UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE GESTÃO E NEGÓCIOS
GRADUAÇÃO EM GESTÃO DA INFORMAÇÃO

ADRYELL ALEXANDRE CAMPOS MEDEIROS

Comparativo entre o algoritmo Apriori e FP-Growth na detecção de regras de associação para análise de sintomas respiratórios

ORIENTADOR: PROF. DR. JOSÉ EDUARDO FERREIRA LOPES

UBERLÂNDIA – MG
2026

ADRYELL ALEXANDRE CAMPOS MEDEIROS

Comparativo entre o algoritmo Apriori e FP-Growth na detecção de regras de associação para análise de sintomas respiratórios

Monografia apresentada ao Curso de Graduação em Gestão da Informação, da Universidade Federal de Uberlândia, como exigência parcial para a obtenção do título de Bacharel.

Orientador: Prof. Dr. José Eduardo Ferreira Lopes

UBERLÂNDIA – MG

2026

RESUMO

Objetivou-se com este relato tecnológico aplicar os algoritmos de regras de associação Apriori e FP-Growth aos dados de Síndrome Respiratória Aguda Grave (SRAG) do Ministério da Saúde, para identificar associações entre sintomas respiratórios. A situação-problema encontrada foi a disponibilidade dos dados passíveis de estudo que devido sua alta volumetria de dados traz uma dificuldade para extrair e comparar regras de associação entre os sintomas, que assim podem contribuir para o entendimento entre a relação dos sintomas respiratórios. Como solução do problema foi desenvolvido um algoritmo em PySpark e pandas (ferramentas do ecossistema Python) para analisar as regras de associação. Como resultados alcançados, destacam-se as regras de associação identificadas entre desconforto respiratório e saturação com registro de O_2 inferior ao limiar da ficha, com destaque para a regra que relaciona desconforto respiratório e saturação, com suporte de 0,4150, confiança de 0,7194 e *lift* de 1,2742, além do comparativo de desempenho entre Apriori e *FP-Growth* na base analisada.

Palavras-chave: Regras de Associação; Mineração de Dados; Apriori; FP-Growth; Databricks; Python; Aprendizado de Máquina.

ABSTRACT

The objective of this technological report was to apply the Apriori and FP-Growth association rule algorithms to the Ministry of Health's Severe Acute Respiratory Syndrome (SARS) data in order to identify associations between respiratory symptoms. The problem situation encountered was the availability of data suitable for study which, due to its high volume, makes it difficult to extract and compare association rules among symptoms that could contribute to understanding the relationship between respiratory symptoms. As a solution to the problem, an algorithm was developed using PySpark and pandas (Python ecosystem tools) to analyze the association rules. Key results achieved include the association rules identified between respiratory distress and oxygen saturation recorded below the form threshold—highlighting the rule connecting respiratory distress and saturation with a support of 0.4150, confidence of 0.7194, and lift of 1.2742—as well as a performance comparison between Apriori and FP-Growth on the analyzed dataset.

Keywords: Association Rules; Data Mining; Apriori; FP-Growth; Databricks; Python; Machine Learning.

SUMÁRIO

1 INTRODUÇÃO

Os dados armazenam informações úteis que podem nos auxiliar a mapear tendências, regras e variações, por meio de análises, esses dados podem apoiar decisões em diferentes áreas, como varejo, negócios, esportes e saúde. Segundo Thacharodi et al. (2024), a revolução tecnológica traz contribuições significativas para o setor da saúde, como na análise e aplicação dos dados clínicos, e isto influencia de maneira positiva nos tratamentos e nos diagnósticos.

A mineração de dados integra essa transformação tecnológica e constitui uma das etapas do processo de descoberta de conhecimento em bases de dados (KDD, do inglês *Knowledge Discovery in Databases*) A mineração de dados é um campo interdisciplinar que reúne conhecimentos de estatística, aprendizado de máquina e banco de dados, com presença forte na atualidade devido à grande volume de dados que são gerados (*Big Data*), e o valor não está na quantidade alta de dados, mas sim em encontrar os dados que trazem implicações positivas para análise, sendo isto a mineração de dados.

As regras de associação constituem uma técnica de mineração de dados descritiva de aprendizado não supervisionado, na qual o objetivo é descobrir relacionamentos entre atributos, padrões ocultos e ocorrências simultâneas entre itens. Busca descobrir quais atributos são antecedentes de atributos consequentes, ou seja, se um atributo implica em outro.

Dados públicos na área de saúde são disponibilizados pelo governo, assim, objetivou-se com este relato tecnológico aplicar os algoritmos de regras de associação Apriori e *FP-Growth* aos dados de Síndrome Respiratória Aguda Grave (SRAG) do Ministério da Saúde, para identificar associações entre sintomas respiratórios e diagnósticos.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Síndrome Respiratória Aguda Grave (SRAG)

A Síndrome Respiratória Aguda Grave não se trata de uma doença isolada, mas sim de uma condição em que há um agravamento decorrente de infecções respiratórias prévias que o paciente possa estar acometido, que compõem diretrizes do Ministério da

Saúde do Brasil e da Organização Mundial da Saúde (OMS). Essa condição caracteriza-se pelo comprometimento grave do sistema respiratório.

Segundo Araujo et al. (2020), a Síndrome Respiratória Aguda Grave (SRAG) é uma síndrome respiratória viral infecciosa decorrente de um agravamento no trato respiratório, que pode estar associada a diferentes agentes etiológicos.

No Brasil, a vigilância da SRAG é operacionalizada pelo Ministério da Saúde por meio do sistema SIVEP-Gripe, que registra os casos hospitalizados segundo o documento 'Orientações para profissionais de saúde – Síndrome Gripal (SG) e Síndrome Respiratória Aguda Grave (SRAG)' disponibilizado pelo Ministério da Saúde, definição de caso que combina síndrome gripal (febre, tosse ou dor de garganta) com dispneia, saturação de oxigênio inferior a 95% ou desconforto respiratório. A notificação é compulsória e os registros incluem dados demográficos, sintomas, comorbidades, evolução e classificação final do caso, o que torna a base especialmente adequada a estudos de mineração de dados.

