Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Uma Abordagem Eficiente para Classificação de Textos Baseada em Compressão

Bruno Vargas de Souza

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientador
Prof. Dr. Pedro Garcia Freitas

Brasília
2025

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Uma Abordagem Eficiente para Classificação de Textos Baseada em Compressão

Bruno Vargas de Souza

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof. Dr. Pedro Garcia Freitas (Orientador)
CIC/UnB

Prof. Dr. Luís Paulo Faina Garcia
CIC/UnB

Prof.a Dr.a Aletéia Patrícia Favacho de Araújo
CIC/UnB

Prof. Dr. Marcelo Grandi Mandelli
Coordenador do Bacharelado em Ciência da Computação

Brasília, 08 de dezembro de 2025

Dedicatória

Dedico este trabalho aos meus pais e ao meu irmão. Vocês não foram apenas meus maiores incentivadores, mas a base de quem sou hoje. Obrigado por oferecerem todo o suporte, o amor e, muitas vezes, o sacrifício necessário para que eu pudesse priorizar os meus estudos. Cada passo dessa caminhada tem a marca de vocês. Vocês fazem parte desta conquista!

Agradecimentos

Agradeço ao meu orientador, Prof. Pedro Garcia Freitas, por conduzir este trabalho com tanta dedicação. Obrigado por ser sempre atencioso, prestativo e paciente em cada etapa da produção. Sua orientação foi fundamental não apenas para este texto, mas para o meu amadurecimento acadêmico.

Ao Prof. Vinicius Borges, deixo um agradecimento especial por ter sido o grande responsável por despertar meu interesse pelas áreas de Algoritmos, Processamento de Linguagem Natural e Inteligência Artificial. Suas aulas inspiradoras e seu suporte constante foram decisivos para que eu escolhesse trilhar este caminho.

Aos meus amigos e parceiros de pesquisa, Victor Hugo e Ana Sofia. Obrigado por me acompanharem durante toda essa jornada, dividindo não apenas a autoria de estudos e publicações, mas também as angústias e alegrias da vida acadêmica. O incentivo mútuo durante a escrita deste trabalho tornou o processo muito mais leve.

Ao Enzo Yoshio e à Isabel Starling, meus eternos parceiros de time. Vocês foram pilares fundamentais da minha vivência na UnB. As horas que passamos treinando juntos ao UnBalloon e resolvendo problemas de programação competitiva criaram um laço que vai muito além da universidade. Sou grato por cada desafio que superamos juntos e por tudo o que aprendi ao lado de vocês

Estendo minha gratidão a todos os demais familiares e amigos que, de perto ou de longe, se fizeram presentes, torceram por mim e me apoiaram durante essa caminhada. O carinho de vocês foi indispensável.

À Universidade de Brasília (UnB) e ao Departamento de Ciência da Computação, pela excelência no ensino e pelas oportunidades oferecidas.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), por meio do Acesso ao Portal de Periódicos.

Resumo

Nos últimos anos, a comunidade de Aprendizado de Máquina tem desenvolvido modelos cada vez mais complexos para classificação de textos, especialmente com o avanço dos Large Language Models (LLMs), que apresentam bom desempenho, mas exigem alta capacidade computacional e grandes volumes de dados rotulados, o que limita seu uso em cenários com poucos recursos. Como alternativa, métodos baseados em compressão têm sido estudados por seu baixo custo computacional, utilizando a Distância de Compressão Normalizada (do inglês, *Normalized Compression Distance*) (NCD), que usa a compressão para medir a similaridade entre textos, em conjunto com classificadores K-vizinhos mais Próximos (do inglês, *K-Nearest Neighbors*) (KNN), embora a busca exaustiva por vizinhos mais próximos represente um gargalo de desempenho. Este trabalho propõe um método de classificação textual baseado em compressão que utiliza uma *Burkhard-Keller Tree* (BKTree) para otimizar essa busca e compara diferentes algoritmos de compressão sem perda no *dataset* Fake News Filipino. Os resultados experimentais mostram que o método mantém desempenho preditivo semelhante ao de abordagens mais complexas, com ganhos de eficiência expressivos, incluindo acelerações de 20×, 25×, 7×, 6,6×, 8×, 10×, 1,4×, 1,9×, 11× e 12× para os compressores Brotli, FSST, LZ4, LZAV, LZF, QuickLZ, Shoco, Smaz, Snappy e ZLib, respectivamente, configurando uma alternativa eficiente e escalável para cenários com recursos limitados.

Palavras-chave: Classificação de textos, Compressão, BK-Tree, KNN, Eficiência computacional

Abstract

In recent years, the Machine Learning community has developed increasingly complex models for text classification, especially with the advancement of Large Language Models (LLMs), which demonstrate good performance but require high computational capacity and large volumes of labeled data, limiting their use in low-resource scenarios. As an alternative, compression-based methods have been studied for their low computational cost, utilizing the Normalized Compression Distance (NCD), which uses compression to measure similarity between texts, in conjunction with K-Nearest Neighbors (KNN) classifiers, although the exhaustive search for nearest neighbors represents a performance bottleneck. This work proposes a compression-based text classification method that utilizes a Burkhard-Keller Tree (BK-Tree) to optimize this search and compares different lossless compression algorithms on the Fake News Filipino dataset. Experimental results show that the method maintains predictive performance similar to complex approaches, with significant efficiency gains, including speedups of 20×, 25×, 7×, 6.6×, 8×, 10×, 1.4×, 1.9×, 11×, and 12× for Brotli, FSST, LZ4, LZAV, LZF, QuickLZ, Shoco, Smaz, Snappy, and ZLib compressors, respectively, configuring an efficient and scalable alternative for low-resource scenarios.

Keywords: Text classification, Compression, BK-Tree, KNN, Computational efficiency

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

API Interface de Programação de Aplicações (do inglês, *Application Programming Interface*).

BERT *Bidirectional Encoder Representations from Transformers.*

BKTree *Burkhard-Keller Tree.*

BTree *Ball Tree.*

BWT Transformada de Burrows-Wheeler (do inglês, *Burrows-Wheeler Transform*).

CDM Métrica de Distância de Compressão (do inglês, *Compression Distance Metric*).

CLM Modelos de Linguagem Baseados em Compressão (do inglês, *Compression-based Language Models*).

CPU Unidade Central de Processamento (do inglês, *Central Processing Unit*).

DNN Redes Neurais Profundas (do inglês *Deep Neural Networks*).

GCC Coleção de Compiladores GNU (do inglês *GNU Compiler Collection*).

GRU Unidade Recorrente com Portas (do inglês, *Gated Recurrent Unit*).

KDTree *K-Dimensional Tree.*

KNN K-vizinhos mais Próximos (do inglês, *K-Nearest Neighbors*).

LLM Large Language Model.

LSTM *Long Short-Term Memory.*

MIMD Múltiplas Instruções, Múltiplos Dados (do inglês, *Multiple Instruction Multiple Data*).

NCD Distância de Compressão Normalizada (do inglês, *Normalized Compression Distance*).

OpenMP *Open Multi-Processing*.

PLN Processamento de Linguagem Natural.

PPM Predição por Correspondência Parcial (do inglês, *Prediction by Partial Matching*).

RNN Redes Neurais Recorrentes (do inglês, *Recurrent Neural Networks*).

SIMD Instrução Única, Múltiplos Dados (do inglês, *Single Instruction Multiple Data*).

TF-IDF Frequência do Termo-Inverso da Frequência nos Documentos (do inglês, *Term Frequency-Inverse Document Frequency*).

VPTree *Vantage-Point Tree*.

Capítulo 1

Introdução

A classificação automática de textos consolidou-se como uma tarefa central nas áreas de Aprendizado de Máquina e Processamento de Linguagem Natural (PLN) [1]. Essa relevância deve-se à capacidade desses sistemas de organizar e extrair valor de grandes volumes de dados não estruturados de forma eficiente. Na prática, tais algoritmos sustentam aplicações críticas para a sociedade moderna, como a filtragem de notícias falsas [2] e a detecção de *spam* [3]. Além disso, são fundamentais em domínios analíticos e técnicos, variando desde a análise de sentimentos [4] até a categorização automática de problemas de programação [5, 6].

Atualmente, o estado da arte é dominado por Redes Neurais Profundas (do inglês *Deep Neural Networks*) (DNN) e arquiteturas baseadas em Transformers, como o *Bidirectional Encoder Representations from Transformers* (BERT) e LLM [7, 8]. Embora esses modelos demonstrem desempenho superior impulsionado por vastos volumes de dados e mecanismos de atenção, essa hegemonia cobra um preço elevado. Tais arquiteturas exigem treinamento intensivo, hardware de alto desempenho (GPUs) e consomem quantidades significativas de energia [9]. Esse custo computacional limita severamente a aplicabilidade desses modelos em cenários de recursos escassos (*low-resource*), em dispositivos de borda ou em tarefas onde a simplicidade e a interpretabilidade seriam preferíveis.

Em resposta a essas limitações, a comunidade científica tem revisitado métodos *parameter-free* baseados em compressão. A fundamentação para tal alternativa reside na NCD [10], uma métrica que opera sob o princípio da teoria da informação de que textos semanticamente semelhantes compartilham redundância. O raciocínio intuitivo sugere que a concatenação de textos similares resulta em uma compressão muito mais eficiente do que o processamento isolado das partes, pois o algoritmo consegue reaproveitar padrões comuns para reduzir o tamanho final. Essa estratégia funciona como uma estimativa prática da Complexidade de Kolmogorov e permite classificar diretamente os dados brutos, eliminando assim a necessidade de etapas custosas como a engenharia de atributos ou a

vetorização.

Recentemente, [11] demonstraram empiricamente a validade dessa abordagem com o método `NPC_Gzip`, provando que a combinação simples de um compressor (como Gzip) com um classificador KNN pode rivalizar com redes neurais modernas em precisão para diversas tarefas. Contudo, a simplicidade desse método esbarra em um gargalo crítico de escalabilidade. A implementação original depende de uma busca exaustiva por força bruta, exigindo que cada nova consulta seja comprimida em conjunto com todas as instâncias do treinamento. Isso torna o processo computacionalmente proibitivo à medida que o volume de dados aumenta, dificultando sua adoção em larga escala.

O presente trabalho aborda diretamente essa lacuna, propondo uma reformulação estrutural do algoritmo de classificação. A solução desenvolvida baseia-se na integração de estruturas de dados métricas para indexação eficiente, combinada com o uso de computação paralela para acelerar o processo de inferência.

1.1 Justificativa

Diante desse cenário, a limitação de escalabilidade impõe uma restrição severa ao uso prático desses classificadores. Especificamente, a implementação ingênua do KNN com NCD exige o cálculo da distância entre o texto de consulta e todas as m instâncias do conjunto de treinamento. Considerando que a operação de compressão é computacionalmente custosa, a complexidade resultante, descrita como $O(m \cdot r \cdot n)$ para *strings* de tamanhos r e n, torna o método inviável para bases de dados extensas ou aplicações que demandem tempo de resposta rápido.

Além da questão de eficiência, existe uma necessidade clara de transparência nas decisões dos modelos. Enquanto as DNN funcionam muitas vezes como "caixas pretas"[12], dificultando a compreensão do motivo de uma classificação, o método proposto oferece uma vantagem direta, que é o fato de ser possível identificar exatamente quais exemplos do treinamento foram utilizados para classificar um novo texto. Essa capacidade de auditoria é fundamental em domínios sensíveis, tais como o jurídico, o médico ou o educacional, nos quais não basta apenas o resultado, mas também a justificativa por trás dele [12].

Portanto, este trabalho se justifica pela necessidade premente de transformar modelos teóricos promissores, frequentemente restritos ao ambiente acadêmico, em ferramentas tecnicamente aplicáveis no mundo real. A proposta preenche uma lacuna importante ao atacar o gargalo de escalabilidade que atualmente impede a adoção massiva de classificadores baseados em compressão. Ao mitigar o custo computacional excessivo sem sacrificar a acurácia, valida-se uma alternativa robusta às redes neurais profundas. Consequentemente, abrem-se portas para o desenvolvimento de sistemas de classificação que sejam,

simultaneamente, eficientes, transparentes e capazes de operar com total independência de vocabulário e idioma.

1.2 Objetivos

O objetivo geral deste trabalho consiste em desenvolver e validar uma abordagem computacionalmente eficiente para a classificação de textos baseada em compressão, visando superar os desafios de escalabilidade inerentes aos métodos atuais. Identifica-se que, apesar da robustez teórica das métricas de compressão, a sua aplicabilidade prática é frequentemente limitada pelo alto custo da busca exaustiva em grandes bases de dados. Nesse sentido, a proposta central foca na otimização do tempo de inferência, substituindo a verificação linear pela integração de estruturas de dados métricas e processamento paralelo. Com isso, busca-se entregar uma solução que mantenha a acurácia dos modelos originais, tornando-os viáveis para cenários que exigem respostas rápidas e uso racional de recursos.

Para alcançar este propósito, a pesquisa desdobra-se em objetivos específicos interconectados. Inicialmente, visa-se implementar uma estrutura de indexação baseada em BKTree [13], adaptando-a para organizar o espaço métrico gerado pela NCD e permitindo a poda eficiente de ramos de busca por meio da desigualdade triangular. Concomitantemente, busca-se desenvolver uma estratégia de busca paralela capaz de explorar a árvore de forma concorrente, maximizando o uso de arquiteturas *multi-core* para acelerar a recuperação dos vizinhos mais próximos. Adicionalmente, o trabalho propõe analisar comparativamente diversos algoritmos de compressão *lossless*, como Zstd, LZ4, Brotli e Snappy [14], para compreender o impacto de cada um no equilíbrio entre acurácia preditiva e velocidade. Por fim, avalia-se empiricamente o desempenho da solução, mensurando os ganhos de *speedup* e a manutenção das métricas de qualidade em relação à abordagem ingênua de força bruta.

1.3 Hipótese de Pesquisa

A hipótese deste trabalho estipula que a integração da estrutura BKTree com o processamento paralelo supera as limitações de escalabilidade da classificação baseada em NCD. A premissa central defende que a poda eficiente do espaço de busca, somada à execução concorrente, proporciona ganhos significativos de *speedup* e mantém a acurácia preditiva equivalente à da abordagem de força bruta. A confirmação desta hipótese viabiliza a aplicação prática de modelos baseados em compressão em cenários reais de classificação textual.

1.4 Organização da Monografia

A estrutura do restante deste trabalho segue uma ordem progressiva de fundamentação, proposição e validação. O Capítulo 2 estabelece a base teórica e revisa conceitos essenciais como KNN, NCD e a estrutura BKTree. Na sequência, os Capítulos 3 e 4 detalham, respectivamente, a arquitetura da solução para superar as limitações de escalabilidade e os aspectos concretos da implementação e da orquestração paralela. A validação empírica ocorre no Capítulo 5 através da análise de desempenho preditivo e curvas de *speedup*. Por fim, o Capítulo 6 apresenta as considerações finais, limitações do estudo e perspectivas para trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Este capítulo tem como objetivo apresentar os conceitos teóricos fundamentais para o entendimento da metodologia e dos experimentos realizados neste trabalho. Inicialmente, é feita uma breve revisão sobre a tarefa de classificação de textos, mostrando como as abordagens nessa área evoluíram ao longo do tempo. Na sequência, detalha-se o funcionamento do algoritmo KNN, destacando sua natureza não paramétrica e a importância da escolha da métrica de distância. Além disso, são exploradas as métricas de similaridade para *strings*, introduzindo a NCD e explicando como os compressores influenciam nesse cálculo. Por fim, descreve-se a BKTree, estrutura de dados essencial para a otimização de busca proposta neste estudo.

2.1 Classificação de Textos

A classificação de textos desempenha um papel central no Aprendizado de Máquina Supervisionado e serve de base para muitas aplicações de processamento de linguagem natural. A tarefa tem como meta a atribuição automática de categorias a documentos através da análise de padrões no texto. O histórico da área mostra mudanças importantes nas últimas décadas. O desenvolvimento começou com abordagens baseadas em regras manuais e símbolos, avançou para modelos estatísticos e chegou ao cenário atual dominado por representações distribuídas e arquiteturas de atenção [15].

