação de caracteres. Dê exemplos de como um mesmo script pode ser usado para escrever várias línguas e como línguas diferentes podem compartilhar certos caracteres, mas não necessariamente o mesmo sistema de escrita.

Exercício 3.17:

Descreva o impacto da escolha entre diferentes esquemas de codificação (como UTF-8, UTF-16, UTF-32) no tamanho do arquivo e na eficiência de processamento de texto. Inclua uma discussão sobre as situações em que cada esquema é preferencialmente utilizado e os trade-offs envolvidos.

3.16.4 Questões de Programação

Exercício 3.18:

Analise os arquivos “*.txt” do diretório “Capítulo Codificação de Caracteres” e obtenha quantos bytes possuem e quantos caracteres mostram, se visualizados com um editor de texto. Leia os arquivos no formato correto.

Exercício 3.19:

Faça duas rotinas, uma que codifique um arquivo binário em Base64 e uma que decodifique e gere um arquivo exatamente igual ao original. Prove que as rotinas funcionam. Compare a velocidade de suas rotinas com as fornecidas pelo pacote base64.

Exercício 3.20:

O objetivo deste exercício é ter uma experiência básica com o Scikit-Learn. Exige algum conhecimento de aprendizado de máquina.

Para isso você deve fazer uma modificação do programa de aprendizado de máquina apresentado no Programa 23.1:

1. Usar a codificação adicional "latin 1", que é a mesma que a ISO-8859-1, ainda para o português. Isso vai causar uma confusão com a "cp1252", ou "Windows-1252". A diferença entre os dois é pequena. Mostre o gráfico da árvore, a acurácia e a matriz de confusão. (fácil)
2. Usar um algoritmo de agrupamento (um simples, como K-means), com os 4 grupos que são bem separados, na coleção original, e veja se os grupos correspondem às classes originais. (médio)
3. Busque um texto em espanhol (razoavelmente grande) e veja o impacto que tem na detecção da codificação em 4 grupos (programa original do professor). O Projeto Gutenberg pode ajudar com o texto, ou o nltk. Execute a árvore de decisão com textos em espanhol e mostre o gráfico da árvore, matriz de confusão e acurácia mais uma vez. Mantenha todos os textos com o mesmo tamanho aproximado (2 a 3K).
4. Usar o K-NN como algoritmo, variando K entre 1 e 20 e buscando o melhor K. Mostre a acurácia e a matriz de confusão. (fácil)

CAPÍTULO 4

ARQUIVOS TEXTUAIS

Nesse capítulo é feita uma introdução aos formatos de arquivos, como o formato PDF, que são mais usados na internet e que devem ser processados para que possam ser realizadas tarefas de indexação, busca e aprendizado de máquina com textos.

Alguns desses formatos não podem ser lidos diretamente por seres humanos, ou podem ser lidos apenas em certas condições, enquanto outros são de fáceis compreensão.

O objetivo de conhecer os formatos é compreender os métodos de transformar o conteúdo desses arquivos em arquivos simples de texto, que serão usados nas tarefas de processamento de texto.

4.1 Os formatos mais comuns

Os formatos de arquivo são criados com diferentes motivações, o que leva a uma série de requisitos, e restrições, e também por escolhas pessoais ou institucionais. Assim, enquanto o formato em que são guardados os arquivos fonte de L^AT_EX deve poder ser lido por seres humanos, o formato básico resultante do seu processamento, o dvi, se destina a ser processado por programas de computador e é de difícil leitura para seres humanos.

A maioria do processamento que é feito hoje em textos usa poucos tipos de arquivos: arquivos de texto simples, arquivos no formato do Word, e arquivos do tipo PDF, HTML, XML ou JSON.

Esses formatos podem ser divididos, de forma geral, em três tipos: formatos de edição, formatos de marcação ou formatos de impressão.

Formatos de edição e marcação são criadas para atender simultaneamente o homem e a máquina, podendo variar entre formatos facilmente interpretáveis por humanos, até alguns que são de difícil interpretação.

Finalmente, formatos de impressão são feitos para indicar como um texto deve ser impresso ou visualizado. Um dos formatos de impressão mais conhecidos, PostScript, é na verdade uma linguagem de programação que pode ser entendida por um especialista.

Ainda nesse caso, se um arquivo em um formato, ou codificação, for lido com a suposição que foi escrito em outro formato, erros aparecerão.

Além disso, vamos tratar de alguns formatos que podem ser encontrados não só em arquivos, mas também em *streams* de dados. O processamento, porém, é semelhante.

4.2 Formatos de edição

Pouco é preciso falar de formatos de edição que não tenha sido tratado no Capítulo 3, pois normalmente a única maneira deles causarem algum problema é serem lidos de acordo com uma codificação diferente da planejada.

Arquivos desse tipo normalmente são chamados de arquivos texto e têm como característica usarem a extensão “.txt”, mas isso não é uma regra determinante. Outros formatos, com mais estrutura, como um *Comma Separated File* ou “.cvs” podem ser tratados, algumas vezes, como um arquivo texto simples.

Os **formatos de edição** podem ser editados diretamente por editores simples e não possuem nenhuma marcação. Normalmente programas de computador são guardados nesse formato. Eles estão sempre codificados de alguma forma, como visto no Capítulo 3.

Antes do aparecimento do Unicode, os formatos de edição normalmente tinham uma relação um para um entre seu tamanho e o número de caracteres, onde cada byte era um caractere em um *codepage*. Como limite, as codepage só podiam ter 256 caracteres e tinham que ser escolhidas de forma ideal para um arquivo, geralmente baseada na língua em que eram escritos. Atualmente, como caracteres podem ser representados por mais de um byte, isso não é mais verdade. Isso significa, também, que os editores “simples” não são mais tão simples. A Figura 4.1 mostra um arquivo desse tipo visualizado em um editor de texto simples e compatível com UTF-8, enquanto a Figura 4.2 mostra o mesmo arquivo visualizado em editor hexadecimal, onde cada byte corresponde a um número ou uma letra. Atenção como certas letras não aparecem na listagem à direita e são substituídas por dois símbolos, já que as letras acentuadas usam dois bytes no UTF-8. Atenção para a sequência final “0D 0A”, que são, respectivamente, os caracteres CR e LF. Já na penúltima linha temos a sequência “70 72 C3 AA 6D 69 6F”, que corresponde a “prêmio”, onde o “ê” precisa de dois bytes, “C3 AA” para ser representado.

Muitos sistemas e experimentos têm como primeiro passo do processamento de um arquivo textual a sua transformação em um formato simples de texto.

Um exemplo de arquivo em UTF-8.
 Na primeira linha temos caracteres atendidos
 por 7 bits.
 Acentos, cedilha e til precisam de 2 bytes, em
 palavras como ação, prêmio e índice.

Figura 4.1: Arquivo texto visualizado em um editor de texto simples compatível com UTF-8.

00000000	55 6D 20 65 78 65 6D 70	6C 6F 20 64 65 20 61 72	Um exemplo de ar
00000010	71 75 69 76 6F 20 65 6D	20 55 54 46 2D 38 2E 0D	quivo em UTF-8..
00000020	0A 4E 61 20 70 72 69 6D	65 69 72 61 20 6C 69 6E	.Na primeira lin
00000030	68 61 20 74 65 6D 6F 73	20 63 61 72 61 63 74 65	ha temos caracte
00000040	72 65 73 20 61 74 65 6E	64 69 64 6F 73 20 70 6F	res atendidos po
00000050	72 20 37 20 62 69 74 73	2E 0D 0A 41 63 65 6E 74	r 7 bits...Acent
00000060	6F 73 2C 20 63 65 64 69	6C 68 61 20 65 20 74 69	os, cedilha e ti
00000070	6C 20 70 72 65 63 69 73	61 6D 20 64 65 20 32 20	l precisam de 2
00000080	62 79 74 65 73 2C 20 65	6D 20 70 61 6C 61 76 72	bytes, em palavr
00000090	61 73 20 63 6F 6D 6F 20	61 C3 A7 C3 A3 6F 2C 20	as como a <u>é</u> úo,
000000A0	70 72 C3 AA 6D 69 6F 20	65 20 C3 AD 6E 64 69 63	pr <u>ñ</u> mio e <u>ñ</u> ndic
000000B0	65 2E 0D 0A 0D 0A 0D 0A	+	e.....

Figura 4.2: Arquivo texto visualizado em um editor hexadecimal.

4.3 Linguagens de marcação

Muitos arquivos textuais são construídos por meio de linguagens de marcação. Em uma linguagem de marcação, caracteres são usados tanto para o conteúdo como para definir, de forma imperativa ou declarativa, o que deve ser feito por um processador específico, como um editor de texto ou uma impressora.

Os **formatos de marcação** ainda podem ser editados diretamente, apresentando um arquivo legível para o ser humano. Editores de texto podem esconder a marcação, usando uma apresentação WYSIWYG, mas sempre vão guardá-los de forma que possam ser abertos por um editor comum. A Figura 4.3 mostra um arquivo L^AT_EX, que é um formato de marcação, visualizado pelo software Notepad. Marcações como \textit{ são vistos pelos humanos, mas interpretados pelo software L^AT_EX como instruções, no caso colocar o texto em itálico.

A marcação não é parte do texto, mas está inserida nele. Assim, em L^AT_EX, por exemplo, escrevemos

```
\textbf{Negrito}
```

quando queremos que o texto “Negrito” esteja em negrito, como em “**negrito**”.

Entre as linguagens de marcação mais utilizadas estão: HTML, XML, SGML, L^AT_EX, RTF, Open XML Document (formato usado no Microsoft Word), OpenDocument (.odt).

Stand-off markup

Uma alternativa ao uso de linguagens de marcação seria usar o que é conhecido como “*stand-off markup*”, ou “*stand-off annotation*”: anotações separadas que fazem referência ao texto original. Por exemplo, um arquivo adicional dizendo onde começa o negrito, por exemplo no caracter 12302 do arquivo, e onde acaba, no caracter 12308, no caso.

A ideia foi defendida por Nelson (1997), e tem uma versão descrita por Thompson e McKelvie (1997), porém não é difundida atualmente.

As linguagens de marcação também podem ser divididas em(Coombs, Renear e DeRose, 1987)¹:

- Descritivas², como HTML, SGML, XML;

¹Os autores também citam um tipo *Puntuational*, referente a marcações sintáticas que dão informações sobre enunciados escritos, como espaços, e que não é preciso considerar no momento.

²Provavelmente o nome mais adequado seria declarativas

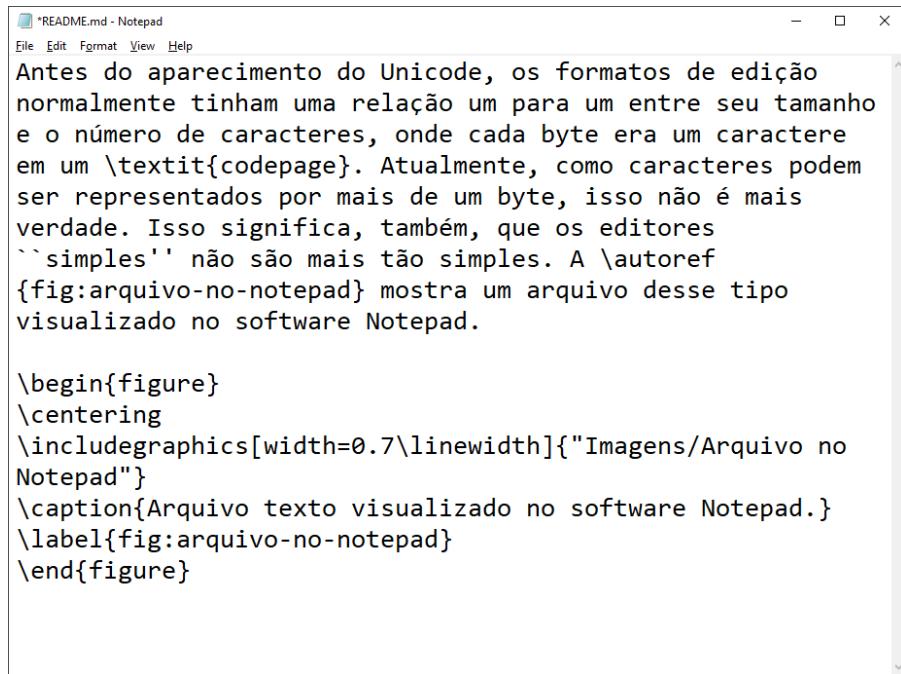


Figura 4.3: Arquivo de marcação L^AT_EXvisualizado pelo software Notepad.

- Procedurais, como T_EX, PostScript ou Lout, e
- De Apresentação, como Wikis.

4.3.1 HTML

HyperText Markup Language foi um formato originalmente criado por Tim Berners-Lee, baseado em SGML, para servir de linguagem de marcação para o World Wide Web (WWW). Junto com o protocolo HTTP ele formou uma das tecnologias de maior sucesso na história.

Apesar de hoje HTML 5 possuir vários tipos de marcação, inicialmente possuía poucos rótulos, que permitiam descrever efeitos desejados na apresentação do texto e a criação de ligações de hipertexto via rede. Ao longo do tempo os rótulos foram se avolumando e novas características apareceram, como a possibilidade de executar programas dentro de uma página HTML, em JavaScript(Mozilla, 2021), ou de criar estilos variados e possivelmente dinâmicos, com CSS(W3C, 2021b).

Não é fácil processar corretamente HTML, porque os arquivos desse formato são criados com muita liberdade e é comum que possuam erros. Na prática, os principais processadores de HTML, os navegadores WWW, perdoam muitos erros.

Porém, normalmente a única necessidade, ou a necessidade principal quando realizando atividades de indexação ou aprendizado de máquina é limpar o HTML, ficando apenas com o texto.

Um arquivo HTML moderno, e muito simples, pode ter a seguinte aparência:

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>Título da Página</title>
5 </head>
6 <body>
```

```

7
8 <h1>Cabeçalho</h1>
9 <p>Um parágrafo simples</p>
10
11 Usando entidades HTML &copy;
12
13 </body>
14 </html>
```

Sua visualização seria a da Figura 4.4

Cabeçalho

Um parágrafo simples

Usando entidades HTML ©

Figura 4.4: A visualização de um arquivo HTML muito simples.

A questão em HTML é que palavras escolher, isto é, como retirar o texto desejado do arquivo. Em primeiro lugar existem as *tags*, cujo seu texto pode desprezado, isto é, não queremos processar algo como TITLE. Além disso, existem as entidades HTML, que podem ser interessantes, ou não, e normalmente têm que ser traduzidas, nesse caso um ´, por exemplo, deve virar um 'á' ou um 'a', de acordo com o que é necessário para o sistema.

Os padrões da Internet/WWW

Os padrões da Internet são estabelecidos atualmente pela Internet Engineering Task Force, sendo denominados RFC, de *Request for Comments*, e numerados sequencialmente (IETF, 2021).

A maioria dos padrões relacionados a Web, ou ao WWW, é estabelecida pelo *World Wide Web Consortium* (W3C) W3C, “an international community where member organizations, a full-time staff, and the public work together to develop Web standards. Led by Web inventor and Director Tim Berners-Lee and CEO Jeffrey Jaffe, W3C’s mission is to lead the Web to its full potential.” (W3C, 2021a).

Por exemplo, no arquivo HTML acima, as partes que devem ser trabalhadas incluem a palavra “Cabeçalho” e a frase “Um parágrafo simples”, mas o que fazer com “Título da Página”? Devemos considerá-lo ou não como parte do documento para indexação, aprendizado de máquina, ou ainda outra aplicação?

Uma página web geralmente possui um HTML complexo, gerado por um conjunto de programas e manipulação humana. A Figura 4.5 mostra um fragmento do HTML da página pessoal do autor, gerado por meio do WordPress.

```

.amaz_td {background:#dddddd;}
</style>
<meta name="description" content="This is the personnal site of Geraldo Xexéo."><meta
name="Keywords" content="Computing, Games, Software Engineering, Information Retrieval"><!--
personal --><meta name="Generator" content="--><!-- BOF: ./personal-templates/simple/themes.show --
--><!-- EOF: ./personal-templates/simple/themes.show --><!--<BASE HREF="http://162.215.248.21/
index.html"-->
<title></title>
</head>
<body onload="" data-gr-c-s-loaded="true"><!-- BOF: layouts/personal/L15/new/splash.html -->
<table border="0" cellpadding="0" cellspacing="0" class="t_height" dir="LTR" style="background:
url()" width="100%">
<tbody>
<tr>
<td height="101" valign="top">
<table border="0" cellpadding="0" cellspacing="0" width="100%">
<tbody>
<tr>
<td align="left" width="492"></td>
<td align="right" height="79" style="background-image:url(widgets/gen_90.1.gif)">
</td>
</tr>
<tr>

```

Figura 4.5: Imagem parcial do HTML usado em uma página pessoal

4.4 HTML 5

HTML 5 complica as coisas um pouco, porque é uma junção de HTML, Javascript, CSS e XML. Isso faz com que não só que seja mais difícil fazer o *parsing* do arquivo, como também implica em decidir o que deve e o que não deve ser aceito como texto, de acordo com necessidades específicas.

Parte da mudança da mentalidade da Web ao longo do tempo foi que antes o navegador só era responsável por tarefas básicas de visualização. Hoje é um ambiente de execução. A Figura 4.6 mostra a evolução de HTML ao longo do tempo até a versão inicial do HTML 5 (WHATWG, 2021), com as principais influências para o desenvolvedor, e que continua recebendo adições, principalmente no CSS (W3C, 2021b).

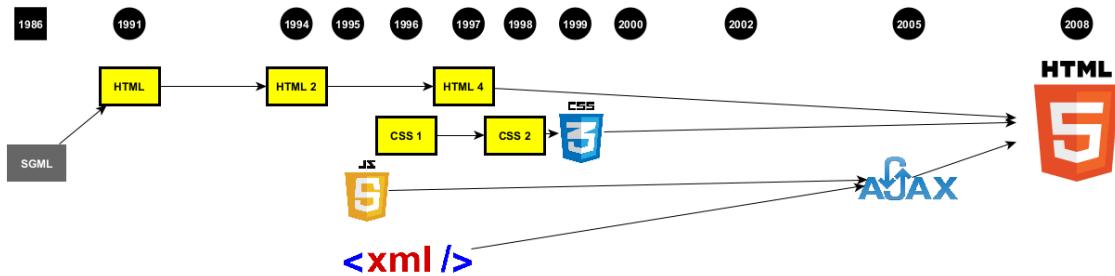


Figura 4.6: Linha do tempo do HTML ao HTML 5.

Já a Figura 4.7 dá uma visão geral da plataforma Web (WHATWG, 2021).

4.4.1 Limpando HTML de arquivos com Python

Para se obter o texto que está contido em um arquivo HTML, qualquer que seja sua versão, é necessário então limpar, ou filtrar, a linguagem de marcação do arquivo.

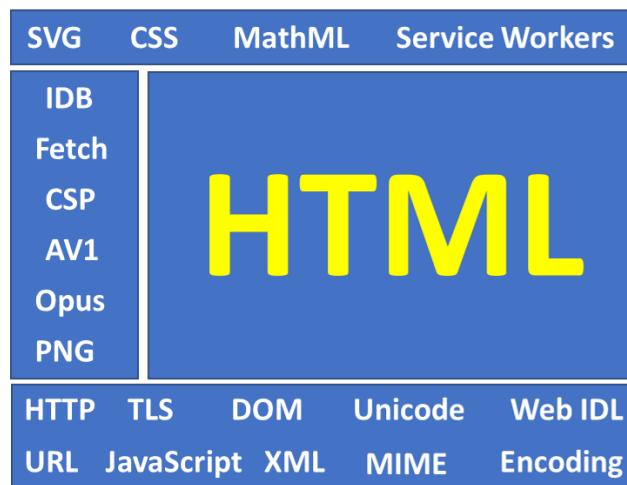


Figura 4.7: Visão geral da plataforma Web, segundo WHATWG (2021)

Existem várias opções para fazer essa limpeza. `lxml` é um pacote Python para tratar XML e HTML. Com ele é possível eliminar toda a marcação com facilidade, como no Programa 4.1, e obter como resultado uma string contendo o texto.

Programa 4.1: Programa simples para obter todo o texto dentro de um arquivo HTML

```

1 import lxml
2
3 f = open('processahtmlsimplesexemplo.html', 'r', encoding='utf-8')
4 origin = f.read()
5
6 html = lxml.html.fromstring(origin)
7 text = str(html.text_content())
8
9 print(text.strip())

```

A resposta, apresentada a seguir, remove os rótulos HTML e trata as entidades:

1 Título da Página	Saída do Programa 4.1
2	
3	
4	
5 Cabeçalho	
6 Um parágrafo simples	
7	
8 Usando entidades HTML ↴	

4.4.2 Limpando HTML de arquivos com KNIME

Usando as extensões é possível desenhar um workflow KNIME que lê um arquivo indicado por uma URL e depois remove todo o markup dele, como é demonstrado na sequência de sequências de Figuras 4.8, 4.9, 4.10, 4.11, 4.12.

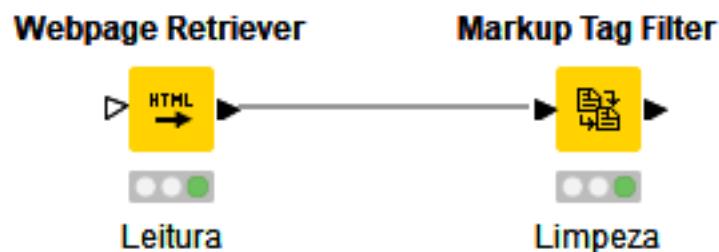


Figura 4.8: Workflow para ler, via URL, e limpar um arquivo HTML

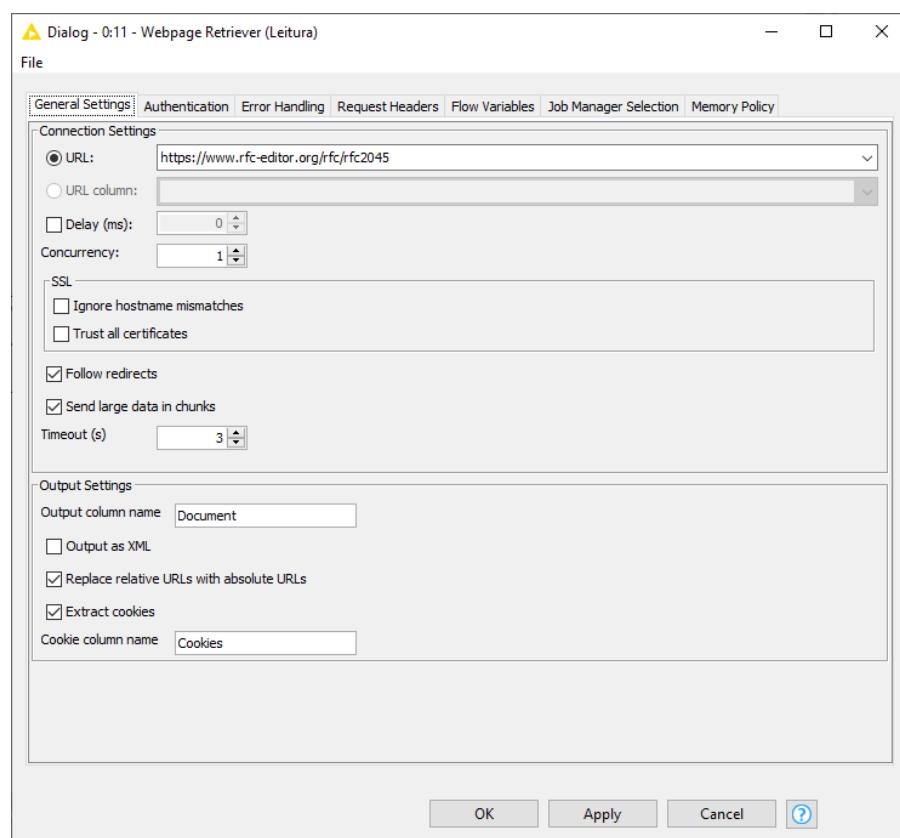


Figura 4.9: Recuperando uma página via sua URL.

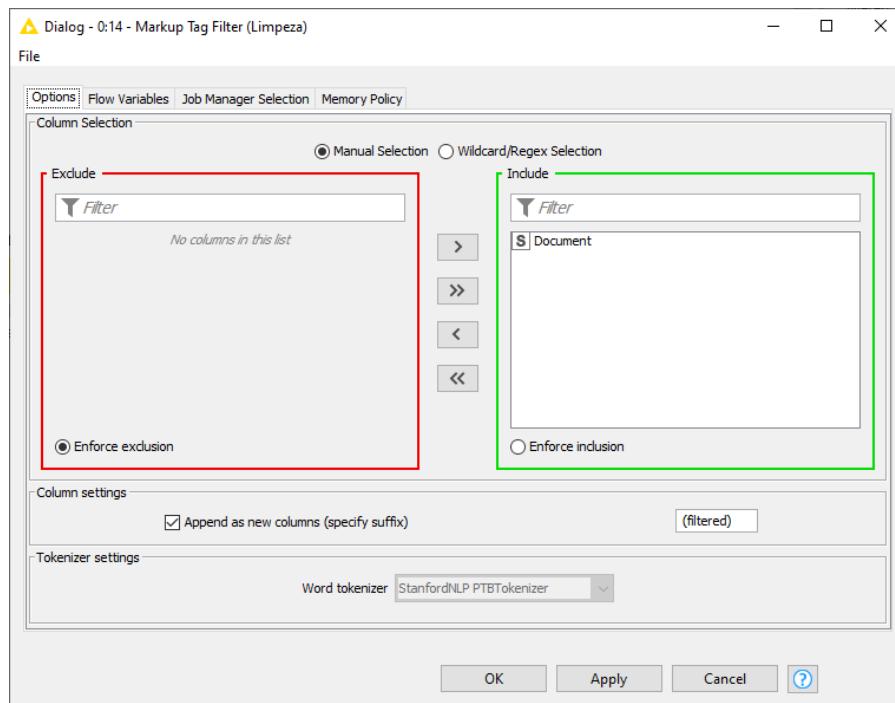


Figura 4.10: Configurando a remoção de tags de markup

Webpage Retriever results - 0:11 - Webpage Retriever (Leitura)	
File Edit Hilite Navigation View	
Table "default" - Rows: 1 Spec - Columns: 2 Properties Flow Variables	
Row ID	S Document
Row0	<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"> <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en"> <head profile="http://dublincore.org/documents/2008/08/04/dc-html"/> <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/> <meta name="robots" content="index,follow"/> <meta name="creator" content="rfcmarkup version 1.129b"/>

Figura 4.11: Resultado recuperado pelo nó de Leitura.

Preprocessed strings/documents. - 0:14 - Markup Tag Filter (Limpeza)			
File Edit Hilite Navigation View			
Table "default" - Rows: 1 Spec - Columns: 3 Properties Flow Variables			
Row ID	S Document	[...]	S Document (filtered)
Row0	<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 ... <html xmlns="http://www.w3.org/1999/xhtml" xml:... <head profile="http://dublincore.org/documents/2... <meta http-equiv="Content-Type" content="text/... <meta name="robots" content="index,follow"/> <meta name="creator" content="rfcmarkup versio...	?	RFC 2045: Multipurpose Internet Mail Extensions (MIME) Part One... Request for Comments: 2045 Microsoft Obsoletes: 1521, 1522, 1590 N. Borenstein Category: Standards Track First Virtual November 1996

Figura 4.12: Resultado pós remoção do markup.

4.5 Outros formatos específicos

Ao longo do tempo, provavelmente milhares de formatos foram desenvolvidos contendo texto, alguns deles certamente perdidos. Por exemplo, durante os anos 1980 se utilizava no Brasil o processador de texto Carta Certa, que saiu de linha em meados da década de 90. Já em torno de 2010, em um projeto que participei, foi impossível ler um arquivo em seu formato.

Mais informação sobre vários formatos de arquivo pode ser obtida em <https://docs.fileformat.com/word-processing/> e outros sites da internet.

4.6 Formatos de impressão

Formatos de impressão são arquivos utilizados primariamente para impressão, mesmo que possam ser compreendidos por seres humanos. A Adobe chama essas linguagens de Linguagem de Descrição de Página (Adobe Systems Inc., 1999), e as divide entre linguagens de formato estático e de formato dinâmico.

A principal linguagem de impressão é PostScript, porém, hoje em dia, o principal formato usado não é o PostScript (PS, arquivos .ps), mas sim o Page Description Format (PDF, arquivos .pdf), que utiliza um subset de PostScript para descrever páginas, dentro de um esquema de arquivo específico, composto de objetos.

4.6.1 PostScript

PostScript é uma linguagem de programação interpretada baseada em pilha com capacidades gráficas, criada com a finalidade de “descrever a aparência de textos, formas gráficas e imagens em páginas impressas ou mostradas em um monitor” (Adobe Systems Inc., 1999) em dispositivos de saída do tipo *raster*. Ele é uma linguagem de formato dinâmica, Turing completa (Adobe Systems Inc., 1999).

Um exemplo que imprime “Hello World!” em PostScript seria.

```
/Helvetica-Bold findfont 26 scalefont setfont
20 40 moveto
.5 setgray
(Hello World!) show
```

Apesar do PS ainda ser fortemente usado como formato de comunicação com impressoras, a maioria dos arquivos digitais atualmente usa o formato PDF.

4.6.2 PDF

Portable Document Format (ou **PDF**) é um formato de arquivo, desenvolvido pela Adobe Systems em 1993, para representar documentos de maneira independente do aplicativo, hardware, e sistema operacional usados para criá-los. Um arquivo PDF pode descrever documentos que contenham texto, gráficos e imagens num formato independente de dispositivo e resolução. Além disso, o formato PDF evolui para permitir recursos interativos, links e formulários.

A versão mais recente quando este texto foi escrito é o **PDF 2.0**, ou **ISO 32000-2:2020**, que oferece entre outras características:

- descrição de página, incluindo texto, gráficos, imagens e outros elementos;
- suporte a fontes embutidas no arquivo;
- compressão de imagens;
- segurança, incluindo senhas e assinatura digital, e
- comentários e recursos iterativos.

Os blocos básicos de um arquivo PDF são chamados de objetos. Além de um objeto nulo, existem também os tipos: booleano, numérico (inteiro e real), nomes, strings, vetores, dicionários, árvores de nomes e árvores de números, *streams*. Em especial, o conteúdo de uma página é guardado dentro de um *content stream*, cuja sintaxe é derivada de PostScript (Rosenthal, 2013).

Um arquivo PDF tem 4 seções, o *Header*, o *Body*, a *Cross-reference Table* e o *Trailer*. O *Header* identifica o arquivo como PDF e dá sua versão, iniciando com %PDF-. O *Trailer* é principalmente um dicionário do tipo chave-valor com informações sobre o documento. O *Body* é onde se encontram os objetos, organizados de uma forma específica. Finalmente, na *Cross-reference Table* está um dos conceitos principais do arquivo PDF: um ponteiro para cada objeto no corpo do arquivo. Após o cabeçalho, as três seções seguintes podem ser atualizadas por meio da criação de três novas seções com os novos dados Rosenthal (2013).

Tendo em vista a complexidade de um arquivo PDF, é bastante incomum que sejam tratados diretamente, sendo que a abordagem mais usada é extrair o texto por meio de um pacote como o `pdfminer.sex` (Shinyama, Guglielmetti e Marsman, 2019).

4.6.3 Extraindo texto do PDF em Python

Vários pacotes ajudam a extrair o texto de um arquivo PDF, como o citado `pdfminer.sex`. O Programa 4.2 mostra uma forma simples de obter todo o texto de um arquivo pdf. Já o Programa 4.3 mostra o uso de funções específicos, dando maior controle ao programador.

Programa 4.2: Usando o `pdfminer.sex` para extrair texto de um arquivo PDF

```

1 import pdfminer.high_level as pdfh
2
3 text = pdfh.extract_text("arquivo.pdf")
4
5 print(len(text))

```

Programa 4.3: Usando uma forma mais complexa de extrair texto de um arquivo PDF

```

1 from io import StringIO
2
3 from pdfminer.converter import TextConverter
4 from pdfminer.layout import LAParams
5 from pdfminer.pdfdocument import PDFDocument
6 from pdfminer.pdfinterp import PDFResourceManager, PDFPageInterpreter
7 from pdfminer.pdfpage import PDFPage
8 from pdfminer.pdfparser import PDFParser
9
10 output_string = StringIO()

```

```
11 with open("arquivo.pdf", 'rb') as in_file:  
12     parser = PDFParser(in_file)  
13     doc = PDFDocument(parser)  
14     rsrccmgr = PDFResourceManager()  
15     device = TextConverter(rsrccmgr, output_string, laparams=LAPParams())  
16     interpreter = PDFPageInterpreter(rsrccmgr, device)  
17     for page in PDFPage.create_pages(doc):  
18         interpreter.process_page(page)  
19  
20 print(output_string.getvalue())
```

4.6.4 Extraindo texto do PDF em KNIME

Também não é difícil extrair o texto de um documento PDF usando o KNIME, já que ele possui o bloco **PDF Parser**. A Figura 4.13 mostra um *workflow* simples que lê e permite visualizar o documento gerado. Já a Figura 4.16 mostra a configuração do bloco.

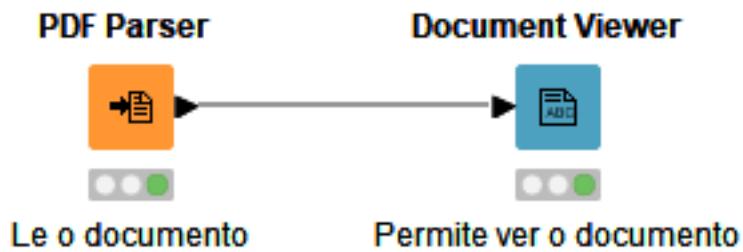


Figura 4.13: Workflow para ler e transformar um arquivo PDF em um Document do KNIME

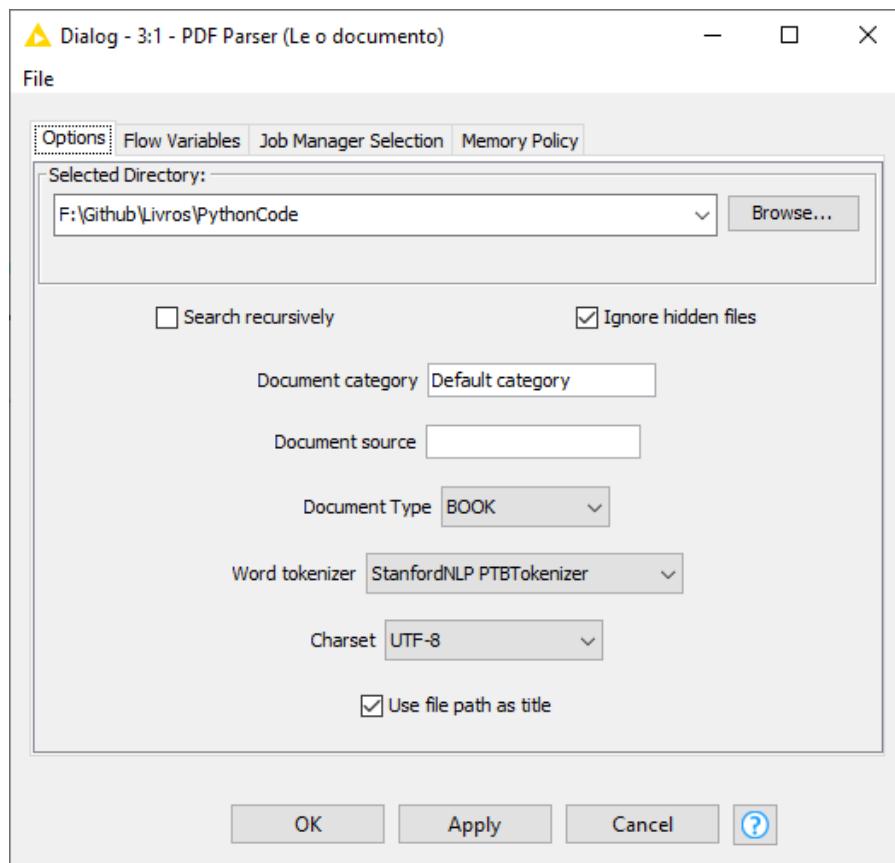


Figura 4.14: Configuração do nós PDF Parser

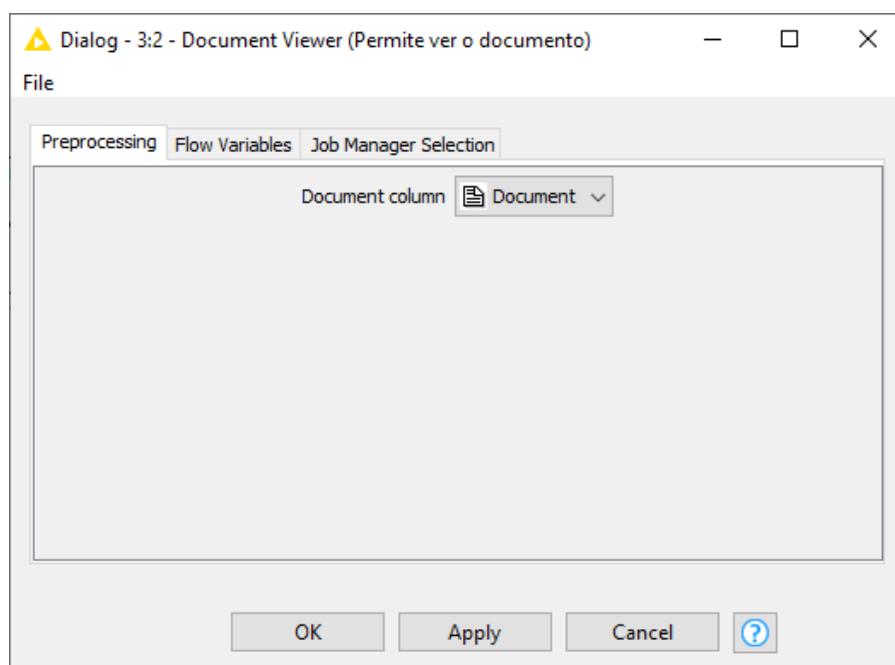


Figura 4.15: Configuração do nó Document Viewer

Document View - 3:2 - Document Viewer (Permite ver o documento)				
File				
Quick Search:		Document Title	Aut...	Sou...
#	Document Title	Aut...	Sou...	Category
1	F:\Github\Livros\PythonCode\arquivo.pdf			Default category

Figura 4.16: Vendo a lista de documentos lidos

Document Details

F:\Github\Livros\PythonCode\arquivo.pdf

Search ▶

Document information ▶

Authors ▶

Meta information ▶

o período de tempo que dura o empréstimo ou a aplicação financeira, podendo ser medido em dias, meses, anos... • prestação ou pagamento (P, PGT, PMT) se refere ao valor de pagamentos quando feitos em um número maior do que 1. • taxa de juros (i), ou simplesmente taxa, é o quociente entre juros e o principal, ver equação 1.2; $S = P + J(1+i)$
 $J = (1+i)P - P$ 21 Analisando essas definições podemos detectar uma propriedade importante do dinheiro e que é a base da Matemática Financeira: o dinheiro muda de valor ao longo do tempo. Normalmente, como os cenários de inflação são muitas mais comuns que os de deflação, R\$100,00 hoje compram mais que comprariam daqui a um ano e menos do que compravam um ano atrás. Por meio de cálculos como o do Valor Presente, como veremos no caso do Valor Presente Líquido, podemos comparar opções de investimentos ou empréstimos. Calculadoras financeiras, como a HP-12C e o Excel permitem calcular facilmente o Valor Presente e o Valor Futuro com pagamentos, ou retiradas, fixas, em um período determinado, ao com uma taxa de juros fixas. Por exemplo, se você tiver R\$10.000,00 por mês e puder aplicar em um projeto, deve procurar uma boa alternativa de investimento para poder saber se, financeiramente, vale a pena usar o dinheiro no projeto, ou se é melhor usar esse investimento. Com a fórmula do Valor Futuro (VF em português ou FV em inglês) esse cálculo pode ser facilmente feito no Excel, como mostrado na Figura 1.7. (a) Valores e formatação (b) Fórmulas Figura 1.7: Cálculo do valor futuro de um investimento de R\$10.000,00 por mês com uma taxa de juros de 1%. Existem várias práticas que são utilizadas para calcular o valor financeiro de um projeto, entre elas (Satpathy et al., 2016): • Custo Total de Propriedade (TCO - Total Cost of Ownership), que calcula todo custo de um serviço ou produto, incluindo a realização e a uso do mesmo. • Retorno do Investimento (ROI - Return on Investment), o método mais usado que é basicamente o lucro sobre o custo; • Valor Presente Líquido - VPL (NPV - Net Present Value), onde se calcula a diferença entre a receita do projeto e seus custos ao longo do tempo, corrigido para o tempo presente, e 22 Tabela 1.3: Valores usados para calcular o TCO de uma impressora. Impressora Laser A Ink A Laser B Preço R\$

Figura 4.17: Vendo o detalhe de um documento, principalmente seu conteúdo textual.

4.7 XML

XML, de *eXtensible Markup Language*³ (W3C, 2008), é um formato projetado para armazenar e transferir conteúdo, de forma legível para humanos, mas não necessariamente fácil de entender. Muitas coleções de documentos usadas em experimentos, como vários Corpora, são fornecidos em XML.

É interessante notar que o XML usado atualmente é o XML 1.0 (W3C, 2008), cuja versão mais nova é mais recente que a versão mais nova do XML 1.1 (W3C, 2006).

A linguagem possui um conjunto de *tags* fixas, mas em cada arquivo a grande parte delas é definida arbitrariamente pelo criador, sendo algumas vezes descritas em esquemas. A Figura 4.18 mostra um arquivo desse tipo.

```
<?xml version="1.0" encoding="UTF-8"?>
<breakfast_menu>
<food>
    <name>Belgian Waffles</name>
    <price>$5.95</price>
    <description>
        Two of our famous Belgian Waffles with plenty of real maple syrup
    </description>
    <calories>650</calories>
</food>
<food>
    <name>Strawberry Belgian Waffles</name>
    <price>$7.95</price>
    <description>
        Light Belgian waffles covered with strawberries and whipped cream
    </description>
    <calories>900</calories>
</food>
<food>
    <name>Berry-Berry Belgian Waffles</name>
    <price>$8.95</price>
    <description>
        Belgian waffles covered with assorted fresh berries and whipped cream
    </description>
    <calories>900</calories>
</food>
```

Figura 4.18: Exemplo de fragmento de arquivo xml.

Um dos problemas de tratar XML é que os dados podem aparecer tanto dentro das *tags* como na forma de texto livre entre o início e o fim de uma marcação.

Um arquivo XML é considerado bem-formado quando segue as regras de sintaxe da linguagem, e válido quando segue as regras estabelecidas em um esquema, isto é, é validado frente a um esquema. Esse esquema pode ser definido em *XML Schema Definition*, um arquivo XSD (W3C, 2006, 2012), ou em um formato mais antigo, o *Document Type Definition*, o DTD. XSD usa a própria XML para definir o esquema de outro arquivo XML, enquanto DTD usa um formato diferente.

Por exemplo, um arquivo XML, como o apresentado no Programa 4.4, pode mostrar uma mensagem. Ele segue as regras sintáticas de XML:

³A pronúncia em inglês é ex-em-el.

- tem um elemento raiz (mensagem);
- todos os elementos tem um *tag* que os fecha (com a barra no início).
- os *tags* são diferenciam maiúsculas de minúsculas (apensa subtendido no exemplo).
- os elementos estão hierarquizados em árvore corretamente, e
- os valores de atributo estão entre aspas.

Programa 4.4: Exemplo de arquivo XML usando um DTD.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE mensagem SYSTEM "Mensagem.dtd">
3 <mensagem>
4 <para>Alice</para>
5 <de>Bob</de>
6 <titulo>Lembrete</titulo>
7 <corpo>Não esqueça a sua promessa!</corpo>
8 </mensagem>
```

Como segue as regras da sintaxe, o Programa 4.4 é um arquivo bem formado. Mas será ele válido? Para ser válido, deve usar, da forma correta, apenas os elementos descritos no DTD a que ele se refere, como o da Programa 4.5.

Programa 4.5: Exemplo de arquivo DTD.

```

1 <!DOCTYPE mensagem
2 [
3   <!ELEMENT mensagem (para,de,titulo,corpo)>
4   <!ELEMENT para (#PCDATA)>
5   <!ELEMENT de (#PCDATA)>
6   <!ELEMENT titulo (#PCDATA)>
7   <!ELEMENT corpo (#PCDATA)>
8 ]
```

É possível notar, no Programa 4.5, que não é permitida a repetição de qualquer elemento. Outra maneira, mais moderna, de descrever as mesmas regras que estão nesse arquivo é o Programa 4.6.

Programa 4.6: Exemplo de arquivo XSD.

```

1 <xsschema attributeFormDefault="unqualified" elementFormDefault="qualified"
  ↳ xmlns:xss="http://www.w3.org/2001/XMLSchema">
2 <xselement name="mensagem">
3 <xsccomplexType>
4 <xsequence>
5 <xselement type="xss:string" name="para"/>
6 <xselement type="xss:string" name="de"/>
7 <xselement type="xss:string" name="titulo"/>
8 <xselement type="xss:string" name="corpo"/>
9 </xsequence>
10 </xsccomplexType>
11 </xselement>
12 </xsschema>
```

4.8 Programas Exemplo

Programa 4.7: Funções de apoio, arquivo birtutils

```

1 import glob
2
3 def fix_path(path):
4     if path[-1] != "/":
5         return path + "/"
6     else:
7         return path
8
9 def get_all_files(path,extension,recursive=False):
10    if recursive:
11        return glob.glob(fix_path(path)+"**/*."+extension,recursive=True)
12    else:
13        return glob.glob(fix_path(path)+"*."+extension)

```

Programa 4.8: Lendo arquivos do corpus Folha, resulta em um dicionário com id dos artigos e
indicando o texto sem tratamento.

```

1 from bs4 import BeautifulSoup
2 import birtutils as bu
3
4
5 def grab_folha_docs(file):
6     all_data = {}
7     with open(file,"r",encoding="utf_8") as f:
8         text = f.read()
9         data_as_xml = BeautifulSoup(text,"xml")
10        the_docs = data_as_xml.find_all('DOC')
11        for d in the_docs:
12            all_data[d.find('DOCNO').text] = d.find("TEXT").text
13    return all_data
14
15
16 if __name__ == "__main__":
17
18     path = '<path to folder>'
19
20     print(path)
21     path = bu.fix_path(path)
22
23     print(path)
24
25     all_data = {}
26     for fn in bu.get_all_files(path,"sgml",recursive=True):
27         all_data |= grab_folha_docs(fn)

```

4.9 Exercícios

4.9.1 Questões Objetivas

Exercício 4.1:

Qual o principal objetivo de conhecer os formatos de arquivos textuais?

1. Converter arquivos PDF em imagens.
2. Transformar conteúdo em código HTML.
3. Compreender métodos para transformar arquivos em texto simples para processamento.
4. Aprender a programar em diferentes linguagens de marcação.
5. Desenvolver novos formatos de arquivo para uso pessoal.

Exercício 4.2:

Que tipos de arquivos são normalmente usados em tarefas de processamento de texto?

1. Arquivos de áudio e vídeo.
2. Arquivos executáveis e binários.
3. Arquivos de texto simples, Word, PDF, HTML, XML ou JSON.
4. Imagens digitais e fotografias.
5. Arquivos comprimidos e arquivos zip.

Exercício 4.3:

Quais são os três tipos principais de formatos de arquivo mencionados?

1. Formatos de edição, marcação e impressão.
2. Formatos de áudio, vídeo e texto.
3. Formatos criptografados, comprimidos e ocultos.
4. Formatos dinâmicos, estáticos e interativos.
5. Formatos de código aberto, proprietários e personalizados.

Exercício 4.4:

O que é característico dos formatos de edição?

1. Podem ser editados apenas por softwares especiais.
2. Não contêm texto, apenas gráficos e imagens.
3. São usados para criptografar informações.
4. São chamados de arquivos texto e normalmente têm extensão ".txt".
5. São incompatíveis com a codificação Unicode.

Exercício 4.5:

Qual a principal diferença entre formatos de marcação e formatos de impressão?

1. Formatos de marcação são apenas para impressão, enquanto formatos de impressão são para leitura em tela.
2. Formatos de marcação são legíveis por humanos e editáveis, enquanto formatos de impressão são destinados à impressão ou visualização.
3. Formatos de impressão contêm apenas imagens, enquanto formatos de marcação contêm apenas texto.

4. Formatos de marcação usam criptografia, enquanto formatos de impressão não.
5. Formatos de impressão são desenvolvidos exclusivamente para a web, enquanto formatos de marcação são para documentos offline.

Exercício 4.6:

Quais linguagens de marcação são comumente utilizadas?

1. C++, Java e Python.
2. HTML, XML, SGML, LaTeX, RTF, Open XML Document e OpenDocument.
3. PostScript, PDF e Markdown.
4. JPEG, PNG e GIF.
5. Bash, PowerShell e CMD.

Exercício 4.7:

O que define um arquivo XML como bem formado?

1. Conter apenas elementos gráficos.
2. Seguir as regras de sintaxe da linguagem XML.
3. Estar criptografado e seguro.
4. Ser validado contra um esquema DTD ou XSD.
5. Ter uma única linha de texto sem marcações.

Exercício 4.8:

Qual formato é derivado do PostScript e utilizado para descrever documentos?

1. HTML.
2. XML.
3. PDF.
4. CSV.
5. RTF.

Exercício 4.9:

Qual ferramenta pode ser utilizada para extrair texto de um arquivo PDF em Python?

1. Microsoft Word.
2. Adobe Photoshop.
3. pdfminer.six.
4. Google Docs.
5. LaTeX compiler.

Exercício 4.10:

O que permite a linguagem PostScript?

1. Descrever a aparência de formas gráficas e imagens em páginas impressas.
2. Codificar vídeos para transmissão na web.
3. Criar sistemas operacionais.
4. Comprimir arquivos de texto para economizar espaço.
5. Traduzir documentos automaticamente entre diferentes idiomas.

4.9.2 Questões Discursivas

Exercício 4.11:

Discuta como os formatos de arquivo influenciam o processamento de texto na web. Considerando o papel do HTML e do XML, como esses formatos facilitam ou dificultam a extração de informações relevantes dos documentos online?

Exercício 4.12:

Explique a importância do Unicode para a representação de texto em diferentes idiomas na internet. Como esse padrão contribui para a globalização da comunicação e do conteúdo digital?

Exercício 4.13:

Descreva o processo de conversão de um documento em formato PDF para um formato de texto simples. Quais são os principais desafios envolvidos nesse processo e como eles podem ser superados?

Exercício 4.14:

Discuta as diferenças entre formatos de edição, marcação e impressão de arquivos textuais. Como a escolha do formato afeta a maneira como os dados são apresentados, armazenados e transmitidos?

Exercício 4.15:

Explique o conceito de linguagens de marcação e sua aplicação no desenvolvimento web e na gestão de documentos. Quais são as vantagens e desvantagens de usar uma linguagem de marcação em comparação com um formato de edição simples?

4.9.3 Questões de Programação

Exercício 4.16:

Escreva um programa em Python que leia um arquivo de texto (.txt) e conte o número de palavras no arquivo. Considere uma palavra como qualquer sequência de caracteres delimitada por espaços em branco, quebras de linha ou pontuação.

Exercício 4.17:

Desenvolva um programa em Python que leia um arquivo HTML e extraia todos os links (elementos dentro de `...`). O programa deve imprimir na tela cada URL encontrada no documento HTML.

Exercício 4.18:

Crie um programa em Python que converta um arquivo de texto (.txt) para o formato CSV. O programa deve ler um arquivo de texto onde cada linha representa um campo e cada registro começa com a palavra “#REGISTRO-INICIO” e termina com a palavra “#REGISTRO-FIM”. O programa deve criar um arquivo CSV onde cada linha corresponde a um registro e os campos são delimitados por vírgulas. O programa deve tratar registros de tamanho diferente de forma que todos os registros tenham o tamanho do maior registro.

Exercício 4.19:

Escreva um programa em Python que converta um arquivo XML para o formato CSV. O arquivo XML segue um esquema fixo com campos baseados no padrão Dublin Core. Cada registro no XML representa um item de mídia digital, com os seguintes campos: título (`<dc:title>`), criador (`<dc:creator>`), data (`<dc:date>`) e descrição (`<dc:description>`). O programa deve ler o arquivo XML e criar um arquivo CSV onde cada linha corresponde a um item, com os campos delimitados por vírgulas. Suponha que o XML tenha a seguinte estrutura:

```
<catalog>
<item>
<dc:title>Exemplo de Título</dc:title>
<dc:creator>Nome do Autor</dc:creator>
<dc:date>2024-03-14</dc:date>
<dc:description>Descrição do item.</dc:description>
</item>
<!-- Mais itens -->
</catalog>
```

Exercício 4.20:

Escreva um programa em Python para converter metadados de documentos no formato Dublin Core para o formato BibTeX, considerando que todos os documentos sejam do tipo livro. Cada registro Dublin Core deve ser convertido para uma entrada `@book` em BibTeX, incluindo os seguintes campos Dublin Core: título (`dc:title`), autor (`dc:creator`), data de publicação (`dc:date`), e editora (`dc:publisher`). O programa deve ler um arquivo XML contendo os metadados Dublin Core e gerar um arquivo `.bib` contendo as entradas BibTeX correspondentes.

Exemplo de entrada (XML):

```
1      <catalog>
2      <book>
3          <dc:title>Exemplo de Título</dc:title>
4          <dc:creator>Nome do Autor</dc:creator>
5          <dc:date>2024</dc:date>
6          <dc:publisher>Editora Exemplo</dc:publisher>
7      </book>
8      <!-- Mais livros -->
9  </catalog>
```

Exercício 4.21:

Escreva um programa em Python que leia um arquivo CSV contendo dados de publicações em formato Dublin Core e gere um histograma mostrando a distribuição do número de publicações por ano. O arquivo CSV terá uma linha para cada publicação com as seguintes colunas: título (`dc:title`), autor (`dc:creator`), data de publicação (`dc:date`), e editora (`dc:publisher`). Assuma que a coluna `dc:date` contém apenas o ano de publicação. O histograma deve ter no eixo x os anos e no eixo y o número de publicações nesse ano.

Exemplo de entrada (CSV):

```
"dc:title","dc:creator","dc:date","dc:publisher"
"Exemplo de Título 1","Nome do Autor 1","2024","Editora Exemplo 1"
"Exemplo de Título 2","Nome do Autor 2","2023","Editora Exemplo 2"
"Exemplo de Título 3","Nome do Autor 3","2024","Editora Exemplo 3"
```

Solução:

Para resolver essa questão, você pode usar a biblioteca pandas para ler o arquivo CSV e matplotlib para gerar o histograma.

```
import pandas as pd
import matplotlib.pyplot as plt

# Lendo o arquivo CSV
df = pd.read_csv('publicacoes.csv')

# Criando um histograma de publicações por ano
plt.figure(figsize=(10, 6))
df['dc:date'].value_counts().sort_index().plot(kind='bar')
plt.title('Histograma de Publicações por Ano')
plt.xlabel('Ano de Publicação')
plt.ylabel('Número de Publicações')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Este programa primeiro lê os dados do arquivo CSV para um DataFrame do pandas. Em seguida, conta o número de publicações por ano usando o método `value_counts()` e ordena os anos em ordem crescente com `sort_index()`. Finalmente, usa o matplotlib para criar e exibir um histograma com os dados.

Exercício 4.22:

Escreva um programa em Python que leia um arquivo BibLaTeX, extraia informações relevantes de cada publicação e crie um documento XML no formato Dublin Core. O programa deve ser capaz de lidar com vários campos BibLaTeX, incluindo, mas não se limitando a: `@book`, `@article`, `@inproceedings`, e seus respectivos campos como `author`, `title`, `year`, `journal`, `publisher`, entre outros. O objetivo é gerar um documento XML que represente essas publicações no formato Dublin Core. Use uma biblioteca Python de leitura de arquivos BibTeX.

Exemplo de entrada (BibLaTeX):

```
@book{examplebook,
author    = {Nome do Autor},
title     = {Título do Livro},
publisher = {Editora do Livro},
year      = {2024},
address   = {Local da Publicação}
```

}

```
@article{examplearticle,  
  author  = {Nome do Autor},  
  title   = {Título do Artigo},  
  journal = {Nome do Periódico},  
  year    = {2023},  
  volume  = {1},  
  number  = {1},  
  pages   = {1-10}  
}
```


Parte II

Pré-Processamento e Algorimos Básicos

CAPÍTULO 5

PRÉ-PROCESSAMENTO

Após o processamento do arquivo original, por exemplo em HTML ou PDF, um texto está representado como um arquivo simples, que pode ser lido por um humano, contendo caracteres codificados em UTF-8, possivelmente contendo marcações ou outros artefatos característicos de um ou outro formato. Nesse caso duas atividades básicas devem ser feitas: a limpeza do texto para obter apenas uma sequência válida de palavras que queremos processar, ou seja, a conversão em texto puro, e a alteração desse texto puro para uma coleção de termos, de forma a melhorar o funcionamento do sistema sendo construído.

Tarefas adicionais podem incluir entender a organização das palavras ou re-organizá-las de alguma forma, por exemplo, quando o arquivo original as posiciona fora da sequência em que aparece no arquivo (como é possível fazer, por exemplo, com arquivos PostScript).

A lista de tarefas que compõe o pré-processamento de texto que podem ser necessárias é grande e inclui:

- eliminação de símbolos indesejados no processamento, como TABs, CRs e LFs;
- eliminação de comandos ou marcações que não pertencem ao texto, como marcações HTML
- eliminação ou tradução de códigos que representam imagens, como , que não pertencem a língua sendo processada, ou que, em sequência, possuem um significado específico, como “:-)” para um rosto sorrido.
- correção de erros de digitação ou de ortografia
- eliminação de acentos e variações aplicadas às letras, como diacríticos, ou criação de representações alternativas, de modo a atender erros ;
- transformação para minúsculas ou maiúsculas, como desejado;
- tokenização, i.e., separação do documento em tokens, ou palavras;
- eliminação das conjugações verbais;
- eliminação das variações de gênero, número, etc;
- lematização ou radicalização (*stemming*).

5.1 Noções de morfologia da palavra

Palavras recebem, a partir de um elemento central, flexões que as caracterizam em categorias gramaticais. Nas línguas, em geral, existem muitas formas de flexões, nem todas existentes em português:

- Modo indicativo, subjuntivo, imperativo, interrogativo, injuntivo, optativo, potencial, jussivo, cohortativo, deôntico (comissivo, diretivo, volitivo), etc.;
- Tempo - presente, pretérito, futuro, não pretérito, não-futuro;
- Voz - ativa, passiva, reflexiva, média;
- Aspecto - perfeito, imperfeito, mais que perfeito, contínuo, progressivo, habitual;
- Caso (não usado em português) - dativo, locativo, subjetivo, nominativo, acusativo, genitivo, objetivo, possessivo;
- Pessoa - primeira, segunda, terceira;
- Número - singular, plural, dual, trial, quadral, paucal menor, paucal maior;
- Gênero – masculino, feminino, neutro;
- Grau (adjetivo) – comparativo, superlativo, e
- Grau (substantivo) - aumentativo, diminutivo.

Essas flexões existem como conjugações, característica dos verbos, e as declinações, características dos substantivos, adjetivos e pronomes. Essas partes são elementos mórficos ou morfemas.

Assim, um **morfema** é um elemento constituinte da palavra, que pode ser (Lucca e Graças Volpe Nunes, 2002):

- **raiz**, o elemento mais simples a que a palavra pode ser reduzida, obtido quando todos os outros elementos são eliminados, por exemplo “bandon” para “abandonar”;
- **tema**, “contuído pelo radical mais uma vogal temática à qual são acrescentadas as desinências casual para substantivos e adjetivos, e verbal para os verbos” (Lucca e Graças Volpe Nunes, 2002);
 - **radical**, o elemento que dá o significado básico da palavra, como “pedr” para “pedra”, “pedrinha”, “pedregulho” e “apedrejar”;
 - **vogal temática**, um elemento somado ao tema para que ele se junta com outros morfemas, podendo ser nominais ou verbais;
- **afixos**, elementos que são agregados a uma raiz ou radical que alteram o sentido da palavra;
 - **prefixos**, os afixos agregados no início da palavra;
 - **sufixos**, os afixos agregados no fim das palavras;
- **desinência**, elementos que indicam as flexões da palavra;
 - **desinência nominais**, indicam gênero e número (em português);
 - **desinência verbal**, indicam tempo, modo, número, pessoa e formal nominal do verbo.

5.1.1 Lematização e Stemming

Um **lema**, ou **forma canônica**, é a forma dicionarizada de uma palavra. No português o lema é o singular masculino, para substantivos e adjetivos, e o infinitivo para os verbos.

Já o termo em inglês *stem* é o lema ou o radical de uma palavra, assim, um *stemmer* deveria ser um algoritmo que, dada uma palavra, fornece o lema ou o radical da mesma (Lucca e Graças Volpe Nunes, 2002). O *stemmer* mais conhecido, porém, é o **Porter Stemmer** (Porter, 1980), que se autodenomina um “removedor de sufixos”. *Stemmers* normalmente fazem simplificações ou premissas práticas sobre o processo, e, por causa disso, podem agregar palavras que no dicionário não estariam agregadas, como

“casa” (onde moramos), “casar” (juntar pares) e “casamento” (o evento), mesmo que de certa forma tenham a mesma origem. Supõe-se que um lematizador, ao contrário do *stemmer* siga estritamente as regras da gramática.

Lematizar, então, é representar uma palavra pelo seu lema. Essa atividade existe no contexto lexicográfico (Lucca e Graças Volpe Nunes, 2002), como na criação de dicionários, e pode ser feita por um algoritmo, enquanto um *stemmer*, que não é usado no contexto lexicográfico, é normalmente um algoritmo aplicado a palavras que, em suas várias variações, resulta em um elemento comum que contém significado, que se segue a estratégia de Porter, remove apenas os sufixos.

5.2 Tokenização

A tarefa de **tokenização**¹ realiza a separação de textos contínuos em unidades chamadas *tokens* (Freitas, 2022). Segundo Christopher D. Manning, Raghavan e Schütze (2009a) “Dada um sequência de caracteres e uma unidade definida de documentos, tokenização é a tarefa de cortar ele em pedaços, chamados *tokens*, talvez jogando fora alguns caracteres como os de pontuação” (Christopher D. Manning, Raghavan e Schütze, 2009a). Um token pode ser: uma palavra; uma palavra composta; qualquer sinal que compõe o texto, como sinais de pontuação; emoticons; os símbolos que indicam moedas, como o ‘\$’ e ‘č’, etc.

Essa descrição faz que a tarefa pareça fácil, e realmente é uma das mais fáceis a princípio, porém, como tudo, aparecem dificuldades no mundo real. Isso se inicia com a definição de palavra (Freitas, 2022), que pode ser vista de vários modos: fonológico, morfossintático, ou semântico. Porém, a abordagem mais comum é a gráfica: “uma palavra é tudo que em um texto escrito aparece entre espaços em branco” (Freitas, 2022).

Mesmo assim, ainda há questões a serem tratadas. Por exemplo, como tokenizar uma palavra como “guarda-chuva”? E palavras compostas que não têm hífen, como “Rio de Janeiro”, conhecidas como expressões multipalavra? Em duas ou três palavras ou em uma só? E como tratar um hífen que separa a palavra entre duas linhas, para juntar suas partes? Que símbolos devem ser considerados como separadores de palavras? Como tratar todos os símbolos possíveis dentro de um conjunto de caracteres? E se forem usados comandos HTML para representar letras, como em ’ação’? Em que momento do tratamento das frases devemos tokenizar e ficar com as palavras? A pontuação será tokenizada ou desprezada?

Na prática, devido a tantos problemas, temos que ou nos dedicar a tokenização com muito detalhe, ou usar uma ferramenta que já faça isso. Por sorte, existem muitas. Andrade (2021) sugere cinco formas de fazê-lo em Python:

1. usar a função `split`, o que só funciona em casos simples;
2. usar o NLTK;
3. usar o sklearn (Pedregosa et al., 2011);
4. usar o spaCy (Honnibal e Montani, 2017), ou
5. usar o Gensim (Rehurek e Sojka, 2010);

O resultado, em Python, é normalmente uma lista de palavras, ou um vetor quando são usados pacotes adicionais. Assim, uma string como ’Essa é uma frase’, tokenizada resulta em uma lista [’Essa’, ’é’, ’uma’, ’frase’].

¹Essa tarefa já foi traduzida como atomização, porém esse nome é muito menos usado.

Não vamos tratar aqui da função `split` do Python porque ela já foi usada no exemplo do Programa 6.1 e só é capaz de usar um único caractere como separador. Deixaremos o uso do sklearn e do spaCy como exercícios.

Usando o NLTK (Bird, Loper e Klein, 2009), o processo é fácil, como apresentado no Programa 5.1, que produz o resultado que o segue.

Programa 5.1: Tokenização com o NLTK

```
1 text = "Um exemplo de texto, com vírgulas e palavras como guarda-chuva."
2 from nltk.tokenize import word_tokenize
3 tokens = word_tokenize(text)
4 print(tokens)
```

Saída do Programa 5.1

```
1 ['Um', 'exemplo', 'de', 'texto', ',', 'com', 'vírgulas',
2 'e', 'palavras', 'como', 'guarda-chuva', '.']
```

Tokenizadores diferentes podem ter abordagens diferentes de tokenização. O NLTK padrão nos dá palavras e pontuações, já usando o Gensim, por exemplo, ficamos só com as palavras.

Programa 5.2: Tokenização com o Gensim

```
1 text = "Um exemplo de texto, com vírgulas e palavras como guarda-chuva."
2 from gensim.utils import tokenize
3 tokens = list(tokenize(text))
4 print(tokens)
```

Saída do Programa 5.2

```
1 ['Um', 'exemplo', 'de', 'texto', 'com', 'vírgulas',
2 'e', 'palavras', 'como', 'guarda', 'chuva']
```

Existem outras duas formas de tokenizar um texto, por caracteres ou símbolos, ou por subpalavras (Freitas, 2022). Ambas são pouco usadas na prática e tem interesse limitado no uso computacional.

5.3 Normalização de maiúsculas e minúsculas

A **normalização de maiúsculas e minúsculas**, em inglês *case folding*, é o processo de colocar todas as letras na forma minúscula ou maiúscula.

Essa ação é útil para processos de indexação, levando em conta que as palavras que começam frases não devem ser indexadas de outras formas, e também que as consultas podem não ser feitas com a regra culta do uso de maiúsculas ou minúsculas na língua. Porém, em outros processos pode ser prejudicial. Por exemplo, na extração de entidades nomeadas uma dica para indicar um nome próprio é ele iniciar em letra maiúscula.

Normalmente a operação é feita transformando em minúsculas. Em Python, a função `str.lower()` faz isso rapidamente para a forma minúscula.

Caracteres maiúsculos e minúsculos

Os primeiros alfabetos de Latim, de onde derivam as línguas românicas, só possuíam um tipo de letra. Com o tempo apareceram novas formas de escrita, havendo a escrita uncial, mais semelhante às nossas maiúsculas, e a meia-uncial, com alguns aspectos das nossas minúsculas, porém os textos usavam uma ou outra (Muzdakis, 2020).

O aparecimento das minúsculas se dá na forma de uma forma de escrita feita para ser mais rápida, sendo mais eficiente para a cópia que manual, e feita com penas em papiro, ao invés das maiúsculas que eram criadas com ferramentas em monumentos ou pedra. Elas são, na verdade, destinadas à escrita cursiva (Diringer, 1953).

Diringer (1953) afirma que a principal influência para usar minúsculas e maiúsculas juntas acontece logo antes ou na época de Carlos Magno (724-814), com o aparecimento da caligrafia Carolíngia, ou Carolina, que por sua clareza foi transformada na caligrafia padrão do Império Carolíngio (800-888) e passou a ser usada em livros da Europa Ocidental. Ela incluía espaços claros entre as palavras e letras maiúsculas.

Os nomes em inglês, *uppercase* e *lowercase*, é proveniente do fato das máquinas de tipografia guardarem as maíscula em uma caixa, ou estojo, mais ao alto, a frente do tipógrafo, e as minúsculas em uma caixa inferior. Em português também se usa **caixa alta** e **caixa baixa**.

5.4 Stemmers

Stemmers são algoritmos, normalmente heurísticas, que fornecem um possível raiz ou radical de uma palavra, evitando que sistemas de processamento de texto tenham que tratar de todas as suas variações e ajudando na interpretação de um possível significado do texto e, na prática, reduzindo o número de chaves de um sistema de busca ou o número de atributos de um sistema de classificação de texto.

Lovins (1968) publicou o primeiro artigo sobre *stemmers*. Ela define um algoritmo de stemming como “um procedimento computacional que reduz todas as palavras para uma mesma raiz (ou, se prefixos não são tratados, para um mesmo radical) para uma forma comum, usualmente por remover de cada palavra seus sufixos derivacionais e inflecionais” (Lovins, 1968). Seu artigo cita três abordagens anteriores: um por John W. Tukey, o segundo por Michael Lesk orientado por Gerard Salton, e o terceiro por James I. Dolby.

A definição de Lovins deixa claro que um algoritmo de stemming é um algoritmo de fusão (*conflation*). A Figura 5.1 representa uma unificação das classificações de Frakes e Baeza-Yates (1992), (Galvez, MoyaAnegón e Solana, 2005) e Amri e Zenkouar (2018) para esse tipo de algoritmo. Nesse diagrama, o *stemmer* mais conhecido hoje em dia, o de Porter (1980), segue o caminho marcado em amarelo. Muitas heurísticas de remoção de afixos, porém, removem apenas os sufixos. Neste texto tratamos apenas dos algoritmos mais tradicionais, e mais usados, por estarem amplamente disponíveis.

Lovins (1968) cita dois princípios que norteiam o desenvolvimento de **stemmers**. O primeiro é a iteração, que busca remover os sufixos por meio de iterações planejadas, já que eles normalmente são adicionados às palavras em uma ordem específica. O segundo é o do casamento mais longo, que escolhe o maior final possível quando é dado uma opção por mais de um final. Além disso, discute que os *stemmers* podem ser livres ou sensitivos ao contexto, onde a existência de um contexto ajuda decidir se uma regra pode ou não ser usada. Por exemplo, exceções podem ser criadas para evitar

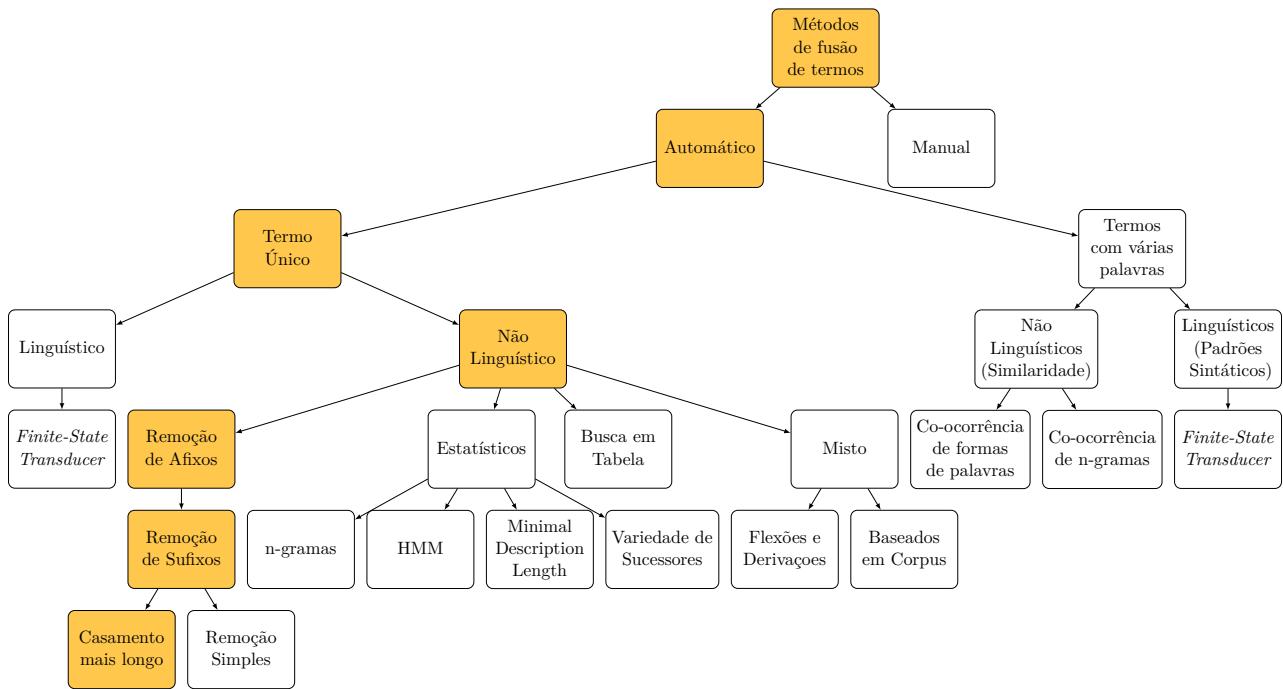


Figura 5.1: Classificação algoritmos de fusão, a partir de Frakes e Baeza-Yates (1992), Galvez, MoyaAnegón e Solana (2005) e Amri e Zenkouar (2018). O caminho em amarelo indica os algoritmos de Porter e o RSLP.

que um contexto comum, como “abilidade”, que pode ser aplicado a palavras como “sociabilidade” e “computabilidade”, não seja aplicado a palavra “habilidade” (Lovins, 1968). Sua proposta é um *stemmer* sensitivo ao contexto, não iterativo e do tipo casamento mais longo.

A seguir são descritos alguns algoritmos, os de truncagem implementados no NLTK, porém é necessário avisar que essas descrições são baseadas nos textos originais, já antigos, e pecam por falta de uma linguagem mais adequada.

5.4.1 Stemmers por Busca em Tabela

Os stemmers mais simples que podemos imaginar são os de busca em tabela. Nesse caso só é necessário ter uma lista de pares palavra-radical. Sua implementação por meio de uma tabela de espalhamento ou de uma estrutura de busca em árvore deixa esse algoritmo muito rápido, apesar de depender de memória.

Essa abordagem, se existisse uma tabela feita por linguistas, seria bastante eficiente e correta, porém não existe uma tabela padrão desse tipo. Além disso, podem existir palavras fora da lista, porque foram recém criadas, ou porque a lista é limitada em algum aspecto.

Abordagens semelhantes, porém, são feitas, por exemplo, em NLTK existe um lematizador feito a partir do WordNet², que retorna a mesma palavra se ela não é encontrada. Também é possível ser mais agressivo e substituir cada palavra por seu significado (*synset* no WordNet).

²https://www.nltk.org/_modules/nltk/stem/wordnet.html

5.4.2 Porter Stemmer

Porter (1980) propôs um dos stemmers que mais influenciou a área, criado de forma empírica. Ele possui uma heurística com 5 passos:

1. remoção de plurais;
2. unificação de padrões de sufixos;
3. manipulação de transformações necessárias para palavras especiais
4. remoção adicional de sufixo
5. remoção de vogal final

O algoritmo é fornecido no NLTK e uma implementação em Python, feita por Vivake Gupta é oferecida no site mantido pelo próprio Porter³, que documenta algumas melhorias no artigo original.

Esse *stemmer* é considerado congelado e melhorias são feitas no *snowball*

A segunda versão, não congelada, do algoritmo de Porter é conhecida como Porter 2. Elas apresenta, em geral, os mesmos passos, com detalhes adicionais em cada passo. Ambas estão implementadas no NLTK^{4,5}.

5.4.3 Snowball Stemmer

Snowball é uma “pequena linguagem de processamento de strings projetada para criar algoritmos de stemming para uso na Recuperação da Informação”(Betts e Bouton, 2022) criada por Porter (2001). A partir dessa linguagem é mantido um algoritmo baseado no original de Porter, conhecido como *porter2*, que é traduzido para várias linguagens⁶, além de stemmers para várias outras linguagens, incluindo o português. O site também fornece programas em Snowball para vários outros stemmers⁷. O algoritmo está presente na NLTK (Bird, Loper e Klein, 2009). O nome é uma homenagem a linguagem de processamento de texto SNOBOL4.

5.4.4 Lancaster Stemmer

O algoritmo de Lancaster (Paice, 1990), proposto por Paice é baseado em regras, compostas de 5 partes (Paice, 1990):

1. um final de palavra, com um ou mais caracteres;
2. um flag indicando que a forma deve estar intacta;
3. um dígito especificando quantas letras devem ser removidas do final da palavra, sendo zero um valor válido;
4. uma string opcional a ser somada no final da palavra, e
5. um símbolo de continuação.

As regras então são organizadas em seções, de acordo com o final das palavras, e o seguinte algoritmo é usado para tratar cada palavra:

- Selecionar a seção relevante
 - Inspecionar a última letra da forma

³<https://tartarus.org/martin/PorterStemmer/>

⁴<https://www.nltk.org/api/nltk.stem.porter.html>

⁵<https://www.nltk.org/api/nltk.stem.snowball.html>

⁶<https://github.com/snowballstem/snowball>

⁷<https://snowballstem.org/algorithms/>

- Se não há seção correspondente a essa letra, terminar
- Se há, considerar a primeira regra da seção relevante
- Verificar aplicabilidade da regra
 - Se as letras finais da forma não correspondem a regra, vá para 4
 - Se as letras casam, e o *flag* de intacto está ativo, e a forma não está intacta, vá para 4
 - Se as condições de aceitação não são válidas, vá para 4 (ver passo seguinte)
- Aplicar regra
 - Apagar no final da forma o número de caracteres especificado na regra
 - Se há uma string a adicionar, adicioná-la
 - Se o símbolo de continuação é “.”, terminar
 - Se o símbolo de continuação é “>”, vá para 1 (isto é, busque mais uma regra)
- Busque por outra regra
 - Vá para a próxima regra
 - Se a seção relevante terminou, terminar
 - Se não, vá para 2

A implementação desse algoritmo no NLTK⁸ por Steven Tomcavage é bastante clara,

5.4.5 *Stemmers Criados para o Português*

Vários *stemmers* foram criados especificamente para o português, entre eles o STEMBR (Alvares, Garcia e Ferraz, 2005), o *Snowball Stemmer* para português (Snowball Project, 2022) e o RSLP (Lopes, 2004), para os dois últimos há implementações no NLTK.

O algoritmo em português do Snowball define 3 regiões, R1, R2 e RV, relativas as posições de vogais e consoantes da palavra, porém não cabe defini-las aqui. Maiores detalhes podem ser encontrados na página oficial do algoritmo (Snowball Project, 2022).

A partir daí, ele segue 5 passos (Snowball Project, 2022):

1. Remoção de sufixos padronizados;
2. Se não foi removido um final no passo 1, remover sufixos de verbos;
3. Remover sufixos em RV;
4. Remover sufixos residuais, e
5. Tratar final restante da palavra (casos específicos).

O **RSLP** (Lopes, 2004) também é um algoritmo de truncagem e remoção de sufixos, muito interessante porque definido em regras simples de serem entendidas, semelhantes as de Paice (1990), e usa uma heurística de 8 passos:

1. Eliminar o “s” final das palavras
2. Reduzir para o masculino, para palavras que terminam em “a”
3. Reduzir para o singular
4. Reduzir o advérbio
5. Reduzir o substantivo
6. Reduzir a conjugação, se o sufixo ainda não foi removido
7. Remoção da vogal, se o sufixo ainda não foi removido
8. Remoção dos acentos

Cada passo é formado por um conjunto de regras examinadas em sequência, onde apenas a primeira regra válida é aplicada. As regras são caracterizadas por quatro componentes:

⁸<https://www.nltk.org/api/nltk.stem.lancaster.html>

1. sufixo a ser removido
2. tamanho mínimo do radical restante
3. sufixo de substituição, se necessário
4. lista de exceções a regra

Esse stemmer possui 199 regras, sendo que uma regra típica é a tupla `{ona, 3, ão, {carona}}`.

O RSLP foi implementado no `org.apache.lucene.analysis.pt`⁹. Um outra implementação foi feita em Ruby por Andrew S. Aguiar¹⁰. Uma implementação em Java foi feita por Anael Carvalho para o Elastic Search¹¹. As regras foram reimplementadas em Python para o NLTK por Tiago Tresoldi para o NLTK^{12,13} e podem ser obtidas na Kaggle¹⁴.

5.4.6 Usando os *Stemmers* em Python

O Programa 5.3 apresenta uma implementação do uso dos 3 *stemmers* aqui descrito com o NLTK.

Programa 5.3: Uso dos stemmers para inglês no NLTK

```

1 from nltk.stem.porter import PorterStemmer
2 from nltk.stem.snowball import SnowballStemmer
3 from nltk.stem.lancaster import LancasterStemmer
4 from nltk.tokenize import word_tokenize
5 import string
6
7 porter_st = PorterStemmer()
8 porter_nltk_st = PorterStemmer(mode=PorterStemmer.NLTK_EXTENSIONS)
9 snow_st = SnowballStemmer("english")
10 lanc_st = LancasterStemmer()
11
12 # passage from Moby Dick from Herman Melville
13 text = """Shipmates, this book, containing only four chapters four yarns is
14 one of the smallest strands in the mighty cable of the Scriptures.
15 """
16 # remove punctuation
17 text = text.translate(str.maketrans(' ', ' ', string.punctuation))
18 tokens_en = word_tokenize(text)
19
20 p_tokens = list(map(porter_st.stem, tokens_en))
21 pn_tokens = list(map(porter_nltk_st.stem, tokens_en))
22 ln_tokens = list(map(lanc_st.stem, tokens_en))
23 sn_tokens = list(map(snow_st.stem, tokens_en))
24
25 print("-"*65)
26 print("{0: <10} | {1: <10} | {2: <11} | {3: <10} | {4: <10}".format(

```

⁹https://lucene.apache.org/core/7_6_0//analyzers-common/org/apache/lucene/analysis/pt/RSLPStemmerBase.html

¹⁰<https://gist.github.com/andrewaguiar/07e04b694d578d6c855dce02853c4410>

¹¹<https://github.com/anaelcarvalho/elasticsearch-analysis-rslp>

¹²<https://www.nltk.org/api/nltk.stem.rslp.html>

¹³https://www.nltk.org/_modules/nltk/stem/rslp.html

¹⁴<https://www.kaggle.com/datasets/nltkdata/rslp-stemmer>

```

27     "original",
28     "Porter",
29     "Porter NLTK",
30     "Lancaster",
31     "Snowball"))
32 print("-"*65)
33 for i in range(len(p_tokens)):
34     print("{0: <10} | {1: <10} | {2: <11} | {3: <10} | {4: <10}".format(
35         ↪ tokens_en[i], p_tokens[i], pn_tokens[i], ln_tokens[i], sn_tokens[i]))
36 print("-"*65)

```

Saída para o Programa 5.3

1	-----
2	original Porter Porter NLTK Lancaster Snowball
3	-----
4	Shipmates shipmat shipmat shipm shipmat
5	this thi thi thi this
6	book book book book book
7	containing contain contain contain contain
8	only onli onli on onli
9	four four four four four
10	chapters chapter chapter chapt chapter
11	four four four four four
12	yarns yarn yarn yarn yarn
13	is is is is is
14	one one one on one
15	of of of of of
16	the the the the the
17	smallest smallest smallest smallest smallest
18	strands strand strand strands strand
19	in in in in in
20	the the the the the
21	mighty mighti mighti mighty mighti
22	cable cabl cabl cabl cabl
23	of of of of of
24	the the the the the
25	Scriptures scriptur scriptur scriptures scriptur
26	-----

O Programa 5.4 apresenta o uso dos *stemmers* Snowball e RSLP para português com o mesmo software.

Programa 5.4: Uso dos stemmers para português no NLTK

```

1 from nltk.stem import RSLPStemmer
2 from nltk.stem.snowball import SnowballStemmer
3 #Trecho de Dom Casmurro por Machado de Assis
4 texto = """Uma noite destas, vindo da cidade para o Engenho Novo, encontrei
5 num trem da Central um rapaz

```

```

6 aqui do bairro, que eu conheço de vista e de chapéu. Cumprimentou-me,
7 sentou-se ao pé de mim, falou da lua e dos ministros,
8 e acabou recitando-me versos."
9 texto = texto.translate(str.maketrans('—,.?!', '      '))
10 snow_p_st = SnowballStemmer("portuguese")
11 rslp_st = RSLPStemmer()
12 tokens_ma = word_tokenize(texto)
13 pt_tokens = list(map(snow_p_st.stem, tokens_ma))
14 pt_n_tokens = list(map(rslp_st.stem, tokens_ma))
15 print("-"*35)
16 print("{0: <12} | {1: <10} | {2: <10}".format("Original","Snowball","RSLP"))
17 print("-"*35)
18 for i in range(len(pt_tokens)):
19     print("{0: <12} | {1: <10} | {2: <10}".format(tokens_ma[i],pt_tokens[i],
20         pt_n_tokens[i]))
20 print("-"*35)

```

Saída para o Programa 5.4

1	2 Original	3 -----	4 Snowball	5 RSLP
4	Uma	uma	uma	
5	noite	noit	noit	
6	destas	dest	dest	
7	vindo	vind	vind	
8	da	da	da	
9	cidade	ciudad	ciudad	
10	para	par	par	
11	o	o	o	
12	Engenho	engenh	engenh	
13	Novo	nov	nov	
14	encontrei	encontr	encontr	
15	num	num	num	
16	trem	trem	tr	
17	da	da	da	
18	Central	central	centr	
19	um	um	um	
20	rapaz	rapaz	rapaz	
21	aqui	aqu	aqu	
22	do	do	do	
23	bairro	bairr	bairr	
24	que	que	que	
25	eu	eu	eu	
26	conheço	conhec	conheç	
27	de	de	de	
28	vista	vist	vist	
29	e	e	e	
30	de	de	de	

```

31 chapéu      | chapéu      | chapéu
32 Cumprimentou | cumpriment | cumpriment
33 me          | me          | me
34 sentou      | sent        | sent
35 se          | se          | se
36 ao          | ao          | ao
37 pé          | pé          | pé
38 de          | de          | de
39 mim         | mim         | mim
40 falou       | fal         | fal
41 da          | da          | da
42 lua         | lua         | lua
43 e           | e           | e
44 dos         | dos         | do
45 ministros   | ministr    | ministr
46 e           | e           | e
47 acabou      | acab        | acab
48 recitando   | recit       | recit
49 me          | me          | me
50 versos      | vers        | vers
51 -----

```

Um **stemmer** pode se favorecer de uma estratégia de cache se processando muitas palavras.

Programa 5.5: Usando stemmer com cache em Python

```

1 from nltk.corpus import gutenberg
2 from nltk.tokenize import word_tokenize
3 from nltk.stem.porter import PorterStemmer
4 from functools import lru_cache
5 from time import perf_counter
6 import string
7
8 # Junta todos os textos do Gutenberg
9 text = ''
10 for f in gutenberg.fileids():
11     print(f)
12     text += gutenberg.raw(f)
13 text = text.translate(str.maketrans(' ', ' ', string.punctuation))
14
15 # Stem todos sem cache
16 print("Iniciando stemming Porter sem cache...")
17 tic = perf_counter()
18 tokens_en = word_tokenize(text)
19 porter_st = PorterStemmer()
20 p_tokens = list(map(porter_st.stem, tokens_en))
21 toc = perf_counter()
22
23 print(f"Tempo sem cache {toc - tic:.4f} seconds")
24

```

```

25 #Stem all with cache
26
27 @lru_cache(maxsize=1000)
28 def cached_stem(word):
29     return porter_st.stem(word)
30
31 print("Iniciando stemming Porter com cache...")
32 tic = perf_counter()
33 pc_tokens = list(map(cached_stem,tokens_en))
34 toc = perf_counter()
35 print(f"Tempo com cache {toc - tic:.4f} seconds")
36
37 print(cached_stem.cache_info())

```

Saída para o Programa 5.5

```

1 austen-emma.txt
2 austen-persuasion.txt
3 austen-sense.txt
4 bible-kjv.txt
5 blake-poems.txt
6 bryant-stories.txt
7 burgess-busterbrown.txt
8 carroll-alice.txt
9 chesterton-ball.txt
10 chesterton-brown.txt
11 chesterton-thursday.txt
12 edgeworth-parents.txt
13 melville-moby_dick.txt
14 milton-paradise.txt
15 shakespeare-caesar.txt
16 shakespeare-hamlet.txt
17 shakespeare-macbeth.txt
18 whitman-leaves.txt
19 Iniciando stemming Porter sem cache...
20 Tempo sem cache 59.8562 seconds
21 Iniciando stemming Porter com cache...
22 Tempo com cache 14.6462 seconds
23 CacheInfo(hits=1684211, misses=451115, maxsize=1000, currsize=1000)

```

Apesar do algoritmos de Porter ser rápido, processando mais de 2 milhões de palavras em aproximadamente 60s, com o uso de um cache LRU esse tempo pode ser levado a aproximadamente 2 ou 3 segundos, como mostra a Figura 5.2 , o que pode ser bom no processamento de grandes coleções. A análise do número de *misses* indica o número de formas distintas, 66392 palavras. No computador onde foi testado, um cache de 32768 itens já apresentou um desempenho de 2,8s, enquanto um cache de 65536, onde cabem quase todas, apresenta um desempenho de 2,6s. Um cache onde cabiam todas as palavras apresentou um desempenho equivalente.

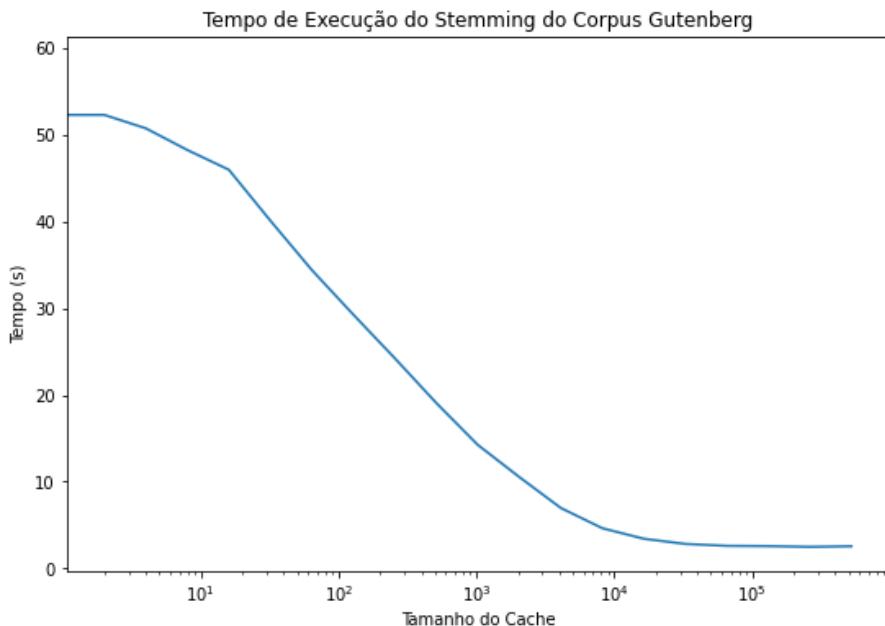


Figura 5.2: Tempo de execução do stemmer de Porter com diferentes tamanhos de cache, potências de 2 de 0 a 524288 entradas, mostrando um limite prático para essa coleção, onde há 66392 formas únicas. Um cache de 32768 foi suficiente para atingir um desempenho próximo do máximo.

5.5 Remoção de *stopwords*

Stopwords são palavras filtradas do texto antes de um processamento (Christopher D. Manning, Raghavan e Schütze, 2009a). Outra definição é a de serem palavras funcionais que não trazem informação semântica interessante sobre o assunto do texto. Além disso, são palavras que não precisam ser indexadas porque sua presença é tão comum nos documentos que elas não fazem diferença nos algoritmos de busca e classificação, já que não diferenciam um do outro (Nothman, Qin e Yurchak, 2018).

O Programa 5.6, calcula e gera a Figura 5.3, com o gráfico da frequência das 50 palavras mais usadas no corpus Gutenberg do NLTK. É possível notar duas propriedades: a distribuição se parece com a distribuição de Zipf e as palavras mais frequentes são as mesmas que encontramos nas listas de *stopwords*.

Programa 5.6: Contagem das 50 primeiras palavras do corpus Gutenberg no NLTK

```

1 from nltk.probability import FreqDist
2 from nltk.corpus import gutenberg
3 import string
4 import matplotlib.pyplot as plt
5
6 text = ''
7 for f in gutenberg.fileids():
8     print(f)
9     text += gutenberg.raw(f)
10 text = text.translate(str.maketrans(' ', ' ', string.punctuation))
11
12 fd = FreqDist(word.lower() for word in text.split())

```

```

13 print("Frequencies done!")
14 p = fd.plot(50, show=False, title="Distribuição das palavras no corpus Gutemberg")
15 p.set_xlabel("Amostra")
16 p.set_ylabel("Frequência")
17 plt.show()

```

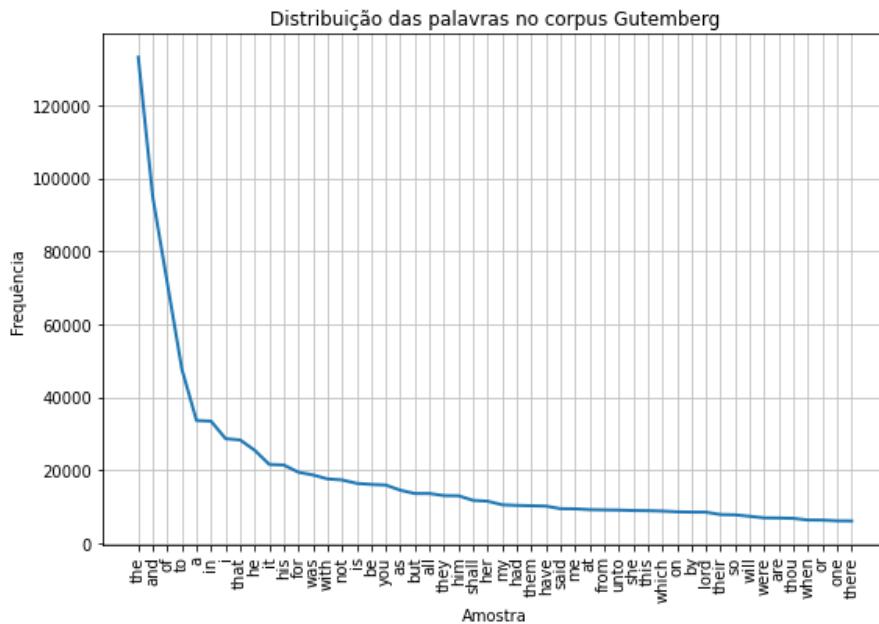


Figura 5.3: Contagem das 50 palavras mais frequentes do corpus Gutenberg, pelo Programa 5.6

Normalmente as *stopwords* são retiradas do índice a partir de um filtro feito com uma lista predefinida por seres humanos, mas elas podem também ser criadas a partir da coleção. Não há um acordo específico sobre que palavras devem ou não ser removidas, ou mesmo se devem ser removidas.

No passado, devido ao tamanho dos índices e escassez de espaço em disco, era muito importante remover essas palavras, porém atualmente muitos sistemas não o fazem, principalmente porque podem ser usadas em n-gramas. Muitos experimentos de classificação, hoje em dia, são feitos com e sem *stopwords* para analisar seu efeito no processo.

Vários pacotes de processamento de texto fornecem listas de *stopwords* em diferentes línguas. Além disso, elas podem ser encontradas em:

- <http://snowball.tartarus.org/>
- <https://countwordsfree.com/stopwords>
- <http://www.ranks.nl/stopwords/>
- <http://members.unine.ch/jacques.savoy/clef/>
- <https://cran.r-project.org/web/packages/stopwords/>
- <https://github.com/igorbrigadir/stopwords/tree/21fb2ef>
- http://ir.dcs.gla.ac.uk/resources/linguistic_utils/

O sistema *Smart* (Salton e McGill, 1983) usava a lista de *stopwords* da Tabela 5.1, sendo ela considerada a primeira lista disponível (Nothman, Qin e Yurchak, 2018).

Programa 5.7: Removendo as *stopwords* com NLTK

```
1 import nltk
```

Tabela 5.1: Lista de *stopwords* do sistema *Smart* (Salton e McGill, 1983).

a a's able about above according accordingly across actually after afterwards again against ain't all allow allows almost alone along already also although always am among amongst an and another any anybody anyhow anyone anything anyway anyways anywhere apart appear appreciate appropriate are aren't around as aside ask asking associated at available away awfully b be became because become becomes becoming been before beforehand behind being believe below beside besides best better between beyond both brief but by c c'mon c's came can can't cannot cant cause causes certain certainly changes clearly co com come comes concerning consequently consider considering contain containing contains corresponding could couldn't course currently d definitely described despite did didn't different do does doesn't doing don't done down downwards during e each edu eg eight either else elsewhere enough entirely especially et etc even ever every everybody everyone everything everywhere ex exactly example except f far few fifth first five followed following follows for former formerly forth four from further furthermore g get gets getting given gives go goes going gone got gotten greetings h had hadn't happens hardly has hasn't have haven't having he he's hello help hence her here here's hereafter hereby herein hereupon hers herself hi him himself his hither hopefully how howbeit however i i'd i'll i'm i've ie if ignored immediate in inasmuch inc indeed indicate indicated indicates inner insofar instead into inward is isn't it it'd it'll it's its itself j just k keep keeps kept know knows known l last lately later latter latterly least less lest let let's like liked likely little look looking looks ltd m mainly many may maybe me mean meanwhile merely might more moreover most mostly much must my myself n name namely nd near nearly necessary need needs neither never nevertheless new next nine no nobody non none noone nor normally not nothing novel now nowhere o obviously of off often oh ok okay old on once one ones only onto or other others otherwise ought our ours ourselves out outside over overall own p particular particularly per perhaps placed please plus possible presumably probably provides q que quite qv r rather rd re really reasonably regarding regardless regards relatively respectively right s said same saw say saying says second secondly see seeing seem seemed seeming seems seen self selves sensible sent serious seriously seven several shall she should shouldn't since six so some somebody somehow someone something sometime sometimes somewhat somewhere soon sorry specified specify specifying still sub such sup sure t t's take taken tell tends th than thank thanks thanx that that's thaths the their theirs them themselves then thence there there's thereafter thereby therefore therein theres thereupon these they they'd they'll they're they've think third this thorough thoroughly those though three through throughout thru thus to together too took toward towards tried tries truly try trying twice two u un under unfortunately unless unlikely until unto up upon us use used useful uses using usually uucp v value various very via viz vs w want wants was wasn't way we we'd we'll we're we've welcome well went were weren't what what's whatever when whence whenever where where's whereafter whereas whereby wherein whereupon wherever whether which while whither who who's whoever whole whom whose why will willing wish with within without won't wonder would would wouldn't x y yes yet you you'd you'll you're you've your yours yourself yourselves z zero

Tabela 5.2: Lista de *stopwords* em português brasileiro obtida em <http://www.ranks.nl/stopwords/>.

a ainda alem ambas ambos antes ao aonde aos apos aquele aqueles as assim com como contra contudo cuja cujas cujo cujos da das de dela dele deles depois desde desta deste dispoe dispoem diversa diversas diversos do dos durante e ela elas ele eles em entao entre essa essas esse esses esta estas este estes ha isso isto logo mais mas mediante menos mesmas mesmo mesmos na nas nao nas nem nesse neste nos o os ou outra outras outro outros pelas pelo pelos perante pois por porque portanto proprio propios quais qual qualquer quando quanto que quem quer se seja sem sendo seu seus sob sobre sua suas tal tambem teu teus toda todas todo todos tua tuas tudo um uma umas uns

```

2 from nltk.corpus import stopwords
3 from nltk.tokenize import word_tokenize
4
5 stop_en = stopwords.words('english')
6 stop_pt = stopwords.words('portuguese')
7
8
9 print("Stopwords in English:\n")
10 print(stop_en)
11
12 test_frase = "There is no such thing as a free lunch"
13
14 print("\n\nFiltrando: ",test_frase)
15
16 word_tokens = word_tokenize(test_frase)
17 filtered_tokens = [w for w in word_tokens if not w.lower() in stop_en]
18
19 print(filtered_tokens)
20
21 print("\n\nStopwords em português:\n")
22 print(stop_pt)
23
24 frase_teste ="Não existe almoço grátis"
25
26 print("\n\nFiltrando: ",frase_teste)
27
28 tokens_palavra = word_tokenize(frase_teste)
29 tokens_filtrados = [p for p in tokens_palavra if not p.lower() in stop_pt]
30
31 print(tokens_filtrados)

```

Saída do Programa 5.7

1 Stopwords in English:

2

3 ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves',

4 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours',

```

5 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she',
6 "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself',
7 'they', 'them', 'their', 'theirs', 'themselves', 'what',
8 'which', 'who', 'whom', 'this', 'that', "that'll", 'these',
9 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been',
10 'being', 'have', 'has', 'had', 'having', 'do', 'does',
11 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or',
12 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for',
13 'with', 'about', 'against', 'between', 'into', 'through',
14 'during', 'before', 'after', 'above', 'below', 'to', 'from',
15 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under',
16 'again', 'further', 'then', 'once', 'here', 'there', 'when',
17 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few',
18 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not',
19 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't',
20 'can', 'will', 'just', 'don', "don't", 'should', "should've",
21 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren',
22 "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn',
23 "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven',
24 "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't",
25 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't",
26 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't",
27 'won', "won't", 'wouldn', "wouldn't"]
28
29

```

30 Filtrando: There is no such thing as a free lunch

31 ['thing', 'free', 'lunch']

32

33

34

35

36 Stopwords em português:

```

37
38 ['de', 'a', 'o', 'que', 'e', 'é', 'do', 'da', 'em',
39 'um', 'para', 'com', 'não', 'uma', 'os', 'no', 'se',
40 'na', 'por', 'mais', 'as',
41 'dos', 'como', 'mas', 'ao', 'ele', 'das', 'à', 'seu',
42 sua', 'ou', 'quando', 'muito', 'nos', 'já', 'eu', 'também',
43 'só', 'pelo', 'pela', 'até', 'isso', 'ela', 'entre',
44 'depois', 'sem', 'mesmo', 'aos', 'seus', 'quem', 'nas',
45 'me', 'esse', 'eles', 'você', 'essa', 'num', 'nem',
46 'suas', 'meu', 'ás', 'minha', 'numa', 'pelos', 'elas',
47 'qual', 'nós', 'lhe', 'deles', 'essas', 'esses', 'pelas',
48 'este', 'dele', 'tu', 'te', 'vocês', 'vos', 'lhes', 'meus',
49 'minhas', 'teu', 'tua', 'teus', 'tuas', 'noso', 'nossa',
50 'nossos', 'nossas', 'dela', 'delas', 'esta', 'estes',
51 'estas', 'aquele', 'aquela', 'aqueles', 'aqueelas', 'isto',
52 'aquilo', 'estou', 'está', 'estamos', 'estão', 'estive',

```

```

53 'esteve', 'estivemos', 'estiveram', 'estava', 'estávamos',
54 'estavam', 'estivera', 'estivéramos', 'esteja', 'estejamos',
55 'estejam', 'estivesse', 'estivéssemos', 'estivessem', 'estiver',
56 'estivermos', 'estiverem', 'hei', 'há', 'havemos', 'hão',
57 'houve', 'houvemos', 'houveram', 'houvera', 'houvéramos',
58 'haja', 'hajamos', 'hajam', 'houvesse', 'houvéssemos',
59 'houvessem', 'houver', 'houvermos', 'houverem', 'houverei',
60 'houverá', 'houveremos', 'houverão', 'houveria', 'houveríamos',
61 'houveriam', 'sou', 'somos', 'são', 'era', 'éramos', 'eram',
62 'fui', 'foi', 'fomos', 'foram', 'fora', 'fôramos', 'seja',
63 'sejamos', 'sejam', 'fosse', 'fôssemos', 'fossem', 'for',
64 'formos', 'forem', 'serei', 'será', 'seremos', 'serão',
65 'seria', 'seríamos', 'seriam', 'tenho', 'tem', 'temos',
66 'tém', 'tinha', 'tínhamos', 'tinham', 'tive', 'teve',
67 'tivemos', 'tiveram', 'tivera', 'tivéramos', 'tenha',
68 'tenhamos', 'tenham', 'tivesse', 'tivéssemos', 'tivessem',
69 'tiver', 'tivermos', 'tiverem', 'terei', 'terá', 'teremos',
70 'terão', 'teria', 'teríamos', 'teriam']
71
72
73 Filtrando: Não existe almoço grátis
74 ['existe', 'almoço', 'grátis']

```

No NLTK é possível estender a stopwords padrão usando o método `extend`, cujo parâmetro é uma lista de *strings*.

5.5.1 Usando o KNIME

O KNIME possui um nó dedicado para stopwords, o “Stop Word Filter”. Ele recebe uma tabela de documentos e pode aplicar tanto uma lista padrão, usando uma entre 12 línguas, incluindo o português, ou uma lista customizada, proveniente de uma tabela.

Duas são as abas de configuração importantes no nó “Stop Word Filter”: a *Preprocessing* (Figura 5.4) e a *Filter Options* (Figura 5.5).

5.5.2 Problemas com *stopwords*

A primeira pergunta que vem com o uso de *stopwords* é quando removê-las dentro do fluxo de pré-processamento. Essa pergunta pode ficar mais complicada caso seja desejada uma análise léxica ou sintática da frase.

Também é possível tratar *stopwords* automáticas, tanto as eliminando como as substituindo. Siglas, por exemplo, podem ser, ou não, desconsideradas. Siglas especiais, como “007” ou “3D” podem ser substituídas por palavras com significado semelhante, como “James Bond” ou “tridimensional”. Outras siglas, fora de um vocabulário de controle, podem ser filtradas.

Outra questão interessante é como fazer com termos compostos que desejamos indexar. Por exemplo, pode ser interessante indexar o trígrama “banco de dados”. Esses termos podem estar em uma lista positiva, conhecida como *go-words*.

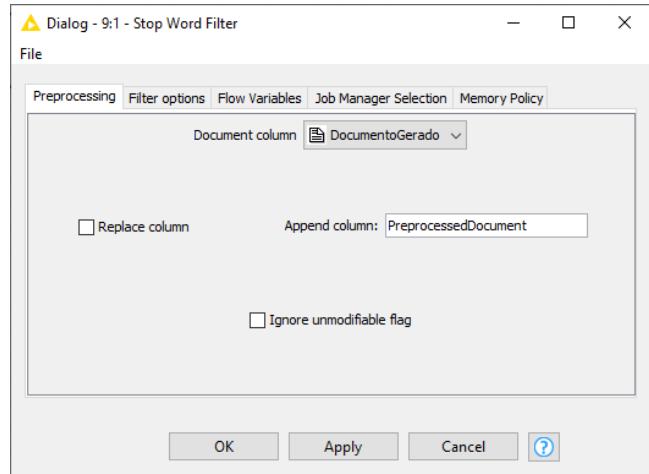


Figura 5.4: Aba Preprocessing do nó Stop Word Filter no KNIME

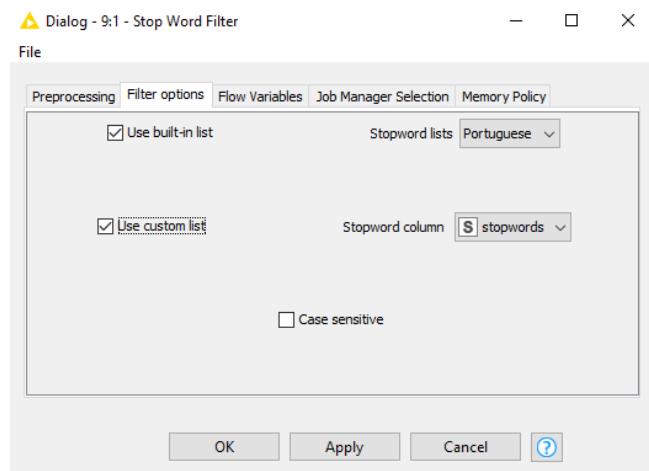


Figura 5.5: Aba Filter Options do nó Stop Word Filter no KNIME

Programa 5.8: Programa que usa *go-words* compostas

```

1 import nltk
2 from nltk.corpus import stopwords
3 from nltk.tokenize import word_tokenize
4
5
6 def check_a_go(frase, indice, comb):
7     seq = [ w.lower() for w in frase[indice+1:indice+len(comb)+1] ]
8     if seq == comb:
9         return True
10    else:
11        return False
12
13 def check_go(frase, word, indice, go):
14     if word in go:
15         for comb in go[word]:

```

```

16     if check_a_go(frase, indice, comb):
17         return [word]+comb
18     else:
19         return False
20
21
22
23
24 stop_pt = stopwords.words('portuguese')
25 frase_teste ="A criação dos bancos de dados foi muito influente no \
26     desenvolvimento de software, permitindo o uso de diferentes \
27     linguagens de programação. São Paulo é uma cidade grande."
28
29 go_words = {"bancos" : [[["de", "dados"]], ["bancos", "financeiros"]], \
30             "linguagens" : [[["de", "programação"]]], \
31             "são" : [[["paulo"]]]}
32
33
34 word_tokens = word_tokenize(frase_teste)
35
36 filtered = []
37
38 i = 0
39 while i <len(word_tokens):
40     cur_word = word_tokens[i].lower()
41     if comb := check_go(word_tokens, cur_word, i, go_words):
42         filtered.append(comb)
43         i += + len(comb)
44     elif cur_word not in stop_pt:
45         filtered.append(cur_word)
46     i += 1
47 print(filtered)

```

5.6 Normalização de texto

Normalização é o ato de corrigir um texto com erros ortográficos, isto é, palavras escritas erradas. Esse cuidado pode ser muito importante, já que erros em palavras funcionam como ruídos que degradam tanto a busca como algoritmos baseados em representações do tipo *bag-of-words* (Hacohen-Kerner, D. Miller e Yigal, 2020) como em redes de neurais profundas e modelos de linguagem, como o BERT (A. Kumar, Makhija e Gupta, 2020).

Os erros podem vir de:

- o autor não sabe escrever a palavra, ou tem um dúvida e opta pelo modo errado, ou
- o autor sabe escrever a palavra, mas erra ao fazê-lo, e não percebe o erro, por exemplo, digitando uma tecla muito próxima da tecla correta, como trocar um “o” por um “p”.
- o autor escolher usar um vocabulário informal, uma gíria, uma abreviação, a oralidade.

Uma palavra pode estar errada absolutamente, sendo chamada de palavra não real, como a palavra “atrazo”, ou apenas em um contexto, como a palavra “contínua”, que é usada no lugar da palavra “continua”, na frase “João continua a fazer o mesmo erro”.

Parte dos erros provindo do desconhecimento da forma correta de escrever a palavra vêm do fato que a relação entre fonemas e grafemas não é biunívoca, isto é, não existe exatamente um grafema para cada fonema e um fonema para cada grafema. Isso pode ser visto em francês, onde quase todas as conjugações de um verbo regular são faladas do mesmo jeito, mas são escritas diferentes, ou dos vários sons que uma mesma letra pode assumir, até mesmo em um mesma palavra, em inglês. As relações entre grafemas e fonemas podem ser:

- Poligâmicas, quando letras representam diferentes sons, dependendo da posição em que estão na palavra. Em português, o “s” tem o som da palavra “sapo”, semelhante ao c-cedilha, quando aparece no início da palavra, mas tem som de “z” quando aparece entre duas vogais, como em “asa”, e
- Poliândrica, quando duas letras podem gerar o mesmo som, como o som de “k” gerados pelas letras “c”, quando sucedida das vogais “a”, “o” ou “u”, e pelo ditongo “qu”, quando sucedido pelas vogais “e” ou “i”. Assim escrevemos “copo” e “quinto”, e falamos “kopo” e “kinto”.

Os erros podem ser categorizados segundo a forma como são consertados, de acordo com a necessidade de inserção, deleção, substituição ou transposição de letras. A noção de distância entre palavras, vista na seção 7.4 pode ser útil nesse caso.

5.6.1 Estratégias gerais de correção

Existem duas estratégias gerais de correção:

1. com aprendizado de máquina, e
2. sem aprendizado de máquina.

Norvig (2007) propõe um algoritmo simples que explica, de forma geral, como funciona um corretor de textos. A ideia básica é:

1. Ter um corpus com muitos exemplos de palavras corretamente escritas e gerar um dicionário;
2. Pegar uma palavra para corrigir, sem ideia do contexto;
3. Se a palavra está no dicionário, ela é aceita;
4. Caso contrário, pesquisa no dicionário todas as palavras candidatas a substitui-la que possuam uma distância de edição igual a 1;
5. Se existirem essas palavras, aceita a com maior frequência;
6. Caso contrário, faz o mesmo com a distância 2;
7. Caso ainda não tenha encontrado, aceita a palavra.

Como gerar todas as palavras com distância 1:

1. Apaga todos os caracteres 1 a 1;
2. Transpõe todos os caracteres, posição por posição;
3. Troca todos os caracteres por todos os outros possíveis;
4. Insere um caractere extra em todos os intervalos.

Esse passo a passo geraria 234 palavras para tentar corrigir a palavra “word”, caso ela não estivesse no dicionário. Se forem geradas todas as palavras com distância 1 a partir dessas palavras, temos todas as palavras com distância 2, que serão 60580 no caso da palavra “word”.

