UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO
DEPARTAMENTO DE ESTATÍSTICA E INFORMÁTICA
PROGRAMA DE PÓS–GRADUAÇÃO EM INFORMÁTICA APLICADA

JOSÉ CARLOS DOS SANTOS

APRENDIZADO DE MÁQUINA APLICADO À PREDIÇÃO DE FALHAS DE SOFTWARE NA COMPUTAÇÃO EM NUVEM

RECIFE – PE

2023

JOSÉ CARLOS DOS SANTOS

APRENDIZADO DE MÁQUINA APLICADO À PREDIÇÃO DE FALHAS DE SOFTWARE NA COMPUTAÇÃO EM NUVEM

Dissertação submetida à Coordenação do Programa de Pós-Graduação em Informática Aplicada da Universidade Federal Rural de Pernambuco, como parte dos requisitos necessários para obtenção do grau de Mestre.

ORIENTADOR: Profª. Dra. Erica Teixeira Gomes de Sousa

COORIENTADOR: Prof. Dr. Tiago Buarque Assunção de Carvalho

RECIFE – PE

2023

Dados Internacionais de Catalogação na Publicação
Universidade Federal Rural de Pernambuco
Sistema Integrado de Bibliotecas
Gerada automaticamente, mediante os dados fornecidos pelo(a) autor(a)

D722a Dos Santos, José Carlos
Aprendizado de Máquina aplicado à predição de falhas de software na computação em nuvem / José Carlos Dos Santos. - 2023.
127 f. : il.

Orientadora: Erica Teixeira Gomes de Sousa.
Coorientador: Tiago Buarque Assuncao de Carvalho.
Inclui referências, apêndice(s) e anexo(s).

Dissertação (Mestrado) - Universidade Federal Rural de Pernambuco, Programa de Pós-Graduação em Informática Aplicada, Recife, 2024.

1. Previsão de falhas. 2. Computação em nuvem. 3. Avaliação de dependabilidade. 4. Aprendizado de máquina. 5. Árvore de decisão. I. Sousa, Erica Teixeira Gomes de, orient. II. Carvalho, Tiago Buarque Assuncao de, coorient. III. Título

CDD 004

JOSÉ CARLOS DOS SANTOS

APRENDIZADO DE MÁQUINA APLICADO À PREDIÇÃO DE FALHAS DE SOFTWARE NA COMPUTAÇÃO EM NUVEM

Dissertação submetida à Coordenação do Programa de Pós-Graduação em Informática Aplicada da Universidade Federal Rural de Pernambuco, como parte dos requisitos necessários para obtenção do grau de Mestre.

Aprovada em: 20 de dezembro de 2023.

BANCA EXAMINADORA

Profª. Dra. Erica Teixeira Gomes de Sousa (Orientadora)
Universidade Federal Rural de Pernambuco
Departamento de Computação

Prof. Dr. Eduardo Antônio Guimarães Tavares
Universidade Federal de Pernambuco
Cin - Centro de Informática

Profª. Dra. Kádna Maria Alves Camboim Vale
Universidade Federal do Agreste de Pernambuco

Dedico este trabalho à minha amada família,
minha esposa Stéfany e meu filho Daniel.

Agradecimentos

Em primeiro lugar, expresso minha sincera gratidão a Deus, que me deu força e sabedoria para superar os diversos desafios na realização deste trabalho. Agradeço à minha família, em especial minha esposa Stéfany pela paciência e compreensão demonstrada nos momentos de ausência necessários para a elaboração deste trabalho. Não poderia deixar de citar meus pais, Sr. Severino e Sra. Fátima, que sempre me apoiam e motivam na busca dos meus objetivos. Por último, mas não menos importante, agradeço aos meus orientadores, professora Erica Sousa e professor Tiago Carvalho, cujas orientações, sempre pontuais e assertivas, foram fundamentais para a conclusão deste estudo. Agradeço-lhes pela paciência e comprometimento demonstrado durante todo o desenvolvimento deste trabalho.

A arte da previsão consiste em antecipar o que acontecerá e depois explicar o porque não aconteceu. (Winston Churchill)

Resumo

Atualmente, os serviços de computação em nuvem são fundamentais para a economia moderna. Diversos setores têm reorientado seus serviços para a nuvem. No entanto, a complexidade e natureza distribuída desses sistemas contribuem para a ocorrência de diversos tipos de falhas que podem resultar em danos catastróficos. Nesse sentido, diversas pesquisas foram conduzidas com o objetivo de mitigar os efeitos das falhas nos sistemas de computação em nuvem. Em particular, as tecnologias de previsão de falhas têm atraído bastante atenção nos últimos anos. Como diversos métodos de previsão de falhas na nuvem foram propostos, foi conduzida uma revisão sistemática da literatura com o objetivo de identificar as técnicas e métodos atualmente utilizados. Como resultado da revisão sistemática, ficou evidenciada a predominância de técnicas de aprendizado de máquina na previsão de falhas na nuvem, assim como a escassez de conjuntos de dados do mundo real para os avanços no estado da arte. Nesse contexto, o objetivo deste trabalho é prover um modelo de previsão de falhas de software para sistemas de computação nuvem. Para alcançar esse objetivo, foram treinados e avaliados modelos de previsão de falhas utilizando aprendizado de máquina com um conjunto de dados do mundo real. O uso de aprendizado de máquina, especificamente técnicas baseadas em árvore de decisão, apresentou resultados promissores na previsão de falhas em sistema de computação em nuvem. Nossa abordagem experimental alcançou acurácia de 94%, assim como recall de 86%, precisão de 85%, F1 score de 85% e AUC de 89%. Entre as contribuições deste trabalho, estão uma revisão do estado da arte da previsão de falhas nos sistemas em nuvem, uma análise detalhada do conjunto de dados utilizado na abordagem experimental, assim como uma abordagem de previsão de falhas com baixo custo computacional e alta interpretabilidade. Finalmente, a abordagem proposta neste trabalho possibilitou a previsão de 76% das tarefas que falhariam com antecedência de até 10 minutos com a interrupção de 4% das tarefas que seriam finalizadas com sucesso.

Palavras-chave: Previsão de falhas; Computação em nuvem; Avaliação de dependabilidade; Aprendizado de máquina; Árvore de decisão.

Abstract

Currently, cloud computing services are fundamental to the modern economy. Various sectors have redirected their services to the cloud. However, the complexity and distributed nature of these systems contribute to the occurrence of various types of failures that can result in catastrophic damage. In this sense, several researches have been conducted to mitigate the effects of failures in cloud computing systems. In particular, failure prediction technologies have attracted considerable attention in recent years. As various cloud failure prediction methods have been proposed, a systematic literature review was conducted to identify the techniques and methods currently used. As a result of the systematic review, the predominance of machine learning techniques in cloud failure prediction was evidenced, as well as the scarcity of real-world data sets for advances in the state of the art. In this context, the objective of this work is to provide a software failure prediction model for cloud computing systems. To achieve this goal, failure prediction models were trained and evaluated using machine learning with a real-world data set. The use of machine learning, specifically decision tree-based techniques, showed promising results in predicting failures in cloud computing systems. Our experimental approach achieved 94% accuracy, as well as 86% recall, 85% precision, 85% F1 score, and 89% AUC. Among the contributions of this work is a review of the state of the art of failure prediction in cloud systems, a detailed analysis of the data set used in the experimental approach, as well as a failure prediction approach with low computational cost and high interpretability. Finally, the approach proposed in this work made it possible to predict 76% of the tasks that would fail up to 10 minutes in advance with the interruption of 4% of the tasks that would be completed.

Keywords: Failure prediction; Cloud computing; Dependability assessment; Machine learning; Decision tree.

Lista de Figuras

Lista de Tabelas

Lista de Siglas

ANN	*Artificial Neural Network*
ARIMA	*Autoregressive Integrated Moving Average*
AUC	*Area Under the Curve*
CART	*Classification and Regression Trees*
CFDR	*The Computer Failure Data Repository*
CNN	*Convolutional Neural Network*
CPU	*Central Processing Unit*
DMOTF	Dias com Muitos *Outliers* em Tarefas que Falharam
DOE	*Design of Experiment*
FN	Falso Negativo
FNN	*Feed-forward Neural Network*
FP	Falso Positivo
GB	*Gradient Boosting*
GRB	*Gated Recurrent Unit*
HD	*Hard Disk*
IA	Inteligência Artificial
KNN	*K-Nearest Neighbors*
LSTM	*Long Short Term Memory*
MAE	*Mean Absolute Error*
MASE	*Mean Absolute Scaled Error*
MaOA	*Many-objective Optimization Algorithm*
NaN	*Not a Number*
NN	*Neural Network*
NTAM	*Neighborhood-Temporal Attention Model*
QDA	*Quadratic Discriminant Analysis*
RAM	*Random Access Memory*

RMTL	*Robust Multi-task Learning Model*
RNN	*Recurrent Neural Networks*
ROC	*Receiver Operating Characteristic*
RMSE	*Root Mean Square Error*
RSL	Revisão Sistemática da Literatura
SMOTE	*Synthetic Minority Oversampling Technique*
SSD	*Solid State Drive*
SVM	*Support Vector Machine*
TIC	Tecnologia da Informação e Comunicação
TFP	Taxa de Falsos Positivos
TPS	*Temporal Progressive Sampling*
TVP	Taxa de Verdadeiros Positivos
VM	*Virtual Machine*
VP	Verdadeiro Positivo
VN	Verdadeiro Negativo

Sumário

1 Introdução

1.1 Contextualização

O modelo de computação em nuvem possui diversas vantagens, sendo uma solução amplamente utilizada no apoio ao fornecimento de serviços públicos e privados dos mais variados setores (DOMINGOS, 2019). Entre as principais vantagens está o fato da computação em nuvem consistir em um paradigma no qual os serviços de tecnologia da informação e comunicação (TIC) são fornecidos sob demanda, ou seja, os recursos podem ser ajustados dinamicamente para possibilitar a realização de diferentes cargas de trabalho (HERBST *et al.*, 2018).

Embora o termo "nuvem" tenha sido utilizado como metáfora para representar uma infraestrutura acessível por rede desde o início da Internet (MITREVSKI *et al.*, 2019), atualmente, representa um mercado significativo com índices crescentes de aceitação, uma vez que as aplicações em nuvem já representam uma parcela significativa de todo mercado na Europa, América do Norte, Ásia e Oriente Médio (HERBST *et al.*, 2018). Os serviços de computação em nuvem são fundamentais para a economia moderna, além de facilitar a expansão de empresas em todo o mundo (BUYYA *et al.*, 2018). Estima-se que em 2025 mais de três quartos dos negócios globais possam residir na nuvem (HERBST *et al.*, 2018).

Tipicamente, os sistemas em nuvem fornecem diversos serviços, além de possuírem diversos subsistemas, os quais são compostos por inúmeros componentes interligados, o que contribui para o aumento da complexidade nesses sistemas. (CHEN *et al.*, 2019). A complexidade e natureza distribuída dos sistemas em nuvem contribuem para a ocorrência dos mais variados tipos de falhas, tais como: falha de software, falha de hardware, congestionamento de rede e falta de recursos (sobrecarga de servidor).

Mesmo em provedores de computação em nuvem classificados como provedores de alto nível, podem ocorrer interrupções do fornecimento de serviço. As falhas são o principal motivo de indisponibilidade na computação em nuvem (GAO *et al.*, 2020; RAWAT; BHADORIA, 2021). Estima-se que as falhas na nuvem representem um prejuízo anual de 700 bilhões de dólares (LI *et al.*, 2020).

Nesse sentido, diversas pesquisas foram conduzidas com o objetivo de mitigar os efeitos das falhas nos sistemas de computação em nuvem. Em particular, as tecnologias de

previsão de falhas têm atraído bastante atenção nos últimos anos (LIU *et al.*, 2017). A previsão de falhas é definida como um conjunto de ações proativas para detectar condições anormais antes que elas realmente aconteçam (GOKHROO *et al.*, 2017). Na prática, um preditor prevê falhas correlacionando dados passados do sistema na presença de falhas com dados atuais obtidos por monitoramento de variáveis do sistema em tempo de execução (IRRERA, 2016).

1.2 Motivação e Justificativa

Devido a complexidade ocasionada pela evolução do software nos últimos anos, há consenso de que nenhum sistema computacional está livre de falhas. Dessa forma, a implantação de sistemas complexos sem falhas é uma meta inatingível (IRRERA, 2016; GUPTA *et al.*, 2017; MARIANI *et al.*, 2018; MONNI *et al.*, 2019). Entretanto, a capacidade de detectar falhas antes de sua ocorrência pode garantir a resiliência desses sistemas e consequentemente a continuidade dos serviços (BHATTACHARYYA *et al.*, 2017a).

A diversidade dos serviços e o aumento da popularidade dos sistemas de computação em nuvem tem como resultado a elevação do tráfego de rede e maior utilização dos recursos, elevando a probabilidade de ocorrência de falhas (KABIR *et al.*, 2021). Normalmente, a ocorrência de falhas nesses sistemas está associada a perdas financeiras, mas em alguns casos pode resultar em perdas de vidas humanas (IRRERA, 2016). Dessa forma, a implantação de um mecanismo de previsão de falhas possibilita mitigar os possíveis impactos decorrentes da interrupção dos serviços (MOHAMMED *et al.*, 2019).

Diversas técnicas de previsão de falhas foram propostas, no entanto, há predominância do uso de técnicas de aprendizado de máquina no desenvolvimento de modelos de previsão (DOMINGOS, 2019). Nos últimos anos, o uso de aprendizado de máquina na previsão de falhas em sistemas de computação em nuvem recebeu maior atenção (MOHAMMED *et al.*, 2019). Por exemplo, algoritmos como *Random Forest* e *Decision Tree* têm sido amplamente utilizados (BHANAGE *et al.*, 2021).

Neste contexto, a inevitabilidade das falhas, a necessidade de manter a resiliência desses sistemas em função dos possíveis danos ocasionados pela ocorrência de falhas, assim como a diversidade de métodos de previsão atualmente utilizados nos conduz ao seguinte questionamento: como prever falhas de software em sistemas de computação em nuvem?

1.3 Objetivos

1.3.1 Objetivo Geral

O principal objetivo deste trabalho é prover um modelo de previsão de falhas de software para sistemas de computação em nuvem. A fim de atingir o objetivo principal, os seguintes objetivos específicos foram definidos:

1.3.2 Objetivo Específico

- Revisar o estado da arte de forma sistemática, analisando os métodos mais recentes utilizados no processo de previsão de falhas em sistemas de computação em nuvem.
- Avaliar o desempenho de modelos baseados em *Machine Learning* para previsão de falhas em sistemas em nuvem.
- Analisar a relação das características da carga de trabalho com a ocorrência de falhas.

1.4 Estrutura da Dissertação

Os próximos capítulos estão organizados da seguinte forma:

Capítulo 2: Neste capítulo são apresentados os conceitos básicos necessários para a compreensão das ideias apresentadas neste estudo.

Capítulo 3: Neste capítulo é apresentada uma revisão sistemática da literatura a respeito do estado atual dos estudos relacionados à previsão de falhas nos sistema em nuvem.

Capítulo 5: Neste capítulo é apresentada uma abordagem experimental de previsão de falhas na nuvem. Os resultados obtidos são comparados com resultados de estudos anteriores.

Capítulo 6: Este capítulo conclui o trabalho com as contribuições, limitações e trabalhos futuros.

2 Fundamentação Teórica

Este capítulo apresenta os conceitos básicos indispensáveis à compreensão das ideias apresentadas neste estudo. O capítulo está organizado nas seguintes seções: A Seção 2.1 define os conceitos básicos relacionados a computação em nuvem. A Seção 2.2 apresenta os conceitos de dependabilidade. A Seção 2.3 apresenta técnicas de previsão de falhas. A Seção 2.4 define os conceitos relacionados ao aprendizado de máquina. Finalmente, a Seção 2.5 detalha os conceitos do design experimental.

2.1 Computação em Nuvem

A computação em nuvem consiste de um modelo que possibilita acesso onipresente, conveniente e sob demanda a um conjunto compartilhado de recursos de computação configuráveis que podem ser provisionados e liberados rapidamente com o mínimo de esforço de gerenciamento e interação do provedor de serviços (MELL; GRANCE, 2011).

Moghaddam *et al.* (2015) definem a computação em nuvem como uma tecnologia que utiliza o conceito de virtualização, poder de processamento, armazenamento e conectividade para fornecer um conjunto de recursos armazenados ou compartilhados entre diversos dispositivos através da Internet. Moravcik *et al.* (2018) definem quatro modelos de implementação dos sistemas de computação em nuvem: nuvem pública, nuvem privada, nuvem comunitária e nuvem híbrida.

- **Nuvem pública:** É um modelo de implementação cujos serviços estão disponíveis para o público em geral. Pode ser administrada por organizações públicas ou privadas, assim como instituições acadêmicas ou governamentais. É o modelo de implementação mais difundido. Os serviços, em sua maioria, são ofertados mediante compensação monetária. Todos os recursos estão localizados no ambiente do provedor de serviço.
- **Nuvem privada:** É definida como uma infraestrutura utilizada e administrada pela mesma organização, embora possa ser administrada por um terceiro. Como toda infraestrutura é internamente concentrada na própria organização, esse modelo de implementação tem como principal característica o gerenciamento do ambiente e, consequentemente, maiores níveis de segurança.

- **Nuvem comunitária:** É definida como uma infraestrutura utilizada exclusivamente por uma comunidade com interesses comuns. Os recursos podem ser gerenciados pela própria comunidade ou por terceiros. Este modelo de implementação prioriza quem tem permissão para usar os recursos, independentemente de onde estão localizados. Ou seja, enquanto em nuvens privadas a infraestrutura é interna à organização, em nuvens comunitárias a localização da infraestrutura não é relevante.
- **Nuvem híbrida:** Este modelo de implementação consiste da combinação de pelo menos dois outros modelos (pública, privada, comunitária). A utilização desse modelo de implantação objetiva combinar os benefícios de outros modelos. Por exemplo, uma organização pode utilizar uma nuvem privada como infraestrutura primária e uma nuvem pública para realização de backup, garantindo a continuidade dos serviços em casos de desastres na infraestrutura local.

Os sistemas de computação em nuvem ainda podem ser classificados em pelo menos três modelos de serviços: Infraestrutura como serviço (IaaS), Plataforma como serviço (PaaS) e Software como serviço (SaaS) (BOKHARI *et al.*, 2018).

- **Infraestrutura como serviço (IaaS):** Este modelo de serviço é caracterizado pelo fornecimento de recursos de computação por meio de virtualização. O fornecimento dos serviços são contabilizados de acordo com o tempo e capacidade dos recursos como CPU, memória e armazenamento utilizados. Os clientes possuem a capacidade de modificar configurações da infraestrutura como reiniciar os servidores, instalar discos virtuais e instalar pacotes de software (BOKHARI *et al.*, 2018).
- **Plataforma como serviço (PaaS):** Este modelo de serviço refere-se a disponibilização de um ambiente de execução e desenvolvimento de aplicações (BELLO *et al.*, 2021). Os provedores de serviço fornecem ferramentas de desenvolvimento hospedadas na nuvem, eliminando a complexidade de configuração e gerenciamento do ambiente (MORAVCIK *et al.*, 2018).
- **Software como serviço (SaaS):** Este modelo de serviço permite que os clientes utilizem software do provedor de serviço através de uma interface web sem a necessidade de instalação (BOKHARI *et al.*, 2018). Como as aplicações são executadas pelo provedor de serviço, os usuários não realizam atividades como atualizações e backups dos dados (MORAVCIK *et al.*, 2018).

Conforme Buyya *et al.* (2018), a medida que mais organizações demonstram interesse

nas tecnologias de computação em nuvem, cresce a demanda por métodos que possam garantir o fornecimento dos serviços com garantia de desempenho e resiliência.

2.2 Dependabilidade

O conceito de dependabilidade é definido como a capacidade de um sistema fornecer um conjunto de serviços confiáveis (MACIEL, 2023). Conforme Souza *et al.* (2022), o conceito de dependabilidade é dividido em três categorias: ameaças, atributos e os meios pelos quais a dependabilidade é alcançada.

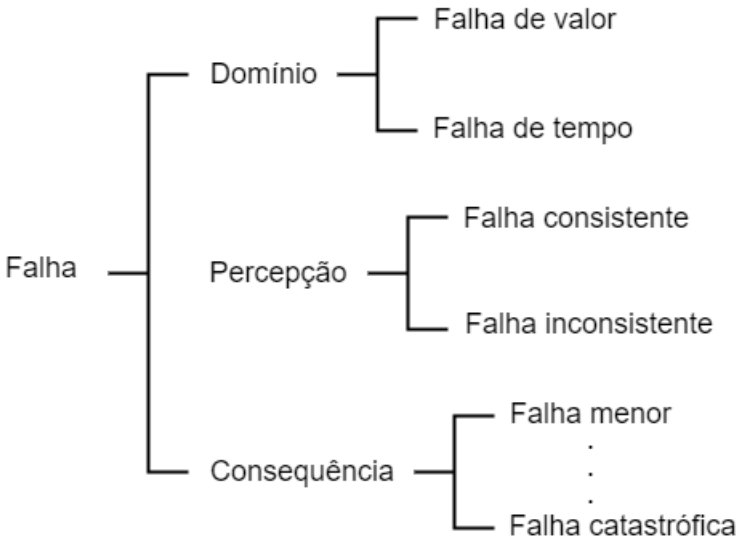
- **Ameças:** Correspondem aos eventos que ocasionam a entrega do serviço de forma indesejada, ou seja, defeito, erro e falha.
- **Atributos:** Corresponde a disponibilidade, confiabilidade, integridade, confidencialidade, segurança e manutenibilidade do sistema.
- **Meios:** São os métodos utilizados para alcançar a dependabilidade. Esses métodos incluem a prevenção, tolerância, previsão e remoção de falhas.

A avaliação de dependabilidade está diretamente relacionada aos estudos dos defeitos, erros e falhas no sistema (SOUZA *et al.*, 2022). Apesar de estarem relacionados, defeito, erro e falha têm significados diferentes. O **defeito** (*fault*) ocorre quando algum elemento do sistema está inoperante ou operando de forma incorreta. O **error** (*error*) é um estado posterior ao defeito, que ocorre quando o sistema utiliza um recurso defeituoso e a função correspondente não é realizada corretamente (CABAN; WALKOWIAK, 2016). A **falha** (*failure*) é definida como um desvio do serviço prestado em relação ao serviço esperado, ou seja, o sistema não é capaz de fornecer a funcionalidade esperada. É importante lembrar que um sistema pode funcionar corretamente mesmo na presença de defeitos que podem permanecer inativos ou que podem ser ativados ocasionando um erro. Os erros, por sua vez, podem permanecer latentes ou propagar-se para a interface do sistema, provocando uma falha (IRRERA, 2016).

As falhas podem ser classificadas de acordo com os modos como se manifestam no sistema. Conforme exibido na Figura 1, esses modos são caracterizados pelo domínio, percepção e consequência das falhas. Em relação ao domínio, as falhas podem ser classificadas como falhas de valor ou de tempo. A falha de valor ocorre quando a informação entregue pelo serviço diverge da esperada. A falha de tempo é caracterizada pela entrega da

informação fora do tempo especificado. Quanto a percepção do usuário, as falhas podem ser consistentes ou inconsistentes. Uma falha é consistente quando seus efeitos são percebidos da mesma forma pelos usuários do sistema, enquanto os efeitos das falhas inconsistentes são percebidos de formas diferentes. As consequências das falhas são classificadas em níveis que vão desde uma falha menor até uma falha catastrófica. As falhas menores causam baixo impacto no fornecimento do serviço, enquanto as falhas catastróficas comprometem o seu fornecimento, sendo algumas vezes inadmissíveis (MACIEL, 2023).

Figura 1 – Modos de falhas.



Fonte: Adaptado de Maciel (2023).

Entre aos atributos da dependabilidade, a disponibilidade é definida como a probabilidade de um sistema estar operacional em um determinado período de tempo (SOUZA *et al.*, 2022). A disponibilidade de um sistema é definida na Equação 2.1.

$$\text{Disponibilidade} = \frac{\text{T\_operacional}}{\text{T\_operacional} + \text{T\_interrupção}}, \tag{2.1}$$

em que T_operacional representa o tempo operacional do sistema e T_interrupção corresponde ao tempo de interrupção do fornecimento do serviço. Por sua vez, a confiabilidade é um atributo que avalia a capacidade de entrega contínua de um serviço (SOUZA *et al.*, 2022).

Avizienis *et al.* (2004) define os meios pelos quais a dependabilidade é alcançada.

- **Prevenção de falhas:** Consiste na melhoria dos processos de desenvolvimento de hardware e software com o objetivo de reduzir o número de falhas introduzidas no desenvolvimento.
- **Tolerância a falhas:** Tem como objetivo evitar as falhas através da detecção de erros e recuperação do sistema.
- **Previsão de falhas:** Consiste na avaliação dos componentes do sistema para classificar combinações de eventos que possam resultar em falhas.
- **Remoção de falhas:** É um processo que ocorre durante a fase de desenvolvimento do sistema. Consiste na verificação e correção de condições no sistema que possam impedir o cumprimento de sua função.

2.3 Técnicas de Previsão de Falhas

A previsão de falhas é definida como um conjunto de ações proativas para detectar condições anormais antes que elas realmente aconteçam (GOKHROO *et al.*, 2017). Preditores de falhas modernos utilizam métricas coletadas dos nós de computação nos sistemas em nuvem para inferir a presença de falhas. Os dados coletados são utilizados no processo de treinamento de modelos que possam prever a ocorrência de falhas em um futuro próximo (MONNI *et al.*, 2019).

Conforme descrito por Jauk *et al.* (2019), diversos métodos para previsão de falhas foram propostos. Por exemplo, métodos baseados em análise de logs e probabilidade, métodos baseados em regras e modelos de Markov estão entre os métodos de previsão de falhas propostos. Entretanto, os métodos de previsão de falhas atualmente utilizados podem ser classificados em três categorias: abordagem estatística, abordagem de aprendizado de máquina e abordagem de aprendizagem profunda (GAO *et al.*, 2020).

O modelo Autorregressivo Integrado de Média Móvel (ARIMA) é uma das abordagens estatísticas mais utilizadas para a previsão de séries temporais (HERMIAS *et al.*, 2017). Uma série temporal é um conjunto de dados que possui suas amostras de acordo com um determinado intervalo de tempo. Esses dados podem ser utilizados para prever o comportamento futuro do sistema (SHAO; ZHANG, 2018). Os métodos existentes para previsão de séries temporais em sistemas em nuvem são principalmente baseados em ARIMA (ADEGBOYEGA, 2017).

O aprendizado de máquina é um ramo da inteligência artificial que permite que os computadores aprendam de forma autônoma, ou seja, sem uma programação explícita (SU *et al.*, 2019). As técnicas de aprendizado de máquina têm sido utilizadas com maior ênfase desde 2016, isso se deve à sua capacidade de classificação e previsão. Algoritmos como *Support Vector Machine* (SVM), *Random Forest* e *Naive Bayes* são exemplos dos algoritmos mais utilizados para detecção e previsão de falhas (BHANAGE *et al.*, 2021).

A aprendizagem profunda é uma subárea do aprendizado de máquina, no entanto, seus algoritmos necessitam de maiores quantidades de dados para o processo de treinamento (SU *et al.*, 2019). Com o crescimento da quantidade de dados de logs disponibilizados pelos sistemas, soluções de aprendizagem profunda passaram a ser utilizadas na previsão de falhas (BHANAGE *et al.*, 2021). Uma outra característica dos modelos de aprendizagem profunda é o custo computacional elevado, ou seja, exigem maior desempenho computacional (FAHIM; SILLITTI, 2019).