2.2 Mineração de Dados (Data Mining) e o processo de KDD

A mineração de dados é um processo ou tarefa dentro do KDD (*Knowledge Discovery in Databases*), sendo uma etapa na descoberta de informações em bases de dados. Dessa forma, a mineração de dados consiste em entender e encontrar padrões através de análises estatísticas como a regra de associação, ferramentas de visualização de dados como o Power BI ou Tableau, e técnicas de inteligência artificial como *machine learning* e *deep learning*.

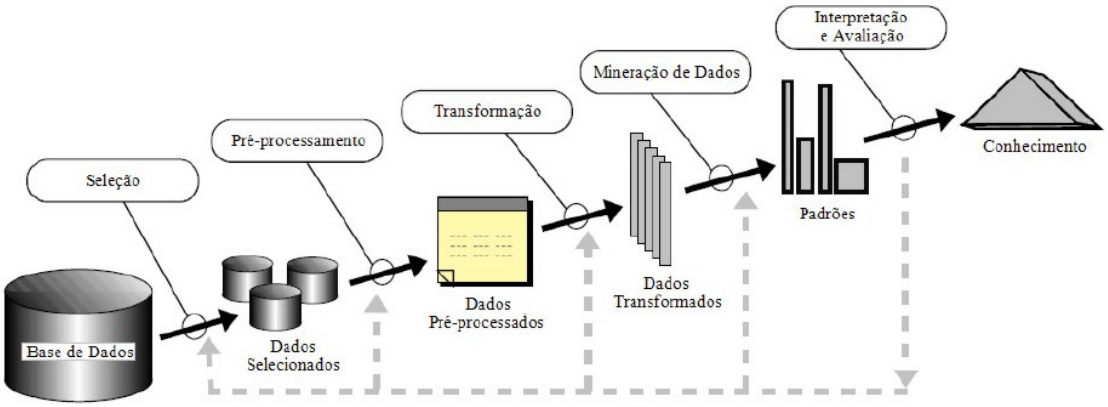
A descoberta de conhecimento em bases de dados (KDD) é um conjunto de etapas que objetiva extrair informações em bases de dados, que possam ser utilizadas para estudos posteriores e tomadas de decisões. As etapas que compõem o KDD são seleção, pré-processamento, transformação, mineração de dados e avaliação.

1. Seleção: Seleção da base de dados a ser trabalhada.
2. Pré-processamento: Tratamento dos dados que estão inconsistentes, faltantes ou fora de ordem.
3. Transformação: Transforma os dados no tipo, escala e formatos necessários para a mineração de dados.
4. Mineração de dados: Processo em que são realizadas análises e estudos através de métodos que visam extrair padrões e informações dos dados.

5. Avaliação: Análise dos resultados que foram encontrados durante a mineração de dados, e estruturação para consolidar as informações e padrões encontrados.

A figura a seguir ilustra o processo de KDD em ordem.

Figura 1 – Fases do processo de KDD



Fonte: Fayyad et al. (1996).

Segundo Mariano (2011), os termos de mineração de dados e KDD são comumente tratados como sinônimos, devido à mineração de dados ser um processo essencial no KDD. Além disso, a mineração de dados não se resume apenas a uma única técnica que resolva os problemas a serem solucionados, as técnicas de mineração de dados com maior utilização são: associação, classificação, agrupamento e regressão, a escolha da técnica deve considerar suas vantagens e limitações em relação ao problema, à análise pretendida e aos objetivos do estudo.

2.3 Regras de Associação e Métricas Estatísticas

As regras de associação é uma das técnicas de ampla utilização na mineração de dados, proposta por Agrawal et al. (1993), como na análise de cesta de supermercados, de fatores climáticos, na ocorrência de sintomas concomitantes como a problemática do presente estudo, entre outros contextos. Para esta técnica, se faz necessário que os dados da base sejam organizados como um conjunto de transações, onde essas transações sejam compostas por um ou mais itens. Assim, é possível analisar se há uma associação entre a

ocorrência dos itens que estão presentes nas transações, desta forma encontram-se implicações da presença de um ou mais itens em outros.

A seguir, serão destacados alguns pontos que contribuem para uma melhor compreensão sobre regras de associação, e os seus princípios para a determinação de associações sobre itens:

- Uma base de dados pode ser tratada com um conjunto de transações, em que cada é contém um conjunto de itens, ou seja, seja I um conjunto de k itens onde I = $\{i_1, ..., i_k\}$, e T um conjunto de j transações onde T = $\{t_1, ..., t_j\}$. Assim sendo, cada T contém um I.
- Um *itemset* é uma coleção de um ou mais itens que ocorrem em uma mesma transação, portanto seja um *itemset* um conjunto IS, então um IS está contido de I.

Algumas métricas são estipuladas ao executar a técnica de regra de associação, essas métricas têm como função estabelecer critérios de avaliação para as regras geradas, além de contribuir para que não haja um acaso das regras geradas.

As métricas utilizadas nesta técnica são o suporte, a confiança e o *lift*, onde cada uma contribui de uma maneira para os cálculos estatísticos e determinísticos das regras.

Conforme falado anteriormente os *itemsets* são gerados a partir de um conjunto de itens, estes conjuntos podem ser de um ou vários itens, assim sendo todo e qualquer item é um *itemset*, mas nem todo *itemset* é considerado frequente. O suporte é uma métrica com a finalidade de medir a frequência da ocorrência de um *itemset* na base de transações, assim um *itemset* é considerado frequente se a sua ocorrência é maior que o limite mínimo estabelecido para a problemática. Seja X o *itemset*, então:

$$\text{Suporte}(X) = \frac{\text{Número de transações com o conjunto } X}{\text{Número total de transações}}$$

Uma regra de associação é formada por um item antecedente que implica em um item consequente, ou seja, uma regra de associação é uma implicação lógica condicional, onde a existência de um item implica na existência de outro item nas transações. Dessa forma a extração de regras de associação busca identificar padrões de coocorrência ou dependência estatística entre dois ou mais itens.

