UNIVERSIDADE FEDERAL DO VALE DO SÃO FRANCISCO
CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

RUAN DE MEDEIROS BAHIA

Estudo investigativo sobre o desempenho de atributos de Recuperação de Informação em tarefas de Mineração de Textos

JUAZEIRO - BA
2020

UNIVERSIDADE FEDERAL DO VALE DO SÃO FRANCISCO
CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

RUAN DE MEDEIROS BAHIA

Estudo investigativo sobre o desempenho de atributos de Recuperação de Informação em tarefas de Mineração de Textos

Trabalho apresentado à Universidade Federal do Vale do São Francisco - Univasf, Campus Juazeiro, como requisito da obtenção do título de Bacharel em Engenharia de Computação. Orientador: Prof. Dr. Rosalvo Ferreira de Oliveira Neto

JUAZEIRO - BA

2020

AGRADECIMENTOS

O conhecimento humano não é desenvolvido individualmente, inúmeros indivíduos colaboraram para a realização deste trabalho. Muitos livros, artigos, trabalhos, e linhas de código escritas por pessoas de vários países deixaram um pouco de conhecimento comigo. Diversas pessoas contribuíram para minha formação como ser humano, família, amigos, professores, colegas, conhecidos e desconhecidos, e a essas pessoas eu sou grato.

Dedico este trabalho ao meu pai, Antonio, pois se segui esse caminho com certeza foi devido à influência dele.

RESUMO

O mundo informatizado gera uma quantidade gigantesca de dados textuais diariamente, e a Mineração de Texto objetiva transformar esses dados em informações úteis, em conhecimento, tendo aplicação inclusive na área forense. A área de Recuperação de Informação contribui para o desenvolvimento da Mineração de Texto no pré-processamento, porém sua utilização direta na criação de atributos não é usual, sendo proposto pela primeira vez por Weren, Moreira e Oliveira (2014).

A partir de uma revisão bibliográfica das áreas de Recuperação de Informação e de Mineração de Texto, o autor sugere uma metodologia para criação de atributos utilizando a função de ranqueamento BM25, similar à utilizada por Weren (2014), e utilizando ferramentas de armazenamento e indexação já existentes para o cálculo. Nesta metodologia, a análise do desempenho dos novos atributos sugeridos é feita por meio da mensuração do desempenho de classificador por medidas consolidadas na literatura de Mineração de Texto como acurácia F_1-score.

Os resultados obtidos a partir da reprodução da metodologia sugerida demonstram o Zettair como a ferramenta com melhor desempenho para indexação de documentos, e o Elasticsearch como o melhor SGBD tanto no quesito desempenho de indexação quanto de consulta.

Os atributos sugeridos, derivados de Recuperação de Informação, apresentam ganho de desempenho em classificadores de um corpus pequeno com documentos longos, 645 artigos. Estes mesmos atributos aplicados em outro classificador para um corpus grande com documentos pequenos, 300 mil *tweets*, causa perda de desempenho desse classificador.

Palavras-chave: Mineração de Texto, Recuperação de Informação, criação de atributos, avaliação de desempenho, engenharia de atributos.

ABSTRACT

In the information age currently going on, the world generates a huge amount of textual data on a daily basis, and Text Mining aims to turn this data into useful information, knowledge, and that has applications even in the forensic area. The study field of Information Retrieval contributes to the development of Text Mining in preprocessing, however it can also used in the creation of attributes for classifiers.

From a literature review of the study fields of Information Retrieval and Text Mining, we suggest to create attributes using the BM25 ranking function, similar to attributes created by Weren (2014), and using existing storage and indexing tools to calculate them. A methodology for analyzing the performance of the suggested new attributes is established, and the measurement of classifier performance by consolidated measures in the Text Mining literature is proposed, such measurements are accuracy and F_1-score.

The results obtained by reproducing the suggested methodology demonstrate Zettair as the tool with the best perfomance for document indexing, and Elasticsearch as the best DBMS both in terms of indexing and query performance.

The suggested attributes, derived from Information Retrieval, exibith performance gains in classifiers of a small corpus with long documents, 645 articles. These same attributes applied in another classifier for a large corpus with small documents, 300 thousand textit tweets, cause this classifier to lose performance.

Key-words: *Text Mining, Information Retrieval, feature creation, performance evaluation, feature engineering.*

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE CÓDIGOS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface* (interface de programação de aplicação)
AQL	*ArangoDB Query Language*
atrib.	atributos
BD	Banco de Dados
BIM	*Binary Independence Model* (modelo de independência binária)
camadas.	cam
CLEF	*Conference and Labs for the Evaluation Forum*
CNN	*Convolutional Neural Network* (Rede Neural Convolucional)
cons.	consulta
dim.	dimensões
esc.	escondidas
ESRC	*ELMo Sentence Representation Convolutional*
HTML	*HyperText Markup Language* (linguagem de marcação de hipertexto)
JSON	*JavaScript Object Notation* (notação para objetos JavaScript)
KDD	*Knowledge Discovery in Data* (descoberta de conhecimento em dados)
MT	Mineração de Texto
NoSQL	*Not Only SQL* ou *no SQL* (não apenas SQL ou sem SQL)
núm.	número
PAN	Organização que se originou do *International Workshop on Plagiarism Analysis, Authorship Identification, and Near-Duplicate Detection*
PDF	*Portable Document Format* (formato de documento portátil)
pont.	pontuação
pos.	posição
PRP	Probability Ranking Principle (princípio de ranqueamento probabilístico)

RI	Recuperação de Informação
RNG	*Random Number Generator* (gerador de números aleatórios)
SGBD	Sistemas Gerenciadores de Banco de Dados
SQL	*Structured Query Language* (linguagem de consulta estruturada)
SVC	*C-Support Vector Classification* (classificação por Máquinas de Vetor de Suporte com parâmetro C)
SVM	*Support Vector Machines* (Máquinas de Vetor de Suporte)
TREC	*Text REtrieval Conference format* (formato da Conferência de Recuperação de Texto)
W3C	*World Wide Web Consortium*
XML	*Extensive Markup Language* (formato de documento portátil)

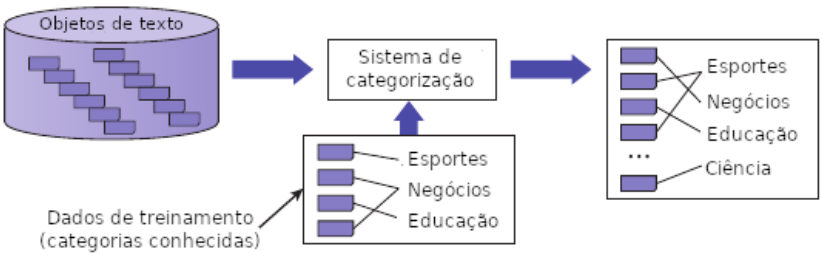
SUMÁRIO

1 INTRODUÇÃO

A informatização do mundo, juntamente com a evolução dos equipamentos computacionais, vem permitindo a geração e coleta de enormes volumes de dados das mais variadas fontes, podendo-se afirmar que vivemos na era dos dados (HAN; KAMBER; PEI, 2011, p. 1). Grande parte dos dados criados *online* está em forma de texto (escrito em linguagem natural) e um estudo feito pela Universidade da Califórnia em Berkeley em 2003 apontou, por exemplo, que somente notícias de jornais (considerando armazenamento digital do texto dos mesmos) representavam cerca de 13,5 terabytes por ano, livros cerca de 5,5 terabytes por ano, e e-mails mais de 440 exabytes por ano (LYMAN; VARIAN, 2003) (ZHAI; MASSUNG, 2016, p. 3).

A coleta e análise da sobrecarga diária de dados textuais se apresenta como um problema que a Mineração de Texto (MT) tenta resolver utilizando de técnicas de mineração de dados, aprendizado de máquina, processamento de linguagem natural, Recuperação de Informação (RI) e gerenciamento do conhecimento (HAN; KAMBER; PEI, 2011, p. 1) (FELDMAN; SANGER, 2006) (SAMMUT; WEBB, 2017, p. 1241). Técnicas de Mineração de Texto são aplicadas na classificação e clusterização de documentos, sumarização de opiniões na internet, acesso de dados biomédicos (AGGARWAL; ZHAI, 2012, p. 4–8), e também em tarefas de identificação de perfis de autoria[1] (RANGEL et al., 2014, p. 906) (RANGEL et al., 2018, p. 6–7), auxiliando em investigações forenses linguísticas (CHASKI, 2012).

Figura 1 – Tarefa de categorização de texto (com exemplos de treinamento disponíveis).



Fonte: Figura adaptada de Zhai e Massung (2016, p. 300).

A Mineração de Texto aborda, dentre seus tipos de tarefas oriundas da mineração de dados, a tarefa de classificação nas coleções de documentos, geralmente chamada de classificação de texto ou categorização de texto (ZHAI; MASSUNG, 2016, p. 35). Classificação é definido como o processo de designar uma ou mais categorias a cada objeto de texto, dentre categorias predefinidas, sendo que predominantemente é utilizado um conjunto de textos já classificados para treinamento (JO, 2018, p. 7) (ZHAI; MASSUNG, 2016, p. 299). Esse processo está exemplificado na Figura 1.

[1] A identificação de perfis de autoria consiste na extração de características do autor com base no conteúdo e estilo do texto. Essas características podem ser gênero, faixa etária, escolaridade, entre outras (WEREN et al., 2014, p. 266).

No processo de classificação de texto são derivados atributos dos objetos de texto originais, um passo necessário para funcionamento do modelo de classificação proveniente da área de aprendizado de máquina (FELDMAN; SANGER, 2006, p. 64). Diferentes conjuntos de atributos podem impactar diretamente no desempenho de um classificador (ZHAI; MASSUNG, 2016, p. 304–306), o qual é tipicamente mensurado pela acurácia[2] (ZHAI; MASSUNG, 2016, p. 313–314) (JO, 2018, p. 9).

A criação de atributos[3] é algo que pode melhorar a acurácia dos classificadores utilizados em tarefas de mineração de dados (MA; TANG; AGGARWAL, 2018, p. 118), os mesmos classificadores também são utilizados em tarefas de Mineração de Texto (SAMMUT; WEBB, 2017, p. 1241). Portanto, sugerir a criação de novos atributos, e avaliar o impacto destes atributos no desempenho de classificadores, contribui para o refinamento das técnicas de Mineração de Texto. Apesar de técnicas de Recuperação de Informação fundamentarem e servirem de base, principalmente, para o pré-processamento dos textos na MT, a utilização de funções de ranqueamento de RI na criação de atributos para MT é recente, sendo encontradas na literatura somente as pesquisas de Weren (2014).

A área de Recuperação de Informação abarca os processos de armazenamento, indexação, recuperação e ranqueamento de consultas sobre coleções de documentos (BAEZA-YATES; RIBEIRO-NETO, 2011, p. 5–8). A RI sempre busca a otimização destes processos, tanto em questão de melhorias na velocidade de resposta, como também na questão da satisfação da necessidade de informação dos usuários dos sistemas de RI (SAMMUT; WEBB, 2017, p .671–672), e segundo Kowalski (2010, p. 2, tradução nossa):

> O objetivo principal de um sistema de Recuperação de Informação é minimizar a sobrecarga do usuário em localizar informação de valor. Na perspectiva do usuário, sobrecarga pode ser definido como o tempo que decorre para localizar a informação necessária. O tempo inicia quando um usuário começa a interagir com o sistema e termina quando encontra os itens de interesse.

Implementações de sistemas de RI que atendam os objetivos da área se tornam complexas por estes sistemas terem que lidar com a integração dos processos de forma a trazer a melhor experiência ao usuário. Portanto, a utilização de ferramentas que subsidiem as tarefas de armazenamento, indexação, recuperação e ranqueamento da Recuperação de Informação é vantajosa por facilitar a introdução de atributos derivados de RI nas tarefas de MT.

O uso de diferentes ferramentas, para realizar a geração dos mesmos atributos derivados de RI, possibilita ainda a comparação do desempenho dessas ferramentas nas tarefas de indexação e de consulta para o ranqueamento.

---

[2] Número de previsões corretas dividido pelo número total de previsões feitas (ZHAI; MASSUNG, 2016, p. 313).

[3] Processo da área de engenharia de atributos (*feature engineering*), também chamado de extração, ou construção, de atributos (SAMMUT; WEBB, 2017, p. 498–503).

1.1 OBJETIVO GERAL

Avaliar o desempenho de atributos oriundos de Recuperação de Informação para tarefas de Mineração de Textos.

1.2 OBJETIVOS ESPECÍFICOS

- Avaliar o ganho de desempenho de classificadores de Mineração de Texto com adição de atributos derivados da função de ranqueamento BM25 da Recuperação de Informação, em pelo menos 2 corpus de competições diferentes, utilizando medidas consolidadas na literatura;
- Reproduzir soluções disponíveis *online* para os corpus selecionados, comprovando as medidas dos resultados das competições;
- Elencar em qual dos corpus selecionados os atributos criados proporcionam maior ganho de desempenho de classificador;
- Comparar o desempenho computacional de ferramentas de armazenamento e indexação de textos:
  - na questão de indexação;
  - na questão de consulta utilizando as implementações do BM25 nativas das ferramentas;

1.3 ORGANIZAÇÃO DO TRABALHO

Este trabalho está dividido em 5 capítulos. O primeiro capítulo foi introdutório e apresentou a motivação para o desenvolvimento desta pesquisa, e qual o problema específico que o projeto aborda.

No segundo capítulo é desenvolvida a fundamentação teórica dos principais campos de abordagem, sendo realizadas revisões bibliográficas das áreas de Recuperação de Informação e de Mineração de Texto.

O terceiro apresenta a metodologia para execução do projeto, ilustrada por meio de um diagrama, que é logo em seguida detalhado textualmente, e os tópicos principais são expandidos em subseções.

Detalhes da configuração experimental, da execução da pesquisa e os resultados obtidos são apresentados, e brevemente discutidos, no quarto capítulo. A disposição dos resultados é feita por meio da tabulação dos dados e também disposição em gráficos.

O quinto e último capítulo traz as considerações finais e visiona trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

A utilização de funções de ranqueamento da Recuperação de Informação engloba o conhecimento acerca de fundamentos da área que serão abordados na Seção 2.1, que discute desde o surgimento da área até sua evolução para os métodos probabilísticos culminando na função de ranqueamento BM25.

Para a criação de atributos em tarefas de Mineração de Texto é necessário entender o processo de descoberta do conhecimento da Mineração de Dados, as peculiaridades para criação destes atributos, e ainda métodos de avaliação dos atributos criados. Isso é feito na Seção 2.2.

2.1 RECUPERAÇÃO DE INFORMAÇÃO

A busca por informação é uma necessidade humana, e uma das principais maneiras de obtê-la é consultar outras pessoas. No entanto, devido ao grande acúmulo de informação das sociedades, uma pessoa não pode carregar consigo todo o conhecimento do mundo. Assim, um modo considerado primordial de transferir esse conhecimento, que tratamos aqui como informação, é por meio de registros físicos em papel, livros e similares (GROSSMAN; FRIEDER, 2004, p. 1).

No processo de organização desses registros físicos, é notória a função dos bibliotecários de separar os vários tipos de conhecimento que os mais diversos livros podem abrigar e, portanto, os sistemas de classificação de áreas e subáreas do conhecimento são um auxílio para as necessidades de busca por informação que uma pessoa pode ter (MANNING; RAGHAVAN; SCHüTZE, 2008, p. 1) (SANDERSON; CROFT, 2012, p. 1446) (BAEZA-YATES; RIBEIRO-NETO, 1999, p. 6). Esses sistemas de classificação facilitam a localização de informações específicas a partir de uma pergunta que uma pessoa pode fazer, mas é necessário saber como o sistema de classificação funciona para que o usuário possa tentar suprir sua necessidade, ou pelo menos será necessário um especialista no sistema (o bibliotecário) para lhe guiar. E mesmo assim, depois de adquirir os diversos materiais que podem responder à sua pergunta, esta pessoa ainda terá que conferir nos textos se estes contêm a informação desejada.

Os sistemas de classificação manual se mostram ineficazes devido ao surgimento crescente e constante de novas informações (BAEZA-YATES; RIBEIRO-NETO, 1999, p. 6) desde o início do século 20. Desta maneira, além da grande quantidade de novas pesquisas científicas sendo publicadas e livros surgindo, entre outros registros históricos, os quais precisam ser classificados, existe também o problema da classificação não poder abranger todo tipo de necessidade que tal material pode satisfazer (SANDERSON; CROFT, 2012, p. 1444).

A preocupação com sistemas que possam indexar todo esse material e fornecer um acesso rápido às necessidades de informação de uma pessoa surgiu também no início do século 20 (BUSH, 1979), onde sistemas mecânicos de recuperação de informação foram vislumbrados.

A partir da criação de sistemas computacionais na década de 1940, foi vista a possibilidade de criação de sistemas que armazenassem informações e possibilitassem essa consulta rápida sobre as informações armazenadas, sendo necessário estabelecer algoritmos que retornassem informação relevante ao que o usuário do sistema procura. Teve início nesse momento o campo científico da Recuperação de Informação (RI, do inglês *Information Retrieval*) que envolve encontrar material (geralmente documentos) de natureza desestruturada (geralmente texto) que satisfaça uma necessidade de informação dentro de grandes acervos (geralmente armazenados em computadores) (MANNING; RAGHAVAN; SCHüTZE, 2008, p. 1).

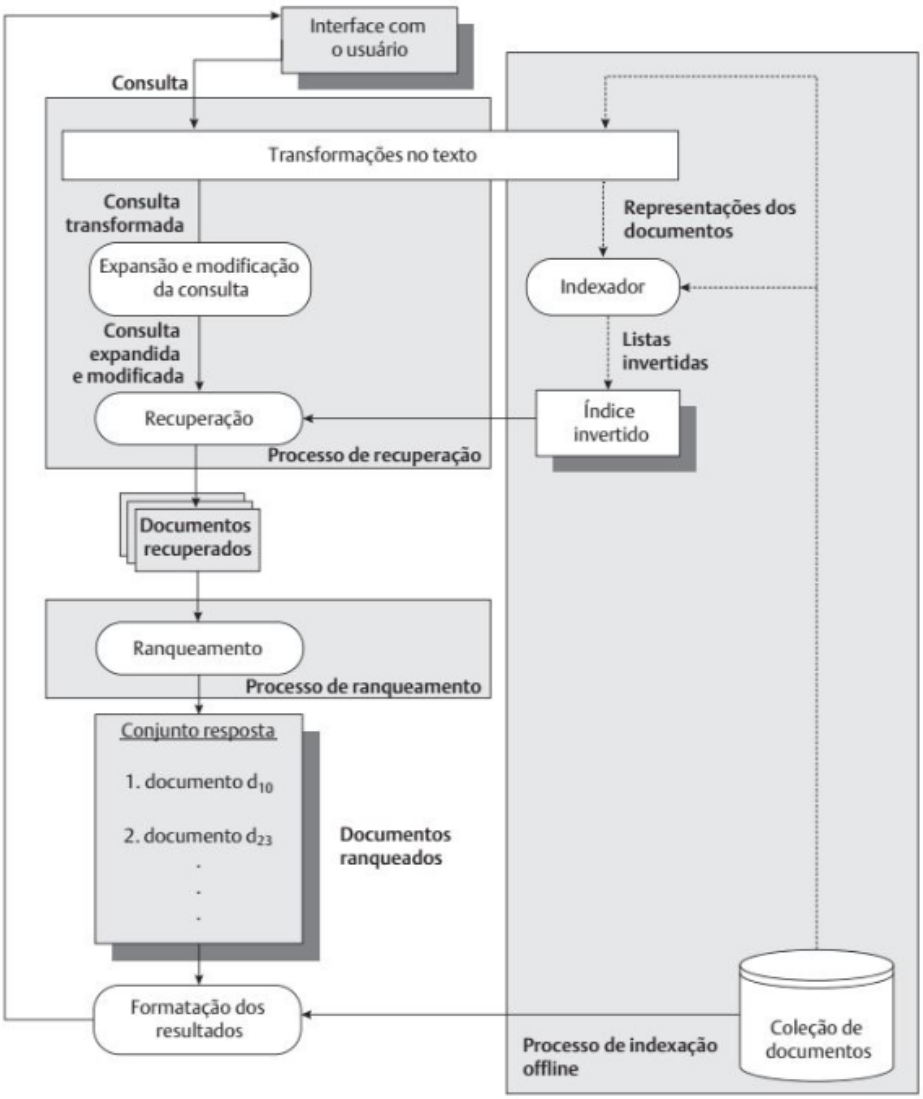
A lei de Moore diz que o crescimento da velocidade de processamento é contínuo (MOORE, 1975), e de maneira similar existe uma duplicação constante da capacidade de armazenamento digital a cada dois anos (WALTER, 2005). Logo, a necessidade de sistemas de Recuperação de Informação surge do crescimento exponencial das coleções de informação derivadas do crescimento de armazenamento, e consequente inabilidade das técnicas tradicionais de catalogação de lidar com isso (SANDERSON; CROFT, 2012). Ter um amontoado de conhecimento, informação, e não poder acessar o que é relevante de modo rápido não é interessante pois o desenvolvimento de pesquisas, por exemplo, fica comprometido e pode perder relevância (BUSH, 1979).

A RI como uma disciplina de pesquisa iniciou no final da década de 50 a partir do uso de computadores na busca de referências de texto associadas com um assunto (SANDERSON; CROFT, 2012, p. 3), as preocupações iniciais dessa área eram (a) *como indexar documentos* e (b) *como recuperá-los*, sendo a busca da melhor maneira de executar tais tarefas o principal objetivo da RI.

Logo no início do seu desenvolvimento as técnicas de RI buscaram se basear em sistemas existentes já consolidados no campo bibliotecário para indexar coleções de itens, tendo como uma técnica de abordagem clássica atribuir códigos numéricos a essas coleções, como por exemplo o feito pelo sistema de Classificação Decimal de Dewey (SANDERSON; CROFT, 2012, p. 1446). No entanto, foi demonstrado por Cleverdon (1959) que um sistema baseado em palavras, como o sistema Uniterm proposto por Taube, Gull e Wachtel (1952), era tão bom e até melhor que outras abordagens clássicas, sendo a indexação por palavras posteriormente adotada pelos sistemas de RI (SANDERSON; CROFT, 2012, p. 1446).

Um exemplo do funcionamento de um sistema moderno de RI pode ser visto na Figura 2, onde está apresentado um fluxograma dos processos de indexação, recuperação, e ranqueamento de documentos. Sendo o ranqueamento feito por sistemas de RI o interesse deste trabalho, alguns dos termos apresentados nessa figura serão abordados logo mais.

Figura 2 – Processos de indexação, recuperação, e ranqueamento dos documentos.



Fonte: Figura extraída de Baeza-Yates e Ribeiro-Neto (2011, p. 8).

Segundo Baeza-Yates e Ribeiro-Neto (2011, p. 7) a arquitetura de funcionamento de um *software* de RI inicia com o processo de indexação executado *offline*, antes do sistema estar pronto para processar consultas, e a estrutura de indexação mais popular é a chamada de índice invertido. Após a indexação da coleção de documentos, o sistema está preparado para o processo de recuperação, no qual o usuário especifica sua consulta, que é modificada pelo sistema para ficar mais próxima do sistema de indexação utilizado. A consulta modificada é utilizada para obter o conjunto de documentos recuperados, os quais, idealmente, satisfazem as necessidades de informação do usuário. Por fim, os documentos recuperados são ranqueados pela sua pertinência de relevância para o usuário.

A arquitetura apresentada na Figura 2 inclui o ranqueamento feito pelos sistemas de RI mordernos, que são uma expansão dos sistemas de RI booleanos, estes últimos contam com uma indexação e recuperação mais simples que os sistemas de RI ranqueados como será

esclarecido nas subseções a seguir.