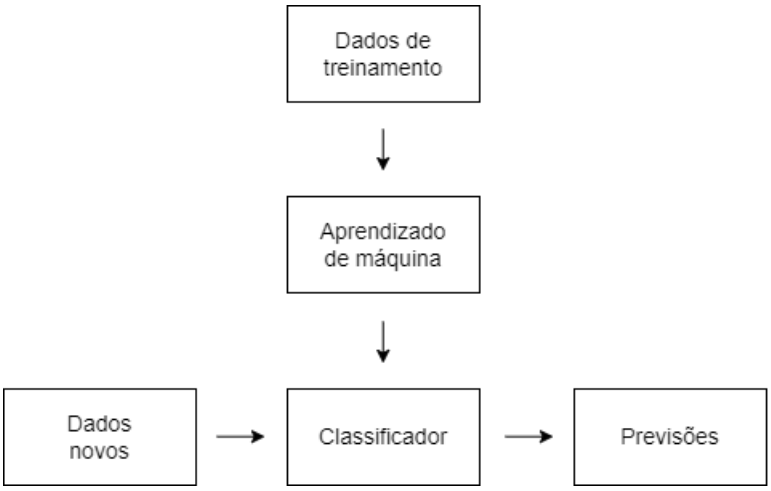
2.4 Técnica de Classificação na IA

A Inteligência Artificial (IA) é definida como qualquer técnica que permita que os computadores resolvam tarefas complexas de forma independente ou com mínima intervenção humana (JANIESCH *et al.*, 2021). Por sua vez, o Aprendizado de Máquina é um dos campos da IA amplamente utilizado em aplicações inteligentes (MRABET *et al.*, 2021). O Aprendizado de Máquina é classificado em aprendizado supervisionado, aprendizado não supervisionado e aprendizado por reforço (JANIESCH *et al.*, 2021).

A classificação é uma das técnicas do aprendizado supervisionado utilizada em problemas onde o objetivo da previsão consiste em determinar como saída um valor discreto, ou seja, um rótulo ou classe (MRABET *et al.*, 2021). Assim como todas as técnicas de aprendizado supervisionado, a classificação requer um conjunto de dados de treinamento com os exemplos para as entradas, assim como os rótulos ou classes de destino. Os dados de treinamento são utilizados para construir um modelo capaz de prever o rótulo de dados não vistos anteriormente (NASTESKI, 2017). A Figura 2 ilustra o processo de aprendizagem. Como pode ser observado, os dados de treinamento são utilizados como entrada de um modelo de aprendizado de máquina, que após se ajustar aos dados de treinamento, é capaz de classificar dados novos, ou seja, prever a classe a qual os dados pertencem. É

importante salientar que na avaliação do modelo, os dados não vistos anteriormente são denominados dados de teste, e costumam ser uma fração dos dados utilizados no processo de treinamento.

Figura 2 – Processo de aprendizagem.



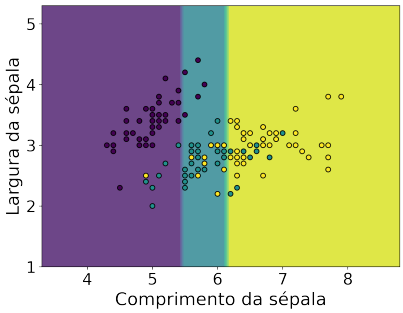
Fonte: Adaptado de Nasteski (2017).

Para garantir o bom desempenho do modelo, alguns aspectos são fundamentais no processo de treinamento. Por exemplo, o ajuste dos hiperparâmetros do algoritmo e a qualidade dos dados utilizados. Quando não observados, esses aspectos podem resultar em *overfitting* ou *underfitting*. Ocorre *overfitting* quando o modelo tem um bom desempenho nos dados de treinamento e baixo desempenho nos dados de teste. Por outro lado, o *underfitting* é caracterizado por baixo desempenho ainda na fase de treinamento (MRABET *et al.*, 2021). Enquanto o *overfitting* está em geral associado à alta complexidade do modelo ou a utilização de poucos dados com muitas variáveis, o *underfitting* ocorre quando o modelo é muito simples ou quando os dados utilizados são pouco informativos (BADILLO *et al.*, 2020). Por sua vez, o modelo que obtém bom desempenho na classificação de dados não vistos (dados de teste) possui alta capacidade de generalização (YETURU, 2020).

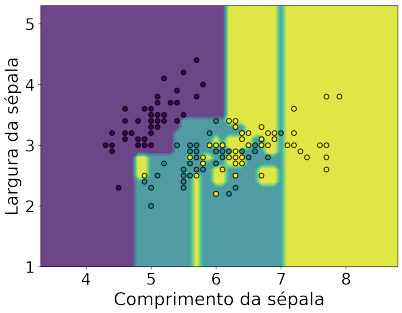
A Figura 3 ilustra o comportamento de modelos de classificação com *underfitting*, *overfitting* e com boa generalização. Em todos os exemplos, foi utilizado o conjunto de dados Iris (FISHER, 1936). Este conjunto de dados é composto por 50 amostras de três diferentes espécies da flor Iris - Setosa, Versicolor e Virginica, totalizando 150 amostras. Cada amostra possui o comprimento e largura da sépala, assim como o comprimento e largura da pétala. As amostras de cada espécie são identificadas por pontos de uma

determinada cor. O objetivo da classificação consistem em traçar divisões de forma que as amostras de uma determinada espécie fique na cor correspondente. A Figura 3a ilustra um modelo com *underfitting*. É possível observar que o modelo classificou incorretamente diversas amostras de cada uma das classes. Esses erros de classificação são justificados pela simplicidade do modelo. Por outro lado, conforme exibido na Figura 3b, o modelo se ajustou para classificar corretamente a maior quantidade possível de amostras de treino, caracterizando o *overfitting*. Por sua vez, o modelo exibido na Figura 3c não se ajustou excessivamente aos dados, ou seja, possui maior capacidade de generalização.

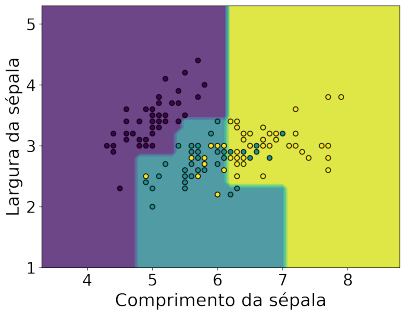
Figura 3 – Representação de modelos de classificação com *underfitting*, *overfitting* e com boa generalização.



(a) Modelo com *underfitting*



(b) Modelo com *overfitting*



(c) Modelo com melhor ajuste

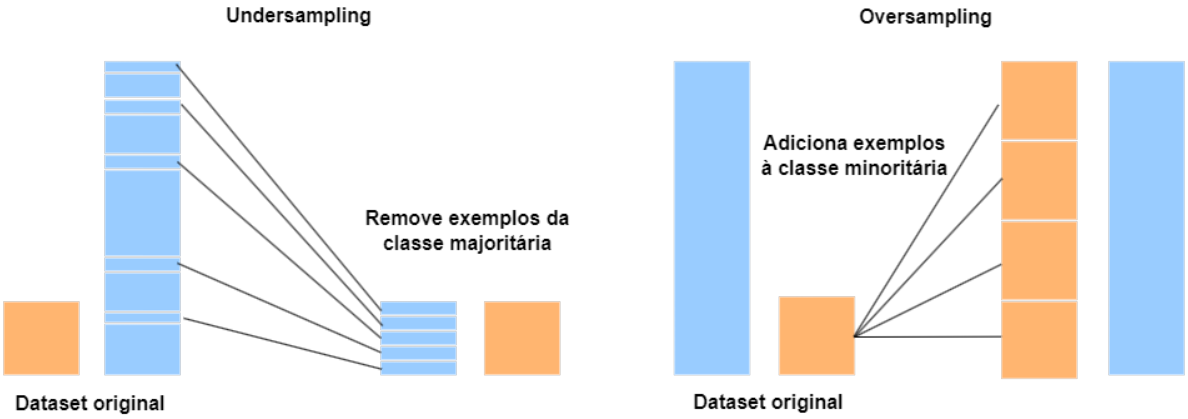
Fonte: O autor.

2.4.1 Balanceamento de classes

Em aprendizado de máquina e estatística, uma característica recorrente nos conjuntos de dados é o desbalanceamento das classes. Ou seja, as classes que se deseja prever não têm a mesma proporção no conjunto de dados e, normalmente, a classe alvo é

a minoritária. Por exemplo, na abordagem de previsão de falhas, objetivo deste estudo, as falhas são exceção, ou seja, corresponde à classe minoritária. Essa característica pode resultar em modelos com baixo desempenho (MOHAMMED *et al.*, 2020). O desbalanceamento de classes é normalmente tratado com estratégias de amostragem, por exemplo, *oversampling* ou *undersampling*. O *oversampling* adiciona amostras a partir da duplicação aleatória da classe minoritária, enquanto o *undersampling* remove aleatoriamente amostras da classe majoritária (LI *et al.*, 2020). A Figura 4 ilustra o funcionamento do *undersampling* e *oversampling*.

Figura 4 – *Undersampling* e *Oversampling*.



Fonte: Adaptado de Mohammed *et al.* (2020).

2.4.2 Árvore de Decisão

Árvore de decisão é uma das abordagens de aprendizado supervisionado mais utilizadas em problemas de classificação (HAMAD; ZEKI, 2018). Os algoritmos de Árvore de decisão classificam os dados dividindo-os em uma estrutura em forma de árvore. Essa estrutura é composta por nós denominados nó raiz, nós internos e nós folhas. O nó raiz corresponde ao primeiro nó da árvore e contém todas as observações dos dados. Os nós internos representam os testes em relação aos atributos de entrada. Por sua vez, os nós folhas correspondem às classes previstas, ou seja, os resultados das previsões (MRABET *et al.*, 2021). As árvores de decisão são implementadas selecionando as variáveis que melhor dividam os dados em relação às classes que representam o objetivo das previsões (OBULESU *et al.*, 2018).

Os modelos de árvore de decisão podem ser baseados em diversos algoritmos, por

exemplo ID3, C4.5 e CART (*Classification and Regression Trees*) (JIAO *et al.*, 2020). Como o algoritmo CART possibilita a implementação de árvores de decisão para resolver problemas de classificação e regressão. Nas árvores de decisão baseadas no algoritmo CART, o critério utilizado para selecionar a melhor divisão dos dados é o Índice Gini. O Índice Gini varia entre 0 e 1 e determina o quão puros estão os dados em cada nó da árvore. Quando todos os dados pertencem a mesma classe, o Índice Gini corresponde a 0 e os dados são considerados puros. Por outro lado, um Índice Gini próximo de 1 indica que os dados estão distribuídos em classes diferentes (LU *et al.*, 2022). A Equação 2.2 exibe a formulação do Índice Gini.

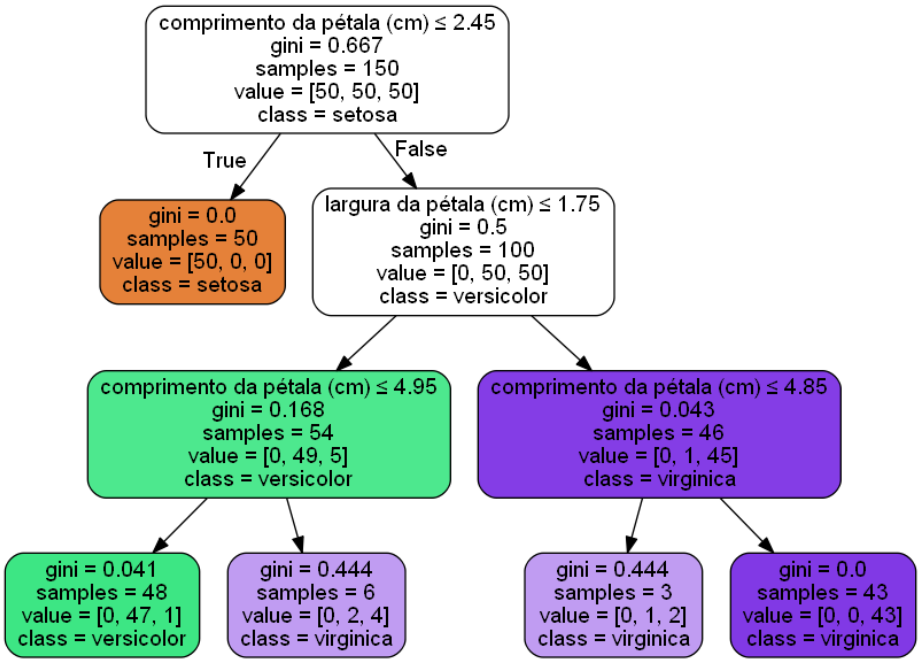
$$Gini(p) = 1 - \sum_{i=1}^{n} (p_i)^2, \tag{2.2}$$

em que n corresponde ao número de classes e p_i é a probabilidade de um elemento ser da i-ésima classe.

A Figura 5 ilustra a aplicação de um modelo de árvore de decisão para classificar as espécies no conjunto de dados Iris (FISHER, 1936). Para facilitar a interpretação do modelo, foram utilizadas apenas duas variáveis do conjunto de dados (comprimento e largura da pétala). Como pode ser observado, a árvore resultante possui três nós internos e cinco folhas. Em cada nó da árvore é exibido o parâmetro de teste, o valor do Índice Gini, o total de exemplos, a quantidade de exemplos de cada classe, assim como a classificação dos dados no respectivo nó. Para impedir o aumento da complexidade do modelo, a profundidade máxima da árvore foi limitada em três níveis. Como resultado, apenas duas folhas ficaram puras (possuem Índice Gini igual a 0).

O controle da complexidade é um aspecto fundamental nos modelos baseados em árvore de decisão. Árvores mais complexas estão associadas a *overfitting* do modelo (NASTESKI, 2017). Como os nós da árvore de decisão continuarão sendo divididos até que todos os nós sejam puros ou exista pelo menos uma obervação em cada nó resultante da divisão, os modelos de árvore de decisão são naturalmente propensos a *overfitting* (BERTSIMAS; DUNN, 2017). Para tratar essa característica são normalmente utilizadas técnicas de poda da árvore. A poda consiste em impedir o crescimento da árvore, atribuindo condições de parada na divisão dos nós. Essas condições incluem principalmente o ajuste da profundidade máxima da árvore (MISHRA, 2020). Por exemplo, na árvore de decisão

Figura 5 – Exemplo de árvore de decisão com conjunto de dados Iris.



Fonte: O autor.

ilustrada na Figura 5, é possível observar que os nós internos no segundo nível foram divididos em folhas com uma quantidade reduzida de exemplos, caracterizando *overfitting*. O modelo teria maior capacidade de generalização com apenas dois níveis.

Assim como os demais modelos de classificação, os modelos de árvore de decisão são normalmente implementados com o uso de bibliotecas de aprendizado de máquina. Entre as bibliotecas atualmente utilizadas, o *Scikit-learn*[1] é um pacote de software muito popular (DOMINGOS, 2019). Em relação aos modelos de árvore de decisão, o *Scikit-learn* utiliza uma versão modificada do algoritmo CART (PEDREGOSA *et al.*, 2011), que quando aplicado em problemas de classificação, é implementado na classe *DecisionTreeClassifier*. Essa classe possui os parâmetros que controlam o aprendizado do modelo, alguns desses parâmetros são listados a seguir. O correto ajuste desses parâmetros é fundamental para evitar *underfitting* ou *overfitting* do modelo.

- max_depth: Profundidade máxima da árvore.
- max_features: Número máximo de atributos considerados para a divisão de um nó.
- max_leaf_nodes: Número máximo de folhas.
- min_samples_split: Número mínimo de amostras para divisão de um nó interno.
- min_samples_leaf: Número mínimo de amostras em um nó folha.

[1] https://scikit-learn.org

Os modelos de árvore de decisão fornecem a importância das variáveis para o resultado das previsões, ou seja, o quanto cada variável contribuiu para o resultado alcançado. Essa métrica pode ser utilizada para selecionar as variáveis mais preditivas nos dados de treinamento e consequentemente reduzir a complexidade do modelo (ASMAWI *et al.*, 2022). A Equação 2.3 descreve a importância da variável na árvore de decisão.

$$\text{Importância da variável} = \frac{\sum_{i=1}^{K} \text{Importância do Nó}_i}{\sum_{i=1}^{N} \text{Importância do Nó}_i}, \tag{2.3}$$

em que K corresponde ao número de nós que dividem a variável e N é o número total de nós. Por sua vez, a importância do nó é calculada pela redução da impureza ponderada pelo número de amostras no nó, conforme a equação 2.4

$$\text{I. Nó} = \frac{N_t}{N} \times (\text{impureza} - \frac{N_{t_R}}{N_t} \times \text{impureza\_nó\_direito} - \frac{N_{t_L}}{N_t} \times \text{impureza\_nó\_esquerdo}), \tag{2.4}$$

em que N_t é o número de amostras no nó atual, N é o número total de amostras, N_{t_R} é o número de amostras no nó direito e N_{t_L} é o número de amostras no nó esquerdo.

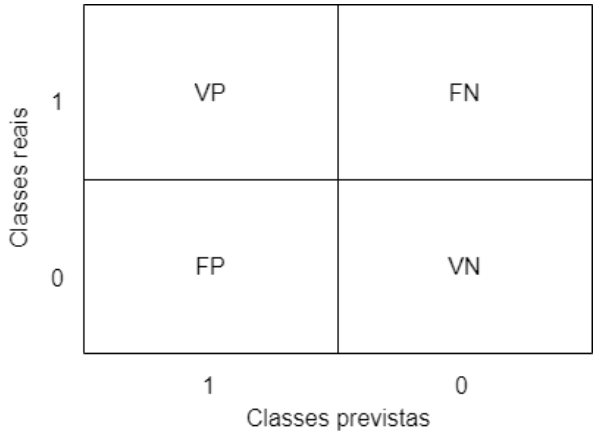
2.4.3 Métricas de Avaliação

Vários aspectos são levados em consideração na avaliação de modelos preditivos, como desempenho, interpretabilidade e recursos computacionais utilizados. As métricas relacionadas a desempenho avaliam quão bem o modelo produziu o resultado esperado. As métricas comumente utilizadas em modelos de classificação são acurácia, *recall*, precisão, e F1 score (JANIESCH *et al.*, 2021).

Essas métricas são calculadas a partir dos resultados da matriz de confusão, que consiste em uma tabela onde são exibidas as proporções das classes previstas em relação às classes reais. Na matriz de confusão, as colunas representam as previsões e as linhas as representam as classes reais. Em problemas de classificação binária, quando o objetivo da previsão se limita a duas classes, a matriz de confusão é composta por duas linha e duas colunas (YETURU, 2020). A Figura 6 ilustra uma matriz de confusão binária, onde os rótulos das classes correspondem a 0 (classe negativa) e 1 (classe positiva). No

exemplo, VP corresponde aos verdadeiros positivos, FP corresponde aos falsos positivos, VN representa os verdadeiros negativos e FN os falsos negativos.

Figura 6 – Exemplo de matriz de confusão binária.



Fonte: O autor.

Utilizada para avaliar o desempenho geral do modelo, a acurácia é definida como a proporção de previsões corretas em relação ao total de previsões. A precisão é a proporção de previsões positivas corretas em relação ao total de previsões positivas. O *recall* é a proporção de exemplos positivos corretamente identificados pelo modelo em relação ao total de exemplos positivos (JAUK *et al.*, 2019). O F1 score consiste da média harmônica da precisão e *recall* (NOTARO *et al.*, 2021). A acurácia, precisão, *recall* e F1 score são descritos nas Equações 2.5, 2.6, 2.7 e 2.8, respectivamente. É importante destacar que a precisão, *recall* e F1 score podem ser calculados para a classe negativa.

$$\text{Acurácia} = \frac{VP + VN}{VP + VN + FP + FN} \tag{2.5}$$

$$\text{Precisão} = \frac{VP}{VP + FP} \tag{2.6}$$

$$Recall = \frac{VP}{VP + FN} \tag{2.7}$$

$$F1 = 2 \times \frac{\text{Precisão} \times Recall}{\text{Precisão} + Recall} \tag{2.8}$$

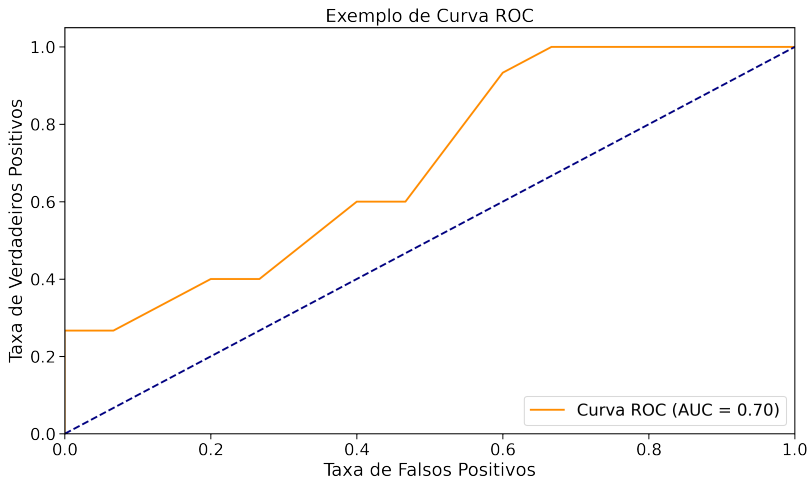
A curva ROC (*Receiver operating characteristic*) é um gráfico utilizado na avaliação e comparação de classificadores binários. Essa métrica é principalmente recomendada para a avaliação de modelos com classes desbalanceadas (LIU; XIE, 2023). A curva ROC consiste de um gráfico onde o eixo X corresponde a taxa de falsos positivos e o eixo Y corresponde a taxa de verdadeiros positivos (YETURU, 2020). A taxa de falsos positivos (TFP) é descrita na Equação 2.9, enquanto a taxa de verdadeiros positivos (TVP) é descrita na Equação 2.10.

$$TFP = \frac{FP}{FP + VN} \tag{2.9}$$

$$TVP = \frac{VP}{VP + FN} \tag{2.10}$$

A partir da curva ROC é calculada a AUC (*area under the curve*), que consiste da área sob a curva ROC. A AUC resume os resultados da curva ROC e tem seus valores distribuídos entre 0 e 1 (YETURU, 2020). A Figura 7 ilustra um exemplo de curva ROC AUC. Um modelo cujas previsões sejam 100% corretas possui AUC igual a 1. Por outro lado, um modelo que realize todas as previsões incorretamente possui AUC igual a 0. Por sua vez, um modelo cujas previsões sejam aleatórias resultará em AUC de 0,5 (representada no gráfico pela diagonal ascendente) (FLACH, 2016).

Figura 7 – Exemplo de curva ROC AUC.



Fonte: O autor.

2.5 Design Experimental

O *Design of Experiment* (DOE) é definido como um método estruturado utilizado para determinar as relações entre as variáveis de entrada e saída de um experimento. O principal objetivo do DOE é obter o máximo de informação com a realização mínima de experimentos (LEE *et al.*, 2022). Jain (1991) define os termos básicos necessários à compreensão do DOE. O DOE consiste em determinar o número de experimentos e as combinações de fatores e níveis utilizados. O DOE pode ser definido através de fatores, níveis e variável resposta.

- **Fatores:** São as variáveis que afetam a variável de resposta, também são denominadas variáveis preditoras. Fatores que têm um impacto quantificável são classificados como fatores primários. Por outro lado, fatores cuja quantificação não é de interesse são classificados como fatores secundários.
- **Níveis:** São os valores que os fatores podem assumir.
- **Variável de resposta:** É o resultado de um experimento, a métrica a qual se pretende avaliar, consiste no objetivo do experimento.

Embora existam diversas variações de DOE, Jain (1991) define os tipos mais comuns que são: Planejamento simples, Planejamento fatorial completo e Planejamento fatorial fracionado. No Planejamento simples, a partir de uma configuração inicial, um único fator é alterado a cada execução do experimento enquanto o desempenho é avaliado. Esse tipo de DOE apresenta eficiência reduzida e pode levar a conclusões equivocadas se os fatores possuírem interação entre si (dependência entre fatores). A quantidade de experimentos possíveis com o Planejamento simples é representada na Equação 2.11.

$$n = 1 + \sum_{i=1}^{k} (n_i - 1), \tag{2.11}$$

em que k corresponde ao número de fatores e n_i é o número de níveis no i-ésimo fator.

Por sua vez, o Planejamento fatorial completo tem como principal característica a utilização de todas as possíveis combinações de fatores e níveis. Como consequência, esse tipo de planejamento possui alto custo de implementação. O Planejamento fatorial completo também possibilita a avaliação das interações entre os fatores. Um estudo com o Planejamento fatorial completo com k fatores e i-ésimo fator com n_i níveis reque n experimentos, conforme representado na Equação 2.12.

$$n = \prod_{i=1}^{k} n_i \tag{2.12}$$

O Planejamento fatorial fracionado tem como principal objetivo reduzir a quantidade de experimentos em relação ao Planejamento fatorial completo. Essa redução ocorre pela utilização de uma fração do Planejamento fatorial completo. Por exemplo, em um Planejamento fatorial completo com 4 fatores e 3 níveis em cada fator seria necessária a realização de 3^4 experimentos, ou seja, 81 execuções. Por outro lado, um Planejamento fatorial fracionado poderia reduzir o número de execuções para 3^{4-2}, ou seja, 9 execuções. Embora possua custo reduzido quando comparado com o Planejamento fatorial completo, o Planejamento fatorial fracionado fornece menos informação.

Este capítulo apresentou os conceitos básicos necessários à compreensão do tema abordado neste estudo. Conceitos relacionados a computação em nuvem, dependabilidade, design experimental, técnicas de previsão de falhas e árvore de decisão estão entre os temas abordados neste capítulo. O próximo capítulo apresenta um revisão do estado da arte da previsão de falhas em sistemas em nuvem.

3 Revisão Sistemática

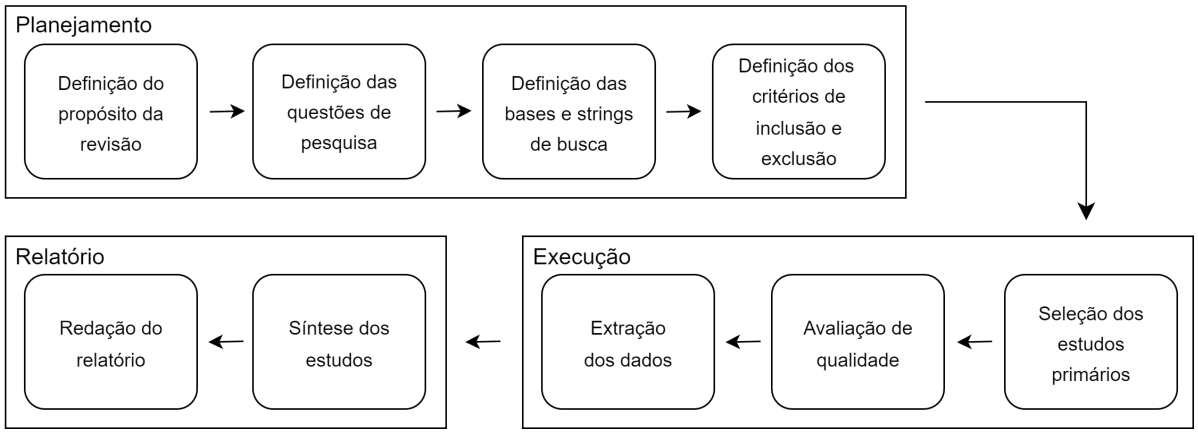
Este capítulo tem como objetivo apresentar uma revisão sistemática da literatura a respeito dos métodos e técnicas utilizadas no processo de previsão de falhas nos sistemas em nuvem. O capítulo está organizado nas seguintes seções: A Seção 3.2 apresenta o planejamento da revisão sistemática. A Seção 3.3 detalha a execução da pesquisa. A Seção 3.4 apresenta a análise dos resultados. A Seção 3.5 apresenta a discussão dos resultados. Por fim, a Seção 3.6 conclui o capítulo.

3.1 Introdução

Uma revisão sistemática da literatura (RSL) é um meio de interpretar e avaliar pesquisas relevantes disponíveis para responder a uma questão de pesquisa ou avaliar um tópico de uma área específica ou fenômeno de interesse (STAPIC *et al.*, 2012). A revisão sistemática da literatura busca caracterizar, sintetizar e comparar os últimos estudos referentes a um determinado tema de pesquisa, possibilitando a transferência de conhecimento na comunidade de pesquisa (KASHANI; MAHDIPOUR, 2022).

Nesse sentido, esta revisão segue os procedimentos apresentados por Majid e Anuar (2022), definidos como: planejamento, execução e relatório. A Figura 8 apresenta uma síntese dos procedimentos utilizados.

Figura 8 – Fluxograma da revisão sistemática.



Fonte: Adaptado de Dias *et al.* (2021).

3.2 Planejamento de Pesquisa

O planejamento da RSL inicia-se com a definição de um protocolo de revisão, ou seja, a definição das regras que serão utilizadas na condução da RSL, mitigando um possível viés de pesquisa em razão das expectativas dos pesquisadores (DIAS *et al.*, 2021). No protocolo de revisão são definidos os seguintes aspectos: objetivo da pesquisa, questões de pesquisa, palavras-chave que serão utilizadas nos mecanismos de busca, critérios de seleção das bases de busca, idioma das publicações, métodos de busca, bases de buscas, critérios de inclusão e exclusão, critérios de qualidade, bem como os dados que serão extraídos dos estudos com objetivo de responder as questões de pesquisa.