```

1 def P(word, N=sum(WORDS.values())):
2     "Calcula a probabilidade da palavra aparecer"
3     return WORDS[word] / N
4
5 def correction(word):
6     "Busca a correção mais provável da palavra"
7     return max(candidates(word), key=P)
8
9 def candidates(word):
10    "Gera todas as correções possíveis de distância de edição 1 e 2"
11    return (known([word]) or known(edits1(word)) or
12           known(edits2(word)) or [word])
13
14 def known(words):
15    "O conjunto de palavras existentes no texto"
16    return set(w for w in words if w in WORDS)
17
18 def edits1(word):
19    "Calcula todas as palavras a 1 edição de distância"
20    letters = 'abcdefghijklmnopqrstuvwxyz'
21    splits = [(word[:i], word[i:]) for i in range(len(word) + 1)]
22    deletes = [L + R[1:] for L, R in splits if R]
23    transposes = [L + R[1] + R[0] + R[2:] for L, R in splits if len(R)>1]
24    replaces = [L + c + R[1:] for L, R in splits if R for c in
25                 letters]
26    inserts = [L + c + R for L, R in splits for c in letters]
27    return set(deletes + transposes + replaces + inserts)
28
29 def edits2(word):
30    "Calcula todas as palavras a duas edições de distância"
31    return {e2 for e1 in edits1(word) for e2 in edits1(e1)}

```

5.6.2 Corrigindo textos com algoritmos genéticos

Algoritmos genéticos são técnicas de otimização inspiradas na evolução biológica. Eles são usados para resolver problemas complexos que envolvem a busca por soluções ótimas em um espaço de busca muito grande.

Um algoritmo genético começa com uma população de soluções candidatas representadas por uma codificação, ou genótipo. São comuns as codificações binárias. Cada indivíduo da população é então avaliado, e o algoritmo gera novas soluções a partir da seleção, reprodução e mutação desses indivíduos. O processo de seleção, probabilístico e baseado na qualidade de cada solução, permite que as soluções mais aptas sejam mantidas na população, enquanto a reprodução e mutação permitem que novas soluções sejam criadas. Ao longo do tempo, o algoritmo evolui a população de soluções candidatas para se aproximar da solução ótima (Goldberg, 1989).

Programação genética é uma técnica de computação evolucionária onde são construídos de forma automática programas capazes de resolver problemas específicos. Uma das formas possíveis é partir de

um conjunto de componentes para a construção de uma árvore sintática desses programas, além do conjunto de dados que represente o problema. O algoritmo 5.1 mostra de forma geral como acontece a programação genética (Poli, Langdon e McPhee, 2008).

Algoritmo 5.1: Algoritmo geral de programação genética. Fonte (Poli, Langdon e McPhee, 2008)

- 1 Criar um população inicial de programas a partir de primitivas disponíveis ;
 - 2 **repita**
 - 3 Execute cada programa e calcule sua adequação;
 - 4 Selecione um ou dos programas da população com probabilidade baseada na adequação para participar de operações genéticas;
 - 5 Crie um novo programa (ou mais) aplicando operações genéticas com probabilidades específicas;
 - 6 **até uma solução aceitável ou outra condição de parada ser alcançada;**
 - 7 **retorna o melhor programa**
-

Ferman (2016) propõe uma técnica que usa árvores representando programas, onde um texto considerada com erros é considerada um token que percorre todas os nós da árvores a partir da raiz até chegar as folhas, retornando então ao ponto de partida. No final são obtidas todas as formas de normalização encontradas para os vocábulos do texto.

Cada nó realiza uma entre várias operações possíveis. Existem cinco operações básicas para os nós: a tokenização, a modificação e codificação, a ramificação, o mapeamento em possibilidades e a filtragem ou desempate. Cada nó é formado de um módulo e de uma função. Nós são construído a partir de módulos que contêm funções. Os módulos são estruturas fixas pré-estabelecidas devido a arquitetura do sistema, enquanto as funções podem ser desenvolvidas pelos usuários, apesar de já haver uma quantidade delas disponível no sistema (Ferman, 2016).

Existem diferentes tipos de módulos em função de sua operação e de sua função arquitetural, por exemplo, um nó pode ter canais de comunicação com bifurcação, com prioridades ou com múltiplos filhos. Cada módulo pode exigir uma função que deve possuir uma assinatura específica. Os módulos existentes são: gerador, codificador com prioridade alta ou baixa, separador, seletor, identificador (terminal), duplicador com e sem prioridade (Ferman, 2016).

Para cada tipo de módulo existe um conjunto de funções que pode executar dentro dele. Por exemplo, em um módulo identificador pode existir uma função que retorna os tokens com a mesma sonoridade dos recebidos (Ferman, 2016).

O sistema então é capaz de propor programas aleatórios, que são árvores compostas de nós que contém um módulo e uma função, e processar uma quantidade de exemplos que mostram versões erradas e corrigidas de um mesmo texto, obtendo assim uma avaliação da capacidade de cada programa corrigir o texto. A cada ciclo a população de programa evolui segundo as normas da Programação Genética, sendo obtido um programa final capaz de normalizar o texto.

A implementação foi feita em Java 1.7 usando o *framework EpochX Genetic Programming for Research*¹⁵, versão 1.4.1.

A avaliação usa a medida BLEU (iBilingual Evaluation Understudy), uma medida também usada na avaliação de traduções:

$$BLEU = BP \times e^{(\sum_{n=1}^N w_n \times \log p_n)} \quad (5.1)$$

¹⁵<http://www.epochx.org/>

onde BP é um fator de ajuste, p_n é a precisão da normalização com n-gramas e w_n são fatores que balanceiam os termos.

5.7 Uma cadeia de pré-processamento em KNIME

É comum que todas as etapas do pré-processamento sejam encadeadas. Para isso, o KNIME permite uma descrição rápida, possuindo vários nós dedicados ao processo.

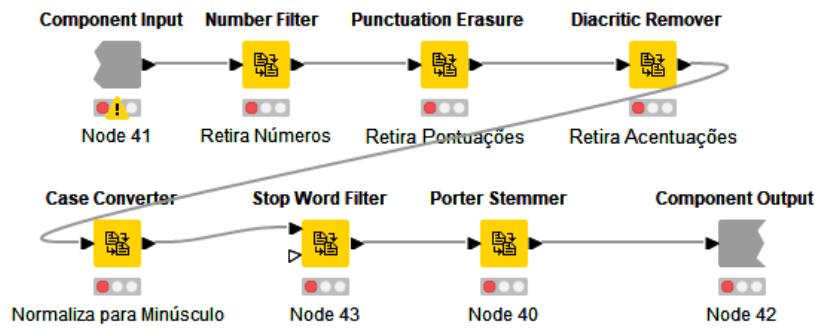


Figura 5.6: Cadeia de preprocessamento em KNIME

5.8 Sentenciação

Em algumas tarefas de processamento de texto é necessária também a sentenciação, ou separação dos textos, ou a junção dos *tokens*, em frases. Novamente, um problema que parece simples se torna complexo, tanto porque há complicações normais da língua, quanto porque temos a garantia que as normas cultas são seguidas nos textos que serão processados (Freitas, 2022).

Uma complicação normal da língua é o uso do símbolo do ponto tanto para indicar abreviações, como em “Dr.”. Outra é o uso de dois pontos para indicar que um personagem vai falar em um texto, o que normalmente deve ser considerado outra sentença (Freitas, 2022).

5.9 Codificação de Texto

Para finalmente representar o texto de uma forma indexável é necessário codificá-lo de alguma forma. Os modelos mais usados se baseiam no contexto de “bag of words”, onde não é usada a ordem das palavras, ou tokens, nos documentos, e é usado no modelos tradicionais de recuperação de informação e mineração de texto.

5.10 Codificação *one hot*

Nessa forma de classificação é criado um vocabulário com todos os termos disponíveis em todos os documentos, que define um vetor. A posição de cada palavra no vetor pode ser dada de várias

formas, como posição na ordem alfabética, pela primeira vez que a palavra aparece ou no final, pela frequência da palavra. Cada documento é então representado por um vetor onde cada posição toma o valor de 1 ou 0, se termo aparece ou não no documento. Essa é a representação do modelo booleano. No modelo vetorial esse valor é substituído por um peso calculado de alguma maneira que indique a importância do termo como definidor do documento dentro da coleção, como a medida tf-idf, discutida na Subseção 14.2.3.

5.10.1 Mapeamento dos termos em números

É comum também que, em certo momento de um processamento do texto, os *tokens* sejam transformadas em números. Por exemplo, no modelo vetorial, ou derivados, os *tokens* vão ser os índices dos vetores. Para essa transformação, tanto podem ser usadas as palavras originais como seus lemas ou radicais.

Essa transformação é feita normalmente por uma tabela de dispersão (*hash table*), para garantir a velocidade do processo. Em Python, o uso de um dicionário é provavelmente a melhor opção. No caso do uso de vetores, cada entrada do dicionário guardará a posição do vetor correspondente ao *token*.

5.11 Exercícios

Exercício 5.1:

Faça um programa que remova todos os caracteres de um texto em português que não seja uma letra ou letra acentuada.

Exercício 5.2:

Faça um programa que corrija as palavras de um texto em português seguindo o algoritmo proposto por Jurafsky.

Exercício 5.3:

Faça um programa que coloque as palavras de um texto em português em minúsculas, funcionando para os acentos e o c-cedilha.

Exercício 5.4:

Faça um programa que coloque todos os verbos de um texto em português no infinitivo.

Exercício 5.5:

Faça um programa que coloque todas as palavras de um texto em português no singular.

Exercício 5.6:

Faça um programa que coloque todas as palavras de um texto em português no gênero masculino (ou feminino).

Exercício 5.7:

Faça um programa que remova um grupo de palavras definidas como *stop-words* de um texto em português.

CAPÍTULO 6

ESTRUTURAS DE DADOS

Este capítulo apresenta algumas estruturas de dados adequadas para o modelo de recuperação de informação, onde dada uma chave deve ser obtido um valor.

6.1 Lista Invertida

A **lista invertida** (Zobel e Moffat, 2006; Zobel, Moffat e Ramamohanarao, 1998), ou o **índice invertido**, ou ***inverted files***, **arquivo invertido**, ou ainda ***posting file***, é a estrutura básica usada na maioria dos sistemas de indexação de documentos. O nome vem do fato que normalmente a indexação permite que se recupere todas as partes de um objeto conhecendo uma identificação do objeto, porém na lista invertida o objeto é recuperado a partir de suas partes.

Em qualquer representação de um documento por meio de suas partes constituintes, os termos, é muito provável que, comparado com todos os termos encontrados em todos os documentos de um conjunto, cada documento tenha apenas uma pequena quantidade de termos. Isso faz com que as representações matriciais termo × documento sejam esparsas, isto é, contenham muitos zeros. A lista invertida é uma forma de fazer essa representação de forma a economizar espaço.

6.1.1 Exemplo de listas invertidas

Sejam três documentos:

1. Maria comprou um quilo de banana prata no mercado.
2. Um quilo de prata vale muito dinheiro.
3. Maria tem dinheiro para comprar um quilo de banana, mas não de prata.

Um lista invertida desses documentos pode ser representada como na Tabela 6.1, após a substituição das maiúsculas por minúsculas.

É importante notar que para processá-las é importante que duas propriedades sejam mantidas: para o acesso rápido as chaves elas devem estar em uma estrutura que favoreça a busca, como em uma lista ordenada (complexidade para a busca $O(n)$), árvore binária (complexidade para a busca $O(\log n)$), ou em uma *BTree* ou *B+Tree* (complexidade para a busca $O(\log n)$), ou colocadas em uma estrutura de espalhamento (*hash*) (complexidade para a busca $O(1)$). Além disso, os documentos apontados

Tabela 6.1: Representação de um lista invertida.

Termo (chave)	Documentos
banana	1, 3
comprar	3
comprou	1
de	1, 2, 3
dinheiro	2, 3
maria	1, 3
mas	3
mercado	1
muito	2
no	1
não	3
para	3
prata	1, 2, 3
quilo	1, 2, 3
tem	3
um	1, 2, 3
vale	2

também devem estar ordenados, de modo que seja possível usar algoritmos rápidos para testar se uma chave está em dois documentos (E lógico).

Na Tabela 6.1 podemos ver algumas características que estão ou não tratadas. Foi ignorada a quantidade de vezes que uma palavra aparece em um texto, como a palavra “de” no item 3. Também podemos ver que a palavra “Um” foi colocada em minúsculas, e que a conjugação dos verbos separa “comprou” de “comprar”, o que muitas vezes é indesejado. Podemos também ver que, em uma coleção, é possível que algumas palavras apareçam em muitos documentos, e outras só em um. A própria escolha das palavras, ou sequência de palavras, ou tokens, é importante. Essas e outras questões são tratadas na fase de pré-processamento do texto.

Um uso comum da lista invertida, além de obter todos os documentos que possuem uma palavra, é obter todos os documentos que possuem um conjunto de palavras, usando o E ou ou OU lógico. O algoritmo da conjunção, conhecido como *merge*, implica em usar dois ponteiros que andam nas listas ordenadas de documentos e é descrito claramente em Christopher D. Manning, Raghavan e Schütze (2009a).

6.2 Melhorias na lista invertida

Várias melhorias podem ser feitas na lista invertida (Christopher D. Manning, Raghavan e Schütze, 2009a):

- usar *skip-pointers* para acelerar o algoritmo de merge;
- guardar a quantidade de documentos por palavra na lista, para poder escolher as menores listas em caso de uma conjunção de múltiplas palavras, acelerando o processo;
- guardar todas as posições da palavra em um documento;
- guardar quantas instâncias da palavra aparecem em um documento;

- processar bigramas (duas palavras em sequência) e outros n-gramas¹;
- usar estruturas de disco para manter listas maiores que a memória RAM, e
- usar cache com as estruturas de disco.

6.3 Listas Invertidas em Python

Em Python puro, a maneira mais eficiente de realizar uma lista invertida em memória é por meio de um dicionário, que podem ser salvas diretamente em um arquivo, se necessário, com pickle, ou sua versão mais rápida, cPickle, ou mesmo em JSON, para compatibilidade com outras linguagens. Se a lista não couber em memória, será necessário uma estrutura em disco mais complexa, o que não é uma alternativa comum em Python. Também pode se usar sqlite 3, que é leve e rápido, com uma tabela contendo a chave e a lista de documentos, evitando *joins*.

Existem alguns pacotes que oferecem estruturas em disco acessadas no paradigma chave-valor. BTrees² é um pacote criado para ZODB³, um banco de dados de objeto nativo em Python, que também é uma opção de uso.

Outras opções são possíveis para bases chave-valor: Berkeley DB, HDF5 ou Kyoto Cabinet.

6.3.1 Implementação trivial em Python

A implementação em Python do Programa 6.1 é bastante simples, e foi usada para gerar as linhas da Tabela 6.1.

Programa 6.1: Criação simples de uma lista invertida

```

1 # o método split só admite um separador, então removemos as vírgulas
2 base = [ "Maria comprou um quilo de banana prata no mercado" ,
3          "Um quilo de prata vale muito dinheiro" ,
4          "Maria tem dinheiro para comprar um quilo de banana mas não de prata"]
5
6 lista_invertida = {}
7
8 for i in range(len(base)):
9     fraselc = str.lower(base[i]).split()
10    for palavra in fraselc:
11        docs = lista_invertida.get(palavra, set())
12        docs.add(i+1)
13        lista_invertida[palavra]=docs
14
15 chaves = [i for i in lista_invertida.keys() ]
16 chaves.sort()
17
18 for i in chaves:
19     print(" ",i," & ",str(lista_invertida[i]), " \\\\"")
```

¹Na verdade, se a busca contém um trígrama ou uma sequência maior de palavras, o uso de bigramas que compõe o n-grama é capaz de atender a demanda rapidamente(Christopher D. Manning, Raghavan e Schütze, 2009a)

²<https://pypi.org/project/BTrees/>

³<https://zodb.org/en/latest/> e tutorial em <https://zodb.org/en/latest/tutorial.html>

6.4 Caches

Um **cache** é uma estrutura intermediária entre um processo que usa objetos e a fonte que os guarda. Em geral, essa estrutura intermediária fornece um acesso mais rápido que essa fonte, porém tem um tamanho menor, só podendo guardar alguns objetos. Como o cache tem um tamanho limitado, é necessária uma estratégia para liberar uma posição quando fica cheio e um objeto novo aparece. Existem várias estratégias possíveis, entre elas algumas simples de manter, como:

- **LRU** - *least recently used* - como guardar apenas os N mais recentemente acessados, descartando os que não são acessados há mais tempo;
- **FIFO** -*first in, first out* - guardar os N que entraram na lista mais recentemente, liberando, se necessário, o mais antigo na lista, e
- **LFU** - *least frequently used* - guardar os mais acessados, descartando o menos acessado.

Uma forma de usar um cache é para guardar resultados que são calculados, mas só precisam ser calculados uma vez, pois o resultado do cálculo será sempre o mesmo em todo processamento. Por exemplo, um processo de *stemming* leva algum tempo, mas dá sempre o mesmo resultado se for feito para a mesma palavra. Então, uma solução é, a cada palavra calculada, guardar o resultado em um cache que seja mais rápido que o processo de cálculo, o que evita que, em sua segunda aparição, tenha que ser calculada de novo. Aplicar um cache a uma função, o que se chama memoização, pode ser feito por meio de programação, mas apenas com uma anotação em Python, como a `@lru-cache`⁴

Caches podem ser muito úteis no processamento de palavras já que textos se caracterizam por usar muitas vezes as mesmas palavras.

6.5 Tabelas *Hash*

Tabelas *Hash* ou Tabelas de Espalhamento são uma estrutura de dado cuja complexidade de tempo esperado de inserção, busca e remoção é $O(1)$, porém ocupam um espaço que deve ser pré-alocado, logo maior do que o necessário a cada momento. Elas podem implementar tanto listas invertidas quanto caches.

Uma tabela de espalhamento é um estrutura de dados baseada em um espaço reservado, um arranjo ou vetor, onde o índice é calculado a partir da chave, por meio de uma função específica que pode ser calculada rapidamente, a função de espalhamento (Cormen et al., 2009). O vetor tem um tamanho proporcional ao número esperado de chaves, que devem ser distribuídas de forma pseudo-aleatória no mesmo por meio da função de espalhamento. Se duas chaves tiverem o mesmo *hash*, uma estratégia de tratamento de colisões é usada.

Um caso especial, conhecido como *hash* perfeito, faz com que todas as chaves se distribuam no vetor completo sem colisão, sendo $O(1)$ o tempo de pior caso. Um caso degenerado, todas as chaves vão ser mapeadas na mesma célula do vetor, e tem o tempo $O(n)$, supondo que o tratamento de colisão usa uma lista encadeada ou outra estrutura de busca sequêncial.

A forma mais simples de função de *hash* é o endereçamento direto. Nesse caso a chave é usada diretamente como o índice do vetor. Se palavras forem mapeadas em um número inteiro representando os primeiros caracteres das mesmas, isso pode implicar em uma vetor que pode ser usado. Os sufixos, porém, podem causar um grande número de colisões.

⁴Veja mais sobre memoização em Python em <https://realpython.com/lru-cache-python/>

Usando as duas primeiras chaves do exemplo como referência, a Tabela 6.2 mostra um passo a passo das inserções das chaves para elas. É usada uma tabela de endereçamento direto (Cormen et al., 2009) baseada na primeira letra da palavra. Como estratégia de tratamento de colisão vamos usar a de pegar a célula vazia seguinte, o que é o tipo mais simples de endereçamento aberto (Cormen et al., 2009). No exemplo da Tabela 6.2 as palavras “dinheiro”, “mercado” e “muito” colidem, respectivamente com “de” e “maria”, e acabam algumas células a frente.

Tabelas de Espalhamento são tratadas detalhadamente no capítulo 11 de Cormen et al. (2009). Em Python, os dicionários, `dict`, são implementados como tabelas de espalhamento com endereçamento aberto. Ao serem criados, os dicionários possuem 8 slots, e mudam de tamanho quando estão com dois terços de ocupação⁵. A implementação em CPython tem mais de 5500 linhas⁶.

⁵<https://stackoverflow.com/questions/327311/how-are-pythons-built-in-dictionaries-implemented>

⁶<https://github.com/python/cpython/blob/main/Objects/dictobject.c>

	Passo e Palavra											
Índice Direto	1	2	3	4	5	6	7	8	9	10	11	12
a	maria	comprou	um	quilo	de	banana	prata	no	mercado	vale	muito	dinheiro
b						banana		banana		banana		banana
c		comprou	comprou	comprou	comprou	comprou		comprou		comprou		comprou
d					de	de	de	de		de	de	
e												dinheiro
f												
g												
h												
i												
j												
k												
l												
m	maria	maria	maria	maria	maria	maria	maria	maria	maria	maria	maria	maria
n								no	no	no	no	no
o								mercado	mercado	mercado	mercado	mercado
p							prata	prata	prata	prata	prata	prata
q					quilo	quilo	quilo	quilo	quilo	quilo	quilo	quilo
r												muito
s												
t												
u			um	um	um	um	um	um	um	um	um	um
v										vale	vale	vale
x												
z												

Tabela 6.2: Exemplo de inserções em uma tabela de endereçamento direto com endereçamento aberto. Palavras que colidem estão em marcadas.

6.6 Arquivos de assinatura, impressões Digitais e LSH

Arquivos de assinatura, impressões digitais e ***Locality Sensitive Hashing*** (LSH) são técnicas relacionadas usadas para pesquisa de vizinhos aproximados em dados de alta dimensionalidade.

Um **arquivo de assinatura**, é uma representação compacta de um conjunto de documentos que permite uma busca rápida de similaridade. Um arquivo de assinatura mapeia cada documento para uma assinatura de comprimento fixo que resume seu conteúdo. A assinatura é geralmente obtida através de uma função de hash com a característica de levar documentos semelhantes a valor semelhantes. A ideia é tornar possível comparar rapidamente as assinaturas de dois documentos para determinar se eles são semelhantes.

As **impressões digitais**, do inglês *fingerprint*, são semelhantes aos arquivos de assinatura, pois também resumem o conteúdo de um documento ou ponto de dados em uma representação compacta. Impressões digitais são comumente usadas em informática química e bioinformática para representar moléculas. Elas são geralmente obtidas através de uma combinação de funções de hash e redução, que levam em conta propriedades específicas dos dados.

O **LSH** é uma técnica usada para realizar pesquisa de vizinhos aproximados em dados de alta dimensionalidade. É uma forma de hash que mapeia pontos de dados para **baldes** (*buckets*), de forma que pontos próximos na alta dimensionalidade tenham mais chances de serem mapeados para o mesmo balde. O LSH pode ser visto como uma generalização do *fingerprinting*, onde, em vez de resumir o conteúdo de um documento em uma representação de comprimento fixo, ele mapeia os pontos de dados para um conjunto de baldes de maneira probabilística.

CAPÍTULO 7

SIMILARIDADE

Este capítulo introduz a questão da similaridade, já que grande parte das soluções para os problemas de recuperação, classificação e agrupamento de documentos parte do princípio de calcular a similaridade entre documentos ou entre um documento e uma consulta.

É possível começar com termos muito simples. "Objeto", por exemplo, é uma palavra que é usada, a maior parte do tempo, sem que seja dada uma definição específica para ela. Um objeto pode ser qualquer coisa, uma coisa, um ser, um construto filosófico, etc.

Um objeto é qualquer coisa que pode ser destacada como tendo uma identidade. A identidade de um objeto é a propriedade de ser distinguível de qualquer outro objeto. Dois objetos são idênticos se eles são realmente o mesmo objeto, ou seja, não podem ser distinguidos um do outro de forma alguma. Dois objetos são iguais se não podem ser distinguidos um do outro de forma alguma, exceto por sua identidade. Em algumas áreas ou teorias, principalmente as mais abstratas, a igualdade entre dois objetos tem o significado de identidade, isto é, objetos iguais são o mesmo objeto. Já quando os objetos considerados são concretos, igualdade e identidade tendem a ser conceitos distintos.

Semelhança é uma avaliação psicológica e contextual de como dois objetos estão próximos de serem iguais. É importante registrar a semelhança como um mecanismo psicológico porque, em geral, deve ser considerada pessoa por pessoa. Além disso, é contextual porque depende altamente dos parâmetros sendo considerados naquele momento. Por exemplo, uma cadeia de supermercados considerará o guaraná e uma bebida de cola como produtos similares. No entanto, se você é um cliente em um restaurante que não tem uma marca específica de cola, pode não considerar o guaraná como um produto semelhante.

Dois objetos são similares se podem ser distinguidos um do outro mas têm alguma dimensão, propriedade ou comportamento que pode ser considerada semelhante, no sentido geral do Português, de alguma forma. Em muitos sistemas é importante definir uma medida de similaridade.

Como medida, a similaridade assume normalmente um valor entre 0, representando nenhuma similaridade, até 1, que representa a igualdade. Existem algumas medidas que indicam similaridade que variam no intervalo $[-1,1]$, como a correlação. Uma **medida de similaridade** é uma função que recebe dois objetos e responde com uma avaliação de sua similaridade na forma de um número real.

De acordo com Blough (2001), que discutem a similaridade no contexto da psicologia e da literatura de aves, existem cinco abordagens teóricas para ela:

Abordagem do elemento comum onde a representação dos estímulos é considerada como um conjunto e a similaridade é medida como o número de elementos comuns entre eles.

Modelos de modelo geralmente aplicados no problema de reconhecimento de objetos, usam uma verificação ponto a ponto para verificar a similaridade entre um objeto alvo e um objeto modelo.

Modelos geométricos onde objetos são representados como uma coordenada em um espaço abstrato, enquanto a dissimilaridade é representada como a distância entre os objetos e a similaridade pode ser calculada a partir dela.

Modelos de recurso que supõem que um objeto é representado por um conjunto de atributos, e a similaridade entre dois objetos A e B é uma função dos atributos comuns, dos atributos que estão em A mas não em B , e dos atributos que estão em B mas não em A .

Teoria Geon também geralmente aplicada ao reconhecimento de objetos, usa formas primitivas e partes elementares para representar os objetos, e a similaridade é uma medida da relação entre eles.

Todas essas abordagens baseiam-se no princípio de que a similaridade entre objetos é resultado da comparação de representações internas dos objetos de estímulo (Blough, 2001). A partir dessas descrições breves, podemos detectar três abstrações matemáticas básicas que são geralmente usadas para lidar com a similaridade: geometria, conjuntos e grafos¹.

Também é possível classificar as abordagens de similaridade em relação à quantidade de informação disponível para a função ou algoritmo de similaridade.

Similaridade de caixa-preta (ou funcional) onde não comparamos os objetos diretamente, mas sim alguma propriedade calculada como uma função desses objetos. Um exemplo é medir a similaridade de dois conjuntos olhando apenas para o número de elementos dos conjuntos.

Similaridade de caixa branca onde não se compara o objeto propriamente dito, mas suas partes. Essa abordagem pode ser rasa ou profunda, dependendo se o raciocínio é aplicado apenas as partes do objeto, ou recursivamente para as partes das partes. Um exemplo é comparar conjunto pela quantidade de elementos que são iguais ou diferentes dentro deles.

7.1 Teoria da similaridade baseada em características de Tversky

Em um artigo seminal de similaridade aplicada à psicologia, Tversky (1977)² apresenta uma teoria axiomática geral de similaridade que o descreve com base em uma função de correspondência ordinal entre conjuntos de atributos.

Tversky questiona a visão geométrica da similaridade, baseada em medidas de distância em um espaço vetorial, pelo fato de experimentos psicológicos mostrarem que a similaridade não segue pressupostos básicos das medidas geométricas, como minimalidade, simetria, desigualdade triangular e transitividade. Por exemplo, pode-se achar que “Brasil” e “Portugal” são países semelhantes, porque ambos falam português, e também descobrir que “Portugal” e “Inglaterra” também são países semelhantes, porque eles estão na Europa, e, no entanto, nunca encontraram qualquer semelhança entre “Brasil” e “Inglaterra”.

¹Os modelos de modelo e teoria Geon podem ser vistos como similaridade baseada em grafos, onde pontos ou formas são nós e suas relações são vértices

²Amos Tversky (1937-1996), um psicólogo cognitivo e matemático israelense e colaborador de longa data do Prêmio Nobel de Ciências Econômicas de 2002, Daniel Kahneman(Ryerson, 2002). Embora o prêmio Nobel não seja distribuído postumamente, seu nome foi citado pelo comitê do Nobel e pelo próprio Kahneman no discurso de aceitação(Kahneman, 2003).

Tversky (1977) começa definindo um conjunto $\Delta = \{a,b,c,\dots\}$ de objetos (ou estímulos), onde cada elemento de Δ está associado a um conjunto de atributos, respectivamente anotados como A,B,C,\dots . Essas características podem variar de atributos abstratos a concretos, como simpatia ou cor do olho. A similaridade entre dois objetos será função da comparação de seus conjuntos de atributos.

Autores posteriores fizeram uma diferença entre estímulos (e suas características) e suas percepções.

Dados dois objetos a e b e seus respectivos conjuntos de atributos A e B , a similaridade entre A e B é uma função de quais atributos são comuns a A e B , quais feições atributos a A e não a B e, por fim, quais atributos pertencem a B e não a A .

7.1.1 Axiomas de Tversky

Axioma 1 (Correspondência (*matching*) (Tversky, 1977)). $s(a,b) = F(A \cap B, A - B, B - A)$, onde F é alguma função de valor real em três argumentos e s é uma função de correspondência.

Axioma 2 (Monotonicidade (Tversky, 1977)). $s(a,b) \geq s(a,c)$ sempre que $(A \cap B) \supset (A \cap C)$, $(A - B) \subset (A - C)$ e $(B - A) \subset (C - A)$. Além disso, se qualquer uma das inclusões for própria, então a desigualdade é estrita.

Axioma 3 (Independência (Tversky, 1977)). Suponha que os pares (a,b) e (c,d) , assim como os pares (a',b') e (c',d') , possuem correspondência nos mesmos dois atributos, enquanto os pares (a,b) e (a',b') , bem como os pares (c,d) e (c',d') , possuem correspondência nos com o restante (terceiro) atributos. Então $s(a,b) > s(a',b') \iff s(c,d) > s(c',d')$.

Axioma 4 (Solvabilidade (Tversky, 1977)). . Para todos os pares $(a,b), (c,d), (e,f)$ de objetos em Δ existe um par (p,q) que corresponde a eles com eles, respectivamente, no primeiro, segundo e terceiro atributo, ou seja, $P \cap Q \simeq A \cap B$, $P - Q \simeq C - D$, e $Q - P \simeq F - E$. (ii). Suponha que $s(a,b) > t > s(c,d)$. então existe e,f com $s(e,f) = t$, tal que se (a,b) e (c,d) possuem correspondência em um ou dois atributos, então (e,f) possui correspondência com eles sobre esses atributos. (iii). Existem pares (a,b) e (c,d) de objetos em Δ que não possuem correspondência em nenhum atributo.

Axioma 5 (Invariância (Tversky, 1977)). Suponha que V, V', W, W' pertençam a $\Phi_i e \Phi_j$, onde $i, j \in \{1, 2, 3\}$, então $(V, V')_i \simeq (W, W')_j \iff (V, V')_j \simeq (W, W')_i$.

Teorema 1 (Teorema da Representação (Tversky, 1977)). Suponha que os Axiomas de 1 a 5 sejam válidos. Então existe uma escala de similaridade S e uma escala não negativa f tal que para todo a, b, c, d em A

1. $S(a,b) > S(c,d) \iff s(a,b) > s(c,d)$,
2. $S(a,b) = \theta f(APiB) - \alpha f(A - B) - (\beta(B - A))$, para algum $\theta, \alpha, \beta > 0$.
3. f e S são escalas de intervalo³.

7.2 Similaridade em conjuntos

Toda medida de similaridade S para conjuntos deve atender a duas condições de contorno: um conjunto é totalmente similar a ele mesmo e não tem nenhuma similaridade com o conjunto vazio. Além disso, ela deve ser comutativa. Considerando que a medida de similaridade é uma medida fuzzy

³Uma escala de intervalo é um conjunto de valores onde as distâncias podem ser comparadas, mas não necessariamente a razão entre eles.

que responde um número no intervalo [0,1], as condições mais comuns estão expressas nas equações a seguir.

$$S : U \rightarrow [0,1] \quad (7.1)$$

$$S(A,A) = 1 \quad (7.2)$$

$$S(A,\emptyset) = 0 \quad (7.3)$$

$$A \cap B = \emptyset \implies S(A,B) = 0 \quad (7.4)$$

$$S(A,B) = S(B,A) \quad (7.5)$$

$$(7.6)$$

Uma função amplamente usada para calcular a **similaridade entre dois conjuntos** é definida como a razão entre o tamanho da interseção e o tamanho da união desses conjuntos, o que é conhecido como o Coeficiente de Similaridade de Jaccard.

$$S_J(A,B) = \frac{\|A \cap B\|}{\|A \cup B\|} \quad (7.7)$$

Outro **coefficientes de similaridade** conhecido é o **Coeficiente de Dice**.

$$S_{\text{Dice}} = \frac{2 \times \|A \cap B\|}{\|A\| + \|B\|} \quad (7.8)$$

Embora as duas medidas sejam semelhantes, elas têm diferenças sutis em como medem a sobreposição entre os conjuntos. A similaridade de Jaccard tende a ser mais sensível a diferenças no tamanho dos conjuntos, enquanto a similaridade de Dice tende a ser mais sensível a diferenças no número de elementos compartilhados entre os conjuntos. As duas se relacionam da seguinte forma:

$$S_J = \frac{S_{\text{Dice}}}{2 - S_{\text{Dice}}} \quad (7.9)$$

$$S_{\text{Dice}} = \frac{2S_J}{S_J + 1} \quad (7.10)$$

Outra característica é que a similaridade de Jaccard atende a desigualdade do triângulo, isso é, $1 - S_J$ pode ser usado como uma métrica, porém Dice não a satisfaz, logo $1 - S_{\text{Dice}}$ não pode ser usado como distância.

No entanto, essas definições tem limitações. Por exemplo, elas discutem a semelhança de conjuntos com base na igualdade de seus elementos. Por exemplo, o conjunto de cores

$$\{\text{azul}, \text{vermelho}\}$$

teria semelhança zero quando comparado ao

$$\{\text{ciano}, \text{magenta}\}.$$

Isso é desejável? Algumas vezes, sim, mas outras vezes, não, pois pode ser interessante ter uma medida de semelhança de conjuntos baseada na medida de semelhança de seus membros. Também não leva em consideração outras diferenças. Por exemplo, não haverá diferença na semelhança entre dois conjuntos

do mesmo tamanho e dois conjuntos de tamanhos diferentes se sua interseção e sua união também têm o mesmo tamanho.

Além disso, os Coeficientes de Jaccard e de Dice só avaliam as co-ocorrências, não tratando das ausências comuns. Isso é tratado por uma medida conhecida como coeficiente simples de casamento (*simple matching coefficient*) (Cross e Sudkamp, 2002):

$$S_{smc}(X,Y) = \frac{|X \cap Y| + |\bar{X} \cap \bar{Y}|}{|X \cup Y| + |\bar{X} \cup \bar{Y}|} \quad (7.11)$$

Outra medida que é algumas vezes usada é o coeficiente de sobreposição (Vijaymeena e Kavitha, 2016):

$$S_s = \frac{|X \cap Y|}{\min(|X|, |Y|)} \quad (7.12)$$

7.3 Similaridade em vetores

Muitos sistemas de similaridade usam o inverso da distância entre vetores como uma medida de similaridade, porém no caso do uso de vetores para representar palavras, é mais comum usar o cosseno do ângulo entre os vetores. Isso acontece porque o tamanho do vetor é normalmente ligado a quantidade de vezes que uma palavra aparece em documentos, enquanto a direção do vetor é ligada ao significado da palavra.

7.3.1 Similaridade como inverso da distância

Uma medida diretamente associada a similaridade é a distância. Uma **função de distância**, ou **métrica**, para um **espaço**, que se torna um **espaço métrico**, \mathbb{S} é definida como uma função:

$$D : \mathbb{S} \times \mathbb{S} \rightarrow \mathbb{R} \quad (7.13)$$

que deve satisfazer quatro propriedades.

1. A distância entre um objeto e ele mesmo é zero, $D(a,a) = 0$;
2. A distância é sempre positiva, $D(a,b) \geq 0$;
3. A distância é simétrica, isto é, $D(a,b) = D(b,a)$
4. a distância segue a desigualdade do triângulo, i.e., dados a , b , e c : $D(a,b) + D(b,c) \geq D(a,c)$.

Sempre que é possível caracterizar uma medida de distância D , a similaridade pode ser definida a partir dela. É comum usar simplesmente D^{-1} como medida de similaridade, com o cuidado de não calcular o valor para $D = 0$ e considerar, no caso, a similaridade como 1. Várias são as transformações possíveis entre similaridade e distância, principalmente se a distância já está normalizada no intervalo $[0,1]$ (caso da Equação 7.15):

$$S = \begin{cases} \frac{1}{D} & \text{se } D \geq 0 \\ 1 & \text{se } D = 0 \end{cases} \quad (7.14)$$

$$S = 1 - D \quad (7.15)$$

$$S = \frac{1}{D+1} \quad (7.16)$$

$$S = 1 - D^2 \quad (7.17)$$

$$(7.18)$$

As medidas de distância mais reconhecidas em geral são as euclidiana e a de Manhattan. Outra medida reconhecida é a Chebyshev, todas apresentadas a seguir. Essas medidas podem ser vistas como casos especiais da distância genérica L_p , ou distância de Minkowski, definida para um espaço n -dimensional como:

$$l_p = \left(\sum_{i=1}^n (\|x_i - y_i\|^p) \right)^{\frac{1}{p}} \quad (7.19)$$

A **distância euclidiana**, ou L_2 é a que estamos acostumados da Geometria euclidiana, é altamente intuitiva, sendo o tamanho da reta que liga dois pontos. Apesar de muito comum, ela é sensível a mudanças de escala, o que pode exigir a normalização dos vetores antes de usá-la. Por exemplo, se estamos comparando características de profissionais e uma dimensão que representa o salário anual, normalmente em milhares, e uma que representa a idade, sempre menor que 100, pode ser interessante normalizar para que as dimensões de salário não domine a distância. Outra característica importante é que em muitas aplicações não é necessário calcular a raiz quadrada, sendo usada a distância euclídea quadrada, D_{se} , o que acelera o uso dessa função Harmouch, 2021.

$$D_{\text{euclidiana}}(A,B) = L_2 = \sqrt{\sum_i (a_i - b_i)^2} \quad (7.20)$$

$$D_{se}(A,B) = \sum_i (a_i - b_i)^2 \quad (7.21)$$

A **distância de Manhattan**, ou L_1 é também chamada de distância de quarteirões ou do taxista, e é comparável a caminho mínimo entre dois pontos de um grid quadriculado, como um táxi atravessando uma cidade onde todas as vias são de mão dupla, de um ponto a outro. Essa distância é mais adequada para *datasets* com atributos discretos ou binários.

$$D_{\text{Manhattan}}(A,B) = L_1 = \sum_i \|a_i - b_i\| \quad (7.22)$$

A distância de Canberra é uma versão ponderada da distância de Manhattan:

$$D_{\text{Canberra}}(A,B) = \sum_i \frac{\|a_i - b_i\|}{\|a_i\| + \|b_i\|} \quad (7.23)$$

A **distância de Chebyshev** é definida como a maior diferença entre todas as diferenças em cada dimensão, e é chamada também de distância do tabuleiro de xadrez, ou L_∞ .

$$D_{\text{Chebyshev}}(A,B) = L_\infty = \max_i (a_i - b_i) \quad (7.24)$$

Outra distância possível é $L_{-\infty}$:

$$L_{-\infty} = \min_i (a_i - b_i) \quad (7.25)$$

Uma possível necessidade é normalizar as dimensões, ou dar pesos diferentes as dimensões. Normalmente isso é feito com a distância euclidiana, criando a euclidiana ponderada.

$$D_{we} = \sqrt{\sum_i w_i (a_i - b_i)^2} \quad (7.26)$$

A **distância entre dois conjuntos** também pode ser medida, sendo a similaridade a sua inversa. Para isso é necessário codificar os conjuntos como um vetor. Seja um conjunto A , $A \subset U$, e o vetor que o representa $\vec{A} = a_i$, onde $a_i = 1$ se o i -ésimo elemento de U representado pela i -ésima posição do vetor está no conjunto A e $a_i = 0$ se não está, as equações a seguir apresentam o cálculo dessas distâncias. Uma aplicação deve tomar cuidado porque a distância zero não possui uma inversa matemática, mas a similaridade, no caso, deve ser considerada 1.

7.3.2 Similaridade do cosseno

Ao usar o inverso da distância se considera o tamanho do vetor, porém em alguns casos isso pode não ser interessante, como veremos no Modelo Vetorial. Especificamente, quando o interesse é maior na direção dos vetores do que em sua magnitude, a **similaridade do cosseno** é mais adequada.

$$\text{sim}_{\cos}(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\|_2 \|\vec{y}\|_2} = \frac{\sum_{i=1}^n x_i y_i}{\|\vec{x}\|_2 \|\vec{y}\|_2} \quad (7.27)$$

Onde “.” é o produto escalar.

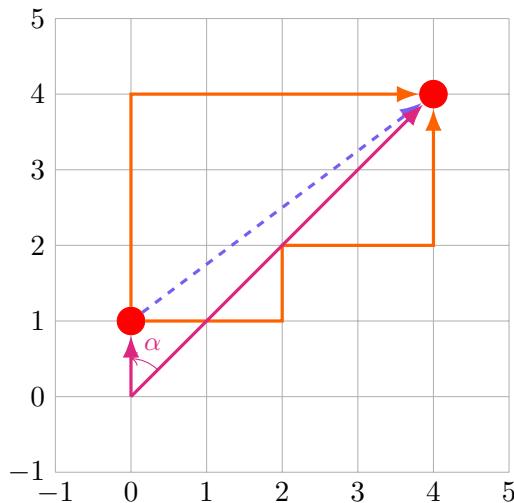


Figura 7.1: Formas de medir a distância entre dois pontos em um espaço. Em magenta a distância do cosseno, em azul tracejado a distância euclidiana e em laranja duas medidas (que vão resultar no mesmo valor) da distância de Manhattan

7.3.3 Funções de erro

Funções de erro, ou **funções de perda (loss functions)**, também são usadas como funções de distância ou dissimilaridade em vários algoritmos, mesmo que algumas não obeleçam a desigualdade do triângulo.

Duas funções comuns de erro são a **soma da diferença absoluta** e a **soma do quadrado das diferenças**, que são, respectivamente, L_1 e L_2^2 .

Porém as mais usadas normalmente são o **erro médio absoluto (MAE)** e o **erro médio quadrático (MSE)**, apresentados a seguir, são as versões normalizadas dos erros, e não são distâncias no sentido teórico da palavra.

$$D_{\text{MAE}}(x,y) = \frac{\|\vec{x} - \vec{y}\|_1}{n} = \frac{1}{n} \sum_{i=1}^n \|x_i - y_i\| \quad (7.28)$$

$$D_{\text{MSE}}(x,y) = \frac{\|\vec{x} - \vec{y}\|_2^2}{n} = \frac{1}{n} \sum_{i=1}^n (x_i - y_i)^2 \quad (7.29)$$

(7.30)

7.4 Similaridade entre strings

A similaridade entre strings pode ser medida pela **distância de edição**, ou seja, quantas operações simples devem ser feitas para transformar uma string na outra. Existem versões diferentes dos conjuntos de edições, mas sempre incluem a troca ou a exclusão de letras. A **distância de Levenshtein** é o número de operações de inserção, exclusão ou substituição necessárias para transformar uma string na outra. A **distância de Damerau-Levenshtein** inclui também a operação de transposição. A **maior subsequência comum** conta apenas inserções e exclusões, mas não substituições. A **distância de Hamming** só é aplicada a strings de mesmo tamanho, e só conta as substituições. A **distância de Jaro** conta apenas transposições.

A definição formal da função de Levenshtein permite definir claramente como contar substituições, inserções e exclusões:

$$\text{lev}(a,b) = \begin{cases} |a| & , \text{ se } |b| = 0 \\ |b| & , \text{ se } |a| = 0 \\ \text{lev}(a[1, :], b[1, :]) & , \text{ se } a[0] = b[0] \\ 1 + \min \begin{cases} \text{lev}(a[1, :], b) \\ \text{lev}(a, b[1, :]) \\ \text{lev}(a[1, :], b[1 :]) \end{cases} & , \text{ em outro caso} \end{cases} \quad (7.31)$$

Seu cálculo a partir da definição é pouco eficiente, mas simples, e aparece na Programa 7.1. Para tentar acelerar a função é usada a técnica de memoização por meio de um cache lru.

Programa 7.1: Função para calcular a distância Levenshtein segundo sua definição

```

1 def distancia_levenshtein(s1, s2):
2     # Caso base: se uma das strings é vazia, a distância é o comprimento da
        # → outra string
3     if len(s1) == 0:
4         return len(s2)
5     elif len(s2) == 0:
6         return len(s1)
7
8     # Caso recursivo:
9     # Se os últimos caracteres de s1 e s2 são iguais, a distância é a mesma que
        # → a distância entre os prefixos

```