2.1.1 Métodos booleanos

Uma consulta (chamada de *query*) representa uma necessidade de informação a ser saciada por um sistema de RI, e essa consulta é composta de termos (um sinônimo para palavras) que nos primeiros desses sistemas era limitada a combinações lógicas e eram recuperados os documentos que tinham correspondência exata com ela (SANDERSON; CROFT, 2012, p. 1446). Esse método de recuperação de informação é conhecido como recuperação booleana, e para indexar os documentos é utilizada, geralmente, uma matriz binária de incidência de termo-documento. Exemplificamos uma matriz dessas na Tabela 1, que é o exemplo dado por Manning, Raghavan e Schütze (2008, p. 3–4) de uma matriz de incidência de termo-documento para o livro *Shakespeare's Collected Works*, que reúne as obras completas de Shakespeare.

Tabela 1 – Matriz de incidência de termo-documento do livro *Shakespeare's Collected Works*. Cada elemento (i, j) da matriz é 1 se a peça de teatro na coluna j contém a palavra na linha i, caso contrário o elemento é 0.

Palavra \ Peça de teatro	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
Antony	1	1	0	0	0	1	
Brutus	1	1	0	1	0	0	
Caeasar	1	1	0	1	1	1	
Calpurnia	0	1	0	0	0	0	
Cleopatra	1	0	0	0	0	0	
mercy	1	0	1	1	1	1	
worser	1	0	1	1	1	0	
...							

Fonte: Tabela adaptada de Manning, Raghavan e Schütze (2008, p. 4).

2.1.2 Ranqueamento

Devido à limitação dos métodos booleanos de somente retornar resultados conforme a presença ou não dos termos da consulta nos documentos (MANNING; RAGHAVAN; SCHüTZE, 2008, p. 100), foi proposto em 1957 por Luhn e em 1959 por Maron *et al.* uma abordagem de recuperação ranqueada (SANDERSON; CROFT, 2012, p. 1446) a qual, em contraste com recuperação booleana, era baseada nos termos de consulta e estabelecia uma pontuação para cada artigo de modo probabilístico e retornava os artigos de modo ordenado. Desta maneira, demonstraram que essa técnica superava a recuperação booleana.

O procedimento fundamental para ranqueamento dos documentos, conforme os termos de consulta, consiste na atribuição de pontuação aos documentos a partir da con-

tabilização do número de aparições (chamada de frequência) de cada um dos termos no documento. Essa pontuação é calculada considerando que além da frequência do termo, denotada como $\text{tf}_{t,d}$ que é o número de ocorrências do termo t em um documento d, existe também a sua relevância, que depende do número de aparições do termo na coleção de documentos inteira. Quanto mais um termo aparece na coleção menos relevante ele é, e este valor de relevância é denotado por idf_t que é o inverso da frequência de um termo t em uma coleção de documentos. Segundo Manning, Raghavan e Schütze (2008, p. 108) este valor da relevância é calculado do seguinte modo:

$$\text{idf}_t = \log \frac{N}{\text{df}_t}, \tag{2.1}$$

Onde N é o número total de documentos na coleção, e df_t é a contagem de ocorrências do termo t em toda coleção de documentos.

O valor resultante da relação entre a frequência do termo e o inverso da frequência nos documentos é chamado de $\text{tf-idf}_{t,d}$ (*term frequency-inverse document frequency*), sendo este valor um dos pesos mais utilizados para ranqueamento (MANNING; RAGHAVAN; SCHüTZE, 2008, p. 107–110), e é calculado como segue:

$$\text{tf-idf}_{t,d} = \text{tf}_{t,d} \times \text{idf}_t. \tag{2.2}$$

Tabela 2 – Exemplo de cálculo do valor de tf-idf.

Termo / Documento			D_1		D_2		D_3	
	df_t	idf_t	$\text{tf}_{t,d}$	$\text{tf-idf}_{t,d}$	$\text{tf}_{t,d}$	$\text{tf-idf}_{t,d}$	$\text{tf}_{t,d}$	$\text{tf-idf}_{t,d}$
car	18165	1,65	27	44,55	4	6,6	24	39,6
auto	6723	2,08	3	6,24	33	68,64	0	0
insurance	19241	1,62	0	0	33	54,46	29	46,98
best	25235	1,5	14	21	0	0	17	25,5

Fonte: Tabelas disponíveis em Manning, Raghavan e Schütze (2008, p. 109–110).

Na Tabela 2 temos um exemplo de cálculo dos valores de tf-idf para posterior cálculo da pontuação para ranqueamento, conforme alguma determinada consulta. A pontuação de um documento d é a soma dos pesos de tf-idf de cada termo t em d, sendo os termos t presentes na consulta realizada (MANNING; RAGHAVAN; SCHüTZE, 2008, p. 109), representamos esse cálculo do seguinte modo:

$$\text{Pontuação}(q,d) = \sum_{t \in q} \text{tf-idf}_{t,d}. \tag{2.3}$$

Utilizando a Equação 2.3 uma consulta com os termos *auto car* retornaria no seu ranqueamento os documentos com a seguinte pontuação, calculamos Pontuação({*auto*,*car*},D_x) para cada documento, por exemplo:

- D_1: 50,79
- D_2: 75,24
- D_3: 39,60

A ordenação dos documentos apresentados como resultado à consulta *auto car* seria então a seguinte: 1º - D_2; 2º - D_1; e 3º - D_3, que se observamos a Tabela 2 é um bom resultado, já que o D_2 contém uma grande frequência do termo *auto* e o D_3 não possui este termo.

Ao longo dos anos foi demonstrada a superioridade da recuperação ranqueada sobre a recuperação booleana (JONES, 1981), e são as técnicas de recuperação ranqueadas que trazem maior interesse para a área de Mineração de Textos, em específico estamos interessados nos modelos vetoriais e os modelos probabilísticos de RI que são evoluções da recuperação ranqueada.

2.1.2.1 Modelo de espaço vetorial

O modelo vetorial surge a partir das limitações do modelo Booleano, que não considera frequência dos termos, e nele são representados um conjunto de documentos num espaço vetorial comum (MANNING; RAGHAVAN; SCHüTZE, 2008, p. 110). Oferece a possibilidade de resultados parciais por meio da atribuição de pesos não binários para os termos da consulta e também para os termos presentes nos documentos, que são utilizados para determinar o grau de similaridade entre cada documento armazenado no sistema e uma determinada consulta (BAEZA-YATES; RIBEIRO-NETO, 2011, p. 77).

Neste modelo os resultados são apresentados em ordem decrescente de similaridade, considerando a correspondência parcial, e não obrigatoriamente total, com os termos da consulta, provendo assim uma resposta mais precisa para as necessidades de informação do usuário. A similaridade nos modelos de espaço vetorial é tratada como uma noção de relevância, e a principal premissa é de que a relevância de um documento em relação a uma consulta está correlacionada com a similaridade entre o documento e consulta (ZHAI; MASSUNG, 2016, p. 110).

Segundo Baeza-Yates e Ribeiro-Neto (2011, p. 77), os pesos $w_{i,j}$ associados com um par termo-documento são não negativos e não binários. Os termos de indexação são considerados mutualmente independentes e são representados como vetores unitários de um espaço t-dimensional, aonde t é número total de termos de indexação. A representação de um documento d_j e uma consulta q são vetores t-dimensionais representados como segue

nas Equações 2.4 e 2.5 abaixo:

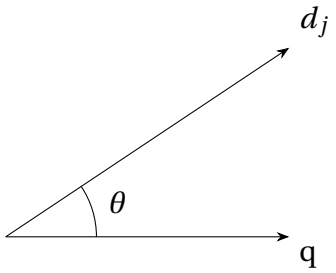
$$\vec{d}_j = (w_{1,j}, w_{2,j}, \cdots, w_{t,j}) \tag{2.4}$$

$$\vec{q} = (w_{1,q}, w_{2,q}, \cdots, w_{t,q}) \tag{2.5}$$

O valor tf-idf$_{t,d}$ (apresentado na Subseção 2.1.2) é um dos padrões de pesos mais comumente utilizados, sendo aplicado diretamente para os pesos de cada termo do documento d_j. E como $w_{i,q}$ é o peso associado com o par termo-consulta (k_i, q), a aplicação do tf-idf$_{t,d}$ vira tf-idf$_{k_i,q}$ para os pesos associados à consulta q (BAEZA-YATES; RIBEIRO-NETO, 2011, p. 77–78).

Logo, tanto o documento d_j quanto uma consulta q feita pelo usuário são representados como vetores t-dimensionais como ilustrado na Figura 3, posteriormente modulados pelos pesos associados.

Figura 3 – O cosseno de θ é adotado como Pontuação$(\vec{d}_j, \vec{q})$.



Fonte: Baseado na figura disponível em Baeza-Yates e Ribeiro-Neto (2011, p. 78).

A avaliação do grau de similaridade entre esses vetores, sendo esta correlação, ou pontuação que o documento d_j vai receber para a consulta q, quantificada pelo cosseno do ângulo entre esses dois vetores, conforme demonstra a Equação 2.6 (BAEZA-YATES; RIBEIRO-NETO, 2011, p. 78).

$$\text{Pontuação\_COS}(\vec{d}_j, \vec{q}) = \frac{\vec{d}_j \bullet \vec{q}}{\left\|\vec{d}_j\right\| \times \left\|\vec{q}\right\|} \tag{2.6}$$

Baeza-Yates e Ribeiro-Neto (2011, p. 79) apontam como vantagens do modelo vetorial:

- **Melhora da qualidade dos resultados** devido à esquemática de pesos utilizada;
- Capacidade de **correspondência parcial** possibilita que documentos *próximos* da consulta sejam retornados;
- **Organização dos resultados pelo grau de similaridade com a consulta** devido à fórmula de ranqueamento pelo cosseno dos vetores;

- **Normalização dos tamanhos dos documentos** embutida.

Os mesmos autores ainda apontam que a principal desvantagem é a presunção de que os termos indexados são mutualmente independentes, e ressaltam que a tarefa de considerar a dependência dos termos é algo bastante complicado e geram resultados ruins caso não seja feita de modo adequado.

2.1.2.2 Modelo probabilístico

Existem diferentes modelos probabilísticos para RI, como por exemplo o modelo linguístico, o modelo de divergência do aleatório (*divergence from randomness*), e o *Framework* de Relevância Probabilística (também chamado de modelo clássico) (ZHAI; MASSUNG, 2016, p. 87), que é o mais conhecido por ter dado origem à função BM25 para ranqueamento de documentos (ROBERTSON, 2010, p. 334–335) (ZHAI; MASSUNG, 2016, p. 111), a qual é nosso foco de discussão. BMxx foi a notação utilizada para nomear a funções de ranqueamento do *software* Okapi de RI, desenvolvido na *City, University of London* na década de 90. BM se refere a *Best-matching*, em português melhor correspondência, e a função popularizada como BM25 foi apresentada por Robertson et al. (1996).

Como já abordado, os sistemas de RI tentam determinar o quão bem os documentos satisfazem as necessidades de informação do usuário, porém existe um grau de incerteza sobre quais documentos tem conteúdo relevante. Partindo desse princípio, da existência da incerteza na relevância dos documentos, temos os modelos probabilísticos, que baseiam-se na teoria probabilística que fundamenta o raciocínio em cima de incertezas (MANNING; RAGHAVAN; SCHüTZE, 2008, p. 201).

Nos modelos probabilísticos, a função de ranqueamento é definida baseada na probabilidade que um documento d é relevante para uma consulta q, ou estatisticamente $P(R = 1|d, q)$ onde $R \in \{0, 1\}$ é uma variável binária aleatória que denota a relevância (ZHAI; MASSUNG, 2016, p. 111–112), sendo esta a base para o princípio do ranqueamento probabilístico (PRP) (MANNING; RAGHAVAN; SCHüTZE, 2008, p. 203).

O PRP é a base para o chamado *Framework* de Relevância Probabilística o qual, por sua vez, deu origem ao modelo de independência binária (BIM), aos modelos de *feedback* de relevância, ao nosso caso de interesse, o BM25, e também a diversas variações do BM25 (ROBERTSON, 2010, p. 333).

O BIM faz a implementação do princípio de ranqueamento probabilístico com documentos e consultas sendo representados por vetores binários de incidência dos termos, podendo assim ser comparado ao modelo Booleano (MANNING; RAGHAVAN; SCHüTZE, 2008, p. 204). A evolução das implementações dos modelos probabilísticos clássicos levou à função de recuperação conhecida como **Okapi BM25**, ou simplesmente BM25, que integra os conceitos do modelo vetorial apresentado na Subsubseção 2.1.2.1, como frequência dos

termos, normalização de tamanho, e correspondência parcial. Devido à sua similaridade com os modelos vetoriais, alguns autores, como Zhai e Massung (2016, p. 111), apresentam a função BM25 função junto à dos modelos vetoriais.

Toda teoria probabilística do PRP, que fundamenta os modelos do Framework de Relevância Probabilística, inclusive o BM25, é extensa e portanto ela não será desenvolvida detalhadamente aqui. Abaixo, na Equação 2.7, está demonstrada a fórmula do Okapi BM25 similar às fórmulas apresentadas por Manning, Raghavan e Schütze (2008, p. 213–215) e por Zhai e Massung (2016, p. 107–108).

$$\text{Pontuação\_BM25}(d_j, q) = \sum_{t \in q} \text{idf}_t \cdot \frac{(k_1 + 1)\text{tf}_{t,d}}{k_1((1-b) + b \times (\frac{L_d}{L_{\text{avg}}})) + \text{tf}_{t,d}} \tag{2.7}$$

A fórmula apresentada possui termos já apresentados, como os pesos $\text{tf}_{t,d}$ e idf_t, ambos apresentados na Subseção 2.1.2. O termo $\text{tf}_{t,d}$ tem o mesmo significado aqui, é uma contagem do número de ocorrências do termo t no documento d, no entanto o termo idf_t na fórmula original do BM25 é conhecido como o peso de Robertson/Spark Jones e pode ser simplificada para a fórmula apresentada na Equação 2.8 (ROBERTSON, 2010, p. 347–349).

$$\text{idf}_t = \log \frac{N - \text{df}_t + \frac{1}{2}}{\text{df}_t + \frac{1}{2}}. \tag{2.8}$$

É necessário dizer que este termo idf_t pode ser incrementado por outras considerações feitas pelo modelo probabilístico, e também simplificado pela demanda da implementação a ser feita. Alguns autores como Manning, Raghavan e Schütze (2008, p. 214) e Zhai e Massung (2016, p. 108) apresentam este termo como o inverso da frequência do termo t, da mesma forma já demonstrada na Equação 2.1.

A fórmula do BM25 tem dois parâmetros de refinamento, os termos b e k_1. O parâmetro b ($0 \leq b \leq 1$) é utilizado para controlar o grau de normalização por tamanho de documento, onde $b = 1$ significa uma escalonamento completo e $b = 0$ corresponde a não realizá-lo. Esse refinamento toma como ponto de referência o tamanho médio dos documentos na coleção inteira, L_{avg}, em relação ao tamanho do documento d, representado por L_d.

Já o termo k_1 ($0 \leq k_1 \leq \infty$) calibra o efeito da correção de frequência dos termos presente na fórmula, $k_1 = 0$ significa que não haverá consideração da frequência dos termos, e valores elevados fazem com a fórmula se aproxime da utilização pura do $\text{tf-idf}_{t,d}$ vista na Equação 2.2 (MANNING; RAGHAVAN; SCHüTZE, 2008, p. 214). Para consultas extensas, com grande número de termos, Manning, Raghavan e Schütze (2008, p. 214) afirmam que pode ser adicionado à fórmula um segundo fator k_3 que possibilita calibrar o escalonamento de frequência dos termos da consulta, chegando assim à fórmula da Equação 2.9 com a

simplificação do termo idf_t pela versão simples da Equação 2.1.

$$\text{Pontuação\_BM25}(d_j, q) = \sum_{t \in q} \left[\log \frac{N}{\text{df}_t} \right] \cdot \frac{(k_1 + 1)\text{tf}_{t,d}}{k_1((1-b) + b \times (\frac{L_d}{L_{\text{avg}}})) + \text{tf}_{t,d}} \cdot \frac{(k_3 + 1)\text{tf}_{t,q}}{k_3 + \text{tf}_{t,q}} \tag{2.9}$$

Os parâmetros de refinamento devem ser definidos para otimizar o desempenho na recuperação em uma coleção de teste, com a utilização de métodos exaustivos manuais para a busca dos melhores valores, ou com métodos de otimização de funções como por exemplo o *grid search*. Não sendo possível realizar a etapa de otimização, experimentos mostram que é razoável definir tanto k_1 como k_3 para valores entre 1.2 e 2, e definir $b = 0.75$ ou valores entre 0.5 e 0.8 (MANNING; RAGHAVAN; SCHüTZE, 2008, p. 215) (ROBERTSON, 2010, p. 360–361).

2.2 MINERAÇÃO DE TEXTO

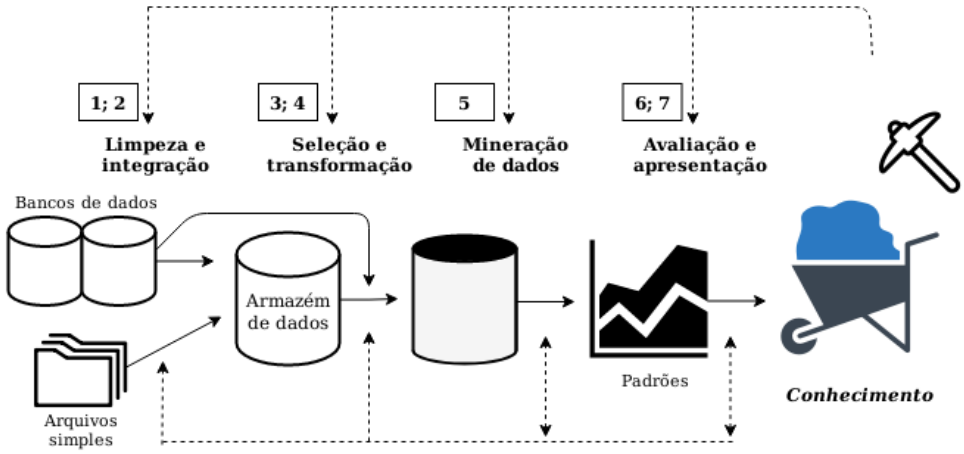
A Mineração de Textos (MT) é definida como o processo de extrair conhecimento implícito de dados textuais (JO, 2018; FELDMAN; SANGER, 2006) e por isso é às vezes tratada como *knowledge discovery in text* (livremente traduzido para descoberta de conhecimento em texto) (KODRATOFF, 1999; FELDMAN; DAGAN, 1995), sendo análogo ao termo *knowledge discovery in data* (KDD) que se refere à Mineração de Dados, ramo da Inteligência Artificial que dá suporte à MT. Apesar de haver um uso sinônimo entre Mineração de Dados e KDD, alguns autores tratam a Mineração de Dados como somente uma parte desse processo de descoberta de conhecimento (HAN; KAMBER; PEI, 2011, p. 6), sendo este um processo iterativo, conforme ilustrado na Figura 4, composto pelas seguintes fases (ou etapas) segundo Han, Kamber e Pei (2011, p. 6–7):

1. **Limpeza dos dados**: remoção de ruído e dados inconsistentes;
2. **Integração dos dados**: combinação de múltiplas fontes de dados;
3. **Seleção dos dados**: dados relevantes para a tarefa de análise são recuperados do banco de dados;
4. **Transformação dos dados**: dados são transformados e consolidados em formas apropriadas para mineração sendo realizadas, por exemplo, ações de agregação ou resumo;
5. **Mineração dos dados**: métodos inteligentes são aplicados para extrair padrões de dados;
6. **Avaliação de padrões**: são identificados os padrões que realmente tão interessantes para representar o conhecimento baseado em medidas de nível de interesse;
7. **Apresentação do conhecimento**: o conhecimento minerado é apresando aos usuários por meio de técnicas de visualização e representação de conhecimento.

É importante notar essas 7 etapas de desenvolvimento de Mineração de Dados para abordamos a definição de MT, pois esta deriva muitas técnicas desenvolvidas na pesquisa do campo de Mineração de Dados para seu campo de aplicação, logo sistemas baseados em ambas áreas vão apresentar similaridades estruturais (FELDMAN; SANGER, 2006, p. 1).

A Mineração de Dados assume que os dados, que vão ser tratados durante seu processo, já foram armazenados em um formato estruturado, logo a maior parte de seu pré-processamento vai estar ligado às etapas 1 e 2 do processo de KDD citado, as de limpeza e integração dos dados (FELDMAN; SANGER, 2006, p. 1). Já na MT, como os dados de trabalho são textos, sendo texto configurado como dados desestruturados que consistem de *strings* (palavras) organizadas de forma coerente e sendo pertencentes a uma linguagem natural

Figura 4 – Mineração de dados como uma fase do processo de descoberta do conhecimento (KDD).



Fonte: Figura baseada na original de Han, Kamber e Pei (2011, p. 7).

(JO, 2018, p. 1), temos que as operações de pré-processamento vão estar mais focadas em etapas adicionais, prévias às citadas para o processo de KDD, sendo estas novas direcionadas à identificação e extração de *features* (atributos) representativas para documentos escritos em linguagem natural, transformando os dados não estruturados, que estão armazenados em coleções de documentos, em um formato mais explicitamente estruturado (FELDMAN; SANGER, 2006, p. 1).

As operações de pré-processamento para MT utilizam de várias técnicas adaptadas dos campos de Recuperação de Informação, extração de informação e linguística computacional para transformar as coleções de documentos desestruturados em dados intermediários cuidadosamente estruturados (FELDMAN; SANGER, 2006, p. 2–3). Essa estrutura intermediária é definida por um modelo representacional dos documentos de texto composto por um conjunto de atributos, sendo sempre preferidos os modelos com menor número de variáveis significativas para a representação (FELDMAN; SANGER, 2006, p. 4).

Apesar da Mineração de Texto utilizar de técnicas da Recuperação de Informação, ambos são campos independentes com objetivos diferentes conforme ressalta Jo (2018, p. 4, tradução nossa) (apresentados de modo resumido na Tabela 3):

> A saída da mineração de dados é o conhecimento implícito que é necessário diretamente para a tomada de decisões, enquanto a saída da recuperação é composta por alguns dos itens de dados que são relevantes para a consulta dada. Por exemplo, no domínio de preços de ações, a previsão dos preços futuros de ações é uma tarefa típica da mineração de dados, enquanto que obter alguns dos preços de ações passadas e atuais é tarefa da recuperação de informação. Observe que a certeza perfeita nunca existe na mineração de dados, em comparação com a recuperação. A computação mais avançada para obter conhecimento dos dados brutos, chamada de síntese, é necessária para executar as tarefas de mineração de dados.

Tabela 3 – Mineração de Dados versus Recuperação de Informação (em específico para objetos de texto a comparação vale para Mineração de Texto versus Recuperação de Texto).

	Mineração	**Recuperação**
Saída	Conhecimento	Itens relevantes
Exemplo	Valores previstos	Valores anteriores ou atuais
Certeza	Probabilística	Nítida
Síntese	Necessária	Opcional

Fonte: Jo (2018, p. 4).

A definição dos atributos para MT busca tirar proveito dos mais variados elementos presentes em um documento escrito em linguagem natural, no entanto é necessário um cuidado pois existe um grande número de palavras, frases e outros artefatos que podem comprometer o desempenho de um sistema de Mineração de Texto ou tornar a tarefa infactível (FELDMAN; SANGER, 2006, p. 4), por isso a necessidade de identificar os melhores atributos, que trazem mais informação sobre o texto. Nesse ponto que a MT pode se auxiliar de técnicas de RI para incrementar seu grupo de atributos, sendo alguns, como por exemplo o BM25 para RI ranqueada, utilizadas em competições de identificação de perfil de autores (WEREN, 2014; WEREN; MOREIRA; OLIVEIRA, 2014; WEREN et al., 2014).