3.2.1 Questões de Pesquisa

Este capítulo busca revisar o estado da arte de forma sistemática, analisando os métodos mais recentes utilizados no processo de previsão de falhas em sistemas de computação em nuvem e, como resultado, responder as seguintes questões de pesquisa:

- Quais os métodos utilizados para previsão de falhas em sistemas em nuvem?
- Quais as características dos datasets utilizados na previsão de falhas nos sistemas em nuvem?
- Quais as métricas utilizadas na avaliação de modelos de previsão de falhas em sistema em nuvem?

3.2.2 Fontes de Busca

Foram selecionadas quatro bases de busca cientificamente reconhecidas nessa revisão sistemática. Além da relevância das bases, também foram levadas em consideração a possibilidade de utilização com acesso acadêmico, possibilidade de busca por palavras-chave e exportação dos resultados. As bases escolhidas foram:

- ACM Digital Library
- IEEE Xplore
- Scopus
- SpringerLink

3.2.3 Estratégia de Busca

Nesta fase da RSL foram definidas as palavras-chave e respectivos sinônimos utilizados nas bases de busca. Após uma pesquisa exploratória e estudo dos trabalhos, verificou-se os termos recorrentes utilizados pela comunidade científica, possibilitando a extração das seguintes palavras-chave: *fault prediction, failure prediction, cloud computing, cloud service, cloud infrastructure, cloud platform e cloud datacenter.*

A string de busca adequada à cada mecanismo de busca foi definida após vários testes, pois uma string geral geraria milhares de resultados, enquanto uma string muito específica eliminaria diversos resultados relevantes. Após realizar uma pesquisa prévia manual em cada base de busca, as peculiaridades de cada mecanismo de busca puderam ser verificadas, e com o uso dos operadores booleanos (AND, OR), foram definidas as strings de busca adequadas a cada mecanismo. Na Tabela 1, é possível visualizar as bases de busca utilizadas, bem como as respectivas strings de busca.

Tabela 1 – String por base de busca

Base de Busca	**String de Busca**
ACM Digital Library	[[All: “fault prediction”] OR [All: “failure prediction”]] AND [[All: “cloud computing”] OR [All: “cloud service”] OR [All: “cloud infrastructure”] OR [All: “cloud platform”] OR [All: “cloud datacenter”]]
IEEE Xplore	((“Full Text & Metadata”:“fault prediction”) OR (“Full Text & Metadata”:“failure prediction”)) AND ((“Full Text & Metadata”:“cloud computing”) OR (“Full Text & Metadata”:“cloud service”) OR (“Full Text & Metadata”:“cloud infrastructure”) OR (“Full Text & Metadata”:“cloud platform”) OR (“Full Text & Metadata”:“cloud datacenter”))
Scopus	(ALL (“fault prediction”) OR ALL (“failure prediction”) AND ALL (“cloud computing”) OR ALL (“cloud service”) OR ALL (“cloud infrastructure”) OR ALL (“cloud platform”) OR ALL (“cloud datacenter”))
SpringerLink	((“fault prediction”) OR (“failure prediction”)) AND ((“cloud computing”) OR (“cloud service”) OR (“cloud infrastructure”) OR (“cloud platform”) OR (“cloud datacenter”))

Fonte: O autor.

3.2.4 Critérios de Inclusão e Exclusão

Abaixo, são listados os critérios utilizados para inclusão e exclusão na RSL dos artigos retornados nas bases de busca.

Inclusão

- Publicação decorrente de estudos primários
- Publicação em língua inglesa
- Publicação disponível na íntegra

Exclusão

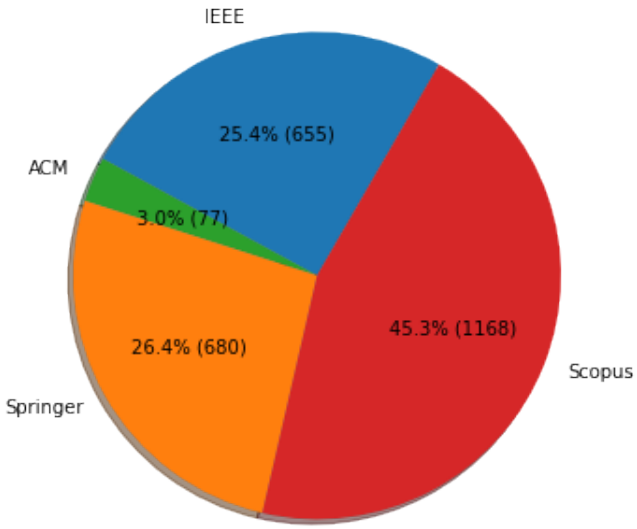
- Publicação decorrente de estudos secundários
- Publicação duplicada nas bases de busca
- Publicação que fuja ao escopo da pesquisa
- Publicação relacionada ao atributo segurança de dependabilidade
- Publicação em idioma diferente do inglês

3.3 Execução da Pesquisa

O processo de busca apresentou os seguintes critérios: Não foram considerados trabalhos com ano de publicação anterior a 2017, pois poderiam apresentar soluções obsoletas; também foram desconsiderados artigos de revisão, pois muitas vezes apresentam poucos detalhes a respeito dos métodos utilizados, descrevendo o estado da arte a partir de aspectos conceituais (SANTOS *et al.*, 2020). A Figura 9 apresenta os resultados da busca primária, onde um total de 2.580 artigos foram retornados, sendo 1.168 da base Scopus, 680 da Springer, 655 da IEEE e 77 da ACM.

Após a realização da busca primária, foram aplicados os critérios de inclusão e exclusão. Inicialmente, 119 artigos que correspondem a 4,61% do número total de artigos selecionados na busca primária foram eliminados por estarem duplicados nas bases de busca. Em seguida, foi realizada a leitura dos títulos e resumos, introdução e conclusão e finalmente, quando necessária, a leitura das obras na íntegra para verificar a relevância

Figura 9 – Resultado da busca primária.



Fonte: O autor.

e relação de cada trabalho com o escopo da pesquisa bem como a adequação com os critérios de inclusão e exclusão. Após as etapas de leitura necessárias, 2.421 artigos que correspondem a 93,8% do total de artigos selecionados na busca primária foram eliminados, resultando na inclusão de 40 artigos na revisão, o que corresponde a 1,55% do total de artigos.

3.4 Análise dos Resultados

A última etapa da RSL tem o objetivo de sintetizar e correlacionar os dados extraídos das publicações para formular respostas às questões de pesquisa (DIAS *et al.*, 2021). A Tabela 2 apresenta uma síntese dos artigos incluídos na RSL, onde são apresentadas as referências, métodos, anos de publicação, conjuntos de dados utilizados e métricas de avaliação. Para facilitar a análise dos resultados, os trabalhos estão indexados por um ID (identificador) no formato: TR_xx, onde xx refere-se a ordem do trabalho na tabela, utilizada para gerar um identificador único.

Tabela 2 – Síntese dos Artigos

ID	Referência	Ano	Método	Dataset	Métrica
TR_01	(PRATHIBHA, 2019)	2019	*K-Means, Decision Tree*, KNN	*WorkFlowSim*	Precisão 83,33%
TR_02	(SHETTY *et al.*, 2019)	2019	XGBoost	Google 2011 V.2	Precisão 92%, *Recall* 94,8%
TR_03	(LIU *et al.*, 2020b)	2020	SVM, XGBoost, *Random Forest*	Logs Baidu	Precisão 93,33%
TR_04	(MA *et al.*, 2020)	2020	RNN	Alibaba V.2018	Acurácia 97,1%
TR_05	(ADAMU *et al.*, 2017)	2017	SVM, Regressão Linear	*The Computer Failure Data Repository* (CFDR)	*
TR_06	(GUPTA *et al.*, 2017)	2017	LSTM, BI-LSTM	Logs do Doker	*Recall* 94%, Precisão 12%, F1 Score 22%
TR_07	(MARIANI *et al.*, 2018)	2018	IBM ITOA-PI	Logs de servidor SIP	*Recall* 92%, Precisão 93%, F1 Score 92%
TR_08	(JASSAS; MAHMOUD, 2021)	2021	ANN	Google 2009, Google 2011 V.2, Mustang Trinity	*Recall* 99%, Precisão 99%, F1 Score 99%
TR_09	(ISLAM; MANIVANNAN, 2017)	2017	LSTM	Google 2011 V.2	Acurácia 87%
TR_10	(LIU *et al.*, 2017)	2017	OS-ELM, OS-SVM	Google 2011 V.2	Precisão 93%, Acurácia 93%
TR_11	(DAS *et al.*, 2020)	2020	LSTM	Logs Hadoop e Cassandra	Precisão 88%, *Recall* 86%, Acurácia 80%
TR_12	(GAO *et al.*, 2020)	2020	BI-LSTM	Google 2011 V.2	Acurácia 93%, AUC 90%
TR_13	(ALAHMAD *et al.*, 2021)	2021	ANN e CNN	Google 2011 V.2, Trinity, Alibaba V.2017	Acurácia 94%

TR_14	(SU *et al.*, 2019)	2020	LSTM	Backblaze (Smart HD)	*
TR_15	(PADMAKUMARI; UMAMAKESWARI, 2019)	2019	*Naive Bayes*, *Random Forest*, *Rule-based*, MF2N2	*WorkFlowSim*	*Recall* 92,1%, Especificidade 95,1%, F1 Score , AUC 97,7%
TR_16	(CHEN *et al.*, 2019)	2019	XGBoost	Microsoft Cloud	Precisão 82,75%, *Recall* 76,74%, F1 Score 79,63%
TR_17	(BHATTACHARYYA *et al.*, 2017b)	2017	SVM	Google 2011 V.2 e *BugBench*	Acurácia 100%
TR_18	(LIN *et al.*, 2018)	2018	*Random Forest*, LSTM	Sistema X Alibaba	*Recall* 63,5%, Precisão 92,4%, F1 Score 75,2%
TR_19	(LUO *et al.*, 2021)	2021	NTAM	Microsoft Azure (Smart HD) Backblaze (SMART HD)	*Recall* 57,58%, Precisão 81,7%, F1 Score 67,34%
TR_20	(CHAKRABORTTII; LITZ, 2020)	2020	*Isolation Forest*, *Autoencoder*	Google Cloud SSD	AUC 99%
TR_21	(LI *et al.*, 2020)	2020	MING, LSTM, *Random Forest*	Sistema X Alibaba	AUC 92%
TR_22	(TEHRANI; SAFI-ESFAHANI, 2017)	2017	SVM	Simulado no CloudSim	Precisão 99%, Acurácia 99,89%
TR_23	(VU *et al.*, 2021)	2021	LSTM, BI-LSTM	Cloud Bitbrain	Acurácia 97%
TR_24	(KHALIL *et al.*, 2017)	2017	SVM	Google 2011 V.2	Acurácia 99,04%, Precisão 96,51%, *Recall* 90,22%, F1 Score 93,26%
TR_25	(LIU *et al.*, 2020)	2020	RMTL	Google 2011 V.2	Acurácia 97%, F1 Score 98,46%
TR_26	(NAM *et al.*, 2021)	2021	CNN	Logs de VM OpenStack	Acurácia 95%, F1 Score 67%
TR_27	(LIU *et al.*, 2020a)	2020	RNN	Smart Dataset W, M e S.	Acurácia 95%

TR_28	(RAWAT *et al.*, 2021)	2021	ARIMA	Simulados em R	RMSE 0,0457443 MAE 0,0344786 MASE 0,6036391
TR_29	(JASSAS; MAHMOUD, 2020)	2020	*Decision Tree*, *Random Forest*, *Naive Bayes*, QDA	Google 2011 V.2, Mustang e Trinity	Precisão 99%, *Recall* 99%, F1 Score 99%
TR_30	(CHHETRI *et al.*, 2022)	2022	*Random Forest*, GB, LSTM, GRU	Nuvem Universidade de Tartu	Precisão 99%, Acurácia 99%, *Recall* 100%, F1 Score 99%
TR_31	(GOLLAPALLI; MAISSA, 2022)	2022	ANN e SVM	Google 2019	Precisão 99,8%, Acurácia 99,8%
TR_32	(ABRO *et al.*, 2022)	2022	*Naive Bayes*, *Random Forest*, Regressão Linear	Simulado no CloudSim	Acurácia 99%, Sensibilidade 98%, Especificidade 99%
TR_33	(YANG; KIM, 2022)	2022	LSTM	Logs OpenStack	Precisão 96,1%, *Recall* 95%, F1 Score 95,5%, AUC 98%
TR_34	(JASSAS; MAHMOUD, 2022)	2022	*Random Forest*, *Decision Tree*, QDA, Naive Bayes, XGBoost	Google 2011 V.2, Mustang e Trinity	Precisão 98%, *Recall* 95%, F1 Score 97%, AUC 100%
TR_35	(ASMAWI *et al.*, 2022)	2022	XGBoost, *Random Forest*, Regressão Logistica, *Decision Tree*, LSTM	Google 2011 V.2	Precisão 94,31%, Acurácia 94,35%, Sensibilidade 91,92%, Especificidade 96,07%, F1 Score 93,1%
TR_36	(MOHAMMED *et al.*, 2019)	2019	ARIMA, KNN, SVM, *Random Forest*	CFDR NERSC 2019	Acurácia 90,76%, Sensibilidade 67,53%
TR_37	(SAXENA; SINGH, 2022a)	2022	SVM, FNN, Regressão Linear	Google 2011 V.2	Acurácia 97,8%

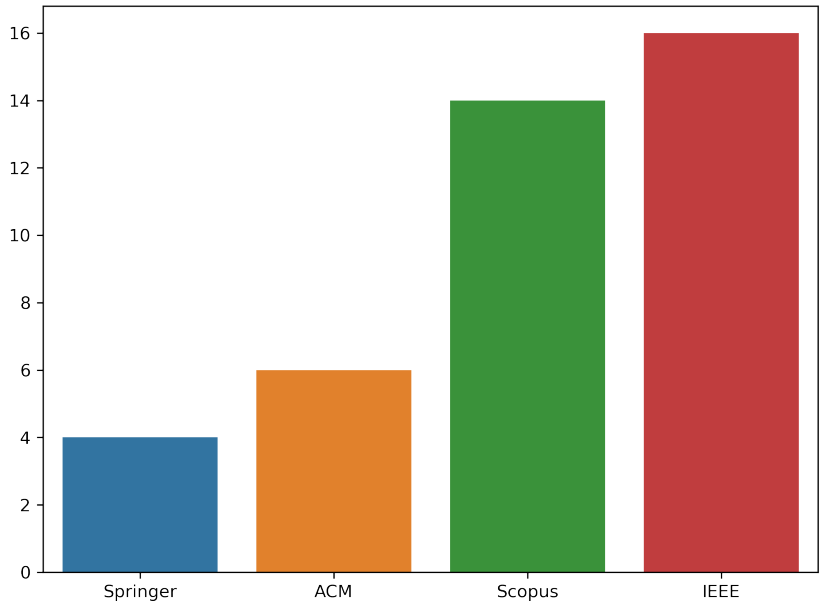
TR_38	(LIANG *et al.*, 2022)	2022	*Random Forest*	*Backblaze* (Smart HD)	Precisão 98,44%, Acurácia 97,57%
TR_39	(ZHANG *et al.*, 2022)	2022	XGBoost	Alibaba	Precisão 48,65%, *Recall* 79,34%, F1 Score 60,31%
TR_40	(SAXENA; SINGH, 2022b)	2022	SVM, *Random Forest*, Regressão Linear, NN	Google 2011 V.2	Acurácia 98,14%

Fonte: O autor.

3.4.1 Base de Publicações

A Figura 10 apresenta a distribuição dos artigos incluídos na revisão sistemática por base de publicação. Embora a base de busca Scopus tenha retornado 45,3% (1.168) do total de artigos na busca primária, a base de busca IEEE foi a que contou com a maior quantidade de artigos incluídos, 16 artigos, seguida das bases Scopus, ACM e Springer com 14, 6 e 4 artigos, respectivamente.

Figura 10 – Artigos aceitos por base de busca.

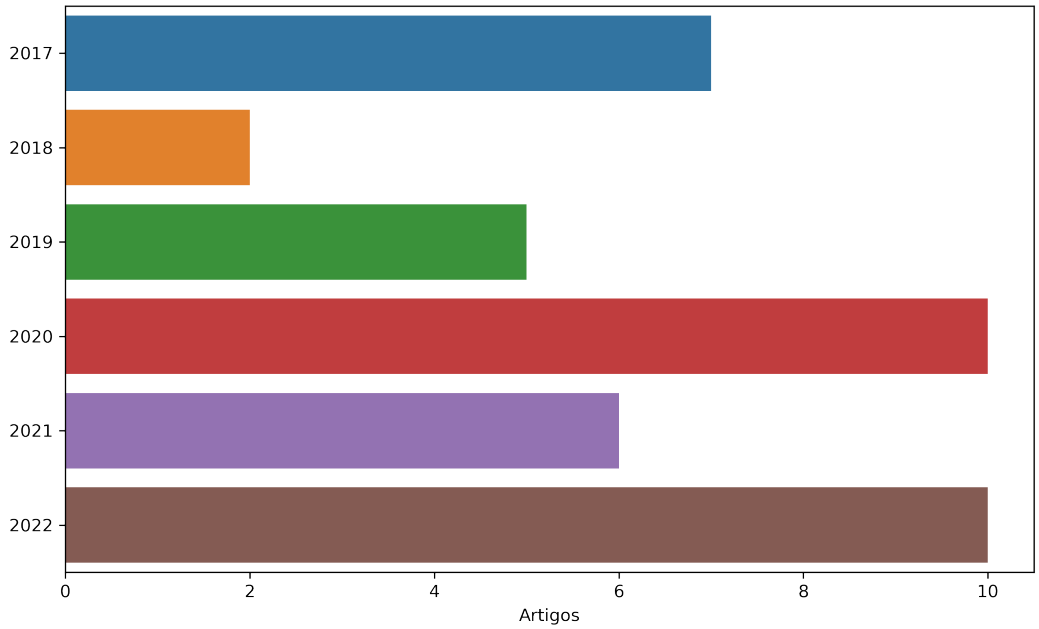


Fonte: O autor.

3.4.2 Anos de Publicações

A Figura 11 apresenta a distribuição dos trabalhos incluídos na revisão sistemática por ano de publicação. É possível observar uma maior concentração de publicações nos anos de 2020 e 2022, indicando maior interesse da comunidade acadêmica por esse tema de pesquisa.

Figura 11 – Artigos aceitos por ano de publicação.

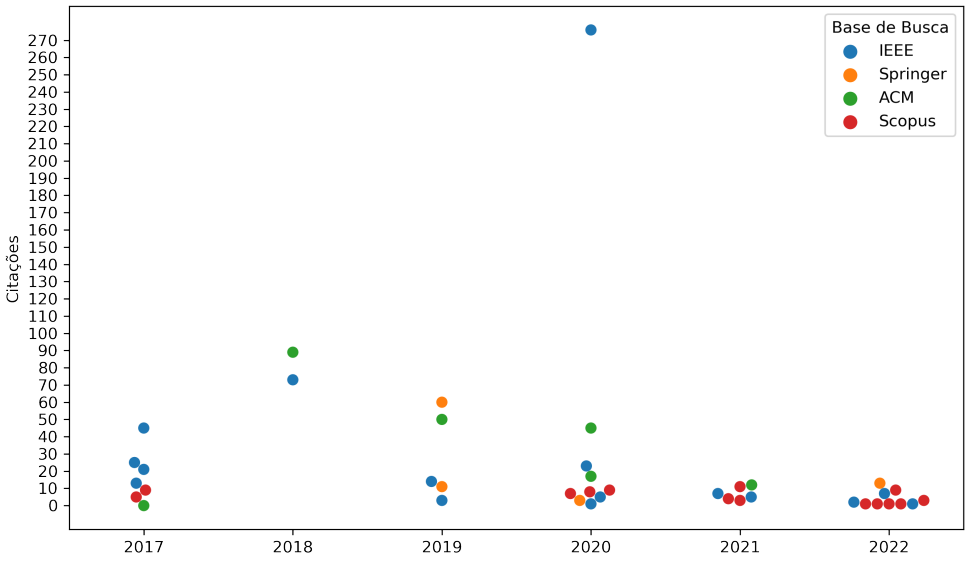


Fonte: O autor.

3.4.3 Citações das Publicações

Na Figura 12, é possível visualizar o número de citações dos artigos aceitos na RSL em relação às bases de busca e anos de publicação. Destaca-se o trabalho de Gao *et al.* (2020) com 276 citações. De forma geral, mesmo em trabalhos mais recentes, observa-se um número expressivo de citações, demonstrando a relevância do tema aqui abordado.

Figura 12 – Citações por ano e base de busca.



Fonte: O autor.

3.5 Discussão dos Resultados

Nesta seção, são discutidos os principais resultados obtidos com o objetivo de responder às questões de pesquisa. Também são apresentadas observações com base nos trabalhos analisados.

3.5.1 Datasets

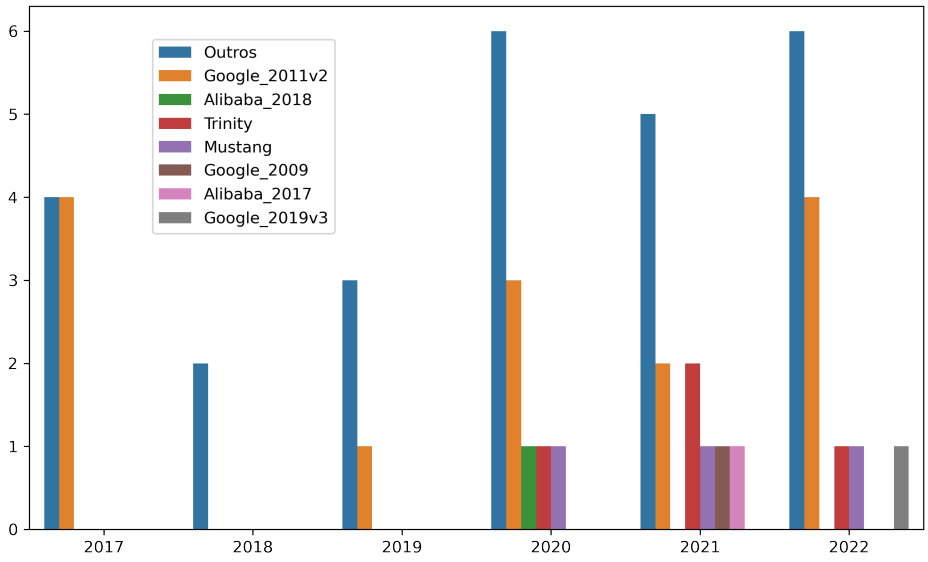
O gerenciamento de falhas na computação em nuvem e o desenvolvimento de ambientes de nuvens confiáveis dependem de dados de falhas de sistemas do mundo real. No entanto, um número escasso desses conjuntos de dados foi disponibilizado publicamente (ISLAM; MANIVANNAN, 2017). Alguns desses conjuntos de dados mais difundidos foram relatados nos artigos analisados. Conforme apresentado na Tabela 2, entre os conjuntos de dados utilizados nos trabalhos incluídos na RSL estão, na ordem de predominância:

- O dataset Google 2011v2 utilizado em catorze publicações: TR_02 , TR_08, TR_09, TR_10, TR_12, TR_13, TR_17, TR_24, TR_25, TR_29, TR_34, TR_35, TR_37 e TR_40;
- O dataset Trinity utilizado em quatro publicações: TR_08, TR_13, TR_29 e TR_34;

- O dataset Mustang utilizado em três publicações: TR_08, TR_29 e TR_34;
- Os datasets Alibaba v.2018 TR_04, Alibaba v.2017 TR_13, Google 2009 TR_08 e Google 2019v3 TR_31 utilizados em um trabalho.

As demais publicações utilizaram outros datasets pouco difundidos ou gerados pelos próprios autores e não disponibilizados publicamente. A Figura 13 apresenta a distribuição dos dataset por ano de publicação.

Figura 13 – Datasets utilizados nos artigos incluídos na RSL.



Fonte: O autor.

O dataset Google 2011v2 é o conjunto de dados mais popular e consequentemente foi o mais utilizado nos últimos anos. Seis anos após seu lançamento, o dataset já havia sido utilizado em mais de 450 publicações (AMVROSIADIS *et al.*, 2018). O dataset Google 2011v2 foi o conjunto de dados utilizado neste trabalho e será descrito na Seção 5.2. Embora o dataset Google 2009 tenha sido utilizado com propósitos de comparação por Jassas e Mahmoud (2021), atualmente esse conjunto de dados é considerado obsoleto e sua utilização não é recomendada. O Google ainda disponibilizou uma terceira versão do dataset, Google 2019v3 (WILKES, 2020), entretanto um único trabalho identificado nesta RSL utilizou esse conjunto de dados (GOLLAPALLI; MAISSA, 2022).

Com o objetivo de reduzir a dependência excessiva do dataset disponibilizado pelo Google, os autores Amvrosiadis *et al.* (2018) lançaram o repositório Atlas que, entre outros conjuntos de dados, é composto pelos datasets Trinity e Mustang, dois conjuntos

de dados coletados do *Los Alamos National Laboratory*. O dataset Trinity, segundo mais predominante nesta RSL, consiste em um registro de eventos de supercomputador composto por 9.408 nós de computação idênticos, com um total de 301.056 núcleos Intel Xeon E5-2698v3 de 2.3 GHz e 1.2 PB de RAM. Essas especificações de hardware tornam este o maior cluster com dataset disponibilizado publicamente por número de núcleos de CPU. Os dados abrangem um período de três meses, entre fevereiro e abril de 2017, e consistem em 25.237 trabalhos de 88 usuários. Por sua vez, o dataset Mustang abrange um período de 61 meses, de outubro de 2011 a novembro de 2016, o que o torna o dataset mais longo disponível publicamente. O cluster conta com 1.600 nós de computação idênticos com um total de 38.400 núcleos AMD Opteron 6176 de 2.3 GHz e 102 TB de RAM. O dataset consiste em 2.1 milhões de trabalhos de 565 usuários (JASSAS; MAHMOUD, 2021)(AMVROSIADIS *et al.*, 2018).

Outro conjunto de dados identificado neste trabalho foi o dataset Alibaba 2018 que consiste em um registro de eventos de um cluster com 4.000 máquinas, 9.000 serviços e 4 milhões de trabalhos no período de 8 dias. O dataset apresenta detalhes do ciclo de vida dos trabalhos e tarefas. Por motivo de confidencialidade, as informações de hardware como memória e disco foram normalizadas nos datasets (GUO *et al.*, 2019).

Em oposição aos trabalhos anteriores, alguns autores utilizaram seus próprios conjuntos de dados obtidos por meio de registros de eventos não disponibilizados publicamente ou datasets sintéticos obtidos em ambientes de simulação. Por exemplo, Chen *et al.* (2019) utilizaram um dataset da nuvem Microsoft gerado pelos próprios autores. Devido a confidencialidade, não foram apresentadas informações relevantes a respeito do dataset, tampouco foi disponibilizado publicamente. Luo *et al.* (2021) utilizaram dois conjuntos de dados estatísticos de unidades de disco gerados do Microsoft Azure, não disponibilizados publicamente. Os autores Zhang *et al.* (2022) utilizaram um conjunto de dados resultante de um registro de eventos da nuvem Alibaba por um período de um ano, não disponibilizado publicamente. Os autores Prathibha (2019); Padmakumari e Umamakeswari (2019); Tehrani e Safi-Esfahani (2017); Rawat *et al.* (2021) e Abro *et al.* (2022) utilizaram datasets obtidos a partir de estratégias de simulação, impossibilitando a reprodução dos experimentos.

Em relação a disposição dos dados, ou seja, como os dados estão formatados, foi observado que alguns conjuntos de dados disponibilizados publicamente, como Google

2011v2 e Google 2019v3 possuem dados ofuscados ou normalizados. A ofuscação tem o objetivo de manter a confidencialidade dos dados considerados sigilosos pela organização. Por sua vez, a normalização é uma técnica comumente utilizada nas abordagens de modelagem preditiva com o objetivo de redimensionar os dados para um intervalo comum, contribuindo para o aprendizado dos algoritmos de aprendizagem de máquina (SINGH; SINGH, 2020). Tanto o conjunto de dados de 2011 quanto o de 2019 possuem os dados referentes à utilização de memória RAM e CPU normalizados para o valor máximo de 1. Essa característica impossibilita a identificação dos valores reais disponíveis e utilizados nas operações do sistema. Por exemplo, não é possível identificar a quantidade de memória e capacidade de processamento disponíveis nos servidores.