```

10     # de s1 e s2 sem o último caractere
11     if s1[0] == s2[0]:
12         return distancia_levenshtein(s1[1:], s2[1:])
13
14     # Caso contrário, a distância é o mínimo de:
15     # - a distância entre s1 e o prefixo de s2 sem o último caractere, mais 1 (
16     #   ↪ inserção)
17     # - a distância entre o prefixo de s1 sem o último caractere e s2, mais 1 (
18     #   ↪ deleção)
19     # - a distância entre o prefixo de s1 sem o último caractere e o prefixo de
19     #   ↪ s2 sem o último caractere,
20     #   mais 1 (substituição)
21
22 else:
23     return 1 + min(distancia_levenshtein(s1[1:], s2), # deleção
24                     distancia_levenshtein(s1, s2[1:]), # inserção
25                     distancia_levenshtein(s1[1:], s2[1:])) # substituição

```

A implementação mais eficiente da distância de Levenshtein usa matrizes e programação dinâmica para evitar a repetição de cálculos desnecessários. A ideia é armazenar os resultados parciais em uma matriz e reutilizá-los posteriormente para calcular a distância de Levenshtein entre as duas strings.

A distância de Damerau-Levenshtein também pode ser definida recursivamente, desta vez a partir de uma função auxiliar $d_{a,b}(i,j)$:

$$d_{a,b}(i,j) = \min \begin{cases} 0 & \text{se } i = j = 0, \\ d_{a,b}(i-1,j) + 1 & \text{se } i > 0, \\ d_{a,b}(i,j-1) + 1 & \text{se } j > 0, \\ d_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} & \text{se } i, j > 0, \\ d_{a,b}(i-2,j-2) + 1_{(a_i \neq b_j)} & \text{se } i, j > 1 \text{ e } a_i = b_{j-1} \text{ e } a_{i-1} = b_j, \end{cases} \quad (7.32)$$

Sendo que a função de Damerau-Levenshtein entre duas strings a e b , de tamanho $|a|$ e $|b|$ é:

$$d_{\text{dl}}(a,b) = d_{a,b}(|a|,|b|) \quad (7.33)$$

7.5 Outras medidas de similaridade e distância

Este capítulo discute de forma geral os conceitos de similaridade e distância, porém existem questões específicas as áreas de aplicações. Por exemplo, distâncias entre dois nós de um grafo pode ser calculadas como a soma dos pesos das arestas do caminho mais curto entre eles e existem várias medidas de similaridade de texto que tentam compensar o efeito do tamanho do texto no valor dos vetores que os representam. Algoritmos de agrupamento e também usam outras medidas de distância. Especificamente, os modelos de recuperação de informação e as modelagens feitas para aprendizado de máquina usam medidas específicas que serão tratadas ao longo deste livro.

7.6 Exercícios

Exercício 7.1:

Implemente uma função que calcule a distância de Levenshtein usando matrizes e programação dinâmica.

Exercício 7.2:

Implemente todas as funções de distância descritas para vetores.

Exercício 7.3:

Implemente todas as funções de distância descritas para vetores.

Exercício 7.4:

Implemente todas as funções de distância descritas para conjuntos.

Exercício 7.5:

Implemente todas as funções de distância descritas para strings.

Parte III

Recuperação de Texto

CAPÍTULO 8

RECUPERAÇÃO DA INFORMAÇÃO

8.1 Motivação

Pessoas precisam de informações, seja por questões ligadas ao trabalho, ao estudo, a diversão ou algum interesse pessoal. As duas fontes de informação existentes são pessoas ou registros criados por essas pessoas, como textos, vídeos e sons, que chamamos de documentos.

Este capítulo trata da busca de documentos. Esse sempre foi um problema razoavelmente importante, porém com a explosão da quantidade de informações disponíveis na forma de documentos digitais, sejam eles banco de dados, documentos, páginas web ou outros, a busca passou a ser uma tarefa muito mais difícil. Onde antes o interesse maior era achar algo relevante sobre uma necessidade da pessoa dentro de uma coleção fechada em um ambiente como uma biblioteca, hoje é um ato diário, por meio do uso do WWW ou de aplicativos. Além disso, a quantidade de documentos que podem atender a demanda de quem faz a busca pode ser enorme, então organizá-los em uma ordem ótima passou a ser uma questão ainda mais crítica.

8.1.1 Escalas do Problema de Busca

Imagine uma criança procurando saber a origem do Homem Aranha em sua biblioteca particular contendo 50 revistas em quadrinhos. Seria simples para ela folhear (*browse*) suas 50 revistas e verificar se há alguma informação relevante.

Um passo além, imagine um estudante de Computação chegando à biblioteca de sua universidade em busca de aprender a programar em Python, onde há no total 50 mil livros. Já não faz mais sentido o estudante olhar livro a livro, percorrendo os corredores. Nas bibliotecas, porém, existe várias tecnologias que permitem ao estudante saber melhor qual livro: uma classificação usando um método padronizado e catálogos por assunto, por título, ou por autor. Algumas bibliotecas podem até ter todos seus livros classificados de formas mais sofisticadas, como sistemas computacionais que registram inclusive resumos. Por um catálogo desse tipo, o estudante vai saber que livros de Python estão disponíveis e até sua posição na biblioteca, mas qual o livro que será útil para ele? O estudante provavelmente ainda vai ter que selecionar alguns e folheá-los de forma a entender sua complexidade, pois pode ser que nada saiba sobre programação, ou ao contrário, que já saiba programar até mesmo em Python e só queira tirar uma dúvida.

Finalmente, a situação mais comum hoje em dia. Nossa usuária da internet busca, na Web, um material sobre empreendedorismo, para descobrir como pode pensar um negócio que deseja abrir. Quantos milhares ou milhões de documentos ou páginas web tratam sobre o assunto, com que profundidade, em que língua¹? Qual desses documentos atende a necessidade do usuário?

8.1.2 Como Atender o Usuário da Busca

É difícil atender o usuário por vários motivos. Alguns deles estão relacionados ao discutido no Capítulo 2, ou seja, basicamente a interpretação dos textos, ou da própria comunicação com o sistema de busca, pelas pessoas.

Outros motivos estão ligados à tecnologia. Primeiro, é muito difícil compreender automaticamente o que está no documento. Em segundo lugar, a conversação do sistema com o usuário comum tem que ser simples, a não ser em casos muito especializados. Muitas vezes o usuário espera um resultado fornecendo apenas uma ou duas palavras.

Ainda um terceiro grupo de motivos está ligado a qualidade dos documentos disponíveis. São muito documentos, alguns cópias de outros, outros versões que já não valem mais, antigos ou com informações erradas, o que traz a questão da atualidade do que é encontrado. Porém, nem sempre o mais atual é o mais correto. Além disso, a qualidade da informação deve ser analisada em relação ao leitor. Uma criança querendo entender a lei da gravidade não pode ler o mesmo texto que um doutorando em Física. A linguagem pode ser ou não adequada a idade, experiência, *background*, opiniões ou mesmo o estado de espírito do usuário. Mais recentemente, a questão dos documentos falsos se tornou muito importante, principalmente porque algumas mentiras podem ser não óbvias e sutis.

Tudo isso nos leva a alguns conceitos discutidos a nível conceitual neste capítulo e em nível prático em outras partes do livro.

8.1.3 A Recuperação da Informação na História

Antes dos computadores já existiam livros e catálogos. Bibliotecas existem desde aproximadamente 2750 A.C. (W. G. Stock e M. Stock, 2013). Logo, o problema de achar algo desejado dentro de uma coleção, existe há mais de 4000 anos. Ao longo desse tempo houve alguns marcos que são listados a seguir.

- Circa 1900 a.c., dessa época foi encontrado um tablete babilônico em sumério, com escrita cuneiforme, que lista em suas duas faces 62 obras literárias, que é a evidência mais antiga de uma lista de documentos conhecida²(Baltar e Albuquerque, 2018);
- Circa 1300 AC, dessa época foram encontradas lista com títulos e informações descritivas detalhadas encontradas em tabletes hititas (Norman, 2023);
- Séc III e II AC, um marco na história é a Biblioteca de Alexandria e os Pinakes, os documentos que a organizavam. Eles introduzem os conceitos de autor e ordem alfabética, entre outros. Acreditasse que ela continha 700.000 rolos de papiro. Até hoje a destruição da Biblioteca de Alexandria, quando estava para ser transportada para Roma, é lastimada pelos historiadores;
- Séc VI, São Bento ensina os monges a copiar documentos, tornando-os os primeiros copistas e catalogadores de livros;
- Séc VII até o IX, surgem os inventários de livros;

¹Uma busca feita por Google com a palavra empreendedorismo em 2 de janeiro de 2021 encontrou 132 milhões de hits.

²<https://cdli.mpiwg-berlin.mpg.de/artifacts/255993>

- Entre 822 e 842 surgem catálogos mais organizados, entre eles, em 831, o do mosteiro beneditino de Sait-Requier, na França. Bibliotecas crescem de tamanho, mas ainda na ordem de 600 ou 700 livros.
- Final do Séc. XIII, as bibliotecas monásticas inglesas criam um catálogo coletivo;
- 1389, é criado o primeiro catálogo organizado como tal, no convento de St. Martin, em Dover;
- 1560, Trefler, um monge beneditino, desenvolve um sistema de classificação e localização, e um catálogo de 5 partes.
- Século XVIII, se desenvolve a pesquisa científica e aparece o primeiro código de catalogação nacional, na França, em 1791. Catalogação passa a ser feita em fichas.
- 1839, regras de Pannizzi
- 1876 Cutter publica um código de catalogação com normas;
- 1876 Melvil Dewey inventa o sistema Dewey de classificação, usado até hoje;
- 1895 Paul Otlet e La Fontaine inventam o CDU (Classificação Decimal Universal), uma notação que permite falar da interseção de assuntos;
- 1933 Classificação facetada é proposta por Shiiali Ramamrita Ranganathan;
- 1945-55 Infância da Recuperação da Informação
- 1945 Vannevar Bush escreve “As We May Think” e praticamente inventa o hipertexto e a referência (link) (Bush, 1945);
- 1958, WRU Searching Selector, primeiro catálogo computadorizado;
- 1960s, estabelecimento da área de Recuperação da Informação, conceituação de Coleção de Texto, Revocação, Precisão, etc...
- 1963, Ted Nelson inventa o termo Hipertexto e propõe um sistema similar ao WWW;
- 1972, início da DARPANET;
- 1991, criação do WWW;
- 1993, criação do primeiro *crawler*, World Wide Web Wanderer, para medir o tamanho da Web, por Matthew Gray, ainda usado nesse ano para criar um índice da Web, o Wandex;
- 1994, lançado o WebCrawler, especificamente para indexar o WWW pelo texto completo das páginas, por Brian Pinkerton;
- 1998 Google, o lançamento da Google é um marco na evolução da capacidade de busca. A empresa, em 2022, domina o mercado, tendo um algoritmo complexo capaz de lidar com uma quantidade enorme de documentos rapidamente, filtrando efeitos que tentam se aproveitar do comportamento do algoritmo e com customizações criadas a partir do perfil do usuário, ao ponto de ser difícil imaginar ser possível alcançar resultados de qualidade semelhante.
- 2023, a introdução no mercado dos Modelos de Linguagem Grandes (*Large Language Models - LLM*) traz oportunidades de encontrar informação na forma conversacional, mudando novamente o mercado de buscas.

8.2 Tarefas da Recuperação da Informação

A Recuperação da Informação considera que o usuário, em busca de informações, pode estar interessado em três classes de tarefas:

- a **busca ad-hoc**, que é busca feita em função de uma necessidade de informação imediata, onde se considera que a consulta pode variar mas a coleção é fixa;
- a **filtragem**, que é a seleção de documentos em um fluxo de documentos em função de uma necessidade de informação permanente, resultando em uma consulta fixa ao longo do tempo;
- a **navegação (browsing)**, ou folheamento, onde o usuário investiga documentos um a um, por algum caminho entre os documentos, de maneira a buscar a informação.

Neste livro a questão de como é feita a navegação ou leitura dos documentos não é tratada, mas basicamente ela depende de haver estruturas internas ou externas nos documentos (organização e links). Hoje em dia ela é uma atividade normal, com a web e seus links, porém existem algumas aplicações interessantes, como os links automáticos gerados no código de programas guardados no GitHub, que permite navegar, por exemplo, de uma chamada de função para a definição da função.

A filtragem, apesar de ser um problema bem diferente anteriormente, com o avanço da tecnologia acaba se tornando tecnicamente um problema semelhante, se não igual, ao da busca ad-hoc. Documentos acabam sendo indexados e comparados da mesma forma.

8.2.1 A Busca Ad-Hoc

Dado um conjunto de documentos, que é chamado de *corpus*, e uma necessidade de informação de um usuário, a tarefa de recuperação, ou busca *ad-hoc*, consiste em encontrar todos os documentos relevantes à necessidade de informação, possivelmente de forma ordenada, do documento mais para o menos relevante. Isso inclui evitar recuperar documentos não relevantes.

Esta tarefa apresenta um nível de dificuldade muito alto, já que, para o problema geral de recuperar um documento dada uma necessidade de informação, tratado anteriormente, as seguintes condições se aplicam:

- O usuário pode não saber qual a sua verdadeira necessidade de informação;
- Os documentos são representados de uma forma que não indica toda a sua condição de preencher a necessidade de informação;
- O sistema não é capaz de fazer todas as operações necessárias para descobrir o que é realmente relevante para o usuário;
- A relevância é um relacionamento e não um conceito absoluto;
- A necessidade de informação, e com isso a noção de relevância de um documento, muda ao longo do uso do sistema, e
- Existem várias formas de relevância.

O Paradoxo da Busca

Se o usuário soubesse o que deseja, não precisaria de um mecanismo de busca. Necessariamente, há um estado cognitivo confuso ou incompleto para que uma busca seja necessária.

Formar uma consulta de qualidade é uma das partes mais difíceis para responder a uma necessidade de informação.

Enquanto tenta resolver sua necessidade de informação, é provável que o usuário de um sistema de recuperação da informação o usará de forma iterativa, atualmente por meio de uma interface interativa³. Essa iteração cumpre um ciclo:

1. o usuário se dá conta que possui uma necessidade de informação;
2. a partir da necessidade de informação o usuário constrói mentalmente uma consulta ideal;
3. o usuário traduz essa consulta ideal em uma consulta que ele supõe adequada ao sistema de recuperação de informações;
4. o usuário submete essa consulta ao sistema;
5. o sistema realiza sua “mágica” e devolve uma lista de documentos;
6. o usuário analisa a lista de documentos;

³No passado ele poderia enviar seu pedido via correio para uma bibliotecária, por exemplo

7. o usuário navega e lê alguns documentos, possivelmente modificando sua necessidade de informação de várias formas, como tornando-a mais sofisticada ou entendendo que documentos não o interessam, e
8. o usuário fica satisfeito ou volta ao passo 2.

Mizzaro (1998) explica que há uma “**Real Necessidade de Informação**” (**RIN**), mas que como o usuário está em um estado alterado, ou incompleto de conhecimento (Belkin, N. e Brooks, 1982), o que ele possui é uma “Percepção de Necessidade de Informação”, não necessariamente igual.

8.3 Principais problemas

Esta seção apresenta uma pequena revisão dos principais problemas da Recuperação da Informação. Alguns desses problemas são tratados detalhadamente ao longo do livro.

- Tamanho da “Coleção”
 - O tamanho da coleção sempre foi um problema na área de Recuperação de Informações, pois cada evolução dos computadores, ao mesmo tempo que trouxe um aumento na capacidade de processar os documentos, também trouxe um aumento da capacidade dos computadores de guardar documentos. Assim, se os dados iniciais era apenas títulos e resumos de pequenas coleções de documentos, eles foram evoluindo para grandes coleções, até chegar ao tamanho incontrolável da internet atual, que tudo abarca.
- Simplicidade da consulta
 - Apesar dos seres humanos precisarem de informação sofisticada, dificilmente possuem o conhecimento para produzir consultas sofisticadas, que expressem bem essa necessidade de informação. A maioria dos sistemas de uso genérico, como o Google, tem como desafio buscar a partir de palavras chaves ou de perguntas em linguagens naturais. Sistemas especializados, porém, como sistemas dedicados a busca de precedentes legais, costumam possuir linguagens mais sofisticadas que permitem declarar conjunções, negativas, etc.
- Quantidade de documentos que atendem a consulta
 - Sempre um desafio, a quantidade de documentos que atende uma consulta chegou ao número de milhões com os mecanismos de busca na internet. Escolher qual o documento mais adequado, ou colocar os documentos em uma ordem de prioridade, é um problema ainda em aberto.
- Existência de informação de baixa qualidade, desatualizada, incompleta ou intencionalmente falsa
 - Os mecanismos de busca, basicamente, tentam achar documentos que versem sobre o assunto determinado pela consulta. Um questão importante é se os documentos retornados apresentam informações corretas e atualizadas.
- Multiplicidade de línguas
 - Leitores podem ser capazes, ou não, de ler em diversas línguas, o que aumenta a utilidade do mecanismo de buscas. As línguas, porém, apresentam particularidades e dificuldades próprias.
- Múltiplas interpretações do texto, incluindo as literais e as não literais
 - Cada texto pode ser interpretado de várias formas e com várias intenções, porém as palavras do texto normalmente só são reconhecidas por seu significado quando isoladas, não por seus significados não literais
- Usos do texto além da leitura do seu conteúdo, como análise de estilo
 - Alguns usuários dos sistemas de busca não estão interessados na mensagem propriamente dita, mas na forma como ela é expressada.
- Meta-dados

- Documentos possuem meta dados, como autor, data da publicação, que podem estar ou não explícitos no documento
- Entender o usuário
 - Sendo a finalidade do mecanismo de busca resolver uma necessidade real de informação do usuário, entender o usuário, tanto cognitivamente como psicologicamente, e tanto de forma global quanto no momento exato da pesquisa, é uma tarefa que pode levar a melhorias no desempenho.
- Entender as alterações do estado do usuário ao longo de um processo de buscas de muitos passos
 - Ao fazer a busca o usuário passa por vários estados mentais, e conhecer esses estados permite também melhorar o desempenho do buscador
- Relacionamentos entre os documentos
 - Que permite entender mais sobre o significado de cada documento e quando um pode ser útil para entender o outro, ou substitui-lo.

8.4 Aboutness

Aboutness é um conceito sem uma tradução específica para o português já tendo sido traduzido para “atinência”, “tematicidade”, “sobrecidade”, “concernência” ou como sinônimo de “assunto”. É usado, ao mesmo tempo, para indicar a determinação pelo assunto, tema ou tópico de um documento, que não seja necessariamente mencionado no título ou no texto, e para indicar sobre o que é o documento, sendo uma questão antiga da Ciência da Informação, e possivelmente sua principal pergunta (Guedes, 2009). O tópico também é usado em outras áreas, como a Filosofia.

Não há, por um lado, entre os principais autores da Ciência da Informação, uma concordância absoluta com uma definição específica da palavra, porém, por outro lado, algumas ideias prevalecem, o que permite um entendimento aceitável do significado do termo. Uma delas é que “documentos possuem um assunto intrínseco, inalterável, e outro, ou outros, que podem ser atribuídos (de acordo com a conveniência do sistemas em que esteja inserido ou das necessidades e expectativas de quem o venha consultar)” (Guedes, 2009).

Essa ideia está desenvolvida nos conceitos de *aboutness* extensional e intensional de Fairthorne (1969), ou, respectivamente, de *aboutness* e significado (*meaning*) (Beghtol, 1986). É semelhante a questão tratada na Seção 2.2 sobre as várias interpretações que podem ser dadas ao documento, como discutido por Meyriat (1981).

A determinação do assunto identifica o motivo do autor ao escrever o documento, mas estará sempre subordinada aos diferentes elementos em consideração pelo leitor ao fazer a leitura. Cabe então, ao responsável por indexar um documento, criar um bom índice, ou uma boa entrada em um índice para um documento, que forneça uma boa formulação do “aboutness” de um documento (Halliday e Hasan, 1976; Hutchins, 1978), em seus vários momentos de uso. Isso cabe tanto a um humano escolhendo palavras-chave em um vocabulário controlado, quanto a uma máquina criando índices automaticamente.

P. Bruza, Song e K.-F. Wong (1999) apresenta algumas propriedades básicas da *Aboutness*, partindo do conceito de portador da informação (*Information Carrier*) (P. Bruza e T. Huibers, 1996). Por definição, um portador da informação *A* é dito *sobre* (*about*) um outro portador da informação *B* se a informação contida em *B* é válida em *A*, com a notação $A \models_a B$.

Dado um *framework* $\{IC, \rightarrow, \oplus, \perp\}$, onde *IC* é um conjunto de portadores de informação, $IC = \{A, B, C, \dots\}$, $A \rightarrow B$ indica que *A* contém a informação que *B* carrega, \oplus indica a composição

e \perp representa a preclusão (incompatibilidade de assuntos) (P. D. Bruza e T. W. C. Huibers, 1994; P. Bruza e T. Huibers, 1996; P. Bruza, Song e K.-F. Wong, 1999):

1. Reflexividade: $A \rightarrow A$;
2. Transitividade: $A \rightarrow B, B \rightarrow C \implies A \rightarrow C$
3. Assimetria: $A \rightarrow B \neq B \rightarrow A$;
4. Contenção e Composição: $A \oplus B \rightarrow A; A \oplus B \rightarrow B$
5. Absorção: $A \rightarrow B \implies A \oplus B = A$
6. Não contenção conflituosa: $A \rightarrow B \implies A \not\rightarrow B$
7. Contenção-Preclusão: $A \rightarrow B, B \perp C \implies A \perp C$

Essas sentenças podem ser lidas como: um documento é sobre si mesmo. Se um documento é sobre um segundo documento e este é sobre um terceiro, então o primeiro também é sobre o terceiro. Se um documento é sobre outro não é necessário que o segundo seja sobre o primeiro. Uma composição de documentos é sobre cada documento da composição. Se um documento é sobre outro, então não é incompatível com o outro, e finalmente se um documento é sobre um segundo e esse segundo é incompatível com um terceiro, então o primeiro documento também é incompatível com o terceiro.

Esse framework permite o desenvolvimento de um sistema axiomático e teoremas. Porém, um dos problemas atingidos foi que, dentro de certos parâmetros, *Aboutness* não é monotônica em relação a termos específicos, pois podem ser dados exemplos como “Surfe é sobre ondas” e “Surfe e internet não é sobre ondas”. Isto pode acontecer, por exemplo, porque termos tem acepções diferentes tanto isoladamente quanto em conjunto.

Os trabalhos sobre *Aboutness* são comuns em outras áreas que não a Recuperação de Informação, como a Filosofia, porém tentativas de encontrar uma Teoria da *Aboutness* (P. D. Bruza e T. W. C. Huibers, 1994; P. Bruza, Song e K.-F. Wong, 1999) que pudesse ser usada de maneira eficaz, ou seu uso em sistemas de recuperação, não parecem ter evoluído na literatura.

Apesar de a *Aboutness* de um documento, em sua concepção mais ampla, parecer ser a norma da indexação, a verdadeira questão não é se o documento é sobre algo, mas sim se interessa ao futuro leitor. Esse outro conceito é conhecido como relevância.

Em todo caso, é importante notar que toda representação de um documento busca representar sobre o que ele é, isto é, sua *aboutness*.

8.5 Relevância

A palavra **relevância** pode ser usada como um termo não definido, pois todos possuem uma ideia do que significa (Saracevic, 2017). Porém, para ser possível discuti-la, é melhor desenvolver um entendimento do que significa, e mesmo uma definição. van Rijsbergen (1979, pg. 4) deixa claro que a relevância é estabelecida por humanos, e sistemas computacionais precisam ter um modelo do que é relevância de forma a quantificá-la.

Algo é relevante caso se enquadre no que o buscador está querendo, mas não necessariamente a pergunta feita. Para uma mesma pergunta, um documento pode ser importante para uma pessoa e não ser para outra. Imagine um biólogo e um fã de automobilismo pesquisando por “Jaguar”: os documentos que consideram relevantes serão provavelmente diferentes. Além disso, um documento pode ou não ser relevante por ser conhecido ou não do usuário.

Borlund (2000) afirma que a relevância, ou não, de um documento, entre vários retornados em uma consulta a um sistema de busca de informações, pode mudar após o usuário ler um outro documento.

Por exemplo, buscando conhecer um evento histórico, digamos que a Revolução dos Cravos em Portugal, após ler um texto, um segundo texto pode não trazer mais nenhuma informação adicional, se tornando não-relevante.

Em slides de uma aula proferida na ESSIR 2003, (van Rijsbergen, 2003) cita alguns autores, sem dar a referência completa, que trazem as seguintes afirmações:

- Goffman (1969) diz que “a relevância da informação em um documento depende do que já se sabe sobre o assunto, e afeta, em seu turno, a relevância dos documentos examinados a seguir”;
- Maron⁴ diz que “apenas por um documento ser sobre o assunto buscado pelo cliente, não quer dizer que ele o julgará relevante”, e
- Borlund (2000)⁵ “a relevância, ou irrelevância, de um documento recuperado pode afetar o estado atual de conhecimento do usuário, resultando na mudança de sua necessidade de informação, o que pode levar à mudança das percepções e interpretações, pelos usuário, dos documentos recuperados a seguir”.

Já Raber (2003) cita uma definição de (van Rijsbergen, 1990): “A medida ou grau de correspondência ou utilidade existente entre um texto ou documento e uma consulta ou requisito de informação determinada por uma pessoa”, que é bastante objetiva, e chama atenção para o fato que a relevância é sempre em função de uma pessoa, que possui um estado cognitivo específico em um certo momento, e pode se referir tanto ao texto desejado como a um requisito de informação.

Um definição razoavelmente formal, mas genérica, de **relevância** foi dada por Saracevic (2017), em um artigo seminal na área: “A relevância é o **A** de um **B** existindo entre um **C** e um **D** como determinado por um **E**”, sendo que:

- **A** pode ser “medida, grau, estimativa,...”;
- **B** pode ser “correspondência, utilidade, encaixe, ...”;
- **C** pode ser “documento, informação fornecida, fato, ...”;
- **D** pode ser “consulta, pedido, requisito de informação, ...”, e
- **E** pode ser “usuário, juiz, especialista de informação, ...”

(Saracevic, 2017)

É possível perceber que a definição de Saracevic aceita, em seu formato, a definição de van Rijsbergen, o que é, de fato, intencional.

8.5.1 Atributos da relevância

Mais modernamente, Saracevic (2017) faz um sumário dos atributos da relevância:

- **Relação**, a relevância aparece expressando uma relação, frequentemente em comunicações que envolvem pessoas, informações ou objetos de informação;
- **Intenção**, a relação de relevância envolve motivação e intenções, além de objetivos, papéis e expectativas;
- **Contexto**, a intenção sempre vem em um contexto, sendo direcionada para ele, havendo um contexto interno, ligado a estados cognitivos e afetivos, e um contexto externo, direcionado para situações, tarefas ou problemas, podendo ainda envolver componentes sociais e culturais;
- **Inferência**, essa relação precisa ser avaliada, sendo criada nessa base;
- **Seleção**, a inferência pode envolver uma seleção entre fontes que competem em direção a maximização dos resultados, e a minimização do esforço de trabalhar com eles;
- **Interação**, inferência é obtida em um processo dinâmico, interativo onde a interpretação dos outros atributos pode mudar, quando o contexto muda, e

⁴Não foi possível identificar esta citação

⁵Esta publicação é provavelmente sua tese de doutorado.

- **Medição (Mensuração)**, a relevância envolve uma avaliação (graduada) da efetividade ou grau de maximização de um relação dada para uma intenção direcionada a um contexto.

(Saracevic, 2017)

Isto é, a relevância é uma relação, entre pessoas e objetos de informação, que ocorre dentro de um contexto, interno e externo, onde são estabelecidas intenções que permitem inferências, onde são atribuídas medidas, que levam a seleção entre fontes de informação por meio de interações.

Além disso, Saracevic (2017) ainda afirma que a relevância pode ou não estar conectada com a verdade⁶.

8.5.2 Manifestações de Relevância

Saracevic (2017) faz um sumário das manifestações da relevância, de acordo com vários autores:

- **Sistêmica ou Algorítmica**
 - É a relação entre a consulta e a informação (ou objetos de informação) no sistema, como recuperados ou não recuperados, por um algoritmo ou procedimento dado.
 - O critério é a efetividade comparada ao inferir a relevância.
 - Cada sistema tem seus meios de representar, organizar e comparar os objetos com a consulta.
 - A intenção é recuperar um conjunto de objetos que o sistema inferiu (construiu) como sendo relevante para uma consulta.
- **Tópica**
 - É a relação entre tópico (assunto) expresso em uma consulta e o tópico coberto pela informação (objeto de informação).
 - *Aboutness* é o critério sobre o qual a topicalidade é inferida, porém a teoria do *aboutness* não continuou a progredir.
 - Assume-se que consulta e objeto de informação podem ser associados a um tópico.
- **Cognitiva**
 - É a relação entre o estado cognitivo do usuário e a informação (objeto de informação)
 - Os critérios de inferência são: correspondência cognitiva, quantidade de informação, (informativeness) novidade, qualidade da informação, etc.
- **Situacional**
 - É a relação entre a situação/tarefa ou problema e os objetos de informação (recuperados, arquivados ou existentes).
 - Os critérios de inferência são: utilidade na tomada de decisão, propriedade da informação, redução da incerteza (ser apropriada, não ownership).
 - Pode envolver fatores sociais e culturais.
- **Afetiva**
 - É a relação entre as intenções, objetivos, emoções e motivações do usuário e os objetos de informação (recuperados, arquivados ou existentes).
 - Os critérios de inferência são: satisfação, sucesso, realização.
 - É razoável defender que todos as outras manifestações de relevância estão sujeitas a afetiva, principalmente a situacional.

No mesmo livro, Saracevic ainda reapresenta um modelo estratificado da relevância(Saracevic, 2007), adaptado na Figura 8.1, inserido no contexto da Recuperação da Informação, que considera camadas diferentes interagindo, integrando diversos pontos de vista. O modelo é dividido em duas grandes

⁶O que parece relevante em tempos de *fake-news*

partes, o usuário e o computador, que se comunicam interativamente por meio de interface (Saracevic, 2017).

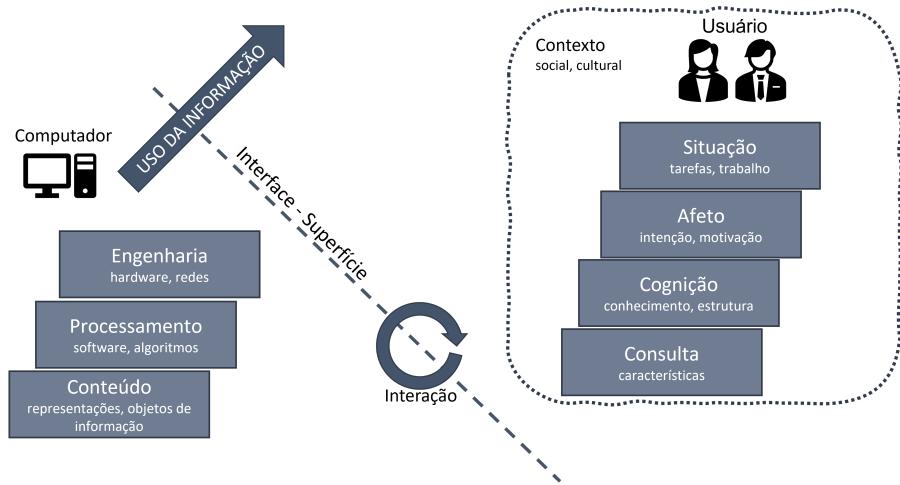


Figura 8.1: Modelo estratificado da relevância, baseado em Saracevic (2017).

Considerando os conceito de Necessidade Real de Informação (RIN) e de Necessidade Percebida de Informação (PIN), propostos por (Mizzaro, 1998), a Figura 8.2 descreve o seguinte quadro sobre a relevância: ao realizar uma tarefa o usuário tem uma Necessidade Real de Informação, para o qual ele não é capaz, dado que falta informação, de perceber precisamente, havendo então uma Necessidade Percebida de Informação. Para poder fazer a consulta, então, ele imagina uma forma de expressar essa PIN, na forma de tópicos em que estar interessado. Essa expressão é então traduzida por ele para a linguagem de consulta. É essa consulta que vai permitir ao sistema, por meio de algoritmos, determinar uma resposta, que deve atender os tópicos, tendo uma relevância cognitiva e situacional. O usuário, então, analisa a resposta, o que mudará principalmente o seu PIN, e possivelmente mesmo a sua RIN, iniciando um novo ciclo da interação.

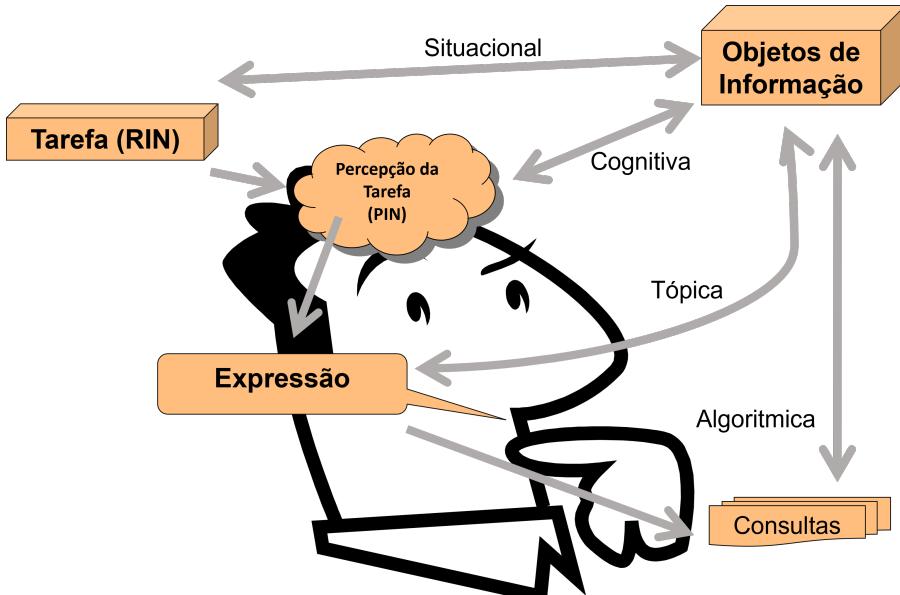


Figura 8.2: Tipos de relevância, baseado em Saracevic (2017).

8.5.3 O Espaço das Relevâncias

Mizzaro (1998) apresenta a ideia de que existem várias relevâncias e que podem ser definidos um conjunto de espaços de relevâncias, em quatro dimensões, que descreve como a relevância deve ser entendida ao longo de um processo de busca de informações:

1. Recursos de Informação, que são encontrados e avaliados, como documentos, substitutos de documentos (*surrogates*), e a informação;
2. Representação do Problema do Usuário, que são usados para fazer a busca, que possui uma necessidade real (RIN), uma necessidade percebida (PIN), a representação mental da RIN, um pedido ou solicitação (*request*), em linguagem natural, e uma consulta, em uma linguagem específica;
3. o tempo, e
4. Componentes, que são o tópico, a tarefa e o contexto.

A RIN se transforma em PIN por um processo de percepção. Já a PIN se transforma em um pedido por um processo de expressão. Finalmente, o pedido se transforma em consulta por um processo de formalização (Mizzaro, 1998). Todos esses processos estão sujeitos a erros, ruídos e incertezas.

Há uma grande dificuldade em perceber a real necessidade de informação, já que se busca algo que não se sabe, ou seja, enquanto há um vazio cognitivo, a pessoa está com uma imagem incompleta do mundo, o que é conhecido como *Anomalous State of Knowledge* (ASK) (Belkin, N. e Brooks, 1982), ou *Incomplete State of Knowledge* (ISK), *Uncertain State of Knowledge* (USK), ou seja, um Estado de Conhecimento Anômalo, Incompleto ou Incerto (Mizzaro, 1998).

Já a expressão sofre do problema do descasamento entre o vocabulário usado no pedido e os termos do documento. Pode haver ambiguidade, sinonímia, homonímia, e outras relações que interferem na facilidade de relacionar informação e expressão. Além disso, o uso de rótulos, e não expressões completas, também dificulta esse processo (Mizzaro, 1998).

Os recursos de informação e a representação do problema podem ser vistos como se associando em pares. Essas associações vão evoluindo pelo tempo, em função do usuário estar realizando a busca dentro de um, ou mais, dos 3 componentes.

Assim a relevância pode ser vista como acontencendo, e evoluindo no tempo dentro de 3 tipos de recursos, 4 tipos de representação e 7 combinações dos Componentes (Mizzaro, 1998). Essa evolução sempre começa em um tempo inicial, com a RIN, que vai evoluindo até a construção da consulta, que pode se repetir várias vezes. A análise das repostas a consulta pode alterar tanto a RIN, quanto a PIN, quanto a expressão, quanto a consulta, causando uma evolução do espaços.

1. RecInfo = { *Surrogate*, Documento, Informação }
2. Repr = { Consulta, Expressão, PIN, RIN}
3. Comp = {{Tópico} {Tarefa}, {Contexto},{Tópico, Tarefa}, {Tópico, Contexto}, {Tarefa, Contexto},{Tópico, Tarefa, Contexto}}
4. Tempo = { $t(rin_0), t(pin_0), t(r_0), t(q_0), t(q_1)$ }

Mizzaro (1998) tenta representar dimensões em diagramas que indicam como o processo de busca de uma informação pode ser visto nessas várias dimensões.

Mizzaro (1998) ainda chama a atenção que o julgamento de relevância é feito por um juiz, dentro de um tipo de relevância, mas também por um tipo de juiz, que pode ou não ser usuário do sistema de busca, como pode ter disponível um ou mais tipos de recursos e também várias formas de representar seu julgamento, tudo isso caracterizado em um momento do tempo.

8.6 Modelo conceitual da Recuperação da Informação

Fuhr (1992) propõe um modelo conceitual, ilustrado na Figura 8.3, para a Recuperação de Informação: dado um conjunto finito de documentos \underline{D} , e um conjunto finito de consultas \underline{Q} , em um certo momento, para certa pessoa, existe um conjunto finito R de julgamentos de relevância, $R : \underline{D} \times \underline{Q} \rightarrow \mathbb{R}$.

Um sistema de recuperação da informação faz esse mapeamento segundo um certo algoritmo. Porém, o sistema trabalha sobre representações D e Q dos documentos e consultas, o que exige dois mapeamentos α_D e α_Q , de documentos em representações de documentos, e de consultas em representações de consultas. Por exemplo, no modelo vetorial, documentos são representados como vetores de termos.

Genericamente falando, não são as representações que são processadas pelo algoritmo, mas sim suas descrições, implementações, por exemplo, um vetor pode ser implementado diretamente ou por uma lista encadeada, se for esparsa. Para chegar a essas descrições, D' e Q' , são necessários outros mapeamentos, β_D e β_Q , e delas é obtida um **valor de status de recuperação**, \mathfrak{R} , que é resultado da aplicação de um algoritmo nas descrições das representações dos documentos, e consultas, e logo uma aproximação, ou estimativa, do julgamento de relevância. Mais tarde, Baeza-Yates e Ribeiro-Neto (2011) removem um nível do modelo, mostrando apenas documento e representação.

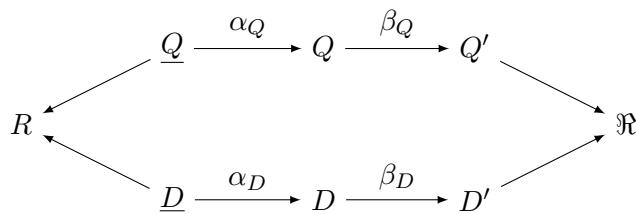


Figura 8.3: Modelo Conceitual da Recuperação da Informação, segundo (Fuhr, 1992).

8.7 Modelos dos sistemas de Recuperação da Informação

van Rijsbergen (1979) propõe um modelo mínimo para explicar como funciona um sistema de recuperação de informação, adaptado na Figura 8.4. Nesse modelo um mecanismo de busca usa duas entradas, uma coleção de documentos e uma consulta, e produz uma saída, um conjunto menor de documentos. Esse conjunto pode ser usado como *feedback* para produzir outra consulta.

Um mecanismo de busca pode ser descrito de forma abstrata em três componentes: um **mecanismo de descoberta de documentos**, que busca documentos em um espaço de documento e entrega para um **mecanismo de representação de documentos**, que gera uma representação local do documento, e um **mecanismo de busca** propriamente dito, capaz de analisar a relevância dessas representações em função de uma consulta. Usando termos mais simples, a Google descreve isso como três estágios: rastreamento, indexação e exibição dos resultados da pesquisa. A Figura 8.5 mostra uma visão bastante abstrata da arquitetura de um mecanismo de busca que segue esse modelo.

A análise do Modelo Abstrato da Figura 8.5 também leva à conclusão da necessidade de definir a representação dos documentos na base e a forma como consultas serão comparadas com esses documentos para gerar a resposta. Isso é função dos Modelos de Recuperação de Informação.

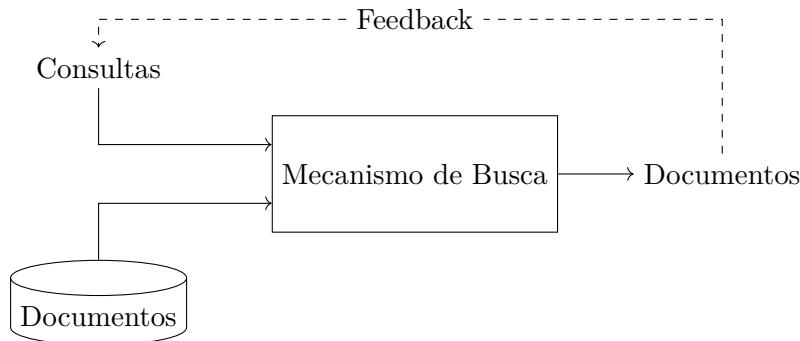


Figura 8.4: Um modelo mínimo para os sistemas de Recuperação da informação. Baseado em van Rijsbergen (1979)

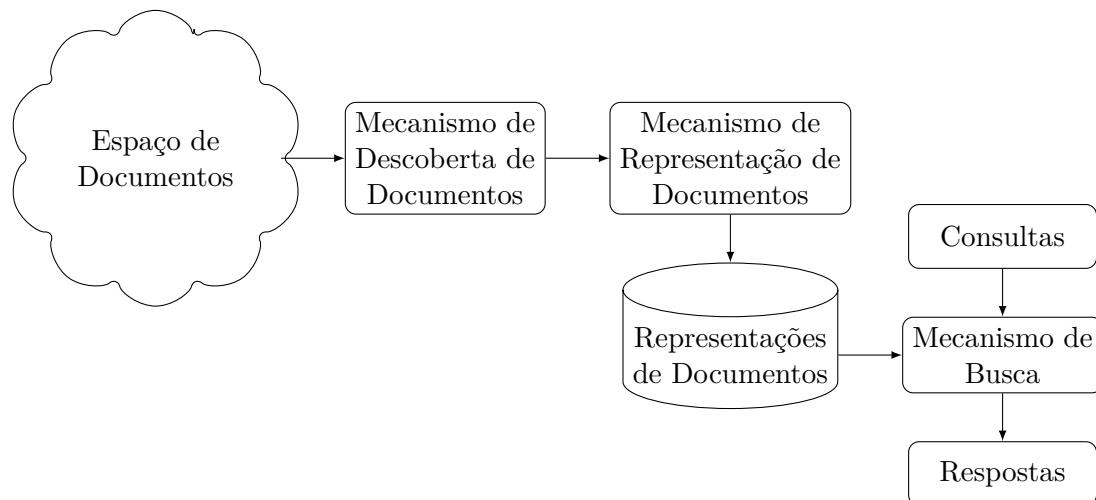


Figura 8.5: Modelo abstrato de um mecanismo de busca

As estratégias clássicas de representação usam a presença de termos nos documentos ou o peso dos termos dos documentos. Estratégias posteriores incluem formas de redução de dimensionalidade e modelos que buscam trazer informações semânticas.

Os mecanismos de busca dependem primeiramente de funções de similaridade para achar os documentos relevantes, porém é muito importante a ordenação desses documentos, o que leva a busca de funções que compensam características como o tamanho do documento.

De grande importância na ordenação dos resultados da consulta foi a introdução de conceitos relacionados a rede formada pelos documentos a partir dos trabalhos dos algoritmos HITS (Kleinberg, 1999), and PageRank (Brin e Page, 1998).

No caso mais interessante hoje em dia, o espaço de documentos é a web, o mecanismo de descoberta de documentos é um *crawler*, o mecanismo de representação é um indexador e o mecanismo de busca propriamente dito é extremamente sofisticado, incluindo processamentos como expansão de consultas, adaptação da consulta e da resposta ao perfil do usuário, etc. Outra entrada possível é uma lista inicial de documentos a serem pesquisados, de onde outros podem ser derivados. Isso resulta em uma arquitetura genérica como a da Figura 8.6

Na prática, um sistema atual, principalmente de busca na web, é mais avançado, pois inclui a construção da base, normalmente por meio da navegação em uma coleção de documentos, possivelmente a web. O Capítulo 19 descreverá estas arquiteturas em mais detalhes.

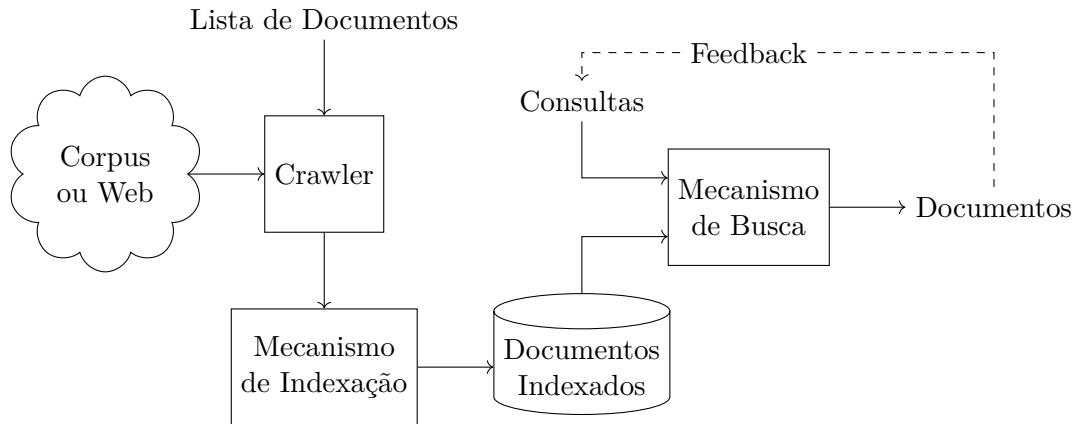


Figura 8.6: Um modelo simples para um sistema de busca.

8.8 Conceitos clássicos de Recuperação da Informação

Baeza-Yates e Ribeiro-Neto (2011) definem alguns conceitos básicos que são usados na Recuperação da Informação, especialmente nos modelos clássicos.

Um **termo índice**, ou algumas vezes apenas **termo**, é uma palavra, ou grupo de palavras consecutivas (nesse caso também chamado de **n-grama**).

Termos índices são quaisquer palavras de uma coleção, ou mais especificamente, quaisquer palavras usadas para representar um conceito ou tópico de um documento. No caso da busca de texto completo, todas as palavras do documento são candidatas a termos índices, possivelmente depois de algum processamento que inclui normalização e confluência, como transformar as formas conjugadas de verbos em seu infinitivo, ou o uso de stemmers.

Em alguns sistemas os termos índices podem ser pré-escolhidos, como em índices por assunto, e um grande esforço é feito para criar listas de termos, principalmente na criação de sistemas de indexação anteriores a busca por texto completo. Nesse caso também é comum chamá-los de **palavras-chave** ou **keywords**.

Um dos processos mais comuns nas tarefas de processamento de texto para busca ou mineração de texto é criar uma associação entre todos os termos de uma coleção de documentos e os documentos onde eles aparecem. Normalmente isso é feito na forma de uma **Matriz Termo-Dокументo**.

Essa prática também é conhecida como uma abordagem **bag-of-words**, e implica normalmente na desconsideração da ordem ou posição dos termos no documento (apesar dessa informação poder estar presente). Essa matriz também pode ser chamada de **Matriz de Incidência** e muitas vezes é armazenada na forma de uma lista invertida.

A Tabela 8.1 mostra um exemplo de um fragmento de uma matriz termo documento onde os termos são a forma singular de algumas palavras e os documentos os cinco primeiros parágrafos desta seção.

Tabela 8.1: Fragmento de uma Matriz Termo-Documento para os quatro primeiros parágrafos dessa seção.

Termos	Documentos				
	<i>p</i> ₁	<i>p</i> ₂	<i>p</i> ₃	<i>p</i> ₄	<i>p</i> ₅
<i>termo</i>	0	1	1	1	1
<i>conceito</i>	1	0	0	0	0
<i>palavra</i>	1	1	1	0	0
...

8.9 Representação do Documento

Uma das características dos modelos de recuperação da informação é a necessidade de representar documentos.

Em geral, um objeto qualquer na Computação é representado por um conjunto de atributos, características, ou *features*, o termo mais frequente no Aprendizado de Máquina. Cada atributo pode assumir um valor qualquer dentre de um domínio, isto é, um conjunto de valores possíveis.

Em relação a documentos, que no caso deste livro são usados em tarefas de Recuperação da Informação ou Aprendizado de Máquina, a dois tipos clássicos de atributos:

- Metadados, ou informações sobre os documentos, normalmente geradas por seres humanos, mas que podem também ser geradas por processos automatizados, como as informações definidas no *Dublin Core*, DoCoMo, MARC e LRM.
- Dados, que é o próprio texto contido no próprio documento, e possivelmente imagens ou sons, ou informações derivadas desses objetos, e que podem ser também usadas para determinar metadados.

A primeira abordagem que pode vir a cabeça para representar sobre o que é um texto é o uso de palavras que indiquem o significado do texto, muito usada em bibliotecas. Por exemplo, um texto jornalístico sobre o Fluminense⁷ poderia receber as palavras-chave “tricolor”, “clube” e “futebol”, mesmo que elas não fossem usadas no artigo.

Esse conceito é chamado de palavras-chave, *keywords* ou *index terms* e é amplamente adotado em bibliotecas, por exemplo em catálogos de assunto. O uso de palavras-chave tem como foco sobre o que é o documento, ou seja, sua *aboutness*.

Palavras-chave não precisam pertencer ao documento, e atualmente continuam úteis justamente por não serem encontradas nos documentos. Ao longo do tempo, a evolução de abordagens desse tipo levaram a escolha cuidadosa das palavras-chave e muitas vezes ao controle das mesmas por um glossário, taxonomia ou ontologia.

Outra forma baseada no significado do documento é a classificação dele dentro de um sistema de classificação, ou taxonomia. Segundo Tunkelang (2008), Aristóteles foi um dos primeiros a estabelecer uma taxonomia científica. Um sistema de classificação tradicional é composto de uma ou mais hierarquias de tópicos, na forma de árvores, que partem de um conceito geral, como “Áreas de Pesquisa” e avançam até detalhes como “Tabagismo”. Um exemplo é o Sistema de Classificação para a Computação de 2012 da ACM, que é uma “ontologia poli-hierárquica que pode ser usada em aplicações da Web Semântica”. Nesse sistema, o Capítulo 3 seria classificado como “Information systems→Information→retrieval→Document representation→ Data encoding and canonicalization”.

⁷O Fluminense é um time de futebol do Rio de Janeiro cuja bandeira tem três cores.

Uma forma mais moderna de classificação é a **classificação facetada** (Tunkelang, 2008), que busca descrever o documento por facetas ou dimensões diferentes, que fornecem visões ortogonais de um documento. Uma faceta pode indicar, por exemplo, o estilo literário de um livro, e a outra um assunto tratado. Para um documento, as facetas assumem valores como em “estilo=barroco, tipo=romance, assunto=traição, língua=português do Brasil”.

Com crescimento do poder computacional, sistemas informatizados foram desenvolvidos com coleções maiores e cada vez admitindo mais palavras no vocabulário, e no índice, o que tornou possível usar todas as palavras dos documentos, supondo que elas trazem o significado do mesmo, no que é chamado **full-text indexing**.

É importante chamar a atenção que todas as palavras do documento, mesmo organizadas de maneiras diferentes, podem não representar alguns significados (*aboutness*) de um documento. A viabilidade do uso de todas as palavras de um documento em um processo de busca implica em um tipo de consulta que busca as palavras que estão no documento.

Este livro trata principalmente das representações de documentos a partir de suas palavras, ou termos obtidos no pré-processamento, mas todo o material aqui apresentado pode ser usado também para n-gramas, isto é, sequência de palavras, ou representações conceituais obtidas a partir destas palavras.

Logo, a principal representação dos documentos aqui considerada é relativa à presença dos termos no documento em cada documento de um corpus. Essa representação é conhecida como **Matriz Termo Documento**. Nessa matriz, os termos estão nas linhas e os documentos estão nas colunas. Cada célula então representa uma medida de importância do termo como atributo do documento. Os modelos de recuperação de informação escolhem formas de criar e interpretar essa matriz. Em aplicações de classificação ou agrupamento, usa-se a Matriz Documento Termo.

Além disso, essa matriz pode ser usada para criar uma matriz termo-termo, que indica a similaridade ou distância entre os termos, ou uma matriz documento-documento, que indica também a similaridade ou distância entre documentos.

A matriz termo documento é uma abstração, podendo ser usada diretamente em um mecanismo de recuperação da informação ou de aprendizado de máquina, mas também pode existir junto com informações adicionais, como as posições onde as palavras aparecem nos documentos.

Outra característica dessa matriz é ser esparsa, isto é, muitos termos não aparecem em muitos documentos. (Nguyen, 2013) reporta uma matriz com 1.870 documentos e 20.176 termos com apenas 80.702 termos não zero, em um total de 37.729.120 células. Uma base de tweets, por exemplo, é extremamente esparsa, já que o número de palavras de cada documento é muito pequeno. Isso leva à necessidade de representações adequadas, como a lista invertida apresentada anteriormente, ou ainda estruturas mais sofisticadas. Normalmente em inglês ou português o número de termos é na ordem de 60 a 80 mil, mas os maiores dicionários contêm em torno de 300 mil palavras, e isso não inclui a conjugação dos verbos, as variantes em número e gênero, etc.

Outra possibilidade para tratar o fato que a matriz é esparsa, principalmente em certos algoritmos que não aceitam muitos atributos, é aplicar processos de redução de dimensionalidade, de forma a extrair dessa matriz uma representação mais compacta e a que muitos autores atribuem a capacidade de representar o verdadeiro significado do documento.

Além disso, essa matriz muitas vezes é modificada para tratar problemas como a similaridade entre termos, a sinonímia e outras relações linguísticas entre os significados dos termos, e a não-independência entre a presença dos termos no mesmo documento.

De qualquer forma, essa matriz é de suma importância na área. Muitas vezes o seu uso é chamado de Bag of Words (BoW), porque ela transforma o documento em um conjunto não ordenado de palavras, com apenas alguma contabilização da importância da palavras. A metáfora é clara, as palavras são jogadas em uma sacola e todas misturadas, mas ao mesmo tempo um *bag* é uma estrutura matemática que pode ser descrita informalmente como um conjunto que aceita a repetição de elementos.

Os capítulos a seguir voltam a tratar da matriz-termo documentos, principalmente nos modelos clássicos de recuperação da informação.

8.9.1 O mundo externo também representa o documento

Durante algum tempo, o *full-text indexing* foi o paradigma computacional mais sofisticado, e independente dos seres humanos, para os sistemas de busca e recuperação. Isso se refletiu nos primeiros mecanismos de busca na Web, ainda bem no início da web, até que a quantidade de documentos relevantes se tornou tão grande para as consultas típicas dos usuários, com poucas palavras, que a resposta dos mecanismos de busca se tornou ineficiente. Listas de centenas de documentos escondiam no seu meio os documentos verdadeiramente relevantes.

A Web, porém, fornecia, de forma estática, uma nova informação além do texto, que são os links entre documentos, além de outras informações que podem ser usadas de forma dinâmica, como o uso dos documentos, e a semelhança entre os usuários. Além disso, passou a ser possível a utilização de uma quantidade massiva de dados, ricas em informações sobre os conteúdos e sobre os usuários. Desta forma, a área de Recuperação de Informação se tornou muita mais rica em tipos de algoritmos, como os algoritmos que se baseiam na estrutura da rede, como o HITS e o PageRank, algoritmos que se baseiam na navegação, como o BrowseRank, e muitas outras abordagens.

8.10 Os modelos de Recuperação da Informação

Costumou-se chamar cada abordagem para a Busca e Recuperação de Documentos (*Information Retrieval*) de um “modelo”. Segundo Baeza-Yates e Ribeiro-Neto, 2011, “um modelo de Recuperação da Informação é uma quádrupla $[\mathbf{D}, \mathbf{Q}, \mathcal{F}, R(q_i, d_j)]$ ” onde:

1. \mathbf{D} é um conjunto de visões ou representações lógicas de documentos de uma coleção;
2. \mathbf{Q} é um conjunto de visões ou representações das necessidades de informação do usuário, chamadas consultas;
3. \mathcal{F} é um arcabouço para modelar representações de documentos, consultas e seus relacionamentos, e
4. $R(q_i, d_j)$ é uma função de ordenação (*ranking*) que associa um número real a um representação de consulta $q_i \in \mathbf{Q}$ e um documento $d_j \in \mathbf{D}$, que define a ordem dos documentos em relação a q_i .

(Baeza-Yates e Ribeiro-Neto, 2011)

Essa definição traz a luz as principais características de um sistema de recuperação da informação:

- É preciso representar tanto os documentos quanto as consultas;
- É preciso modelar os relacionamentos entre documentos e consultas, e
- É preciso ordenar a resposta.

As principais formas de representar os documentos são feitas em função dos termos encontrados no mesmo, como os modelos booleano e vetorial, sendo uma alternativa comum a criação de representações que modelam conceitos, como o uso da Indexação por Semântica Latente (LSI). Outras formas de representar falam da probabilidade do texto ser gerado.

Já as principais formas de relacionar consultas e documentos e ordenar a resposta de uma consulta estão ligadas a questões de similaridade, como no modelo vetorial, ou de consultas a partir de conectivos lógicos, como no modelo booleano. A alternativa principal é calcular a probabilidade de relevância, ou a probabilidade de geração do documento pela consulta, ou vice-versa.

8.11 Taxonomia de modelos de Recuperação da Informação

Baeza-Yates e Ribeiro-Neto (2011) também fizeram uma taxonomia para esses modelos, apresentada na Figura 8.7 que é bastante aceita, mas que vai além do texto. Nessa taxonomia foi incluído um grupo de modelos Baseado em Sinais, baseado no trabalho com *FFT* e *Wavelets* deste autor.

Trabalhando apenas com a questão da busca e recuperação de documentos com texto, é possível propor uma segunda classificação, que considera a forma como o texto é visto. Assim, uma classe seria a dos modelos do tipo *bag of words*, que consideram os documentos como conjuntos de termos, mesmo que sejam n-gramas, e que inclui praticamente todos os modelos baseados em Teoria dos Conjuntos e Algebraicos, e alguns probabilísticos. A segunda classe considera a sequência de palavras, e inclui os modelos de linguagem de vários tipos. Por muito tempo os modelos de *bag of words* dominaram a área de mineração de texto, como forma de gerar atributos para algoritmos de classificação ou agrupamento. Com o aparecimento dos *embeddings* não contextuais, como o *Word2Vec*, e os algoritmos de Redes Neurais Profundas, como o BERT e o GPT, que são modelos de linguagem e fornecem *embeddings* contextuais, o interessado da área migrou para representações desse tipo.

Este livro não trata dos mecanismos dedicados para documentos semi-estruturados, pois não têm tido impacto real tanto na academia quanto no mercado, ou de multimídia. Uma forma tradicional de tratar documentos XML sem grandes sofisticações, por exemplo, é simplesmente indexar cada nó como um documento diferente.

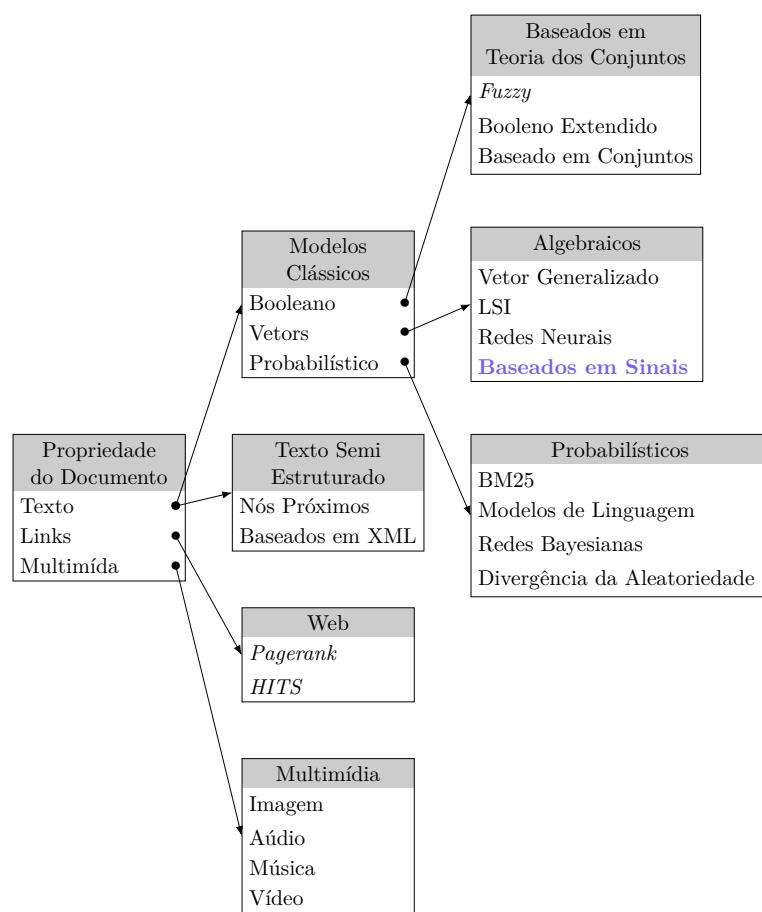


Figura 8.7: Taxonomia da Recuperação da Informação. Fonte: (Baeza-Yates e Ribeiro-Neto, 2011).

CAPÍTULO 9

MEDIDAS DE AVALIAÇÃO

Mecanismos de busca ou de aprendizado de máquina podem ser avaliados de várias formas. Em muitas áreas de conhecimento onde o resultado esperado dos algoritmos ou heurísticas é exato há um interesse na complexidade do algoritmo, ou na velocidade de execução em certa configuração de computadores. Essas diferentes formas de desempenho são calculadas para uma resposta correta e precisa.

Os algoritmos de busca, recuperação e mineração de texto, porém, têm como característica principal não retornarem respostas precisas, principalmente quando avaliados por seres humanos diferentes e em contextos onde a relevância não é clara. Logo, apesar do desempenho continuar importante, as medidas mais interessantes nesse caso buscam avaliar a qualidade da resposta encontrada em função de uma expectativa específica. A princípio, o que se busca é uma avaliação que permita comparar a satisfação do usuário com as respostas trazidas por diferentes algoritmos ou mecanismos de busca.

Essa satisfação não é fácil medir, principalmente em experimentos automatizados, e com o interesse de ter um grande número de dados para garantir um certo grau de generalização. Em busca de uma medida dessa satisfação, então, foi estabelecido o conceito de relevância, que, em suas manifestações descritas na Subseção 8.5.2, é um proxy adequado para a satisfação. Mesmo assim, há muitas questões relacionadas aos tipos de relevância que podem ser medidas de forma repetitiva. Isso levou a adoção de medidas específicas.

O precursor desses estudos comparativos foi Cyril Cleverdon, um dos “pais” da Recuperação da Informação, nos Experimentos Cranfield (Cleverdon, 1960, 1967). Cinco anos antes, porém, Kent et al. (1955) já definiam as medidas básicas de avaliação de mecanismos de busca, precisão¹ e revocação, em “Machine literature searching VIII. Operational criteria for designing information retrieval systems”.

9.1 Modelo clássico de avaliação

A forma clássica de avaliar um mecanismo de busca é realizar um *benchmark* a partir de três tipos de informação: uma coleção de documentos, uma coleção de consultas, e uma avaliação de cada documento da coleção como relevante ou não-relevante para cada consulta.

¹Kent et al. (1955) chama de *pertinence factor*.

Para coleções de tamanhos razoáveis, é comum que essa avaliação seja feita por vários especialistas, e que haja alguma discordância entre eles (Baeza-Yates e Ribeiro-Neto, 2011; Cleverdon, 1960). Para atender os métodos de avaliação, cada julgamento pode ser binário, isto é, apenas dizer se o documento é relevante ou não, ou usar alguma escala, como dar um valor no conjunto 0, 1, 2. Esse trabalho, porém, não é fácil, porque é necessário julgar toda a coleção em relação a cada consulta, e ao mesmo tempo é interessante que a coleção seja muito grande. Soluções alternativas incluem, em uma comparação de mecanismos, considerar como relevantes a uma consulta os documentos que sejam retornados por todos ou por uma quantidade específica de mecanismos.

Existem várias coleções usadas como padrão para experimentos de busca e recuperação. Duas coleções muito usadas são a NIST-GOV2, com 25 milhões de documentos, e a Reuters-RCV1, com mais de 800 mil documentos.

9.2 Medidas clássicas de avaliação

As medidas clássicas de avaliação de mecanismos são a precisão e a revocação, introduzidas por Kent et al. (1955). No aprendizado de máquina é também comum usar a acurácia. Elas podem ser definidas da seguinte forma: seja T uma tarefa realizada sobre um conjunto C para a qual se espera como resposta um conjunto A , onde $A \subset C$, e seja H_T uma heurística que busca implementar a tarefa T e responde com um conjunto B , onde $B \subset C$, então é possível definir, para H_T as seguintes medidas de avaliação:

- a **precisão** P , representando a proporção de acertos da heurística (Equação 9.1) em relação ao que foi obtido por ela;
- a **revocação**, do inglês *recall*², R , ou **sensibilidade**, representando a proporção do que devia ter sido obtido pela heurística (Equação 9.2) pelo que foi obtido por ela;
- a **acurácia** A_c , representando a quantidade de acertos entre o que foi encontrado e também entre o que não devia ser encontrado e não foi (Equação 9.3).

$$P(H_T) = \frac{|A \cap B|}{|B|} \quad (9.1)$$

$$R(H_T) = \frac{|A \cap B|}{|A|} \quad (9.2)$$

$$A_c(H_T) = \frac{|A \cap B| + |C - A - B|}{|C|} \quad (9.3)$$

O desempenho de heurística de busca ou classificação também podem ser entendido por meio da **matriz de confusão** (Tabela 9.1).

Levando em conta a notação da Tabela 9.1, as fórmulas podem ser reescritas como:

$$P(H_T) = \frac{VP}{VP + FP} \quad (9.4)$$

²A palavra revocação foi adotada como a tradução padrão de *recall* na área de Recuperação da Informação no Brasil e significa “chamar de novo, chamar do passado”, possivelmente relembrar, porém também significa “tornar nulo”, sendo sinônimo de revogar, que é exatamente trazer as coisas a uma situação passada. Não é muita palavra de fácil reconhecimento ou totalmente correta como tradução, porém palavras que poderiam ser mais adequadas como abrangência, cobertura ou até relembrança têm outros significados específicos ou não foram aceitas.

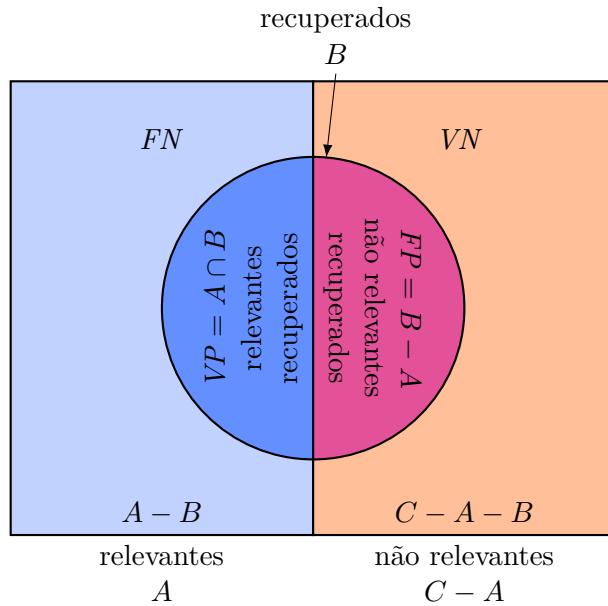


Figura 9.1: Sendo C o conjunto com todos os elementos, A o conjunto dos elementos relevantes, e B o conjunto dos elementos recuperados. Então $A \cap B$ é o conjunto dos verdadeiros positivos, $A - B$ é o conjunto dos falsos negativos, $B - A$ é o conjunto dos falsos positivos e $C - A - B$ é o conjunto dos verdadeiros negativos.

Tabela 9.1: Matriz de Confusão

	Recuperados (B)	Não-Recuperados ($C - B$)
Deviam ser recuperados (A)	Verdadeiros Positivos (VP) $A \cap B$	Falso Negativo (FN) $A - B$
Não Deviam ser recuperados ($C - A$)	Falsos Positivos (FP) $B - A$	Verdadeiros Negativos (VN) $C - A - B$

$$R(H_T) = \frac{VP}{VP + FN} \tag{9.5}$$

$$A_c(H_T) = \frac{VP + VN}{VP + VN + FP + FN} \tag{9.6}$$

Se $B = A$ então $P = R = A_c = 1$. Porém, se $B = C \implies R = 1$, porém $A_c = \frac{A}{C}$ e $P = \frac{A}{C}$, mas esse resultado não tem interesse nenhum. Já $P = 1$ pode acontecer quando B é pequeno, e esse resultado pode ser de algum interesse.

É importante notar que essas medidas não incluem a ordem de recuperação dos documentos, mas sim apenas o conjunto recuperado.

Estamos usando as medidas certas?

Su (1992) encontrou na literatura da época 5 critérios para avaliar sistemas de recuperação: relevância, eficiência, utilidade, satisfação do usuário e sucesso. Eles se dividem em 20 medidas, incluindo precisão, tempo da sessão de busca, custo real de busca, valor do resultado da busca em dólares, e chegando até o julgamento do usuário sobre o sistema como um todo. Ela não avaliou revocação por causa da dificuldade operacional de avaliar quantos documentos relevantes existem em uma coleção. Sua lista é bem diferente das medidas mais usadas hoje em dia, apresentadas neste texto e encontradas na maioria dos livros e também usadas na maioria dos artigos sobre o tema, já que o foco mudou da avaliação de sistemas por diferentes visões do ser humano para medidas ligadas ao conceito de relevância algorítmica. Sua conclusão é que a melhor medida única do sucesso de um mecanismo de Recuperação da Informação seria o “Valor dos resultados da busca como um todo” e que precisão, uma medida fortemente usada hoje em dia, não é um bom indicador de desempenho da Recuperação da Informação. Além disso, ela detectou que a revocação é mais importante que a precisão para os usuários.

9.2.1 Problemas com a acurácia na recuperação da informação

Se o conjunto B é de tamanho muito diferente do conjunto $C - B$, o que é o caso mais comum na tarefa de recuperação da informação, então a acurácia não é uma boa medida, o que exemplificado na matriz de confusão.

Tabela 9.2: Matriz de Confusão Desbalanceada

	Recuperados	Não-Recuperados
Deviam ser recuperados	$VP = 1$	$FN = 9$
Não Deviam ser recuperados	$FP = 0$	$VN = 990$

$$A_c(H_T) = \frac{1 + 990}{1000} = 0.991 \quad (9.7)$$

Como a revocação é baixa, 10%, mesmo com a precisão alta, esse resultado é insuficiente na maioria dos casos. Mesmo assim a acurácia é alta. Por isso a acurácia não é usada na recuperação da informação, e apesar de ter valor nas tarefas de classificação e agrupamento deve ser sempre analisada a partir da matriz de confusão.

9.2.2 Outras medidas clássicas

A Tabela 9.3 apresenta uma lista de medidas que são usadas a partir dos valores obtidos para a matriz de confusão. Nela cada documento recuperado pelo mecanismo de busca é classificado em uma de 4 situações: um documento que devia ser recuperado e foi recuperada (**verdadeiro positivo** ou **VP**), um documento que não devia ser recuperado e não foi recuperado (**verdadeiro negativo** ou **VN**), um documento que foi recuperado e não devia ser recuperado (**falso positivo** ou **FP**), e um documento que não foi recuperado mas devia ter sido recuperado (**falso negativo** ou **VN**).

Tabela 9.3: Algumas medidas muito usadas, seus sinônimos e fórmulas.

Medida	Acrônimo	Fórmula
Positivos	<i>Pos</i>	$VP + FN$
Negativos	<i>Neg</i>	$VN + FP$
População Total	-	$Pos + Neg$
Verdadeiro Positivo	<i>VP</i> ou <i>TP</i>	
Falso Negativo		
Erro Tipo II		<i>FN</i>
Perda		
Falso Positivo		
Erro Tipo I		<i>FP</i>
Alarme Falso		
Verdadeiro Negativo	<i>VN</i> ou <i>TN</i>	
Prevalência		$\frac{Pos}{Pos+Neg}$
Acurácia	<i>A_c</i>	$\frac{VP+VN}{VP+VN+FP+FN}$
Taxa de Verdadeiro Positivos		
Revocação		
<i>Recall</i>		
<i>Sensitivity</i>	<i>R</i>	$\frac{VP}{VP+FN}$
Sensibilidade		
Probabilidade de Detecção		
Precisão	<i>P</i>	$\frac{VP}{VP+FP}$
Valor Preditivo Positivo		
Especificidade		
Seletividade		
Taxa de Verdadeiro Negativo		$\frac{VN}{VN+FP}$
Discriminação		
<i>Fallout</i>		$1 - \frac{VN}{VN+FP}$

9.3 Fatores que governam a precisão e revocação

Não é possível discutir os mecanismos de busca sem considerar a parte onde acontece a indexação. A forma como a indexação possui algumas características. Soergel (1994) discutindo sistemas com vocabulários controlados, descreve exaustividade, especificidade, correção e consistência e seus efeitos na busca.

A **exaustividade** descreve a extensão com que determinado documento é indexado. A ideia é que um documento sendo indexado é relevante, em graus variados, para muitos conceitos. Quanto mais conceitos forem cobertos, mais exaustiva é a indexação (Soergel, 1994).

A exaustividade depende do ponto de vista, ou seja, a existência de descritores que cubram o conceito, e a importância, que é o limite exigido para que o conceito seja descrito. Em sistemas com vocabulário restrito essas questões são muito importantes, porém em sistemas *full-text* também existem,

dependendo de estratégias diversas como o uso de *stopwords*, n-gramas, tesaurus, LSA e outras técnicas que diminuem ou aumentam a exaustividade (Soergel, 1994).

A **especificidade** descreve o quanto genérica é a descrição dos documentos na indexação. Novamente, quando a indexação usa um vocabulário restrito isso é mais fácil de determinar ou controlar, enquanto em sistemas *full-text* depende das propriedades dos algoritmos usados (Soergel, 1994).

Como exemplo, ao indexar um conjunto de artigos científicos, a exaustividade é aumentada se mais áreas da Ciência são cobertas, como Medicina, Engenharia, etc., já a especificidade depende de detalhar os conceitos, como Cirurgia Cardíaca e Cálculo de Estruturas como detalhamento das duas áreas anteriores.

A exaustividade e a especificidade da indexação afetam diretamente as propriedades dos algoritmos de busca. Quanto maior a exaustividade, maior será a revocação, ou seja, quanto maior for a capacidade de indexar conceitos diferentes, mais o mecanismo de busca será capaz de encontrar documentos. Porém, isso também tem o efeito colateral de diminuir a precisão. Por outro lado, quanto maior for a especificidade, maior será a precisão, novamente com um efeito colateral: diminuir a revocação.

9.4 Relação entre precisão e revocação

Tendo em vista o discutido sobre exaustividade e especificidade, e usando um pouco de senso comum, é razoável entender que quanto mais preciso um mecanismo de busca deseja ser, mais documentos que deviam ser encontrados correm o risco de não ser encontrados, e quanto mais ele tenta encontrar todos os documentos possíveis, mais documentos que não deveriam ser encontrados são encontrados.

Ao mesmo tempo, para cada consulta que retorna uma lista ordenada de documentos de acordo com uma regra qualquer que prevê sua relevância, é possível escolher um limite que define até que ponto da lista a resposta será considerada. Ou, entendendo de outra maneira, cada consulta pode ser vista como a ordenação de todos os documentos da coleção de acordo com sua relevância e, novamente, pode se aplicar um limite de aceitação para escolher a resposta.

Em ambos os casos, cada resposta a uma consulta implica em uma lista ordenada onde a revocação é crescente, já que ao longo da lista cada vez mais documentos relevantes são encontrados, porém a precisão pode variar.

Assim, para cada consulta é possível construir uma curva de precisão por revocação. A partir de alguma manipulação matemática, todas essas curvas poderiam ser resumidas em uma curva de precisão por revocação para um mecanismo de busca, como a da Figura 9.2.

A partir dessa curva, a qual até esse ponto é uma construção teórica, seria possível determinar se um mecanismo de busca, ou uma configuração específica do mesmo mecanismo de busca, é melhor ou pior que o outro. Por exemplo, se a curva de um primeiro mecanismo apresentasse sempre uma precisão melhor do que a de um segundo, seria um sinal de que o primeiro é melhor do que o segundo.

Na próxima seção será apresentado um gráfico específico que mostra essa relação e permite a comparação de mecanismos distintos.

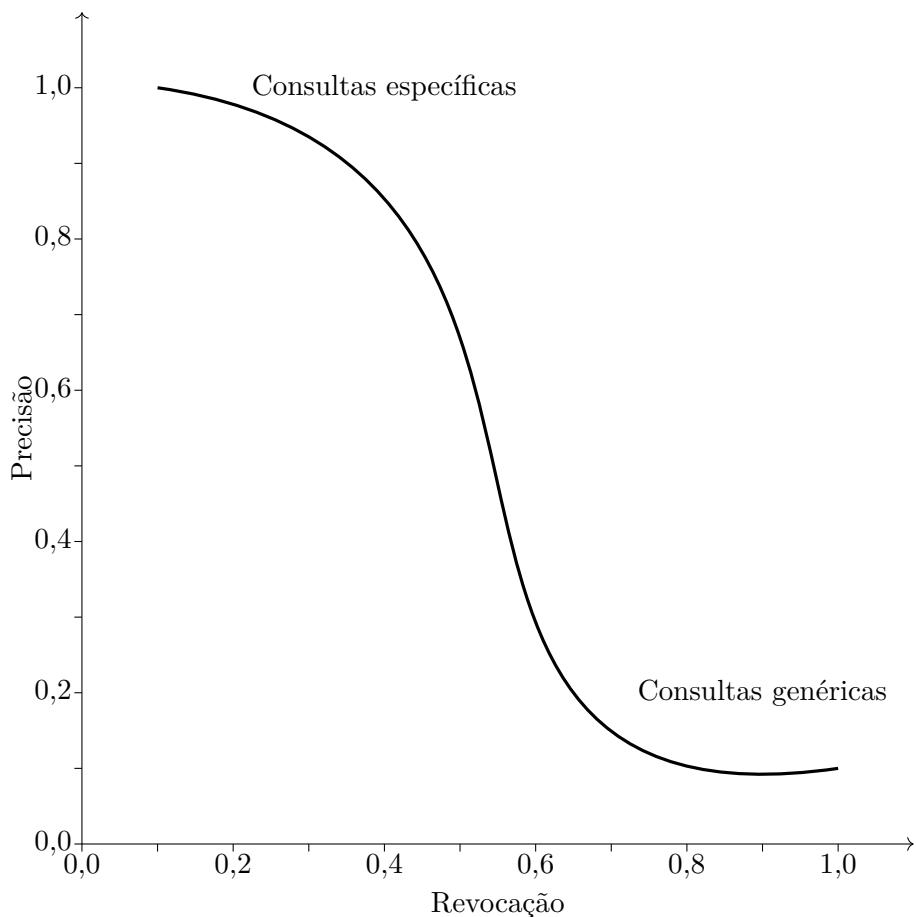


Figura 9.2: Visão abstrata de uma curva média de precisão por revocação média. Fonte:(Salton e McGill, 1983)

9.5 Gráfico de precisão em 11 níveis de revocação

Outra forma de comparar heurísticas é analisar como a precisão varia de acordo com a revocação. A ideia aqui é que um algoritmo de busca retorna uma lista ordenada de documentos. A partir do primeiro, a cada documento que pegamos, ele é relevante ou não. Isso faz com que a cada documento adicional haja uma possível mudança nos valores da precisão e revocação.

Por exemplo, dado uma consulta C_1 com a resposta esperada, não-ordenada,

$$\{d_1, d_2, d_3, d_4, d_5\}.$$

Suponha que um mecanismo de busca M retorna, para essa consulta, a lista ordenada

$$\langle \mathbf{d}_1, d_6, \mathbf{d}_5, d_7, \mathbf{d}_2, d_8, \mathbf{d}_4, d_9, \mathbf{d}_3 \rangle,$$

onde o negrito indica os documentos esperados. Então é possível construir a Tabela 9.4 que indica a revocação e a precisão a cada item obtido na sequência e, a partir desses dados, construir um gráfico de precisão para revocação para uma consulta (C_1), apresentado na Figura 9.3. Observe que em um gráfico desse tipo, a revocação nunca diminui, pois ela mede a proporção de documentos encontrados entre os desejados, mas a precisão varia, tendo uma tendência a cair em relação a precisão inicial. Isso

acontece por que, em geral, se espera que para encontrar todos os documentos seja necessário ampliar muito as condições de aceitação para colocar documentos em um resultado de consulta, gerando ruído.

Tabela 9.4: Evolução das medidas de revocação e precisão a cada item analisado da resposta à consulta.

Posição	Documento	R	P
1	d_1	20%	100%
2	d_2	20%	50%
3	d_5	40%	67%
4	d_7	40%	50%
5	d_2	60%	60%
6	d_8	60%	50%
7	d_4	80%	57%
8	d_9	80%	50%
9	d_3	100%	50%

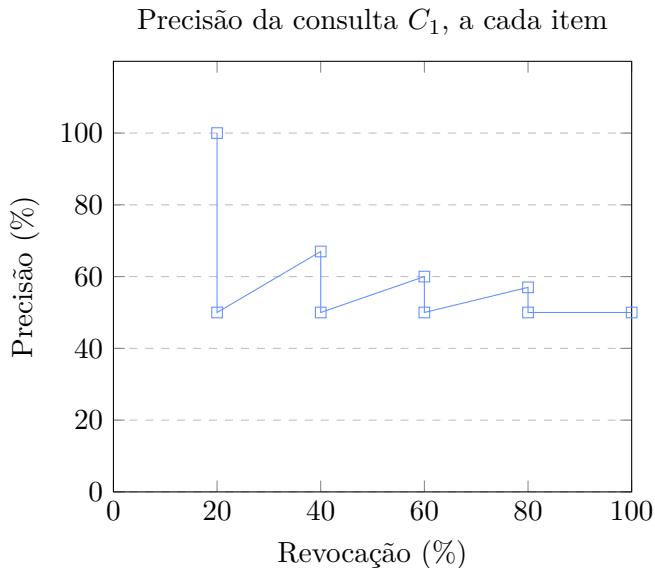


Figura 9.3: Gráfico de precisão por revocação para uma consulta (C_1).

A Tabela 9.4 e o Figura 9.3 indicam o desempenho do mecanismo M para uma consulta apenas (C_1). Porém, para avaliar um mecanismo de busca são usadas várias consultas. Além disso, elas mostram resultados onde não há mudança da revocação, o que não é muito interessante e gera um efeito de serra. Para eliminar esse efeito, o próximo gráfico marcará apenas os pontos onde um documento relevante é recuperado.

Suponha que uma consulta C_2 esperasse como respostas

$$\{d_1, d_2, d_3\}$$

e o mecanismo M retornasse como resposta

$$\langle \mathbf{d}_1, d_9, \mathbf{d}_3 \rangle .$$

O gráfico com as duas consultas, C_1 e C_2 é o da Figura 9.4. Este um **gráfico de precisão por revocação**, no caso, para todas as consultas que foram feitas.

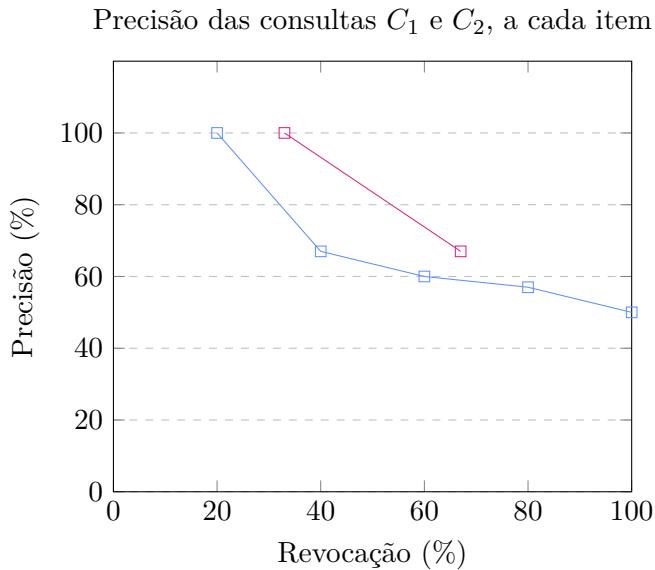


Figura 9.4: Gráfico de precisão por revocação para as consultas C_1 e C_2 .

A pergunta que essas imagens geram é: como fazer um gráfico que represente de forma geral o desempenho do mecanismo de consulta, de forma que mecanismos possam ser comparados pelos seus gráficos? Ou seja, como representar todas as consultas em uma só curva, para representar o desempenho médio?

Olhando para o gráfico, é possível ver a dificuldade de fazer uma média. Isso acontece porque os pontos de cada curva não têm abscissas comuns. Em geral, com muitas curvas, haverá alguma coincidência, mas mesmo assim, como fazer as médias?

A solução que acabou se tornando prática da área é o **gráfico de precisão em 11 pontos de revocação**. Para construir esse gráfico, partimos da definição dos valores de revocação para os quais são desejadas medidas de precisão: de 0% de revocação até 100%, com passos de 10%.

Logo, a questão é agora é como obter esses pontos, já que cada consulta tem um comportamento diferente? O que se faz é uma interpolação dos pontos do gráfico obtidos na consulta para os 11 pontos de revocação. Essa interpolação usa a regra:

$$p_i(r) = \max_{r_i \geq r} p(r_i) \quad (9.8)$$

A ideia básica é “olhar a direita” no gráfico, ou abaixo na tabela. Cada um dos 11 pontos, de abscissas variando de 10 em 10, tem como ordenada o maior valor a sua direita, isto é a maior ordenada para todas as abscissas iguais ou maiores que dele. Isso para cada curva isoladamente, isso é, para cada consulta. Caso uma revocação não tenha sido alcançada, como no caso de nossa consulta C_2 , a precisão é considerada 0.

A Tabela 9.5 mostra uma forma de construção do gráfico de precisão por 11 pontos de revocação. Nela são colocados todos os pontos de revocação encontrados por todas as curvas, e ainda os 11 pontos de revocação desejados. As precisões então são colocadas para cada curva isolada, $p(C_1)$ e $p(C_2)$, e a curva interpolada é calculada simplesmente pegando o maior valor na mesma linha ou abaixo dela, $p_i(C_1)$ e $p_i(C_2)$. Finalmente, o valor médio nesses pontos permite a geração da curva para o mecanismo de consulta. A Figura 9.5 mostra as três curvas, sendo que a curva resultante é o **gráfico de precisão por 11 pontos de revocação**.

Tabela 9.5: Tabela para construção do gráfico de 11 pontos

R (%)	$p(C_1)$	$p(C_2)$	$p_i(C_1)$	$p_i(C_1)$	\bar{p}_i
0	-	-	100	100	100
10	-	-	100	100	100
20	100	-	100	100	100
30	-	-	67	100	84
33	-	100	-	-	-
40	67	-	67	67	67
50	-	-	60	67	64
60	60	-	60	67	64
67	-	67	-	-	-
70	-	-	57	0	29
80	57	-	57	0	29
90	-	-	56	0	28
100	57	-	56	0	28

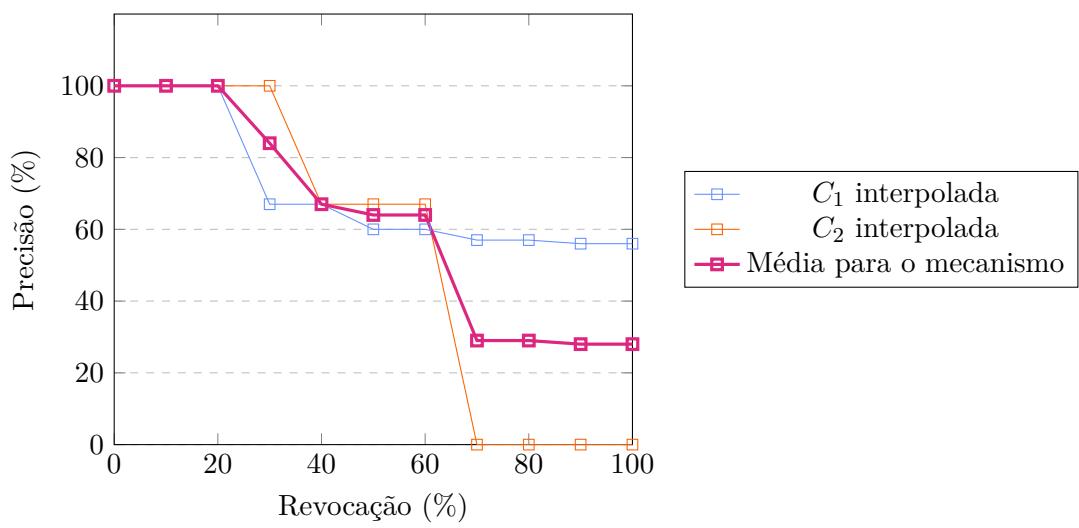


Figura 9.5: Gráfico de precisão por 11 pontos de revocação para as duas consultas e para o mecanismo.

9.6 Comparando com apenas um valor

Apesar do padrão *de facto* dos artigos ser o uso de precisão e revocação, eles dificultam a comparação de dois mecanismos, já que implicam em duas dimensões com uma forte correlação e alterar o desempenho do mecanismo para uma medida pode afetar a outra negativamente.

Assim, foram propostas algumas medidas com apenas uma dimensão apenas e que permitem a comparação direta de dois mecanismos ou duas versões do mesmo mecanismo.

Uma medida muito fácil de calcular é conhecida como ***precision at 10***, ou **precisão em 10**, indicada como **P@10**, ou variantes. Esse medida simplesmente indica a precisão em um ponto fixo, como as 10 primeiras repostas. Sua variante mais comum é **P@5**, que usa as 5 primeiras.

Outra medida fácil é a **R-Precision**, ou **Precisão R**. A ideia aqui é definir uma posição na lista ordenada de resposta com R documentos relevantes e calcular a precisão nesse ponto. Usando as consultas C_1 e C_2 da Seção 9.5 podemos calcular, por exemplo, R_2 . Em C_1 o segundo documento aparece na posição 3, fazendo que $R_2 = 0.67$. Coincidemente, para C_2 , o segundo documento relevante também é encontrado na posição 3, logo, novamente $R_2 = 0.67$.

9.6.1 Mean Average Precision

Uma medida um pouco mais complexa é a **Mean Average Precision**, ou **MAP**, cuja tradução para português seria “Precisão Média Média”. A ideia dessa medida é indicar a média das médias das precisões obtida em cada consulta, sendo que a média da consulta é feita a partir da média das precisões a cada documento obtido.

Usando como exemplo as consultas C_1 e C_2 , já da Tabela 9.5, é possível ver que C_1 apresenta a lista de precisões

$$\langle 1, 0.67, 0.60, 0.57, 0.56 \rangle$$

, e C_2 apresenta a lista de precisões

$$\langle 1, 0.67, 0 \rangle$$

, onde se assume zero para os documentos não apresentados. A média de C_1 é então 0,68 e a média de C_2 é 0,62. Logo a média das médias é 0,62.

A fórmula para o valor médio de precisão, MVP_c , de uma consulta c , é:

$$MVP_c = \frac{1}{|R_c|} \sum_{k=1}^{|R_c|} P(R_c[k]), \quad (9.9)$$

onde $R_i[k]$ é o k -ésimo documento de R_c , o conjunto de documentos relevantes na resposta a consulta c e $P(R_c[k])$ é a precisão nessa posição.

Com isso, a MAP para um conjunto de N_c consultas é:

$$MAP = \frac{1}{N_c} \sum_{i=1}^{N_c} MVP_i \quad (9.10)$$

9.6.2 Teste F

A revocação e a precisão mantém uma relação, já que, para uma heurística qualquer, é comum que com o aumento da revocação, isto é, quanto mais objetos buscados são encontrados, se espera

uma diminuição da precisão. A medida F tenta transformar essa relação em um número, e se aplica a todos os objetos retornados na busca, permitindo assim comparar, com um número só, as heurísticas avaliadas.

O **teste-F**, **medida-F**, ou simplesmente **F**, ou ainda $F + \beta$, é uma medida de acurácia baseada na precisão e na revocação, com uma parâmetro β positivo real, apresentada na Equação 9.11. Sua forma mais comum é F_1 , equivalente média harmônica. Outros valores comuns de β são 0,5 e 2 (Christopher D. Manning, Raghavan e Schütze, 2009a).

$$F_\beta = (1 + \beta^2) \frac{P \times R}{\beta^2 P + R} = \frac{(1 + \beta^2) TP}{(1 + \beta^2) TP + \beta^2 FN + FP} \quad (9.11)$$

$$F = F_1 = 2 \times \frac{P \times R}{P + R} \quad (9.12)$$

9.7 DCG

As medidas apresentadas até agora consideravam se um documento é relevante ou não, não considerando um grau de relevância. Essa informação, porém, se disponível, é importante, já que claramente alguns documentos são mais relevantes do que outros. Um bom mecanismo de busca, que retorna uma resposta ordenada, deve buscar ordenar os documentos pela ordem decrescente de relevância, isto é, mostrar primeiro os mais relevantes.

O **Discounted Cumulative Gain**, (DCG), ou ainda Ganho Acumulado Descontado, é uma medida da utilidade dos documentos em uma lista ordenada por relevância (Järvelin e Kekäläinen, 2000), que foi criada para comparar dois mecanismos de busca em função da qualidade de suas respostas ordenadas.

Essa medida parte da premissa que a cada documento da resposta pode ser associada uma relevância numérica, ou um ganho, G , obtido quando o documento é encontrado. Isso é comum de acontecer nas coleções, tanto com avaliações feitas por um, quanto por vários juízes, e ainda aproximado por outras estratégias, como análise de *clicks*.

O DCG e suas variantes foram criados a partir de duas premissas sobre o resultado de uma consulta:

1. Documentos mais relevantes tem mais valor que documentos parcialmente ou não relevantes, e
2. Documentos mais relevantes devem aparecer mais perto do início da lista que documentos menos ou não relevantes.

Para atender a primeira premissa, é possível apenas somar o ganho de todos os documentos encontrados até uma posição. Assim é definido o **ganho Acumulado Direto**, **Direct Cumulated Gain** (CG), que representa a soma dos ganhos até uma posição i (Järvelin e Kekäläinen, 2000):

$$CG_i = \sum_{i=1}^i G_i \quad (9.13)$$

Assim, com o ganho acumulado direto, um mecanismo de busca é melhor que outro se, até uma posição especificada, entrega documentos com mais relevância.

Isso permite, dado um vetor de ganhos da consulta, calcular um vetor do CG da consulta. Assim, um vetor

$$\langle 3, 1, 0, 2, 4, 0, 0 \rangle$$

gera o vetor de CG

$$\langle 3, 4, 4, 6, 10, 10, 10 \rangle$$

, onde

$$V_{CG} = \{v_i = \sum_{j=1}^p iCG_j\} \quad (9.14)$$

Para atender a segunda premissa é importante colocar a ordem de entrega na medida, assim, o **DCG** é calculado aplicando um desconto ao ganho do documento relativo a posição do documento na lista ordenada. Assim, quanto mais longe do início um documento aparecer, menor vai ser seu ganho. Sua fórmula, como proposta inicialmente, era³:

$$DCG_p = \begin{cases} G_1, & \text{se } i = 1 \\ G_{i-1} + \frac{G_i}{\log i}, & \text{caso contrário} \end{cases} \quad (9.15)$$

Essa definição também permite gerar um vetor do DCG para uma consulta. No caso da mesma consulta, agora teríamos o vetor de DCG:

$$\langle 3.0, 4.0, 4.0, 5.0, 6.7, 6.7, 6.7 \rangle$$

Ambas as medidas, CG e DCG, podem também ser analisadas em sua média, por posição. Isso pode ser usado, então, para comparar dois mecanismos. As medidas são calculadas para N_c sendo o tamanho da resposta (Baeza-Yates e Ribeiro-Neto, 2011):

$$\overline{CG}_i = \frac{1}{N_c} \sum_{i=1}^{N_c} CG_i \quad (9.16)$$

$$\overline{DCG}_i = \frac{1}{N_c} \sum_{i=1}^{N_c} DCG_i \quad (9.17)$$

$$(9.18)$$

Mais tarde, Järvelin e Kekäläinen (2002) propuseram uma medida adicional que indica a qualidade do DCG em função de um DCG ideal, o **Normalized DCG**, (NDCG), ou **DCG normalizado**. Esse DCG ideal seria o obtido com os documentos ordenados de forma decrescente exatamente por sua relevância.

O ganho dessa nova medida é entender a diferença em magnitude entre dois mecanismos de busca, já que o DCG não dá ideia da importância de um ou outro valor.

Para realizá-la é preciso então saber a ordenação ideal da resposta para uma consulta e aplicar o DCG sobre ela, obtendo um DCG'. O valor normalizado, para uma consulta, é então:

$$NDCG_p = \sum_{i=1}^p \frac{DCG_i}{DCG'_1} \quad (9.19)$$

Supondo um vetor da resposta ideal sendo

$$\langle 4, 3, 2, 1, 0, 0, 0, 0 \rangle$$

³A notação recursiva evita ter que colocar um fato para evitar o log ser zero

, os vetores ideais seriam

$$\text{CGI} = \langle 4, 7, 9, 10, 10, 10, 10 \rangle$$

e

$$\text{DCGI} = \langle 4.0, 7.0, 8.26, 8.76, 8.76, 8.76, 8.76 \rangle$$

Dessa forma, o vetor do NDCG seria:

$$\text{NDCG} = \langle 0.75, 0.57, 0.48, 0.57, 0.77, 0.77, 0.77 \rangle$$

Da mesma maneira que é possível fazer com os valores de CG e DCG, os valores de NDCG para várias consultas podem ser agregados em uma média e assim mecanismos de busca podem ser comparados.

CAPÍTULO 10

REPRESENTAÇÕES DE TEXTO

Este capítulo faz uma introdução as várias formas de representação de texto.

As representações de palavras, fragmentos de textos e textos, são criadas com intuições específicas, normalmente ligados a alguma forma de interpretar o significado do mesmo. Na Busca e Recuperação da Informação, o interesse geral é buscar documentos que sejam relevantes a uma consulta feita, de certa forma representando sua *aboutness*. Já nas tarefas hoje ligadas a Inteligência Artificial e ao Aprendizado de Máquina, o significado específico do texto é mais importante, porém as técnicas de Recuperação da Informação ainda são úteis.

10.1 Representação Booleana

Os primeiros sistemas de Busca de Documentos partiram de iniciativas ad-hoc, e tinham o foco na modelagem de documentos por meio de palavras chaves. Assim, o importante era se um documento era representado ou não por uma palavra chave. As consultas eram operações lógicas sobre palavras chave. Com o avanço da informática, as palavras chave passaram de determinadas por seres humanos, possivelmente especialistas no processo de indexação, para a indexação automática dos títulos, resumos e textos completos dos documentos. Dessa maneira, os primeiros modelos trabalhavam com uma representação do documento pela presença ou não de termos, por consultas lógicas sobre essa presença, e acabou sendo denominado, mais tarde, de Modelo Booleano (Baeza-Yates e Ribeiro-Neto, 2011; Salton e McGill, 1983).

Deixando claro, o Modelo Booleano usa representações dos documentos por meio da presença ou não de termos, e consultas usando operações lógicas sobre essa presença. Mais modernamente essas representações são estudadas na forma de vetores de termos, isto é, vetores onde cada posição representa a ocorrência ou não do termo no documento, o que é chamado de codificação ***one-hot*** (Baeza-Yates e Ribeiro-Neto, 2011).

Como a busca é feita pelos termos, estruturas como a lista invertida são usadas. Matematicamente, porém, estudamos a Matriz Termo x Documento, que no caso do Modelo Booleano, é preenchida apenas com zeros e uns.

Além disso, o uso da palavra termo é feito para indicar que o que está sendo indexado não é necessariamente uma palavra do texto, mas uma representação que pode ser obtida dessa palavra, como seu radical ou seu lema.

10.2 Representações Geométricas ou Vetoriais de Documentos

Um das proposta seminais da área de BRI foi o Modelo Vetorial (Salton, 1965). Nesse modelo, documentos são considerados como vetores em espaço vetorial onde as dimensões são definidas pelos termos. Isso também é indicado na Matriz Termo x Documento (ou na sua inversa, a Documento x Termo). A diferença principal é que ao invés de considerar apenas a presença do termo no documento é considerado algum número que indique a importância dessa presença. Rapidamente foi proposta a ideia de que essa importância era proporcional a sua frequência no documento e inversamente proporcional a sua frequência na coleção.

Os Modelo Vetorial fez grande sucesso e escala bem para grandes coleções, e permite várias sofisticações, como a inclusão de pesos para palavras que não aparecem mas são sinônimas daquelas que aparecem, criação de Theasurus automáticos, medidas de similaridade entre documentos, e ainda mais, levou a consideração de outras formas de usar representações vetoriais. Uma das motivações para outras formas de representar é a quantidade enorme de dimensões, na ordem de dezenhas de milhares, e o fato que a matriz Termo x Documento é muito esparsa.

Isso levou a ideia de manipulações algébricas que buscasse representar significados em vez de termos, o que levou a modelos como o LSI (Thomas K. Landauer, McNamara et al., 2007), que usa os conceitos de auto-vetores e auto-valores para determinar um conjunto de dimensões que busca representar a semântica gerada por conjuntos de termos. Outras propostas se seguiram com a ideia de aplicar manipulações algébricas nas Matrizes Termo Documento para buscar espaços mais densos, mas de menor dimensionalidade.

Mais tarde, isso vai evoluir para a noção de *embeddings* (Mikolov et al., 2013).

10.3 Modelos Probabilísticos e de Linguagem

Uma outra abordagem, que também se inicia junto com o início da Busca e Recuperação de Informação, é a de buscar a probabilidade de um documento responder a uma consulta (van Rijsbergen, 1979).

O Modelo Probabilístico é proposto inicialmente sobre o modelo Booleano, e o que propõe são novas fórmulas de calcular a probabilidade de um documento ser relevante. Essas fórmulas imediatamente modificadas para considerar representações não booleanas e vão levar, mais tarde, a um dos modelos de maior sucesso, o BM25.

Por outro lado, uma linha de pesquisa busca Modelos de Linguagem. Modelos de Linguagem buscam calcular a probabilidade de uma sentença ser gerada na linguagem dentro de certas condições, normalmente uma sequência de termos. Nos modelos de linguagem se calcula a probabilidade da consulta gerar o documento, ou vice-versa. Esses modelos de linguagem, junto com os embeddings, vão, mais recentemente, levar as principais ferramentas de Aprendizado de Máquinas para textos nos últimos anos, as LLMs.

10.4 Embeddings

Na linha de evolução das ideias sobre como representar significados dos termos a Matriz Termo x Documento foi durante muito tempo o fator determinante que permitia construir relação entre os termos (em Matrizes Termo x Termo). Além disso, representações na forma de tesouros, como o Wordnet, também traziam informações semânticas.

Um ***embedding*** é uma maneira de representar dados em um espaço vetorial onde as estruturas e relações nesses dados são de alguma forma preservadas em termos de distâncias ou estruturas de vizinhança nesse espaço (Dan Jurafsky e J. H. Martin, 2022).

O conceito de *word embeddings*, uma maneira específica de representar palavras como vetores em um espaço de dimensão reduzida, foi popularizado por uma equipe de pesquisadores do Google através de um projeto chamado Word2Vec. Esta equipe, liderada por Tomas Mikolov, propôs Word2Vec em 2013. Vale ressaltar que a ideia de representar palavras como vetores numéricos existe há muito tempo no campo da linguística computacional (Mikolov et al., 2013).

Ao representar *word embeddings* temos representações de palavras, não de documentos, e aparece o problema de como representar documentos em certas tarefas. Uma maneira seria concatenar os *embeddings*, outra somá-los ou realizar outro operação de agregação.

De toda forma, foram encontrados usos para os *embeddings* em várias tarefas, tanto a nível de palavras, como entender se elas mudam de significado com o tempo, como a nível de documentos, porém seu uso ficou mais importante com o aparecimento das LLMs.

Uma das principais características das LLMs é ter um primeiro passo onde os termos são transformados em *embeddings*. Esses *embeddings* são aprendidos nas próprias LLMs. Por exemplo, no BERT, o primeiro passo é a busca do *embedding* correspondente a palavra naquele instante do modelo. Esse primeiro *embedding* pode ser considerado não-contextual pois é único para cada palavra. No final da rede, porém, cada *embedding* correspondente a palavra original é considerado um *embedding* contextual, já que leva em conta o contexto em que a palavra está, isto é, a influência de todas as palavras ao seu redor na entrada de acordo com um modelo de atenção de várias cabeças e camadas, em uma rede profunda.

CAPÍTULO 11

MODELOS DE LINGUAGEM

Os **modelos de linguagem** formam a base probabilística de muitas técnicas modernas de Processamento de Linguagem Natural (PLN). Em essência, um modelo de linguagem atribui uma probabilidade a cada sequência possível de palavras em um determinado idioma. Essa distribuição de probabilidade reflete quão **provável** é uma sentença ocorrer naquela língua. Modelos de linguagem são usados em diversas aplicações, incluindo reconhecimento de fala (avaliando a plausibilidade de hipóteses de transcrição), correção ortográfica, digitação preditiva e sistemas de diálogo, assim como na tradução automática e resumo de texto, entre outras. Neste capítulo, discutimos os fundamentos probabilísticos dos modelos de linguagem, abrangendo desde modelos baseados em *n*-gramas de Markov até os modelos neurais mais avançados, incluindo *transformers* e modelos de linguagem de larga escala (LLMs, do inglês *Large Language Models*), como os da família GPT. Também abordaremos arquiteturas de sequência-para-sequência (*seq2seq*), mecanismos de atenção e a evolução histórica que conectou essas ideias.

11.1 Fundamentos Probabilísticos e Modelos de *n*-gramas

A ideia de modelar matematicamente a linguagem natural data dos primórdios da teoria da informação. Shannon (1948), em seu trabalho pioneiro, introduziu o conceito de calcular a **entropia** de uma língua e demonstrou como sequências de caracteres ou palavras podem ser geradas aleatoriamente com base em distribuições de probabilidade estimadas a partir de texto. Essa foi possivelmente a primeira demonstração de um modelo estatístico de linguagem, ao usar cadeias de *Markov* para aproximar a estrutura do inglês escrito. Um **modelo de linguagem** hoje é formalmente definido como uma função que atribui uma medida de probabilidade a cada sequência de palavras (ou símbolos) possível. Ou seja, para uma linguagem L definida sobre um vocabulário Σ , o modelo de linguagem M deve satisfazer $\sum_{s \in L} P(s) = 1$, onde a soma ocorre sobre todas as sentenças s na língua L definidas sobre o alfabeto Σ^1 .

Na prática, calcular $P(s)$ para cada sentença s arbitrária é inviável, pois o número de sentenças possíveis cresce exponencialmente com o tamanho. Em vez disso, assumimos a chamada **hipótese de Markov**, segundo a qual a probabilidade de uma palavra em uma sentença depende apenas de

¹Note que esta notação, inspirada em Christopher D. Manning, Raghavan e Schütze (2009b), usa o símbolo de somatório Σ tanto para denotar a soma quanto o alfabeto, mas o contexto elimina a ambiguidade.

um número limitado de palavras anteriores. Essa suposição simplifica o cálculo de probabilidades de sentenças. Por exemplo, em um modelo de linguagem de **unígrama**, assume-se independência completa entre as palavras, de modo que a probabilidade de uma sentença é apenas o produto das probabilidades individuais de suas palavras. Já em um modelo de **bigramas**, cada palavra depende apenas da palavra imediatamente anterior; em um **trígrama**, depende das duas anteriores, e assim por diante. De forma geral, em um modelo de n -gramas (onde n indica o tamanho da janela de contexto), a probabilidade de uma sequência de palavras w_1, w_2, \dots, w_m pode ser aproximada por:

$$P(w_1 w_2 \dots w_m) \approx \prod_{i=1}^m P(w_i | w_{i-(n-1)}, \dots, w_{i-1}), \quad (11.1)$$

onde para $i = 1$ usamos convenções especiais (por exemplo, um símbolo de início de sentença). No caso extremo de $n = 1$ (unígrama), isso reduz-se a $P(w_1 \dots w_m) = \prod_{i=1}^m P(w_i)$, ignorando completamente a ordem das palavras. Em n -gramas com $n > 1$, capturamos parcialmente a estrutura sequencial da linguagem: um modelo de bigramas considera dependências de primeira ordem (cada palavra depende só da anterior), e um de trígramas, dependências de segunda ordem (cada palavra depende das duas anteriores), e assim por diante.

Para estimar os parâmetros de um modelo de n -gramas, utilizamos corpora de texto. Em sua forma mais simples, a estimativa de máxima verossimilhança para a probabilidade condicionada $P(w_j | w_{j-(n-1)}, \dots, w_{j-1})$ é dada pela frequência relativa:

$$P_{\text{MLE}}(w_j | w_{j-(n-1)}, \dots, w_{j-1}) = \frac{\#(w_{j-(n-1)}, \dots, w_{j-1}, w_j)}{\#(w_{j-(n-1)}, \dots, w_{j-1})}, \quad (11.2)$$

onde $\#(x)$ denota a contagem de ocorrências da sequência x no corpus de treinamento. Essa estimativa, contudo, sofre com o problema de atribuir probabilidade zero a qualquer sequência n -grama não observada no corpus. Como mesmo corpora muito grandes não cobrem todas as construções possíveis da língua, é essencial aplicar técnicas de **suavização** (*smoothing*) para ajustar as probabilidades, redistribuindo um pouco de massa de probabilidade para sequências não vistas. Diversos métodos de suavização clássicos incluem Laplace (ou *add-one*), Good-Turing e Kneser-Ney, além de abordagens por interpolação.

Por exemplo, na **suavização por interpolação linear**, combina-se a estimativa de diferentes ordens de modelo. No caso de bigramas interpolados com unígramas:

$$P_\lambda(w_j | w_{j-1}) = \lambda P_{\text{MLE}}(w_j | w_{j-1}) + (1 - \lambda) P_{\text{MLE}}(w_j), \quad (11.3)$$

onde $0 \leq \lambda \leq 1$ é um hiperparâmetro que controla a influência relativa do modelo de bigramas e do modelo de unígrama. Já na **suavização de Dirichlet** (também conhecida como suavização de Bayes com conjugado), incorporamos uma estimativa de um modelo de coleção (por exemplo, as frequências globais no corpus) como pseudocontagens. A fórmula resultante para um modelo de unígramas suavizado por Dirichlet é:

$$P_{\text{Dir}}(w | d) = \frac{\#(w, d) + \alpha P(w | C)}{|d| + \alpha}, \quad (11.4)$$

onde $\#(w, d)$ é a contagem bruta da palavra w no documento (ou conjunto de palavras) d , $|d|$ é o tamanho total de d , $P(w | C)$ é a probabilidade da palavra w de acordo com um modelo de coleção C

(por exemplo, frequência relativa no corpus inteiro), e α é um parâmetro de suavização (equivalente a adicionar α ocorrências virtuais de acordo com o modelo de coleção). Essa forma de suavização é amplamente usada, por exemplo, em modelos de linguagem para Recuperação de Informação, como no ranking probabilístico de documentos.

A qualidade de um modelo de linguagem de n -gramas pode ser avaliada por métricas como **perplexidade**, que é definida como a inversa da raiz m -ésima da probabilidade atribuída a uma sequência de m palavras: $PP(w_1^m) = P(w_1 \dots w_m)^{-1/m}$. Intuitivamente, perplexidade menor indica um modelo que melhor prevê sequências de teste (ou seja, atribui maior probabilidade a elas). Modelos com n maior tendem a ter perplexidades menores em corpus conhecidos, pois capturam mais dependências, mas sofrem com maior sparsidade de dados.

Historicamente, modelos de linguagem baseados em n -gramas dominaram o PLN por décadas devido à sua simplicidade e eficácia razoável. Ferramentas importantes foram desenvolvidas para lidar com esses modelos, incluindo algoritmos eficientes de contagem e armazenamento (como *tries* e listas invertidas) e técnicas de suavização sofisticadas citadas acima. Pesquisas clássicas em reconhecimento de voz e outras áreas mostraram que trigramas bem suavizados podem ser extremamente úteis. No entanto, mesmo com avanços incrementais, modelos de n -gramas enfrentam limitações fundamentais: eles não generalizam bem para contextos muito além de $n - 1$ palavras e não capturam similaridades semânticas entre palavras além do que está explicitamente observado como contexto imediato.

11.2 Modelos de Linguagem Neurais: do *feed-forward* ao *recurrent*

Um avanço significativo ocorreu quando pesquisadores começaram a incorporar **redes neurais** aos modelos de linguagem, permitindo escapar, em parte, da maldição da dimensionalidade dos n -gramas. Bengio et al. (2003) propuseram o primeiro modelo de linguagem neural eficaz, conhecido como *neural probabilistic language model*. A ideia central foi aprender, simultaneamente com a modelagem da sequência, uma representação distribuída para cada palavra do vocabulário. Em vez de tratar cada palavra como um símbolo independente (como no n -grama clássico), a rede neural aprende vetores contínuos de baixa dimensionalidade (embeddings, discutidos em detalhe no próximo capítulo) para as palavras, capturando similaridades semânticas e sintáticas. O modelo de Bengio et al. (2003) usava uma rede do tipo *feed-forward* com uma camada de projeção que mapeava palavras (representadas inicialmente como vetores one-hot) para embeddings, seguida de camadas ocultas totalmente conectadas que previam a próxima palavra. Esse modelo, apesar de computacionalmente custoso para os padrões da época, já demonstrou melhor desempenho que modelos de trigramas com suavização avançada.

Posteriormente, **Mikolov2010RNNLM**² e colegas introduziram o uso de **redes neurais recorrentes (RNNs)** para modelagem de linguagem. Diferentemente das redes feed-forward, as RNNs apresentam conexões recorrentes que lhes permitem manter um estado interno, funcionando como uma memória do contexto já visto na sequência. Isso torna as RNNs naturalmente adequadas para lidar com sequências de comprimentos variados, já que, em teoria, podem usar todo o histórico arbitrário de palavras anteriores ao prever a próxima palavra. Um modelo conhecido como **RNNLM (Recurrent Neural Network Language Model)** mostrou reduções significativas de perplexidade em relação a modelos n -grama equivalentes, comprovando o poder dessa abordagem.

Contudo, as primeiras RNNs sofriam para capturar dependências de longo prazo devido ao problema do **desvanecimento do gradiente**. A introdução das arquiteturas de memória de longa curta duração, ou **LSTM** (*Long Short-Term Memory*), por Hochreiter e Schmidhuber (1997) mitigou esse problema.

²Por exemplo, Mikolov et al. introduziram uma série de trabalhos entre 2010 e 2012 explorando redes neurais recorrentes para modelos de linguagem.

As LSTMs, e mais tarde as variações conhecidas como GRUs (*Gated Recurrent Units*), incorporam portas de controle de fluxo de informação (entrada, esquecimento e saída) que preservam gradientes e informações relevantes ao longo do tempo. Na prática, modelos de linguagem baseados em LSTM tornaram-se estado da arte por volta da década de 2010, conseguindo aproveitar contextos muito maiores do que n -gramas fixos e melhorando resultados em tarefas diversas. Essas redes recorrentes profundas passaram a ser empregadas não apenas para prever a próxima palavra em sequência, mas também como componentes em sistemas mais complexos, como veremos adiante.

11.3 Sequence-to-Sequence e Mecanismos de Atenção

Embora as RNNs unidirecionais (por exemplo, que leem a sequência do início para o fim) funcionem bem para modelar linguagem de forma genérica, algumas aplicações específicas requerem gerar uma sequência de saída a partir de uma sequência de entrada diferente. Um caso emblemático é a **tradução automática**: gerar uma frase em português a partir de uma frase em inglês, por exemplo. Modelos clássicos de tradução estatística utilizavam pipelines complexos com alinhamento de palavras, modelos de tradução e modelos de linguagem separados. Em 2014, **Sutskever2014** introduziram o paradigma de **aprendizado sequência-para-sequência** (*sequence-to-sequence learning*) usando redes neurais, que revolucionou o campo. Em um sistema típico *seq2seq*, uma RNN chamada **encoder** lê a sequência de entrada (por exemplo, a frase em inglês) e codifica seu significado em um vetor contínuo (às vezes chamado de vetor de contexto ou estado final). Em seguida, uma segunda RNN chamada **decoder** gera a sequência de saída (frase em português) palavra por palavra, condicionada apenas nesse vetor de contexto inicial provido pelo encoder. Durante o treinamento, o decoder aprende a produzir saídas desejadas (traduções corretas) a partir do vetor produzido pelo encoder para cada sentença de entrada.

No modelo seq2seq básico de **Sutskever2014**, o vetor de contexto era o estado final do encoder (por exemplo, a última saída de uma LSTM após ler toda a frase de entrada). Porém, logo percebeu-se que forçar todo o conteúdo semântico de uma frase potencialmente longa a caber num único vetor de dimensão fixa era uma limitação severa, especialmente para frases mais extensas ou complexas. A solução veio com os **mecanismos de atenção** (*attention mechanisms*). Em vez de confiar apenas no último estado do encoder, **Bahdanau2015** propuseram que o decoder pudesse *atender* a diferentes partes dos estados do encoder durante a geração de cada palavra de saída. Em outras palavras, a cada passo de geração, o decoder calcula pesos de atenção que indicam quanta importância deve dar a cada palavra (ou estado oculto correspondente) da sequência de entrada. Esses pesos são tipicamente obtidos passando-se o estado atual do decoder e os estados do encoder por uma função de similaridade ou rede neural simples (chamada de função de alinhamento ou pontuação), seguida de uma normalização softmax. O resultado é um conjunto de pesos $\alpha_{i,j}$ que indica a relevância da i -ésima palavra de entrada para a geração da j -ésima palavra de saída. Em seguida, calcula-se uma combinação linear ponderada dos estados do encoder (também chamada de vetor de contexto dinâmico) que é fornecida ao decoder para auxiliar na previsão da próxima palavra.

Matematicamente, se (h_1, h_2, \dots, h_T) são os estados ocultos do encoder para uma sequência de entrada de comprimento T , e s_{j-1} é o estado oculto do decoder antes de gerar a j -ésima palavra da tradução, então um modelo de atenção global ao estilo de **Bahdanau2015** calcula:

$$e_{i,j} = \text{score}(s_{j-1}, h_i) \quad (11.5)$$

$$\alpha_{i,j} = \frac{\exp(e_{i,j})}{\sum_{k=1}^T \exp(e_{k,j})} \quad (11.6)$$

$$c_j = \sum_{i=1}^T \alpha_{i,j} h_i, \quad (11.7)$$

onde $e_{i,j}$ é o *score* calculado para o i -ésimo elemento de entrada em relação à posição j de saída (por exemplo, usando uma pequena rede feed-forward ou produto escalar entre s_{j-1} e h_i), $\alpha_{i,j}$ são os pesos de atenção (softmax sobre os scores) e c_j é o vetor de contexto resultante para o passo j . Este vetor de contexto c_j é então usado, junto com s_{j-1} , para determinar s_j (o novo estado oculto do decoder) e a distribuição de probabilidade sobre a próxima palavra de saída a ser gerada.

O mecanismo de atenção não apenas melhorou drasticamente a qualidade de traduções automáticas neurais, permitindo que o decoder focasse efetivamente nas palavras relevantes da entrada para cada parte da saída, mas também provou ser útil em muitas outras tarefas de PLN, como resumo automático, legendagem de imagens e perguntas e respostas. Além disso, a atenção introduziu um nível de interpretabilidade aos modelos neurais de sequência: os pesos de atenção muitas vezes podem ser visualizados como uma matriz de alinhamento entre entrada e saída, semelhante aos alinhamentos explícitos dos sistemas estatísticos tradicionais.

11.4 *Transformers*: Atenção Multi-cabeças e Pré-treinamento em Larga Escala

Após a adoção bem-sucedida do mecanismo de atenção em arquiteturas recorrentes, um próximo passo na evolução foi remover completamente a dependência em recorrências, usando a atenção como principal meio de modelar dependências em sequências. Isso levou à criação da arquitetura conhecida como **Transformer**. Proposta por Vaswani et al. (2017), a frase de efeito do artigo já indicava o cerne da ideia: "*Attention Is All You Need*". Os Transformers eliminam as RNNs e, em vez disso, processam a sequência inteira em paralelo por meio de camadas de atenção auto-regressiva (no decoder) e auto-atenção (no encoder).

Um Transformer típico consiste em um **encoder** e um **decoder** empilhados em múltiplas camadas (blocos). Cada camada de encoder tem subcamadas: uma de **auto-atenção multi-cabeças** e outra de **rede feed-forward** posicionada em sequência, com conexões residuais e normalização layer-norm em cada subcamada. A auto-atenção do encoder permite que cada posição da sequência de entrada preste atenção a outras posições dessa mesma sequência, ou seja, o encoder aprende representações para cada palavra levando em conta o contexto global de toda a frase de entrada. O termo "multi-cabeças" refere-se a usar múltiplos conjuntos de projeções lineares (chamadas cabeças) para projetar as representações de entrada em subespaços distintos e aplicar a atenção em cada um deles independentemente; isso permite que o modelo capture diferentes tipos de relações semânticas ou sintáticas simultaneamente. Os resultados de todas as cabeças são concatenados e passados por outra projeção linear, compondo assim a subcamada de atenção completa.

Matematicamente, a atenção em cada cabeça h recebe três matrizes: Q (consultas), K (chaves) e V (valores), que são obtidas projetando as representações de entrada usando matrizes de peso aprendidas (W_h^Q , W_h^K , W_h^V). Para uma cabeça de atenção, computa-se:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V, \quad (11.8)$$

onde d_k é a dimensão das chaves (um fator de escala para estabilizar os gradientes). A operação QK^\top produz uma matriz de similaridades entre todas as posições (consultas contra chaves); a normalização softmax em cada linha produz pesos de atenção que somam 1 para cada elemento de Q , e esses pesos então ponderam a combinação linear dos valores V . No caso de auto-atenção no encoder, $Q = K = V$ corresponde às representações das palavras da própria sentença de entrada. No decoder, há duas camadas de atenção distintas: uma de auto-atenção (onde $Q = K = V$ nas representações já geradas do próprio decoder, com máscara para impedir acesso a passos futuros) e uma de atenção cruzada mirando o encoder (onde Q são as representações do decoder e $K = V$ vêm das saídas do encoder), permitindo que o decoder consulte informações codificadas do input.

Graças à ausência de recorrência, os Transformers podem ser treinados de forma muito mais paralelizada, aproveitando o poder das GPUs e TPUs de forma eficiente, uma vez que a atenção e as camadas feed-forward podem ser aplicadas a todos os tokens simultaneamente. Para incorporar a noção de ordem sequencial (já que, sem recorrência, o modelo por si só não sabe a posição relativa das palavras), os Transformers utilizam **codificações posicional** adicionadas às embeddings iniciais das palavras na entrada. Essas codificações podem ser fixas (por exemplo, senóides de diferentes frequências, como no trabalho original) ou aprendidas.

Os Transformers rapidamente superaram as arquiteturas baseadas em RNN em diversos benchmarks de tradução e outras tarefas, estabelecendo novos estados da arte. No entanto, talvez o impacto mais profundo dos Transformers tenha sido habilitar o treinamento de modelos de linguagem em larga escala de forma mais eficaz. A partir de 2018, emergiu a tendência de treinar modelos de linguagem cada vez maiores em enormes quantidades de texto não supervisionado e depois adaptá-los (por fine-tuning ou outras técnicas) para tarefas específicas. Esse paradigma de **pré-treinamento de modelos de linguagem** seguido de **ajuste fino** transformou a área de PLN.

Um dos primeiros resultados de grande destaque nessa linha foi o modelo **BERT** (*Bidirectional Encoder Representations from Transformers*), introduzido por J. Devlin et al. (2018a). O BERT é um modelo Transformer do tipo encoder puro, treinado de forma bidirecional em texto. Diferente dos modelos de linguagem tradicionais que predizem a próxima palavra, o pré-treinamento do BERT utiliza duas tarefas: (1) **modelagem de linguagem mascarada**, onde 15% das palavras de cada sequência são substituídas por um símbolo especial (ou ocasionalmente por outra palavra ou deixadas inalteradas) e o modelo deve prever quais eram as palavras originais; e (2) **previsão de próxima sentença**, onde o modelo recebe pares de sentenças e aprende a indicar se a segunda frase segue a primeira em um documento original. A primeira tarefa faz o BERT aprender representações internas ricas que incorporam contexto à esquerda e à direita (ou seja, bidirecionais) para cada palavra, e a segunda adiciona alguma noção de coerência de sentenças. O resultado foi um modelo com 12 camadas (na versão BERT-base, com 110 milhões de parâmetros) ou 24 camadas (BERT-large, 340 milhões de parâmetros) que, após pré-treinado em um grande corpus (Wikipedia + BooksCorpus), pode ser *fine-tuned* para diversas tarefas de PLN como classificação de texto, reconhecimento de entidades nomeadas e respostas a perguntas, obtendo resultados líderes em muitas delas. Modelos derivados e variantes do BERT logo surgiram, como o **RoBERTa** ([liu2019roberta](#)) (que aprimorou o treinamento do BERT ao remover a tarefa de próxima sentença e treinar por mais tempo com mais dados) e outros modelos focados em diferentes línguas ou domínios.

Paralelamente, outra linha de pesquisa explorou modelos de linguagem pré-treinados de forma unidirecional (ou auto-regressiva) para geração de texto. Um precursor foi o **OpenAI GPT** (Generative

Pre-Training) original, lançado em 2018, que utilizava um Transformer decoder treinado para prever a próxima palavra (modelagem de linguagem clássica) em grandes quantidades de texto, e depois ajustado para tarefas condicionalmente (por exemplo, adicionando um cabeçote de classificação e treinando no conjunto de treino de uma tarefa específica). A segunda geração, *GPT-2* (2019), chamou atenção por seu tamanho e capacidade de gerar texto assustadoramente coerente; o modelo de maior porte dessa família tinha 1,5 bilhão de parâmetros e foi inicialmente não divulgado integralmente ao público devido a preocupações com uso indevido. Logo depois, **Brown2020** introduziram o *GPT-3*, um modelo de linguagem com impressionantes 175 bilhões de parâmetros, treinado sobre praticamente toda a internet (centenas de bilhões de palavras). O GPT-3 demonstrou uma habilidade inédita de realizar tarefas de PLN sem ajuste fino explícito: apenas fornecendo alguns exemplos (contexto de prompt) na própria entrada, ele consegue resolver problemas de tradução, responder perguntas, resumir textos, entre outras tarefas – o chamado *few-shot learning*. Esse resultado evidenciou que a escala do modelo e dos dados pode induzir capacidades emergentes.

O sucesso de GPT-3 e modelos contemporâneos inaugurou a era dos **grandes modelos de linguagem** (LLMs) em ampla escala. Empresas e instituições lançaram arquiteturas similares, muitas vezes com ainda mais parâmetros ou treinadas em corpora multilingues e multimodais. Por exemplo, o modelo **PaLM** (Pathways Language Model) do Google alcançou 540 bilhões de parâmetros, e o projeto colaborativo **BLOOM** treinou um modelo de 176 bilhões de parâmetros disponível abertamente. Em 2022, a OpenAI apresentou o **ChatGPT**, uma versão conversacional afinada a partir da família GPT (com técnicas de ajuste fino usando feedback humano, conhecidas como RLHF, *Reinforcement Learning from Human Feedback*). O ChatGPT popularizou mundialmente o uso de LLMs, demonstrando de forma interativa o que esses modelos são capazes de fazer, de responder perguntas a compor textos complexos mediante instruções em linguagem natural. Em 2023, o lançamento do *GPT-4* (**OpenAI2023GPT4**) elevou ainda mais o patamar, incorporando capacidade multimodal e exibindo desempenho em testes padronizados próximo ao nível humano em várias áreas de conhecimento.

Esses modelos de linguagem profunda, apesar de sua enorme complexidade, conceitualmente ainda são modelos de linguagem no sentido que aprendem uma distribuição $P(s)$ sobre sequências. A grande diferença está em sua escala (bilhões de parâmetros, treinados em trilhões de palavras) e nas arquiteturas avançadas (Transformers) que conseguem capturar dependências de longo alcance e vários aspectos do significado. Enquanto modelos n -gramas tradicionais se baseavam puramente em estatísticas locais de coocorrência, os LLMs capturam padrões linguísticos em múltiplos níveis (léxico, sintático, semântico, pragmático) devido ao seu treinamento massivo. Vale ressaltar que, junto com seus avanços, surgem novos desafios: estes modelos tendem a ser opacos, podem incorporar vieses presentes nos dados, e podem gerar informação incorreta ou conteúdo inapropriado se não forem cuidadosamente filtrados e controlados. Ainda assim, eles representam hoje o estado da arte em uma variedade de tarefas de PLN, muitas vezes funcionando como plataformas gerais sobre as quais sistemas específicos podem ser construídos.

11.5 Aplicações e Considerações Finais

Modelos de linguagem constituem um pilar fundamental em PLN. Além das aplicações já mencionadas (como digitação preditiva, reconhecimento de voz, tradução automática e diálogo), eles também são empregados em tarefas como detecção de spam (modelando a probabilidade de sequências de palavras típicas de spam vs. não spam), preenchimento automático de consultas de busca, análise de sentimento (como parte de arquiteturas maiores) e geração de conteúdo criativo (poemas, código-fonte, etc.). Os LLMs recentes estendem essas fronteiras, servindo como motores para assistentes virtuais avançados e possivelmente revolucionando a forma como interagimos com computadores.

Apesar dos progressos, entender e aprimorar modelos de linguagem continua sendo um campo ativo de pesquisa. Questões de **eficiência** são importantes: modelos enormes são custosos de treinar e executar, o que tem levado a investigações em compressão de modelos, distilação e técnicas mais eficientes de atenção. Há também o aspecto de **controle e alinhamento**: garantir que modelos de linguagem gerativos sigam instruções humanas de forma confiável e não produzam saídas inadequadas. Nesse sentido, técnicas de alinhamento com preferência humana (como o RLHF mencionado no contexto do ChatGPT) são cada vez mais relevantes.

Em resumo, os modelos de linguagem evoluíram de simples cadeias de Markov para sofisticadas arquiteturas neurais que capturam uma variedade impressionante de fenômenos linguísticos. Essa evolução histórica desde os experimentos de Shannon, passando pelos *n*-gramas, modelos neurais, mecanismos de atenção e Transformers, até os gigantescos LLMs atuais reflete não apenas avanços algorítmicos, mas também a disponibilidade crescente de recursos computacionais e dados em larga escala. Nos próximos capítulos, continuaremos explorando aspectos relacionados a representações linguísticas e técnicas derivadas desses fundamentos, como os *embeddings* de palavras e sentenças, que tiveram um papel crucial nesse progresso.

CAPÍTULO 12

EMBEDDINGS

No contexto do PLN, chamamos de *embedding* uma representação vetorial de um objeto discreto (por exemplo, uma palavra, frase ou documento) em um espaço contínuo de menor dimensionalidade, de forma que propriedades ou relações semânticas do objeto original sejam preservadas na estrutura geométrica desses vetores (Dan Jurafsky e J. H. Martin, 2022). Em outras palavras, *embeddings* são vetores densos (com a maioria dos componentes não nulos) que conseguem capturar similaridades e relações entre palavras ou textos melhor do que representações esparsas tradicionais. A motivação para usar embeddings deriva da chamada **hipótese distribucional**, formulada por linguistas nos anos 1950 (por exemplo, Zellig Harris) e resumida na famosa citação de Firth: *"Você os conhecerá pela companhia que eles mantêm"*. Essa hipótese postula que o significado de uma palavra pode ser inferido do contexto em que ela aparece. Assim, se duas palavras ocorrem em contextos semelhantes, espera-se que seus significados (ou usos) tenham relação. Os embeddings levam essa ideia ao extremo, mapeando palavras com contextos similares para pontos próximos em um espaço vetorial.

12.1 Representações Discretas vs. Distribuídas

Antes do advento dos embeddings densos, a representação dominante para texto em tarefas de PLN era baseada em vetores esparsos de dimensionalidade muito alta. Por exemplo, na modelagem clássica de documentos para recuperação de informação, usa-se a representação de **saco de palavras** (*bag-of-words*), na qual um documento é representado por um vetor cujo comprimento é o tamanho do vocabulário e que indica a presença ou contagem de cada termo no documento. Essa representação pode ser binária (presença/ausência, conhecida como codificação *one-hot*) ou ponderada por frequência. Uma versão refinada e amplamente utilizada é a representação TF-IDF (Term FrequencyInverse Document Frequency), que pondera cada termo pelo produto entre sua frequência no documento e o logaritmo inverso da sua frequência no corpus, destacando termos que caracterizam melhor o documento em comparação à coleção geral (Baeza-Yates e Ribeiro-Neto, 2011; Salton e McGill, 1983). Embora simples e efetivas em certos cenários, essas abordagens geram vetores extremamente esparsos e de alta dimensionalidade (tipicamente dezenas de milhares de dimensões correspondentes ao vocabulário). Além disso, elas não capturam relações semânticas entre termos: palavras diferentes são representadas em posições totalmente distintas do vetor, de modo que sinônimos ou termos correlatos não têm nenhuma proximidade automática nessa representação.

Uma forma inicial de contornar a alta dimensionalidade e esparsidão foi a utilização de técnicas de **redução de dimensionalidade** em matrizes termo-documento ou termo-termo. A **Análise Semântica Latente** (LSA, do inglês *Latent Semantic Analysis*), proposta por Thomas K. Landauer, McNamara et al. (2007), é um exemplo notável. A LSA aplica uma decomposição por valores singulares (SVD) à matriz termo-documento, fatorando-a em componentes latentes. Mantendo-se apenas as k maiores singularidades e vetores singulares correspondentes, obtém-se uma representação dos termos e documentos em um espaço de dimensão k (tipicamente algumas centenas) que maximiza a captura da variância dos dados originais. Essa técnica consegue agrupar termos que aparecem em contextos similares e, empiricamente, mostrou capturar relações de sinônimia e tópicos latentes. Em essência, a LSA produziu uma das primeiras formas de embeddings para palavras e documentos, ainda que derivadas de álgebra linear sobre contagens.

12.2 Embeddings de Palavras Não Contextuais

O termo *word embedding* ganhou proeminência após os trabalhos de Mikolov et al. (2013), que introduziram modelos simples e eficazes para aprender representações vetoriais densas de palavras treinando redes neurais em grandes corpus de texto. Esses modelos, conhecidos coletivamente como **Word2Vec**, criaram vetores de dimensão tipicamente 100, 300 ou 600 para cada palavra de um vocabulário possivelmente com milhões de palavras, capturando relações semânticas de forma impressionante. Por exemplo, operações vetoriais com esses embeddings apresentaram resultados anedóticos famosos, como $\text{vec}(\text{rei}) - \text{vec}(\text{homem}) + \text{vec}(\text{mulher}) \approx \text{vec}(\text{rainha})$, sugerindo que o modelo codificou analogias de gênero e realeza em sua estrutura geométrica.

O Word2Vec possui duas arquiteturas principais de treinamento, ambas baseadas em redes neurais rasas (isto é, de poucas camadas): o modelo *CBOW* (*Continuous Bag-of-Words*) e o modelo **Skip-gram**. Em ambos os casos, o objetivo é usar o contexto local de palavras em uma sentença para aprender as representações. No modelo CBOW, a rede neural tenta prever uma palavra alvo dado seu contexto (por exemplo, dadas as 2 palavras anteriores e 2 posteriores, prever a palavra central); já no Skip-gram, o objetivo é inverso: dado um termo central, prever as palavras de seu contexto próximo. Formalmente, o modelo Skip-gram procura maximizar a probabilidade de ocorrência de palavras de contexto $c \in C(w)$ ao redor de uma palavra alvo w . Seja V o vocabulário e v_w o vetor (embedding) associado à palavra w . Um esquema simplificado da função objetivo a ser maximizada é:

$$L = \sum_{w \in V} \sum_{c \in C(w)} \log P(c | w), \quad (12.1)$$

onde $P(c | w)$ é modelado via uma função softmax utilizando o produto escalar dos vetores das palavras (também chamados de vetores de entrada e de saída no modelo Skip-gram):

$$P(c | w) = \frac{\exp(v_c \cdot v_w)}{\sum_{u \in V} \exp(v_u \cdot v_w)}. \quad (12.2)$$

Em práticas de implementação, refinamentos como *negative sampling* (amostragem negativa) ou *hierarchical softmax* são usados para tornar o treinamento computacionalmente viável para vocabulários muito grandes, aproximando a função de perda de forma eficiente (Mikolov et al., 2013). Ao final do treinamento, as camadas intermediárias da rede (os vetores v_w aprendidos) atuam como embeddings de cada palavra. Importante ressaltar que, embora a arquitetura seja motivada por um modelo de linguagem (prever palavras omitidas em um contexto), o objetivo final aqui não é construir um modelo

que gera texto com alta probabilidade, mas sim aprender vetores de palavras que sejam úteis para medir similaridade semântica e servir de insumos a outras tarefas.

Outra contribuição fundamental foi o modelo **GloVe** (do inglês *Global Vectors for Word Representation*), de Pennington, Socher e Christopher D Manning (2014). O GloVe segue uma abordagem diferente mas relacionada: ele parte de estatísticas globais de cocorrência de palavras no corpus (por exemplo, uma matriz contando quantas vezes cada par de palavras ocorre no mesmo contexto) e aprende embeddings ao fatorar essas contagens de forma aproximada. A função de custo do GloVe é construída para que o produto escalar entre os vetores das palavras i e j somado aos seus vieses (b_i e b_j) tente aproximar o logaritmo da contagem de cocorrência X_{ij} entre essas palavras:

$$J = \sum_{i,j=1}^{|V|} f(X_{ij})(v_i^\top \tilde{v}_j + b_i + \tilde{b}_j - \log X_{ij})^2, \quad (12.3)$$

onde v_i e \tilde{v}_j são, respectivamente, o vetor da palavra i (considerada como palavra de linha) e da palavra j (como palavra de contexto ou de coluna) e $f(X_{ij})$ é uma função de ponderação que reduz a influência de pares muito frequentes ou muito raros (no trabalho original, $f(x) = \min(1, (x/x_{\max})^\alpha)$ com hiperparâmetros escolhidos empiricamente). Em resumo, o GloVe integra aspectos de modelos baseados em contagem e de modelos preditivos: ele utiliza contagens globais como insumo, mas realiza aprendizagem iterativa de vetores via descida de gradiente. Os embeddings GloVe resultantes, assim como os do Word2Vec, demonstraram capturar diversas relações semânticas e sintáticas, e tornaram-se populares por sua facilidade de treinamento e disponibilidade em conjuntos pré-treinados.

Tanto Word2Vec quanto GloVe produzem um único vetor por palavra, independentemente do contexto onde a palavra aparece. Por isso, esses métodos são frequentemente chamados de **embeddings estáticos** ou não contextuais. Há outras extensões e variações dignas de nota nesse paradigma. Por exemplo, Bojanowski et al. (2016) propuseram incorporar subunidades de palavra nos embeddings, no modelo conhecido como **FastText**. Em vez de aprender vetores apenas para palavras inteiras, o FastText também aprende vetores para n-gramas de caracteres (subpalavras); o vetor final de uma palavra é a soma de seus n-gramas constituintes. Essa abordagem melhora a representação de palavras raras ou formas derivadas (p.ex. plural, flexões verbais), pois mesmo que uma palavra completa não tenha aparecido com frequência no corpus, suas partes (prefixos, sufixos, raízes) podem ter aparecido em outras palavras, permitindo estimar um vetor razoável. Embeddings de subpalavras também lidam de forma elegante com vocabulário aberto, permitindo obter vetores para palavras não vistas (bastando decompor a palavra em n-gramas conhecidos). Tanto Word2Vec quanto GloVe e FastText geram o que podemos chamar de **espaço vetorial semântico**: um espaço contínuo onde proximidade geométrica correlaciona-se com proximidade semântica (ou de uso) entre palavras. Esses embeddings estáticos de palavras tornaram-se insumo padrão para inúmeros sistemas de PLN durante a segunda metade da década de 2010, substituindo gradualmente as representações esparsas em muitas tarefas.

Convém mencionar também que técnicas semelhantes foram desenvolvidas para obter vetores representativos de unidades maiores de texto, como frases, parágrafos ou documentos. Uma estratégia simples para representar um documento a partir de embeddings de suas palavras é fazer a média (ou soma) dos vetores de palavra dessa abordagem, embora perca a ordem, provou ser um baseline forte em tarefas de classificação de texto e recuperação semântica (Clinchant e Perronnin, 2013). Modelos mais sofisticados, como o **Doc2Vec** (Le e Mikolov, 2014) (também chamado de Paragraph Vector), estendem o Skip-gram introduzindo vetores dedicados para cada documento (além dos vetores de palavra) e treinando conjuntamente para que o vetor do documento, junto com os vetores de algumas palavras nele contidas, possa prever palavras adjacentes. Assim, após o treinamento, cada documento (ou parágrafo) adquire um vetor próprio que captura seu conteúdo temático. No entanto, com o

advento de modelos neurais profundos e contextualizados (que veremos a seguir), técnicas estáticas como Doc2Vec tornaram-se menos utilizadas em aplicações de ponta.

12.3 Embeddings Contextuais: ELMo, BERT, GPT e além

Embora embeddings estáticos tenham sido revolucionários, eles apresentam uma limitação clara: não adaptam a representação de uma palavra de acordo com o seu contexto. Isso é problemático especialmente para palavras **polissemias** (com múltiplos sentidos). Por exemplo, a palavra "banco" pode significar uma instituição financeira ou um assento, dependendo do contexto. Um embedding estático terá que ficar em algum ponto médio entre esses significados ou refletir apenas o sentido predominante no corpus de treinamento. Já um **embedding contextual** é aquele gerado dinamicamente considerando a sentença ou parágrafo em que a palavra ocorre, de modo que a mesma palavra terá vetores diferentes em contextos diferentes, conforme o sentido utilizado.

Os embeddings contextuais despontaram com força em 2018. Um marco inicial foi o modelo **ELMo** (*Embeddings from Language Models*), proposto por Peters et al. (2018). O ELMo consistia em um modelo de linguagem neural bidirecional profundo (com duas camadas de LSTM para esquerda-direita e duas para direita-esquerda) treinado em larga escala. Em vez de utilizar apenas a última camada do modelo de linguagem, o ELMo aproveita uma combinação linear aprendida das camadas internas para gerar, para cada palavra em uma sentença, um vetor contextualizado que incorpora informações de toda a sentença (tanto à esquerda quanto à direita da palavra). Esses vetores demonstraram melhorar significativamente o desempenho de uma ampla gama de tarefas (da análise sintática à detecção de sentimento) quando utilizados como características adicionais ou entradas para os modelos dessas tarefas, superando facilmente abordagens anteriores baseadas apenas em embeddings estáticos.

Na esteira do ELMo, modelos baseados em Transformer rapidamente assumiram o protagonismo em embeddings contextuais. Já discutimos no capítulo anterior o BERT (J. Devlin et al., 2018a), que foi treinado com máscara de palavras e obteve representações ricas. Do ponto de vista de embeddings, o BERT (ou suas variantes, como RoBERTa) pode ser visto como uma função $f_{BERT}(S, i)$ que recebe uma sentença S e um índice i e produz um vetor de dimensão fixa para a palavra na posição i dentro de S , levando em conta todo o contexto de S . Assim, $f_{BERT}(\text{Eu fui ao banco sacar dinheiro}, i = \text{banco})$ resultará em um vetor diferente de $f_{BERT}(\text{O banco da praça está ocupado}, i = \text{banco})$, permitindo desambiguar automaticamente os sentidos. Esses embeddings contextualizados de BERT são de alta dimensão (tipicamente 768 ou 1024 dimensões) e carregam informação muito rica sobre a palavra e seu papel na sentença (tanto que as camadas finais do BERT já são suficientes para resolver tarefas como reconhecimento de entidade ou análise de sentimento com mínimo ajuste fino).

Outro modelo contextual notável é o próprio GPT-2/GPT-3 (baseado no Transformer decoder unidirecional). Embora a aplicação principal do GPT seja geração de texto, ele também produz embeddings contextuais: a cada posição de palavra em um texto de entrada, podemos tomar o vetor de estado oculto dessa posição (após as múltiplas camadas Transformer) como um embedding contextual unidirecional, incorporando apenas o contexto à esquerda. Em tarefas de linguagem natural, às vezes combina-se esses embeddings do GPT (ou de outros modelos auto-regressivos) com embeddings de direção contrária para obter uma representação bidirecional semelhante à do BERT. Entretanto, em geral modelos como BERT ou seus sucessores (por exemplo, **XLNet**, **ELECTRA**, etc.) são preferidos quando o objetivo principal é extrair características contextuais, enquanto modelos tipo GPT são preferidos para geração livre.

Vale ressaltar que embeddings contextuais não são mais vetores associados unicamente a um item lexical no léxico; eles são produzidos por modelos complexos que, dado um texto, produzem uma

representação para cada token. Portanto, o uso de embeddings contextuais em pipelines de PLN geralmente implica usar todo o modelo pré-treinado para codificar os textos de interesse. Isso pode ser computacionalmente custoso, mas o benefício em desempenho tem se mostrado compensador. Na prática, a maioria dos sistemas atuais integra esses modelos de forma fim-a-fim: por exemplo, para classificação de frases, em vez de primeiro extrair embeddings contextuais e depois treinar um classificador separado, simplesmente se toma um modelo como BERT e realiza o ajuste fino adicionando uma camada de classificação ao [CLS] (token de classificação do BERT) ou a alguma agregação dos embeddings das palavras.

Exemplos importantes de embeddings contextuais modernos incluem: - **BERT Multilíngue** e variações específicas para outros idiomas, que permitem obter embeddings contextuais em diferentes línguas, muitas vezes alinhados em um espaço semântico compartilhado (útil para tradução e transferência cross-lingual). - **RoBERTa** ([liu2019roberta](#)), que mostrou que com mais dados e ajuste de hiperparâmetros (removendo tarefa de próxima sentença, aumentando tamanho de batch e taxa de aprendizado), as representações BERT podiam ser ainda melhores. - **ALBERT** e **DistilBERT**, que são versões mais leves de BERT, produzindo embeddings semelhantes com menos parâmetros via fatorização ou distilação. - **ELECTRA**, que treinou um modelo BERT-like não para predizer tokens mascarados, mas para distinguir tokens originais de tokens substituídos por outro modelo, aprendendo assim de maneira mais eficiente e produzindo embeddings de qualidade com menos custo de pré-treino. - **Embeddings do Sentence-BERT (sBERT)**, que são embeddings para sentenças inteiras obtidos a partir de BERT treinado especificamente para que as sentenças tivessem representações de uso direto em medidas de similaridade sem precisar de ajuste supervisionado a cada nova comparação ([reimers2019sentencebert](#)). Esses embeddings de sentença têm aplicações em busca semântica e recuperação de respostas, por exemplo.

O panorama de embeddings contextuais está em constante evolução, mas todos compartilham a ideia central de representar palavras ou textos de forma sensível ao contexto, superando a rigidez dos embeddings não contextuais.

12.4 Vantagens, Limitações e Desenvolvimentos Atuais

Comparação estáticos vs. contextuais: Embeddings estáticos (Word2Vec, GloVe, etc.) apresentam vantagens como simplicidade e eficiência: uma vez treinados, podem ser armazenados e utilizados facilmente como tabelas de vetores para *lookup*, incorporando instantaneamente conhecimento semântico em modelos sem exigir muito custo computacional adicional. Eles também costumam ser treináveis com menos dados e recursos; por exemplo, é possível treinar embeddings de 100 dimensões para um domínio específico (jurídico, biomédico, etc.) com corpora de alguns milhões de palavras, capturando terminologia daquele domínio relativamente bem. Em contrapartida, sua principal limitação é não diferenciarem sentidos: todos os usos de "banco" convergem para o mesmo vetor. Isso também significa que eles não se adaptam a ambiguidades sintáticas ou diferenças de contexto (por exemplo, voz ativa vs. passiva podem demandar nuances diferentes na representação de uma mesma palavra, algo que embeddings estáticos não conseguem refletir).

Já os embeddings contextuais oferecem representações muito mais ricas e acuradas em contextos reais. Eles possibilitam que modelos de PLN interpretem palavras de acordo com o enunciado, resolvendo ambiguidades lexical e referencial em muitos casos. De fato, a incorporação de embeddings contextuais elevou o desempenho em praticamente todas as tarefas de PLN, do *parsing* à resposta a perguntas e inferência textual. A desvantagem principal é o custo: os modelos geradores de embeddings contextuais são grandes e caros de rodar. Enquanto uma tabela de Word2Vec de 3 milhões de palavras

cabe em algumas centenas de MB e pode ser acessada em tempo constante por palavra, um BERT com 340 milhões de parâmetros precisa executar dezenas de multiplicações de matrizes grandes para cada frase, incorrendo em latência e gasto energético significativos. Além disso, embeddings contextuais, por serem produzidos por modelos treinados em enormes quantidades de dados gerais, podem não refletir perfeitamente nuances de um domínio específico a menos que o modelo seja ajustado ou que se use técnicas de adaptação (como *fine-tuning* ou *prompting* apropriado). Em cenários com dados limitados ou hardware restrito, embeddings estáticos ainda podem ser a opção mais viável.

****Integração e uso conjunto:**** Em alguns casos, combina-se informações de embeddings estáticos e contextuais. Por exemplo, em sistemas de recuperação de informação, pode-se expandir a consulta de um usuário buscando termos similares de acordo com embeddings estáticos (que são fáceis de pré-computar para todo vocabulário), enquanto se utiliza um modelo contextual para refinar o ranking considerando o contexto dos termos nos documentos. Da mesma forma, há pesquisas sobre utilizar embeddings estáticos como insumo inicial para modelos contextuais a fim de acelerar seu treinamento ou melhorar sua robustez para palavras raras.

****Desafios persistentes:**** Mesmo com técnicas de ponta, alguns desafios permanecem. Um deles é a **robustez a texto ruidoso** ou não padrão (por exemplo, gírias da internet, erros ortográficos, linguagem de redes sociais). Embeddings estáticos treinados em linguagem formal podem não ter entradas para palavras com grafias criativas, e embora sub-palavras ajudem, o significado pode se perder. Modelos contextuais também podem falhar com textos muito fora do domínio de seu treino. Trabalhos como o de Doval, Vilares e Gómez-Rodríguez (2020) exploram maneiras de treinar embeddings mais robustos a ruído, combinando informações fonéticas e de caracteres para tolerar variações. Outro desafio é a **generalização para múltiplos domínios**: modelos enormes tendem a captar muito do domínio predominante nos dados de treino (por exemplo, texto da web). Ao aplicá-los em domínios especializados (médico, jurídico), pode ser necessário adaptá-los ou até treinar do zero em dados daquele domínio, o que é custoso. Técnicas de ajuste incremental e treinamento contínuo de embeddings estão em pesquisa para facilitar transferências de domínio.

A **interpretabilidade** das representações é outro ponto de atenção. Embeddings, sejam estáticos ou dinâmicos, são notoriamente opacos: as dimensões em geral não têm um significado intrínseco claro (não correspondem a temas únicos, por exemplo). Vu e Iyyer (2019) investigam maneiras de tornar as dimensões mais interpretáveis, por meio de regularizações que incentivem a dispersão de informação ou alinhamento com características linguísticas conhecidas. Ainda assim, compreender exatamente o que foi codificado em um vetor de 300 ou 1024 dimensões é não trivial. Isso também dificulta identificar e mitigar vieses e estereótipos que podem estar presentes nos embeddings. Por exemplo, pesquisadores já demonstraram que embeddings estáticos refletem vieses societais (associando gênero a certas profissões, por exemplo), e modelos contextuais também podem reproduzir ou até amplificar vieses presentes nos dados de treinamento. Desenvolver técnicas para auditar e corrigir essas questões nos espaços embeddados é um campo ativo.

****Tendências recentes:**** Uma tendência recente é incorporar conhecimento estruturado (por exemplo, de ontologias ou bases de conhecimento) diretamente nos embeddings. Também vê-se um movimento em direção a **embeddings multimodais**, que alinham vetores de palavras com vetores de imagens, áudio ou vídeo correspondentes, permitindo, por exemplo, recuperar uma imagem relevante a partir de uma descrição textual. Além disso, com o surgimento de modelos gigantescos de última geração, a fronteira entre embedding e modelo completo começa a se difundir: por exemplo, as últimas camadas de um GPT-3 (com 175B parâmetros) ou de um PaLM (540B) não são facilmente armazenáveis ou utilizáveis isoladamente como vetores estáticos, mas esses modelos funcionam como oráculos capazes de gerar ou avaliar texto de forma condicionada, muitas vezes substituindo tarefas que antes seriam feitas via comparação de embeddings.

Ainda assim, a noção de embeddings se mantém central: mesmo os modelos de linguagem mais modernos produzem internamente representações vetoriais (por exemplo, embeddings de frases ou de instruções) que mediam suas capacidades. Assim, entender e melhorar embeddings sejam estáticos ou contextuais permanece relevante. Pesquisas como as de Roy, Ganguly et al. (2018) mostram que integrar embeddings com mecanismos de normalização de termos pode melhorar sistemas de busca; Ji et al. (2016) propuseram algoritmos (como o WordRank) para aprender embeddings fazendo uma ordenação robusta, em vez de apenas predição local, buscando reduzir a influência de ruídos e outliers nos dados de treino. Essas e outras inovações apontam que há espaço para aperfeiçoar tanto a qualidade quanto a robustez dos embeddings.

Conclusão: A introdução de embeddings densos transformou a área de PLN, proporcionando um salto de desempenho ao capturar de forma distribuída relações de similaridade semântica que antes eram perdidas por representações discretas. Embeddings estáticos como Word2Vec e GloVe inauguraram essa revolução, e hoje embeddings contextuais como os derivados de modelos BERT e GPT elevam ainda mais o patamar, permitindo que cada palavra ou sentença seja interpretada à luz de seu contexto único. O campo continua avançando, explorando embeddings mais especializados, eficientes e interpretáveis. Apesar dos desafios (robustez, vieses, custo computacional), os embeddings agora parte intrínseca de modelos gigantes seguem como alicerces sobre os quais construímos sistemas de linguagem cada vez mais capazes e inteligentes.

DRAFT

CAPÍTULO 13

MODELO BOOLEANO

Uma maneira simples de representar sobre o que trata um texto é por um conjunto de palavras especialmente escolhidas que, de acordo com uma ou mais perspectivas, expliquem sobre o que o texto fala. Ela vem a tona baseada na prática dos bibliotecários: o uso de palavras chave cuidadosamente escolhidas e provavelmente controladas por um glossário.

Os primeiros sistemas de recuperação da informação partiam da ideia de automatizar e estender com funcionalidades de consulta complexas os catálogos de cartões disponíveis nas bibliotecas. Até hoje, sistemas de bibliotecas dependem de bibliotecários registrarem palavras chave para os livros.

A segunda abordagem, que exige um poder computacional maior, porém totalmente exequível hoje em dia, é indexar todas as palavras do documento, e até combinação de palavras, com a premissa que elas tragam o significado do documento junto com elas, no que é chamado *full-text indexing*.

Essa premissa não é óbvia. Por exemplo, um pesquisador em busca de livros escritos com um estilo específico, como o literatura barroca, provavelmente pouco se aproveitará das palavras que realmente representam seu desejo, como “barroco”, e terá que buscar palavras esperadas na literatura desse tipo. Já um pesquisador usando um mecanismo de palavras chaves escolhidas certamente encontrará, entre elas, a descrição do estilo do documento que procura.

Levando isso em consideração, esse capítulo trata de como representar documentos por meio de palavras da forma mais simples, isto é, considerando apenas se a palavra está ou não presente na descrição do documento. Essa descrição, que não precisa ser única, pode ser na forma de um conjunto de palavras-chave escolhidas intencionalmente, ou pela coleção de palavras do documento, o que é conhecido como *full-text* na literatura, provavelmente pré-processadas de alguma forma, como explicado no Capítulo 5.

Essa representação, quando relativa ao texto completo do documento, é conhecida também como *bag of words*.

13.1 Pequena História do Início de Tudo

Esse modelo foi definido teoricamente *a posteriori* do seu uso. Autores importantes da área, que explicam o modelo, não registram sua invenção, como Baeza-Yates e Ribeiro-Neto (2011), ou mesmo Salton e McGill (1983), que fala do modelo como “Sistemas Baseados em Arquivos Invertidos”, hoje

conhecidos como listas invertidas ou índices invertidos. Cooper (1997) afirma que a proposta de usar consultas baseadas em expressões booleanas de descritores de documentos é da década de 50.

Investigando essa questão foi possível chegar a artigos da fase de transição entre catálogos de cartões bem desenvolvidos e os primeiros sistemas que discutiam o uso de computadores (Harman, 2019; Sanderson e Croft, 2012). Allen Kent e coautores, como J.W. Perry, Janet Rees, descrevem necessidades para a recuperação de informação mecanizada em diferentes artigos na década de 50, sendo que uma das propostas mais conhecidas foi a do **WRU Searching Selector**. Esse sistema poderia buscar em 25000 resumos de artigos em fitas de papel com 5 buscas booleanas concorrentes, sendo a semente do primeiro serviço pago de busca bibliográfica (Harman, 2019). Um artigo interessante de H. Mitchell (1953) faz cálculos de quantos documentos poderiam ser processados em uma busca que é um “E” de várias palavras chave codificadas em números e prevê que a resposta poderia ser dada em quatro horas para um milhão de documentos com um UNIVAC Fac-Tronic, que possuía uma memória de “12.000 caracteres de sete pulsos”, que supomos ser 7-bits (Remington Rand, 1953).

Versões similares dessa história inicial da Recuperação da Informação pode ser encontrada em Harman (2019) e também em Sanderson e Croft (2012).

13.2 Modelo Booleano

O Modelo Booleano (Baeza-Yates e Ribeiro-Neto, 2011) é baseado na Teoria dos Conjuntos e na Lógica Booleana. Ele parte da ideia que as consultas q são expressões booleanas, onde cada termo representa se uma palavra-chave está ou não presente na representação do documento, ou na representação da consulta. Uma vantagem do Modelo Booleano é sua semântica precisa, o que permite também a definição formal.

Seja um conjunto de termos $K = k_1, k_2, \dots, k_T$ e um conjunto de representações de documentos $D = d_1, d_2, \dots, d_N, d_i \in \wp(K)$, uma consulta Q tem a forma de uma expressão booleana formada com os termos k_j e os operadores E , OU e NO , onde k_j indica que o termo k_j deve pertencer a representação do documento desejado, e $NO(k_j)$ indica que o termo não deve pertencer ao documento (Baeza-Yates e Ribeiro-Neto, 2011).

Consultas típicas então seriam “FLUMINENSE”, “FLUMINENSE E FLAMENGO” ou “FLUMINENSE OU TRICOLOR”. Uma consulta mais complexa poderia ser “(FLUMINENSE OU TRICOLOR) E (FLAMENGO E RUBRO-NEGRO) E NÃO(MARACANÃ)”.

A resposta a uma consulta $R(Q)$ é a lista de documentos que atende aos requisitos da consulta. Ou seja, a resposta é $\{x | x \in D \wedge R(x) = 1\}$. Dessa forma os documentos são relevantes ou não, não havendo um grau de relevância.

Um maneira de entender um sistema baseado no Modelo Booleano é que ele faz um mapeamento de um conjunto de documento $D = d_1, d_2, \dots, d_N$ em um conjunto de vetores em $\{0,1\}^T$, onde T é a quantidade de termos. Dessa maneira, cada documento pode ser visto como descrito por um vetor de 0s e 1s de tamanho fixo. Isso é equivalente a representar uma consulta como a Forma Normal Disjuntiva (Baeza-Yates e Ribeiro-Neto, 2011), da seguinte maneira, dada uma consulta q :

$$\begin{aligned} q &= k_a \wedge (k_b \vee \neg k_c) \\ q_{dnf} &= (k_a \wedge k_b \wedge k_c) \vee (k_a \wedge k_b \wedge \neg k_c) \vee (k_a \wedge \neg k_b \wedge \neg k_c) \end{aligned}$$

E cada termo k_i pode ser representado como um vetor, no caso com 3 dimensões apenas:

$$q_{dbf} = (1,1,1) \vee (1,1,0) \vee (1,0,0)$$

onde cada q_{cc} , como $(1,0,0)$ é um componente conjuntivo.

Além disso, considera-se que existe uma função $g_i(x)$, que retorna o peso associado ao termo k_i em x , que pode ser um documento ou uma consulta. No modelo booleano, $g_i(x) \in \{0,1\}$ (Baeza-Yates e Ribeiro-Neto, 2011).

A similaridade entre a consulta q e um documento d_j é:

$$sim(d_j, q) = \begin{cases} 1, & \text{if } \exists \vec{q}_{cc} | (\vec{q}_{cc} \in \vec{q}_{fnd}) \wedge (\forall k_i, g_i(\vec{d}_j) = g_i(\vec{q}_{cc})) \\ 0, & \text{otherwise} \end{cases} \quad (13.1)$$

O que podemos compreender dessa formalização é que o sistema vai buscar documentos que possuam uma representação exatamente igual a de um dos componentes conjuntivos.

13.3 Usando a Lista Invertida

Uma maneira de implementar o Modelo Booleano é pelo uso da Lista Invertida (Salton, 1989). Como visto no Seção 6.1, uma lista invertida é, de uma forma bem geral, uma estrutura de dados onde as partes permitem recuperar o todo. É chamada invertida porque normalmente do todo se recuperam as partes.

Então se um documento d_i é descrito pelas palavras-chave $k_i^1, k_i^2, \dots, k_i^n$, a lista invertida permitirá que, conhecido um k_i^j , o documento d_i seja encontrado.

Uma lista-invertida pode ser implementada como uma lista encadeada, porém em Python e em memória, a maneira mais simples é implementá-la como um *dictionary*.

Assim, supondo que temos um conjunto de 3 documentos d_1, d_2, d_3 , onde $d_1 = k_1, k_2$, $d_2 = k_1, k_3$ e $d_3 = k_2, k_3$, a lista-invertida poderia ser representada como: `li = { 'k1': set('d1', 'd2'), 'k2': set('d1', 'd3'), 'k3': set('d2', 'd3') }`.

Essa lista invertida é construída com conjuntos (`sets`) para que seja possível aproveitar as funções de união e interseção. Outras representações são possíveis e a escolha depende de fatores como a quantidade de documentos, quantidade de palavras nos documentos, tamanho da memória, etc.

Teoricamente, processo básico de construção das listas invertidas partiria da Matriz de Incidência ou Matriz Termo-Documento, que indica, para cada termo, qual o seu peso em cada documento, sendo que no caso do Modelo Booleano, esse peso é apenas 0 ou 1, caso não apareça ou apareça no documento, respectivamente. Na prática as listas são construídas diretamente e a matriz termo documento é só uma abstração.

Seja um conjunto de documentos com a Matriz de Incidência apresentada na Tabela 13.1. A matriz apresenta 4 termos e 5 documentos, sendo que o documento d_1 e d_5 possuem a mesma representação, o que significa que possuem os mesmos termos, mas não necessariamente são iguais. É óbvio que documentos iguais possuem os mesmos termos, mas o inverso não é verdade. Por exemplo, as sentenças “O Fluminense ganhou do Flamengo” e “O Flamengo ganhou do Fluminense” possuem os mesmos termos, mas tem significados opostos. Quando a coleção de termos é controlada, é comum haver muitos documentos que possuem exatamente os mesmos termos, como livros sobre os mesmos assuntos em uma biblioteca. Quando se usa a indexação por texto completo, isso é bem mais raro, mesmo com técnicas de confluência, como o stemming.

Tabela 13.1: Matriz Termo-Documento, ou Matriz de Incidência, simples.

Termos	Documentos				
	d_1	d_2	d_3	d_4	d_5
k_1	1	1	0	0	1
k_2	0	1	1	0	0
k_3	1	0	1	0	1
k_4	0	0	0	1	0

13.4 Consultas booleanas

Uma consulta típica a um sistema de palavras-chaves é a consulta booleana. Esse tipo de consulta foi, e ainda é, muito usado em sistemas de biblioteca e permite operações rápidas.

Usando o exemplo da Tabela 13.1, suponha a consulta $(k_1 \wedge k_2) \vee k_4$. Esse modelo leva ao dbf $(1,1,0,0) \vee (1,1,0,1) \vee (1,1,1,1) \vee (1,0,0,1) \vee (1,0,1,1) \vee (0,1,0,1) \vee (0,1,1,1) \vee (0,0,0,1) \vee (0,0,1,1)$. Esses vetores devem ser comparados com as colunas da matriz. A resposta é d_2, d_4 .

13.5 Simulando o Modelo Booleano

O Programa 13.1 simula um corpus indexado por uma lista invertida, com capacidade de fazer operações de interseção e união, dadas as palavras chave.

Programa 13.1: Programa simples para fazer consultas do tipo E e OU a uma lista-invertida

