Jack o Minerador: Apriori Por Partes
Machine Learning particionando dados do Censo escolar e do ENEM

Julia de Araújo Ramos[1],**Maria Clara Simões de Jesus**[1], **Jorge Lima de Oliveira Filho**[1], **Jauberth Weyll Abijaude**[1], **Álvaro Vinícius de Souza Coêlho**[1]

[1]Departamento de Engenharias e Computação, Universidade Estadual de Santa Cruz
Rodovia Jorge Amado, Km 16, Bairro Salobrinho CEP 45662-900. Ilhéus-Bahia

juramosar@gmail.com,{mcsjesus.cic,jlofilho,jauberth,degas}@uesc.br

Resumo. *Este artigo apresenta uma abordagem para o algoritmo Apriori em hardwares modestos, explorando a possibilidade de cumprir o objetivo do algoritmo com múltiplas execuções em dados particionados, seguidas de uma verificação final. Demonstra-se analiticamente, mas uma implementação sobre dados reais também é mostrada, usando dados do ENEM e do Censo Escolar.*

1. Introdução

Muitos pesquisadores de diversas áreas como pedagogia, licenciaturas, psicologia entre outras utilizam dados anuais do resultado do Exame Nacional do Ensino Médio - ENEM ou do Censo Escolar para proceder às suas investigações. Ocorre, porém, que cada pesquisa que se debruça sobre esses dados tem seu próprio objetivo, e portanto busca aspectos específicos dentro das respectivas bases de dados, tentando encontrar correlações e inferências úteis. Como são múltiplos os pesquisadores, com múltiplos interesses, também múltiplas serão as necessidades de processamento destes dados. Neste contexto, um método que faça um processamento o mais geral possível, encontrando o máximo possível de correlações, e ainda partindo de fontes diversas, se torna particularmente interessante para municiar os pesquisadores de todo o conhecimento necessário sobre as correlações nos dados, independente de seu objetivo específico.

O Algoritmo Apriori [Agrawal et al. 1993] tem se mostrado particularmente útil neste contexto, pois permite encontrar quaisquer correlações entre dados tanto de um mesmo domínio quanto de domínios distintos, sendo flexível o suficiente para ser executado tanto sobre bancos de dados relacionais como dados desestruturados da Web. Todavia, a complexidade espacial do Aprori [Purdom et al. 2004] pode fazer com que sua execução seja proibitiva em alguns hardwares mais modestos, e isso é particularmente verdadeiro quando se consideram os dados consolidados da junção entre o ENEM e o Censo Escolar.

Neste trabalho, implementamos e testamos uma abordagem alternativa que executa o Apriori a partir do particionamento dos dados, e assim conseguimos processar a junção dos dados do Censo Escolar e do ENEM utilizando um equipamento relativamente modesto, se comparado à capacidade de grandes *data centers*.

2. Contexto da Pesquisa

As possibilidades de emprego de técnicas de *machine learning* são diversas, com contextos que vão desde apoio à tomada de decisões em gestão estratégica ao suporte a políticas públicas aplicadas a setores fundamentais, como saúde e educação. Como exemplo, há estudos sobre dados do Censo escolar, como a utilização destes na busca do entendimento e

correlação de fatores que atuam positiva ou negativamente no resultado final de estudantes que buscam concluir o ensino fundamental[Ferreira 2015] ou verificação da eficácia de estratégias para inclusão de estudantes com necessidades especiais [Vizzotto 2020].

Com dados do ENEM, levantamentos permitiram avaliar o desempenho de estudantes com necessidades especiais[Silva and Meletti 2012] ou proposição, adequação e avaliação de políticas e práticas pedagógicas [Hilário 2008], entre outros mais generalizados, com objetivos os mais variados, disponíveis em grande quantidade de fontes bibliográficas[Lima et al. 2019].

Já as abordagens que consideram dados tanto do ENEM quanto do Censo Escolar, abrem possibilidades mais promissoras como um diagnóstico da eficiência e eficácia de políticas públicas, considerando aspectos de gestão e de financiamento, com foco em programas de educação profissionalizante[Machado and Ferreira 2019].

