UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Câmpus de São José do Rio Preto

INSTITUTO DE BIOCIÊNCIAS, LETRAS E CIÊNCIAS EXATAS
IBILCE / UNESP - CÂMPUS DE SÃO JOSÉ DO RIO PRETO

BEATRIZ FERREIRA DE LIMA

Otimização de desempenho de algoritmo para detecção de *outliers* em séries temporais

São José do Rio Preto
2021

BEATRIZ FERREIRA DE LIMA

Otimização de desempenho de algoritmo para detecção de *outliers* em séries temporais

Trabalho de Conclusão de Curso apresentado ao Departamento de Ciências de Computação e Estatística do Instituto de Biociências Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Carlos Roberto Valêncio

São José do Rio Preto
2021

L732o

Lima, Beatriz Ferreira de

Otimização de desempenho de algoritmo para detecção de outliers em séries temporais / Beatriz Ferreira de Lima. -- São José do Rio Preto, 2022

45 p. : il., tabs.

Trabalho de conclusão de curso (Bacharelado - Ciência da Computação) - Universidade Estadual Paulista (Unesp), Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto

Orientador: Carlos Roberto Valêncio

1. Ciência da computação. 2. Banco de dados. 3. Detecção de outliers. 4. Séries temporais. 5. Algoritmos paralelos. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca do Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

BEATRIZ FERREIRA DE LIMA

Otimização de desempenho de algoritmo para detecção de *outliers* em séries temporais

Trabalho de Conclusão de Curso apresentado ao Departamento de Ciências de Computação e Estatística do Instituto de Biociências Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Banca examinadora:

Prof. Dr. Carlos Roberto Valêncio
UNESP – São José do Rio Preto
Orientador

Prof. Dr. Geraldo Francisco Donegá Zafalon
UNESP – São José do Rio Preto

Profª Dra. Carina Alexandra Rondini
UNESP – São José do Rio Preto

São José do Rio Preto
2021

"Happiness can be found, even in the darkest of times, if one only remembers to turn on the light"

Harry Potter e o Prisioneiro de Azkaban

Agradecimentos

Agradeço primeiramente a Deus e todos os seres de luz que me ajudaram e sustentaram até aqui. Agradeço também aos meus pais, Leila e Vanderlei, que me apoiaram em todos os passos dessa caminhada. Ao meu orientador, Prof. Dr. Carlos Roberto Valêncio, meu muito obrigada por aceitar me orientar nesses tempos caóticos que estamos vivendo e por tornar possível a conclusão da minha formação acadêmica. Agradeço aos meus irmãos, Isabela e Thiago, que trilharam seus caminhos antes de mim e me guiaram sempre que possível. À minha Mel por ajudar nos tempos complicados, obrigada. Aos meus colegas e amigos da graduação que tornaram os meus dias melhores ao compartilharem comigo todos os desafios enfrentados durante o curso, meus mais sinceros agradecimentos. Ao primeiro grande amigo que eu fiz na faculdade (aquela amizade nascida na fila do RU), Luís Vinícius, meu muito obrigada. À Vitoria e à Viviane (que gritou comigo quando necessário) que se dispuseram a me ajudar quando eu achei que a execução deste trabalho não seria possível, minha eterna gratidão. Agradeço ao Douglas, o parceiro que a graduação me trouxe, que me acompanhou por todos os momentos, sendo eles bons ou ruins, e me ajudou a chegar até aqui.

Resumo

As séries temporais têm se mostrado presentes em diversas áreas de grande valor econômico, como o mercado de ações e a indústria. Em circunstância de características Big Data, pode-se ter os dados processados em tempo real (real time) ou quase em tempo real (near real time), e fontes capazes de gerar volumes elevados de dados, o que impõe a fase de preparação a necessidade de execução da limpeza destes dados de forma eficaz e eficiente ao lidar com estes requisitos. Existem diversos algoritmos que podem ser utilizados para realizar esse processo, porém esses podem conter limitações como baixo rendimento, distorção dos dados, tempo elevado de processamento, entre outros. Um dos problemas a ser tratado na preparação dos dados é a detecção de outliers, dados que podem refletir distorções e que podem implicar em custos adicionais na fase de limpeza dos dados. Assim, este trabalho teve como objetivo propor um algoritmo que realize a detecção de outliers e, posteriormente, a limpeza dos dados discrepantes de forma eficaz e eficiente, em que se buscou manter a integralidade da informação obtida através dos dados de séries temporais. A partir dos testes realizados com o algoritmo, foi possível constatar uma redução significativa no tempo de processamento, de até 70%, sem que os dados originais sofressem alterações.

Palavras-chave: Banco de Dados, Séries Temporais. Algoritmos paralelos. Limpeza de dados. Detecção de outliers. Apache Spark.

ABSTRACT

Time series have been present in several areas of great economic value, such as the stock market and industry. In circumstances of Big Data characteristics, data can be processed in real time or near real time, and sources capable of generating high volumes of data, which impose the preparation phase to need to perform cleaning of these data effectively and efficiently when dealing with these requirements. There are several algorithms that can be used to carry out this process, but they may have limitations such as low yield, data distortion, high processing time, among others. One of the problems to be addressed in data preparation is the detection of outliers, data that can reflect distortions and that can imply additional costs in the data cleaning phase. Thus, this work aimed to propose an algorithm that performs the detection of outliers and, subsequently, the cleaning of outliers in an effective and efficient way, which sought to maintain the completeness of the information obtained through time series data. From the tests carried out with the algorithm, it was possible to verify a significant reduction in processing time, up to 70%, without altering the original data.

Lista de Ilustrações

Lista de Tabelas

Lista de Abreviaturas e Siglas

EWMA - *Exponential Weighted Moving Average*

GAN - *Generative Adversarial Networks*

GBD - Grupo de Banco de Dados

GPU - *Graphics Processing Unit*

HPC - *High Performance Computing Clusters*

IBILCE - Instituto de Biociências, Letras e Ciências Exatas

KDD - *Knowledge Discovery in Databases*

RDD - *Resilient Distributed Datasets*

SQL - *Structured Query Language*

UNESP - Universidade Estadual Paulista "Júlio de Mesquita Filho"

VPN - *Virtual Private Network*

WMA - *Weighted Moving Average*

Sumário

1 Introdução

Ao longo dos anos, as séries temporais têm sido utilizadas em diversos campos, como economia, processamento de sinais, sistemas de geolocalização, mercado de ações, dentre outros. Tais dados podem ser entendidos como uma sequência de variáveis aleatórias, na qual cada uma das variáveis denota um valor arbitrário obtido em um determinado ponto do tempo (WANG; WANG, 2019). Uma série temporal pode ser ilustrada por uma sequência de valores, como: ...100,00;110,00;120,00...., na qual cada valor está associado a um ponto no tempo.

Os dados são coletados normalmente através de sensores, principal fonte desses dados no campo industrial — o que nesse cenário representa dispositivos físicos acoplados aos maquinários utilizados, como por exemplo: por aproximação, acelerômetro, de temperatura etc.—, o que possibilita o processamento dos mesmos em tempo real (*real time*) ou quase em tempo real (*near real time*). Além disso, essa e demais fontes costumam gerar volumes elevados de dados. Assim, a limpeza dos mesmos deve ser feita de forma eficaz e eficiente, de forma a lidar com esse volume de dados (WANG; WANG, 2019).

Embora existam diversos algoritmos que possam ser utilizados para a limpeza desses dados, estes possuem limitações, seja rendimento abaixo do esperado, pelo tempo elevado de processamento ou pela obtenção de resultados que não são plenamente confiáveis — como é o caso de alguns dos algoritmos que utilizam-se de métodos de suavização para a eliminação de ruídos, em que os dados originais são modificados de modo a distorcê-los e, assim, produz-se resultados incertos para as análises (WANG; WANG, 2019) — , dentre outros fatores.

Para auxiliar na limpeza de dados, Wang e Wang (2019) indicam que se pode incluir a detecção de outliers — também chamados pontos discrepantes, que representam dados anômalos — antes da realização da limpeza em si, de modo a reparar apenas os dados considerados errados.

Gupta et al. (2013) explora dois tipos de detecção: identificar pontos como outliers e subsequências como outliers. As quais podem ser feitas de modo a detectar outliers em bancos de dados de séries temporais ou em uma determinada série temporal.

Para a execução dessa tarefa, há na literatura a proposição de diversos algoritmos. Para a detecção de pontos como outliers, pode-se citar o trabalho de Hill e Minsker (2010), que primeiro agrupam os pontos de dados e tomam a média dos agrupamentos como o valor previsto do ponto.

Na detecção de subsequências como valores discrepantes, o algoritmo heurístico por reordenação de sequências candidatas, proposto por Keogh et al. (2005), além do algoritmo de aceleração que se utiliza de valores de hash sensíveis locais proposto por Wei et al. (2006) podem ser destacados. Tem-se também o proposto por Keogh et al. (2006), que é destinado a encontrar subsequências de dados anômalos em séries de dados temporais. Neste caso, é proposto um algoritmo que se baseia na descoberta de dados discordantes por meio de "força bruta".

Embora a lógica para a detecção destas distorções nos dados seja muito simples, a complexidade e tempo de processamento são elevados, de modo que a utilização do mesmo se torna inviável quando se considera grandes volumes. Deste modo, tem-se a indicação de que a detecção de outliers é um caminho a ser considerado para limpeza de dados, com o propósito de facilitar a realização da mesma, além do crescente destaque dos dados de séries temporais, cuja limpeza ganha relevância nesse cenário e o estudo desses conceitos e aprimoramento das ferramentas.

1.1 Motivação e Escopo

Na literatura, podem ser encontrados diversos trabalhos que buscam resolver os problemas de qualidade em dados de séries temporais, de modo a contornar desafios vindos de questões como sazonalidade, dados tendenciosos, autocorrelações e até mesmo lacuna nos dados (DASU; DUAN; SRIVASTAVA, 2016). Além destes obstáculos, deve-se ressaltar que os dados coletados são processados em tempo real (*real time*) ou quase em tempo real (*near real time*), e as fontes costumam gerar volumes elevados de dados.

Wang e Wang (2019) reduzem os erros de séries temporais a duas categorias: erro de registro de data/hora (*timestamp errors)* e erros de valor observado (*observed value errors*). Para a primeira categoria, Song et al. (2016) propõem um método para a limpeza dos dados. Por outro lado, para dados de série temporal cujos erros sejam de valores observados, existem

duas abordagens de processamento para lidar com os erros: descartar os dados errados, que se baseia em identificar a série por meio de um algoritmo de detecção de anomalias e descartar os dados anormais; e limpar os dados, o que pode ser feito manual ou automaticamente. Indubitavelmente, a limpeza manual possui alta acurácia, porém a sua implementação é difícil pois o tempo e esforço dedicados são proporcionais a sua precisão.