2.2.1 Corpus

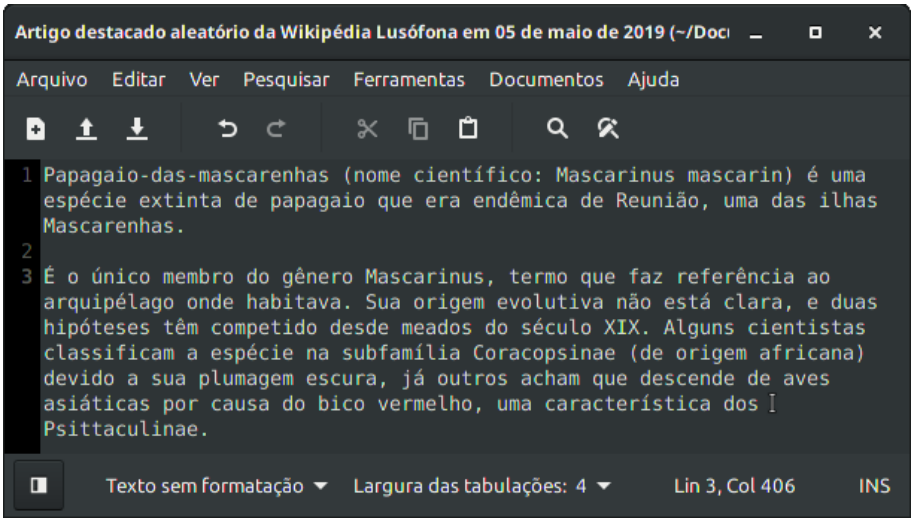
Vários formatos de texto são utilizados para o processamento computacional de texto, segundo Jo (2018, p. 6) os mais populares são os distribuídos pelo *software* de escritório proprietário MS Office, como o arquivo padrão do MS Word com extensão “doc”, MS PowerPoint com extensão “ppt”, e o MS Excel com o “xls”, e ainda para transferência entre computadores o mais utilizado é o formato PDF (*Portable Document Format*). Como alternativa aos formatos proprietários do MS Office, foi aprovada a norma ISO/IEC 263000 em 8 de maio de 2006 que define o formato aberto ODF (*Open Document Format for Office Applications*). A partir da padronização feita com o ODF são definidas diversas extensões que passaram a ser suportadas por praticamente todos os *softwares* de escritório, como por exemplo a extensão “.odt” para documentos de texto, “.ods” para folhas de cálculo e “.odp” para apresentações.

O texto simples, ou texto sem formatação, é o formato mais elementar de texto que é feito por um editor de texto, exemplificado na Figura 5 por um arquivo aberto no editor xed[1]. Usualmente cada texto corresponde a único arquivo, sendo geralmente armazenado com a extensão “txt” nos sistemas operacionais Windows (JO, 2018, p. 6).

O XML (*Extensive Markup Language*) pode ser considerado como outro formato de texto conforme exibido na Figura 6. É o formato de documento *web* mais flexível e foi projetado com base no HTML, o XML 1.0 foi padronizado pela W3C (*World Wide Web Consortium*)

[1] xed é um editor de texto leve e pequeno feito para o X-Apps (LINUX MINT, 2019).

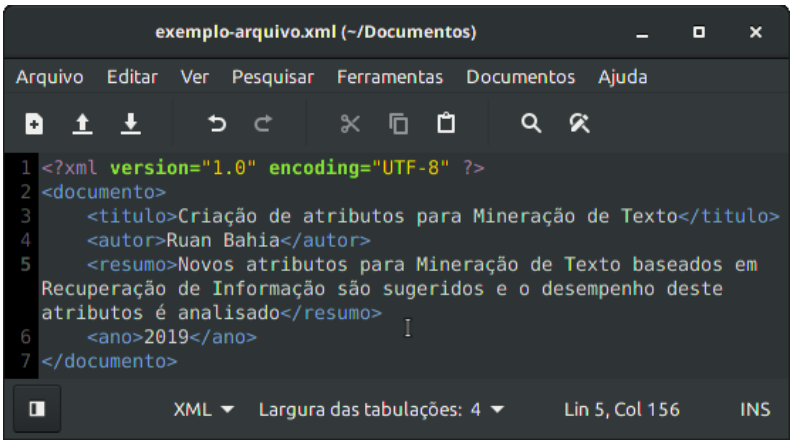
Figura 5 – Arquivo de texto simples, texto não formatado, aberto para edição no xed.



Fonte: O autor, conteúdo do texto obtido de Wikipédia (2019).

em sua primeira versão em 1998, sendo a quinta edição a mais recente publicada em novembro de 2008 (W3C, 2008). Atualmente o XML é utilizado como padrão para descrever itens de dados textuais de forma relacional, onde cada campo possui uma etiqueta de início e de fim, e o valor do campo fica entre estas etiquetas (JO, 2018, p. 6).

Figura 6 – Exemplo de documento XML aberto no editor xed.



Fonte: O autor.

Uma coleção de textos simples é chamada de corpus, geralmente sendo referenciado pelo diretório que contém os arquivos de texto, e excepcionalmente um único arquivo pode também corresponder a um corpus ao invés de ser somente um único arquivo de texto (JO, 2018, p. 6). Num conceito mais abrangente, Kwartler (2017, p. 9) considera como corpus qualquer corpo, ou conjunto, grande de texto organizado, assim um conjunto de arquivos de texto XML também é tratado como um corpus.

2.2.2 Tarefas de Mineração de Dados

As tarefas de Mineração de Dados podem ser divididas em duas categorias, segundo Tan, Steinbach e Kumar (2014, p. 7) e Han, Kamber e Pei (2011, p. 15):

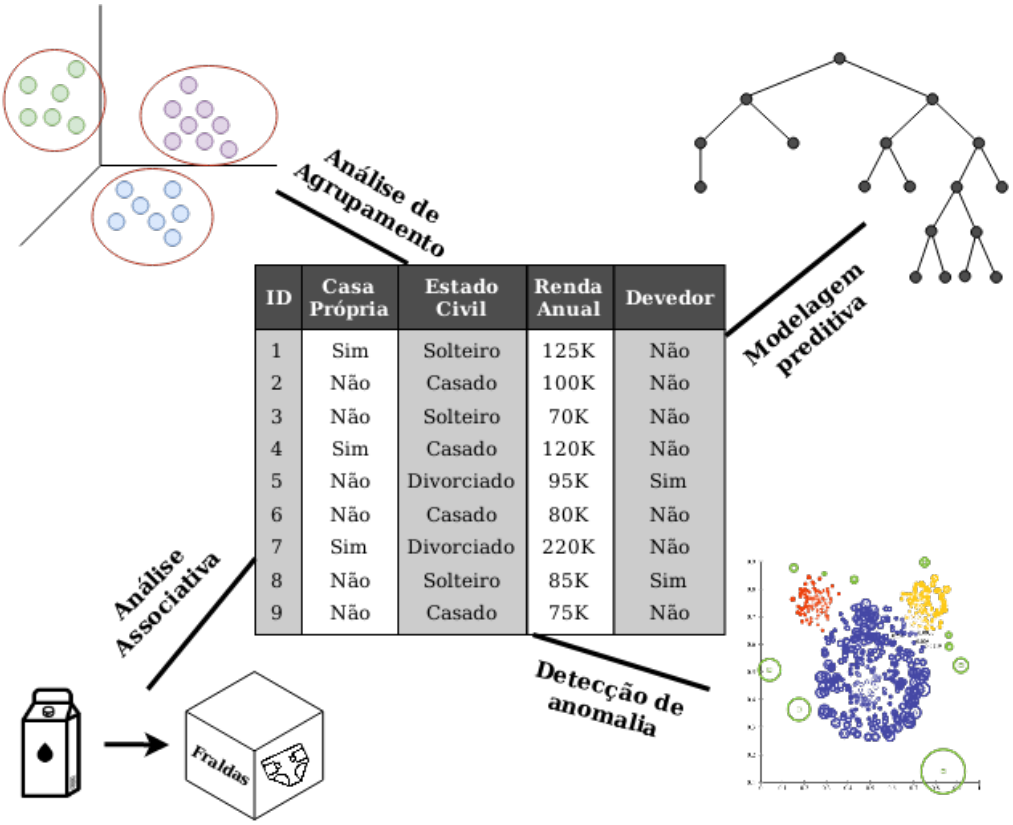
- **Descritivas**: tem como objetivo caracterizar propriedades dos dados no conjunto objetivo por meio da derivação de padrões que resumem as relações nos dados; e
- **Preditivas**: realizam uma indução nos dados presentes objetivando prever valores de um atributo em particular, sendo este atributo a ser previsto chamado de objetivo.

Quanto às tarefas em si, a mineração de dados possui tarefas de agrupamento, associação, classificação, descrição, detecção de anomalias, e de regressão. As quatro principais estão ilustradas na Figura 7. As tarefas de classificação e de regressão entram na categoria de tarefas preditivas, e são às vezes referidas como tarefas de modelagem preditiva. Ainda, tarefas de classificação trabalham com atributos objetivos discretos, enquanto que tarefas de regressão tem seus atributos objetivos como variáveis contínuas (TAN; STEINBACH; KUMAR, 2014, p. 7–8), e ambas focam na análise de conjuntos de dados rotulados, os chamados de conjuntos de treinamento (HAN; KAMBER; PEI, 2011, p. 19). Na Mineração de Textos uma tarefa de classificação normalmente recebe o nome de **categorização de texto** (TURCHI; MAMMONE; CRISTIANINI, 2009, p. 6) (FELDMAN; SANGER, 2006, p. 61).

Segundo os autores Han, Kamber e Pei (2011, p. 15–21), Tan, Steinbach e Kumar (2014, p. 8–11) e Larose e Larose (2014, p. 8–14) as demais tarefas da mineração de dados são descritivas e podem ser caracterizadas como segue:

- **Agrupamento**: consiste numa análise dos dados sem consulta à rótulos de classe, podendo estes até não estarem presentes, e este tipo de tarefa é utilizada para gerar rótulos de classes para grupos de dados. Na literatura é mais comum se referir ao termo em língua inglesa, *clustering*;
- **Associação**: são descobertos relacionamentos mais frequentes entre os atributos, utilizada para descoberta de padrões, geralmente por meio de regras de implicação;
- **Descrição**: os dados são associados com conceitos, consistindo da caracterização dos dados, no qual é feito o resumo das características gerais de uma classe objetivo; ou da discriminação dos dados, na qual é feita a comparação, contraste, dos atributos de uma classe objetivo com um conjunto de outras classes;
- **Detecção de anomalias**: também chamado de detecção de dados discrepantes (*outliers*), foca na identificação de exemplos dos dados que possuem características significativamente diferentes do restante. Muitos métodos de mineração de dados descartam os *outliers* por considerarem estes como ruídos nos dados.

Figura 7 – As quatro principais tarefas da mineração de dados.



Fonte: Figura baseada na original de Tan, Steinbach e Kumar (2014, p. 7).

2.2.2.1 Classificação binária

O problema da classificação consiste na aprendizagem da estrutura dos exemplos do conjunto de dados, os quais estão classificados em grupos chamados de categorias ou classes (AGGARWAL, 2015, p. 285). A entrada consiste de uma coleção de registros, conhecidos como exemplos, cada um caracterizado pela tupla $(\mathbf{x}, y)$ onde $\mathbf{x}$ é um conjunto de atributos e y é o atributo chamado de rótulo da classe. Tan, Steinbach e Kumar (2014, p. 146, tradução nossa) dizem que "classificação é a tarefa de aprender uma função objetivo f que mapeia cada conjunto de atributos x a um rótulo de classe predefinido y".

O aprendizado da função objetivo é feito com um modelo, chamado de modelo de treinamento, pois este é baseado no conjunto de treinamento. Então, este modelo de treinamento é utilizado para prever a classe de exemplos não vistos anteriormente, isso quer dizer que estes exemplos não fazem parte do conjunto de treinamento. Segundo Aggarwal (2015, p. 286, tradução nossa), então, a maior parte dos algoritmos de classificação possui duas fases:

1. **Fase de treinamento**: nesta fase um modelo de treinamento é construído à parte das instâncias de treinamento. Isto pode ser entendido, intuitivamente, como um resumo do modelo matemático dos grupos rotulados no conjunto de dados de

treinamento;

2. **Fase de teste**: nesta fase o modelo de treinamento é utilizado para determinar os rótulos de classe (ou identificadores de grupos) de uma ou mais instâncias não vistas.

A tarefa de classificação é também chamada de aprendizado supervisionado porque exemplos dos dados são utilizados para aprender as estruturas de agrupamento das classes (AGGARWAL, 2015, p. 285). Técnicas de classificação são mais apropriadas para dados com categorias binárias ou nominais, não sendo indicados para categorias com significado ordinal, pois a tarefa de classificação não considera a ordem implícita entre as categorias (TAN; STEINBACH; KUMAR, 2014, p. 147). A tarefa de categorização dos dados pode ser com rótulo único ou com múltiplos rótulos, na categorização com múltiplos rótulos as categorias se sobrepõem, e já na categorização com rótulo único cada exemplo de dado pertence a exatamente uma categoria (FELDMAN; SANGER, 2006, p. 67) (TAN; STEINBACH; KUMAR, 2014, p. 306).

A classificação binária é um caso especial da categorização de rótulo único na qual quantidade de categorias é dois, sendo este o caso mais simples das tarefas de classificação, pois nele só existem duas possibilidades de rótulo de classe e cada objeto de dados pertence, exclusivamente, a uma das classes (FELDMAN; SANGER, 2006, p. 67) (JO, 2018, p. 81). Um exemplo clássico de categorização binária de texto é um analisador de *spam*[2] para e-mails, nele só existem duas possibilidades de classificação para dado e-mail, ou **é *spam*** ou **não é *spam***.

2.2.3 Engenharia de atributos

O processo para extrair atributos passíveis de serem utilizados por classificadores da mineração de dados é chamado por Zheng e Casari (2018) de *feature engineering*, que em uma tradução livre tentando preservar o contexto pode ser chamado de *manejo de atributos*. Num conceito mais generalista, Dong e Liu (2018, p. 3, tradução nossa) define *feature engineering* como uma área que abrange "os tópicos de transformação de atributos, geração de atributos, extração de atributos, seleção de atributos, análise e avaliação de atributos, metodologias de manejo generalista e automatizado de atributos, e aplicações do manejo de atributos", sendo esta a definição de engenharia de atributos no processo de KDD.

Cada um dos tópicos da engenharia de atributos é definido por Dong e Liu (2018, p. 3) como segue:

- **Transformação de atributos**: construção de novos atributos a partir de atributos existentes, frequentemente feito por meio de mapeamentos matemáticos;

[2] Conteúdo indesejado, tratado como lixo eletrônico.

- **Geração de atributos**: também chamado de criação de atributos ou de extração de atributos, é a geração de novos atributos que não são resultados de transformações. Por exemplo, atributos derivados de padrões e de técnicas de domínio específico se encontram na categoria de geração de atributos;

- **Seleção de atributos**: selecionar um pequeno conjunto de atributos dentre um montante, com objetivo de reduzir o conjunto de atributos para tornar a tarefa de classificação computacionalmente viável. A seleção de atributos também pode visar medição de atributos úteis e a melhora do desempenho por meio da escolha do melhor subconjunto;

- **Metodologias de manejo generalista e automatizado de atributos**: abordagens para geração automática de uma grande quantidade de atributos e seleção de um subconjunto dentre os atributos gerados;

- **Aplicações do manejo de atributos**: a utilização das técnicas de engenharia de atributos para resolver outras tarefas de análise de dados.

2.2.3.1 Atributos comuns para documentos

Ao atacar um problema de MT em uma coleção de documentos é de suma importância a definição dos atributos a serem utilizados na tarefa, e mesmo em pequenas coleções o problema da alta dimensionalidade dos atributos aparece. Feldman e Sanger (2006, p. 4, tradução nossa) exemplificam que até mesmo "em uma coleção extremamente pequena de 15 mil documentos selecionados de notícias da Reuters, podem ser identificadas mais de 25 mil raízes de palavras não triviais".

A alta dimensionalidade dos atributos pode aparecer como um empecilho computacional para a realização da MT, por fazer com que os cálculos necessários sejam muito mais demorados e em alguns casos tornando-se um problema computacionalmente inviável. Então, a identificação e escolha de um modelo representacional, um bom conjunto de atributos, para representar os documento é algo de suma importância (FELDMAN; SANGER, 2006, p. 4).

Os tipos de atributos mais utilizados para representar documentos, segundo Feldman e Sanger (2006, p. 5–7):

- **Caracteres**: são as letras, números, e caracteres especiais presentes nos documentos utilizados para construir a semântica do mesmo;

- **Palavras**: são símbolos linguísticos nativos do espaço de atributos de um documento. Atributos a nível de palavra geralmente são palavras únicas selecionadas de um documento nativo, e também é possível que sejam utilizados todas as palavras de um documento para representá-lo;

- **Termos**: são palavras únicas e frases com mais de uma palavra selecionadas de um corpus de documentos nativos por meio de técnicas específicas para extração de termos;

- **Conceitos**: são os atributos gerados de modo manual, baseado em regras, ou via categorização híbrida feita no pré-processamento. Por exemplo, um documento sobre carros esportivos pode não incluir a palavra "automóvel", mas este conceito pode ser utilizado para identificar e representar esse documento.

2.2.3.2 Criação de atributos

A criação de atributos também é referida como geração de atributos, e consiste em derivar novos conjuntos de atributos que capturem, de forma mais efetiva, a informação carregada pelos dados (TAN; STEINBACH; KUMAR, 2014, p. 55).

Algumas metodologias de criação de atributos são apresentadas por Tan, Steinbach e Kumar (2014, p. 55–57):

- **Extração de atributos**: a criação de um novo conjunto de atributos a partir dos dados brutos. Por exemplo, no domínio da classificação de imagens os dados brutos são os *pixels*, que não são apropriados para tarefas de classificação, no entanto os dados podem ser processados para fornecer informações de presença de contornos, que podem então ser utilizados por técnicas de classificação para identificar rostos;

- **Mapeamento dos dados para um novo espaço**: mudar completamente a visualização dos dados. Considerando uma série temporal, por exemplo, os dados podem ser transformados na informação de frequência dessa stem-seérie por meio da transformada de Fourier;

- **Construção de atributos**: os atributos presentes nos dados possuem a informação necessária para o processo de mineração, mas não estão na forma adequada, assim novos atributos podem ser construídos na forma adequada. Um exemplo é numa tarefa de classificação que os dados contém informação de volume e massa de itens, pode ser construído o atributo de densidade.

2.2.4 Medidas de avaliação de classificadores

Tarefas de classificação (categorização) de texto produzem modelos de classificadores, os quais tem efetividade avaliada em termos de suas medidas de **precisão** e de **revocação**[3] (BERRY; KOGAN, 2010, p. 48), e também há uma preocupação grande com a **acurácia** do classificador (ZHAI; MASSUNG, 2016, p. 313). As medidas de avaliação de um modelo de

[3] As medidas na língua inglesa são chamadas de *precision* e *recall*.

classificador devem ser calculadas com base na aplicação do modelo num conjunto de teste, consistindo de tuplas não utilizadas para treinar o modelo (HAN; KAMBER; PEI, 2011, p. 364).

Antes de descrever as medidas, é necessário abordar o conceito de exemplos positivos (ou tuplas positivas) e de exemplos negativos (ou tuplas negativas) nos dados a serem classificados. Os exemplos positivos são aqueles da classe de maior interesse, e os exemplos negativos são todos o restante (HAN; KAMBER; PEI, 2011, p. 364). Considerando uma classificação binária, pode-se ter os exemplos positivos referentes a *spam* = *sim* e os negativos *spam* = *não*. Seguindo, como descrito por (HAN; KAMBER; PEI, 2011, p. 364–365), ao construir um modelo de classificador treinado com exemplos deste tipo, e então ao testá-lo em um conjunto com a classe objetivo conhecida, serão previstas as classes desse conjunto. Sendo P' o número de exemplos classificados positivamente, e N' o número de exemplos classificados negativamente. Cada exemplo pode então ter a classe prevista comparada com as classes objetivos já conhecidas, gerando a matriz de confusão da Tabela 4 como um sumário das previsões corretas e incorretas.

Tabela 4 – Matriz de confusão para uma tarefa de classificação binária, exibida com os totais para exemplos positivos e negativos.

		Classe prevista		
		sim	*não*	Total
Classe real	*sim*	*TP*	*FN*	P
	não	*FP*	*TN*	N
	Total	P'	N'	$P+N$

Fonte: Tabela disponível em Han, Kamber e Pei (2011, p. 366).

A matriz de confusão tem tamanho $n \times n$, onde n é o número de classes (sempre maior ou igual a 2). Cada entrada $MC_{i,j}$ da matriz corresponde ao número de exemplos da classe i rotulado pelo classificador como pertecente à classe j. A terminologia utiliza na Tabela 4, caso específico para classificação binária, possui o seguinte significado:

- **Verdadeiros positivos** (*TP*): referente ao número de exemplos positivos rotulados corretamente;
- **Verdadeiros negativos** (*TN*): número de exemplos negativos rotulados corretamente;
- **Falsos positivos** (*FP*): número de exemplos positivos rotulados incorretamente, a classe real na verdade era negativa, no entanto o modelo de classificação rotulou como a classe positiva;
- **Falsos negativos** (*FN*): número de exemplos negativos rotulados incorretamente, por exemplo a rotulação de um exemplo como *spam* = *não* sendo que sua classe real na verdade é *spam* = *sim*.

A adição de totais à matriz de confusão é comum (HAN; KAMBER; PEI, 2011, p. 366), na matriz apresentada na Tabela 4 estão apresentados os totais P' e N', respectivamente o número total de exemplos rotulados como positivos ($TP+FP$) e o número de exemplos rotulados como negativos ($TN+FN$). P é o número de exemplos com classe real positiva ($TP+FN$) e N é o número de exemplos com classe real negativa ($FP+TN$).

A razão entre o número de classificações verdadeiras positivas e o número classificações positivas dá a precisão (p) do classificador, conforme mostra a Equação 2.10, e segundo Han, Kamber e Pei (2011, p. 368, tradução nossa) "precisão pode ser pensada como uma medida de exatidão".

A revocação (r) é dada pela Equação 2.11, que é a razão entre o número o número de classificações verdadeiras positivas e o número de exemplos positivos da classe real, e também é chamada de sensitividade (HAN; KAMBER; PEI, 2011, p. 364–365). Ainda, segundo Han, Kamber e Pei (2011, p. 368, tradução nossa) "revocação é uma medida de completude".

$$p = \frac{TP}{TP+FP} = \frac{TP}{P'} \qquad (2.10) \qquad\qquad r = \frac{TP}{TP+FN} = \frac{TP}{P} \qquad (2.11)$$

As medidas de precisão e revocação podem ser combinadas em uma única medida chamada de F (também conhecida como F-*score* ou pontuação F_1), que é a média harmônica entre elas, definida como disposto na Equação 2.12.

O nível geral de reconhecimento do classificador é medido pela acurácia (acc), que é definida pela Equação 2.13, representando a razão de exemplos do conjunto de teste que foram classificados corretamente.

$$F = \frac{2 \times p \times r}{p+r} \qquad (2.12) \qquad\qquad acc = \frac{TP+TN}{P+N} \qquad (2.13)$$

Apesar das medidas F, precisão, revocação e acurácia serem bastante úteis na medida de efetividade de classificadores, Berry e Kogan (2010, p. 48) afirmam que elas não consideram o custo da classificação errônea em problemas com classes desbalanceadas, e então sugere um critério de acurácia ponderada e similarmente Han, Kamber e Pei (2011, p. 369) sugerem o uso da medida F_β definida como descrito na Equação 2.14.