É importante notar que todos estes trabalhos utilizam conhecimentos extraídos das bases de dados do Censo Escolar ou do Enem (ou Ambos) com objetivos bem específicos. Este trabalho parte da premissa de que é interessante aplicar um algoritmo de *machine learning* que encontre todo tipo de conhecimento, indiscriminadamente, de forma que este conhecimento possa ser utilizado para qualquer fim, por mais específico ou generalizado que seja.

2.1. O Apriori

Didaticamente, a melhor forma de conhecer o Apriori [Agrawal et al. 1993] é imaginar sua aplicação a dados de uma rede de supermercados, em que cada carrinho possui um conjunto de **Itens**. Do ponto de vista deste trabalho, o objetivo é encontrar conjuntos de Itens que ocorram juntos em uma quantidade relevante de vezes.

2.1.1. Definições

Para compreender o algoritmo, e ainda usando o exemplo didático do supermercado, faz-se necessário definir os elementos que guiam sua execução.

- Item: é o elemento atômico, a partir do qual se deseja encontrar as regras de associação.
- Conjunto de Itens: é um conjunto (não importam quantidades) de diferentes Itens num carrinho, mas também serão produzidos durante a execução.
- Transação e Conjunto de transações: uma Transação é o Conjunto de Itens que está em cada carrinho que foi registrado. E o Conjunto de Transações é o conjunto de todos carrinhos registrados.
- Suporte mínimo e Itens Frequentes: apenas Itens que sejam referidos uma certa quantidade de vezes interessam. Este limiar é chamado de **Suporte Mínimo** S e é parâmetro do algoritmo. Além do suporte mínimo, cada conjunto de itens possui seu próprio suporte, referente à quantidade de vezes que ele ocorre. Os conjuntos cujo suporte supere o limiar S são referidos como **Itens Frequentes**.

  Por exemplo, o suporte de { a, b, c } é igual a 2 (ou 50%) num conjunto de Transações formado por$\{a, b, c, d\}\{a, b, d, e\}\{a, b, c, e\}\{b, c, d, e\}$.

2.1.2. Funcionamento

Para entender o funcionamento do Apriori, é necessário observar a seguinte propriedade dos conjuntos frequentes: seja A um conjunto frequente, então todo subconjunto de A também será frequente, já que qualquer combinação de itens de A ocorre pelo menos em todas as transações junto ao próprio conjunto A.

Isto define a estratégia para gerar os conjuntos frequentes: a geração de **conjuntos de conjuntos** com tamanhos (quantidade de elementos) sucessivamente maiores, começando por conjuntos unitários.

Usa-se os conjuntos frequentes de tamanho $L-1$ para, após combiná-los entre si, gerar **candidatos** a conjuntos de tamanho L. O conjunto de candidatos de tamanho L é chamado C_L. Depois de gerado C_L, verifica-se, no conjunto de transações, o suporte efetivo de cada candidato e se procede à **poda** dos que não alcançam suporte mínimo, obtendo-se assim os conjuntos frequentes de tamanho L (F_L). O procedimento se repete até que a quantidade de conjuntos em F_L seja ≤ 1. O funcionamento do algoritmo está descrito abaixo.

Algorithm 1 Algoritmo Apriori

procedure APRIORI(T) ▷ T é o conjunto de transações
  $C_1 \leftarrow gera1Candidatos(T)$ ▷ Os candidatos de tamanho 1 são buscados em T
  $F_1 \leftarrow podaCandidatos(C_1, T)$ ▷ Gera-se os frequentes de tamanho 1
  $i \leftarrow 2$
  while $|F_{i-1}| \geq 1$ **do** ▷ Mais do que 1 conjunto de tamanho $i-1$
    $C_i \leftarrow geraCandidatos(F_{i-1}, T)$
    $F_i \leftarrow podaCandidatos(C_i, T)$
    $i++$
  end while
  return $\bigcup_{k=1}^{i} F_i, 1 \leq k < i$
end procedure

2.2. Considerações de Complexidade

Para efeito de análise de complexidade computacional vamos considerar uma distribuição equiprovável dos itens, desprezando o suporte (que, em casos práticos costuma ser muito baixo). Vamos supor que há um total de m itens diferentes. Então temos que, em cada etapa l (onde se está gerando os conjuntos de tamanho L), temos uma combinação (binomial) de $\binom{m}{l}$ conjuntos gerados, o que dá um total de

$$\frac{m!}{l! \cdot (m-l)!} \tag{1}$$

[Purdom et al. 2004]

candidatos. Assim, considerando que houve um total de S níveis de execução (gerando conjuntos frequentes de tamanho 1 a S), o esforço para gerar todos os conjuntos frequentes impõe uma complexidade espacial total *potencial*[1] dada por