Dentre as dificuldades encontradas em dados de séries temporais, tem-se que a principal fonte de dados de séries temporais são dados coletados por meio de sensores que, geralmente, coletam dados em uma frequência de segundos, de modo que o volume de dados é significativo (WANG; WANG, 2019). Além disso, sabe-se que dados de sensores tendem a não ser confiáveis (JEFFERY et al, 2006).

As séries temporais são contínuas (WANG; WANG, 2019), deste modo é importante que o algoritmo de limpeza seja capaz de lidar com uma grande quantidade de dados, além de possuir um bom rendimento, isto é, a limpeza a ser realizada proporcione resultados satisfatórios, de modo a ser efetiva em um grande volume de dados.

Existem algumas abordagens para mitigar o problema. Uma delas pode ser a utilização de algoritmos para detecção de *outliers* antes da realização do processo de limpeza em si (WANG; WANG, 2019). Para este fim, existem diversos algoritmos propostos na literatura. Um que deve ser considerado por tratar o problema com extrema simplicidade, é o proposto por Keogh et al. (2006), que utiliza de "força bruta" para detectar subsequências de dados anômalos em séries temporais.

O algoritmo, no entanto, possui alta complexidade e elevado tempo de processamento, uma vez que as comparações feitas para a detecção de valores discrepantes são feitas uma a uma, em que se compara todas as subsequências da série temporal (KEOGH et al., 2006). Para suavizar esse problema, algumas soluções foram propostas, porém, embora melhorem o desempenho do algoritmo, a sua precisão ainda pode ser aprimorada. Para alcançar este objetivo, isto é, aprimorar o algoritmo, pode-se recorrer à técnica de processamento paralelo e distribuído, de forma a dividir as tarefas em diversos núcleos ou máquinas, aumentando, assim, a sua escalabilidade e eficiência (HILDEBRANDT et al., 2017).

1.2 Objetivo

1.2.1 Objetivos Gerais

De maneira geral, o objetivo deste trabalho é adaptar o algoritmo proposto por Keogh et al. (2006) utilizado para a detecção de outliers e adicionar à sua execução a realização de

limpeza de dados de séries temporais. Para isso, será utilizado o conceito de computação paralela e distribuída, que consiste na distribuição da tarefa a ser executada em diversos núcleos 14 ou máquinas, de modo a aumentar a sua escalabilidade e eficiência, além da diminuição do tempo de processamento necessário para a sua execução, em que se proporciona a limpeza de forma a abranger o elevado volume de dados gerado em uma série temporal.

Em suma, a contribuição científica deste trabalho baseia-se em propor um algoritmo eficiente e eficaz para a detecção de *outliers*, com a posterior limpeza dos dados identificados como anômalos que será realizada por meio, a princípio, da abordagem de limpeza baseada em restrições, de modo que um estudo será conduzido para definir o algoritmo mais adequado para esse processo. Isso objetiva permitir que o elevado volume de dados gerado por séries temporais possa ser tratado, de modo a visar a integralidade da informação e de seu valor.

1.2.2 Objetivos Específicos

Em relação aos objetivos específicos, tem-se os seguintes:

a) desenvolver um algoritmo eficaz e eficiente, baseado no algoritmo para detecção de *outliers* proposto por Keogh et al. (2006), por meio do *framework* Apache Spark;
b) agregar a este o processo de limpeza dos dados anômalos, de modo a tornar esta etapa mais eficiente, na tentativa de garantir a integralidade da informação, bem como um bom desempenho ao limpar grandes volumes de dados.

1.3 Metodologia

Para este trabalho, adotou-se as seguintes etapas: revisão bibliográfica, desenvolvimento do projeto proposto, testes e conclusão.

O processo de revisão bibliográfica baseou-se em um levantamento de estudos que compõem o estado da arte do assunto abordado, bem como daqueles pertinentes a assuntos correlatos, isto é, estudos sobre detecção de dados anômalos, limpeza de dados, séries temporais e limpeza de dados em séries temporais, além dos conceitos e recursos de paralelização.

Depois, selecionou-se os estudos de maior relevância para o assunto, de modo a estabelecer um método de validação para o trabalho quando este fosse concluído.

Após essa etapa, iniciou-se, então, o desenvolvimento do projeto proposto, ou seja, foi realizada a paralelização do algoritmo proposto por Keogh et al. (2006), e agregação a este do

algoritmo de limpeza escolhido, por meio do *framework* Apache Spark, baseado nos conceitos de detecção de *outliers* e limpeza para grandes volumes de dados.

Em sequência, foram realizados os testes a fim de validar os resultados obtidos pelo algoritmo desenvolvido, de modo a observar a precisão alcançada, bem como o tempo de processamento, para comprovar a hipótese apresentada neste trabalho.

1.4 Organização da Monografia

Esta monografia será dividida em cinco capítulos da seguinte forma:

a) Capítulo 1 - Considerações iniciais, motivação e escopo, objetivo, metodologia e exequibilidade;
b) Capítulo 2 – Revisão Bibliográfica sobre séries temporais, limpeza de dados, detecção de *outliers*, escalabilidade, trabalhos correlatos e estado da arte;
c) Capítulo 3 - Desenvolvimento do projeto, testes e resultados;
d) Capítulo 4 - Exposição dos testes e resultados obtidos;
e) Capítulo 5 - Conclusões, contribuições e trabalhos futuros.

2 Revisão Bibliográfica

Neste capítulo são apresentados os conceitos de séries temporais, limpeza de dados, detecção de outliers e escalabilidade. É efetuada também a apresentação das principais funcionalidades da ferramenta Apache Spark. Por fim, descrição dos trabalhos correlatos e estado da arte das áreas de estudo nos temas referentes aos conceitos apresentados.

2.1 Séries Temporais

Os dados de séries temporais (*time series data*) podem ser definidos como uma sequência de variáveis aleatórias, na qual uma determinada variável aleatória denota o valor obtido pela série em determinado ponto no tempo (*time point*). Por exemplo, dada a sequência de variáveis aleatórias $x1, x2, \ldots, xn$, tem-se que a variável aleatória $x1$ denota o valor obtido pela série no primeiro ponto de tempo, a variável $x2$ denota o valor para o segundo período de tempo, e assim por diante (WANG; WANG, 2019). Na Tabela 1 mostra-se um exemplo de série temporal.

Tabela 1 - Exemplo de série temporal

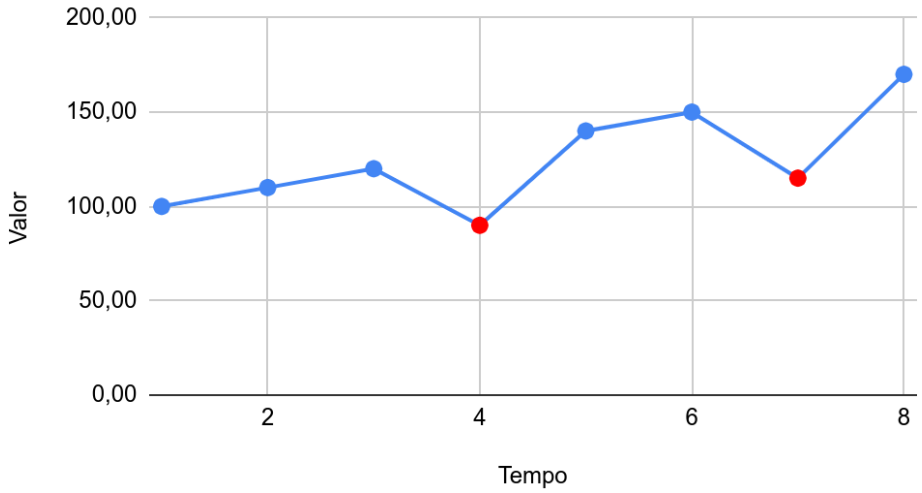
Tempo	Valor
t1	100,00
t2	110,00
t3	120,00
t4	130,00
t5	140,00
t6	150,00
t7	160,00
t8	170,00

Fonte: Elaborado pela autora

A utilização dos dados de séries temporais encontra-se em vasta expansão, de forma que esses são utilizados em diversas áreas, como economia, processamento de sinais, sistemas de geolocalização, mercado de ações, dentre outros (WANG; WANG, 2019), o que os torna um ativo de valor inestimável. Tais dados são coletados, principalmente, por meio de sensores, que,

geralmente, coletam dados em uma frequência de segundos, de modo que o volume de dados é significativo (WANG; WANG, 2019). Além disso, sabe-se que dados de sensores tendem a não ser confiáveis (JEFFERY et al, 2006), o que pode gerar uma série de dados com dados anômalos, o que prejudica a qualidade dos dados. Um exemplo de dados anômalos pode ser visualizado no gráfico apresentado na Figura 1, no qual os dados anômalos estão destacados na cor vermelha.

Figura 1 - Exemplo de dados anômalos



Fonte: Elaborado pela autora

Assim, garantir a qualidade dos dados representa desafios relevantes nesse campo, uma vez que os dados podem indicar autocorrelações, tendências, sazonalidade e lacunas (WANG; WANG, 2019; DASU; DUAN; SRIVASTAVA, 2016). Por esse motivo, se faz necessário o desenvolvimento de estratégias que contornem esses problemas, de modo a assegurar a integralidade da informação, além de dados limpos que não comprometam as posteriores análises.

2.2 Limpeza de dados

O processo de limpeza, ou depuração, de dados é uma das etapas da descoberta de conhecimento em uma base de dados — KDD, do inglês, *Knowledge Discovery in Databases* — e consiste na detecção e remoção de erros e inconsistências dos dados, visando elevar a qualidade dos mesmos. Este processo é importante para que seja possível obter acesso a dados precisos e consistentes (RAHM; DO, 2000).

Para o campo das séries temporais, existem diversas abordagens para este problema. Pode-se citar, dentre os algoritmos existentes, aqueles baseados em suavização, em estatísticas ou em restrições (WANG; WANG, 2019).