```

1 class Corpus:
2
3     def __init__(self, dict):
4         self.dict = dict.copy()
5
6     def processa_e(self, chaves, current = None):
7         if not current:
8             current = set()
9
10        for k in chaves:
11            if current:
12                current &= self.dict[k]
13            else:
14                current = self.dict[k].copy()
15        return current
16
17    def processa_ou(self, chaves, current = None):
18        if not current:
19            current = set()
20        for k in chaves:
21            current |= self.dict[k]
22        return current
23
24

```

```

25 if __name__ == '__main__':
26
27     corpus = Corpus({ 'k1' : {'d1', 'd2', 'd5'}, 'k2' : {'d2', 'd3'},
28                       'k3' : {'d1', 'd3', 'd5'}, 'k4' : {'d4'} }
29     )
30
31     # (k1 e k2) ou k4
32     exp1 = corpus.processa_e(['k1', 'k2'])
33     corpus.processa_ou(['k4'], exp1)
34     print(exp1)

```

Saída do Programa 13.1

```
1 {'d2', 'd4'}
```

13.6 Estratégias para busca em longas listas invertidas

Uma consulta booleana é feita na forma de operações lógicas que o usuário pode organizar de várias formas. Cabe ao sistema de consultas otimizar a execução das mesmas em uma ordem ótima. Para isso, algumas propriedades podem ser aproveitadas, como o fato da interseção de um conjunto ser sempre menor ou igual ao conjunto. Christopher D. Manning, Raghavan e Schütze (2009a) sugerem pelos menos três práticas:

- iniciar pelos “E”, que diminuem o tamanho dos resultados intermediários;
- iniciar pelas chaves com menos documentos;
- usar skip pointers;

13.7 Exemplos com o Modelo Booleano

Para exemplificar o funcionamento do Modelo Booleano, é apresentada uma implementação simples em memória construída em Python, com apoio de textos selecionados do corpus Gutenberg do NLTK. Os textos estão em inglês, sendo: Sense and Sensibility de Jane Austen; Julius Caesar, Hamlet e Macbeth, de Shakespeare; Moby Dick, de Herman Melville e Alice in Wonderland, de Lewis Carroll. Nos exemplos usaremos termos da língua inglesa, mas preferencialmente nomes de nomes de personagens que sabemos estar, ou não, nas obras: Hamlet, Macbeth, Caesar, Antony, Cleopatra, John, Ishamel, Elinor, Marianne e Alice.

Deve ficar claro que todos esses programas se aproveitam de estruturas e funções de Python que funcionam bem em memória, porém não escalam para o mundo real, onde a quantidade de documentos varia de milhares em um biblioteca para milhões na Internet. Um sistema real, em vez de um dicionário, teria que utilizar a memória em disco e uma estrutura de dados de arquivos.

O programa da Listagem 13.2 que prepara na memória os seis documentos e dez termos do nosso exemplo.

Programa 13.2: Preparação do Exemplo.

```

1 import nltk
2 nltk.download('gutenberg')

```

```

3 docs=[0]*6 # Vetor de Documentos
4
5 # Escolhe 6 documentos entre os fornecidos
6 docs[0]=nltk.corpus.gutenberg.words('austen-sense.txt')
7 docs[1]=nltk.corpus.gutenberg.words('shakespeare-caesar.txt')
8 docs[2]=nltk.corpus.gutenberg.words('shakespeare-hamlet.txt')
9 docs[3]=nltk.corpus.gutenberg.words('shakespeare-macbeth.txt')
10 docs[4]=nltk.corpus.gutenberg.words('melville-moby_dick.txt')
11 docs[5]=nltk.corpus.gutenberg.words('carroll-alice.txt')
12
13 # Define os termos
14 words = ["Hamlet",
15         "Macbeth",
16         "Caesar",
17         "Antony",
18         "Cleopatra",
19         "John",
20         "Ishmael",
21         "Elinor",
22         "Marianne",
23         "Alice"]
24
25 words.sort()
26
27
28 mtd = [0]*len(words) # Matriz Termo Documentos
29 for i in range(len(words)):
30     mtd[i] = [0]*len(docs)

```

O programa da Listagem 13.3 é uma abordagem simples feita em Python para gerar um matriz termo documento, com a especificidade que a palavra sendo indexada foi escolhida antes e está listada na variável `words`.

Programa 13.3: Cálculo da uma matriz termo documento.

```

1 # Verifica se os termos estão presentes nos documentos
2 # e preenche a matriz termo documentos
3
4 for docn in range(len(docs)):
5     for wordn in range(len(words)):
6         if words[wordn] in docs[docn]:
7             mtd[wordn][docn]=1
8
9 # imprime para verificação
10
11 print("    DOCS→ ", "0 1 2 3 4 5")
12 for i in range(len(words)):
13     print("%10s  "%words[i], end=' ')
14     for j in range(len(docs)):
15         print(mtd[i][j], end=' ')

```

16 **print()**

O resultado principal da execução desse programa é a matriz termo documento:

	Resultado da execução do Programa 13.3						
1	DOCS->	0	1	2	3	4	5
2	Alice	0	0	0	0	0	1
3	Antony	0	1	0	0	1	0
4	Caesar	0	1	1	1	1	0
5	Cleopatra	0	0	0	0	1	0
6	Elinor	1	0	0	0	0	0
7	Hamlet	1	0	1	0	0	0
8	Ishmael	0	0	0	0	1	0
9	John	1	0	0	0	1	0
10	Macbeth	0	0	0	1	0	0
11	Marianne	1	0	0	0	0	0

Nas coleções no mundo real, a matriz termo documento é muito grande e esparsa. Isso implica que em vez de representa-la como matriz ela é normalmente representada como uma lista invertida.

O programa de Listagem 13.3 utiliza de algumas facilidades permitidas por tudo caber em memória e algumas funções já estarem implementadas em Python. Um programa mais parecido com um programa de indexação, ainda todo em memória, usaria o laço da Listagem 13.4, onde é necessário passar pelo documento palavra por palavra para criar a lista invertida.

Programa 13.4: Leitura de um documento palavra a palavra e registro dos documentos que possuem os termos índices em uma “lista invertida” construída por meio de um dicionário.

```

1 lis_inver = {}
2
3 for docn in range(len(docs)):
4     for term in docs[docn]:
5         if term in words:
6             if docn not in lis_inver.get(term, []):
7                 lis_inver[term] = lis_inver.get(term, [])+[docn]
8
9 for key in lis_inver:
10    print("%10s ->%key, end=" )
11    print(lis_inver[key])

```

O resultado do programa da Listagem 13.4 é:

	Resultado da execução do Programa 13.4					
1	John	->	[0,	4]		
2	Elinor	->	[0]			
3	Marianne	->	[0]			
4	Hamlet	->	[0,	2]		
5	Caesar	->	[1,	2,	3,	4]
6	Antony	->	[1,	4]		
7	Macbeth	->	[3]			
8	Ishmael	->	[4]			

```

9 Cleopatra -> [4]
10 Alice -> [5]

```

Em nossos exemplos, como em muitos sistemas reais, não é possível aplicar o NÃO em uma só palavra, sendo ele um operador binário que equivale a diferença de conjuntos.

Para realizar as funções lógicas E, OU e NÃO, são necessárias funções específicas se queremos fazer o processo nós mesmos. Considerando que as listas estão ordenadas pelo número do arquivo na Listagem 13.5 são apresentadas: a função `pandq`, que calcula a conjunção de duas listas; a função `pnotb`, que calcula a lista de elementos de `a` que não estão em `b`, e a função `porb`, que calcula a lista da união das listas `a` e `b`.

Programa 13.5: Calcula operadores E, OU e NÃO para duas listas.

```

1 def pandq(p,q):
2     ans = []
3     while p and q:
4         if p[0] == q[0]:
5             ans.append(p[0])
6             p = p[1:]
7         elif p[0]<q[0]:
8             p = p[1:]
9         else:
10            q = q[1:]
11    return ans
12
13 def pnotq(p,q):
14     ans = []
15     while p and q:
16         if p[0] == q[0]:
17             p = p[1:]
18             q = q[1:]
19         elif p[0]<q[0]:
20             ans.append(p[0])
21             p = p[1:]
22         else:
23             q = q[1:]
24     if p:
25         ans.extend(p)
26    return ans
27
28 def porq(p,q):
29     ans = []
30     while p and q:
31         if p[0] == q[0]:
32             ans.append(p[0])
33             p = p[1:]
34             q = q[1:]
35         elif p[0]<q[0]:
36             ans.append(p[0])

```

```

37     p = p[1:]
38 else:
39     ans.append(q[0])
40     q = q[1:]
41 if p:
42     ans.extend(p)
43 if q:
44     ans.extend(q)
45 return ans

```

Python fornece funções mais eficientes para todas essas opções, mas o objetivo do código apresentado neste livro é demonstrar como os algoritmos são utilizados, mesmo que algumas necessidades de sistemas reais sejam deixadas de fora.

Um maneira fácil de interpretar uma expressão booleana tradicional, na notação infixa é transformá-la para uma notação pós-fixa que pode ser imediatamente executada. Nesse caso uma expressão lógica ($A \text{ NOT } B \text{ E } C \text{ NOT } (D \text{ OU } \text{NOT}(F))$) é traduzida para a lista ['A', 'B', 'NOT', 'C', 'D', 'F', 'NOT', 'OU', 'NOT', 'E']. O Programa da Listagem 13.6 faz isso.

Programa 13.6: Transforma um expressão lógica infixa em pós-fixa.

```

1 prioridade = {"(" : 10 , ")" : 10 , "NÃO" : 9 ,
2                 "NOT" : 9 , "E" : 8 , "AND" : 8,
3                 "OU" : 7 , "OR" : 7 , "+" : 7 ,
4                 "." : 8 , "~" : 9,
5                 "*" : 8 , "_" : 9}
6
7
8 def i2p(expr):
9     pilha = []
10    saída = []
11    # conserta espaços faltando
12    entrada = expr.replace("(", " ( ").replace(")", " ) ").split()
13    for token in entrada:
14        pt = prioridade.get(token,0)
15        if pt == 0: # if operando, output
16            saída.append(token)
17        elif token == "(":
18            pilha.append(token)
19        elif token == ")":
20            while pilha and pilha[-1] != "(":
21                saída.append(pilha.pop())
22            pilha.pop()
23        elif not pilha: # if stack empty
24            pilha.append(token)
25        elif pt>prioridade.get(pilha[-1],0):
26            pilha.append(token)
27        elif "(" in pilha[-1]: # if top stack is (
28            pilha.append(token)
29        else:

```

```

30     while pilha and prioridade[pilha[-1]] >= pt and pilha[-1] != "(":
31         saida.append(pilha.pop())
32         pilha.append(token)
33     while pilha:
34         saida.append(pilha.pop())
35     return(saida)

```

O programa da Listagem 13.7 processa uma expressão de consulta lógica de memória sem usar estruturas de índice, já que Python possui o operador `in`.

Programa 13.7: Processa uma busca booleana na coleção em memória usando o documento original.

```

1 def processa_booleano(expr,docs):
2     consulta = i2p(expr)
3     pilha = []
4     ops = prioridade.keys()
5     for tk in consulta:
6         if tk not in ops:
7             found = set()
8             for docn in range(len(docs)):
9                 if tk in docs[docn]:
10                     found.add(docn)
11             pilha.append(found)
12         if tk in ["E","*",".", "AND"]:
13             pilha.append(pilha.pop().intersection(pilha.pop()))
14         if tk in ["OU","+","OR"]:
15             pilha.append(pilha.pop().union(pilha.pop()))
16         if tk in ["NOT","~","-", "NÃO"]:
17             buf = pilha.pop()
18             pilha.append(pilha.pop().difference(buf))
19     return(pilha.pop())

```

Já o programa da Listagem 13.8 processa uma expressão de consulta lógica de memória usando as estruturas de índice e comparando-as por meio das funções mostradas na Listagem 13.5.

Programa 13.8: Processa uma busca booleana na coleção em memória usando índice.

```

1 def processa_booleano(expr,docs):
2     consulta = i2p(expr)
3     pilha = []
4     ops = prioridade.keys()
5     for tk in consulta:
6         if tk not in ops:
7             found = set()
8             for docn in range(len(docs)):
9                 if tk in docs[docn]:
10                     found.add(docn)
11             pilha.append(found)
12         if tk in ["E","*",".", "AND"]:
13             pilha.append(pilha.pop().intersection(pilha.pop()))
14         if tk in ["OU","+","OR"]:

```

```

15     pilha.append(pilha.pop().union(pilha.pop()))
16     if tk in ["NOT", "~", "-", "NÃO"]:
17         buf = pilha.pop()
18         pilha.append(pilha.pop().difference(buf))
19     return(pilha.pop())

```

13.8 Similaridade de Documentos no Modelo Booleano

É possível calcular a **similaridade** entre documentos no Modelo Booleano a partir de várias fórmulas de similaridade usadas em diferentes abordagens. Aqui serão apresentadas medidas ligadas a similaridade entre conjuntos, entendendo que, segundo o Modelo Booleano, cada documento é visto como um conjunto de termos.

Provavelmente, a medida de similaridade mais usada é a **Similaridade de Jaccard**, que é aplicada a conjuntos de elementos. Segundo essa medida, a similaridade entre dois conjuntos pode ser calculada pela razão entre a interseção e a união dos dois conjuntos.

$$S_{\text{Jaccard}}(A, B) = \frac{\|A \cap B\|}{\|A \cup B\|} \quad (\text{Similaridade de Jaccard})$$

13.9 Extensões ao Modelo Booleano

Principalmente no início das pesquisas em Recuperação da Informação, foram propostas muitas extensões ao Modelo Booleano. Dois fatores contribuem para isso: sua formulação matemática permite fazer extensões usando conceitos matemáticos, como os modelos baseados em Lógica Fuzzy (Baeza-Yates e Ribeiro-Neto, 2011), e o fato de sua resposta não possuir um *ranking*, o que motiva a busca por extensões nessa direção.

Salton e McGill (1983) resume as extensões do Modelo Booleano em duas características, uma função que fornece um peso para o documento diferente de zeros e uns, e um forma de processamento das expressões de consulta, basicamente formadas de e, ou e não, que trate desses valores.

A Lógica Fuzzy é uma escolha quase que óbvia. Já em 1979, Radecki (1979) descreve teoricamente um modelo de recuperação baseado na Lógica Fuzzy. Mais tarde, Fox (1983), que foi orientado por Salton, apresenta um modelo Fuzzy baseado em P-Normas, que mais tarde compara com o uso do mínimo e o máximo como operadores fuzzy (Fox e Sharan, 1986).

13.10 Uso Atual

Atualmente a visão booleana, ou modelos que podem ser chamados de booleanos estendidos, ainda é usada em alguns sistemas, principalmente aqueles que usam consultas meio das expressões de consulta com operadores como E e OU.

Simultaneamente, com a separação do algoritmo de ordenação (*ranking*) da busca propriamente dita, representações booleanas, isto é, a contabilização apenas da presença ou não dos termos no documento, são usadas, com operadores explícitos ou implícitos, para gerar um conjunto de documentos a ser ordenado. Isso é o que acontecia, por exemplo, no algoritmo *Pagerank* original Page et al., 1999.

13.11 Exercícios

Exercício 13.1:

Implemente um programa similar ao implementado nesse capítulo, porém usando estruturas persistentes, como as citadas na Seção 6.1.

Exercício 13.2:

Escolha uma coleção de documentos e construa uma matriz de similaridade entre eles.

CAPÍTULO 14

REPRESENTAÇÃO VETORIAL

No Capítulo 13, foi mostrada uma alternativa de representação para o Modelo Booleano onde os documentos são mapeados em vetores cujos pesos são sempre 0 ou 1. A consulta é feita por meio de uma expressão lógica dos termos, e a conclusão é que o elemento pertence ou não a resposta.

Essa estrutura não permitia o cálculo de um ranking, isto é, todas as respostas aparecem com igual importância, vários autores buscaram alternativas, como o Modelo Probabilístico(S. E. Robertson, 1977; van Rijsbergen, 1979), Modelos Booleanos Estendidos, o uso de Lógica Fuzzy, e outros(Baeza-Yates e Ribeiro-Neto, 2011).

Um dos modelos de maior sucesso foi o Modelo Vetorial(Salton e McGill, 1983), que até hoje influencia toda a área de Recuperação da Informação e de Aprendizado de Máquina, dada a representação que propõe para o documento.

No **Modelo Vetorial**, cada documento é representado por um vetor em que cada elemento representa o peso de um termo específico no documento, sendo que a noção de peso não é definida *a priori*, sendo exigido apenas um número real, e existindo várias formas de calculá-lo. Também não é feita nenhuma exigência na ordem dos termos, porém é necessário que todos os elementos de cada posição sejam associados aos mesmos termos em todos os vetores. Os termos definem as dimensões do espaço vetorial onde estão os documentos. O peso é uma medida da importância do termo no documento. A ideia básica é que vetor de termos de um documento indique sobre o que ele fala (*aboutness*).

Nesse caso, são calculadas similaridades entre documentos ou entre documentos e consultas por meio de funções entre dois vetores, que são propostas para atender certas propriedades.

Valores simples para o peso usados em cada elemento do vetor são a própria frequência ou a frequência relativa do termo, porém as medidas de maior sucesso partiram do conceito conhecido como tf-idf, *term frequency-inverse document frequency*, que considera a importância do termo dentro do documento e a importância do termo como discriminador de documentos na coleção. Assim, um termo que aparece muito em um documento tem o *tf*, de ***term frequency*** alto, e um documento que aparecem em todos os documentos não tem importância, tendo um *idf*, de ***inverse document frequency*** baixo.

A Figura 14.1 mostra a ideia básica do Modelo Vetorial, usando apenas duas dimensões: os termos presentes no corpus definem um espaço vetorial, e indicam o significado dos documento, e da consulta.

Além disso, o modelo considera que a similaridade entre a consulta e os documentos da coleção pode ser calculada dentro desse espaço, por exemplo como o inverso da distância, e mais frequentemente como uma distância angular entre os vetores, possuindo gradações.

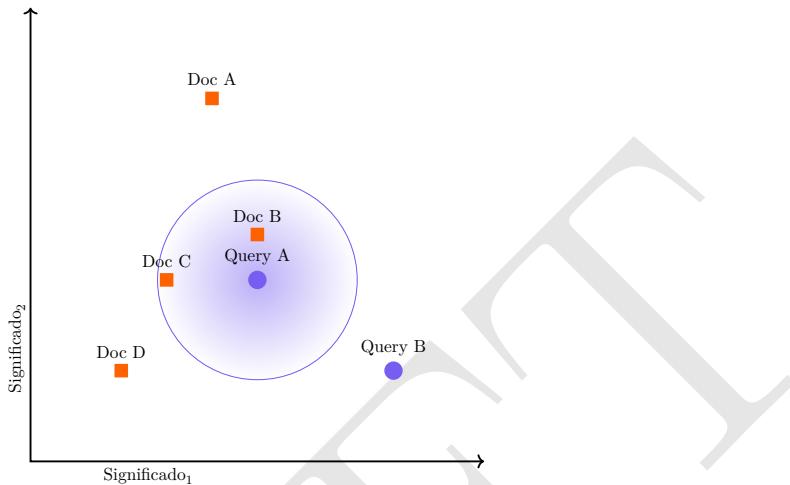


Figura 14.1: A ideia básica do Modelo Vetorial é que os termos do documento, e da consulta, indicam seu significado, e que a similaridade entre a consulta e os documentos da coleção pode ser calculada em um espaço vetorial.

Algumas premissas do Modelo Vetorial original não são verdade. A principal é que as dimensões dos vetores são ortonormais. Como as dimensões são geradas pelos termos, isso significa que os termos seriam independentes entre si, ou seja, em última instância, que o fato de um termo aparecer em um documento é independente do fato de qualquer outro termo aparecer no documento. Isso, obviamente, não é verdade. Alguns termos costumam aparecer mais frequentemente juntos, o que é chamado de co-ocorrência, do que outros. Isso foi tratado em modelos subsequentes, mas sem impactar fortemente o resultados já de boa qualidade do modelo vetorial.

A primeira vista, a similaridade no Modelo Vetorial poderia ser vista como o inverso da distância entre os vetores de frequência. Isso, porém, acaba por não funcionar bem, porque o que importa no significado de um documento não deve ser o peso de um outro termo, mas sim a relação entre os pesos do termo. Ao usar o inverso da distância, poderiam ocorrer situações como a demonstrada na Figura 14.2, onde uma consulta q seria mais próxima dos documentos d_2 e d_3 , porém a proporção entre os pesos dos termos mais semelhante a q é a do documento d_1 .

Por isso, a medida mais simples e razoável para o Modelo Vetorial é relacionada ao ângulo entre dois vetores, e na prática se usa o coseno do ângulo, que é máximo quando o ângulo é zero. Na Figura 14.3, é possível ver que $\beta < \theta < \alpha$.

Atualmente, fórmulas mais complexas são utilizadas de forma a melhorar o desempenho do sistema, por exemplo corrigindo os efeitos do tamanho do documento, ou usando conceitos dos modelos probabilísticos.

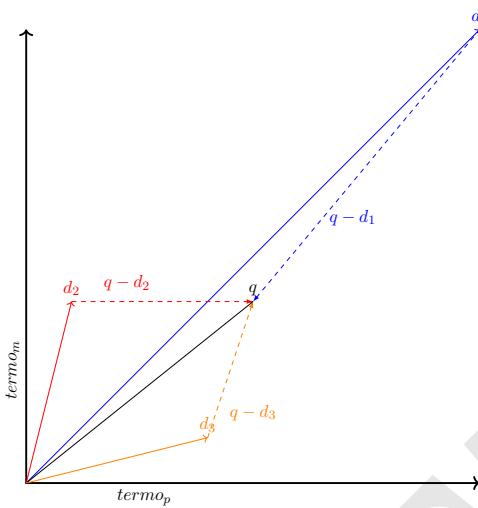


Figura 14.2: Exemplo de problema com o uso do tamanho dos vetores

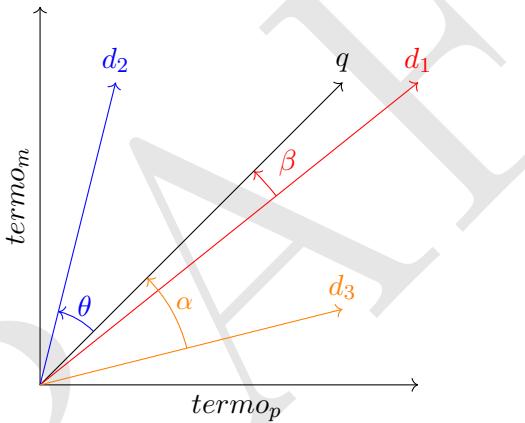


Figura 14.3: Usando o cosseno dos vetores.

14.1 Definições do Modelo Vetorial

Seja um conjunto D de N documentos d_i , ou corpus, $D = \{d_1, d_2, \dots, d_N\}$, compostos de T termos k_j onde cada documento é representado na forma $d_i = (w_{i1}, w_{i2}, \dots, w_{iT})$, onde w_{ij} representa o peso do j -ésimo termo no i -ésimo documento i . No Modelo Vetorial, $w_{ij} > 0$ sempre que $k_i \in d_j$ (Baeza-Yates e Ribeiro-Neto, 2011).

Para cada termo k_j é associado um vetor unitário $\vec{k}_j = (0, 0, \dots, 1, \dots, 0)$ onde todos os valores são 0, menos o da j -ésima posição, que é 1¹.

Por definição, esses vetores são considerados ortonormais, o que supõe que os termos ocorrem de forma independente nos documentos. Esta é uma decisão importante, que levará a tratamentos mais sofisticados que corrigem essa suposição.

As seguintes afirmações são válidas:

- Ambos documento $\vec{d}_i = (w_{i0}, w_{i1}, \dots, w_{iT})$ e consulta $\vec{q} = (w_{q0}, w_{q1}, \dots, w_{qT})$ são elementos do espaço euclidiano E^n , $n = T + 1$.

¹Também é possível representar um documento como um dicionário: $d = \{t_0 : w_{d0}, t_1 : w_{d1}, \dots, t_T : w_{dT}\}$.

id	Documento	Ordem dos Termos	Vetores de Frequência
1	O Pedro viu o homem.	o, pedro, viu, ho- mem	1: (2,1,1,1)
2	O homem era velho.	o, pedro, viu, homem, era, velho	1: (2,1,1,1,0,0) 2: (1,0,0,1,1,1)
3	Pedro tinha um binóculo.	o, pedro, viu, homem, era, velho, tinha, um, binóculo	1: (2,1,1,1,0,0,0,0,0) 2: (1,0,0,1,1,1,0,0,0) 3: (0,1,0,0,0,0,1,1,1)

Tabela 14.1: Exemplo de criação passo a passo dos vetores de documentos de uma coleção.

- Cada termo t_i corresponde a um vetor base \vec{e} do espaço E^n .
- O grau de relevância r de um documento D representado por um vetor \vec{w} relativo a consulta Q representada pelo vetor \vec{q} é baseado no produto escalar $\langle \vec{w}, \vec{q} \rangle$, e em geral pode ser considerado uma medida de similaridade $\text{sim}(\vec{d}, \vec{q})$.
- Se $\langle \vec{w}, \vec{q} \rangle = 0$, então o documento não é relevante e não é recuperado.
- Se $\langle \vec{w}, \vec{q} \rangle \neq 0$, então o documento é considerado relevante e é recuperado.
- Seja $\langle \vec{w}_i, \vec{q} \rangle \neq 0$, $i = 1, \dots, m$ correspondente aos documentos d_i . Então os documentos d_1, \dots, d_m são usados para construir a lista de acertos: d_1, \dots, d_m são ordenados descendente mente em ordem do grau de relevância e são apresentados ao usuário nessa ordem.

(Baeza-Yates e Ribeiro-Neto, 2011)

É importante notar que o Modelo Vetorial não supõe ordem nas palavras, e a ordem escolhida no vetor pode ser arbitrária (Baeza-Yates e Ribeiro-Neto, 2011). O importante é que todos os vetores tenham a mesma ordem. Normalmente, em *full-text retrieval*, os termos são adicionados a estrutura de dados usada para a criação dos vetores na primeira vez em que são encontrados. A Tabela 14.1 mostra um exemplo onde 3 documentos, frases no caso, são inseridos em ordem e o vetor vai mudando de tamanho para considerar as novas palavras. Cada palavra é inserida na ordem em que é encontrada. Na prática são criadas estruturas de dados temporárias, como uma árvore binária ou uma tabela de dispersão, para acelerar a busca pelos termos na contagem, e depois processá-las para gerar os vetores. Tipicamente os vetores são esparsos.

14.2 Medindo o peso de um termo em um documento

O peso de um termo em um documento pode ser medido de várias formas, que foram evoluindo ao longo do tempo. Nesta seção será discutida a função **tf-idf**, uma das mais usadas para essa finalidade..

14.2.1 TF, a frequência do termo

A frequência de uma palavra em um documento é normalmente associada a representatividade dela como indicadora do assunto do mesmo documento (Salton e McGill, 1983). Sua importância foi relatada já em Luhn, 1958, que inclusive discute como as palavras mais e menos frequentes em um documento ou coleção são menos significativas na sua representação.

A medida de frequência é conhecida como **term frequency**, e pela sigla **TF** e possui várias versões (Baeza-Yates e Ribeiro-Neto, 2011; Roelleke, 2013), sendo algumas apresentadas a seguir.

Sendo D uma coleção de documento d_j e K uma coleção de termos k_i , $f_{i,j}$ é a frequência do termo k_i no documento d_j , ou seja, a quantidade de vezes que o termo aparece no documento. Um termo pode ser uma palavra ou outro item único que pode ser identificado no texto, possivelmente após algum pré-processamento como o uso de um *stemmer*, podendo também incluir bigramas ou outras composições de palavras.

$$TF_{\text{total}}(i,j) = f_{i,j} \quad (14.1)$$

$$TF_{\text{soma}}(i,j) = \frac{f_{i,j}}{\sum_k f_{k,j}} \quad (14.2)$$

$$TF_{\text{max}}(i,j) = \frac{f_{i,j}}{\max_k f_{k,j}} \quad (14.3)$$

$$TF_{\log}(i,j) = \begin{cases} \log(1 + f_{i,j}) & , \text{ if } f_{i,j} > 0 \\ 0 & , \text{ caso contrário} \end{cases} \quad (14.4)$$

$$TF_K(i,j) = \frac{f_{i,j}}{f_{i,j} + K} \quad (14.5)$$

$$TF_{\text{dupla}}(i,j) = K + K \frac{f_{i,j}}{\max_i f_{i,j}} \quad (14.6)$$

A Equação 14.1, TF_{total} , é uma indicação simples da contagem do termo no documento. Roelleke (2013) afirma que escolher TF_{total} ou TF_{soma} corresponde a assumir independência dos termos. Sua principal vantagem é ser uma medida muito simples, porém ela é influenciada pelo tamanho do documento. TF_{soma} e TF_{max} são normalizações simples, a primeira pelo total de termos do documento, a segunda pela frequência do termo máximo no documento. A primeira acaba resultando em valores muito pequenos para documentos grandes, levando a um viés que favorece documentos pequenos. TF_{max} e TF_K mitigam esse viés. Mais tarde será mostrada a função BM25, com bases probabilísticas, que é ainda melhor.

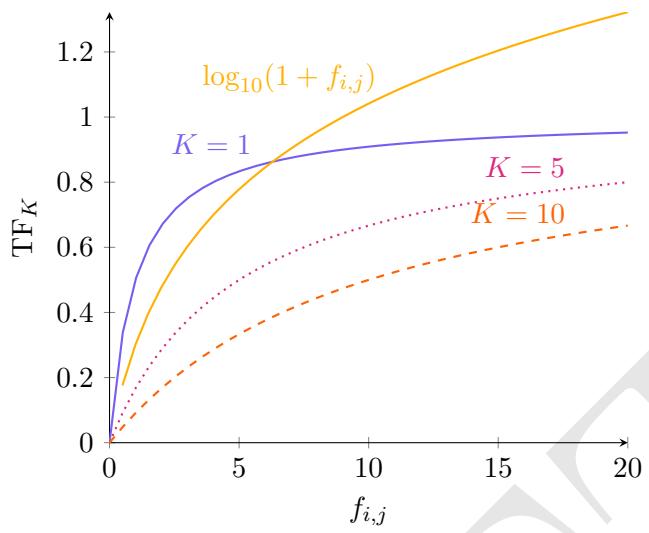
Já a fórmula para TF_{\log} tenta evitar que as palavras com frequência muito alta dominem totalmente o significado do documento, passando a controlar a ordem de grandeza². A fórmula TF_K , conhecida como TF fracionário, considera uma dependência maior do que a logaritmica e garante um $TF < 1$. Enquanto TF_K dá a k -ésima ocorrência de um termo um impacto de $1/k^2$, o impacto em TF_{\log} é de $1/k$ (Roelleke, 2013). A Figura 14.4 compara as duas opções para diferentes valores de K . Ambas operações, a aplicação do log e a inclusão do termo K , tem como resultado prático o “achatamento” da imagem da função original, $y = f(x)$, onde são aplicadas, i.e., as diferenças no eixo y são menores para as mesmas diferenças no eixo x .

14.2.2 IDF, a frequência inversa nos documentos

O fato de um termo ser importante em um documento não ajuda a definir a relevância se ele aparecer em todos os documentos. Isso foi reconhecido já inicialmente, como por Luhn (1958) e tratado especificamente por Sparck Jones, 1972 que trouxe o conceito da Frequência Inversa nos Documentos, conhecida pela sigla *IDF*.

Seja N o número de documentos e n_i o número de documentos com o termo i , várias fórmulas também foram propostas na literatura, como (Baeza-Yates e Ribeiro-Neto, 2011; Roelleke, 2013):

²A base do logaritmo é pouco importante, já que $\log_{\text{base}}(x) = \frac{\ln(x)}{\ln(\text{base})}$, e a diferença é uma constante

Figura 14.4: Comparaçao das curvas para \log_{10} e variações de K para TF_K .

$$IDF_{\text{unário}} = 1 \quad (14.7)$$

$$IDF_{\text{total}}(i) = \frac{1}{n_i} \quad (14.8)$$

$$IDF_{\text{soma}}(i) = \log \frac{N}{n_i} \quad (14.9)$$

$$IDF_{\text{suave}}(i) = \log 1 + \frac{N}{n_i} \quad (14.10)$$

$$IDF_{\text{max}}(i) = \log 1 + \frac{\max_i n_i}{n_i} \quad (14.11)$$

$$IDF_{\text{BIR}}(i) = \log \frac{N - n_i}{n_i} \quad (14.12)$$

$$IDF_{\text{BIRsuave}}(i) = \log \frac{N - n_i + K}{n_i + K} \quad (14.13)$$

14.2.3 TF-IDF

Combinando TF e IDF vários esquemas podem ser obtidos, como por exemplo (Baeza-Yates e Ribeiro-Neto, 2011; Roelleke, 2013; Sparck Jones, 1972):

$$TF - IDF = \frac{f_{i,j}}{n_i} \quad (14.14)$$

$$TF - IDF = \frac{f_{i,j}}{\sum_k f_{k,j}} \log \frac{N}{n_i} \quad (14.15)$$

$$TF-IDF(i,j) = (1 + \log f_{i,j} \times \log \frac{N}{n_i}) \quad (14.16)$$

14.3 Medindo a similaridade entre documentos

Existem muitas medidas de similaridade entre os documentos, dado que eles estão mapeados em um espaço vetorial. O coseno do ângulo é uma medida de similaridade normalizada, que desconsidera o tamanho de documentos. Dados dois vetores d_1 e d_2 , o coseno é medido por:

$$\cos(\angle(d_1, d_2)) = \frac{d_1 \cdot d_2}{\|w_1\|_2 \|w_2\|_2} = \frac{\sum_{i=0}^T w_{1i} \times w_{2i}}{\sum_{i=0}^T w_{1i}^2 \sum_{i=0}^T w_{2i}^2} \quad (14.17)$$

Quando d_2 é uma consulta, a fórmula do *score* de relevância a ser usado na ordenação da resposta à consulta é:

$$r(d, q) = \cos(\angle(d, q)) = \frac{d \cdot q}{\|d\|_2 \|q\|_2} = \frac{\sum_{i=0}^T w_{di} \times w_{qi}}{\sum_{i=0}^T w_{di}^2 \sum_{i=0}^T w_{qi}^2} \quad (14.18)$$

14.3.1 Introduzindo a qualidade

Christopher D. Manning, Raghavan e Schütze (2009a) sugere a introdução de um fator de qualidade estático para cada documento $g(d)$, o que permite modificar a equação de similaridade para fornecer um novo *score* a ser usado na resposta:

$$s(d, q) = g(d) + r(d, q) \quad (14.19)$$

14.4 Uso atual do Modelo Vetorial

A representação vetorial é usada em muitos sistemas, como o Lucene e o sistema textual do MySQL, porém as funções de similaridade variam entre as propostas originalmente para ele e, principalmente, a BM25, criada no contexto dos modelos probabilísticos.

14.5 Extensões ao Modelo Vetorial

Uma das questões mais importantes relacionadas ao Modelo Vetorial é a suposição da independência dos termos, o que permite considerar os vetores dos termos formarem uma base ortonormal para o espaço de documentos.

Essa suposição também significa dizer que cada termo no vetor é totalmente diferente dos outros semanticamente, o que não é verdade. Várias estratégias podem ser usadas para calcular uma semelhança entre os termos e considerar isso na busca e na ordenação da resposta.

Uma opção é sofisticar o Modelo Vetorial, como no Modelo Vetorial Generalizado (S. K. M. Wong, Ziarko e P. C. N. Wong, 1985), ou usar técnicas de redução de dimensionalidade, em busca de trabalhar com conceitos em vez de termo, como no LSI (Furnas et al., 1988). Essas soluções, porém, podem tornar o custo computacional mais alto, o que leva a soluções alternativas, normalmente baseadas em *Thesaurus* e expansão de consulta.

Outra questão importante no modelo vetorial são as consultas muito simples fornecidas pelos usuários. Nesse caso, novamente a expansão de consulta pode ser a solução, mas também o feedback de relevância.

14.5.1 Modelo vetorial generalizado

Uma estratégia para tratar a questão das palavras não serem realmente independentes entre si foi proposta por S. K. M. Wong, Ziarko e P. C. N. Wong (1985): o Modelo Vetorial Generalizado. Nesse modelo é criada uma base ortonormal, formada a partir de *minterms*, que são vetores que mostram todas as combinações possíveis de termos. Assim um corpus com T termos possuir 2^T *minterms* possíveis. Só são usados, na verdade, os *minterms* que aparecem na coleção. Além disso, cada *minterm* pode ser visto como um “documento template”, indicando a co-ocorrência de todos aqueles termos. O modelo é mais caro computacionalmente e não ficou demonstrado ser melhor que o Vetorial.

14.6 Expansão de consulta

Expandir a consulta significa buscar, por algum mecanismo, outras palavras que podem auxiliar encontrar documentos relevantes, melhorando a consulta original. A ideia foi proposta inicialmente por Maron e Kuhns (1960), sendo motivada pelo fato dos usuários se limitarem a algumas poucas palavras em suas consultas.

O uso de poucas palavras na busca é reconhecido já há bastante tempo. Um estudo recente que analisou 160 milhões de consultas populares identificou que 51% delas usava apenas 1 ou 2 palavras, enquanto que 43,3% usava entre 3 e 5 palavras. Contando só as consultas únicas, e não o volume total, 67% eram de 3 ou 5 palavras e 15,9% de 1 ou 2 palavras. Com poucas palavras há uma definição muito ampla do que realmente é a necessidade de informação do usuário (Oberstein, 2022), havendo problemas de precisão.

Expandir significa então colocar novos termos na consulta de forma automática, possivelmente usando pesos diferentes dos pesos dados aos termos da consulta original.

Por exemplo, uma consulta apenas uma palavra, “fotos gato”, poderia ser expandida para “foto imagem fotografia gato bichano felino”. Também são possíveis utilizar outras formas de escrita, expandir siglas, e o uso de palavras fortemente associadas. Assim, “Livro 007” pode ser expandido para “livro 007 James Bond Ian Fleming”.

Uma técnica básica de expansão é simplesmente pegar os termos da consulta e buscar em um thesauros termos que sejam sinônimos, ou possuam outra relação como hiperonímia, com o termo original. É possível uso de um dicionário ou thesaurus externo, como o Wordnet (Fellbaum, 1998; G. A. Miller, 1995; Princeton University, 2010), descrito na Seção 21.1, ou o uso de um *thesaurus* criado automaticamente a partir da própria coleção e a relação estatística da frequência em que os termos aparecem nos mesmos documentos. Maron e Kuhns (1960) chamam isso de relações semânticas, paradigmática, ou seja, baseada nos significados do texto, e estatísticas, baseadas na frequência, ou ainda sintagmáticas (Rapp, 2002).

Como é difícil que uma ou duas palavras não gerem resultado algum quando a busca é feita na internet, técnicas desse tipo eram mais adequadas quando a coleção era bem menor, e podem ainda ser adequadas em ambientes de coleções restritas, como base de documentos de empresas. Elas ainda podem ajudar a encontrar documentos mais variados, porém têm o risco de diminuir a precisão.

14.6.1 Feedback de relevância

O Feedback de Relevância também é uma técnica de expansão de consulta, onde, a partir do primeiro resultado apresentado ao usuário, se espera uma resposta do usuário de que documentos

foram relevantes e que documentos não foram relevantes. A partir dessa resposta é possível construir uma nova consulta, gerada a partir dos documentos relevantes. Nesse caso, a consulta é expandida com os termos encontrados nos documentos retornados e aprovados pelo usuário.

Na prática, o **feedback de relevância explícito** a partir da resposta real do usuário, não é usado, mas sim através de estratégias como acompanhar a navegação do usuário, o que Baeza-Yates e Ribeiro-Neto (2011) chama **feedback de relevância através de clicks**. Uma variação importante é o **feedback de pseudo-relavância, ou feedback de relevância implícito**, onde já se faz a expansão a partir do resultado da busca, sem a intervenção do usuário.

Rocchio (1965) propôs o algoritmo inicial de feedback de relevância a partir da ideia que um vetor de consulta ideal maximizaria a diferença média entre os documentos relevantes e os documentos irrelevantes. Esse vetor ideal q_{ideal} , teoricamente, faria C ser máximo em:

$$C = \frac{1}{|R|} \sum_{\vec{d}_i \in R} \text{sim}(\vec{q}_{\text{ideal}}, \vec{d}_i) - \frac{1}{|\bar{R}|} \sum_{\vec{d}_i \in \bar{R}} \text{sim}(\vec{q}_{\text{ideal}}, \vec{d}_i) \quad (14.20)$$

onde $R \cup \bar{R}$ é o conjunto total de documentos, sendo R os documentos relevantes e \bar{R} os não relevantes. Para a similaridade do cosseno, isso resulta em:

$$\vec{q}_{\text{ideal}} = \frac{1}{|R|} \sum_{\vec{d}_i \in R} \frac{\vec{d}_i}{|\vec{d}_i|} - \frac{1}{|\bar{R}|} \sum_{\vec{d}_i \in \bar{R}} \frac{\vec{d}_i}{|\vec{d}_i|} \quad (14.21)$$

Para fazer o feedback de relevância então é possível fazer uma estimativa a partir dos subconjuntos de documentos recuperados relevantes r e não relevantes \bar{r} (Christopher D. Manning, Raghavan e Schütze, 2009a; Rocchio, 1965) e usá-la para modificar a consulta original. Rocchio (1965) sugere mas não usa pesos, mas Salton (1971a) descreva a fórmula mais geral, que ficou conhecida como “O algoritmo de Rocchio para o feedback de relevância” (Christopher D. Manning, Raghavan e Schütze, 2009a), que inclui não considerar termos negativos:

$$\vec{Q}_{fr} = \alpha \vec{Q}_0 + \beta \frac{1}{|r|} \sum_{\vec{d}_i \in r} \vec{d}_i - \gamma \frac{1}{|\bar{r}|} \sum_{\vec{d}_i \in \bar{r}} \vec{d}_i \quad (14.22)$$

$$\vec{Q}'_{fr} = \begin{cases} \vec{Q}_{fr} & , \text{for } \vec{Q}_{fr} \geq 0 \\ 0 & , \text{for } \vec{Q}_{fr} < 0 \end{cases} \quad (14.23)$$

Segundo Christopher D. Manning, Raghavan e Schütze (2009a), valores razoáveis para as constantes são $\alpha = 1$, $\beta = 0.75$ e $\gamma = 0.15$, sendo que em alguns sistemas, $\gamma = 0$.

A interpretação geométrica desta fórmula é clara: dado um vetor de consulta, respostas relevantes e não relevantes, um novo vetor de consulta é construído aproximando a consulta do “documento relevante médio” e afastando do “documento não relevante médio”. A Figura 14.5 representa essa operação.

```

1 import numpy as np
2
3 def rochho_algorithm(query_vector, relevant_docs_vectors,
4     ↪ irrelevant_docs_vectors, alpha=1, beta=0.75, gamma=0.15):
5     """
6         Implements the Rocchio algorithm for relevance feedback.

```

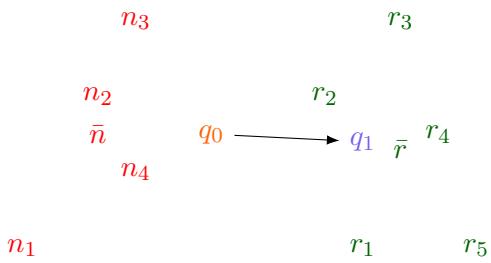


Figura 14.5: Interpretação geométrica do feedback de relevância.

```

7     Parameters:
8         query_vector (numpy array): The original query vector.
9         relevant_docs_vectors (list of numpy arrays): The vectors of the documents
10            ↪ that were marked as relevant.
11         irrelevant_docs_vectors (list of numpy arrays): The vectors of the
12            ↪ documents that were marked as irrelevant.
13         alpha (float): The weight of the original query vector.
14         beta (float): The weight of the relevant documents vectors.
15         gamma (float): The weight of the irrelevant documents vectors.
16
17     Returns:
18         A numpy array representing the new query vector.
19         """
20         relevant_docs_matrix = np.stack(relevant_docs_vectors, axis=0) if len(
21             ↪ relevant_docs_vectors) > 0 else np.zeros((0, len(query_vector)))
22         irrelevant_docs_matrix = np.stack(irrelevant_docs_vectors, axis=0) if len(
23             ↪ irrelevant_docs_vectors) > 0 else np.zeros((0, len(query_vector)))
24
25         new_query_vector = alpha * query_vector + beta * (relevant_docs_matrix.mean
26             ↪ (axis=0) if relevant_docs_matrix.shape[0] > 0 else np.zeros(len(
27                 ↪ query_vector))) - gamma * (irrelevant_docs_matrix.mean(axis=0) if
28                 ↪ irrelevant_docs_matrix.shape[0] > 0 else np.zeros(len(query_vector)))
29
30     return new_query_vector

```

14.6.2 Feedback de Pseudo-Relevância

No **feedback de pseudo-relevância**, existem estratégias conhecidas como **análise local** ou **análise global**. Na análise local, os resultados são agrupados e geram novas consultas, já na análise global são usados todos os documentos da coleção e um *thesaurus*. Uma descrição mais detalhada pode ser obtida em Baeza-Yates e Ribeiro-Neto (2011).

Lavrenko e Croft (2001) propuseram o uso de modelos de linguagem baseados em relevância, onde o peso de um termo w em uma expansão de consulta para uma consulta $Q = \{q_1, q_2, \dots, q_k\}$ e D um conjunto de documentos pseudo-relevantes é computado como:

$$FW_{rm}(w) = P(w|R) = \sum_{d \in D} P(w|D) \prod_{q \in Q} P(q|D) \quad (14.24)$$

No algoritmos RM3 (Abdul-Jaleel et al., 2004), descrito por Roy, S. Bhatia e Mitra, 2019, esse mesmo peso é estimado como:

$$FW_{rm3}(w) = P(w|R') = \alpha P(w|Q) + (1 - \alpha)P(w|R) \quad (14.25)$$

onde $\alpha \in [0,1]$ e $P(w|Q)$ é calculado usando uma estimativa por máxima verossimilhança *maximum likelihood estimation*.

Roy, S. Bhatia e Mitra (2019) propõem então três melhorias ao RM3. Sendo C um conjunto de documentos, R um sub conjunto de C com documentos pseudo-relevantes, w um termo de expansão, $r(w)$ uma medida de raridade do termo w , as três melhorias são descritas a seguir. O melhor desempenho foi obtido por $RM3_3^+$.

$$FW_{rm3_1^+}(w) = \alpha P(w|Q) + (1 - \alpha)NFW(w) \quad (14.26)$$

$$FW_{rm3_2^+}(w) \approx \alpha P(w|Q) \times r(w) + (1 - \alpha)P(w|R) \times r(w) \quad (14.27)$$

$$FW_{rm3_3^+}(w) = \alpha P(w|Q) + (1 - \alpha)P(w|R) \quad (14.28)$$

Para $RM3_1^+$ Como $r(w)$ foi usada o IDF. São escolhidos então os N termos com maior FW , e os pesos de FW normalizados sobre a somam resultando em NFW , que são então combinados com o modelo de verossimilhança máxima Q com um fator α . $RM3_1^+$ resulta em um peso baseado na combinação entre um modelo de linguagem da consulta e a razão entre a probabilidade de relação de RM e um modelo de não relevância.

CAPÍTULO 15

REDUÇÃO DE DIMENSIONALIDADE E EMBEDDINGS

Um dos problemas típicos com algoritmos clássicos de aprendizado de máquina para o uso em texto é a enorme quantidade de atributos a serem usados, na prática ao menos um por palavra chave. Por isso, antes de usar esses algoritmos, é razoável usar algum método de dimensão de dimensionalidade. Esse grande número de palavras chave também tem impacto na busca, inclusive por questões semânticas, pois as palavras podem significar a mesma coisa e várias palavras podem representar a mesma coisa, sem contar as semelhanças e similaridades.

A ideia principal da maioria dos métodos de redução de dimensionalidade é remover de antemão os termos que não ajudam, ou mesmo atrapalham, a busca ou o aprendizado de máquina. Isso pode ser visto como uma remoção de ruído, considerando o texto um sinal onde os termos de “frequência mais alta” (aqui a frequência mais alta indica na verdade a raridade da palavra, não a contagem) são removidos para evitar que esse ruído de alta frequência atrapalhe a classificação.

A primeira abordagem, usando o modelo vetorial, é usar apenas os termos mais frequentes, possivelmente depois de retirar as stopwords. Na prática, essa técnica funciona razoavelmente já a partir de 1024 termos com o aprendizado de máquina, mas apresenta um resultado ruim para a busca.

Uma proposta que teve sucesso é fazer uma representação dos documentos usando um algoritmo conhecido como LSI, tratado neste capítulo. LSI segue os mesmos princípios do que chamamos hoje de *embeddings*.

15.1 Embeddings

Embeddings são representações de objetos dentro de espaços matemáticos, mantendo propriedades estruturais essenciais (J. A. Lee e Verleysen, 2007). Mas precisamente, um **embedding** é uma função que representa um objeto matemática dentro de um espaço, preservando certas propriedades estruturais (J. A. Lee e Verleysen, 2007) A motivação para o uso de embeddings deriva do interesse de representar objetos de alta dimensionalidade em espaços mais tratáveis, mantendo relações e estruturas inerentes aos dados originais.

No modelo vetorial, documentos são representados por termos em um espaço de alta dimensão, dezenas de milhares de termos. Maior ainda é a quantidade de variáveis que representam os termos, na ordem de milhões (ou mais) documentos. Sabemos porém que nem todas as dimensões tem a mesma

importância. Na representação de documentos, podemos eliminar as *stop words* sem muita perda na busca, e também costumamos reduzir todas as conjugações de um verbo para a sua forma infinitiva, ou ainda o *stem* dessa forma. Já na representação de termos, podemos imaginar que para representar o significado de uma palavra em uma certa época, apenas documentos daquela época devem ser usados na representação. O desafio é lidar com a vasta dimensionalidade e encontrar uma representação eficaz e significativa.

A relevância das variáveis é crítica na determinação de quais dimensões são preservadas nos *embeddings*. Mesmo que todas as variáveis sejam relevantes, elas podem conter informação redundante, havendo dependência entre elas. A dependência entre as variáveis, indicada pela correlação, sugere uma estrutura subjacente que deve ser capturada pela representação, isto é, o *embedding*.

Na Redução de Dimensionalidade, vista de forma geral, como na Análise de Componentes Principais, procura eliminar dependências e reduzir o espaço sem perder informações essenciais. Já na Separação de Variáveis Latentes, o foco está na descoberta de variáveis latentes que explicam a origem dos dados observados, mesmo que não possam ser levantadas diretamente.

15.1.1 Topologia e Embeddings

A topologia estuda propriedades de objetos que são preservadas sob deformações, torções e estiramentos, mas não cortes. Quando uma ou mais variáveis que descrevem um objeto dependem uma das outras, sua distribuição conjunta, ou melhor, o suporte dessa distribuição conjunta, não cobre todo o espaço. Essa dependência induz uma estrutura na distribuição, que forma um *locus*, que pode ser visto como um objeto no espaço. Isso é crucial para entender como *embeddings* podem manter a “forma” dos dados mesmo quando representados em dimensões reduzidas (J. A. Lee e Verleysen, 2007).

15.1.2 Definição Formal

Em um conjunto Y , uma topologia T é o conjunto dos subconjuntos de Y com as seguintes propriedades:

$$\emptyset \in T \tag{15.1}$$

$$Y \in T \tag{15.2}$$

$$A \in T \wedge B \in T \longrightarrow A \cap B \in T \tag{15.3}$$

$$A \in T \wedge B \in T \longrightarrow A \cup B \in T \tag{15.4}$$

Um espaço X está *embedded* em um outro espaço Y quando as propriedades de Y restringidas a X são as mesmas de X .

15.1.3 Buscando as Variáveis Latentes

Quando consideramos a estrutura dos dados representados por funções como o Rolo Suiço (Figura 15.1), estamos diante do desafio de identificar as variáveis latentes que descrevem a forma intrínseca dos dados. Estas são as dimensões essenciais que capturam a variação significativa dos dados. No contexto do *Swiss roll*, as variáveis latentes correspondem aos parâmetros que geram a espiral bidimensional antes de ser enrolada na terceira dimensão.

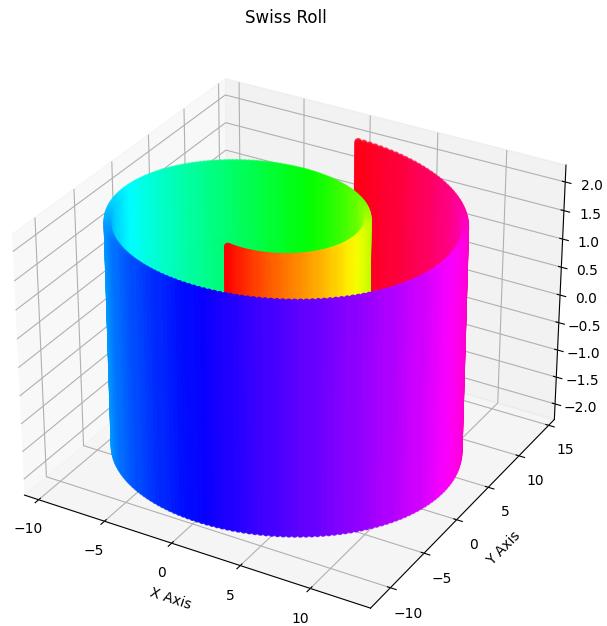


Figura 15.1: O Rolo Suiço é um exemplo clássico usado para ilustrar desafios em técnicas de embedding e redução de dimensionalidade. Ele representa uma superfície bidimensional que foi enrolada em uma terceira dimensão de maneira que não pode ser desdobrada em um plano sem distorção. O objetivo de usar o Rolo Suiço em técnicas de embedding é demonstrar a capacidade de um algoritmo de desenrolar superfícies curvas enquanto preserva a estrutura local dos dados.

Fonte: elaboração do autor, baseada em (J. A. Lee e Verleysen, 2007)

15.1.4 Variáveis Latentes e a Estrutura dos Dados

A descoberta de variáveis latentes é fundamental para entender a dimensionalidade intrínseca dos dados. Se a dimensão intrínseca for igual à dimensão do espaço de embedding, então os dados são considerados sem uma estrutura topológica complexa. Por outro lado, se a dimensão intrínseca for menor, isso indica que os dados possuem uma estrutura que não preenche completamente o espaço de embedding. Nesse caso, o processo de re-embedding, ou seja, a representação dos dados em um espaço de menor dimensão, pode revelar a estrutura topológica subjacente de forma mais clara.

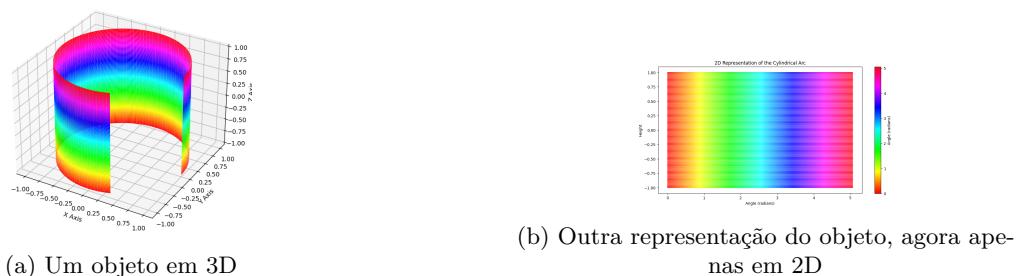


Figura 15.2: Exemplificando o uso de uma transformação de um objeto representado em um espaço tridimensional para um espaço bidimensional.

15.2 Indexação por Semântica Latente

A Indexação por **Semântica Latente** (**LSI**¹), e a Análise por Semântica Latente, (**LSA**²) são baseadas em um método vetorial de representação de texto que visa, por meio de manipulações algébricas na Matriz Termo Documento, encontrar tópicos que representem o significado dos documentos a partir de combinações dos termos (S. K. M. Wong, Ziarko e P. C. N. Wong, 1985).

LSI pode ser visto como uma manipulação algébrica, ou uma operação de redução de dimensionalidade, que busca encontrar uma aproximação da matriz de termos e documento de forma que:

- sejam filtrados os ruídos causado por termos,
- seja economizado espaço na busca,
- seja economizado tempo na busca.

Seus defensores, porém, dizem que LSA é uma teoria do significado, além de um método computacional, pois permite que computadores façam tarefas típicas dos seres humanos. LSA aprende o significado de palavras por meio da exposição a conjuntos muito grande de textos (Thomas K. Landauer, s.d.), de forma similar ao que chamamos hoje de *embeddings*.

As matrizes termo-documento são muito esparsas, enquanto as matrizes geradas pela LSI são densas, logo, para que haja economia de tempo e espaço, é necessário que a dimensão da matriz aproximada seja muito menor que a da matriz original.

De forma abstrata, o objeto é mudar de um mapeamento termo-documento, como na Figura 15.3a, para uma mapeamento dos termos para conceitos e dos conceitos para documentos, como na Figura 15.3b. Esse mapeamento é feito por uma operação da Álgebra Linear conhecida como Decomposição em Valores Singulares.

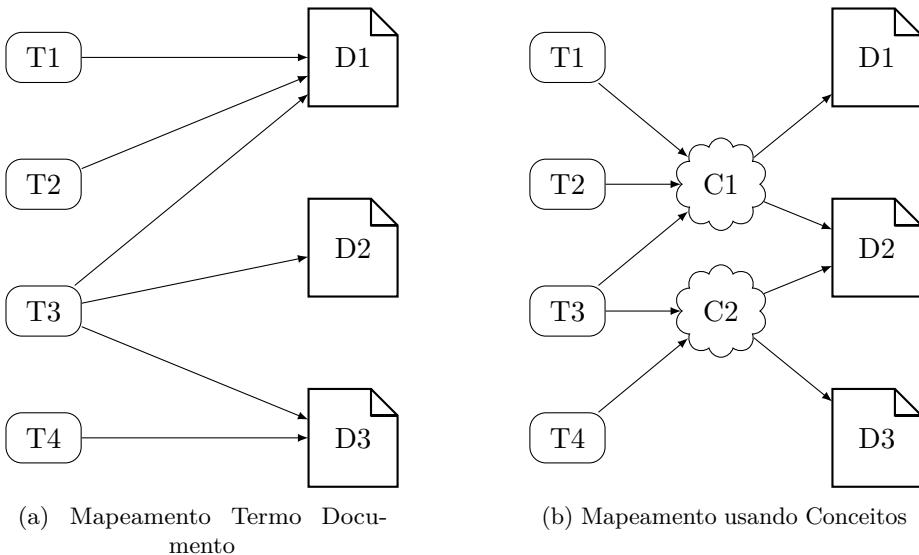


Figura 15.3: Visão abstrata do LSI.

¹Do inglês *Latent Semantic Indexing*

²Do inglês *Latent Semantic Analysis*

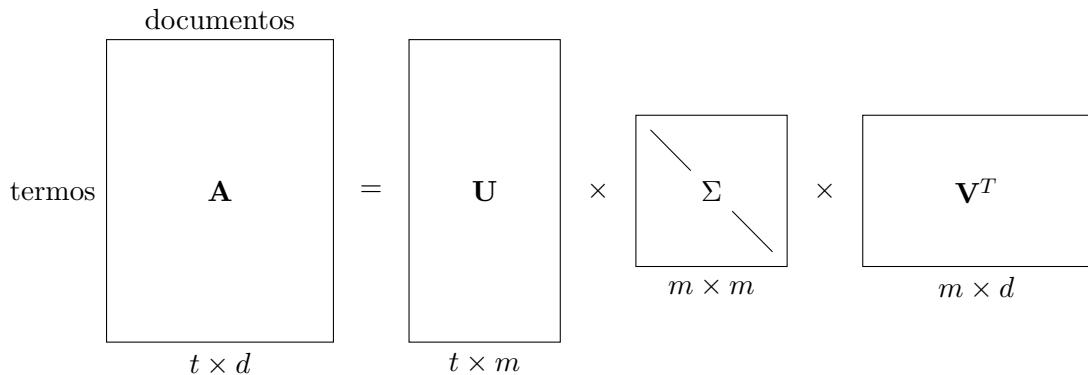
15.2.1 A Decomposição em Valores Singulares

Uma **decomposição em valores singulares** de uma matriz $\mathbf{A}_{t \times d}$, conhecida pela sigla **SVD**, é definida pela equação (Cohen, 2022; Golub e Van Loan, 2013):

$$\mathbf{A}_{t \times d} = \mathbf{U}_{t \times m} \boldsymbol{\Sigma}_{m \times m} \mathbf{V}^T_{m \times d} = \sum_{i=1}^r \sigma_i \mathbf{u}_i \mathbf{v}^T \quad (15.5)$$

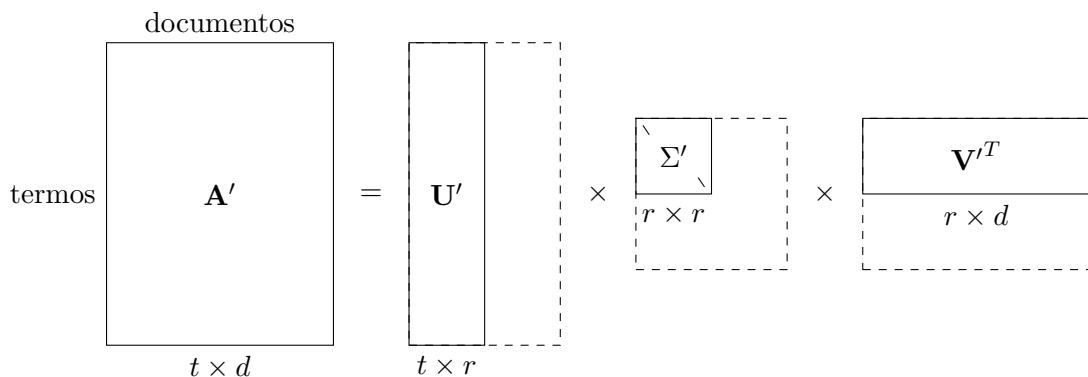
Onde $m = \min(t, d)$, \mathbf{U} e \mathbf{V} possuem colunas ortonormais e $\boldsymbol{\Sigma}$ é uma matriz diagonal onde $\sigma_{ii} > \sigma_{(i+1,i+1)}$, isto é, seus valores estão ordenados do maior para o menor, sendo todos positivos³. A matriz $\boldsymbol{\Sigma}$ é chamada de matriz dos valores singulares. Essa decomposição está esquematizada na Figura 15.4. O valor de m é o número de vetores linearmente independentes em \mathbf{A} , isto é, o posto da matriz (ou *rank*).

Figura 15.4: Esquema do SVD. Fonte: (Cohen, 2022)



Uma aproximação de $\mathbf{A}_{t \times d}$ pode ser obtida escolhendo os r maiores valores singulares de **Sigma** e cortando as colunas e linhas equivalentes em \mathbf{U} e \mathbf{V}^T , como esquematizado na Figura 15.5.

Figura 15.5: Esquema do SVD após a redução de *rank*. Fonte: (Cohen, 2022)



A ideia aqui é que reduzindo a dimensão de A , se remove também ruído, ou seja, informações não realmente relevantes ao significado de cada termo. A criação de A' captura a estrutura semântica da relação entre termos e documentos.

Também é importante notar que há uma interpretação geométrica para os valores singulares. Os valores singulares de uma matriz são as raízes quadradas dos autovalores da matriz produto $\mathbf{A} \cdot \mathbf{A}^*$, onde

³Se não for exigida a ordenação, existem várias decomposições, porém, sendo exigida, só existe uma.

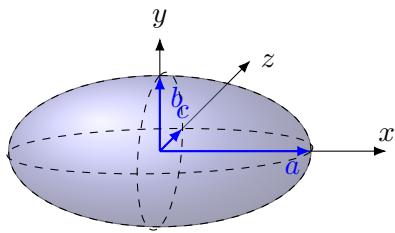


Figura 15.6: Representação de um elipsóide em 3 dimensões mostrando a diferença de tamanho do objeto entre os eixos.

\mathbf{A}^* é a matriz adjunta de \mathbf{A} . Geometricamente, esses valores singulares correspondem às distâncias dos eixos principais da elipse de espalhamento da transformação linear representada pela matriz \mathbf{A} . Assim, os valores singulares representam o grau de esticamento ou encolhimento que ocorre em cada direção nos dados de entrada quando passam pela transformação linear gerada por \mathbf{A} .

Uma explicação visualmente mais simples é ver a matriz \mathbf{A} como um objeto em um espaço n-dimensional, os valores singulares são a magnitude dos eixos desse hiper-elipsóide, logo seus valores indicam as dimensões mais significativas. Pensando em um espaço tridimensional, é como caracteriza-se as dimensões mais importantes. Uma esfera vai ser projetada pela transformação linear em uma elipsóide. Por exemplo, pode-se pensar em uma caneta, descrita por 3 eixos, sendo que um é seu comprimento, muito maior que a largura e a altura. Nesse caso, o valor singular equivalente ao comprimento seria muito mais alto. A Figura 15.6 representa essa ideia.

O cálculo do SVD para uma matriz sai do escopo deste texto, sendo descrito em livros de algoritmos matemáticos ou Álgebra Linear. É difícil que um programador consiga implementar versões mais eficientes do algoritmo do que as disponíveis nas bibliotecas que atendem a Álgebra Linear. Neste texto, adotamos, como exemplo, a função fornecida pelo Numpy. Existem métodos exatos e aproximados de cálculo do SVD, além de métodos incrementais, como GHA, ou *Generalized Hebbian Algorithm* (Gorrell e Webb, 2005).

15.2.2 Quantas dimensões usar no LSI?

Uma questão importante é quantas dimensões usar no LSI. T. K. Landauer, Foltz e Laham (1988) fez uma análise baseada em acertos em perguntas sobre sinônimo no TOEFL e encontrou valores ótimos em torno de 300 dimensões. Kontostathis e Pottenger (2006), em outra análise propõe valores entre 150 e 500 para coleções “grandes” e entre 50 e 200 para coleções “menores”, porém hoje em dia todas as suas coleções seriam consideradas muito pequenas, apesar da maior coleção, com menos de 6000 documentos, já apresentar mais de 81.879 termos.

Thomas K. Landauer e Dumais (2008) sugere que 300 dimensões são normalmente quase-ótimas, e que um número entre 200 e 2000 será útil. Nesse mesmo artigo, eles também sugerem que não há necessidade de recalcular os vetores pelo menos até 20% de mudanças em um corpus inicial com mais de 20 mil passagens. Como tamanho de documento, defendem que passagens coerentes entre 50 a 100 palavras são unidades ótimas para o treinamento. Eles afirmam ainda que o LSI vem sendo usado com sucesso em coleções pequenas com resultados de precisão e revocação 30% maiores que outros métodos.

Certamente grandes índices genéricos, com milhões de documento e que atendem uma variedade muito grande de consultas precisarão de mais dimensões, porém coleções específicas, como as usadas em portais de empresas, podem ser satisfeitas com essas ordens de grandeza.

15.2.3 Exemplo de uso da LSI em Python

O exemplo que se segue é adaptado de um exemplo teórico de Grossman e Frieder (2004). Aqui é mostrada uma implementação em Python usando apenas o `numpy`, mas outros pacotes podem ser usados. Os Programas de 15.1 a 15.6 compõem na verdade um só programa.

Seja uma base composta de três documentos:

- d_1 – Carregamento de outro destruído em um fogo.
- d_2 – Entrega de prata chegou em um caminhão prata.
- d_3 – Carregamento de outro chegou em um caminhão.

Nessa base será feita uma consulta q : ouro prata caminhão.

O Programa 15.1 realiza a preparação dos dados para fazer uma Indexação por Semântica Latente. Apesar de existirem outras bibliotecas que possam ajudar, vamos iniciar apenas com a `numpy` (linha 1). A `base` é uma lista que contém três sentenças na forma de *strings*.

A LSI ainda é método de similaridade de vetores que representam os documentos e a consulta, onde os vetores são uma modificação dos vetores originais do Modelo Vetorial. Assim, nas linhas 11 e 12, é definida uma função de similaridade do cosseno.

Além disso, na linha 14, é feita uma tokenização de forma muito simples. Como os exemplos (artificiais) não possuem sinais de pontuação, foi usada apenas a função `split`. Deve-se notar que como `base` é uma lista de *strings*, então aplicamos a função `split` a todos os elementos da lista, por meio da função `map`.

Programa 15.1: Declarações

```

1 import numpy as np #
2 np.set_printoptions(precision=2)
3
4
5 base = ["Carregamento de ouro destruído em um fogo",
6         "Entrega de prata chegou em um caminhão prata",
7         "Carregamento de ouro chegou em um caminhão"]
8
9 q = ["ouro", "prata", "caminhão"]
10
11 def cos_sim(A,B): #
12     return A.dot(B)[0]/(np.linalg.norm(A)*np.linalg.norm(B)) #
13
14 base_tokens = map(str.split,base) #
```

Continuando, agora com o Programa 15.2, é necessário construir a matriz termo documento. Como estamos usando um biblioteca numérica, e para facilitar o mapeamento entre as palavras e os números de linha, criamos dois dicionários, um onde dado uma palavra se obtém um número, e outro inverso. Como é um problema simples, já sabemos o tamanho exato da matriz, que chamaremos de `A`, na linha 19.

Entre a linha 21 e a linha 28 é feito o preenchimento correto da matriz termo documento. Já nas linhas 30 a 32 é criado o vetor de consulta `Q`. Ambos são impressos a seguir.

Programa 15.2: Construindo Termo Documento

```

15 d = 0
16 k = 0
17 numero = {}
18 palavras = []
19 A = np.zeros((11,3),int)
20
21 for frase in base_tokens:
22     for token in frase:
23         if token not in numero:
24             numero[token] = k
25             palavras[k] = token
26             k += 1
27             A[numero[token],d] += 1
28     d += 1
29
30 Q = np.zeros((11,1),int)
31 for palavra in q:
32     Q[numero[palavra],0] = 1
33
34 print(A)
35 print(Q)

```

O SVD é feito no Programa 15.3, entre algumas impressões que mostram o que está sendo realizado. É importante chamar atenção para o fato que esse operação sofisticada é feita em apenas uma linha (44). Tendo em vista que a fórmula do SVD é $A = U\Sigma V^T$, a função svd do subpacote de álgebra linear responde com o V transposto. Para obter o V original da forma, é feita a operação de transposição na linha 45.

Programa 15.3: Fazendo o SVD

```

36 for i in range(len(A)):
37     print("{0:13} {1}".format(palavras[i],A[i]))
38
39 print("-"*30)
40
41 for i in range(len(Q)):
42     print("{0:13} {1}".format(palavras[i],Q[i]))
43
44 U, S, VT = np.linalg.svd(A,full_matrices=False) # svd!
45 V = VT.transpose() # transpõe VT para achar V
46
47 print("-"*15+ "U" + "-"*15)
48 print(U)
49 print("-"*15+ "S" + "-"*15)
50 print(S)
51 print("-"*15+ "V" + "-"*15)
52 print(VT)
53 print("-"*15+ "VT" + "-"*15)
54 print(V)

```

Feito o SVD, no Programa 15.4 é feita a diminuição dos ranks das matrizes, de forma a atender a aglutinação de palavras em conceitos proposta pelo SVD. É importante notar aqui que as operações podem ser realizadas apenas com operadores muito comuns em Python, sobre as matrizes retornadas pelo `svd`.

Na chamada a função `svd` foi usada a forma reduzida de U . Além disso, a resposta S não é uma matriz diagonal, mas um vetor de valores singulares. Para poder reconstruir A é necessário usar a função `diag`.

Programa 15.4: Diminuindo o rank

```

55 A_reconstructed = U @ np.diag(S) @ VT
56 print("—" * 15 + "Arec" + "—" * 15)
57 print(A_reconstructed)
58
59 m, n = A.shape
60 r = 2
61 U_r = U[:, :r]
62 VT_r = VT[:r, :]
63 V_r = VT_r.transpose()
64 S_r = S[:r]
65 A_r = U_r @ np.diag(S_r) @ VT_r
66 print("—" * 15 + "Ar" + "—" * 15)
67 print(A_r)
68
69 print("—" * 15 + "U_r" + "—" * 15)
70 print(U_r)
71 print("—" * 15 + "S_r" + "—" * 15)
72 print(S_r)
73 print("—" * 15 + "V_r" + "—" * 15)
74 print(V_r)
75 print("—" * 15 + "VT_r" + "—" * 15)
76 print(VT_r)

```

Para fazer a consulta é necessário fazer uma modificação no vetor de consulta para atender as novas dimensões das matrizes reduzidas, isso é feito com a fórmula $q_r = q^T U_r \Sigma_r^{-1}$, como pode ser visto no Programa 15.5.

No mesmo programa são então calculadas as similaridades. Segundo o exemplo, as similaridades estão ordenadas como $d_2 > d_3 > d_1$.

Programa 15.5: Fazendo a consulta

```

77 QLSI = Q.transpose() @ U_r @ np.linalg.inv(np.diag(S_r))
78 print("—" * 15 + "QLSI" + "—" * 15)
79 print(QLSI)
80
81 sim = [0, 0, 0]
82 for d in range(3):
83     sim[d] = cos_sim(QLSI, V_r[d])
84     print(f"Query {q} Document — {d} Similarity={sim[d]}")

```

A Programa 15.6 mostra todos os vetores reduzidos, em duas dimensões, e é possível verificar a ordenação das similaridades de forma visual. Esse gráfico foi gerado com o Figura 15.7

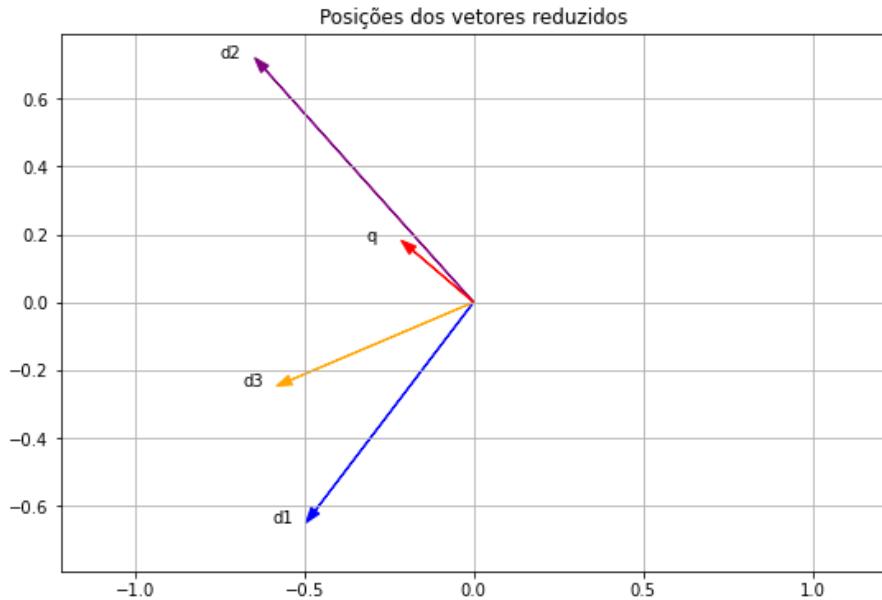


Figura 15.7: Vetores reduzidos do exemplo.

Programa 15.6: Plot dos vetores

```

85 import matplotlib.pyplot as plt
86
87 def plotvecs(M, labels=[], colors=[], do_axis=True, **kwargs):
88     rows, cols = M.shape
89     if not colors :
90         colors = ["black"]*rows
91
92     for i in range(rows):
93         xs = M[i,0]
94         ys = M[i,1]
95         plt.arrow(0,0,xs,ys,
96                   length_includes_head=True,
97                   head_width=.03,
98                   color=colors[i],**kwargs)
99         if labels:
100             plt.text(xs-.1,ys,labels[i])
101
102     if do_axis:
103         maxesx = 1.1*abs(M[:,0]).max()
104         maxesy = 1.1*abs(M[:,1]).max()
105         plt.axis('equal')
106         plt.xlim([-maxesx,maxesx])
107         plt.ylim([-maxesy,maxesy])
108         plt.grid(visible=True, which='major')
109     return plt

```

```

110
111
112 plotvecs(V_r,["d"+str(i+1) for i in range(len(V_r)) ],
113           ["azul","roxo","laranja"])
114 plotvecs(QLSI,[ "q"],[ "magenta"],do_axis=False)
115 plt.title("Posições dos vetores reduzidos")
116 plt.show()

```

A seguir, a saída da execução do programa completo.

Saída do programa completo de LSI

