Learned Index Baseado na Predição de Vértices iniciais em Grafos de Proximidade

Matheus B. Bastos[1], **Daniel S. Kaster**[1]

[1]Departamento de Computação – Universidade Estadual de Londrina (UEL)
Caixa Postal 10.011 – 86.057-970 – Londrina – PR – Brasil

mathheusb@gmail.com, dskaster@uel.br

Abstract. *Similarity searches on complex data rely on methods to speed up queries, such as spatial approximation on proximity graphs, which suffer from the initial vertex selection problem. Recently, learned indexes have emerged using machine learning to model the data distribution and other characteristics, showing promising results. This paper presents an efficient learned index on graph-based methods for similarity searching. Our method employs supervised learning to predict the seed vertices. The method reduced the distance computations needed to obtain high recall rates on various datasets evaluated.*

Resumo. *Buscas por similaridade em dados complexos utilizam métodos para acelerar consultas, como a aproximação espacial em grafos de proximidade, que sofre com o problema da escolha do vértice inicial. Recentemente, learned indexes surgiram utilizando aprendizado de máquina para modelar a distribuição de dados e outras características, apresentando resultados promissores. Este artigo apresenta um learned index eficiente sobre métodos baseados em grafos para buscas por similaridade. O método proposto aplica aprendizado supervisionado para predição dos vértices iniciais de busca. O método reduziu o número de computações de distância necessárias para alcançar uma alta taxa de recall em vários conjuntos de dados avaliados.*

1. Introdução

Recentemente, aplicações que usam dados complexos viram sua utilização crescer, assim como a necessidade de armazenar esses dados. Exemplos incluem imagens, sons e dados georreferenciados. Diferentemente dos estruturados, dados complexos não podem ser ordenados e comparações exatas não fazem tanto sentido. No caso desses tipos de dados, buscas por similaridade são mais adequadas, retornando, por exemplo, quais são os k elementos mais similares a um elemento de consulta. Para realizar buscas por similaridade, várias estruturas de índice foram propostas na literatura, entre elas as baseadas em grafos de proximidade [Hajebi et al. 2011].

O *HGraph* [Shimomura and Kaster 2019] carrega grande importância para este trabalho. Ele é um método para construção de grafos de proximidade, como grafos *k-NN*. A construção é feita através de uma estratégia de divisão e conquista, que cria várias partições. Elas são conectadas através de regiões de sobreposição e *long-range edges*, arestas criadas posteriormente que aumentam a qualidade das consultas. Com o grafo construído, um algoritmo baseado em *aproximação espacial* [Navarro 2002] pode ser utilizado para realizar consultas sobre ele. Um problema nesses algoritmos é quando o

vértice inicial está "muito longe" do resultado ideal. Dependendo do tamanho do grafo, essa condição causa uma necessidade de percorrer uma região muito grande do grafo [Shimomura and Kaster 2019]. Para contornar esse problema, é necessário determinar os vértices iniciais de busca de maneira mais adequada.

Nos últimos anos, *learned indexes* surgiram como fortes alternativas ou complementos a estruturas tradicionais [Antol et al. 2021, Kraska et al. 2018]. Utilizando modelos de aprendizado de máquina, muitos deles conseguem se aproveitar de padrões existentes nos dados para acelerar consultas. O *Learned Metric Index* [Antol et al. 2021], por exemplo, pode se basear em índices como o M-Index [Novak et al. 2011] para construir sua própria estrutura constituída por modelos, eliminando cálculos de distância em passos intermediários da busca.

Este trabalho propõe um novo *learned index* baseado em grafos construídos por particionamento, como é o caso do *HGraph*. A ideia principal é aproveitar a estrutura do grafo para treinar modelos que terão o papel de determinar os vértices iniciais das consultas por similaridade. Com a predição, as buscas são iniciadas em regiões mais adequadas e necessitam de menos cálculos de distância para atingir o mesmo resultado. Para tal, uma estrutura auxiliar ao *HGraph* foi implementada utilizando duas abordagens: (1) utilizando um único modelo e (2) utilizando uma hierarquia de modelos. As predições dos modelos são utilizadas como ponto de partida para as buscas.