A cada regra de associação extraída da base de transações são atribuídas três métricas como o suporte, a confiança e o *lift* da regra. O suporte da regra mede a

probabilidade conjunta de um paciente apresentar tanto o item antecedente (X) quanto o item consequente (Y), em relação ao total de transações da base. A confiança mede o grau de certeza de uma regra de associação formulada na implicação do item antecedente (X) no item consequente (Y), ou seja, expressa a probabilidade condicional de P (Y | X) de que um registro que contenha X também contenha Y. Desta forma, a métrica é calculada:

$$\text{Confiança}(X \rightarrow Y) = \frac{\text{Suporte}(X \cup Y)}{\text{Suporte}(X)}$$

O suporte mede a frequência, a confiança mede a precisão da regra, enquanto o *lift* mede que mede a relação do antecedente com o consequente. O *lift* atua como um corretor de viés para os itens que possuem uma alta ocorrência e alta prevalência isolada nas transações, o que implicaria em um alto suporte, sem que isso represente uma relação de causalidade, e sim apenas por ser um item comum. Desta forma, é definido como a razão entre a probabilidade condicional de Y dado X e o suporte de Y. Para valores de *lift* > 1 indica uma associação positiva, para valores de *lift* = 1 indica uma independência e para valores de *lift* < 1 indica uma associação negativa.

$$Lift(X \rightarrow Y) = \frac{\text{Confiança}(X \rightarrow Y)}{\text{Suporte}(Y)}$$

2.3.1 Algoritmo Apriori

O algoritmo Apriori foi proposto por Agrawal e Srikant (1994), sendo o pioneiro na mineração de *itemsets* frequentes e na geração de regras de associação. O nome "Apriori" se baseia no fato de o método de resolução do algoritmo ser derivado do conhecimento prévio e dos resultados obtidos nas etapas anteriores até a sua finalização. A cada etapa é guiado pelos *itemsets* gerados na etapa anterior para então prever os candidatos a *itemsets* frequentes nas etapas subsequentes. (Tan; Steinbach; Kumar, 2018).

Este algoritmo tem como objetivo analisar as combinações de itens de forma iterativa, identificando os conjuntos que são aprovados e atingem ou superam o limite estabelecido do suporte mínimo.

2.3.1.1 Mecanismo de Funcionamento do Algoritmo Apriori

O algoritmo realiza uma abordagem de busca em largura, onde executa um ciclo iterativo de geração de candidatos e contagem de suporte organizado por nível de cardinalidade. De modo semelhante a um funil, o primeiro nível é mais geral e analisa todos os conjuntos de itens, e a cada nível se torna mais específico e com menos conjuntos de itens.

1. Passo 1: O algoritmo realiza a leitura inicial, uma varredura por completo na base de dados para contabilizar a frequência de cada item individualmente. Neste passo os conjuntos de itens gerados contêm apenas um item, assim são conjuntos de tamanho 1.
2. Passo 2: Realiza-se o filtro nos conjuntos gerados no passo anterior, onde os conjuntos que não atingirem o limiar do suporte mínimo não são considerados candidatos a *itemsets* frequentes.
3. Passo 3: A partir do conjunto de *itemsets* frequentes gerados no nível anterior, é realizada a criação de conjuntos de *itemsets* unindo os *itemsets* frequentes. Nesta etapa é realizada a poda.
4. Passo 4: Realiza-se o filtro novamente nos conjuntos gerados, e apenas *itemsets* com suporte maior que o suporte mínimo são considerados *itemsets* frequentes e continuam no próximo nível.
5. Passo 5: O critério de parada acontece quando os passos anteriores são repetidos iterativamente, até que nenhum dos *itemsets* gerados possui valor de suporte igual ou maior do que o suporte mínimo.

Vale ressaltar que na etapa de poda, se um itemset não for frequente, os itemsets do nível seguinte que o contenham também são descartados, dispensando o cálculo de seus suportes.

2.3.2 Algoritmo FP-Growth

O algoritmo *FP-Growth* (*Frequent Pattern Growth*) foi proposto por Han, Pei e Yin (2000), surgindo como uma alternativa de alto desempenho e menor custo computacional ao algoritmo Apriori, já que foi proposto para evitar a geração explícita de candidatos. Desta forma, o algoritmo comprime o banco de dados de transações em uma estrutura compacta na memória RAM, denominada *FP-Tree* (*Frequent Pattern Tree*).

A construção da *FP-Tree* se dá por uma árvore prefixada estendida que contém de maneira condensada as transações da base de dados sem perda de informação sobre a frequência das ocorrências de maneira simultânea dos itens. (Tan, Steinbach e Kumar, 2018).

2.3.2.1 Mecanismo de Funcionamento do Algoritmo *FP-Growth*

Diferente do algoritmo Apriori, este algoritmo se divide em duas etapas, na primeira etapa, constrói-se a FP-Tree; na segunda, realiza-se a mineração de padrões frequentes.

A etapa 1 é composta pelos seguintes passos:

1. Varredura na base de dados onde o algoritmo calcula a frequência de cada *itemset* individual de apenas um item. Os *itemsets* cujo suporte for menor que o mínimo são eliminados, e então ordena os *itemsets* restantes em ordem decrescente de suporte.
2. Reordena as transações da base para conter apenas os *itemsets* frequentes em ordem decrescente de ocorrência.
3. Realiza uma nova varredura na base de dados reordenada no passo anterior, inserindo os *itemsets* na *FP-Tree*. Transações que contenham os mesmos itens iniciais pertencem ao mesmo nó na árvore, incrementando os seus contadores de ocorrência.

A etapa 2, é composta pelos seguintes passos:

1. O algoritmo percorre a tabela que continha os *itemsets* frequentes encontrados na etapa anterior.
2. Para cada item, o algoritmo rastreia na árvore e extrai todos os caminhos de prefixo que levam a este item.
3. Uma sub-árvore é gerada para aquele item, caso a sub-árvore contenha frequências acima do suporte mínimo, novos *itemsets* frequentes compostos são extraídos recursivamente. (Han, Pei e Yin, 2000).