A técnica de suavização baseia-se, como o próprio nome diz, na suavização dos dados, isto é, eliminação de ruídos por meio da modificação dos mesmos. Deste modo, embora o tempo de processamento dos métodos baseados nessa abordagem não seja alto, durante a execução os dados originais podem sofrer grandes alterações, de modo a distorcê-los e tornar os resultados das análises realizadas incertos, de modo que esta, dentre as abordagens existentes, pode não ser a mais adequada (WANG; WANG, 2019). Como exemplo, pode-se mencionar o algoritmo *Moving Average* (BRILLINGER, 2001), cuja versão simplificada baseia-se no cálculo da média de uma parcela dos dados da série temporal, de modo a prever o valor da mesma em um determinado período de tempo. Ainda que a abordagem de suavização não seja a mais recomendada para a limpeza de dados de séries temporais, o *Moving Average* é amplamente utilizado.

Por outro lado, os algoritmos baseados em estatísticas estão alicerçados no treinamento de modelos com os dados para a posterior limpeza da base, de modo a ser um novo aspecto a ser observado sobre o processo de limpeza de dados (WANG; WANG, 2019). Como exemplo, tem-se o algoritmo *Maximum Likelihood*, que tem como conceito básico a realização de um teste aleatório baseado numa função de distribuição. Considerando a existência de diversas possibilidades de valores para determinado ponto da série temporal, o resultado do teste é considerado aquele com maior probabilidade de ocorrência, de modo a ser utilizado para a limpeza de dados com ruídos (WANG; WANG, 2019).

Tem-se ainda algoritmos baseados em restrição que são aqueles cujo processo de limpeza implica na utilização de algum critério restritivo. Dentre as restrições, pode-se citar as dependências de ordem ou de sequência, e a restrição de velocidade (WANG; WANG,2019). Para exemplificar, pode-se mencionar aquele proposto por Song et al. (2015), que se baseia em restrição de velocidade, de modo a considerá-la ao observar os valores em uma série temporal. Com o objetivo de determinar se um dado é ou não discrepante, um cálculo é realizado utilizando-se os valores da série temporal, cujo resultado é posteriormente comparado a valores pré-definidos de velocidade mínima e máxima.

Assim, dada uma série temporal $s = \{100, 110, 80, 130, 140\}$ e períodos de tempo $t = \{1,2,3,4,5\}$, pode-se considerar uma janela — isto é, o par de valores para as velocidades mínima e máxima — $V = 2$, velocidade mínima $v_{min} = -20$ e velocidade máxima $v_{max} =$

20, de modo a analisar os dados x_2 e x_3, tem-se: $\frac{80 - 110}{3-2} = -40 < -20$, de modo que a velocidade mínima não é respeitada. De forma similar, para os dados x_3 e x_4, tem-se: $\frac{130 - 80}{4-3} = 50 > 20$, e a velocidade máxima também não é respeitada. Deste modo, sabe-se que o dado x_3 possui um valor discrepante que precisa ser corrigido, o que pode ser feito de modo a encontrar um valor que respeite as restrições de velocidade. Neste exemplo, um valor que poderia satisfazer as condições seria x_3= 120.

2.2.1 Limpeza de dados em séries temporais

Há na literatura, conforme mencionado, diversas técnicas para a realização da limpeza de dados em séries temporais. De modo a entender melhor quais os desafios enfrentados para a realização desse processo, Wang e Wang (2019) listam as quatro maiores dificuldades neste campo: a quantidade de dados é grande e possui alta taxa de erro, os erros ocorrem por razões diversas e complexas, os dados originais podem sofrer grandes alterações e a geração e armazenamento de dados ocorre de forma contínua.

As informações que compõem as séries são coletadas, normalmente, por meio de sensores, o que gera um volume elevado de dados de forma contínua, muitos deles com ruídos. Para os erros com naturezas diversas e complexas, pode-se citar aqueles provocados por ambientes complexos, como os decorrentes da Internet das Coisas — IOT, do inglês, *Internet of Things* — que Karkouch et al. (2016) explica em detalhes.

Muitos procedimentos de limpeza utilizam o princípio de filtragem suave, o que pode gerar grandes alterações nos dados originais, ocasionando perda de informações. Assim, a limpeza deve se basear no princípio de modificação mínima (AFRATI; KOLAITIS, 2009; CHOMICKI; MARCINKOWSKI, 2005; FAGIN; KIMELFELD; KOLAITIS, 2015), quanto menos divergências entre os dados originais e os tratados, menor será a perda de informações.

De modo a considerar as limitações dos métodos existentes, bem como os possíveis erros nos dados, Wang e Wang (2019) sugerem que se pode utilizar a detecção de *outliers* antes de um algoritmo para a limpeza propriamente dita, de modo a separar os dados discrepantes dos demais, facilitando a identificação do erro e sua posterior correção.

2.3 Detecção de *outliers*

Há muitas definições na literatura para um *outlier*. Como exemplo, pode-se mencionar as definições de Hawkins (1980) e de Barnett e Lewis (1994). O primeiro explica *outlier* como

sendo uma observação que se desvia acentuadamente de outras observações, podendo indicar a geração deste dado por um mecanismo diferente. Já os outros dois definem como sendo uma observação que foge ao padrão identificado no restante do conjunto de dados analisados.

A detecção de *outliers* é um passo primário em muitas aplicações do cenário *Big Data* e baseia-se em encontrar, dentre uma gama de dados, aqueles que fogem do padrão estabelecido, e apesar de serem frequentemente considerados erros, dados anômalos, ou discrepantes, podem carregar informações importantes, de modo que podem indicar falhas no modelo, parâmetros enviesados ou resultados incorretos (BEN-GAL, 2005).

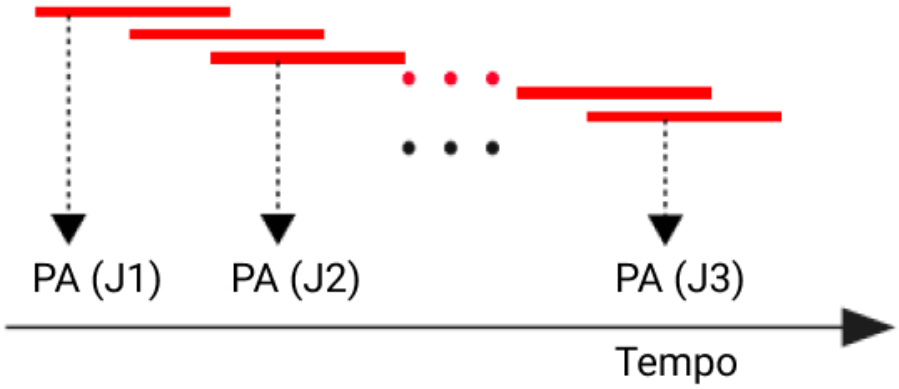
Na área de estatística, o estudo sobre detecção de *outliers* é de suma importância e vem sendo realizado há décadas. Porém, este tópico abrange ainda outros campos, como por exemplo o campo industrial, e pode ser utilizado em diversos tipos de dados (GUPTA et al., 2013) como dados de alta dimensão (AGGARWAL; YU, 2001), dados incertos (AGGARWAL; YU, 2008) e dados de séries temporais, de modo que o trabalho de Fox (1972) foi o primeiro realizado para este último tipo de dado.

Para os dados de séries temporais, como mencionado anteriormente, Gupta et al. (2013) explora dois tipos de detecção: identificar pontos como *outliers* e subsequências como *outliers*, as quais podem ser feitas de modo a detectar *outliers* em bancos de dados de séries temporais ou em uma determinada série temporal.

A detecção para pontos únicos como *outliers* é feita, normalmente, com a utilização de algoritmos de predição, isto é, dado o valor predito para um determinado ponto da série temporal no modelo estabelecido, compara-se os valores observados para cada ponto da série, posteriormente, comparando a diferença entre eles a um limiar pré-determinado. Se esta for maior que o limiar, considera-se o dado anômalo (WANG; WANG, 2019).

Contudo, para a detecção de subsequências de uma série temporal como uma sequência de *outliers*, pode-se lidar da seguinte forma: primeiro, realiza-se a divisão em janelas, isto é, subdivide-se a série em múltiplas subsequências que se sobrepõem, para as quais, posteriormente, calcula-se uma pontuação para determinar o quão anômala é cada subsequência. Após, a partir dos resultados dos cálculos anteriores, determina-se o quão anômala é a série que está sendo analisada (WANG; WANG, 2019). Um exemplo para esta detecção pode ser visto na Figura 2. Considera-se PA como Pontuação Anômala, resultado do cálculo para cada janela, e J como janela.

Figura 2: Exemplo de detecção de subsequências como *outliers*.



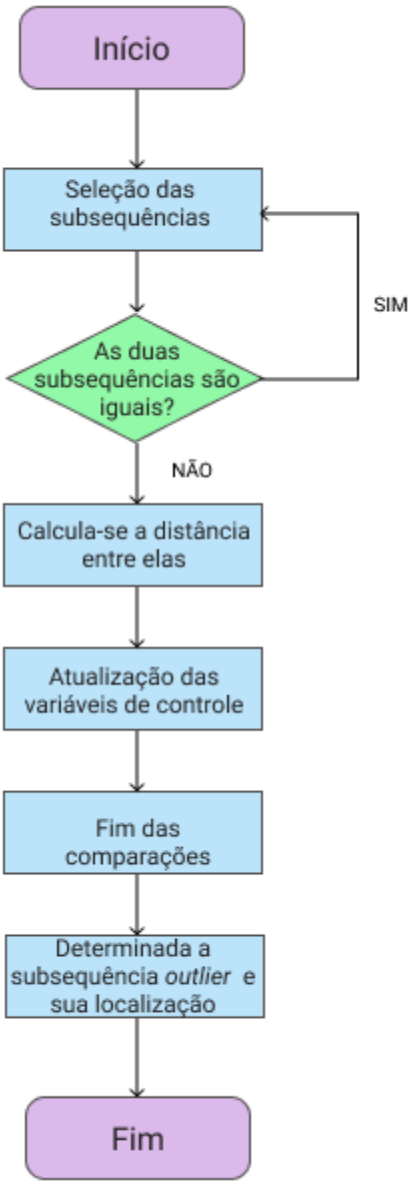
Fonte: Adaptado de Wang e Wang (2019).

2.3.1 Algoritmo baseado em "força bruta"

Para as duas categorias de detecção de *outliers* analisadas por Gupta et al. (2013) existem diversos algoritmos propostos. Dentre eles, pode-se citar o proposto por Keogh et al. (2006) que visa identificar dentre uma sequência de dados uma subsequência discrepante. Este busca identificar subsequências como *outliers* em séries temporais.