Inicialmente, métodos baseados em contagem e modelos de linguagem simples foram amplamente utilizados. A representação *bag-of-words* [16] e o uso de pesos Frequência do Termo-Inverso da Frequência nos Documentos (do inglês, *Term Frequency-Inverse Document Frequency*) (TF-IDF) tornaram-se padrões para transformar documentos em vetores numéricos, facilitando a classificação e recuperação [17]. Nessa fase, algoritmos clássicos de classificação, como o KNN, foram consolidados como linhas de base teóricas e práticas, juntamente com classificadores probabilísticos simples (por exemplo, Naive Bayes),

que demonstraram bom desempenho na categorização textual com custo computacional reduzido [18, 19].

Com o aumento de *corpora* e poder computacional, modelos de linguagem baseados em *n-gramas* [20] e modelos probabilísticos mais robustos surgiram, seguidos por arquiteturas neurais. Redes Neurais Recorrentes (do inglês, *Recurrent Neural Networks*) (RNN) [21] e suas variantes *Long Short-Term Memory* (LSTM) [22] mostraram capacidade de capturar dependências sequenciais em textos, superando técnicas baseadas em *n-gramas* em diversas tarefas de PLN. A arquitetura LSTM, com seus mecanismos de portas (*gates*) para controlar o fluxo de informação, resolveu problemas de gradientes desaparecentes presentes em RNNs tradicionais, permitindo o aprendizado de dependências de longo prazo. Posteriormente, a Unidade Recorrente com Portas (do inglês, *Gated Recurrent Unit*) (GRU) surgiu como uma alternativa mais simples e eficiente, mantendo boa capacidade de modelagem sequencial [23]. Adicionalmente, abordagens convolucionais para texto revelaram eficácia na extração de padrões locais e na classificação de sentenças e documentos, utilizando filtros convolucionais para identificar *n-gramas* e estruturas sintáticas relevantes [24].

Uma segunda mudança importante aconteceu com a chegada das representações distribuídas (*embeddings*). Essa abordagem mapeia palavras e frases para espaços vetoriais densos e permite medir a similaridade semântica através de operações geométricas [25]. O uso de técnicas como Word2Vec [25] transformou a engenharia de características ao oferecer vetores pré-treinados que melhoraram a capacidade de generalização dos modelos. Mais tarde, o mecanismo de atenção e a arquitetura de transformadores trouxeram grandes progressos. Modelos baseados em atenção (p. ex. BERT) conseguem capturar relações de contexto profundas entre *tokens* e definiram novos padrões de desempenho em classificação de textos e tarefas afins [26, 7].

Mesmo com a popularidade das representações vetoriais, métodos focados na similaridade direta entre sequências continuam importantes. Eles são valiosos em situações com poucos dados, quando é preciso explicar a classificação ou lidar com restrições de idioma. Abordagens baseadas em distância (como Levenshtein [27]) mantêm seu valor prático em problemas onde erros de edição ou variações locais trazem informação. Da mesma forma, medidas baseadas em compressão (como a NCD) permitem comparar documentos sem depender de vocabulário, pois medem a informação compartilhada entre as sequências [10, 28, 29].

2.2 Algoritmo dos K-vizinhos mais Próximos

O algoritmo KNN representa uma das técnicas mais clássicas e intuitivas do aprendizado supervisionado. Cover e Hart apresentaram o método na década de 1960 [18] sob a

premissa de que exemplos com características similares costumam fazer parte da mesma classe. Ao contrário de modelos paramétricos [30, 31], que dependem de uma função de decisão explícita, o KNN opera de maneira não paramétrica e usa as próprias amostras de treinamento como referência para a classificação.

Na sua versão básica, o algoritmo trabalha com um conjunto de dados rotulados. Diante de uma nova entrada, o sistema calcula a distância até todos os elementos do conjunto de treinamento. Em seguida, seleciona os k vizinhos mais próximos e define a categoria da nova instância através de uma regra de decisão. A estratégia mais comum utiliza a *moda*, ou seja, a nova amostra recebe a classe mais frequente entre os vizinhos. Outra possibilidade envolve ponderar a influência de cada vizinho pela distância e dar mais peso aos exemplos mais próximos [32].

O valor de k exerce influência direta no desempenho do algoritmo. Valores pequenos tornam o modelo mais sensível a ruídos e *outliers*, enquanto valores muito grandes podem levar à perda de detalhes locais e à suavização excessiva do espaço de decisão. Assim, a escolha de k envolve um equilíbrio entre estabilidade e sensibilidade local [18, 33].

Uma implementação direta do KNN para classificação de textos, proposta por [11] no método `NPC_Gzip`, utiliza uma estratégia de força bruta. Nessa abordagem, para cada consulta, calcula-se a distância entre o texto de entrada e todas as instâncias do conjunto de treinamento, ordenando os resultados e selecionando os k vizinhos mais próximos para determinar a classe por voto majoritário. O Algoritmo 1 descreve formalmente esse procedimento.

Assim, embora essa implementação seja intuitiva e de fácil compreensão, sua complexidade computacional é $O(m)$ cálculos de distância por consulta, onde m é o tamanho do conjunto de treinamento. Quando a métrica de distância envolve operações custosas, como compressão no caso da NCD, esse custo pode tornar o método impraticável para conjuntos de dados grandes. Por essa razão, estruturas de indexação como a BKTree, a ser apresentada na Seção 2.5, oferecem alternativas mais eficientes para acelerar a busca por vizinhos, reduzindo significativamente o número de comparações necessárias.

A principal vantagem do KNN está em sua simplicidade e facilidade de interpretação. Como não há uma fase de treinamento propriamente dita, o algoritmo é classificado como um método de aprendizado *preguiçoso* (*lazy learning*) [34], realizando o esforço computacional apenas no momento da predição. Essa característica permite rastrear cada decisão, identificando exatamente quais vizinhos influenciaram a classificação final. Além disso, o KNN é flexível quanto ao tipo de dado analisado, podendo empregar diferentes métricas de distância conforme o domínio, por exemplo, a distância Euclidiana para dados contínuos, a distância de Hamming para dados binários, ou a distância de edição para *strings* [35].

Algorithm 1 Algoritmo KNN por força bruta para classificação de textos

Require: Conjunto de treinamento $\mathcal{D} = \{(\mathbf{s}_1, \mathbf{c}_1), \ldots, (\mathbf{s}_m, \mathbf{c}_m)\}$
Require: Texto de consulta $\mathbf{q}$
Require: Número de vizinhos $k \in \mathbb{N}^+$
Require: Métrica de distância $d(\cdot, \cdot)$
Ensure: Classe prevista para $\mathbf{q}$

```
Inicializar uma lista vazia de pares (distância, classe): L ← [ ]
for cada (s_i, c_i) em D do
    Calcular a distância entre q e s_i: dist_i ← d(q, s_i)
    Adicionar (dist_i, c_i) à lista L
Ordenar L em ordem crescente de distância
Selecionar os k primeiros elementos: N ← {(dist_j, c_j) | primeiros k elementos}
Extrair as classes dos vizinhos mais próximos: C_{N_K} ← {c_j | (dist_j, c_j) ∈ N}
Definir a classe prevista como a moda das classes vizinhas: c̄ ← Mode(C_{N_K})
return c̄
```

Entretanto, o desempenho do KNN depende fortemente da métrica de similaridade adotada, pois ela define o que significa "ser semelhante". Em domínios textuais, métricas tradicionais como as distâncias Euclidiana ou Manhattan tendem a ser pouco informativas devido à alta dimensionalidade e à esparsidade das representações. Por essa razão, este trabalho dedica a próxima seção à discussão de métricas especializadas para sequências textuais, incluindo medidas de edição e de compressão, que serão fundamentais para a formulação do classificador proposto.

2.3 Métricas de Similaridade para *Strings*

As métricas de similaridade textual são funções matemáticas que expressam numericamente o nível de parentesco ou distância entre duas cadeias de caracteres. Essas ferramentas convertem a comparação entre textos em dados quantificáveis e permitem o tratamento computacional da linguagem. Tais medidas assumem papel central em tarefas de classificação e recuperação de informação ao definir a vizinhança entre exemplos, onde valores baixos de distância indicam alta similaridade semântica ou estrutural.

Historicamente, as medidas de edição (*edit distances*) surgiram inicialmente para representar alterações básicas entre cadeias, incluindo inserções, remoções e substituições. A natureza exata dessas métricas favoreceu o uso em corretores ortográficos e na comparação de textos curtos. Ao mesmo tempo, a área progrediu com coeficientes que analisam correspondências locais, como *n-gramas*, e regras que valorizam posições específicas, como prefixos. Essas técnicas ganharam espaço em problemas pontuais, como o emparelhamento de nomes e a eliminação de duplicatas em bancos de dados [36].

Mais recentemente, abordagens inspiradas em princípios de informação motivaram o uso de medidas de similaridade baseadas em compressão, como a NCD. Essas métricas quantificam a informação compartilhada entre duas sequências sem a necessidade de recorrer a vocabulários fixos ou modelos pré-treinados [10, 28]. Nesse contexto, a eficiência do processo de compressão conjunta reflete diretamente o quanto duas cadeias compartilham padrões informacionais subjacentes. Essa característica fundamenta-se na teoria da complexidade de Kolmogorov [28], oferecendo uma perspectiva universal para a comparação de dados.

2.3.1 Distâncias Empregadas e Breve Descrição

Levenshtein. A distância de Levenshtein mede o número mínimo de operações atômicas (inserção, deleção ou substituição de caracteres) necessárias para transformar uma cadeia x em outra y [37]. É amplamente utilizada em busca aproximada e correção ortográfica. O cálculo clássico, realizado por programação dinâmica, possui custo de tempo $O(|x| \cdot |y|)$ e pode ter seu custo de espaço reduzido para $O(\min\{|x|, |y|\})$ mediante otimizações conhecidas [38].

Damerau–Levenshtein. Essa variação amplia o conjunto de operações permitidas ao incluir transposições de caracteres adjacentes, além de inserção, deleção e substituição [39]. Essa inclusão torna a métrica mais adequada para capturar erros humanos comuns de digitação, como a troca de letras vizinhas. É frequentemente aplicada em deduplicação e correção de entradas fornecidas por usuários [40].

Jaro–Winkler. O coeficiente de Jaro avalia a similaridade com base em correspondências e transposições dentro de uma janela ajustável. O refinamento proposto por Winkler adiciona um fator de bonificação para prefixos coincidentes (*prefix scale*), valorizando cadeias que compartilham o mesmo início [41, 42]. Essa métrica é amplamente utilizada em *record linkage* e emparelhamento de nomes, apresentando boa tolerância a pequenas discrepâncias internas e dando ênfase a coincidências no início das palavras [43].

Coeficiente de Dice (Strike-a-Match) - Simon White. Frequentemente referida como “Strike-a-Match” na literatura de desenvolvimento, essa heurística compara os conjuntos de bigramas (ou *n-gramas* de ordem baixa) formados a partir de duas cadeias e calcula uma similaridade baseada no coeficiente de Dice/Sørensen entre esses conjuntos. Essa abordagem é eficiente e, em geral, robusta a pequenas reordenações e variações leves de forma. Por depender de *n-gramas*, sua sensibilidade está relacionada ao comprimento das cadeias e à granularidade adotada [44].

2.3.2 Distância de Compressão Normalizada (NCD)

A NCD [10] é uma medida de similaridade derivada de princípios de informação e compressão. A relação entre compressão e similaridade textual decorre do princípio de que textos semelhantes compartilham padrões internos. Quando duas sequências x e y possuem trechos em comum, a compressão conjunta tende a ser mais eficiente que a compressão individual, resultando em $C(xy) < C(x) + C(y)$. Essa propriedade fundamenta o uso da NCD para estimar a semelhança entre cadeias: valores menores indicam maior sobreposição informacional entre os textos.

Proposta por Cilibrasi e Vitányi [10] como uma medida universal de similaridade baseada em compressão, a NCD deriva dos princípios de complexidade de Kolmogorov [28]. A sua definição formal é expressa pela Equação 2.1:

$$\mathrm{NCD}(x, y) = \frac{C(xy) - \min\{C(x), C(y)\}}{\max\{C(x), C(y)\}}, \tag{2.1}$$

em que $C(\cdot)$ representa o tamanho comprimido (um valor inteiro) retornado por um compressor prático. Matematicamente, o resultado dessa equação pertence ao intervalo real contínuo $[0, 1]$. No entanto, para adequar a métrica à estrutura de indexação proposta neste trabalho, aplica-se uma discretização que mapeia esses valores para o intervalo de inteiros $[0, 100]$. A NCD é atrativa por ser independente de vocabulário e aplicável a qualquer sequência representável, mas sua eficácia depende fortemente do compressor adotado e apresenta custo computacional elevado devido à necessidade de compressões conjuntas para cada par comparado.

Além da NCD, a literatura apresenta outras métricas e abordagens baseadas em compressão. A Métrica de Distância de Compressão (do inglês, *Compression Distance Metric*) (CDM) utiliza formulações alternativas de normalização e oferece perspectivas complementares sobre a similaridade entre sequências [29]. Por sua vez, as Modelos de Linguagem Baseados em Compressão (do inglês, *Compression-based Language Models*) (CLM) exploram a entropia cruzada entre um documento e modelos de linguagem construídos a partir de compressores, permitindo a classificação textual através da comparação de probabilidades de compressão [45, 46]. Essas abordagens compartilham o princípio fundamental de que a eficiência de compressão reflete padrões informacionais comuns, mas diferem na forma como quantificam essa relação.

2.3.3 Observações Comparativas e Critérios para Experimentos

As medidas descritas apresentam vantagens e limitações distintas. As distâncias de Levenshtein [27] e Damerau–Levenshtein [39] capturam com precisão alterações locais, o que

é essencial em tarefas como, por exemplo, correção ortográfica. A métrica Jaro–Winkler [41, 42] favorece correspondências com prefixos idênticos, característica útil na comparação de nomes próprios. A heurística Strike-a-Match [44], também chamada de similaridade de Simon–White [44], baseada em *bigramas*, é rápida e robusta a pequenas reordenações. Já a NCD [47] oferece uma perspectiva mais ampla e independente de vocabulário, mas exige maior custo computacional e depende da qualidade do compressor utilizado. Para fins experimentais, essas métricas serão aplicadas como critérios comparativos, avaliando acurácia, sensibilidade e custo computacional, tanto de forma isolada quanto combinadas a estratégias de indexação (como BKTree).

2.4 Compressores

Os compressores são algoritmos que reduzem o volume de arquivos ou sequências de dados através da identificação de repetições e redundâncias [14]. Na aplicação da NCD, esses componentes assumem papel vital, pois definem os valores de $C(x)$ e $C(xy)$. Essas funções retornam, invariavelmente, um valor inteiro que corresponde ao tamanho total (em bytes ou bits) da versão comprimida da sequência isolada e da concatenação. Logo, a seleção do algoritmo impacta diretamente a eficácia da métrica em representar a similaridade textual.

2.4.1 Compressão sem Perdas

Neste trabalho utiliza exclusivamente compressores sem perdas (do inglês, *lossless*) [14]. A categoria engloba métodos capazes de reduzir o tamanho de uma *string* e garantir a recuperação exata do dado original, bit a bit, a partir da versão comprimida [48]. O processo assegura a integridade total do conteúdo após a descompressão. Tal característica é indispensável na análise de textos, já que pequenas alterações de caracteres podem mudar completamente o sentido de uma sequência. O método difere da compressão com perdas (*lossy*) [14], comum em áudio e imagem [48], que descarta dados menos perceptíveis para aumentar a taxa de compactação, prática inviável em comparações de *strings*.