```

1 [[1 0 1]
2 [1 1 1]
3 [1 0 1]
4 [1 0 0]
5 [1 1 1]
6 [1 1 1]
7 [1 0 0]
8 [0 1 0]
9 [0 2 0]
10 [0 1 1]
11 [0 1 1]]
12 [[0]
13 [0]
14 [1]
15 [0]
16 [0]
17 [0]
18 [0]
19 [0]
20 [1]
21 [0]
22 [1]]
23 Carregamento [1 0 1]
24 de [1 1 1]
25 ouro [1 0 1]
26 destruído [1 0 0]
27 em [1 1 1]
28 um [1 1 1]
29 fogo [1 0 0]
30 Entrega [0 1 0]
31 prata [0 2 0]
32 chegou [0 1 1]
33 caminhão [0 1 1]
34 -----
35 Carregamento [0]
36 de [0]
37 ouro [1]
38 destruído [0]

```

```
39 em [0]
40 um [0]
41 fogo [0]
42 Entrega [0]
43 prata [1]
44 chegou [0]
45 caminhão [1]
46 -----U-----
47 [[-0.26 -0.38 -0.15]
48 [-0.42 -0.07 0.05]
49 [-0.26 -0.38 -0.15]
50 [-0.12 -0.27 0.45]
51 [-0.42 -0.07 0.05]
52 [-0.42 -0.07 0.05]
53 [-0.12 -0.27 0.45]
54 [-0.16 0.3 0.2 ]
55 [-0.32 0.61 0.4 ]
56 [-0.3 0.2 -0.41]
57 [-0.3 0.2 -0.41]]
58 -----S-----
59 [4.1 2.36 1.27]
60 -----V-----
61 [[-0.49 -0.65 -0.58]
62 [-0.65 0.72 -0.25]
63 [ 0.58 0.26 -0.77]]
64 -----VT-----
65 [[-0.49 -0.65 0.58]
66 [-0.65 0.72 0.26]
67 [-0.58 -0.25 -0.77]]
68 -----Arec-----
69 [[ 1.00e+00 -7.49e-16 1.00e+00]
70 [ 1.00e+00 1.00e+00 1.00e+00]
71 [ 1.00e+00 -7.15e-16 1.00e+00]
72 [ 1.00e+00 -5.55e-16 1.44e-15]
73 [ 1.00e+00 1.00e+00 1.00e+00]
74 [ 1.00e+00 1.00e+00 1.00e+00]
75 [ 1.00e+00 -5.55e-16 1.44e-15]
76 [ 7.22e-16 1.00e+00 -1.28e-15]
77 [ 1.44e-15 2.00e+00 -2.55e-15]
78 [ 7.77e-16 1.00e+00 1.00e+00]
79 [ 7.77e-16 1.00e+00 1.00e+00]]
80 -----Ar-----
81 [[ 1.11 0.05 0.85]
82 [ 0.97 0.99 1.05]
83 [ 1.11 0.05 0.85]
84 [ 0.67 -0.15 0.45]
85 [ 0.97 0.99 1.05]
86 [ 0.97 0.99 1.05]
```

```

87 [ 0.67 -0.15  0.45]
88 [-0.15  0.93  0.2 ]
89 [-0.3   1.87  0.4 ]
90 [ 0.3   1.13  0.6 ]
91 [ 0.3   1.13  0.6 ]]
92 -----U_r-----
93 [[-0.26 -0.38]
94 [-0.42 -0.07]
95 [-0.26 -0.38]
96 [-0.12 -0.27]
97 [-0.42 -0.07]
98 [-0.42 -0.07]
99 [-0.12 -0.27]
100 [-0.16  0.3 ]
101 [-0.32  0.61]
102 [-0.3   0.2 ]
103 [-0.3   0.2 ]]
104 -----S_r-----
105 [4.1  2.36]
106 -----V_r-----
107 [[-0.49 -0.65]
108 [-0.65  0.72]
109 [-0.58 -0.25]]
110 -----VT_r-----
111 [[-0.49 -0.65 -0.58]
112 [-0.65  0.72 -0.25]]
113 -----QLSI-----
114 [[-0.21  0.18]]
115 Query ['ouro', 'prata', 'caminhão'] Document - 0 Similarity=-0.05395084366642496
116 Query ['ouro', 'prata', 'caminhão'] Document - 1 Similarity=0.9909874267484721
117 Query ['ouro', 'prata', 'caminhão'] Document - 2 Similarity=0.4479594658282973

```

15.2.4 Considerações sobre o LSI

Como o LSI se inicia com a construção da matriz termo documento, ele necessita de uma coleção estática, não sendo originalmente adequado para coleções que evoluem rápido.

Apesar de funcionar melhor com mais termos na consulta, possivelmente por uma delimitação conceitual mais bem feita do documento desejado, o LSI normalmente retorna documentos mesmo quando a consulta no modelo vetorial não retorna nenhum, porque ele é capaz de retornar outros documentos que falam sobre o mesmo assunto que o termo único representa. Em contrapartida, ele também acaba retornando documentos para as várias acepções relativas ao termos presentes na consulta. Em geral, experimentos mostram uma melhoria tanto na revocação quanto na precisão, além de melhorias no ranking (Giesbers, Rusman e Bruggen, 2006; Yu, Maciej Ceglowski e Payne, 2002).

Yu, Maciej Ceglowski e Payne (2002) ainda afirmam que LSI pode ser usado como primeiro passo de uma operação de *feedback* de relevância, assistência a escrita de trabalhos de cadeira, ou correção, identificando tópicos não tratados pelos alunos, análise da coerência do texto (Foltz, Kintsch e T. K. Landauer, 1998) e filtragem da informação.

15.3 Modelo Baseado em Sinais

15.4 *Embeddings*

Um ***embedding*** é uma técnica que mapeia uma representação de alta dimensão em uma representação de dimensão mais baixa.

Como visto anteriormente, os vetores que representam os documentos são muito longos, na prática eles tem mais de 50 mil termos, porém são também esparsos. Para o cálculo de medidas de similaridade de termos, a representação do termo, a partir da transposição de uma matriz termo-documento, pode ter milhões de dimensões, sendo uma para cada documento. Uma alternativa é buscar um vetor menor, mais curto e mais denso. Também foi visto que o LSI faz isso de alguma forma. Os embeddings descritos nesta seção foram criados com essa função.

A busca por vetores curtos vem do fato que eles são melhores para o aprendizado de máquina. Além da questão da eficiência, o que é potencializado pelo fato de que alguns algoritmos são fortemente dependentes da quantidade de atributos, uma representação mais densa pode generalizar melhor conceitos do que contagens explícitas. Essa afirmativa, novamente, vai ao encontro dos resultados do LSI. A ideia é que vetores densos capturam melhor a sinonímia e outras características de similaridade, funcionando melhor quando o resultado final é analisado.

15.5 Histórico e Contexto

A tarefa de representar palavras e documentos é fundamental para a maioria das aplicações de Processamento de Linguagem Natural (PLN). O Modelo de Espaço Vetorial, proposto por Salton (1971b), estabelece a base matemática para representar documentos como vetores em espaços multidimensionais. Este modelo foi complementado por técnicas de Modelagem de Linguagem Estatística, que buscam prever palavras com base em seus contextos (Bahl, Jelinek e Mercer, 1983; Goodman, 2001).

Bengio et al. (2003) introduziram modelos de linguagem baseados em redes neurais, onde as palavras eram projetadas em uma camada de embedding antes de serem processadas por camadas subsequentes. Essa abordagem inovadora mitigava os efeitos da alta dimensionalidade, facilitando a generalização e reduzindo o custo computacional.

15.6 Abordagens Baseadas em Predição e Contagem

Modelos de embeddings podem ser divididos em duas categorias principais:

- **Baseados em predição:** Modelos como Word2Vec (Mikolov et al., 2013) e FastText (Bojanowski et al., 2016) utilizam redes neurais para prever palavras com base no contexto. Esses modelos capturam regularidades semânticas e sintáticas, como relações de gênero e pluralidade.
- **Baseados em contagem:** Modelos como GloVe (Pennington, Socher e Christopher D Manning, 2014) utilizam matrizes de coocorrência globais e fatorizações para construir representações densas, capturando relações semânticas a partir de estatísticas de coocorrência.

15.6.1 Word2Vec

Mikolov et al. (2013) propuseram uma representação extremamente densa para a representação de palavras em corpus muito grandes, o Word2Vec, que atinge boa qualidade em tarefas onde é importante uma medida de similaridade entre palavras. Essa representação usa vetores de tamanhos fixo, gerados a partir de uma rede neural. Os vetores usados são de tamanho 50, 100, 300 e 600. Foi inspirada em modelos de linguagens implementados em redes neurais que tentam prever a próxima palavra, o NNLM (Bengio et al., 2003) e o RNNLM.

A geração é feita por dois modelos alternativos: o *Bag-of-Words* e o **Skip-gram**. Nesses modelos uma rede neural é treinada com um corpo muito grande e uma camada intermediária da rede é considerada a representação da palavra. No artigo original, é utilizada uma janela móvel de texto com 5 palavras, sendo que a palavra do meio é a que está sendo representada pelo vetor gerado.

As arquiteturas principais são usadas da seguinte forma:

- **Skip-gram:** Prevê palavras de contexto dado uma palavra central. Esta abordagem é útil para representar palavras raras.
- **CBOW (Continuous Bag of Words):** Prevê uma palavra central com base em palavras de contexto.

Ambos os modelos utilizam amostragem negativa para acelerar o treinamento e reduzir a influência de palavras muito frequentes.

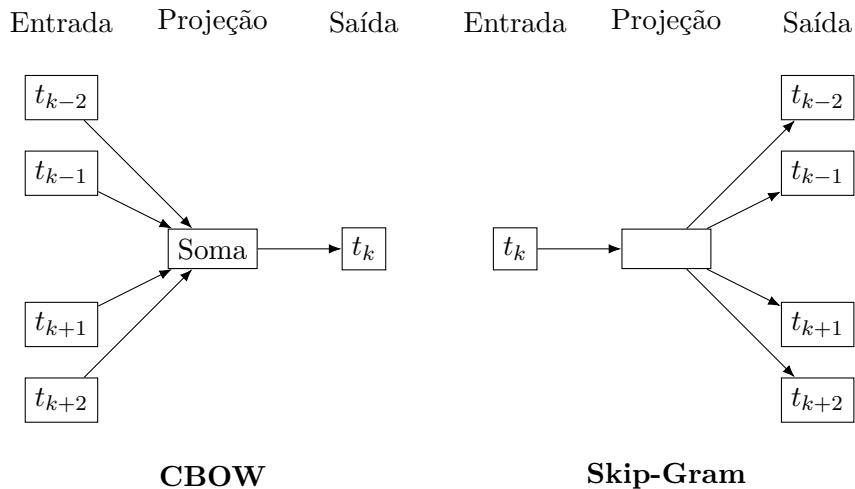


Figura 15.8: Representação dos modos CBOW e Skip-Gram. Fonte: (Bengio et al., 2003)

O modelo *Bag-of-Words* é baseado em uma rede neural que recebe um conjunto de palavras encontrados ao redor de uma palavra específica, em uma janela de texto de tamanho fixo, e que deve aprender a gerar a palavra central da janela.

No modelo **Skip-gram** a entrada da rede é a palavra e a saída da rede são as palavras ao redor dela na janela.

O Word2vec se tornou popular por ser rápido de treinar e disponibilizar o código na rede. A tarefa de aprendizado não é importante no modelo, mas sim os pesos aprendidos. No Skip-gram o treinamento é feito com exemplos positivos, tirados diretamente dos textos por meio de uma janela deslizante, e exemplos negativos gerados aleatoriamente.

15.6.2 GloVe

O GloVe constrói embeddings baseados em estatísticas globais de coocorrência. Sua função de perda considera razões de coocorrências em vez de contagens brutas, capturando melhor a semântica entre pares de palavras.

15.7 Modelos Contextuais

Modelos como ELMo (Peters et al., 2018) e BERT (J. Devlin et al., 2018b) introduzem embeddings contextuais, que variam dependendo do uso de uma palavra em um texto. Essa característica resolve ambiguidades semânticas e melhora a representação de significados complexos em documentos.

15.7.1 Detalhamento do Word2Vec

15.8 Seq2Seq

15.9 *Encoder-Decoder*

15.10 Atenção

Atenção é uma técnica utilizada em redes neurais que simulam a atenção, de forma a aumentar a importância de alguns nós de uma camada da rede, como a entrada, em relação aos outros nós da camada. A ideia básica é fazer a rede focalizar em partes específicas dos dados, e para isso é necessário realizar um processo de aprendizado de máquinas. A atenção é o mecanismo básico de redes LSTM e dos Transformadores, utilizados no BERT e no GPT.

Silva (2007) propôs um modelo de representação de documentos baseado em sinais, fazendo experimentos com FFT e **Wavelets**, concluindo pela melhor eficácia do segundo. O modelo, mais tarde, foi usado para classificação de textos (Xexéo et al., 2008).

Wavelets são transformadas, isto é, funções matemáticas utilizadas para análise de sinais, que permitem descrever e decompor um sinal em diferentes escalas de resolução e frequências. As wavelets possuem uma propriedade chamada de "escala e deslocamento", o que significa que podem ser dilatadas ou comprimidas e deslocadas no tempo sem perder suas características principais.

A análise com wavelets é bastante utilizada em áreas como processamento de sinais, compressão de dados, reconhecimento de padrões, processamento de imagens, etc.. Como outras técnicas típicas de processamento de sinal, é praticamente esquecida na área de Recuperação da Informação e Processamento de Linguagem Natural.

A proposta de Silva (2007) tem como característica principal a reorganização dos vetores de termos por meio da correlação entre os termos, assim construindo um vetor mais similar a um sinal como os tratados pelas transformadas analisadas, o que permite o funcionamento das Wavelets de forma adequada.

15.11 Desafios e Perspectivas

Embora avanços significativos tenham sido alcançados, desafios permanecem:

- **Ruído em textos:** Dados ruidosos podem prejudicar a qualidade dos embeddings (Doval, Vilares e Gómez-Rodríguez, 2020).
- **Generalização em múltiplos domínios:** Modelos podem apresentar desempenho inferior em domínios especializados.
- **Interpretação:** A opacidade dos modelos dificulta a compreensão das representações aprendidas.

Pesquisas futuras devem focar na adaptação de embeddings para tarefas específicas e na composição de embeddings para entidades mais complexas, como sentenças e documentos (Kiros et al., 2015; Le e Mikolov, 2014).

DRAFT

CAPÍTULO 16

MODELO PROBABILÍSTICO

O nome **Modelo Probabilístico**, que na verdade é uma família de modelos, não é baseado na representação, mas sim na forma como é calculada a relevância de um documento. Em vez de lógica ou em uma abordagem geométrica, o que se busca é calcular a probabilidade de um documento ser relevante para o usuário ou para a consulta. S. Robertson e Zaragoza (2009) chamam essa família de **Arcabouço de Relevância Probabilística**, ou ***Probabilistic Relevance Framework***, (**PRF**)

Possivelmente, Maron e Kuhns (1960), em um artigo seminal, foram os primeiros a propor o que eles chamaram de **indexação probabilística**, baseada em pesos para os índices e em retornar como resposta um número que indicasse a probabilidade de relevância do documento. Além disso, esses autores também propuseram a existência de um espaço de indexação, onde termos mais semanticamente parecidos deveriam estar mais próximos. Em especial, eles definiram relevância exatamente como a probabilidade de um usuário usando um termo ser satisfeita com o documento, o que não é, hoje, a definição mais aceita do termo (S. E. Robertson, 1977; Saracevic, 2017).

A ideia básica de Maron e Kuhns (1960), explicada por (S. E. Robertson, 1977) é “Dada uma variável de critério¹ dicotômica e um sistema que tem alguma informação (essencialmente probabilística), parece ser suficientemente óbvio que os documentos que são mais prováveis de satisfazer os usuários devem ser apresentados a eles antes.”

A abordagem principal, que inspira também o *feedback de relevância*, é que a probabilidade de um documento ser relevante pode ser calculada, iterativa e interativamente, a partir da avaliação do usuário sobre os documentos apresentados em uma primeira consulta, ou sobre o próprio resultado inicial de uma comparação entre a consulta e todos os documentos da base, buscando ao longo das iterações aumentar a probabilidade de um documento ser relevante na resposta. Em algumas versões a interatividade é simulada.

Existem várias versões de modelos probabilísticos, sendo que as mais fáceis de entender partem como extensões do modelo booleano, onde os documentos são indexados apenas a partir da presença ou não do termo no documento, i.e., a matriz termo-documento contém apenas 0s e 1s.

S. E. Robertson (1977) atribui a Cooper, em uma comunicação privada, o **Princípio da Ordenação Probabilística**, ou **PRP**, ou *Probability Ranking Principle* “Se a resposta de um sistema de recuperação

¹Uma variável de critério é uma variável dependente, ou predita, porém apenas em situações não experimentais.(Glen, 2023)

de informação a cada solicitação é uma classificação dos documentos na coleção em ordem decrescente de probabilidade de relevância ao usuário que enviou a solicitação, onde as probabilidades são estimadas com a maior precisão possível com base em quaisquer dados disponibilizados ao sistema para este propósito, a eficácia geral do sistema para seu usuário será a melhor possível com base nesses dados.” Ele mesmo questiona o princípio, mas apresenta argumentos para adotá-lo (van Rijsbergen, 1979). Cooper, o próprio criador do princípio, também apresentou um contra exemplo, considerando o desempenho de um sistema ao apresentar a mesma resposta para duas classes de usuário (S. E. Robertson, 1977).

Esse princípio assume, também, que é possível calcular a probabilidade de relevância de um documento, $P(\text{relevância}|\text{documento})$, que van Rijsbergen também diz estar sujeito a questionamentos, baseado no fato que, durante o uso do sistema, não se sabe quais são ou quantos são os documentos relevantes. Porém é possível, por tentativas, fazer uma conjectura desse valor e depois, por meio de iterações, melhorar essa conjectura (van Rijsbergen, 1979).

16.1 Abordagem Geral do Modelo Probabilístico

Essa seção espelha o trabalho de van Rijsbergen (1979), também explicado por Baeza-Yates e Ribeiro-Neto (2011), que propõe um modelo probabilístico que se inicia com a representação de um documento por um vetor binário, como no modelo booleano:

$$d = (x_1, x_2, \dots, x_n), \text{ onde } \begin{cases} x_i = 1, \text{ se o } i\text{-ésimo termo está presente no documento} \\ x_i = 0, \text{ caso contrário} \end{cases} \quad (16.1)$$

Para um documento qualquer d , existem dois eventos exclusivos, e_1 , o documento é relevante, e e_0 , o documento não é relevante. A ideia do modelo probabilístico é calcular a probabilidade de um documento ser relevante. Isso é escrito como a probabilidade de um documento dado ser relevante. A notação para isso é $P(e_1|d)$. Já a probabilidade de não ser relevante, dado o documento, tem a notação $P(e_0|d)$.

A probabilidade de ser relevante dado o documento é um valor desconhecido, mas o teorema de Bayes fornece um caminho para seu cálculo. Segundo o teorema de Bayes:

$$P(A|B) \cdot P(B) = P(B|A) \cdot P(A) \quad (\text{Teorema de Bayes})$$

ou, para o cálculo de $P(A|B)$:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \quad (16.2)$$

No caso do modelo probabilístico, usando i para representar tanto 1 quanto 0, a probabilidade de um documento (não) ser relevante é

$$P(e_i|d) = \frac{P(d|e_i)P(e_i)}{P(d)}, \quad (16.3)$$

onde $P(e_i)$ é a probabilidade *a priori* de (não) relevância e é proporcional a verossimilhança de relevância, ou não relevância.

Além disso, dado um documento d , van Rijsbergen (1979) lembra que existe uma probabilidade de observar o documento de forma aleatória, seja ele relevante ou não, e ela pode ser descrita da forma:

$$P(d) = \sum_{i=0}^1 P(x|e_i)P(e_i) \quad (16.4)$$

$$P(d) = P(d|e_0)P(e_0) + P(d|e_1)P(e_1) \quad (16.5)$$

o que é trivial se só forem possíveis os eventos e_0 e e_1 e eles forem exclusivos.

É possível defender que, na Equação 16.3, $P(d)$ e $P(e_i)$ podem ser considerados fáceis de calcular ou estimar, ou mesmo considerados constantes em alguns casos, então **o problema mais difícil é calcular $P(d|e_i)$** .

16.1.1 Discussão do valor a ser otimizado

Voltando ao problema original, calculados os valores necessários para decidir se um documento é relevante ou não, van Rijsbergen (1979) usa a Regra de Decisão de Bayes, que minimiza a probabilidade média de erro (van Rijsbergen, 1979):

$$P(e_1|d) > P(e_0|d) \Rightarrow d \text{ é relevante, caso contrário não é relevante.} \quad (16.6)$$

Como só são possíveis dois eventos, a probabilidade de erro pode ser vista como a probabilidade da outra opção, ou seja, ela é

$$P(\text{erro}|d) = \begin{cases} P(e_1|d), & \text{se decidimos por } e_0 \\ P(e_0|d), & \text{se decidimos por } e_1 \end{cases} \quad (16.7)$$

o que significa que, uma vez escolhida uma opção, isto é, escolhido considerar relevante (ou irrelevante), a probabilidade de erro é a probabilidade do evento oposto (van Rijsbergen, 1979). Portanto, para minimizar a probabilidade média de erro, temos que minimizar

$$P(\text{erro}) = P(e_0|d) \times P(d) + P(e_1|d) \times P(d) \quad (16.8)$$

que é mínima se for escolhida a Equação 16.6.

Porém, há um custo associado a recuperar um documento, relevante ou não. Esse custo pode ser modelado como uma **perda** (*loss*) l_{ij} , que indica a perda por decidir e_i quando o verdadeiro evento é e_j . Isso leva a possibilidade de calcular uma **função de risco** $R(e_i|d)$, que é o custo total de todas as decisões (van Rijsbergen, 1979).

$$R(e_i|d) = l_{i0}P(e_0|d) + l_{i1}P(e_1|d) + l_{01}P(e_1|d) + l_{00}P(e_0|d) \quad (16.9)$$

Nesse ponto, o problema passa a ser minimizar o risco, e isso é feito garantindo que (van Rijsbergen, 1979)

$$R(e_1|d) < R(e_0|d) \Rightarrow d \text{ é relevante, caso contrário não é relevante.} \quad (16.10)$$

E as equações 16.6 e 16.10 são equivalentes, sob algumas condições (van Rijsbergen, 1979).

Reescrevendo Equação 16.10, encontra-se a regra para calcular se um documento deve ser considerado relevante:

$$P(d|e_1)P(e_1) > P(d|e_0)P(e_0) \Rightarrow d \text{ é relevante, caso contrário não é relevante.} \quad (16.11)$$

Tudo que foi obtido até agora pode ser levado a uma regra de decisão $g(d)$, que dá um número positivo se o documento for relevante.

Transforma-se a Equação 16.11 em uma subtração, já que $a > b \Rightarrow a/b > 1$, logo, o objetivo deve garantir que

$$\frac{P(d|e_1)P(e_1)}{P(d|e_0)P(e_0)} > 1. \quad (16.12)$$

Usando a função logaritmo, isso permite definir $g(d)$ como uma função que dá um número positivo se o documento é relevante, ou um número negativo se é não relevante.

$$g(d) = \log \frac{P(d|e_1)P(e_1)}{P(d|e_0)P(e_0)} \quad (16.13)$$

16.1.2 Supondo termos independentes

Neste ponto, podemos voltar a considerar $P(d|e_i)$. van Rijsbergen (1979) agora faz o que chama de “suposição principal”: que cada presença de um termo em d é independente das outras e isso leva a probabilidade do documento ser escolhido dado que é (não) relevante é o produto da probabilidade de todos os termos serem escolhidos dados que são (não) relevantes:

$$P(d|e_i) = P(x_1|e_i)P(x_2|e_i)\dots P(x_n|e_i) = \prod_{j=1}^n P(x_j|e_i). \quad (16.14)$$

A suposição de independência entre os termos quanto a sua presença em documentos não é verdade, mas é comum em todos os modelos mais simples de Recuperação da Informação, inclusive no Modelo Vetorial. Muitos modelos avançados são criados a partir da ideia de corrigir as distorções criadas nos modelos que a fazem. No próprio texto de (van Rijsbergen, 1979) ela é posteriormente relaxada.

Continuando, e para facilitar o trabalho, van Rijsbergen cria duas novas variáveis, $p_i = P(x_i = 1|e_1)$ e $q_i = P(x_i = 1|e_2)$. Isto é, p_i indica a probabilidade de um termo i aparecer dado que o documento é relevante, e q_i a probabilidade do documento dado que o resultado não é relevante.

$$p_i = Prob(d_i = 1|e_1) \quad (16.15)$$

$$q_i = Prob(d_i = 1|e_0) \quad (16.16)$$

Com isso, já que os eventos de aparecimento de termo são independentes, a Equação 16.14 pode ser reescrita como:

$$p(d|e_1) = \prod_{i=1}^n p_i^{x_i} (1 - p_i)^{1-x_i} \quad (16.17)$$

$$p(d|e_2) = \prod_{i=1}^n q_i^{x_i} (1 - q_i)^{1-x_i} \quad (16.18)$$

onde x_i só assume os valores 0 e 1, e nenhum termo é zero, já que $y^0 = 1$ ou os produtórios seriam zerados.

Nesse caso, p_i (q_i) é a probabilidade de que o documento é relevante (ou não), então o i -ésimo termo estará presente.

Agora, é feita a substituição com as Equações 16.17 e 16.18:

$$g(d) = \log \left(\frac{\prod_{i=1}^n p_i^{x_i} (1 - p_i)^{1-x_i}}{\prod_{i=1}^n q_i^{x_i} (1 - q_i)^{1-x_i}} \right) \quad (16.19)$$

Usando propriedades do logaritmo, e reorganizando, van Rijsbergen (1979) apresenta então a função g em termos de somatórios²:

$$g(d) = \sum_{i=1}^n c_i x_i + C \quad (16.20)$$

onde

$$c_i = \log \frac{p_i(1 - q_i)}{q_i(1 - p_i)} \quad (16.21)$$

e

$$C = \sum_{i=1}^n \log \frac{(1 - p_i)}{1 - q_i} + \log \frac{P(e_1)}{P(e_0)} + \log \frac{l_{01} - l_{11}}{l_{12} - l_{22}} \quad (16.22)$$

van Rijsbergen (1979) chama a atenção, nesse ponto, que c_i são os pesos dos termos nos documentos, e que C é uma constante para cada consulta. Isso faz com que c_i possa ser calculado apenas para os termos presentes. Ainda mais, as funções de perda podem ser também consideradas constantes, sendo que podem ser adotadas $l_{11} = l_{22} = 0$, isto é, não há custo em recuperar um documento relevante e também para não recuperar um documento não relevante, e l_{21}/l_{22} pode ser escolhido.

16.2 Estimando as probabilidades

Considerando então que C é constante para a consulta, então ele pode ser desprezado no cálculo de $g(d)$. Além disso, é possível agora dar valores para as variáveis. Para isso, seria necessário ter a informação completa sobre a coleção, de forma a estimar $p_i = r/R$ e $q_i = (n - r)/(N - R)$, onde

- r , a quantidade de documentos relevantes que contém o termo;
- R , a quantidade total de documentos relevantes;
- n , a quantidade de documentos que contém o termo;
- N , a quantidade total de documentos na coleção.

Chances

Para evitar confusão entre as palavras que usamos no dia a dia e uma terminologia formal, é importante lembrar a definição de **chance** (*odds*) de ocorrência de um evento como a razão entre a probabilidade do evento acontecer e a probabilidade de não acontecer. Isto é:

$$\text{chance}(A) = \frac{A}{A_c} = \frac{P(A)}{1 - P(A)}.$$

Disso é possível calcular:

- r/R , a probabilidade de um documento relevante conter o termo;
- $(n - r)/(N - R)$ a probabilidade de um documento não relevante conter o termo;
- n/N , a probabilidade de um documento qualquer conter o termo;
- $(R - r)/R$ a probabilidade de um termo relevante não conter o termo;
- $(N - n - R + r)/(N - R)$, a probabilidade de um documento não relevante não conter o termo;
- $(N - n)/N$, a chance que um documento não contenha o termo;
- $(n - r)/(N - n - R + r)$, a chance que um documento não relevante contenha o termo, e
- $n/(N - n)$ a chance de um documento conter o termo.

²Apesar dessa transformação usar apenas matemática básica apenas, ela é não trivial, apesar de van Rijsbergen considerá-la “óbvia”.

Isso leva a $g(x)$ poder ser modelado como:

$$g(d) = \sum_{i=1}^n x_i \log \frac{r/(R-r)}{(n-r)/(N-n-R+r)} + C, \quad (16.23)$$

lembrando que x_i assume apenas os valores 0 ou 1, e indicam a presença do termo no documento, e deve ser considerado apenas para os termos que estão na consulta e no documento.

E van Rijsbergen (1979) usa a fórmula F4 de S. E. Robertson e Spärk-Jones (1976) de :

$$K(N,r,n,R) = \sum_{i=1}^n x_i \log \frac{r/(R-r)}{(n-r)/(N-n-R+r)} \quad (16.24)$$

O cálculo de K , para um termo, é o seu peso, sendo que pode ser interpretado como “o poder com que o i -ésimo termo discrimina entre documentos relevantes e não relevantes”(van Rijsbergen, 1979).

O modelo probabilístico então depende de conhecer 4 valores, entre eles o número de documentos relevantes na coleção e o número de documentos relevantes encontrados. Uma alternativa é considerar $r = R = 0$ (Baeza-Yates e Ribeiro-Neto, 2011).

Outra suposição é considerar $p_{iR} = 0.5$ e $q_{iR} = \frac{n}{N}$ o que leva a (Baeza-Yates e Ribeiro-Neto, 1999)

$$g(x) = \sum \log \left(\frac{N-n}{n} \right).$$

16.3 Modelo RSJ

S. E. Robertson e Spärk-Jones (1976) propuseram um modelo probabilístico com duas possíveis suposições e dois possíveis princípios, conhecido como **Modelos RSJ**.

As suposições são sobre a independência dos termos:

- I_1 A distribuição dos termos nos documentos relevantes é independente e sua distribuição em todos os termos é independente, isto é, a presença de um termo em um documento relevante não impacta a presença de outros termos em qualquer outro documento relevante.
- I_2 A distribuição dos termos nos documentos relevantes é independente e sua distribuição nos documentos irrelevantes os termos é independente, isso é, estende I_1 para a presença em documentos não-relevantes.

Os princípios de ordenação também são dois:

- O_1 A probabilidade de relevância é baseada apenas na presença de termos nos documentos, e
- O_2 A probabilidade de relevância é baseada na presença e na ausência de termos em um documento.

Escolhendo entre I_1 e I_2 e também entre O_1 e O_2 são possíveis quatro combinações, que são chamadas de F_1, F_2, F_3, F_4 .

$$F_1 = \log \frac{r/R}{n/N} \quad (16.25)$$

	i_1	i_2
O_1	F_1	F_2
O_2	F_3	F_4

F_1 avalia a razão entre a proporção de documentos relevantes em que o termo ocorre para a proporção de ocorrência em toda a coleção.

$$F_2 = \log \frac{r/R}{(n-r)/(N-R)} \quad (16.26)$$

F_2 avalia a razão entre a proporção de documentos relevantes e os documentos não relevantes que foram encontrados.

$$F_3 = \log \frac{r/(R-r)}{(n)/(N-n)} \quad (16.27)$$

F_3 avalia a razão entre o número de documentos relevantes em que o termo ocorre e o número de documentos relevantes onde ele não ocorre.

$$F_4 = \log \frac{r/(R-r)}{(n-r)/(N-n-R+r)} \quad (16.28)$$

16.4 BM25

Stephen Robertson e Karen-Spärck Jones são dois pesquisadores britânicos que contribuíram muito para a Recuperação da Informação, principalmente através do estudo do modelo probabilístico. No projeto **Okapi** eles propuseram e avaliaram várias medidas de similaridade de documentos, sendo que a **BM25**³, se mostrou a mais eficiente e vem resistindo com o tempo, sendo usada no Lucene, descrito na seção 19.3, e outros softwares.

S. Robertson e Zaragoza (2009) explicam essa função de recuperação para um modelo probabilístico a partir da frequência dos termos no documento e mais dois fatores: a **eliteness** e a **saturação**.

A **eliteness** seria um conceito próximo da *aboutness*, onde quanto mais alto o valor, mais significativo é o termo como definidor do que é o documento. Eles assumem que a quantidade de ocorrência de um termo no documento depende de sua **eliteness**. Logo a relevância de um termo para uma consulta seria derivada da **eliteness** do termo, que seria derivada da sua frequência, crescendo inicialmente de forma monotônica com ela.

Além disso, a **eliteness** se aproxima assintoticamente de um valor máximo, onde, para um peso de termo w_i^{elite} :

$$\lim_{tf \rightarrow \infty} w_i^{\text{elite}} = \log \frac{p_i(1 - q_i)}{(1 - p_i)q_i}, \quad (16.29)$$

que é exatamente o peso no modelo BIM apresentado anteriormente.

Os autores continuam por supor que autores podem ter diferenças, incluindo sua verbosidade e o escopo do trabalho. Além disso, há a questão do tamanho do documento. Com isso, propõe normalizar os tamanhos de documentos, com um parâmetro b , que varia de 1, normalização total, a 0, desligando a normalização:

$$B = (1 - b) + b \frac{\text{len}(d)}{\sum_{j=1}^N \text{len}(d_j)} \quad (16.30)$$

³BM significa **B**est **M**atch

Aplicando isso a fórmula da frequência (tf), chega a:

$$w_i^{\text{BM25}} = \frac{tf}{k_1((1-b) + b \frac{\text{len}(d_i)}{\sum_{j=1}^N \text{len}(d_j)}) + tf} w_i^{\text{RSJ}} \quad (16.31)$$

Onde w_i^{RSJ} é uma fórmula para o peso dos termos propostas por Robertson e Sparck Jones:

$$w_i^{\text{RSJ}} = \log \frac{(r_i + 0.5)(N - R - n_i + r_i + 0.5)}{(n_i - r_i + 0.5)(R - r_i + 0.5)} \quad (16.32)$$

16.4.1 BM25

16.5 Modelo Probabilístico hoje

A maioria das aplicações que implementam mecanismos de buscas hoje em dia afirmam usar o modelo vetorial porque o usam para a representação dos documentos, porém a função de similaridade mais aceita nas ferramentas é a BM25.

CAPÍTULO 17

MODELOS DE LINGUAGEM

Os Modelos de Linguagem partem da ideia que usuários que fazem uma boa consulta buscam palavras que devem pertencer ao documento relevante, logo um documento é uma boa resposta para uma consulta se é provável que ele gere essa consulta (Christopher D. Manning, Raghavan e Schütze, 2009a).

A ideia é construir um modelo probabilístico, M_d , para cada documento e ranquear os documentos baseado na probabilidade do modelo gerar a consulta, $P(q|M_d)$.

Uma proposta simples é usar como modelo um **autômato finito** que pode gerar várias sentenças, ou seja, gerar uma linguagem. O autômata da Figura 17.1, por exemplo, gera sentenças como:

- Eu quero
- Eu Quero Eu Quero
- Eu Quero Eu Quero
- ...

Além disso, é possível notar que ele não gera “Quero Eu” ou “Quero Quero”.

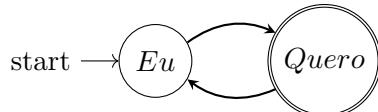


Figura 17.1: Um autômata simples só com dois estados

Se, ao invés de gerar um termo, houver uma distribuição de probabilidade de gerar diferentes termos, temos um modelo de linguagem (Christopher D. Manning, Raghavan e Schütze, 2009a).

Assim, um **modelo de linguagem** é uma função que coloca uma medida de probabilidade sobre *strings* de alguns vocabulários. Para uma linguagem M e sobre um alfabeto Σ , temos que $\sum_{s \in \Sigma} P(s) = 1$ ¹ (Christopher D. Manning, Raghavan e Schütze, 2009a).

Com apenas um nó, temos uma distribuição de probabilidade para diferentes termos, o que é chamado um modelo de linguagem de unígrafo, que na prática é uma previsão da próxima palavra a partir de um estado único.

¹Cuidado porque essa notação, original de Christopher D. Manning, Raghavan e Schütze (2009b), usa o símbolo de somatório, que é a mesma letra grega que a representação usada para o alfabeto

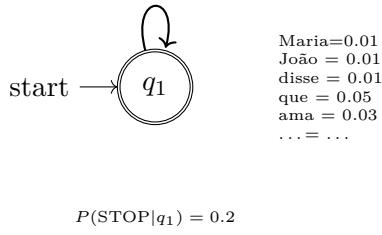


Figura 17.2: Um modelo de linguagem de unígrafo, parcialmente descrito como um autômato.
Fonte: (Christopher D. Manning, Raghavan e Schütze, 2009a)

A questão agora é, usando esse modelo, M_1 , achar a probabilidade dele emitir uma sequência de palavras. Por exemplo, supondo a frase: “Maria disse que ama João”, a probabilidade dela ser emitida pelo modelo da Figura 17.2 é a probabilidade de emissão de cada palavra dado que a frase termina em João. Isso pode ser calculado como:

$$\begin{aligned} P(\text{“Maria disse que ama João”}) &= P(\text{não terminar}) \times P(\text{Maria}) \times \\ &\quad P(\text{não terminar}) \times P(\text{disse}) \times P(\text{não terminar}) \times P(\text{que}) \times \\ &\quad P(\text{não terminar}) \times P(\text{ama}) \times P(\text{não terminar}) \times P(\text{João}) \times P(\text{terminar}) \end{aligned} \quad (17.1)$$

Usando as probabilidades do modelo da Figura 17.2, essa equação pode ser calculada:

$$P(\text{frase}) = 0.8 \times 0.01 \times 0.8 \times 0.01 \times 0.8 \times 0.05 \times 0.8 \times 0.03 \times 0.8 \times 0.01 \times 0.2 \quad (17.2)$$

$$= 9.8394e^{-10} \quad (17.3)$$

Se existem vários modelos, um para cada documento, é possível compará-los em relação a consulta e escolher aquele com mais probabilidade.

Modelos de linguagem de unígrafo são fáceis de calcular, e consideram, como o modelo vetorial e o modelo probabilístico que descrevemos anteriormente, que a probabilidade de uma palavra aparecer é independente das palavras anteriores, logo $P(q) = \prod_{w_i \in q} P(w_i)$.

Já um modelo de bigramas considera a palavra anterior, dessa forma, dada uma consulta que é uma sequência de termos $q = w_0 w_1 \dots w_n$, para cada documento d :

$$P_d(q) = P_d(w_0) P_d(w_1|w_0) \dots P_d(w_n|w_{n-1}) = P_d(w_0) \times \prod_{i=1}^n P_d(w_i|w_{i-1}) \quad (17.4)$$

Christopher D. Manning, Raghavan e Schütze (2009a) afirma que unígramas são suficientes em BRI, apesar de em outras áreas modelos mais complexos serem bem melhores.

17.1 Modelos multinomiais

Com unígramas, a ordem das palavras não é importante, o que leva a uma outra conta em um modelo multinomial, onde se remove a importância da ordenação:

$$P(d) = \frac{L_d!}{t f_{t_1,d}! t f_{t_2,d}! \dots t f_{t_M,d}!} P(t_1)^{t f_{t_1,d}} P(t_2)^{t f_{t_2,d}} \dots P(t_M)^{t f_{t_M,d}} \quad (17.5)$$

onde L_d é o tamanho do documento d .

O cálculo de $P(d|q)$ pode ser resolvido com o Teorema de Bayes

$$P(d|q) = \frac{P(q|d)P(d)}{P(q)} \quad (17.6)$$

Como $P(q)$ é igual para todos os documentos, e só precisamos comparar os $P(d|q)$, pode ser ignorado. Também podemos considerar $P(d)$ uniforme para a coleção, e novamente ignorá-lo. Assim,

$$P(q|M_d) = K_q \prod_{t \in V} P(t|M_d)^{tf_{t,d}} \quad (17.7)$$

Então, no modelo de linguagem, inferimos um M_d para cada documento e estimamos $P(q|M)d$ para cada documento em relação a consulta, resultado em um *score* para o *ranking*:

$$\hat{P}(q|M_d) = \prod_{t \in q} \hat{P}(t|M_d) = \prod_{t \in q} \frac{tf_{t,d}}{L_d} \quad (17.8)$$

Matematicamente, o problema é que os termos que não aparecem tem valor zero, e isso é um produtório. Então existe a opção de fazer *smoothing* ou evitar a exigência da conjunção, considerando as palavras não presentes no documento.

17.2 Smoothing

Por interpolação linear:

$$P(t|d) = \lambda \hat{P}_{\text{mle}}(t|M_d) + (1 - \lambda) \hat{P}_{\text{mle}}(t|M_c) \quad (17.9)$$

Bayesiano:

$$\hat{P}(t|d) = \frac{tf_{t,d} + \alpha \hat{P}(t|M_c)}{L_d + \alpha} \quad (17.10)$$

E uma função de ranking básica pode ser:

$$P(d|q) \propto P(d) \prod_{t \in q} ((1 - \lambda)P(t|M_c) + \lambda P(t|M_d)) \quad (17.11)$$

17.3 Modelos de aprendizado profundo

Nos últimos anos, as redes neurais profundas, têm obtido resultados fortíssimos em várias áreas, que podem ser aplicados a buscas ou aprendizado de máquina. Quando aplicados a problemas de linguagem natural são chamados de ***Large Language Models*** ou ***LLM***. Dois modelos representativos são o BERT (J. Devlin et al., 2018b), com cerca de 335 milhões de parâmetros, e o GPT-3, com cerca de 17 bilhões de parâmetros, mas o quadro tem se alterado rapidamente, principalmente a partir da versão disponibilizada para conversação com o público em geral, o Chat-GPT.

Conceitualmente, esses sistemas baseados em redes neurais profundas também são modelos de linguagem, mas partem de uma abordagem diferente do que os modelos tratados anteriormente neste capítulo, por isso serão tratados a parte, junto com as abordagens de redes neurais.

CAPÍTULO 18

A WEB, ALÉM DO TEXTO

As técnicas discutidas até aqui tem basicamente relação com o texto completo (*full-text*) dos documentos disponíveis, porém há formas alternativas de buscar por documentos, ou de avaliar sua relevância. Com o advento da web elas cresceram em importância, se tornando os principais fatores na ordenação da resposta apresentada ao usuário.

18.1 Curta Perspectiva Histórica Pré-Web

Antes da existência dos computadores, e ainda na infância deles, as primeiras formas de busca foram os catálogos e os índices, construídos ao redor de vários temas. A catalogação se tornou uma Ciência, com métodos detalhados e acordos internacionais. Como vimos na revisão histórica¹, catálogos de autores e títulos evoluíram para catálogos de assuntos e por formas sofisticadas como a indexação facetada ou o CDU².

O tratamento computacional desses catálogos, porém, se assemelha mais a questão da criação de um modelo de dados estruturados e de uma linguagem de consulta que permita consultas precisas, se aproximando mais da teoria e prática de Banco de Dados. Na verdade, existe uma longa discussão na Ciência da Informação sobre o uso de bancos de dados relacionais para registros bibliográficos, e muitos dos sistemas de gerência de biblioteca preferem adotar mecanismos semelhantes ao que chamamos hoje de NoSQL, em especial bases de dados, ou apenas implementações diretas, de sistemas de chave-valor.

Logo, há uma grande área de trabalho na Ciência da Informação relacionada a outras formas de indexar que não o texto completo. A maior parte dessas pesquisas, porém, estão relacionadas a informações específicas dos documentos, e que podem ser levantadas olhando exclusivamente para eles. A possibilidade de trabalhar com o texto-completo, trazida pelo desenvolvimento da memória e velocidade dos computadores, e a popularização dos mecanismos de busca a partir de poucas palavras encontradas no texto, mudou o panorama da busca, principalmente para as necessidades de informação mais simples. Ainda assim, hoje, em aplicações específicas, ainda é importante contar com informações fornecidas por catálogos, justamente porque nem toda a informação sobre um documento pode ser derivada diretamente do texto, sendo necessário contexto³.

¹A história resumida da Recuperação da Informação, pode ser encontrada na Subseção 8.1.3

²

³O aparecimento no mercado das IA baseadas em Redes Neurais Profundas pode mudar esse quadro

O aparecimento da web, e seu crescimento vertiginoso, trouxe a necessidade de catálogos ou lista de documentos. Assim, no começo da internet, sites como Yahoo! e DMOZ faziam sucesso, e até livros físicos foram criados catalogando sites (O'Reilly et al., 1992). Porém, claramente o que inicialmente foi chamado de *crawlers*, que indexavam automaticamente a web por meio de navegação pelos links dos documentos, mostraram que a capacidade de indexar texto completo se mostrou mais rápida que a criação de catálogos, a abordagem alternativa.

Com a grande quantidade de páginas disponíveis, a tarefa de recuperação encontrou um limite para os mecanismos de BRI baseados em texto-completo, e eles não eram mais realmente úteis para encontrar a melhor informação possível. A geração inicial de buscadores automáticos e catálogos, que talvez tenha encontrado seu ápice em um site de busca chamado Altavista, começou a falhar em sua missão do meio para o fim da década de 90.

Tudo isso mudou com o aparecimento do Google, em 1998, que trouxe uma informação adicional para o ranking dos documentos: uma avaliação de sua importância baseada na rede de links da web, o algoritmo Pagerank (Page et al., 1999). Outro algoritmo, o HITS Kleinberg, 1999, influenciou a criação do Pagerank, mas não teve impacto tão forte no mercado.

O sucesso da Google mostrou que os documentos poderiam ser ordenados na resposta a partir da informação gerada pelos links entre eles, o que outros pesquisadores mais tarde usram como inspiração para usar informações relacionadas não ao documento, mas com a forma que eram usados, a navegação entre eles, o tempo de leitura, a similaridade entre usuários, etc.

Porém, não foram os links da Web, ou do hipertexto, que criaram a referência de um documento para outro. Isso já acontecia nas citações, usadas principalmente nos textos científicos, e isso foi percebido anteriormente. Alguns artigos sobre o tema são citados por Kleinberg (1999), e esses trabalhos podem ser considerados antecessores do Pagerank.

18.1.1 Uso de Citações na Busca

Citações são a recompensa e a indicação dos bons artigos científicos. Para o bem ou para o mal, revistas com artigos muito citados são consideradas melhores que revistas com poucos artigos citados, e isso leva os autores a procurarem essas revistas, esperando que seus artigos recebam muitas citações. Os autores, da mesma forma, tendem a confiar mais em revistas mais citadas. Todo esse raciocínio acabou levando a criação dos índices de citação, hoje também usados para avaliar pesquisadores.

O estudo de índices de citações, e outros fatores, acabou sendo denominado Cienciometria⁴, uma área interdisciplinar que estuda a produção científica, sua disseminação e impacto. Este texto não tratará desse assunto, mas sim de precursores que propuseram a rede de citações para auxiliar na busca.

Eugene Garfield, o pesquisador que propôs em 1955 e criou o uso do índice de citações, ideia que foi consolidada em vários índices largamente usados: *SCI*, *SSCI* e *JCR* explica como índices de citação podem ser usados para a busca.

Ele mostra como o *Citation Index* pode ser usado como uma ferramenta para “navegar pela literatura”, principalmente detectando “habilitando o buscador a encontrar descendentes atuais de artigos ou livros específicos”. Dessa forma, ele demonstra que as citações criam uma rede de documentos.

A mesma ideia já tinha sido proposta tanto por Bush (1945), que concebeu uma espécie de escrivaninha para textos ligados, o Memex, quanto por Paul Otlet, circa 1920, que propôs o Mundaneum,

⁴Em inglês, *Scientometrics*

e seria mais tarde transformada no conceito de hipertexto por Ted Nelson em 1965 e amplamente divulgado em seu livro *Literary machines: The report on, and of, Project Xanadu concerning word processing, electronic publishing, hypertext, thinkertoys, tomorrow's intellectual revolution and certain other topics including knowledge, education and freedom* (Nelson, 1987). Tudo isso foi implementado em vários sistemas de hipertexto em torno de 1990, porém a WWW (Tim Berners-Lee, 1989), proposta aberta baseada no http e no HTML, foi o sistema que progrediu e hoje é ubíquo.

Além disso, ele também defende que a análise histórica de uma área de pesquisa deve ser feita a partir dos índices de citação das publicações, o que equivale a dar uma medida de qualidade às publicações baseadas na sua influência na área.

Kleinberg (1999), inspirado nas citações, propôs o primeiro algoritmo que criava uma avaliação, para páginas web, a partir da sua importância no grafo criado pelos links entre as páginas.

18.2 A Web como um Grafo Direcionado

Uma das considerações importantes de vários algoritmos é entender a Web como um grafo direcionado. Esse grafo pode ser construído diretamente a partir das ligações entre as páginas ou por dados correspondentes a navegação entre as páginas, o que seria uma referência mais dinâmica. Além disso, com o advento das redes sociais, passaram a existir também tipos de links, como o “Reply-to” e o “Retweet”.

Em todo caso, o modelo básico assume que as páginas são os nós e os links, tanto referências HTML ou outra forma, são as arestas. Por exemplo, a partir do seguinte conjunto de páginas com suas referências listadas, é possível criar o grafo da Figura 18.1 e a matriz de incidência da Equação 18.1.

- www.um.com
 - www.dois.com
 - www.tres.com
- www.dois.com
 - www.tres.com
 - www.quatro.com
 - www.cinco.com
- www.tres.com
 - www.um.com
- www.quatro.com
 - www.tres.com
- www.cinco.com
 - www.um.com
 - www.quatro.com

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix} \quad (18.1)$$

É possível construir outros grafos, inclusive não direcionados, por exemplo, um grafo de co-citação, pode ser construído para o mesmo conjunto de páginas (Henzinger, 2001). Para o mesmo conjunto de páginas, o grafo de co-citação aparece na Figura 18.2.

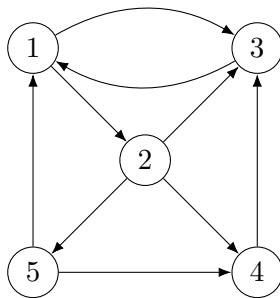


Figura 18.1: Grafo para um conjunto de páginas web descrito no texto.

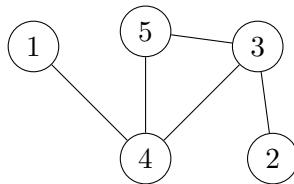


Figura 18.2: Um grafo de co-citação para o mesmo conjunto de páginas (Figura 18.1). Dois nós estão ligados se um mesmo nó apontar para os dois.

Ver a web como um grafo inspirou vários avanços, desde melhorias na busca, como também ferramentas de análise social e de outras formas de rede. Ao estudar a estrutura da WWW, Barabási (2002) descobriu que a rede apresentava uma estrutura distribuição de potência, onde alguns nós possuem muito mais ligações que outros nós. Em seu livro *Linked: The New Science of Networks*, ele argumenta que a compreensão da estrutura e da dinâmica das redes é crucial para compreender muitos fenômenos da nossa sociedade e do mundo natural. O autor também discute como as redes impactam a disseminação da informação, a tomada de decisões e o comportamento coletivo. Ele mostra como o controle sobre certas redes pode ser usado para influenciar a sociedade e apresenta exemplos de como a manipulação da rede pode ser usada para o bem ou para o mal.

18.3 HITS

HITS é a sigla para *Hypertext Induced Topic Search*, um algoritmo proposto por Kleinberg (1999) baseado nos conceitos de **hubs** e **authorities**. Um *hub* é um nó que fornece links para muitos nós, enquanto um *authority* é um nó que recebe links de muitos nós. Enquanto os primeiros representam de certa forma um catálogo de informações, os segundos representam alguma informação importante.

A Figura 18.3 mostra uma rede representando uma rede com *hubs* e autoridades, calculados com o algoritmo HITS, normalizando de forma que a soma de todas autoridades, ou *hub*, dos nós seja 1. Os nós ‘d’ e ‘b’ são facilmente identificados como possíveis hubs, e realmente atingem, cada um, .31 como valor normalizado. Os nós ‘c’ e ‘g’ também parecem escolhas óbvias como autoridades, e realmente atingem cada um .18 como valor normalizado. Porém os nós ‘f’ e ‘j’ também atingem bons valores como autoridades, .17 e .15, respectivamente.

O princípio do algoritmo HITS é ordenar as páginas recuperadas em uma resposta a uma consulta por um algoritmo de busca por texto completo por sua importância, e essa importância é dada pela quantidade de vezes que a página é citada por páginas importantes. Kleinberg (1999) diz que uma página é importante a partir de duas características: sua qualidade como *hub* e sua qualidade como

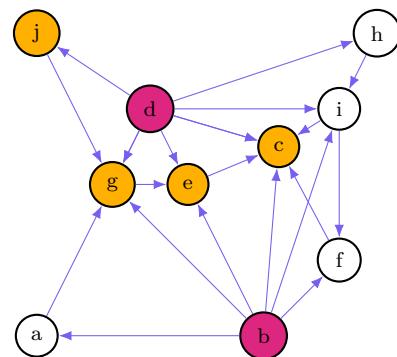


Figura 18.3: A web pode ser vista como um grafo direcionado. Nessa imagem estão marcados os *hubs* em magenta e autoridades em laranja. Os valores calculados estão na Tabela 18.1.

Tabela 18.1: Valor de autoridade e hub para o grafo da Figura 18.3.

Nó	Autoridade	Hub
0	0.08	0.07
1	0.00	0.31
2	0.18	0.00
3	0.00	0.31
4	0.17	0.07
5	0.08	0.07
6	0.18	0.06
7	0.08	0.03
8	0.08	0.03
9	0.15	0.07

authority (autoridade) ou ambos. Além disso, duas premissas são feitas: boas autoridades são apontadas por bons *hubs* e bons *hubs* apontam para boas autoridades.

O algoritmo básico do HITS pode ser descrito em dois grandes passos:

1. Construir um sub-grafo focado da WWW, e
2. Computar os valores das páginas como *hubs* e autoridades grafo formado por elas.

O primeiro passo do algoritmo HITS, descrito no algoritmo 18.1 e ilustrado na Figura 18.4, tem como entrada a resposta de uma consulta. Esse conjunto é então aumentado com todas as páginas que as páginas originais apontam (*forward link pages*) e mais um conjunto limitado, por um k escolhido, de páginas que apontam para alguma das páginas do conjunto original (*backward link pages*). O conjunto original é chamado de *root set* e o conjunto resultante resultante é chamado de *base set*.

Algoritmo 18.1: Algoritmo 1 do HITS, construção do sub-grafo focado

Entrada: R , resultados de uma consulta
 $k \in \mathbb{N}$

```

1  $S \leftarrow R$ 
2 para todo  $r \in R$  faça
3    $I \leftarrow$  páginas que apontam para  $r$ 
4    $S \leftarrow S \cup I$ 
5    $O \leftarrow$  páginas apontadas por  $r$ 
6   se  $|O| < k$  então
7     |  $S \leftarrow S \cup O$ 
8   senão
9     |  $S \leftarrow S \cup \{k$  páginas aleatórias de  $O\}$ 
10  fim
11 fim
12 retorna  $S$ 

```

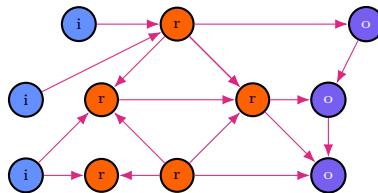


Figura 18.4: Ilustração do primeiro passo do algoritmo HITS, aa construção do *base set*. No início um mecanismo de busca retorna os nós marcados com r , em laranja, o *root set*. O conjunto é então extendido com todos os nós apontados por eles, marcados com o , em roxo, e por no máximo k nós que apontam para eles, i , em azul.

Definido o *Baseset*, a segunda parte do algoritmo HITS busca calcular iterativamente os valores de *hub* e autoridade para cada nó, a partir de um valor inicial. Para isso a ideia é que o valor de autoridade de um nó seja a soma do valor dos hubs que apontam para ele, e o valor de hub de um nó seja a soma dos valores de autoridade dos nós que apontam para ele. Para evitar uma escalada de valores, as somas são normalizadas. As equações a seguir e o algoritmo 18.2 mostram como são calculadas cada iteração do HITS. Em especial, o algoritmo 18.2 apresenta uma versão com k iterações,

mas ela pode ser substituída por uma versão similar onde se calcula se o algoritmo já convergiu.

$$t_i^{(k+1)} = \sum_{j=1}^{|G|} \begin{cases} h_j^{(k)}, & \text{se } p_j \text{ aponta para } p_i \\ 0, & \text{se } p_j \text{ não aponta para } p_i \end{cases} \quad (18.2)$$

$$a_i^{k+1} = \frac{t_i^{k+1}}{\sum_{j=1}^{|G|} t_j^{k+1}} \quad (18.3)$$

$$u_i^{(k+1)} = \sum_{u=1}^{|G|} \begin{cases} a_j^{(k+1)}, & \text{se } p_i \text{ aponta para } p_j \\ 0, & \text{se } p_i \text{ não aponta para } p_j \end{cases} \quad (18.4)$$

$$h_i^{k+1} = \frac{u_i^{k+1}}{\sum_{j=1}^{|G|} u_j^{k+1}} \quad (18.5)$$

Ainda pode ser considerado mais um passo no algoritmo HITS, que escolhe as páginas com maior pontuação como *hub* e autoridade para mostrar ao usuário.

Também é possível calcular o HITS de forma matricial. Seja L a matriz de adjacência do grafo, então (Langville e Meyer, 2006):

$$a^{(k+1)} = L^T h^{(k)} \text{ e } h^{(k+1)} = La^{(k+1)} \quad (18.6)$$

Para o HITS fazer sentido como algoritmo para ordenar as páginas, ele deve convergir, o que pode ser provado (Kleinberg, 1999). Além disso, os relatos disponíveis dizem que ele converge rapidamente (Henzinger, 2001).

Existem várias questões teóricas e práticas sobre o HITS, como se ele converge e qual a forma mais rápida de calcular cada iteração, por exemplo usando operações matriciais (Berry e Browne, 2005; Kleinberg, 1999).

O Algoritmo HITS tem alguns problemas.

- O algoritmo é sensível a alteração dos nós, já que só trata de um sub-conjunto da rede a cada geração dos valores Henzinger (2001). Isso significa que ele é dependente das consultas, e Langville e Meyer (2006) propõem calcular previamente o HITS para toda a rede como solução para esse problema.
- Se as páginas que são adicionadas ao conjunto inicial, por algum motivo, tratam de outro assunto, os ranking mais altos podem não corresponder ao assunto desejado pelo usuário, o que é chamado de *topic drift*.
- Ele é sensível a spam, e adicionar links as páginas influencia os *scores* delas mesmas, mesmo que a página não seja apontada por outras.

Algoritmo 18.2: Algoritmo 2 do HITS - atenção a direção de $p \neq q$

Entrada: G , Grafo representando as páginas
 $k \in \mathbb{N}$

```

1 para todo  $p \in G$  faça
2   |    $A[p] \leftarrow 1$ 
3   |    $H[p] \leftarrow 1$ 
4 fim
5 para  $j \leftarrow 1, k$  faça
6   |    $s \leftarrow 0$ 
7   |   para todo  $p \in G$  faça
8     |     |    $A[p] \leftarrow 0$ 
9     |     para todo  $q \in G$  faça
10    |       |       se  $q$  aponta para  $p$  então
11      |         |         |    $A[p] \leftarrow A[p] + H[q]$ 
12      |       fim
13    |     fim
14    |      $s \leftarrow s + A[p]$ 
15  fim
16  para todo  $p \in G$  faça
17    |    $A[p] \leftarrow A[p]/s$ 
18  fim
19   $s \leftarrow 0$ 
20  para todo  $p \in G$  faça
21    |    $H[p] \leftarrow 0$ 
22    |   para todo  $q \in G$  faça
23      |     |       se  $p$  aponta para  $q$  então
24        |       |       |    $H[p] \leftarrow H[p] + A[q]$ 
25        |       fim
26      |     fim
27      |      $s \leftarrow s + H[p]$ 
28  fim
29  para todo  $p \in G$  faça
30    |    $H[p] \leftarrow H[p]/s$ 
31  fim
32 fim
33 retorna  $A, H$ 

```

18.4 Pagerank

O *Pagerank* (Page et al., 1999) foi o algoritmo original do Google, grande responsável pelo seu sucesso comercial. A ideia básica é dar um valor fixo de importância para cada nó na rede e usar esse valor como *score* para ordenar o resultado de uma busca simples de texto completo. Uma página é considerada importante se for apontada por páginas importantes. Isso leva a um algoritmo iterativo que pode ser provado que converge (Langville e Meyer, 2006, pgs. 36-46, 75-84).

Essa importância atribuída pelo *Pagerank* é criada a partir de uma nova medida de centralidade do grafo direcionado construído a partir da rede, definida pelo próprio algoritmo, que pode ser interpretada como a probabilidade de um passeio aleatório pela grafo criado passar pelo nó, com uma chance adicional de pular de um nó para outro sem seguir um link. Outra maneira de olhar, mais matemática, é que os *scores* de importância que o *Pagerank* fornece são um estado estacionário de uma Cadeia de Markov enorme (Langville e Meyer, 2006, pg. 31).

Essa probabilidade pode ser calculada por um algoritmo iterativo. Dada uma página A , o seu *pagerank*, $P_r(A)$, pode ser calculado a partir do *pagerank* das páginas T_i que apontam para A , $P_r(T_i)$. Considerando $N(T_i)$ o total de links saindo de T_i , e uma taxa de amortecimento d , que indica a probabilidade de sair de uma página sem seguir nenhum link, $(1 - d)$, então, se A é apontada por n páginas:

$$P_r(A) = (1 - d) + d \sum_{i=1}^n \frac{P_r(T_i)}{N(T_i)} \quad (18.7)$$

Na Equação 18.7 pode ser interpretada como o *pagerank* de uma página ser a soma da probabilidade de chegar na página a partir de um pulo aleatório mais a soma do *pagerank* “doados” a partir das probabilidades de chegar na página a partir de outras páginas, supondo que em cada página se escolhe o nó de destino de maneira aleatória entre todos os *links* possíveis.

Algoritmo 18.3: Algoritmo do PageRank

Entrada: Grafo $G = (V, E)$, damping factor d

Saída: Ranking de páginas r

```

1 para todo página  $p \in V$  faça
2   |    $r(p) \leftarrow 1/|V|$ ;
3 fim
4 repita
5   |   para todo página  $p \in V$  faça
6     |     |    $s(p) \leftarrow 0$ ;
7     |     |   para todo página  $q$  que linka para  $p$  faça
8       |       |    $s(p) \leftarrow s(p) + r(q)/\deg(q)$ ;
9     |   fim
10   |   fim
11   |   para todo página  $p \in V$  faça
12     |     |    $r(p) \leftarrow (1 - d) \cdot s(p) + d \cdot (1/|V|)$ ;
13   |   fim
14 até convergência;
15 retorna  $r$ 

```

O cálculo é iterativo e na forma matricial onde um passo pode ser descrito por:

$$pA + (1 - p)E, \text{ onde } E = \frac{\bar{e}\bar{e}^T}{|G|} \quad (18.8)$$

O algoritmo 18.3 define o cálculo iterativo do *Pagerank*, página a página. O Programa 18.1 define a função `pagerank` que recebe como entrada uma matriz de adjacência `matrix` e dois parâmetros opcionais, `d` e `eps`, que correspondem ao fator de amortecimento e à precisão, respectivamente. A função retorna o vetor PageRank dos nós do grafo. Nesse caso são usadas operações matriciais.

A matriz de transição é preparada a partir da matriz de adjacência. Primeiro, a soma de cada linha da matriz é calculada e armazenada. Se alguma linha tiver soma igual a zero, é atribuído o valor 1 para a soma dessa linha para evitar divisão por zero. Também será necessário somar uma probabilidade adicional naqueles nós, evitando o que é conhecido como efeito *sink*, o que é dado pela variável `ADD`. Uma outra opção seria tirar esses nós da matriz, já que eles não afetam o *Pagerank* de outros nós, pois não “doam” nada para eles. Em seguida, a matriz é normalizada dividindo cada elemento pela soma da linha correspondente.

O vetor com as *pageranks* iniciais é inicializado com $1/n$ para cada nó, onde `n` é o número de nós. O vetor com os valores anteriores também é inicializado, com uma valor alto qualquer que permite passar no primeiro teste do *loop*⁵.

A seguir, a matriz de transição ponderada com o fator de amortecimento é construída. Em cada iteração, o vetor é atualizado com o produto escalar entre o vetor e a matriz de transição ponderada. O *loop* é interrompido quando a diferença entre o vetor atual e o vetor anterior é menor que o erro máximo aceitável.

Programa 18.1: Cálculo do Pagerank

```

1 import numpy as np
2
3 def page_rank(matrix, d=0.85, eps=1e-8):
4     n = matrix.shape[0]
5     M = np.array(matrix, dtype=float)
6     row_sums = np.array(M.sum(axis=1))
7     ADD = np.zeros((n,n), dtype=float)
8     ADD[row_sums == 0, :] = d * (1/n)
9     row_sums[row_sums == 0] = 1
10    M = M / row_sums[:, np.newaxis]
11    v = np.ones(n) / n
12    last_v = np.ones(n) * 100
13    G = (d * M) + ((1 - d) / n) + ADD
14    print(G)
15    while np.sum(np.abs(v - last_v)) > eps:
16        last_v = v
17        v = np.dot(v, G)
18    return v

```

O *Pagerank* parte do princípio que a rede indica a relevância ou qualidade do documento, o que já provou não ser verdade em alguns casos. Por exemplo, em 2016, a palavra “jew” era considerada

⁵Seria mais adequado usar uma expressão Repita-Até, mas não está disponível em Python.

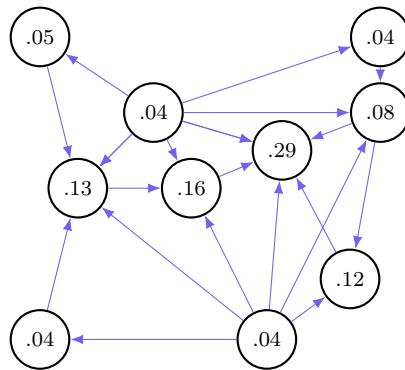


Figura 18.5: Grafo com os nós anotados com seu *Pagerank*

pejorativa aos judeus, não sendo usada normalmente em sites sobre judeus ou judaísmo, porém sendo usada em sites negacionistas do holocausto, logo uma procura por essa palavra trazia no topo esse tipo de site, o que causou uma reação e posteriormente uma mudança no algoritmo para evitar esse tipo de problema. O algoritmo, como alguns outros, também confunde popularidade com qualidade. Além disso, sua proposta inicial só permitia a atualização sobre os dados da rede completa, mas isso foi mudado. O custo computacional também era alto. Como vantagens, é um algoritmo razoavelmente robusto, porém não resistiu às técnicas de manipulação.

Hoje, atualizações do *Pagerank* são apenas um dos algoritmos que compõem o serviço de busca da Google, que é extremamente complexo e possui muitas funcionalidades, como personalização dos resultados e técnicas para combater práticas não éticas de **SEO (Search Engine Optimization)**.

Quando foi lançado, porém, o *Pagerank* mudou o panorama de busca na rede, sendo um dos algoritmos com maior impacto na história, permitindo a criação do que passaria a ser uma gigante do mercado.

A Figura 18.5 mostra um grafo onde cada nó tem anotado seu *Pagerank*. Esse é o mesmo grafo da Figura 18.3, e comparando os dois é possível perceber que o nó *d*, que tem um valor de *hub* alto, tem um *pagerank* baixo, já que ninguém o aponta. Isso é uma demonstração de como o *HITS* pode ser manipulado mais facilmente que o *Pagerank*.

18.5 A Vida Pós-*Pagerank*

Apesar de não ser o primeiro algoritmo, o *Pagerank* mostrou com sucesso que o problema de ordenar a resposta de um mecanismo de busca não precisava olhar para os termos do documento, mas poderia ser independente deste. Com isso, apareceram vários outros algoritmos. Em paralelo, surgiu também uma área de pesquisa conhecida com “Learning to Rank”, que busca usar técnicas de aprendizado de máquina para melhorar a ordenação.

Y. Liu et al. (2008) apresentaram o algoritmo ***BrowseRank***, que se baseia em informação da navegação do usuário em vez do link das páginas, incluindo não só a passagem entre páginas, mas também a forma como se chegou a página e um cálculo do tempo de permanência na página.

O algoritmo do *BrowseRank*, algoritmo 18.4, é mais sofisticado matematicamente que o *PageRank*, mas o princípio é o mesmo: a partir de uma matriz de transição, é gerado um modelo e por meio do método da potência se chega a uma distribuição estacionária. Nesse algoritmo q_i é função do tempo que o usuário fica em cada página.

Algoritmo 18.4: Algoritmo do BrowseRank. Fonte: (Y. Liu et al., 2008)

Entrada: dados do comportamento do usuário
Saída: o *score* de importância da página, π

- 1 Construa o grafo de navegação do usuário;
- 2 Estime q_{ii} para todas as páginas;
- 3 Estime a matriz de transição de probabilidade do EMC;
- 4 Obtenha a distribuição de probabilidade estacionária pelo método da potência;
- 5 Compute a distribuição de probabilidade estacionária do Processo-Q usando a equação;

O BrowseRank precisa de informações adicionais do que a apenas a página estática, logo não cabe em muitas aplicações.

Outra abordagem similar é o *Trustrank* (Gyöngyi, Garcia-Molina e Pedersen, 2004), que busca separar páginas confiáveis de *spam*. Esse algoritmo parte de um conjunto inicial de páginas confiáveis, determinadas por especialistas, para calcular a reputação das outras páginas da rede. Para isso, é usado o princípio que boas páginas apontam para boas páginas, porém são incluídos a attenuação da confiança com a distância e a divisão da confiança com o número de páginas que uma página com a confiança já calculada aponta.

Algoritmo 18.5: Algoritmo Trustrank

Entrada: T , a matriz de transição
 N , o número de páginas
 L , um limite para invocar avaliações de um oráculo
 α , uma fator de decaimento para um *PageRank* com viés
 M , um número de iterações para o *PageRank*

Saída: t , os *scores* do *Trustrank*

- 1 $s \leftarrow$ conjunto de páginas iniciais
- 2 $\sigma = \text{Ordenar}(S)$
- 3 $d = \text{zeros}(N)$
- 4 **para** $i = 1$ até 9 **faz**
- 5 | **se** $O(\delta(i)) == 1$ **então**
- 6 | | **d**(σ_i) = 1
- 7 | **fim**
- 8 **fim**
- 9 $d \leftarrow \frac{d}{\|d\|}$
- 10 $t \leftarrow d$
- 11 **para** $i = 1$ até M **faz**
- 12 | **t** $\leftarrow \alpha \cdot T \cdot t + (1 - \alpha) \cdot d$
- 13 **fim**
- 14 **retorna** d

18.6 Exercícios

Exercício 18.1:

Implemente uma versão do HITS em Python. Teste com a rede da Figura 18.3, com os valores da Tabela 18.1.

Exercício 18.2:

Implemente uma versão do HITS em Python usando apenas operações matriciais (Berry e Browne, 2005), usando o NumPy. Teste com a rede da Figura 18.3, com os valores da Tabela 18.1.

Exercício 18.3:

Implemente uma versão do HITS em Python que para porque convergiu, e não em um número k de passos. Teste com a rede da Figura 18.3, com os valores da Tabela 18.1.

Exercício 18.4:

Implemente uma versão do Pagerank em Python, usando apenas `for`, em vez da versão matricial apresentada. Teste com a rede da Figura 18.5.

Exercício 18.5:

Escolhendo uma rede muito grande disponível para trabalhar com algoritmos de análise de links, faça um estudo da convergência dos algoritmos, mostrando o erro em função do número de passos.

Exercício 18.6:

Implemente um programa que calcule o *Pagerank* simulando uma *random walk* e analise sua convergência em função do número de passos. Use a rede da Figura 18.5.

DRAFT

CAPÍTULO 19

MECANISMOS DE BUSCA

Os mecanismos de busca atuais possuem arquiteturas complexas. O Capítulo 8 apresentou arquiteturas abstratas e simplificadas para que fosse possível discutir as três partes mais importantes dos mecanismos de busca: encontrar os documentos, indexá-los e permitir a consulta dos usuários finais a base gerada. Neste capítulo são discutidos mais alguns detalhes da arquitetura de alguns sistemas e softwares dedicados a essa tarefa.

A Figura 19.1 tenta representar um mecanismo de busca moderno, com seus dois principais clientes, o anunciante e o usuário final. Normalmente a presença do usuário “Anunciante” não é desenvolvida nos textos de Recuperação da Informação, e de certa forma, há uma diferença na forma como anunciantes e não-anunciantes são tratados pelos mecanismos de busca na rede, como Google ou Bing, porém a sua existência não pode ser simplesmente esquecida, já que interferem nos resultados.

Por exemplo, inicialmente os links patrocinados que apareciam na resposta da Google, isso é, os resultados pagos pelos anunciantes, vinham em caixas separadas, claramente indicados. Em 2023 é difícil separar uma resposta que é um anúncio de uma resposta “comum”, conhecidos como links orgânicos. Essa mudança aconteceu a partir de 2016 (Marvin, 2016) e hoje pode ser analisada pela forma que a Google discute como aparecem os anúncios (Google, 2023c; Lomas, 2020).

Outra característica da figura é a análise estar separada em três partes: o indexador, a análise de rede e outras análises. O indexador e o índice são a parte mais tradicional da Recuperação da Informação, que passaram a precisar do apoio de sistemas de análise de rede e ligações entre os documentos com o advento da Web. Porém, atualmente, como será visto na descrição do funcionamento do Google, muitos outros dados são necessários. A Microsoft já declarou que usa 1.000 sinais para classificar um documento, e enquanto a Google normalmente usa “mais de 200”, porém já declarou que cada um deles tem mais de 50 variantes, o que levaria a 10.000 sinais (Sullivan, 2010). Quais os sinais usados, porém, além dos óbvios, é um segredo comercial das empresas. Também está incluída uma funcionalidade de aprendizado que busca fornecer informações para um modelo de usuário que pode modificar o resultado das perguntas. A Figura 19.1 é, então, ainda uma simplificação de uma rede muito mais complexa de produtos, sub-sistemas, software e hardware, porém é um modelo que já permite um *insight* mais detalhado.

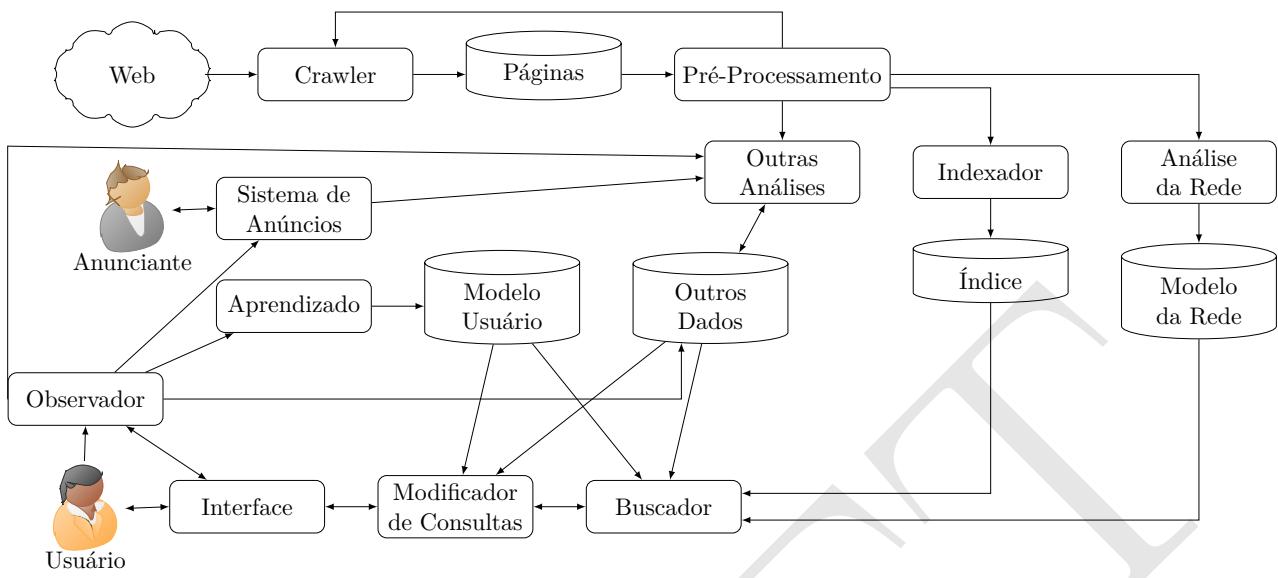


Figura 19.1: Modelo genérico de um mecanismo de busca moderno, baseado em diversas descrições do funcionamento do Google.

19.1 Funcionamento do Google

Certamente o **Google** é o mecanismo de busca mais conhecido e mais usado por todos, sendo uma das maiores empresas do planeta. Isso aconteceu por causa da capacidade da Google monetizar o processo de busca, e para que isso acontecesse a Google teve que primeiro mostrar que era o melhor mecanismo de busca disponível, e manter essa liderança. Em fevereiro de 2023 StatCounter (2023) indicava que a Google tinha uma fatia de 93.37% do mercado mundial de mecanismos de busca, sendo que no Brasil atingia 96.83%.

A discussão a seguir sobre como funciona o Google se baseia basicamente em mensagens em blogs ou artigos nos sites da Google para divulgação ou explicação de seus mecanismos para clientes e desenvolvedores, e ainda por textos criados por especialistas em *Search Engine Optimization* (SEO). Nenhuma destas fontes dá uma visão completa ou detalhada de como funciona o mecanismo da Google, e muito que se fala fora da Google é suposição. Porém muitas informações relevantes podem ser obtidas, pelo menos a nível superficial, mais relacionado ao impacto das mudanças da Google na posição de sites, o que é de interesse dos webmasters e daqueles interessados em SEO,. Não é possível, porém, saber os segredos do negócio da Google e como realmente todos esses sistemas funcionam juntos. Outras fontes possíveis são artigos científicos produzidos por membros da Google e patentes da Google, mas esses geralmente se dedicam a detalhes específicos e também dão pouca informação de como as “coisas realmente funcionam”, sendo mais usados em outras partes deste livro.

Seus fundadores, Brin e Page (1998) explicaram a arquitetura inicial da Google, apresentada na Figura 19.2. O intuito era demonstrar como vários processos, representados aqui como ovais, ocorriam em paralelo, e que também existiam várias bases de dados. Sua principal proposta era selecionar as páginas por uma busca simples conjuntiva por palavras chave e avaliar a qualidade da página, para o *ranking*, pelo algoritmo Pagerank (Page et al., 1999).

Foi a mudança do conceito do ranqueamento por relevância para a importância do documento na rede, fornecida pelo PageRank, o grande diferencial de qualidade, ou pelo menos de qualidade percebida pelo usuário, da Google sobre os principais concorrentes da época. Por exemplo, o Altavista (Seltzer,

E. J. Ray e D. S. Ray, 1997), líder do mercado anteriormente, já não conseguia ordenar as páginas de forma a que o usuário ficasse satisfeito com os primeiros resultados listados.

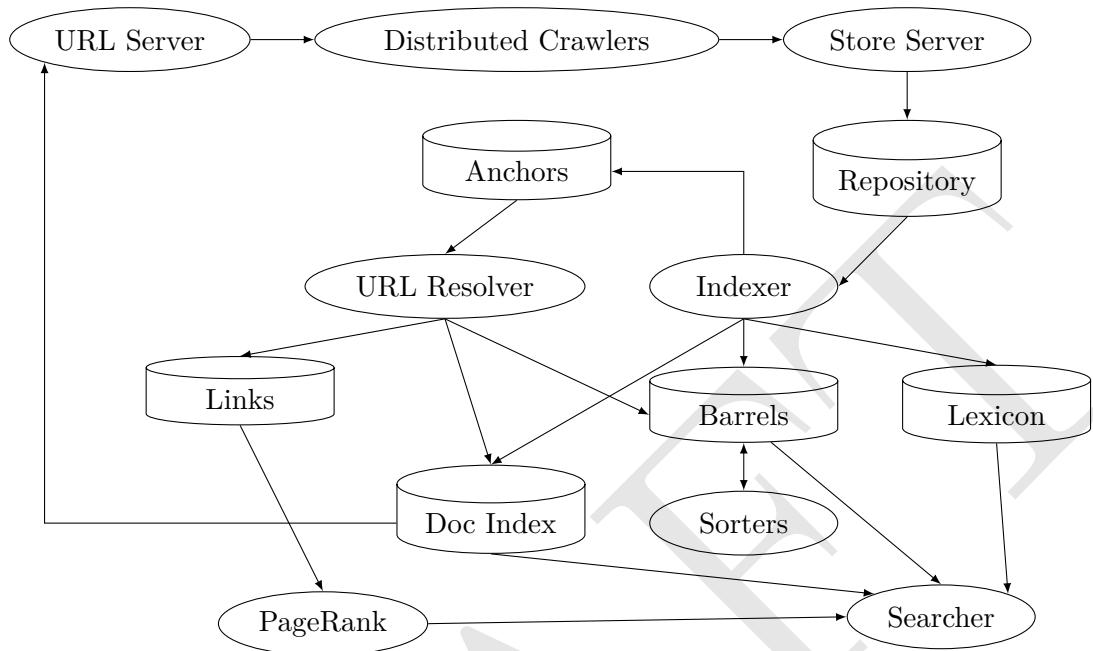


Figura 19.2: Arquitetura original do Google (redesenho). Fonte: (Brin e Page, 1998)

Desde sua versão original o mecanismo do Google já passou por milhares de pequenas mudanças e algumas grandes mudanças, logo a Figura 19.2 tem apenas interesse histórico. Algumas foram explicadas pela própria Google, outras percebidas pelo mercado. Algumas versões tiveram grande impacto nos resultados da busca imediatamente, como a *Caffeine* ou a introdução do Panda, outras, como a reescrita total do código com a versão Hummingbird tiveram um impacto imediato menor, mas permitiram mudanças mais aprofundadas ao longo do tempo.

A Google descreve seu mecanismo como composto de três etapas, como esperado de um mecanismo de Recuperação da Informação tradicional (Google, 2023b):

1. **Rastreamento**, que se inicia pela descoberta de URLs. Algumas páginas já são conhecidas e usadas como origem de navegação para novas páginas, outras podem ser recebidas pelo Google por meio de *sitemaps*. Quem faz o rastreamento é o Googlebot, um crawler. O Googlebot não só analisa a página como executa todo o código JavaScript, pois ele pode ser necessário para captar o conteúdo (Google, 2023b). Além disso, usa dois *crawlers* diferentes, o **Googlebot** para computadores e o Googlebot para *smartphones*, cada um simulando um tipo de usuário diferente. O Capítulo 20 discute mais sobre o comportamento de [crawler]crawlers.
2. **Indexação**, que trata todo o conteúdo da página e usa tanto técnicas básicas de Recuperação de Informação quanto técnicas avançadas e algoritmos proprietários discutidos nesse capítulo.
3. **Exibição dos resultados da pesquisa**, buscando encontrar uma ordem de apresentação (*ranking*) que atenda o usuário, também de acordo com vários algoritmos descritos neste capítulo.

Deve ficar claro que a indexação guarda a informação que será usada pela exibição, porém é importante lembrar que tanto no rastreamento e recuperação de páginas quanto na indexação são guardados vários **sinais**, isto é, informações levantadas ao acessar ou ao analisar a página que, mais tarde, serão usadas para melhor ordenar os resultados, tanto para todos quanto para um usuário específico.

Entre os dados utilizados no Google estão mais de 200 sinais, entre eles:

- Link score, o principal e baseado no PageRank;
- Relevância do texto âncora;
- Conteúdo da página, considerando especialmente:
 - Título;
 - Palavras Chave;
 - Tamanho da página;
 - Compreensibilidade;
- Velocidade da páginas;
- Facilidade para dispositivos móveis, e
- Experiência do usuário, incluindo o *click-through rate*, que é uma forma de feedback de relevância.

Para o usuário, talvez a consulta seja a parte mais importante. Para atender a consulta, a Google descreve cinco fatores principais :

1. **Significado**, identificar o resultado da consulta, principalmente a partir de modelos de idiomas que buscam a intenção da consulta. Um exemplo é a capacidade de usar sinônimos precisos das palavras, mas algoritmos como o BERT (Pandu Nayak, 2019) permitem buscas a partir do significado da palavra dentro de contextos. Um exemplo dado pela própria Google foi a capacidade de entender o uso da palavra “alterar” na frase “alterar o brilho do portátil”, na consulta, para “ajustar”, na frase “ajustar o brilho de um portátil” no documento;
2. **Relevância**, determinar a relevância do conteúdo, pelas presença de palavras-chave, por sua posição na página (por exemplo, no título), e outras formas qua avaliam o conteúdo como um todo a partir de alguns sinais (não especificados);
3. **Qualidade**, determinar a qualidade do conteúdo, também a partir de diferentes sinais, como links ao conteúdo e credibilidade;
4. **Usabilidade**, capacidade de utilização da página, incluindo a experiência na página, facilidade de consulta em dispositivos móveis, tempo de carregamento de conteúdo, etc., e ainda
5. **Contexto** e definições, por exemplo considerando a região onde está sendo feita a busca, a linguagem sendo usada, se a pesquisa é segura, preferências de conteúdo e personalização.

19.1.1 Alguns algoritmos na história da Google

A lista a seguir mostra alguns algoritmos usados em algum momento da história da Google. Nessa história, tanto o Caffeine quanto o Hummingbird foram grandes mudanças, outros, como o Panda, se iniciaram como sistemas separados e foram unificados ao sistema principal.

- **Pagerank**, o algoritmo inicial da Google, tratado na Seção 18.4. Uma versão evoluída ainda é usada nos sistemas de classificação.
- **Caffeine**, a grande atualização do Google de 2010 e que teve um grande impacto nas buscas, permitindo uma atualização contínua do índice do Google, indexando documentos quase em tempo real (Grimes, 2010).
- **Panda** que foi descrito pela Google pela primeira vez em fevereiro de 2011 e já passou por várias versões, tem a finalidade de fornecer sites mais úteis aos usuários, penalizando sites com conteúdo de baixa qualidade. Usa para isso vários fatores de avaliação como a originalidade do conteúdo, a relevância do conteúdo para a consulta de pesquisa e a presença de anúncios excessivos ou pop-ups que atrapalham a experiência do usuário. Ele também leva em consideração a estrutura do site, como a facilidade de navegação e a presença de conteúdo duplicado. O Panda foi uma

das atualizações de algoritmo mais importantes do Google, tendo um grande impacto no tráfego dos sites MOZ, 2023. Desde 2015 foi incorporado aos sistemas de classificação Google, 2023b.

- **Penguin**, de 2012, que penaliza sites que usam técnicas de manipulação da quantidade de links que apontam para uma página. Essa algoritmo passou a fazer parte dos sistemas de classificação em 2016, em sua versão 4.0 Illyes, 2016.
- **TopHeavy** (2012), ou *Page layout algorithm improvement*, que penalizava sites com muitos anúncios ou pouco conteúdo Cutts, 2012.
- **Hummingbird**, foi a reescrita completa do *core* do algoritmo de busca, que evitava conteúdo de má qualidade, interpretava as consultas e não exigia palavras exatamente iguais. Ele foi uma das maiores atualizações da Google, mas com impacto sutil nas buscas, em 2013. Uma de suas características foi passar também a prever também as necessidades da busca a partir de dispositivos móveis, e outra foi entender melhor consultas em linguagem natural. A Google nunca publicou uma explicação de como realmente funcionava Montti, 2022, porém existe uma patente que indica algumas de suas funcionalidades Mahabal et al., 2013. Além disso, permitiu várias outras inovações que vieram a seguir. Continuou a evoluir e hoje é considerado um sistema desativado.
- **Knowledge Graph**, lançado em 2012, é um mecanismo que permite responder perguntas factuais, como “quantas pessoas cabem no Maracanã”. Muitos dados são obtidos diretamente de fontes públicas, mas a Google também licencia algumas informações, como resultados de partidas esportivas, preços de ações, etc. A Google também fornece *knowledge panels* que podem conter informações dados pelas pessoas ou empresas caracterizadas nesses painéis. ’ Google, 2023d; Patel, 2023.
- **Payday Loan Algorithm** (2013), busca melhorar respostas em áreas muito sujeitas a spam, principalmente na área de empréstimos, jogos, pornografia, etc. Cutts, 2013;
- **Pigeon** (2014) e **Possum**¹ (2016), que tratam a questão da localidade na busca MOZ, 2021;
- **RankBrain** (2015), um algoritmo de aprendizado de máquina que atualizou o Humminbird, sendo o primeiro sistema de IA usado na busca;
- **SpamBrain** é mais um sistema de prevenção a spam baseado em IA da Google. Lançado em 2018, só entre 2020 e 2021 o SpamBrain aumentou em 6 vezes a quantidade de sites de spam identificados Google, 2022b. Em 2019, por exemplo, esse sistema detectou mais de 25 bilhões de páginas diariamente com spam Google, 2020.
- **Medic** (2018), que trata especificamente de sites que entram no que a Google chama de “Your Money Your Life”, uma categoria onde a qualidade da informação é de extrema importância, especializado na qualidade de sites com informações de saúde Conversion, 2019.
- **BERT** (2018), um *Large Language Model* que busca o significado da palavra em um contexto específico e será tratado em outro capítulo.
- o **Page Experience** (2020) inclui métricas como o atendimento as características dos dispositivos móveis, segurança https, e os **Core Web Vitals**, cujos três principais são o tempo do maior elemento da página carregar (LCP), o tempo que uma página demora para ser interativa Anderson, 2021. (FID) eo *Cumulative Layout Shift*, que mede quanto as páginas do site mudam de forma inesperada Estrella, 2023.
- **MUM** (2021), um algoritmo 1000 vezes mais poderoso que o BERT Pandu Nayak, 2021, usando uma nova arquitetura denominada T5 Raffel et al., 2019.

Em 2023, como o aparecimento do ChatGPT, a Google, como outras empresas, tentam alinhar um sistema de conversação com a busca. A Google propôs o Bard, capaz de ajudar na busca.

¹Possum não é um nome oficial da Google, mas dado pelo mercado

19.1.2 Sistemas de Classificação do Google

A seguir é apresentada uma lista fornecida pelo Google sobre os vários sistemas que ajudam a ordenação da resposta (Google, 2022c):

- BERT (J. Devlin et al., 2018b);
- Sistemas de informação de crises (Google, 2023a), que trata de alertas e notícias sobre situações de crise, como tempestades, terremotos, e tc.;
- Sistemas de eliminação de duplicação;
- Sistema de domínio de correspondência exata, que busca evitar nomes de domínio criados para atender consultas específicas;
- Sistemas de atualização, que favorecem as informações mais novas;
- Sistema de conteúdo útil (Google, 2023f);
- Sistemas de análise de links e PageRank (Page, 2001; Page et al., 1999);
- Sistemas de notícias locais;
- MUM - Modelo Unificado Multitarefas (Panduk Nayak, 2022);
- Correspondência neural (***Neural Matching***), que olha para páginas e consultas inteiras, buscando entender representações vagas de conceitos (Panduk Nayak, 2022);
- Sistemas de conteúdo originais (Gingras, 2019);
- Sistemas de rebaixamento baseado em remoção, seja ela jurídica ou de dados pessoais solicitados diretamente ao Google.
- Sistema de experiência na página (Anderson, 2021);
- Sistema de classificação de trecho (Raghavan, 2020);
- Sistema de avaliações do produto (Google, 2023e; P. Liu, 2022);
- RankBrain, que associa palavras a conceitos (Panduk Nayak, 2022);
- Sistemas de informações confiáveis (Google, 2011; Pandu Nayak, 2022);
- Sistema de diversidade de sites, que busca não repetir páginas do mesmo site e
- Sistemas de detecção de spam, incluindo o SpamBrain Google, 2020, 2022b.

19.2 Apache Solr

Solr é uma das plataformas de busca mais usadas em situações locais, sendo que seu sistema de indexação é o Lucene, suportando várias funcionalidades, como busca de termos e busca facetada.

A arquitetura de Solr é descrita na Figura 19.3. Ela é baseada no modelo cliente-servidor, sendo que os componentes de interface são os *handlers*, que atendem a *requests*, ou consultas, e a pedidos de atualização. As consultas são fornecidas pelos *Response Writers*. O funcionamento básico implica em um sistema ou usuário enviar uma consulta para o *Requests Handler*, que determina um *Query Parser* apropriado, que usa o Lucene para gerar uma resposta que será apresentada pelo *Response Writer*.

Os pedidos feitos ao Solr são atendidos por *Request Handlers*. Eles são processos configuráveis que indicam como o Solr deve tratar esses pedidos. Entre eles, está o **SearchHandler**, que é um acesso flexível a ação de busca nos índices mantidos pelo Solr e pelo Lucene.

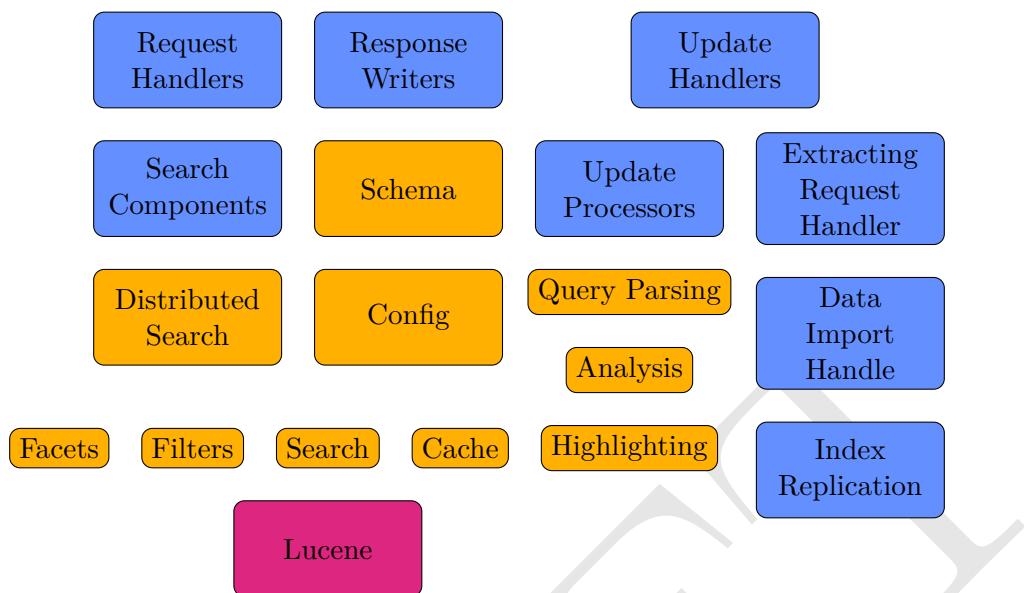


Figura 19.3: Visão geral da arquitetura do Solr. Fonte: (Sun, 2019)

19.3 Lucene

Lucene, uma “biblioteca Java que fornece mecanismos poderosos de indexação e busca” (Apache Software Foundation, 2011a), é o padrão *de-facto* para mecanismos de busca *full-text* a nível organizacional, possivelmente sendo usada dentro de uma ferramenta mais poderosa. Ele é o *engine* que faz funcionar dois outros produtos muito utilizados na indústria, o Solr e o ElasticSearch. Lucene foi desenvolvido por Doug Cutting, que também desenvolveu o Nutch, a partir de 1996, e o nome foi uma homenagem a sua mãe. Segundo Yang, Fang e J. Lin (2018) “Hoje, só um grupo pequeno de empresas [...] usa sua própria infraestrutura de busca. Para todas as outras, o mecanismo de busca de código aberto Lucene [...] se tornou *de facto* a plataforma para construir e implantar aplicações de busca”.

Construído como uma biblioteca, o Lucene só pode ser usado diretamente de Java, integrado ao seu programa, ou por meio de outros sistemas que forneçam interfaces de acesso remotas e guardem seus dados por meio dele, como o Solr. Porém, existem adaptações para usá-lo em outras linguagens. Ele pode ser visto como a parte interna de uma base de dados NoSQL, onde os registros tem formato livre, ou como uma base de dados de estrutura chave-valor sem esquema².

Seu funcionamento básico pode ser dividido em duas funções principais: salvar e recuperar conteúdo em um índice, retornando documentos ordenados por relevância a consulta ou por um campo indexado. Para isso, o Lucene fornece 5 funcionalidades básicas:

- a análise de documentos,
- a indexação de documentos (analisados),
- a construção da consulta,
- , o processamento da consulta e
- a apresentação de resultados.

²Em alguns dos sistemas onde é usado internamente, como o Solr, pode ser necessária existência de um esquema para atender outras características.

A unidade de trabalho do Lucene é um **Document**. Um documento do Lucene não é exatamente um documento, mas um registro com campos. O que está no documento é responsabilidade do programador.

Todos os documentos devem estar indexados. Um índice é composto de vários documentos, e indexar corresponde a adicionar documentos ao índice por meio de um **IndexWriter**. Buscar envolve recuperar documento do índice por meio de um **IndexSearcher**.

Um **Document** é composto de campos, **Fields**. Cada documento pode ter campos diferentes, não existindo um esquema para a base. Um campo é uma dupla par-valor. Cada **Field** pode ser incluído em um **Document** isoladamente. Para criar um campo é necessário: dar um nome ao campo, dizer o valor do campo, e opcionalmente dar um tipo para ele. Para cada campo também devem ser escolhidas algumas propriedades, sendo as mais importantes se o campo deve ser armazenado na base e se o valor do campo deve ser indexado. Documentos propriamente dito normalmente não são guardados diretamente em um campo, mas sim processados para gerar o índice e apontados para algum lugar de origem, como um diretório de documentos ou uma url.

Algumas opções para os campos são: **indexed**, **tokenized**, **stored**, **compressed** e **binary** (que não podem ser indexados).

O Programa 19.1 é um exemplo simples de uso do Lucene. Nele vemos o uso de um **Analyzer**, no caso o **BrazilianAnalyzer**, que, associado a um **IndexWriter**, da linha 30 a 33, vai tokenizar e eliminar as stopwords. Isso permitirá a indexação *full-text*.

Depois disso o programa vai criar três documentos e indexá-los, para depois recuperar os documentos que contém a palavra “texto”.

Programa 19.1: Exemplo do uso do Lucene.