A lógica do algoritmo é muito simples. A ideia é separar previamente a série temporal em subsequências de dados e, a partir disso, comparar todas as subsequências, de modo a realizar todas as combinações possíveis, para determinar qual a subsequência é a mais distante das demais e qual a sua localização na série temporal. Na figura 3 está ilustrado o fluxograma do algoritmo proposto por Keogh et al. (2006).

Figura 3 - Fluxograma do algoritmo original.



Fonte: Elaborado pela autora.

Embora seja a forma intuitiva de realizar o processo de identificar subsequências como *outliers*, esta não é a mais efetiva. O algoritmo, da forma proposta, possui complexidade $\boldsymbol{O(n^2)}$, advinda das comparações das subsequências umas com as outras, o que implica em alto custo computacional para realizar a sua execução, além de elevado tempo de processamento, fatores que tornam inviável a sua utilização em cenários reais. Para um maior detalhamento do algoritmo, consultar Apêndice A.

2.4 Escalabilidade

O escalonamento consiste na capacidade do sistema de se adequar de modo a conciliar demandas crescentes de processamento (SINGH; REDDY, 2015) comuns em aplicações do cenário *Big Data*. Tal processo pode ser realizado de duas formas: vertical ou horizontalmente, detalhadas no trabalho de Singh e Reddy (2015).

A escalabilidade horizontal, também conhecida como *scale out*, baseia-se na distribuição da carga de trabalho em diversos servidores, na qual máquinas são acrescentadas para aumentar a capacidade de processamento do sistema. Geralmente, nesse cenário, diversas instâncias do sistema operacional estão sendo executadas em diferentes máquinas (SINGH; REDDY, 2015).

Por outro lado, a escalabilidade vertical, também chamada *scale up*, implica no aumento do número de processadores e memória, além de um hardware mais rápido, normalmente, em um único servidor. Também costuma envolver apenas uma única instância de um sistema operacional (SINGH; REDDY, 2015).

Existem diversas formas para realizar as escalabilidades horizontal e vertical. Para a escalabilidade horizontal, pode-se citar soluções como redes *peer-to-peer* e plataformas como Apache Hadoop e Apache Spark. Já para a escalabilidade vertical, tem-se clusters de computação de alto desempenho (HPC) — do inglês, *High Performance Computing Clusters* —, processadores com vários núcleos (*multicore*), Unidade de Processamento Gráfico (GPU) — do inglês, *Graphics Processing Unit* —, entre outras (SINGH; REDDY, 2015).

Em termos de escalabilidade propriamente dita, o escalonamento vertical possui limitações. No caso do HPC, a implantação de muitos núcleos torna uma possível escalabilidade custosa. Há também um limite para a quantidade de GPUs que podem ser adicionadas a uma única máquina, de forma a ocasionar um gargalo na transferência de dados (SINGH; REDDY, 2015). Assim, as plataformas de escalonamento horizontal, mostram-se mais adequadas para sistemas que precisam comportar uma demanda crescente de processamento de dados.

2.4.1 Apache Spark

O Apache Spark é descrito em sua plataforma como “um mecanismo de análise unificado para processamento de dados em grande escala” (APACHE SPARK, 2021) e foi desenvolvido na AMPLab da Universidade da Califórnia de Berkeley, em 2009 (DATABRICKS, 2021). Este é uma alternativa ao Apache Hadoop, de modo a superar as

limitações de entrada e saída do disco e com desempenho superior aos sistemas anteriores (SINGH; REDDY, 2015).

Seu recurso mais notável é a capacidade de realizar cálculos na memória, permitindo que os dados sejam armazenados em *cache*, o que impede a sobrecarga de disco para tarefas iterativas, presente no Hadoop (SINGH, REDDY, 2015), além de ser capaz de acelerar a execução de uma aplicação em cem vezes utilizando a memória e em dez vezes, o disco em comparação ao Hadoop MapReduce (DATABRICKS, 2021).

O Spark fornece suporte a mais técnicas de processamento de dados, a exemplo de consultas do tipo *Structured Query Language* (SQL) — traduzido livremente como Linguagem de Consulta Estruturada — e processamento de fluxo de dados, além de estender o modelo MapReduce e possuir interfaces de programação de aplicações — APIs, do inglês *Application Programming Interface* — em Python, Java, Scala e SQL, de modo a possuir ainda muitas bibliotecas agregadas (MAVRIDIS; KARATZA, 2017).

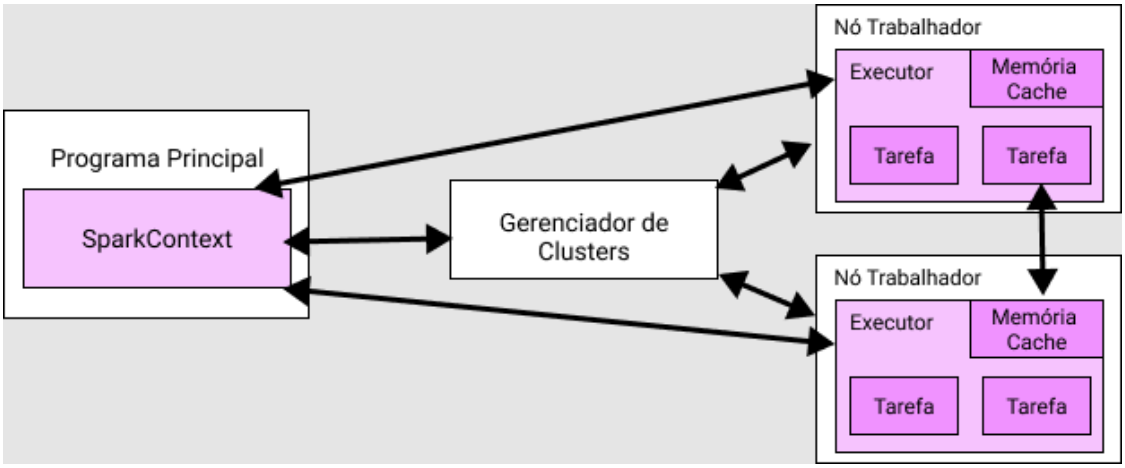
Dentre as bibliotecas encontradas no Spark, pode-se mencionar SQL e DataFrame, as quais permitem a consulta de dados estruturados, por meio de SQL ou uma API DataFrame, além de possibilitar o acesso a diferentes fontes de dados — como JSON, Parquet, entre outras — e a integração das mesmas. Ademais, inclui também métodos de armazenamento e geração de código que permitem consultas mais rápidas, e é altamente escalável (APACHE SPARK, 2021). Tem-se também a MLlib, adequada para aprendizado de máquina, que pode ser utilizada em APIs e interopera com a biblioteca NumPy em Python e bibliotecas R. Além disso, é possível utilizar quaisquer fontes de dados Hadoop, de modo a tornar a conexão com projetos Hadoop mais fácil (APACHE SPARK, 2021). Pode-se ainda citar a GraphX, que é uma API utilizada para grafos e computação paralela baseada em grafos, e Spark Streaming, que permite construir aplicações *streaming* escalonáveis e com alta tolerância a falhas (APACHE SPARK, 2021).

De modo a entender melhor o Spark, é necessário entender o conceito de *Dataset,* que é a abstração primária do Spark. Este é um repositório distribuído de itens que podem ser criados, por meio de algumas ações, seja com o *Hadoop Distributed File System* (HDFS), ou a partir da transformação de outros conjuntos de dados, de modo que é possível encadear ações e transformações (APACHE SPARK, 2021). Também é importante destacar o recurso que contempla o acesso repetido a um mesmo conjunto de dados, que é o suporte à extração de *datasets* em *cache* na memória de todo o cluster (APACHE SPARK, 2021).

A arquitetura de uma aplicação feita com o Spark, normalmente, é constituída por um

programa principal que comporta o SparkContext — objeto que coordena os conjuntos de processos compõem a aplicação — um gerenciador de clusters — que alocam os recursos para as aplicações — e nós trabalhadores, de modo que este último item possui um executor, atribuído a ele pela aplicação. Este possui uma memória *cache* e é responsável por executar as tarefas, enviadas pelo SparkContext. Na Figura 4 está ilustrada a arquitetura de uma aplicação Spark (APACHE SPARK, 2021).

Figura 4 - Arquitetura de uma aplicação Apache Spark



Fonte: Adaptado de Apache Spark (2021).

O Spark pode, ainda, ser executado utilizando diversos gerenciadores de cluster, como, por exemplo, por meio de seu modo de cluster autônomo (*standalone cluster*), Hadoop YARN — no qual o gerenciador de recursos e monitoramento do Hadoop são utilizados —, Kubernetes — sistema baseado em contêineres que permite a automação de implantação, escalonamento e gerenciamento de aplicações (KUBERNETES, 2021) —, entre outros. Ressalta-se que em seu modo autônomo, o gerenciador de recursos nativo fornece meios para inicializar o programa principal e os nós trabalhadores por meio de *scripts*, sendo possível executá-los em uma única máquina para testes (APACHE SPARK, 2021).

2.5 Trabalhos correlatos e estado da arte

Na literatura, é possível trabalhos que contemplam a etapa de limpeza de dados e detecção de *outliers* em dados de séries temporais.

No campo de detecção de *outliers*, um trabalho recente a ser mencionado é o realizado por Duggimpudi et al. (2019). Este propõe três algoritmos voltados para a detecção de *outliers*, cuja utilização é adequada para dados de séries temporais. O primeiro é o *Spatio-Temporal*

Behavioral Outlier Factor (ST-BOF) — traduzido livremente como Fator de Anomalia Comportamental Espaço-Temporal. Este mede o grau de anomalia de um objeto considerando apenas os seus atributos comportamentais, de modo que os atributos espaço-temporais são utilizados como atributos contextuais, isto é, para definir a vizinhança. O segundo é o S*patio-Temporal Behavioral Density Based Clustering of Applications with Noise* (ST-BDBCAN) — traduzido livremente como Agrupamento de Aplicações com Ruído Baseado em Densidade Comportamental Espaço-Temporal. Sua execução consiste em identificar *outliers* espaço-temporais e agrupamentos comportamentais espaço-temporais, de modo que o algoritmo agrupa objetos com atributos semelhantes, o que possibilita a identificação de objetos com atributos comportamentais anômalos, como valores discrepantes. Por último, é proposto o algoritmo Approx-ST-BDBCAN. No aspecto conceitual, este é igual ao ST-BDBCAN, excetuando-se o fato de que o Approx-ST-BDBCAN particiona os dados em blocos sobrepostos, de modo que cada um pode ser executado separada e paralelamente.