2.4 Modelos de Memória Distribuída e arquitetura do *Databricks*

O avanço da computação em grande escala, termo conhecido como *Big Data*, contribuiu e necessitou de uma evolução nos modelos de processamento distribuídos. O

MapReduce (Dean e Ghemawat, 2004) trouxe ganhos à área ao permitir o processamento de dados massivos em *clusters* de computadores. Contudo, eram necessárias repetidas operações de gravação e leitura em disco entre as etapas de processamento, o que trazia impactos negativos em algoritmos iterativos e mineração exploratória de dados como o caso do algoritmo.

Devido a essa limitação, modelos de memória distribuída foram desenvolvidos, nestes modelos os dados são armazenados de maneira temporária na memória RAM distribuída entre múltiplos nós de um cluster, contribuindo para uma melhora no tempo de execução.

A consolidação dos modelos de memória distribuída contribuiu para o desenvolvimento de frameworks e linguagens que realizem processamento neste cenário, além de impulsionar o surgimento de plataformas com sistemas alocados e gerenciados em nuvem, caso do *Databricks*. O *Databricks* se destaca como uma IDE (ambiente de desenvolvimento) e um ambiente de computação em nuvem para gerenciar banco de dados. Este é projetado para otimizar a infraestrutura distribuída e simplificar o desenvolvimento de pipelines de aprendizado de máquina e mineração de dados (Gomez et al., 2021; Silva, 2022).

3 CONTEXTO INVESTIGADO E SITUAÇÃO PROBLEMA

Este estudo foi desenvolvido baseado nos dados disponibilizados pelo governo, que são obtidos quando pacientes com um quadro de síndrome respiratória aguda grave são atendidos em UPA (Unidades de Pronto Atendimento), o setor do governo responsável por armazenar e disponibilizar estes dados, além de gerenciar as unidades de atendimento a pacientes é o Ministério da Saúde.

Os dados disponíveis de SRAG são desde 2019, durante este ano até hoje o país enfrentou uma pandemia de 2020 a 2022 da covid-19, doença causada pelo vírus SARS-CoV-2 que causava complicações pulmonares com sintomas respiratórios como tosse, falta de ar, entre outros. Além de outras doenças respiratórias graves como pneumonia, quadros gripais em pessoas de faixa de risco como idosos, imunossuprimidos, com isso a análise destes dados para entender como exploração epidemiológica de coocorrências e geração de hipóteses. Já que sabendo os sintomas que este paciente apresenta é possível mapear quais sintomas podem ocorrer concomitantemente ou ao longo da evolução do

quadro, além de relacionar com as possíveis causas permitindo um tratamento mais eficiente e assertivo.

Para esta análise e resolução da situação-problema, utilizou-se a base SIVEP-Gripe, disponível no OpenDataSUS (Brasil, 2026), com o período utilizado de 2019 ao presente momento, onde os dados foram baixados em maio de 2026. Esta base possui 1.206.920 registros de pacientes atendidos, mas apenas 1.167.671 registros possuem valores para os sintomas que serão analisados, portanto não serão todos os registros da base a serem analisados. Os campos de sintomas para serem analisados são: febre, tosse, dor de garganta, dispneia, desconforto respiratório, saturação, diarreia, vômito e outros sintomas.

Portanto os dados existem e são públicos, mas as associações entre sintomas e diagnósticos não são evidentes na base bruta, e é essa lacuna analítica que a intervenção descrita na Seção 4 busca superar.

4 INTERVENÇÃO ADOTADA

Diante da situação observada e dos dados disponíveis, realizou-se um estudo aprofundado da base de dados, por meio de técnicas de mineração de dados e utilizando regras de associação para identificar padrões entre os sintomas, a fim de compreender melhor os impactos e subsidiar investigações futuras.

Para este estudo, uma série de etapas e ferramentas foram necessárias, utilizou-se o ambiente de desenvolvimento Databricks, uma plataforma que funciona no navegador e com a ideia de processamento distribuído desta forma possuindo a biblioteca '*pyspark'* nativa. Já as etapas percorridas para o desenvolvimento do trabalho são: limpeza e tratamento da base de dados em código *python* no *databricks*, desenvolvimento do código de regra de associação, ajuste dos hiperparâmetros e e, por fim, análise dos resultados obtidos.

Foram desenvolvidos dois códigos em *notebooks* diferentes no *databricks*, um contendo o código com o algoritmo *FP-Growth* e o outro contendo o algoritmo Apriori, em ambos a sequência da estrutura do código foi a mesma.

A primeira etapa foi realizar o *download* da base de dados no portal OpenDataSUS, onde o arquivo foi baixado no formato .csv. Após o download, o arquivo foi importado.

Desta maneira foi desenvolvido o código, onde as bibliotecas necessárias foram importadas, como ‘os’, ‘psutil’, ‘time’, ‘pyspark’, ‘pandas’, ‘mlxtend’. Foram realizadas a leitura do arquivo que já estava importado no *databricks*, e então foi realizada a limpeza na base de dados, onde os registros que não tinham valores para as colunas de sintomas foram excluídos.