```

1 import java.nio.file.Files;
2 import java.nio.file.Path;
3
4 import org.apache.lucene.analysis.Analyzer;
5 import org.apache.lucene.analysis.pt.PortugueseAnalyzer;
6 import org.apache.lucene.document.Document;
7 import org.apache.lucene.document.Field;
8 import org.apache.lucene.document.TextField;
9 import org.apache.lucene.index.DirectoryReader;
10 import org.apache.lucene.index.IndexWriter;
11 import org.apache.lucene.index.IndexWriterConfig;
12 import org.apache.lucene.queryparser.classic.ParseException;
13 import org.apache.lucene.queryparser.classic.QueryParser;
14 import org.apache.lucene.search.IndexSearcher;
15 import org.apache.lucene.search.Query;
16 import org.apache.lucene.search.ScoreDoc;
17 import org.apache.lucene.store.Directory;
18 import org.apache.lucene.store.FSDirectory;
19 import org.apache.lucene.util.IOUtils;
20
21 import java.io.*;
22
23 public class LuceneTest {

```

```

25  public static void main(String[] args) throws IOException,
26      ↪ ParseException {
27
28      System.out.println("Lá vou eu...");
29
30      // Escolhendo um analisador
31      Analyzer analyzer = new PortugueseAnalyzer(); //
32      Path indexPath = Files.createTempDirectory("tempIndex");
33      Directory directory = FSDirectory.open(indexPath);
34      IndexWriterConfig config = new IndexWriterConfig(analyzer); //
35      IndexWriter iwriter = new IndexWriter(directory, config);
36
37      // criando o documento 1 que vou procurar e colocando o único
38      ↪ campo
39      Document doc = new Document();
40      String text = "Esse é o texto para encontrar.";
41      System.out.printf("Indexei esse: %s\n",text);
42      doc.add(new Field("fieldname", text, TextField.TYPE_STORED));
43
44      // indexando o documento
45      iwriter.addDocument(doc);
46
47      // outro documento, reuso variáveis para mostrar que pode
48      // esse não pode ser achado
49      doc = new Document();
50      text = "Esse não pode ser achado.";
51      System.out.printf("Indexei esse: %s\n",text);
52      doc.add(new Field("fieldname", text, TextField.TYPE_STORED));
53      iwriter.addDocument(doc);
54
55      // outro documento, reuso variáveis para mostrar que pode
56      // esse também tem que ser ser achado
57      doc = new Document();
58      text = "Outro texto para achar.";
59      System.out.printf("Indexei esse: %s\n",text);
60      doc.add(new Field("fieldname", text, TextField.TYPE_STORED));
61      iwriter.addDocument(doc);
62
63      // Fecha o índice para abrir na busca
64      iwriter.close();
65
66      //Preparando para a busca
67      DirectoryReader ireader = DirectoryReader.open(directory);
68      IndexSearcher isearcher = new IndexSearcher(ireader);
69
70      // Criando a busca
71      QueryParser parser = new QueryParser("fieldname", analyzer);

```

```

71         String busca = "texto";
72         Query query = parser.parse(busca);
73         System.out.printf("Busquei esse: %s\n",busca);
74
75         // Fazendo a busca
76         ScoreDoc[] hits = isearcher.search(query, 10).scoreDocs;
77
78         // Passando pelos resultados
79         System.out.printf("Achei %d documentos.\n",hits.length);
80         for (int i = 0; i < hits.length; i++) {
81             Document hitDoc = isearcher.doc(hits[i].doc);
82             String achei = hitDoc.get("fieldname");
83             System.out.printf("Achei esse: %s\n",achei);
84         }
85
86         // Limpando a casa
87         ireader.close();
88         directory.close();
89         IOUtils.rm(indexPath);
90
91         System.out.println("Acabou...");
```

Saída para o Programa 19.1

1 Lá vou eu...
 2 Indexei esse: Esse é o texto para encontrar.
 3 Indexei esse: Esse não pode ser achado.
 4 Indexei esse: Outro texto para achar.
 5 Busquei esse: texto
 6 Achei 2 documentos.
 7 Achei esse: Outro texto para achar.
 8 Achei esse: Esse é o texto para encontrar.
 9 Acabou...

19.3.1 PyLucene

PyLucene (Apache Software Foundation, 2011b) é um conjunto de *bindings* para Lucene em Python, feito em Java. PyLucene inclui uma máquina virtual Java com o Lucene dentro do processo Python. Desta forma, o PyLucene é usado diretamente do programa em Python.

O principal defeito de PyLucene é a enorme dificuldade de instalação. Para instalar o PyLucene é preciso: construir o JCC, uma extensão de Python que usar Python e C++, o que exige um compilador C++; construir o PyLucene, o que exige um JDK, sendo que o recomendado é o Temurin. Vários utilitários devem estar instalados, sendo que alguns não são comuns no Windows, como o `make`. Para a maioria dos usos, é mais interessante usar o Lucene diretamente em Java, ou usar o Apache Solr ou o

ElasticSearch. Ambos incluem o Lucene e permitem chamadas REST. Existem algumas bibliotecas Python que servem de front-end ao Pylucene.

19.4 Whoosh

Woosh é um biblioteca de full-text search nativa em Python, inspirada em Lucene (Chaput, 2012). É um projeto interessante, de código aberto, mas sua última *release*, 2.4.7, foi em 2016. É uma opção razoável para trabalhos acadêmicos, porém não para aplicações profissionais, já que no momento que esse livro estava sendo escrito era lento e possuía bugs, mas não possuia manutenção. Uma vantagem é sua simplicidade, apesar de seguir um modelo bem parecido com o Lucene, então serve bastante para fins educacionais.

Uma das características do Woosh é exigir um esquema, porém os documentos não precisam ter todos os campos. O passo a passo básico é similar ao do Lucene. É necessário criar um índice e usar um `writer` para adicionar documentos no mesmo.

No exemplo do Programa 19.2 é criado um schema e um índice, e depois 3 documentos são inseridos. Também são feitas quatro buscas que mostram como o programa deve se comportar.

Woosh também permite analisadores de vários tipos. Além disso, tudo pode ser estendido ou trocado. Como em Lucene, possui uma linguagem de consulta.

Woosh pode ser lento para muitos documentos, um projeto chamado Pyindex³, no GitHub, afirma conseguir melhores resultados dentro de algumas limitações, como ter um índice estático.

Programa 19.2: Exemplo do uso do Woosh.

```

1 import os.path
2
3 from whoosh.index import create_in
4 from whoosh.fields import Schema, TEXT
5 from whoosh.qparser import QueryParser
6 from whoosh.index import open_dir
7
8 # precisamos de um diretório para guardar o índice
9 if not os.path.exists("indexdir"):
10     os.mkdir("indexdir")
11
12 # Definindo o Schema, campos do documento
13 schema = Schema(titulo=TEXT(stored=True), conteudo=TEXT(stored=True))
14
15 # criar um índice no diretório com o esquema e recuperar um writer
16 ix = create_in("indexdir", schema)
17 writer = ix.writer()
18
19 # inserir três documentos
20 docs = { "Primeiro" : "Este é o primeiro documento",
21           "Segundo" : "Este é o segundo documento, não deve ser achado",
22           "Terceiro" : "Este é o numero 3, deve ser achado com o primeiro"}
```

³<https://github.com/spitis/PyIndex>

```

24 for t in docs:
25     writer.add_document(titulo=t, conteudo=docs[t])
26
27 # Commit as mudanças e fecha o índice
28 writer.commit()
29 ix.close()
30
31 # abre de novo índice para busca
32 sx = open_dir("indexdir")
33
34 # busca e relata
35 with sx.searcher() as searcher:
36     print("Buscando 'primeiro' em conteúdo")
37     query = QueryParser("conteúdo", sx.schema).parse("primeiro")
38     results = searcher.search(query)
39     for r in results:
40         print(" - Encontrei : ", r)
41
42 with sx.searcher() as searcher:
43     print("Buscando 'documento' em conteúdo")
44     query = QueryParser("conteúdo", sx.schema).parse("documento")
45     results = searcher.search(query)
46     for r in results:
47         print(" - Encontrei: ", r)
48
49 with sx.searcher() as searcher:
50     print("Buscando 'Terceiro' em conteúdo")
51     query = QueryParser("conteúdo", sx.schema).parse("Terceiro")
52     results = searcher.search(query)
53     for r in results:
54         print(" - Encontrei: ", r)
55
56 with sx.searcher() as searcher:
57     print("Buscando 'Terceiro' em título")
58     query = QueryParser("título", sx.schema).parse("Terceiro")
59     results = searcher.search(query)
60     for r in results:
61         print(" - Encontrei: ", r)
62
63 sx.close()

```

Saída para o Programa 19.2

```

1 Buscando 'primeiro' em conteúdo
2 - Encontrei : <Hit {'conteúdo': 'Este é o primeiro documento', 'título': 'Primeiro'}>
3 - Encontrei : <Hit {'conteúdo': 'Este é o numero 3, deve ser achado com o primeiro', 'título': 'Terceiro'}>
4 Buscando 'documento' em conteúdo
5 - Encontrei: <Hit {'conteúdo': 'Este é o primeiro documento', 'título': 'Primeiro'}>
6 - Encontrei: <Hit {'conteúdo': 'Este é o segundo documento, não deve ser achado', 'título': 'Segundo'}>
7 Buscando 'Terceiro' em conteúdo

```

8 Buscando 'Terceiro' em titulo

9 - Encontrei: <Hit {'conteudo': 'Este é o numero 3, deve ser achado com o primeiro', 'titulo': 'Terceiro'

19.5 Outras Ferramentas Úteis

Uma abordagem profissional a índices *full-text* certamente se beneficiaria de ferramentas como **Solr** (Apache Software Foundation, 2022a) ou **ElasticSearch**. Bancos de dados como MySQL, PostgreSQL, Oracle e MS-SQL também fornecem funcionalidades de Full Text, assim como SQLite⁴ que pode ser mais simples de usar. **Sphinx** é uma ferramenta que tem maturidade suficiente para ser usada também, e várias bases NoSQL, tanto de documentos como do tipo chave-valor, podem ser úteis.

19.5.1 Terrier

O Terrier⁵ é um mecanismo de busca de código aberto que afirma ser “flexível, eficiente e eficaz, facilmente implantável em grandes coleções de documentos.” Ele busca implementar funcionalidades de indexação e recuperação de última geração e fornecer uma plataforma para o desenvolvimento e avaliação rápidas de aplicativos de recuperação em larga escala. A pesquisa pode ser facilmente realizada em coleções de teste padrão TREC e CLEF. O Terrier é escrito em Java e desenvolvido pelo Grupo de Recuperação de Informação dentro da Escola de Ciência da Computação da Universidade de Glasgow (Macdonald et al., 2012; Ounis, Amati, Plachouras et al., 2006; Ounis, Amati, V. et al., 2005; Ounis, Lioma et al., 2007).

19.5.2 Anserini e Pyserini

Pyserini (J. Lin et al., 2021) é um toolkit em Python de pesquisa reproduzível com representações densas e esparsas, disponível publicamente, baseado no Anserini (Yang, Fang e J. Lin, 2017, 2018), que tem o mesmo objetivo e é baseado no Lucene.

Anserini tem como objetivo criar *baselines* reproduzíveis para pesquisa em recuperação da informação em modelos baseados em *bags of words*. Os criadores se preocupam com a capacidade de reproduzir experimentos e citam que é comum que outros autores façam afirmações vagas em suas descrições, como dizer que usam o BM25 sem citar os parâmetros. Usando Java e Lucene e padrões TREC, seus autores afirmam ter alcançado um sistema escalável, eficiente, que provê modelos de *ranking* modernos e rápidos, com baixa latência.

19.5.3 Ainda outras ferramentas disponíveis

- Capreolus, a toolkit for constructing flexible ad hoc retrieval pipelines that can be run via a Python or command line interface.⁶
- PISA, Performant Indexes and Search for Academia is a text search engine able to run on large-scale collections of documents. It allows researchers to experiment with state-of-the-art techniques, allowing an ideal environment for rapid development.⁷

⁴<https://www.sqlite.org/fts5.html>

⁵<http://terrier.org/>

⁶<https://capreolus.ai/en/latest/>

⁷<https://github.com/pisa-engine/pisa>

- OpenNIR⁸⁹ .
- OpenMatch An all-in-one toolkit for information retrieval¹⁰.
- Tevatron (Gao et al., 2022)
- Lemur, Indri ¹¹
- Zettair ¹²

DRAFT

⁸⁹<https://opennir.net/>

⁹<https://github.com/Georgetown-IR-Lab/OpenNIRACcompleteNeuralAd-HocRankingPipelinefromGeorgetownInformationRetrievalSystem>
citep{MacAvaney2020}

¹⁰<https://github.com/OpenMatch/OpenMatch>

¹¹<https://www.lemurproject.org/>

¹²<http://www.seg.rmit.edu.au/zettair/>

CAPÍTULO 20

CRAWLERS

Uma rápida reação ao aparecimento da WWW foi a necessidade de catalogar seus documentos, dispersos pela rede de servidores httpd. Outros sistemas que tentavam disponibilizar documentos na época, como o Gopher, já possuíam formas de permitir a busca, e até a busca em mecanismos de busca, como os sistemas Archie, que indexava arquivos FTP, Veronica, que indexava o Gopher, ou o WAIS, que indexava bases variadas (Smith e Updegrove, 1993).

A WWW trazia, porém, uma características: os seus arquivos possuíam links para outros arquivos. Assim, ao se indexar um arquivo, era possível encontrar outros arquivos para indexar. Isso deu a oportunidade de criar mecanismos específicos de navegação automática pela rede de documentos, de forma a recuperá-los.

Esses programas, capazes de navegar pela rede recuperando documentos, foram chamados de ***robots***, ***crawlers***, ***spiders*** ou ainda outros nomes, e passaram a ser parte importante dos mecanismos de busca, já que uma forma ingênua de busca seria muito ineficiente. Mais formalmente, Najork (s.d.) define “web crawler é um programa que, dada uma ou mais URLs iniciais, recupera as páginas web associadas a essas URLs, extraí todos os hyperlinks nelas contidas e, recursivamente, continua a recuperar as páginas identificadas por esses hyperlinks”.

O primeiro *crawler*, o World Wide Web Wanderer, criado por Matthew Gray (Najork, s.d.), tinha como finalidade medir o tamanho da WWW.

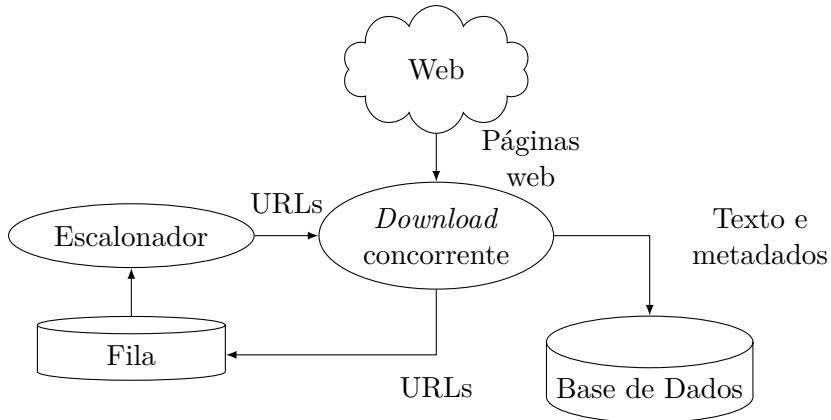
Questões que são levantadas para tornar a navegação e obtenção dos documentos na rede incluem, mas não estão limitadas a:

- Como descobrir um documento que não é apontado;
- Com que frequência voltar a consultar um documento para ver se houve modificações;
- Com que profundidade investigar um servidor/site e quando buscar outro, ou seja, como estabelecer uma boa relação entre a profundidade nos sites e a quantidade de sites buscados;
- Como respeitar os recursos nos sites;
- Como dar prioridade aos sites na busca, e
- Como usar os recursos locais e remotos de forma justa.

Castillo (2004) descreve minimamente a arquitetura de um *crawler* como na Figura 20.1.

Uma sequência básica de ações para um *crawler*, divididas entre o escalonador e o *download*, pode ser descrita como:

Figura 20.1: Arquitetura abstrata de um crawler. Fonte: Castillo (2004)



1. Decidir o próximo link a verificar;
2. Decidir como recuperar o arquivo indicado pelo próximo link;
 - (a) Verificar se o arquivo deve ser recuperado
3. Recuperar o arquivo indicado pelo próximo link;
4. Processar o arquivo recuperado para obter novos links e outras informações sobre esses links;
5. Adicionar os links obtidos, e informações que possam ajudar a decidir qual será o próximo link a verificar, ao conjunto de links a verificar;
6. Entregar o arquivo a outro sistema que vai utilizá-lo, e
7. Voltar ao passo 1.

Todos esses passos podem ser simples ou muito complexos. Por exemplo, em um esquema de crawlers distribuídos, como o proposto por (Mayworm, 2007), todos os passos estão sujeitos a análises de desempenho sobre quanto custa fazer algo em cada nó da rede de crawlers. Porém, é fácil fazer um programa simples em Python que seguirá esse processo de alguma forma ingênua.

Por exemplo, a partir de modelos propostos por Cho, Garcia-Molina e Page (1998) é possível obter um algoritmo básico, algoritmo 20.1, para crawling que exige várias funções: *enfileirar*, que coloca uma url no fim da fila; *primeira*, que remove e retorna a primeira url da fila; *reordenar*, que reordena a fila considerando as informações sobre os links; *recuperarPaginas*, que baixa a página indicada pela url, e *extrairUrls*, que extrai as urls em uma página.

O algoritmo 20.1 funciona para experimentos, porém provavelmente não é eficiente para um sistema real. Entre outras coisas, as atividades são realizadas sequencialmente, enquanto em um sistema real é mais adequado que cada uma dessas funções funcione de forma independente e concorrente. Além disso, algumas páginas devem ser revisitadas periodicamente. Também, em um sistema real, páginas têm que ser revisitadas, pois podem mudar. Finalmente, não mostra como evitar páginas que não devem ser indexadas.

Castillo (2004) e recentemente revisitado por M. Kumar, R. Bhatia e Rattan (2017) tratam essas questões ao indicar que o comportamento de um *Crawler* é “resultado da combinação de políticas”:

- uma política de seleção, que define as páginas a serem visitadas;
- uma política de revisitação, que define quando verificar por mudanças nas páginas;
- uma política de educação (*politeness*), que define como evitar a sobrecarga nos servidores Web;
- uma política de paralelização, que define como coordenar *crawlers* distribuídos Castillo (2004), e

Algoritmo 20.1: Algoritmo básico de um crawler

```

Entrada: urlInicial
1 enfileirar(filaUrls, urlInicial);
2 enquanto filaUrls ≠ ∅ faça
3   url = primeira(filaUrls);
4   pagina = recuperarPagina(url);
5   paginasRecuperadas ← paginasRecuperadas + (url, pagina);
6   novasUrls = extrairUrls(pagina);
7   para u in novasUrls faça
8     se (u ∈ filaUrls) ∧ ((u, -) ∈ paginasRecuperadas) então
9       | enfileirar(urlQueue, u);
10      |
11    fim
12  fim
13 fim

```

- uma política de robuste que indica como o *crawler* trata as armadilhas intencionais ou não que aparecem nos sites e que são prejudiciais ao seu desempenho.

20.1 Tipos de *crawler*

Vários autores fazem uma revisão dos tipos de crawlers que existem, propondo classes criadas a partir das principais características das propostas originais. *Crawlers* usados em sistemas reais, porém, tem provavelmente estruturas híbridas.

Deshmukh e Vishwakarma (2021) e Chang (2022), em revisões recentes, acabam por descrever sete tipos de *crawlers*:

1. Genéricos;
2. *Focused*;
3. Baseados em inferência;
4. Incrementais;
5. Paralelos;
6. Distribuídos, e
7. da Internet das Coisas.

Um *Focused Crawler* (Chakrabarti, 2009; Deshmukh e Vishwakarma, 2021) visita apenas um conjunto desejado de páginas. Esse tipo de *crawler* precisa de um mecanismo de classificação para determinar se uma página deve ser visitada ou não e outro que indica prioridades. Esse tipo de *crawler* usa técnicas de aprendizado de máquina para prever a relevância dos links a serem visitados e escolher a prioridade (Chang, 2022; Deshmukh e Vishwakarma, 2021).

Crawlers baseados em inferência dependem de regras e de um mecanismo de inferência que determinam a qualidade do conteúdo das páginas visitadas (Deshmukh e Vishwakarma, 2021).

A ideia dos *Crawlers* incrementais é visitar as páginas repetidas vezes, melhorando a qualidade da informação devido a atualização das páginas (Chang, 2022; Deshmukh e Vishwakarma, 2021).

Como os nomes dizem, *crawlers* paralelos e distribuídos executam diferentes cópias, respectivamente no mesmo local ou em locais diferentes, buscando maior velocidade ou cobertura da rede (Chang, 2022; Deshmukh e Vishwakarma, 2021).

M. Kumar, R. Bhatia e Rattan (2017) descrevem alguns outros tipos e subtipos de *crawlers*, como aqueles que são dedicados a fóruns. Entre eles, é interessante notar os *crawlers* dedicados a web escondida, que neste texto é chamada de *deep web*, e a ideia de Hammer e Fiedler (2000) de usar *crawlers* móveis, onde o mecanismo de busca envia ao servidor um agente capaz de fazer as buscas no local e enviar o resultado de volta (R. Kumar, 2009).

Castillo (2004) propõe uma classificação baseada em 3 conceitos que seriam propriedades desejadas pelos sistemas que usam o *crawler*:

- Qualidade representacional;
- Qualidade intrínseca, e
- Atualização.

Nesse caso, um sistema de espelhamento está mais interessado na qualidade representacional e na atualização, enquanto um *focused crawler* está muito interessado na qualidade intrínseca e tem algum interesse apenas na atualização. Já um sistema de arquivamento está muito interessado na qualidade representacional e um pouco interessado na qualidade intrínseca.

20.2 O que não indexar

Em 1994, após um debate de robots de busca eram “bons ou maus”, ou seja, se eram prejudiciais ou benéficos para o funcionamento de um site, Koster (1994) propôs como padrão um formato de arquivo que permitiria declarar como um robot deveria se comportar em um site, basicamente avisando que certos arquivos, ou certas partes da estrutura, não deveriam ser consultadas pelo robot.

É importante ver a relevância da discussão: enquanto robots permitiam que um site fosse encontrado, a existência de muitos robots, ou robots mal comportados, poderia esgotar os recursos de um site, sendo equivalentes a um ataque de negação de serviço (DoS Attack). Então, é necessário um equilíbrio entre indexar um site e exigir recursos do mesmo, para benefício do próprio site e da rede em geral. Isso é uma preocupação que um robot deve ter.

Na prática, o protocolo é apenas um arquivo com um formato específico, e simples, que indica que seções de um site não deveriam ser indexadas por um robot. É importante frisar que essa é uma escolha voluntária do robot, que pode tanto não investigar a existência dessa informação no site, quanto intencionalmente quebrar as regras.

Basicamente um arquivo robot.txt é composto de grupos, sendo que cada grupo inicia com o comando **user-agent**, que indica a que robot o grupo se refere, e segue com um grupo de regras que podem ser do tipo **allow** ou **disallow**, que respectivamente informam que o acesso liberado ou indesejado a um arquivo ou um grupo de arquivos.

Um exemplo de arquivo robot.txt é dado no próprio *RFC 9309 Robots Exclusion Protocol* e aparece na figura Figura 20.2. Nesse exemplo existem 3 grupos de regras. A primeira, destinada a todos os robots que não estão nas outras regras, não permite que sejam indexados arquivos .gif ou arquivos no diretório **example**, enquanto permite que o diretório **publications** seja indexado. As seguintes são regras especializadas para robots que se identificam como agentes **foobot**, que não pode indexar nada a não ser dois arquivos; **barbot** e **bazbot**, que não podem indexar uma página específica, e o agente **quxbot**, que tem acesso irrestrito.

Essa proposta se tornou o padrão de fato da indústria, porém só se tornou um RFC em 2022, sendo o *RFC 9309 Robots Exclusion Protocol* (Koster et al., 2022).

```
User-Agent: *
Disallow: *.gif$
Disallow: /example/
Allow: /publications/
User-Agent: foobot
Disallow:/
Allow:/example/page.html
Allow:/example/allowed.gif
User-Agent: barbot
User-Agent: bazbot
Disallow: /example/page.html
User-Agent: quxbot
```

Figura 20.2: Exemplo de arquivo robot.txt. Fonte: Koster et al. (2022)

Tabela 20.1: Alguns identificadores de agentes

Nome do Agente	Identificador do Agente	Referência
Googlebot	Googlebot	(Google, 2022a)
APIs-Google	APIs-Google	(Google, 2022a)
AdesBot Mobile Web Android	AdsBot-Google-Mobile	(Google, 2022a)
Mobile AdSense	Mediapartners-Google	(Google, 2022a)
Bingbot	bingbot	

Quanto aos código dos agentes, é necessário investigar. A Google, por exemplo, informa 19 agentes diferentes (Google, 2022a). A Tabela 20.1 informa alguns agentes:

O arquivo `robots.txt` nomeia áreas do seu site ou arquivos que não devem ser buscados por um *crawler*.

Ele é instalado na raiz de uma estrutura de domínio.

Existem outras formas de evitar que uma página seja indexada, e que deve ser observada por robots. Uma delas é uma *meta-tag* do HTML com nome `robots` e o conteúdo `noindex`, como na Programa 20.1. Outra é usar um rótulo `X-Robots-Tag: noindex` no cabeçalho da resposta.

Programa 20.1: Exemplo de como evitar robots no HTML.

```
1 <html><head>
2 <meta name="robots" content="noindex" />
3 ...
4 </head>
5 <body>...</body>
6 </html>
```

20.2.1 Sitemaps

Sitemaps “são arquivos usados para fornecer informações sobre páginas, vídeos e outros arquivos do site e indicar a relação entre eles. Os mecanismos de pesquisa, como o Google, leem esses arquivos para rastrear seu site com mais eficiência” (Google, 2022d). Eles podem ser criados com vários formatos: XML, que pode ser visto em Sitemaps.org (2022), RSS, mRSS, Atom 1.0 e texto. Podem ser vistos como a ideia contrária dos arquivos robots.txt, pois eles são feitos para indicar ao mecanismo de busca o que se deseja que ele indexe.

Alguns gerenciadores de conteúdo, como o Wordpress, geram sitemaps automaticamente. Para consultá-lo, basta baixar o arquivo `sitemap.xml`.

20.3 Qualidade do *crawling*

Considere um conjunto de páginas U , onde cada página u tem uma importância $I(u)$. Um *crawler* ótimo que visite K páginas em U deve visitar as $|K|$ páginas mais importantes, u_1, u_2, \dots, u_k onde $\forall i, j | i \leq k \wedge j > k, I(u_i) > I(u_j)$. Ou seja, a importância de uma página em K é sempre maior que a importância que uma página em $U - K$. Seja R uma coleção qualquer de páginas recuperadas, que possui um subconjunto M , $M = R \cap K$, que são importantes (Chakrabarti, 2018; Cho, Garcia-Molina e Page, 1998).

A métrica de importância pode ser construída de várias formas. Cho, Garcia-Molina e Page (1998) sugere tanto métricas muito dinâmicas, como a similaridade a uma consulta, quanto métricas que podem ser vistas como estáticas em relação a um estado da rede, como o *PageRank* Brin e Page, 1998.

Uma figura de mérito possível para R deve comparar as páginas recuperadas em com as páginas que deviam ser recuperadas (Chakrabarti, 2018), M/K , que é na prática o conceito de revocação apresentado Seção 9.2.

Outra forma possível de calcular a qualidade Q é considerar o peso das importâncias, como em:

$$Q(M) = \frac{\sum_{i=1}^{|K|} I(u_i)}{\sum_{j=1}^{|M|} I(u_j)} \quad (20.1)$$

Na prática, várias medidas podem ser usadas para configurar a importância. Uma delas é o *PageRank* (Brin e Page, 1998; Cho, Garcia-Molina e Page, 1998).

20.4 Como decidir o próximo link a visitar?

Uma das questões mais importantes ao construir um *crawler*, e que definirá todo o seu desempenho, independente da medida de qualidade escolhida, é a escolha do próximo link a visitar (Cho, Garcia-Molina e Page, 1998). Essa é uma tarefa de previsão e que visa maximizar a qualidade coleção de páginas visitadas e o uso da rede.

Esse problema pode ser tratado a partir de várias premissas (Cho, Garcia-Molina e Page, 1998):

- visita a um número limitado de páginas;
- visita a um número limitado de páginas, considerando um limite mínimo de qualidade para as páginas, e

- visitação sujeita a um limite de páginas na memória, que exige que, depois de um limite, páginas de menor qualidade sejam eliminadas da memória, mantendo a coleção em um tamanho fixo.

Crawlers podem ser construídos em torno de objetivos que eles se propõe a cumprir. Duas abordagens genéricas podem ser consideradas, atualmente pensadas de forma híbrida: ter objetivos relativos ao grafo, como fazer uma busca a mais ampla possível, e só depois aprofundar (*breath-first*), ou ter objetivos relativos a tópicos, o que é chamado de **Focused Crawling** (Chakrabarti, 2009).

A questão mais geral é como priorizar o *crawling*, tendo em vista o enorme tamanho da Web, que pode ser considerada infinita, o que implica na escolha de uma forma de avaliar a qualidade tanto da coleção quanto do próximo documento a ser recuperado (Castillo, 2004). Essa estratégia pode levar em conta fatores como: assuntos, profundidade da página no site, propriedades de sites similares, padrões de navegação, padrões de construção de site, feedbacks obtidos nos mecanismos de busca, logs, taxas de atualização das páginas, etc.

O **escalonador** é a parte do *crawler* responsável por priorizar as urls a serem visitadas. As estratégias de escalonamento podem ser divididas entre estratégias de longo prazo, onde se busca recuperar primeiro as páginas mais importantes, e estratégias de curto prazo, onde se deseja usar a conectividade da rede de forma eficiente (Castillo, 2004).

Com o objetivo de capturar um conjunto de páginas com o maior PageRank agregado possível dentro de subconjunto de tamanho limitado de uma rede, Castillo (2004) propõe várias estratégias baseadas em propriedades do grafo da rede, definindo três grupos principais:

- Estratégias sem informação extra além da obtida no *crawling*
- Estratégias com informação histórica, baseadas no PageRank do último *crawling*, e
- Estratégias com toda a informação da rede.

Entre as estratégias com apenas a informação do *crawling*, podem ser feitas estimativas do *PageRank* a partir dos dados conhecidos até um certo momento do *crawling*, como usar aproximações, como o número de referência a uma url.

Quanto a estratégias de curto prazo, Mayworm (2007) implementou um crawler com arquitetura distribuída na forma peer-to-peer, onde crawlers colaborativos buscavam balancear a carga e otimizar o uso da rede em cada nó. Nesse crawler, cada nó fica responsável por um servidor, identificado pelo endereço IP, em função do desempenho do nó para recuperar as páginas daquele servidor.

20.5 Arquitetura de um *crawler*

Um dos crawlers de código aberto mais poderosos disponíveis é o Apache Nutch, escrito em Java (Apache Software Foundation, 2022c). Ele é usado no Apache Solr (Apache Software Foundation, 2022b), uma ferramenta para busca planejada para o nível organizacional, junto com ferramentas de indexação como o Apache Lucene. Todas essas ferramentas, ao contrário da maior parte do que é mostrado neste texto, são feitas em Java.

A arquitetura do Nutch¹, apresentada na Figura 20.3, é altamente paralela. Nele, A *Crawl db* mantém informações sobre todas as páginas conhecidas, incluindo plano e status de recuperação e

¹A informação sobre a arquitetura do Nutch é razoavelmente confusa, sendo que alguns autores praticamente eliminam o Link db, enquanto outros parecem misturar outras partes do Solr como funções do Nutch. Além disso, no tempo, o Nutch parece ter se tornado mais dedicado ao Crawler, e outra

metadados. Já o *Link db* mantém a informação sobre os links que chegaram, incluindo o texto âncora e de onde foram obtidos, a base de segmentos contém o conteúdo das página em sua forma original, e também processados pelo *parser*, incluindo os meta-dados descobertos e para onde ela aponta. Além disso, possui o texto para indexação e outros índices opcionais (Apache Software Foundation, 2022c; Cafarella e Cutting, 2004; Nagel, 2014; Srivastav, 2014; Thang, 2016).

O processo de execução do Nutch é:

1. Injetar as URLs sementes (alimentar o Link db que inicia vazia);
2. Gerar uma nova lista de segmentos a obter;
3. Recuperar o conteúdo;
4. Analisar (*Parse*) o conteúdo;
5. Atualizar o Crawl db com informação dos segmentos, e
6. Atualizar o Link db com informação dos segmentos.

Normalmente o Nutch interage com um indexador, como o Lucene (Apache Software Foundation, 2011a), sendo considerado por alguns um “último passo” da sua execução. Em sua concepção original (Cafarella e Cutting, 2004) era um mecanismo de busca na web, mas atualmente é mais visto como um *crawler* no ambiente Solr (Apache Software Foundation, 2022b).

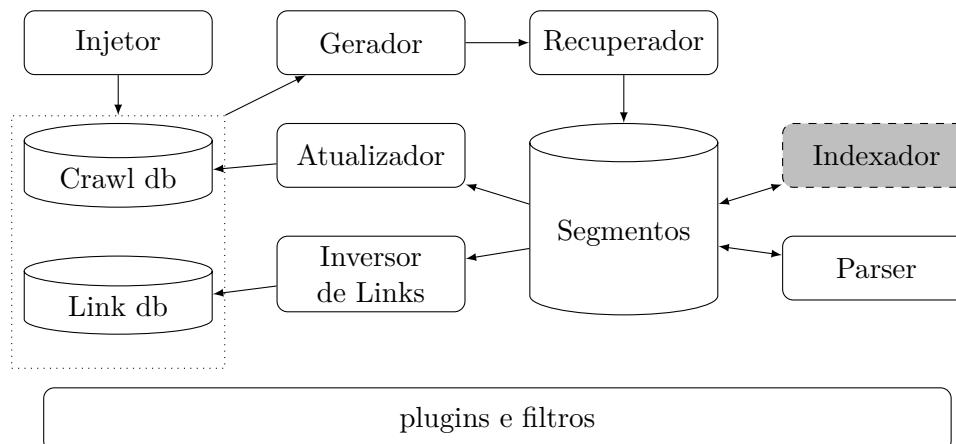


Figura 20.3: Arquitetura do Nutch

20.6 Armadilhas para *crawler*

Uma **armadilha para crawler** é uma url que pode gerar uma navegação infinita ou longa demais para interessar a um *crawler*. Um exemplo disso é uma página de um dia de calendário que tenha um link para o link seguinte, gerado automaticamente. Um crawler poderia ficar infinitamente navegando nos links de “próximo dia” e ficar preso nessa navegação (Castillo, 2004).

Existem algumas alternativas simples para tratar isso (Castillo, 2004):

- Baixar apenas páginas estáticas;
- Baixar páginas dinâmicas com apenas um conjunto de valores nos seus parâmetros;
- Limitar o número de páginas;
- Limitar o número máximo de páginas por domínio, e
- Limitar o número de níveis por web site.

20.7 Tópicos relacionados ao *crawling*

20.7.1 Scrapping

Um das tarefas que pode estar associada, em diferentes graus de coesão, com o *crawling* é o **scrapping** ou, com um nome traduzido mas pouco usado, **raspagem**. Esta tarefa consiste em extrair, de uma página, informações importantes, muitas vezes a partir de campos pré-definidos. Em geral, extrair o texto de uma página já pode ser considerado *scrapping*, apesar de não ser um termo normalmente utilizado nesse contexto.

Todo sistema de *crawling* é obrigado a fazer algum *scrapping*, já que tem que descobrir, na página sendo lida, as próximas páginas sendo lida, extraindo a informação dos links que obtém.

Na maioria dos sistemas de indexação e extração de dados, essas tarefas podem ser feitas de maneira totalmente autônoma. O *crawler* pode simplesmente guardar o HTML obtido, ou outro formato, em alguma base temporária para posterior processamento.

20.7.2 Deep web

Deep Web é o termo dado a partes da WWW que não podemos atingir diretamente por uma URL fixa. Como exemplo, vários sites que na prática são aplicações e a consulta deve ser feita dinamicamente, possivelmente usando comandos internos em javascript.

Não se deve confundir a *deep web* com a **dark web**, o que é comum em artigos jornalísticos. *Dark web* é o nome dado a web “ilegal”, ou a sites que usam alguma técnica que evitar que sejam encontrados, ou manter sua anonimidade, para evitar complicações legais em algum país, por questões políticas, ou outras. A *deep web* é facilmente encontrada, é pública, mas é difícil de consultar de forma automática, não necessariamente de forma intencional pelos criadores do site, mas por causa das dificuldades que algumas tecnologias geram para fornecer consultas automáticas.

Por exemplo, em um site que guarde informações sobre produtos a venda pode ser que a única forma de chegar a um produto é conhecer o seu nome. Nesse caso, um *crawler* seria obrigado a conhecer o nome de todos os produtos para poder visitar todas as páginas. Como isso não é possível em geral, essa informação, mesmo sendo pública, é considerada profunda. O projetista do site, pode, por algum meio, fornecer informações suficientes para o *crawler* atingir todas as páginas, como ter uma lista de todas elas em algum lugar que o consumidor normal do site não esteja interessado em ler.

20.8 Aplicações de *crawling*

A principal aplicação usada para estudar os *crawlers* são os mecanismos de busca na Web, tratados no Capítulo 19. Porém, outras aplicações que seguem a mesma estrutura são possíveis.

Uma delas é a recuperação de Tweets, já que é possível seguir trilhas como *replies* e *retweets*, além de navegar as redes de seguidores².

Crawlers também podem ser usados para testar sites. Nessa aplicação eles são feitos para se manter dentro de um domínio e procuram encontrar todos os links.

²<https://github.com/twitterdev/search-tweets-python>

20.9 Exemplo de implementação em Python

Criado para servir como um exemplo de conceito, o crawler do Programa 20.2 é bastante ingênuo e supõe uma rede com poucos recursos. Por exemplo, ele não trata o cabeçalho das resposta, que pode conter mensagens de erro, mensagens de redirecionamento, e outros tipos. Também não tem nenhuma defesa contra armadilhas. O código, porém, mostra as ações básicas de um crawler, seguindo o algoritmo 20.1.

- O código define uma classe de objetos Crawler, que pode ser inicializada com uma url ou uma lista de urls;
- Mantém duas listas, uma de urls a visitar, `fila_url`, e outra de urls já visitadas, `paginas_recuperadas`;
- A política de seleção de links é a de busca em largura, já que todas as páginas encontradas em uma página são imediatamente colocadas na fila, o que é feito em `extrair_urls`;
- Nunca revisita uma página já visitada, o que é garantido na função `enfileirar`, que testa a presença da url nas duas listas antes de incluir em `fila_url`;
- Evita urls claramente erradas recuperadas pela função `extrair_urls` com uma validação na linha 33;
- Após recuperar a página na função `recuperar_pagina`, dá de 1 a 3 segundos de espera após recuperar uma página, na linha 25;
- É frágil quanto a defesas existentes em vários sites, que evitam pedidos muito rápidos e analisam também o agente, não conseguindo abrir alguns deles;
- Declara um user-agent no `request`, linha 21, o que falta em muitos exemplos e evita que o `request` seja recusado por alguns sites;
- Na função `crawl`, que recebe uma url, executa a recuperação de página e depois extrai as urls dela, enfileirando todas as urls encontradas;
- Possui o comando `run(maximo=5)` para recuperar as páginas a partir das urls na fila, com um limite máximo de páginas a serem recuperadas, que é 5 por *default*;
- Usa a função `print` para ficar mais interativo e comprehensível, mas deveria usar funções de *logging* com diferentes tipos graus de importância, e
- Não respeita o arquivo robots.txt.

Programa 20.2: Um crawler simples em Python que respeita um intervalo de 1 a 3 segundos.

```

1 from urllib.parse import urljoin
2 import urllib.request
3 from bs4 import BeautifulSoup
4 import validators
5 from random import randint
6 from time import sleep
7
8 class Crawler:
9
10    def __init__(self, urls=[]):
11        self.paginas_recuperadas = []
12        if type(urls) == list:
13            self.fila_urls = urls
14        elif type(urls) == str:
15            self.fila_urls = [urls]
16        else:
17            raise Exception("use uma url ou lista de urls e não "+urls)

```

```

18
19     def recuperar_pagina(self, url):
20         texto = "<HTML></HTML>"
21         the_headers = {'User-Agent': 'Experimental 1.0'} #
22         req = urllib.request.Request(url, headers=the_headers)
23         with urllib.request.urlopen(req) as response:
24             texto = response.read()
25             sleep(randint(1,3)) #
26             self.paginas_recuperadas.append(url)
27             return texto
28
29     def enfileirar(self, url):
30         if url and \
31             url not in self.paginas_recuperadas and \
32             url not in self.fila_urls and \
33             validators.url(url): #
34                 self.fila_urls.append(url)
35
36     def primeira(self):
37         return self.fila_urls.pop(0)
38
39     def extrair_urls(self, url_base, html):
40         soup = BeautifulSoup(html, 'html.parser')
41         for referencia in soup.find_all('a'):
42             nova_url = referencia.get('href')
43             if nova_url and nova_url.startswith('/'):
44                 nova_url = urljoin(url_base, nova_url)
45                 yield nova_url
46
47     def crawl(self, url):
48         try:
49             html = self.recuperar_pagina(url)
50             try:
51                 for url_candidata in self.extrair_urls(url, html):
52                     self.enfileirar(url_candidata)
53             except Exception as e:
54                 print("Não enfileirou porque "+str(e))
55             except Exception as e:
56                 print("Não recuperou porque "+str(e))
57
58
59     def run(self, maximo = 5):
60         vistos = 1
61         while self.fila_urls and vistos <= maximo:
62             url = self.primeira()
63             print(f'Crawling {vistos}: {url}')
64             try:
65                 self.crawl(url)

```

```

66         except Exception as e:
67             print(f'Failed to crawl: {url} '+str(e))
68     finally:
69         vistos += 1
70
71 if __name__ == "__main__":
72     d = Crawler(["http://www.umsite.com/"])
73     d.run(5)

```

A classe `Crawler` do Programa 20.2 pode ser extendida para tratar os arquivos `robots.txt`, o que é feito no Programa 20.3. Nessa classe é feita uma nova função `recuperar_pagina` que, antes de recuperar a página verifica se ela pode recuperar. Para isso, é mantida uma lista, `sites_visitados` indexada pelo nome do domínio do site e onde os valores são um objeto `robotparser`, que permite testar se uma url pode ser visitada, com a função `can_fetch`. O arquivo `robot.txt` só é buscado, então, uma vez por site. Além disso, caso não exista um arquivo `robot.txt`, ou ele não possa ser recuperado, o programa considera que pode baixar a página.

Programa 20.3: Crawler com tratamento da exclusão pelos robot.txt

```

1 from crawlersimples import Crawler
2
3 import urllib.robotparser as urobot
4 from urlparse import urlparse
5
6 class CrawlerRobot(Crawler) :
7
8     def __init__(self, urls=[]):
9         super().__init__(urls)
10        self.sites_visitados = {}
11
12    def recupera_robot(self, dominio):
13        try:
14            rp = urobot.RobotFileParser()
15            rp.set_url(dominio+"robots.txt")
16            rp.read()
17            self.sites_visitados[dominio] = rp
18        except Exception as e:
19            print("Erro no robot "+str(e))
20
21    def checa_robot_url(self,url):
22        try:
23            dominio = urlparse(url).scheme+"://"+urlparse(url).netloc+"/"
24            if dominio not in self.sites_visitados:
25                self.recupera_robot(dominio)
26            if rp := self.sites_visitados[dominio]:
27                return rp.can_fetch("*",url)
28            else:
29                return True
30        except Exception as e:
31            print("erro na checagem de robot "+e)

```

```

32         return True
33
34     def recuperar_pagina(self, url):
35         if self.checa_robot_url(url):
36             return super().recuperar_pagina(url)
37         else:
38             print("Não posso visitar (robot.txt) "+url)
39             return ""
40
41 if __name__ == "__main__":
42     d = CrawlerRobot(["http://www.umsite.com/"])
43     d.run(6)

```

20.9.1 Testando Crawlers

Muito cuidado deve ser tomado ao testar,e usar, *crawlers* na web, pois em algum momento vai ser necessário usar servidores que não pertencem ao desenvolvedor, e isso deve ser feito respeitando o desempenho do mesmo.

No início, é mais fácil estabelecer um servidor próprio com links internos, e fazer o *crawler* apenas na sua máquina. Limitar o número de páginas visitadas, como é feito na função `run` do Programa 20.2 é importante. Mais tarde, se for necessário testes mais longos em web sites reais, é melhor fazê-lo em horas de baixo uso.

Todas as impressões, e ainda outras, feitas nos programas exemplos podem ser alteradas para um log em um arquivo, o que pode facilitar o debug.

Em 2022, dois sites existentes podem ajudar a testar *crawlers*. O mais relevante é o “Crawler Test two point oh!”³. Esse site fornece dezenas de testes que criam situações que um *crawler* genérico vai encontrar, divididas em 13 grupos:

- *Mobile*, como uma página servida dinamicamente;
- *Description Tags*, como tags duplicadas ou ausentes;
- *Encoding*, como codificações em várias línguas;
- *Titles*, como títulos duplicados, ausentes, longos demais e com espaços;
- *Robots Protocol*, como proibição de navegação profunda;
- *Redirects*, como redireção infinita e códigos específicos;
- *Links*, como links internos e externos quebrados;
- *URLs*, como diferentes erros na URL e urls com caracteres codificados;
- *Caninical Tags*, como tags no cabeçalho ou auto-referencias;
- *Status Codes*, como todos os tipos de respostas HTTP;
- *Social Tags*, como passar do tamanho máximo;
- *Content*, como página de erro e ausência de H1 e
- *Other*, como HTML quebrados e tempo longo de carregamento.

O site interessante é o *Web Scraping Sandbox*⁴. Ele fornece dois subsites, um com uma loja e outro com citações, que podem ser usados para crawling e scrapping.

³<https://crawler-test.com/>

⁴<https://toscrapе.com/>

20.9.2 Recursos para Crawling em Python

Chang (2022) raz uma lista de bibliotecas clássicas que são usadas em de *crawlers*, entre elas algumas em Python como BeautifulSoup (Richardson, 2007), `Urllib`, `Requests`, `Mechanize`, `Selenium`, `Robotbrowser`, `Scrapely`. Além disso sugere três bancos de dados NoSQL para implementar a base de dados: MongoDB, Redis, ou Cassandra. Aponta ainda arcabouços de *crawler*, sendo três em Python: Scrapy⁵, PySpider⁶ e Robot Framework⁷.

20.10 Exercícios

Exercício 20.1:

Altere o *crawler* exemplo para usar uma lista de prioridades em Python, em vez de uma lista comum. Para isso calcule uma prioridade com as seguintes regras:

1. Uma url cujo nome é apenas o nome de domínio ganha 1000 pontos de prioridade;
2. A primeira url de um domínio não visitado ganha 500 pontos de prioridade;
3. Toda url cujo nome de arquivo contém “index” ganha 250 pontos de prioridade;
4. Toda url que acaba em “htm” ou “html” ganha 100 pontos de prioridade, e
5. Toda url ganha 100 de prioridade e desconta o seu tamanho da sua prioridade.

Exercício 20.2:

Altere o exercício anterior para somar 50 na prioridade a cada vez que uma url ainda não recuperada for enfileirada de novo.

Exercício 20.3:

Altere os crawlers para fazer todos os testes de url, como validade e se pode buscar, no momento de enfileirar a url, incluindo aí adiantar a busca do robots.txt.

Exercício 20.4:

Altere os crawlers para serem configurados para limitar a profundidade que navega em cada site.

Exercício 20.5:

Altere o *crawler* exemplo para ler o cabeçalho da resposta e tratar os status HTTP de re-direção.

Exercício 20.6:

Altere o *crawler* exemplo para se manter no domínio do primeiro link enviado e reportar os problemas encontrados, principalmente erros em urls e páginas não encontradas.

⁵<https://scrapy.org/>

⁶<http://docs.pySpider.org/en/latest/>

⁷<https://robotframework.org/>

Parte IV

Recursos

CAPÍTULO 21

RECURSOS LEXICAIS

Em muitas técnicas relacionadas ao tratamento de texto, e principalmente naquelas que se baseiam na Línguistica Computacional, é interessante usar informações sobre as palavras. Essas informações podem ser desde uma lista de todas as palavras que devem ser consideradas existentes como também incluir propriedades e relações morfológicas, semânticas, ou outros, tantos gerais de uma língua quanto para um uso específico (Freitas, 2022).

Entre os recursos mais usados em recuperação de informação estão lista de *stopwords*, **tesauros**, que são listas de sinônimos, redes de palavras (*wordnets*), bases de conhecimento e ontologias (Freitas, 2022).

(Freitas, 2022), como exemplo, cita o OpLexicon¹, um léxico de sentimento da língua portuguesa (Souza e Vieira, 2012; Souza, Vieira et al., 2011). Nesse léxico, cada palavra está classificada segundo sua forma gramatical, sua polaridade, ou seja, se é positiva ou negativa, e como a atribuição foi dada, manual ou automática.

Ainda segundo Freitas (2022), ontologias fornecem um conhecimento estruturado e estático, o que, em sentido amplo, incluem taxonomias. Já as bases de conhecimento seriam mais flexíveis e dinâmicas. Outras formas de representação semelhantes seriam as *wordnets* e *framenets*.

21.1 Wordnets

WordNet é uma grande base de dados léxica, semelhante a um dicionário ou uma ontologia, mantida pela Universidade de Princeton, que cobre a maioria dos substantivos, verbos, adjetivos e advérbios em inglês, com mais de 125 mil significados (Fellbaum, 1998; G. A. Miller, 1995; Princeton University, 2010). Essa base é desenvolvida e mantida manualmente por uma equipe de psico-linguistas há mais de 25 anos.

Apesar de WordNet ser o nome da base original, ele foi adotado por outras bases similares, como a EuroNet a Open Multilingual Wordnet², a Open English WordNet³ e outras versões para inglês e

¹<https://www.inf.pucrs.br/linatural/wordpress/recursos-e-ferramentas/oplexicon/>

²<http://compling.hss.ntu.edu.sg/omw/>

³<https://en-word.net/>

outras línguas. Em português, entre outras, existe openWordnet-PT⁴ (V. d. Paiva, Rademaker e Melo, 2012), o WordNet.BR⁵ (da Silva, s.d.), e o PULO⁶ (Simões e Guinovart, 2014).

A WordNet possui vários usos, sendo os principais no Processamento de Linguagem Natural baseado em conhecimento linguístico e no apoio a ferramentas de aprendizado de máquina com texto. Em seu site⁷ é possível consultar a base, como mostram as figuras 21.1 e 21.2.

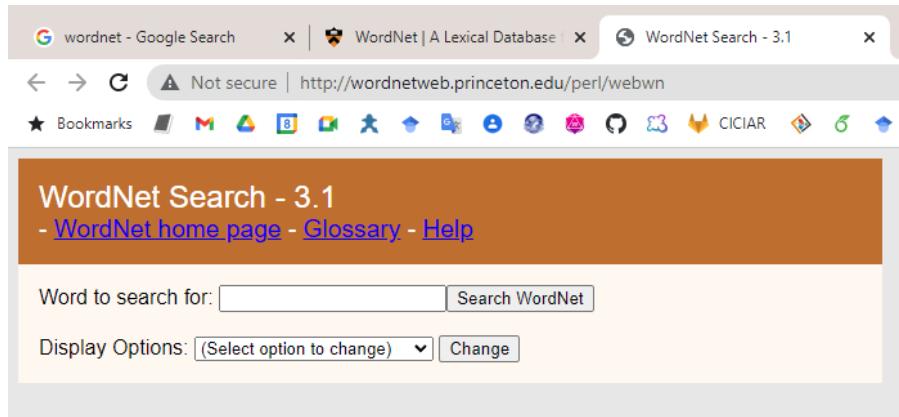


Figura 21.1: Tela de consulta no site da WordNet

Alguns sites utilizam a base da WordNet para visualizações, como o Visual WordNet⁸, Figura 21.3 , e o Visuwords⁹, Figura 21.4.

Existem várias abordagens similares a WordNet criadas para o português:

- PULO;
- WordNet.PT¹⁰;
- Onto.PT¹¹
- PAPEL¹²;
- TeP¹³, e
- OpenWordnet-PG¹⁴.

A principal característica da WordNet é a organização das palavras por significado, em conjuntos de sinônimos, conhecidos por *synsets*. Além disso, são criadas várias relações entre esses conjuntos. No caso de substantivos temos, por exemplo, as relações de hiperonímia/hiponímia, ou de generalização/especialização e a holonímia/meronímia, ou todo/parte, e termos coordenados, isto é, que compartilham um mesmo hiperônimo. No caso de verbos, há a relação de implicação (*manner*), ou forma de (fazer algo), e a hiperonímia (Freitas, 2022).

A Figura 21.5 mostra um exemplo da relação de hiperonímia presente no WordNet, cuja denominação é “is-a”, que é uma relação transitiva. Assim, como {*convertible*} é um {*car*} e {*car*}, por sua vez, é um

⁴<https://github.com/own-pt/openWordnet-PT>

⁵Disponibilizado como TeP em <http://www.nilc.icmc.usp.br/tep2/>

⁶<http://wordnet.pt/>

⁷<https://wordnet.princeton.edu/>

⁸<https://www.visual-thesaurus.com/wordnet.php>

⁹<https://visuwords.com/>

¹⁰no momento que este livro está sendo escrito, sem acesso

¹¹http://ontopt.dei.uc.pt/index.php?sec=download_ontopt

¹²<https://www.linguateca.pt/PAPEL/>

¹³<http://www.nilc.icmc.usp.br/tep2/index.htm>

¹⁴<https://github.com/own-pt/openWordnet-PT>

WordNet Search - 3.1
[- WordNet home page](#) - [Glossary](#) - [Help](#)

Word to search for: talk

Display Options: (Select option to change) ▾

Key: "S:" = Show Synset (semantic) relations, "W:" = Show Word (lexical) relations
 Display options for sense: (gloss) "an example sentence"

Noun

- S: (n) [talk](#), [talking](#) (an exchange of ideas via conversation) "*let's have more work and less talk around here*"
- S: (n) [talk](#) (discussion; ('talk about' is a less formal alternative for 'discussion of')) "*his poetry contains much talk about love and anger*"
- S: (n) [talk](#) (the act of giving a talk to an audience) "*I attended an interesting talk on local history*"
- S: (n) [lecture](#), [public lecture](#), [talk](#) (a speech that is open to the public) "*he attended a lecture on telecommunications*"
- S: (n) [talk](#), [talk of the town](#) (idle gossip or rumor) "*there has been talk about you lately*"

Verb

- S: (v) [talk](#), [speak](#) (exchange thoughts; talk with) "*We often talk business*"; "*Actions talk louder than words*"
- S: (v) [talk](#), [speak](#), [utter](#), [mouth](#), [verbalize](#), [verbalise](#) (express in speech) "*She talks a lot of nonsense*". "*This depressed patient does not verbalize*"
- S: (v) [speak](#), [talk](#) (use language) "*the baby talks already*", "*the prisoner won't speak*", "*they speak a strange dialect*"
- S: (v) [spill](#), [talk](#) (reveal information) "*If you don't oblige me, I'll talk!*", "*The former employee spilled all the details*"
- S: (v) [spill the beans](#), [let the cat out of the bag](#), [talk](#), [tattle](#), [blab](#), [peach](#), [babble](#), [sing](#), [babble out](#), [blab out](#) (divulge confidential information or secrets) "*Be careful--his secretary talks*"
- S: (v) [lecture](#), [talk](#) (deliver a lecture or talk) "*She will talk at Rutgers next week*", "*Did you ever lecture at Harvard?*"

Figura 21.2: Resposta a consulta "talk", fornecida no site da WordNet.

{*vehicle*}, então {*convertible*} é um {*vehicle*}. A notação de conjunto é usada para mostrar que esse é um conceito, não apenas a palavra.

A relação “is-part-of” representa a meronímia, e é exemplificada na Figura 21.6. Ela também é transitiva, logo um {*cylinder*} é parte de um {*car*}.

Entre os verbos, que representam eventos na WordNet, as relações são outras. Uma delas é a “manner”, ou seja, a que diz uma forma em que um conceito mais geral pode ser acontecer. Pelo VisualWords é possível navegar na palavra *talk* e seus vários significados, entre eles o verbo *talk*, no conceito {*talk*, *speak*} e o substantivo *talk* no conceito {*talk*, *talking*}. O conceito {*talk*, *speak*} “é um tipo de” {*communicate*, *intercommunicate*}.

As relações transformam o WordNet em uma grande rede semântica, porém apenas com as relações hierárquicas, a rede é desconectada. Para auxiliar a conexão da rede foi criado o conceito de domínio(Bentivogli et al., 2004). Um domínio é um *synset* que define uma área a que vários outros *synsets* pertencem, assim {*medicine*} é um tópico a que pertencem outros *synset* como {*patient*}.

Mais recentemente, os *synsets* do WordNet receberam explicações para os seu significado, o que é conhecido como *glosses*.

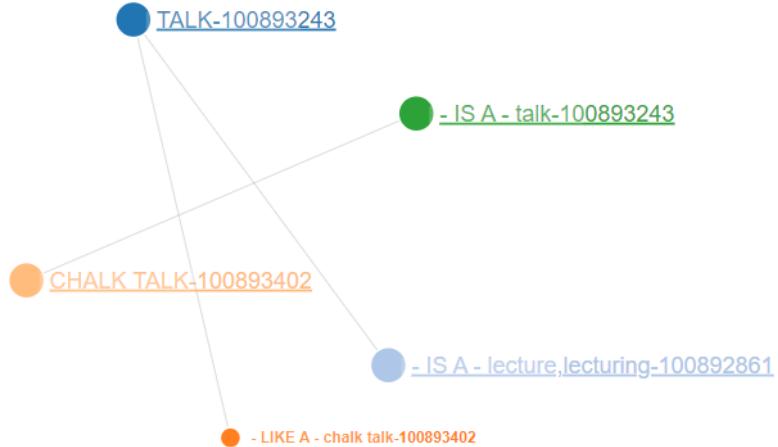


Figura 21.3: Visualização da WordNet pelo Visual Wordnet

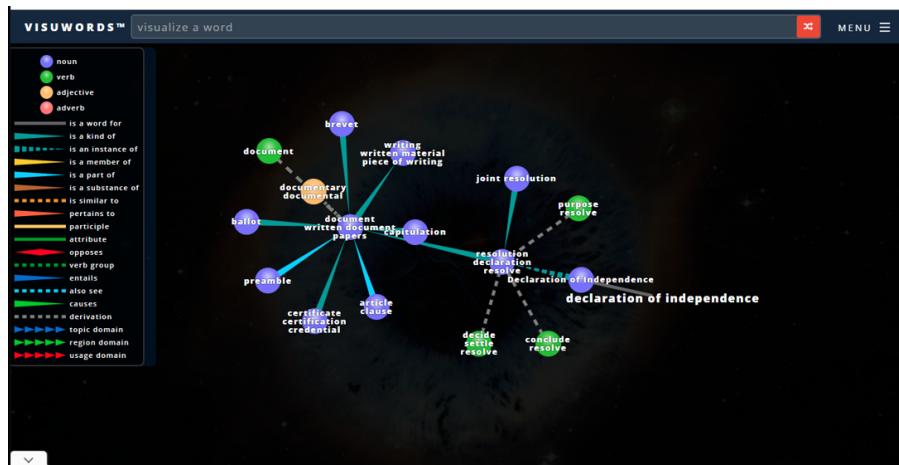


Figura 21.4: Visualização da WordNet pelo site Visual Words.

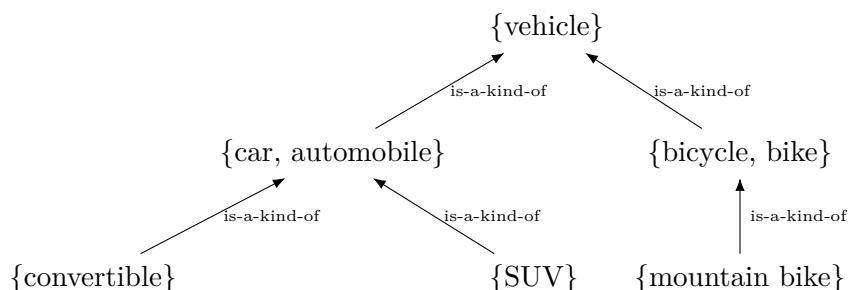


Figura 21.5: Exemplo de relação de hiperonímia presente no WordNet.

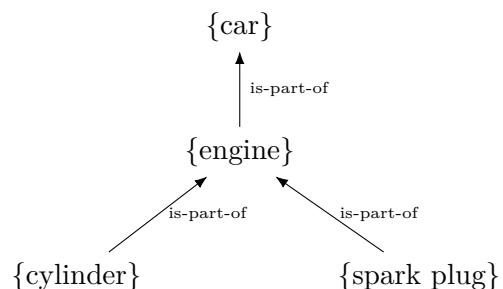


Figura 21.6: Exemplo da relação de meronímia no WordNet

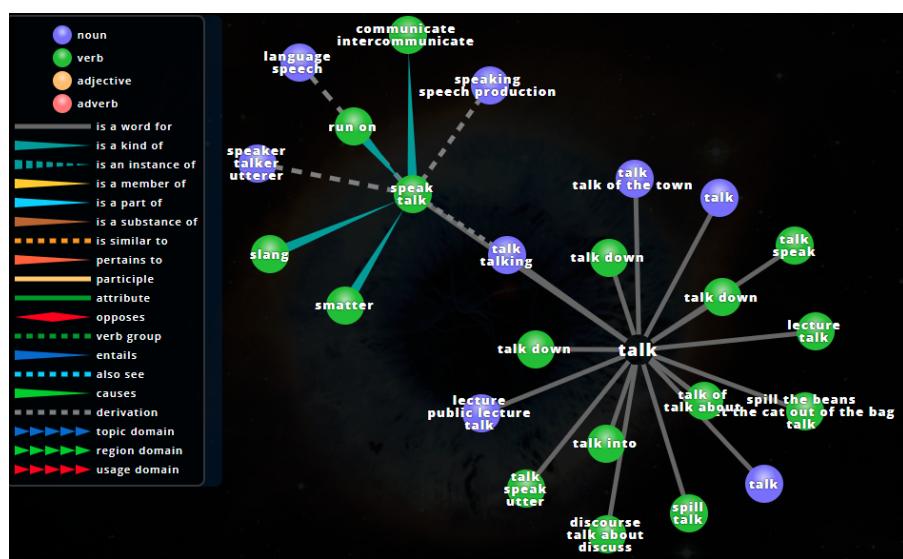


Figura 21.7: Visualização, no VisualWord da palavra talk, ligada a vários conceitos, sendo um o verbo talk no sentido de falar, que é "a kind of" Communicate.

21.2 Uso do WordNet

O WordNet pode ser usado para determinar a distância semântica entre dois termos. Para isso, uma forma é buscar um hiperônimo comum a ambos.

Outro uso do WordNet é tratar a tarefa de *Word Sense Disambiguation*. Isto pode ser feito procurando termos próximos ao termo onde a dúvida, em busca de termos com uma distância semântica menor do termo sendo investigado e que possa usar para escolher o significado. Por exemplo, a palavra “*bank*” pode ter mais de um significado, porém se aparecer junto de “*clerk*” ou “*financial*”, é mais provável que seja um banco financeiro do que outras interpretações.

21.3 WordNet e Prolog

Uma maneira razoavelmente fácil de manipular o WordNet é por meio de sua versão em Prolog. Essa versão fornece vários arquivos, cada um com um predicado. O arquivo `wn_s.pl` fornece a informação principal, com o predicado `s/6`, que indica:

- número do synset, um campo de 9 bytes onde o primeiro byte define a categoria do synset, que pode ser
 - 1, substantivo
 - 2, verbo
 - 3, adjetivo
 - 4, advérbio
- sequência do termo no synset, ordenados da esquerda para direita, se esse número for usado em algum lugar como 0, significa que a relação é com todo o synset.
- o termo ou palavra,
- um caracter que indica o tipo do synset, podendo ser
 - n, para substantivo (*noun*)
 - v, para verbo,
 - a, para adjetivo,
 - s, para adjetivo satélite, e
 - r, para advérbio;
- um número indicando uma posição entre os vários significados do termo (não é possível saber quantos existem sem achar o máximo);
- um número que indica quantas vezes o sentido específico do termo foi identificado em vários textos de concordância semântica;

O número do synset, ou o par número do synset, sequência do termo, será usado nos outros arquivos. Por exemplo, o arquivo com os *glosses* usa apenas o número do synset, enquanto o arquivo com os rótulos para os sentidos usa os dois, já que é importante identificar a palavra ou termo usado.

Programa 21.1: Programa Simples em Prolog

```

1 explica(T,E)  :-  s(X,_,T,_,_,_),  g(X,E) .
2
3 eum(P,F)  :-  s(X,_,P,_,_,_), s(Y,_,F,_,_,_), hyp(Y,X) .
4 eum(P,F)  :-  s(X,_,P,_,_,_),  s(Y,_,Z,_,_,_), hyp(Y,X) , eum(Z,F) .
5
6 mesmopai(P,F,K)  :-  eum(P,K) , eum(F,K) .

```

arquivo	predicado	uso
wn_s.pl	s/6	fornecer todos os synsets e seus termos
wn_sk	sk/3	fornecer uma codificação apropriada dos termos quando usados em sistemas de rotulação
wn_hyp	hyp/2	par de synsets onde o segundo é hierônimo do primeiro
wn_mm	mm/2	par de synsets onde o segundo é parte do primeiro
wn_g	g/2	<i>glosses</i> para os synsets

DRAFT

CAPÍTULO 22

CONJUNTOS DE DADOS DISPONÍVEIS

22.1 Coleções Clássicas

Algumas coleções razoavelmente pequenas são comuns em artigos antigos.

22.2 Coleções TREC

22.3 Acesso as coleções

O **ir datasets**: **Catalog**(MacAvaney et al., 2021)¹ é “um site que fornece informações sobre conjuntos de dados disponíveis para pesquisa em recuperação de informação. `ir\datasets` é um pacote Python que fornece uma interface comum para vários benchmarks de classificação de recuperação de informação ad-hoc, conjuntos de dados de treinamento, entre outros. O pacote cuida do download dos conjuntos de dados (incluindo documentos, consultas, julgamentos de relevância, etc.) quando disponíveis em fontes públicas. Instruções sobre como obter conjuntos de dados são fornecidas quando eles não estão disponíveis publicamente. `ir\datasets` fornece um formato de iterador comum para permitir que eles sejam facilmente usados em Python. Ele tenta fornecer os dados de forma inalterada (ou seja, mantendo todos os campos e marcações), enquanto lida com diferenças em formatos de arquivo, codificação, etc. Adaptadores fornecem funcionalidade extra, como permitir buscas rápidas de documentos por ID. Uma interface de linha de comando também está disponível.(MacAvaney et al., 2021)”

¹<https://ir-datasets.com/>

DRAFT

Parte V

Aprendizado de Máquina

DRAFT

CAPÍTULO 23

INTRODUÇÃO AO APRENDIZADO DE MÁQUINA

Em 1959, Arthur Samuel escreveu um artigo considerado um marco inicial do aprendizado de máquina, dizendo “Um computador por ser programado de forma a aprender a jogar um jogo melhor de damas do que é jogado pela pessoa que escreveu o programa”, e também “Programar computadores para aprender pela experiência deve eventualmente eliminar a necessidade para muito desse esforço detalhado de programação”(Samuel, 1959)¹.

Já Tom Mitchell (1997) explica “Podemos dizer que um programa de computador aprende pela experiência E a respeito de alguma tarefa T e com uma medida de desempenho D se o desempenho em T, medido D, melhora com a experiência E”(T. M. Mitchell, 1997).

Existem várias escolas de Aprendizado de Máquina. Domingos (2015) descreve 5 principais:

1. os **Simbolistas** defendem que “toda inteligência pode ser reduzida a manipulação de símbolos” (Domingos, 2015);
2. os **Evolucionistas** partem do princípio que “a mãe de todo aprendizado é a seleção natural” (Domingos, 2015);
3. os **Analogistas** dizem que “a chave do aprendizado é o reconhecimento de semelhanças entre situações e, a partir daí, a inferência de outras semelhanças” (Domingos, 2015);
4. os **Bayesianos** defendem que “todo conhecimento aprendido é incerto e o próprio aprendizado é um tipo de inferência incerta” (Domingos, 2015), e
5. os **Conexionistas** buscam uma “engenharia reversa do cérebro” (Domingos, 2015).

Como toda classificação da forma como as pessoas fazem ciência, Domingos lembra que existem outras visões e abordagens híbridas.

Normalmente, o **aprendizado de máquina** é feito com um entre quatro objetivos, que aumentam de complexidade, mas também de valor para a organização:

1. **Descrição** de uma situação;
2. **Diagnóstico** de uma situação;
3. **Predição** de uma situação, e
4. **Prescrição** de uma linha de ação.

¹Muitos textos atribuem a Arthur Samuel, e citam este mesmo artigo, pela frase: “Field of study that gives computers the ability to learn without being explicitly programmed”, mas não há nenhuma menção a esta frase em seu artigo, ou nos subsequentes. Talvez o autor tenha falado, mas é uma prova que muita gente não lê o artigo que cita.

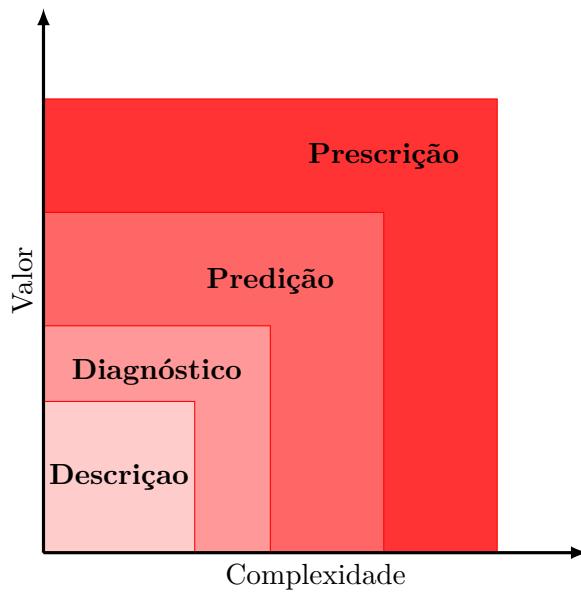


Figura 23.1: Quatro objetivos comuns de aprendizado de máquina.

Na **Descrição** a questão principal é encontrar, no conjunto de dados disponível, informações que permitam entender o comportamento do passado. Essas informações podem, mais tarde, ser usadas para prever o futuro, supondo que as condições continuem as mesmas. De certa forma, todo o processo de aprendizado de máquina inclui a capacidade de descrever, por diversos tipos de modelos, compreensíveis por humanos ou não, os dados disponíveis. Uma tarefa típica de descrição é gerar uma nuvem de palavras, ou *tag cloud*, que descreva um texto, ou um grupo de textos (Xexeo, F. Morgado e Fiúza, 2009).

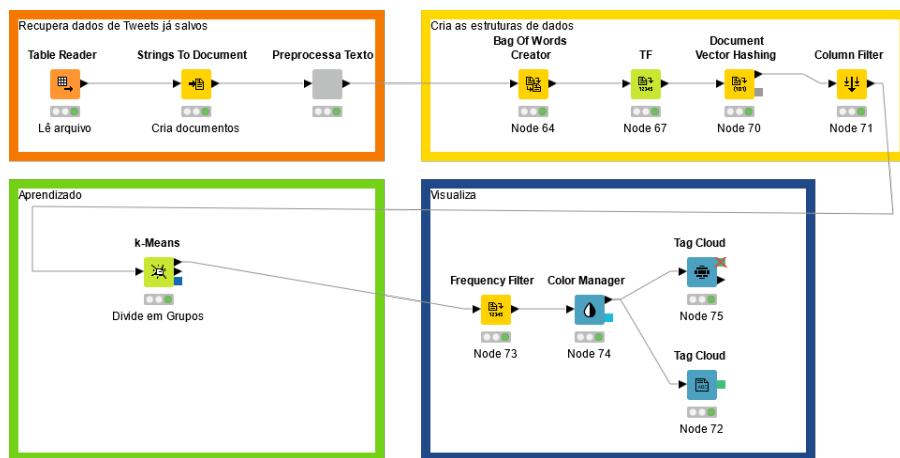


Figura 23.2: Workflow KNIME para criação de um tag cloud.

Tarefas de **diagnóstico** já procuram, além da descrição, detectar algum comportamento ou situação específica a detecção de *spam* e *fake news*, ou a análise de sentimento.

A tarefa de **predição** busca prever uma situação ou um evento futuro. Um exemplo com texto é tentar prever o comportamento da bolsa por meio das notícias, ou tentar prever a duração de um relacionamento a partir de conversas entre casais Pennebaker, 2011.

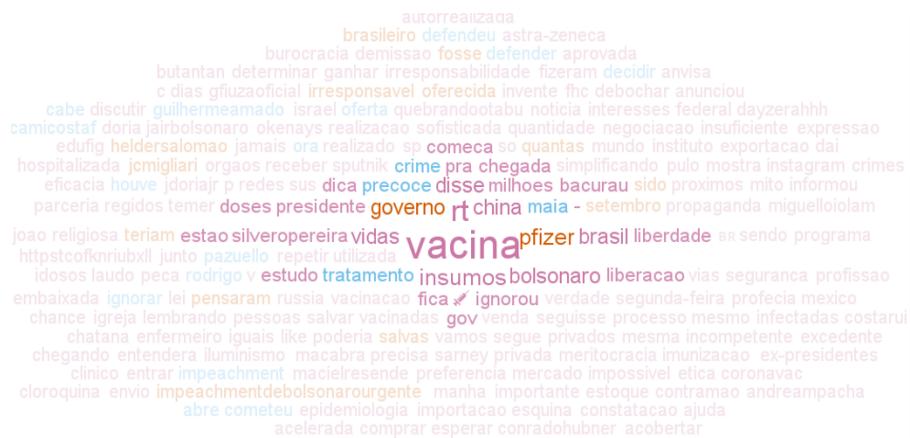


Figura 23.3: Tag Cloud criado com o modelo da Figura 23.2.

Finalmente, a tarefa de **diagnóstico** tenta sugerir uma linha de ação para o futuro, como por exemplo indicar uma pessoa para resolver um problema descrito em um email.

23.1 Visão Geral do Aprendizado de Máquina

O aprendizado de máquina pode ser visto, como o processo de estimar(Cherkassky e Mulier, 2007):

- a estrutura de um sistema, ou seja, criar um modelo de caixa-branca de um sistema em análise, ou
- a relação entre a entrada e a saída de um sistema, ou seja, criar um modelo de caixa preta de um sistema em análise.

Esse processo genérico de aprendizado envolve três componentes: “um Gerador de vetores de entrada aleatórios, um Sistema que retorna uma saída para esse valor, e uma Máquina de Aprendizado que estima um mapeamento desconhecido entre entradas e saídas”(Cherkassky e Mulier, 2007). É possível adicionar a essa descrição a necessidade que a Máquina de Aprendizado também forneça um Modelo, aproximado, do sistema, como na Figura 23.4.

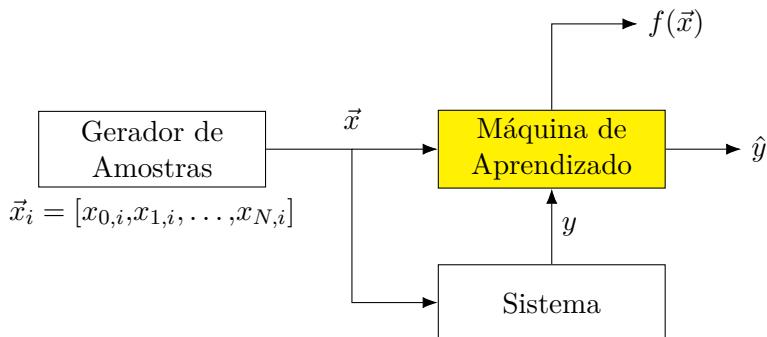


Figura 23.4: Modelo genérico do aprendizado de máquina, adaptado de (Cherkassky e Mulier, 2007)

Formalmente o gerador fornece vetores aleatórios que seguem alguma distribuição probabilística, $\vec{x} \in \mathbb{R}^N$, porém, a não ser em condições de simulação em experimentos com dados sintéticos, o que está disponível ao pesquisador são os dados reais, em uma situação **observacional**, sobre os quais ele não tem controle(Cherkassky e Mulier, 2007). Além disso, os dados fornecidos também já incluem as saídas y do Sistema, que resultam do processamento de \vec{x} .

Normalmente, então, o Gerador de Amostras e o Sistema estão escondidos, juntos, em uma caixa preta a que o pesquisador não tem acesso, ou tem acesso limitado.

Nessa situação, é comum que os dados possuam um **viés**, ou vários vieses. Um viés é um direcionamento nos dados, que tanto pode vir da forma como foram obtidos, da amostragem, ou mesmo da prática atual que sistema possui, mas não é desejada, como o aprendizado de preconceitos em sistemas de tomada de decisão(Emspak, 2016).

Também é possível, e também comum, que a amostra não possua todos os atributos que influenciam o sistema. Por exemplo, ao tentar predizer a venda de sorvete na praia, podem estar disponíveis a temperatura e o dia da semana, mas não o fato de ser feriado, ou se vai chover ou não. Esses dados, ou variáveis não observadas, podem levar a conclusões erradas.

Finalmente, em relação a amostra e as variáveis disponíveis, é importante frisar que “correlação não é causalidade”. Além de fatores como dois elementos correlacionados poderem possuir a mesma causa, um terceiro fator, também há a questão da direção da possível causalidade e ainda correlações espúrias. Um exemplo simples: apesar de haver correlação entre as vendas de sorvetes na praia e o número de queimaduras de Sol, não há causalidade entre um ou outro, sendo ambos causados pelo fatores que levam as pessoas a praia, como clima, presença de nuvens no céu, temperatura, dia da

semana, etc. Vigen (2015) publicou um livro, *Spurious Correlation*, onde apresenta várias correlações espúrias e engraçadas, como uma que existe entre a renda tuta gerada por *arcades* e o número de títulos de doutorado em Ciência da Computação nos EUA por ano².

Ao definir a Máquina de Aprendizado, são feitas duas escolhas: a primeira é a o algoritmo de aprendizado, a segunda é uma hipótese, composta pelo conjunto de funções que são candidatas para a definição do modelo(Abu-Mostafa, Magon-Ismail e H. T. Lin, 2012).

Por exemplo, ao tentar encontrar uma curva que passe por certos pontos, é possível escolher um algoritmo de regressão paramétrica que encontre um polinômio de grau fixo, como a regressão linear, que busca encontrar a e b em uma função $f(x) = ax + b$ que melhor explique os pontos, ou escolher um algoritmo que encontre um polinômio de grau não definido, na forma $y = \sum_{i=0}^m w_i x^i$, onde m passa a ser mais um parâmetro a ser determinado.

Então o papel da Máquina de Aprendizado é selecionar uma função aplicável as entradas do sistema, entre o conjunto de funções que suporta, que tenha o erro mínimo quando comparada com a saída do sistema, com acesso a apenas um conjunto finito de exemplos (Cherkassky e Mulier, 2007). O que está sendo feito, então, é calcular uma estimativa $p(\hat{y}|\vec{x}) = p(y|\vec{x})p(\vec{x})$, dado um conjunto limitado de exemplos de x e y , possivelmente com viés e erros. A essa função são aplicados os parâmetros w que minimizam o erro. Esse erro define uma qualidade, que é medida como a discrepância entre o resultado fornecido pelo sistema e o resultado fornecido pelo modelo, ou função, conhecida como **Perda** ou **Loss** (Cherkassky e Mulier, 2007):

$$L(y, f(\vec{x}, w)) \quad (23.1)$$

23.2 Modelos gerativos vs. discriminativos

Em geral, algoritmos discriminativos, como SVM, fazem predições baseados em probabilidades condicionais, enquanto algoritmos gerativos³, como Classificadores Naïve Bayes, constroem modelos de como uma classe gera os dados de entrada (Dan Jurafsky e J. H. Martin, 2022). Um algoritmo discriminativo supõe uma função para calcular $P(Y|X)$ e buscam estimar seus parâmetros a partir dos dados de treinamento. Já um algoritmo gerativo buscam estimar $P(Y|X)$ a partir de entender $P(X|Y)$ e $P(Y)$, para calcular $P(Y|X)$ usando o teorema de Bayes (A. Ng e M. Jordan, 2001). Ainda outra explicação: os modelos discriminativos criam fronteiras no espaço de dados, enquanto os modelos gerativos focam na explicação de como o dado foi gerado.

Exemplos de modelos discriminativos são: Regressão Logística, SVM, Redes Neurais (tradicional), K-NN, CRF, Árvores de Decisão e *Random Forest*. Já exemplos de modelos gerativos são: *Naïve Bayes*, *Bayesian Networks*, *Markov Random Fields* e *Latent Dirichlet Allocation* (LDA).

A. Ng e M. Jordan (2001) chamam a atenção para uma observação de Vapnik (2000), que diz “quando resolvendo um problema dado, tente evitar resolver um problema mais geral como um passo intermediário” (Vapnik, 2000, p. 30), o que levaria a escolher diretamente os modelos discriminativos. Porém, ao comparar os dois tipos de algoritmos, eles concluem que existem regimes distintos de desempenho com o aumento do número de casos de treinamento, e os algoritmos gerativos podem ser, em alguns casos, melhores que os discriminativos (A. Ng e M. Jordan, 2001) por convergirem mais rapidamente.

Mais ainda, A. Ng e M. Jordan (2001) definem um par gerativo-discriminativo a partir da escolha da distribuição de $p(x|y)$ e $p(y)$. Assim, ao escolher uma algoritmo de um tipo e um do outro,

²Alguns dos gráficos podem ser vistos em <http://www.tylervigen.com/spurious-correlations>

³Em inglês, *generative*, o que leva algumas traduções para português os chamarem de gerativos

para poder fazer uma análise do resultado, é interessante escolher um par desse tipo. Por exemplo, com entradas discretas, então os classificadores *Naïve Bayes* e Regressão Logística fazem um par gerativo-discriminativo (A. Ng e M. Jordan, 2001).

23.3 Tarefas Básicas do Aprendizado de Máquina

As principais tarefas do aprendizado de máquina são:

- **Classificação**, analisar uma amostra e classificá-la dentro um conjunto, normalmente pequeno, sempre limitado, de classes. Conhecida também como rotulação. A forma mais simples é responder sim ou não para uma pergunta, como o resultado de um exame de gravidez. No contexto de tratamento de texto é comum haver vários rótulos possíveis. Normalmente exige um conjunto de amostras pré-rotuladas para o aprendizado, onde é usado o aprendizado supervisionado;
- **Agrupamento**, analisar um conjunto de amostras e separá-las em grupos por similaridade, normalmente sem conhecimento anterior desses grupos, exigindo uma medida de distância entre as amostras, normalmente usando o aprendizado não-supervisionado, e
- **Regressão**, que busca encontrar os melhores parâmetros para uma função que passa por um conjunto de pontos em um espaço multi-dimensional, normalmente usando o aprendizado supervisionado.

23.4 Formas de Aprendizado de Máquina

As principais formas do aprendizado de máquina são:

- **Aprendizado Supervisionado**, onde todas as amostras usadas no aprendizado possuem a entrada e a saída;
- **Aprendizado Não-Supervisionado**, onde as amostras usadas no aprendizado possuem apenas as entradas;
- **Aprendizado por Reforço**, onde as amostras são fornecidas uma a uma para a máquina de aprendizado, que gera uma resposta. Essa resposta é avaliada e resulta em um *feedback* que altera o modelo da máquina de aprendizado;
- **Aprendizado Semi-Supervisionado**, que é um modo híbrido.

23.5 Modelo do Scikit-Learn

Uma das ferramentas que será utilizada para o aprendizado de máquina neste livro é o Scikit-Learn, um pacote de código aberto que fornece algoritmos para aprendizado de máquina em Python(Pedregosa et al., 2011).

O modelo de aprendizado do Scikit-Learn é simples, cada máquina de aprendizado é uma classe. Para usá-la é necessário criar uma instância e fornecer um vetor de entradas e um vetor de saída. Um processo de aprendizado básico com o Scikit-Learn segue os seguintes passos (Figura 23.5):

1. Ler os dados, \vec{x} e y , e colocá-los em uma array ou vetor;
2. Dividir os dados em aprendizado, \vec{x}_a e y_a , e teste \vec{x}_t e y_t ;
3. Criar a máquina de aprendizado, de acordo com o modelo desejado, usando um construtor de classe;
4. Fazer o aprendizado propriamente dito, a partir dos dados de aprendizado, \vec{x}_a e y_a ;

5. Fazer a predição \hat{y} a partir do modelo gerado e dos dados de teste \vec{x}_t , e
6. Comparar y_t com \hat{y} e obter as medidas de qualidade do aprendizado.

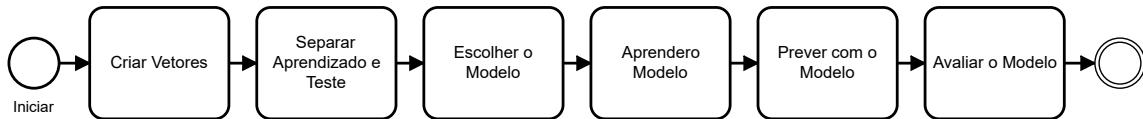


Figura 23.5: Processo básico do aprendizado de máquina usando o Scikit-Learn, modelo em BPMN

O Programa 23.1 implementa uma tarefa classificação com aprendizado supervisionado, usando o Scikit-Learn. O modelo escolhido é uma árvore de decisão, cujos parâmetros devem ser determinados pelo algoritmo. O objetivo é o problema de determinar a codificação de arquivos em português, entre 4 possíveis. Os arquivos estão todos em português⁴, com acentos, e foram extraídos do Corpus CETEN Folha 1.0, sendo que cada extrato aparece uma vez em cada uma das codificações possíveis.

Programa 23.1: Programa para árvore de decisão que busca uma maneira de determinar a codificação de um arquivo em português

```

1 # Bibliotecas usadas
2 from glob import glob           # trata diretórios
3 import numpy as np              # vetores e matrizes
4 from sklearn.model_selection import train_test_split # divisão dos dados
5 from sklearn import tree        # árvores de decisão
6 from sklearn import metrics     # avaliação
7 import matplotlib.pyplot as plt # gráficos
8
9 # set folders
10 FOLDERTEMP = "F:/Datasets Text/CetenFolha/Minis/" # diretório dos arquivos
11 XSIZE = (1,256) # tamanho da matriz inicial e das que são somadas a ela
12 CODEPAGES = ["cp1252","utf-8","utf-32","utf-16"] # codepages que vamos usar
13
14 def read_chunk(file_object, chunk_size=1):
15     """ Generator that provides
16     chuncks from a binary file, usually a binary one. No optimization
17
18     Attributes
19     -----
20     file_object : File
21         a file object created with open
22     chunk_size : int
23         size of the chunk to be read, default 1
24     """
25     while True:
26         file = file_object.read(chunk_size)
27         if not file:
28             break
29         yield file
30

```

⁴Os arquivos foram gerados com o Programa C.1

```

31 allfiles = glob(FOLDERTEMP+"*") # para procurar todos os arquivos no folder
32 x = np.zeros(XSIZE,float) # # inicializa a matriz inicial
33 y = [0]
34
35 # lê os arquivos e preenche
36 i = 0
37 count = 0
38 for file in allfiles: # em todos os arquivos do folder
39     count += 1
40     for j in range(len(CODEPAGES)): # guarda o encoding declarado no nome do arq
41         if file.find(CODEPAGES[j]) > -1:
42             y[i] = j
43     with open(file,"rb") as f: # conta quantos bytes de cada valor no arquivo
44         for byte in read_chunk(f,1):
45             number = int.from_bytes(byte,"big")
46             x[i,number] += 1
47     if file != allfiles[-1]:
48         x = np.r_[x,np.zeros(XSIZE,float)]
49         y.append(0)
50         i += 1
51
52 # cria as matrizes/vetores de treinamento e teste
53 x_train, x_test, y_train, y_test = train_test_split(x,y,
54                                                     test_size=0.3,
55                                                     random_state=1)
56
57 # cria o classificador
58 clf = tree.DecisionTreeClassifier()
59 # aprende
60 clf = clf.fit(x_train,y_train)
61 # obtem as predições
62 y_pred = clf.predict(x_test)
63 # mostra a imagem da árvore gerada
64 plt.figure(figsize=(15,15))
65 tree.plot_tree(clf,filled=True)
66 plt.show()
67 # analisa a qualidade
68 print("Accuracy: ",metrics.accuracy_score(y_test, y_pred))
69 print("Confusion Matrix:\n",metrics.confusion_matrix(y_test, y_pred))

```

Uma árvore de decisão pode ser representada graficamente ou em regras do tipo se-então. Para o algoritmo usado na árvore de decisão no Scikit-Learn, o que é feito é a escolha de um atributo para o corte do conjunto atual a ser classificado em dois, de modo a minimizar o índice Gini de cada conjunto restante. Aos conjuntos restantes se aplica o mesmo algoritmo. Como se pode ver nessa execução do Programa 23.1, no primeiro nível da árvore já são separados os 709 elementos usados no aprendizado e que pertencem a segunda classe (utf-8), pela observação que o byte 231, que representa o “ç” nessa codificação, não aparece (sendo sua frequência menor que 0,5). Cada classe recebe uma cor no diagrama

Como é uma tarefa de classificação, dos arquivos nas suas respectivas codificações, então usamos a medida de Acurácia para determinar a qualidade, sendo que seu valor pode ser definido como a proporção de previsões corretas sobre o total de previsões, e podemos analisar o comportamento entre as classes por meio de uma Matriz de Confusão, que apresenta nas suas linhas as classes corretas, nas colunas as classes previstas, e nas células a quantidade de elementos em cada categoria.

1 Acurácia: 0.9958402662229617 Resultado do Programa 23.1

2 Matriz de Confusão:

3 [[308 0 0 0]
4 [0 292 0 0]
5 [0 0 288 4]
6 [0 0 1 309]]

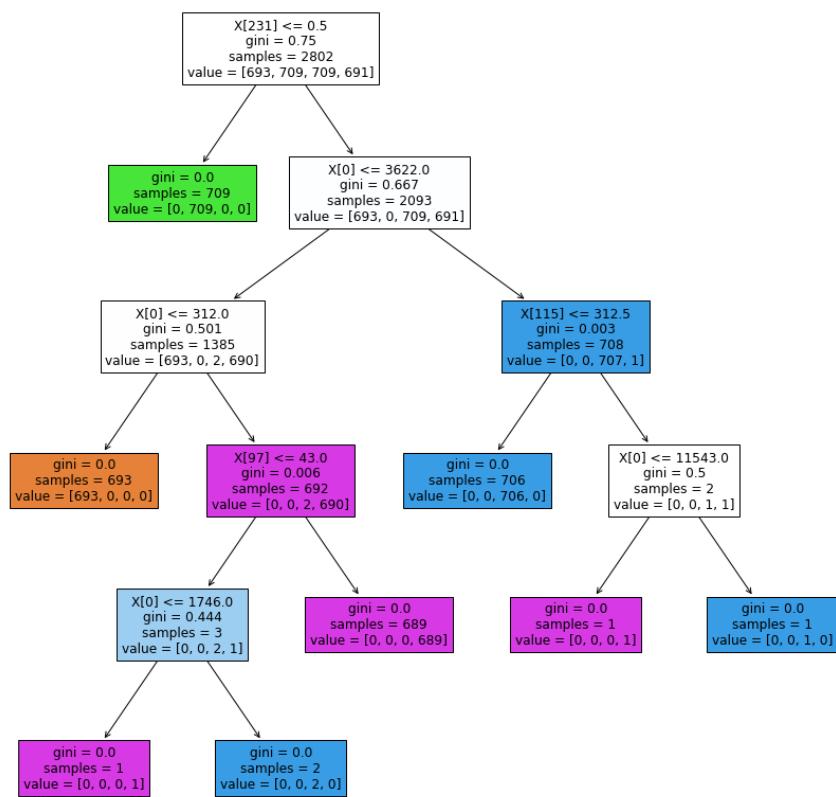


Figura 23.6: Árvore de decisão gerada pelo Programa 23.1.

CAPÍTULO 24

CLASSIFICAÇÃO DE TEXTOS

A tarefa de classificação de textos implica em receber como entrada um texto ou fragmento de texto e dar a ele uma classe entre um conjunto de classes pré-determinada. Por exemplo, dado um texto que contém um pedido de manutenção de software, decidir que setor da empresa, que são as classes, deve atender o pedido, tarefa conhecida como *ticket routing*.

Classificar um texto é sempre mais fácil quando a classificação é binária, isto é, quando se tem que decidir entre apenas duas classes, mas é muito comum que existam muitas classes no problema. Em geral, quanto menos classes, mais fácil é classificar, logo uma solução comum para o problema multi-classe é criar uma classificação hierárquica. Por exemplo, em uma aplicação que precisa identificar os times referenciados em mensagens em mídia social sobre times de futebol, pode ser primeiro feita uma separação entre mensagem que contém algo sobre times de futebol e depois uma segunda classificação escolhe o time da mensagem.

24.1 Análise de Sentimentos

Uma tarefa que tem chamado muita atenção na Mineração de Textos é a **Análise de Sentimentos**, ou **Mineração de Opinião**, que “o estudo computacional das opiniões, sentimentos, emoções, humores e atitudes das pessoas” (B. Liu, 2020).

As aplicações da Análise de Sentimentos são muitas. Hoje em dia, as empresas vigiam as redes sociais para obter a opinião dos clientes sobre ela, seus serviços e seus produtos. B. Liu (2020), por exemplo, afirma ter criado sistemas para análise de sentimentos para empresas em mais de 40 domínios. Aplicações em política também são comuns, tanto para especialistas quanto na academia.

A análise de sentimentos pode ocorrer em vários níveis, desde um documento completo até sentenças a sentença, podendo ser orientada também a aspectos sendo discutidos no texto. Além disso, opiniões podem ser absolutas, como dizer que “Feijoada é muito gostoso”, ou relativas, como dizer que “Feijoada é muito melhor que feijão simples” (B. Liu, 2020).

Essa tarefa é feita normalmente em duas etapas: a primeira decide se o texto contém ou não uma opinião, e a segunda dá a polaridade, ou seja, se é positiva ou negativa, da opinião. Em alguns casos, um terceiro passo pode avaliar a intensidade da polaridade.

DRAFT

CAPÍTULO 25

ALGORITMOS CLÁSSICOS DE CLASSIFICAÇÃO

25.1 Naïve-Bayes

25.2 SVM

DRAFT

Parte VI

Redes Neurais

DRAFT

CAPÍTULO 26

INTRODUÇÃO ÀS REDES NEURAIS

O interesse por redes neurais artificiais (RNAs) surge da tentativa de simular a habilidade do cérebro humano em resolver problemas complexos. Desde os primeiros avanços na inteligência artificial (IA), cientistas buscam criar sistemas capazes de aprender, reconhecer padrões e tomar decisões de forma autônoma (Russell e Norvig, 2009). A evolução das RNAs tem sido catalisada por avanços na computação e pela disponibilidade de grandes volumes de dados, que são essenciais para o treinamento desses modelos (Neisser et al., 1996).

26.1 Inspiração Biológica

As redes neurais são inspiradas no sistema nervoso biológico, onde os neurônios se conectam através de sinapses, formando redes altamente complexas (Gottfredson, 1994). Um neurônio artificial é um modelo computacional que busca replicar esse comportamento, permitindo que sistemas computacionais aprendam a partir de exemplos (M. Taylor, 2017).

26.2 Os Primeiros Modelos: O Perceptron

O perceptron, introduzido por Rosenblatt (1958), foi o primeiro modelo de RNA amplamente reconhecido. Ele era capaz de resolver problemas de classificação linear utilizando uma simples regra de atualização de pesos. Apesar de suas limitações, como a incapacidade de resolver problemas não-lineares, o perceptron abriu caminho para desenvolvimentos posteriores (McCulloch e Pitts, 1943).

26.3 Redes Neurais Feedforward e Backpropagation

As redes neurais feedforward são as mais simples, onde os dados percorrem a rede em uma *direção direta*, da entrada até a saída, sem laços. Matematicamente, cada neurônio realiza a seguinte operação:

$$y = f \left(\sum_{i=1}^n w_i x_i + b \right), \quad (26.1)$$

onde x_i são as entradas, w_i são os pesos sinápticos, b é o termo de viés (bias) e f é a função de ativação.

O treinamento dessas redes frequentemente utiliza o algoritmo de retropropagação do erro (back-propagation). Esse algoritmo calcula o gradiente da função de custo em relação aos pesos, ajustando-os para minimizar o erro:

$$\Delta w = -\eta \frac{\partial E}{\partial w}, \quad (26.2)$$

onde η é a taxa de aprendizado e E é a função de custo, frequentemente definida como o erro quadrático médio:

$$E = \frac{1}{N} \sum_{j=1}^N (y_j - \hat{y}_j)^2. \quad (26.3)$$

26.3.1 Principais Funções de Ativação

- **Função Sigmoide:** $f(x) = \frac{1}{1+e^{-x}}$
- **ReLU (Rectified Linear Unit):** $f(x) = \max(0, x)$
- **Tanh:** $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

Essas funções permitem a modelagem de relações não-lineares entre os dados.

26.4 Deep Learning: Redes Profundas

Com o aumento do poder computacional e o acesso a grandes volumes de dados, as redes neurais profundas (DNNs) emergiram como uma evolução natural das MLPs. Modelos como AlexNet (Krizhevsky, Sutskever e Hinton, 2012) demonstraram que redes profundas são capazes de obter resultados superiores em tarefas como classificação de imagens, reconhecimento de fala e processamento de linguagem natural.

26.5 Redes Recorrentes e LSTM

As redes neurais recorrentes (RNNs) foram introduzidas para lidar com dados sequenciais, como séries temporais e texto. No entanto, as RNNs tradicionais sofriam com o problema do desvanecimento do gradiente. Para resolver essa limitação, Hochreiter e Schmidhuber (1997) propuseram as redes LSTM (Long Short-Term Memory), que utilizam mecanismos de portas para controlar o fluxo de informação, permitindo o aprendizado de dependências de longo prazo.

26.6 Bibliotecas Python para Redes Neurais

Uma das ferramentas mais populares para implementação de redes neurais é o **PyTorch**, uma biblioteca de código aberto que facilita a criação de modelos de aprendizado profundo. A seguir, apresentamos um exemplo básico de uso:

```
import torch
import torch.nn as nn
import torch.optim as optim

# Definição do modelo
```

```

class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fc = nn.Linear(2, 1) # Camada totalmente conectada

    def forward(self, x):
        return torch.sigmoid(self.fc(x))

# Inicialização do modelo
model = SimpleNN()
criterion = nn.BCELoss() # Função de custo (Binary Cross Entropy)
optimizer = optim.SGD(model.parameters(), lr=0.01)

# Dados de exemplo
inputs = torch.tensor([[0.0, 0.0], [0.0, 1.0], [1.0, 0.0], [1.0, 1.0]])
labels = torch.tensor([[0.0], [1.0], [1.0], [0.0]])