Ainda, pode-se citar o trabalho de Li et al. (2019). Com o avanço das tecnologias baseadas em aprendizado de máquina — do inglês, *machine learning* —, tem-se a utilização de *Generative Adversarial Networks* — GAN, traduzido livremente como "redes adversárias generativas". Li et al. (2019) propõem o uso dessa tecnologia para a detecção de dados anômalos em dados de séries temporais. As GANs, por meio do treinamento de dois modelos, capturam a distribuição dos dados e os classifica como dados reais ou falsos. Com a utilização de uma variável aleatória, gera-se uma distribuição nova para estes dados, de forma que a distribuição real capturada e a gerada são comparadas, a fim de otimizar a rede. O processo é feito seguindo alguns passos: gera-se uma entrada distribuída de maneira uniforme, processa-se então essa entrada com a rede, de modo a coletar a saída gerada. Assim, compara-se a distribuição coletada com a gerada anteriormente, e, por fim, utiliza-se os conceitos de retropropagação e gradiente descendente para calcular erros e atualizar os pesos dos dados. Esse processo tem por objetivo otimizar o processo de treinamento dos modelos de captura de distribuição dos dados e de classificação mencionados.

O conceito de GANs está presente também nos trabalhos relacionados à limpeza de dados. Neste campo, pode-se citar os trabalhos realizados por Sun et al. (2018) e Fang et al. (2019). Os métodos propostos pelos autores baseiam-se em supor que o dado anômalo não existe, está ausente na base, e repará-lo. Sun et al. (2018) aplicam isso em um modelo simples, de forma a utilizar uma base composta por dados sobre vagas em um estacionamento. Primeiro, é realizada a análise de semelhanças entre dados de vagas em estacionamentos e dados de

estacionamentos, e então GANs são utilizadas para gerar dados de estacionamento, como dados para substituir os dados anômalos. Tal processo pode ser utilizado, por meio das devidas alterações, para tratar de dados em séries temporais. No trabalho de Fang et al. (2019), é proposto o FuelNet, um algoritmo que é utilizado para corrigir dados que sejam inconsistentes numa base que trata dados sobre consumo de combustível, além de classificar os dados incompletos. Este algoritmo é baseado em Redes Neurais Convolucionais — CNNs, do inglês *Convolutional Neural Networks* —, também muito utilizadas no campo de aprendizado de máquina, e GANs.

Para a limpeza de dados, pode-se citar também o trabalho de Milani et al. (2019) que propõem o *Spatio-Temporal Probabilistic Model* (STPM) — traduzido livremente como "Modelo Probabilístico Espaço-Temporal". Este estende uma classe de modelos gráficos que visa capturar correlações espaciais e temporais entre atualizações baseando-se no princípio da localidade — também chamada Localidade de Referência (DENNING, 2006). Considera-se que células atualizadas recentemente — localidade temporal — possivelmente serão atualizadas novamente em um futuro próximo. De forma similar, células próximas às que foram atualizadas, provavelmente, também serão atualizadas — localidade espacial. A partir deste estabelecimento de padrões de atualização, o algoritmo é capaz de detectar e reparar os *outliers*.

3 Desenvolvimento

Conforme detalhado na seção 1.3, foram realizados estudos que contemplam o funcionamento do algoritmo, bem como das ferramentas e conceitos utilizados para a elaboração do projeto proposto. As etapas pertinentes à seleção de bases de dados, implementação do algoritmo no qual o projeto foi baseado, bem como o desenvolvimento do projeto em si, serão descritas nas subseções subsequentes.

3.1 Seleção das bases de dados e implementação do algoritmo de base

Durante a implementação do algoritmo de base, assim como na implementação por meio da ferramenta Apache Spark, utilizou-se uma base com dados mais genéricos e com uma volumetria menor, a fim de atestar as funcionalidades desenvolvidas em ambas as situações. A base em questão contemplava dados sobre produção elétrica analisada durante três anos (KAGGLE, 2018), de forma que esta possuía a data de produção e a volumetria produzida.

A fim de possibilitar a medição do tempo de processamento do algoritmo com diversas volumetrias de dados, a base de dados inicialmente utilizada foi substituída por outras mais robustas. Para este propósito, utilizou-se bases que compreendem dados pertinentes à variação nos preços de ações de importantes empresas do mercado atual, que possui informações como o preço das ações ao serem disponibilizadas, as altas e baixas dos valores e as datas em que esses dados foram observados (KAGGLE, 2018).

Ressalta-se que todas as bases utilizadas possuem dados públicos e disponibilizados gratuitamente pela plataforma Kaggle.

Inicialmente, o processo de implementação iniciou-se com a execução do algoritmo base, de forma a seguir o proposto por Keogh et al. (2006). Para isso, a linguagem escolhida foi Python, tal escolha baseou-se no fato de esta ser uma linguagem que permite a manipulação de dados objetiva e efetivamente, de modo a permitir a integração e elaboração de sistemas de forma simplificada (PYTHON, 2021). Utilizou-se também o Google Colaboratory (GOOGLE, 2021), ferramenta disponibilizada gratuitamente a todos os usuários das contas Google, que possibilita a implementação de códigos na linguagem Python, sem a necessidade de

configurações específicas, além de fornecer acesso gratuito a GPUs e o compartilhamento de dados de forma facilitada.

Para a validação do algoritmo, foi realizado um pré-processamento na base de dados, que consiste na divisão das séries temporais em diversas subsequências, de modo a adequar a entrada àquela exigida.

3.2 Implementação do algoritmo utilizando Apache Spark

A implementação por meio da ferramenta Apache Spark iniciou-se com a integração da API Pyspark, fornecida pela mesma, ao ambiente de desenvolvimento Google Colaboratory.

Após a integração, foi realizada a etapa de leitura e pré-processamento dos dados. De modo a possibilitar a utilização de clusters, a ferramenta Apache Spark, disponibiliza um formato de dados específico, conhecido como RDD — *Resilient Distributed Datasets*, traduzido livremente como "conjuntos resilientes de dados distribuídos". Para a inicial leitura dos dados, estes foram lidos utilizando RDD, porém foram posteriormente convertidos para *dataframes* — traduzido livremente como "quadros de dados" —, que permitem o armazenamento de dados no formato de tabela, de modo a facilitar o acesso e manipulação algébrica dos mesmos, essencial para a execução do algoritmo.

Após a leitura, o pré-processamento foi realizado em duas etapas. Primeiramente, realizou-se uma triagem que permitiu que dados corrompidos — isto é, armazenados de forma incorreta ou incompletos —, fossem eliminados, uma vez que o algoritmo base não foi construído para lidar com tais dados e os testes utilizando-os poderiam gerar resultados enviesados ou discrepantes, diferentes dos obtidos com o algoritmo base, de forma que a integralidade da informação e a confiabilidade dos dados não poderiam ser garantidas. Depois, dividiu-se a série temporal, isto é, os dados da base escolhida, em várias subséries de tamanhos iguais, armazenando-as em estruturas distintas que, posteriormente, compuseram uma lista de subséries, de forma a ser a entrada adequada para o algoritmo.

De modo a possibilitar o processamento em paralelo, além dos RDDs, utilizou-se o conjunto disponível de bibliotecas fornecidas pela ferramenta Apache Spark, mais especificamente as da API Pyspark, por meio das quais, foi viável utilizar comandos SQL no fluxo de execução, em conjunto com a linguagem Python, novamente visando facilitar a manipulação dos dados.

Uma vez que os dados pré-processados por meio dos recursos do Pyspark já possibilitam e facilitam o processamento paralelo, a construção do algoritmo seguiu um modelo similar ao

original. Assim, iniciou-se a implementação pelo *core* do algoritmo, isto é, construiu-se as etapas de seleção de subsequências de séries temporais. Visto que a comparação deve ser feita entre sequências distintas, houve a implementação de um tratamento para evitar que o algoritmo atuasse sobre duas sequências iguais.

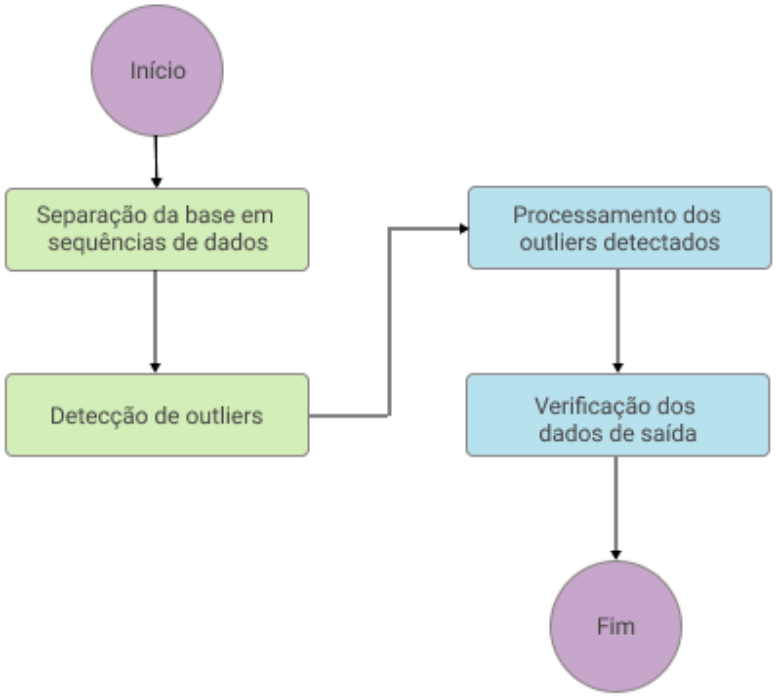
É relevante ressaltar que a detecção de *outliers* realizada pelo algoritmo baseia-se no cálculo da distância entre as subsequências componentes da série original. Para este fim, utilizou-se a distância euclidiana, também utilizada por Keogh et al. (2006), de forma que as principais características do algoritmo original fossem mantidas, de modo a obter-se resultados mais acurados em relação às melhorias implementadas. Para o cálculo, considera-se duas subsequências C e Q, ambas com tamanho n. A distância é calculada a partir da diferença ao quadrado de cada elemento das subsequências, qi e ci, soma-se então todas as diferenças e, por fim, calcula-se a raiz quadrada deste valor. A fórmula para o cálculo da distância euclidiana pode ser expressa como na Equação 1 a seguir, retirada de Keogh et al. (2006):

$$\mathrm{Dist}(\mathrm{Q}, \mathrm{C}) = \sqrt{\sum_{i=1}^{n} (q_i - c_i)^2} \quad (1)$$

Por fim, implementou-se a função *main* do algoritmo, por meio da qual o algoritmo foi alimentado e executado com os dados pré-processados, e obteve-se os seguintes resultados: a localização da sequência *outlier* dentro da base de dados, a distância desta para as demais subsequências e a subsequência *outlier* em forma de *dataframe*. Tais saídas do algoritmo compõem as etapas de "processamento dos dados" e "verificação dos dados", que compreendem a comparação dos resultados do algoritmo aos do algoritmo original. Destaca-se que, para possibilitar a comparação dos tempos de processamento, foi acrescentado ao código esse cálculo.