[1]Isto é afetado pela distribuição dos dados e também pelo suporte mínimo

$$\sum_{l=1}^{S} \frac{m!}{l! \cdot (m-l)!} \tag{2}$$

É importante observar que se trata de uma distribuição binomial. Logo, a quantidade de conjuntos vai crescendo mas necessariamente há um nível $l^{critico}$ a partir do qual a quantidade de conjuntos gerados decresce, com valores progressivamente menores, de maneira simétrica. Além disso, em casos reais é esperado que m seja muito maior do que qualquer um dos l, pois nenhum conjunto frequente deverá incluir todos os itens disponíveis.

Para ilustrar, supondo que m seja igual a 140. Então, numa execução hipotética que alcance nível $l = 12$ temos, pela Equação 1, um total de

$$\frac{140!}{12! \cdot 128!}$$

conjuntos frequentes candidatos, apenas considerando o nível ($l = 12$). O valor aproximado chega a, potencialmente, $7,3 \cdot 10^{16}$ conjuntos. Considerando o somatório descrito na Equação 2, e variando os níveis de 1 a 12 numa distribuição binomial, encontraremos um total aproximado de $8,03 \cdot 10^{16}$. Valor da mesma ordem de grandeza do nível $l = 12$.

A avaliação acima considera o custo *potencial* de memória, com suporte igual (ou muito próximo a) 0. Isto pode ser reduzido, por exemplo, se considerarmos um suporte mínimo muito alto [2]. Todavia, na maior parte das aplicações o suporte considerado é muito baixo [Koh and Rountree 2005], o que torna razoável considerar que os custos de geração de conjuntos candidatos se aproximem da análise potencial apresentada acima, o que torna proibitiva a implementação prática do Apriori senão em hardware com alta capacidade.

3. Aplicações do Apriori

Desde sua apresentação, o Apriori mostrou-se versátil. Por exemplo, é possível se adaptar o Apriori para trabalhar em classificação de dados, a partir de uma etapa de pré-processamento que mapeia cada um dos possíveis valores de atributos em "Itens"[Jovanoski and Lavrač 2001]. Neste contexto de aplicações em classificação, entre muitas, destacam-se trabalhos que implementam o Apriori a partir de dados de educação [Jha and Ragha 2013]. Todavia, a quantidade de dados disponíveis a respeito de censos e avaliações na Educação, mais a complexidade inerente ao próprio algoritmo Apriori desencoraja sua implementação em máquinas com limitações de quantidade de memória (veja na Seção 2.2). Mais especificamente, tomando a junção dos dados do Enem com os do Censo Escolar, temos um total de 119.295 linhas de dados, cada uma delas com 69 atributos. Considerando-se os possíveis valores de cada atributo, e mapeando-os em "Itens", chegamos a um total de $m = 140$ Itens.

Observe que, aplicando $m = 140$ e $l = 12$ à Equação 1, encontramos que o potencial de conjuntos frequentes gerados chega a $7,3 \cdot 10^{16}$ conjuntos candidatos apenas neste nível. Aplicações assim direcionam as pesquisas na área para a implementação deste algoritmo em grandes estruturas computacionais, que disponham de alta capacidade de armazenamento e paralelismo, além de *throughput* suficientemente alto para eventuais necessidades de *swap* durante o processamento[Yahya et al. 2012].

[2] Se o suporte requerido for 100%, provavelmente nenhum conjunto será gerado

Este trabalho apresenta uma implementação do Apriori que particiona os dados de maneira que o esforço computacional em cada partição esteja ao alcance de máquinas mais modestas. Isto torna viável sua execução mesmo numa máquina cujas características mencionadas acima sejam limitadas.

4. Solução Proposta

Para implementar a solução que trazemos neste trabalho, precisaremos começar pelo que chamaremos de **Teorema do Particionamento do Apriori**.