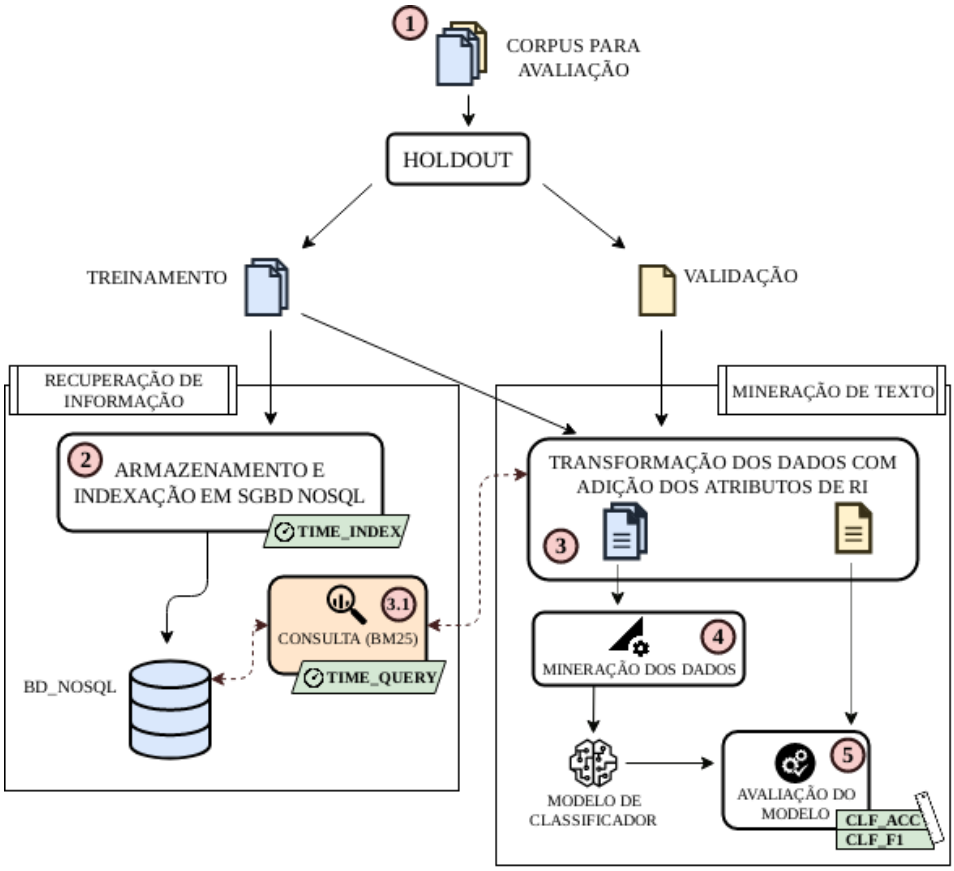
$$F_\beta = \frac{(1+\beta^2) \times p \times r}{\beta^2 \times p + r} \qquad (2.14)$$

As medidas de avaliação de classificadores possibilitam a comparação de diferentes modelos de classificadores e assim é possível efetuar a seleção do modelo, que é a escolha do melhor classificador dentre os modelos gerados para um problema específico (HAN; KAMBER; PEI, 2011, p. 364).

3 MATERIAIS E MÉTODOS

Este capítulo apresenta o modo de avaliação do desempenho dos atributos, gerados a partir da função de ranqueamento BM25, em Mineração de Texto. A metodologia proposta está ilustrada na Figura 8 e o processo é detalhado a seguir.

Figura 8 – Metodologia proposta para avaliação de desempenho, em verde estão as variáveis mensuráveis sugeridas.



Fonte: O autor.

Inicialmente é necessário selecionar os **(1) Corpus para Avaliação** que servem para efetuar as avaliações de desempenho, e, caso esses corpus ainda não estejam segmentados em exemplos para treinamento e exemplos para validação, é feita essa separação com o método de *holdout*[1] de $\frac{2}{3}$ para treinamento e $\frac{1}{3}$ para validação.

Em seguida, o conjunto de treinamento passa pela fase de RI, aonde é feito o **(2) Armazenamento e Indexação em SGBD NoSQL** por meio das ferramentas apresentadas na

[1] Particionamento aleatório do dados em dois conjuntos independentes, geralmente chamados de treinamento e teste. O conjunto de treinamento é utilizado para derivar o modelo e o de teste para estimar a sua acurácia. (HAN; KAMBER; PEI, 2011, p. 370).

Seção 3.2, sendo mensurado o tempo necessário para concluir essa operação em cada um dos Sistemas Gerenciadores de Banco de Dados (SGBD) NoSQL selecionados.

O processo de Mineração de Texto pode ser segmentado em 7 etapas, conforme ressaltado anteriormente na Seção 2.2. Na metodologia proposta aqui a fase de MT assume que as 3 etapas iniciais de limpeza, integração, e seleção dos dados, já foram realizadas no banco de dados de teste, assim os conjuntos divididos em treinamento e validação passam diretamente pela etapa seguinte que é a **(3) Transformação dos dados com adição dos atributos de RI**. Nesta etapa, os banco de dados NoSQL indexados recebem **(3.1) Consultas** com a utilização das funções de ranqueamento baseadas no BM25 que cada ferramenta implementa, sendo mensurado o tempo para efetuar cada consulta e gerar os novos atributos sugeridos adiante na Seção 3.3.

O processo de **(4) Mineração dos Dados** é feito por meio da réplica de soluções dos corpus para avaliação selecionados, que possuem seus códigos-fonte disponíveis, sendo então reproduzida cada solução (a) sem nenhuma alteração, e (b) com adição dos novos atributos sugeridos. Esse processo gera dois modelos de classificador, o primeiro criado sem atributos de RI e o segundo com esses atributos. A etapa de **(5) Avaliação do Modelo** permite que para cada modelo sejam geradas medidas da literatura de MT, detalhadas logo mais na Subseção 3.4.2, a partir do teste com o conjunto de validação, possibilitando a comparação do ganho de desempenho de classificador com/sem atributos de RI.

3.1 CORPUS PARA AVALIAÇÃO

Foram definidos dois corpus de competições promovidas pela PAN, sigla da organização que se originou do *International Workshop on Plagiarism Analysis, Authorship Identification, and Near-Duplicate Detection* em 2007 (PAN'07 Workshop, 2007), sendo estes descritos a seguir:

- **DB_AUTHORPROF - *Author Profiling - PAN @ CLEF 2018*:** Uma tarefa da competição *CLEF 2018* promovida pela PAN na classe de análise de autoria, a qual foca na identificação de gênero no Twitter em três linguagens distintas, inglês, espanhol, e árabe (PAN, 2018).

  Os dados consistem de 100 *tweets*[2] e 10 imagens para cada autor, sendo que o conjunto de treinamento possui (a) 3000 autores em inglês, (b) 3000 autores em espanhol, e (c) 1500 autores em árabe, e o conjunto de validação possui (a) 1900 autores em inglês, (b) 2200 autores em espanhol, e (c) 1000 autores em árabe (RANGEL et al., 2018).

  Segundo Rangel et al. (2018) esse banco de dados da *CLEF 2018* com um total de 12600 autores é um subconjunto da tarefa de *Author Profiling* da *CLEF 2017* e

[2] Termo utilizado para designar as publicações feitas na rede social do Twitter.

eles foram classificados em dois passos, (i) automaticamente com a ajuda de um dicionário de nomes próprios, e (ii) manualmente verificando a foto, descrição e outros elementos de cada perfil (RANGEL et al., 2017).

- **DB_HYPARTISAN - *Hyperpartisan News Detection - PAN @ SemEval 2019 Task 4*:** Esta tarefa da competição *SemEval 2019* promovida pela PAN consiste em, dada uma notícia, avaliar se esta segue uma argumentação hiperpartidária, que significa verificar se ela possui fidelidade cega, preconceituosa, ou irracional a um partido, grupo, causa, ou pessoa (PAN, 2019b). Portanto pode-se dizer que se trata de um problema de classificação binária, a classe real é a presença ou ausência de hiperpartidarismo em cada exemplo.

  Os dados consistem em artigos de notícias divididos em múltiplos arquivos onde cada arquivo com parte inicial "articles-" consiste em uma notícia e a classificação real da notícia está presente em um arquivo com inicial "ground-truth-". Além disso os dados estão divididos em duas partes, a primeira composta de 750 mil artigos está classificada pelo enviesamento geral do editor, onde metade está classificada como hiperpartidária e a outra metade não. Dessa parte, 80% (600 mil artigos) estão no conjunto de treinamento e 20% (150 mil) estão no conjunto de validação, sendo que nenhum editor de artigos do conjunto de treinamento se repete no conjunto de validação (KIESEL et al., 2018).

  A segunda parte dos dados foi rotulada através de *crowdsourcing*[3]. Essa parte dos dados contêm um total de 645 artigos, sendo estes apenas artigos para os quais existia um consenso entre os trabalhadores de crowdsourcing. Destes, 238 (37%) são hiperpartidários e 407 (63%) não são (KIESEL et al., 2018).

Os dois corpus selecionados possuem soluções de participantes nas competições da PAN que tem seu código fonte aberto e disponível em repositórios online. Das equipes participantes na competição do corpus DB_AUTHORPROF, foram localizadas em repositórios de código aberto no GitHub[4] as supostas soluções apresentadas na Tabela 5, ordenadas pela classificação disponível na parte de resultados da página da competição PAN (2018).

Quanto à competição referente ao corpus DB_HYPARTISAN, foram encontrados os repositórios disponíveis na ordem de classificação da página da competição (PAN, 2019b), dispostos abaixo na Tabela 6.

As soluções com código fonte encontradas foram analisadas para garantir que já não fazem uso da criação de atributos baseados em RI. Então, assim foram selecionadas as soluções 1_bertha e 4_tom do corpus DB_HYPARTISAN, e a solução 2_daneshvar18 do corpus

[3] Obter entrada para uma tarefa ou projeto específico contando com os serviços de um número de pessoas, pagas ou não, tipicamente através da Internet.

[4] Plataforma de hospedagem de código-fonte com controle de versão usando o Git.

Tabela 5 – Soluções encontradas de participantes da competição DB_AUTHORPROF.

Posição	Equipe	Repositório de código no site <https://github.com/>
2	daneshvar18	/SamanDaneshvar/pan18ap/
4	laporte18	/arthur-sultan-info/PAN2018-AP/
12	gouravdas18	/brajagopalcse/PAN2018/
16	schaetti18	/nschaetti/PAN18-Author-Profiling/
21	raiyani18	/kraiyani/author-profiling-pan-clef-2018
23	karlgreen18	/jussikarlgren/pan18/

Fonte: Classificações obtidas de PAN (2018), e repositórios encontrados pelo autor.

Tabela 6 – Soluções encontradas de participantes da competição DB_HYPERPARTISAN.

Posição	Equipe	Repositório de código no site <https://github.com/>
1	bertha-von-suttner	/GateNLP/semeval2019-hyperpartisan-bertha-von-suttner/
4	tom-jumbo-grumbo	/chialun-yeh/SemEval2019/
10	clint-buchanan	/hmc-cs159-fall2018/final-project-team-mvp-10000/
13	paparazzo	/ngannlt/semeval2019-hyperpartisan-paparazzo/
17	spider-jerusalem	/amal994/hyperpartisan-detection-task/
19	doris-martin	/ixa-ehu/ixa-pipe-doc/

Fonte: (PAN, 2019a).

DB_AUTHORPROF para execução e verificação das pontuações obtidas na competição e obtenção das medidas de desempenho de classificador (ver a Subseção 3.4.2). Em seguida, é executado o processo completo da metodologia apresentada na Figura 8, para mensurar o desempenho computacional das ferramentas de armazenamento e indexação, e o desempenho do classificador com os atributos de RI.

3.2 ARMAZENAMENTO E INDEXAÇÃO

Para armazenar e indexar os corpus das tarefas de Mineração de Textos em bancos de dados, possibilitando o cálculo da função BM25 para que sejam gerados os atributos sugeridos na Seção 3.3 para cada exemplo, são utilizadas as seguintes tecnologias que fazem implementações do BM25:

- **TOOL_ELASTIC**: Elasticsearch 7.2 é o mecanismo distribuído de análise e busca baseado no Apache Lucene[5], desenvolvido em Java, e possui código aberto sob diversas licenças sendo a principal a Licença Apache[6] (ELASTIC, 2019a; ELASTIC, 2019b).

  O Elasticsearch possui diversas APIs que possibilitam sua integração fácil com variadas linguagens de programação (ELASTIC, 2019a), e tenta deixar todas suas funcionalidades disponíveis via sua API JSON pois internamente é no formato

[5] Biblioteca de software livre e de código aberto para ferramentas de buscas em texto, escrita originalmente em Java (LUCENE, 2019).

[6] Licença de software livre permissiva de autoria da Apache Software Foundation (ASF) (NEW MEDIA RIGHTS, 2015).

JSON[7] que o Elasticsearch guarda os dados. Suporta também requisições GET em tempo real, o que o torna apropriado para armazenamento como um banco de dados NoSQL (Peter - Karussel, 2011; YAZıCı, 2018).

Dentre suas funções de indexação, o Elasticsearch possui um módulo de similaridade (*similarity module*) que é responsável pela implantação de funções para ranqueamento de documentos. Esse módulo realiza uma implementação da função BM25 como sua função padrão para cálculo de similaridade sobre o nome de *BM25 similarity* (ELASTIC, 2019c).

- **TOOL_ARANGO**: ArangoDB v3.4.6 é um banco de dados multi-modelo nativo, desenvolvido principalmente em C++ com extensões em JavaScript, que possui código-fonte aberto e possibilita modelos de dados flexíveis, tanto para documentos, gráficos, e valores-chave (ARANGODB INC, 2019b; ARANGODB INC, 2019a).

  Utiliza da linguagem de consulta AQL (*ArangoDB Query Language*) para recuperar e modificar dados, que, por meio das *views*[8] do tipo arangosearch, introduz uma camada de integração com a biblioteca IResearch[9]. Assim, por meio da AQL integrada ao IResearch, o ArangoDB fornece funções de ordenação de documentos mediante uma consulta, e, dentre elas, a função BM25() faz uma implementação do algoritmo da função de ranqueamento BM25 (ARANGODB INC, 2019c).

- **TOOL_ZETTAIR**: Zettair v0.9.3 é um mecanismo de busca de código-fonte aberto escrito na linguagem C, projetado para ser compacto e pesquisar rapidamente em texto, desenvolvido pelo Grupo de Mecanismos de Busca do Instituto Real de Tecnologia de Melbourne em 2009 (RMIT University, 2009a). O Zettair permite que coleções no formato HTML ou TREC sejam indexadas para busca, e possui como característica principal a habilidade de lidar com grandes quantidades de texto, ele constrói índices invertidos por meio da análise feita no seu modo de construção de índex (RMIT University, 2009b).

  Os arquivos de índices invertidos construídos pelo Zettair podem então ser consultados utilizando duas medidas diferentes (RMIT University, 2009c):

  - **–cosine**: Uma implementação da medida da similaridade do cosseno vista na Subsubseção 2.1.2.1; e
  - **–okapi**: Implementação da função BM25, também chamada de Okapi BM25, como apresentada na Subsubseção 2.1.2.2.

[7] *JavaScript Object Notation*, formato de objeto utilizado pela linguagem de programação JavaScript.

[8] "Uma *view* (visão) em terminologia SQL é uma única tabela que é derivada de outras tabelas [···] é considerada uma tabela virtual" (ELMASRI; NAVATHE, 2010, p. 88).

[9] Biblioteca de mecanismo de busca orientada a documentos, multiplataforma, e de alto desempenho, escrita inteiramente em C++, com o foco em uma conectividade de diferentes modelos de ranqueamento/similaridade (IRESEARCH, 2019).

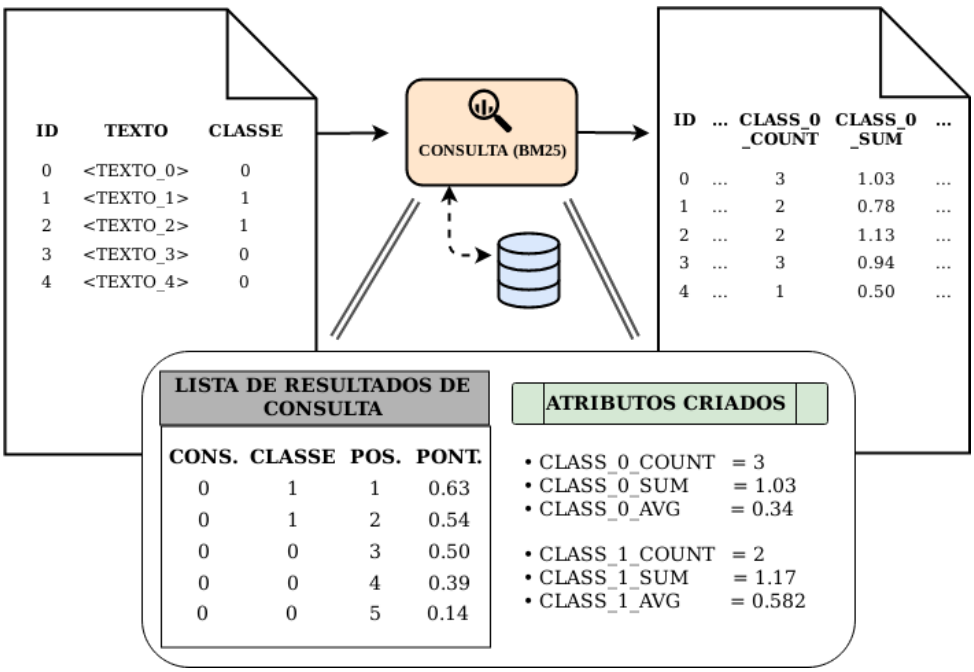
Vale então ressaltar que o Zettair se trata somente de um mecanismo de busca, enquanto que tanto o ArangoDB quanto o Elasticsearch são bem mais ricos em funcionalidades, estando assim bem mais próximos de SGBDs convencionais.

3.3 ATRIBUTOS DE RI SUGERIDOS

Os atributos baseados em Recuperação de Informação sugeridos para a presente pesquisa são fundamentados na investigação de Weren (2014), derivada de seus trabalhos anteriores (WEREN; MOREIRA; OLIVEIRA, 2014; WEREN et al., 2014), os quais mostram resultados positivos da utilização de atributos derivados de RI numa tarefa de Mineração de Texto. Nesses trabalhos, para identificação de perfis de autoria, são utilizados grupos de atributos derivados de RI, os quais representam a seguinte hipótese: **os autores de um mesmo grupo de gênero ou idade tendem a usar termos semelhantes, e que a distribuição desses termos difere entre os grupos** (WEREN, 2014, p. 20).

Esse mesmo pressuposto, da investigação feita por Weren (2014), é assumido aqui, sendo generalizado para outras classes de identificação de autoria, como por exemplo que autores de artigos hiperpartidários tendem a usar termos semelhantes, e a distribuição desses termos difere de autores não hiperpartidários.

Figura 9 – Metodologia de consulta aos BD para geração dos atributos sugeridos, exemplificação da lista de resultados para uma única consulta.



Fonte: O autor.

Cada exemplo do corpus a ser classificado é usado como consulta ao banco de dados NoSQL indexado, e cada uma destas consultas tem como resultado uma lista dos *top-k*

exemplos presentes no banco de dados, ordenada por pontuação de similaridade da função BM25, sendo esta pontuação calculada pela ferramenta de armazenamento indexação. Esse procedimento está representado na Figura 9.

O cálculo dos novos atributos é feito a partir da lista retornada dos *top-k* exemplos, e são utilizadas as três funções de agregação sugeridas por Weren (2014, p. 21–23): (a) média, (b) soma, e (c) contagem, levando em conta a classe binária da tarefa de classificação. Assim, são gerados os atributos apresentados na Tabela 7.

Tabela 7 – Atributos derivados de RI sugeridos.

Exemplo / Agregação	Não faz parte da classe da tarefa	Faz parte da classe da tarefa
Média aritmética das pontuações	**CLASS_0_BM25_AVG**	**CLASS_1_BM25_AVG**
Contagem do número de resultados	**CLASS_0_BM25_COUNT**	**CLASS_1_BM25_COUNT**
Soma das pontuações	**CLASS_0_BM25_SUM**	**CLASS_1_BM25_SUM**

Fonte: O autor.

A definição do valor de *top-k* altera diretamente os atributos gerados. Contudo, a influência desta alteração não foi caso de estudo de Weren ao propor seus atributos de RI, e também nos seus estudos falta a informação de qual o valor de *top-k* utilizado. A ferramenta de indexação utilizada nos estudos de Weren é o Zettair, e, quando não especificado explicitamente, o padrão de resposta a uma consulta é retornar 20 resultados. Então, utilizar *top-k* = 20 pode ser uma referência, no entanto o autor do presente estudo propõe-se a utilizar *top-k* = 100 para gerar os atributos de RI.

3.4 MEDIDAS PARA AVALIAÇÃO DE DESEMPENHO

São avaliados (a) o desempenho das ferramentas utilizadas para indexação e consulta, por meio da avaliação temporal do desempenho computacional, e também (b) o ganho de desempenho propiciado pelo uso dos atributos de RI em termos das medidas de classificadores de Mineração de Texto.

3.4.1 Desempenho computacional das ferramentas

Para avaliar o desempenho das ferramentas de armazenamento e indexação são utilizadas duas medidas temporais:

- **TIME_INDEX**: Tempo de execução para indexar o conjunto de treinamento de cada um dos corpus para avaliação elencados na Seção 3.1. Dadas as 3 diferentes ferra-

mentas de indexação e os 2 corpus selecionados, são computadas 6 TIME_INDEX para comparação;

- **TIME_QUERY**: Tempo para consulta de cada exemplo do conjunto de teste e geração dos atributos sugeridos na Seção 3.3 para o item específico. Dadas as 3 soluções para os dois corpus selecionados, e dadas as 3 ferramentas de indexação, 9 TIME_QUERY são computadas.

Essas variáveis serão computadas para cada uma das 3 ferramentas selecionadas sendo executadas no mesmo sistema computacional a fim de oferecer maior confiança aos números obtidos. O sistema computacional utilizado para efetuar o experimento é especificado no capítulo de resultados.

3.4.2 Desempenho de classificador

O ganho de desempenho propiciado pelos atributos de RI criados é mensurado por medidas de desempenho de classificadores da literatura de mineração de dados, as mesmas também utilizadas na Mineração de Texto. Nesse estudo serão computadas as seguintes medidas:

- **CLF_ACC**: Acurácia do classificador no conjunto de validação.
- **CLF_F1**: F_1-score do classificador no conjunto de validação.

A acurácia de um classificador, conforme apresentada na Subseção 2.2.4 definida pela Equação 2.13, é a razão entre o número de exemplos classificados corretamente, e o número total de exemplos, no caso em questão, a variável **CLF_ACC** a ser computada é referente ao conjunto de validação. O F_1-score é uma medida de classificador que engloba a precisão e revocação (ou sensitividade), e conforme apresentado na Subseção 2.2.4, é uma boa medida da efetividade de classificadores em termos comparação pois engloba mais informação do que a acurácia. É calculado conforme a definição dada pela Equação 2.12 da Subseção 2.2.4.

4 RESULTADOS

Este capítulo apresenta os resultados obtidos neste estudo investigativo do desempenho de atributos de RI em classificadores de Mineração de Texto.

Nas subseções a seguir primeiro é abordada a configuração experimental utilizada para realizar o estudo, e logo em seguida é apresentada uma visão geral das soluções selecionadas dos corpus DB_AUTHORPROF e DB_HYPERPARTISAN, na qual o pré-processamento realizado e os classificadores utilizados em cada uma das soluções são descritos brevemente. Por fim, são apresentados os resultados mensurados por meio das medidas escolhidas para avaliação de desempenho, na Seção 4.3 estão as medidas de desempenho das ferramentas de armazenamento e indexação, e na Seção 4.4 são expostas as medidas de desempenho dos classificadores.