2.4.2 Tipos de Compressores

De maneira geral, os compressores *lossless* empregados para textos podem ser agrupados em três categorias principais:

- Baseados em dicionário: identificam *substrings* recorrentes e as substituem por referências a um "dicionário"de padrões previamente encontrados. Compressores como LZ77, LZ78 e LZW são exemplos clássicos desse tipo [49, 50]. Eles apresentam bom desempenho em textos com muitas repetições lexicais ou estruturas regulares;

- Por modelagem estatística: constroem modelos probabilísticos do contexto anterior para prever o próximo símbolo, atribuindo códigos mais curtos a elementos mais prováveis. Métodos como o Predição por Correspondência Parcial (do inglês, *Prediction by Partial Matching*) (PPM) e a Transformada de Burrows-Wheeler (do inglês, *Burrows-Wheeler Transform*) (BWT) pertencem a essa categoria [51, 52]. Esses algoritmos são mais sensíveis a dependências de longo alcance e capturam características estilísticas do texto;
- Híbridos: combinam técnicas de dicionário e de modelagem estatística, buscando equilibrar eficiência e profundidade de compressão. Um exemplo é o compressor DEFLATE, que combina LZ77 com codificação de Huffman, sendo amplamente utilizado em formatos como ZIP e GZIP [53].

2.4.3 Impacto Prático

A definição do algoritmo de compressão impacta diretamente a precisão e a eficiência do cálculo da NCD. Compressores robustos, capazes de capturar dependências distantes e contextos complexos, costumam gerar medidas de similaridade mais consistentes. Contudo, tal sofisticação implica maior custo computacional e aumenta o tempo de resposta das consultas. Em contrapartida, algoritmos mais simples garantem velocidade computacional superior, embora apresentem risco de falha na detecção de semelhanças semânticas ou estruturais sutis [10].

A eficácia da ferramenta depende também da natureza do algoritmo. Métodos baseados em dicionário, por exemplo, demonstram maior aptidão para identificar repetições locais e padrões lexicais exatos [14]. Já os modelos estatísticos sobressaem na percepção de dependências de estilo e estruturas gramaticais abstratas [45]. Qualquer que seja o tipo, o tamanho final do arquivo comprimido atua como indicador direto da complexidade informacional: sequências redundantes geram arquivos compactos, enquanto textos ricos produzem arquivos maiores [28].

Do ponto de vista da implementação, algumas estratégias podem ser adotadas para mitigar o custo computacional sem prejudicar o resultado final. Técnicas como o pré-cálculo de $C(x)$ para o conjunto de treinamento e a limitação do comprimento máximo das sequências são essenciais para reduzir o tempo de execução. Além disso, o uso de amostragem controlada pode acelerar o processo sem comprometer drasticamente a precisão da medida de distância. Dessa forma, o compressor não atua apenas como um componente auxiliar da NCD, mas sim como uma variável experimental crítica que influencia a capacidade do método de refletir a similaridade textual de maneira fiel e eficiente.

2.5 BKTree e Buscas em Espaços Métricos Discretos

Burkhard e Keller [13] propuseram a BKTree como uma estrutura de dados para permitir buscas eficientes em espaços métricos discretos. A sua compatibilidade com métricas de edição (por exemplo, distância de Levenshtein [37]) ou informacionais (como a NCD [47]) a torna ideal para acelerar o processo de busca de vizinhos em algoritmos como o KNN, reduzindo significativamente o número de comparações necessárias. O funcionamento dessa estrutura baseia-se na propriedade fundamental da desigualdade triangular [54], que define uma relação entre as distâncias de três elementos quaisquer a, b e c pertencentes a um mesmo espaço métrico. Tal propriedade é expressa na Equação 2.2.

$$d(a, c) \leq d(a, b) + d(b, c) \tag{2.2}$$

ou, de forma equivalente, como apresentado na Equação 2.3,

$$|d(a, b) - d(b, c)| \leq d(a, c) \leq d(a, b) + d(b, c). \tag{2.3}$$

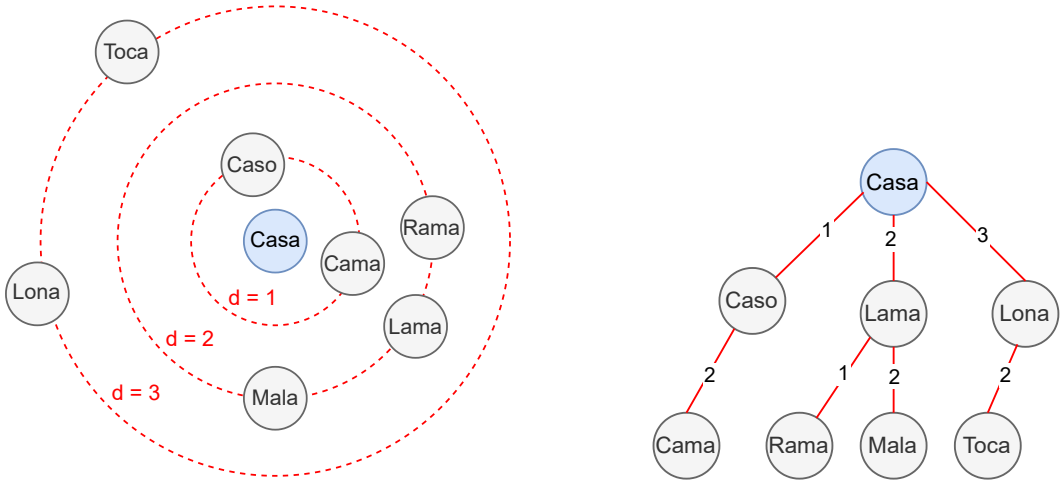
Essa relação é essencial, pois permite restringir as regiões de busca, eliminando cálculos desnecessários durante a procura por elementos semelhantes. Em outras palavras, a estrutura se beneficia da geometria do espaço métrico para reduzir o número de comparações e, consequentemente, o custo computacional.

2.5.1 Construção da Árvore

Para construir a BKTree, parte-se de um conjunto de elementos $S = \{x_1, x_2, \ldots, x_n\}$ e de uma função de distância $d : S \times S \to \mathbb{R}^+$. O primeiro elemento do conjunto é escolhido como raiz, e os demais são inseridos recursivamente. Para cada novo elemento x_i, calcula-se a distância $d(x_i, x_j)$ em relação ao nó atual x_j, sendo essa distância utilizada como rótulo da aresta que conecta o novo nó ao nó pai. Cada nó, portanto, mantém um conjunto de filhos indexados pelas distâncias inteiras resultantes, conforme o mapeamento apresentado na Equação 2.4:

$$\text{children}(x_j) = \{(d(x_j, x_k), x_k) \mid x_k \text{ é filho de } x_j\} \tag{2.4}$$

Essa organização hierárquica permite que a árvore seja explorada seletivamente durante a busca, evitando o cálculo redundante de distâncias entre elementos que não podem satisfazer os critérios de similaridade definidos. Para ilustrar esse processo, a Figura 2.1 demonstra como o agrupamento por distância radial no espaço métrico (Figura 2.1a) é



(a) Visualização do Espaço Métrico (b) Estrutura Hierárquica BKTree

Figura 2.1: Comparação visual da organização de dados utilizando a palavra “Casa” como raíz. Para fins de clareza didática, utilizou-se neste exemplo a Distância de Levenshtein para demonstrar as relações de proximidade. (a) No espaço métrico, observa-se o agrupamento por distância radial. (b) Na BKTree, esse agrupamento é convertido em uma hierarquia, onde colisões (como em “Cama”, “Rama”, “Mala” e “Toca”) são resolvidas criando subníveis.

convertido diretamente na estrutura hierárquica de nós e arestas da BKTree (Figura 2.1b), utilizando a palavra “Casa” como exemplo de raiz.

2.5.2 Processo de Busca

No processo de busca, dada uma consulta q e um raio máximo r, calcula-se a distância $d(q, x_j)$ em relação ao nó atual x_j. Com base na desigualdade triangular, apenas são explorados aqueles nós cujas arestas $d(x_j, x_k)$ satisfazem a Equação 2.5.

$$d(q, x_j) - r \leq d(x_j, x_k) \leq d(q, x_j) + r. \tag{2.5}$$

Dessa forma, é possível eliminar subárvores inteiras que não contenham elementos dentro do raio de busca, reduzindo significativamente o número de comparações necessárias. Esse mecanismo de poda faz com que a BKTree seja especialmente eficiente em espaços nos quais a função de distância apresenta distribuição desigual de valores, como ocorre com distâncias de edição entre cadeias de caracteres.

A Figura 2.2 ilustra uma simulação de busca pela palavra $q =$ “Cama” com um raio de tolerância fixo $r = 1$. Na raiz (“Casa”), a distância para a consulta é $d(\text{Casa}, \text{Cama}) = 1$.

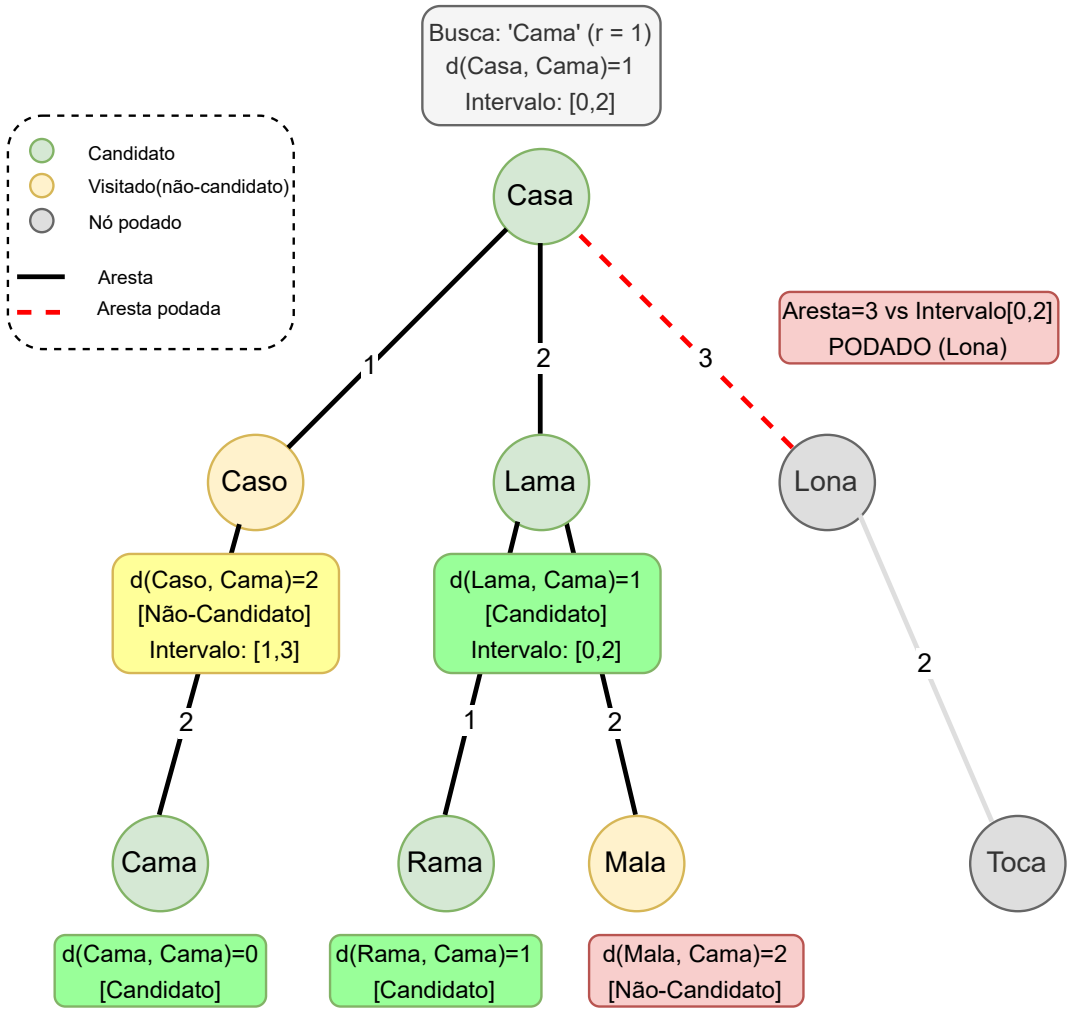


Figura 2.2: Exemplo de busca na BKTree com a consulta q = "Cama" e raio $r = 1$. Observa-se a poda do ramo "Lona" na raiz e a filtragem do nó "Mala".

Aplicando a inequação do intervalo de busca, temos $[1-1, 1+1]$, ou seja, apenas filhos conectados por arestas de peso entre 0 e 2 devem ser visitados. Consequentemente, a aresta para o nó "Lona" (peso 3) é imediatamente podada, descartando toda a sua subárvore sem a necessidade de comparar "Cama" com "Lona" ou seus descendentes. Por outro lado, ao visitar o nó "Lama" (onde $d = 1$, intervalo $[0, 2]$), o nó filho "Mala" é acessado pois sua aresta (peso 2) está no intervalo, mas é descartado do resultado final pois sua distância real para a consulta excede o raio r ($d(\text{Mala}, \text{Cama}) = 2 > 1$).

É importante ressaltar que, no exemplo acima, o valor de r foi mantido fixo por questões didáticas. Entretanto, na aplicação do algoritmo combinada com o método do KNN, o raio de busca r torna-se dinâmico. O processo inicia-se com $r = \infty$ (ou um valor máximo

arbitrário) até que os primeiros K candidatos sejam encontrados (por exemplo, $K = 5$). Uma vez preenchida a lista com K vizinhos, r é atualizado para a distância do candidato mais distante dessa lista (o k-ésimo vizinho). À medida que a busca progride e novos nós com distâncias menores são encontrados, a lista de melhores candidatos é atualizada e o valor de r é reduzido. Esse ajuste contínuo torna o critério de poda progressivamente mais restritivo, otimizando o desempenho ao descartar ramos que, com certeza, não contêm candidatos melhores que os já encontrados.

2.5.3 Complexidade e Aplicabilidade

A complexidade de construção da BKTree, no pior caso, é $O(n^2)$, uma vez que cada elemento pode ser comparado com todos os demais já inseridos. Entretanto, na prática, a estrutura tende a apresentar comportamento próximo de $O(n \log n)$, devido à natureza hierárquica da inserção e à eficiência da poda durante o processo. As buscas também possuem custo $O(n)$ no pior cenário, mas geralmente são sublineares, especialmente quando o espaço métrico possui boa separabilidade [55].

Diferentemente de estruturas como *K-Dimensional Tree* (KDTree) ou *Ball Tree* (BTree), que são projetadas para espaços vetoriais contínuos e métricas euclidianas, a BKTree é adequada para espaços discretos. Por essa razão, tem sido amplamente empregada em aplicações que envolvem comparação de cadeias de caracteres, como correção ortográfica [56], recuperação de informações e classificação de textos. Sua compatibilidade com métricas de edição, como Levenshtein e Damerau-Levenshtein, e também com medidas de similaridade informacional, como a NCD, torna-a uma estrutura versátil para tarefas de busca aproximada em grandes coleções de dados textuais.

2.6 Computação Paralela

A eficiência na busca pelos vizinhos mais próximos dentro de uma BKTree é o fator determinante para a viabilidade do classificador proposto. Embora a estrutura da árvore permita uma redução drástica no espaço de busca por meio da desigualdade triangular, o custo acumulado de múltiplas computações da NCD pode tornar a inferência lenta em bases de dados extensas. Nesse cenário, a computação paralela é aplicada para acelerar a travessia dos ramos da árvore, permitindo que múltiplos núcleos de processamento colaborem na identificação dos candidatos de forma simultânea e independente [57].

2.6.1 Taxonomia de Flynn

A organização desse processamento paralelo fundamenta-se na Taxonomia de Flynn [58], que classifica as arquiteturas conforme o fluxo de instruções e dados. A arquitetura Múltiplas Instruções, Múltiplos Dados (do inglês, *Multiple Instruction Multiple Data*) (MIMD) é a que melhor descreve o funcionamento dos processadores multinúcleo modernos. Em sistemas MIMD, cada núcleo possui autonomia para executar fluxos de instruções distintos sobre subárvores variadas da BKTree. Adicionalmente, o conceito de Instrução Única, Múltiplos Dados (do inglês, *Single Instruction Multiple Data*) (SIMD) é relevante no nível de instrução, permitindo que o hardware processe vetores de dados simultaneamente para otimizar operações de baixo nível dentro de cada thread de execução [59].

