Junções por Similaridade em Espaços Vetoriais Semânticos

Douglas Rolins de Santana[1], **Leonardo Andrade Ribeiro**[1]**(orientador)**

[1]Programa de Pós-Graduação em Ciência da Computação
Instituto de Informática (INF) – Universidade Federal de Goiás (UFG)

douglasrolins@discente.ufg.br,laribeiro@inf.ufg.br

Nível: Mestrado, **Ingresso:** Março de 2021, **Previsão da defesa:** Março de 2023, **Exame de qualificação:** Junho de 2022.

Abstract. *Similarity join returns all pairs of objects whose similarity is not less than a specified threshold. This operation is of fundamental importance for data cleaning and integration. A popular approach is to adopt a vector-space representation and use cosine to measure the similarity of two vectors. Calculating cosine similarity over all vector pairs is prohibitively expensive for large datasets. State-of-the-art similarity join algorithms exploit the sparsity of the vector space generated by traditional methods to derive filters and, thus, reduce the comparison space. Recent advances in natural language processing have enabled representations based on semantically richer vectors, thereby increasing the quality of the results. However, such vectors exhibit different characteristics from those generated by traditional methods. In particular, these vectors are dense and have lower dimensionality. In this context, we present a research proposal centered around two main questions: 1) how do these characteristics affect the performance of existing similarity join algorithms?; 2) is it possible to achieve performance in this new representation comparable to previous methods while maintaining the quality of the results? This paper further describes our research methodology, presents some initial results, and outlines future work.*

Resumo. *Junção por similaridade retorna todos os pares de objetos cuja similaridade não é menor que um limite especificado. Essa operação é de fundamental importância para limpeza e integração de dados. Uma abordagem popular é adotar uma representação em espaço vetorial e usar cosseno para medir a similaridade de dois vetores. Calcular a similaridade de cosseno em todos os pares de vetores é proibitivamente custoso para grandes conjuntos de dados. Algoritmos de junção de similaridade exploram a esparsidade do espaço vetorial gerado por métodos tradicionais para derivar filtros e, assim, reduzir o espaço de comparação. Avanços recentes no processamento de linguagem natural possibilitaram representações baseadas em vetores semanticamente mais ricos, aumentando assim a qualidade dos resultados. No entanto, tais vetores apresentam características diferentes daquelas geradas por métodos tradicionais. Em particular, esses vetores são densos e possuem menor dimensionalidade. Nesse contexto, apresentamos uma proposta de pesquisa centrada em duas questões principais: 1) como essas características afetam o desempenho dos algoritmos de junção por similaridade existentes?; 2) é possível obter desempenho nesta nova representação comparável aos métodos anteriores mantendo a qualidade dos resultados? Este artigo descreve ainda a metodologia de pesquisa, apresenta alguns resultados iniciais e delineia trabalhos futuros.*

1. Introdução

A qualidade dos dados é um aspecto de extrema relevância em *Data Science*, pois inconsistências podem comprometer os resultados. Por este motivo, as fases de preparação e limpeza são imprescindíveis em um processo de análise de dados. Um tipo comum de inconsistência é presença de duplicatas, isto é, múltiplas e não idênticas representações de uma entidade do mundo real. Neste contexto, a tarefa em um processo de limpeza de dados consiste em identificar e corrigir essas duplicatas antes do início das atividades de análise. Esta tarefa foi amplamente pesquisada por uma variedade de abordagens diferentes mas ainda continua sendo um problema desafiador e com espaço para melhorias [Mudgal et al. 2018].

Uma das principais operações usadas para identificação de duplicatas é a junção por similaridade. Primeiramente, para se definir o valor em que dois objetos são semelhantes podem ser utilizadas funções de similaridade [Ribeiro and Härder 2011]. Uma função de similaridade busca calcular o valor da similaridade entre dois objetos no conjunto de dados. Neste contexto, a junção por similaridade retorna todos os pares (x, y) em um conjunto de dados cuja similaridade seja maior que t, ou seja, $sim(x, y) \geq t$ [Chaudhuri et al. 2006]. Além da identificação de duplicatas, junções por similaridade são amplamente utilizadas em uma variedade de aplicações como detecção de plágio, refinamento de consultas, filtragem colaborativa, classificação de texto, extração de entidades, integração de dados, mineração de dados, dentre outras.