4.1 CONFIGURAÇÃO EXPERIMENTAL

Para programação e execução dos experimentos foi utilizado o sistema computacional disponível para o autor, com a configuração disposta na Tabela 8.

Tabela 8 – Configuração do computador de mesa utilizado neste estudo.

Processador	Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz
Memória RAM	32370 MB
Sistema Operacional	Linux Mint 19.2 Tina
Placa-mãe	Gigabyte Z97-D3H
Gráficos	Mesa DRI Intel(R) Haswell Desktop
Disco	HP SSD EX920 512GB

Fonte: O autor.

As ferramentas de armazenamento e indexação receberam os mesmos parâmetros de refinamento na configuração de suas funções BM25, sendo estes configurados em $k_1 = 1.2$, $k_3 = 0$ e $b = 0.75$. O parâmetro k_3 (escalonar frequência de termos na consulta) é exclusivo do Zettair, as demais ferramentas não implementam este parâmetro.

O ambiente de programação utilizado foi o VSCodium e a linguagem de programação utilizada foi o Python 3.7.5. Para permitir reprodutibilidade dos resultados obtidos, o gerador de número aleatórios (RNG) do ambiente do Python, assim como os RNGs das bibliotecas utilizadas pelas soluções, tiveram as suas sementes de geração fixadas.

Todos os códigos fontes dos scripts Python criados para este estudo estão disponíveis em um repositório de código online e público criado pelo autor, acessível pelo seguinte link: https://github.com/ruanmed/tcc-ii-ir-features-text-mining/. A seguir são tratadas as dependências de *software* para execução do estudo.

4.1.1 Dependências de *software*

Para execução dos scripts Python gerados por esse estudo é necessário Python >= 3.6 com as bibliotecas elasticsearch-py >= 7.0.5 e python-arango >= 5.2.1. Ambas bibliotecas se encontram disponíveis no PyPI (*Python Package Index*, ou índice de pacotes do Python em tradução livre).

Também é necessário possuir instâncias do Elasticsearch >= 7.4.0 e do ArangoDB >= 3.5.1 rodando e disponíveis na rede local do sistema, isso pode ser feito por meio da instalação dessas ferramentas no sistema local. O Zettair >= 0.94 também é necessário estar instalado no sistema, sendo possível executá-lo pelo terminal de comandos com o comando 'zet'.

A instalação das três ferramentas em sistemas Linux foi simplificada com a criação de um arquivo Docker que utiliza de imagens disponíveis online do ArangoDB e do Elasticsearch, reduzindo o número de comandos necessários para executar as ferramentas no sistema computacional. Detalhes de como utilizar esse arquivo Docker se encontram em: https://github.com/ruanmed/tcc-ii-ir-features-text-mining#docker-file.

Além dessas dependências, é necessário também instalar as bibliotecas específicas de cada uma das soluções selecionadas. Essas estão listadas nos arquivos README disponibilizados junto às soluções.

4.2 VISÃO GERAL DAS SOLUÇÕES SELECIONADAS E ADAPTAÇÕES FEITAS

As soluções selecionadas para reprodução e posterior adição dos atributos de RI utilizam de diferentes técnicas para pré-processamento dos corpus e também de classificadores diferentes, tornando necessário, em alguns casos, ajustes antes da adição dos atributos de RI.

Nas subseções a seguir são brevemente abordadas as soluções 1_bertha e 4_tom do corpus DB_HYPARTISAN, e a solução 2_daneshvar18 do corpus DB_AUTHORPROF.

4.2.1 Corpus DB_HYPERPARTISAN

O corpus DB_HYPERPARTISAN possui dois conjuntos de dados, o primeiro com 750 mil artigos classificados por enviesamento do editor, e o segundo com 645 artigos rotulados como hiperpartidários ou não com consenso entre avaliadores via *crowdsourcing*. O objetivo da competição que utilizou esse corpus era classificar artigos como hiperpartidários ou não, e assim os participantes puderam utilizar os dois conjuntos para fazer suas avaliações. No entanto, a utilização dos 750 mil artigos classificados por enviesamento de editor não foram úteis para ajudar na classificação, e das duas soluções selecionadas nenhuma delas utilizou esses 750 mil artigos nos seus modelos finais, pois utilizar a informação de editor em conjunto reduziu o desempenho de seus classificadores (JIANG et al., 2019, p. 840) pois segundo Yeh, Loni e Schuth (2019, p. 1067) o conjunto de dados classificado por enviesamento do editor possui muito ruído.

O conjunto de validação final da competição, com 628 artigos também classificados manualmente, não foi disponibilizado ao público, portanto para reprodução das soluções foi necessário dividir o conjunto de 645 artigos em treinamento e validação pelo método de *holdout*, ficando assim com 430 artigos para treinamento e 215 artigos para teste. Essa divisão foi feita de modo que as classes mantiveram-se com o mesmo balanceamento do conjunto original (37% dos artigos hiperpartidário e 63% não hiperpartidários).

Então, os 430 artigos foram indexados em todas as ferramentas de indexação permitindo a posterior geração dos atributos de RI. A indexação foi feita com auxílio da classe *IndexToolManager* criada pelo autor, que é abordada com detalhes na seção de desempenho das ferramentas de armazenamento e indexação.

4.2.1.1 Solução 1_bertha

A solução da equipe Bertha von Suttner (4_bertha) para essa tarefa da *SemEval 2019* consiste em uma abordagem de classificação com uma Rede Neural Convolucional (*Convolutional Neural Network*, CNN) e o pré-processamento dos documentos do corpus em uma representação destes por meio de uma Rede ESRC (*ELMo Sentence Representation Convolutional Network*), uma representação sugerida e nomeada pela equipe.

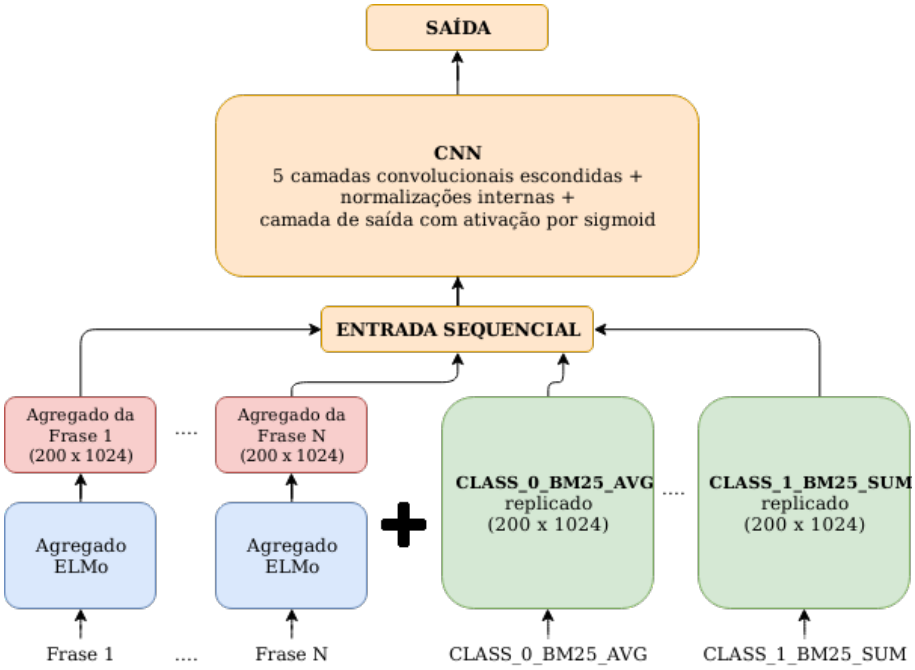
A representação na Rede ESRC é uma alternativa à representação usual dos documentos por termos (ou *tokens*). Essa representação consiste no cálculo da média dos agregados ELMo para os documentos, a nível de frase, e cada documento é representado como uma sequência desses agregados, que são vetores de tamanho de 1024 *floats*. Na implementação da solução cada documento é limitado a ocupar até 200 vetores de tamanho 1024, sendo então cada documento padronizado com forma 200 x 1024. ELMo é uma representação de palavras com contextualização profunda, que modela características complexas de uso das palavras, como sintaxe e semântica, e o uso dessas palavras em contextos linguísticos (PETERS et al., 2018).

A CNN do classificador consiste de 5 camadas convolucionais paralelas internamente que recebem como entrada os agregados ELMo, essas 5 camadas convolucionais se conectam a camada de saída da rede neural, com função de ativação sigmoid, mais detalhes da implementação da CNN podem ser vistos em Jiang et al. (2019).

Para adicionar os 6 atributos de RI à rede foram embutidos, ao final de cada representação de documento, 6 vetores de tamanho 1024, preenchidos pelo valor calculado de cada atributo para o respectivo documento, resultando em documentos com forma 206 x 1024. A adaptação da solução está ilustrada na Figura 10.

A Figura 10 é baseada na figura de representação de arquitetura do sistema apresentada no artigo de descrição da solução da equipe (JIANG et al., 2019), alguns detalhes da CNN são abstraídos e os atributos de RI adicionados estão representados em verde.

Figura 10 – Arquitetura de sistema da solução 1_bertha após adaptação.



Fonte: O autor.

4.2.1.2 Solução 4_tom

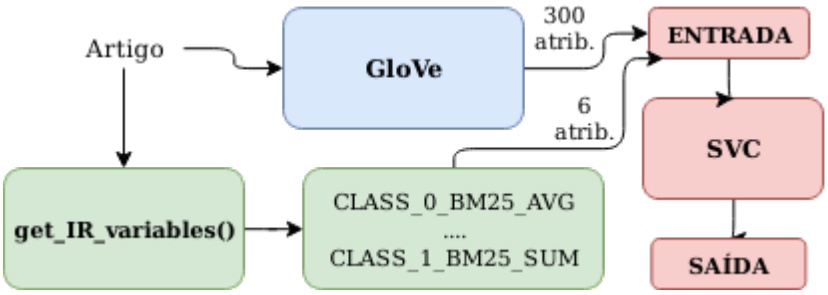
A solução da equipe Tom Jumbo-Grumbo (4_tom) utilizou de dois classificadores, um de Regressão Logística e outro de Máquinas de Vetor de Suporte (SVMs), sendo o utilizado no modelo final o classificador SVC (C-Support Vector Classification) da biblioteca sklearn para Python. Uma SVM funciona transformando os dados de treinamento para uma dimensão maior e então executa uma busca pelo melhor limite de decisão, chamado de hiperplano, para separar as classes (HAN; KAMBER; PEI, 2011, p. 408). Em especial, o classificador SVC da biblioteca sklearn possui um parâmetro de regularização de sua função de custo (parâmetro chamado de C) que deve ser estritamente positivo, o valor padrão é definido como $C = 1,0$, no entanto a solução da equipe, conforme disposto no código fonte disponível online, utilizou $C = 0,9$ após uma análise com diferentes valores.

No pré-processamento, a equipe utilizou três estratégias para converter os documentos em atributos para o classificador. A primeira foi de criar vetores baseados na frequência dos termos, descartando termos que apareçam em mais de 90% dos documentos e utilizando os 50 mil termos mais frequentes dos que sobrarem, guardando o valor tf-idf para cada termo respectivo ao documento. A segunda foi treinar um modelo PV-DM para gerar os atributos referentes a cada documento. E a terceira utilizou um agregado pré-treinado em textos da Wikipédia com o algoritmo GloVe, que, similarmente aos agregados ELMo, tentam também inferir o significado das palavras dos documentos.

A arquitetura com melhor resultado foi a utilização dos atributos GloVe com o classificador SVM da biblioteca sklearn. Os documentos são convertidos para os vetores GloVe

pré-treinados a partir se suas primeiras 1000 palavras, esses vetores GloVe possuem 300 dimensões por palavra, o que resultaria numa representação com forma de 1000 x 300, no entanto o vetor final que representa cada documento é a média dos vetores de palavras respectivo.

Figura 11 – Arquitetura do sistema da solução 4_tom após adaptação.



Fonte: O autor.

Para adicionar os 6 atributos de RI estes foram calculados para os respectivos documentos e embutidos no vetor GloVe do documento, resultando em documentos representados por 306 dimensões, ou 306 atributos. A arquitetura da solução adaptada está ilustrada na Figura 11.

4.2.2 Corpus DB_AUTHORPROF

O corpus DB_AUTHORPROF consiste de *tweets* de autores em três línguas diferentes, inglês, espanhol e árabe. Para avaliação dos atributos de RI o escopo de melhoria de desempenho foram considerados somente os classificadores das soluções para os autores de língua inglesa, e as imagens não foram consideradas.

O conjunto de treinamento de autores da língua inglesa consiste de 300 mil *tweets* de 3000 autores diferentes, e como o conjunto de validação final está disponível ao público, necessário somente solicitar o acesso, não foi necessário fazer o *holdout* do conjunto de treinamento. O conjunto de validação da língua inglesa consiste de 190 mil tweets de 1900 autores.

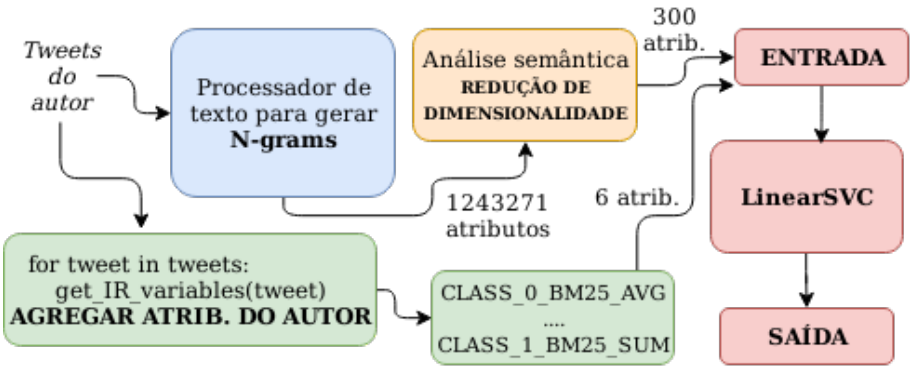
Os 300 mil *tweets* foram indexados em todas as ferramentas de indexação permitindo a geração dos atributos de RI posteriormente para cada *tweet* específico.

4.2.2.1 Solução 2_daneshvar18

A solução de Daneshvar e Inkpen (2018) utilizou um classificador SVM com diferentes tipos de n-grams de palavras e caracteres como atributos, e o melhor resultado obtido foi com a criação dos n-grams, posterior redução de dimensionalidade e então utilização do classificador LinearSVC da biblioteca sklearn para Python para criar o modelo de classificação.

A adaptação da solução para adicionar os atributos de RI poderia ser feita adicionando os atributos antes ou depois da redução de dimensionalidade utilizada pela equipe. Os documentos inicialmente são representados por vetores de 1243271 dimensões, após a diminuição da dimensionalidade cada documento é representado por um vetor de 300 dimensões. No entanto, uma peculiaridade da solução está em como é feita a classificação, os 100 *tweets* de cada autor são tratados como um único documento, pois o objetivo é classificar o gênero dos autores em homens ou mulheres. Como os *tweets* foram indexados individualmente, então para adicionar os atributos de RI para cada autor foi feita a geração dos 6 atributos para todos os tweets e então estes foram agregados por autor. A arquitetura adaptada do sistema pode ser vista na Figura 12.

Figura 12 – Arquitetura do sistema da solução 2_daneshvar18.



Fonte: O autor.

Por fim, os 6 atributos de RI agregados por autor foram embutidos ao final da representação dos documentos após a redução de dimensionalidade, resultando em documentos represetados por 306 atributos.

4.2.3 Resumo das soluções

Na Tabela 9 a seguir estão as principais características das soluções selecionadas.

Tabela 9 – Resumo dos detalhes das soluções selecionadas.

Solução	Pré-processamento	Núm. atrib.	Redução dim.	Núm. atrib. após	Classificador
1_bertha	ELMo	200 x 1024	Não	200 x 1024	CNN (5 cam. esc.)
4_tom	GloVe	300	Não	300	SVC
2_daneshvar18	N-gram palavras+caracteres	1243271	Sim	300	LinearSVC

Fonte: O autor.

4.3 DESEMPENHO DAS FERRAMENTAS DE ARMAZENAMENTO E INDEXAÇÃO

Para cálculo das medidas TIME_INDEX e TIME_QUERY, conforme sugeridas no Capítulo 3, foi empregada a linguagem de programação Python, utilizada para implementada uma classe chamada *IndexToolManager* que abstrai a indexação e o cálculo das variáveis de RI com as ferramentas.

Essa classe foi central para todo o estudo, a utilização dela concentrou as funções para acesso e manipulação, quando disponível, aos dados das ferramentas de armazenamento e indexação, centralização das diferentes bibliotecas do Python já disponíveis para interagir com o ArangoDB e com o Elasticsearch, python-arango e elasticsearch-py respectivamente. Trechos da implementação da classe *IndexToolManager* estão no Apêndice B.

4.3.1 Tempo de indexação

Para cálculo da medida TIME_INDEX foi criado um script Python nomeado `time_index.py`, o qual utilizou da classe *IndexToolManager* em duas funções feitas para executar a indexação dos banco de dados, DB_AUTHORPROF e DB_HYPERPARTISAN, nas 3 ferramentas, ArangoDB, Elasticsearch e Zettair. As função principal do script `time_index.py` possui nome *measure_TIME_INDEX*, e um trecho do código que implementa essa função pode ser visto a seguir no Código 1.

Código 1 – Função *measure_TIME_INDEX* do script `time_index.py`.

```
def measure_TIME_INDEX(normal=False, clean=False):
    mylogger.info('START OF TIME_INDEX MEASUREMENTS')
    exp_id = str(datetime.datetime.now())
    mylogger.info(exp_id)

    initial = time.time()
    if (clean):
        mylogger.info('CLEANING DATABASES')
        testTool = IndexToolManager(indexName=authorprof_db_name)
        testTool.clean_default()
        final = time.time()
        mylogger.info(f'CLEANING FINISHED: {final - initial}')

    tools = ['arango', 'elastic', 'zettair']
    dbs = ['authorprof', 'hyperpartisan', 'hyperpartisan_split_42']
    for db in dbs:
        mylogger.info('')
        mylogger.info('DB_' + db)
        for tool in tools:
            if (normal and tool != 'zettair'):
                index(idx_type='normal', db=db, tool=tool,
                      db_name=db, exp_id=exp_id)
            index(idx_type='bulk', db=db, tool=tool,
                  db_name=str(db+'_bulk'), exp_id=exp_id)
    mylogger.info(str(datetime.datetime.now()))
    mylogger.info('END OF TIME_INDEX MEASUREMENTS')
```

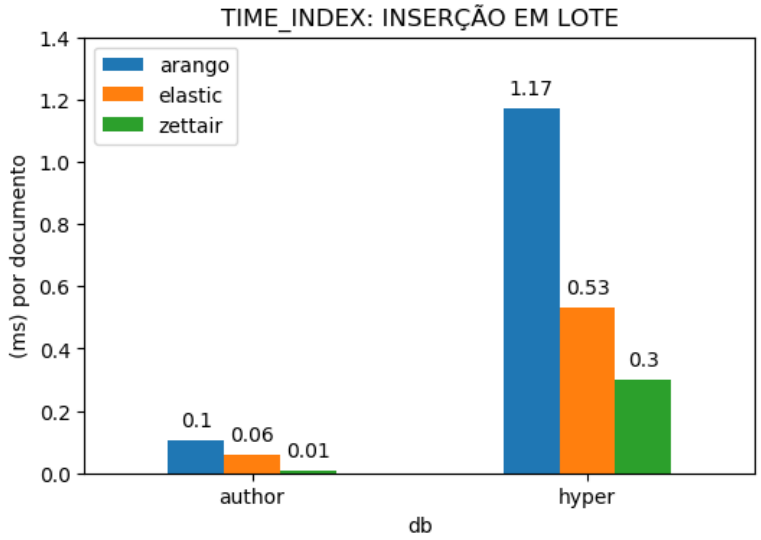
Fonte: O autor.

Essa função é responsável por efetuar uma chamada à função *index* com os parâmetros de tipo de indexação, corpus a ser indexado, e qual a ferramenta utilizada nessa indexação, além de uma identificação do experimento. A função *index*, que pode ser vista no Apêndice A, faz os devidos tratamentos para chamar os métodos da classe *IndexToolManager* responsáveis por fazer a indexação dos documentos do corpus especificado utilizando a ferramenta especificada, isso enquanto mensura o tempo que cada indexação levou e registra todas as operações num arquivo de texto.

Como as ferramentas ArangoDB e Elasticsearch se assemelham bastante a sistemas gerenciadores de bancos de dados completos, preparados, inclusive, para distribuição geográfica dos dados, há de ser citada essa grande diferença deles para o Zettair, este último que é somente um sistema para indexação em lotes e consulta local dos dados, não permitindo, por exemplo, adição de novos documentos em um índice. A ação de inserção unitária é um procedimento comum em SGBDs.

A operação de indexação foi executada de dois modos, em lote e unitária, sendo que o Zettair só permite a inserção em lote. Na Figura 13 pode ser visto o tempo mensurado, em milissegundos gastos por documento, para inserção em lote dos documentos dos corpus em cada ferramenta.

Figura 13 – Medidas de desempenho TIME_INDEX mensuradas para as ferramentas de armazenamento e indexação, com inserções feitas em lote.



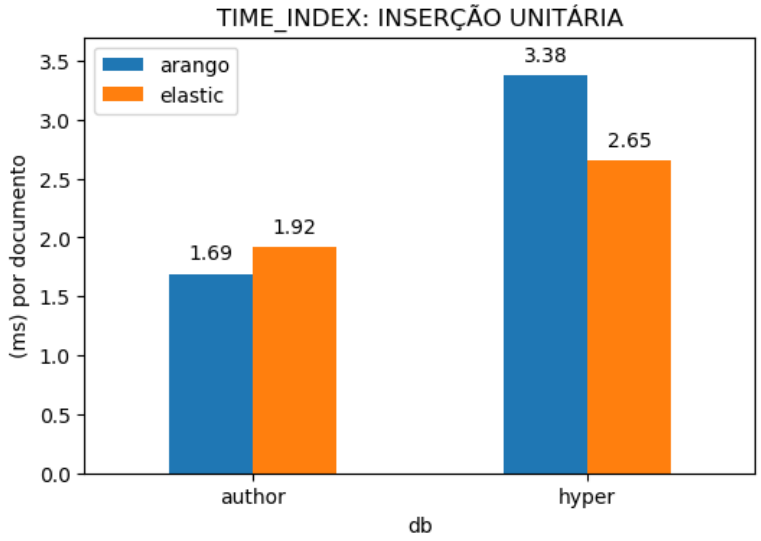
Fonte: O autor.

O Zettair é a ferramenta mais rápida para completar a indexação de ambos os corpus selecionados, levando somente 2,29 segundos para indexar os 300 mil documentos do corpus DB_AUTHORPROF, menos de 0,01 milissegundos por documento. Dentre os SGBDs avaliados, o Elasticsearch supera o ArangoDB para inserções em lote, gastando 17,61 segundos para indexar os 300 mil *tweets*, cerca de 0,06 milissegundos por documento. Nota-se ainda que a indexação do corpus DB_HYPERPARTISAN é mais lenta quando se consi-

dera o tempo por documento inserido, porque, apesar do corpus DB_AUTHORPROF ter um número bem maior de documentos, 300 mil em comparação com 645, cada artigo do DB_HYPERPARTISAN é bem mais extenso que qualquer dos *tweets* do DB_AUTHORPROF. A mesma relação de velocidade entre as ferramentas observada para as inserções feitas com os documentos do DB_AUTHORPROF se mantem para as inserções feitas com os documentos do DB_HYPERPARTISAN, o Zettair é o mais rápido, e, dentre os SGBDs, o Elasticsearch é mais rápido que o ArangoDB.