2.6.2 Programação com OpenMP

A implementação da busca paralela foi realizada utilizando o padrão *Open Multi-Processing* (OpenMP) [60], uma interface baseada em diretivas de compilador amplamente adotada em computação de alto desempenho [61]. O funcionamento do OpenMP segue o modelo *fork-join*, onde uma thread mestre coordena a execução e, ao encontrar uma região paralela, cria threads trabalhadoras para processar os nós da árvore de forma concorrente. Ao término da exploração dos ramos, as threads são sincronizadas e o controle retorna à thread mestre para a finalização da consulta.

A gestão de memória no OpenMP é crucial para assegurar que a busca pelos vizinhos mais próximos ocorra de forma correta. O modelo permite definir variáveis como privadas a cada thread ou compartilhadas entre o grupo de execução. Na BKTree, a estrutura da árvore e os parâmetros de busca são compartilhados, enquanto os buffers de cálculo locais são privados para evitar condições de corrida [61]. Para garantir a integridade da fila de vizinhos e a atualização segura do limite de corte, utilizam-se seções críticas e operações atômicas, permitindo que o acesso concorrente aos dados globais não comprometa o resultado final.

2.7 Métricas e Comparação de Eficiência

A avaliação do desempenho preditivo dos modelos utilizou métricas amplamente consolidadas em tarefas de classificação [62], como acurácia, precisão, *recall* e *F1-score* (macro). O cálculo dessas medidas fundamenta-se nos quatro possíveis desfechos de uma classificação binária derivados da matriz de confusão. Esses desfechos incluem o Verdadeiro Positivo (VP) e o Verdadeiro Negativo (VN), que representam os acertos do modelo para as classes positiva e negativa, respectivamente. Em contrapartida, os erros recebem

mapeamento como Falso Positivo (FP), quando a classe positiva é predita incorretamente, e Falso Negativo (FN), quando o modelo falha ao identificar uma instância que era de fato positiva.

A acurácia é a métrica mais intuitiva para avaliar um classificador, pois indica a proporção global de acertos do sistema, conforme expresso pela Equação 2.6:

$$\text{Acurácia} = \frac{VP + VN}{VP + VN + FP + FN} \tag{2.6}$$

A precisão foca na qualidade das predições positivas, expressando a fração de instâncias classificadas como positivas que realmente pertencem a essa classe (Equação 2.7), enquanto o *recall* avalia a completude, quantificando a capacidade do modelo de recuperar as instâncias positivas reais (Equação 2.8):

$$\text{Precisão} = \frac{VP}{VP + FP} \tag{2.7}$$

$$\text{Recall} = \frac{VP}{VP + FN} \tag{2.8}$$

O *F1-score* macro, apresentado na Equação 2.9, surge como uma medida agregadora que estabelece a média harmônica entre precisão e *recall*, garantindo um equilíbrio na avaliação de todas as classes independentemente do volume de amostras.

$$\text{F1-score} = 2 \times \frac{\text{Precisão} \times \text{Recall}}{\text{Precisão} + \text{Recall}} \tag{2.9}$$

Este trabalho dedica especial atenção à avaliação do custo computacional das soluções. Para quantificar os ganhos de eficiência introduzidos pela proposta paralela discutida na Seção anterior, utilizam-se as métricas de *speedup* e eficiência, que permitem isolar os ganhos advindos da estrutura de dados daqueles provenientes da execução concorrente.

2.7.1 Speedup Relativo

O *speedup* relativo [59] é a métrica utilizada para comparar o desempenho do método proposto paralelizado diretamente com a implementação ingênua de referência. Esse valor indica quantas vezes a versão otimizada com BKTree é mais rápida que a abordagem de força bruta ao empregar o mesmo algoritmo de compressão. O cálculo segue a Equação 2.10:

$$\text{Speedup Relativo}(p) = \frac{\text{Tempo do método Naive}}{\text{Tempo da BKTree com } p \textit{ threads}} \tag{2.10}$$

2.7.2 Speedup Paralelo e Eficiência

O *speedup* paralelo mede o ganho de escalabilidade obtido ao migrar o método proposto de uma execução serial para uma execução com múltiplas *threads*, conforme a Equação 2.11. Essa métrica revela o quanto o algoritmo consegue se beneficiar do uso de múltiplos núcleos de processamento.

$$\text{Speedup Paralelo}(p) = \frac{\text{Tempo da BKTree com 1 } \textit{thread}}{\text{Tempo da BKTree com } p \textit{ threads}} \tag{2.11}$$

Complementarmente, a eficiência (E_p) é calculada para observar o aproveitamento real do hardware. Dessa forma, ela relaciona o *speedup* alcançado com o número de recursos empregados (p), sendo definida pela Equação 2.12:

$$E_p = \frac{\text{Speedup Paralelo}(p)}{p} \tag{2.12}$$

Capítulo 3

Solução Proposta

Este capítulo detalha a arquitetura desenvolvida para mitigar as limitações de desempenho identificadas no método `NPC_Gzip`. A solução concebida tem como base a substituição da busca exaustiva linear por uma abordagem hierárquica e concorrente. O modelo integra a estrutura de dados BKTree a um sistema de processamento paralelo. O objetivo central é viabilizar a escalabilidade da classificação baseada em compressão sem comprometer a acurácia da métrica NCD.

A Figura 3.1 apresenta a visão geral da solução e ilustra o fluxo desde a organização dos dados até a estratégia de consulta.

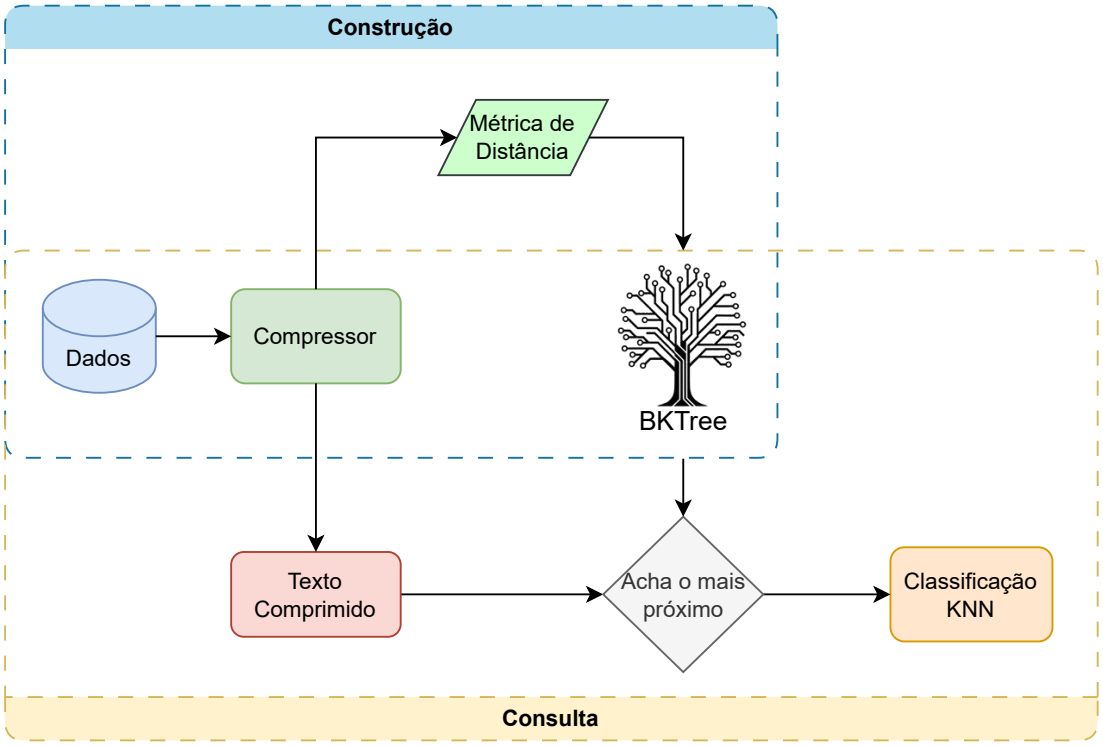


Figura 3.1: Arquitetura da solução: indexação via BKTree e classificação hierárquica baseada em NCD.

3.1 Estruturação do Espaço de Busca

A otimização do algoritmo KNN em cenários de alto custo computacional exige o abandono da força bruta em favor de estruturas de indexação eficientes. O uso de compressores impõe esse cenário. A premissa da solução consiste em reduzir a complexidade da consulta de linear $O(m)$ para sublinear ou logarítmica $O(\log m)$ [63]. No entanto, a natureza da métrica NCD impõe restrições específicas sobre qual estrutura de dados é adequada.

3.1.1 Limitações de Árvores Espaciais Tradicionais

Diversas estruturas de indexação consolidadas na literatura partem do pressuposto de que os dados residem em um espaço vetorial euclidiano. Exemplos incluem KDTree [64], BTree [65] e *Vantage-Point Tree* (VPTree) [63]. Tais métodos dependem de operações geométricas como o cálculo de centróides, divisões de eixos ou projeções ortogonais.

Essas premissas são inadequadas para a extensão do trabalho de [11]. A métrica NCD e as operações sobre *strings* não possuem representação vetorial nativa nem eixos coordenados explícitos. A tentativa de forçar uma representação vetorial (*embedding*) para utilizar árvores espaciais adicionaria uma camada de complexidade indesejada. Além disso,

causaria perda de informação e contradição com a proposta *parameter-free* da classificação por compressão.

3.1.2 Adoção da BKTree para Espaços Métricos Discretos

Diante da ausência de geometria euclidiana, a solução adota a BKTree [13]. O desenho desta estrutura contempla especificamente espaços métricos discretos. A justificativa para a escolha reside no fato de a BKTree não requerer coordenadas e exigir apenas que a função de distância respeite a desigualdade triangular.

Compressores reais podem apresentar leves desvios teóricos. Mesmo assim, a literatura demonstra que a NCD aproxima a desigualdade triangular de forma robusta o suficiente para permitir indexação métrica [10]. A BKTree permite organizar o conjunto de treinamento com base puramente na informatividade relativa entre os textos. A estrutura cria ramos para agrupar instâncias com taxas de compressão similares. Isso viabiliza a poda eficiente de subárvores inteiras durante a busca.

3.2 Modelo de Processamento Paralelo

A segunda vertente da solução proposta ataca o custo de Unidade Central de Processamento (do inglês, *Central Processing Unit*) (CPU) individual de cada operação de compressão. Mesmo com a poda eficiente da árvore, percorrer os ramos restantes de forma sequencial subutilizaria a capacidade dos processadores modernos.

O modelo arquitetural desenhado tem como princípio a decomposição da busca em tarefas granulares e independentes. Ao contrário da recursão tradicional onde cada passo bloqueia o anterior, a solução propõe um sistema produtor-consumidor dinâmico. A exploração de um nó da árvore gera novas tarefas para distribuição entre múltiplos núcleos de processamento.

Essa estratégia visa amortizar o custo de latência da compressão através do *throughput* elevado do paralelismo. O modelo prevê o uso de estruturas de sincronização globais para manter a consistência dos resultados. Isso garante que a versão paralela produza resultados determinísticos e idênticos à versão sequencial. O ganho de desempenho (*speedup*) em ambientes *multi-core* passa a ser significativo com essa abordagem.

Capítulo 4

Metodologia

Este capítulo descreve os procedimentos de implementação da solução arquitetural proposta. O texto detalha os algoritmos desenvolvidos para a construção da estrutura de dados. Também apresenta a adaptação da métrica NCD para o domínio discreto e os mecanismos de sincronização utilizados na orquestração da busca paralela.

4.1 Implementação e Construção da BKTree

A implementação da BKTree neste trabalho difere da versão clássica utilizada para distância de Levenshtein devido à natureza contínua da NCD. Para adequar a métrica à estrutura o projeto introduziu uma etapa de discretização. A estrutura opera eficientemente com arestas rotuladas por inteiros e a discretização viabiliza esse funcionamento.

Os valores de NCD estão originalmente no intervalo contínuo $[0, 1]$. O algoritmo submete esses valores a um fator de escala de 100 e realiza o arredondamento. Isso resulta em distâncias operacionais inteiras no intervalo $[0, 100]$. O valor 0 representa identidade informacional e o valor 100 denota máxima dissimilaridade. Essa transformação permite mapear os vizinhos em *buckets* discretos na árvore e otimiza a dispersão dos nós.

O procedimento de construção consta no Algoritmo 2. A execução ocorre de forma sequencial como uma etapa de pré-processamento. A inserção de cada nó respeita a hierarquia métrica e garante a organização necessária para a poda futura.

Algorithm 2 Construção de BKTree com Discretização de NCD

Require: Conjunto de treinamento $\mathcal{D}$.
Require: Função de NCD $d(\cdot, \cdot)$.
Ensure: Raiz da BKTree $\mathcal{T}_{\text{root}}$.

$\mathcal{T}_{\text{root}} \leftarrow$ null
for cada $(\mathbf{x}_i, y_i)$ em $\mathcal{D}$ **do**
    if $\mathcal{T}_{\text{root}} =$ null **then**
        $\mathcal{T}_{\text{root}} \leftarrow$ create_node$(\mathbf{x}_i, y_i)$
    else
        InsertNode$(\mathcal{T}_{\text{root}}, \mathbf{x}_i, y_i, d)$
return $\mathcal{T}_{\text{root}}$
procedure InsertNode($node$, $\mathbf{x}$, y, d)
    $raw\_dist \leftarrow d(node.\text{string}, \mathbf{x})$
    $dist \leftarrow \text{round}(raw\_dist \times 100)$ ▷ Discretização
    if não existe chave $dist$ em $node$.children **then**
        $node$.children$[dist] \leftarrow$ create_node$(\mathbf{x}, y)$
    else
        InsertNode($node$.children$[dist], \mathbf{x}, y, d$)

4.2 Orquestração da Busca Paralela

A metodologia de busca implementada substitui a recursão em profundidade tradicional. O sistema utiliza uma abordagem baseada em fila de tarefas (*Task Queue*) executada por um *pool* de *worker threads*.

A Figura 4.1 ilustra o fluxo de dados implementado. O funcionamento ocorre através de um ciclo de realimentação. O processo consome tarefas e pode gerar novas tarefas que retornam à fila caso satisfaçam os critérios de poda.

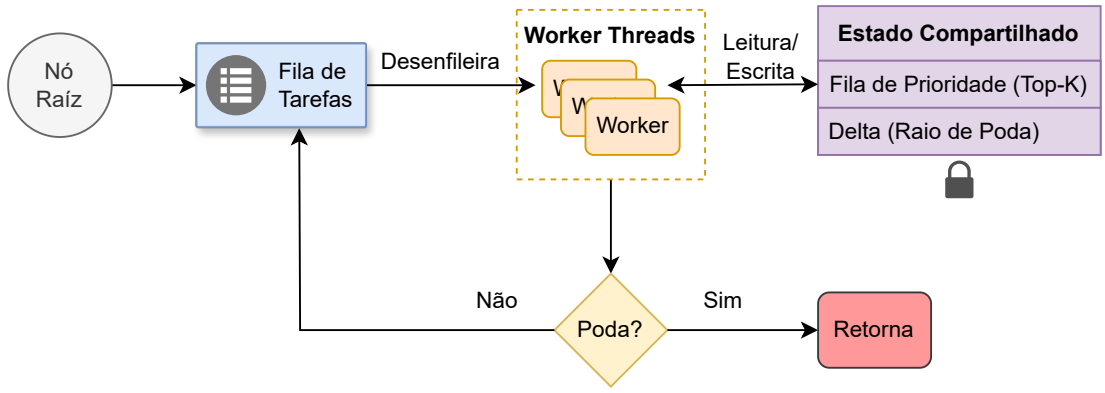


Figura 4.1: Fluxo de execução concorrente: consumo de nós, verificação de poda e atualização sincronizada das estruturas globais.

4.2.1 Mecanismos de Sincronização e Poda

O sistema possui duas estruturas globais protegidas por mecanismos de exclusão mútua (*mutex/locks*) para garantir a consistência dos dados em um ambiente concorrente. A primeira estrutura fundamental é a Fila Prioritária Compartilhada ($\mathcal{PQ}_{shared}$), responsável por armazenar os k vizinhos mais próximos encontrados até o momento por qualquer *thread*. O acesso de escrita a esta fila é bloqueante para evitar condições de corrida. A segunda estrutura é o Limite Global de Corte (δ_{shared}), uma variável atômica ou protegida que armazena a pior distância presente na fila prioritária. Ela atua como um limiar dinâmico e qualquer *thread* pode ler esse valor instantaneamente para decidir sobre a poda de um ramo sem necessidade de bloqueio constante.