Experimentos utilizando diferentes conjuntos de dados indicam que as abordagens propostas possibilitaram uma redução significativa de cálculos de distância necessários para obter um nível de *recall* equivalente, quando comparadas ao grafo original.

2. Fundamentos e Trabalhos Relacionados

Grafos de proximidade são grafos comumente utilizados para a realização de consultas por similaridade [Ocsa et al. 2007]. Neles, uma propriedade P, chamada de *critério de vizinhança*, é definida, e cada par de vértices $u, v \in V$ é conectado por uma aresta $e \in E$ se, e somente se, eles satisfizerem a propriedade P. As arestas geralmente tem como peso a distância entre os dois elementos correspondentes aos vértices. Esses grafos refletem a proximidade entre os elementos conectando objetos espacialmente próximos.

Nesse tipo de grafo, consultas de similaridade podem ser respondidas através da *aproximação espacial*, definida por [Navarro 2002]. Sendo $N(u)$ os vizinhos de u, inicia-se o processo em um elemento qualquer de V, que é atribuído à variável a. Em cada passo, é escolhido um dos vizinhos $b \in N(a)$, o qual deve ter menos distância ao elemento de consulta do que a e todos os seus outros vizinhos, e atribuído b à variável a. O processo é repetido até que não seja possível escolher um vizinho de a com esse critério, o que significa que a é o vértice mais próximo ao objeto de consulta.

Os algoritmos de aproximação espacial sofrem com um problema: se os vértices iniciais escolhidos estiverem muito "longe" do objeto de consulta – o que não é incomum, principalmente quando a escolha é aleatória – será necessário percorrer uma grande parte do grafo para atingir a resposta ideal, realizando cálculos de distância excessivos. Ainda mais, em consultas aproximadas, buscas gulosas podem retornar respostas longe da ideal. Algoritmos como o GNNS [Hajebi et al. 2011] contornam o problema realizando várias buscas, denominadas *restarts*, cada uma iniciando em um vértice aleatório dife-

rente. Porém, em alguns casos é necessário realizar muitos *restarts* para atingir uma taxa de *recall* satisfatória.

Para melhorar esse cenário e evitar que vértices inadequados sejam escolhidos, técnicas de aprendizado de máquina podem ser empregadas. O uso de aprendizado de máquina para realizar consultas em conjuntos de dados tem sido o foco de vários trabalhos recentes. Esse tipo de índice é chamado de ***learned index***.

Um dos precursores dos *learned indexes* é o *Recursive Model Index*, ou RMI [Kraska et al. 2018], constituído por uma hierarquia de modelos. Seu objetivo é aproximar a função de distribuição acumulada do conjunto de dados ordenado, assim sendo capaz de predizer posições para chaves de entrada. Na predição, cada nível na hierarquia é responsável por escolher um modelo do nível abaixo, até que na folha a posição é predita. Por exigir que o domínio de dados seja ordenável, a estrutura não é adequada para dados complexos e buscas por similaridade.

Outro *learned index* é o *Learned Metric Index*, ou LMI [Antol et al. 2021]. Ele usa como base outro índice como o *M-Index* [Novak et al. 2011] para criar uma nova estrutura que substitui a original. Essa estrutura também é constituída por uma hierarquia de modelos, usando como *label* na predição o *cluster* em que cada cada objeto foi inserido na estrutura original. Ao final, são retornados possíveis *clusters* onde a resposta se encontra, que devem ser varridos sequencialmente. O LMI é capaz de indexar conjuntos de dados em um espaço métrico, não exigindo que sejam ordenáveis, assim sendo adequado para o uso em dados complexos.