4.1. Teorema do Particionamento do Apriori

Enunciado: independente do número ou do tamanho de cada partição, não é possível que um conjunto seja frequente global se ele não for frequente **em pelo menos uma partição**.

4.2. Prova do Teorema

Vamos demonstrar por redução ao absurdo. Suponha um certo suporte mínimo (tomado em percentual) $S, 0 \leq S \leq 1$. Seja P o tamanho total do conjunto de treinamento. Logo, um conjunto frequente ocorre pelo menos $P \cdot S$ vezes. E sejam $p_1, p_2, ...p_n$ os tamanhos (em quantidade de transações) das n partições em que P será dividido logo

$$P \cdot S = \sum_{i=1}^{n} p_i \cdot S \tag{3}$$

Suponha que G seja um conjunto frequente (ocorre pelo menos $P \cdot S$ vezes). Então a quantidade de vezes que G ocorre, S_g tem a seguinte propriedade:

$$S_g \geq P \cdot S \tag{4}$$

Mas G ocorre uma certa quantidade de vezes em cada partição; então, sendo $g_1, g_2, ..., g_n$ as n quantidades que G ocorre em cada partição, temos que a soma das ocorrências em cada partição tem que ser igual à soma total de ocorrências de G. Mas, por hipótese, G não é frequente em nenhuma partição. Logo, temos que $\forall i, g_i < S \cdot p_i$ (g_i nunca alcança o suporte mínimo na partição p_i). E, portanto,

$$S_g = \sum_{i=1}^{n} g_i < \sum_{i=1}^{n} p_i \cdot S \tag{5}$$

Então, da Equação 4 temos que $S_g \geq P \cdot S$ e, da Equação 5 temos que $S_g < P \cdot S$. Absurdo, Q.E.D.

4.3. Descrição da Solução

A solução apresentada neste trabalho consiste, então, em particionar o conjunto original de dados em partes pequenas o suficiente para serem executados com limitação de memória. Sobre cada partição se executa completamente o algoritmo Apriori, na busca por conjuntos frequentes locais a cada uma delas. Então, sejam $p_1, p_2, ...p_n$ as n partições, e sejam $F_1, F_2, ...F_n$ os n conjuntos de conjuntos frequentes que foram gerados em cada partição. Agora geramos de CG, Candidatos Globais, a partir da união de todos os conjuntos frequentes de cada partição.

$$CG = \bigcup_{i=1}^{n} F_i$$

.

Note que, conforme demonstrado na Seção 4.2, **não há falsos negativos** (conjuntos que sejam globalmente frequentes mas não tenham aparecido em algum L_i, $(1 \leq i \leq n)$, embora **possam existir falsos positivos** (conjuntos que aparecem como frequente em pelo menos uma partição, mas que não são globalmente frequente). Por este motivo, o conjunto CG contem todos os frequentes globais, embora alguns que não o são.

Executa-se um segundo percurso no conjunto completo de transações, verificando-se o suporte global de cada um dos conjuntos $f \in CG$, determinando os conjuntos Frequentes Globais (FG) pela poda dos que não alcançam o suporte mínimo S considerado. Assim, os conjuntos frequentes encontrados são dados por: $FG = \forall f_i | f_i \in CG \wedge suporte(F_i) \geq S$. É importante observar que esta segunda etapa não requer nenhuma combinação de dados, e é basicamente o percurso linear dos dados, tendo, portanto, complexidade na ordem de $O(T)$, T o número de transações.

5. Metodologia

A metodologia para desenvolvimento e avaliação deste trabalho ficou suportada em três pilares, que, na verdade, respondem às três perguntas que estão implicitamente postas aqui: demonstrar sua *Viabilidade*, com um método que encontre partições pequenas o suficiente para serem executáveis mas não pequenas demais para reduzir o custo de processamento; *Eficácia* pois, apesar do Teorema apresentado (Seção 4.2) precisamos atestar que nossa implementação está correta; e finalmente um *Experimento Completo* utilizando os dados do ENEM e do Censo Escolar.

5.1. Viabilidade

A ideia geral é fazer divisões sucessivas do conjunto de dados em partições cada vez menores até encontrar o tamanho ideal: não tão grande para que sua execução caiba na memória RAM disponível, não tão pequena para não gerar execuções desnecessárias do Apriori.