Embora tenha passado mais de uma década desde o lançamento do dataset Google 2011 v2, nesta pesquisa ficou evidenciada a preferência dos pesquisadores por esse conjunto de dados em detrimento dos demais identificados neste trabalho e disponibilizados publicamente. Essa preferência reforça a dependência excessiva apontada por Amvrosiadis *et al.* (2018). Como alternativa ao dataset de 2011v2, o dataset Google 2019v3 lançado no ano de 2020, pode ser utilizado nas pesquisas relacionadas à previsão de falhas em nuvem. Por se tratar de um dataset mais recente, pode refletir melhor a realidade dos sistemas atualmente utilizados.

3.5.2 Desbalanceamento de classes

Uma outra característica em relação aos conjuntos de dados é o desbalanceamento de classes. Entre os trabalhos analisados, poucos autores utilizaram técnicas para lidar com o desbalanceamento de classes. Os autores Shetty *et al.* (2019) e Chen *et al.* (2019) utilizaram *Synthetic Minority Oversampling Technique* (SMOTE) que consiste em uma técnica de amostragem da classe minoritária com o objetivo de gerar amostras sintéticas, reduzindo ou eliminando o desbalanceamento entre as classes. Embora os autores Lin *et al.* (2018) também tenha utilizado SMOTE, relataram resultados pouco satisfatórios, com altas taxas de falsos positivos.

Li *et al.* (2020) utilizaram uma técnica de *oversampling* aprimorada pelos autores para selecionar mais amostras positivas, capturando mais sintomas de falhas. Nos experimentos, os autores relataram melhores resultados com o uso da técnica proposta

em comparação com outras técnicas de amostragem. Em virtude do desbalanceamento excessivo dos dados utilizados, Nam *et al.* (2021) utilizaram *oversampling* em 2 vezes nos dados de falhas e *undersampling* em 60 vezes nos dados normais.

Por outro lado, Luo *et al.* (2021) propuseram um método para lidar com o desbalanceamento de classes denominado *Temporal Progressive Sampling* (TPS). O método proposto gera novas amostras da classe minoritária em função do tempo e é definido pelos autores como um método para aprimoramento de dados. Os demais trabalhos incluídos nesta revisão sistemática não utilizaram nenhuma técnica para tratar o desbalanceamento de classes.

3.5.3 Métodos

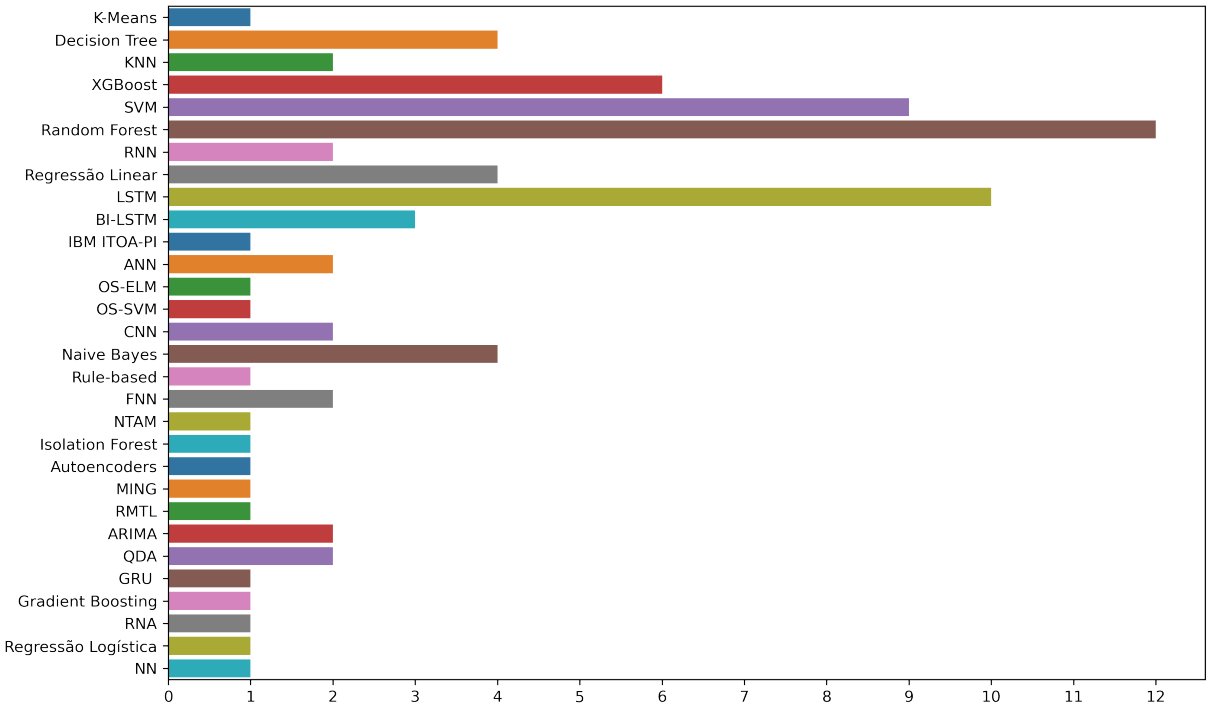
Em relação às técnicas de previsão de falhas utilizadas, conforme apresentado na Tabela 2, observa-se, na ordem de predominância, a utilização dos algoritmos:

- *Random Forest* utilizado nos trabalhos: TR_21, TR_15, TR_18, TR_21, TR_29, TR_30, TR_32, TR_34, TR_35, TR_36, TR_38 e TR_40, totalizando doze artigos com publicações em todos os anos abrangidos por esta pesquisa, com exceção dos anos de 2017 e 2021;
- *Long Short Term Memory* (LSTM) utilizado nos trabalhos: TR_06, TR_09, TR_11, TR_14, TR_18, TR_21, TR_23, TR_30, TR_33 e TR_35;
- *Support Vector Machine* (SVM) utilizado em nove publicações: TR_03, TR_05, TR_17, TR_22, TR_24, TR_31, TR_36, TR_37 e TR_40;
- O algoritmo XGBoost utilizado em seis trabalhos: TR_02, TR_03, TR_16, TR_34, TR_35 e TR_39;
- *Naive Bayes* utilizado em quatro trabalhos: TR_15, TR_29, TR_32 e TR_34;
- Regressão Linear utilizada em quatro trabalhos: TR_05, TR_32, TR_37 e TR_40;
- *Decision Tree* utilizado em quatro trabalhos: TR_01, TR_29, TR_34 e TR_35;
- *Bidirectional long short term memory* (BI-LSTM) utilizado em três trabalhos: TR_06, TR_12 e TR_23.

Os demais algoritmos apresentados na Tabela 2 foram utilizados por um ou dois autores. A Figura 14 apresenta a distribuição dos algoritmos nos trabalhos analisados.

Os autores Islam e Manivannan (2017) propuseram um modelo para prever falhas

Figura 14 – Algoritmos utilizados nos artigos incluídos na RSL.



Fonte: O autor.

de trabalhos e tarefas de aplicações em nuvem com o algoritmo LSTM. Nos experimentos, os autores constataram alta correlação entre tempo e consumo de recursos com tarefas que falharam, além de apresentarem um método para estimar uma redução no consumo de recursos como CPU e memória em decorrência da previsão de falhas.

Embora os autores Islam e Manivannan (2017) tenham apontado o algoritmo SVM como inadequado para previsão de falhas com o dataset do Google 2011v2, Khalil *et al.* (2017) propuseram um método para prever falhas de máquinas por sobrecarga de trabalhos em um sistema em nuvem com uso do SVM. Para isso, o modelo foi treinado somente com as informações de CPU e memória. O uso do SVM foi possível graças ao método que os autores utilizaram para categorizar as falhas. Selecionaram as tarefas de maior prioridade e calcularam as utilizações máximas de CPU e memória desse conjunto de tarefas, possibilitando a classificação das próximas tarefas de acordo com o consumo de CPU e memória, ou seja, caso o consumo atual de CPU ou memória de uma tarefa seja maior que o consumo máximo calculado, pressupõe-se uma sobrecarga de utilização de recursos e iminente falha.

Em contraste com o trabalho anterior, Liu *et al.* (2020b) alcançaram melhores resultados em previsão de falhas em nuvem com o algoritmo *Random Forest* em detrimento do algoritmo SVM. Os autores modelaram e implantaram a ferramenta COMPASS que tem o objetivo de prever falhas de hardware nos servidores da nuvem Baidu. A solução foi avaliada durante um período de 3 meses e alcançou 93.33% de precisão para previsões de falhas com até 30 minutos de antecedência. Com base na análise dos dados da nuvem Baidu, constatou-se que 63% das interrupções do sistema são provocadas por falhas de hardware.

Ainda no contexto de previsão de falhas em hardware na nuvem, Chakraborttii e Litz (2020) propuseram um método para prever falhas em SSDs em servidores na nuvem com uso dos algoritmos *Isolation Forest* e *Autoencoder*. Na abordagem, utilizaram um dataset com dados de mais de 30.000 SSDs coletados por um período de seis anos da nuvem do Google, e utilizaram somente a classe majoritária (sem falhas) no processo de treinado. O método proposto possui as vantagens de não sofrer com *overfitting* (sobreajuste aos dados de treinamento), consegue prever falhas inéditas não vistas no conjunto de teste, além de não ser afetado pelo desbalanceamento de classes, comum em modelos de classificação.

Em conformidade com a pesquisa recente (BHANAGE *et al.*, 2021) ficou evidenciada neste trabalho a predominância de técnicas de aprendizagem de máquina nas abordagens de previsão de falhas. Especificamente, algoritmos baseados em árvore como *Random Forest* e *Decision Tree*. Possivelmente, essa tendência pode ser explicada pelo fato dos modelos baseados em *Random Forest* apresentarem excelentes resultados, superando técnicas como SVM e Redes Neurais (JAUK *et al.*, 2019). Em concordância com Bhanage *et al.* (2021) constatou-se nesta pesquisa um aumento no uso de técnicas de aprendizagem profunda nos últimos anos. Em particular, o algoritmo LSTM tem sido amplamente utilizado.

Em relação à natureza das falhas as quais se pretende prever nos sistemas em nuvem, observou-se a predominância de métodos voltados à previsão de falhas de trabalhos e tarefas (14 artigos), seguido por falhas de servidor (10 artigos), serviços ou software de forma geral (6 artigos), falhas de máquina virtual (5 artigos) e falhas de disco (5 artigos).

É importante notar que em relação aos métodos que utilizaram o conjunto de dados do Google 2011v2, alguns autores consideraram mais de um estado final dos trabalhos e tarefas como falhas. Por exemplo, Shetty *et al.* (2019) e Bhattacharyya *et al.* (2017b)

classificaram as tarefas encerradas como falhas. Por outro lado, Islam e Manivannan (2017), Liu *et al.* (2017) e (LIU *et al.*, 2020) classificaram as tarefas removidas e encerradas como falhas. Além das tarefas removidas e encerradas, Asmawi *et al.* (2022) classificaram as tarefas e trabalhos perdidos como falhas. A classificação desses estados de trabalhos e tarefas como falhas deve ser cuidadosamente avaliada, pois pode impactar nos resultados finais dos métodos propostos, uma vez que adicionam dados aos modelos.

Um outro aspecto importante relacionado aos métodos que utilizaram os dados do Google é a quantidade de dados utilizada nos modelos de previsão de falhas. Embora o conjunto de dados Google 2011v2 consista de um registro de eventos de 29 dias, alguns autores utilizaram uma fração desses dados, possivelmente devido às limitações de recursos de computação para realização dos experimentos. Liu *et al.* (2017) utilizaram as doze primeiras horas do dataset, enquanto Liu *et al.* (2020) utilizaram os três primeiros dias. Por sua vez, Jassas e Mahmoud (2020) se limitaram à utilização dos primeiros sete dias. Por outro lado, Asmawi *et al.* (2022) utilizaram os primeiros catorze dias do dataset. Como uma fração dos dados pode não representar o comportamento da totalidade dos dados, os resultados apresentados podem ser distorcidos.

3.5.4 Métricas

Além das métricas geralmente utilizadas na avaliação de modelos de previsão de falhas, alguns autores apresentaram outras métricas significativas na avaliação da abordagem proposta. Devido à particularidade do método utilizado por Chakraborttii e Litz (2020), onde somente a classe majoritária foi utilizada no processo de treinamento do modelo, os autores utilizaram a curva *Receiver Operating Characteristic* (ROC) *area under the curve* (AUC) (ZHANG *et al.*, 2022) para avaliar o modelo. A utilização de métricas tradicionais poderiam apresentar resultados distorcidos, uma vez que havia a possibilidade de *overfitting* para classe majoritária (prever todas as amostras para classe majoritária). Os autores também apresentaram as métricas referentes ao tempo de treinamento para cada algoritmo avaliado. De forma semelhante, mas com objetivo de manter o acordo de confidencialidade com o Alibaba, Li *et al.* (2020) utilizaram exclusivamente a curva ROC AUC na avaliação do método proposto. Também avaliaram o desempenho do método com diferentes janelas de previsão, obtendo melhores resultados com o algoritmo *Random Forest* que obteve AUC de 92% para previsões entre 2 e 48 horas de antecedência.

Além da acurácia do modelo proposto, Gao *et al.* (2020) levaram em consideração a sobrecarga de tempo de treinamento e teste dos modelos avaliados, buscando equilíbrio entre ambas as métricas com objetivo de manter a qualidade do serviço na nuvem. Na abordagem, os autores alcançaram 90% de acurácia na previsão de falhas de trabalhos e tarefas com 15 minutos de antecedência. Entre os trabalhos que apresentaram o tempo para previsão de falhas, Das *et al.* (2020) foram os que apresentaram métricas de tempo de previsão mais curtas. Os autores apresentaram um *framework* para prever falhas em nós de computação em nuvem com tempo de previsão de 3 minutos. O *framework* obteve *recall*, precisão e acurácia de 86%, 88% e 80%, respectivamente. Por outro lado, Chakraborttii e Litz (2020) avaliaram o desempenho de dois diferentes algoritmos (*Isolation Forest* e *Autoencoder*) com tempos de previsão entre um e quatro dias, caracterizando a abordagem com maior tempo de previsão identificado nesta RSL.

Embora tenha apresentado o fluxograma de atividades do método proposto, Su *et al.* (2019) não apresentaram nenhuma métrica para avaliação da abordagem proposta, tampouco apresentaram os resultados obtidos na avaliação experimental. De forma semelhante, Adamu *et al.* (2017) não apresentaram a métrica utilizada na avaliação do método proposto, entretanto apresentou a quantidade de falhas previstas no dataset.

Saxena e Singh (2022a), Saxena e Singh (2022b), Alahmad *et al.* (2021), Bhattacharyya *et al.* (2017b), Ma *et al.* (2020), Islam e Manivannan (2017), Liu *et al.* (2020a) e Vu *et al.* (2021) utilizaram exclusivamente a acurácia na avaliação dos métodos propostos. Entretanto, o uso exclusivo dessa métrica não é adequado para avaliação de modelos com conjuntos de dados desbalanceados. Por exemplo, no contexto de previsão de falhas em tarefas, um modelo que sempre classifica todas as tarefas como sem falhas terá alta acurácia, mesmo que ignore as tarefas com falhas. Isso ocorre porque, nos conjuntos de dados, as falhas são exceções (JAUK *et al.*, 2019). Notaro *et al.* (2021) rotulam a acurácia como enganosa para avaliação da qualidade de previsões, pois na maioria das vezes essa métrica explora a assimetria dos dados. Embora a AUC seja apontada como apropriada para avaliação de modelos com dados desbalanceados (FAHIM; SILLITTI, 2019), somente seis autores utilizaram essa métrica (PADMAKUMARI; UMAMAKESWARI, 2019)(CHAKRABORTTII; LITZ, 2020)(LI *et al.*, 2020)(YANG; KIM, 2022)(GAO *et al.*, 2020)(JASSAS; MAHMOUD, 2022).

3.6 Conclusão

Tendo em vista que diversos métodos foram propostos para a previsão de falhas nos sistemas em nuvem, este capítulo apresentou uma revisão sistemática da literatura onde foram examinadas as características desses métodos, incluindo as técnicas utilizadas, os conjuntos de dados e as métricas empregadas para avaliar os métodos propostos. Nesse sentido, foram analisados 40 artigos de um total de 2.580 retornados na busca nas bases ACM, IEEE, Scopus e SpringerLink.

Muitos dos artigos analisados nesta Revisão Sistemática da Literatura não fornecem informações suficientes para permitir a reprodução adequada dos estudos. Isso dificulta a avaliação do método proposto, bem como sua reprodução e possível aprimoramento. Por exemplo, um número escasso de autores detalharam o pré-processamento dos dados utilizados. Informações como métodos utilizados para agregar as amostras no dataset e técnicas empregadas no balanceamento de classes (quando aplicável) foram omitidas.

Outra característica que dificulta ou impossibilita a reprodução dos estudos é a utilização de conjuntos de dados não disponibilizados publicamente. Nos trabalhos analisados foram relatados diversos conjuntos de dados pouco difundidos e não disponibilizados publicamente. Isso pode ser explicado pela escassez de conjuntos de dados de cargas de trabalho do mundo real.

Embora o dataset Google 2011v2 (predominante nesta pesquisa) tenha sido utilizado em várias publicações, o detalhamento precário dos métodos que utilizaram esse conjunto de dados, bem como a utilização de apenas parte do dataset na maioria dos experimentos evidenciam a necessidade de um aprofundamento nos estudos de previsão de falhas com esse conjunto de dados.

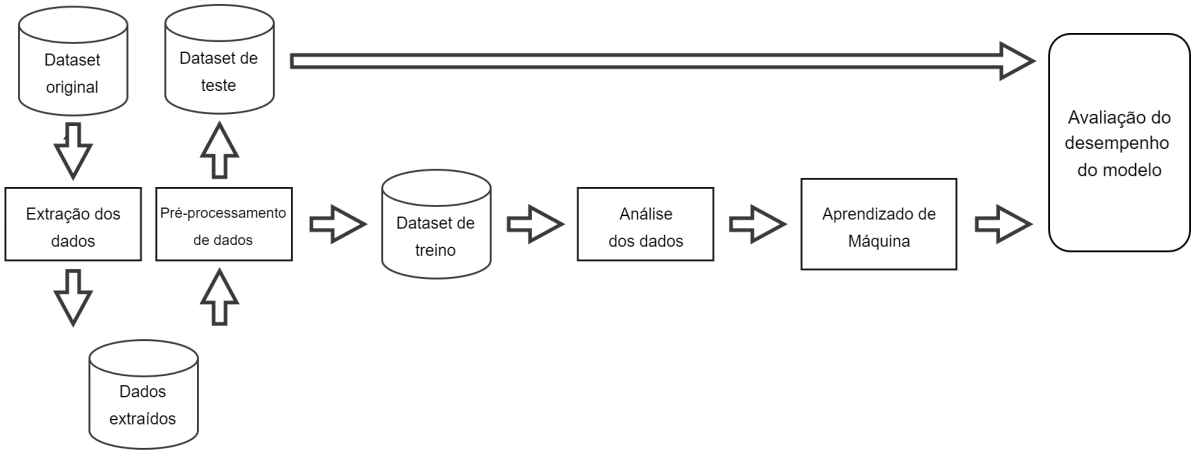
Em virtude dos fatos observados, o próximo capítulo apresenta uma abordagem experimental de previsão de falhas em sistemas em nuvem que inclui uma análise detalhada e caracterização das falhas no dataset Google 2011v2. Uma vez que alguns parâmetros, como a utilização de alguns recursos, podem ser indicadores de falhas, a relação entre utilização de recursos e a ocorrência de falhas nos sistemas em nuvem também é objeto de estudo.

4 Metodologia para Previsão de Falhas na Nuvem

Este capítulo apresenta a metodologia proposta para previsão de falhas de software em sistemas em nuvem. A metodologia proposta consiste no treinamento de modelos baseados em aprendizado de máquina com dados de cargas de trabalho de sistemas de computação em nuvem do mundo real para prever a ocorrência de falhas de software com até dez minutos de antecedência. As etapas da metodologia proposta incluem a extração dos dados de interesse do *dataset* original, pré-processamento dos dados, análise dos dados, implementação do aprendizado de máquina e avaliação de desempenho.

Como pode ser observado na Figura 15, a metodologia proposta tem início na extração dos dados de interesse do *dataset* original. Após o processo de extração, os dados de treino e teste são pré-processados em conjuntos de dados distintos. Em seguida, os dados de treino são analisados com o objetivo de identificar as variáveis com maior poder preditivo. A próxima etapa consiste no ajuste dos hiperparâmetros e no treinamento do modelo de aprendizado de máquina. Por fim, o desempenho do modelo é avaliado com os dados de teste. O detalhamento de cada etapa da abordagem proposta é apresentado nas próximas seções.

Figura 15 – Fluxograma da metodologia proposta.



Fonte: O autor.

4.1 Pré-processamento dos Dados

O pré-processamento dos dados é antecedido pela extração dos dados de interesse do *dataset* original. Por vezes, alguns conjuntos de dados possuem campos sem relevância para os objetivos da modelagem preditiva. Além disso, os dados de interesse podem estar em tabelas diferentes. Assim, essa etapa é responsável pela extração apenas dos dados que podem contribuir para o objetivo das previsões.

O pré-processamento de dados é fundamental no desenvolvimento de modelos preditivos, pois a utilização de dados inadequados resultará em modelos inválidos ou com baixo desempenho (LONES, 2021). Nesta etapa, para possibilitar a exploração das características e posterior limpeza dos dados, os registros em arquivos distintos são concatenados em um único aquivo. O próximo passo consiste na exclusão dos registros duplicados e tratamento dos valores NaN. Embora os valores NaN possam ser eliminados ou substituídos (EMMANUEL *et al.*, 2021), neste contexto, optou-se pela exclusão desses valores. Destaca-se que os valores NaN representam valores vazios, que apesar de serem comuns em grandes conjuntos de dados, precisam ser tratados ou removidos, pois sua utilização pode resultar em complicações ou perda de desempenho (MCKINNEY *et al.*, 2011).

Para evitar o vazamento de dados (quando os dados de teste tornam-se parte dos dados de treinamento), os dados de treinamento e teste devem ser divididos antes de atividades como a normalização dos dados ou a realização de cálculos com base nos registros. É importante salientar que a ordem dos registros no conjunto de dados deve ser preservada, ou seja, os dados de treinamento devem ser cronologicamente anteriores aos dados de teste (LONES, 2021).

Após a divisão dos dados, os registros de cada tarefa são agregados pela média aritmética. A agregação objetiva transformar todos os registros de uma determinada tarefa em um único registro. Neste processo, os últimos registros referentes aos últimos dez minutos de monitoramento de cada tarefa são desconsiderados, pois a intenção é prever a ocorrência de falhas com antecedência de dez minutos. Quando não disponível no conjunto de dados, a duração das tarefas pode ser obtida pelo cálculo da diferença entre os *timestamps* do primeiro e último registro da respectiva tarefa.

A atividade anterior resulta em um conjunto de dados, normalmente, com classes desbalanceadas. O desbalanceamento de classes é geralmente tratado com técnicas de

amostragem, como *undersampling* ou *oversampling* (LI *et al.*, 2020). Um dos critérios para a seleção da técnica de amostragem é a quantidade de dados disponíveis, pois a utilização de *undersampling*, por exemplo, pode levar à perda de informações potencialmente importantes para a classificação (KAUR; GOSAIN, 2018).

Uma vez realizado o balanceamento das classes, o próximo passo consiste na análise de correlação das variáveis no conjunto de dados. Com o objetivo de reduzir a quantidade de variáveis no conjunto de dados, bem como eliminar atributos redundantes que podem influenciar o desempenho do modelo, reduzindo a capacidade de generalização, o coeficiente de Pearson (ZHANG *et al.*, 2018) é utilizado para identificar as variáveis altamente correlacionadas.

Conforme descrito por Chen *et al.* (2021), o coeficiente de Pearson avalia o quão correlacionadas são duas ou mais variáveis. Essa correlação varia entre -1 e 1. Uma correlação maior que 0 indica que as variáveis são correlacionadas positivamente (diretamente proporcionais). Por outro lado, uma correlação menor que 0 indica que as variáveis são correlacionadas negativamente (inversamente proporcionais). Valores próximos a -1 ou 1 indicam uma correlação forte. Valores próximos a 0 indicam uma correlação fraca. Neste contexto, variáveis com correlação positiva ou negativa superior a 80% (correlação muito forte) são eliminadas.

4.2 Protocolo de Experimento

Nesta etapa são definidos os fatores e níveis que serão utilizados nos experimentos, assim como a quantidade de experimentos que serão realizados. Um experimento é definido como um ou mais testes nos quais são realizadas mudanças intencionais nas variáveis de entrada de um processo para identificar a influência sobre a variável de saída (MONTGOMERY, 2017). Diversas estratégias podem ser utilizadas no planejamento dos experimentos. A escolha da estratégia dependerá de vários fatores, incluindo o tempo e recursos disponíveis. Mais detalhes relacionados ao Planejamento de experimentos podem ser obtidos na Seção 2.5.

4.3 Aprendizado de Máquina

Uma vez realizado o planejamento dos experimentos, a próxima etapa consiste na modelagem preditiva com a utilização do algoritmo de aprendizado de máquina. Embora este trabalho utilize a implementação do algoritmo de árvore de decisão do *Scikit-learn* (PEDREGOSA *et al.*, 2011), outros algoritmos de classificação podem ser utilizados.

Assim como todo algoritmo de aprendizado de máquina, os algoritmos de árvore de decisão possuem hiperparâmetros que definem os aspectos relacionados à forma como o algoritmo irá aprender, influenciando significativamente a eficácia do modelo (MITROFANOV; SEMENKIN, 2021). Como não é possível determinar os valores ótimos de hiperparâmetros para cada tipo de problema, esses valores são frenquentemente definidos manualmente (ARDEN; SAFITRI, 2022).

Como conjuntos de dados com um grande número de variáveis podem apresentar informações redundantes ou irrelevantes, influenciando o desempenho do algoritmo (GARCÍA *et al.*, 2016), a importância das variáveis (*Feature Importance*) fornecida por modelos baseados em árvore de decisão pode ser utilizada para redução da quantidade de variáveis (ASMAWI *et al.*, 2022).

4.4 Avaliação de Desempenho

A última etapa consiste na avaliação do desempenho do modelo. Normalmente, as métricas utilizadas em modelos de classificação são acurácia, *recall*, precisão, e F1 score (JANIESCH *et al.*, 2021). Vale salientar que a acurácia pode apresentar resultados distorcidos, principalmente quando utilizada com conjuntos de dados desbalanceados (NOTARO *et al.*, 2021). Por outro lado, a AUC (*Area Under the Curve*) fornece estimativas precisas em relação ao desempenho desses modelos (ZHANG *et al.*, 2022).

Um aspecto importante na avaliação de desempenho do modelo é a apresentação detalhada das métricas para cada classe, em vez de se concentrar apenas no desempenho médio geral. A Seção 2.4.3 detalha as métricas utilizadas em problemas de classificação.

4.5 Considerações Finais

Este capítulo apresentou uma metodologia para previsão de falhas de software em sistemas de computação em nuvem. O próximo capítulo apresenta uma abordagem experimental onde são aplicados os conceitos e métodos aqui discutidos.

5 Estudo de Caso

Este capítulo apresenta uma abordagem experimental com o objetivo de prover um modelo de previsão de falhas de software em aplicações em nuvem. Para isso, foram realizados quatro experimentos de aprendizado de máquina, nos quais são avaliados os impactos das variações de carga de trabalho no desempenho dos modelos preditivos. O estudo de caso apresentado neste capítulo segue a metodologia proposta no Capítulo 4.

5.1 Introdução

O escopo deste trabalho limita-se a previsão de falhas de trabalhos e tarefas em aplicações em nuvem. Para isso, foi utilizado o conjunto de dados do Google 2011v2 (REISS *et al.*, 2011), descrito na Seção 5.2, bem como a implementação do algoritmo de árvore de decisão do Scikit-learn (PEDREGOSA *et al.*, 2011), descrito na Seção 2.4.2.