Para empregar aprendizado de máquina em uma busca por aproximação espacial, é necessário que haja alguma noção de posição ou *cluster* no grafo que possa ser usada na predição. Nesse contexto, este trabalho utiliza o *HGraph* [Shimomura and Kaster 2019] como base. O *HGraph* é um método que constrói grafos de proximidade através do particionamento do conjunto de dados, utilizando uma estratégia de divisão e conquista. Seu objetivo é acelerar a construção dos grafos através do potencial de paralelização, ao mesmo tempo que adiciona características capazes de melhorar a qualidade das consultas.

O particionamento no *HGraph* é feito de forma recursiva: em cada passo, são selecionados elementos aleatórios para serem pivôs. Então, cada um dos elementos restantes é adicionado ao subconjunto do pivô mais próximo. O processo se repete até que seja atingida uma determinada cardinalidade em cada subconjunto, quando em cada um é construído um grafo como o *k-NNG*. Os vários grafos construídos são conectados através de: (1) arestas de longa distância, que conectam pivôs e não seguem o critério de proximidade; e (2) regiões de sobreposição, onde vértices próximos à "borda" da partição são duplicados, ficando presentes em duas partições e, assim, conectando-as. Ao final, tem-se uma aproximação do grafo original. A próxima seção apresenta a proposta do trabalho.

3. Proposta de um *Learned Index* Baseado na Predição de Vértices Iniciais

A proposta apresentada neste trabalho tira proveito do particionamento realizado pelo *HGraph*, que tende a agrupar elementos similares, para determinar em que parte do grafo uma busca por aproximação espacial será iniciada. Isso pode ser feito através da predição da partição mais próxima ao objeto de consulta com modelos de aprendizado de máquina. Dessa forma, o *label* atribuído aos elementos e utilizado no treinamento dos modelos

é uma posição na estrutura original, que no caso é um grafo. O objetivo é trazer uma nova abordagem ao problema da escolha dos vértices iniciais, na tentativa de iniciar a aproximação espacial em um vértice mais próximo do resultado ideal, assim necessitando menos cálculos de distância.

A proposta adiciona alguns passos após a construção do *HGraph*, onde modelos de classificação são treinados. Além disso, o GNNS [Hajebi et al. 2011], principal algoritmo de busca utilizado nos experimentos, foi alterado para predizer os vértices iniciais ao invés de escolhê-los aleatoriamente. A implementação foi feita parte em C++, junto ao código do *HGraph*, e parte em Python, utilizando a biblioteca *scikit-learn* [Pedregosa et al. 2011] nas partes correspondentes ao treinamento e predição. O classificador *Random Forest* [Cutler et al. 2012] foi utilizado para induzir os modelos.

Logo após o término do processo de construção do *HGraph*, é gerado um conjunto de dados contendo as características dos elementos correspondentes a cada vértice do grafo, rotulados com o identificador da partição onde se encontram. O identificador é formado pelo "caminho" na árvore de recursão percorrido durante o processo de particionamento. Dessa forma, ele carrega informações que podem ser aproveitadas para agrupar partições. O conjunto de dados gerado é tratado antes de ser utilizado para treinamento. Devido às regiões de sobreposição do *HGraph*, muitos elementos aparecem duplicados. Para evitar problemas no treinamento, os dados são deduplicados, mantendo-se os mais próximos a algum pivô.

Com o conjunto de dados gerado e tratado, são propostas duas abordagens para realizar o treinamento e predição em cima dele: (1) modelo único, onde o conjunto de dados é usado diretamente para treinar um único classificador, que então já é capaz de realizar as predições desejadas; e (2) hierarquia de modelos, onde há dois ou mais níveis de modelos, sendo que cada um dos modelos de um nível inferior é responsável por uma parte do conjunto. A primeira abordagem pode ser suficiente em vários casos, principalmente em conjuntos de dados com baixa cardinalidade/dimensionalidade. A segunda, mais frequentemente adotada em *learned indexes*, tem a motivação de tornar a tarefa de classificação mais simples para os modelos, pois o número de classes envolvidas é reduzido nos casos individuais, o que é particularmente relevante para conjuntos de dados maiores. Contudo, é importante manter um balanço entre o número reduzido de classes e o número significativo de instâncias de cada classe, para possibilitar um aprendizado adequado.