Importante começar no pré-processamento, organizando os dados aleatoriamente pois tipicamente (e com os dados do ENEM e Censo Escolar não é diferente) o formato original tem algum ordenamento e isto gera viés: ordem geográfica ou temporal por exemplo. A partir daí, implementamos um método bisseção, particionando o conjunto original em duas metades, e tentando a execução do algoritmo numa delas. Pode-se conseguir sua execução com êxito ou não (provavelmente não, nos primeiros particionamentos) e, doravante, a ideia é muito simples: seguir particionando em $1/4, 1/8, ...$ até se encontrar uma partição tal que possa ser executada.

Naturalmente, caso a execução em alguma partição falhe por falta de memória, esta partição especificamente é re-dividida e o processamento segue normalmente.

5.2. Eficácia

Apesar do que está demonstrado no Teorema (Seção 4), a tarefa específica de junção dos frequentes locais e sua verificação final (descritos na Seção 4.3) precisa de uma verificação independente, pois a programação destas etapas poderia conter erros.

Para isso, uma mesma implementação do Apriori deverá ser utilizada. Serão feitas execuções diferentes sobre o mesmo conjunto de dados: uma com o conjunto completo,

outra com o método de particionamento que foi implementado. Os resultados obtidos são comparados e, assim, poderemos atestar que a implementação adotada não gera erros de processamento.

5.3. Experimento Completo

Havendo evidência de que o método funciona corretamente, é possível aplicá-lo sobre os dados do Enem e do Censo Escolar, como uma prova de conceito da implementação proposta, descrevendo: configuração da máquina, detalhes de software, número de partições e resultados encontrados, de forma a tornar o experimento reprodutível.

6. Resultados

Neste trabalho, foi realizado um experimento com a utilização da biblioteca *mlxtend* [Raschka 2018], que traz uma versão otimizada do Apriori.

A máquina utilizada no experimento foi tem processador **Intel Core i7-10750H**, com 16 GB de memória RAM, com armazenamento em *SSD NVMe* de com capacidade de 512 GB, rodando o MS Windows 11 Pro™. Os métodos de particionamento dos dados, união dos frequentes locais, verificação dos frequentes globais, foram implementados da forma que estão descritos na Seção 4, utilizando a linguagem Python versão 3.9, acrescida, além do *mlxtend*, da biblioteca *Panda*[Elsner 2023] para simplificar os processos de manipulação de dados e avaliação de resultados.

6.1. O experimento

Após pre-processamento, unindo dados do ENEM com dados do Censo escolar, e organizando-os em ordem aleatória, obteve-se uma base com dados socio-econômicos de estudantes, infra-estrutura pedagógica e física das escolas, além de dados geográficos e sociais. Estes dados produziram uma tabela com 119.295 linhas, cada uma delas com 69 colunas. Definindo-se um item para cada valor de atributo, chegou-se a um total de 140 itens diferentes.

Apesar de os números parecerem modestos, a quantidade de conjuntos frequentes mostrou-se surpreendentemente alta nas primeiras tentativas de execução, devido ao alto custo de memória necessário, conforme discutido na Seção 2.2.

Uma das maneiras de se reduzir a quantidade de candidatos a conjuntos frequentes é elevar o suporte mínimo considerado. Por exemplo, se considerarmos um suporte de 100%, provavelmente nenhum conjunto candidato será gerado. Por outro lado, um suporte perto de 0% gera uma quantidade fatorial de combinações (veja na Seção 2.2). Dessa maneira, apenas elevando-se substancialmente o suporte mínimo foi possível processar em uma única execução todos os dados considerando-se as 69 colunas.

Esta relação entre o suporte e a quantidade de recursos necessários é melhor apreciada nas descrições dos experimentos para a prova de conceito (Seção 6.1.2).

6.1.1. Experimento de Eficácia

Para verificar a eficácia, conforme discutido metodologicamente (Seção 5.2), era necessário configurar uma execução que fosse viável executar numa única execução (sem particionamento). Reduziu-se quantidade de colunas para 40 e, consequentemente, a quantidade de itens disponíveis caiu para 80. Manteve-se, porém, todas as 119.295 linhas.

Posta esta configuração, executamos o Apriori sobre todo o conjunto de dados e, em seguida, executamos o método com particionamento, usando um total de 5 partições, de acordo com o processo descrito na Seção 4.