Os experimentos foram conduzidos em um computador com processador AMD Ryzen™3 4300U com *clock* de 2.70GHz, 4MB de cache e 4 Núcleos, 32 GB de memória RAM DDR4 3200 MHz e placa gráfica AMD Radeon™1400 MHz. O sistema operacional utilizado foi o Windows® 11 Pro versão 22H2. Em relação ao ambiente de desenvolvimento utilizado, todos os experimentos foram codificados na linguagem Python™[1] e executados no Jupyter Notebook™[2]. Diversas bibliotecas foram utilizadas neste trabalho, as principais são apresentadas na Tabela 3, juntamente com uma breve descrição de cada uma e suas respectivas referências.

No processo de extração de dados, descrito na Capítulo 4, as tabelas EVENTOS DE TAREFA (disponibilizada no Anexo A) e USO DE TAREFA (disponibilizada no Anexo B) são obtidas a partir do conjunto de dados Google 2011v2 . Como cada trabalho é composto por uma ou mais tarefas (REISS *et al.*, 2012), a ocorrência de falha em uma tarefa implica na falha do trabalho ao qual a tarefa pertence. Desse modo, este trabalho limitou-se à utilização dessas duas tabelas.

[1] https://www.python.org
[2] https://jupyter.org

Tabela 3 – Bibliotecas utilizadas.

Biblioteca	Descrição	Referência
Pandas[3]	Análise e manipulação de dados	Team (2020)
Scikit-learn[4]	Módulo Python com ampla variedade de algoritmos de aprendizado de máquina	Pedregosa *et al.* (2011)
Imbalanced-learn[5]	Ferramenta Python para lidar com o problema de conjuntos de dados desbalanceados	Lemaître *et al.* (2017)
Numpy[6]	Pacote para computação científica em Python	Harris *et al.* (2020)
Matplotlib[7]	Ferramenta para visualização estática de dados	Hunter (2007)
Seaborn[8]	Ferramenta de visualização de dados Python baseada em Matplotlib	Waskom (2021)

Fonte: O autor.

5.2 Descrição da Base de Dados

O dataset Google 2011v2 consiste em um registro de eventos de um cluster de computação em nuvem com aproximadamente 12.500 nós de computação realizado no ano de 2011 com duração de 29 dias. O dataset inclui informações sobre as especificações de máquinas e o ciclo de vida das tarefas e trabalhos que foram solicitados pelos usuários (ALAHMAD *et al.*, 2021). O conjunto de dados possui aproximadamente 200 GB e é composto pelas seguintes tabelas:

- EVENTOS DE MÁQUINA: fornece informações sobre a adição, remoção ou atualização de máquinas no cluster.
- ATRIBUTOS DE MÁQUINA: fornece informações como versão do kernel, velocidade de clock e presença de endereço IP externo.
- EVENTOS DE TRABALHO: fornece informações relacionadas ao ciclo de vida dos trabalhos.

[3] https://pandas.pydata.org
[4] https://scikit-learn.org
[5] https://imbalanced-learn.org
[6] https://numpy.org
[7] https://matplotlib.org
[8] https://seaborn.pydata.org

- EVENTOS DE TAREFA: fornece informações relacionadas ao ciclo de vida das tarefas.
- RESTRIÇÕES DE TAREFA: descreve as restrições relacionadas às máquinas às quais as tarefas podem ser agendadas.
- USO DE TAREFA: fornece métricas de consumo de recursos das tarefas como CPU e memória.

Como este trabalho objetiva a previsão de falhas de software (tarefas e consequentemente trabalhos), a descrição apresentada nesta seção limita-se às métricas da tabela EVENTOS DE TAREFA. Conforme especificado na documentação oficial do dataset (REISS *et al.*, 2011), as tarefas podem apresentar os seguintes estados finais: removida (*evict*), falha (*fail*), finalizada (*finish*), encerrada (*kill*) e perdida (*lost*). Esses estados possuem os seguintes significados:

- Removida: a tarefa foi removida por outra tarefa com maior prioridade.
- Falha: a tarefa foi encerrada por uma condição anormal.
- Finalizada: a tarefa foi finalizada com êxito.
- Encerrada: a tarefa foi finalizada pelo usuário ou uma de suas dependências foi finalizada de forma anormal.
- Perdida: o registro que indica o término da tarefa está ausente.

A Tabela 4 apresenta a distribuição das tarefas em função do estado final. Embora um número expressivo de tarefas tenham falhado, os motivos das falhas são desconhecidos (REISS *et al.*, 2011). Vale salientar que os estados removida, falha, encerrada e perdida são considerados malsucedidos. Conforme apresentado na Seção 5.3, foram consideradas neste estudo somente as tarefas finalizadas e com falhas.

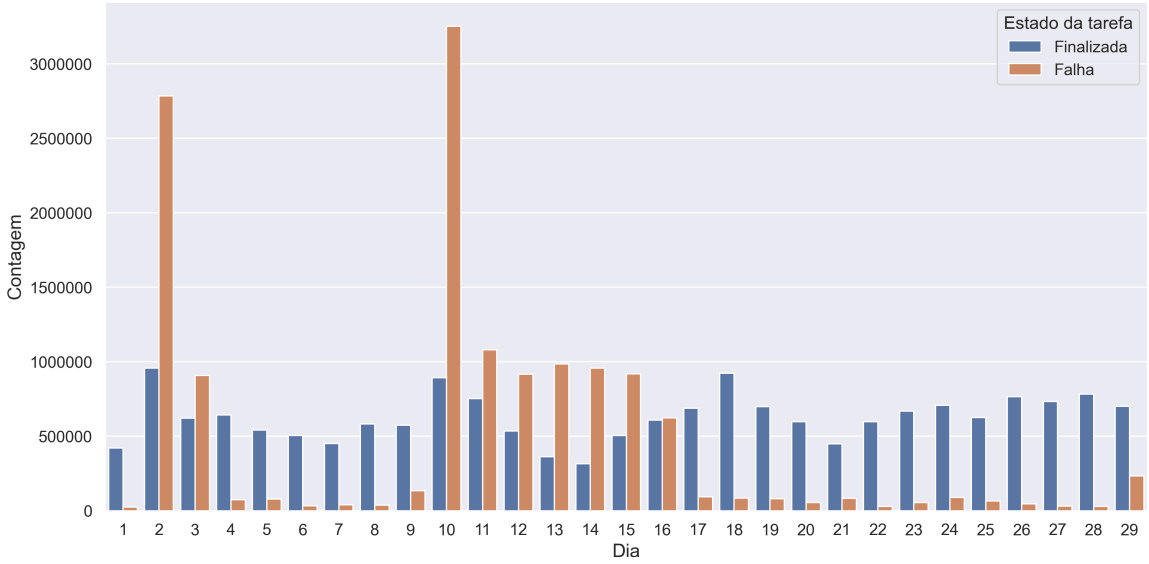
Tabela 4 – Distribuição das tarefas por estado final.

Estado final	**Quantidade**	**Percentual**
Removida	5.864.353	12,15%
Falha	13.829.769	28,65%
Finalizada	18.217.975	37,74%
Encerrada	10.349.680	21,44%
Perdida	8.754	0,02%

Fonte: O autor.

A Figura 16 apresenta a distribuição das tarefas finalizas e com falhas nos 29 dias de monitoramento. É possível observar uma alta incidência de falhas nos dias 2 e 3, bem como entre os dias 10 e 16. O percentual de falhas ocorridas no segundo dia corresponde a 20,12% de todas as falhas observadas, enquanto 23,52% das falhas ocorreram no décimo dia do monitoramento. Assim como a documentação oficial, o dataset Google 2011v2 não apresenta evidências que permitam identificar os possíveis motivos para o número elevado de falhas em determinados dias.

Figura 16 – Distribuição das tarefas finalizadas e com falhas no dataset.

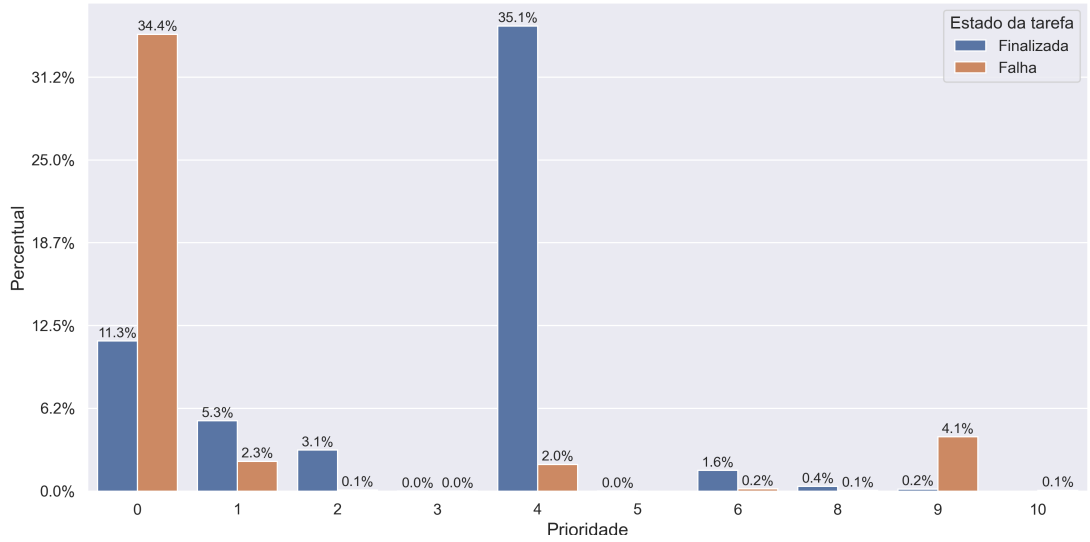


Fonte: O autor.

No dataset Google 2011v2 cada tarefa está associada a uma prioridade de execução. Essa prioridade varia entre 0 e 11, onde os números maiores representam uma maior prioridade (REISS *et al.*, 2012). A Figura 17 apresenta o estado final das tarefas por nível de prioridade. É possível observar que um número elevado de tarefas com prioridade 0 falharam. Por outro lado, destaca-se o quantitativo elevado de tarefas finalizadas com prioridade 4. Esse comportamento pode ser explicado pelo fato das tarefas de maior prioridade possuírem preferência na utilização dos recursos de computação (REISS *et al.*, 2011). Ainda em relação à prioridade, observa-se a predominância de tarefas que falharam em detrimento das finalizadas no nível de prioridade 9. Como tarefas com maior prioridade possuem preferência na utilização de recursos, esse comportamento é considerado anômalo. Como o dataset, assim como a documentação oficial não detalham o comportamento do

agendador de tarefas em relação à prioridade, não é possível identificar os possíveis motivos para essas falhas. 45,75% das tarefas possuem prioridade 0, enquanto 37,07% possuem prioridade 4. A predominância de tarefas de baixa prioridade pode estar relacionada ao percentual elevado de falhas no dataset Google 2011v2. Não foram identificadas tarefas com os níveis de prioridade 7 e 11.

Figura 17 – Estado das tarefas por nível de prioridade.



Fonte: O autor.

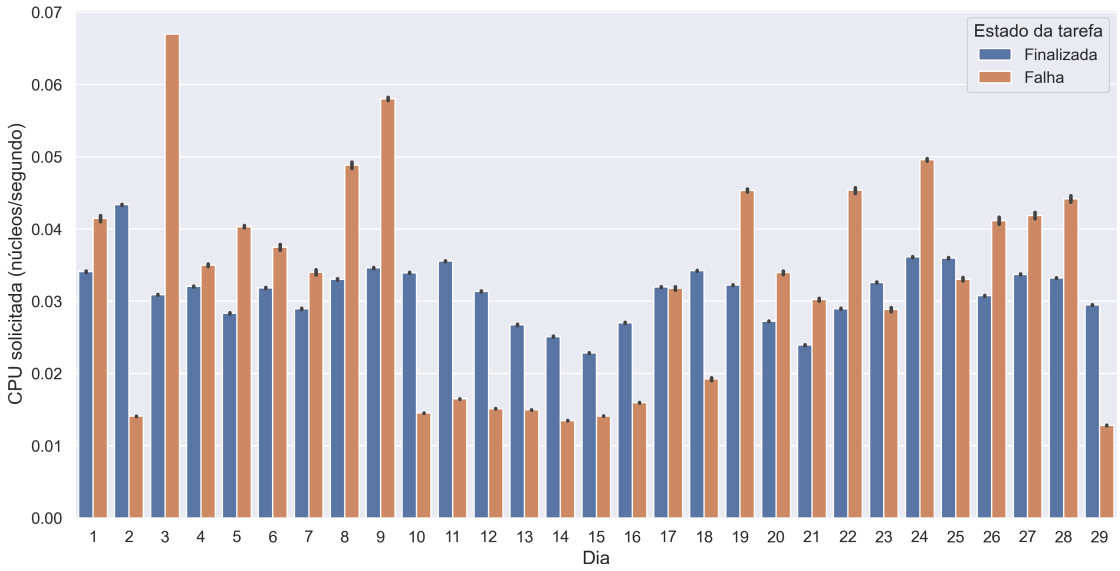
A Figura 18 exibe o estado das tarefas em relação à CPU solicitada. É possível observar que, em alguns dias do conjunto de dados (15 dias), a solicitação média de CPU das tarefas que falharam foi superior à solicitação das tarefas finalizadas. Isso sugere uma possível correlação positiva entre as falhas e os níveis de solicitação de CPU.

O estado final das tarefas em relação à memória RAM solicitada pode ser visualizado na Figura 19. Como pode ser observado, a solicitação média de memória RAM das tarefas que falharam foi superior à solicitação média das tarefas finalizadas em 11 dos 29 dias do conjunto de dados.

Por sua vez, a Figura 20 apresenta o estado das tarefas em relação à solicitação média de disco. Como pode ser observado, em 7 dos 29 dias, a média desse recurso foi maior para tarefas que falharam em comparação com as tarefas finalizadas.

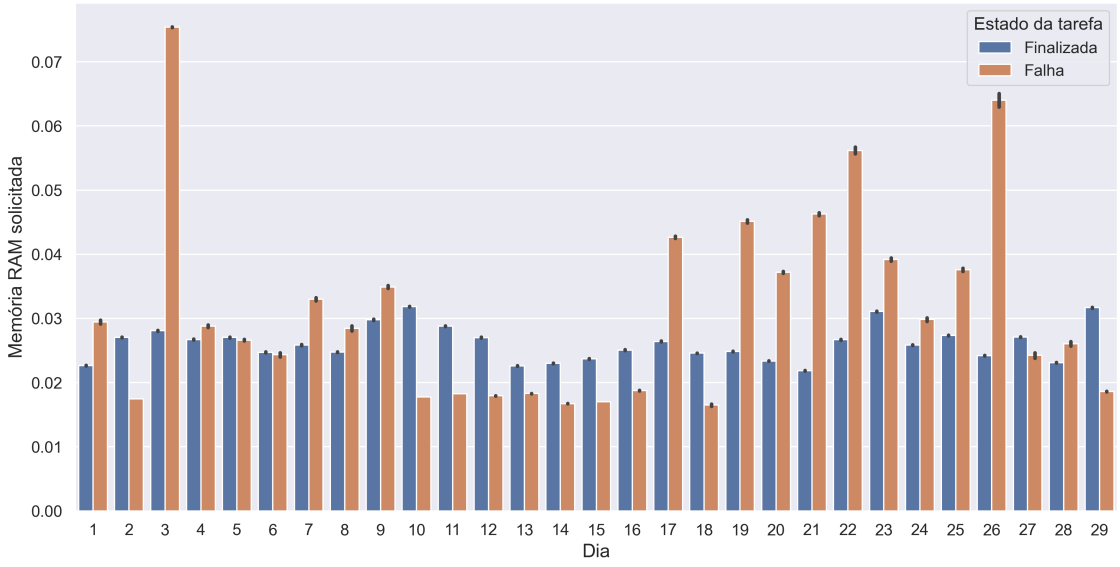
Conforme apresentado nesta seção, ficou evidenciado um maior provisionamento de recursos por tarefas que falharam em alguns dos 29 dias do conjunto de dados. Esses recursos incluem CPU, disco e memória RAM. Como as variáveis observadas se referem

Figura 18 – Estado das tarefas por CPU solicitada.



Fonte: O autor.

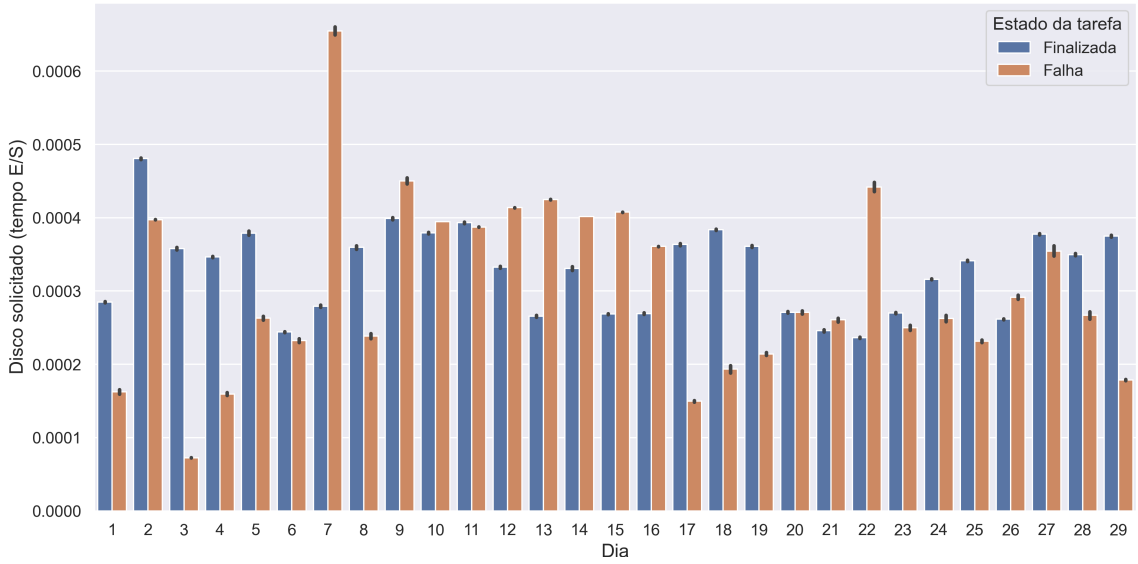
Figura 19 – Estado das tarefas por memória RAM solicitada.



Fonte: O autor.

aos recursos provisionados no início do ciclo de vida das tarefas, a previsão de falhas neste estágio pode ser utilizada para interromper as tarefas que falhariam e como consequência mitigar possíveis desperdícios de recursos.

Figura 20 – Estado das tarefas por disco solicitado.



Fonte: O autor.

5.3 Pré-processamento da Base de Dados

Neste trabalho foram utilizadas duas das tabelas disponíveis no conjunto de dados Google 2011v2, USO DE TAREFA e EVENTOS DE TAREFA. A tabela USO DE TAREFA contém as métricas dinâmicas das tarefas, ou seja, o quanto de recurso foi consumido por cada tarefa durante seu ciclo de vida (média das métricas agregadas a cada 5 minutos). Já a tabela EVENTOS DE TAREFA, analisada na Seção 5.2, contém as métricas estáticas das tarefas, ou seja, o quanto de recurso foi provisionado para cada tarefa. A tabela EVENTOS DE TAREFA também registra as mudanças do status das tarefas, bem como o *timestamp* em que a mudança ocorreu (REISS *et al.*, 2011). Um exemplo da tabela EVENTOS DE TAREFA pode ser visualizado no Anexo A, enquanto o Anexo B exibe um exemplo da tabela USO DE TAREFA.

O pré-processamento dos dados inicia-se com a importação e concatenação dos arquivos da tabela EVENTOS DE TAREFA. Por se tratar de uma grande quantidade de registros, o total de arquivos (500 arquivos csv) excede a quantidade máxima de memória disponível no computador onde foram realizados os experimentos. Para contornar essa limitação, os arquivos foram importados em conjuntos de dez, do primeiro ao décimo e assim sucessivamente. Por se tratar de tarefas que estavam em execução antes do início do rastreamento, todas as amostras com *timestamp* iguais a zero foram removidas, também

foram removidas as linhas duplicadas e todas as linhas que apresentaram pelo menos um registro NaN (*Not a Number*), conforme metodologia utilizada por Khalil *et al.* (2017) e Asmawi *et al.* (2022).

Em relação a remoção indiscriminada dos registros NaN, vale salientar a coluna Informação ausente na tabela EVENTOS DE TAREFA, que indica se registros ausentes em tarefas foram sintetizados. Quando o campo informação ausente apresenta o valor NaN, indica que o registro em questão não possui qualquer valor ausente. Neste trabalho, optou-se por excluir a coluna informação ausente.

Como o objetivo é prever se uma tarefa irá ou não falhar e estamos lidando com um problema de classificação, todas as amostras cujos rótulos (tipo de evento) sejam diferentes de 3 (tarefas que falharam) ou 4 (tarefas finalizadas normalmente) foram excluídas. Embora alguns autores tenham considerado outros estados das tarefas como falhas (removidas, encerradas ou perdidadas) (ISLAM; MANIVANNAN, 2017)(LIU *et al.*, 2017)(LIU *et al.*, 2020)(ASMAWI *et al.*, 2022), e uma vez que a documentação oficial do conjunto de dados não apresenta os motivos claros para esses estados de tarefa (REISS *et al.*, 2011), optou-se por manter somente os dois estados, falha e finalizada, conforme os trabalhos de Gao *et al.* (2020) e Jassas e Mahmoud (2020), pois a intenção é prever se as tarefas falharão ou serão finalizadas corretamente.

Após o pré-processamento e limpeza dos dados da tabela EVENTOS DE TAREFA, o próximo passo consiste no pré-processamento da tabela USO DE TAREFA. O processamento da tabela USO DE TAREFA foi realizado de forma semelhante ao da tabela EVENTOS DE TAREFA. Para superar as limitações de memória, os arquivos foram importados e concatenados em conjuntos de dez, do primeiro ao décimo e assim sucessivamente. Após a importação, foram removidas as linhas duplicadas e os registros com valores NaN. Como as amostras no conjunto de dados estão originalmente agrupadas em intervalos de cinco minutos e apresentam o *timestamp* inicial e final para cada tarefa, foi adicionada uma coluna (duração da tarefa) ao conjunto de dados com o resultado da diferença entre o *timestamp* final e inicial com o objetivo de calcular a duração total de cada tarefa. Após o cálculo da duração de cada tarefa, foram removidas as seguintes colunas do conjunto de dados, pois tais colunas não apresentavam utilidade para as etapas seguintes:

- Tempo inicial (*timestamp* inicial da amostra da tarefa);
- Tempo final (*timestamp* final da amostra da tarefa);
- ID de máquina (identificador único da máquina);
- Proporção da amostra (indica se a amostragem demorou mais tempo do que deveria);
- Tipo de agregação (indica se o registro foi obtido a partir de subcontainers).

Os *timestamps* foram removidos porque a duração das tarefas foi calculada anteriormente (campo duração de tarefa), logo esses campos não apresentam utilidade para o desenvolvimento deste trabalho. O identificador de máquina não possui relevância, pois pretende-se prever falhas de tarefas, assim os atributos de máquina fogem ao escopo deste trabalho. Os campos proporção da amostra e tipo de agregação fornecem detalhes intrínsecos ao cluster do qual os dados foram coletados. Dessa forma, não estão no escopo deste trabalho.

A próxima etapa consiste em agregar as métricas dinâmicas das tarefas (registros das tarefas dispostos em intervalos de cinco minutos) da tabela USO DE TAREFA por tarefa. Nessa ocasião, os campos ID de trabalho e índice de tarefa foram utilizados como identificador único, pois consistem no identificador único de cada trabalho e índice que identifica as tarefas que compõem os respectivos trabalhos. Todos os campos foram agregados pela média aritmética, exceto o campo duração da tarefa que foi agregado pela soma, pois a intenção é identificar a duração total de cada tarefa. No processo de agregação, as duas últimas amostras referentes a cada tarefa foram excluídas. Uma vez que as amostras estão dispostas em intervalos de cinco minutos, a exclusão das duas últimas amostras implica em excluir pelo menos os últimos cinco minutos de dados referentes à tarefa, pois a intenção é prever o estado final da tarefa com no mínimo cinco minutos de antecedência.

Finalizado o processo de agregação das métricas por tarefa, os registros estáticos e dinâmicos das tarefas ainda se encontram em tabelas distintas. Assim, foi necessária a utilização de um método para unificar as tabelas EVENTOS DE TAREFA e USO DE TAREFA, possibilitando a análise dos dados, bem como sua utilização no processo de treinamento do algoritmo de Árvore de Decisão (PEDREGOSA *et al.*, 2011). Para essa finalidade, foi utilizado o método merge da biblioteca Pandas (TEAM, 2020), fixando os campos ID de trabalho e índice de tarefa como identificador único. Ou seja, todos os registros em ambas as tabelas foram unificados para suas respectivas tarefas. Após

a unificação das tabelas anteriormente citadas, não há mais a necessidade de manter identificadores únicos nos registros, além de outros campos sem relevância para o processo de treinamento do algoritmo de ML. Dessa forma, os seguintes campos foram excluídos: tempo (*timestamp* de cada registro), ID de trabalho, índice de tarefa, ID de máquina, nome de usuário, classe de agendamento e restrição de máquina.

Após o pré-processamento dos dados, temos como resultado um único dataset com os seguintes campos: prioridade, CPU solicitada, RAM solicitada, disco solicitado, utilização de CPU média, memória canônica, memória atribuída, cache não mapeado, cache total, memória máxima, tempo médio de disco, espaço médio em disco, CPU máxima, tempo máximo de disco, ciclos por instrução, memória acessada por instrução, amostra de CPU, duração da tarefa, e estado da tarefa. Totalizando 18 *features* mais o variável *target* (estado final da tarefa). O código utilizado no pré-processamento dos dados foi disponibilizado no Apêndice A, enquanto um fragmento do conjunto de dados após as etapas descritas anteriormente pode ser visualizado no Anexo C.

Em relação a distribuição das classes, como ocorreram diversas falhas no período de rastreamento realizado pelo Google, optou-se por utilizar *undersampling* (LI *et al.*, 2020) (sub-amostragem da classe majoritária). Além disso, a técnica de sub-amostragem foi utilizada com o intuito de reduzir a quantidade total de dados no dataset, possibilitando a análise e utilização de todo o conjunto de dados sem o uso de recursos de computação de alto desempenho. É importante destacar que somente os dados de treino foram balanceados. Os dados de teste foram utilizados com a proporção original de classes, que é naturalmente desbalanceada. O balanceamento dos dados de teste resultaria em desempenho superestimado do modelo (LONES, 2021).

5.4 Protocolo de Experimento

Conforme descrito na Seção 5.2, ao analisar a base de dados do Google 2011v2, foram constatadas algumas anomalias e características que podem influenciar os resultados experimentais. Por exemplo, a ausência dos tempos de disco a partir do 15º dia de rastreamento e alta incidência de falhas em determinados dias do rastreamento. Dessa forma, para garantir a legitimidade dos resultados, optou-se por conduzir vários experimentos, avaliando o impacto das anomalias e características identificadas. Como a documentação oficial do dataset não apresenta nenhuma explicação ou justificativa

para a ocorrência elevada de falhas em determinados dias do rastreamento, optou-se pela classificação de alguns desses dias como Dias com muitos *outliers* em tarefas que falharam (DMOTF). Por sua vez, a ausência dos tempos de disco é apontada como consequência de uma mudança no sistema de monitoramento do Google (REISS *et al.*, 2011).

A técnica de planejamento de experimentos adotada neste trabalho é definida por Jain (1991) e detalhada na Seção 2.5. Na Tabela 5, são apresentados os fatores e níveis utilizados nos experimentos. Embora seja possível um total de 12 experimentos diferentes com os fatores e níveis utilizados na abordagem do Planejamento fatorial completo, neste trabalho, decidiu-se realizar um experimento inicial com os primeiros 14 dias, incluindo os tempos de disco e mantendo os dias com alta incidência de falhas. Em seguida, foram realizados outros três experimentos com todos os dias do rastreamento, sem os tempos de disco, indisponíveis a partir do 15º dia, e removendo gradativamente os dias em que ocorreram um número elevado de falhas. A exclusão de determinados dias de rastreamento objetiva investigar o impacto da alta incidência de falhas no desempenho do modelo preditivo. Por exemplo, no Experimento 2, os dias com alta incidência de falhas foram mantidos. No entanto, no Experimento 3, o 2º e o 10º dias foram excluídos. Por sua vez, o Experimento 4 foi realizado excluindo-se o 2º e o 3º dias, bem como do 10º ao 16º dias.

Tabela 5 – Fatores e níveis dos experimentos.