Para operação de inserção unitária, avaliada somente com o ArangoDB e Elasticsearch, foi levado em conta o tempo total para indexação de todos os documentos de cada corpus, inseridos em sequência, e posteriormente esses valores mensurados foram divididos pelo número de documentos. Na Figura 14 estão dispostos os tempos totais para indexação de todos os documentos dos corpus com ambas as ferramentas.

Figura 14 – Medidas de desempenho TIME_INDEX mensuradas para as ferramentas de armazenamento e indexação, com inserções unitárias.



Fonte: O autor.

O ArangoDB teve um melhor desempenho para as inserções dos documentos do DB_AUTHORPROF, com 1,69 milissegundos por documento, em contraste aos 1,92 milissegundos por documento do Elasticsearch. Porém, o Elasticsearch foi mais rápido para inserção dos documentos do DB_HYPERPARTISAN, com 2,65 milissegundos por documento contra 3,38 milissegundos por documento do ArangoDB.

Percebe-se então que o ArangoDB se saiu um pouco melhor que o Elasticsearch na inserção de grande quantidade de documentos pequenos em sequência, evidenciado pelos tempos de inserção unitária da Figura 14. E pode-se supor que isto é derivado de alguma sobrecarga na operação de inserção unitária do Elasticsearch, ou talvez no custo de recálculo do índice que varia com o tamanho dos documentos inseridos, porém com um custo inicial maior, já que para inserção unitária dos documentos do DB_HYPERPARTISAN, que são artigos

extensos, o Elasticsearch teve melhor desempenho.

No geral, o desempenho da indexação por inserção unitária fica abaixo da por inserção em lotes, como era esperado.

4.3.2 Tempo de consulta

Para medir o TIME_QUERY utilizando cada ferramenta durante a execução das soluções foi necessário adaptar os códigos das soluções para fazer o registro do tempo que cada consulta, e posterior geração dos atributos de RI, levou.

A fim de facilitar a geração dos atributos, foram implementados métodos na classe *IndexToolManager* para que: dado um texto, seja cálculo direto das variáveis de RI já com as ferramentas específicas, como por exemplo o método *arango_get_IR_variables*. A utilização desses métodos pode ser visto no trecho disposto a seguir do script Python `feat_GloVe.ipynb` da solução 4_tom adaptada.

Código 2 – Trecho da adaptação feita ao script `feat_GloVe.ipynb` da solução 4_tom.

```
...
ir_top_k = 100
...
testTool = IndexToolManager(
    indexName=str(hyperpartisan_db_name), top_k=ir_top_k)
...
def gloveVectorize(glove, text):
    ...
    for text_id, t in enumerate(text):
        ...
        if (exp_dict['add_ir_variables']):
            initial = time.time()
            ign_first = ignore_first_result and (text_id in id1)
            if (exp_dict['tool'] == 'arango'):
                ir_variables = testTool.arango_get_IR_variables(
                    t, 'true', ignore_first_result=ign_first)
            elif (exp_dict['tool'] == 'elastic'):
                ir_variables = testTool.elastic_get_IR_variables(
                    t, 'true', ignore_first_result=ign_first)
            elif (exp_dict['tool'] == 'zettair'):
                ir_variables = testTool.zettair_get_IR_variables(
                    t, 'true', interactive=False,
                    ignore_first_result=ign_first)
            final = time.time()
            time_query_list.append(float(final-initial))
            ir_vars_dict = [
                ir_variables['CLASS_0_BM25_AVG'],
                ir_variables['CLASS_0_BM25_COUNT'],
                ir_variables['CLASS_0_BM25_SUM'],
```

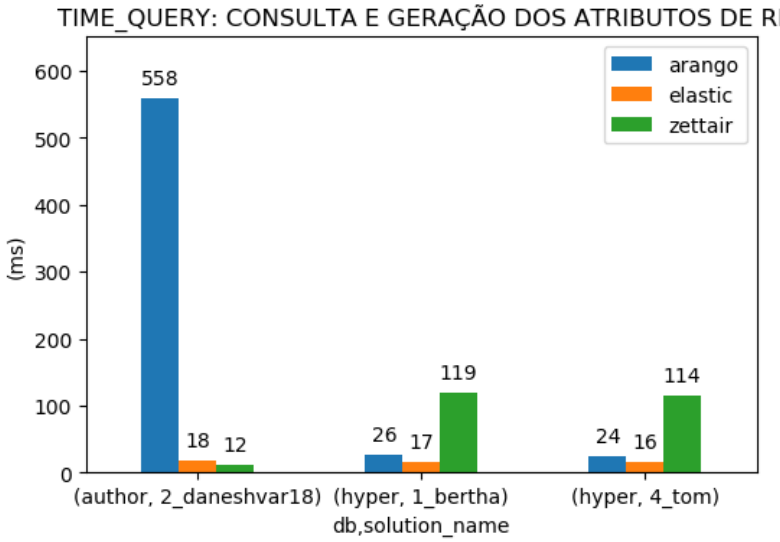
```
                ir_variables['CLASS_1_BM25_AVG'],
                ir_variables['CLASS_1_BM25_COUNT'],
                ir_variables['CLASS_1_BM25_SUM'],
            ]
        ...
```

Fonte: O autor.

Neste trecho, Código 2, *toolTest* é uma instância da classe *IndexToolManager* com *top-k* igual a 100. Esta classe que é responsável pelo cálculo das variáveis de RI, e o tempo de consulta leva em conta esse cálculo, como fica exposto pela posição das variáveis *initial* e *final*.

O mesmo tipo de adaptação feito para a solução 4_tom foi feito para as demais, e as medidas coletadas estão dispostas na Figura 15.

Figura 15 – Medidas de desempenho TIME_QUERY mensuradas para consulta e criação dos 6 atributos de RI sugeridos, utilizando as ferramentas de armazenamento e indexação.



Fonte: O autor.

Ao ver os resultados percebe-se uma discrepância entre os resultados do Zettair nos dois corpus, levando somente 12 milissegundos para o cálculo dos atributos de RI no corpus corpus DB_AUTHORPROF e mais de 110 milissegundos para as soluções do corpus DB_HYPERPARTISAN. Isso acontece pois o Zettair possui dois modos de operação para consultas, o (a) modo iterativo e o (b) modo de consulta única. Em ambos os modos o Zettair efetua uma etapa de inicialização, copiando o índice gerado para a memória, e para efetuar consultas em série o modo iterativo é o ideal, no entanto ele possui uma limitação não documentada de cerca de 2048 caracteres nas consultas, como no corpus DB_AUTHORPROF todos os documentos são pequenos, portanto menores que esse limite, as consultas para gerar os atributos de RI com o Zettair puderam ser executadas no modo iterativo. No entanto, para o corpus DB_HYPERPARTISAN isso não foi possível pois diversos documentos ultrapassam

essa limitação de 2048 caracteres, portanto, para este corpus, as consultas realizadas com o Zettair utilizaram o modo de consulta única.

A outra surpresa foi o ArangoDB, que levou 558 milissegundos para efetuar cada consulta no corpus DB_AUTHORPROF, o que indica que seu algoritmo para cálculo do BM25 tem um custo de desempenho proporcional ao número de documentos indexados no banco de dados, pois o corpus DB_AUTHORPROF possui 300 mil documentos indexados, e o DB_HYPERPARTISAN somente 430. Enquanto isso, o Elasticsearch teve desempenho similar para cálculo dos atributos de RI nos dois corpus, chegando no máximo a 18 milissegundos por consulta no corpus DB_AUTHORPROF.

O Zettair é a ferramenta mais rápida para a consulta e geração dos atributos de RI no corpus DB_AUTHORPROF, porém o Elasticsearch é a melhor ferramenta, em quesito geral de tempo de consulta, devido à limitação do tamanho de documento do modo iterativo do Zettair.

4.4 DESEMPENHO DOS CLASSIFICADORES COM ATRIBUTOS DE RI

O desempenho dos classificadores foi mensurado conforme as medidas CLF_ACC e CLF_F1 estabelecidas na Subseção 3.4.2 do Capítulo 3, esses valores foram coletados reproduzindo as soluções originais e então reproduzindo as soluções com as adaptações para incluir os atributos de RI gerados por cada uma das ferramentas.

A seguir são apresentados as medidas coletadas para as soluções dos corpus.

4.4.1 DB_HYPERPARTISAN

Antes da adaptação das soluções selecionadas foi necessário reproduzi-las, no entanto, como o conjunto de validação da competição de 628 não foi disponibilizado ao público e foi necessário fazer o *holdout* no conjunto de treinamento, não é possível comparar diretamente os resultados finais divulgados na página da competição com os obtidos. Pois, para submissão dos classificadores finais para a da competição, ambas as soluções 1_bertha e 4_tom treinaram seus classificadores com os 645 artigos e os resultados de acurácia e F_1-score disponíveis no ranking da competição foram calculados para as predições feitas nos 628 exemplos do conjunto de validação. Além disso, outro problema é que o script Python da solução 1_bertha disponibilizado online não possuia números aleatórios fixos, o que torna praticamente impossível reproduzir a mesma solução que foi submetida pela equipe para a competição.

Encontram-se na Tabela 10 as medidas divulgadas na página da competição *SemEval 2019* (PAN, 2019a) e as reproduções feitas com treinamento em 430 artigos e validação nos 215 restantes.

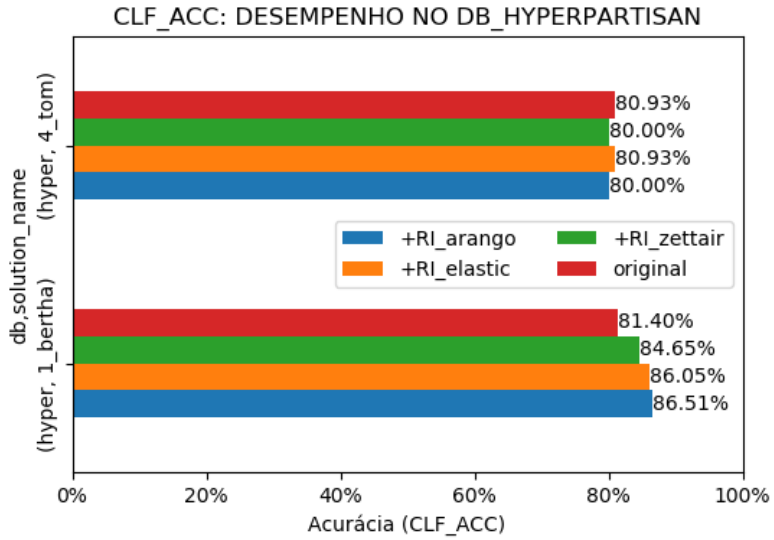
Tabela 10 – Comparação das medidas das soluções do corpus DB_HYPERPARTISAN divulgadas pelas competição e das reproduções.

Solução	**Acurácia**		**F_1-Score**	
	Competição	Reprodução	Competição	Reprodução
1_bertha	0,822	0,814	0,809	0,762
4_tom	0,806	0,809	0,790	0,707

Fonte: O autor.

Após essa validação que não há nada de muito errado com as soluções reproduzidas em comparação com os resultados divulgados pela competição, pois os resultados reproduzidos ficam dentro do esperado para um conjunto de treinamento reduzido, as soluções foram adaptadas para incluir os atributos de RI conforme o descrito nas seções anteriores. Na Figura 16 estão dispostas as medidas CLF_ACC para ambas as soluções.

Figura 16 – Desempenho CLF_ACC das soluções do corpus DB_HYPERPARTISAN.



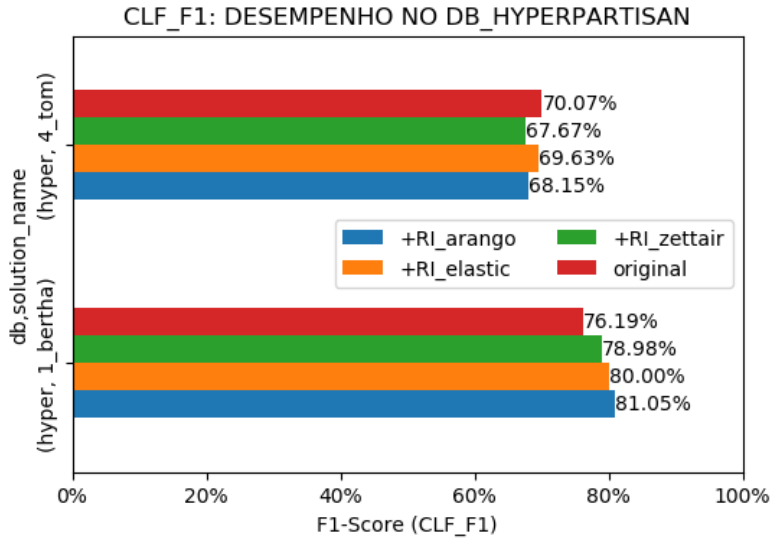
Fonte: O autor.

E na Figura 17 estão dispostas as medidas CLF_F1.

A adição dos atributos de RI proporciona ganho de acurácia e também de F_1-Score para a solução 1_bertha, alcançando valores maiores até mesmo que os finais da competição com acurácia de 0,8651 (86,51%) e F_1-Score de 0,8105 (81,05%) quando adicionados os atributos de RI gerados pelo ArangoDB. Já para a solução 4_tom inicialmente a adição dos atributos de RI não proporciona nenhum ganho de acurácia ou F_1-Score, pelo contrário, com todas as ferramentas o F_1-Score foi menor ao adicionar os atributos de RI, e somente com os atributos de RI do Elasticsearch que a acurácia se manteve a mesma, com o ArangoDB e com o Zettair houve também diminuição da acurácia do classificador da solução 4_tom.

Deve ser observado um detalhe importante da solução 4_tom, que é o classificador utilizado e o modo escolhido para definição dos parâmetros desse classificador pela equipe, que no caso foi o classificador SVC e na elaboração de sua solução a equipe avaliou diferentes

Figura 17 – Desempenho CLF_F1 das soluções do corpus DB_HYPERPARTISAN.

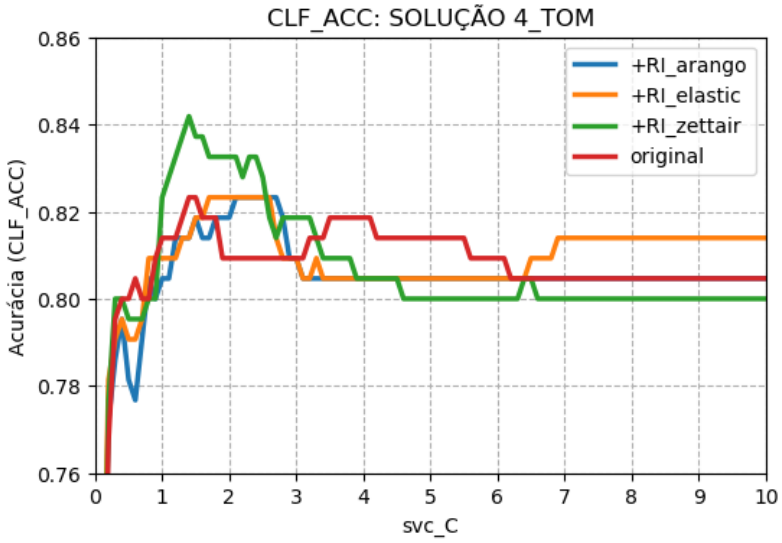


Fonte: O autor.

valores do parâmetro C do classificador e escolheu o valor de C com melhor resultado nesse teste, que foi $C = 0,9$. Como o conjunto de treinamento para submissão da equipe consistiu de todos os 645 artigos, e na reprodução feita para este estudo isso não pode ser feito devido à necessidade de um conjunto isolado de validação, foi feita a suposição que o melhor valor de C para a reprodução da solução seria diferente, pois o conjunto de treinamento da reprodução possui somente 430 artigos.

Nas Figuras 18 e 19 estão plotadas as medidas CLF_ACC e CLF_F1 obtidas com a reprodução da solução 4_tom com valores de C na faixa de 0,00001 a 10,0.

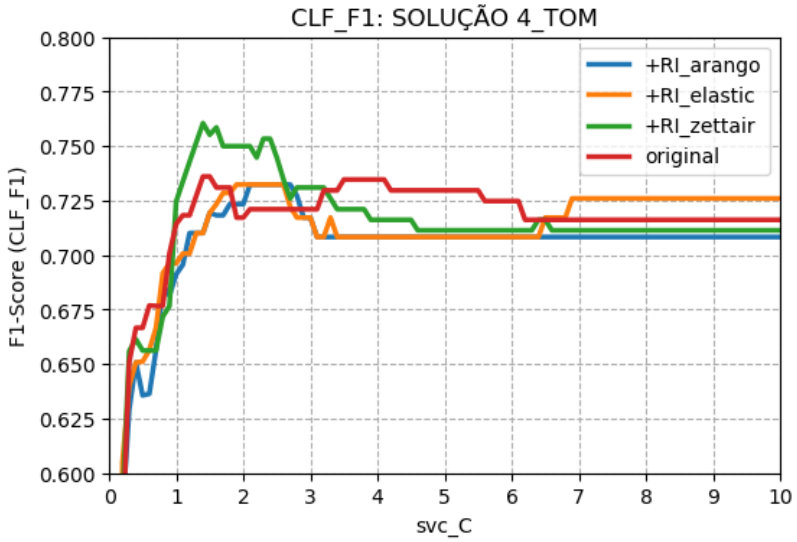
Figura 18 – Desempenho CLF_ACC da solução 4_tom para diferentes valores de refinamento da função custo, parâmetro C do classificador SVC.



Fonte: O autor.

Uma rápida análise dos gráficos permite perceber que os melhores valores do parâ-

Figura 19 – Desempenho CLF_F1 da solução 4_tom para diferentes valores de refinamento da função custo, parâmetro C do classificador SVC.



Fonte: O autor.

metro C agora se encontram na faixa de 1,0 a 3,0 para as reproduções, tanto em termos de CLF_ACC quanto de CLF_F1. O valor de $C = 0,9$ de fato não está entre os melhores possíveis para o classificador.

Nota-se que, mesmo com a adição dos atributos de RI, utilizando tanto o ArangoDB quanto o Elasticsearch, o classificador em nenhum momento consegue superar o melhor valor de C do classificador original, porém, com adição dos atributos de RI gerados pelo Zettair, ele atinge os valores de CLF_ACC = 0,8419 e CLF_F1 = 0,7606 quando $C = 1,4$. Os valores máximos atingidos com os respectivos valores de C estão dispostos na Tabela 11.

Tabela 11 – Melhores valores de CLF_ACC e CLF_F1 do classificador SVC da solução 4_tom após reproduzida com diferentes valores de C.

Solução	**C**	**Acurácia**	**F_1-Score**
original	1,4	0,8233	0,7361
+RI_arango	2,1	0,8233	0,7324
+RI_elastic	1,9	0,8233	0,7324
+RI_zettair	1,4	**0,8419**	**0,7606**

Fonte: O autor.

Observa-se então que a adição dos atributos de RI, utilizando o Zettair para gerá-los, oferece um ganho de desempenho para o classificador SVC da solução 4_tom quando efetuada a otimização de hiperparâmetros. Porém, ainda assim o classificador SVC não consegue superar os resultados obtidos com o classificar CNN da solução 1_bertha ao adicionar os parâmetros de RI com quaisquer das ferramentas. O que pode parecer um tanto estranho, já que na solução 4_tom somente o Zettair produziu ganho de desempenho do classificador. Talvez isso esteja relacionado ao modo de classificação final utilizado pela solução 1_bertha, no qual é feito o *ensemble* (junção) dos três melhores modelos de qualquer época durante

o treinamento da CNN, é possível que os atributos de RI não sejam os responsáveis pelo ganho de desempenho observado na solução 1_bertha, mas somente a iteração diferente, devido a alteração no conjunto de atributos feita por cada ferramenta, tenha gerado modelos que tenham melhor desempenho no *ensemble*. Pois, mesmo com a fixação da semente do gerador de número aleatório, como os valores dos atributos de RI são alterados (já que cada ferramenta sua própria implementação do BM25 com pequenas diferenças entre si), os modelos gerados pela reprodução com adição dos atributos de RI gerados pelo Zettair é diferente dos gerados pelo Elasticsearch, que também é diferente dos gerados pelo ArangoDB, e todos estes também são diferentes dos modelos gerados sem adição dos atributos de RI.

4.4.2 DB_AUTHORPROF

A solução 2_daneshvar18 foi reproduzida sem nenhuma adaptação para comparação com os valores que se encontram na página da competição. Na Tabela estão a medida de acurácia divulgada na página da competição (PAN, 2018) junto com as medidas da reprodução feita do mesmo modo que a submissão original, treinamento com os tweets de 3000 autores de língua inglesa e validação com os tweets de 1900 autores de língua inglesa.

Tabela 12 – Comparação das medidas da solução 2_daneshvar18 do corpus DB_AUTHORPROF divulgadas pela competição e da reprodução da solução.

Solução	**Acurácia**		F_1**-Score**	
	Competição	Reprodução	Competição	Reprodução
2_daneshvar18	0,8221	0,822105	–	0,820785

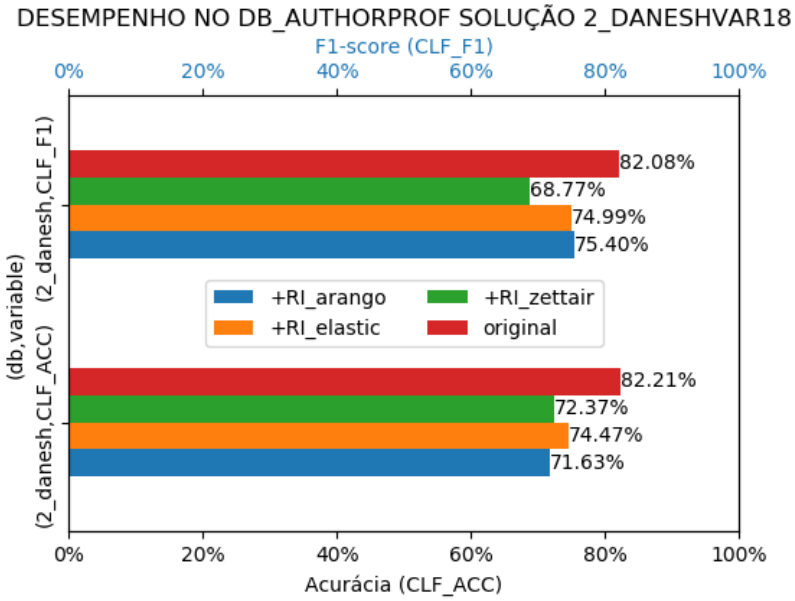
Fonte: O autor.

Ao truncar o resultado de acurácia da solução reproduzida em 4 dígitos, confirmamos que o mesmo valor divulgado na página da competição é o obtido ao reproduzir a solução. Tanto a página da competição quanto o artigo de descrição da solução (DANESHVAR; INKPEN, 2018) só mencionam a acurácia da solução, portanto não há como comparar o F_1-score obtido ao reproduzir a solução.

Depois da validação inicial da solução 2_daneshvar por meio da reprodução dos resultados da competição, ela foi adaptada para incluir os atributos de RI junto aos atributos de entrada do classificador antes da redução de dimensionalidade. As medições de CLF_ACC e CLF_F1 obtidas ao reproduzir a solução incluindo os atributos com as diferentes ferramentas estão dispostas nas Figura 20.