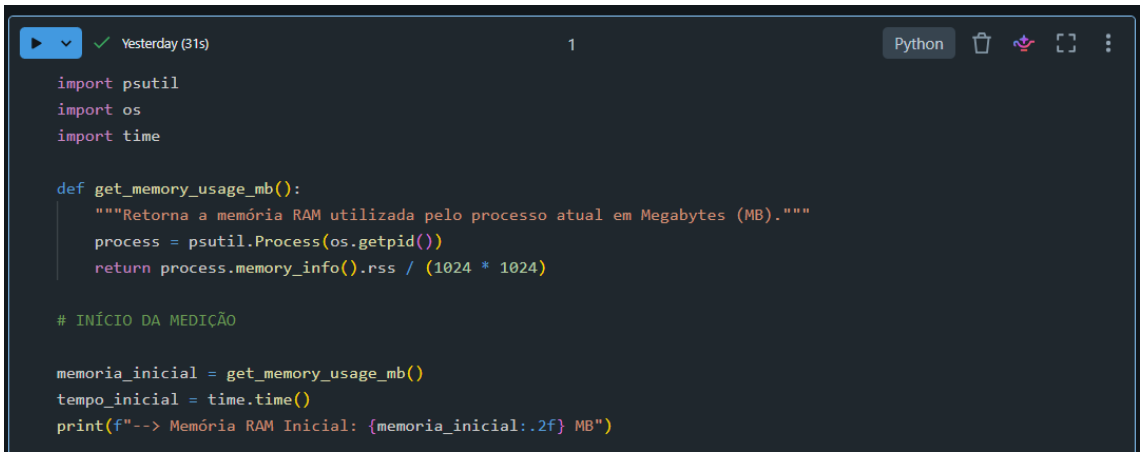
Por se tratar de uma análise binária já que o paciente poderia possuir ou não o sintoma, quando o valor fosse ‘1’ para a coluna do sintoma indicava que este paciente o apresentou, caso fosse outro valor indicava que o paciente não o apresentou ou foi ignorado. Assim sendo nas colunas de sintomas, mantivemos o valor ‘1’ e qualquer outro valor foi substituído por um campo nulo, desta maneira transformamos a base em uma lista de transações (assim os sintomas representaram itens).

Após isso foram desenvolvidos os códigos para execução dos algoritmos *FP-Growth* e Apriori, onde os valores de parâmetros estabelecidos foram: suporte = ‘0,4’, confiança = ‘0,6’ e *lift* > 1,1. Os valores de suporte e confiança foram estabelecidos de maneira exploratória, e o valor do *lift* segue recomendação do autor Mariano (2011). E então foram geradas as regras que atendiam à estes parâmetros.

Destaca-se que ao iniciar a execução do código foi armazenado o valor da memória RAM sendo utilizada no momento, e ao final da execução do código foi calculada a variação da memória final menos a inicial, desta maneira sabendo o quanto foi utilizado de memória. Além disso foi calculado o tempo de execução do código, para poder analisar o custo computacional envolvido na execução dos diferentes algoritmos.

Abaixo, seguem imagens referentes ao código desenvolvido para a análise das regras de associação e do custo computacional:

Figura 2 – Trecho 1 do código utilizando o algoritmo *FP-Growth*



Fonte: elaborado pelo autor (2026)

Na figura 2, o trecho do código destacado realiza as etapas de importação das bibliotecas *psutil, os* e *time*, logo em seguida é declarado a função '*get_memory_usage_mb*' que faz o cálculo e armazena o valor da memória RAM inicial. Depois é armazenado o tempo inicial de quando o código é executado.

Figura 3 – Trecho 2 do código utilizando o algoritmo *FP-Growth*

```
from pyspark.sql import functions as F
from pyspark.ml.fpm import FPGrowth

caminho_arquivo = "/Volumes/workspace/default/tcc/Influenza_2026.csv"
sintomas_cols = ["FEBRE", "TOSSE", "GARGANTA", "DISPNEIA", "DESC_RESP", "SATURACAO", "DIARREIA", "VOMITO",
"OUTRO_SIN"]

df_raw = spark.read.csv(caminho_arquivo, header=True, inferSchema=True, sep=';')

df_transformed = df_raw
for sintoma in sintomas_cols:
    df_transformed = df_transformed.withColumn(
        sintoma,
        F.when(F.col(sintoma) == 1, F.lit(sintoma)).otherwise(F.lit(""))
    )
```

Fonte: elaborado pelo autor (2026)

Na figura 3, é feito a importação do módulo '*functions*' da biblioteca '*pyspark.sql*' e o módulo '*FPGrowth*' da biblioteca '*pyspark.ml.fpm'*. Então segue-se para a leitura do arquivo no formato .csv, e é armazenado em uma lista as colunas com os sintomas que serão analisados. Depois é feito as transformações necessárias de valores, onde caso tenha o valor '1' na coluna mantém esse valor, e caso o valor seja diferente deste então transforma para valores de '*string*' nula.

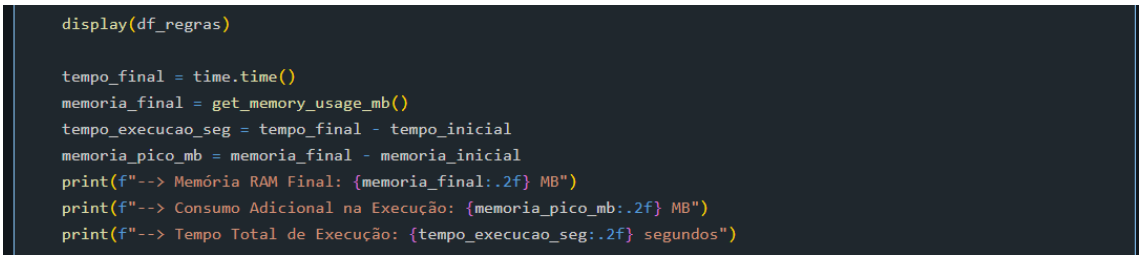
Figura 4 – Trecho 3 do código utilizando o algoritmo *FP-Growth*

```
df_basket = df_transformed.withColumn(
    "items",
    F.array_remove(F.array([F.col(c) for c in sintomas_cols]), "")
).select("items")

df_basket_valido = df_basket.filter(F.size(F.col("items")) > 0)

# Configurar e Treinar o FP-Growth
fp_growth = FPGrowth(itemsCol="items", minSupport=0.4, minConfidence=0.6)
model = fp_growth.fit(df_basket_valido)
df_freq = model.freqItemsets.orderBy(F.desc("freq"))

df_regras = model.associationRules \
    .filter(F.col("lift") > 1.1) \
    .orderBy(F.desc("lift"))
```

Fonte: elaborado pelo autor (2026)

Na figura 4, o *dataframe* que armazena os dados é transformado em uma cesta, onde os atendimentos de pacientes são transformados em transações, e é feito o cálculo das regras de associação entre os sintomas, ao final do código após gerado as regras de associação são filtradas apenas regras com o valor de *lift* > 1,1.