Experimentos	**Dados Utilizados**	**Tempos de Disco**	**DMOTF**
1	Primeiros 14 dias	Incluídos	Incluídos
2	Todos os 29 dias	Excluídos	Incluídos
3	Todos os 29 dias	Excluídos	Excluindo 2º e 10º dias
4	Todos os 29 dias	Excluídos	Excluindo 2º e 3º, 10º ao 16º dias

Fonte: O autor.

As proporções dos dados de treino e teste de cada experimento são apresentadas na Tabela 6. Em todos os experimentos, os dados de treino são balanceados, ou seja, têm proporções iguais de ambas as classes. Por exemplo, no primeiro experimento foram utilizados 5.289.766 registros (para o subconjunto de treino), sendo 2.644.883 da classe Falha e 2.644.883 da classe Finalizada. Por sua vez, os dados de teste do primeiro experimento possuem 2.720.497 registros da classe Falha e 1.217.254 registros da classe Finalizada,

totalizando 3.937.751 registros. No primeiro experimento, os dados de treino correspondem aos dez primeiros dias de rastreamento, enquanto os dados de teste representam os quatro dias posteriores. Ou seja, a intenção é utilizar os dez dias iniciais para prever os quatro dias seguintes. Os dados de teste dos Experimentos 2, 3 e 4 correspondem a 4.311.642 registros (467.628 da classe Falha e 3.844.014 da classe Finalizada). Nos Experimentos 2, 3 e 4, os dados de treino correspondem aos vinte e três dias iniciais do conjunto de dados, enquanto os dados de teste correspondem aos últimos seis dias. Ou seja, a intenção é utilizar os vinte e seis dias iniciais para prever os últimos seis dias do dataset.

Tabela 6 – Proporção dos dados de treino e teste nos experimentos.

Experimentos	**Dados de treino**	**Dados de teste**
1	5.289.766 Registros	3.937.751 Registros
2	10.539.958 Registros	4.311.642 Registros
3	7.551.820 Registros	4.311.642 Registros
4	1.525.580 Registros	4.311.642 Registros

Fonte: O autor.

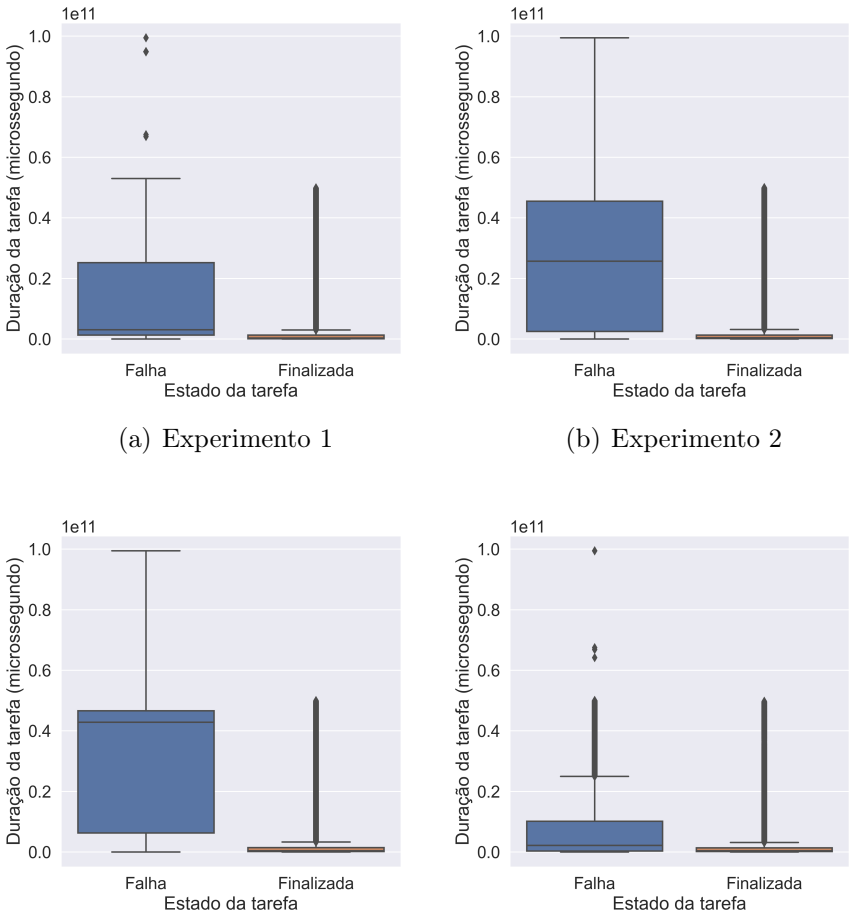
5.5 Análise dos Dados

Além de analisar as variáveis dinâmicas, que representam os recursos efetivamente utilizados em relação ao estado final das tarefas, esta seção também apresenta uma análise de correlação das variáveis disponíveis no conjunto de dados Google 2011v2 e utilizadas na abordagem experimental. As análises apresentadas foram realizadas a partir dos dados de treino dos experimentos descritos na Seção 5.4.

5.5.1 Variáveis Dinâmicas

As variáveis dinâmicas representam o consumo de recursos durante o ciclo de vida das tarefas. Na Figura 21 é possível visualizar a duração das tarefas em relação ao seu estado final, ou seja, falha ou finalizada. Ainda que em proporções diferentes em cada experimento, é possível observar que as tarefas que falharam apresentam maior duração em relação às tarefas que foram finalizadas com sucesso.

Figura 21 – Duração das tarefas por experimento



(a) Experimento 1

(b) Experimento 2

(c) Experimento 3

(d) Experimento 4

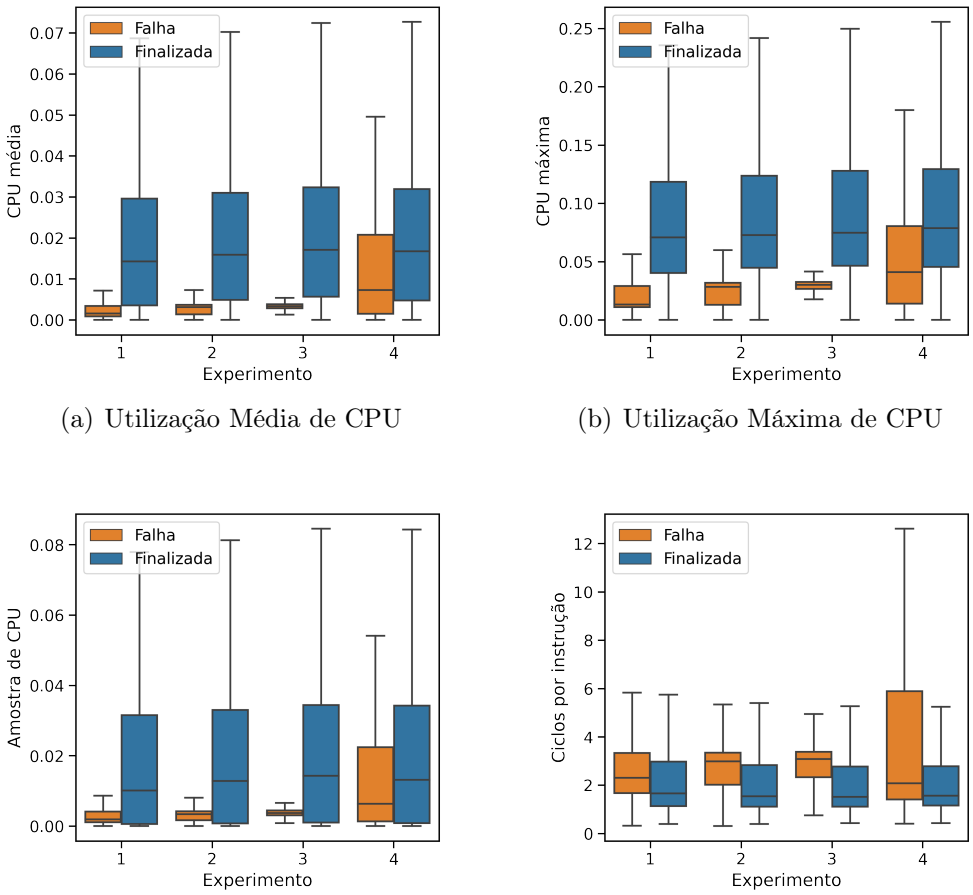
Fonte: O autor.

Inicialmente, pressupõe-se que as tarefas que falharam utilizaram os recursos computacionais por períodos superiores de tempo aos utilizados pelas tarefas que finalizaram, caracterizando desperdício de recursos. Por exemplo, no Experimento 1 as tarefas que falharam consumiram em média 607% mais tempo que as tarefas finalizadas. Esse percentual é ainda maior no Experimento 2, alcançando 1.274%. No Experimento 3, as tarefas que falharam consumiram 1.502% mais tempo que as tarefas finalizadas. Essa diferença foi reduzida para 433% no Experimento 4. A previsão de falhas em um estado inicial pode ser utilizada para mitigar esse desperdício de recursos.

Com o objetivo de avaliar o consumo desses recursos, esta seção apresenta uma análise das variáveis que quantificam os recursos utilizados em relação ao estado final das tarefas. A Figura 22 apresenta as métricas de utilização de CPU em relação ao estado final das tarefas.

Como pode ser observado na Figura 22a, as tarefas finalizadas com êxito

Figura 22 – Métricas de utilização de CPU



(a) Utilização Média de CPU

(b) Utilização Máxima de CPU

(c) Amostra aleatória de CPU

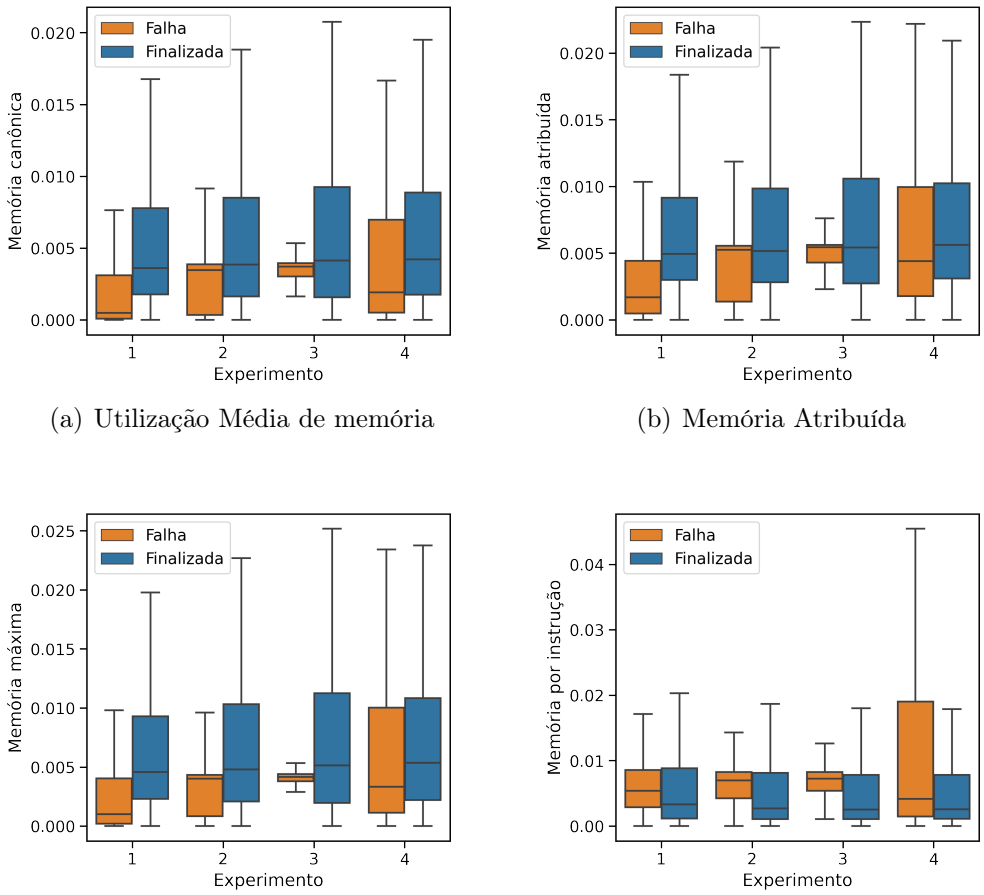
(d) Ciclos por Instrução

Fonte: O autor.

apresentaram maior utilização média de CPU em todos os experimentos. Um comportamento semelhante pode ser observado nas Figuras 22b e 22c, onde as tarefas concluídas com êxito apresentaram maiores estimativas de CPU máxima e amostra de CPU em relação às tarefas que falharam. Vale salientar que a métrica amostra de CPU corresponde à utilização média de CPU a partir de amostras aleatórias com duração de um segundo coletadas em intervalos de cinco minutos (REISS *et al.*, 2011). Por outro lado, conforme apresentado na Figura 22d, as tarefas que falharam apresentaram, predominantemente, maior utilização de ciclos por instrução em todos os experimentos. Com destaque para o Experimento 4, onde mais de 25% das tarefas que falharam utilizaram mais recurso que o total de tarefas finalizadas.

A Figura 23 apresenta as métricas de utilização de memória em relação ao estado final das tarefas.

Figura 23 – Métricas de utilização de memória



(a) Utilização Média de memória

(b) Memória Atribuída

(c) Utilização Máxima de memória

(d) Memória acessada por instrução

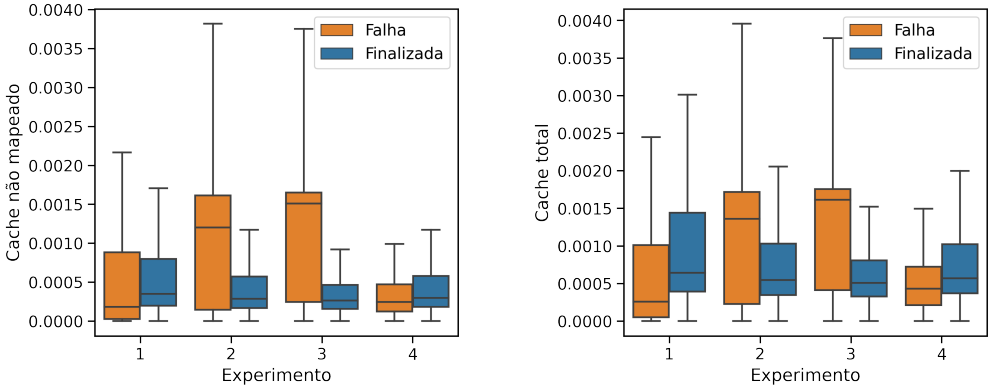
Fonte: O autor.

Como exibido na Figura 23a, em todos os experimentos, as tarefas finalizadas com êxito apresentaram uma utilização média de memória superior em relação às tarefas que falharam. As métricas de memória atribuída, exibidas na Figura 23b, apresentaram comportamento semelhante nos Experimentos 1, 2 e 3, onde as tarefas finalizadas consumiram mais recursos. No entanto, no Experimento 4, as tarefas finalizadas e que falharam apresentaram estimativas semelhantes de utilização desse recurso. Vale ressaltar que a memória canônica corresponde à utilização média de memória, enquanto a memória atribuída se refere ao uso de memória com base na memória atribuída a um determinado contêiner, mas não necessariamente utilizada (REISS *et al.*, 2011). A Figura 23c exibe a utilização máxima de memória em relação ao estado final das tarefas. Observa-se que em todos os experimentos as tarefas finalizadas apresentaram maior utilização desse recurso. Por sua vez, a Figura 23d exibe as métricas de memória acessada por instrução. Em todos os experimentos, a maioria das tarefas que falharam apresentaram maior utilização desse

recurso em comparação com as tarefas finalizadas. Em particular o Experimento 4, onde mais de 25% das tarefas que falharam apresentaram maior utilização desse recurso.

As métricas relacionadas à utilização de cache são apresentadas na Figura 24.

Figura 24 – Métricas de utilização de cache



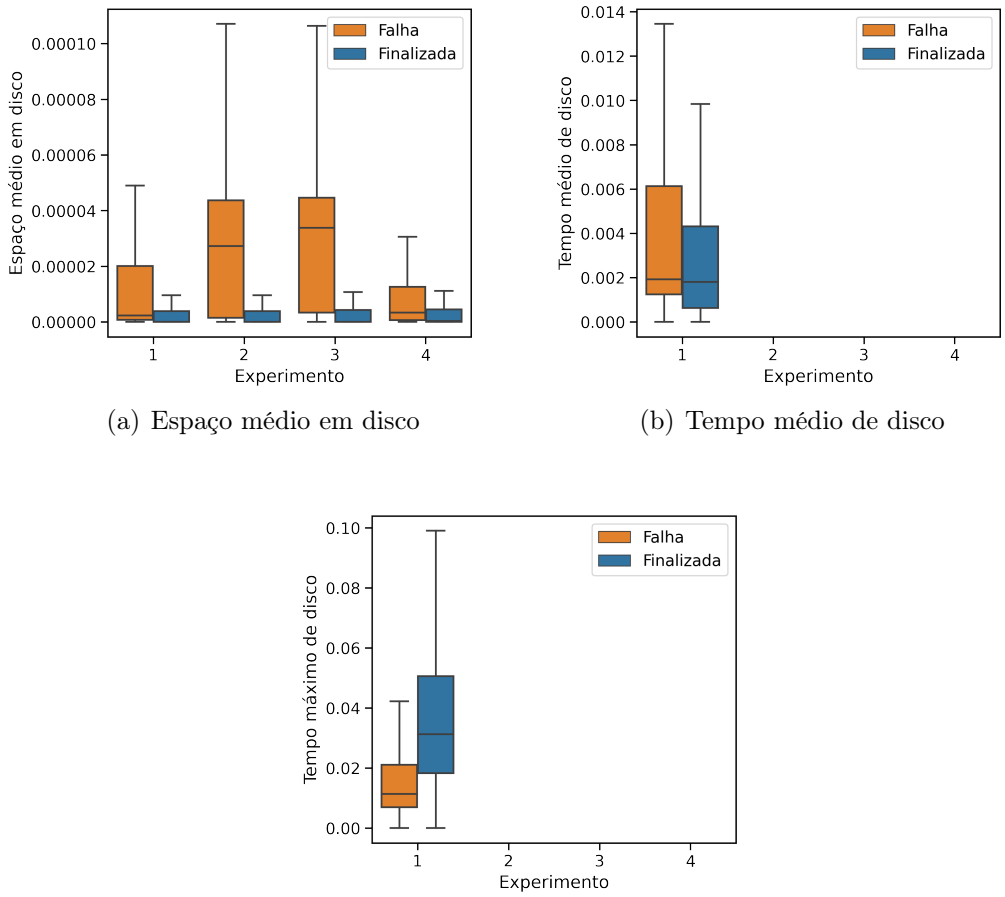
(a) Utilização de cache não mapeado (b) Utilização de cache total

Fonte: O autor.

Conforme exibido na Figura 24a, no Experimento 4, as tarefas finalizadas apresentaram estimativas superiores de utilização de cache não mapeado em relação às tarefas que falharam. Entretanto, nos Experimentos 1, 2 e 3, as tarefas com falha apresentaram maiores estimativas de utilização. Por sua vez, as métricas de cache total, ilustradas na Figura 24b, apresentaram comportamento diverso. Nos Experimentos 1 e 4, as tarefas finalizadas apresentaram maiores estimativas de utilização quando comparadas com as tarefas que falharam. Por outro lado, nos Experimento 2 e 3 as tarefas que falharam apresentaram maior utilização desse recurso.

A Figura 25 apresenta as métricas relacionadas à utilização de disco. Na Figura 25a, é possível observar que, em todos os experimentes, as tarefas que falharam utilizaram maior proporção de espaço em disco em relação às tarefas finalizadas. Conforme apontado na Seção 5.4, os tempos médios e máximos de disco estão disponíveis somente até o 15º dia no dataset. Dessa forma, essas métricas são apresentadas somente no Experimento 1. Na Figura 25b, observa-se que as tarefas com falha apresentaram estimativas de tempo médio de disco superiores em relação às tarefas finalizadas. Entretanto, como apresentado na Figura 25c, as tarefas finalizadas com sucesso apresentaram maior proporção de tempo máximo de disco.

Figura 25 – Métricas de utilização de disco



(a) Espaço médio em disco

(b) Tempo médio de disco

(c) Tempo máximo de disco

Fonte: O autor.

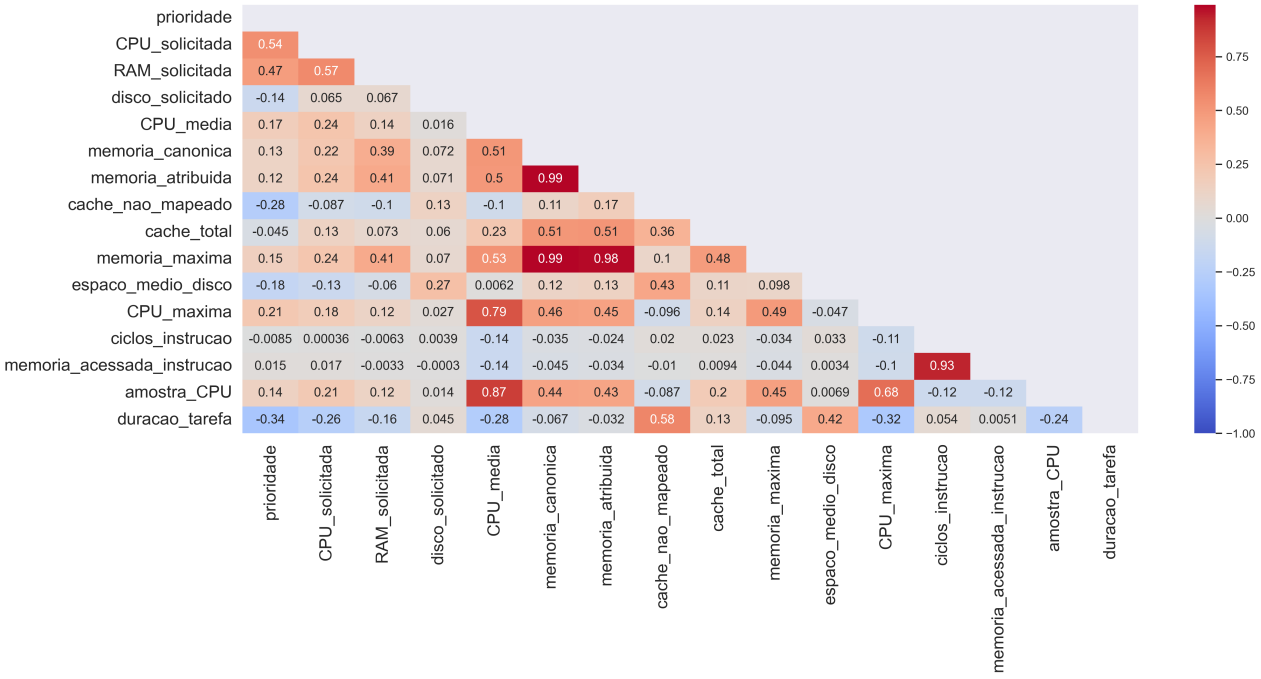
A análise das variáveis dinâmicas realizada nesta seção indica que as tarefas finalizadas utilizaram mais recursos em relação às tarefas que falharam. As tarefas finalizadas utilizaram predominantemente maiores proporções de recursos como CPU média, CPU máxima, amostra de CPU, memória canônica, memória atribuída, memória máxima e tempo máximo de disco. Por outro lado, as tarefas que falharam utilizaram maiores proporções de ciclos por instrução, memória acessada por instrução, cache não mapeado, espaço médio em disco e tempo médio de disco.

5.5.2 Correlação das Variáveis

A Figura 26 apresenta a matriz de correlação das variáveis. Embora tenham sido realizados quatro experimentos com proporções de dados diferentes, não houve mudanças significativas nos valores de correlação. Assim, a matriz de correlação exibida corresponde

aos dados dos primeiros vinte e três dias do dataset (dados de treino do Experimento 2 descrito na Tabela 6), pois incluem o maior número de registros. Neste trabalho, optou-se por eliminar as variáveis que apresentavam uma correlação positiva ou negativa superior a 80% (correlação muito forte) (CHEN *et al.*, 2021). Como resultado, as variáveis memoria_atribuida, memoria_maxima, CPU_media e memoria_acessada_instrucao foram removidas dos dados de treinamento.

Figura 26 – Matriz de correlação.



Fonte: O autor.

5.6 Treinamento do Classificador

Este trabalho utiliza a implementação do algoritmo de árvore de decisão do *Scikit-learn*, descrito na Seção 2.4.2. Neste estudo, os hiperparâmetros foram empiricamente definidos, ou seja, foram ajustados manualmente enquanto o desempenho do modelo foi avaliado. Como os modelos baseados em árvores de decisão fornecem a importância das variáveis (*Feature Importance*) para o resultado das previsões (ASMAWI *et al.*, 2022) e a utilização de muitas variáveis contribui para o aumento da complexidade e possível superajuste do modelo (COSTA, 2022). Os modelos foram treinados em duas etapas. Inicialmente, todas as variáveis foram utilizadas no processo de ajuste dos hiperparâmetros

do modelo com o objetivo de identificar as variáveis com maior importância. Após a primeira etapa, a importância das variáveis foi utilizada para selecionar as variáveis mais preditivas, reduzindo a quantidade de variáveis para posterior treinamento do modelo. É importante salientar que somente os dados de treinamento foram utilizados no ajuste dos hiperparâmetros. A importância das variáveis em modelos baseados em árvore de decisão é detalhada na Seção 2.4.2.

A Tabela 7 apresenta os hiperparâmetros inicialmente utilizados por experimento realizado. É possível observar a profundidade máxima definida para cada árvore de decisão (max_depth), bem como o número de recursos que serão considerados pelo algoritmo para selecionar a melhor divisão dos dados em cada nó (max_features). Os demais hiperparâmetros do classificador foram utilizados com os valores padrão. Por exemplo, os parâmetros criterion (critério utilizado na divisão dos nós), splitter (estratégia utilizada para selecionar a melhor divisão em cada nó), min_samples_split (número mínimo de amostras para divisão de um nó interno) e min_samples_leaf (número mínimo de amostras presentes em um nó folha) apresentaram os valores: 'gini', 'best', 2 e 1, respectivamente.

Tabela 7 – Hiperparâmetros iniciais por experimento.

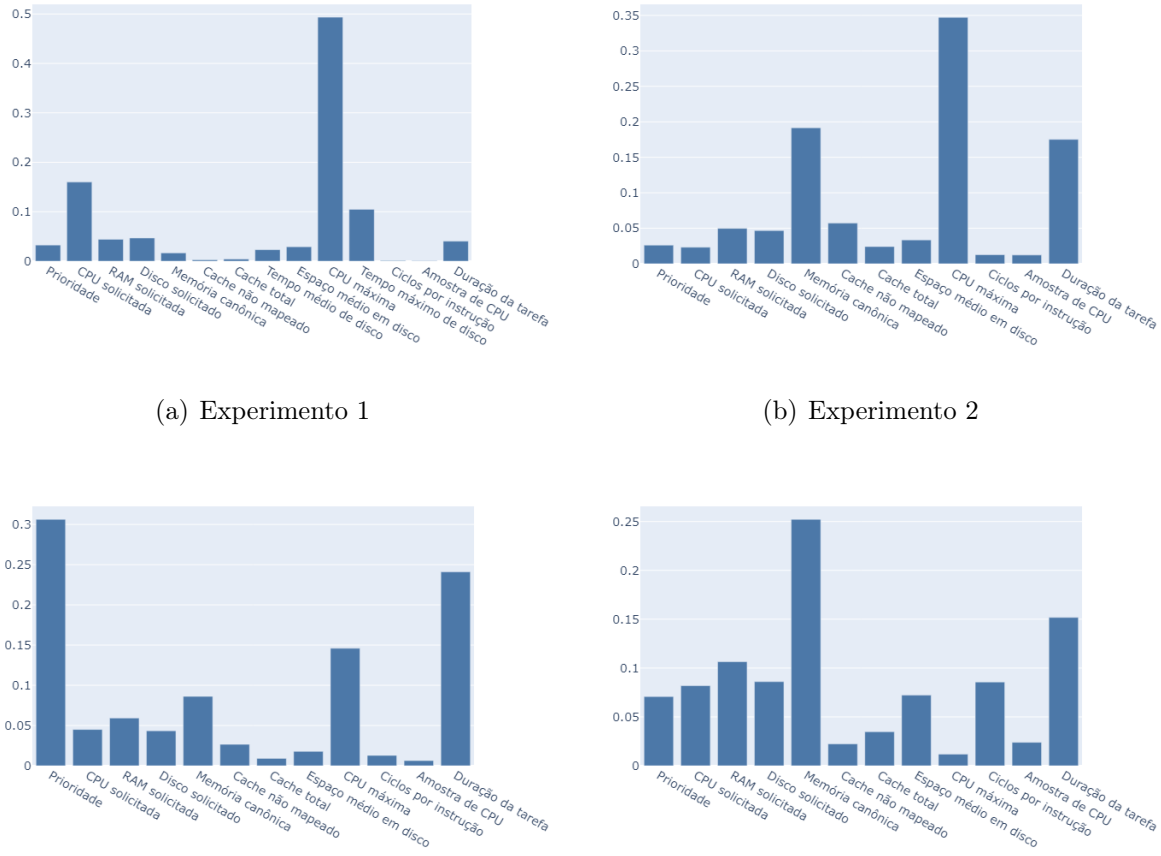
Experimentos	**max_depth**	**max_features**
1	10	None
2	20	'sqrt'
3	15	'log2'
4	10	'sqrt'

Fonte: O autor.