Antes da aplicação de uma função de similaridade, tipicamente os dados são organizados em uma representação que possibilite a realização de operações nos mesmos. Uma das estruturas utilizada é o vetor [Salton et al. 1975] e uma abordagem comum consiste em representar os dados em formato numérico com uso da medida estatística TF-IDF. Nesta representação em espaço vetorial, uma função comum utilizada para definir a similaridade é a função cosseno. Trabalhos mais recentes, ao invés de definir um formato com o TF-IDF e aplicar uma função tradicional de similaridade, utilizam técnicas de *machine learning* para aprender uma representação que capture o contexto e relacionamento entre objetos. Esta abordagem resultou em maior qualidade nos resultados em relação à semântica, especialmente em dados textuais não estruturados [Mudgal et al. 2018, Li et al. 2020]. Porém, estas técnicas demandam tempo considerável para treinar modelos, o que para aplicações em tempo real pode representar um obstáculo.

Com a chegada da arquitetura *Transformer* [Vaswani et al. 2017], surgiram modelos pré-treinados em grandes bases de dados e que, devido maior capacidade de generalização, conseguem gerar melhores representações dos dados textuais bem como representam o estado da arte em diversas aplicações. A partir destas melhores representações semânticas dos dados textuais, este trabalho de pesquisa investiga a junção por similaridade com uso de função cosseno nessas representações. O objetivo é verificar se com dados textuais representados com a arquitetura *Transformer* e com aplicação de algoritmo otimizado para junção por similaridade com função cosseno é possível aliar eficácia (i.e., qualidade de resultados) e eficiência.

Neste trabalho, foram definidas as seguintes questões de pesquisa: (1) como as características dos espaços vetoriais semânticos impactam no desempenho de algoritmos de junção por similaridade existentes?; e (2) é possível obter desempenho nesta nova representação comparável aos métodos anteriores mantendo a qualidade dos resultados?

As seguintes contribuições são esperadas com este trabalho de pesquisa: (1) algoritmo otimizado para junção por similaridade em vetores densos e (2) aumento na qualidade dos resultados em comparação com abordagens tradicionais em cenários de aplicação que demandam contextualização semântica. Este trabalho é voltado para dados textuais, porém, as técnicas desenvolvidas possivelmente poderão ser relevantes para outros tipos de dados, como imagens e vídeos.

2. Trabalhos Relacionados

Dentre os vários trabalhos de junção por similaridade com função cosseno destaca-se o L2AP [Anastasiu and Karypis 2014] por demonstrar superioridade nas técnicas de filtragem e assim reduzir substancialmente os custos computacionais da operação. O desenvolvimento e experimentos do trabalho de Anastasiu e Karypis é voltado para vetores esparsos e nesta pesquisa iremos aplicá-lo, e investigar possibilidades de otimização, em espaços vetoriais semânticos que basicamente são vetores densos de tamanho fixo.

Com objetivo de obter melhores resultados no contexto semântico, especialmente em aplicações que exigem maior significado em linguagem natural, *DeepMatcher* [Mudgal et al. 2018] fornece uma variedade técnicas baseada em *Deep Learning* para diversas tarefas, de forma a produzir o aprendizado das representações que capturam a similaridade entre as instâncias de dados. Já em *Ditto* [Li et al. 2020], utiliza-se modelos baseados em *Transformers* pré-treinados bem como *fine-tuning* na aplicação alvo para superar o *DeepMatcher* na qualidade dos resultados. *Ditto* utiliza ainda blocagem para reduzir a quantidade de pares a serem considerados. Mais recentemente, o *Ember* [Suri et al. 2021], através de uma arquitetura em três etapas (pré-processamento, aprendizado da representação e junção), obtém resultados superiores ao *Ditto* em relação a revocação e nos tempos de consulta referentes a único registro; porém, *Ditto* ainda permanece com melhores resultados na busca de todos os pares. *Ember* utiliza a biblioteca baseada em GPU Faiss [Johnson et al. 2019] para criar um índice na fase de junção.

Estes trabalhos que envolvem o contexto semântico apresentam os custos computacionais como desafio pois, como observado em *Ditto*, um pipeline completo foi realizado em 6.3 horas em uma base de dados com 10 milhões de pares candidatos, além da necessidade de se usar *hardware* com GPU, no caso do *Ember*, para otimizar a fase de junção. Desta forma, este trabalho busca investigar técnicas que possam ser executada em CPU, tendo como *baseline* o algoritmo L2AP. Os conjuntos de dados utilizados nos trabalhos do *DeepMatcher*, *Ditto* e *Ember* foram as mesmas, bem como serão utilizadas nesta pesquisa para fins de comparação dos resultados. Vale ressaltar que otimizações em técnicas para CPU podem ser empregadas e/ou adaptadas para GPU [Ribeiro-Júnior et al. 2017].