# Treinamento
for epoch in range(1000):
    optimizer.zero_grad()
    outputs = model(inputs)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()

print("Treinamento concluído")

```

Esse exemplo demonstra como criar um modelo simples, definir uma função de custo e um otimizador, e treinar a rede utilizando o PyTorch.

26.7 Uma Ponte para os Transformers

As limitações das LSTMs em termos de paralelismo e capacidade de modelar dependências complexas levaram ao desenvolvimento de arquiteturas baseadas em transformers (Vaswani et al., 2017). Esses modelos representam um avanço significativo no processamento de linguagem natural, possibilitando aplicações como o BERT (J. Devlin et al., 2018a).

DRAFT

CAPÍTULO 27

EMBEDDINGS

A Seção 15.1 apresentou uma definição geral do conceito de *embeddings*. Neste capítulo tratamos de alguns *embeddings* publicados recentemente, dividindo-os em *embeddings* contextuais e não contextuais, isso é, que são relativos a um termo de forma isolada ou a um termo dentro de um contexto. A publicação do artigo de Mikolov et al. (2013) trouxe novas formas de tratar problemas de aprendizado de máquina, propondo representações eficientes das palavras, mas trazendo também desafios para definir como representar sentenças, parágrafos, trechos de documentos ou documentos inteiros.

27.1 *Embeddings* não contextuais

Mikolov et al. (2013) propôs uma representação extremamente densa para a representação de palavras em corpus muito grandes, o Word2Vec, que atinge boa qualidade em tarefas onde é importante uma medida de similaridade entre palavras. Essa representação usa vetores de tamanhos fixo, gerados a partir de uma rede neural. Os vetores usados são de tamanho 50, 100, 300 e 600. Foi inspirada em modelos de linguagens implementados em redes neurais que tentam prever a próxima palavra, o NNLM (Bengio et al., 2003) e o RNNLM.

A geração é feita por dois modelos alternativos: o *Bag-of-Words* e o ***Skip-gram***. Nesses modelos uma rede neural é treinada com um corpo muito grande e uma camada intermediária da rede é considerada a representação da palavra. No artigo original, é utilizada uma janela móvel de texto com 5 palavras, sendo que a palavra do meio é a que está sendo representada pelo vetor gerado.

O modelo *Bag-of-Words* é baseado em uma rede neural que recebe um conjunto de palavras encontrados ao redor de uma palavra específica, em uma janela de texto de tamanho fixo, e que deve aprender a gerar a palavra central da janela.

No modelo ***Skip-gram*** a entrada da rede é a palavra e a saída da rede são as palavras ao redor dela na janela.

O Word2vec se tornou popular por ser rápido de treinar e disponibilizar o código na rede. A tarefa de aprendizado não é importante no modelo, mas sim os pesos aprendidos. No Skip-gram o treinamento é feito com exemplos positivos, tirados diretamente dos textos por meio de uma janela deslizante, e exemplos negativos gerados aleatoriamente.

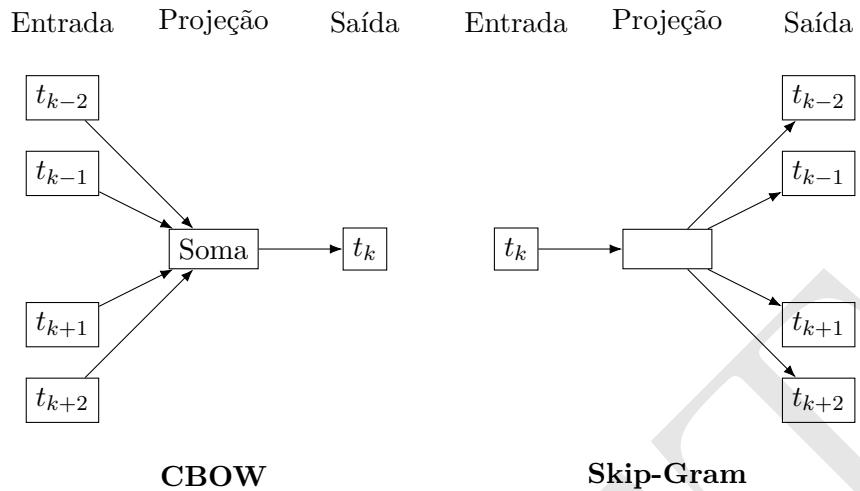


Figura 27.1: Representação dos modos CBOW e Skip-Gram. Fonte: (Bengio et al., 2003)

27.1.1 Detalhamento do Word2Vec

27.2 Seq2Seq

27.3 *Encoder-Decoder*

DRAFT

DRAFT

CAPÍTULO 28

AVANÇOS COM REDES NEURAIS

28.1 LSTM

DRAFT

CAPÍTULO 29

ATENÇÃO, TRANSFORMADORES, BERT E MODELOS GRANDES DE LINGUAGEM

29.1 Atenção

Atenção é uma técnica utilizada em redes neurais que simulam a atenção, de forma a aumentar a importância de alguns nós de uma camada da rede, como a entrada, em relação aos outros nós da camada. A ideia básica é fazer a rede focalizar em partes específicas dos dados, e para isso é necessário realizar um processo de aprendizado de máquinas. A atenção é o mecanismo básico de redes LSTM e dos Transformadores, utilizados no BERT e no GPT.

29.2 Transformadores

29.3 BERT

29.4 LLMs

DRAFT

Parte VII

Aplicações

DRAFT

CAPÍTULO 30

RAG

DRAFT

CAPÍTULO 31

NUVENS DE RÓTULOS

Este capítulo é uma adaptação do artigo “Differential Tag Clouds: Highlighting Particular Features in Documents” (Xexeo, F. Morgado e Fiúza, 2009), que foi desenvolvido de forma mais extensa por F. F. Morgado (2010).



Figura 31.1: Nuvem de palavras geradas a partir de uma versão do capítulo 2 deste livro, usando o site <https://www.wordclouds.com/>, em 9 de abril de 2022. Algumas stop words foram removidas manualmente.

31.1 Introdução

Nuvens de rótulos podem ser explicadas como diagramas formados por símbolos, normalmente palavras, que têm como intenção indicar o assunto de um documento ou coleção de documentos.

Usualmente a importância de cada rótulo no documento é indicada por características da fonte adotada por cada palavra no diagrama, principalmente tamanho e cor. A Figura 31.1 exemplifica o conceito.

É difícil precisar o primeiro uso de uma nuvem de rótulos. A Wikipedia cita o site Flickr, e seu co-fundador Stewart Butterfield, como a principal influência em sua divulgação, e sites como Del.icio.us e Technorati como seus popularizadores. Butterfield teria se apoiado em um implementação de Jim Flanagan, hoje apenas disponível na Way Back Machine¹ (Wikipedia, 2022).

Algumas nuvens apareceram antes na literatura, como no livro “Microserfs” de Douglas Coupland, de 1995, e a capa alemã do livro de Giles Deleuze e Félix Guattari “Tausend Plateaus. Kapitalismus und Schizophrenie”, de 1992. A mais antiga referência impressa, porém, é uma página da revista Littérature, de 1923, com o título ERUTARETTIL, que lista a importância de alguns autores surrealistas, criada por André Breton e Robert Desnos (Ombre Blanches, 2012) e apresentada na Figura 31.2. Não se tem notícia se houve influência dessas imagens na proposta de Flanagan.

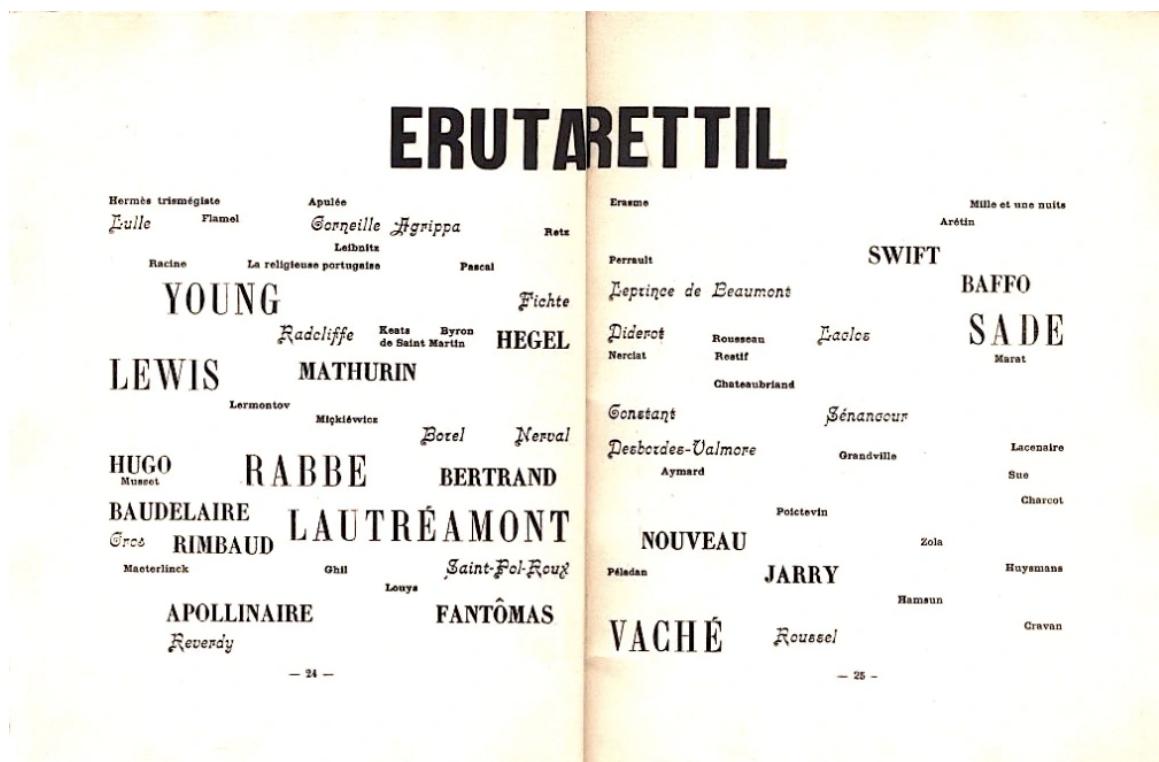


Figura 31.2: Primeira imagem impressa conhecida de uma nuvem de palavras. Fonte: Ombre Blanches (2012)

Inicialmente, nuvens de rótulos foram usadas por sites, como o Flickr, Del.icio.us e Technorati, para classificar postagens ou coleções de postagens. Esses rótulos poderiam ser explicitamente dados por seres humanos, o que mais tarde fica conhecido como *folksonomies*, ou obtidos por algoritmos. Mais tarde, dada a suposta facilidade que as nuvens de rótulo parecem dar a análise de significado de um corpus, isso é, um documento ou conjunto de documentos determinado, passaram a ser usadas para outras atividades, como o análise política (Dias e T. M. Paiva, 2022), ou a análise da relevância das acomodações no Airbnb (B. D. d. Paiva e Pinho, 2022).

¹<https://web.archive.org/web/20041204231120/http://twiki.tensegrity.net/bin/view/Main/SearchReferralZeitgeist>

É inegável o sucesso das nuvens de rótulo em algumas aplicações, principalmente quando é necessário ter uma compreensão abstrata de uma quantidade grande de texto. Porém existem críticas, e seu uso como ferramenta de navegação em sites praticamente se encerrou. Segundo (Viégas e Wattenberg, 2008), as nuvens de rótulo quebram quase todas as regras de ouro do design. Se por um lado isso parece surpreendente, também permite que sejam estudadas melhorias, ou quem sabe descobertas novas formas do ser humano perceber a informação, que afetariam o estudo do design. Já Nielsen (2009) diz “Nuvens de rótulo são usadas demais. Apesar de parecerem bonitas, elas usam espaço de tela de forma inefficiente e muitos usuários não sabem como usá-las”.

As principais ideias tratadas por esse artigo são como entender formalmente a geração de uma nuvem de rótulos de forma automática e para que elas podem ser usadas. Apesar de existirem muitos algoritmos para gerá-las, Xegeo, F. Morgado e Fiúza (2009) propuseram uma modelo conceitual que explica, de forma geral, como são geradas. Esse modelo é apresentado na Seção 31.3. A partir desse modelo esses autores também propuseram duas formas diferentes de criar nuvens de rótulo, as nuvens de rótulos diferenciais, que mostram como um documento se difere dentre um conjunto de documentos descrito por uma nuvem de rótulo de sumário. Isso foi explorado, mais tarde, por F. F. Morgado (2010) em sua dissertação de mestrado.

Para exemplificar esses conceitos, apresentamos imagens originais de F. F. Morgado (2010). A Figura 31.3 mostra uma nuvem de rótulos de sumário para todos os documentos retornados em uma busca sobre a palavra Jaguar. É possível notar as palavras “cars”, “animals” e “games”, que se referem a três tipos diferentes de jaguares: o carro, o animal e o console de videogame. Apesar de existirem formas específicas para gerar nuvens de rótulo de resumo, elas podem, na prática, serem vistas como nuvens de rótulos de vários documentos.

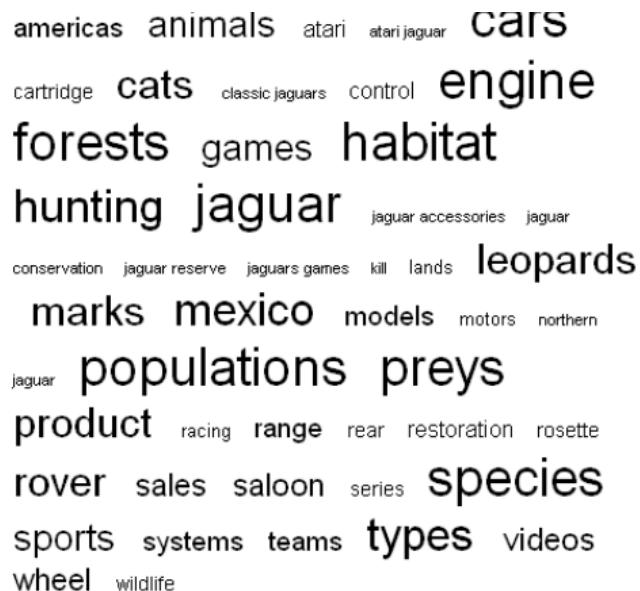


Figura 31.3: Nuvem de rótulos sumário para o conjunto de documentos gerados por uma busca sobre a palavra Jaguar. Fonte: F. F. Morgado (2010)

Já uma nuvem de rótulo diferencial exige não só um documento, mas uma coleção de referência, pois ela mostra como um documento (ou documentos) se diferem de uma coleção. A Figura 31.4 mostra como um documento específico que trata o animal jaguar se difere da coleção completa (e sua nuvem de rótulos).

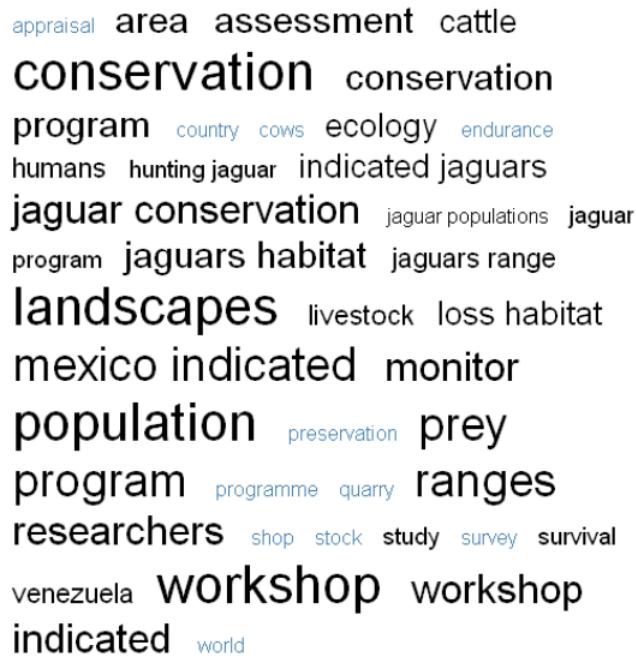


Figura 31.4: Nuvem de rótulos diferencial para um documento sobre o animal jaguar, em referência ao conjunto de documentos gerado por uma busca pela palavra jaguar. Fonte:F. F. Morgado (2010)

31.1.1 Um pequena questão de nomenclatura

O termo “nuvem de rótulos”, adotado neste texto, resume uma variedade de termos usados na literatura e no dia a dia. Os mais comuns, além do que adotamos, são “nuvens de palavras”, ou as versões em inglês, “word clouds” ou “tag clouds”. Alguns autores, buscando uma forma supostamente mais científica, usam o nome “weighted list”, ou “lista ponderada”, porém esse nome se confunde com um conceito pouco conhecido de estrutura de dados. Outro nome que vem sendo usado é “wordle”, que seriam uma versão das tag clouds que “dão atenção cuidadosa a tipografia, cor e composição” (Viégas, Wattenberg e Feinberg, 2009), mas que na prática resultam na mesma coisa.

Existe uma leve diferença semântica entre “palavra” e “rótulo”. Idealmente, quando é usado o termo “palavra” se supõe que as palavras que aparecem na nuvem foram tiradas diretamente do texto. Já no caso do termo “rótulo”, se supõe que as palavras que aparecem na nuvem foram obtidas de certa forma a partir do texto, mas não necessariamente pertencem ao texto, podendo apenas ser rótulos calculados de alguma forma ou fornecidos por seres humanos. Nielsen (2009) explicita essa diferença, dizendo que nuvens de palavras falam dos termos mais frequentes encontrados em um corpus. Em seu exemplo ele mostra como a comparação visual de duas nuvens de palavras pode levar a compreensão das diferenças e semelhanças entre dois documentos, mesmo que afirme que um resumo de um parágrafo seria mais útil.

Essa separação, porém, é indiferente perante a teoria desenvolvida neste artigo. Assim, para normalizar a nomenclatura, será usado sempre o termo “nuvem de rótulos”, já que rótulos podem ser palavras, mas nem todo rótulo é necessariamente uma palavra. Sempre que possível, os termos em inglês serão traduzidos.

31.2 Visão geral das nuvens de rótulos

Segundo Rivadeneira et al. (2007), “Nuvens de rótulos são representações visuais de um conjunto de palavras, tipicamente um conjunto de rótulos selecionados por algum método racional, na qual atributos do texto como tamanho, estilo, ou cor são usados para representar características dos termos associados”.

Apesar de seres humanos poderem criar nuvens de palavras, o método racional normalmente é um algoritmo, compostos de vários passos típicos do processamento de linguagem natural: obtenção do arquivo original, extração do texto, limpeza de caracteres indesejados e outras formas de pré-processamento, extração das palavras mais significativas ou de conceitos, e a organização das palavras na forma de um diagrama.

Uma característica adicional importante das nuvens de palavras é que eles se referem a alguma coisa, um objeto, normalmente um documento ou uma coleção de documentos. Essa relação pode ser feita de forma explícita, por exemplo por meio de títulos, legendas ou URLs, ou implícita, por exemplo por meio da localização da nuvem de palavras dentro do próprio documento.

Ou seja, **nuvens de rótulos são descrições de objetos**.

Rótulos são normalmente palavras, mas podem ser qualquer símbolo associado a um objeto, como uma imagem, um ícone, uma sequência de palavras (conhecidas como n-gramas), radicais, ou lemas.

O objetivo de cada rótulo é representar um conceito, que é ligado, de alguma forma, ao objeto sendo descrito pela nuvem. Marinchev (2006) propõe que para cada objeto existe um conjunto de conceitos abstratos associados a ele. Um **campo semântico** é “o conjunto de conceitos conectados a um objeto foco, de tal forma que é independente das pessoas que atribuíram os rótulos (rotuladores originais) e é possível que outras pessoas tenham o entendimento destes conceitos” (Marinchev, 2006).

Uma nuvem de rótulos pode então ser interpretada como uma representação visual de um campo semântico de um objeto. Assim, a criação de uma nuvem de rótulos pode ser entendida como um processo de três passos (Marinchev, 2006; Xexeo, F. Morgado e Fiúza, 2009):

1. Entendimento do objeto foco e dos conceitos a ele aplicáveis;
2. Compreensão do campo semântico associado ao objeto, e
3. Transcrição do campo semântico em uma nuvem de rótulos.

É importante notar que há uma dificuldade semântica entre cada passo, o que permite a introdução de ruídos. Pode ser que no primeiro passo o entendimento não seja completo ou correto, pode haver um problema na tradução desses conceitos em uma visão completa e finalmente pode haver também uma dificuldade de transformar os conceitos em rótulos. Por exemplo, o conceito de “saudade” não possui uma tradução em uma palavra apenas em outras línguas, enquanto os alemães possuem uma palavra, “schadenfreude”, para o conceito de sentir prazer com a desgraça alheia.

O processo de interpretação de uma nuvem de palavras também é sujeito a ruídos. Ele é composto de (Marinchev, 2006):

1. Leitura dos rótulos da nuvem, dentro de um contexto;
2. Interpretação dos rótulos em conceitos;
3. (Re)Criação do campo semântico, e
4. Criação de uma imagem mental do que é o objeto foco associada ao campo semântico.

Toda esse processo é mediado por propriedades visuais. Essas propriedades influenciam a percepção do usuário sobre o objeto associado. Bateman, Gutwin e Nacenta (2008) discutem as características visuais das nuvens de rótulos e como elas influenciam a atenção do usuário, informações que podem

ser usadas pelo criador da nuvem para direcionar a atenção do usuário e, consequentemente, sua interpretação do documento. Segundo Bateman, Gutwin e Nacenta (2008), as propriedades visuais das fontes mais importantes são: tamanho, peso e intensidade das cores. As cores parecem ser importantes como indicadores, apesar desses autores não terem conseguido descobrir quais cores atraem mais atenção. Além disso, a posição dos rótulos é importante na percepção, devendo os mais importantes estar no centro da nuvem.

Várias táticas de posicionamento também podem ser usadas para manipular a atenção, como a proximidade semântica dos rótulos (Hassan-Montero e Herrero-Solana, 2006), a distância do centro em nuvens circulares (Bielenberg e Zacher, 2006), a movimentação dos rótulos e o uso de 3D.

Tudo isso leva a compreensão de que a interpretação de uma nuvem de rótulos está sujeita a vários ruídos, tanto na sua criação, quanto na sua interpretação. Esses ruídos podem criar, alterar, ou eliminar símbolos na nuvem, o que por sua vez pode criar, alterar ou eliminar conceitos do campo semântico formado na mente do leitor. Assim, quanto melhor o processo de criação de nuvens de rótulos, possivelmente melhor será o resultado. Atualmente esse processo é normalmente automatizado, por meio de algoritmos de processamento de linguagem natural (NLP), aprendizado de máquina, ou outras estratégias e heurísticas.

Neste artigo, mostramos um modelo conceitual geral que atende a qualquer forma de criação das nuvens de rótulos, o que permite entender os passos e estabelecer formas de avaliar o resultado.

31.2.1 Características das Nuvens de Rótulos

Como visto pelo exemplo da Figura 31.1, nuvens de rótulos são construídas pela disposição de símbolos, normalmente palavras, em um espaço, normalmente bi-dimensional. Esses rótulos podem ser colocadas de várias maneiras, o que é feito usualmente por motivos estéticos. No exemplo da Figura 31.1 a imagem de uma nuvem é usada para limitar o espaço, enquanto as palavras, que representam os rótulos, são colocadas em várias posições e rotações.

A praxe é que o tamanho do rótulo indique a importância do mesmo como representante do significado do objeto para o qual foi criada a nuvem. Outros atributos, como cor, centralidade da posição, etc. podem ser usados tanto com significado como por motivos decorativos. Na Figura 31.1, cor, tipo e peso da fonte e rotação são decorativos. O algoritmo usado também favorece colocar próximo ao centro as palavras mais importantes.

Outras nuvens de rótulos seguem estratégias diferentes. Uma abordagem é listar as palavras sequencialmente, sem variar muitos elementos das fontes, de modo a oferecer menos distratores. A Figura Figura 31.5a mostra uma nuvem ordenada em ordem alfabética, enquanto a Figura Figura 31.5b mostra uma nuvem ordenada por importância do rótulo.

Outra forma de organizar é por semelhança conceitual entre os rótulos, como a nuvem da Figura 31.6. Para isso, no caso da criação automática, é necessário um processamento mais sofisticado, como o uso de LDA (F. F. Morgado, 2010).

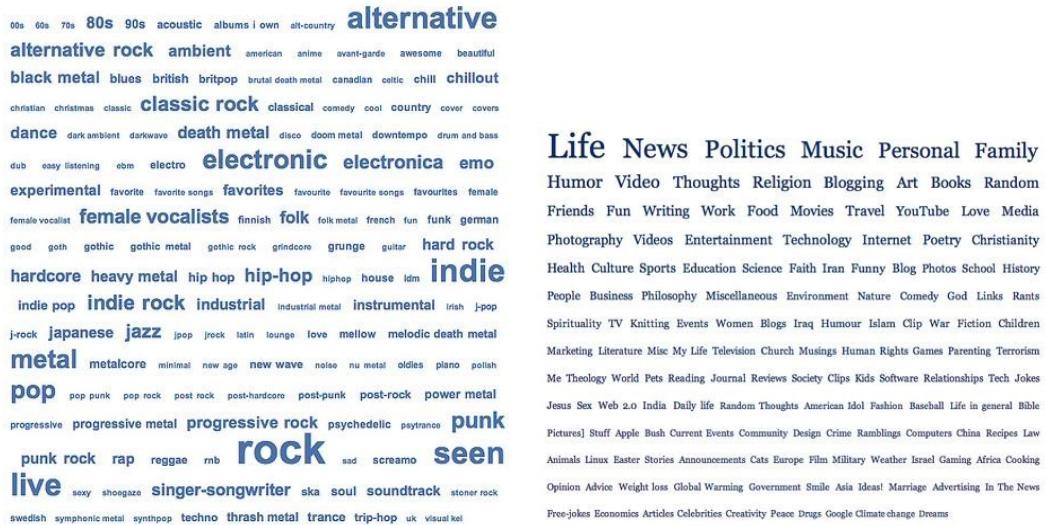


Figura 31.5: Duas maneiras de construir uma nuvem de rótulos.



Figura 31.6: Nuvem ordenada por importância do rótulo.

31.3 Um modelo formal para nuvens de rótulo

Um modelo formal para nuvens de rótulo foi proposto por Xexeo, F. Morgado e Fiúza (2009) e descrito de detalhadamente na dissertação de mestrado de F. F. Morgado (2010). Nesta seção ele é reapresentado buscando uma melhor ordem para a compreensão de conceitos.

O modelo começa com definições muito gerais, que servirão de base para a construção do resto do modelo, partindo do conceito de objetos. A esses objetos podem ser associados atributos. Cada atributo descreve uma propriedade do objeto, de acordo com um conjunto de valores. O conjunto de atributos define, implicitamente, a classe do objeto. Mapas permitem associar objetos a atributos, atributos a conjunto de valores e, um valor a cada atributo de um objeto. Esse modelo é descrito em UML na Figura 31.7. A Subseção 31.3.1 descreve esse modelo, por meio de conjuntos e funções, introduzindo restrições, podendo ser considerada um meta-modelo de referência. A seguir, na Subseção 31.3.2 são introduzidos os conceitos de recursos, o que queremos representar nas nuvens de rótulos, e suas coleções, contextos, e as funções necessárias para trabalhar com eles. O modelo, porém, se baseia na

ideia que essa representação existe por causa da existência de campos semânticos, e isso é tratado na Subseção 31.3.3. Estas definições são então finalmente usadas para gerar rótulos (Subseção 31.3.4), e criar nuvens de rótulos (Subseção 31.3.5).

Esse modelo tenta deixar claro que **nuvens de rótulos são representações físicas de campos semânticos, onde valores associados aos conceitos, como relevância, são traduzidos para valores associados aos rótulos, como peso da fonte ou posição no diagrama.**

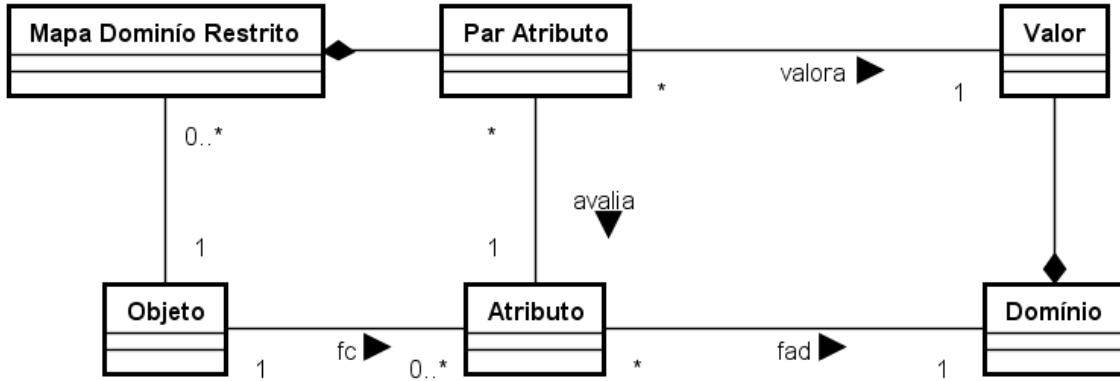


Figura 31.7: Modelo UML que descreve como são construídos os atributos de um objeto

31.3.1 Definições iniciais do modelo

Um **objeto**, representado por o , é um conceito primitivo, semelhante a definição de elemento na Teoria Ingênua de Conjuntos (K. Devlin, 1993). O conjunto de objetos é o conjunto universal U .

Como no modelo relacional (Date, 2004), um **domínio** é um conjunto de valores, notado como V_i . O conjunto de todos os domínios é indicado pela letra V . Valores de um domínio podem ser indicados pela letra v com índices, como em v_{ij} , que indica o j -ésimo valor do i -ésimo domínio.

Um **atributo**, representado por a , de um objeto o , é uma propriedade que descreve o . O conjunto de todos atributos possíveis é denotado por A . Um conjunto de atributos, isto é, um subconjunto de A , é representado por A_i . Um atributo de A_i é representado por a_i , e um atributo de um objeto o_j é representado por a_{ij} .

Para associar um domínio a um atributo, é definida uma **função de atribuição de domínio**, representada por f_{ad} . Atributos só podem possuir um domínio, mas vários atributos podem possuir o mesmo domínio.

$$f_{ad} : A \rightarrow V \quad (31.1)$$

Um **par atributo**, é um par ordenado (a_i, v_{ij}) que indica um atributo a seu valor.

$$(a_i, v_{ij}) \in A \times V_i, \text{ onde } f_{ad}(a_i) = V_i \quad (31.2)$$

Para que um objeto ser descrito, é preciso que ele possua atributos e que os atributos possuam valor. Para isso é necessário existir um **conjunto par atributo**, ou um **mapa de domínio restrito**, representado por m , que é um conjunto de pares atributos onde todo primeiro elemento de um par

ordenado é único entre todos os componentes do mapa.

$$\begin{aligned}
 m = & \{(a_i, v_{ij}) | \\
 & (((a_i, v_{ij}) \in A \times V_i) \wedge (f_{ad}(a_i) = V_i)) \wedge \\
 & ((\text{se } (a_i, v_{ij}) \in m) \wedge (a_i, v_{ik} \in m)) \\
 & \text{então } (a_i = a_k) \Rightarrow (v_{ij} = v_{ik}))\}
 \end{aligned} \tag{31.3}$$

O conjunto de todos os mapas possíveis é denotado por M .

Uma **função de classificação**, representado por f_c , gera um conjunto de atributos $A_i, A_i \subset A$, a partir de um objeto o . A ideia aqui é que essa função, reconhecendo um objeto, gera todos os atributos que o descrevem, mas não seus valores. Ela representa a abstração de classificação dos seres humanos². A notação $\wp(A)$ significa o conjunto potência de A , indicando que um objeto pode ter quaisquer quantidade de atributos

$$f_c : U \rightarrow \wp(A) \tag{31.4}$$

$$f_c(o) = \{a_{ij} | a_{ij} \text{ é uma propriedade de } o\} \tag{31.5}$$

Uma função de atribuição de mapa é uma função f_{am} que, dado um objeto, e um conjunto de atributos $A_i, A_i \subset A$, gera um mapa no qual para cada atributo de A_i há um par atributo correspondente. Esse mapa é, na prática, a atribuição de valores a cada atributo de cada objeto.

$$f_{am} : U \times \wp(A) \rightarrow M \tag{31.6}$$

$$\begin{aligned}
 f_{am}(o, A_i) = & \{(a_i, v_{ij}) | \forall a_{ij} \in A_i \Rightarrow \\
 & (f_{ad}(a_{ij}) = V_i) \wedge ((a_i, v_{ij}) \in f_{am}(o, A_i))\}
 \end{aligned} \tag{31.7}$$

Essas duas funções permitem classificar um objeto, isto é, determinar seus atributos, e dar valores aos mesmos, ambos de forma dinâmica.

31.3.2 Introduzindo recursos e contextos

Como nuvens de rótulos são criadas na Internet, o modelo formal restringe os objetos a recursos. A definição de **recurso** é tomada de do RFC 3986 (T. Berners-Lee, Fielding e Masinter, 2005): qualquer conceito abstrato ou entidade física que pode ser identificado unicamente. Nesse modelo formal, um **recurso**, representado por r , é também qualquer objeto que possa ser descrito, ao menos parcialmente, por rótulos (F. F. Morgado, 2010). Um **contexto**, representado por w , é uma coleção de recursos. O conjunto de todos os contextos tem a notação W , inspirado na *World Wide Web*.

Como, nas nuvens de rótulos, deseja-se representar recursos, o modelo basicamente repete as definições do meta-modelo, mas restringindo os objetos classificáveis ao conjunto de recursos. Assim, ele se inicia por definir uma *função de classificação de recursos*, f_{cr} , e depois uma função de atribuição de mapa de recurso, f_{amr} .

$$f_{cr} : W \rightarrow \wp(A) \tag{31.8}$$

$$f_{cr}(r) = \{a_{ij} | a_{ij} \text{ é uma propriedade de } r\} \tag{31.9}$$

²É possível observar que apesar do modelo falar de atributos, nada no modelo evita que o atributo seja uma função, já que funções também podem ser valores, e na prática ser equivalente a um método na orientação a objeto

$$f_{amr} : W \times \wp(A) \rightarrow M \quad (31.10)$$

$$\begin{aligned} f_{amr}(r, A_i) = & \{(a_i, v_{ij}) | \forall a_{ij} \in A_i \Rightarrow \\ & ((a_i, v_{ij}) \in f_{amr}(o, A_i) \wedge f_{ad}(a_{ij} = V_i))\} \end{aligned} \quad (31.11)$$

Tendo essas definições, é possível agora buscar **representações de recursos**. A ideia básica é que um recurso possui atributos valorados, e pode ser representado de alguma forma, a partir desses atributos e valores. Assim **representação de recursos**, ou mapa de recurso, representado como MR , é uma função de atribuição de mapa que tem como parâmetros um recurso e uma função de classificação de recursos, isto é, dado um recurso e seus atributos, retorna um mapa com os valores do recurso. Esse mapa é uma representação abstrata do recurso.

$$MR(r) = f_{amr}(r, f_{cr}(r)) \quad (31.12)$$

31.3.3 Criando campos semânticos

Um campo semântico, representado por CS , seguindo a definição de Marinchev (2006), é um subconjunto do conjunto de todos os conceitos abstratos, representado pelo conjunto C . Supondo um predicado lógico $D(c, r)$ que indica se um conceito descreve de alguma forma um recurso, um campo semântico para um recurso r pode ser definido como:

$$CS(r) = \{c_i | c_i \in C \wedge D(c_i, r)\} \quad (31.13)$$

Cada conceito, para um recurso específico, permite que seja descrito por um conjunto de atributos. Agora é possível ver como o meta-modelo pode ser usado para descrever tanto recursos como conceitos, pois ambos são objetos. Um **função de classificação de conceito**, representada por f_{cc} , é definida por:

$$f_{cc} : W \times C \rightarrow \wp(A) \quad (31.14)$$

$$f_{cc}(r, c) = \{\text{atributos de } c \text{ quando descreve } r\} \quad (31.15)$$

E, para uma função de classificação de conceito, existe uma **função de atribuição de mapa para um conceito**:

$$f_{amc} : W \times C \times \wp(A) \rightarrow M \quad (31.16)$$

$$\begin{aligned} f_{amc}(r, c, A_i) = & \{(a_{ij}, v_{ij}) | \forall a_{ij} \in A_i \Rightarrow \\ & ((a_{ij}, v_{ij}) \in f_{amc}(r, c, A_i) \wedge \\ & f_{ad}(a_{ij} = V_i))\} \end{aligned} \quad (31.17)$$

Com isso é possível criar um **campo semântico avaliado** para um recurso r , que é um conjunto de pares ordenados no qual o primeiro elemento é um conceito c_i aplicável ao recurso r e o segundo elemento é um conjunto par atributo composto dos atributos induzidos por c_i em r .

$$\begin{aligned} CSA(r) = & \{(c_i, m_i) | c_i \in CS(r) \wedge \\ & m_i = f_{amc}(r, c_i, f_{cc}(r, c_i))\} \end{aligned} \quad (31.18)$$

O campos semânticos e os campos semânticos avaliados, que devem ser gerados, respectivamente, por uma **função geradora de campos semânticos**, f_{gcs} e por uma **função geradora de campos semânticos avaliados**, f_{gcsa} .

$$f_{gcs} : W \times \wp(W) \rightarrow C \quad (31.19)$$

$$f_{gcs}(r, w) = \{c_i \mid c_i \in C \wedge \text{representa } w(c_i, r)\} \quad (31.20)$$

$$f_{gcs}(r, w) = CS_w(r) \quad (31.21)$$

$$f_{gcsa} : W \times \wp(W) \rightarrow C \times M \quad (31.22)$$

$$\begin{aligned} f_{gcsa}(r, w) = & \{(c_i, m_i) \mid c_i \in CS_w(r) \\ & \wedge m_i = f_{amc}(r, c_i f_{cc}(r, c_i))\} \end{aligned} \quad (31.23)$$

Neste ponto é importante entender o que é possível: não só associar conceitos aos recursos, mas também associar atributos e valores aos conceitos, e também aos recursos. O modelo da Figura 31.8 mostra essas relações.

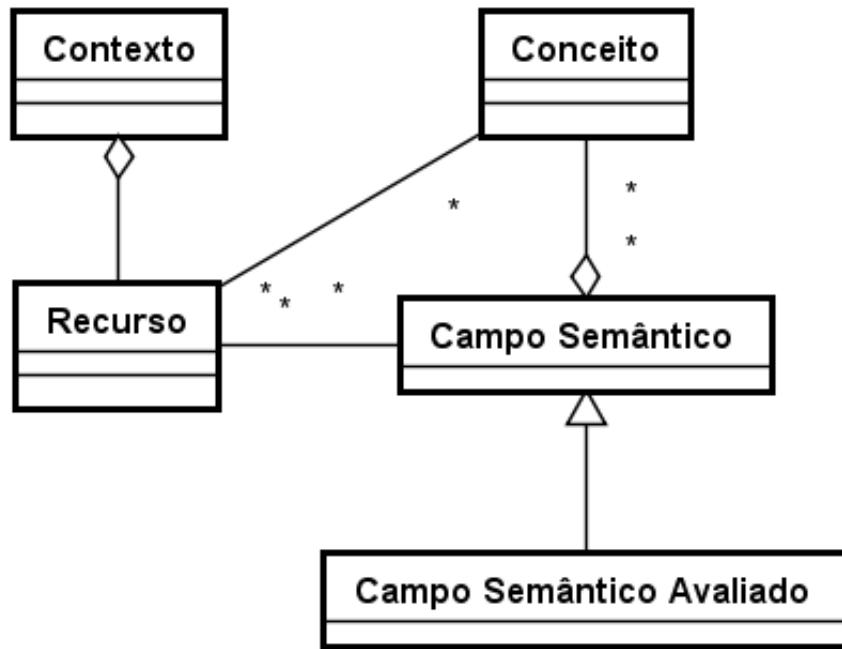


Figura 31.8: Campos semânticos e recursos em UML.

31.3.4 Gerando rótulos

Os conceitos de um campo semântico devem ser transformados em rótulos a serem usados nas nuvens, com vários atributos. Para isso são necessárias mais algumas definições. Rótulos são normalmente palavras únicas ou n-gramas, mas podem na verdade ser qualquer símbolo.

Uma **função de classificação de rótulo** é uma função f_{ct} , que, dado um recurso r e um rótulo t (do inglês *tag*) gera um conjunto de atributos $A_i, A_i \subset A$, que pode ser utilizado para representar os atributos da rótulo t quando se referem ao recurso r . O conjunto de todos os rótulos é descrito pela

letra T .

$$f_{ct} : W \times T \rightarrow \wp(A) \quad (31.24)$$

$$= \{a_{ij} | a_{ij} \text{ é um atributo de } t \text{ quando se refere a } r\} \quad (31.25)$$

Uma **função de atribuição de mapa para uma rótulo** t , é uma função f_{amt} que, dado uma rótulo t , um recurso r , e um conjunto de atributos $A_i, A_i \subset A$, gera um mapa no qual para cada atributo de A_i há um par atributo correspondente o descrevendo.

$$f_{amt} : W \times T \times \wp(A) \rightarrow M \quad (31.26)$$

$$\begin{aligned} f_{amt}(r, t, A_i) = & \{(a_{ij}, v_{ij}) \mid \forall a_{ij} \in A_i \Rightarrow \\ & ((a_{ij}, v_{ij}) \in f_{amt}(r, t, A_i) \wedge \\ & (f_{ad}(a_{ij}) = V_i))\} \end{aligned} \quad (31.27)$$

Uma **representação de rótulo** ou um **mapa de rótulo** para uma rótulo t , $MT(t)$, é um mapa:

$$MT(t) = f_{amt}(r, t, f_{ct}(r, t)) \quad (31.28)$$

Mapas de rótulo agem como as representações dos rótulos. Por exemplo, o mapa de uma rótulo pode ser formado por tuplas que representam suas propriedades na nuvem de rótulos.

Um **gerador de conjunto de rótulos** é uma função f_{gct} que dado um contexto w , um recurso específico $r, r \subset w$, gera um conjunto de rótulos $CT(r)$ que representa o grupo de rótulos que podem ser consideradas, sob alguma razão, serem aplicadas a r no contexto w .

$$f_{gct} : W \times \wp(W) \rightarrow T \quad (31.29)$$

$$\begin{aligned} f_{gct}(r, w) = & \{t_i \mid \exists c_i \in CS_w(r) \wedge \\ & \text{representa}_w(t_j, c_j)\} = CT_w(r) \end{aligned} \quad (31.30)$$

Dado um conjunto de rótulos $TC(r)$, uma **nuvem de rótulos abstrata** $TCA(r)$ é um conjunto de tuplas

$$TCA(r) = \{(t_j, m_i)\}, \quad (31.31)$$

31.3.5 Criando nuvens de rótulos

A partir das definições do modelo, é possível entender que para criar uma nuvem de rótulo é necessário seguir o seguinte processo:

1. Escolher um recurso, normalmente um documento, dentro de um contexto;
2. Criar um campo semântico avaliado para o objeto;
3. Utilizar o campo semântico avaliado para definir uma nuvem de rótulos abstrata para o objeto, e
4. Gerar uma representação visual para a nuvem de rótulos abstrata, a partir de uma mapeamento de atributos a propriedades dos rótulos.

31.3.6 Um modelo para nuvens de rótulo de sumário

Uma **nuvem de rótulos de sumário** é uma nuvem que representa todos os recursos existentes de um determinado contexto. Ela pouco se difere de uma nuvem de rótulos de um conjunto maior, mas tem seu papel teórico para a definição de nuvens de rótulo diferenciais.

Uma **nuvem de rótulo de união** (UTC) é na prática uma união de conjuntos:

$$UTC(w) = \bigcup_{r \in w} TCA(r) \quad (31.32)$$

Essa nuvem então representa todos os conceitos possíveis de todos os recursos, mas não o resumo da coleção de recursos como um todo. Para isso, é necessário criar uma **função de sumarização** é uma função $f_s(t)$, que dado um rótulo t sabe decidir se ele deve ou não aparecer na nuvem de rótulo de sumário.

$$f_s(t) \rightarrow \mathbb{B}, \quad (31.33)$$

onde \mathbb{B} é o conjunto dos booleanos $\{F, V\}$.

Sendo assim, uma **nuvem de rótulo de sumário** é definida como:

$$STC = \{t | (t \in UTC(w)) \wedge (f_s(t) = V)\} \quad (31.34)$$

Portanto, resumidamente esta nuvem de rótulos representa o conjunto de rótulos que estão relacionados a pelo menos a algum dos recursos do contexto e, ao mesmo tempo, atendem às condições especificadas por $f_s(t)$. Geralmente, esta função utilizará algum valor limite como filtro para selecionar somente os rótulos que pertencem a nuvem de rótulos de sumário.

31.3.7 Um modelo para nuvens de rótulo diferenciais

Finalmente é possível definir formalmente o conceito de **nuvem de rótulos diferencial**. Seguindo o esquema das subseções anteriores, pretende-se fornecer um modelo que possa ser utilizado por um processo para gerá-las.

O interesse é representar as diferenças de um recurso específico em relação aos outros integrantes da coleção.

O conceito de diferença é usado no sentido de realçar, destacar ou evidenciar as características ou conceitos particulares de um recurso que não estejam sendo abordados pelos demais recursos neste mesmo contexto. Portanto, uma $DTC(r,w)$ é definida como uma nuvem de rótulos, que fornece uma representação visual dos conceitos presentes em um específico recurso r , e que de alguma forma, se destacam ou se diferenciam dos demais conceitos presentes nos outros recursos, que são componentes de um determinado contexto w .

Para começar, é apresentada a **função de diferenciação** que é uma função $f_d(t,r,w)$, que irá fornecer uma condição para definir se o rótulo t deverá ser utilizado para representar um conceito que existe no recurso r que está sendo analisado mas não aparece na STC .

$$f_d : T \times W \times C \rightarrow \mathbb{B} \quad (31.35)$$

O modelo aqui apresentado cria um arcabouço de conjuntos e funções que guiam, como um template, as técnicas de construção de nuvens de rótulos, descritas na próxima seção.

Desta forma, é possível definir uma **nuvem de rótulo diferencial**, como:

$$DTC(r,w) = \{t | (t \in CRC(w) \vee t \in TCA(r)) \wedge f_d(t,r,w) = V\} \quad (31.36)$$

Logo, na $DTC(r,w)$ são encontrados rótulos que descrevem os documentos individualmente, mas não são necessariamente consideradas na nuvem de rótulos de sumário de resumo do conjunto. Sendo assim, a **principal função de uma nuvem de rótulos diferencial é representar características individuais e importantes de cada recurso** e por consequência auxiliar aos usuários no momento em que estes realizam uma busca, em certo contexto, a encontrar os recursos que eles desejam.

31.4 Principais Técnicas Utilizadas

As principais técnicas utilizadas na criação de nuvens de rótulos são típicas das áreas de Busca e Recuperação da Informação (Baeza-Yates e Ribeiro-Neto, 2011), Mineração de Texto, e Processamento de Linguagem Natural (Dan Jurafsky e J. H. Martin, 2022). Um processo para gerar uma nuvem de rótulos pode ser dividido em 7 passos:

1. Recuperação do documento, ou coleção de documentos, que pode ser tanto apenas a leitura de um arquivo conhecido como a recuperação automática de uma série de documentos na web por meio de *crawlers* (Mayworm, 2007);
2. Leitura do arquivo e extração do texto, muitas vezes exigindo o uso de bibliotecas especializadas, como o pacote `pdfminer.six` (Shinyama, Guglielmetti e Marsman, 2019), no caso de arquivos PDF;
3. Pré-processamento do texto (Dan Jurafsky e J. H. Martin, 2022), que exige passos como:
 - (a) eliminação de símbolos indesejados no processamento, como TABs, CRs e LFs, acentos e variações aplicadas às letras, como diacríticos, ou criação de representações alternativas, de modo a atender erros ;
 - (b) eliminação ou tradução de códigos que representam imagens, que não pertencem a língua sendo processada, como emoticons, ou que, em sequência, possuem um significado específico, como :-) para um rosto sorrindo;
 - (c) eliminação de comandos ou marcações que não pertencem ao texto, como marcações HTML ou XML, normalmente usando pacotes como *Beautiful Soup* (Richardson, 2007);
 - (d) transformação para minúsculas ou maiúsculas, conhecida como *case-folding*, se e como desejado;
 - (e) tokenização, i.e., separação do documento em tokens, ou palavras, e possível geração de n-gramas (Christopher D. Manning, Raghavan e Schütze, 2009b);
 - (f) correção de erros de digitação ou de ortografia (Norvig, 2007, 2009);
 - (g) tratamento linguístico, em busca da descoberta da classe gramatical, função sintática, significados, etc.
 - (h) eliminação de palavras indesejadas na nuvem, tanto *stopwords*, aquelas que não trazem isoladamente diferenciação do significado do texto, como conjunções, quanto palavras que aparecem sempre em uma coleção (Baeza-Yates e Ribeiro-Neto, 2011; Christopher D. Manning, Raghavan e Schütze, 2009b), como o nome do site onde as palavras foram obtidas;
 - (i) eliminação das variações de gênero, número, tempo, grau, etc, e ainda, em alguns algumas vezes, a lematização, i.e., buscar a forma padrão da língua, possivelmente por meio de tabelas, ou a radicalização (*stemming*), i.e., remoção de sufixos e prefixos Baeza-Yates e Ribeiro-Neto (2011), Dan Jurafsky e J. H. Martin (2022), Christopher D. Manning, Raghavan e Schütze (2009b) e Porter (1980).
4. Geração de estruturas de dados intermediárias, na forma de índices, ou estruturas chave-valor, como listas invertidas, matrizes termo-documento ou ainda baseados na semântica latente, modelos de linguagem, etc; incluindo preocupações com n-gramas (Baeza-Yates e Ribeiro-Neto, 2011; Christopher D. Manning, Raghavan e Schütze, 2009b);

5. Extração da informação desejada, na forma de uma lista ordenada de rótulos e propriedades, geralmente apenas uma lista palavra-peso (F. F. Morgado, 2010);
6. Mapeamento das propriedades associadas aos rótulos um forma de representação dos mesmos, geralmente apenas o tamanho da fonte, e
7. Geração final do diagrama, com posicionamento, embelezamento estético, etc. (F. F. Morgado, 2010).

Apesar do pré-processamento de texto ser composto de tarefas simples, ou pelo menos amplamente conhecidas, ele é crítico na qualidade das nuvens de rótulo, principalmente quando é necessário levar mais em conta o conceito que as palavras representam do propriamente as palavras. A confluência de palavras, por meio da eliminação de suas variações. Por exemplo, a confluência pela variação em número pode garantir que a forma plural e singular sejam representadas de maneira unificada na nuvem, juntando palavras como “nuvem” e “nuvens”. Por outro lado, pode não ser interessante fazer a confluência em gênero, para poder separar palavras como “cidadão” e “cidadã” em um estudo de gênero. A eliminação de *stopwords*, que hoje é pouco preocupante em outras técnicas de Processamento de Linguagem Natural, também é tipicamente essencial, para evitar que a nuvem fique repleta de palavras como “que” e “mas”. Por outro lado ela deve também ser feita com cuidado, já que palavras que constam de listas de *stopwords* podem ser importantes na compreensão do texto, como o uso de pronomes pessoais e possessivos.

Muito do que é feito no passo 4 parte de técnicas básicas, como a simples contagem de frequência, no caso de um único documento. Técnicas mais avançadas buscam conceitos em vez das palavras retiradas diretamente do texto, como a Análise por Semântica Latente (LSA) (Thomas K. Landauer, McNamara et al., 2007), ou a Análise Latente de Dirichlet (LDA) (Blei, A. Y. Ng e M. I. Jordan, 2003), usadas no passo 4, são na prática técnicas de dimensão de dimensionalidade, que tentam fazer com que palavras encontradas no texto se juntem em grupos, permitindo usar dessas palavras como rótulos. Essa unificação de palavras em um conceito também pode ser conseguido por meio de tesouros, como o *WordNet* (Fellbaum, 1998).

Um tesouro é uma lista de palavras com significados semelhantes, sendo que a semelhança pode ocorrer por sinônima, hiperónima, ou outras formas. O *WordNet* foi criado na Universidade de Princeton e se tornou um padrão *de facto* para a língua inglesa, e uma referência de formato adotada por várias outras iniciativas em outras línguas. No *WordNet*, as palavras estão reunidas em conjuntos de um mesmo significado conhecidos como *synset*. Iniciativas semelhantes em português incluem o OpenWordNet-PT (V. d. Paiva, Rademaker e Melo, 2012).

Recentemente, há um interesse em “semantic word clouds”, que buscam fazer uma representação que aproxima mas os rótulos que tem significado similar Hearst et al., 2020; Jo, B. Lee e Seo, 2015, como mostrado na Figura 31.6. Esses modelos usam as técnicas citadas nos parágrafos anteriores ou similares.

F. F. Morgado (2010) mostra uma instância do processo de criação de nuvens de rótulo por meio de um diagrama de atividades UML, adaptado na Figura 31.9. Além disso, F. F. Morgado (2010) propõe algoritmos para duas abordagens para criação de nuvens de rótulos abstratas.

Para implementação dessas técnicas, existem pacotes bastante completos oferecendo uma grande variedade de técnicas de Processamento de Linguagem Natural, como o NLTK (Bird, Loper e Klein, 2009) para Python e Weka para Java (Workbench, 2106). Existem também pacotes específicos para a geração de nuvens de rótulos, como o *wordclouds* (Mueller, 2020).

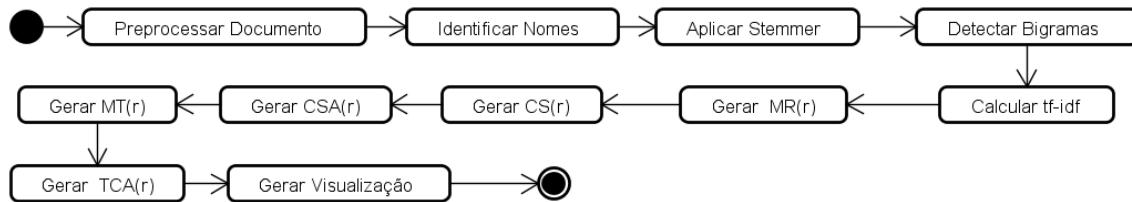


Figura 31.9: Uma instância de um processo de geração de nuvens de rótulo. Adaptado de: F. F. Morgado (2010).

31.5 Aplicações

Esta seção exemplifica o uso de nuvens de rótulos. É importante lembrar que o objetivo geral de uma nuvem de rótulos é representar, de uma forma facilmente compreendida, um objeto, normalmente um documento, ou um conjunto de objetos. Outro uso comum é na diferenciação entre duas nuvens, e aí entram as nuvem diferenciais e de resumo.

Dessa forma, havendo duas coleções de documentos, é possível compará-los das seguintes formas: pela comparação visual de duas nuvens de rótulos, uma para cada coleção, ou pela comparação de três nuvens de rótulos: uma que mostra o que todos os documentos tem em comum, a nuvem de rótulo sumário, e uma para cada coleção de documentos, que mostra como esse subconjunto se diferencia do todo.

B. D. d. Paiva e Pinho (2022) apresentam uma análise visual, na forma de nuvens de rótulos, de projetos de lei da Câmara de Deputados, sendo que essa análise visual é criada por meio de um algoritmo de LSA Thomas K. Landauer, McNamara et al., 2007. Essa é uma forma sofisticada de fazer a visualização por nuvens de rótulos, pois busca mostrar não as palavras encontradas no texto por sua frequência, mas sim por seu significado. Nesse trabalho, foram analisados apenas Projetos de Lei propostos por deputados, entre 1934 e 2022, do site Dados Abertos da Câmara³. Para exemplificar, mostramos nas Figuras 31.10 e 31.11 como comparar os tópicos mais relevantes nos períodos de 2000 a 2019 e ao período da pandemia, 2020 a 2022. É possível notar o aparecimento da palavra pandemia, a diminuição da relevância do termo trânsito, e o aparecimento dos termos penal e crime de forma bastante relevante, possivelmente indicando medidas de segurança pública. Nesse caso são comparadas nuvens de rótulos tradicionais.

Dias e T. M. Paiva (2022) apresentam outra aplicação de nuvens de rótulos tradicionais, que mostra quais os cômodos mais comentados em relação aos aluguéis no *Airbnb*. Nesse caso, fazem também uma análise de sentimentos, permitindo a separação dos cômodos mais elogiados dos mais criticados. Esse tipo de trabalho tem grande interesse da indústria. Uma rápida análise visual das nuvens geradas a partir do texto das revisões em inglês no site da empresa mostra que enquanto banheiros e cozinhas recebem a maior quantidade de opiniões positivas, banheiros recebem a maior quantidade de opiniões negativas. As palavras que aparecem na nuvem foram determinadas por uma lista criada pelos autores, o que também é uma opção para casos onde se busca informação específica. A análise de sentimentos foi feita a partir da atribuição de polaridades a partir da biblioteca `textblob` Loria, 2018 e as nuvens com a biblioteca `wordcloud` (Mueller, 2020).

Já Boechat e Kuchpil (2022), apresentaram análise das publicações de Computação e como elas vêm se alterando ao longo do tempo na UFRJ, desta vez baseados em nuvens de rótulos diferenciais. Para isso o autor gerou um nuvem de rótulos de sumário a partir dos títulos de todas as publicações

³<https://dadosabertos.camara.leg.br>



Figura 31.10: Relevância de termos para o período 2000-2019. Fonte: B. D. d. Paiva e Pinho (2022)

entre 2002 e 2022, apresentada na Figura 31.12 e ainda a nuvem de rótulos diferencial para cada três anos. Como exemplo, as Figuras 31.13 e 31.14 permitem avaliar o que fica constante e o que tem mudado pesquisa nos últimos 20 anos, e indica o compromisso com a inovação, que aparece fortemente nos dois casos. Pode-se observar que a palavra “innovation” não aparece nos anos intermediários como relevante.

Esses exemplos mostram diferentes tipos de análise que podem ser feitas: comparando itens no tempo, pela atuação de partidos políticos, pelo interesse específico em negócios, etc. Muitos usos podem ser imaginados.

Outros grupos de trabalho investigaram o uso ou aplicaram diretamente nuvens de rótulos. Não é o foco deste trabalho fazer uma revisão ou uma comparação, porém alguns trabalhos são interessantes tanto por fornecer indicações sobre o uso das nuvens de rótulos, como por servir como inspiração para aplicações.

Relativo ao uso de nuvens como ferramenta de recuperação da informação, Sinclair e Cardew-Hall (2008) concluíram que a busca por meio de nuvens de rótulos ser útil como apoio a navegação e busca por serendipidade mas, por outro lado, a interface de busca é melhor para os usuários com objetivos claros. Quanto à visualização, já citamos anteriormente os trabalhos de Viégas e Wattenberg (2008) e Viégas, Wattenberg e Feinberg (2009), que estudaram como tornar nuvens de rótulo mais corretas em relação as regras de design, propondo a ferramenta *wordle*.

Quanto a usos específicos, Egler, Costa e Gonçalves (2021) usam nuvens de palavras para trazer significado a descrição dos assuntos tratados por grupos políticos de direita brasileiro nas redes sociais, usando a cor para identificar o tipo de mídia onde as palavras foram encontradas. O uso de nuvens de rótulos para avaliar simultaneamente uma grande quantidade de documentos é uma das suas melhores aplicações.

Kui et al. (2022), por exemplo, afirmam que nuvens de rótulos, no contexto educational, são uma ferramenta para predição e recomendação de recursos de aprendizado.

Outra aplicação interessante foi proposta por Al-Msie'deen (2019): o uso de nuvens de rótulos para visualização de código fonte orientado a objeto, criando nuvens para identificadores em geral e também

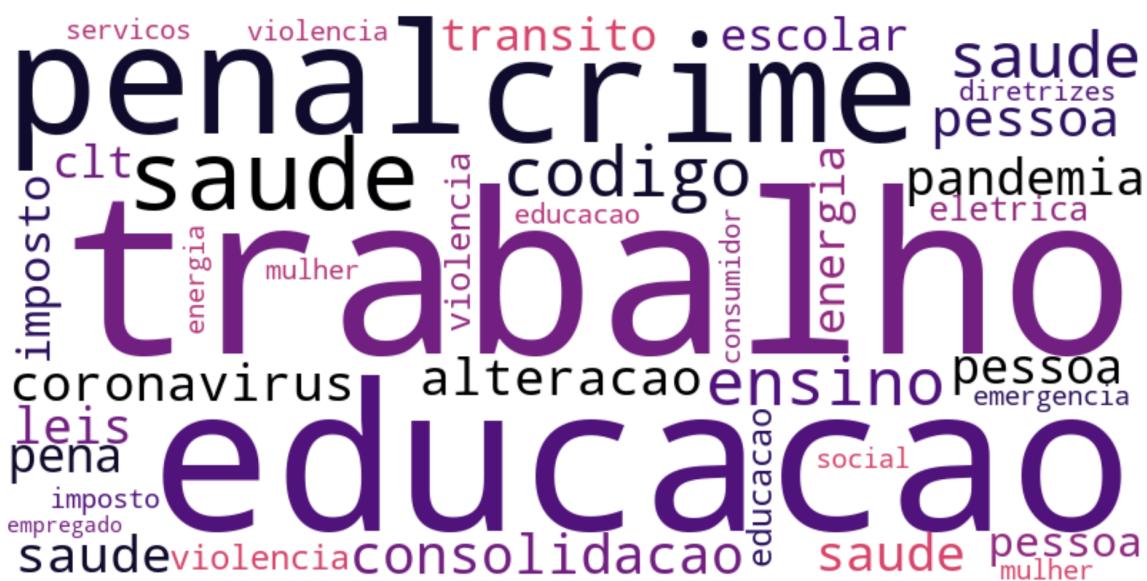


Figura 31.11: Relevância de termos para o período 2020-2022. Fonte: B. D. d. Paiva e Pinho (2022)

divididos em pacotes, classes, métodos e atributos. Ela mostra que atributos específicos podem ser obtidos dos conceitos, no caso o tipo de identificador, e representados na nuvem.

Alguns artigos também mostram variações possíveis. Momin, Kays e Sadri (2022) apresenta Redes de Bigramas, uma representação gráfica onde palavras que se relacionam para a par são mostradas como um grafo, onde as palavras são os vértices e os relacionamentos são os arcos.

Li, Dong e Yuan (2018) mostram algumas alternativas para incluir nuvens de rótulos em mapas como estratégia de visualização de que pontos de interesse existem em uma região. Assim, cada área do mapa é na verdade uma nuvem de rótulos diferente.

Outra variante é uma nuvem de dados, *data cloud*. Nessa variante, os rótulos são algum campo específico de uma base de dados, como o nome de um país, e os atributos são outros campos, como usar a população para o tamanho da fonte e a cor para o continente.

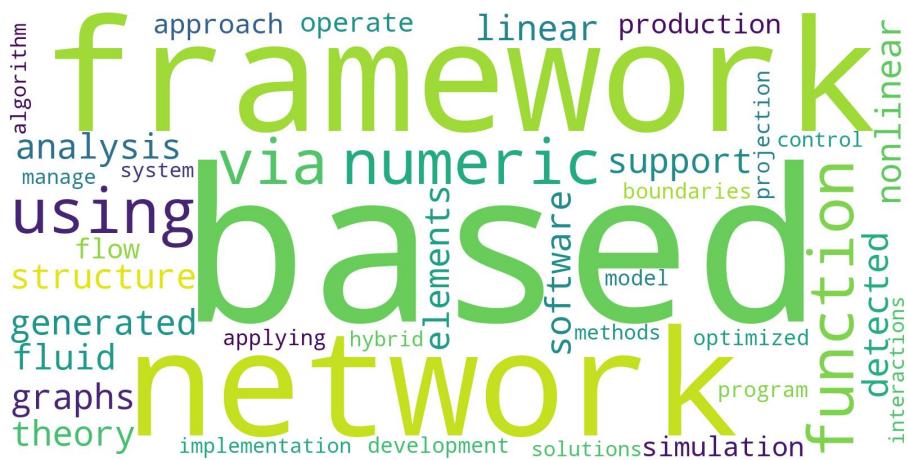


Figura 31.12: Nuvem de rótulos de sumário para os títulos das publicações entre 2002 e 2022. Fonte: Boechat e Kuchpil (2022)

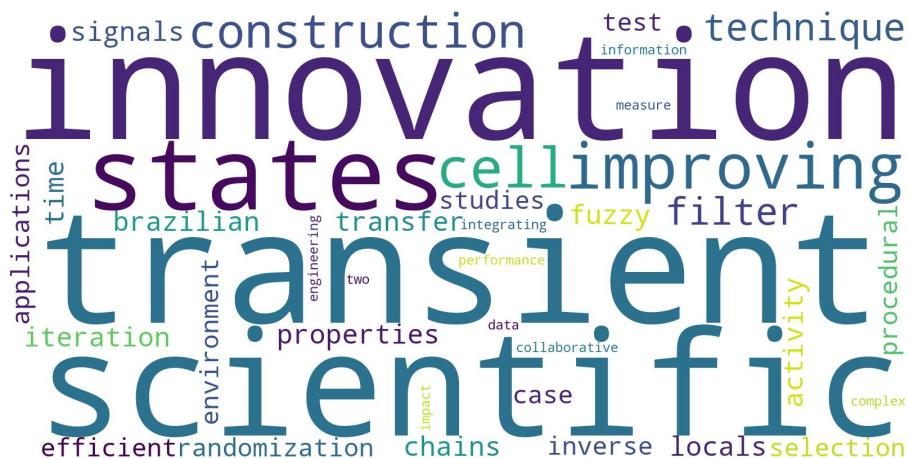


Figura 31.13: Nuvem de rótulos de diferencial para os títulos das publicações entre 2002 a 2004.
Boechat e Kuchpil (2022)

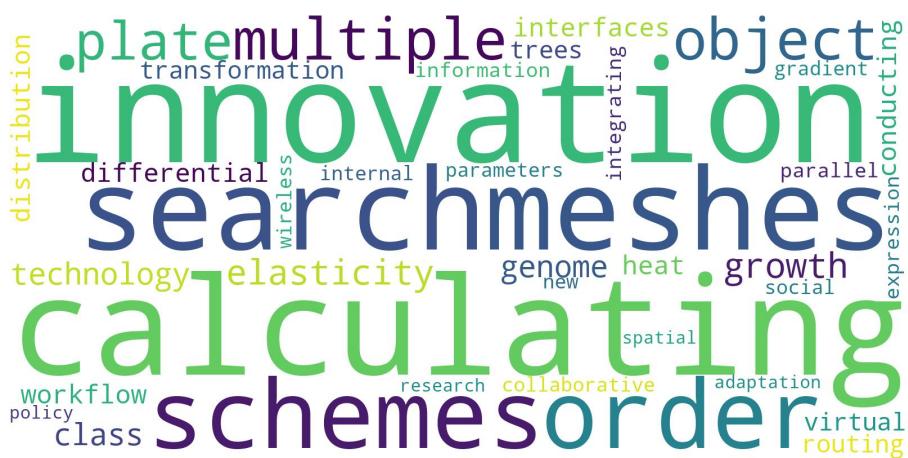


Figura 31.14: Nuvem de rótulos de diferencial para os títulos das publicações entre 2020 a 2022. Boechat e Kuchpil (2022)

31.6 GTAGLIB

GTAGLIB é uma biblioteca em Python, descrita em Mattos (2022), capaz de gerar nuvens de rótulos diferenciais e de sumário. A biblioteca implementa dois métodos, sendo que o primeiro usa os termos mais relevantes no documento e o segundo usa, além desses termos, sinônimos presentes no WordNet (Fellbaum, 1998). Além disso, o primeiro parte das nuvens diferenciais e depois gera a nuvem sumário, enquanto o segundo as gera a nuvem sumário antes de gerar as diferenciais.

O método 1 segue os seguintes passos:

1. tokenização;
2. uso do stemmer de Porter (Porter, 1980);
3. remoção das *stopwords*;
4. filtragem para aceitar apenas substantivos, por meio de POS-Tagging (Dan Jurafsky e J. H. Martin, 2022), que rotula a classe gramatical da palavra;
5. geração de bigramas;
6. cálculo do TF-IDF de cada palavra;
7. escolha dos termos mais relevantes como campo semântico;
8. criação da nuvem abstrata a partir do campo semântico;
9. criação da nuvem diferencial (são criadas primeiros);
10. criação da matriz termo-documento;
11. aplicação do LSA;
12. aplicação do Coeficiente de Correlação de Pearson na matriz reconstruída para obter uma matriz de correlação de termos;
13. escolha de um termo raiz;
14. escolha dos termos mais correlacionados com o termo raiz, e
15. construção da matriz de sumário.

O método 2 repete o processo até o passo 7, e segue:

8. expansão do campo semântico com os sinônimos das palavras contidas no próprio campo, usando o Wordnet;
9. criar a nuvem abstrata de cada documento;
10. ordenar a união dos termos das nuvens;
11. escolher os primeiros quarenta termos como os de maior relevância, e
12. para cada documento, construir a nuvem diferencial com os termos não presentes no sumário, colorindo os termos expandidos em outra cor.

A Programa 31.1 apresenta um exemplo de uso.

Programa 31.1: Fragmento de código com exemplo de uso da GTAGLIB

```

1 from gtaglib.generator import TagGenerator
2
3 set_summary, differential = TagGenerator(
4     semantic_field_size=40,
5     stemmer="porter",
6     generate_bigrams=True,
7     use_tfidf=True
8 ).generate_tag_cloud(dataset,1,root="step",
9     outputdir="myoutputdir")

```

O tf-idf, acrônimo em inglês para frequência do termo e frequência inversa nos documentos, é uma medida usada na busca e recuperação da informação que indica a importância relativa de um termo em um documento em relação a uma coleção de documentos. Apesar de existirem muitas fórmulas, que produzem valores diferentes, para o conceito, uma das mais citadas é (Baeza-Yates e Ribeiro-Neto, 2011):

$$TF - IDF = f_i \log \frac{N}{n_i}, \quad (31.37)$$

onde f_i é a frequência do termo i no documento em questão, N o número de documentos na coleção e n_i o número de documentos da coleção onde o termo i aparece.

A biblioteca pode ser encontrada em <https://github.com/GuilhermeCaeiro/gtaglib>. Sua classe principal é a `TagGenerator`. Para gerar a nuvem de rótulos abstratas podem ser usados os métodos `generate`, ou `generate_tag_clouds`. Para gerar as imagens, a `gtaglib` atualmente usa a biblioteca `wordcloud` internamente.

DRAFT

CAPÍTULO 32

DETECÇÃO DE PLÁGIO

Plagiar é apresentar como sua uma obra intelectual, ou ideias, que pertença a outra pessoa. O plágio é uma violação dos direitos do autor ou dos seus direitos conexos, descritos na lei 9.610 de 1998, e é um crime previsto no Art. 184 do Código Penal, sendo prevista as penas de multa e detenção. Conceitualmente, plagiar é roubar a propriedade intelectual.

Plagiar não é apenas copiar o texto, o que é chamado de **plágio direto**, mas também fazer transliterações, ou mesmo apresentar como sua uma ideia, sem referenciar corretamente o autor. Também é plágio suprimir uma fonte secundária, citando diretamente a fonte primária sem consultá-la.

Abreu (2011) e Duarte (2017) trabalharam com o problema de plágio e fizeram revisões do assunto, tanto conceituais como técnicas, e são usados como guias para este capítulo.

32.1 Propriedade intelectual e as leis brasileiras

Segundo a lei nº 9610 de 19 de fevereiro de 1998, que trata da proteção da propriedade intelectual na forma de direitos autorais: “São obras intelectuais protegidas as criações do espírito, expressas por qualquer meio ou fixadas em qualquer suporte, tangível ou intangível, conhecido ou que se invente no futuro” (Brasil, 1998, Art. 7º) sendo que “No domínio das ciências, a proteção recairá sobre a forma literária ou artística, não abrangendo o seu conteúdo científico ou técnico, sem prejuízo dos direitos que protegem os demais campos da propriedade imaterial” (Brasil, 1998, Art. 7º, §3º). Essa proteção independe de registro (Brasil, 1998, Art. 18º). A lei cita exemplo de tais obras, como: texto de obras literárias, composições musicais com ou sem letra, adaptações apresentadas como criação intelectual novas, programas de computador, ilustrações e arte cinética (Brasil, 1998, Art. 7º).

Ainda segundo a mesma lei, são direitos morais do autor (Brasil, 1998, Art. 24):

- I. o de reivindicar, a qualquer tempo, a autoria da obra;
- II. o de ter seu nome, pseudônimo ou sinal convencional indicado ou anunciado, como sendo o do autor, na utilização de sua obra;
- III. o de conservar a obra inédita;

- IV. o de assegurar a integridade da obra, opondo-se a quaisquer modificações ou à prática de atos que, de qualquer forma, possam prejudicá-la ou atingi-lo, como autor, em sua reputação ou honra;
- V. o de modificar a obra, antes ou depois de utilizada;
- VI. o de retirar de circulação a obra ou de suspender qualquer forma de utilização já autorizada, quando a circulação ou utilização implicarem afronta à sua reputação e imagem;
- VII. o de ter acesso a exemplar único e raro da obra, quando se encontre legitimamente em poder de outrem, para o fim de, por meio de processo fotográfico ou assemelhado, ou audiovisual, preservar sua memória, de forma que cause o menor inconveniente possível a seu detentor, que, em todo caso, será indenizado de qualquer dano ou prejuízo que lhe seja causado.

A lei também prevê usos que não constitui ofensa ao direito autoral, como “a citação em livros, jornais, revistas ou qualquer outro meio de comunicação, de passagens de qualquer obra, para fins de estudo, crítica ou polêmica, na medida justificada para o fim a atingir, indicando-se o nome do autor e a origem da obra” (Brasil, 1998, Art. 46).

Finalmente, é importante dizer que no Brasil, os direitos morais do autor são inalienáveis e irrenunciáveis (Brasil, 1998, Art. 27).

Não cumprir essa exigência do enunciado incorre na violação dos direitos do autor, crime contra a propriedade intelectual previsto no Título III, Capítulo I, Art. 184 do Código penal: “reprodução total ou parcial, com intuito de lucro direto ou indireto, por qualquer meio ou processo, de obra intelectual, interpretação, execução ou fonograma, sem autorização expressa do autor”, sendo que o mesmo artigo do Código Penal ressalta no §4 as exceções e limitações da Lei 9610/1998 (Brasil, 1940).

A partir dessa breve revisão das leis brasileiras, é fácil entender que se a obra de um autor é utilizada, então deve ter seu nome anunciado como autor da mesma, e que a citação está autorizada por lei. Não cumprir esse dever resulta em crime.

32.2 Classificações para o plágio

Diferentes autores propõe formas de classificar o plágio. Maurer, Kappe e Zaka (2006) propõe quatro classificações para o plágio:

1. **plágio acidental**, causado pela falta de conhecimento do conceito de plágio e das regras existente para referenciar um trabalho;
2. **plágio não-intencional**, causado pela geração paralela de ideias a partir de conhecimento existente;
3. **plágio intencional**, o ato deliberado de plagiar, e
4. **auto-plágio**, quando o próprio autor repete trabalhos anteriores seus sem citá-los.

F. K. Taylor (1965) já citava o efeito da criptoamnésia no plágio considera não intencional. Ela pode ser definida como a “presença de um fenômeno na consciência normal que objetivamente são memórias, mas subjetivamente não são reconhecidas como tal”. Isso leva a que a pessoa experiente esse fenômeno como “uma criação original da sua mente”, levando ao plágio não intencional (F. K. Taylor, 1965).

Duarte (2017) sintetiza as contribuições de B. Martin (1994), Maurer, Kappe e Zaka (2006) e Weber-Wulff (2010) em 11 formas de plágio (Duarte, 2017):

1. Copiar e colar, a forma mais simples e mais fácil de detectar;
2. Plágio de fontes secundários, onde não se registra a existência de uma fonte secundária de onde o texto foi obtido, e se cita as fontes primárias sem consultá-las;
3. Plágio da forma da fonte, onde se repete a estrutura da fonte secundária sem citá-la, citando-se apenas as fontes primárias;
4. Plágio artístico, feito por uso de meios diferentes para reproduzir o mesmo conteúdo, como na gravação de um texto de outrem;
5. Plágio de ideia, forma mais sofisticada, onde apenas a ideia é usada;
6. Plágio de tradução, sem que seja feita a devida referência a fonte;
7. Paráfrase, que se utiliza de uma nova apresentação de um texto para repetir suas ideias;
8. Coleções misturadas e coladas, onde o texto se assemelha a uma colagem aleatória de outras fontes;
9. Colcha de retalho ou mosaicos de orações, caracterizado pela mudança de poucas palavras das fontes;
10. Plágio estrutural, onde é feita a paráfrase e ainda o uso de estruturas, como a estrutura argumentativa, e
11. Plágio de conspiração, típico de situações escolares onde um aluno resolve uma questão e os outros apresentam soluções copiadas e modificadas para esconder o plágio.

DRAFT

Apêndices

DRAFT

APÊNDICE A

TÓPICOS EM PYTHON

A.1 Dicionários

Dicionários em Python são implementados com uma hash-table com endereçamento aberto que cresce dinamicamente quando está 2/3 ocupada, iniciando com 8 *slots*.

d = create an empty dictionary and assign it to d
d[key] = value assign a value to a given dictionary key
d.keys() the list of keys of the dictionary
list(d) the list of keys of the dictionary
sorted(d) the keys of the dictionary, sorted
key in d test whether a particular key is in the dictionary
for key in d iterate over the keys of the dictionary
d.values() the list of values in the dictionary
dict([(k1,v1), (k2,v2), ...]) create a dictionary from a list of key-value pairs
d1.update(d2) add all items from d2 to d1
defaultdict(int) a dictionary whose default value is zero

Tabela A.1: Alguns formas de usar dicionários em Python

Código	Resultado
<code>d = {}\\</code>	cria um dicionário vazio
<code>d = {1 : 10 , 2 : 20 , ...}\\</code>	cria um dicionário a partir de duplas chave:valor
<code>d=defaultdict(f)</code>	cria um dicionário cujo valor default é fornecido por uma função f
<code>dict([(k1,v1), (k2,v2), ...])</code>	cria um dicionário a partir de uma lista de tuplas chave-valor
<code>d[key] = value</code>	insere um par chave valor no dicionário
<code>d1.update(d2)</code>	adiciona os itens do d2 no d1
<code>key in d</code>	testa se uma chave está em um dicionário
<code>list(d)</code>	retorna uma lista de chaves do dicionário
<code>sorted(d)</code>	retorna as chaves do dicionário ordenadas
<code>d.keys()</code>	retorna uma <i>iterable view</i> das chaves do dicionário
<code>d.values()</code>	retorna uma <i>iterable view</i> dos valores do dicionário
<code>for key in d</code>	faz uma iteração nas chaves do dicionário, e não nos pares

A.2 Strings

A maneira de representar texto em Python é por meio do Tipo **str**, conhecido como string¹.

Exemplos de string são:

```

1 "A"
2 'A'
3 "Uma frase"
4 'Uma frase'
5 """ Uma frase longa
6 muito longa
7 com várias linhas"""

```

Exemplos de strings.

Como visto no exemplo acima, strings devem ser delimitadas por aspas (""), apóstrofes (''), ou três aspas quando se deseja uma string que seja muito longa e use muitas linhas².

Existem muitas funções que trabalham com strings no pacote **string**, que serão vistas mais tarde.

As strings podem ser concatenadas para criar outras strings. Para isso também se usa o operador '+'. Uma string vazia, sem caracteres, é indicada abrindo e fechando a string sem nada no meio, como em """. Uma string pode conter espaços, como em " ", uma string com um só espaço.

¹Em português muitas vezes usamos o termo sequência de caracteres, mas não há uma tradução direta e a palavra "string" será usada no texto

²Muitas linguagens de programação possuem dois tipos, um de caracteres, ou seja, uma letra apenas, e um de strings, possivelmente sendo o segundo um vetor do primeiro. Em Python, não existe um tipo caractere. Uma string de uma letra é também uma string

O tamanho de uma string é dado pela função `len(s)`³. Essa é uma função muito usada, inclusive com outros tipos de dados.

Seguem alguns exemplos:

```
1 s1 = "alfa" + "beto"
2 s2 = "alfa" + " " + "beto"
3 t1 = len("alfabeto")
4 t2 = len("a l f a b e t o")
```

Não existe um operador para subtrair strings, mas existem funções que permitem modificá-las de várias formas, ou buscar strings dentro de strings.

Na seção A.2.5 veremos mais funções sobre strings.

A.2.1 Índices de Strings

Um dos operadores mais poderosos sobre strings é o `[]`. Ele permite indexar um caractere específico da String ou pegar uma fatia da String. Ele também permite contar do início para o fim, ou do fim para o início (nesse caso usando números negativos).

A primeira letra de uma string está na posição zero.

Programa A.1: Obtendo um caractere indexado em uma string

```
1 print("beto"[0])
2 print("beto"[3])
```

Saída para o Programa A.1

```
1 b
2 o
```

Na contagem ao contrário, a última letra está na posição `-1`.

Programa A.2: Obtendo um caractere indexado negativo em uma string

```
1 print("beto)[-1]
2 print("beto)[-3])
```

Saída para o Programa A.2

```
1 o
2 e
```

Como vemos, há uma pequena diferença na forma de contar. Ela existe por alguns motivos. Primeiro, não existe o número `-0`, segundo, ela permite que o trabalho com fatias funcione melhor.

³Nesse livro usamos uma marca especial para permitir que o leitor veja o espaço, ele não existe e não deve ser digitado no programa.

A.2.2 Fatiadas de Strings

Fatiadas são pedaços de strings.

Mais interessante ainda que poder pegar caracteres um a um, em Python podem pegar pedaços ou fatiadas de strings usando também o operador [] .

Primeiro vamos ver alguns exemplos mais simples, usando a notação início e fim da fatia, para frente e para trás. Para isso, usamos o operador fatia no formato [`<início>:<fim>`].

Programa A.3: Fatiando uma string

```
1 print("beto"[0:1])
2 print("beto"[0:2])
3 print("beto"[-3:-1])
```

Saída para o programa Programa A.3

1 b
2 be
3 et

Para entender como funciona devemos entender que as posições nas fatiadas, positivas ou negativas, não indicam verdadeiramente as posições, mas sim os intervalos entre as letras. A Figura A.1 demonstra isso. Na figura devemos notar que a posição positiva final não pode ser usada em índices, apenas em fatiadas.

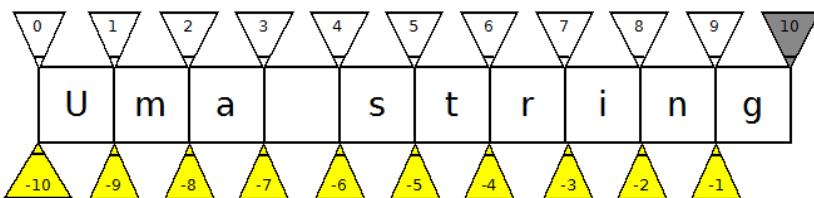


Figura A.1: As posições de indexação das strings estão na verdade entre as letras

A notação de fatiadas também pode ser usada com 3 argumentos, sendo que o terceiro é responsável por indicar o passo, ou pulo, que é dado, ao pegar os caracteres, usando para isso a notação [`<início>:<fim>:<passo>`].

Dessa maneira, podemos criar fatiadas especiais que pulam letras, como em:

Programa A.4: Fatiadas que pulam letras.

```
1 print("alfabeto brasileiro"[1:12:2])
```

Saída para o programa Programa A.4

1 laeoba

Outra característica: se a posição inicial, ou final, é deixada em branco, então significa que queremos a fatia a partir do início, ou fim, da string.

Programa A.5: Marcando só a posição final.

```
1 print("alfabeto)[:5]
2 print("alfabeto"[5:])
```

1 alfab _____ Saída para o programa Programa A.5 _____
2 eto

E se deixamos início e fim vazios uma cópia da string é criada.

A.2.3 O operador in

Strings, como todas as sequências, possui um operador `in` que diz se um elemento está dentro dela. Também é possível verificar se um elemento não pertence a uma string usando o operador `not in`.

```
>>> "1" in "1234"
True
>>> "5" in "1234"
False
>>> "1" not in "1234"
False
>>> "5" not in "1234"
True
```

A.2.4 Comparando Strings

A princípio, strings são comparadas em sua ordem lexicográfica, isto é, primeiro na "ordem alfabética", depois pelo tamanho da string.

Na verdade, cada caractere tem um valor inteiro, ou seu *Unicode code point*. Esse valor pode ser calculado com a função `ord(c)`, onde `(c)` é uma string de 1 caractere apenas. Também podemos calcular o caractere dado um número, com a função `chr(i)`, onde `i` é um número inteiro até 1.114.111, ou `0x10FFFF`, o maior caractere Unicode.

Por necessidades de representação, "A" e "a" são caracteres diferentes para o computador.

Programa A.6: Cada letra equivale a um número.

```
1 print(ord("a"))
2 print(ord("A"))
3 print("a" < "A")
```

1 97 _____ Saída para o programa Programa A.6 _____
2 65
3 False

A.2.5 Funções com String

- `lower()`, coloca todas as letras em minúscula;
- `slipt()`, devolve uma lista de tokens, considerando apenas uma caractere para separação, e
- `upper()`, coloca todas as letras em maiúscula.

A.2.6 Exercícios

Exercício A.1:

Ana recebeu de Beto uma mensagem em código, onde cada terceira letra apenas deve ser lida. Se a mensagem recebida foi "asefduds grtrwewe jyacgmsdo", qual a mensagem original?

```
print("asefduds grtrwewe jyacgmsdo" [2::3])
```

Veja que como as letras de uma string são contadas a partir do número zero, a primeira terceira letra é a de índice 2.

Exercício A.2:

Dê o resultado para as seguintes expressões:

```
print("alfabeto brasileiro" [::2])
print("alfabeto brasileiro" [11:5:-1])
print("alfabeto brasileiro" [::-1])
```

Exercício A.3:

Ana resolveu responder a Beto, usando novamente um código, porém agora eles combinaram inverter a string e depois escolher apenas as letras pares. Lembre, porém, que eles sempre começam contando a primeira letra como 1. A mensagem de Ana para Beto foi 'fohduaeryoemwatny sokhrnjertm wudEn'

¹ `'fohduaeryoemwatny sokhrnjertm wudEn' [::-1][1::2]`

Nessa solução usamos dois passos. Como o operador `[]` acontece primeiro a esquerda, então primeiro invertemos a string para depois pegar as letras pares.

APÊNDICE B

TÓPICOS SELECIONADOS DE MATEMÁTICA

B.1 Autovetores e Autovalores

Seja uma matriz quadrada \mathbf{A} , que pode ser vista como indicando uma transformação linear $\mathbb{R}^N \rightarrow \mathbb{R}^N$ em um mesmo espaço vetorial V . Como \mathbf{A} opera sobre o mesmo espaço vetorial, é também chamada de um operador (Axler, 2015).

Um subespaço U de V é invariante sob \mathbf{A} se $u \in U \implies \mathbf{A}u \in U$ (Axler, 2015).

Seja $v \in V$ e $v \neq 0$, e U definido como o conjunto de todos os múltiplos escalares de v (Axler, 2015):

$$U = \{\lambda v : \lambda \in \mathbb{R}\} \quad (\text{B.1})$$

Se U é invariante sob um operador \mathbf{A} , então existe um λ tal que (Axler, 2015).:

$$\mathbf{A}v = \lambda v \quad (\text{B.2})$$

Seja \mathbf{A} um operador em V , um número $\lambda \in \mathbb{R}$ é um **autovalor** de \mathbf{A} se existe um $v \in V$ tal que $v \neq 0$ e $\mathbf{A}v = \lambda v$ (Axler, 2015).

Supondo que o operador \mathbf{A} tem o autovalor λ , então o vetor $v \in V$ é tal que $v \neq 0$ e $\mathbf{A}v = \lambda v$ é chamado de **autovetor** de \mathbf{A} (Axler, 2015).

DRAFT

APÊNDICE C

PROGRAMAS ÚTEIS

Programa C.1: Programa que gera os arquivos para o aprendizado da detecção de codificações

```

1 from bs4 import BeautifulSoup
2
3 FOLDERIN = "F:/Datasets Text/CetenFolha/" # diretório dos arquivos
4 AROW = (1,257) # tamanho da matriz inicial e das que são somadas a ela
5 CODEPAGES = ["cp1252","utf-8","utf-32","utf-16"] # codepages que vamos usar
6 FOLDERTEMP = FOLDERIN+"Minis/"
7 FILEIN = "CETENFolha-1.0"
8 FILEOUTTEXT = ".temp"
9 SIZEMINIFILE = 50 # linhas por arquivo
10 TOTALFILES = 1000 # número de arquivos
11 f = open(FOLDERIN+FILEIN,"r",encoding="cp1252")
12
13 print(f.encoding)
14
15 ff = {}
16
17 lcounter = 0
18 fcounter = 0
19 try:
20     for line in f:
21         if lcounter % SIZEMINIFILE == 0: # if it is a new file
22             for cps in CODEPAGES: # open all files
23                 ff[cps] = open(FOLDERTEMP+cps+
24                               str(fcounter)+FILEOUTTEXT,
25                               "w", encoding=cps)
26             cleantext = BeautifulSoup(line, "lxml").text # Limpa HTML
27             for cps in CODEPAGES: # write in all files
28                 ff[cps].write(cleantext)
29             lcounter += 1 # a new line done
30         if lcounter % SIZEMINIFILE == 0: # lines enough

```