A construção de uma hierarquia de modelos é feita de maneira *bottom-up*. Primeiramente, é necessário formar agrupamentos de partições, os quais corresponderão a um modelo de nível folha. Assim, um modelo de nível folha será responsável por realizar predições apenas entre as partições que estão no seu grupo. É importante que partições similares sejam agrupadas para que as predições tenham boa qualidade. Há várias maneiras de realizar os agrupamentos. A proposta apresentada neste artigo é chamada *união de partições irmãs*. Nesta proposta, partições irmãs são partições que surgiram de uma mesma divisão no processo de particionamento do *HGraph*, ou seja, têm o mesmo nó pai na árvore de recursão. Assim, são formados grupos com no máximo N partições, tentando unir sempre partições irmãs, porém não formando grupos com apenas uma partição. A implementação usa os identificadores das partições para determinar se duas partições são irmãs: basta verificar se o caminho até as duas na recursão é o mesmo. Na Figura 1(a) é

exibido um exemplo de aplicação do algoritmo. Com os agrupamentos feitos, é treinado um modelo para cada grupo, utilizando do conjunto de dados apenas os elementos pertencentes às partições do grupo. Por fim, o modelo raiz é treinado, e será responsável por escolher qual dos modelos de nível folha utilizar. Para seu treinamento, é utilizado o conjunto gerado por inteiro, mas os *labels* são trocados: cada elemento deve ser rotulado por um identificador do grupo de partições no qual ele está inserido, e não mais da partição. Dessa maneira, o número de classes é dividido por aproximadamente N.

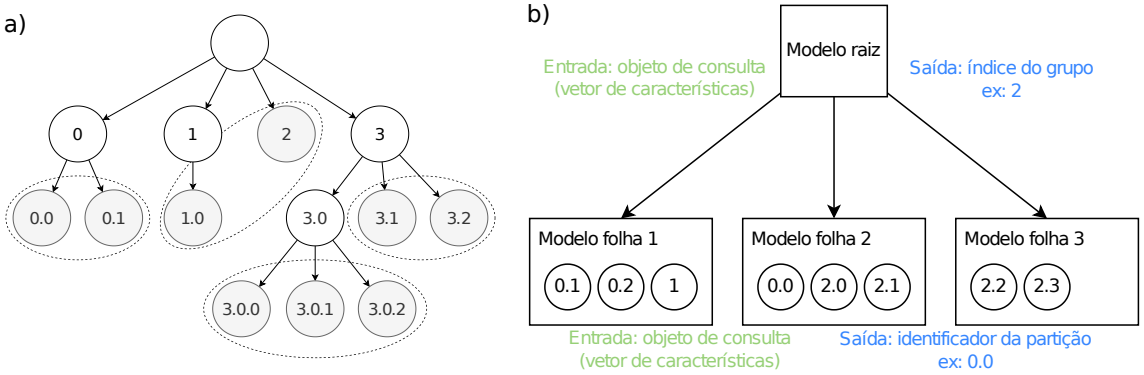

Figura 1. a) Exemplo de aplicação do agrupamento por partições irmãs com $N = 3$. b) Processo de predição em uma hierarquia de modelos.

Assim, os modelos treinados podem ser utilizados na fase de busca. O algoritmo GNNS, antes de iniciar a busca, foi alterado para realizar uma predição com o objeto de consulta. No caso da abordagem com modelo único, o identificador da partição inicial é retornado diretamente. Já na hierarquia de modelos, o modelo raiz prediz um modelo de nível folha, o qual efetivamente retorna uma partição com base no objeto de consulta, como pode ser visto na Figura 1(b). Em ambos os casos, com a partição escolhida, os *restarts* do algoritmo são feitos em seus vértices.