3. Fundamentação Teórica

Uma tarefa inicial para representar dados textuais em vetores, é transformar o texto em unidades atômicas chamadas *tokens*; esta tarefa tem o nome de *tokenização*. *Tokens* podem ser palavras, ou ainda sub-divisões do texto denominados *q-grams*. A partir da tokenização é possível associar valores aos tokens para obter o vetor numérico que representa o texto. Estratégias tradicionais utilizam medidas estatísticas, como TF-IDF:

$$TF\text{-}IDF(t) = tc \times \log(\frac{N}{sc(t)}), \quad (1)$$

onde tc representa a quantidade de vezes que o token t aparece na sentença, N o total de sentenças do conjunto de dados e $sc(t)$ a quantidade de sentenças em que o token t aparece no conjunto de dados. Para junção por similaridade em vetores normalizados para uma unidade de comprimento, o cosseno é igual ao produto escalar entre os vetores:

$$sim(x,y) \;=\; \frac{\sum_{i=1}^{d} x_i \times y_i}{||x|| \times ||y||} \;=\; dot(x,y) \;= \sum_{i=1}^{d} x_i \times y_i \tag{2}$$

Na representação TF-IDF, seja com tokenização por palavras ou q-gram, o valor de cada token é definido de maneira independente capturando apenas a sintaxe; o contexto e o relacionamento semântico com demais tokens não é capturado. Desta forma, outras maneiras de representar o texto foram desenvolvidas, e principalmente com o trabalho de Mikolov *et al.* [Mikolov et al. 2013], houve avanços significativos para capturar os relacionamentos semânticos do texto. Neste caso são gerados vetores densos e de tamanho fixo, com técnica chamada de *word embeddings*.

A introdução da arquitetura denominada *Transformer* [Vaswani et al. 2017] causou grande impacto na área de processamento de linguagem natural (PLN), demonstrando desempenho e capacidade de generalização superior às demais técnicas. Com isso, surgiram modelos pré-treinados em grandes quantidades de dados e disponibilizados publicamente, como o modelo *BERT* [Devlin et al. 2019]. Apesar de gerar melhores representações do texto, o modelo *BERT* não foi construído para verificar a similaridade semântica entre sentenças, pois utiliza representação em nível de palavra e não em nível de sentença. Assim, para realizar uma pesquisa por similaridade entre duas sentenças, necessitaria que ambas sejam alimentadas na rede, o que gera grande custo computacional em situações como encontrar o par mais semelhante em todo o conjunto de dados, necessitando de diversas horas para os cálculos de inferência [Reimers and Gurevych 2019].

Como alternativa foi publicado o modelo SBERT [Reimers and Gurevych 2019], que produz representações vetoriais de sentenças sem realizar cálculos de inferência completos. A partir destas representações, para verificar a semelhança semântica de pares, pode ser utilizado a similaridade de cosseno. Outros modelos similares ao SBERT foram desenvolvidos e até o superaram, bem como estão disponíveis publicamente e hospedados pela organização Hugging Face[1]. Através de um framework, desenvolvido também por Reimers e Gurevych, nomeado de *Sentence-Transformers*[2], é possível trocar por modelos mais recentes e realizar *fine-tuning* de forma simplificada.

A partir dos dados devidamente representados em vetores, uma abordagem ingênua da junção por similaridade com função cosseno é executar comparações entre os objetos com todas possibilidades possíveis, o que gera uma complexidade de $O(n^2)$. Portanto, algoritmos de junção por similaridade dependem de técnicas de filtragem para permitir eliminar um maior número de candidatos e assim reduzir o espaço de comparação. Dentre os algoritmos existentes, o estado da arte é representado pelo algoritmo L2AP [Anastasiu and Karypis 2014], que emprega uma variedade de filtros baseados em limites derivados da desigualdade de Cauchy-Schwarz.