Na figura 5 está ilustrado o fluxograma do algoritmo, pertinente às etapas de detecção de *outliers* e processamento posterior dos dados.

Figura 5 - Fluxograma do algoritmo apenas com a detecção de outliers.



Fonte: Elaborado pela autora.

3.3 Seleção e implementação do algoritmo de limpeza de dados

Uma vez que a subsequência de *outliers* tenha sido encontrada, é necessário que os dados presentes na mesma sejam tratados, isto é, passem pelo processo de *data cleaning*. Para este propósito, há uma gama de algoritmos que podem ser escolhidos, conforme citado anteriormente.

Nas seções 2.2 e 2.5 foram discutidos alguns dos algoritmos presentes na literatura e, dentre estes, o algoritmo *Moving Average*. Conforme destacado na seção 2.2, este algoritmo, embora amplamente utilizado, não é o mais adequado em tratando-se de limpeza de dados em séries temporais. No entanto, assim como mencionado na seção 2.5, alguns estudos indicaram adaptações deste algoritmo que são igualmente eficazes e que não possuem as desvantagens do mesmo, como é o caso do *Weighted Moving Average* (ZHUANG et al., 2007) — WMA, traduzido livremente como "média móvel ponderada" — e do *Exponential Weighted Moving Average* (GARDNER JR., 2006) — EWMA, traduzido livremente como "média móvel exponencial".

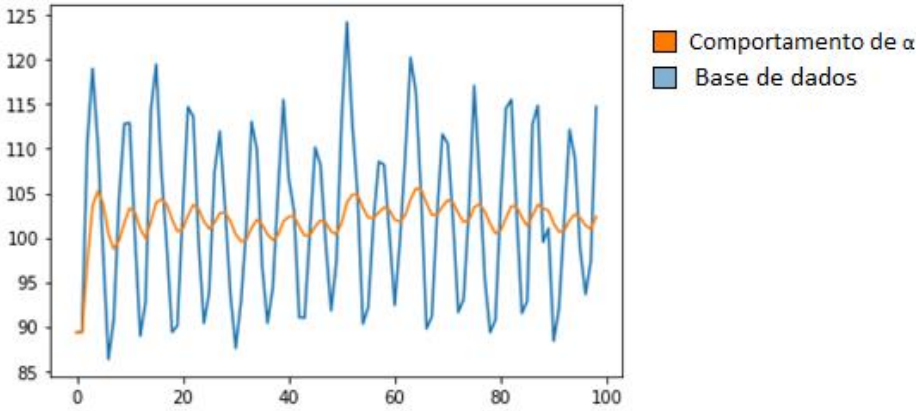
Após uma análise cuidadosa dos algoritmos existentes na literatura, optou-se por utilizar o EWMA neste projeto. O funcionamento deste algoritmo é semelhante ao do WMA, a diferença principal baseia-se no critério para a atribuição de pesos a cada dado da série temporal. O peso conferido a cada um depende exclusivamente do quão antigo o dado é, isto é, a data de sua ocorrência, uma vez que os dados mais recentes recebem pesos maiores que os mais antigos, de forma a estabelecer uma maior relevância para estes nos cálculos. Tal abordagem permite que o algoritmo seja mais responsivo às tendências observadas nos dados que a versão mais simples do *Moving Average*.

A limpeza com este algoritmo é similar às realizadas por outros algoritmos utilizados para a limpeza e predição de dados. Baseado em observações anteriores, este realiza um cálculo para prever qual o próximo valor da série temporal. Dentro do escopo deste trabalho, tal método foi utilizado da seguinte maneira: uma vez que a subsequência *outlier* foi encontrada, os valores presentes nesta foram processados pelo algoritmo de limpeza, de modo que foram encontrados valores mais adequados para substituir os dados discrepantes, por meio dos demais dados da série, de modo a prever os valores apropriados, e, assim, realizar a limpeza na série temporal original. O EWMA é um algoritmo recursivo, cuja formulação matemática pode ser expressa como na Equação 2 a seguir:

$$EWMA = \alpha \times r_t + (1 - \alpha) \times EWMA_{t-1} \quad (2)$$

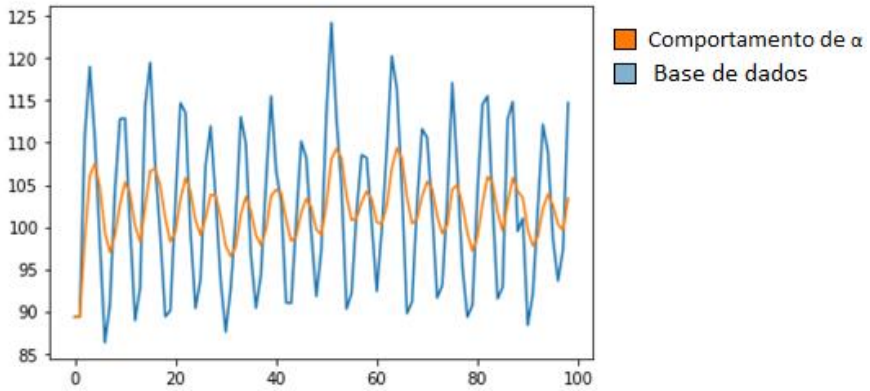
Os parâmetros requeridos para realizar a limpeza dos dados, são r e α. O primeiro é o valor da série temporal no período t. Já o segundo, é definido pelo analista que está utilizando o algoritmo, e deve estar dentro do intervalo $0 < \alpha < 1$, de modo que quanto mais próximo de 1, mais semelhante à série original serão os resultados obtidos. A escolha deste fator está diretamente ligada ao objetivo de sua utilização, pois este é determinante para o grau de suavização sofrido pelos dados. Nas Figuras 6, 7, 8 e 9 a seguir, está ilustrado o comportamento dos dados em relação aos dados originais de uma das bases selecionadas para $\alpha = 0.10$, $\alpha = 0.25$, $\alpha = 0.50$ e $\alpha = 0.75$, respectivamente. Destaca-se que a linha laranja corresponde ao comportamento de *alpha* enquanto a azul refere-se à base de dados.

Figura 6 - EWMA com α = 0.10.



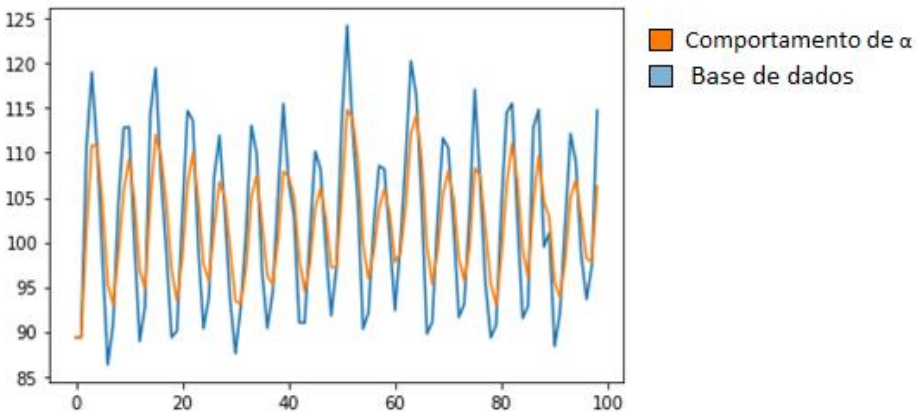
Fonte: Elaborado pela autora.

Figura 7 - EWMA com α = 0.25



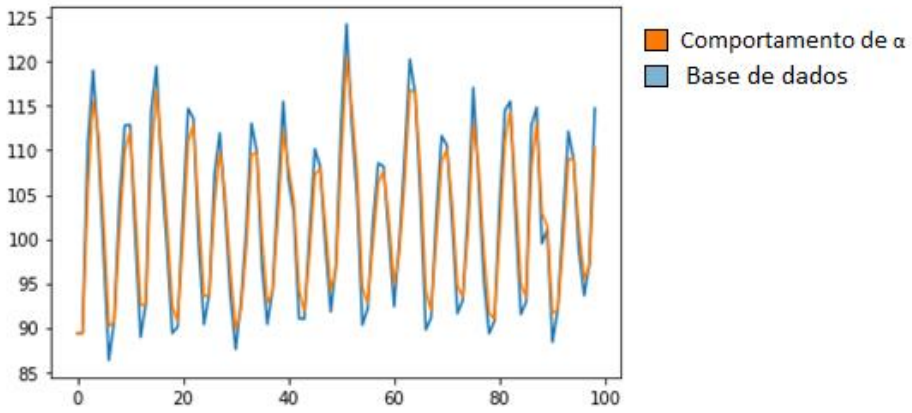
Fonte: Elaborado pela autora.

Figura 8 - EWMA com = 0.50.



Fonte: Elaborado pela autora

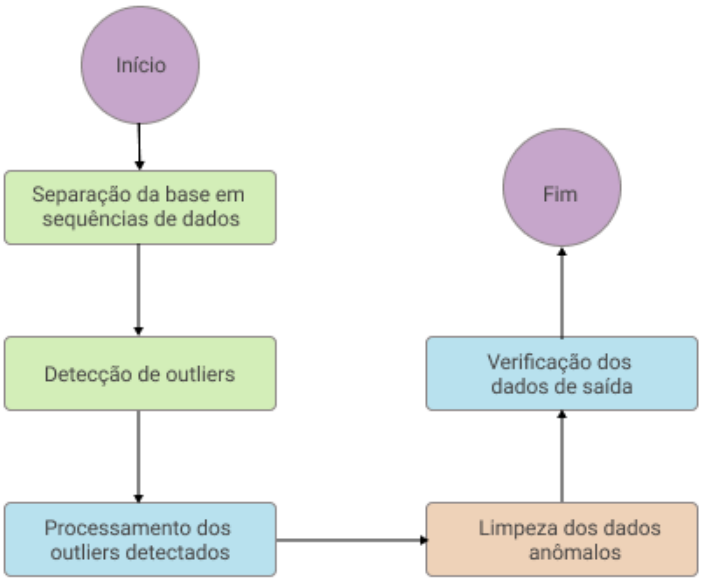
Figura 9 - EWMA com = 0.75.