O Algoritmo 3 detalha a lógica de execução. A poda utiliza a desigualdade triangular. Um ramo com distância de aresta d_{edge} só recebe exploração se $|d_{node\_query} - d_{edge}| \leq \delta_{shared}$. A atualização contínua de δ_{shared} permite que a descoberta de um bom vizinho por uma *thread* acelere a poda em todas as outras *threads* imediatamente.

Algorithm 3 Algoritmo de Busca Paralela na BKTree

Require: Raiz $\mathcal{T}_{\text{root}}$, Consulta q, k, Função $d(\cdot, \cdot)$.
Ensure: Classe prevista y_{pred}.

$\mathcal{PQ}_{shared} \leftarrow$ PriorityQueue(k)
$\delta_{shared} \leftarrow \infty$
TaskQueue.add($\mathcal{T}_{\text{root}}$)
while existem tarefas ou *workers* ativos **do**
    $node \leftarrow$ TaskQueue.pop()
    ProcessNode($node$) em *Thread* disponível
procedure ProcessNode($node$)
    $d_{curr} \leftarrow d(node.\text{string}, q)$
    Lock($\mathcal{PQ}_{shared}$)
    if d_{curr} qualifica para top-k **then**
        $\mathcal{PQ}_{shared}$.update($d_{curr}, node.class$)
        $\delta_{shared} \leftarrow \mathcal{PQ}_{shared}$.max_dist()
    Unlock($\mathcal{PQ}_{shared}$)
    for cada $(d_{edge}, child)$ em $node$.children **do**
        if $|d_{curr} - d_{edge}| \leq \delta_{shared}$ **then** ▷ Critério de Poda
            TaskQueue.add($child$)
return Moda($\mathcal{PQ}_{shared}$)

Capítulo 5

Experimentos

Neste capítulo, o texto avalia empiricamente a eficácia e a eficiência da abordagem proposta. O objetivo central dos experimentos consiste em validar a hipótese de que a integração da estrutura BKTree com a busca paralelizada reduz drasticamente o custo computacional da classificação baseada em compressão sem degradar a qualidade das predições. Para isso, o estudo conduziu uma bateria de testes comparativos entre o método proposto, a implementação ingênua de força bruta e abordagens baseadas em distâncias de edição clássicas utilizando um *corpus* de detecção de *fake news* e uma variedade de algoritmos de compressão.

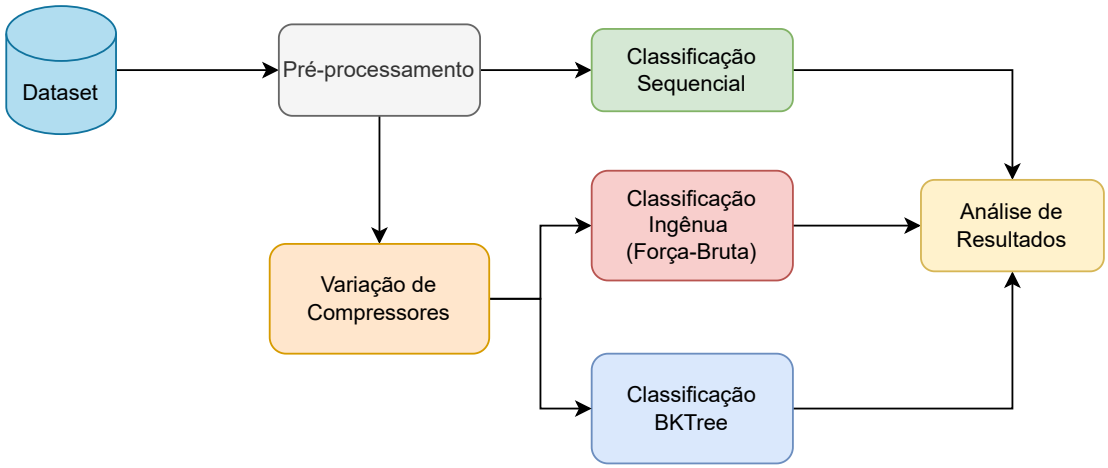


Figura 5.1: Diagrama do fluxo experimental: do pré-processamento às diferentes estratégias de classificação e análise.

A estrutura dos experimentos ilustrada na Figura 5.1 inicia a execução com a ingestão do *dataset* e segue para uma etapa de pré-processamento padronizada. A partir deste ponto, o fluxo bifurca em duas vertentes principais. A primeira encaminha os dados para a classificação sequencial baseada em métricas de edição. A segunda vertente submete

os dados a uma variação de compressores cujos resultados alimentam duas estratégias distintas: a classificação ingênua (força-bruta) e a classificação otimizada via BKTree. Por fim, as saídas das três estratégias (sequencial, ingênua e BKTree) convergem para a etapa de análise de resultados na qual ocorre a consolidação das métricas de desempenho e eficiência.

5.1 Configuração Experimental

Esta seção descreve de forma detalhada o protocolo experimental estabelecido para a validação desta pesquisa. O foco central consiste em avaliar a eficácia e a eficiência do método proposto através do confronto direto com as linhas de base comparativas selecionadas na literatura. Esse nível de detalhamento objetiva não apenas fundamentar a análise dos dados subsequente mas também garantir a total reprodutibilidade dos experimentos e assegurar uma comparação justa entre todas as abordagens analisadas.

5.1.1 Conjunto de Dados e Pré-processamento

Para os experimentos o projeto utilizou o *corpus Low-Resource Fake News Detection Corpora in Filipino* [66] que contém 3.206 notícias rotuladas por especialistas com distribuição balanceada entre classes verdadeira e falsa. O material é majoritariamente em filipino e apresenta inserções ocasionais de termos em inglês presentes no uso coloquial. O protocolo aplicou apenas uma limpeza textual mínima para manter condições experimentais consistentes e reduzir vieses introduzidos por etapas complexas de normalização. Essa limpeza envolveu especificamente a remoção de espaços extremos e normalização básica de *tokens* por meio da Interface de Programação de Aplicações (do inglês, *Application Programming Interface*) (API) do *HuggingFace* [67]. Essa escolha visa preservar as características originais das sequências textuais relevantes para métodos baseados em compressão.

A divisão dos dados adotou uma partição fixa de 80% dos exemplos para treino e 20% para teste. O código fixou todas as sementes e parâmetros randômicos de modo que cada execução observasse exatamente as mesmas partições e assegurasse comparações determinísticas entre os diferentes métodos e compressores. Os tempos de execução reportados correspondem à média observada em múltiplas repetições experimentais com o intuito de reduzir a variabilidade introduzida por ruído no sistema.

5.1.2 Métodos e Compressores

A avaliação ampliou o escopo para além do *GZip* utilizado originalmente em [11] e incluiu um conjunto representativo de algoritmos de compressão sem perda agrupados conforme

sua natureza. A seleção incluiu compressores estatísticos representados pelo *Brotli* [68] e *Bz2* [69] bem como métodos baseados em dicionário, categoria que engloba *GZip* [70], *ZLib* [71], *Zstd* [72], *LZAV* [73] e *LZF* [74]. Adicionalmente, o estudo avaliou compressores leves com foco em velocidade tais como *QuickLZ* [75], *Shoco* [76], *Snappy* [77], *FSST* [78] e *Smaz* [79]. A motivação para a escolha desses algoritmos reside no compromisso entre razão de compressão e velocidade uma vez que compressores distintos tendem a inverter esse compromisso e impactar diretamente a NCD utilizada na classificação.

Três categorias distintas de estratégias de classificação foram implementadas e confrontadas. A primeira estabelecida como linha de base ingênua consiste na implementação por força bruta de KNN empregando NCD como métrica conforme a formulação de [11]. A segunda categoria refere-se à abordagem proposta (BKTree) que aplica o KNN sobre representações compactadas indexadas em uma estrutura adaptada para compressão com busca paralela conforme descrito no Capítulo 4. A terceira categoria abrange as versões baseadas em sequência que utilizam KNN de força bruta sem compressão e empregam métricas de distância entre *strings* clássicas como *Levenshtein* [37], *Damerau-Levenshtein* [39], *Jaro-Winkler* [41, 42] e *Simon-White* [44] como linhas de base complementares. Todas as variações que utilizam KNN fixaram $k = 5$ (cinco vizinhos) para a votação majoritária.

5.1.3 Ambiente e Reprodutibilidade

A implementação optou por uma arquitetura híbrida que equilibra desempenho e facilidade de orquestração. O núcleo crítico do sistema engloba os compressores e as estruturas de dados e teve seu desenvolvimento em linguagem C/C++. A compilação utilizou a Coleção de Compiladores GNU (do inglês *GNU Compiler Collection*) (GCC) 13.2 para garantir máxima eficiência enquanto a biblioteca *OpenMP* viabilizou a execução paralela em memória compartilhada. A integração eficiente entre esse núcleo de baixo nível e a camada de orquestração em *Python* 3.10 ocorreu por meio da ferramenta *Cython*. Essa abordagem permitiu a interoperabilidade direta entre as linguagens e eliminou gargalos de comunicação. A manipulação numérica e as rotinas de avaliação estatística em *Python* empregaram a biblioteca *SciPy*. Todas as execuções ocorreram em uma estação de trabalho equipada com um processador AMD Ryzen Threadripper 1950X de 16 núcleos físicos onde o protocolo buscou minimizar cargas concorrentes para assegurar a estabilidade das medições temporais.

A transparência científica e a possibilidade de verificação independente constituem requisitos fundamentais nesta pesquisa. O projeto optou pela abertura total dos artefatos produzidos com o objetivo de facilitar a reprodução exata dos resultados obtidos e encorajar a extensão do estudo pela comunidade. O conjunto completo engloba o código-fonte da

aplicação e os *scripts* de automação experimental e encontra-se disponível publicamente em repositório de controle de versão[1].

5.2 Resultados Experimentais

Esta seção apresenta e discute os resultados obtidos com a aplicação da metodologia proposta e estabelece um confronto direto entre o desempenho preditivo e os ganhos de eficiência computacional. A análise detalha as métricas de *F1-score*, acurácia, precisão e *recall* para fornecer uma visão abrangente. Os experimentos abrangeram três categorias distintas de modelos para garantir uma base comparativa sólida. As categorias avaliadas incluem a linha de base ingênua (força bruta com NCD), a abordagem proposta otimizada com BKTree e busca paralelizada e uma versão de controle baseada em métricas clássicas de similaridade de sequências.

O foco central da análise reside em demonstrar a viabilidade técnica da integração da BKTree com a NCD. O objetivo primário consiste em comprovar que essa nova arquitetura é capaz de manter a qualidade de classificação do método original e preservar sua robustez teórica. Simultaneamente, busca-se evidenciar que a mudança estrutural introduz ganhos significativos de *speedup* em relação à abordagem ingênua. Assim, os dados apresentados a seguir visam validar a hipótese de que a eficiência de execução pode aumentar drasticamente sem o sacrifício da acurácia preditiva.

5.2.1 Desempenho de Predição

A Tabela 5.1 permite observar que o método baseado em BKTree apresenta desempenho semelhante à implementação ingênua, produz resultados praticamente idênticos e demonstra equivalência estatística na maioria dos casos. O método baseado em BKTree obteve pontuações iguais ou ligeiramente superiores às da implementação ingênua para a maioria dos compressores. Isso demonstra que a estratégia de busca otimizada proporciona ganhos de eficiência sem sacrificar a acurácia de classificação. Conforme indicado na tabela 5.1, os resultados do método proposto são referidos como "BKTree" (símbolo $\mathfrak{B}$) e o método de [11] é denominado "ingênuo" (símbolo $\mathfrak{N}$). Além disso, as colunas $\mathcal{T}_{opt}$ e $\mathcal{S}_{max}$ especificam os hiperparâmetros de paralelização utilizados para obter os valores de *F1-score*, acurácia, precisão e *recall* por se tratar de uma estrutura implementada de forma paralela. Mais especificamente, $\mathcal{T}_{opt}$ indica o número de *threads* que proporcionou o melhor *speedup* relativo enquanto $\mathcal{S}_{max}$ revela o *speedup* relativo alcançado com esse número de *threads*.

[1] `https://gitlab.com/lisa-unb/comtext/`

Tabela 5.1: Métricas de predição obtidas usando o método proposto baseado em BKTree comparado com a abordagem ingênua de KNN quando o speedup máximo foi alcançado. $\mathcal{S}_{max}$ representa o speedup relativo máximo alcançado, e $\mathcal{T}_{opt}$ é o número de *threads* que produziu esse speedup ótimo. Os símbolos $\mathfrak{N}$ e $\mathfrak{B}$ representam, respectivamente, o método ingênuo (linha de base) e o BKTree (proposto).

C	F1-Score		Acurácia		Precisão		Recall		$\mathcal{T}_{opt}$	$\mathcal{S}_{max}$
	$\mathfrak{N}$	$\mathfrak{B}$	$\mathfrak{N}$	$\mathfrak{B}$	$\mathfrak{N}$	$\mathfrak{B}$	$\mathfrak{N}$	$\mathfrak{B}$		
Brotli	0.90	**0.92**	0.90	**0.92**	0.91	**0.92**	0.90	**0.92**	12	19.74
FSST	**0.89**	0.88	**0.89**	0.88	**0.89**	0.88	**0.89**	0.88	24	24.73
LZ4	**0.92**	**0.92**	**0.92**	**0.92**	0.92	**0.93**	**0.92**	**0.92**	8	6.8
LZAV	0.88	**0.89**	0.88	**0.89**	0.89	**0.90**	0.88	**0.89**	8	6.36
LZF	0.90	**0.91**	0.90	**0.91**	0.90	**0.91**	0.90	**0.91**	16	8.03
Quicklz	0.81	**0.92**	0.81	**0.92**	0.83	**0.92**	0.81	**0.92**	12	9.93
Shoco	**0.56**	0.33	**0.61**	0.50	**0.68**	0.25	**0.61**	0.5	12	1.31
Smaz	0.43	**0.54**	0.52	**0.55**	**0.57**	0.56	0.52	**0.55**	8	1.89
Snappy	0.49	**0.91**	0.58	**0.91**	0.75	**0.91**	0.58	**0.91**	16	11.38
ZLib	**0.94**	**0.94**	**0.94**	**0.94**	**0.94**	**0.94**	**0.94**	**0.94**	12	11.83
ZStd	**0.95**	0.94	**0.95**	0.94	**0.95**	0.94	**0.95**	0.94	24	6.1

5.2.2 Relação entre Desempenho e Eficiência

A relação entre desempenho e eficiência aparece ilustrada na Figura 5.2 que apresenta um diagrama de dispersão comparando a *F1-score* máxima com o tempo médio de execução de cada modelo. Na visualização, as implementações ingênuas aparecem em vermelho, as abordagens baseadas em BKTree em azul e os métodos sem compressão (baseados em sequência) em verde. Um modelo ideal situa-se no quadrante superior esquerdo (maior *F1* e menor tempo). Nesse quadrante os modelos baseados em BKTree predominam e evidenciam melhor compromisso entre eficiência e acurácia. Em contrapartida, as implementações ingênuas tendem a apresentar maiores tempos de execução e posicionam-se à direita do gráfico. Os métodos sem compressão exibem características temporais semelhantes às das implementações ingênuas.