[1] `https://huggingface.co/sentence-transformers`
[2] `https://www.sbert.net`

Datasets	Atributos	Tam. Min	Tam. Max	Tam. Méd	Registros	Duplicatas	Total
DBLP	Title, Author	8	958	125	20000	5	100000

Tabela 1. Descrição do dataset DBLP

4. Metodologia

O trabalho de pesquisa, precedido pela revisão sistemática da literatura, está sendo conduzido a partir das seguintes etapas:

- Realização de testes e experimentos em algoritmos existentes com diferentes representações de dados;
- Análise e comparação dos resultados com objetivo de identificar relações de causa e efeito nas diferentes bases de dados e possíveis melhorias;
- Desenvolvimento de um novo algoritmo de junção por similaridade a partir de dados com representação semântica. O principalmente requisito é otimizar o tempo de execução e consumo de recursos no processamento da junção por similaridade nestes espaços vetoriais que capturam a semântica, contexto e aspectos sintáticos.
- Realização de estudos comparativos detalhados, onde diversas abordagens alternativas serão executadas em condições idênticas, possibilitando refinamentos sucessivos na proposta inicial.

Os resultados serão analisados empiricamente e os experimentos divididos em duas partes: custos computacionais do algoritmo e qualidade dos resultados. Os critérios utilizados para avaliar os resultados serão: tempo de processamento, consumo de recursos e memória para avaliar os custos computacionais e medida F1 para avaliar a qualidade. Para os custos computacionais na execução do algoritmo em espaços vetoriais semânticos, serão utilizadas as bases DBLP[3] e IMDB[4] devido a maior quantidade de registros e possibilidade de geração de dados sintéticos para experimentos em grandes volumes de dados. Neste experimento, será utilizado como *baseline* a representação vetorial por medida TF-IDF. Para a qualidade dos resultados, e também custos computacionais avaliados no contexto semântico, serão utilizados os datasets do trabalho [Mudgal et al. 2018] para fins de comparação com *DeepMatcher*, *Ditto* e *Ember*.

5. Experimentos e Resultados Preliminares

Experimentos preliminares foram realizados para avaliar os custos computacionais na execução do algoritmo L2AP nos espaços vetoriais semânticos. O objetivo é responder a questão (1) da pesquisa. Para tanto, foi utilizada a base DPLP descrita na Tabela 1.

As representações vetoriais, normalizadas para uma unidade de comprimento, foram realizadas de duas formas: (1) medida TF-IDF com *tokenização q-gram(3)* e (2) representação semântica com uso do *framework sentence-transformers* e modelo pré-treinado multi-qa-MiniLM-L6-cos-v1[5] que gera vetores com 384 dimensões. Este modelo foi escolhido por apresentar maior rapidez na representação e melhores resultados na busca semântica de acordo com o site SBERT[6]. Uma característica comum nos vetores

[3]http://dblp.uni-trier.de
[4]https://www.imdb.com/interfaces
[5]https://huggingface.co/sentence-transformers/multi-qa-MiniLM-L6-cos-v1
[6]https://www.sbert.net/docs/pretrained_models.html

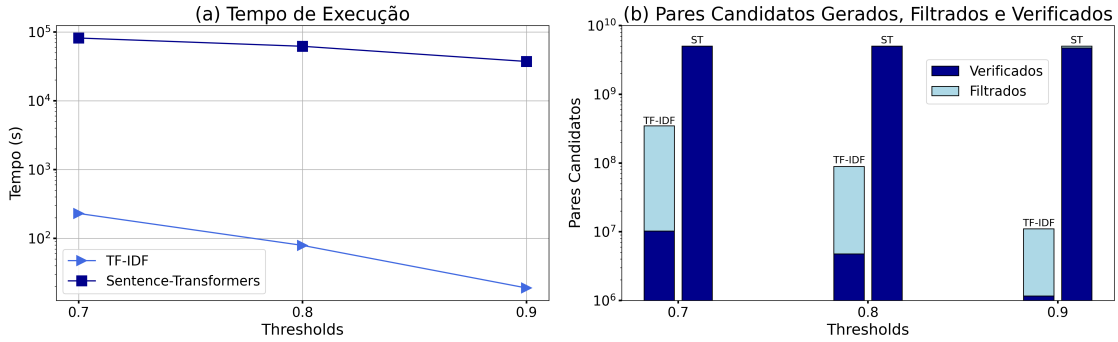

Figura 1. Execução do L2AP em vetores semânticos e com TF-IDF

semânticos são dimensões no intervalo [-1,1]. Portanto foi necessário o ajuste dos valores para o intervalo [0,1] como esperado pelo L2AP. Esta operação foi realizada através da fórmula $N = (x + 1) / 2$. O tempo necessário para gerar a representação vetorial pela medida TF-IDF e pelo *sentence-transformers* foi de 16 segundos e 59 segundos respectivamente. As execuções do algoritmo foram realizadas variando o *threshold* de 0.7 a 0.9 com incrementos de 0.1 e os tempos obtidos são a média de cinco execuções.