Fonte: Elaborado pela autora.

A implementação do algoritmo foi feita por meio da biblioteca Pandas, presente na linguagem Python, através do método Pandas.Series.ewm(). Este permitiu a utilização do algoritmo de forma eficaz e para diversos valores de *alpha*, conforme mostrado anteriormente. Na figura 10 a seguir está ilustrado o fluxograma do algoritmo completo, de modo que abrange as etapas de detecção de *outliers*, processamento e limpeza dos dados.

Figura 10 - Fluxograma do algoritmo completo.



Fonte: Elaborado pela autora.

4 Testes e resultados

A partir da execução dos algoritmos descritos no capítulo 3, foi possível observar se as melhorias implantadas foram efetivas. O modo como os testes foram conduzidos, bem como os resultados obtidos a partir dos mesmos, estão descritos nas subseções subsequentes.

4.1 Realização dos testes

Os testes foram conduzidos utilizando seis bases de dados, de forma que a primeira possui uma volumetria menor e menos atributos e as demais são mais volumosas e possuem dados sobre o mercado de ações para empresas distintas, contemplando dados como a variação dos preços das ações, isto é, o preço das mesmas ao entrarem no mercado, suas altas, baixas e a data nos quais esses eventos foram verificados.

A exceção do primeiro teste, que utilizou a menor base apenas para verificar se tanto o algoritmo de base quanto o desenvolvido para o trabalho funcionavam plenamente, estes basearam-se na medição do tempo de execução necessário para ambos os algoritmos, de forma a verificar se houve uma melhora significativa após a realização das melhorias propostas, além da verificação dos resultados obtidos, isto é, se os algoritmos alcançaram os mesmo resultados quando executados e se os dados sofreram quaisquer distorções.

Ressalta-se que a volumetria de cada base é diferente, de modo que a cada teste realizado, uma volumetria maior que a anterior foi considerada. A fim de simplificar a identificação de cada uma das bases selecionadas para os testes de tempo, estas serão chamadas de bases A, B, C e D, apelidos adotados até o fim deste documento. Na tabela 2 estão contidas as volumetrias das bases.

Tabela 2 - Volumetria das bases de dados utilizadas.

Base de dados	Registros
A	3052
B	7700
C	14991
D	29891

Fonte: Elaborada pela autora.

Destaca-se que os testes para ambos os algoritmos foram realizados considerando apenas o tempo de execução para a etapa de detecção de *outliers*, de modo a realizar uma avaliação imparcial, focada apenas nas melhorias propostas. Assim, as etapas de pré-processamento não foram consideradas para estes. Ainda, cada teste foi realizado cinco vezes, de forma a garantir uma amostragem estatisticamente adequada. Os valores apresentados na seção a seguir foram alcançados a partir das médias desses testes.

Conforme mencionado na subseção 3.2.1, o cálculo do tempo de execução foi integrado aos algoritmos, de modo que a medição fosse mais acurada, assim, não houve quaisquer interferências externas nos resultados obtidos.

Ressalta-se que a comparação entre as saídas adquiridas com a execução dos algoritmos também foi integrada ao código, de forma que cada valor obtido com um dos algoritmos — isto é, a localização da sequência *outlier* dentro da base de dados, a distância desta para as demais subsequências e a subsequência *outlier* — foi comparado individualmente ao seu equivalente, obtido com o outro.

Os testes concernentes às etapas de análise de resultados e limpeza dos dados basearam-se na observação dos dados obtidos após o processamento realizado pelo algoritmo.

4.2 Resultados obtidos

Conforme mencionado anteriormente, os testes foram conduzidos para cinco bases diferentes, porém, considerou-se para os testes de tempo de execução somente quatro.

Na tabela 3 estão contidos os tempos de execução obtidos para cada base de dados, bem como o valor em porcentagem a diminuição do tempo de processamento do algoritmo paralelo para o algoritmo original.

Tabela 3 - Tempos de execução dos algoritmos.

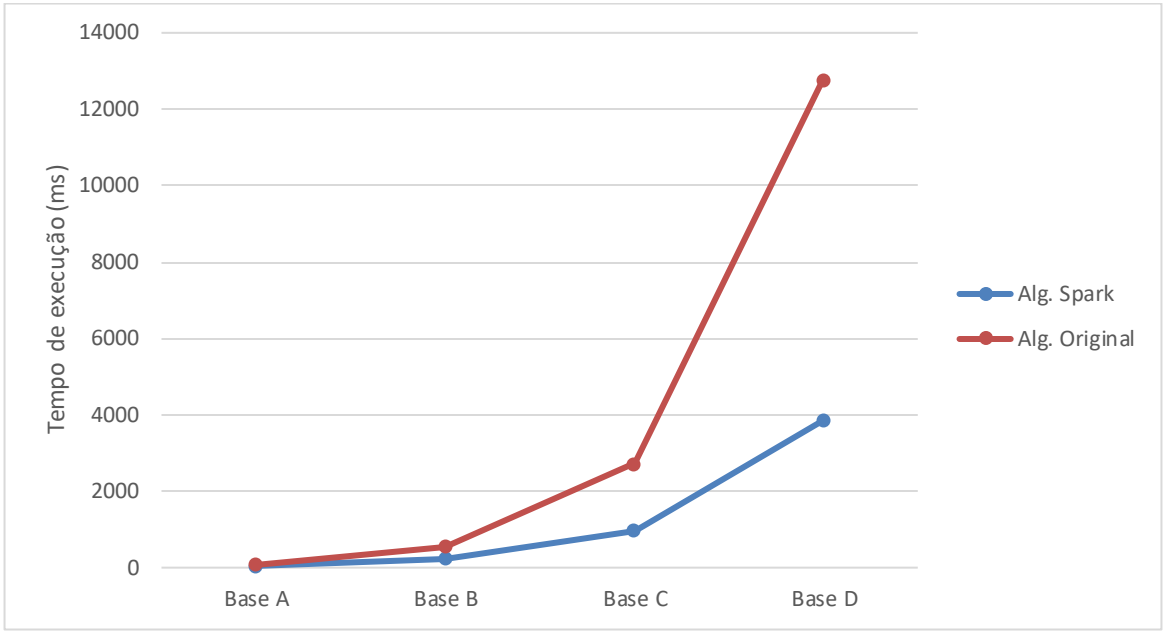
	Tempo de execução dos algoritmos (ms)		
Registros	**Algoritmo original**	**Algoritmo paralelo**	**% de melhoria no tempo**
3052	70,23239	45,55417	35
7700	563,55024	237,60009	58
14991	2731,11081	982,12386	64
29891	12790,06624	3853,83201	70

Fonte: Elaborada pela autora.

De modo a considerar que a volumetria de cada base testada é maior que a da base anterior, pode-se observar que quanto maior a quantidade de dados, melhor o desempenho do algoritmo em relação ao original. Assim, embora nos primeiros testes a melhoria tenha sido de

apenas 35%, tem-se que na maior das bases de dados, a base D, os testes com os algoritmos indicaram que o tempo utilizado para a execução do algoritmo paralelo corresponde a 30% do tempo necessário para a execução do algoritmo base, ou como indicado na tabela, uma redução de 70%, o que representa uma execução 3,3 vezes mais rápida que a do algoritmo original. Ainda, a tendência observada indica que esse tempo tem potencial para melhorar utilizando-se bases ainda mais robustas. Na figura 11 está ilustrada a melhora do algoritmo a cada teste realizado.

Figura 11 - Tempo de processamento dos algoritmos para todas as bases de dados.



Fonte: Elaborado pela autora.

A partir dos testes e das etapas de processamento e verificação de dados presentes no algoritmo, foi possível observar que os dados mantiveram a sua integralidade, de forma que os resultados obtidos por ambos os algoritmos foram os mesmos. Ainda, a limpeza realizada foi capaz de diminuir a distância entre a subsequência indicada como anômala e o restante da série original.

5 Conclusão

De modo a considerar o cenário *Big Data* com a geração, manipulação e armazenamento de elevados volumes de dados, além da relevância das séries temporais no mercado atual, alguns desafios concernentes ao campo que intersecta esses dois conceitos precisam ser superados. Além da grande volumetria de dados, há o problema da existência de dados que dificultam as análises e a utilização das informações contidas nessas bases, que é o caso dos *outliers* — também chamados dados anômalos, ou discrepantes. Tais dados são responsáveis por gerar autocorrelações, tendências, sazonalidade e lacunas nas bases, eventos que precisam ser evitados a fim de garantir que as análises feitas sobre a base sejam confiáveis e acuradas. Dito isso, é essencial que estratégias capazes de limpar grandes volumes de dados sejam estudadas. O algoritmo desenvolvido para a detecção de *outliers* proposto por Keogh et al. (2006) possui alta acurácia, porém a sua complexidade, bem como o seu tempo de processamento são elevados, o que torna a sua utilização inviável para cenários reais com alta volumetria de dados.

No presente trabalho, inicialmente foi apresentada relevância das séries temporais na atualidade, bem como os conceitos de limpeza de dados, detecção de *outliers*, paralelização e o estado da arte, de forma a demonstrar a importância da limpeza de dados no campo das séries temporais e como a detecção de *outliers* pode ser uma ferramenta valiosa nesse processo. A partir do algoritmo de Keogh et al. (2006), foi, então, proposta uma estratégia capaz de permitir que grandes volumes de dados se séries temporais sejam limpos de forma eficaz e eficiente, por meio da aplicação de conceitos de paralelização e escalabilidade, utilizados como estratégia para a redução do tempo de execução.

5.1 Contribuição Científica e Trabalhos Futuros

Por meio da implementação das melhorias propostas, com a utilização dos conceitos de paralelização e escalabilidade, o trabalho desenvolvido alcançou os objetivos propostos, isto é, o algoritmo construído indicou redução significativa no tempo de execução se comparado ao algoritmo original. Ademais, ao final do projeto obteve-se um algoritmo que atua em duas

frentes distintas — detecção de *outliers* e limpeza de dados — sem distorções nas informações e de forma eficiente. Dessa forma, os resultados obtidos demonstraram que a integralidade das informações processadas se manteve, de modo a garantir a confiabilidade dos dados. Portanto, foi possível desenvolver um algoritmo que se mostra eficaz e eficiente, capaz de lidar com grandes volumes de dados, de forma a possibilitar a sua aplicação em diversos cenários.

