Projeto de Reprodução

Igor de Sousa Pereira

July 2024

1 Introdução

Para este projeto de reprodução, utilizamos o trabalho de Karapiperis intitulado *A Suite of Efficient Randomized Algorithms for Streaming Record Linkage*, que apresenta uma técnica inovadora para lidar com dados provenientes de diferentes fontes no contexto de streaming. Essa técnica, denominada Resolução de Entidades, tem como objetivo unir duas fontes de dados e identificar pares de entidades iguais, permitindo assim obter insights valiosos, melhorar a qualidade dos *datasets*, entre outros benefícios.

A principal característica do método descrito neste trabalho é a transformação de valores de string em representações numéricas, chamadas de *hash*. O conceito subjacente a essa abordagem é que, teoricamente, ao gerar valores de *hash* para dois nomes de pessoas, como "Ygor" e "Igor", devido à sua grande similaridade semântica, os valores criados serão bastante parecidos (às vezes até iguais, dependendo do método utilizado para a criação dos *hashes*). Dessa forma, esses algoritmos conseguem capturar a semelhança, facilitando a tarefa de Resolução de Entidades.

Neste projeto, buscamos verificar a consistência dos resultados obtidos pelo autor, garantindo assim que seus resultados são confiáveis.

2 Metodologia

Utilizamos o código disponível no GitHub do autor[1]. Os *datasets* mencionados no artigo estão disponíveis em:

- `https://www.ncsbe.gov/results-data/voter-registration-data`
- `https://dblp.org/xml`
- `https://dbs.uni-leipzig.de/research/projects/object_matching/benchmark_datasets_for_entity_resolution`

Devemos salientar que os dois primeiros *datasets* são muito grandes em tamanho e em quantidade de informações (*records*), sendo respectivamente 2

[1] `https://github.com/dimkar121/Approximate-Blocking/tree/master`

milhões e 16 milhões de *records*. Tentativas de executar tais *datasets* sofreram com *overflow* de memória principal, não permitindo a execução do *dataset*.

O ambiente testado foi uma máquina física contendo 16 GB de RAM com frequência de 3000 MHz, processador Intel Core i5 10400 @ 2.90GHz × 12, SSD NVME de 1 TB de armazenamento, rodando o sistema Ubuntu 22.04.4 LTS 64 bits. O ambiente foi o Conda versão 23.11.0 e Python 3.11.5. As versões das bibliotecas são as mesmas descritas no repositório. Também testamos em um ambiente Google Compute Engine (Google Colab), com 12.7 GB de RAM e 107.7 GB de armazenamento, porém sem sucesso em ler tais *datasets*.

3 Reprodução

A fim de reprodução, recuperamos o artigo e modificamos 4 parâmetros: L, L1, *offsetA* e *offsetB*. Os parâmetros L e L1 referem-se à quantidade de *hashes* gerados para cada *record* que o algoritmo lidará. Colocamos os novos valores de L e L1 para 80 e 35 (anteriormente atribuídos como 115 e 50, respectivamente), a fim de verificar como o código se comportaria tendo menos valores *hashes* gerados para cada *record*. Os valores de *offsetA* e *offsetB* foram ambos colocados para 50 (anteriormente atribuídos como 1) para forçar o código a lidar com mais valores de uma vez.

O *dataset* escolhido foi o SCD, que possui 66 mil *records*, por conseguir rodar na máquina local. O experimento foi executado 10 vezes e obtida a média dos valores retornados, como *recall* e *precision*, tempo de bloqueio e *matching* a fim de obter valores mais generalizáveis.

A partir disso, ao executarmos o código, obtivemos como média:

- *Recall*: 0.9285019637179726
- *Precision*: 0.6739888737150423

4 Conclusões

Comparando aos resultados do artigo, vimos que as métricas para esses 66 mil *records* se comportam de maneira semelhante em relação ao *recall*, também 0.92. Em relação à precisão, o valor que foi evidenciado foi aproximadamente 0.5, o que mostra uma grande melhora em nossa execução, comparada à do autor. Devemos levar em conta que o tamanho do *offset* influenciou no processamento e, consequentemente, melhorou o resultado da precisão.