Comparados os conjuntos frequentes produzidos nas duas execuções, e eles eram idênticos, encontrou-se a evidência de que a implementação do método está correta.

6.1.2. Prova do conceito

Depois de comprovada a correção do método proposto, deu-se a execução da prova de conceito, utilizando-se a totalidade das colunas e linhas a serem processadas.

Diferentes valores de suporte foram verificados, para saber em quantas partições seria necessário se dividir o arquivo original. Note que, conforme discutido na Seção 6.1, valores mais altos de suporte permitem partições maiores. Desta forma, diferentes valores de suporte permitiram diferentes valores de particionamento. A Tabela 1 mostra a quantidade e o tamanho máximo de cada partição para diferentes valores de suporte que consideramos.

Número de Partições	Linhas por Partição (aprox.)	Sup. Mínimo Alcançado
1	119.295	0.70
2	59.647	0.70
4	29.823	0.60
8	14.911	0.60
16	7.455	0.50
32	3.727	0.45

Tabela 1. Relação entre o número de partições e o suporte mínimo alcançado.

Evidentemente, um suporte menor que 0.45 implicaria mais partições para a execução do método. Mas para demonstração do conceito julgou-se desnecessário. Especificamente no cenário com suporte igual a $0,45$, e com 32 partições com (aproximadamente) 3.727 linhas em cada, encontramos, após a união de todos os candidatos locais, um total de $1.952.352$ candidatos globais de diferentes tamanhos. A Tabela 2 mostra a distribuição desses candidatos globais de acordo com seus respectivos tamanhos.

Após o processamento dos candidatos globais para exclusão dos falsos positivos, obtivemos um total $1.866.592$ conjuntos frequentes de tamanhos diferentes, distribuídos conforme mostrado na Tabela 3.

7. Trabalhos Correlatos

A ideia de particionar os dados de treinamento, que costumam ser volumosos, para reduzir o esforço computacional na execução do processo de *machine learning* é um caminho natural de processamento e, com o algoritmo Apriori, não é diferente. Neste sentido, entre as soluções propostas na literatura, destaca-se a execução do Apriori sobre subconjuuntos dos dados, com aplicação de métodos de otimização baseado em algoritmos genéticos sobre cada subconjunto[Ghosh et al. 2010]. Esta abordagem consegue melhorar o desempenho do algoritmo em sua complexidade temporal, mas o ganho espacial, ao contrário do método mostrado neste trabalho, acaba sendo muito pequeno pois a implementação do método de otimização baseado em algoritmo genético impõe que os dados estejam sempre disponíveis na memória.

Tamanho	Quantidade
1	999
2	11.158
3	62.249
4	198.853
5	394.567
6	508.555
7	432.838
8	240.823
9	83.958
10	16.711
11	1.585
12	56

Tabela 2. Candidatos Globais

Tamanho	Quantidade
1	764
2	10.056
3	58.495
4	189.291
5	377.095
6	486.629
7	414.903
8	231.012
9	80.639
10	16.107
11	1.545
12	56

Tabela 3. Frequentes Globais

Dentre trabalhos similares ao que está neste artigo, destacam-se abordagens aplicáveis a métodos paralelos, de forma que os dados sejam processados em partições locais e, em seguida, juntados para uma avaliação final a fim de se selecionar os conjuntos frequentes em todas as partições[Ye and Chiang 2006]. Mas o foco acaba sendo, também, o canho temporal, a partir do processamento paralelo.

8. Conclusões e Trabalhos Futuros

Dentro da infinidade de métodos e abordagens aplicadas em *Machine Learning*, ao longo de décadas, o algoritmo Apriori se posa, ao mesmo tempo que como um clássico, também como uma das referências mais revisitadas em trabalhos acadêmicos e na indústria de software ainda hoje. Isto ocorre porque se trata de um método bastante flexível, que pode ser adaptado e executado em diferentes contextos: útil nos domínios dos bancos de dados relacionais e também no processamento de dados não estruturados da web; útil na extração de conhecimento sobre dados educacionais ou de saúde e também na correlação textual em documentos.