Como pode ser observado nos gráficos, os atributos de RI na verdade provocaram uma perda considerável de desempenho do classificador da solução 2_daneshvar18, ambas as medidas CLF_ACC e CLF_F1 foram menores com a adição dos atributos de RI com quaisquer das ferramentas utilizadas. O classificador da solução se trata de um classificador do tipo de Máquinas de Vetor de Suporte, o LinearSVC, e assim como foi feito com a solução 4_tom

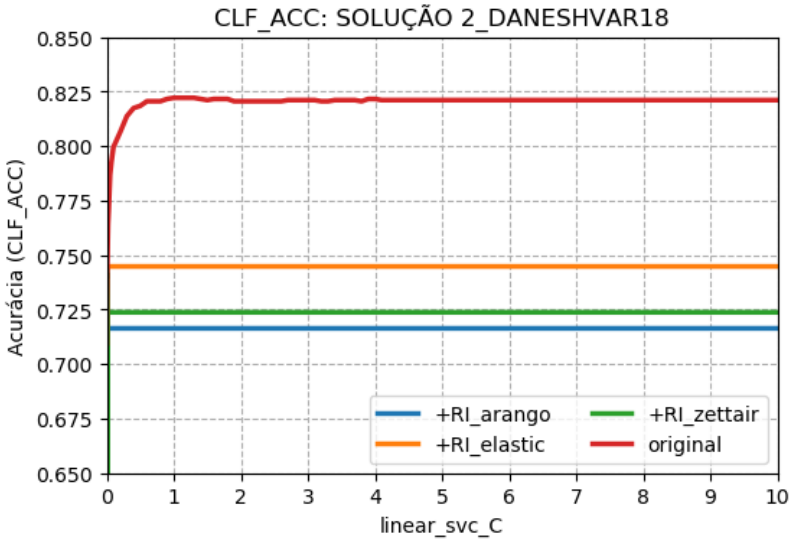
Figura 20 – Desempenho CLF_F1 e CLF_ACC da solução reproduzida para o corpus DB_AUTHORPROF.



Fonte: O autor.

anteriormente, pode-se supor que o parâmetro C do classificador, escolhido como o padrão $C = 1,0$ pela equipe da solução 2_daneshvar18, não se trata do melhor valor possível para os modelos de classificação gerados ao adicionar os atributos de RI. Então, trabalhando com essa hipótese, a solução foi reproduzida alterando o valor de C do LinearSVC na faixa de valores de $0,00001$ a $10,0$. As medidas coletadas estão plotadas nas Figuras 21 e 22.

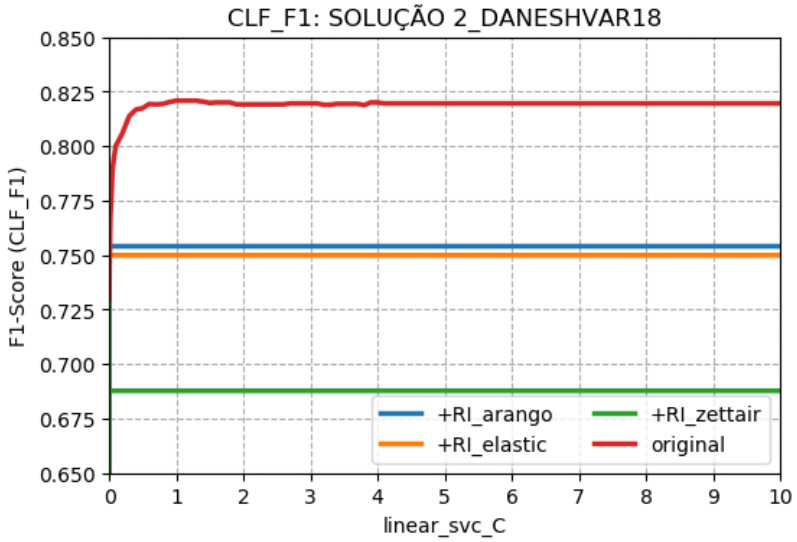
Figura 21 – Desempenho CLF_ACC da solução 2_daneshvar18 para diferentes valores de refinamento da função custo, parâmetro C do classificador LinearSVC.



Fonte: O autor.

Os gráficos de desempenho com diferentes valores de C comprovam que os atributos de RI pioraram os modelos gerados pelo classificador LinearSVC da solução 2_daneshvar18.

Figura 22 – Desempenho CLF_F1 da solução 2_daneshvar18 para diferentes valores de refinamento da função custo, parâmetro C do classificador LinearSVC.



Fonte: O autor.

Pode-se dizer que a introdução dos atributos de RI se trata de uma introdução de ruído que atrapalha o classificador e o impede alcançar os níveis de desempenho atingidos utilizando somente os atributos originais.

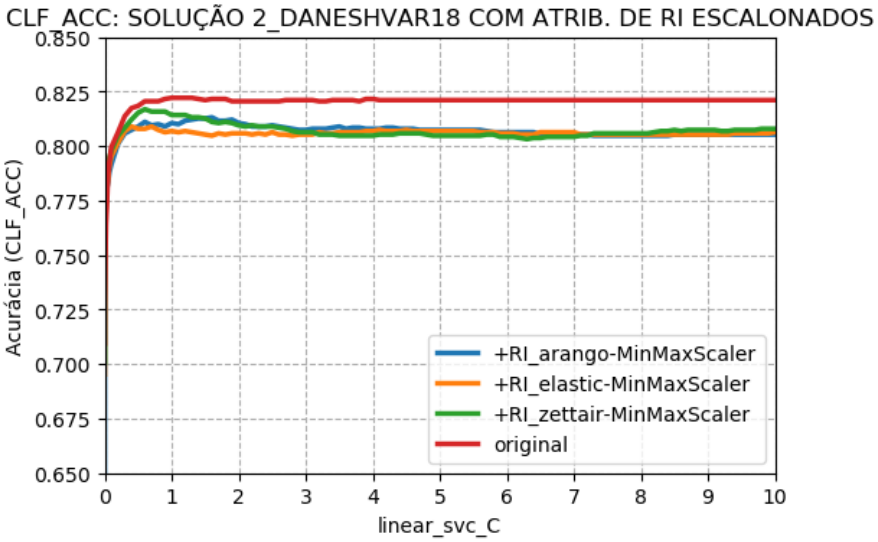
Por que isso ocorreu? Ao analisar os atributos originais após a redução de dimensionalidade foi constatado que os valores máximos e mínimos de cada uma das colunas dos 300 atributos estavam na faixa de $-1,0$ a $1,0$ enquanto que os 6 atributos de RI gerados pelas ferramentas estavam na faixa de $8,0$ até a casa dos milhares. Os classificadores do tipo SVM atribuem maior importância a atributos com maiores magnitudes conforme o núcleo selecionado (KUMAR, 2014), e isso estava acontecendo com o núcleo padrão do classificador LinearSVC do sklearn.

Para tentar resolver o problema foi utilizado o escalonador MinMaxScaler nos 6 atributos de RI para dimensioná-los para a faixa de $-1,0$ a $1,0$. Os medidas coletadas podem ser observadas nas Figuras 23 e 24.

Apesar do desempenho do classificador da solução 2_daneshvar18 ter sido maior com os atributos de RI escalonados, como mostra o gráfico da Figura 23 em comparação com o da Figura 21, ainda assim o desempenho continuou abaixo da reprodução da solução original. Na Tabela 13 podem ser vistos os valores máximos atingidos com os respectivos valores de C.

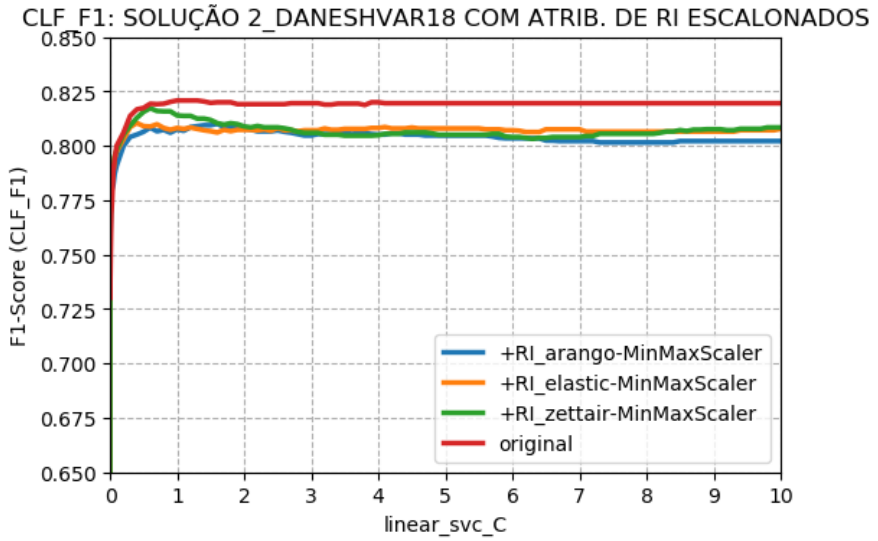
Pode-se perceber que o melhor desempenho com os atributos de RI é quando gerado pelo Zettair, no entanto ainda assim fica abaixo da solução original. Obtendo acurácia de $81,68\%$ contra $82,22\%$ da original, e F_1-score de $81,72\%$ contra $82,08\%$ da original.

Figura 23 – Desempenho CLF_ACC da solução 2_daneshvar18 para diferentes valores de refinamento da função custo, parâmetro C do classificador LinearSVC com atributos de RI escalonados pelo MinMaxScaler.



Fonte: O autor.

Figura 24 – Desempenho CLF_F1 da solução 2_daneshvar18 para diferentes valores de refinamento da função custo, parâmetro C do classificador LinearSVC com atributos de RI escalonados pelo MinMaxScaler.



Fonte: O autor.

Tabela 13 – Melhores valores de CLF_ACC e CLF_F1 do classificador LinearSVC da solução 2_daneshvar18 após reproduzida com diferentes valores de C, com atributos de RI escalonados pelo MinMaxScaler.

Solução	**C**	**Acurácia**	F_1**-Score**
original	1,0	**0,822105**	**0,820785**
+RI_arango	1,6	0,813158	0,810059
+RI_elastic	0,4	0,808947	0,810444
+RI_zettair	0,6	0,816842	0,817227

Fonte: O autor.

5 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Esse estudo investigativo buscou avaliar o desempenho de atributos oriundos da área de Recuperação de Informação em tarefas de Mineração de Texto, com enfoque em atributos gerados pela função BM25, por meio de uma metodologia própria. A geração desses atributos foi subsidiada por ferramentas que implementam a função BM25 e o desempenho dos atributos de RI foi avaliado em dois corpus distintos. O desempenho das ferramentas de armazenamento e indexação também foi avaliado.

Os resultados obtidos com a metodologia proposta mostram que o Zettair foi a ferramenta mais rápida na questão de indexação e também para consultas com textos de tamanho limitado. O Elasticsearch também apresentou bom desempenho, não possuindo limitações de tamanho do texto nas consultas, e foi o melhor dentre os dois SGBDs testados. Além disso, ele possui muito mais funcionalidades que o Zettair, que por sua vez não é um SGBD.

Ainda na avaliação de desempenho das ferramentas, percebe-se que o ArangoDB obteve um resultado muito aquém dos demais na medida TIME_QUERY para o corpus de 300 mil registros, indicando que as consultas utilizando a função BM25 do ArangoDB são lentas para bancos de dados com grande número de documentos.

Dentre as soluções selecionadas para reprodução, só foi possível comprovar as medidas do ranking final das competições para a solução 2_daneshvar18 do corpus DB_AUTHORPROF, para as duas soluções do corpus DB_HYPERPARTISAN isso não foi possível pois o conjunto de validação final da competição não foi publicado. Contudo, as soluções do DB_HYPERPARTISAN selecionadas mantiveram a relação de posição quando reproduzidas, com a solução 1_bertha obtendo melhor acurácia e melhor F1-Score do que a solução 4_tom.

Na avaliação de desempenho dos classificadores adicionando os atributos de RI, a Rede Neural Convolucional (1_bertha) apresentou o maior ganho de desempenho, saltando de uma acurácia de 81,40% para 86,51% e F_1-Score de 76,19% para 81,05% ao utilizar o ArangoDB para gerar os atributos de RI. A adição dos atributos de RI gerados por quaisquer ferramentas proporcionou ganho de desempenho à CNN da solução 1_bertha para o corpus DB_HYPERPARTISAN, porém, não houve qualquer ganho de desempenho ao adicionar os atributos de RI ao classificador SVC da solução 4_tom.

Uma análise e seleção dos parâmetros do classificador SVC mostrou que adicionar os atributos de RI gerados pelo Zettair produz um ganho de desempenho de acurácia de 82,33% para 84,19% e F_1-score de 73,61% para 76,06% quando o melhor valor do parâmetro C é selecionado.

Quanto ao corpus DB_AUTHORPROF, o classificador LinearSVC da solução 2_daneshvar18 não obteve nenhum ganho de desempenho ao serem adicionados os atributos de RI,

nem mesmo ao ser feita a análise e seleção do parâmetro C do classificador. Uma investigação posterior revelou que na verdade a magnitude dos atributos de RI era bem maior que a dos atributos originais, prejudicando o classificador LinearSVC. No entanto, mesmo com o reparo de magnitude por meio do escalonador MinMaxScaler, e nova análise e seleção do parâmetro C do classificador, não foi obtido nenhum ganho de desempenho ao adicionar os atributos de RI à solução 2_daneshvar18.

Então, conclui-se que os 6 atributos de RI sugeridos neste estudo proporcionaram ganho de desempenho somente para classificações do corpus DB_HYPERPARTISAN, um corpus pequeno com documentos extensos. Os atributos de RI funcionaram melhor com o classificador CNN, e para o classificador SVC somente o Zettair proporcionou ganho de desempenho. No corpus DB_AUTHORPROF, um corpus grande com documentos curtos, com o classificador LinearSVC, os atributos de RI causaram perda de desempenho. Para corroborar o ganho de desempenho, ou não, dos atributos de RI sugeridos são necessários mais testes pois os resultados aqui obtidos não são unânimes. São necessários testes com outros classificadores nesses mesmos corpus e também em diferentes corpus, corpus grandes com documentos extensos e corpus pequenos com documentos pequenos. Podem ser pensados também testes com diferentes valores de top-k e parâmetros k_1, k_3 e b para cálculo dos atributos de RI.

5.1 TRABALHOS FUTUROS

Para trabalhos futuros está planejada uma maior análise exploratória dos parâmetros da solução 2_daneshvar18, reproduzindo a solução com maiores variações nos parâmetros do classificador LinearSVC, inclusive com alteração do núcleo do mesmo, para verificar se alguma configuração dele possui desempenho superior constante ao serem adicionados os atributos de RI. Isso não foi feito neste estudo devido à limitação temporal para executar todas as variações necessárias para chegar a mais conclusões, mas alguns resultados já estão guardados.

Quanto maior o número de soluções reproduzidas para cada corpus melhor, mas trabalhar com um mínimo talvez seja o ideal, então para todas as reproduções futuras é bom tomar como base no mínimo duas soluções, ou classificadores distintos, por corpus, para efetuar as comparações. Então, em um trabalho futuro imediato planeja-se reproduzir mais uma das soluções do corpus DB_AUTHORPROF disponíveis online.

Como mencionado antes, trabalhos futuros podem também trabalhar somente analisando diferentes valores que influenciem somente nos atributos de RI gerados, o top-k e os parâmetros k_1, k_3 e b reproduzindo-os para os mesmos corpus.

Além disso, uma outra possibilidade de estudo, é o maior isolamento do índice de geração dos atributos de RI do conjunto de treinamento por meio da separação dos corpus

em 3 pedaços. Nessa separação o primeiro pedaço seria utilizado para para treinamento do classificador, o segundo pedaço seria utilizado como o índice para geração dos atributos derivados da função BM25 para o primeiro pedaço, e o terceiro pedaço ficaria para teste/validação do classificador.

REFERÊNCIAS

AGGARWAL, C. C. **Data Mining: The Textbook**. New York, NY, USA: Springer, 2015. 734 p. ISBN 9783319141411. Disponível em: <https://www.springer.com/gp/book/9783319141411>. Citado 2 vezes nas páginas 31 e 32.

AGGARWAL, C. C.; ZHAI, C. **Mining Text Data**. New York, NY, USA: Springer, 2012. 522 p. ISBN 9781461432227. Disponível em: <https://www.springer.com/gp/book/9781461432227>. Citado na página 13.

ARANGODB INC. **ArangoDB - GitHub - arangodb/arangodb at 3.4.2**. 2019. Disponível em: <https://web.archive.org/web/20190712211053/https://github.com/arangodb/arangodb/tree/3.4.2>. Citado na página 41.

ARANGODB INC. **ArangoDB v3.4.6 Documentation: Introduction to ArangoDB Documentation**. 2019. Disponível em: <https://web.archive.org/web/20190712211033/https://www.arangodb.com/docs/3.4/index.html>. Citado na página 41.

ARANGODB INC. **ArangoSearch Views in AQL**. 2019. Disponível em: <https://web.archive.org/web/20190712211727/https://www.arangodb.com/docs/3.4/aql/views-arango-search.html#bm25>. Citado na página 41.

BAEZA-YATES, R.; RIBEIRO-NETO, B. **Modern Information Retrieval**. 1. ed. Essex, England: Addison-Wesley, 1999. 499 p. ISBN 978-0-201-39829-8. Citado na página 16.

BAEZA-YATES, R.; RIBEIRO-NETO, B. **Modern Information Retrieval: The Concepts and Technology Behind Search**. 2. ed. Essex, England: Addison-Wesley, 2011. 913 p. ISBN 978-0-321-41691-9. Citado 4 vezes nas páginas 14, 18, 21 e 22.

BERRY, M. W.; KOGAN, J. **Text Mining: Applications and Theory**. 1. ed. Chichester, West Sussex, Reino Unido: John Wiley & Sons, 2010. 207 p. ISBN 9780470749821. Disponível em: <https://www.wiley.com/en-br/Text+Mining:+Applications+and+Theory-p-9780470749821>. Citado 2 vezes nas páginas 34 e 36.

BUSH, V. As We May Think. **SIGPC Note.**, ACM, New York, NY, USA, v. 1, n. 4, p. 36–44, abr. 1979. ISSN 0163-5816. Disponível em: <http://doi.acm.org/10.1145/1113634.1113638>. Citado na página 17.

CAPPELLATO, L. et al. (Ed.). **Working Notes for CLEF 2014 Conference**, v. 1180, n. 1180 de **CEUR Workshop Proceedings**, (CEUR Workshop Proceedings, 1180). Aachen: [s.n.], 2014. ISSN 1613-0073. Disponível em: <http://ceur-ws.org/Vol-1180>.

CHASKI, C. E. Author Identification In The Forensic Setting. **The Oxford Handbook of Language and Law**, maio 2012. Disponível em: <hhttp://doi.org/10.1093/oxfordhb/9780199572120.013.0036>. Citado na página 13.

CLEVERDON, C. The Evaluation of Systems Used in Information Retrieval. In: **Proceedings of the International Conference on Scientific Information: Two Volumes**. Washington, DC: The National Academies Press, 1959. p. 687–698. Disponível em: <https://doi.org/10.17226/10866>. Citado na página 17.

DANESHVAR, S.; INKPEN, D. Gender Identification in Twitter using N-grams and LSA—Notebook for PAN at CLEF 2018. In: CAPPELLATO, L. et al. (Ed.). **CLEF 2018 Evaluation Labs and Workshop – Working Notes Papers, 10-14 September, Avignon, France**. CEUR-WS.org, 2018. ISSN 1613–0073. Disponível em: <http://ceur-ws.org/Vol-2125/>. Citado 2 vezes nas páginas 49 e 60.

DONG, G.; LIU, H. **Feature Engineering for Machine Learning and Data Analytics**. 1. ed. Boca Raton, FL, EUA: CRC Press, 2018. 400 p. ISBN 9781315181080. Disponível em: <https://www.taylorfrancis.com/books/e/9781315181080>. Citado na página 32.

ELASTIC. **Elasticsearch: A Distributed RESTful Search Engine - GitHub - elastic/elasticsearch at 7.2**. 2019. Disponível em: <https://web.archive.org/web/20190712125715/https://github.com/elastic/elasticsearch/tree/7.2>. Citado na página 40.

ELASTIC. **Elasticsearch Reference [7.2] » Elasticsearch introduction**. 2019. Disponível em: <https://web.archive.org/web/20190712125432/https://www.elastic.co/guide/en/elasticsearch/reference/7.2/elasticsearch-intro.html>. Citado na página 40.

ELASTIC. **Elasticsearch Reference [7.2] » Index modules » Similarity module**. 2019. Disponível em: <https://web.archive.org/web/20190712131555/https://www.elastic.co/guide/en/elasticsearch/reference/current/index-modules-similarity.html>. Citado na página 41.

ELMASRI, R.; NAVATHE, S. B. **Sistemas de Banco de Dados**. 6. ed. São Paulo, Brasil: Pearson Education do Brasil, 2010. 788 p. ISBN 9788579360855. Citado na página 41.

FELDMAN, R.; DAGAN, I. Knowledge Discovery in Textual databases (KDT). In: FAYYAD, U.; UTHURUSAMY, R. (Ed.). **Proceedings of the First International Conference on Knowledge Discovery and Data Mining**. AAAI Press, 1995. (KDD'95), p. 112–117. Disponível em: <http://dl.acm.org/citation.cfm?id=3001335.3001354>. Citado na página 26.

FELDMAN, R.; SANGER, J. **Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data**. 1. ed. New York, NY, USA: Cambridge University Press, 2006. 410 p. ISBN 0521836573, 9780521836579. Citado 8 vezes nas páginas 13, 14, 26, 27, 28, 30, 32 e 33.

GROSSMAN, D. A.; FRIEDER, O. **Information Retrieval: Algorithms and Heuristics**. 2. ed. Chicago, IL, USA: Springer, 2004. 332 p. (The Kluwer International Series of Information Retrieval). ISBN 9781402030048. Citado na página 16.

HAN, J.; KAMBER, M.; PEI, J. **Data Mining: Concepts and Techniques**. 3rd. ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011. 703 p. ISBN 0123814790, 9780123814791. Citado 8 vezes nas páginas 13, 26, 27, 30, 35, 36, 37 e 48.

IRESEARCH. **IResearch search engine: Version 1.0 - GitHub - iresearch-toolkit/iresearch at arangodb-3.4**. 2019. Disponível em: <https://web.archive.org/web/20190712233055/https://github.com/iresearch-toolkit/iresearch/tree/arangodb-3.4>. Citado na página 41.

JIANG, Y. et al. Team bertha von suttner at SemEval-2019 task 4: Hyperpartisan news detection using ELMo sentence representation convolutional network. In: **Proceedings of the 13th International Workshop on Semantic Evaluation**. Minneapolis, Minnesota, USA: Association for Computational Linguistics, 2019. p. 840–844. Disponível em: <https://www.aclweb.org/anthology/S19-2146>. Citado 2 vezes nas páginas 46 e 47.

JO, T. **Text Mining: Concepts, Implementation, and Big Data Challenge**. 1. ed. Springer International Publishing, 2018. 373 p. (Studies in Big Data, 45). ISBN 9783319918143. Disponível em: <https://doi.org/10.1007/978-3-319-91815-0>. Citado 7 vezes nas páginas 13, 14, 26, 27, 28, 29 e 32.

JONES, K. S. **Information Retrieval Experiment**. Newton, MA, USA: Butterworth-Heinemann, 1981. ISBN 0408106484. Disponível em: <https://dl.acm.org/citation.cfm?id=539571>. Citado na página 21.

KIESEL, J. et al. **Data for PAN at SemEval 2019 Task 4: Hyperpartisan News Detection**. 2018. Disponível em: <https://doi.org/10.5281/zenodo.1489920>. Citado na página 39.

KODRATOFF, Y. Knowledge Discovery in Texts: A Definition, and Applications. In: **Proceedings of the 11th International Symposium on Foundations of Intelligent Systems**. Londres, Reino Unido: Springer-Verlag, 1999. (ISMIS '99), p. 16–29. ISBN 3-540-65965-X. Disponível em: <http://dl.acm.org/citation.cfm?id=646358.689959>. Citado na página 26.

KOWALSKI, G. **Information Retrieval Architecture and Algorithms**. 1. ed. Ashburn, VA, USA: Springer, 2010. 305 p. ISBN 9781441977151. Citado na página 14.