Figura 5 – Trecho 4 do código utilizando o algoritmo *FP-Growth*

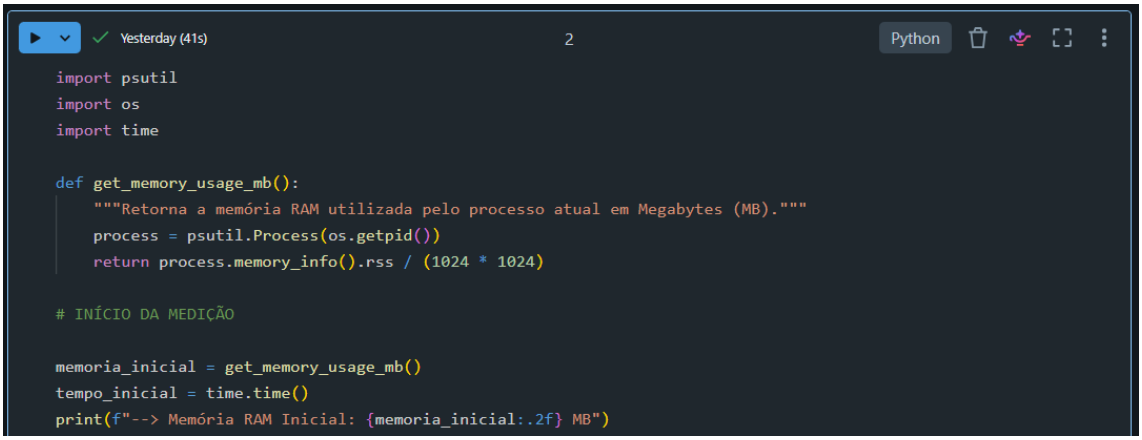


```
display(df_regras)

tempo_final = time.time()
memoria_final = get_memory_usage_mb()
tempo_execucao_seg = tempo_final - tempo_inicial
memoria_pico_mb = memoria_final - memoria_inicial
print(f"--> Memória RAM Final: {memoria_final:.2f} MB")
print(f"--> Consumo Adicional na Execução: {memoria_pico_mb:.2f} MB")
print(f"--> Tempo Total de Execução: {tempo_execucao_seg:.2f} segundos")
```

Fonte: elaborado pelo autor (2026)

Na figura 5, é calculado o tempo final de execução do código e a memória final utilizada, para finalizar é feito o cálculo da variação de tempo de execução e da variação de memória necessária para execução. Então mensagens são escritas na tela com esses valores.

Figura 6 – Trecho 1 do código utilizando o algoritmo Apriori



```
Yesterday (41s)    2    Python

import psutil
import os
import time

def get_memory_usage_mb():
    """Retorna a memória RAM utilizada pelo processo atual em Megabytes (MB)."""
    process = psutil.Process(os.getpid())
    return process.memory_info().rss / (1024 * 1024)

# INÍCIO DA MEDIÇÃO

memoria_inicial = get_memory_usage_mb()
tempo_inicial = time.time()
print(f"--> Memória RAM Inicial: {memoria_inicial:.2f} MB")
```

Fonte: elaborado pelo autor (2026)

Na figura 6, o trecho do código destacado realiza as etapas de importação das bibliotecas *psutil, os* e *time*, logo em seguida é declarado a função '*get_memory_usage_mb*' que faz o cálculo e armazena o valor da memória RAM inicial. Depois é armazenado o tempo inicial de quando o código é executado.

Figura 7 – Trecho 2 do código utilizando o algoritmo Apriori

```
from pyspark.sql import functions as F
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules

caminho_arquivo = "/Volumes/workspace/default/tcc/Influenza_2026.csv"
sintomas_cols = ["FEBRE", "TOSSE", "GARGANTA", "DISPNEIA", "DESC_RESP", "SATURACAO", "DIARREIA", "VOMITO",
"OUTRO_SIN"]

df_raw = spark.read.csv(caminho_arquivo, header=True, inferSchema=True, sep=';')

df_transformed = df_raw
for sintoma in sintomas_cols:
    df_transformed = df_transformed.withColumn(
        sintoma,
        F.when(F.col(sintoma) == 1, F.lit(1)).otherwise(F.lit(0))
    )
```

Fonte: elaborado pelo autor (2026)

Na figura 7, é feito a importação do módulo '*functions*' da biblioteca '*pyspark.sql*', a biblioteca '*pandas*' e o módulo 'apriori, *association_rules'* da biblioteca '*mlxtend.frequent_patterns'*. Então segue-se para a leitura do arquivo no formato .csv, e é armazenado em uma lista as colunas com os sintomas que serão analisados. Depois é feito as transformações necessárias de valores, onde caso tenha o valor '1' na coluna mantém esse valor, e caso o valor seja diferente deste então transforma para valores de '*string*' nula.

Figura 8 – Trecho 3 do código utilizando o algoritmo Apriori

```
df_sintomas_spark = df_transformed.select(sintomas_cols)

soma_sintomas = sum([F.col(c) for c in sintomas_cols])
df_sintomas_validos = df_sintomas_spark.filter(soma_sintomas > 0)

df_pandas = df_sintomas_validos.toPandas().astype(bool)

frequent_itemsets = apriori(df_pandas, min_support=0.4, use_colnames=True)

frequent_itemsets_spark = frequent_itemsets.copy()
frequent_itemsets_spark['itemsets'] = frequent_itemsets_spark['itemsets'].apply(lambda x: ', '.join(list(x))).
astype("string")

df_freq_spark = spark.createDataFrame(frequent_itemsets_spark)

regras_pandas = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.6)
```

Fonte: elaborado pelo autor (2026)

Na figura 8, o *dataframe* que armazena os dados é transformado em uma cesta, onde os atendimentos de pacientes são transformados em transações, e é feito o cálculo

dos i*temsets* frequentes, e então o cálculo das regras de associação entre os sintomas que superem o valor de suporte mínimo e confiança mínimo.