```
31     print(fcounter, lcounter)
32     for cps in CODEPAGES: # close all files
33         ff[cps].close()
34     fcounter += 1 # it is a new file
35     if fcounter > TOTALFILES: # it is ok of files
36         print("Breaking...")
37         break
38 finally:
39     for cps in CODEPAGES:
40         ff[cps].close()
41
42 print(f"End... Total set of files {fcounter}")
```

APÊNDICE D

DESAFIO

Esse exercício está dimensionado para ser feito em Python com a biblioteca NLTK. Você pode escolher outra linguagem por sua conta e risco.

O exercício usará a base CysticFibrosis2, disponível na rede. Você deve fazer um sistema de recuperação da informação dividido em módulos especificados a seguir. O sistema poderá funcionar totalmente na memória, usando arquivos para comunicação entre módulos.

Os módulos devem, em geral, seguir o princípio de processamento em batch:

1. Ler todos os dados
2. Fazer todo o processamento
3. Salvar todos os dados

Alguns desses módulos podem ser reusados ou refeitos em exercícios posteriores. Todos os módulos deve possuir um LOG que permitam pelo menos a um programa posterior, usando o módulo logging de Python:

- Identificar quando iniciaram suas operações
- Identificar quando iniciam cada parte de seu processamento
 - Ler arquivo de configuração
 - Ler arquivo de dados
- Identificar quantos dados foram lidos
- . Identificar quando terminaram os processamentos
- . Calcular os tempos médios de processamento de consultas, documento e palavras, de acordo com o programa sendo usado
- Identificar erros no processamento, caso aconteçam

O modelo do sistema a ser implementado é o seguinte:

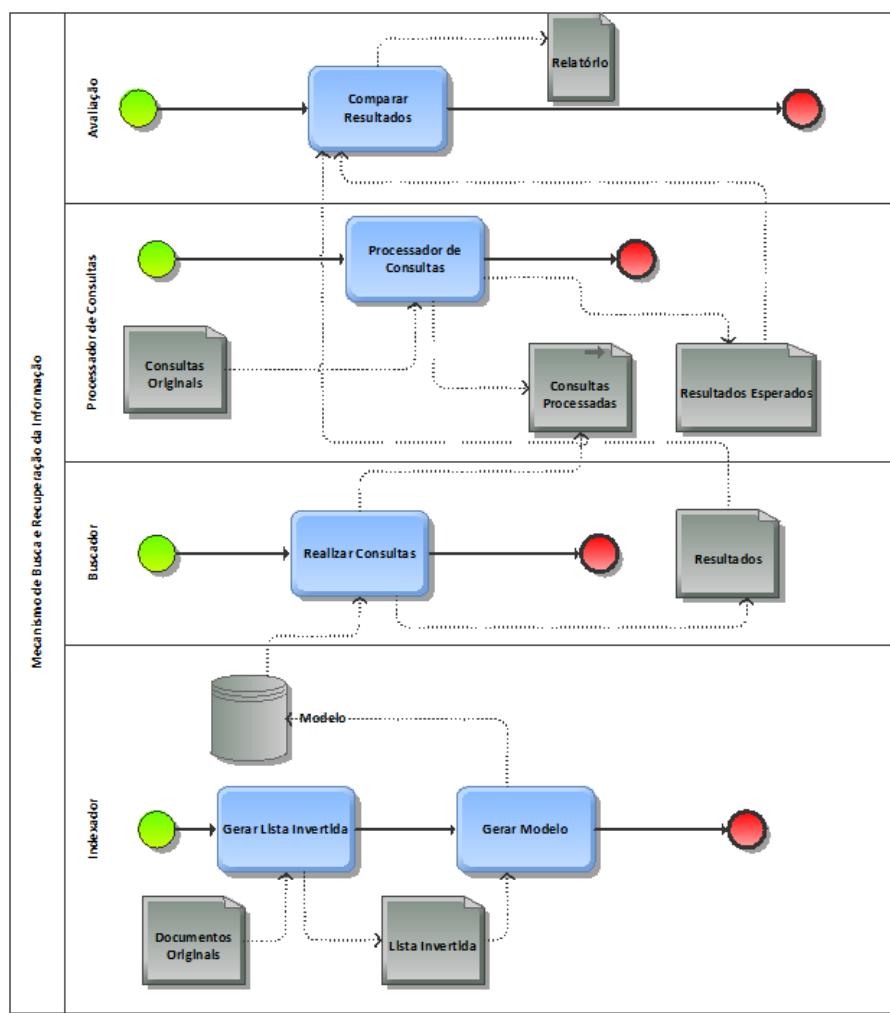


Figura D.1: Modelo de processo para o exercício.

D.1 Processador de Consultas

O objetivo desse módulo é transformar o arquivo de consultas fornecido ao padrão de palavras que estamos utilizando.

- O Processador de Consultas deverá ler um arquivo de configuração
 - O arquivo é criado por vocês
 - O nome do arquivo é PC.CFG
 - Ele contém dois tipos de instruções
 - ◊ LEIA=<nome de arquivo>
 - ◊ CONSULTAS=<nome de arquivo>
 - ◊ ESPERADOS=<nome de arquivo>
 - ◊ As instruções são obrigatórias, aparecem uma única vez e nessa ordem.
- O Processador de Consultas deverá ler um arquivo em formato XML
 - O arquivo a ser lido será indicado pela instrução LEIA no arquivo de configuração
 - ◊ O formato é descrito pelo arquivo cfc2-query.dtd.
 - ◊ O arquivo a ser lido é cfquery.xml.
 - O Processador de Consultas deverá gerar dois arquivos
 - Os arquivos deverão ser no formato cvs
 - ◊ O caractere de separação será o ;, ponto e vírgula
 - Todos os caracteres ; que aparecerem no arquivo original devem ser eliminados
 - ◊ A primeira linha do arquivo cvs deve ser o cabeçalho com o nome dos campos
 - O primeiro arquivo a ser gerado será indicado na instrução CONSULTAS do arquivo de configuração
 - ◊ Cada linha representará uma consulta
 - O primeiro campo de cada linha conterá o número da consulta [Campo QueryNumber]
 - O segundo campo de cada linha conterá uma consulta processada em letras maiúsculas, sem acento, pontuação, etc... [campo QueryText]
 - Cada aluno poderá escolher como criar sua consulta
 - O segundo arquivo a ser gerado será indicado na instrução ESPERADOS
 - ◊ Cada linha representará uma consulta
 - O primeiro campo de cada linha conterá o número da consulta [Campo QueryNumber]
 - O segundo campo conterá um documento [Campo DocNumber]
 - O terceiro campo conterá o número de votos do documento [Campo DocVotes]
 - Uma consulta poderá aparecer em várias linhas, pois podem possuir vários documentos como resposta
 - As linhas de uma consulta devem ser consecutivas no arquivo
 - Essas contas devem ser feitas a partir dos campos Records, Item e do atributo Score de Item
 - Considerar qualquer coisa diferente de zero como um voto

D.2 Gerador de Lista Invertida

A função desse módulo é criar as listas invertidas simples.

- O Gerador Lista Invertida deverá ler um arquivo de configuração

- O nome do arquivo é GLI.CFG
- Ele contém dois tipos de instruções
 - ◊ LEIA=<nome de arquivo>
 - ◊ ESCREVA=<nome de arquivo>
 - ◊ Podem ser uma ou mais instruções LEIA
 - ◊ Deve haver uma e apenas uma instrução ESCREVA
 - ◊ A instrução ESCREVA aparece depois de todas as instruções LEIA
- O Gerador Lista Invertida deverá ler um conjunto de arquivos em formato XML
 - Os arquivos a serem lidos serão indicados pela instrução LEIA no arquivo de configuração
 - O formato é descrito pelo arquivo cfc2.dtd.
 - O conjunto de arquivos será definido por um arquivo de configuração
 - Os arquivos a serem lidos são os fornecidos na coleção
- Só serão usados os campos RECORDNUM, que contém identificador do texto e ABSTRACT, que contém o texto a ser classificado
 - Atenção: Se o registro não contiver o campo ABSTRACT deverá ser usado o campo EXTRACT
- O Gerador Lista Invertida deverá gerar um arquivo
 - O arquivo a ser gerado será indicado na instrução ESCREVA do arquivo de configuração
 - O arquivo deverá ser no formato cvs
 - ◊ O caractere de separação será o ;, ponto e vírgula
 - Cada linha representará uma palavra
 - O primeiro campo de cada linha conterá a palavra em letras maiúsculas, sem acento
 - O segundo campo de cada linha apresentará uma lista (Python) de identificadores de documentos onde a palavra aparece
 - Se uma palavra aparece mais de uma vez em um documento, o número do documento aparecerá o mesmo número de vezes na lista
 - Exemplo de uma linha
 - ◊ FIBROSIS ; [1,2,2,3,4,5,10,15,21,21,21]

D.3 Indexador

A função desse módulo é criar o modelo vetorial, dadas as listas invertidas simples.

- O indexador será configurado por um arquivo INDEX.CFG
 - O arquivo conterá apenas uma linha LEIA, que terá o formato
 - ◊ LEIA=<nome de arquivo>
 - O arquivo conterá apenas uma linha ESCREVA, que terá o formato
 - ◊ ESCREVA=<nome de arquivo>
- O Indexador deverá implementar um indexador segundo o Modelo Vetorial
 - O Indexador deverá utilizar o tf/idf padrão
 - ◊ O tf pode ser normalizado como proposto na equação 2.1 do Cap. 2 do Modern Information Retrieval
 - O indexador deverá permitir a alteração dessa medida de maneira simples
 - O Indexador deverá possuir uma estrutura de memória deve de alguma forma representar a matriz termo documento
 - O Indexador deverá classificar toda uma base transformando as palavras apenas da seguinte forma:
 - ◊ Apenas palavras de 2 letras ou mais

- Apenas palavras com apenas letras
- Todas as letras convertidas para os caracteres ASCII de A até Z, ou seja, só letras maiúsculas e nenhum outro símbolo
- A base a ser indexada estará na instrução LEIA do arquivo de configuração
- O sistema deverá salvar toda essa estrutura do Modelo Vetorial para utilização posterior

D.4 Buscador

O objetivo desse módulo é obter os resultados de um conjunto de buscas em um modelo salvo.

- O Buscador deverá ler o arquivo de consultas e o arquivo do modelo vetorial e realizar cada consulta, escrevendo outro arquivo com a resposta encontrada para cada consulta.
- Para isso, usará o arquivo de configuração BUSCA.CFG, que possuirá duas instruções
 - MODELO=<nome de arquivo>
 - CONSULTAS=<nome de arquivo>
 - RESULTADOS=<nome de arquivo>
- A busca deverá ser feita usando modelo vetorial
- Cada palavra na consulta terá o peso 1
- O arquivo de resultados deverá
 - Ser no formato .csv
 - Separar os campos por ;, ponto e vírgula
 - Cada uma de suas linhas terá dois campos
 - O primeiro contendo o identificador da consulta
 - O segundo contendo uma lista Python de ternos ordenados
 - O primeiro elemento é a posição do documento no ranking
 - O segundo elemento é o número do documento
 - O terceiro elemento é a distância do elemento para a consulta

D.5 Avaliação

Você deve avaliar o seu sistema de recuperação de informação, atualizando-o para trabalhar com e sem o uso do stemmer de Porter, usando os arquivos RESULTADOS.CSV e RESULTADOS_ESPERADOS.CSV para obter as seguintes medidas e diagramas:

1. Gráfico de 11 pontos de precisão e recall
2. F1
3. Precision@5
4. Precision@10
5. Histograma de R-Precision (comparativo)
6. MAP
7. MRR
8. Discounted Cumulative Gain (médio)
9. Normalized Discounted Cumulative Gain

Se alguma decisão do limite para o rank for importante, use 10. Os resultados devem todos ser entregues em um arquivo RELATORIO.MD. Os diagramas também devem ser entregues (cada um) na forma de um arquivo CVS e de um ou mais arquivo gráfico (PDF ou outro), ambos no formato <tipo de

gráfico>-<dado usado>-<sequencial>.<formato do arquivo>. Por exemplo 11Pontos-nostemmer-1.csv ou 11Pontos-stemmer-2.pdf Segue o novo modelo:

D.6 Stemming

- Você deve incluir no arquivo de configuração dos programas que analisam diretamente o texto a possibilidade de usar ou não um stemmer
- A opção é uma linha no início com a palavra STEMMER ou NOSTEMMER
- Você deve usar o Stemmer de Porter disponível em <http://tartarus.org/martin/PorterStemmer/>
- A palavra STEMMER ou NOSTEMMER deve ser somada a palavra RESULTADOS para formar o nome do arquivo de resposta, resultando em RESULTADOS-STEMMER ou RESULTADOS-NOTEMMER.

D.7 Entrega

A entrega é composta de um arquivo .zip ou um repositório Git (GitHub) contendo:

- Todo o código fonte em um diretório SRC
- Um arquivo README.MD na raiz com qualquer instrução adicional para uso do código entregue
- Um arquivo MODELO.(DOC ou TXT) com a descrição do formato do modelo na raiz
- Em um diretório RESULT
 - Todos os arquivos criados por sua execução.
 - O arquivo RESULTADOS.csv
- Os resultados da avaliação devem ficar em um diretório AVALIA.

APÊNDICE E

JOGO DO SIGNIFICADO

E.1 Introdução

Um jogo para discutir o significado das palavras, baseado no “Telefone Sem Fio”¹ e também no “Cadavre Exquis”².

E.2 Componentes

1. Uma folha em branco numerada de 0 até o número de jogadores para cada jogador. N é o número de jogadores
2. lápis ou canetas
3. opcional: lista de palavras, ou cartas com as palavras.

E.3 Modo de jogar

1. Os jogadores sentam e se colocam em ordem, de 1 a N.
2. Cada jogador recebe uma folha numerada de 0 a N.
3. Cada jogador recebe, ou escolhe, uma palavra, que deve escrever na posição 0 da folha.
4. Cada jogador deve passar a página em para o próximo jogador, o último jogador passa para o primeiro.
5. Ao receber a página o jogador tem duas opções
 - Se ele recebe uma palavra, deve escrever uma definição na próxima posição livre da folha e copiá-la para um cartão em branco.
 - Se ele recebe uma definição, deve escrever uma palavra na próxima posição livre da folha e copiá-la para um cartão em branco.
6. O jogador agora deve dobrar (ou redobrar) a página de modo que o próximo jogador **só veja a última palavra ou a última definição, e não as anteriores**.
7. Se as posições da folha não estiverem completas, voltar ao passo 4.
8. Se as posições da folha estiverem completas o jogo terminou.

¹[https://pt.wikipedia.org/wiki/Telefone_sem_fio_\(brincadeira\)](https://pt.wikipedia.org/wiki/Telefone_sem_fio_(brincadeira))

²https://pt.wikipedia.org/wiki/Cad%C3%A1ver_esquisito

Sobre as definições:

- As definições devem ser **verdadeiras**
- Não pode ser usada a própria palavra ou variantes na definição.
- **A definição tem que ter no máximo 7 palavras.**

Deve ficar claro que a cada passo, cada jogador vai, de acordo com o ritmo do jogo, ou ler uma palavra e descrever uma definição, ou ler uma definição e escolher uma palavra, e passar o que escreveu para frente.

E.4 Conclusão

Ao terminar o jogo, se for um número ímpar de jogadores, a última linha de cada página conterá uma definição. Se for um número par, conterá uma palavra.

Cada folha mostra a evolução da palavra/significado pelos jogadores.

A turma deve discutir essa evolução.

REFERÊNCIAS BIBLIOGRÁFICAS

- Abbott, Edwin (1991). *Flatland: A Romance of Many Dimensions*. 2^a ed. Princeton University Press (ver p. 15).
- Abdul-Jaleel, Nasreen et al. (2004). “UMass at TREC 2004: Novelty and HARD”. *Computer Science Department Faculty Publication Series*, p. 189 (ver p. 203).
- Abreu, Rodrigo Mesquita de (set. de 2011). “Proposta de Arquitetura Para Um Sistema de Detecção de Plágio Multi-Algoritmo”. Diss. de mestr. Rio de Janeiro: Programa de Engenharia de Sistemas e Computação - COPPE/UFRJ (ver p. 345).
- Abu-Mostafa, Y. S., M. Magon-Ismail e H. T. Lin (2012). *Learning from Data: A Short Course*. AML Book (ver p. 295).
- Adobe Systems Inc. (1999). *PostScript Language Reference (3rd Ed.)* USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0201379228 (ver p. 62).
- Allen, Julie D. et al. (2007). *The Unicode Standard 5.0 Electronic edition*. Unicode Consortium (ver p. 28).
- Alvares, Reinaldo Viana, Ana Cristina Bicharra Garcia e Inhaúma Ferraz (2005). “STEMBR: A Stemming Algorithm for the Brazilian Portuguese Language”. Em: *EPIA 2005 - 12th Portuguese Conference on Artificial Intelligence*. Vol. Lecture Notes in Artificial Intelligence 3808. Covilhã, Portugal: Springer, pp. 693–701 (ver p. 86).
- Amri, Samir e Lahbib Zenkouar (ago. de 2018). “Stemming and Lemmatization for Information Retrieval Systems in Amazigh Language”. Em: *Third International Conference, BDCA 2018*. Vol. 872. Third International Conference, BDCA 2018, Kenitra, Morocco, April 45, 2018, Revised Selected Papers, pp. 222–233. ISBN: 978-3-319-96291-7. DOI: 10.1007/978-3-319-96292-4_18 (ver pp. xii, 83, 84).
- Anderson, Tiffani (25 de ago. de 2021). *Googles Page Experience Update: What It Is and What It Means for Your Website*. Bluehost. URL: <https://www.bluehost.com/blog/googles-page-experience-update-what-it-is-and-what-it-means-for-your-website> (acesso em 06/03/2023) (ver pp. 253, 254).
- Andrade, Frank (13 de mar. de 2021). *5 Simple Ways to Tokenize Text in Python. Tokenizing text, a large corpus and sentences of different language*. Towards Data Science. URL: <https://towardsdatascience.com/5-simple-ways-to-tokenize-text-in-python-92c6804edfc4> (acesso em 14/03/2022) (ver p. 81).
- Apache Software Foundation (2011a). *Welcome to Apache Lucene*. Versão 9.0.0. Apache Software Foundation. URL: <https://lucene.apache.org/> (acesso em 07/03/2023) (ver pp. 255, 270).
- (2011b). *Welcome to PyLucene*. Apache Software Foundation. URL: <https://lucene.apache.org/pylucene/> (acesso em 10/03/2022) (ver p. 258).

- Apache Software Foundation (2022a). *Apache Solr Reference Guide 9.0*. Apache Software Foundation. URL: <https://solr.apache.org/guide/solr/latest/> (acesso em 16/11/2022) (ver p. 261).
- (2022b). *Learn More About Solr*. Apache Software Foundation. URL: <https://solr.apache.org/> (acesso em 10/03/2022) (ver pp. 269, 270).
- (2022c). *Apache Nutch*. Apache Foundation. URL: <https://nutch.apache.org/> (acesso em 14/11/2022) (ver pp. 269, 270).
- Araújo, Luciana Kuchenbecker (2022). *O que é fonema?* Brasil Escola. URL: <https://brasilescola.uol.com.br/o-que-e/portugues/o-que-e-fonema.htm> (acesso em 19/03/2022) (ver p. 17).
- Atkins, S., J. Clear e N. Ostler (1 de jan. de 1992). “Corpus Design Criteria”. *Literary and Linguistic Computing* 7.1, pp. 1–16. DOI: 10.1093/linc/7.1.1 (ver pp. 6, 21).
- Axler, Sheldon (2015). *Linear Algebra Done Right*. Ed. por Sheldon Axler e Kenneth Ribet. 3^a ed. Undergraduate Texts in Mathematics. Springer (ver p. 357).
- Baeza-Yates, Ricardo e Berthier Ribeiro-Neto (1999). *Modern Information Retrieval*. USA: ACM Press (ver p. 228).
- (2011). *Modern Information Retrieval: The Concepts and Technology behind Search*. USA: Addison-Wesley Publishing Company (ver pp. xii, 138, 140, 143–145, 148, 159, 161, 173, 181–183, 191, 193, 195–198, 201, 202, 224, 228, 336, 343).
- Bahl, Lalit R., Frederick Jelinek e Robert L. Mercer (1983). “A Maximum Likelihood Approach to Continuous Speech Recognition”. *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-5.2*, pp. 179–190. DOI: 10.1109/TPAMI.1983.4767370 (ver p. 218).
- Baltar, Maria Elizabeth e Carneiro de Albuquerque (2018). *Instrumentos de Representação Descritiva da Informação*. Curso de Bacharelado em Biblioteconomia na Modalidade a Distância. Rio de Janeiro: Departamento de Biblioteconomia, FACC/ UFRJ (ver p. 128).
- Barabási, Albert-László (2002). *Linked: The New Science of Networks*. Perseus Publishing (ver p. 238).
- Bateman, Scott, Carl Gutwin e Miguel Nacenta (2008). “Seeing Things in the Clouds: The Effect of Visual Features on Tag Cloud Selections”. Em: *Proceedings of the Nineteenth ACM Conference on Hypertext and Hypermedia*. HT '08. Pittsburgh, PA, USA: Association for Computing Machinery, pp. 193–202 (ver pp. 327, 328).
- Beghtol, C (1986). “Bibliographic Classification Theory And Text Linguistics: Aboutness Analysis, Intertextuality And The Cognitive Act Of Classifying Documents”. *Journal of Documentation* 42.2, pp. 84–113. DOI: <https://doi.org/10.1108/eb026788> (ver p. 132).
- Belkin, N. J., Oddy R. N. e H. M. Brooks (jun. de 1982). “ASK for Information Retrieval Part I. Background and Theory”. *Journal of Documentation* 38.2 (ver pp. 131, 137).
- Bengio, Yoshua et al. (mar. de 2003). “A Neural Probabilistic Language Model”. *The Journal of Machine Learning Research* 3.null, pp. 1137–1155. ISSN: 1532-4435 (ver pp. xiii, xiv, 167, 218, 219, 311, 312).
- Bentivogli, Luisa et al. (ago. de 2004). “Revising WordNet Domains Hierarchy: Semantics, Coverage, and Balancing”. Em: *COLING 2004 Workshop on Multilingual Linguistic Resources*. Geneva, Switzerland, pp. 101–108 (ver p. 281).
- Berners-Lee, T., R. Fielding e L Masinter (jan. de 2005). *Uniform Resource Identifier: Generic Syntax, RFC 3986*. Rel. técn. (ver p. 331).
- Berners-Lee, Tim (1989). *Information Management: A Proposal*. CERN. URL: <http://www.w3.org/History/1989/proposal.html> (acesso em 08/02/2023) (ver p. 237).
- Berry, Michael W. e Murray Browne (2005). *Understanding Search Engines: Mathematical Modeling and Text Retrieval*. 2^a ed. Philadelphia: SIAM (ver pp. 241, 247).
- Betts, O. e R. Bouton (24 de mar. de 2022). *Snowball Stemming language and algorithms*. URL: <https://github.com/snowballstem/snowball> (acesso em 24/03/2022) (ver p. 85).

- Bielenberg, K. e M. Zacher (2006). "Groups in Social Software: Utilizing Tagging to Integrate Individual Contexts for Social Navigation". Diss. de mestr. Bremen (ver p. 328).
- Bird, Steven, Edward Loper e Ewan Klein (2009). *Natural Language Processing with Python*. Sebastopol, California: O'Reilly Media Inc (ver pp. xi, 18, 82, 85, 337).
- Blei, David M., Andrew Y. Ng e Michael I. Jordan (mar. de 2003). "Latent Dirichlet Allocation". *J. Mach. Learn. Res.* 3.null, pp. 993–1022. ISSN: 1532-4435 (ver p. 337).
- Blough, D. S. (2001). "The perception of similarity". Em: *Avian visual cognition*. [On-line]. Available: <http://www.pigeon.psy.tufts.edu/avc/dblough>, last checked in 30 Aug. Department of Psychology, Tufts University. Cap. The perception of similarity (ver pp. 115, 116).
- Boechat, Pedro e Pedro Kuchpil (ago. de 2022). "Análise da evolução dos títulos de pesquisas em computação da UFRJ por meio de Diferencial Tag Clouds". Em: *Explorações em Mineração de Texto*. Ed. por Geraldo Xexéo. ES-782/22. Relatório Técnico, pp. 26–33 (ver pp. xiv, 338, 341).
- Bojanowski, Piotr et al. (2016). "Enriching Word Vectors with Subword Information". *Transactions of the Association for Computational Linguistics* 5, pp. 135–146 (ver pp. 175, 218).
- Borlund, Pia (jan. de 2000). "Experimental components for the evaluation of interactive information retrieval systems". *Journal of Documentation* 56.1, pp. 71–90. ISSN: 0022-0418. DOI: 10.1108/EUM0000000007110. URL: <https://doi.org/10.1108/EUM0000000007110> (ver pp. 133, 134).
- Brasil (7 de dez. de 1940). *Decreto Lei N° 2.848, de 7 de dezembro de 1940*. Brasília, DF: Brasil. URL: http://www.planalto.gov.br/ccivil_03/Decreto-Lei/DeL2848compilado.htm (acesso em 06/11/2022) (ver p. 346).
- (19 de fev. de 1998). *Lei N° 9.610, de 19 de Fevereiro de 1998*. Brasília, DF: Brasil. URL: http://www.planalto.gov.br/ccivil_03/leis/19610.htm (acesso em 06/11/2022) (ver pp. 345, 346).
- Briet, S (1951). *Quest-ce que la documentation*. Paris (ver pp. 6, 7).
- Brin, Sergey e Lawrence Page (1998). "The Anatomy of a Large-Scale Hypertextual Web Search Engine". *Computer Networks and ISDN Systems* 30 (1-7), pp. 107–117. DOI: [https://doi.org/10.1016/S0169-7552\(98\)00110-X](https://doi.org/10.1016/S0169-7552(98)00110-X). URL: <http://infolab.stanford.edu/~backrub/google.html> (ver pp. xiii, 139, 250, 251, 268).
- Brousentsov, N. P. et al. (2021). *Development of ternary computers at Moscow State University*. URL: <https://www.computer-museum.ru/english/setun.htm> (acesso em 25/12/2021) (ver p. 29).
- Bruza, P. D. e T. W. C. Huibers (1994). "Investigating Aboutness Axioms Using Information Fields". Em: *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '94. Dublin, Ireland: Springer-Verlag, pp. 112–121. ISBN: 038719889X (ver p. 133).
- Bruza, Peter e Theo Huibers (out. de 1996). "A study of aboutness in information retrieval". *Artificial Intelligence Review* 10, pp. 381–407. DOI: 10.1007/BF00130692 (ver pp. 132, 133).
- Bruza, Peter, Dawei Song e Kam-Fai Wong (1999). "Fundamental Properties of Aboutness (Poster Abstract)". Em: *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '99. Berkeley, California, USA: Association for Computing Machinery, pp. 277–278. ISBN: 1581130961. DOI: 10.1145/312624.312696. URL: <https://doi.org/10.1145/312624.312696> (ver pp. 132, 133).
- Buckland, Michael K. (1997). "What Is a Document?"? *Journal Of The American Society For Information Science* 48.9. John Wiley & Sons, Inc., pp. 804–809 (ver p. 6).
- Bush, Vannevar (jul. de 1945). "As We May Think". *The Atlantic* (176), pp. 101–108. URL: <https://www.theatlantic.com/magazine/archive/1945/07/as-we-may-think/303881/> (acesso em 06/02/2023) (ver pp. 129, 236).
- Cafarella, Mike e Doug Cutting (abr. de 2004). "Building Nutch: Open Source Search: A Case Study in Writing an Open Source Search Engine". *ACM Queue* 2.2, pp. 54–61. ISSN: 1542-7730. DOI: 10.1145/988392.988408. URL: <https://doi.org/10.1145/988392.988408> (ver p. 270).

- Castillo, Carlos (nov. de 2004). “Effective Web Crawling”. Tese de dout. Dept. of Computer Science - University of Chile (ver pp. xiii, 263, 264, 266, 269, 270).
- Chakrabarti, Soumen (2009). “Focused Web Crawling”. Em: *Encyclopedia of Database Systems*. Ed. por Ling Liu e M. Tamer Özsu. Boston, MA: Springer US, pp. 1147–1155 (ver pp. 265, 269).
- (2018). “Focused Web Crawling”. Em: ed. por Ling Liu e M. Tamer Özsu. Boston, MA: Springer US, pp. 1493–1500 (ver p. 268).
- Chang, Zixiang (2022). “A Survey of Modern Crawler Methods”. Em: *The 6th International Conference on Control Engineering and Artificial Intelligence*. CCEAI 2022. Virtual Event, Japan: Association for Computing Machinery, pp. 21–28. ISBN: 9781450385916. DOI: 10.1145/3522749.3523076. URL: <https://doi.org/10.1145/3522749.3523076> (ver pp. 265, 276).
- Chaput, Matt (2012). *Whoosh 2.7.4 documentation*. URL: <https://whoosh.readthedocs.io/en/latest/> (acesso em 17/11/2022) (ver p. 259).
- Cherkassky, V. e F. Mulier (2007). *Learning from Data: Concepts, Theory and Methods*. 2^a ed. John Wiley & Sons (ver pp. xiv, 294, 295).
- Cho, Junghoo, Hector Garcia-Molina e Lawrence Page (abr. de 1998). “Efficient Crawling through URL Ordering”. *Comput. Netw. ISDN Syst.* 30.17, pp. 161–172. ISSN: 0169-7552 (ver pp. 264, 268).
- Chomsky, Noam (2002). *Syntactic Structures*. 2^a ed. de Gruyter Mouton (ver p. 12).
- Cleverdon, C. W. (jan. de 1960). “The Aslib Cranfield Research Project on the Comparative Efficiency of Indexing Systems”. *Aslib Proceedings* 12.12, pp. 421–431. ISSN: 0001-253X. DOI: 10.1108/eb049778. URL: <https://doi.org/10.1108/eb049778> (ver pp. 147, 148).
- (jan. de 1967). “The Cranfield Tests on Index Language Devices”. *Aslib Proceedings* 19.6, pp. 173–194. ISSN: 0001-253X. DOI: 10.1108/eb050097. URL: <https://doi.org/10.1108/eb050097> (ver p. 147).
- Clinchant, Stéphane e Florent Perronnin (2013). “Aggregating Continuous Word Embeddings for Information Retrieval”. Em: *Proceedings of the Workshop on Continuous Vector Space Models and their Compositionality (CVSC)*. Association for Computational Linguistics, pp. 100–109 (ver p. 175).
- Cohen, Mike X. (2022). *Practical Linear Algebra for Data Science: From Core Concepts to Applications Using Python*. Sebastopol, CA: O'Reilly Media (ver pp. xiii, 209).
- Constantin, A. et al. (2016). “The Document Components Ontology (DoCO)”. *Semantic Web* 7.2, pp. 167–181. DOI: <http://dx.doi.org/10.3233/SW-150177> (ver p. 20).
- Control Data Corporation (1975). *Control Data Cyber 170 Computer Systems: Hardware Reference Manual*. St. Paul, Minnesota (ver p. 29).
- Conversion (13 de mai. de 2019). *Medic Update: como essa atualização do Google impactou sites e seus conteúdos*. Conversion. URL: <https://www.conversion.com.br/blog/medic-update/> (acesso em 06/03/2023) (ver p. 253).
- Coombs, James H., Allen H. Renear e Steven J. DeRose (nov. de 1987). “Markup Systems and the Future of Scholarly Text Processing”. *Commun. ACM* 30.11, pp. 933–947. ISSN: 0001-0782. DOI: 10.1145/32206.32209. URL: <https://doi.org/10.1145/32206.32209> (ver p. 55).
- Cooper, William S. (1997). “Getting Beyond Boole”. Em: *Readings in Information Retrieval*. Ed. por Karen Sparck Jones e Peter Willett. Ed. por Edward Fox. The Morgan Kaufman Series in Multimedia Infomation and Systems. San Francisco: Morgan Kaufman Publishers (ver p. 182).
- Cormen, Thomas H. et al. (2009). *Introdução aos Algoritmos*. 3^a ed. Elsevier (ver pp. 110, 111).
- Cross, Valerie V. e Thomas A. Sudkamp (2002). *Similarity and Compatibility in Fuzzy Set Theory: Assessment and Applications*. Heidelberg, Germany, Germany: Physica-Verlag GmbH. ISBN: 3-7908-1458-X (ver p. 119).
- Crystal, David (2019). *The Cambridge Encyclopedia of The English Language*. 3^a ed. New York: Cambridge University Press (ver pp. 27, 28).

- Cutts, Matt (19 de jan. de 2012). *Page layout algorithm improvement*. Google Search Central Blog. (Acesso em 06/03/2023) (ver p. 253).
- (13 de jun. de 2013). *What should we expect in the next few months in terms of SEO for Google?* Video. Google Search Central. URL: <https://www.youtube.com/watch?v=xQmQeKU25zg#t=2m30s> (acesso em 06/03/2023) (ver p. 253).
- da Silva, B.C. Dias (s.d.). “Brazilian Portuguese Wordnet: A Computational Linguistic Exercise of Encoding Bilingual Relational Lexicons”. *International Journal of Computational Linguistics and Applications* 1.1–2 (), pp. 137–150 (ver p. 280).
- Daniels, Peter T. (2003). “Writing Systems”. Em: *The Handbook of Linguistics*. Ed. por Mark Aronoff e Janie Rees-Miller. Oxford: Blackwell. Cap. 3 (ver p. 12).
- Date, C. J. (2004). *An introduction to database systems*. Eighth. Boston, MA, USA: Pearson/Addison Wesley, pp. xxvii + 983 + 22 (ver p. 330).
- DCMI (2022a). *About DCMI*. URL: <https://www.dublincore.org/about/> (acesso em 04/01/2022) (ver p. 20).
- (2022b). *DCMI Metadata Terms*. URL: <https://www.dublincore.org/about/> (acesso em 04/01/2022) (ver p. 21).
- (2022c). *Dublin Core*. URL: <https://www.dublincore.org/about/> (acesso em 04/01/2022) (ver pp. 20, 21).
- Deshmukh, Smita e Kantilal Vishwakarma (2021). “A Survey on Crawlers used in developing Search Engine”. Em: *2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)*, pp. 1446–1452. DOI: [10.1109/ICICCS51141.2021.9432368](https://doi.org/10.1109/ICICCS51141.2021.9432368) (ver p. 265).
- Devlin, Jacob et al. (2018a). “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. *arXiv preprint arXiv:1810.04805* (ver pp. 170, 176, 309).
- (2018b). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. DOI: [10.48550/ARXIV.1810.04805](https://doi.org/10.48550/ARXIV.1810.04805). URL: <https://arxiv.org/abs/1810.04805> (acesso em 07/02/2023) (ver pp. 220, 233, 254).
- Devlin, K. (1993). *The Joy of Sets: Fundamentals of Contemporary Set Theory*. 2^a ed. Undergraduate Texts in Mathematics. Berlin: Springer New York (ver p. 330).
- Dias, Carlos Cardoso e Tales Mello Paiva (2022). “Análise Semântica Latente (LSA) Aplicada a Projetos de Lei”. Em: *Explorações em Mineração de Texto*. Ed. por Geraldo Xexéo. ES-782/22. Relatório Técnico, pp. 17–24 (ver pp. 324, 338).
- Diringer, David (out. de 1953). *The Alphabet: A Key to the History of Mankind*. 2^a ed. London: Hutchinson’s (ver p. 83).
- Document Academy, The (2022). *What is a document?* URL: <http://documentacademy.org/?what-is-a-document> (acesso em 04/01/2022) (ver pp. 6, 7).
- Domingos, Pedro (2015). *The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World*. Basic Books (ver p. 291).
- Doval, Yerai, Jesús Vilares e Carlos Gómez-Rodríguez (2020). “Towards Robust Word Embeddings for Noisy Texts”. *Applied Sciences* 10.19, p. 6893 (ver pp. 178, 221).
- Duarte, Fellipe Ribeiro (jun. de 2017). “Identificando Plagio Externo Com Locality-Sensitive tHashing”. Tese de dout. Rio de Janeiro: Programa de Engenharia de Sistemas e Computação - COPPE/UFRJ (ver pp. 345, 346).
- Egler, Tamara Tania Cohen, Thiago Costa e Pedro Paulo Gonçalves (1 de out. de 2021). “A (In)Visibilidade Da Rede Tecnopolítica Bolsonarista”. *AR@Cne. Revista Electrónica de Recursos en Internet Sobre Geografía Y Ciencias Sociales* XXV.251 (ver p. 339).
- Emspak, Jesse (29 de dez. de 2016). “How a Machine Learns Prejudice. Artificial intelligence picks up bias from human creators not from hard, cold logic”. *Scientific American*. URL: <https://www.scientificamerican.com/article/how-a-machine-learns-prejudice/>

- www.scientificamerican.com/article/how-a-machine-learns-prejudice/ (acesso em 06/04/2022) (ver p. 294).
- Estrella, Carlos (28 de fev. de 2023). *Cumulative Layout Shift (CLS): O Que é e Como Melhorá-lo.* Hostinger Tutorials. URL: <https://www.hostinger.com.br/tutoriais/cumulative-layout-shift> (ver p. 253).
- Fairthorne, R. A (1969). “Content analysis, specification and control”. *Annual Review of Information Science and Technology* 4, pp. 73–109 (ver p. 132).
- Fellbaum, Christiane (1998). *WordNet: An Electronic Lexical Database*. Cambridge, MA: MIT Press (ver pp. 200, 279, 337, 342).
- Ferman, Fabio (2016). “Programação Genética de Árvores de Regras para Normalização de Textos”. Diss. de mestr. Rio de Janeiro: Programa de Engenharia de Sistemas e Computação - COPPE/UFRJ (ver p. 102).
- Foltz, P. W., W. Kintsch e T. K. Landauer (1998). “The measurement of textual coherence with Latent Semantic Analysis”. *Discourse Processes* 25.2-3, pp. 285–307. URL: <http://lsa.colorado.edu/papers/dp2.foltz.pdf> (acesso em 25/04/2023) (ver p. 217).
- Fox, Edward A. (ago. de 1983). “Extending the Boolean and Vector Space Models of Information Retrieval with P-Norm Queries and Multiple Concept Types”. Tese de dout. Ithaca, NY: Cornell Univ. (ver p. 191).
- Fox, Edward A. e Sharat Sharan (1986). *A Comparison of Two Methods for Soft Boolean Operator Interpretation in Information Retrieval*. Technical Report TR-86-01. Virginia Tech. URL: <http://hdl.handle.net/10919/20272> (acesso em 20/11/2022) (ver p. 191).
- Frakes, William B. e Ricardo Baeza-Yates (30 de jun. de 1992). *Information Retrieval: Data Structures & Algorithms*. Prentice-Hall. 512 pp. ISBN: 0134638379 (ver pp. xii, 83, 84).
- Freed, N. e N. Borenstein (nov. de 1996). *RFC-2045 - Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*. URL: <https://www.rfc-editor.org/rfc/rfc2045> (acesso em 30/12/2021) (ver p. 48).
- Freitas, Cláudia (2022). *Linguística Computacional*. Vol. 13. Linguística para o Ensino Superior. São Paulo: Parábola (ver pp. 81, 82, 103, 279, 280).
- Fuhr, N. (1992). “Probabilistic Models in Information Retrieval”. *The Computer Journal* 35.3, pp. 243–255 (ver pp. xii, 138).
- Furnas, G. W. et al. (1988). “Information Retrieval Using a Singular Value Decomposition Model of Latent Semantic Structure”. Em: *Proceedings of the 11th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '88. Grenoble, France: Association for Computing Machinery, pp. 465–480. ISBN: 2706103094. DOI: 10.1145/62437.62487. URL: <https://doi.org/10.1145/62437.62487> (ver p. 199).
- Galvez, Carmen, Félix de MoyaAnegón e Víctor H. Solana (1 de jan. de 2005). “Term conflation methods in information retrieval”. *Journal of Documentation* 61.4, pp. 520–547. DOI: 10.1108/00220410510607507 (ver pp. xii, 83, 84).
- Gao, Luyu et al. (2022). “Tevatron: An Efficient and Flexible Toolkit for Dense Retrieval”. *ArXiv abs/2203.05765* (ver p. 262).
- Giesbers, Bas, Ellen Rusman e Jan Bruggen (mai. de 2006). *State of the Art LSA*. Cooper project Derivable 3.1. Open University of the Netherlands. URL: https://www.researchgate.net/publication/277222150_State_of_the_Art_LSA (acesso em 25/04/2023) (ver p. 217).
- Gingras, Richard (12 de set. de 2019). *Elevating original reporting in Search*. Google. URL: <https://blog.google/products/search/original-reporting/> (acesso em 07/03/2023) (ver p. 254).
- Glen, Stephanie (2023). *Criterion Variable: Definition, Use and Examples*. StatisticsHowTo.com: Elementary Statistics for the rest of us! URL: <https://www.statisticshowto.com/criterion-variable-2/> (acesso em 04/01/2023) (ver p. 223).

- Goffman, William (1969). "An Indirect Method of Information Retrieval". *Information Storage and Retrieval* 4, pp. 361–373 (ver p. 134).
- Goldberg, David E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. 1st. USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0201157675 (ver p. 101).
- Golub, Gene H. e Charles F. Van Loan (2013). *Matrix Computations*. 4^a ed. Baltimore, Maryland: The Johns Hopkins University Press (ver p. 209).
- Goodman, Joshua (2001). "A Bit of Progress in Language Modeling". *Computer Speech & Language* 15.4, pp. 403–434 (ver p. 218).
- Google (6 de mai. de 2011). *Mais orientações para criar sites de alta qualidade*. URL: <https://developers.google.com/search/blog/2011/05/more-guidance-on-building-high-quality> (acesso em 05/03/2023) (ver p. 254).
- (9 de jun. de 2020). *Como o Google combateu o spam na Pesquisa – Relatório de spam na Web 2019*. URL: <https://developers.google.com/search/blog/2020/06/how-we-fought-search-spam-on-google> (acesso em 05/03/2023) (ver pp. 253, 254).
 - (2022a). *Central da Pesquisa Google: Visão geral dos rastreadores do Google (user agents)*. Google. URL: <https://developers.google.com/search/docs/crawling-indexing/overview-google-crawlers> (acesso em 12/11/2022) (ver p. 267).
 - (4 de abr. de 2022b). *Como o Google combateu o spam na Pesquisa em 2021*. URL: <https://developers.google.com/search/blog/2022/04/webspam-report-2021> (acesso em 05/03/2023) (ver pp. 253, 254).
 - (30 de nov. de 2022c). *Um guia sobre os sistemas de classificação da Pesquisa Google*. URL: <https://developers.google.com/search/docs/appearance/ranking-systems-guide> (acesso em 07/03/2023) (ver p. 254).
 - (2023a). *Googles crisis alerts provide access to trusted safety information across Search, Maps, and Android*. URL: <https://crisisresponse.google/forecasting-and-alerts> (acesso em 07/03/2023) (ver p. 254).
 - (2 de mar. de 2023b). *Guia detalhado de como a Pesquisa Google funciona*. URL: <https://developers.google.com/search/docs/fundamentals/how-search-works> (acesso em 06/03/2023) (ver pp. 251, 253).
 - (2023c). *How Google search results differ from ads*. Google Ads Help. URL: <https://support.google.com/google-ads/answer/1722080> (acesso em 07/03/2023) (ver p. 249).
 - (2023d). *How Google's Knowledge Graph works*. URL: <https://support.google.com/knowledgepanel/answer/9787176> (acesso em 05/03/2023) (ver p. 253).
 - (2023e). *Sistema de avaliações do produto da Pesquisa Google e seu site*. URL: <https://developers.google.com/search/updates/product-reviews-update> (acesso em 07/03/2023) (ver p. 254).
 - (5 de fev. de 2023f). *Sistema de conteúdo útil da Pesquisa Google e seu site*. URL: <https://developers.google.com/search/updates/helpful-content-update> (acesso em 07/03/2023) (ver p. 254).
 - (2022d). *Saiba mais sobre sitemaps*. URL: <https://developers.google.com/search/docs/crawling-indexing/sitemaps/overview> (acesso em 15/11/2022) (ver p. 268).
- Gorrell, Genevieve e B. Webb (2005). "Generalized Hebbian Algorithm for Incremental Singular Value Decomposition in Natural Language Processing". Em: *INTERSPEECH 2005 - Eurospeech, Ninth European Conference on Speech Communication and Technology, ISCA 2005*. Lisboa, Portugal (ver p. 210).
- Gottfredson, Linda S. (1994). "Mainstream Science on Intelligence: An Editorial with 52 Signatories, History, and Bibliography". *The Wall Street Journal* (ver p. 307).
- Grimes, Carrie (8 de jun. de 2010). *Our new search index: Caffeine*. Google. URL: <https://googleblog.blogspot.com/2010/06/our-new-search-index-caffeine.html> (ver p. 252).

- Grossman, David A. e Ophir Frieder (2004). *Information Retrieval: Algorithms and Heuristics*. Netherlands: Springer (ver p. 211).
- Guedes, Emanuel Guedson Ferreira (2009). “O Conceito Aboutness na Organização e Representação do Conhecimento”. Diss. de mestr. Programa de Pós-Graduação em Ciência da Informação da Faculdade de Filosofia e Ciências da Universidade Estadual Paulista UNESP. URL: https://www.marilia.unesp.br/Home/Pos-Graduacao/CienciadaInformacao/Dissertacoes/guedes_egf_me_mar.pdf (acesso em 21/03/2022) (ver p. 132).
- Gyöngyi, Zoltán, Hector Garcia-Molina e Jan Pedersen (2004). “Combating Web Spam with Trustrank”. Em: *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30. VLDB '04*. Toronto, Canada: VLDB Endowment, pp. 576–587. ISBN: 0120884690 (ver p. 246).
- Hacohen-Kerner, Yaakov, Daniel Miller e Yair Yigal (1 de mai. de 2020). “The influence of preprocessing on text classification using a bag-of-words representation”. *PLOS ONE* 15.5, e0232525. DOI: [10.1371/journal.pone.0232525](https://doi.org/10.1371/journal.pone.0232525) (ver p. 99).
- Haliday, M. A. K. e Ruqaiya Hasan (1976). *Cohesion in English*. London: Longman (ver pp. 5, 132).
- Hammer, Joachin e Jan Fiedler (nov. de 2000). “Using Mobile Crawlers to Search the Web Efficiently”. *ACIS Int. J Comp. Inf. Sci.* 1.1, pp. 36–58. ISSN: 1525-9293 (ver p. 266).
- Haralambous, Yannis (2007). *Fonts & Encodings*. Trad. por P. Scott Horne. O'Reilly Media (ver pp. 28, 31, 32).
- Harman, Donna (2019). “Information Retrieval: The Early Years”. *Foundations and Trends in Information Retrieval* 13.5, pp. 425–577. DOI: [10.1561/1500000065](https://doi.org/10.1561/1500000065) (ver p. 182).
- Harmouch, Mahmoud (13 de mar. de 2021). *17 types of similarity and dissimilarity measures used in data science*. URL: <https://towardsdatascience.com/17-types-of-similarity-and-dissimilarity-measures-used-in-data-science-3eb914d2681> (acesso em 02/03/2023) (ver p. 120).
- Hassan-Montero, Y. e V. Herrero-Solana (2006). “Improving Tag-Clouds as Visual Information Retrieval Interfaces”. Em: *Proceedings of the 1st International Conference on Multidisciplinary Information Sciences and Technologies InSCiT* (ver p. 328).
- Hearst, Marti A. et al. (2020). “An Evaluation of Semantically Grouped Word Cloud Designs”. *IEEE Transactions on Visualization and Computer Graphics* 26.9, pp. 2748–2761 (ver p. 337).
- Henzinger, Monika R. (jan. de 2001). “Hyperlink Analysis for the Web”. *IEEE Internet Computing* (ver pp. 237, 241).
- Hochreiter, Sepp e Jürgen Schmidhuber (1997). “Long Short-Term Memory”. *Neural Computation* 9.8, pp. 1735–1780 (ver pp. 167, 308).
- Honnibal, Matthew e Ines Montani (2017). “spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing”. To appear (ver p. 81).
- Hutchins, W. J. (mai. de 1978). “The concept of ‘aboutness’ in subject indexing”. *Aslib Proceedings* 30.5, pp. 172–181 (ver p. 132).
- IBM (2010). *z/OS Basic Skills: The EBCDIC character set - Application programming on z/OS*. URL: <https://www.ibm.com/docs/en/zos-basic-skills?topic=mainframe-ebcdic-character-set> (acesso em 27/12/2021) (ver p. 31).
- IETF (30 de dez. de 2021). *Internet Standards: RFCs*. Internet Engineering Task Force. (Acesso em 30/12/2021) (ver p. 57).
- IFLA (fev. de 2009). *Functional Requirements For Bibliographic Records. Final Report*. Rel. técn. International Federation of Library Associations e Institutions. URL: <https://repository.ifla.org/bitstream/123456789/811/2/ifla-functional-requirements-for-bibliographic-records-frbr.pdf> (acesso em 04/01/2022) (ver pp. xi, 19, 20).

- Illyes, Gary (23 de set. de 2016). *O Penguin agora faz parte do nosso algoritmo principal*. Blog da Central da Pesquisa Google. URL: <https://developers.google.com/search/blog/2016/09/penguin-is-now-part-of-our-core> (acesso em 06/03/2023) (ver p. 253).
- International Phonetic Association (1999). *Handbook of the International Phonetic Association : A Guide to the Use of the International Phonetic Alphabet*. Cambridge University Press (ver p. 17).
- Järvelin, Kalervo e Jaana Kekäläinen (2000). “IR Evaluation Methods for Retrieving Highly Relevant Documents”. Em: *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '00. Athens, Greece: Association for Computing Machinery, pp. 41–48. ISBN: 1581132263. DOI: 10.1145/345508.345545. URL: <https://doi.org/10.1145/345508.345545> (ver p. 158).
- (out. de 2002). “Cumulated Gain-Based Evaluation of IR Techniques”. *ACM Trans. Inf. Syst.* 20.4, pp. 422–446. ISSN: 1046-8188. DOI: 10.1145/582415.582418. URL: <https://doi.org/10.1145/582415.582418> (ver p. 159).
- Ji, Shihao et al. (2016). “WordRank: Learning Word Embeddings via Robust Ranking”. Em: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, pp. 658–668 (ver p. 179).
- Jo, Jaemin, Bongshin Lee e Jinwook Seo (2015). “WordlePlus: Expanding Wordle’s Use through Natural Interaction and Animation”. *IEEE Computer Graphics and Applications* 35.6, pp. 20–28. DOI: 10.1109/MCG.2015.113 (ver p. 337).
- Johnson, Lyndon B. (11 de mar. de 1968). *Memorandum Approving the Adoption by the Federal Government of a Standard Code for Information Interchange*. Ed. por Gerhard Peters e John T. Woolley. The American Presidency Project. URL: <https://www.presidency.ucsb.edu/node/237376> (acesso em 26/12/2021) (ver p. 30).
- Jurafsky, Dan e James H. Martin (12 de jan. de 2022). *Speech and Language Processing*. draft. URL: <https://web.stanford.edu/~jurafsky/slp3/> (acesso em 16/03/2024) (ver pp. 163, 173, 295, 336, 342).
- Jurafsky, Daniel e James H. Martin (2008). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. 2^a ed. Upper Saddle River, NJ: Pearson Prentice Hall (ver p. 9).
- Kahneman, Daniel (2003). “Les Prix Nobel. The Nobel Prizes 2002”. Em: ed. por Tore Frängsmyr. Stockholm: Nobel Foundation. Cap. Maps of Bounded Rationality: A Perspective on Intuitive Judgment and Choice. URL: http://nobelprize.org/%20nobel%5C_prizes/economics/laureates/%202002/kahnemann-lecture.pdf (ver p. 116).
- Kent, Allen et al. (abr. de 1955). “Machine literature searching VIII. Operational criteria for designing information retrieval systems”. *American Documentation* 6.2, pp. 93–101. DOI: 10.1002/asi.5090060209. URL: <https://ideas.repec.org/a/bla/amedoc/v6y1955i2p93-101.html> (ver pp. 147, 148).
- Kiros, Ryan et al. (2015). “Skip-thought vectors”. Em: *Advances in Neural Information Processing Systems*, pp. 3294–3302 (ver p. 221).
- Kleinberg, Jon M. (set. de 1999). “Authoritative Sources in a Hyperlinked Environment”. *Journal of the ACM* 46.5, pp. 604–632. ISSN: 0004-5411. DOI: 10.1145/324133.324140. URL: <https://doi.org/10.1145/324133.324140> (ver pp. 139, 236–238, 241).
- Klensin, J. (out. de 2008). *RFC-5321 - Simple Mail Transfer Protocol*. URL: <https://www.rfc-editor.org/rfc/rfc5321.html> (ver p. 48).
- Knuth, Donald E. (1997). *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0201896842 (ver p. 29).
- Kontostathis, April e William M. Pottenger (2006). “A framework for understanding Latent Semantic Indexing (LSI) performance”. *Information Processing & Management* 42.1. Formal Methods for

- Information Retrieval, pp. 56–73. ISSN: 0306-4573. DOI: <https://doi.org/10.1016/j.ipm.2004.11.007>. URL: <https://www.sciencedirect.com/science/article/pii/S0306457304001529> (ver p. 210).
- Koster, M. (25 de fev. de 1994). *Important: Spiders, Robots and Web Wanderers*. e-mail message 4113 www-talk - via wayback machine. URL: <https://web.archive.org/web/20131029200350/http://inkdroid.org/tmp/www-talk/4113.html> (acesso em 12/11/2022) (ver p. 266).
- Koster, M. et al. (set. de 2022). *RFC 9309 Robots Exclusion Protocol*. Internet Engineering Task Force (IETF). URL: <https://www.rfc-editor.org/rfc/rfc9309.pdf> (acesso em 12/11/2022).
- Krizhevsky, Alex, Ilya Sutskever e Geoffrey E. Hinton (2012). “ImageNet Classification with Deep Convolutional Neural Networks”. Em: *Advances in Neural Information Processing Systems*, pp. 1097–1105 (ver p. 308).
- Kui, Xiaoyan et al. (2022). “A survey of visual analytics techniques for online education”. *Visual Informatics* (ver p. 339).
- Kumar, Ankit, Piyush Makhija e Anuj Gupta (2020). “Noisy Text Data: Achilles Heel of BERT”. Em: *Proceedings of the 2020 EMNLP Workshop W-NUT: The Sixth Workshop on Noisy User-generated Text*, pp. 16–21 (ver p. 99).
- Kumar, Manish, Rajesh Bhatia e Dhavleesh Rattan (2017). “A survey of Web crawlers for information retrieval”. *WIREs Data Mining and Knowledge Discovery* 7.6, e1218. DOI: <https://doi.org/10.1002/widm.1218>. eprint: <https://wires.onlinelibrary.wiley.com/doi/pdf/10.1002/widm.1218>. URL: <https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/widm.1218> (ver pp. 264, 266).
- Kumar, R. (2009). “Web Page Quality Metrics”. Em: *Encyclopedia of Database Systems*. Ed. por Ling Liu e M. Tamer Özsu. Boston, MA: Springer US (ver p. 266).
- Landauer, T. K., P. W. Foltz e D Laham (1988). “An introduction to latent semantic analysis”. *Discourse Processes* 25.2–3, pp. 259–284. DOI: 10.1080/01638539809545028 (ver p. 210).
- Landauer, Thomas K. (s.d.). “LSA as a Theory of Meaning”. Em: cap. 1 (ver p. 208).
- Landauer, Thomas K. e Susan Dumais (2008). *Latent semantic analysis*. Scholarpedia. URL: http://www.scholarpedia.org/article/Latent_semantic_analysis (acesso em 18/02/2023) (ver p. 210).
- Landauer, Thomas K., Danielle S. McNamara et al. (fev. de 2007). *Handbook of Latent Semantic Analysis*. 1^a ed. New York: Routledge (ver pp. 162, 174, 337, 338).
- Langville, Amy N. e Carl D. Meyer (2006). *Google’s Pagerank and Beyond: the science of search engine rankings*. Princeton: Princeton University Press (ver pp. 241, 243).
- Lavrenko, Victor e W. Bruce Croft (2001). “Relevance Based Language Models”. Em: *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’01. New Orleans, Louisiana, USA: Association for Computing Machinery, pp. 120–127. ISBN: 1581133316. DOI: 10.1145/383952.383972. URL: <https://doi.org/10.1145/383952.383972> (ver p. 202).
- Le, Quoc V. e Tomas Mikolov (2014). *Distributed Representations of Sentences and Documents*. arXiv: 1405.4053 [cs.CL]. URL: <https://arxiv.org/abs/1405.4053> (ver pp. 175, 221).
- Lee, John A. e Michel Verleysen (2007). *Nonlinear Dimensionality Reduction*. New York: Springer (ver pp. xiii, 205–207).
- Leibson, Steven (1 de dez. de 2021). “Which Was The First Microprocessor? Intel 4004, AiResearch MP944, or Four-Phase AL1?” *Electronic Engineering Journal*. URL: <https://www.eejournal.com/article/which-was-the-first-microprocessor/> (acesso em 09/03/2022) (ver p. 29).

- Li, Chenlu, Xiaoju Dong e Xiaoru Yuan (2018). “Metro-Wordle: An Interactive Visualization for Urban Text Distributions Based on Wordle”. *Visual Informatics* 2.1. Proceedings of PacificVAST 2018, pp. 50–59. ISSN: 2468-502X (ver p. 340).
- Library of Congress, The (13 de mar. de 2020). *MARC Standards*. URL: <https://www.loc.gov/marc/> (acesso em 04/01/2022) (ver p. 21).
- (2022). *BIBFRAME Ontology*. Versão 2.1.0. URL: <https://id.loc.gov/ontologies/bibframe.html> (acesso em 04/01/2022) (ver p. 21).
- Lin, Jimmy et al. (2021). “Pyserini: A Python Toolkit for Reproducible Information Retrieval Research with Sparse and Dense Representations”. Em: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’21. Virtual Event, Canada: Association for Computing Machinery, pp. 2356–2362. ISBN: 9781450380379. DOI: 10.1145/3404835.3463238. URL: <https://doi.org/10.1145/3404835.3463238> (ver p. 261).
- Liu, B. (2020). *Sentiment Analysis: Mining Opinions, Sentiments, and Emotions*. 2^a ed. Studies in Natural Language Processing. Cambridge University Press. ISBN: 9781108486378 (ver p. 301).
- Liu, Ling e M. Tamer Özsu, ed. (2009). *Encyclopedia of Database Systems*. Boston, MA: Springer US. DOI: 10.1007/978-0-387-39940-9.
- Liu, Perry (23 de mar. de 2022). *More helpful product reviews on Search*. Google. URL: <https://blog.google/products/search/more-helpful-product-reviews/> (acesso em 07/03/2023) (ver p. 254).
- Liu, Yuting et al. (2008). “BrowseRank: Letting Web Users Vote for Page Importance”. Em: *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’08. Singapore, Singapore: Association for Computing Machinery, pp. 451–458. ISBN: 9781605581644. DOI: 10.1145/1390334.1390412. URL: <https://doi.org/10.1145/1390334.1390412> (ver pp. xix, 245, 246).
- Lomas, Natasha (2020). *Googles latest user-hostile design change makes ads and search results look identical*. TechCrunch. URL: <https://techcrunch.com/2020/01/23/squint-and-youll-click-it/> (acesso em 07/03/2023) (ver p. 249).
- Lopes, M. C. S. (2004). “Mineração de Dados Textuais Utilizando Técnicas de Clustering Para o Idioma Português”. Tese de dout. Rio de Janeiro: Programa de Engenharia Civil - COPPE/UFRJ (ver p. 86).
- Loria, Steven (2018). “textblob Documentation”. *Release 0.15.2* (ver p. 338).
- Louro, A. Tavares (2007). *O sentido denotativo e conotativo de provérbios in Ciberdúvidas da Língua Portuguesa*. Ciberdúvidas da Língua Portuguesa. URL: <https://ciberduvidas.iscte-iul.pt/consultorio/perguntas/o-sentido-denotativo-e-conotativo-de-proverbios/20662> (acesso em 01/07/2020) (ver p. 13).
- Lovins, Julie Beth (mar.–jun. de 1968). “Development of a Stemming Algorithm”. *Mechanical Translation and Computational Linguistics* 11.1-2. URL: <https://aclanthology.org/www.mt-archive.info/MT-1968-Lovins.pdf> (acesso em 23/03/2022) (ver pp. 83, 84).
- Löwis, Martin v. (2010). *Flexible String Representation*. PEP 393. URL: <https://www.python.org/dev/peps/pep-0393/> (ver p. 47).
- Lucca, J. L. De e Maria das Graças Volpe Nunes (nov. de 2002). *Lematização versus Stemming*. Rel. técn. NILC-TR-02-22. o Núcleo Interinstitucional de Lingüística Computacional (ver pp. 80, 81).
- Luhn, H.P. (abr. de 1958). “The Automatic Creation of Literature Abstracts”. *IBM Journal of Research and Development* 2 (2), pp. 159–165. DOI: <http://dx.doi.org/10.1147/rd.22.0159> (ver pp. 196, 197).
- MacAvaney, Sean et al. (2021). “Simplified Data Wrangling with ir_datasets”. Em: *SIGIR* (ver p. 287).

- Macbeth, Danielle (2011). "Language, Natural and Symbolic". Em: *The Cambridge Encyclopedia of the Language Sciences*. Ed. por Patrick Colm Hogan. New York: Cambridge University Press, pp. 409–410 (ver p. 27).
- Macdonald, Craig et al. (2012). "From puppy to maturity: Experiences in developing Terrier". *Proc. of OSIR at SIGIR*, pp. 60–63 (ver p. 261).
- Mackenzie, Charles E. (1980). *Coded Character Sets, History, and Development*. Addison-Wesley (ver p. 30).
- Mahabal, Abhijit A. et al. (17 de set. de 2013). "Synonym identification based on co-occurring terms". US8538984B1. Google Inc. URL: <https://patentimages.storage.googleapis.com/87/00/fe/4a26e2c9ad69c7/US8538984.pdf> (ver p. 253).
- Manning, Christopher D., Prabhakar Raghavan e Hinrich Schütze (1 de abr. de 2009a). *An Introduction to Information Retrieval*. Cambridge UP: Cambridge University Press. URL: <https://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf> (ver pp. xiii, 81, 92, 108, 109, 158, 185, 199, 201, 231, 232).
- (1 de abr. de 2009b). "Probabilistic information retrieval". Em: *An Introduction to Information Retrieval*. Cambridge UP: Cambridge University Press, pp. 201–217 (ver pp. 165, 231, 336).
- Marinchev, I. (2006). "Practical Semantic Web - Tagging and Tag Clouds". *Journal Cybernetics and Information Technologies* 6.3, pp. 33–39 (ver pp. 327, 332).
- Maron, M. E. e J. L. Kuhns (jul. de 1960). "On Relevance, Probabilistic Indexing and Information Retrieval". *Journal of the ACM* 7.3, pp. 216–244. ISSN: 0004-5411. DOI: 10.1145/321033.321035. URL: <https://doi.org/10.1145/321033.321035> (ver pp. 200, 223).
- Martin, B (1994). "Plagiarism: a misplaced emphasis". *Journal of Information Ethics* 3.2 (Fall 1994). 36–47. URL: <https://www.uow.edu.au/~bmartin/pubs/94jie.html> (ver p. 346).
- Marvin, Ginny (22 de fev. de 2016). *FAQ: All About The Changes To Googles Ad Layout On Desktop Search Results. A look at what is changing and what is not on Google desktop search results*. Search Engine Land. URL: <https://searchengineland.com/243057-243057> (acesso em 07/03/2023) (ver p. 249).
- Mattos, Guilherme C. de (ago. de 2022). "GTAGLIB: a Library to Generate Tags and Tag Clouds". Em: *Explorações em Mineração de Texto*. Ed. por Geraldo Xexéo. ES-782/22. Relatório Técnico, pp. 35–48 (ver p. 342).
- Maurer, Hermann, Frank Kappe e Bilal Zaka (2006). "Plagiarism - A Survey". *Journal of Universal Computer Science* 12.8, pp. 1050–1084 (ver p. 346).
- Mayworm, Marcelo de Mattos (28 de set. de 2007). "Um Crawler Peer-to-Peer Baseado em Agentes". Diss. de mestr. Rio de Janeiro: Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ (ver pp. 264, 269, 336).
- McCulloch, Warren S. e Walter Pitts (1943). "A Logical Calculus of the Ideas Immanent in Nervous Activity". *Bulletin of Mathematical Biophysics* 5.4, pp. 115–133 (ver p. 307).
- Meyriat, Jean (1981). "Document, documentation, documentologie". *Schéma et Schématisation* 14, pp. 51–63 (ver pp. 7, 132).
- Mikolov, Tomas et al. (2013). "Efficient Estimation of Word Representations in Vector Space". Em: *Proceedings of the International Conference on Learning Representations (ICLR)*. arXiv. DOI: 10.48550/ARXIV.1301.3781. URL: <https://arxiv.org/abs/1301.3781> (acesso em 17/02/2023) (ver pp. 162, 163, 174, 218, 219, 311).
- Miller, George A. (1995). "WordNet: A Lexical Database for English". *Communications of ACM* 38.11, pp. 39–41 (ver pp. 200, 279).
- Miranda, Otávio (2023). *Normalização Unicode em Python*. URL: <https://www.otaviomiranda.com.br/2020/normalizacao-unicode-em-python/> (acesso em 21/11/2023) (ver p. 34).

- Mitchell, Herbert (jan. de 1953). “The use of the Univ AC FAC-tronic system in the library reference field”. *American Documentation* 4.1, pp. 16–17. DOI: 10.1002/asi.5090040105 (ver p. 182).
- Mitchell, Tom M. (1997). *Machine learning, International Edition*. McGraw-Hill Series in Computer Science. McGraw-Hill. ISBN: 978-0-07-042807-2. URL: <https://www.worldcat.org/oclc/61321007> (ver p. 291).
- Mizzaro, Stefano (jun. de 1998). “How many relevances in information retrieval?” *Interacting with Computers* 10.3, pp. 303–320. DOI: 10.1016/s0953-5438(98)00012-5 (ver pp. 131, 136, 137).
- Momin, Khondhaker, H M Imran Kays e Arif Mohaimin Sadri (out. de 2022). *Identifying Crisis Response Communities in Online Social Networks for Compound Disasters: The Case of Hurricane Laura and Covid-19*. URL: <https://arxiv.org/abs/2210.14970> (acesso em 04/11/2022) (ver p. 340).
- Montti, Roger (12 de abr. de 2022). *Googles Hummingbird Update: How It Changed Search*. Search Engine Journal - SEO, Search Marketing News e Tutorials. URL: <https://www.searchenginejournal.com/google-algorithm-history/hummingbird-update/> (acesso em 05/03/2023) (ver p. 253).
- Morgado, Fernando Fernandes (29 de set. de 2010). “Representação de Documentos Através de Nuvens de Termos”. Diss. de mestr. Rio de Janeiro: Programa de Engenharia de Sistemas e Computação - COPPE/UFRJ (ver pp. xiv, 323, 325, 326, 328, 329, 331, 337, 338).
- MOZ (2021). *Google Possum*. What is Google Possum? MOZ. URL: <https://moz.com/learn/seo/google-possum> (acesso em 06/03/2023) (ver p. 253).
- (2023). *Google Panda*. URL: <https://moz.com/learn/seo/google-panda> (acesso em 05/03/2023) (ver p. 253).
- Mozilla (2021). *JavaScript*. URL: <https://developer.mozilla.org/en-US/docs/Web/JavaScript> (acesso em 30/12/2021) (ver p. 56).
- Al-Msie'deen, Ra'Fat (2019). “Tag Clouds for Object-Oriented Source Code Visualization”. *Engineering, Technology & Applied Science Research* 9.3, pp. 4243–4248 (ver p. 339).
- Mueller, Andreas (2020). *WordCloud for Python documentation*. URL: https://amueller.github.io/word_cloud/index.html (acesso em 03/11/2022) (ver pp. 337, 338).
- Muzdakis, Madeleine (30 de set. de 2020). *Uppercase and Lowercase: An Etymological Tale of Two Scripts*. My Modern Met. URL: <https://mymodernmet.com/uppercase-lowercase-letters> (acesso em 05/02/2023) (ver p. 83).
- Nagel, Sebastian (18 de nov. de 2014). *Web Crawling with Apache Nutch*. Presentation. URL: https://www.slideshare.net/sebastian_nagel/aceu2014-snagelwebcrawlingnutch (acesso em 16/11/2022) (ver p. 270).
- Najork, Marc (s.d.). “Web Crawler Architecture”. Em: *Encyclopedia of Database Systems*. Ed. por Ling Liu e M. Tamer Ozsu. Boston, MA: Springer US, pp. 4608–4611 (ver p. 263).
- Nayak, Pandu (25 de out. de 2019). *Understanding searches better than ever before*. Google. URL: <https://blog.google/products/search/search-language-understanding-bert/> (ver p. 252).
- (18 de mai. de 2021). *MUM: A new AI milestone for understanding information*. Google. URL: <https://blog.google/products/search/introducing-mum/> (ver p. 253).
- (11 de ago. de 2022). *New ways we're helping you find high-quality information*. Google. URL: <https://blog.google/products/search/information-literacy> (acesso em 07/03/2023) (ver p. 254).
- Nayak, Panduk (3 de fev. de 2022). *How AI powers great search results*. Google (The Keyword). URL: <https://blog.google/products/search/how-ai-powers-great-search-results/> (acesso em 06/03/2023) (ver p. 254).
- Neisser, Ulric et al. (1996). “Intelligence: Knowns and Unknowns”. *American Psychologist* 51.2, pp. 77–101. DOI: 10.1037/0003-066x.51.2.77 (ver p. 307).

- Nelson, Theodor Holm (1987). *Literary machines: The report on, and of, Project Xanadu concerning word processing, electronic publishing, hypertext, thinkertoys, tomorrow's intellectual revolution and certain other topics including knowledge, education and freedom*. Mindful Press (ver p. 237).
- (2 de out. de 1997). *Embedded Markup Considered Harmful*. URL: <https://www.xml.com/pub/a/w3j/s3.nelson.html> (acesso em 24/12/2021) (ver p. 55).
- Neto, Nelson Fordelone (11 de jul. de 2017). *Uma Homenagem Aos 73 Anos De Chico Buarque. Chico Buarque E A Revolução Dos Cravos*. Esquerda Diário. URL: <https://www.esquerdadiario.com.br/Chico-Buarque-e-a-Revolucao-dos-Cravos> (acesso em 29/03/2022) (ver pp. 13, 14).
- Ng, Andrew e Michael Jordan (2001). “On Discriminative vs. Generative Classifiers: A comparison of logistic regression and naive Bayes”. Em: *Advances in Neural Information Processing Systems*. Ed. por T. Dietterich, S. Becker e Z. Ghahramani. Vol. 14. MIT Press. URL: <https://proceedings.neurips.cc/paper/2001/file/7b7a53e239400a13bd6be6c91c4f6c4e-Paper.pdf> (ver pp. 295, 296).
- Nguyen, Eric (2013). “Text Mining and Network Analysis of Digital Libraries in R”. Em: Zhao, Yanchang. *Data Mining Applications with R*. Elsevier Science Publishing Co Inc. ISBN: 9780124115118 (ver p. 142).
- Nielsen, Jakob (23 de mar. de 2009). *Tag Cloud Examples*. Nielsen Normam Group. URL: <https://www.nngroup.com/articles/tag-cloud-examples/> (acesso em 02/11/2022) (ver pp. 325, 326).
- NISO (20 de fev. de 2013). *The Dublin Core Metadata Element Set*. NISO Standards ANSI/NISO Z39.85-2021. NISO - National Information Standards Organization. ISBN: 978-1-937522-14-8. URL: https://groups.niso.org/apps/group_public/download.php/10258/Z39-85-2012_dublin_core.pdf (acesso em 04/01/2022) (ver pp. 20, 21).
- Norman, Jeremy (4 de fev. de 2023). *The Earliest Surviving Detailed Bibliographical Entries*. HistoryofInformation.com. URL: <https://historyofinformation.com/detail.php?id=2794> (acesso em 06/02/2023) (ver p. 128).
- Norvig, Peter (2007). *How to Write a Spelling Corrector*. URL: <https://norvig.com/spell-correct.html%20consultada%20em%202024/7/2020> (acesso em 19/01/2022) (ver pp. xvii, 100, 336).
- (2009). “Natural Language Corpus Data”. Em: *Beautiful Data: The Stories Behind Elegant Data Solutions*. Ed. por Tom Segaran e Jeff Hammerbacher, pp. 219–242 (ver p. 336).
- Nothman, Joel, Hanmin Qin e Roman Yurchak (20 de jul. de 2018). “Stop Word Lists in Free Open-source Software Packages”. Em: *Proceedings of Workshop for NLP Open Source Software*. Association for Computational Linguistics. Melbourne, Australia, pp. 7–12 (ver pp. 92, 93).
- O'Reilly, Tim et al. (1992). *The Whole Internet User's Guide & Catalog*. O'Reilly Media, Inc. (ver p. 236).
- Oberstein, Mordy (2022). *The State of Search 2022*. Tech Report. SEMRUSH. URL: <https://static.semrush.com/blog/uploads/files/dd/4e/dd4ed0f40b1a50a8959db188176bc3d0/the-state-of-search-2022.pdf> (acesso em 08/03/2023) (ver p. 200).
- Ombre Blanches (10 de mai. de 2012). *Cloud on Title*. URL: <https://ombresblanches.wordpress.com/2012/05/10/cloud-on-title/> (acesso em 30/10/2022) (ver pp. xiv, 324).
- Ounis, I., G. Amati, V. Plachouras et al. (2006). “Terrier: A High Performance and Scalable Information Retrieval Platform”. Em: *Proceedings of ACM SIGIR'06 Workshop on Open Source Information Retrieval (OSIR 2006)*. Seattle, Washington, USA (ver p. 261).
- Ounis, I., G. Amati, Plachouras V. et al. (2005). “Terrier Information Retrieval Platform”. Em: *Proceedings of the 27th European Conference on IR Research (ECIR 2005)*. Vol. 3408. Lecture Notes in Computer Science. Springer, pp. 517–519. ISBN: 3-540-25295-9 (ver p. 261).
- Ounis, I., C. Lioma et al. (2007). “Research Directions in Terrier”. *Novatica/UPGRADE Special Issue on Web Information Access*. Ed. por Ricardo Baeza-Yates et al. (ver p. 261).

- Pagani, Luiz Arthur (2022). *Diagramas em árvore como representação da estrutura sintática (Slides do curso UFPR - HL396 Língua Portuguesa IV)*. URL: https://docs.ufpr.br/~arthur/textos/apr/sintaxe/arv_apr.pdf (acesso em 06/01/2022) (ver pp. xi, 11).
- Page, Lawrence (4 de set. de 2001). "Method for Node Ranking in a Linked Database". US 6,285,999 B1. The Board of Trustees of the Leland Stanford Junior University (ver p. 254).
- Page, Lawrence et al. (nov. de 1999). *The PageRank Citation Ranking: Bringing Order to the Web*. Technical Report 1999-66. Previous number = SIDL-WP-1999-0120. Stanford InfoLab. URL: <http://ilpubs.stanford.edu:8090/422/> (ver pp. 191, 236, 243, 250, 254).
- Paice, Chris D. (nov. de 1990). "Another Stemmer". *SIGIR Forum* 24.3, pp. 56–61. ISSN: 0163-5840. DOI: 10.1145/101306.101310. URL: <https://doi.org/10.1145/101306.101310> (ver pp. 85, 86).
- Paiva, Bruno D. de e João P. Leite Pinho (ago. de 2022). "Most Relevant Rooms in an AirbnbAccommodation: A QuantitativeAnalyses on Reviews". Em: *Explorações em Mineração de Texto*. Ed. por Geraldo Xexéo. ES-782/22. Relatório Técnico, pp. 5–16 (ver pp. xiv, 324, 338–340).
- Paiva, Valeria de, Alexandre Rademaker e Gerard de Melo (dez. de 2012). "OpenWordNet-PT: An Open Brazilian Wordnet for Reasoning". Em: *Proceedings of COLING 2012: Demonstration Papers*. Published also as Techreport <http://hdl.handle.net/10438/10274>. Mumbai, India: The COLING 2012 Organizing Committee, pp. 353–360. URL: <http://www.aclweb.org/anthology/C12-3044> (ver pp. 280, 337).
- Patel, Neil (2023). *O Guia do Iniciante ao Knowledge Graph do Google*. URL: <https://neilpatel.com.br/blog/o-guia-do-iniciante-ao-knowledge-graph-do-google/> (acesso em 06/03/2023) (ver p. 253).
- Pedregosa, F. et al. (2011). "Scikit-learn: Machine Learning in Python". *Journal of Machine Learning Research* 12, pp. 2825–2830 (ver pp. 81, 296).
- Pennebaker, James (2011). *The Secret Life of Pronouns. What Our Words Say About Us*. New York: Bloomsbury Press (ver p. 292).
- Pennington, Jeffrey, Richard Socher e Christopher D Manning (2014). "GloVe: Global Vectors for Word Representation". Em: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, pp. 1532–1543 (ver pp. 175, 218).
- Peters, Matthew E et al. (2018). "Deep contextualized word representations". Em: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. Association for Computational Linguistics, pp. 2227–2237 (ver pp. 176, 220).
- Pilgrim, Mark, Dan Blanchard e Ian Cordasco (2015). *chardet Docs Usage*. URL: <https://chardet.readthedocs.io/en/latest/usage.html> (acesso em 27/12/2021) (ver p. 39).
- Poli, Riccardo, William B. Langdon e Nicholas Freitag McPhee (2008). *A field guide to genetic programming*. With contributions by J. R. Koza. Lulu.com. URL: http://www0.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/poli08_fieldguide.pdf (acesso em 14/03/2023) (ver pp. xix, 102).
- Porter, M. F. (jul. de 1980). "An Algorithm for Suffix Stripping". *Program* 14.3, pp. 130–137. URL: <https://tartarus.org/martin/PorterStemmer/def.txt> (acesso em 22/03/2022) (ver pp. 80, 83, 85, 336, 342).
- (out. de 2001). *Snowball: A language for stemming algorithms*. URL: <https://snowballstem.org/texts/introduction.html> (acesso em 24/03/2022) (ver p. 85).
- Postel, Jonathan B. (ago. de 1982). *RFC-821 - Simple Mail Transfer Protocol*. URL: <https://www.rfc-editor.org/rfc/rfc821> (acesso em 30/12/2021) (ver pp. 47, 48).
- Princeton University (2010). *WordNet: A Lexical Database for English*. URL: <https://wordnet.princeton.edu/> (acesso em 09/04/2022) (ver pp. 200, 279).

- Raber, Douglas (2003). *The Problem Of Information: An Introduction to Information Science*. Lanham, MD: Scarecrow Press (ver p. 134).
- Radecki, Tadeusz (1979). "Fuzzy set theoretical approach to document retrieval". *Information Processing & Management* 15.5, pp. 247–259. ISSN: 0306-4573. DOI: [https://doi.org/10.1016/0306-4573\(79\)90031-1](https://doi.org/10.1016/0306-4573(79)90031-1). URL: <https://www.sciencedirect.com/science/article/pii/0306457379900311> (ver p. 191).
- Raffel, Colin et al. (2019). *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. DOI: 10.48550/ARXIV.1910.10683. URL: <https://arxiv.org/abs/1910.10683> (ver p. 253).
- Raghavan, Prabhakar (15 de out. de 2020). *How AI is powering a more helpful Google*. Google. URL: <https://www.blog.google/products/search/search-on/> (acesso em 07/03/2023) (ver p. 254).
- Rapp, Reinhard (2002). "The Computation of Word Associations: Comparing Syntagmatic and Paradigmatic Approaches". Em: *COLING 2002: The 19th International Conference on Computational Linguistics*. URL: <https://aclanthology.org/C02-1007> (ver p. 200).
- Rehurek, Radim e Petr Sojka (22 de mai. de 2010). "Software Framework for Topic Modelling with Large Corpora". English. Em: *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. <http://is.muni.cz/publication/884893/en>. Valletta, Malta: ELRA, pp. 45–50 (ver p. 81).
- Remington Rand (1953). *UNIVAC fac-tronic systems as used by The Army Map Service* (ver p. 182).
- Richardson, Leonard (2007). "Beautiful soup documentation". *April* (ver pp. 276, 336).
- Riva, Pat, Patrick Le Buf e Maja umer (ago. de 2017). *IFLA Library Reference Model: A Conceptual Model for Bibliographic Information*. URL: <https://www.ifla.org/wp-content/uploads/2019/05/assets/cataloguing/frbr-lrm/ifla-lrm-august-2017.pdf> (acesso em 04/01/2022) (ver p. 19).
- Rivadeneira, A. W. et al. (2007). "Getting our head in the clouds: toward evaluation of tagclouds". Em: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. CHI'07*. New York, NY: ACM (ver p. 327).
- Robertson, S. E. (1977). "The Probability Ranking Principle". *Journal of Documentation* 33.4, pp. 294–304 (ver pp. 193, 223, 224).
- Robertson, S. E. e K. Spärk-Jones (1976). "Relevance Weighting of Search Terms". *Journal of the American Society for Information Science* 27.3 (May-June), pp. 129–146 (ver p. 228).
- Robertson, Stephen e Hugo Zaragoza (2009). "The Probabilistic Relevance Framework: BM25 and Beyond". *Foundations and Trends® in Information Retrieval* 3.4, pp. 333–389. DOI: 10.1561/1500000019 (ver pp. 223, 229).
- Rocchio, Joseph (1965). "Relevance Feedback in Information Retrieval". Em: Salton, Gerard. *Information Storage and Retrieval: Scientific Report No. ISR-9*. No. ISR-9 (ver p. 201).
- Roelleke, Thomas (26 de jul. de 2013). *Information Retrieval Models: Foundations and Relationships*. Vol. 5. Synthesis Lectures on Information Concepts, Retrieval, and Services 3. Morgan & Claypool Publishers LLC, pp. 1–163. DOI: 10.2200/s00494ed1v01y201304icr027 (ver pp. 196–198).
- Rogers, H. (2005). *Writing Systems: A Linguistic Approach*. Blackwell Textbooks in Linguistics. Wiley (ver p. 12).
- Rosenblatt, Frank (1958). "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain". *Psychological Review* 65.6, pp. 386–408 (ver p. 307).
- Rosenthal, Leonard (2013). *Developing with PDF*. Sebastopol, California: O'Reilly Media, Inc. (ver p. 63).
- Rossum, Guido van e Fred L. Drake (2009). *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace. ISBN: 1441412697 (ver p. 37).

- Roy, Dwaipayan, Sumit Bhatia e Mandar Mitra (2019). "Selecting Discriminative Terms for Relevance Model". Em: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR'19. Paris, France: Association for Computing Machinery, pp. 1253–1256. ISBN: 9781450361729 (ver p. 203).
- Roy, Dwaipayan, Debasis Ganguly et al. (2018). "Using Word Embeddings for Information Retrieval: How Collection and Term Normalization Choices Affect Performance". Em: *Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM)*. ACM, pp. 741–750 (ver p. 179).
- Russell, Stuart e Peter Norvig (2009). *Artificial Intelligence: A Modern Approach*. 3rd. Upper Saddle River, NJ, USA: Prentice Hall (ver p. 307).
- Ryerson, James (out. de 2002). "The man who wasn't there". *Boston Globe*. URL: http://www.j-bradford-delong.net/%20movable%5C_type/archives/001025.html (ver p. 116).
- Salton, Gerard (1965). *Information Storage and Retrieval: Scientific Report No. ISR-9*. No. ISR-9 (ver p. 162).
- (1971a). "Relevance Feedback and the Optimization of Retrieval Effectiveness". Em: *The SMART Retrieval System. Experiments in Automatic Document Processing*. Ed. por Gerard Salton. Englewood Cliffs, New Jersey: Prentice-Hall. Cap. 15, pp. 324–336 (ver p. 201).
 - ed. (1971b). *The SMART Retrieval System. Experiments in Automatic Document Processing*. Englewood Cliffs, New Jersey: Prentice-Hall (ver p. 218).
 - (1989). *Automatic Text Processing The Transformation, Analysis, and Retrieval of Information by Computer*. Reading, Massachusetts: Addison-Wesley (ver p. 183).
- Salton, Gerard e Michael J. McGill (1983). *Introduction to Modern Information Retrieval*. New York: McGraw-Hill (ver pp. xii, xv, 93, 94, 153, 161, 173, 181, 191, 193, 196).
- Samuel, A. L. (1959). "Some Studies in Machine Learning Using the Game of Checkers". *IBM Journal of Research and Development* 3.3, pp. 210–219 (ver p. 291).
- Sanderson, Mark e W. Bruce Croft (2012). "The History of Information Retrieval Research". *Proceedings of the IEEE* 100.Special Centennial Issue, pp. 1444–1451. DOI: 10.1109/JPROC.2012.2189916 (ver p. 182).
- Saracevic, Tefko (2007). "Relevance: A Review of and a Framework for the Thinging on the Notion in Information Science: Nature and Manifestations of Relevance". *Journal of the American Society for Information Science and Technology* 58.13 (November-December), pp. 1915–1933 (ver p. 135).
- (2017). *The Notion of Relevance in Information Science Everybody knows what relevance is. But what is it really?* Synthesis Lectures On Information Concepts, Retrieval, And Services 50. Morgan & Claypool. DOI: 10.2200/S00723ED1V01Y201607ICR050 (ver pp. xii, 133–136, 223).
- Seltzer, Richard, Eric J. Ray e Deborah S. Ray (1997). *The AltaVista Search Revolution: How to find anything on the Internet*. McGraw-Hill (ver p. 250).
- Shannon, Claude (1948). *The Mathematical Theory of Communication*. Urbana, IL: University of Illinois Press (ver p. 165).
- Shinyama, Yusuke, Philippe Guglielmetti e Pieter Marsman (2019). *pdfminer.six*. URL: <https://pdfminersix.readthedocs.io/en/latest/> (acesso em 04/11/2022) (ver pp. 63, 336).
- Shotton, David e Silvio Peroni (3 de jul. de 2015). *DoCO, the Document Components Ontology*. URL: <https://sparontologies.github.io/doco/current/doco.html> (acesso em 04/01/2022) (ver pp. xi, 20).
- Silva, Rafael Leonardo Siqueira da (2007). "Modelo de Sinais para Busca e Recuperação de Informação Textual". Diss. de mestr. Programa de Engenharia de Sistemas e Computação - COPPE/UFRJ (ver p. 220).

- Simões, A. e X. Guinovart (2014). "Bootstrapping a Portuguese Wordnet from Galician, Spanish and English wordnets". Em: *Advances in Speech and Language Technologies for Iberian Languages*. Vol. 8854. Springer, pp. 239–248 (ver p. 280).
- Sinclair, James e Michael Cardew-Hall (2008). "The folksonomy tag cloud: when is it useful?" *Journal of Information Science* 34.1, pp. 15–29 (ver p. 339).
- Sitemaps.org (2022). *What are Sitemaps?* URL: <https://www.sitemaps.org/> (acesso em 15/11/2022) (ver p. 268).
- Smiraglia, Richard P. (2001). *The Nature of "a work": implication for the organization of Knowledge*. Lanham, Maryland: Scarecrow Press (ver p. 6).
- Smith, Judy e Daniel Updegrove (fev. de 1993). "Navigating the Internet: Tools for discovery". *Penn-Printout* 9.4 (ver p. 263).
- Snowball Project (2022). *Portuguese stemming algorithm*. URL: <http://snowball.tartarus.org/algorithms/portuguese/stemmer.html> (acesso em 24/03/2022) (ver p. 86).
- Soergel, Dagobert (set. de 1994). "Indexing and Retrieval Performance: The Logical Evidence." *JASIS* 45, pp. 589–599. DOI: 10.1002/(SICI)1097-4571(199409)45:83.0.CO;2-E (ver pp. 151, 152).
- Souza, Marlo e Renata Vieira (2012). "Sentiment Analysis on Twitter Data for Portuguese Language". Em: *Proceedings of the 10th International Conference on Computational Processing of the Portuguese Language*. PROPOR'12. Coimbra, Portugal: Springer-Verlag, pp. 241–247. ISBN: 978-3-642-28884-5. DOI: 10.1007/978-3-642-28885-2_28. URL: http://dx.doi.org/10.1007/978-3-642-28885-2_28 (ver p. 279).
- Souza, Marlo, Renata Vieira et al. (2011). "Construction of a Portuguese Opinion Lexicon from multiple resources". Em: *In 8th Brazilian Symposium in Information and Human Language Technology - STIL, Mato Grosso* (ver p. 279).
- Sparck Jones, Karen (1972). "A Statistical Interpretation of Term Specificity and Its Application in Retrieval". *Journal of Documentation* 28.1, pp. 11–21. DOI: <https://doi.org/10.1108/eb026526> (ver pp. 197, 198).
- Srivastav, Saumitra (set. de 2014). *Friends of Solr - "Nutech and HDFS"*. September-2014 Meetup. Bangalore Apache Solr Groups. URL: <https://www.slideshare.net/saumitra121/friend-of-solr-nutch-hdfs> (acesso em 16/11/2022) (ver p. 270).
- StatCounter (2023). *Search Engine Market Share Worldwide*. URL: <https://gs.statcounter.com/search-engine-market-share> (acesso em 07/03/2023) (ver p. 250).
- Stock, W. G. e M. Stock (2013). *Handbook of Information Science*. Knowledge and Information. De Gruyter (ver p. 128).
- Su, Louise T. (1992). "Evaluation measures for interactive information retrieval". *Information Processing & Management* 28.4. Special Issue: Evaluation Issues in Information Retrieval, pp. 503–516. ISSN: 0306-4573. DOI: [https://doi.org/10.1016/0306-4573\(92\)90007-M](https://doi.org/10.1016/0306-4573(92)90007-M). URL: <https://www.sciencedirect.com/science/article/pii/030645739290007M> (ver p. 150).
- Sullivan, Danny (11 de nov. de 2010). *Dear Bing, We Have 10,000 Ranking Signals To Your 1,000. Love, Google*. Search Engine Land. URL: <https://searchengineland.com/bing-10000-ranking-signals-google-55473> (acesso em 07/03/2023) (ver p. 249).
- Sun, Jimeng (2019). *CSE 6250 Big Data for Healthcare: Solr*. Georgia Institute of Technology. URL: <https://www.sunlab.org/teaching/cse6250/fall2019/nlp/solr.html> (ver pp. xiii, 255).
- Taylor, F. K. (nov. de 1965). "Cryptomnesia and plagiarism". *The British journal of psychiatry : the journal of mental science* 111.480, pp. 1111–8 (ver p. 346).
- Taylor, Michael (2017). *Neural Networks: A Visual Introduction for Beginners*. Blue Windmill Media (ver p. 307).

- Thang, Sungrem (21 de mar. de 2016). *Introduction to Apache Nutch*. Presentation. Sigmoid. URL: https://www.slideshare.net/SigmoidHR/introduction-to-apache-nutch?qid=d2d76108-861c-47c0-aacc-b93328dd3240&v=&b=&from_search=1 (acesso em 16/11/2022) (ver p. 270).
- Thompson, H. S. e D. McKelvie (1997). “Hyperlink semantics for standoff markup of read-only documents”. Em: *Proceedings of SGML Europe*. URL: <http://www.ltg.ed.ac.uk/~ht/sgmleu97.html>. (acesso em 24/12/2021) (ver p. 55).
- Tillett, Barbara (fev. de 2004). *What is FRBR? A Conceptual Model for the Bibliographic Universe*. Functional Requirements for Bibliographic Records. Library of Congress Cataloging Distribution Service. URL: <http://www.loc.gov/cds/FRBR.html> (acesso em 04/01/2022) (ver p. 19).
- Tunkelang, Daniel (2008). *Faceted Search*. Boston: Morgan & Claypool (ver pp. 141, 142).
- Tversky, Amos (1977). “Features of Similarity”. *Psychological Reviews* 84.4, pp. 327–352 (ver pp. 116, 117).
- Unicode, Inc. (24 de jul. de 2017). *What is Unicode?* URL: <https://unicode.org/standard/WhatIsUnicode.html> (acesso em 24/12/2021) (ver p. 33).
- (22 de ago. de 2019). *The Unicode Standard: A Technical Introduction*. URL: <http://www.unicode.org/standard/principles.html> (acesso em 25/12/2021) (ver p. 33).
- (2021). *Unicode Version 14.0 Character Counts*. URL: https://www.unicode.org/versions/stats/charcountv14_0.html (acesso em 24/12/2021) (ver p. 33).
- van Rijsbergen, C. J. Keith (1979). *Information Retrieval* (ver pp. xii, 133, 138, 139, 162, 193, 224–228).
- (17 de mai. de 1990). “The Science of Information Retrieval: Its Methodology and Logic”. Em: *Conference Informatievetenschap in Nederland* (Conference Informatievetenschap in Nederland), RABIN, The Hague, p. 24 (ver p. 134).
- (2003). *Introduction to Information Retrieval (ESSIR 2003) - Slides*. URL: <http://mrim.imag.fr/essir03/PDF/4.Rijsbergen.pdf> (acesso em 01/01/2022) (ver p. 134).
- Vapnik, Vladimir N. (2000). *The Nature of Statistical Learning Theory*. 2^a ed. Information Science and Statistics. Springer (ver p. 295).
- Vaswani, Ashish et al. (2017). “Attention Is All You Need”. Em: *Advances in Neural Information Processing Systems*, pp. 5998–6008 (ver pp. 169, 309).
- Veloso, Caetano (1969). *Os Argonautas*. URL: <https://www.letras.mus.br/caetano-veloso/44761/> (acesso em 29/03/2022) (ver p. 14).
- Viégas, Fernanda B. e Martin Wattenberg (jul. de 2008). “TIMELINES Tag Clouds and the Case for Vernacular Visualization”. *Interactions* 15.4, pp. 49–52. ISSN: 1072-5520 (ver pp. 325, 339).
- Viégas, Fernanda B., Martin Wattenberg e Jonathan Feinberg (nov. de 2009). “Participatory Visualization with Wordle”. *IEEE transactions on visualization and computer graphics* 15, pp. 1137–44 (ver pp. 326, 339).
- Vigen, Tyler (2015). *Spurious Correlation*. New York: Hachette (ver p. 295).
- Vijaymeena, M.K. e K. Kavitha (mar. de 2016). “A Survey on Similarity Measures in TextMining”. *Machine Learning and Applications: An International Journal* 3.1, pp. 19–28 (ver p. 119).
- Vu, Tu e Mohit Iyyer (2019). “Encouraging Paragraph Embeddings to Remember Sentence Identity Improves Classification”. Em: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*. Association for Computational Linguistics, pp. 6331–6338 (ver p. 178).
- W3C (29 de set. de 2006). *Extensible Markup Language (XML) 1.1 (Second Edition)*. *W3C Recommendation 16 August 2006, edited in place 29 September 2006*. URL: <https://www.w3.org/TR/xml11/> (acesso em 29/12/2021) (ver p. 67).
- (26 de nov. de 2008). *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. *W3C Recommendation 26 November 2008*. World Wide Web Consortium. URL: <https://www.w3.org/TR/xml/> (acesso em 29/12/2021) (ver p. 67).

- W3C, ed. (5 de jan. de 2012). *W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures. W3C Recommendation 5 April 2012*. W3C. URL: <https://www.w3.org/TR/xmlschema11-1/> (acesso em 29/12/2021) (ver p. 67).
- (30 de dez. de 2021a). *About W3C*. W3C. URL: <https://www.w3.org/Consortium/> (acesso em 30/12/2021) (ver p. 57).
 - (30 de dez. de 2021b). *CSS Snapshot 2021*. W3C Group Note, 30 December 2021. Ed. por Tab Atkins Jr, Elika J. Etemad e Florian Rivoal. W3C (ver pp. 56, 58).
- Weber-Wulff, Debora (2010). “Why does plagiarism detection software not find all plagiarism?” Em: *Proceedings of the 4th International Plagiarism Conference*. Newcastle upon Tyne, UK: Routledge, pp. 62–73. DOI: 10.4324/9781315166148-5 (ver p. 346).
- WHATWG (21 de dez. de 2021). *HTML Living Standard Last Updated 21 December 2021*. WHATWG (Apple, Google, Mozilla, Microsoft). URL: <https://html.spec.whatwg.org/> (acesso em 30/12/2021) (ver pp. xi, 58, 59).
- Whistler, Ken, ed. (12 de ago. de 2023). *Unicode Standard Annex #15: UNICODE NORMALIZATION FORMS*. URL: <https://www.unicode.org/reports/tr15/> (acesso em 21/11/2023) (ver p. 34).
- Wikipedia (26 de dez. de 2020). *Setun*. URL: <https://en.wikipedia.org/wiki/Setun> (acesso em 25/12/2021) (ver p. 29).
- (30 de out. de 2022). *Tag Cloud*. URL: https://en.wikipedia.org/wiki/Tag_cloud#:~:text=The%20first%20tag%20clouds%20on,designer%20Stewart%20Butterfield%20in%202004. (acesso em 30/10/2022) (ver p. 324).
- Wong, S. K. M., Wojciech Ziarko e Patrick C. N. Wong (1985). “Generalized Vector Spaces Model in Information Retrieval”. Em: *Proceedings of the 8th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '85. Montreal, Quebec, Canada: Association for Computing Machinery, pp. 18–25. ISBN: 0897911598. DOI: 10.1145/253495.253506. URL: <https://doi.org/10.1145/253495.253506> (ver pp. 199, 200, 208).
- Workbench, The WEKA (2106). “Eibe Frank and Mark A. Hall and Ian H. Witten”. Em: *Data Mining: Practical Machine Learning Tools and Techniques*. 4^a ed. Online Appendix. Morgan Kaufmann (ver p. 337).
- Xexéo, Geraldo, Fernando Morgado e Patrícia Fiúza (2009). “Differential Tag Clouds: Highlighting Particular Features in Documents”. Em: *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - Volume 03*. WI-IAT 09. USA: IEEE Computer Society, pp. 129–132. ISBN: 9780769538013. DOI: 10.1109/WI-IAT.2009.247. URL: <https://doi.org/10.1109/WI-IAT.2009.247> (ver pp. 292, 323, 325, 327, 329).
- Xexéo, Geraldo (ago. de 2022). *Explorações em Mineração de Texto*. Rel. técn. ES-782/22. Relatório Técnico. Programa de Engenharia de Sistemas e Computação - COPPE/UFRJ. URL: <https://www.cos.ufrj.br/index.php/pt-BR/publicacoes-pesquisa/details/15/3060>.
- Xexéo, Geraldo et al. (dez. de 2008). “Using Wavelets to Classify Documents”. Em: *2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*. IEEE. DOI: 10.1109/wiat.2008.221. URL: <https://doi.org/10.1109/wiat.2008.221> (ver p. 220).
- Yang, Peilin, Hui Fang e Jimmy Lin (2017). “Anserini: Enabling the Use of Lucene for Information Retrieval Research”. Em: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '17. Shinjuku, Tokyo, Japan: Association for Computing Machinery, pp. 1253–1256. ISBN: 9781450350228. DOI: 10.1145/3077136.3080721. URL: <https://doi.org/10.1145/3077136.3080721> (ver p. 261).
- (out. de 2018). “Anserini: Reproducible Ranking Baselines Using Lucene”. *J. Data and Information Quality* 10.4. ISSN: 1936-1955. DOI: 10.1145/3239571. URL: <https://doi.org/10.1145/3239571> (ver pp. 255, 261).

- Yu, Clara, John Cuadrado and Maciej Ceglowski e J. Scott Payne (2002). *Patterns in Unstructured Data : Discovery, Aggregation and Visualization*. National Institute for Technology in Liberal Education. URL: http://www.seobook.com/lsi/cover_page.htm (acesso em 25/04/2023) (ver p. 217).
- Zobel, Justin e Alistair Moffat (2006). “Inverted Files for Text Search Engines”. *ACM Computing Surveys* 38.2, 6–es. ISSN: 0360-0300. DOI: 10.1145/1132956.1132959. URL: <https://doi.org/10.1145/1132956.1132959> (ver p. 107).
- Zobel, Justin, Alistair Moffat e Kotagiri Ramamohanarao (dez. de 1998). “Inverted Files versus Signature Files for Text Indexing”. *ACM Trans. Database Syst.* 23.4, pp. 453–490. ISSN: 0362-5915. DOI: 10.1145/296854.277632. URL: <https://doi.org/10.1145/296854.277632> (ver p. 107).

ÍNDICE REMISSIVO

- [, 251
8 bits, 28
detecção de *spam*, 292
abjads, 12
Aboutness, 132
abugidas, 12
acurácia, 148
afixos, 80
Agrupamento, 296
agrupamento, 8
alfabeto, 12
alfabéticos, 12
American Standard Code for Information Interchage, 30
Analogistas, 291
análise de sentimento, 292
Análise de Sentimentos, 301
análise global, 202
análise local, 202
aprendizado de máquina, 291
Aprendizado Não-Supervisionado, 296
Aprendizado por Reforço, 296
Aprendizado Semi-Supervisionado, 296
Aprendizado Supervisionado, 296
Arcabouço de Relevância Probabilística, 223
armadilha para crawler, 270
Arquivo, 21
arquivo de assinatura, 113
arquivo invertido, 107
ASCII, 17, 30, 47
Tabela, 30
atributo, 330
authorities, 238
auto-plágio, 346
autovalor, 357
autovetor, 357
autômato finito, 231
Bag of Words, 10
bag of words, 144
Bag-of-Words, 219, 311
bag-of-words, 140
baldes, 113
base set, 240
base64, 48
bases de conhecimento, 279
Bayesianos, 291
BCD, 29
BERT, 253
Biblioteca Eletrônica de Textos, 21
Binary Coded Decimal, 29
BM25, 229
BOM, 39
BoW, 10
Brown Corpus, 21
BrowseRank, 245
browsing, 129
buckets, 113
busca ad-hoc, 8, 129
cache, 110
cache lru, 122
Caffeine, 252
caixa alta, 83

- caixa baixa, **83**
 campo semântico, **327**, 332
 caractere, **28**
 caracteres, **12**
 Carriage Return, 35
 case folding, **82**
 CBOW, 174
 CCITT, 30
 CDC Cyber 170, 29
CG, 158
 chance, **227**
 chardet, 39–41
 Charset-Normalizer, 39
 charset-normalizer, 41
Classificação, 296
 classificação, 8
 classificação facetada, **142**
 Claude Shannon, 29
Code Page
 1252, 32
code page, 32
 detecção automática, 39
Code Pages, 32
code point, 28
Codificação de Binários em Texto, 47
Coeficiente de Dice, 118
 coeficientes de similaridade, **118**
 concatenando strings, 352
Conexionistas, 291
 conflation, 83
 conjunto par atributo, **330**
Contexto, 252
 contexto, **331**
Core Web Vitals, 253
Corpus, 21
 corpus, **8**
 CP1252, 37, 41
CR, 35
 crawler
 ótimo, 268
crawlers, 263
 crawling:priorização, 269
CSS, 39
 custo associado a recuperar um documento, 225
 código Morse, 30
dark web, 271
DCG, 158, 159
 DCG normalizado, **159**
- DCMI, **20**
 decomposição em valores singulares, **209**
Deep Web, 271
Descrição, 291
 descrição, **292**
 desinência, **80**
 desinência nominais, **80**
 desinência verbal, **80**
Diagnóstico, 291
 diagnóstico, **292, 293**
Direct Cumulated Gain, 158
Discounted Cumulative Gain, 158
discurso, 10
 distância de Chebyshev, **120**
 distância de Damerau-Levenshtein, **122**
 distância de edição, **122**
 distância de Hamming, **122**
 distância de Jaro, **122**
 distância de Levenshtein, **122**
 distância de Manhattan, **120**
 distância entre dois conjuntos, **121**
 distância euclidiana, **120**
DoCO, 20
Docoment Components Ontology, 20
Documentalidade, 7
documento, 6, 8
 documentos por atribuição, **7**
 documentos por intenção, **7**
 domínio, **330**
DTD, 68
Dublin Core, 141
Dublin Core Metadata Initiative, 20
EBCDIC, 29, 31
ElasticSearch, 261
ELEMENT, 68
 eliteness, **229**
embedding, 163, 173, 205, 218
 embeddings, 144, 311
 end of line, 36
 entidades HTML, 46
EOL, 36
 equivalência canônica, **34**
 erro médio absoluto, **122**
 erro médio quadrático, **122**
 erros ortográficos, 99
escalonador, 269
escrita, 27
 espaço métrico, **119**

- especificidade, **152**
 Estabilidade, **7**
 Evolucionistas, **291**
 exaustividade, **151**
- F, **158**, **158**
 falso negativo, **150**
 falso positivo, **150**
 fatias de strings, **354**
 feedback de pseudo-relavância, **201**
 feedback de pseudo-relevância, **202**
 feedback de relevância, **199**
 feedback de relevância através de clicks, **201**
 feedback de relevância explícito, **201**
 feedback de relevância implícito, **201**
FIFO, **110**
 filtragem, **129**
 fingerprint, **113**
Focused Crawling, **269**
 fonemas, **100**
 fonte, **28**
 fonético, **9**
 forma canônica, **80**
 formatos de edição, **54**
 formatos de marcação, **55**
FP, **150**
 framenets, **279**
FRBR, **19**
 full-text indexing, **142**
 função de atribuição de domínio, **330**
 função de classificação, **331**
 função de diferenciação, **335**
 função de distância, **119**
 função de risco, **225**
 Funções de erro, **121**
 funções de perda, **121**
 fusão, **83**
- ganho, **158**
 ganho Acumulado Direto, **158**
 Generalized Hebbian Algorithm, **210**
 Gerador de Amostras, **294**
 glifo, **28**
 glyph, **28**
 go-words, **97**
 Googlebot, **251**
 grafemas, **12**, **100**
 grammatical, **10**
- gráfico de precisão em 11 pontos de revocação, **155**
 gráfico de precisão por 11 pontos de revocação, **155**
 gráfico de precisão por revocação, **154**
 hiperonímia, **280**
 hiponímia, **280**
HITS, **238**
 holonímia, **280**
HTML, **38**, **56**, **59**
HTML5, **39**
 hubs, **238**
Hummingbird, **251**, **253**
- identidade, **115**
 identificação de entidades nomeadas, **8**
 idf, **193**
 idioma, **27**
IEEE 470, **29**
 igualdade, **115**
 implicação, **280**
 impressões digitais, **113**
 Indexalidade, **7**
 indexando strings, **353**
 indexação probabilística, **223**
 Intel 4004, **29**
 inverse document frequency, **193**
 inverted files, **107**
 ir datasets: Catalog, **287**
ISO 32000-2:2020, **63**
ISO Latin 1, **32**
ISO-8859, **32**
ISO/IEC 10646, **33**
- JavaScript, **39**
 JSON, **21**
- keywords, **140**
KNIME, **41**, **97**, **103**
Knowledge Graph, **253**
- Large Language Model, **253**
 Large Language Models, **233**
 Latent Semantic, **208**
 lema, **80**
 len
 strings, **353**
 LF, **35**, **41**
LFU, **110**

- Line Feed, 35
 lingua natural, **27**
 linguagem, **27**
 lista invertida, **107**, 140
 lista ponderada, 326
LLM, 233
Locality Sensitive Hashing, 113
 logográficos, **12**
Loss, 295
 loss functions, **121**
LRU, 110
LSA, 208
LSH, 113
LSI, 208
Lucene, 255
 léxico, **9**
 língua, **27**
- Mac OSX, 36
 Machine Readable Cataloging Record, 21
MAE, 122
 maior subsequência comum, **122**
MAP, 157
MARC, 21
 matriz de confusão, **148**
Matriz de Incidência, 140
 matriz documento-documento, 142
Matriz Termo Documento, 142, 208
Matriz Termo-Dокументo, 140
 matriz termo-termo, 142
 Mean Average Precision, **157**
Medic, 253
 medida de similaridade, **115**
medida-F, 158
 memoização, 122
 meronímia, 280
 metadado, **7**
 MIME, 48
 Mineração de Opinião, **301**
minterms, 200
 modelo de linguagem, **231**
Modelo Probabilístico, 223
Modelo Vetorial, 193
 modelos de linguagem, **165**
Modelos RSJ, 228
 morfema, **80**
 morfemas, **12**
 morfologia, 80
 morfológico, **9**
- MSE, **122**
MUM, 253
 Máquina de Aprendizado, 295
 métrica, **119**
- n-grama, **140**
 n-gramas, **9**
 navegação, **129**
NDCG, 159
 necessidade de informação, 8
Neural Matching, 254
NFC, 34
NFD, 34
NFKC, 34
NFKD, 34
 Nikolay Brusentsov, 29
NLTK, 81, 97
 nltk, 97
 normalização, 99
 normalização de maiúsculas e minúsculas, **82**
 normalização Unicode, **34**
Normalized DCG, 159
 notação, **28**
 Nutch, 269
 nuvem de palavras, 292, 326
 nuvem de rótulo de união, **335**
 nuvem de rótulo diferencial, **335**
 nuvem de rótulos, 327
 nuvem de rótulos de sumário, **334**
 nuvem de rótulos diferencial, 325
 nuvem de rótulos sumário, 325
Nuvens de rótulos, 323
- objeto, **330**
 observacional, **294**
odds, 227
Okapi, 229
one-hot, 161, 173
 ontologias, 279
 OpLexicon, 279
- 5, **157**
 10, **157**
Page Description Format, 62
Page Experience, 253
PageRank, 268
Pagerank, 236, 243, 252
 palavras-chave, **140**
Panda, 251, 252
 par atributo, **330**

- Payday Loan Algorithm, **253**
 PCDATA, 68
 PDF, 62, **62**
 PDF 2.0, **63**
 PDF Parser, **64**
 pdfminer, 63
 Penguin, **253**
 Perda, **295**
 perda, **225**
 PHP, 39
 Pigeon, **253**
 Pluralidade, **7**
 plágio, 345
 plágio acidental, **346**
 plágio direto, **345**
 plágio intencional, **346**
 plágio não-intencional, **346**
 polaridade, 279
 ponto flutuante, 29
 Portable Document Format , **62**
 Porter Stemmer, **80**
 Possum, **253**
 posting file, **107**
 PostScript, 62, **62**
 pragmático, **10**
 precision at 10, **157**
 precisão, **148**, 150
 precisão em 10, **157**
 Precisão R, **157**
 Predição, **291**
 predição, **292**
 prefixos, **80**
 Prescrição, **291**
 PRF, **223**
 Princípio da Ordenação Probabilística, **223**
 Probabilistic Relevance Framework, **223**
 probability ranking principle, 223
 processo genérico de aprendizado, 294
 Produtividade, **7**
 PRP, **223**
 PS, 62
 PyLucene, **258**
 Python, 59
 Python 3, 39
 Qualidade, **252**
 R-Precision, **157**
 radical, **80**
 raiz, **80**
 RankBrain, **253**
 raspagem, **271**
 Real Necessidade de Informação, **131**
 recurso, **331**
 Regressão, **296**
 relevance feedback, 223
 Relevância, **252**
 relevância, **133**, **134**
 manifestações, 135
 representação de caracteres, 27
 representação de recursos, **332**
 revocação, **148**
 RFC, 57
 RIN, **131**
 robots, **263**
 root set, **240**
 RSLP, **86**
 ruído, 327
 Rótulos, **327**
 saturação, **229**
 scrapping, **271**
 script, **27**, 33
 Search Engine Optimization, **245**
 semelhança, 115
 Semântica Latente, **208**
 semântico, **10**
 sensibilidade, **148**
 SEO, **245**
 Sergei Sobolev, 29
 Setun, 29
 SGML, 56
 Significado, **252**
 silábicos, **12**
 Simbolistas, **291**
 similaridade, 115, **191**
 Similaridade de Jaccard, **191**
 similaridade do cosseno, **121**
 similaridade entre dois conjuntos, **118**
 simple matching coefficient, 119
 sinais, **251**
 sintático, **10**
 sistema de escrita, **12**, **27**
 sistemas mistos, **12**
 Sitemaps, **268**
 Skip-gram, **174**, **219**, **311**
 Snowball, **85**
 Solr, **261**

- soma da diferença absoluta, **121**
- soma do quadrado das diferenças, **121**
- SpamBrain, **253**
- Sphinx, **261**
- spiders, **263**
- SQL, 39
- sqlite 3, 109
- stemmer, 197
- Stemmers, **83**
- stemmers, 140
- Stopwords, **92**
- stopwords, 97
- str, 352
- string, 352
- strings
 - [] , 353
 - concatenação, 352
 - fatias, 354
 - indexando, 353
 - len, 353
 - tamanho de uma..., 353
- Subcorpus, **21**
- sufixos, **80**
- SVD, **209**
- synsets, **280**
- Tabela ASCII, 31
- tag cloud, 326
- tamanho da CPU, 28
- tamanho de uma string, 353
- tema, **80**
- term frequency, **193, 196**
- termo, **140**
- termo índice, **140**
- termos coordenados, 280
- tesauros, **279**
- teste-F, **158**
- texto, **5**
- TF, **196**
- tf, **193**
- tf-idf, **196**
- thesaurus, 200
- tokenização, **81**
- tokens, **81**
- TopHeavy, **253**
- transcrição, **28**
- tráicos (*featural*), **12**
- Twitter, 21
- Unicode, 17, **33**
- Usabilidade, **252**
- UTF-16, **33**
- UTF-32, **33**
- UTF-8, 17, 29, **33, 38, 39**
- uuencode, 48
- verdadeiro negativo, **150**
- verdadeiro positivo, **150**
- viés, **294**
- VN, **150**
- vogal temática, **80**
- VP, **150**
- W3C, 46, 57
- Wavelets, **220**
- weighted list, 326
- word cloud, 326
- Word2Vec, 144
- wordle, 326
- WordNet, **279**
- wordnets, 279
- WRU Searching Selector, **182**
- XML, 21, 38, 59, 67
- XML 1.0, 67
- XML 1.1, 67
- índice invertido, **107**