Eficiência da BKTree Sequencial vs. Método Naive

Para validar o impacto da estrutura de dados na redução do esforço computacional, realizou-se uma comparação direta entre a busca exaustiva (*Naive*) e a busca fundamentada na BKTree, ambas executadas em uma única *thread*. O objetivo desta etapa é isolar o ganho proporcionado exclusivamente pela aplicação da desigualdade triangular na poda de ramos da árvore. Os resultados consolidados na Tabela 5.2 demonstram que a utilização da

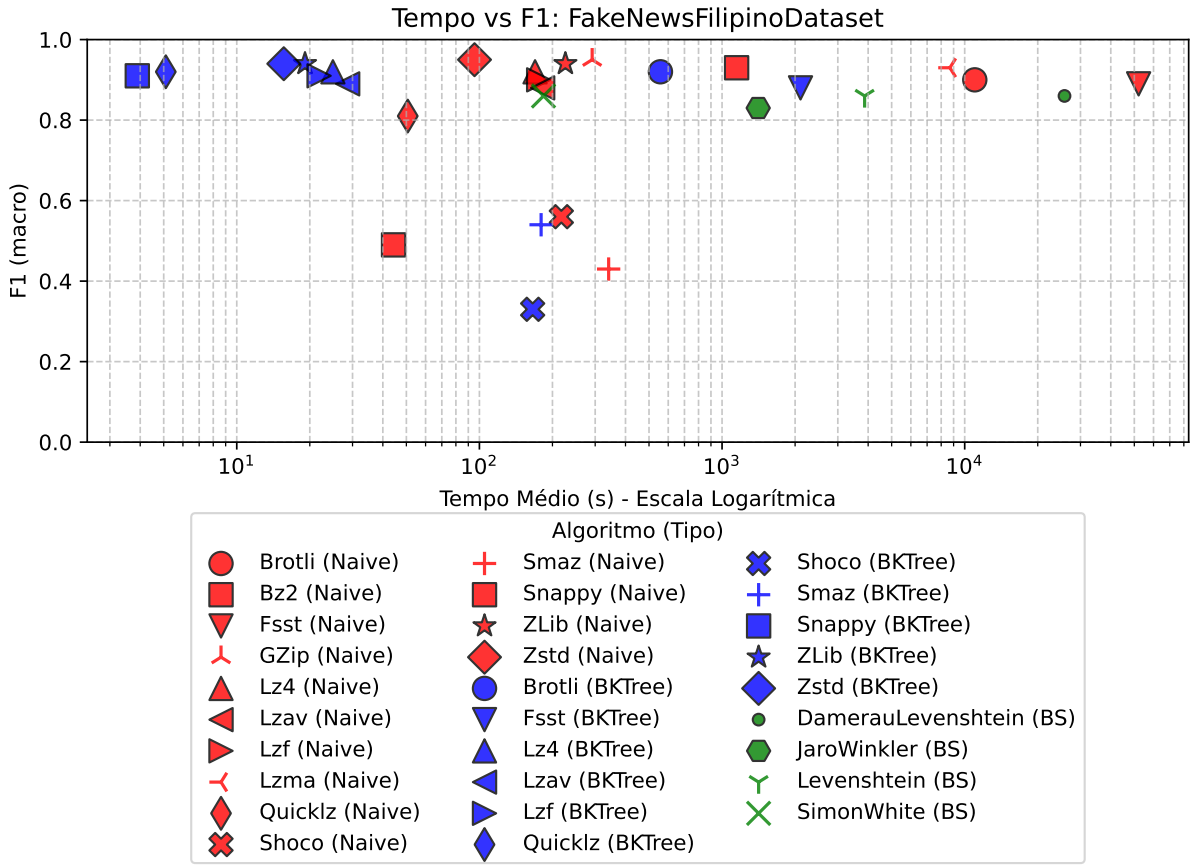


Figura 5.2: Diagrama de dispersão da *F1-score* máxima versus tempo médio de execução (escala *log*) para todos os compressores avaliados. Os tipos de modelo são distinguidos por cor: ingênuo (vermelho), BKTree (azul), baseado em sequência (verde).

BKTree resultou em reduções significativas no tempo de treinamento e inferência total para a maioria dos compressores.

A análise dos dados revela que compressores como o FSST e o Brotli obtiveram os maiores índices de aceleração, com *speedups* de 14,70× e 4,78×, respectivamente. Esse comportamento sugere que a distribuição das distâncias NCD gerada por esses algoritmos permitiu uma poda eficaz da árvore. Por outro lado, observou-se que compressores otimizados para strings muito curtas, como o Shoco, apresentaram um *speedup* ligeiramente inferior à unidade (0,92×). Esse fenômeno ocorre quando o custo fixo de percorrer a estrutura da árvore e gerenciar a fila de busca supera a economia gerada pela redução no número de cálculos de distância, indicando que a eficiência da BKTree é sensível ao tempo de execução intrínseco de cada compressor. Esses resultados sequenciais definem o ponto de partida para a avaliação do processamento paralelo via OpenMP, discutida a seguir.

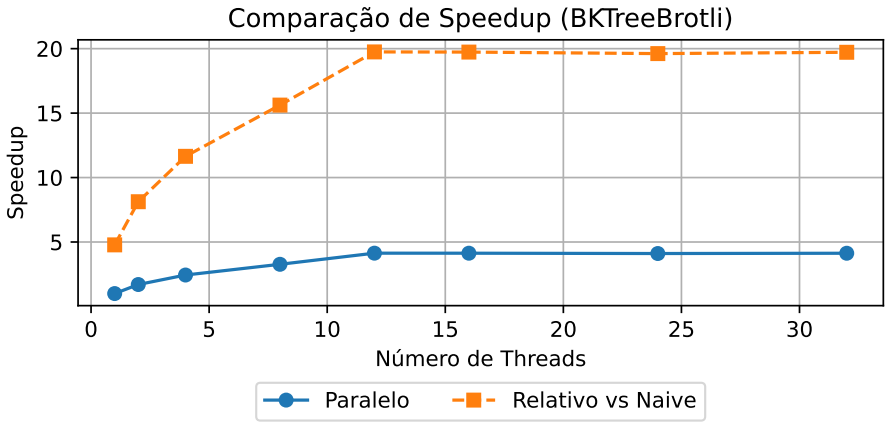
Tabela 5.2: Comparação entre método Naive e BKTree sequencial (1 thread)

Compressor	**Tempo Total Naive (s)**	**Tempo Total BKTree (s)**	**Speedup**
Brotli	11.003,00	2.303,43	4,78×
Fsst	52.098,00	3.544,00	14,70×
Lz4	169,56	71,31	2,38×
Lzav	182,23	74,43	2,45×
Lzf	175,50	68,49	2,56×
Quicklz	50,67	14,82	3,42×
Shoco	217,56	236,48	0,92×
Smaz	340,87	287,58	1,19×
Snappy	44,29	10,14	4,37×
ZLib	226,15	62,85	3,60×
Zstd	95,45	77,12	1,24×

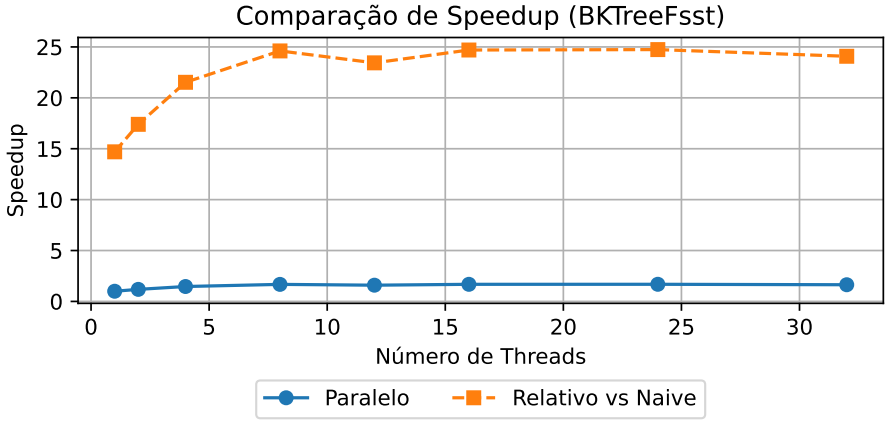
5.2.3 Speedup e Escalabilidade

Os hiperparâmetros de paralelização $\mathcal{T}_{opt}$ e $\mathcal{S}_{max}$ apresentados na Tabela 5.1 correspondem aos pontos máximos nas curvas laranja exibidas na Figura 5.3. Essa figura apresenta a curva número de *threads* versus *speedup* para cada compressor: as curvas azuis representam o *speedup* paralelo (comparação entre BKTree paralelo e BKTree serial) e as laranjas representam o *speedup* relativo (comparação entre BKTree paralelo e a implementação ingênua). Observa-se que a escalabilidade obtida pela paralelização tende a saturar rapidamente, isto é, o *speedup* deixa de aumentar apesar do incremento no número de *threads*, porém a curva laranja permanece assintoticamente acima da curva azul na maioria dos casos. Esse comportamento indica que a vantagem principal do método proposto advém da estrutura em BKTree que reduz a complexidade computacional da busca e não apenas da paralelização. Como exemplo, o compressor *Zstd* atingiu *speedup* relativo de 6× em relação à linha de base ingênua e cerca de 5× de *speedup* paralelo em relação à execução com *thread* única, o que evidencia que o uso de múltiplas *threads* reduz substancialmente o tempo de execução. Outros compressores também se beneficiaram do processamento paralelo, mesmo quando sua escalabilidade é menor, e apresentaram ganhos relevantes de taxa de processamento.

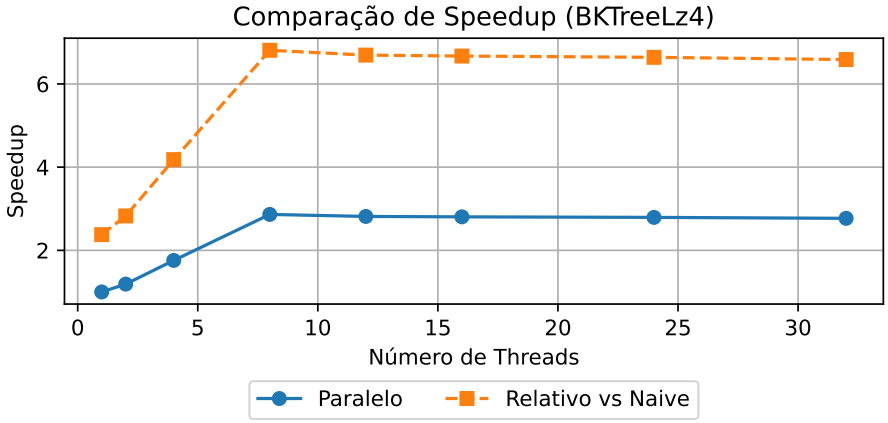
A eficiência de paralelização, calculada conforme a Equação **??**, permite avaliar o grau de aproveitamento dos recursos computacionais à medida que a carga de trabalho é distribuída entre múltiplos núcleos de processamento. Os resultados apresentados na Tabela 5.3 revelam uma tendência comum de declínio na eficiência com o aumento do número de *threads*, fenômeno atribuído à sobrecarga (*overhead*) de sincronização e à contenção de recursos na gestão da fila de tarefas da BKTree.



(a) Brotli

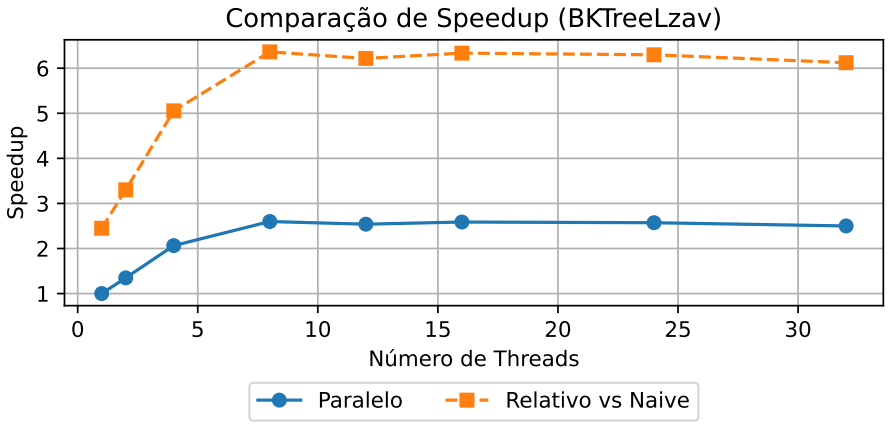


(b) FSST

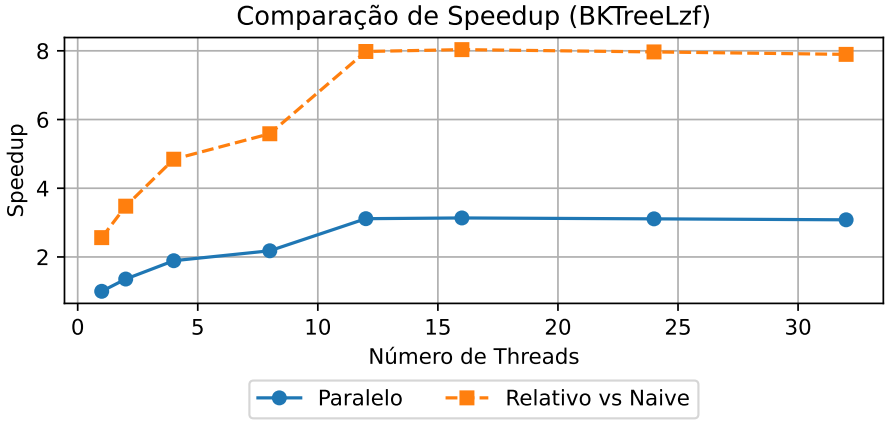


(c) Lz4

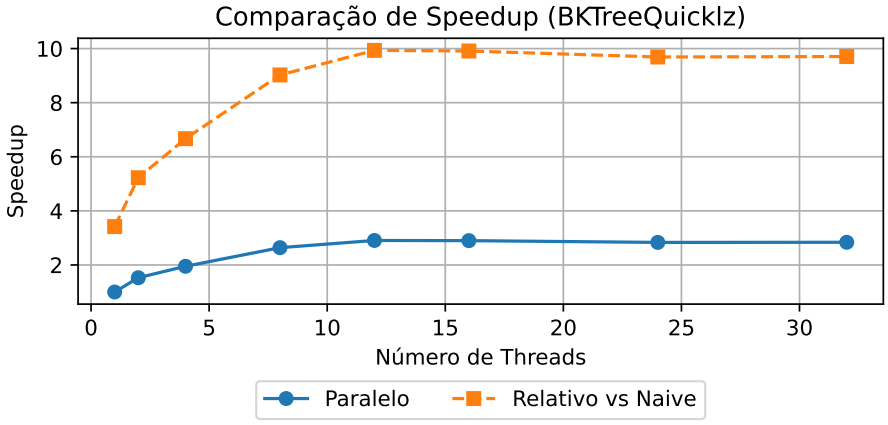
Figura 5.3: Curvas de *speedup* relativo e paralelo por compressor.