4. Experimentos

Esta seção apresenta resultados de experimentos comparando os métodos propostos com o *baseline* sem o seu uso, todos implementados em C++ usando a biblioteca *Non-Metric Space Library* [Boytsov and Naidan 2013], com os métodos propostos fazendo chamadas em Python. O *HGraph* original é rotulado como *HGraph*; o método utilizando a abordagem hierárquica, como *HGraph_H*; e o com a abordagem de modelo único, como *HGraph_S*. Os resultados apresentados neste trabalho referem-se aos conjuntos de dados listados na Tabela 1. A execução dos testes foi *single-threaded* em um Intel(R) Core(TM) i7-8700 e 32GB de *RAM* a 2666 MHz, executando o Ubuntu 20.04.2 LTS.

Os parâmetros utilizados na *RandomForestClassifier*, do *scikit-learn*, foram 100 árvores (*n_estimators*) e 30 de profundidade máxima (*max_depth*). No *HGraph*, foi escolhido o grafo *kNNG*, com $o = 0{,}1$ e $n_P = 10$, $NN = \{5, 10, 55\}$ e $m = \{5000, 10000\}$. Foram feitas consultas *k-NN* com $k = \{1, 10, 30\}$ e número de *restarts* $r = \{1, 5, 10, 20, 40, 80, 120, 240\}$.

Para analisar os ganhos dos métodos propostos, foram feitas comparações entre os diferentes métodos relacionando o número de computações de distância feitas nas consul-

[1]https://archive.ics.uci.edu/ml/datasets/corel+image+features

Tabela 1. Conjuntos de dados utilizados nos experimentos.

Dataset	Elementos	Dimensões	Fonte
Color Moments	68.040	9	*Corel Image Features* [1]
Color Histogram	68.040	32	*Corel Image Features*
CoPhIR Colour Layout	1.000.000	12	CoPhIR [Bolettieri et al. 2009]

tas e o *recall* atingido. A Figura 2 mostra que abordagem hierárquica apresentou melhor desempenho, com menos cálculos sendo exigidos para atingir um mesmo valor de *recall* no conjunto *CoPhIR Colour Layout*. A de modelo único vem em segundo lugar, ainda apresentando ganhos em relação ao *HGraph* original. Os parâmetros utilizados neste teste foram $NN = 10$ e $m = 5000$. A diferença entre a abordagem hierárquica e a simples foi reduzida à medida que o valor do k da consulta k-NN aumentou, mas o ganho em comparação ao original permaneceu existindo.

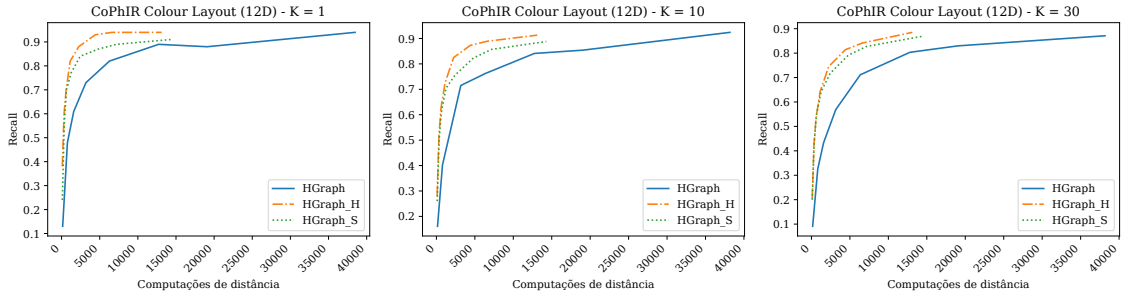

Figura 2. Desempenho das propostas com variação do k em consultas k-NN no conjunto *CoPhIR Colour Layout*.