KUMAR, N. **Using Support Vector Machines Effectively: Getting the best performance out of support vector machines (SVMs)**. 2014. Disponível em: <https://web.archive.org/web/20170929001525/http://neerajkumar.org/writings/svm/>. Citado na página 62.

KWARTLER, T. **Text Mining in Practice With R**. 1. ed. Chichester, Inglaterra: John Wiley & Sons, 2017. 307 p. ISBN 9781119282013. Disponível em: <https://www.wiley.com/en-us/Text+Mining+in+Practice+with+R-p-9781119282013>. Citado na página 29.

LAROSE, D. T.; LAROSE, C. D. **Discovering Knowledge in data: An Introduction to Data Mining**. Hoboken, New Jersey, EUA: Wiley, 2014. 316 p. ISBN 9780470908747. Disponível em: <https://www.wiley.com/en-br/Discovering+Knowledge+in+Data:+An+Introduction+to+Data+Mining,+2nd+Edition-p-9780470908747>. Citado na página 30.

LINUX MINT. **xed - GitHub - linuxmint/xed**. 2019. Disponível em: <https://web.archive.org/web/20190506095954/https://github.com/linuxmint/xed>. Citado na página 28.

LUCENE. **Apache LuceneTM 8.1.1 Documentation**. 2019. Disponível em: <https://web.archive.org/web/20190712130658/https://lucene.apache.org/core/8_1_1/index.html>. Citado na página 40.

LYMAN, P.; VARIAN, H. R. How much information 2003? **http://www.sims.berkeley.edu/research/projects/how-much-info-2003/**, 2003. Disponível em: <http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/>. Citado na página 13.

MA, Y.; TANG, J.; AGGARWAL, C. Feature Engineering for Data Streams. In: DONG, G.; LIU, H. (Ed.). **Feature Engineering for Machine Learning and Data Analytics**. 1. ed. Boca Raton, FL, EUA: CRC Press, 2018. cap. 5, p. 117–143. ISBN 9781315181080. Disponível em: <https://www.taylorfrancis.com/books/e/9781315181080/chapters/10.1201/9781315181080-5>. Citado na página 14.

MANNING, C. D.; RAGHAVAN, P.; SCHüTZE, H. **Introduction to Information Retrieval**. 1. ed. Cambridge, Reino Unido: Cambridge University Press, 2008. 482 p. ISBN 9780511414053. Citado 8 vezes nas páginas 16, 17, 19, 20, 21, 23, 24 e 25.

MOORE, G. E. **Progress in Digital Integrated Eletronics**. 1975. 11–13 p. Disponível em: <http://ai.eecs.umich.edu/people/conway/VLSI/BackgroundContext/SMErpt/AppB.pdf>. Citado na página 17.

NEW MEDIA RIGHTS. **Open Source Licensing Guide**. 2015. Disponível em: <https://web.archive.org/web/20190712132826/https://www.newmediarights.org/open_source/new_media_rights_open_source_licensing_guide>. Citado na página 40.

PAN. **Author Profiling PAN @ CLEF 2018**. 2018. Disponível em: <https://web.archive.org/web/20190712001219/https://pan.webis.de/clef18/pan18-web/author-profiling.html>. Citado 4 vezes nas páginas 38, 39, 40 e 60.

PAN. **Hyperpartisan News Detection: Leaderboard - PAN @ SemEval 2019**. 2019. Disponível em: <https://web.archive.org/web/20190803162634/https://pan.webis.de/semeval19/semeval19-web/leaderboard.html>. Citado 2 vezes nas páginas 40 e 56.

PAN. **Hyperpartisan News Detection PAN @ SemEval 2019**. 2019. Disponível em: <https://web.archive.org/web/20190711231940/https://pan.webis.de/semeval19/semeval19-web/index.html>. Citado na página 39.

PAN'07 Workshop. **International Workshop on Plagiarism Analysis, Authorship Identification, and Near-Duplicate Detection (PAN)**. 2007. Disponível em: <https://web.archive.org/web/20190711212207/https://www.uni-weimar.de/medien/webis/events/pan-07/pan07-web/>. Citado na página 38.

Peter - Karussel. **Jetslide uses ElasticSearch as Database**. 2011. Disponível em: <https://web.archive.org/web/20190712183200/https://karussell.wordpress.com/2011/07/13/jetslide-uses-elasticsearch-as-database/>. Citado na página 41.

PETERS, M. E. et al. Deep contextualized word representations. **CoRR**, abs/1802.05365, 2018. Disponível em: <http://arxiv.org/abs/1802.05365>. Citado na página 47.

RANGEL, F. et al. Overview of the 2nd Author Profiling Task at PAN 2014. In: CAPPELLATO, L. et al. (Ed.). **CLEF 2014**: CLEF2014 Working Notes. Aachen: [s.n.], 2014. (CEUR Workshop Proceedings, 1180), p. 989–927. ISSN 1613-0073. Disponível em: <http://ceur-ws.org/Vol-1180/CLEF2014wn-Pan-RangelEt2014.pdf>. Citado na página 13.

RANGEL, F. et al. Overview of the 6th Author Profiling Task at PAN 2018: Multimodal Gender Identification in Twitter. **Working Notes Papers of the CLEF**, 2018. Disponível em: <http://ceur-ws.org/Vol-2125/invited_paper_15.pdf>. Citado 2 vezes nas páginas 13 e 38.

RANGEL, F. et al. Overview of the 5th Author Profiling Task at PAN 2017: Gender and Language Variety Identification in Twitter. **Working Notes Papers of the CLEF**, 2017. Disponível em: <http://ceur-ws.org/Vol-1866/invited_paper_11.pdf>. Citado na página 39.

RMIT University. **Zettair Homepage: Introduction**. 2009. Disponível em: <https://web.archive.org/web/20190713000005/http://www.seg.rmit.edu.au/zettair/index.html>. Citado na página 41.

RMIT University. **Zettair Index Build**. 2009. Disponível em: <https://web.archive.org/web/20190713000144/http://www.seg.rmit.edu.au/zettair/doc/Build.html>. Citado na página 41.

RMIT University. **Zettair User Guide**. 2009. Disponível em: <https://web.archive.org/web/20190713000200/http://www.seg.rmit.edu.au/zettair/doc/Readme.html>. Citado na página 41.

ROBERTSON, S. The Probabilistic Relevance Framework: BM25 and Beyond. **Foundations and Trends® in Information Retrieval**, v. 3, n. 4, p. 333–389, 2010. ISSN 1554-0669, 1554-0677. Disponível em: <http://www.nowpublishers.com/product.aspx?product=INR&doi=1500000019>. Citado 3 vezes nas páginas 23, 24 e 25.

ROBERTSON, S. E. et al. Okapi at TREC-3. In: HARMAN, D. K. (Ed.). **Overview of the Third Text REtrieval Conference (TREC-3)**. Washington, DC: [s.n.], 1996. (NIST Special Publication, 500-225), p. 109–126. Disponível em: <https://web.archive.org/web/20190723231216/https://www.computing.dcu.ie/~gjones/Teaching/CA437/city.pdf>. Citado na página 23.

SAMMUT, C.; WEBB, G. I. (Ed.). **Encyclopedia of Machine Learning and Data Mining**. 2. ed. [S.l.]: Springer Science+Business Media New York, 2017. 1335 p. Citado 2 vezes nas páginas 13 e 14.

SANDERSON, M.; CROFT, W. B. The History of Information Retrieval Research. **Proceedings of the IEEE**, v. 100, n. Special Centennial Issue, p. 1444–1451, maio 2012. ISSN 0018-9219. Citado 3 vezes nas páginas 16, 17 e 19.

TAN, P.-N.; STEINBACH, M.; KUMAR, V. **Introduction to Data Mining**. Harlow, Essex, Inglaterra: Pearson, 2014. 732 p. ISBN 9781292026152. Disponível em: <https://www.pearsonelt.ch/HigherEducation/Pearson/EAN/9781292026152/Introduction-to-Data-Mining-Pearson-New-International-Edition>. Citado 4 vezes nas páginas 30, 31, 32 e 34.

TAUBE, M.; GULL, C. D.; WACHTEL, I. S. Unit terms in coordinate indexing. **Journal of the Association for Information Science and Technology**, v. 3, n. 4, p. 213–218, abr. 1952. Citado na página 17.

TURCHI, M.; MAMMONE, A.; CRISTIANINI, N. Analysis of Text Patterns Using Kernel Methods. In: SRIVASTAVA, A. N.; SAHAMI, M. (Ed.). 1. ed. Chapman & Hall/CRC, 2009, (Chapman & Hall/CRC Data Mining and Knowledge Discovery Series). cap. 1, p. 1–25. ISBN 9781420059403. Disponível em: <https://www.taylorfrancis.com/books/e/9781315181080/chapters/10.1201/9781315181080-5>. Citado na página 30.

W3C. **Extensible Markup Language (XML) 1.0 (Fifth Edition): W3C Recommendation 26 November 2008**. 2008. Disponível em: <https://web.archive.org/web/20190804210831/https://www.w3.org/TR/2008/REC-xml-20081126/>. Citado na página 29.

WALTER, C. **Kryder's Law**. 2005. Disponível em: <https://web.archive.org/web/20190731213726/https://www.scientificamerican.com/article/kryders-law/>. Citado na página 17.

WEREN, E. R. D. **Atribuição de Perfis de Autoria**. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Sul, Porto Alegre, BR-RS, novembro 2014. Disponível em: <http://hdl.handle.net/10183/108592>. Citado 6 vezes nas páginas 3, 4, 14, 28, 42 e 43.

WEREN, E. R. D. et al. Examining Multiple Features for Author Profiling. **Journal of Information and Data Management**, v. 5, n. 3, p. 266–279, outubro 2014. ISSN 2178-7107.

Disponível em: <https://periodicos.ufmg.br/index.php/jidm/article/view/282>. Citado 3 vezes nas páginas 13, 28 e 42.

WEREN, E. R. D.; MOREIRA, V. P.; OLIVEIRA, J. P. M. de. Exploring Information Retrieval features for Author Profiling. In: CAPPELLATO, L. et al. (Ed.). **CLEF 2014**: CLEF2014 Working Notes. Aachen: [s.n.], 2014. (CEUR Workshop Proceedings, 1180), p. 1164–1171. ISSN 1613-0073. Disponível em: <http://ceur-ws.org/Vol-1180/CLEF2014wn-Pan-WerenEt2014.pdf>. Citado 3 vezes nas páginas 3, 28 e 42.

WIKIPéDIA. **Portal:Conteúdo destacado**. 2019. Disponível em: <https://web.archive.org/web/20190505054741/https://pt.wikipedia.org/wiki/Portal:Conte%C3%BAdo_destacado>. Citado na página 29.

YAZıCı, V. **Using Elasticsearch as the Primary Data Store**. 2018. Disponível em: <https://web.archive.org/web/20190712183146/https://vlkan.com/blog/post/2018/11/14/elasticsearch-primary-data-store/>. Citado na página 41.

YEH, C.-L.; LONI, B.; SCHUTH, A. Tom jumbo-grumbo at SemEval-2019 task 4: Hyperpartisan news detection with GloVe vectors and SVM. In: **Proceedings of the 13th International Workshop on Semantic Evaluation**. Minneapolis, Minnesota, USA: Association for Computational Linguistics, 2019. p. 1067–1071. Disponível em: <https://www.aclweb.org/anthology/S19-2187>. Citado na página 46.

ZHAI, C.; MASSUNG, S. **Text Data Management and Analysis: A Practical Introduction to Information Retrieval and Text Mining**. 1. ed. [S.l.]: ACM Books, 2016. 510 p. ISBN 9781970001174. Citado 6 vezes nas páginas 13, 14, 21, 23, 24 e 34.

ZHENG, A.; CASARI, A. **Feature Engineering for Machine Learning**. 1. ed. Sebastopol, CA, EUA: O'Reilly, 2018. 200 p. ISBN 9781491953242. Disponível em: <http://oreilly.com/catalog/errata.csp?isbn=9781491953242>. Citado na página 32.

Apêndices

APÊNDICE A – TRECHO DE CÓDIGO: FUNÇÃO *INDEX*

O código completo da função *index* pode ser encontrado no seguinte arquivo disponível na internet: https://github.com/ruanmed/tcc-ii-ir-features-text-mining/blob/master/tool-testing/time-index.py.

```
...
def index(idx_type='normal', db='authorprof', tool='arango',
↪  db_name='authorprof', exp_id='unamed'):
    mylogger.info('')
    mylogger.info(f'INDEX TYPE: {idx_type}')
    mylogger.info(f'DB: {db}')
    mylogger.info(f'TOOL: {tool}')
    mylogger.info(f'DB NAME: {db_name}')
    initial = time.time()

    testTool = IndexToolManager(indexName=db_name)

    start = time.time()
    append_class_to_id = False
    if (tool == 'zettair'):
        append_class_to_id = True
    bulk = testTool.get_documents(db,
                                  db_files[db]['xml_folder'],
                                  db_files[db]['truth_txt'],
                                  append_class_to_id)
    end = time.time()
    mylogger.info(f'get_documents {end - start}')
    mylogger.info(f'TOTAL documents {len(bulk)}')

    start = time.time()
    if (tool == 'arango'):
        documentList = testTool.bulkListGeneratorArango(bulk)
        end = time.time()
        mylogger.info(f'bulkListGeneratorArango {end - start}')
        if (idx_type == 'normal'):
            start = time.time()
            for doc in documentList:
                testTool.insertDocumentArango(doc)
            end = time.time()
            mylogger.info(f'for-loop insertDocumentArango {end -
            ↪  start}')
        if (idx_type == 'bulk'):
            start = time.time()
            testTool.bulkImportArango(documentList)
```

```
            end = time.time()
            mylogger.info(f'bulkImportArango {end - start}')

    if (tool == 'elastic'):
        if (idx_type == 'normal'):
            start = time.time()
            for doc in bulk:
                testTool.insertElastic(doc.pop('id'), doc)
            end = time.time()
            mylogger.info(f'for-loop insertElastic {end - start}')

        if (idx_type == 'bulk'):
            start = time.time()
            bulkBody = testTool.bulkHelperInsertGeneratorElastic(bulk)
            end = time.time()
            mylogger.info(f'bulkHelperInsertGeneratorElastic {end -
            ↪  start}')

            start = time.time()
            testTool.bulkHelperElastic(bulkBody)
            end = time.time()
            mylogger.info(f'bulkHelperElastic {end - start}')

        start = time.time()
        testTool.refreshElastic()
        end = time.time()
        mylogger.info(f'refreshElastic {end - start}')

    if (tool == 'zettair'):
        start = time.time()
        testTool.saveToTrecFileZettair(bulk)
        end = time.time()
        mylogger.info(f'saveToTrecFileZettair {end - start}')

        start = time.time()
        testTool.zettair_index()
        end = time.time()
        mylogger.info(f'zettair_index {end - start}')

    final = time.time()
    result_id = datetime.datetime.now().strftime("%Y%m%d-%H%M%S.%f")
    testTool.log_result(result_id, {
        'exp_id': exp_id,
        'variable': 'TIME_INDEX',
        'index_type': idx_type,
        'db': db,
        'tool': tool,
        'db_name': db_name,
```

```
        'value': str((final - initial)),
    })
    mylogger.info(f'index TOTAL TIME: {final - initial}')
```

APÊNDICE B – TRECHOS DA IMPLEMENTAÇÃO DA CLASSE *INDEXTOOLMANAGER*

A implementação completa da classe *IndexToolManager* pode ser encontrada no seguinte arquivo disponível na internet: https://github.com/ruanmed/tcc-ii-ir-features-text-mining/blob/master/tool-testing/indextoolmanager.py.

```
import time
import math
import subprocess
from arango import ArangoClient
from elasticsearch import Elasticsearch
from elasticsearch import helpers as ElasticsearchHelpers
import xml.etree.ElementTree as ET
import pandas as pd
import json
from shlex import quote
...
class IndexToolManager:
    '''
    A class used to manage the database indexation tools used in this
↪   research.
    Provides functions to index a database with ArangoDB, Elasticsearch
↪   and Zettair,
    using the BM25 IR function implemented in each of those tools.
    Also makes it possible to query the indexed database using BM25.

    Attributes
    ----------
    indexName : str
        a string to refer to the current working data set

    bm25_b : float
        BM25 b parameter to adjust the document length compensation

    bm25_k1 : float
        BM25 k1 parameter to adjust the term-frequency weight

    bm25_k3 : float
        BM25 k3 parameter to adjust the term-frequency weight in the
↪   query (used for long queries)

    top_k : int
        Number of results to be retrieved when querying the database

    '''
```

```
def __init__(self, indexName='default_index',
             bm25_b=0.75, bm25_k1=1.2, bm25_k3=0.0, top_k=100):
    self.indexName = indexName
    self.bm25_b = float(bm25_b)
    self.bm25_k1 = float(bm25_k1)
    self.bm25_k3 = float(bm25_k3)
    self.numberResults = int(top_k)
    self.root_path =
    ↪  "/home/ruan/Documentos/git/tcc-ii-ir-features-text-mining/tool-testing/

    self.zettair_query_process = None

    self.initializeArango()
    self.initializeElastic()

    self.resultsIndexName = 'tcc_results'
    body = {
        "settings": {
            "number_of_shards": 1,
        }
    }
    if not
    ↪  self.elasticClient.indices.exists(index=self.resultsIndexName):
        self.elasticClient.indices.create(
            index=self.resultsIndexName, body=body)

    # Create a new database named "resultsIndexName" if it does not
    ↪  exist.
    if not self.arango_sys_db .has_database(self.resultsIndexName):
        self.arango_sys_db .create_database(self.resultsIndexName)

    # Connect to "resultsIndexName" database as root user.
    # This returns an API wrapper for "resultsIndexName" database.
    self.arangoResultsDb = self.arangoClient.db(
        self.resultsIndexName, username=None, password=None)

    db = self.arangoResultsDb
    # Create a new collection named "resultsIndexName" if it does
    ↪  not exist.
    # This returns an API wrapper for "resultsIndexName" collection.
    if db.has_collection(self.resultsIndexName):
        self.arangoResultsCollection =
        ↪  db.collection(self.resultsIndexName)
    else:
        self.arangoResultsCollection = db.create_collection(
            self.resultsIndexName)
```

```
    ...

    def log_result(self, itemKey, itemBody):
        '''
        Inserts a document in the Elasticsearch database.

        Parameters
        ----------
        itemKey : str or number
            Document identifier

        itemBody : dict
            Document body/data.
        '''

        self.elasticClient.index(
            index=self.resultsIndexName,
            ↪  doc_type=self.elasticDocumentType,
            id=itemKey, body=itemBody)

        document = {'_key': itemKey}
        document.update(itemBody)
        self.arangoResultsCollection.insert(document)

    ...

    def get_documents(self, db='authorprof',
                      documents_xml_folder='db_authorprof/en/',
                      truth_txt='db_authorprof/truth.txt',
                      append_class_to_id=False):
        '''
        Generates a list with all documents from db formatted files.

        Parameters
        ----------
        db : str
            Database name.

        documents_xml_folder : str
            Folder that contains the XML files from the authors'
↪   documents (twits),
            must follow the DB_AUTHORPROF task XML format.

        truth_txt : str
            Truth TXT file with authors' classifications of gender {
↪   female | male },
            must follow the DB_AUTHORPROF task TXT format.
        '''
```

```
        if (db == 'authorprof'):
            return self.get_documents_DB_AUTHORPROF(documents_xml_folder,
            ↪  truth_txt, append_class_to_id)
        if (db == 'botgender'):
            return self.get_documents_DB_BOTGENDER(documents_xml_folder,
            ↪  truth_txt, append_class_to_id)
        if (db == 'hyperpartisan'):
            return
            ↪  self.get_documents_DB_HYPERPARTISAN(documents_xml_folder,
            ↪  truth_txt, append_class_to_id)
        if (db == 'hyperpartisan_split_42'):
            return
            ↪  self.get_documents_DB_HYPERPARTISAN_split(documents_xml_folder,
            ↪  truth_txt, append_class_to_id)

        return []
    ...

    def calc_IR(self, result_df, positive_class='true'):
        '''
        Calculates IR attributes suggested in the research:
            CLASS_0_BM25_AVG
            CLASS_0_BM25_COUNT
            CLASS_0_BM25_SUM
            CLASS_1_BM25_AVG
            CLASS_1_BM25_COUNT
            CLASS_1_BM25_SUM
        and returns them as a dictionary.

        Parameters
        ----------
        result_df : DataFrame
            A query result dataframe produced by the query methods.
            Must have the columns:
                * score
                * class

        positive_class : str
            Specifies which 'class' is the positive class.
        '''
        df = result_df.copy()
        CLASS_0 = df.loc[(df['class'] != positive_class)]['score']
        CLASS_1 = df.loc[(df['class'] == positive_class)]['score']
        attrib_IR = {
            'CLASS_0_BM25_AVG': (0 if math.isnan(CLASS_0.mean())
                                 else CLASS_0.mean()),
            'CLASS_0_BM25_COUNT': CLASS_0.count(),
            'CLASS_0_BM25_SUM': CLASS_0.sum(),
```

```
            'CLASS_1_BM25_AVG': (0 if math.isnan(CLASS_1.mean())
                                 else CLASS_1.mean()),
            'CLASS_1_BM25_COUNT': CLASS_1.count(),
            'CLASS_1_BM25_SUM': CLASS_1.sum(),
        }
        return attrib_IR

    ...

    def insertArango(self, itemKey, itemBody):
        '''
        Inserts a document in the ArangoDB 'indexName' collection.

        Parameters
        ----------
        itemKey : str or number
            Document identifier

        itemBody : dict
            Document body/data.
        '''

        document = {'_key': itemKey}
        document.update(itemBody)
        self.arangoCollection.insert(document)

    ...

    def arango_query(self, query, ignore_first_result=False):
        '''
        Query ArangoDB view and returns a Pandas DataFrame with the
↪   results.

        Parameters
        ----------
        query : str
            Text to be queried to the view using BM25 analyzer.
        '''
        initial = time.time()
        escaped_query = str(query).replace('\\', '')
        escaped_query = str(escaped_query).replace("'", "\\\'")

        nResults = int(self.numberResults)
        if ignore_first_result:
            nResults += 1
        aqlquery = (f"FOR d IN {str(self.arangoViewName)} SEARCH "
                    + f"ANALYZER(d.text IN TOKENS('{escaped_query}'"
                    + f", 'text_en'), 'text_en') "
```

```
                + f"SORT BM25(d, {self.bm25_k1}, {self.bm25_b}) "
                + f"DESC LIMIT {nResults} "
                + f"LET sco = BM25(d, {self.bm25_k1}, "
                + f"{self.bm25_b}) RETURN {{ doc: d, score: sco }}")
    # print(aqlquery)
    cursor = self.arangoDb.aql.execute(query=aqlquery,
                                       count=True,
                                       batch_size=self.numberResults,
                                       optimizer_rules=['+all'],
                                       cache=True)
    item_list = []
    # print(1, time.time()-initial)
    initial = time.time()
    for item in cursor.batch():
        # print(item)
        item_list.append([item['score'],
         item['doc']['_id'].split('/')[-1],
                          item['doc']['class']])
    # print(2, time.time()-initial)
    if ignore_first_result and (len(item_list) > 0):
        item_list.pop(0)
    return pd.DataFrame(item_list, columns=['score', 'id', 'class'])

...

def arango_get_IR_variables(self, query, positive_class='true',
    ignore_first_result=False):
    '''
     Query ArangoDB view and returns a dict with the IR variables.

    Parameters
    ----------
    query : str
        Text to be queried to the view using BM25 analyzer.
    '''
    result_df = self.arango_query(query,
     ignore_first_result=ignore_first_result)

    return self.calc_IR(result_df=result_df,
     positive_class=positive_class)

...
```