A Figura 1 apresenta: (a) os tempos de execução do algoritmo L2AP e (b) a quantidade de pares candidatos gerados, filtrados e verificados (b) nos vetores com representação por medida TF-IDF e com representação semântica através do *Sentence-Transformers* indicado pela sigla ST. O L2AP demonstra eficiência nos vetores gerados com medida TF-IDF. Estes vetores possuem uma grande quantidade de dimensões, porém são esparsos e com a minoria das dimensões diferente de zero. As técnicas de filtragem e poda de candidatos deste algoritmo se beneficiam destas características para eliminar um maior número de candidatos e interromper a computação de falsos-positivos e, desta forma, a execução é realizada em poucos segundos ou minutos.

Já nos espaços vetoriais semânticos, que se tratam de vetores densos e de tamanho fixo, o L2AP não apresentou eficiência. Como os vetores possuem todas as dimensões em comum, a fase de geração de candidatos acaba não filtrando candidatos, o que sobrecarrega a fase de verificação que computa a similaridade total de um grande número de falsos-positivos. O principal filtro do L2AP, baseado na norma L2, filtrou um número bastante reduzido de candidatos. Verificou-se ainda que os filtros que se baseiam no tamanho do vetor, do seu prefixo ou sufixo deste algoritmo, não filtraram pares candidatos. Desta forma, o algoritmo acaba calculando a similaridade de uma grande porção do espaço de comparação, o que aumenta substancialmente o tempo de execução.

6. Conclusões

Operações de junção por similaridade são importantes ferramentas para limpeza e integração de dados textuais. Devido ao significativo custo computacional destas operações, algoritmos que buscam reduzir estes custos são cruciais. Outro aspecto importante se refere a qualidade semântica dos resultados, pois características dos dados demanda um maior significado em linguagem natural. Neste contexto, este trabalho de pesquisa investiga a otimização de junções por similaridade em vetores semânticos. Pelos experimentos preliminares verificou-se que as características destes vetores reduzem drasticamente o desempenho do o L2AP, o melhor algoritmo conhecido para junções por

similaridade sobre vetores. Desta maneira, a continuidade dos trabalhos de pesquisa se concentrará no desenvolvimento de um novo algoritmo customizado para espaços vetoriais semânticos. Também serão consideradas nesta pesquisa técnicas correlatas como seleção de características e redução de dimensionalidade, dimensão intrínseca de dados, maldição de dimensionalidade, uso de estruturas de indexação, dentre outras.

Referências

Anastasiu, D. C. and Karypis, G. (2014). L2AP: Fast Cosine Similarity Search with Prefix L-2 Norm Bounds. In *Proceedings of the ICDE Conference*, pages 784–795.

Chaudhuri, S., Ganti, V., and Kaushik, R. (2006). A Primitive Operator for Similarity Joins in Data Cleaning. In *Proceedings of the ICDE Conference*, page 5.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186.

Johnson, J., Douze, M., and Jégou, H. (2019). Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547.

Li, Y., Li, J., Suhara, Y., Doan, A., and Tan, W. (2020). Deep Entity Matching with Pre-Trained Language Models. *Proceedings of the VLDB Endowment*, 14(1):50–60.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. In Bengio, Y. and LeCun, Y., editors, *International Conference on Learning Representations*.

Mudgal, S., Li, H., Rekatsinas, T., Doan, A., Park, Y., Krishnan, G., Deep, R., Arcaute, E., and Raghavendra, V. (2018). Deep Learning for Entity Matching: A Design Space Exploration. In *Proceedings of the SIGMOD Conference*, pages 19–34.

Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. *CoRR*, abs/1908.10084.

Ribeiro, L. A. and Härder, T. (2011). Generalizing Prefix Filtering to Improve Set Similarity Joins. *Information Systems*, 36(1):62–78.

Ribeiro-Júnior, S., Quirino, R. D., Ribeiro, L. A., and Martins, W. S. (2017). Fast Parallel Set Similarity Joins on Many-core Architectures. *Journal of Information and Data Management*, 8(3):255–270.

Salton, G., Wong, A., and Yang, C. (1975). A Vector Space Model for Automatic Indexing. *Communications of the ACM*, 18(11):613–620.

Suri, S., Ilyas, I. F., Ré, C., and Rekatsinas, T. (2021). Ember: No-Code Context Enrichment via Similarity-Based Keyless Joins. *Proceedings of the VLDB Endowment*, 15(3):699–712.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is All you Need. In *Annual Conference on Neural Information Processing Systems*, pages 5998–6008.