A Figura 27 apresenta as estimativas de importância das variáveis por experimento. No Experimento 1 foram selecionadas as quatro variáveis mais preditivas, CPU solicitada, disco solicitado, CPU máxima e tempo máximo de disco, ilustradas na Figura 27a. Por outro lado, no Experimento 2 foram obtidos resultados mais otimistas com a utilização das três variáveis mais preditivas, memória canônica, CPU máxima e duração da tarefa, exibidas na Figura 27b. Por sua vez, no Experimento 3 foram selecionadas as variáveis prioridade, memória canônica, CPU máxima e duração da tarefa, ilustradas na Figura 27c. Finalmente, no Experimento 4 foram utilizadas as variáveis RAM solicitada, memória canônica, ciclos por instrução e duração da tarefa, detalhadas na Figura 27d. Com exceção

do Experimento 1, em todos os experimentos a duração das tarefas foi uma das variáveis mais relevantes para o resultado das previsões.

Figura 27 – Importância das variáveis por experimento



(a) Experimento 1

(b) Experimento 2

(c) Experimento 3

(d) Experimento 4

Fonte: O autor.

A Tabela 8 apresenta os hiperparâmetros finais utilizados nos experimentos após a redução da quantidade de variáveis. Como pode ser observado, a redução da quantidade de variáveis resultou em modelos com menor complexidade. Por exemplo, a profundidade máxima da árvore de decisão (max_depth) foi reduzida significativamente em todos os experimentos. Essa característica pode contribuir para o aumento do desempenho do modelo, assim como reduzir o tempo de treinamento (BADILLO *et al.*, 2020). É importante salientar que, em problemas de classificação, a profundidade máxima da árvore é o parâmetro com maior influência nos modelos de árvore de decisão (ALAWAD *et al.*, 2018). O código utilizado em um dos experimentos (Experimento 1) foi disponibilizado no Apêndice B. Por sua vez, os modelos resultantes de cada experimento podem ser visualizados no Apêndice C.

Tabela 8 – Hiperparâmetros finais por experimento.

Experimentos	**max_depth**	**max_features**
1	6	‘sqrt’
2	4	‘sqrt’
3	5	‘sqrt’
4	5	‘sqrt’

Fonte: O autor.

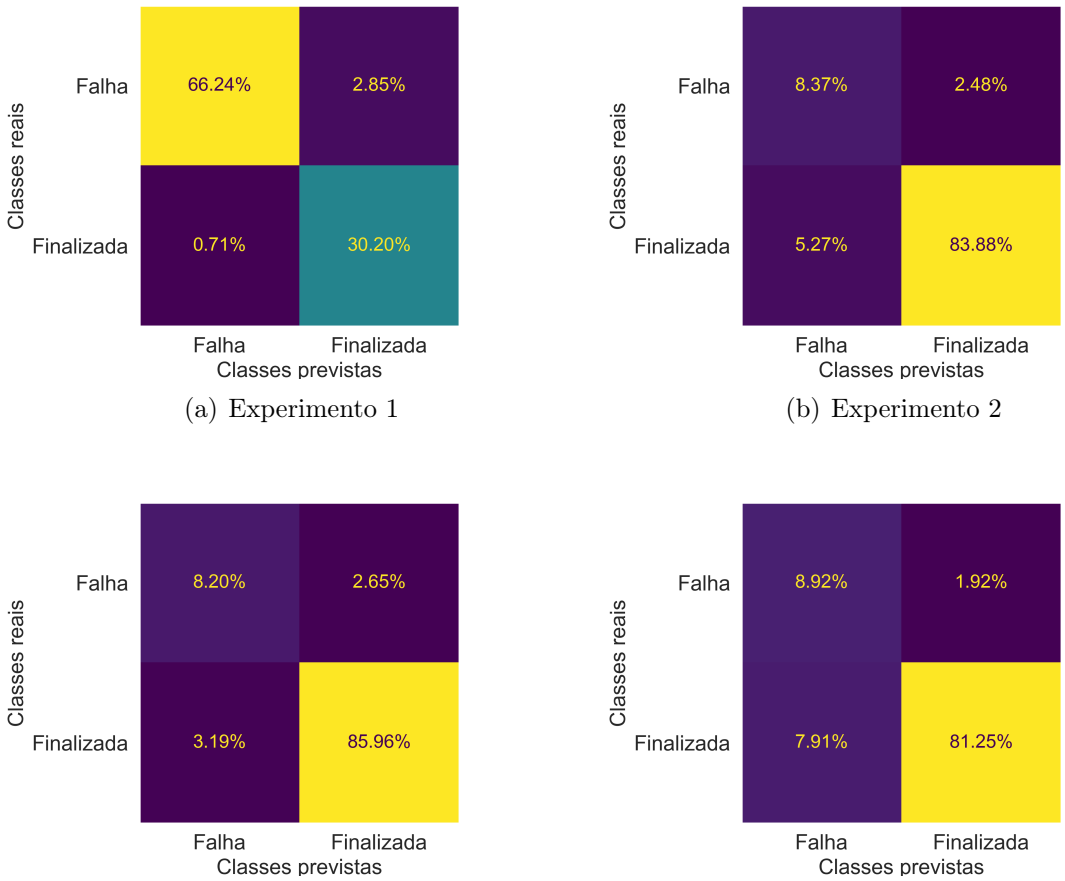
5.7 Resultados

Esta seção apresenta os resultados obtidos na abordagem experimental. Além das métricas comumente utilizadas na avaliação de modelos de classificação, os preditores de falha modelados também são avaliados em relação aos tempos de treinamento e previsão. A fundamentação das métricas de avaliação utilizadas é detalhada na Seção 2.4.3.

Com o objetivo de facilitar a interpretação do desempenho dos modelos, a Figura 28 apresenta as matrizes de confusão dos experimentos. Dessa forma, é possível visualizar os percentuais das previsões corretamente e incorretamente realizadas, em relação às proporções de cada classe nos dados de teste. É importante destacar que as matrizes de confusão foram normalizadas pelo total de instâncias em cada experimento. Dessa forma, a soma de todos os campos da matriz equivale a 100% das observações no experimento. A normalização da matriz também permite visualizar o percentual correspondente de cada classe no experimento.

Como pode ser observado na Figura 28a, o modelo previu corretamente 66,24% das tarefas que falharam. Esse percentual corresponde a 2.608.195 tarefas. Por sua vez, o modelo classificou corretamente 30,20% das tarefas finalizadas, o que corresponde a um total de 1.189.248 tarefas. Como a classe falha constitui o objetivo da previsão (classe positiva), temos o total de verdadeiros positivos e verdadeiros negativos, respectivamente. Ainda em relação ao Experimento 1, o modelo apresentou uma taxa de falso positivo de 0,71%, o que representa 28.006 tarefas. Enquanto a taxa de falso negativo representa 2,85%, o que corresponde a 112.302 tarefas. Neste experimento, as tarefas que falharam equivalem a 69,09% do total de instâncias, enquanto as tarefas finalizadas representam 30,91%.

Figura 28 – Matrizes de confusão por experimento



(a) Experimento 1

(b) Experimento 2

(c) Experimento 3

(d) Experimento 4

Fonte: O autor.

A Figura 28b exibe a matriz de confusão do Experimento 2. Neste experimento, o modelo previu corretamente 8,37% das falhas e classificou corretamente 83,88% das tarefas finalizadas. Esses percentuais correspondem a 360.900 e 3.616.775, respectivamente. Neste experimento, ocorreram maiores taxas de falso positivo em relação ao Experimento 1. Um total de 5,27% das tarefas finalizadas, o que corresponde a 227.239 tarefas, foram classificadas incorretamente como falhas. Enquanto 2,48% das tarefas que falharam, o que corresponde a 106.728 tarefas, foram erroneamente classificadas como finalizadas. Vale salientar que neste experimento, assim como nos Experimentos 3 e 4, as tarefas finalizadas correspondem a 89,15% das instâncias, enquanto 10,85% das instâncias são de tarefas que falharam.

As proporções das previsões do Experimento 3 podem ser visualizadas na Figura 28c. O modelo previu corretamente o estado de 8,20% das tarefas que falharam e 85,96% das

tarefas finalizadas. Essas estimativas correspondem a 353.541 e 3.706.444, respectivamente. Em relação aos falsos positivos e negativos, 3,19% das tarefas finalizadas, o que corresponde a 137.570 tarefas, foram incorretamente classificadas como falhas. Por outro lado, 2,65% das tarefas que falharam, o que corresponde a 114.087 tarefas, foram erroneamente classificadas como finalizadas.

Na Figura 28d, é possível observar o percentual de tarefas que falharam e foram previstas corretamente (8,92%), assim como o percentual de tarefas finalizadas e corretamente classificadas (81,25%) no Experimento 4. Esses percentuais correspondem a 384.662 e 3.503.082, respectivamente. O modelo ainda classificou erroneamente como falhas 7,91% das tarefas finalizadas, o que corresponde a 340.932 tarefas. Um total de 1,92% das tarefas que falharam foram incorretamente classificadas como finalizadas, o que corresponde a 82.966 tarefas.

Inicialmente, é possível concluir que o modelo resultante do Experimento 1 apresentou melhor desempenho em relação aos demais. Entretanto, como a classe majoritária nos dados utilizados no Experimento 1 é a classe falha, não é possível realizar uma comparação direta entre os quatro experimentos. Dessa forma, somente os Experimentos 2, 3 e 4 são diretamente comparáveis. Embora o Experimento 4 tenha previsto corretamente o maior número de falhas (8,92%), esse experimento apresentou o maior número de falsos positivos (7,91%). Por sua vez, O Experimento 2 previu corretamente 8,37% das falhas, mas ainda apresenta um número elevado de falsos positivos em relação à proporção de dados de falhas no experimento. Por outro lado, o Experimento 3 previu corretamente 8,20% das falhas e reduziu os falsos positivos para 3,19%. Levando em consideração que 89,15% dos dados nesses experimentos são da classe finalizada, reduzir a quantidade de falsos positivos resulta no aumento da precisão do modelo.

A Tabela 9 apresenta as métricas do desempenho médio obtidas em cada experimento. É possível observar os tempos de treinamento e previsão, assim como a acurácia, recall, precisão e F1 score alcançados. É importante destacar que os tempos de treinamento e previsão estão representados em segundos. Os Experimentos 1 e 2 apresentaram uma diferença acentuada nos tempos de treinamento quando comparados com os Experimentos 3 e 4. Essa diferença pode ser explicada pelo fato do modelo resultante do Experimento 1 possuir maior complexidade em relação aos demais (profundidade da árvore de decisão) e o Experimento 2 contar com uma maior quantidade de dados de

treinamento. Em todos os experimentos, os tempos de previsão foram menores que um segundo. De forma geral, todos os experimentos apresentaram tempos de treinamento reduzidos. Minimizar os tempos de treinamento é importante para garantir a previsão antecipada das falhas. Como observado, o modelo resultante do Experimento 1 foi o que apresentou os resultados mais promissores. Por exemplo, o Experimento 1 alcançou um recall de 97%, enquanto o Experimento 4 alcançou recall de 87%, seguidos dos Experimentos 2 e 3 com recall de 86%. Por sua vez, a precisão média do Experimento 1 foi de 95%, enquanto o Experimento 3 obteve a segunda melhor precisão (85%). Os Experimentos 2 e 4 obtiveram 79% e 75% de precisão, respectivamente.

Tabela 9 – Desempenho médio por experimento.

Experimento	**Tempo de treinamento (segundos)**	**Tempo de previsão (segundos)**	**Acurácia**	**Recall**	**Precisão**	**F1 Score**
1	11,51	0,30	96%	97%	95%	96%
2	14,01	0,27	92%	86%	79%	82%
3	2,07	0,34	94%	86%	85%	85%
4	3,35	0,36	90%	87%	75%	79%

Fonte: O autor.

Como as classes estão desbalanceadas nos dados de teste, o desempenho médio obtido não permite a visualização detalhada do desempenho do modelo, ou seja, não é possível visualizar as métricas de desempenho em relação a uma determinada classe. A Tabela 10 exibe as métricas obtidas por classe (falha e finalizada), assim como a distribuição das classes nos dados de teste em cada experimento. É possível observar que, com exceção da precisão e do F1 score do Experimento 1, todas as métricas obtidas para a classe finalizada foram superiores as obtidas para classe falha. Esse comportamento é esperado, pois com exceção do Experimento 1, em todos os experimentos a classe predominante é a finalizada. Destaca-se ainda a precisão pouco expressiva (53%) para a classe falha no Experimento 4, contrastando com o recall (82%) do mesmo experimento. Levando em consideração o desbalanceamento excessivo dos dados, alcançar valores promissores para ambas as métricas é uma tarefa desafiadora (LIN *et al.*, 2018).

Nos experimentos, os modelos foram ajustados para maximizar o recall para as previsões positivas (tarefas que falharam) e a precisão para a classe negativa (tarefas

Tabela 10 – Métricas de desempenho por classe.

Experimento	Recall		Precisão		F1 Score		Classes	
	Falha	Finalizada	Falha	**Finalizada**	Falha	Finalizada	Falha	Finalizada
1	**96%**	98%	99%	**91%**	97%	94%	2.720.497	1.217.254
2	**77%**	94%	61%	**97%**	68%	96%	467.628	3.844.014
3	**76%**	96%	72%	**97%**	74%	97%	467.628	3.844.014
4	**82%**	91%	53%	**98%**	64%	94%	467.628	3.844.014

Fonte: O autor.

finalizadas), conforme destacado na Tabela 10. Como o recall avalia o desempenho das previsões corretamente realizadas em relação a todas as observações de uma classe nos dados de teste, maximizar o recall significa prever o maior número possível de falhas no conjunto de dados, ainda que haja uma incidência aceitável de falsos positivos. Por outro lado, como a precisão avalia o desempenho do modelo em relação à proporção das previsões corretamente realizadas entre todas as previsões, maximizar a precisão para a classe falha resulta na redução de falsos positivos.

Os modelos foram ajustados para essa característica, pois no ambiente de computação em nuvem, as falhas podem resultar em prejuízos catastróficos (ABDERRAHIM; CHOUKAIR, 2016), além de ocasionarem danos à reputação do provedor de serviço (CHHETRI *et al.*, 2022). A avaliação dos impactos ocasionados pela classificação de algumas tarefas legítimas como falhas foge ao escopo deste estudo.

Como pode ser observado na Tabela 10, foram obtidos resultados mais promissores no Experimente 1, com recall de 96% para a classe falha e precisão de 91% para a classe finalizada. Nesse experimento foram utilizados os dados referentes aos primeiros catorze dias de monitoramento do dataset Google 2011v2. O maior desempenho alcançado nesse experimento é justificado pela distribuição das falhas no conjunto de dados. 81,77% das falhas ocorreram nos primeiros catorze dias. Em decorrência dessa característica, a classe majoritária nos dados de teste nesse experimento é a classe falha. A predominância dessa classe contribuiu para um maior desempenho do modelo preditivo. É importante lembrar que somente os dados de treinamento foram balanceados em todos os experimentos.

Os Experimentos 2 e 3 obtiveram métricas de recall e precisão semelhantes. O Experimento 2 obteve recall de 77% para a classe falha, enquanto o Experimento 3 obteve recall de 76%, ou seja, entre todas as tarefas que falharam 77% foram previstas corretamente

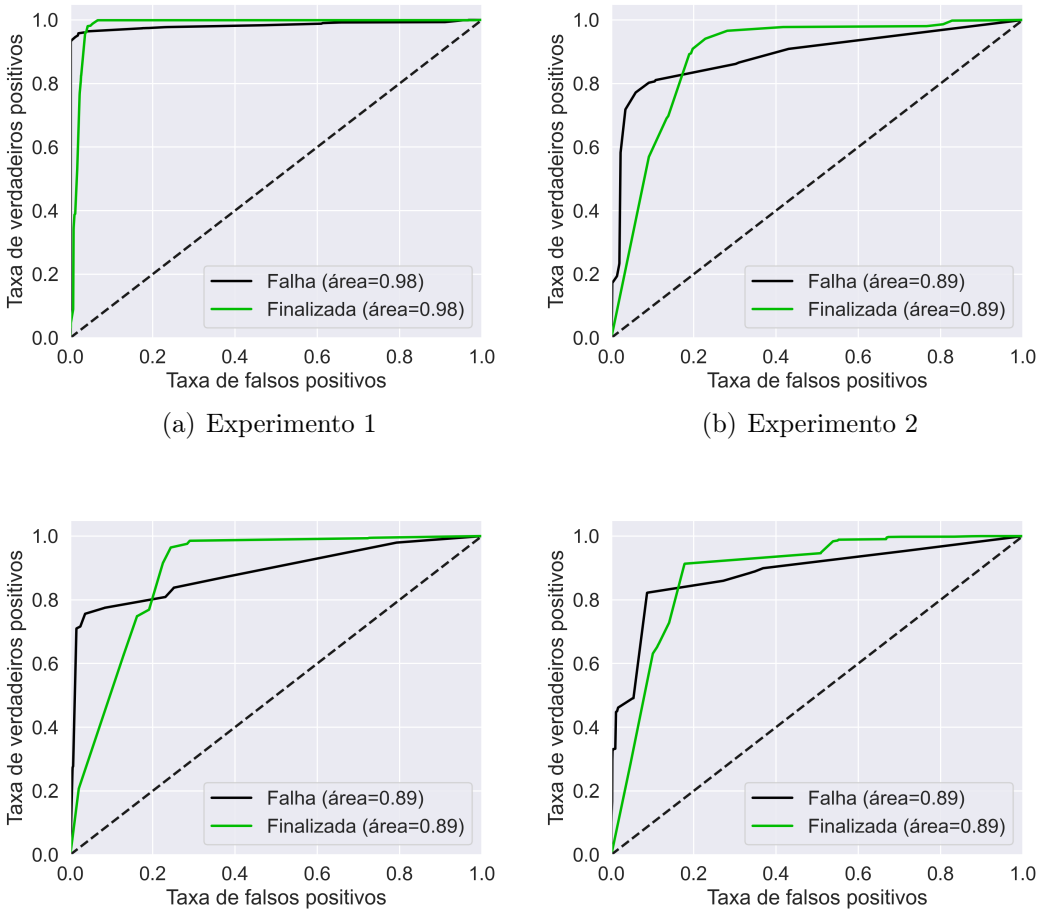
no Experimento 2, enquanto 76% foram previstas no Experimento 3. Entretanto, é importante notar que o recall da classe finalizada no Experimento 2 (94%) é inferior ao recall da classe finalizada no Experimento 3 (96%), isso significa que o Experimento 3 classificou corretamente 96% de todas as tarefas finalizadas, enquanto o Experimento 2 limitou-se a 94%.

Por sua vez, o Experimento 4 obteve recall de 82% para a classe falha e precisão de 98% para a classe finalizada. Essas métricas são superiores às métricas obtidas nos Experimentos 2 e 3. Entretanto, o Experimento 4 obteve recall de 91% para a classe finalizada, o que significa que 91% de todas as tarefas finalizadas foram classificadas corretamente, enquanto 9% dessas tarefas foram classificadas incorretamente como falhas. Em comparação com os Experimentos 2 e 3, o Experimento 4 obteve menor desempenho na classificação de todas as tarefas finalizadas (recall da classe finalizada), o que implica em um maior número de falsos positivos. O menor desempenho do Experimento 4 pode ser explicado pela perda de informação nos dados de teste em decorrência da remoção de uma maior quantidade de dias com alta incidência de falhas.

Como os resultados do Experimento 1 foram superestimados, o modelo com melhor relação entre a previsão do maior número de falhas e a redução dos falsos positivos é o modelo resultante do Experimento 3. Nesse experimento, 76% de todas as falhas foram corretamente previstas, enquanto apenas 4% das tarefas finalizadas foram incorretamente classificadas como falha.

O desempenho dos modelos também foi avaliado em relação à *Receiver Operating Characteristic* (ROC) *area under the curve* (AUC) (ZHANG *et al.*, 2022). A Figura 29 exibe a curva ROC e, consequentemente, as estimativas de AUC obtidas para as duas classes (falha e finalizada) em cada experimento. Conforme descrito na Seção 2.4.3, quanto mais próxima a curva ROC estiver do canto superior esquerdo, melhor é o desempenho do modelo. Como pode ser observado na Figura 29a, as estimativas mais promissoras foram obtidas no Experimento 1, com 98% para ambas as classes. Por sua vez, embora apresentem variações sutis na curva ROC, os Experimentos 2, 3 e 4 obtiveram AUC de 89% para ambas as classes, como pode ser observado nas Figuras 29b, 29c e 29d. De forma geral, foram obtidos resultados promissores em todos os experimentos.

Figura 29 – Curva ROC AUC por experimento



(a) Experimento 1

(b) Experimento 2

(c) Experimento 3

(d) Experimento 4

Fonte: O autor.

5.8 Discussão

Os resultados da abordagem experimental realizada neste estudo indicam resultados superestimados em trabalhos anteriores. A Tabela 11 resume os resultados alcançados em relação aos últimos estudos identificados na revisão da literatura que utilizaram o dataset Google 2011v2. Como este estudo objetiva a previsão de falhas de tarefas e trabalhos, somente os resultados de abordagens com esse objetivo foram comparados. Entre os resultados obtidos neste estudo, foram comparados os resultados do Experimento 1, que foi realizado sem o tratamento da alta incidência de falhas nos dias iniciais do dataset, assim como o Experimento 3, onde foi tratada a alta incidência de falhas ocorridas no segundo e décimo dia do dataset. É importante destacar que as falhas ocorridas nesses dias correspondem a 43,56% de todas as falhas ocorridas nos 29 dias de monitoramento do conjunto de dados.

Tabela 11 – Comparação dos resultados.

Referência	Acurácia	Recall	Precisão	F1 Score	AUC
Experimento 1	96%	97%	95%	96%	98%
Experimento 3	94%	86%	85%	85%	89%
Liu *et al.* (2017)	93%	-	93%	-	-
Shetty *et al.* (2019)	-	95%	92%	-	-
Liu *et al.* (2020)	97%	-	-	98%	-
Islam e Manivannan (2017)	87%	-	-	-	
Bhattacharyya *et al.* (2017b)	100%	-	-	-	-
Gao *et al.* (2020)	93%	-	-	-	90%
Alahmad *et al.* (2021)	94%	-	-	-	-
Jassas e Mahmoud (2020)	-	99%	99%	99%	-
Jassas e Mahmoud (2021)	-	99%	99%	99%	-
Jassas e Mahmoud (2022)	-	95%	98%	97%	100%
Asmawi *et al.* (2022)	94%	92%	94%	93%	-

Fonte: O autor.

Como pode ser observado na Tabela 11, o Experimento 1 obteve acurácia de 96%, recall de 97%, precisão de 95%, F1 score de 96% e AUC de 98%. Por outro lado, os resultados obtidos no Experimento 3, onde a alta incidência de falhas foi tratada, foram mais realistas. O modelo apresentou acurácia de 94%, recall de 86%, precisão de 85%, F1 score de 85% e AUC de 89%. O contraste no desempenho desses experimentos é justificado pela alta concentração de falhas nos dias iniciais do conjunto de dados, como apontado na Seção 5.2. Essas constatações evidenciam resultados superestimados em pesquisas anteriores que utilizaram frações do dataset Google 2011v2 e não consideraram a alta concentração de falhas no conjunto de dados.

Por exemplos, Liu *et al.* (2020) relataram resultados promissores na previsão de falhas de tarefas com a utilização dos três dias iniciais do conjunto de dados. Os autores relataram acurácia e F1 score de 97% e 98%, respectivamente. Embora detalhem o pré-processamento dos dados utilizados, os autores não indicaram ou trataram a alta incidência de falhas.

Asmawi *et al.* (2022) relataram uma proporção anormal de tarefas que falharam em relação às tarefas finalizadas nos primeiros catorze dias do conjunto de dados. Embora indiquem a existência de três tarefas com falha a cada tarefa finalizada, os autores não trataram os dados utilizados na abordagem experimental. Os autores relataram acurácia, recall, precisão e F1 score de 94%, 92%, 94% e 93%, respectivamente. É importante salientar que, como os autores objetivaram a previsão de falhas de tarefas e trabalhos motivadas por falhas de agendamento, os dados referentes ao consumo de recursos não foram utilizados. Por outro lado, na abordagem experimental realizada neste estudo, foram utilizados tanto os dados referentes ao consumo de recursos como os dados relacionados ao agendamento das tarefas.

Embora Jassas e Mahmoud (2020) tenham relatado alta incidência de falhas nos dias iniciais do dataset Google 2011v2, os autores não trataram essa característica nos dados. Os autores relataram desempenho promissor com os primeiros sete dias do dataset, assim como com a totalidade do conjunto de dados. Na abordagem, que objetiva prever falhas de trabalhos na nuvem, os autores alcançaram precisão, recall e F1 score de 99%, como pode ser observado na Tabela 11. Como o pré-processamento dos dados não foi detalhado, não é possível determinar as proporções de dados de treinamento e teste utilizadas em cada experimento, assim como outros fatores que possam ter contribuído para o desempenho elevado do modelo. Em uma abordagem posterior, Jassas e Mahmoud (2022) relataram a remoção de algumas tarefas com falhas do dataset Google 2011v2 por considerá-las atípicas. Entretanto, os autores não informaram a proporção de tarefas removidas, tampouco a distribuição das classes nos experimentos. Os autores relataram métricas idênticas às apresentadas no trabalho anterior, com precisão, recall e F1 score de 99%. Em termos gerais, a descrição do pré-processamento dos dados foi insuficiente nos dois trabalhos anteriormente mencionados. Entretanto, conforme descrito na Seção 5.2, o detalhamento do pré-processamento dos dados realizado neste estudo possibilita sua reprodução.

Em comparação com os trabalhos analisados nesta seção e descritos na Tabela 11, o Experimento 1 deste estudo obteve resultados simulares explorando a assimetria dos dados nos dias iniciais do dataset. Como o dataset Google 2011v2 possui uma taxa de falhas elevada, a modelagem de preditores de falhas com a utilização desse dataset sem o tratamento dessa característica deve ser cuidadosamente avaliada. O dataset do Google

20011v2 possui características únicas em comparação com outros conjuntos de dados do mundo real disponibilizados publicamente (AMVROSIADIS *et al.*, 2018). Dessa forma, o uso desses dados sem a devida atenção pode apresentar resultados distorcidos.

Um outro aspecto importante analisado neste estudo é a relação entre o estado final da tarefa (falha ou finalizada) e o consumo de recursos durante seu ciclo de vida. Embora alguns autores como Jassas e Mahmoud (2018), Shetty *et al.* (2019), Jassas e Mahmoud (2021) e Jassas e Mahmoud (2022) apontem a existência de uma relação positiva entre as falhas de tarefas e o consumo de recursos, ou seja, as tarefas que falham consomem mais recursos em relação às tarefas finalizadas, neste estudo esse comportamento não foi observado. Como pode ser observado na Seção 5.5.1, embora as tarefas que falharam tenham utilizado maiores proporções de alguns recursos pontuais, as tarefas finalizadas utilizaram predominantemente mais recursos. É importante destacar que esses autores chegaram a essa conclusão a partir da análise das tabelas Eventos de trabalho e Eventos de tarefa, que compõem o dataset Google 2011v2. Entretanto, essas tabelas apresentam as métricas relacionas ao provisionamento de recursos no agendador de tarefas, ou seja, o quantitativo de recurso solicitado por cada tarefa no início de seu ciclo de vida. Porém, como descrito na documentação oficial do conjunto de dados, as tarefas podem utilizar uma quantidade de recursos superior a inicialmente solicitada (REISS *et al.*, 2011). Uma abordagem apropriada deve analisar o real quantitativo de recursos utilizados pelas tarefas durante seu ciclo de vida (disponível na tabela Uso de tarefa), conforme realizado na abordagem experimental deste estudo.