Para trabalhos futuros, propõe-se a melhoria da etapa de pré-processamento dos dados, essencial na execução do algoritmo, de forma a aumentar ainda mais a performance do mesmo. Para este propósito, propõe-se o estudo de estratégias capazes de permitir que os conceitos de paralelização também possam ser aplicados para esta etapa. Ainda, sugere-se que uma aplicação baseada no algoritmo seja desenvolvida, isto é, a criação de uma interface para facilitar a experiência de usuários que necessitem realizar a limpeza de séries temporais.

REFERÊNCIAS

AFRATI, Foto N.; KOLAITIS, Phokion G. Repair checking in inconsistent databases: algorithms and complexity. In: **Proceedings of the 12th International Conference on Database Theory**. 2009. p. 31-41.

AGGARWAL, Charu C.; YU, Philip S. Outlier detection for high dimensional data. In: **Proceedings of the 2001 ACM SIGMOD international conference on Management of data**. 2001. p. 37-46.

AGGARWAL, Charu C.; YU, Philip S. Outlier detection with uncertain data. In: **Proceedings of the 2008 SIAM International Conference on Data Mining**. Society for Industrial and Applied Mathematics, 2008. p. 483-493.

APACHE SPARK, Spark. Disponível em: https://spark.apache.org/. Acesso em 10 de julho de 2021.

BARNETT, V.; LEWIS, T. Outliers in Statistical Data. **John Wiley and Sons**. New York, 1994.

BASU, Sabyasachi; MECKESHEIMER, Martin. Automatic outlier detection for time series: an application to sensor data. **Knowledge and Information Systems**, v. 11, n. 2, p. 137-154, 2007.

BEN-GAL, Irad. Outlier detection. In: **Data mining and knowledge discovery handbook**. Springer, Boston, MA, 2005. p. 131-146.

BRILLINGER, David R. **Time series: data analysis and theory**. Society for Industrial and Applied Mathematics, 2001.

CHOMICKI, Jan; MARCINKOWSKI, Jerzy. Minimal-change integrity maintenance using tuple deletions. **Information and Computation**, v. 197, n. 1-2, p. 90-121, 2005.

DASU, Tamraparni; DUAN, Rong; SRIVASTAVA, Divesh. Data Quality for Temporal Streams. **IEEE Data Eng. Bull.**, v. 39, n. 2, p. 78-92, 2016.

DATABRICKS, Apache Spark. Disponível em: https://databricks.com/spark/about. Acesso em 12 de julho de 2021.

DENNING, Peter J. The locality principle. In: **Communication Networks and Computer Systems: A Tribute to Professor Erol Gelenbe**. 2006. p. 43-67.

DUGGIMPUDI, Maria Bala et al. Spatio-temporal outlier detection algorithms based on computing behavioral outlierness factor. **Data & Knowledge Engineering**, v. 122, p. 1-24, 2019.

FANG, Chenguang et al. Fine-grained fuel consumption prediction. In: **Proceedings of the 28th ACM International Conference on Information and Knowledge Management**. 2019. p. 2783-2791.

FAGIN, Ronald; KIMELFELD, Benny; KOLAITIS, Phokion G. Dichotomies in the complexity of preferred repairs. In: **Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems**. 2015. p. 3-15.

FOX, Anthony J. Outliers in time series. **Journal of the Royal Statistical Society: Series B (Methodological)**, v. 34, n. 3, p. 350-363, 1972.

GARDNER JR, Everette S. Exponential smoothing: The state of the art—Part II. **International journal of forecasting**, v. 22, n. 4, p. 637-666, 2006.

GOLAB, Lukasz et al. Sequential dependencies. **Proceedings of the VLDB Endowment**, v. 2, n. 1, p. 574-585, 2009.

GOOGLE COLABORATORY, Google. Disponível em: https://colab.research.google.com/?hl=pt_BR. Acesso em 18 de outubro de 2021.

GUPTA, Manish et al. Outlier detection for temporal data: A survey. **IEEE Transactions on Knowledge and Data Engineering**, v. 26, n. 9, p. 2250-2267, 2013.

HAWKINS, Douglas M. Identification of outliers. **London: Chapman and Hall**, 1980.

HILDEBRANDT, Kai et al. Large-scale data pollution with Apache Spark. **IEEE Transactions on Big Data**, v. 6, n. 2, p. 396-411, 2017.

HILL, David J.; MINSKER, Barbara S. Anomaly detection in streaming environmental sensor data: A data-driven modeling approach. **Environmental Modelling & Software**, v. 25, n. 9, p. 1014-1022, 2010.

JEFFERY, Shawn R. et al. Declarative support for sensor data cleaning. In: **International Conference on Pervasive Computing**. Springer, Berlin, Heidelberg, 2006. p. 83-100.

KARKOUCH, Aimad et al. Data quality in internet of things: A state-of-the-art survey. **Journal of Network and Computer Applications**, v. 73, p. 57-81, 2016.

KEOGH, Eamonn; LIN, Jessica; FU, Ada. Hot sax: Efficiently finding the most unusual time series subsequence. In: **Fifth IEEE International Conference on Data Mining (ICDM'05)**. Ieee, 2005. p. 8 pp.

KUBERNETES, Kubernetes. Disponível em: https://kubernetes.io/. Acesso em 20 de julho de 2021.

LI, Dan et al. MAD-GAN: Multivariate anomaly detection for time series data with generative adversarial networks. In: **International Conference on Artificial Neural Networks**. Springer, Cham, 2019. p. 703-716.

MAVRIDIS, Ilias; KARATZA, Helen. Performance evaluation of cloud-based log file analysis with Apache Hadoop and Apache Spark. **Journal of Systems and Software**, v. 125, p. 133-151, 2017

MILANI, Mostafa; ZHENG, Zheng; CHIANG, Fei. Currentclean: Spatio-temporal cleaning of stale data. In: **2019 IEEE 35th International Conference on Data Engineering (ICDE)**. IEEE, 2019. p. 172-183.

PYTHON, Python. Disponível em: https://www.python.org/. Acesso em 11 de novembro de 2021.

SINGH, D.; REDDY, C. K. A survey on platforms for big data analytics. **Journal of big data**, v. 2, n. 1, p. 8, 2015.

SONG, Shaoxu et al. SCREEN: stream data cleaning under speed constraints. In: **Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data**. 2015. p. 827-841.

SONG, Shaoxu; CAO, Yue; WANG, Jianmin. Cleaning timestamps with temporal constraints. **Proceedings of the VLDB Endowment**, v. 9, n. 10, p. 708-719, 2016.

SUN, Yuqiang et al. Exploration on spatiotemporal data repairing of parking lots based on recurrent gans. In: **2018 21st International Conference on Intelligent Transportation Systems (ITSC)**. IEEE, 2018. p. 467-472.

WANG, Xi; WANG, Chen. Time series data cleaning: A survey. **IEEE Access**, v. 8, p. 1866-1881, 2019.

WEI, Li; KEOGH, Eamonn; XI, Xiaopeng. Saxually explicit images: Finding unusual shapes. In: **Sixth International Conference on Data Mining (ICDM'06)**. IEEE, 2006. p. 711-720.

ZHUANG, Yongzhen et al. A weighted moving average-based approach for cleaning sensor data. In: **27th International Conference on Distributed Computing Systems (ICDCS'07)**. IEEE, 2007. p. 38-38.

APÊNDICE A – Algoritmo baseado em força bruta

O algoritmo realiza o processo de detecção de subsequências como *outliers* da seguinte maneira: primeiro, este define as variáveis de controle melhor_dis_ate_agora e melhor_loc_ate_agora, utilizadas para verificar qual subsequência de dados possui a melhor distância em relação à subsequência que está sendo analisada e qual a localização dessa subsequência. Após, inicia-se o *loop* externo, que define qual será a subsequência analisada e define-se um valor alto para a variável dist_vizinho_mais_prox, que controla qual a distância entre a subsequência que será analisada e a subsequência mais próxima desta.

Inicia-se, então, o *loop* interno. Este define qual a subsequência será comparada à definida anteriormente. Então, a primeira etapa deste trecho consiste em verificar se a sequência analisada é a mesma que está sendo comparada a ela. Caso não seja, calcula-se a distância entre elas e atualiza-se a variável dist_vizinho_mais_prox. Encerra-se o *loop* interno. Ao fim do *loop* externo, verifica-se se a distância para o vizinho mais próximo é maior que a melhor distância encontrada. Caso seja, atualiza-se as variáveis de melhor distância e localização, atribuindo a elas os valores da distância do vizinho mais próximo e sua localização, respectivamente. Retorna-se, por fim, o par de melhor distância e localização.

O pseudocódigo do algoritmo em questão está exposto na Tabela 4.

Tabela 4 - Pseudocódigo do Algoritmo baseado em "força bruta"

1	**Função** [dist, loc] = Forca_Bruta (*T*,*n*)	
2	melhor_dist_ate_agora = 0	
3	melhor_loc_ate_agora = NaN	
4		
5	**Para** p = 1 até |*T*| - *n* + 1	// Começo do Loop externo

6	dist_vizinho_mais_prox = infinito	
7	**Para** q = 1 até |*T*| - *n* +1	// Começo do Loop interno
8	**Se** |p-q| >= *n*	// Verifica se é a mesma subsequência
9	**Se** Dist [*tp,...,tp+n-1*], [*tq,...,tq+n-1*] < dist_vizinho_mais_prox	
10	dist_vizinho_mais_prox = Dist(*tp,...,tp+n-1, tq,...,tq+n-1*)	
11	**Fim se**	
12	**Fim se**	// Fim da verificação
13	**Fim para**	// Fim do loop interno
14	**Se** dist_vizinho_mais_prox > melhor_dist_ate_agora	
15	melhor_dist_ate_agora = dist_vizinho_mais_prox	
16	melhor_loc_ate_agora = *p*	
17	**Fim se**	
18	**Fim para**	// Fim do loop externo
19	**Retorna** [melhor_dist_ate_agora, melhor_loc_ate_agora]	

Fonte: Adaptado de Keogh et al. (2006)