O parâmetro NN do *kNNG* determina o número de vizinhos mais próximos que serão conectados a cada vértice através de arestas no grafo. A fim de analisar o impacto desse parâmetro, foram fixados $m = 5000$ e $k = 10$. A Figura 3 mostra um sumário dos cenários para o conjunto *Corel Color Moments* por meio de *boxplots*. Evidentemente, quanto maior o NN, maior o *recall*. Em contrapartida, mais cálculos de distância são feitos. Nota-se que, à medida que valor de NN cresce, os ganhos das novas abordagens em questão de *recall* são reduzidos. Isso é mais visível em conjuntos de dados menores e pode ser explicado pela seleção do vértice inicial ter um impacto menor na aproximação espacial quando há tantas arestas em relação ao valor de k. Em compensação, esse valor pode causar mais cálculos de distância, reduzidos significativamente pelos métodos propostos, além de aumentar o tempo de construção do grafo.

Por fim, foi feita uma análise para cada conjunto de dados em cima das diferentes configurações que foram capazes de obter valores de *recall* $\geq 0{,}9$ nas consultas. As configurações foram ordenadas por cálculos de distância realizados e as 10 melhores de cada conjunto de dados foram selecionadas. A Figura 4 mostra o resultado dessa análise para consultas com $k = 10$. Várias configurações do *HGraph* com predição de vértice inicial foram capazes de diminuir os cálculos de distância necessários para se obter um recall maior ou igual a $0{,}9$ nos conjuntos avaliados. A figura mostra que houve claros ganhos para o conjunto *CoPhIR Colour Layout*, onde a melhor configuração do *HGraph* original teve que realizar 138% mais cálculos de distância do que a abordagem hierárquica

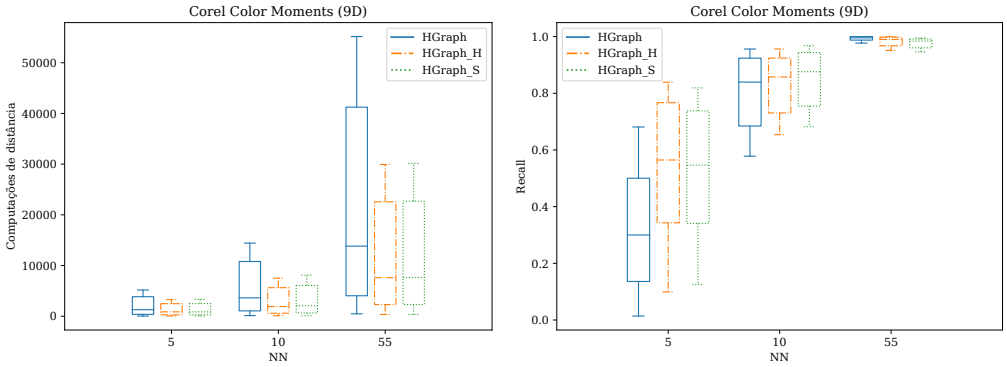

Figura 3. Computação de distâncias e *recall* variando-se o parâmetro NN no conjunto *Color Moments*.

com $m = 10000$ e $NN = 55$. A Figura 4 também mostra que os métodos propostos foram as melhores opções para o conjunto *Corel Color Histogram*, exigindo até 40% menos cálculos do que a melhor configuração do *HGraph* original.