Figura 9 – Trecho 4 do código utilizando o algoritmo Apriori

```
regras_pandas['antecedents'] = regras_pandas['antecedents'].apply(lambda x: ', '.join(list(x))).astype("string")
regras_pandas['consequents'] = regras_pandas['consequents'].apply(lambda x: ', '.join(list(x))).astype("string")

df_regras_spark = spark.createDataFrame(regras_pandas)

df_regras_final = df_regras_spark \
    .filter(F.col("lift") > 1.1) \
    .orderBy(F.desc("lift"))

display(df_regras_final)
```

Fonte: elaborado pelo autor (2026)

Na figura 9, após ser geradas as regras de associação entre os sintomas, estas são filtradas e apenas das regras de associação com valor de *lift* maior que 1,1 são admitidas.

Figura 10 – Trecho 5 do código utilizando o algoritmo Apriori

```
tempo_final = time.time()
memoria_final = get_memory_usage_mb()
tempo_execucao_seg = tempo_final - tempo_inicial
memoria_pico_mb = memoria_final - memoria_inicial
print(f"--> Memória RAM Final: {memoria_final:.2f} MB")
print(f"--> Consumo Adicional na Execução: {memoria_pico_mb:.2f} MB")
print(f"--> Tempo Total de Execução: {tempo_execucao_seg:.2f} segundos")
```

Fonte: elaborado pelo autor (2026)

Na figura 10, é calculado o tempo final de execução do código e a memória final utilizada, para finalizar é feito o cálculo da variação de tempo de execução e da variação de memória necessária para execução. Então mensagens são escritas na tela com esses valores.

5 RESULTADOS ALCANÇADOS

5.1 Regras de Associações Obtidas

Através do algoritmo *FP-Growth*, no Brasil no período de 2019 a 2026, a base reunia 1.206.920 registros e apenas 1.167.671 registros de sintomas respiratórios em

pacientes atendidos no país. No código, para as métricas de confiança e suporte foram utilizados os valores 0.6 (60%) e 0.4 (40%) respectivamente.

Com esses valores, para ambas as técnicas como o Apriori e *FP-Growth* os *itemsets* de sintomas mais frequentes são apresentados na Tabela 1:

Tabela 1 – *Itemsets* de sintomas e suas respectivas frequências

Itemsets	**Frequência**
Dispneia	817571
Tosse	806311
Febre	686423
Desconforto Respiratório	673620
Saturação	659305
Tosse, Dispneia	589830
Desconforto Respiratório, Dispneia	554531
Saturação, Dispneia	542978
Febre, Tosse	530773
Saturação, Desconforto Respiratório	484665
Desconforto Respiratório, Tosse	482523
Febre, Dispneia	480015

Fonte: Elaborado pelo autor (2026).

Utilizando os mesmos valores de confiança e suporte, e com '*lift'* maior ou igual a 1,1 foram determinadas as seguintes para ambas as técnicas como o Apriori e o *FP-Growth*, como mostra a tabela 2:

Tabela 2 – Regra de associação dos sintomas

Antecedente	**Consequente**	**Confiança**	***Lift***	**Suporte**
Desconforto Respiratório	Saturação	0,7194	1,2742	0,4150
Saturação	Desconforto Respiratório	0,7351	1,2742	0,4150
Dispneia	Saturação	0,6641	1,1762	0,4650
Saturação	Dispneia	0,8235	1,1762	0,4650

Desconforto Respiratório	Dispneia	0,8232	1,1757	0,4749
Dispneia	Desconforto Respiratório	0,6782	1,1757	0,4749
Tosse	Febre	0,6582	1,1197	0,4545
Febre	Tosse	0,7732	1,1197	0,4545

Fonte: Elaborado pelo autor (2026).

5.2 Resultados do Desempenho

Conforme apresentado neste trabalho, foram utilizados dois algoritmos para o cálculo das regras de associação, o algoritmo FP-Growth que usa modelo de memória distribuída e o Apriori que refaz o processamento a cada iteração de *itemsets*. Com a ideia de comparar os dois algoritmos para entender qual modelo utilizar, foi realizada uma comparação do custo computacional entre ambos, estes foram desenvolvidos na IDE *Databricks* com um código enxuto, ou seja, os códigos foram reduzidos às etapas necessárias para gerar os resultados, evitando códigos chaves como ‘print’ e ‘display’, e executados no mesmo ambiente de cluster de maneira assíncrona, onde primeiro foi executado o código Apriori e ao finalizar foi executado o código da técnica *FP-Growth* ao final a comparação do custo computacional levou em consideração a memória necessária para a execução e o tempo decorrido, como mostra a tabela 3.

Tabela 3 – Valores do custo computacional para execução dos códigos

Algoritmo	**Tempo de Execução**	**Memória Necessária**
FP-Growth	25,60 segundos	31,96 MB
Apriori	37,24 segundos	78,69 MB

Fonte: Elaborado pelo autor (2026)

Como observado na tabela, o algoritmo FP-Growth necessita de um custo computacional menor que o algoritmo Apriori, quanto ao tempo de execução, o FP-Growth foi aproximadamente 11,64 segundos mais rápido e quanto à memória medida, o FP-Growth utilizou menos da metade do valor registrado para o Apriori. Isso se deve ao fato de assim como o FP-Growth o *Databricks* é uma plataforma gerenciada que executa

workloads Spark distribuídos, além de uma IDE de modelagem de memória distribuída, que permite que tarefas sejam executadas ao mesmo tempo, além do algoritmo Apriori ter a necessidade de percorrer os *itemsets* a cada iteração, enquanto o modelo FP-Growth cria uma árvore com os *itemsets* que são percorridos apenas uma vez (*FP-Tree*), reduzindo varreduras e candidatos em comparação ao Apriori, não precisando de reprocessamento.

6 CONSIDERAÇÕES FINAIS

Objetivou-se com este relato tecnológico aplicar os algoritmos de regras de associação Apriori e FP-Growth aos dados de Síndrome Respiratória Aguda Grave (SRAG), e essas técnicas de mineração de dados foram aplicadas a fim de extrair conhecimento por meio de regras de associação estatisticamente robustas e a plausabilidade clínica permanece como hipótese a ser validada.