Neste trabalho enfrentamos o problema do gerenciamento da memória, que impõe restrições ao uso do Apriori, e mostramos que com o artifício do particionamento se torna possível uma execução alternativa, produzindo rigorosamente o mesmo resultado. No estudo que fizemos, os resultados apontam para milhões de conjuntos que correlacionam dados do ENEM e do Censo Escolar, e que podem ser aproveitados por diversos pesquisadores de diversas áreas do conhecimento, sob diferentes contextos. Além disso, com a estratégia do particionamento, este processamento não requer uma capacidade computacional proibitiva para ser exequível, podendo ser executado em máquinas com quantidades modestas de memória disponível.

Importante observar que a metodologia apresentada neste artigo é, do ponto de vista teorético, uma escolha: ganhamos em complexidade espacial mas, conforme sabemos da Teoria da Computação, pagamos um pedágio no tempo para isto (as múltiplas execuções). Nossos experimentos, porém, sugerem que o compromisso é satisfatório.

Adicionalmente, é possível se usar esta abordagem de particionamento para criar uma execução paralela do Apriori, de forma que cada partição seja executada em um hardware específico e dedicado, e assim o tempo necessário para completar toda a execução se torna muito menor.

Referências

Agrawal, R., Imieliński, T., and Swami, A. (1993). Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 207–216.

Elsner, J. (2023). Taming the panda with python: A powerful duo for seamless robotics programming and integration. *SoftwareX*, 24:101532.

Ferreira, G. (2015). Investigação acerca dos fatores determinantes para a conclusão do ensino fundamental utilizando mineração de dados educacionais no censo escolar da educação básica do inep 2014. In *Anais dos Workshops do Congresso Brasileiro de Informática na Educação*, volume 4, page 1034.

Ghosh, S., Biswas, S., Sarkar, D., and Sarkar, P. P. (2010). Mining frequent itemsets using genetic algorithm. *arXiv preprint arXiv:1011.0328*.

Hilário, R. A. (2008). O enem como indutor de políticas públicas para melhoria da qualidade do ensino médio. *Cadernos de Pós-graduação*, pages 95–108.

Jha, J. and Ragha, L. (2013). Educational data mining using improved apriori algorithm. *International Journal of Information and Computation Technology*, 3(5):411–418.

Jovanoski, V. and Lavrač, N. (2001). Classification rule learning with apriori-c. In *Portuguese conference on artificial intelligence*, pages 44–51. Springer.

Koh, Y. S. and Rountree, N. (2005). Finding sporadic rules using apriori-inverse. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 97–106. Springer.

Lima, P. d. S. N., Ambrósio, A. P. L., Ferreira, D. J., and Brancher, J. D. (2019). Análise de dados do enade e enem: uma revisão sistemática da literatura. *Avaliação: Revista da Avaliação da Educação Superior (Campinas)*, 24:89–107.

Machado, C. T. and Ferreira, L. S. (2019). Políticas de gestão e financiamento da educação profissional e tecnológica no brasil: uma análise das matrículas no censo escolar/inep e a sua relação com os programas brasil profissionalizado e pronatec. *Plurais-Revista Multidisciplinar*, 4(3):93–113.

Purdom, P. W., Van Gucht, D., and Groth, D. P. (2004). Average-case performance of the apriori algorithm. *SIAM Journal on Computing*, 33(5):1223–1260.

Raschka, S. (2018). Mlxtend: Providing machine learning and data science utilities and extensions to python's scientific computing stack. *Journal of open source software*, 3(24):638.

Silva, M. C. V. and Meletti, S. M. F. (2012). Avaliação em larga escala de alunos com necessidades educacionais especiais no município de londrina (pr). *Revista Educação Especial*, 25(44):417–434.

Vizzotto, P. A. (2020). Inclusão na educação básica brasileira: Análise do censo escolar por meio dos microdados do inep. *Ensaios Pedagógicos*, 4(1):102–112.

Yahya, O., Hegazy, O., and Ezat, E. (2012). An efficient implementation of apriori algorithm based on hadoop-mapreduce model. *International Journal of Reviews in Computing*, 12.

Ye, Y. and Chiang, C.-C. (2006). A parallel apriori algorithm for frequent itemsets mining. In *Fourth International Conference on Software Engineering Research, Management and Applications (SERA'06)*, pages 87–94. IEEE.