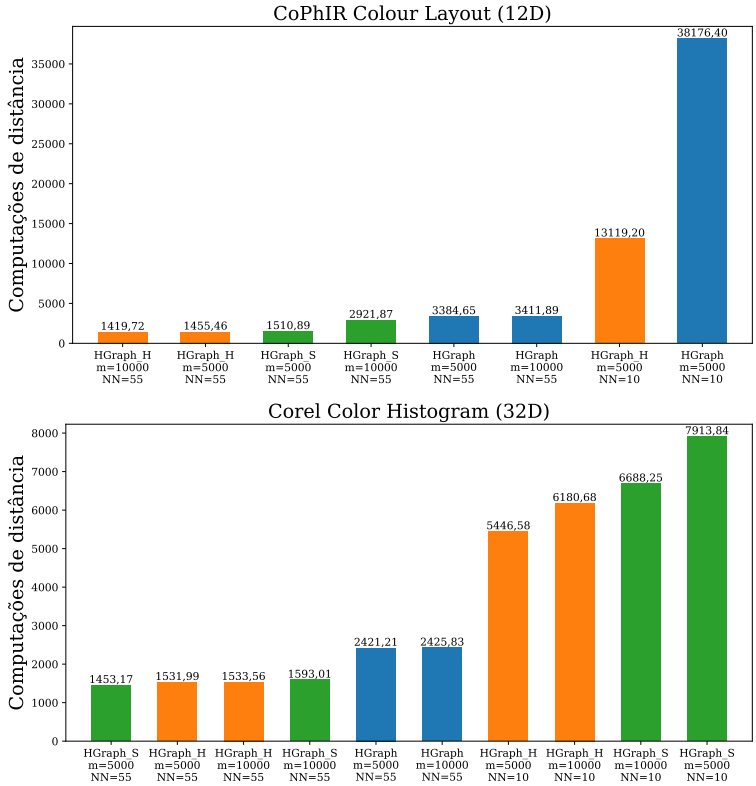

Figura 4. Configurações com *recall* $\geq 0{,}9$ para consultas 10-NN nos conjuntos *CoPhIR Colour Layout* e *Corel Color Histogram*.

5. Conclusão

Neste artigo, foi apresentado um novo *learned index* baseado em grafos construídos com particionamento, como o *HGraph*. O índice se aproveita do particionamento realizado durante a construção do grafo para treinar um classificador (um modelo único ou uma

hierarquia de modelos), o qual é capaz de prever uma posição inicial adequada para determinada busca se iniciar. Nos experimentos, ambas as abordagens foram capazes de reduzir o número de cálculos de distância necessário para se atingir um *recall* satisfatório.

Como trabalhos futuros, destacam-se uma experimentação mais extensa para analisar compromissos da abordagem, como número de modelos (classes) por nível, avaliar impactos nos tempos de construção e consulta e realizar comparações com outras estruturas na literatura, em particular, o LMI. Além disso, conjuntos de dados maiores e/ou com mais dimensionalidade devem ser testados, a fim de verificar se os ganhos são intensificados nesses casos. Também, outros tipos de modelos de classificação devem ser explorados, como redes neurais e redes neurais de grafos.

Referências

Antol, M., Ol'ha, J., Slaninákoválá, T., and Dohnal, V. (2021). Learned metric index—proposition of learned indexing for unstructured data. *Information Systems*, 100:101774.

Bolettieri, P., Esuli, A., Falchi, F., Lucchese, C., Perego, R., Piccioli, T., and Rabitti, F. (2009). Cophir: a test collection for content-based image retrieval. *arXiv preprint arXiv:0905.4627*.

Boytsov, L. and Naidan, B. (2013). Engineering efficient and effective non-metric space library. In *International Conference on Similarity Search and Applications*, pages 280–293. Springer.

Cutler, A., Cutler, D. R., and Stevens, J. R. (2012). Random forests. In *Ensemble machine learning*, pages 157–175. Springer.

Hajebi, K., Abbasi-Yadkori, Y., Shahbazi, H., and Zhang, H. (2011). Fast approximate nearest-neighbor search with k-nearest neighbor graph. In *Twenty-Second International Joint Conference on Artificial Intelligence*.

Kraska, T., Beutel, A., Chi, E. H., Dean, J., and Polyzotis, N. (2018). The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*, pages 489–504.

Navarro, G. (2002). Searching in metric spaces by spatial approximation. *The VLDB Journal*, 11(1):28–46.

Novak, D., Batko, M., and Zezula, P. (2011). Metric index: An efficient and scalable solution for precise and approximate similarity search. *Information Systems*, 36(4):721–733.

Ocsa, A., Bedregal, C., and Cuadros-Vargas, E. (2007). A new approach for similarity queries using neighborhood graphs. In *SBBD*, pages 131–142.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830.

Shimomura, L. C. and Kaster, D. S. (2019). Hgraph: a connected-partition approach to proximity graphs for similarity search. In *International Conference on Database and Expert Systems Applications*, pages 106–121. Springer.