(d) LZAV

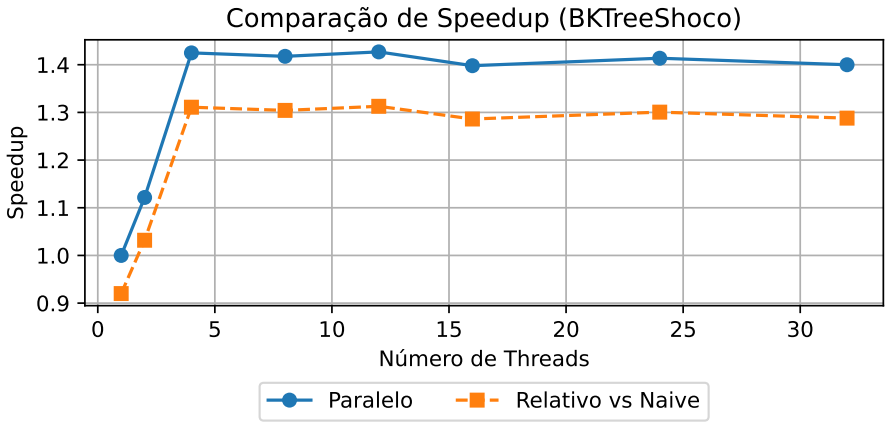


(e) LZF

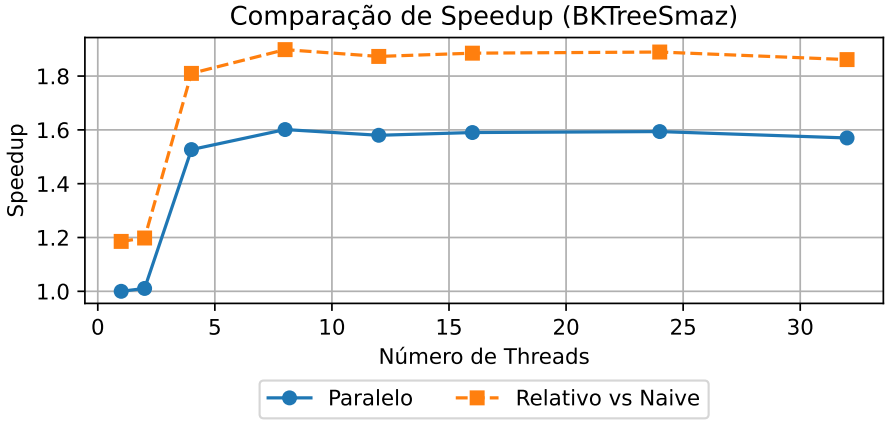


(f) QuickLZ

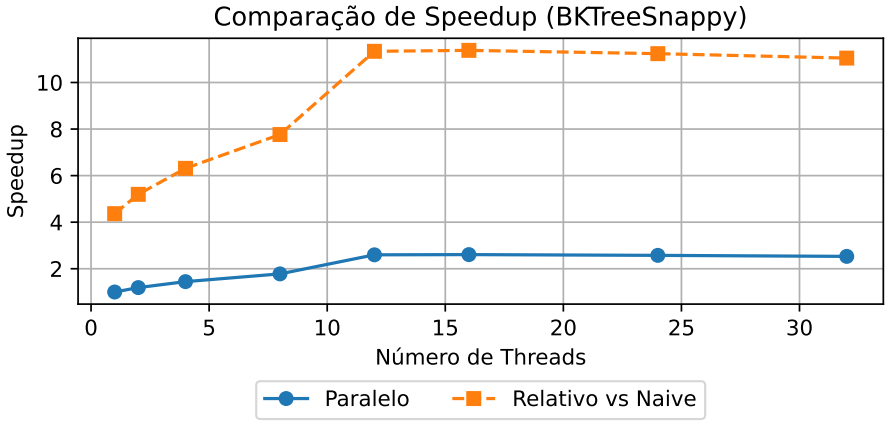
Figura 5.3: Curvas de *speedup* relativo e paralelo por compressor (continuação).



(g) Shoco

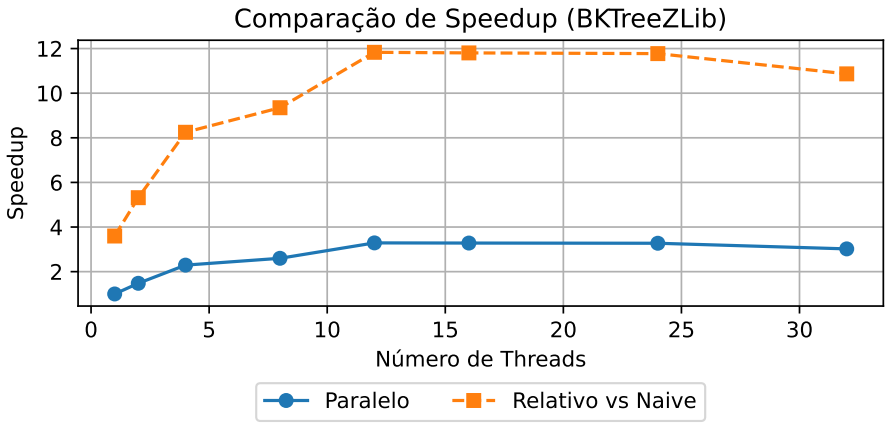


(h) Smaz

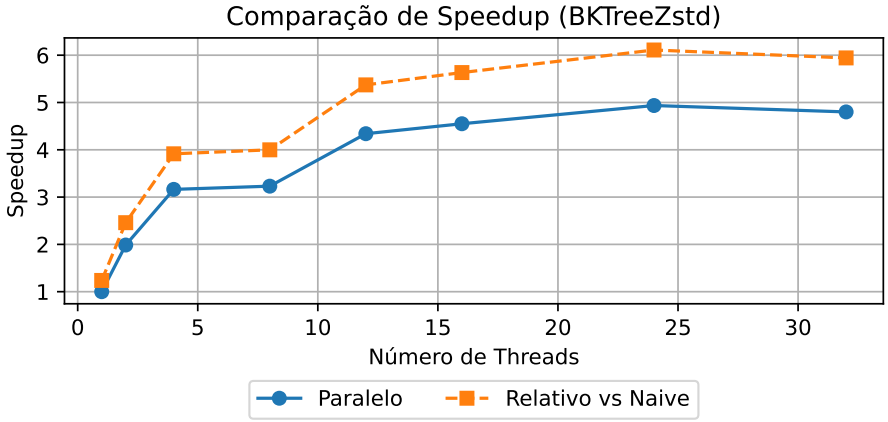


(i) Snappy

Figura 5.3: Curvas de *speedup* relativo e paralelo por compressor (continuação).



(j) ZLib



(k) Zstd

Figura 5.3: Curvas de *speedup* relativo e paralelo por compressor (continuação).

Tabela 5.3: Eficiência de paralelização por compressor e número de threads (%)

Compressor	**1**	**2**	**4**	**8**	**12**	**16**	**24**	**32**
Brotli	100	85,0	61,0	40,9	34,4	25,8	17,1	12,9
Fsst	100	59,2	36,6	20,9	13,3	10,5	7,0	5,1
Lz4	100	59,5	44,0	35,8	23,5	17,5	11,6	8,7
Lzav	100	67,4	51,6	32,5	21,2	16,2	10,7	7,8
Lzf	100	67,9	47,3	27,2	26,0	19,6	13,0	9,6
Quicklz	100	76,4	48,8	33,0	24,2	18,1	11,8	8,9
Shoco	100	56,1	35,6	17,7	11,9	8,7	5,9	4,4
Smaz	100	50,5	38,2	20,0	13,2	9,9	6,6	4,9
Snappy	100	59,5	36,1	22,2	21,6	16,3	10,7	7,9
ZLib	100	73,9	57,3	32,5	27,4	20,5	13,6	9,4
Zstd	100	99,3	79,1	40,4	36,2	28,4	20,6	15,0

5.2.4 Observações por Compressor

Quanto às características individuais, a análise demonstrou que diferentes compressores se destacaram por propriedades específicas e evidenciaram o compromisso entre a precisão da métrica NCD e a eficiência computacional. Os compressores *Snappy* e *QuickLZ* destacaram-se por apresentar eficiência temporal elevada com comprometimento mínimo da acurácia e, portanto, constituem boas escolhas quando o tempo de resposta é crítico. Em contrapartida, *ZLib* e *Zstd* forneceram a acurácia superior (*F1-scores* de até 0,95) em velocidades competitivas e posicionam-se como alternativas mais equilibradas quando a prioridade for a máxima precisão de predição.

Outros algoritmos como *LZ4*, *LZF* e *LZAV* também demonstraram excelente performance e alcançaram *F1-scores* consistentemente próximos ou superiores a 0,90 (variando de 0,88 a 0,92) além de apresentar ganhos de *speedup* relativo notáveis que variaram entre 6,36x e 8,03x em relação à linha de base ingênua. Isso os consolida como alternativas sólidas que conseguem manter um bom balanceamento entre a qualidade preditiva e a eficiência temporal. Por outro lado, o *FSST* e o *Brotli* demonstraram maior sensibilidade à complexidade computacional da compressão e impactaram o tempo de execução. Embora o *FSST* tenha alcançado o maior *speedup* relativo (24,73x), o tempo de execução absoluto de ambos foi mais elevado, o que indica um custo maior por consulta.

No extremo de desempenho, os algoritmos *Smaz* e *Shoco* exibiram as maiores dificuldades em termos de acurácia e eficiência. O baixo desempenho do *Shoco* se explica pelo fato de ser um compressor otimizado especificamente para *strings* curtas em inglês enquanto o conjunto de dados foi testado em um *corpus* de notícias em filipino, o que resultou em uma identificação de padrões insuficiente para textos maiores. O *Smaz* também é voltado para

strings muito curtas e isso limita sua capacidade de capturar os padrões informacionais necessários para a métrica NCD em documentos textuais de maior extensão.

Em síntese, as evidências indicam que a seleção de compressores não deve ser arbitrária mas sim estritamente orientada pelos requisitos da aplicação. É fundamental buscar um equilíbrio criterioso entre a eficiência da taxa de compressão, que afeta a sensibilidade da NCD, e o custo temporal de execução do algoritmo em si. Para cenários que exigem máxima acurácia, deve-se optar por *ZLib* ou *Zstd* ao passo que *Snappy* ou *QuickLZ* tornam-se preferíveis quando a prioridade é a latência mínima. Por fim, vale ressaltar que compressores como *LZ4* e *LZF* oferecem um excelente meio-termo e consolidam-se como opções versáteis para casos de uso gerais.

5.2.5 Sumário dos Resultados

Os resultados experimentais demonstram inequivocamente a eficácia da abordagem proposta para otimizar a classificação de textos baseada na NCD. A estrutura de dados baseada na BKTree mostrou-se crucial pois não apenas preservou mas em alguns cenários de compressores chegou a melhorar marginalmente as métricas de classificação em comparação com a custosa implementação ingênua de força bruta. Este sucesso em manter a qualidade preditiva reside na capacidade da BKTree de restringir eficientemente o espaço de busca a vizinhos próximos sem comprometer significativamente a identificação do vizinho mais próximo com base na NCD.

A principal contribuição prática e o ganho mais substancial residem na eficiência. A organização dos dados em BKTree atua diretamente na redução da complexidade computacional da fase de busca e transforma uma operação de tempo $O(N)$ em uma busca com complexidade esperada muito menor. Esta otimização estrutural combinada com o uso estratégico da paralelização em múltiplos núcleos permitiu a obtenção de *speed-ups* relativos significativos e substanciais em relação à linha de base ingênua. A capacidade de processar consultas em uma fração do tempo do método ingênuo valida o uso da BKTree como um acelerador fundamental para a NCD em grandes coleções de texto e torna o método aplicável em ambientes onde o tempo de resposta é uma limitação crítica.

Para aplicações práticas a seleção do compressor deve ser uma decisão de engenharia baseada no compromisso desejado entre precisão e velocidade. Compressores como *ZLib* e *Zstd* são a escolha ideal quando a prioridade inegociável é maximizar a acurácia devido à sua capacidade superior de capturar estruturas complexas do texto e gerar uma NCD mais informativa. Por outro lado, para cenários de tempo real ou onde o volume de consultas exige a máxima velocidade, compressores como *Snappy* e *QuickLZ*, juntamente com *LZ4* e *LZF*, representam a melhor alternativa pois oferecem ganhos massivos de eficiência temporal com uma degradação de acurácia que se mostrou aceitável na maioria dos casos.

Capítulo 6

Conclusão

Este trabalho apresentou uma reformulação estrutural do método `NPC_Gzip`, originalmente proposto por [11], visando superar suas limitações de escalabilidade. A abordagem desenvolvida substituiu a busca exaustiva por uma estrutura hierárquica de dados baseada em BKTree, além de incorporar estratégias modernas de paralelização. Como resultado, obteve-se uma arquitetura robusta que preserva a simplicidade e a interpretabilidade inerentes ao método original. Simultaneamente, essa nova estrutura proporcionou ganhos substanciais e mensuráveis em termos de desempenho computacional e capacidade de processamento.

6.1 Síntese dos Resultados

Os experimentos realizados validaram a hipótese central da pesquisa, demonstrando que a abordagem proposta alcança acurácia equivalente ou até superior à implementação ingênua. Com *F1-scores* atingindo marcas de até 0,95, confirmou-se que a otimização estrutural via árvores métricas não compromete a qualidade das predições finais. A principal contribuição deste estudo reside, contudo, nos ganhos de eficiência operacional. Foram obtidos *speedups* relativos de até 25× em comparação à linha de base de força bruta, o que evidencia o enorme potencial da combinação entre BKTree e paralelização para acelerar a classificação baseada em compressão.

Quanto à diversidade de algoritmos, a avaliação comparativa de onze compressores distintos revelou padrões importantes sobre o compromisso entre precisão e velocidade. Compressores focados em rapidez, como Snappy e QuickLZ, destacaram-se pelo excelente desempenho em tempo de execução, sendo ideais para aplicações de baixa latência. Por outro lado, algoritmos como ZLib e Zstd alcançaram as maiores taxas de acurácia mantendo uma eficiência temporal competitiva. Esses resultados oferecem subsídios práticos valiosos,

permitindo a escolha informada de compressores conforme as restrições específicas de cada aplicação.

No que se refere à escalabilidade, a análise aprofundada indicou que a vantagem principal do método advém da organização dos dados na estrutura BKTree, que reduz drasticamente a complexidade computacional da busca. Embora a paralelização seja benéfica, observou-se que a escalabilidade tende a saturar com o aumento excessivo do número de *threads*. Ainda assim, os ganhos obtidos pela execução concorrente são substanciais quando comparados à execução serial. Isso torna o método viável e eficiente para o processamento de conjuntos de dados de grande porte.

6.2 Limitações e Considerações

Embora os resultados obtidos sejam promissores, é necessário reconhecer algumas limitações inerentes à abordagem proposta. A construção da BKTree exige uma etapa de pré-processamento computacionalmente intensiva, o que pode representar um custo inicial elevado. Por essa razão, o método se mostra mais adequado para cenários caracterizados por um alto volume de consultas sobre um mesmo conjunto de treinamento estático, onde esse investimento inicial é amortizado ao longo do tempo. Além disso, a eficácia da poda depende estritamente da adequação da métrica de distância ao espaço métrico, sendo imperativo verificar se a desigualdade triangular é satisfeita para garantir a corretude dos resultados.

No que tange à generalização dos achados, os experimentos foram conduzidos sobre um *corpus* específico de detecção de *fake news* no idioma filipino. Consequentemente, a extensão e a validação desses resultados para outros domínios textuais e línguas diferentes ainda precisam ser investigadas em profundidade. Outro ponto de atenção é a influência direta da escolha do compressor tanto na precisão quanto na eficiência do sistema. Isso exige que o usuário possua conhecimento prévio sobre as características do domínio dos dados para realizar uma seleção adequada do algoritmo de compressão.

6.3 Trabalhos Futuros

Como direções para pesquisas futuras, pretende-se expandir as avaliações experimentais para uma variedade maior de domínios textuais e idiomas. O objetivo é investigar a capacidade de generalização do método proposto e identificar padrões de comportamento em contextos linguísticos diversos. Além disso, a exploração de estratégias alternativas de paralelização para a BKTree permanece como um campo aberto. Isso inclui, especifica-

mente, a investigação de técnicas para paralelizar também a fase de construção da árvore, o que representa uma oportunidade significativa de otimização adicional.

Adicionalmente, vislumbra-se a integração de técnicas leves de aprendizado supervisionado ao fluxo de classificação. Abordagens como o ajuste fino de hiperparâmetros baseado em validação cruzada ou a ponderação adaptativa de vizinhos podem aprimorar ainda mais a aplicabilidade prática dos métodos baseados em compressão. Por fim, a investigação de compressores especializados para domínios textuais específicos, bem como a análise de métodos híbridos que combinem diferentes métricas de similaridade, constituem direções promissoras para a continuidade deste trabalho.

6.4 Trabalhos Publicados

Conferências Internacionais

- SILVESTRE, A. S. S.; **DE SOUZA, B. V.**; LISBOA, V. H. F.; BORGES, V. R. P. **A Multi-Label Classification Approach for Categorizing Beginner Programming Problems from Online Judges**. In: *2024 IEEE Frontiers in Education Conference (FIE)*. IEEE, 2024. p. 1-8. DOI: 10.1109/FIE61694.2024.10893153.

Conferências Nacionais

- **SOUZA, B.**; FREITAS, P. **Efficient Compression-Based Low-resource Text Classification**. In: *Anais do XXII Encontro Nacional de Inteligência Artificial e Computacional (ENIAC)*. Porto Alegre: SBC, 2025. p. 1797–1808. DOI: 10.5753/eniac.2025.13966.
- **SOUZA, B.**; SILVESTRE, A.; LISBOA, V.; BORGES, V. **Pre-trained Language Models for Multi-Label Text Classification of Competitive Programming Problems**. In: *Anais do XXI Encontro Nacional de Inteligência Artificial e Computacional (ENIAC)*. Porto Alegre: SBC, 2024. p. 73–84. DOI: 10.5753/eniac.2024.245222.