O algoritmo FP-Growth, desenvolvido em um ambiente de processamento distribuído (*Databricks*) com a biblioteca PySpark, demonstrou ser adequado ao processamento da base nas condições testadas para trabalhar com um grande volume de dados secundários de saúde pública. Haja vista que este algoritmo elimina a necessidade de realizar múltiplos escaneamentos da base, como o algoritmo Apriori, por meio da estrutura de árvore de frequências gerada uma única vez (*FP-Tree*), desta maneira o algoritmo otimizou o tempo decorrido para o processamento em memória, contribuindo para uma rápida extração de *itemsets* frequentes que estavam de acordo com os limiares de suporte e confiança estipulados.

Do ponto de vista analítico, prático e clínico, a análise das regras de associação entre sintomas com valor de *Lift* superior a 1,1 e alta confiança indicou relações fundamentais sobre o comportamento concomitante dos sintomas respiratórios. “A identificação desses sintomas concomitantes com forte associação atua como uma referência e os padrões podem ser discutidos por especialistas como hipóteses para estudos posteriores. Desta maneira evidencia capacidade exploratória de identificar coocorrências na base analisada.

Ressalta-se que as regras identificadas expressam associação estatística entre sintomas, não relação de causalidade nem recomendação clínica; sua aplicação assistencial exigiria validação por profissionais de saúde e estudos específicos. Portanto, ressalvam-se as cautelas clínicas, pois estas devem ser feitas por profissionais de saúde.

Esta pesquisa contou com o apoio do Gemini (Google, modelo Flash 3.6) durante a elaboração deste trabalho. As ferramentas foram usadas para organizar ideias, estruturar seções, aprimorar a redação, tradução de textos e desenvolvimento do código.

7 DECLARAÇÃO DE USO DE FERRAMENTAS DE INTELIGÊNCIA ARTIFICIAL

O uso da IA se restringiu ao apoio textual e ao desenvolvimento do código. As análises, a interpretação dos resultados, a proposta tecnológica e o desenvolvimento da solução foram feitos pelo autor. As decisões sobre o problema investigado, a fundamentação teórica, o tratamento dos dados e as conclusões partiram da experiência adquirida no trabalho e das orientações acadêmicas recebidas. Todo o conteúdo gerado com apoio da ferramenta foi revisado, adaptado ao contexto da pesquisa e validado antes de ser incorporado ao trabalho. A responsabilidade pelo conteúdo final é integralmente do autor.

8 REFERÊNCIAS

AGRAWAL, R.; IMIELINSKI, T.; SWAMI, A. **Mining association rules between sets of items in large databases.** In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 1993, Washington. Proceedings [...]. New York: ACM, 1993. p. 207-216.

AGRAWAL, R.; SRIKANT, R. **Fast algorithms for mining association rules.** In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES (VLDB), 20., 1994, Santiago. Proceedings [...]. San Francisco: Morgan Kaufmann, 1994. p. 487-499.

ARAÚJO, K. L. R.; AQUINO, É. C.; SILVA, L. L. S.; TERNES, Y. M. F. **Fatores associados à Síndrome Respiratória Aguda Grave em uma Região Central do Brasil.** Ciência & Saúde Coletiva, 12/08/2020. v. 25, supl. 2, p. 4121–4130, DOI 10.1590/1413-812320202510.2.26802020.

BRASIL. Ministério da Saúde. **SRAG: banco de dados de Síndrome Respiratória Aguda Grave.** OpenDataSUS, Brasília, DF, 2026. Disponível em: https://opendatasus.saude.gov.br. Acesso em: 4 maio 2026.

BRASIL. Ministério da Saúde. Secretaria de Vigilância em Saúde e Ambiente. **Guia de orientações para profissionais de saúde**: síndrome gripal (SG) e síndrome respiratória

aguda grave (SRAG). Brasília, DF: Ministério da Saúde, 2025. Disponível em: https://www.gov.br/saude/pt-br/centrais-de-conteudo/publicacoes/guias-e-manuais/2025/guia-de-orientacoes-para-profissionais-de-saude-srag.pdf. Acesso em: 3 agosto 2026.

DEAN, J.; GHEMAWAT, S. **MapReduce: simplified data processing on large clusters.** In: SYMPOSIUM ON OPERATING SYSTEM DESIGN AND IMPLEMENTATION (OSDI), 6., 2004, San Francisco. Proceedings [...]. Berkeley: USENIX, 2004. p. 137-150.

FAYYAD, U.; PIATETSKY-SHAPIRO, G.; SMYTH, P. **From data mining to knowledge discovery in databases.** AI Magazine, v. 17, n. 3, p. 37-54, 1996.

GOMEZ, C. E. et al. **Arquiteturas de big data e computação em nuvem aplicadas à análise de dados em saúde pública no Brasil.** Revista Brasileira de Computação Aplicada, v. 13, n. 2, p. 45-56, 2021.

HAN, J.; PEI, J.; YIN, Y. **Mining frequent patterns without candidate generation.** In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 2000, Dallas. Proceedings [...]. New York: ACM, 2000. p. 1-12.

MARIANO, M. **Comparação de algoritmos paralelos para a extração de regras de associação no modelo de memória distribuída.** Dissertação - Universidade Federal de Mato Grosso do Sul, Campo Grande, 2011.

SILVA, M. H. da. **Arquiteturas em nuvem para engenharia de dados: um estudo de caso focado no ecossistema Databricks e PySpark.** Monografia - Universidade Federal de Minas Gerais, Belo Horizonte, 2022.

TAN, P.-N.; STEINBACH, M.; KUMAR, V. **Introduction to data mining.** 2. ed. Boston: Pearson, 2018.

THACHARODI, A. et al. **Revolutionizing healthcare and medicine: the impact of modern technologies for a healthier future - a comprehensive review.** Health Care Science, 2024. https://doi.org/10.1002/hcs2.115