Referências

[1] Sebastiani, Fabrizio: *Machine learning in automated text categorization.* ACM Computing Surveys, 34(1):1–47, 2002. 1

[2] Shu, Kai, Amy Silva, Suhang Wang, Jiliang Tang e Huan Liu: *Fake news detection on social media: A data mining perspective.* ACM SIGKDD Explorations Newsletter, 19(1):22–36, 2017. 1

[3] Sahami, Mehran, Susan Dumais, David Heckerman e Eric Horvitz: *A bayesian approach to filtering junk e-mail.* Em *Learning for Text Categorization: Papers from the 1998 Workshop*, páginas 98–105, Madison, Wisconsin, 1998. AAAI Press. 1

[4] Pang, Bo e Lillian Lee: *Opinion mining and sentiment analysis.* Foundations and Trends in Information Retrieval, 2(1–2):1–135, 2008. 1

[5] Vargas, Bruno, Ana Silvestre, Victor Lisboa e Vinicius Borges: *Pre-trained language models for multi-label text classification of competitive programming problems.* Em *Anais do XXI Encontro Nacional de Inteligência Artificial e Computacional*, páginas 73–84, Porto Alegre, RS, Brasil, 2024. SBC. `https://sol.sbc.org.br/index.php/eniac/article/view/33783`. 1

[6] Silvestre, Ana Sofia S., Bruno Vargas De Souza, Victor Hugo F. Lisboa e Vinicius R. P. Borges: *A multi-label classification approach for categorizing beginner programming problems from online judges.* Em *2024 IEEE Frontiers in Education Conference (FIE)*, páginas 1–8, 2024. 1

[7] Devlin, Jacob, Ming Wei Chang, Kenton Lee e Kristina Toutanova: *Bert: Pre-training of deep bidirectional transformers for language understanding.* Em *Proceedings of NAACL-HLT*, páginas 4171–4186, 2019. 1, 6

[8] LeCun, Yann, Yoshua Bengio e Geoffrey Hinton: *Deep learning.* Nature, 521(7553):436–444, 2015. 1

[9] Strubell, Emma, Ananya Ganesh e Andrew McCallum: *Energy and policy considerations for deep learning in NLP.* Em *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, páginas 3645–3650, Florence, Italy, 2019. Association for Computational Linguistics. 1

[10] Cilibrasi, Rudi e Paul M. B. Vitányi: *Clustering by compression.* IEEE Transactions on Information Theory, 51(4):1523–1545, 2005. 1, 6, 9, 10, 12, 22

[11] Jiang, Zhiying, Matthew Y. R. Yang, Mikhail Tsirlin, Raphael Tang, Yiqin Dai e Jimmy Lin: *"low-resource" text classification: A parameter-free classification method with compressors*. Em Rogers, Anna, Jordan L. Boyd-Graber e Naoaki Okazaki (editores): *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, páginas 6810–6828. Association for Computational Linguistics, 2023. https://doi.org/10.18653/v1/2023.findings-acl.426. 2, 7, 21, 27, 28, 29, 39

[12] Rudin, Cynthia: *Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead*. Nature Machine Intelligence, 1(5):206–215, 2019. 2

[13] Burkhard, Walter A. e Robert M. Keller: *Some approaches to best-match file searching*. Communications of the ACM, 16(4):230–236, 1973. 3, 13, 22

[14] Salomon, David: *Data Compression: The Complete Reference*. Springer Science & Business Media, 4ª edição, 2007. 3, 11, 12

[15] Minaee, Shervin, Nal Kalchbrenner, Erik Cambria, Narjes Nikzad, Meysam Chenaghlu e Jianfeng Gao: *Deep learning-based text classification: A comprehensive review*. ACM Computing Surveys (CSUR), 54(3):1–40, 2021. 5

[16] Salton, Gerard, Anita Wong e Chung Shu Yang: *A vector space model for automatic indexing*. Communications of the ACM, 18(11):613–620, 1975. 5

[17] Salton, G. e C. Buckley: *Term-weighting approaches in automatic text retrieval*. Information Processing & Management, 24(5):513–523, 1988. 5

[18] Cover, T. M. e P. E. Hart: *Nearest neighbor pattern classification*. IEEE Transactions on Information Theory, 13(1):21–27, 1967. 6, 7

[19] McCallum, A. e K. Nigam: *A comparison of event models for naive bayes text classification*. 1998. Também disponível como technical report; referência clássica sobre Naive Bayes em texto. 6

[20] Shannon, Claude E.: *A mathematical theory of communication*. The Bell System Technical Journal, 27(3):379–423, 1948. 6

[21] Elman, Jeffrey L.: *Finding structure in time*. Cognitive Science, 14(2):179–211, 1990. 6

[22] Hochreiter, S. e J. Schmidhuber: *Long short-term memory*. Neural Computation, 9(8):1735–1780, 1997. 6

[23] Cho, Kyunghyun, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk e Yoshua Bengio: *Learning phrase representations using rnn encoder-decoder for statistical machine translation*. arXiv preprint arXiv:1406.1078, 2014. 6

[24] Kim, Yoon: *Convolutional neural networks for sentence classification.* Em *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, páginas 1746–1751, 2014. 6

[25] Mikolov, Tomas, Kai Chen, Greg Corrado e Jeffrey Dean: *Efficient estimation of word representations in vector space.* arXiv preprint arXiv:1301.3781, 2013. 6

[26] Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser e Illia Polosukhin: *Attention is all you need.* Em *Advances in Neural Information Processing Systems (NeurIPS) / Proceedings*, 2017. 6

[27] Levenshtein, V. I.: *Binary codes capable of correcting deletions, insertions, and reversals.* Soviet Physics Doklady, 10:707–710, 1966. 6, 10

[28] Li, Ming e Paul M. B. Vitányi: *An Introduction to Kolmogorov Complexity and Its Applications.* Springer, 2nd edição, 1997. 6, 9, 10, 12

[29] Li, M., X. Chen, X. Li, B. Ma e P. M. B. Vitányi: *The similarity metric.* arXiv preprint (cs/0111054), 2001. 6, 10

[30] Nala, Vinicius: *Modelos ML: Paramétricos x Não-paramétricos.* `https://medium.com/@viniciusnala/modelos-ml-paramÃľtricos-x-nÃčo-paramÃľtricos-0cc68e1a82aa`, fev 2024. Artigo publicado no Medium. Acessado em: 21 de novembro de 2025. 7

[31] Izbicki, Rafael e Tiago Mendonça dos Santos: *Aprendizado de máquina: uma abordagem estatística.* Publicação Independente, 1ª edição, 2020, ISBN 978-65-00-02410-4. `https://rafaelizbicki.com/ame/`, Disponível para download gratuito. Acessado em: 21 de novembro de 2025. 7

[32] Dudani, S. A.: *The distance-weighted k-nearest-neighbor rule.* IEEE Transactions on Systems, Man, and Cybernetics, 6(4):325–327, 1976. 7

[33] Fix, Evelyn e Joseph L. Hodges: *Discriminatory analysis: Nonparametric discrimination, consistency properties.* Technical Report 21-49-004, USAF School of Aviation Medicine, 1951. 7

[34] Aha, David W, Dennis Kibler e Marc K Albert: *Instance-based learning algorithms.* Machine learning, 6(1):37–66, 1991. 7

[35] Duda, Richard O., Peter E. Hart e David G. Stork: *Pattern Classification.* Wiley-Interscience, 2nd edição, 2000. 7

[36] Cohen, William W., Pradeep Ravikumar e Stephen E. Fienberg: *A comparison of string metrics for matching names and records.* Em *KDD Workshop on Data Cleaning and Object Consolidation*, volume 3, páginas 73–78, Washington, DC, 2003. 8

[37] Levenshtein, Vladimir I.: *Binary codes capable of correcting deletions, insertions, and reversals.* Soviet Physics Doklady, 10:707–710, 1966. 9, 13, 28

[38] Wagner, Robert A. e Michael J. Fischer: *The string-to-string correction problem.* J. ACM, 21(1):168–173, janeiro 1974, ISSN 0004-5411. `https://doi.org/10.1145/321796.321811`. 9

[39] Damerau, Fred J.: *A technique for computer detection and correction of spelling errors.* Communications of the ACM, 7(3):171–176, 1964. 9, 10, 28

[40] Christen, Peter: *A comparison of personal name matching: techniques and practical issues.* Em *Sixth IEEE International Conference on Data Mining - Workshops (ICDMW'06)*, páginas 290–294. IEEE, 2006. 9

[41] Jaro, Matthew A.: *Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida.* Em *JSM Proceedings, Social Statistics Section*, 1989. 9, 11, 28

[42] Winkler, William E.: *String comparator metrics and enhanced decision rules in the fellegi–sunter model of record linkage.* Relatório Técnico, U.S. Bureau of Census, 1990. 9, 11, 28

[43] Elmagarmid, Ahmed K, Panagiotis G Ipeirotis e Vassilios S Verykios: *Duplicate record detection: A survey.* IEEE Transactions on Knowledge and Data Engineering, 19(1):1–16, 2007. 9

[44] White, Simon: *How to strike a match: A simple algorithm for string similarity.* `http://www.catalysoft.com/articles/StrikeAMatch.html`, 2000. Accessed 2025-05-20. 9, 11, 28

[45] Teahan, William J e David J Harper: *Using compression-based language models for text categorization.* Language modeling for information retrieval, páginas 141–165, 2003. 10, 12

[46] Frank, Eibe, Chang Chui e Ian H Witten: *Text categorization using compression models.* Em *Proceedings of the Conference on Data Compression*, página 555, 2000. 10

[47] Cebrian, Manuel, Manuel Alfonseca e Alfonso Ortega: *The normalized compression distance is resistant to noise.* IEEE Transactions on Information Theory, 53(5):1895–1900, 2007. 11, 13

[48] Sayood, Khalid: *Introduction to data compression.* Morgan Kaufmann, 2017. 11

[49] Ziv, Jacob e Abraham Lempel: *A universal algorithm for sequential data compression.* IEEE Transactions on Information Theory, 23(3):337–343, 1977. 11

[50] Welch, Terry A.: *A technique for high-performance data compression.* Computer, 17(6):8–19, 1984. 11

[51] Cleary, John G. e Ian H. Witten: *Data compression using adaptive coding and partial string matching.* IEEE Transactions on Communications, 32(4):396–402, 1984. 12

[52] Burrows, Michael e David J. Wheeler: *A block-sorting lossless data compression algorithm*. Relatório Técnico, Digital Equipment Corporation, 1994. 12

[53] Deutsch, P.: *Deflate compressed data format specification version 1.3*. Relatório Técnico, IETF RFC 1951, 1996. 12

[54] Zezula, Pavel, Giuseppe Amato, Vlastislav Dohnal e Michal Batko: *Similarity Search: The Metric Space Approach*. Springer Science & Business Media, New York, 2006. 13

[55] Chávez, Edgar, Gonzalo Navarro, Ricardo Baeza-Yates e José Luis Marroquín: *Searching in metric spaces*. ACM Computing Surveys (CSUR), 33(3):273–321, 2001. 16

[56] Zobel, Justin e Alistair Moffat: *Inverted files for text search engines*. ACM Computing Surveys, 38(2):6–56, 2004. 16

[57] Pacheco, Peter S.: *An Introduction to Parallel Programming*. Morgan Kaufmann, 2011. 16

[58] Flynn, Michael J.: *Some computer organizations and their effectiveness*. IEEE Transactions on Computers, C-21(9):948–960, 1972. 17

[59] Hennessy, John L. e David A. Patterson: *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 6th edição, 2017. 17, 18

[60] OpenMP Architecture Review Board: *OpenMP Application Program Interface Version 6.0*, November 2024. `https://www.openmp.org/specifications/`. 17

[61] Chapman, Barbara, Gabriele Jost e Ruud Van Der Pas: *Using OpenMP: Portable Shared Memory Parallel Programming*. MIT Press, 2007. 17

[62] Sokolova, Marina e Guy Lapalme: *A systematic analysis of performance measures for classification tasks*. Information Processing & Management, 45(4):427–437, 2009. 17

[63] Yianilos, Peter N.: *Data structures and algorithms for nearest neighbor search in general metric spaces*. Em *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '93, página 311–321, USA, 1993. Society for Industrial and Applied Mathematics, ISBN 0898713137. 21

[64] Bentley, J. L.: *Multidimensional binary search trees used for associative searching*. Communications of the ACM, 18(9):509–517, 1975. 21

[65] Omohundro, Stephen M.: *Five balltree construction algorithms*. Em *International Computer Science Institute Technical Report*, número ICSI TR-89-063, 1989. 21

[66] Cruz, Jan Christian Blaise, Julianne Agatha Tan e Charibeth Cheng: *Localization of fake news detection via multitask transfer learning*. Em *Proceedings of the 12th Language Resources and Evaluation Conference*, páginas 2596–2604, 2020. 27

[67] Huggingface: *Fake news filipino (jcblaise/fake_news_filipino)*. `https://huggingface.co/datasets/jcblaise/fake_news_filipino`, 2020. Accessed: 2025-05-20. 27

[68] Alakuijala, Jyrki, Andrea Farruggia, Paolo Ferragina, Eugene Kliuchnikov, Robert Obryk, Zoltan Szabadka e Lode Vandevenne: *Brotli: A general-purpose data compressor*. ACM Transactions on Information Systems, 37:1–30, dezembro 2018. 28

[69] Seward, Julian: *bzip2: A block-sorting file compressor*. Source code and documentation, 1996. Released July 1996; official site: `https://www.bzip.org/`. 28

[70] Gailly, Jean-loup e Mark Adler: *gzip: Gnu file compression utility*. Software and format specification, 1992. Initial release 31 October 1992; official site: `https://www.gnu.org/software/gzip/`. 28

[71] Gailly, Jean-loup e Mark Adler: *zlib: A lossless data compression library*. Software library and documentation, 1995. First released May 1, 1995; official site: `https://zlib.net/`. 28

[72] Collet, Yann: *Zstandard (zstd): A fast lossless compression algorithm*. Reference C implementation and specification, 2016. First released August 31, 2016; format standardized in IETF RFC 8878 (February 2021); official site: `https://facebook.github.io/zstd/`. 28

[73] Avaneev, Dmitry: *LZAV: Fast in-memory lz77-based data compressor (header-only c/c++)*. GitHub repository, 2023. Available at `https://github.com/avaneev/lzav`, with performance around 480 MB/s compression and 2800 MB/s decompression :contentReference[oaicite:1]index=1. 28

[74] Lehmann, Marc A.: *LibLZF: A very small and fast lz77-based compression library*. Project homepage, 2008. Last updated August 25, 2008; BSD-style license; official site: `https://oldhome.schmorp.de/marc/liblzf.html`. 28

[75] Reinhold, Lasse Mikkel: *QuickLZ: A very fast lz compression library*. Official website and source code, 2011. Described as the world's fastest compression library (308 MB/s); original C version v1.5.0 available at `http://www.quicklz.com/`, and ports in Go and Rust exist :contentReference[oaicite:1]index=1. 28

[76] Schramm, Christian: *shoco: A fast compressor for short strings*. GitHub repository, 2015. C library optimized for very short strings; MIT license; `https://github.com/Ed-von-Schleck/shoco`. 28

[77] Dean, Jeff, Sanjay Ghemawat e Steinar H. Gunderson: *Snappy: A fast compressor/decompressor*. Software library, 2011. Open-sourced by Google; C++ implementation with 250MB/s compression, 500MB/s decompression; `https://google.github.io/snappy/`. 28

[78] Boncz, Peter, Thomas Neumann e Viktor Leis: *FSST: Fast static symbol table string compression*. CWI / research publication and GitHub, 2019. Lightweight random-access string compression; `https://github.com/cwida/fsst`. 28

[79] Sanfilippo, Salvatore: *Smaz: Small string compression library*. GitHub repository, 2012. Compresses very short strings (e.g. the → 1 byte); BSD-3; `https://github.com/antirez/smaz`. 28