Em relação às variáveis utilizadas nos modelos de previsão, foi observado que, em alguns experimentos, variáveis estáticas foram apontadas como relevantes para o resultado das previsões. Por exemplo, a CPU solicitada apresentou 15,99% de importância no Experimento 1, enquanto a Prioridade apresentou 30,63% no Experimento 3. Por sua vez, no Experimento 4, a memória RAM solicitada apresentou importância de 10,64%. Isso sugere que o modelo pode ser aprimorado para prever falhas ainda no estágio de agendamento das tarefas. Entretanto, como o modelo resultante depende de características do agendador de tarefas, a implementação pode não ser aplicável em outros cenários. Vale salientar que as pesquisas relacionadas à previsão antecipada de falhas encontram-se em um estágio inicial (LIU *et al.*, 2020). A próxima seção apresenta os argumentos conclusivos deste capítulo.

5.9 Conclusão

Este capítulo apresentou uma abordagem experimental para previsão de falhas em sistemas em nuvem. Nesse propósito, foi utilizado o conjunto de dados Google 2011v2, assim como a implementação do algoritmo de árvore de decisão do *Scikit-learn*. É importante ressaltar que este estudo objetivou a previsão de falhas de trabalhos e tarefas com antecedência de pelo menos cinco minutos. considerando que os registros de cada tarefa estão dispostos no conjunto de dados em intervalos de cinco minutos, os dois últimos registros de cada tarefa foram excluídos, tanto dos dados de treinamento quanto dos dados de teste. Dessa forma, como o preditor não teve acesso aos últimos dez minutos de dados das tarefas, as previsões são realizadas com até dez minutos de antecedência. Com base na revisão sistemática realizada neste trabalho, nenhum trabalho anterior utilizou este método para quantificar o tempo de antecedência das previsões em relação à ocorrência das falhas. A exclusão de registros e consequente perda de informações das tarefas induz um maior nível de dificuldade nas previsões. Dessa forma, não é possível uma comparação direta entre os resultados deste estudo e os resultados de trabalhos anteriores.

Conforme discutido na Seção 5.8, a análise dos métodos utilizados em estudos anteriores aponta para resultados superestimados em decorrência da predominância de tarefas que falharam nos dias iniciais do conjunto de dados. O contraste observado entre os resultados dos Experimentos 1 e 3 evidencia essa característica. Enquanto no Experimento 1 o modelo obteve resultados superestimados explorando a predominância de tarefas com falhas, no Experimento 3 foram obtidos resultados mais realistas com o tratamento da alta incidência de falhas no segundo e décimo dias do conjunto de dados.

A abordagem apresentada neste estudo difere das demais identificadas na literatura por considerar a alta incidência de falhas identificada nos dias iniciais do conjunto de dados utilizado, utilizar um algoritmo de aprendizado de máquina com baixo custo computacional e alta interpretabilidade, além de apresentar um nível de detalhamento que permite a reprodução dos experimentos realizados, contribuindo para os avanços no estado da arte. As contribuições resultantes deste estudo, assim como as limitações e sugestões para trabalhos futuros são apresentadas no próximo capítulo.

6 Conclusão

Nos últimos anos, os sistemas de computação em nuvem têm sido amplamente adotados para a hospedagem dos mais variados tipos de serviços. A flexibilidade e escalabilidade proporcionadas por esses sistemas têm atraído um número crescente de usuários. Entretanto, a natureza distribuída e a heterogeneidade das cargas de trabalho executadas nesses sistemas contribuem para a ocorrência dos mais variados tipos de falhas.

Nesse cenário, as tecnologias de previsão de falhas podem ser utilizadas para garantir a continuidade dos serviços, mitigando os efeitos das falhas que podem resultar em prejuízos financeiros ou a reputação do provedor de serviço. Tendo em vista a escassez de conjuntos de dados do mundo real identificada neste estudo, a modelagem de preditores de falhas confiáveis é uma tarefa desafiadora.

Este estudo propôs um modelo baseado em aprendizado de máquina para previsão de falhas de software em sistemas em nuvem. A utilização de aprendizado de máquina, especificamente técnicas baseadas em árvore de decisão, apresentou resultados promissores. A abordagem experimental alcançou acurácia de 94%, assim como *recall* de 86%, precisão de 85%, F1 score de 85% e AUC de 89%. Efetivamente, a abordagem proposta neste trabalho possibilitou a previsão de 76% das tarefas que falhariam com antecedência de 5 a 10 minutos com a interrupção de 4% das tarefas que seriam finalizadas com sucesso.

6.1 Contribuições

Entre as contribuições resultantes deste estudo estão uma revisão do estado da arte da previsão de falhas nos sistemas em nuvem, uma análise detalhada do conjunto de dados utilizado na abordagem experimental, assim como uma abordagem de previsão de falhas com baixo custo computacional e alta interpretabilidade. Em oposição a abordagens anteriores, a abordagem proposta neste estudo possui um nível de detalhamento que possibilita sua reprodução e análise por outros pesquisadores, contribuindo para os avanços no estado da arte.

6.2 Limitações

A análise do consumo de recursos das tarefas no dataset foi realizada a partir de dados resultantes de técnicas de amostragem (*undersampling*). Dessa forma, informações importantes podem ter sido negligenciadas nesse processo em função da redução da quantidade de dados consequente da amostragem.

Assim como em estudos anteriores, neste estudo a previsão do tempo para a falha é um problema em aberto. Ou seja, conforme constatado na revisão da literatura, não foram identificados estudos conclusivos relacionados à previsão do tempo para a ocorrência de falhas.

6.3 Trabalhos Futuros

Trabalhos futuros podem explorar os seguintes aspectos:

- Investigar se a predominância de tarefas de baixa prioridade pode estar relacionada ao percentual elevado de falhas no dataset Google 2011v2.
- Avaliar diferentes técnicas de amostragem na modelagem de preditores de falhas para sistemas em nuvem.
- Avaliar métodos de agregação das amostras de dados nos conjuntos de dados utilizados na previsão de falhas em nuvem.
- Aplicar aprendizado de máquina na modelagem de preditores de falhas com a utilização do dataset Google 2019v2.
- Aplicar técnicas de aprendizagem profunda na previsão de falhas em sistemas em nuvem.

Referências

ABDERRAHIM, W.; CHOUKAIR, Z. Paas dependability integration architecture based on cloud brokering. In: **Proceedings of the 31st Annual ACM Symposium on Applied Computing**. New York, NY, USA: Association for Computing Machinery, 2016. (SAC '16), p. 484–487. ISBN 9781450337397. Disponível em: <https://doi.org/10.1145/2851613.2851874>.

ABRO, J. H.; LI, C.; SHAFIQ, M.; VISHNUKUMAR, A.; MEWADA, S.; MALPANI, K.; OSEI-OWUSU, J. Artificial intelligence enabled effective fault prediction techniques in cloud computing environment for improving resource optimization. **Scientific Programming**, 2022.

ADAMU, H.; MOHAMMED, B.; MAINA, A. B.; CULLEN, A.; UGAIL, H.; AWAN, I. An approach to failure prediction in a cloud based environment. In: **2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)**. [S.l.: s.n.], 2017.

ADEGBOYEGA, A. Time-series models for cloud workload prediction: A comparison. In: **2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)**. [S.l.: s.n.], 2017.

ALAHMAD, Y.; DARADKEH, T.; AGARWAL, A. Proactive failure-aware task scheduling framework for cloud computing. **IEEE Access**, 2021.

ALAWAD, W.; ZOHDY, M.; DEBNATH, D. Tuning hyperparameters of decision tree classifiers using computationally efficient schemes. In: **2018 IEEE First International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)**. [S.l.: s.n.], 2018. p. 168–169.

AMVROSIADIS, G.; PARK, J. W.; GANGER, G. R.; GIBSON, G. A.; BASEMAN, E.; DEBARDELEBEN, N. On the diversity of cluster workloads and its impact on research results. In: **2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)**. [S.l.: s.n.], 2018.

ARDEN, F.; SAFITRI, C. Hyperparameter tuning algorithm comparison with machine learning algorithms. In: **2022 6th International Conference on Information Technology, Information Systems and Electrical Engineering (ICITISEE)**. [S.l.: s.n.], 2022. p. 183–188.

ASMAWI, T. N. T.; ISMAIL, A.; SHEN, J. Cloud failure prediction based on traditional machine learning and deep learning. **Journal of Cloud Computing**, 2022.

AVIZIENIS, A.; LAPRIE, J.-C.; RANDELL, B.; LANDWEHR, C. Basic concepts and taxonomy of dependable and secure computing. **IEEE Transactions on Dependable and Secure Computing**, v. 1, n. 1, p. 11–33, 2004.

BADILLO, S.; BANFAI, B.; BIRZELE, F.; DAVYDOV, I. I.; HUTCHINSON, L.; KAM-THONG, T.; SIEBOURG-POLSTER, J.; STEIERT, B.; ZHANG, J. D. An introduction to machine learning. **Clinical Pharmacology & Therapeutics**, v. 107, n. 4, p. 871–885, 2020. Disponível em: <https://ascpt.onlinelibrary.wiley.com/doi/abs/10.1002/cpt.1796>.

BELLO, S. A.; OYEDELE, L. O.; AKINADE, O. O.; BILAL, M.; Davila Delgado, J. M.; AKANBI, L. A.; AJAYI, A. O.; OWOLABI, H. A. Cloud computing in construction industry: Use cases, benefits and challenges. **Automation in Construction**, v. 122, p. 103441, 2021. ISSN 0926-5805. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0926580520310219>.

BERTSIMAS, D.; DUNN, J. Optimal classification trees. **Machine learning**, Springer US, New York, v. 106, n. 7, p. 1039–1082, 2017. ISSN 0885-6125.

BHANAGE, D. A.; PAWAR, A. V.; KOTECHA, K. It infrastructure anomaly detection and failure handling: A systematic literature review focusing on datasets, log preprocessing, machine deep learning approaches and automated tool. **IEEE Access**, 2021.

BHATTACHARYYA, A.; SINGH, H.; JANDEGHI, S. A. J.; AMZA, C. Online detection of anomalous applications on the cloud. In: **Proceedings of the 27th Annual International Conference on Computer Science and Software Engineering**. USA: IBM Corp., 2017. (CASCON '17), p. 161–169.

BHATTACHARYYA, A.; SINGH, H.; JANDEGHI, S. A. J.; AMZA, C. Online detection of anomalous applications on the cloud. In: **Proceedings of the 27th Annual International Conference on Computer Science and Software Engineering**. [S.l.: s.n.], 2017.

BOKHARI, M. U.; MAKKI, Q.; TAMANDANI, Y. K. A survey on cloud computing. In: AGGARWAL, V. B.; BHATNAGAR, V.; MISHRA, D. K. (Ed.). **Big Data Analytics**. Singapore: Springer Singapore, 2018. p. 149–164. ISBN 978-981-10-6620-7.

BUYYA, R.; SRIRAMA, S. N.; CASALE, G.; CALHEIROS, R.; SIMMHAN, Y.; VARGHESE, B.; GELENBE, E.; JAVADI, B.; VAQUERO, L. M.; NETTO, M. A. S.; TOOSI, A. N.; RODRIGUEZ, M. A.; LLORENTE, I. M.; VIMERCATI, S. D. C. D.; SAMARATI, P.; MILOJICIC, D.; VARELA, C.; BAHSOON, R.; ASSUNCAO, M. D. D.; RANA, O.; ZHOU, W.; JIN, H.; GENTZSCH, W.; ZOMAYA, A. Y.; SHEN, H. A manifesto for future generation cloud computing: Research directions for the next decade. **ACM Comput. Surv.**, 2018.

CABAN, D.; WALKOWIAK, T. Stochastic analysis of systems exposed to very unlikely faults. In: **2016 Second International Symposium on Stochastic Models in Reliability Engineering, Life Science and Operations Management (SMRLO)**. [S.l.: s.n.], 2016. p. 310–317.

CHAKRABORTTII, C.; LITZ, H. Improving the accuracy, adaptability, and interpretability of ssd failure prediction models. In: **Proceedings of the 11th ACM Symposium on Cloud Computing**. [S.l.: s.n.], 2020.

CHEN, P.; LI, F.; WU, C. Research on intrusion detection method based on pearson correlation coefficient feature selection algorithm. **Journal of physics. Conference series**, IOP Publishing, Bristol, v. 1757, n. 1, p. 12054, 2021. ISSN 1742-6588.

CHEN, Y.; YANG, X.; LIN, Q.; ZHANG, H.; GAO, F.; XU, Z.; DANG, Y.; ZHANG, D.; DONG, H.; XU, Y.; LI, H.; KANG, Y. Outage prediction and diagnosis for cloud service systems. In: **The World Wide Web Conference**. [S.l.: s.n.], 2019.

CHHETRI, T. R.; DEHURY, C. K.; LIND, A.; SRIRAMA, S. N.; FENSEL, A. A combined system metrics approach to cloud service reliability using artificial intelligence. **Big Data and Cognitive Computing**, 2022.

COSTA, S. de V. Modelos preditivos em ambiente cloud. 2022.

DAS, A.; MUELLER, F.; ROUNTREE, B. Aarohi: Making real-time node failure prediction feasible. In: **2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)**. [S.l.: s.n.], 2020.

DIAS, A. H. T.; CORREIA, L. H. A.; MALHEIROS, N. A systematic literature review on virtual machine consolidation. **ACM Comput. Surv.**, 2021.

DOMINGOS, J. L. B. **Failure Prevision in Cloud Applications**. Tese (Doutorado) — Universidade de Coimbra, 2019.

EMMANUEL, T.; MAUPONG, T.; MPOELENG, D.; SEMONG, T.; MPHAGO, B.; TABONA, O. A survey on missing data in machine learning. **Journal of Big Data**, SpringerOpen, v. 8, n. 1, p. 1–37, 2021.

FAHIM, M.; SILLITTI, A. Anomaly detection, analysis and prediction techniques in iot environment: A systematic literature review. **IEEE Access**, 2019.

FISHER, R. A. The use of multiple measurements in taxonomic problems. **Annals of eugenics**, Wiley Online Library, v. 7, n. 2, p. 179–188, 1936.

FLACH, P. A. Roc analysis. In: _____. **Encyclopedia of Machine Learning and Data Mining**. Boston, MA: Springer US, 2016. p. 1–8. ISBN 978-1-4899-7502-7. Disponível em: <https://doi.org/10.1007/978-1-4899-7502-7_739-1>.

GAO, J.; WANG, H.; SHEN, H. Task failure prediction in cloud data centers using deep learning. **IEEE Transactions on Services Computing**, 2020.

GARCÍA, S.; RAMÍREZ-GALLEGO, S.; LUENGO, J.; BENÍTEZ, J. M.; HERRERA, F. Big data preprocessing: methods and prospects. **Big Data Analytics**, BioMed Central, v. 1, n. 1, p. 1–22, 2016.

GOKHROO, M. K.; GOVIL, M. C.; PILLI, E. S. Detecting and mitigating faults in cloud computing environment. In: **2017 3rd International Conference on Computational Intelligence Communication Technology (CICT)**. [S.l.: s.n.], 2017.

GOLLAPALLI, M.; MAISSA, A. Task failure prediction using machine learning tech-niques in the google cluster trace cloud computing environment. **Mathematical Modelling of En-gineering Problems**, 2022.

GUO, J.; CHANG, Z.; WANG, S.; DING, H.; FENG, Y.; MAO, L.; BAO, Y. Who limits the resource efficiency of my datacenter: An analysis of alibaba datacenter traces. In: **Proceedings of the International Symposium on Quality of Service**. [S.l.: s.n.], 2019.

GUPTA, S.; MUTHIYAN, N.; KUMAR, S.; NIGAM, A.; DINESH, D. A. A supervised deep learning framework for proactive anomaly detection in cloud workloads. In: **2017 14th IEEE India Council International Conference (INDICON)**. [S.l.: s.n.], 2017.

HAMAD, M. A.; ZEKI, A. M. Accuracy vs. cost in decision trees: A survey. In: **2018 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)**. [S.l.: s.n.], 2018. p. 1–4.

HARRIS, C. R.; MILLMAN, K. J.; WALT, S. J. van der; GOMMERS, R.; VIRTANEN, P.; COURNAPEAU, D.; WIESER, E.; TAYLOR, J.; BERG, S.; SMITH, N. J.; KERN, R.; PICUS, M.; HOYER, S.; KERKWIJK, M. H. van; BRETT, M.; HALDANE, A.; RÍO, J. F. del; WIEBE, M.; PETERSON, P.; GÉRARD-MARCHANT, P.; SHEPPARD, K.; REDDY, T.; WECKESSER, W.; ABBASI, H.; GOHLKE, C.; OLIPHANT, T. E. Array programming with NumPy. **Nature**, Springer Science and Business Media LLC, v. 585, n. 7825, p. 357–362, set. 2020. Disponível em: <https://doi.org/10.1038/s41586-020-2649-2>.

HERBST, N.; BAUER, A.; KOUNEV, S.; OIKONOMOU, G.; EYK, E. V.; KOUSIOURIS, G.; EVANGELINOU, A.; KREBS, R.; BRECHT, T.; ABAD, C. L.; IOSUP, A. Quantifying cloud performance and dependability: Taxonomy, metric design, and emerging challenges. **ACM Trans. Model. Perform. Eval. Comput. Syst.**, 2018.

HERMIAS, J. P.; TEKNOMO, K.; MONJE, J. C. N. Short-term stochastic load forecasting using autoregressive integrated moving average models and hidden markov model. In: **2017 International Conference on Information and Communication Technologies (ICICT)**. [S.l.: s.n.], 2017.

HUNTER, J. D. Matplotlib: A 2d graphics environment. **Computing in Science & Engineering**, IEEE COMPUTER SOC, v. 9, n. 3, p. 90–95, 2007.

IRRERA, I. **Fault injection for online failure prediction assessment and improvement**. Tese (Doutorado), 2016.

ISLAM, T.; MANIVANNAN, D. Predicting application failure in cloud: A machine learning approach. In: **2017 IEEE International Conference on Cognitive Computing (ICCC)**. [S.l.: s.n.], 2017.

JAIN, R. **The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling**. [S.l.]: Wiley New York, 1991. v. 1.

JANIESCH, C.; ZSCHECH, P.; HEINRICH, K. Machine learning and deep learning. **Electronic Markets**, 2021.

JASSAS, M.; MAHMOUD, Q. H. Failure analysis and characterization of scheduling jobs in google cluster trace. In: **IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society**. [S.l.: s.n.], 2018. p. 3102–3107.

JASSAS, M. S.; MAHMOUD, Q. H. Evaluation of a failure prediction model for large scale cloud applications. In: **Canadian Conference on Artificial Intelligence**. [S.l.: s.n.], 2020.

JASSAS, M. S.; MAHMOUD, Q. H. A failure prediction model for large scale cloud applications using deep learning. In: **2021 IEEE International Systems Conference (SysCon)**. [S.l.: s.n.], 2021.

JASSAS, M. S.; MAHMOUD, Q. H. Analysis of job failure and prediction model for cloud computing using machine learning. **Sensors**, 2022.

JAUK, D.; YANG, D.; SCHULZ, M. Predicting faults in high performance computing systems: An in-depth survey of the state-of-the-practice. In: **Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis**. [S.l.: s.n.], 2019.

JIAO, S. R.; SONG, J.; LIU, B. A review of decision tree classification algorithms for continuous variables. **Journal of Physics: Conference Series**, IOP Publishing, v. 1651, n. 1, p. 012083, nov 2020. Disponível em: <https://dx.doi.org/10.1088/1742-6596/1651/1/012083>.

KABIR, H. M. D.; KHOSRAVI, A.; MONDAL, S. K.; RAHMAN, M.; NAHAVANDI, S.; BUYYA, R. Uncertainty-aware decisions in cloud computing: Foundations and future directions. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 54, n. 4, may 2021. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3447583>.

KASHANI, M. H.; MAHDIPOUR, E. Load balancing algorithms in fog computing: A systematic review. **IEEE Transactions on Services Computing**, 2022.

KAUR, P.; GOSAIN, A. Comparing the behavior of oversampling and undersampling approach of class imbalance learning by combining class imbalance problem with noise. In: SPRINGER. **ICT Based Innovations: Proceedings of CSI 2015**. [S.l.], 2018. p. 23–30.

KHALIL, M. H.; SHETA, W. M.; ELMAGHRABY, A. S. Categorizing hardware failure in large scale cloud computing environment. In: **2017 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)**. [S.l.: s.n.], 2017.

LEE, B.; MAHTAB, M.; NEO, T.; FAROOQI, I.; KHURSHEED, A. A comprehensive review of design of experiment (doe) for water and wastewater treatment application - key concepts, methodology and contextualized application. **Journal of Water Process Engineering**, v. 47, p. 102673, 2022. ISSN 2214-7144. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2214714422001167>.

LEMAîTRE, G.; NOGUEIRA, F.; ARIDAS, C. K. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. **Journal of Machine Learning Research**, v. 18, n. 17, p. 1–5, 2017. Disponível em: <http://jmlr.org/papers/v18/16-365.html>.

LI, Y.; JIANG, Z. M. J.; LI, H.; HASSAN, A. E.; HE, C.; HUANG, R.; ZENG, Z.; WANG, M.; CHEN, P. Predicting node failures in an ultra-large-scale cloud computing platform: An aiops solution. **ACM Trans. Softw. Eng. Methodol.**, 2020.

LIANG, C.; DENG, L.; ZHU, J.; CAO, Z.; LI, C. Disk failure prediction based on sw-disk feature engineering. In: . [S.l.: s.n.], 2022.

LIN, Q.; HSIEH, K.; DANG, Y.; ZHANG, H.; SUI, K.; XU, Y.; LOU, J.-G.; LI, C.; WU, Y.; YAO, R.; CHINTALAPATI, M.; ZHANG, D. Predicting node failure in cloud service systems. In: **Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering**. [S.l.: s.n.], 2018.

LIU, C.; DAI, L.; LAI, Y.; LAI, G.; MAO, W. Failure prediction of tasks in the cloud at an earlier stage: a solution based on domain information mining. **Computing**, 2020.

LIU, C.; HAN, J.; SHANG, Y.; LIU, C.; CHENG, B.; CHEN, J. Predicting of job failure in compute cloud based on online extreme learning machine: A comparative study. **IEEE Access**, 2017.

LIU, D.; WANG, B.; LI, P.; STONES, R. J.; MARBACH, T. G.; WANG, G.; LIU, X.; LI, Z. Predicting hard drive failures for cloud storage systems. In: **International Conference on Algorithms and Architectures for Parallel Processing**. [S.l.: s.n.], 2020.

LIU, X.; HE, Y.; LIU, H.; ZHANG, J.; LIU, B.; PENG, X.; XU, J.; ZHANG, J.; ZHOU, A.; SUN, P.; ZHU, K.; NISHI, A.; ZHU, D.; ZHANG, K. Smart server crash prediction in cloud service data center. In: **2020 19th IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm)**. [S.l.: s.n.], 2020.

LIU, Y. L. Y.; XIE, D. Implications of imbalanced datasets for empirical roc-auc estimation in binary classification tasks. **Journal of Statistical Computation and Simulation**, Taylor Francis, v. 0, n. 0, p. 1–21, 2023. Disponível em: <https://doi.org/10.1080/00949655.2023.2238235>.

LONES, M. A. How to avoid machine learning pitfalls: a guide for academic researchers. **arXiv preprint arXiv:2108.02497**, 2021.

LU, Y.; YE, T.; ZHENG, J. Decision tree algorithm in machine learning. In: **2022 IEEE International Conference on Advances in Electrical Engineering and Computer Applications (AEECA)**. [S.l.: s.n.], 2022. p. 1014–1017.

LUO, C.; ZHAO, P.; QIAO, B.; WU, Y.; ZHANG, H.; WU, W.; LU, W.; DANG, Y.; RAJMOHAN, S.; LIN, Q.; ZHANG, D. Ntam: Neighborhood-temporal attention model for disk failure prediction in cloud platforms. In: **Proceedings of the Web Conference 2021**. [S.l.: s.n.], 2021.

MA, Y.; WU, S.; GONG, S.; XU, C. Artificial intelligence-based cloud data center fault detection method. In: **2020 IEEE 9th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)**. [S.l.: s.n.], 2020.

MACIEL, P. R. M. **Performance, reliability, and availability evaluation of computational systems, Volume 2: Reliability, availability modeling, measuring, and data analysis**. [S.l.]: CRC Press, 2023.

MAJID, H.; ANUAR, S. A systematic literature review of failure prediction in production environment using machine learning technique. **International Journal of Innovative Computing**, 2022.

MARIANI, L.; MONNI, C.; PEZZé, M.; RIGANELLI, O.; XIN, R. Localizing faults in cloud systems. In: **2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)**. [S.l.: s.n.], 2018.

MCKINNEY, W. *et al.* pandas: a foundational python library for data analysis and statistics. **Python for high performance and scientific computing**, Seattle, v. 14, n. 9, p. 1–9, 2011.

MELL, P.; GRANCE, T. The nist definition of cloud computing, document sp 800-145. **Nat. Inst. Stand. Technol., Gaithersburg, MD, USA**, 2011.

MISHRA, A. Creating a decision tree classifier. In: **Machine Learning for iOS Developers**. Hoboken, NJ, USA: John Wiley Sons, Inc, 2020. p. 175–202. ISBN 1119602874.

MITREVSKI, P.; MITREVSKI, F.; GUSEV, M. A decade time-lapse of cloud performance and dependability modeling: Performability evaluation framework. In: **Proceedings of the 2nd International Conference on Networking, Information Systems Security**. New York, NY, USA: Association for Computing Machinery, 2019. (NISS19). ISBN 9781450366458. Disponível em: <https://doi.org/10.1145/3320326.3320400>.

MITROFANOV, S.; SEMENKIN, E. An approach to training decision trees with the relearning of nodes. In: **2021 International Conference on Information Technologies (InfoTech)**. [S.l.: s.n.], 2021. p. 1–5.

MOGHADDAM, F. F.; ROHANI, M. B.; AHMADI, M.; KHODADADI, T.; MADADIPOUYA, K. Cloud computing: Vision, architecture and characteristics. In: **2015 IEEE 6th Control and System Graduate Research Colloquium (ICSGRC)**. [S.l.: s.n.], 2015. p. 1–6.

MOHAMMED, B.; AWAN, I.; UGAIL, H.; YOUNAS, M. Failure prediction using machine learning in a virtualised hpc system and application. **Cluster Computing**, 2019.

MOHAMMED, R.; RAWASHDEH, J.; ABDULLAH, M. Machine learning with oversampling and undersampling techniques: Overview study and experimental results. In: **2020 11th International Conference on Information and Communication Systems (ICICS)**. [S.l.: s.n.], 2020.

MONNI, C.; PEZZè, M.; PRISCO, G. An rbm anomaly detector for the cloud. In: **2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)**. [S.l.: s.n.], 2019.

MONTGOMERY, D. C. **Design and analysis of experiments**. [S.l.]: John wiley & sons, 2017.

MORAVCIK, M.; SEGEC, P.; KONTSEK, M. Overview of cloud computing standards. In: IEEE. **2018 16th International Conference on Emerging eLearning Technologies and Applications (ICETA)**. [S.l.], 2018. p. 395–402.

MRABET, M. A. E.; MAKKAOUI, K. E.; FAIZE, A. Supervised machine learning: A survey. In: **2021 4th International Conference on Advanced Communication Technologies and Networking (CommNet)**. [S.l.: s.n.], 2021. p. 1–10.

NAM, S.; HONG, J.; YOO, J.-H.; HONG, J. W.-K. Virtual machine failure prediction using log analysis. In: **2021 22nd Asia-Pacific Network Operations and Management Symposium (APNOMS)**. [S.l.: s.n.], 2021.

NASTESKI, V. An overview of the supervised machine learning methods. **HORIZONS.B**, v. 4, p. 51–62, 12 2017.

NOTARO, P.; CARDOSO, J.; GERNDT, M. A survey of aiops methods for failure management. **ACM Trans. Intell. Syst. Technol.**, 2021.

OBULESU, O.; MAHENDRA, M.; THRILOKREDDY, M. Machine learning techniques and tools: A survey. In: **2018 International Conference on Inventive Research in Computing Applications (ICIRCA)**. [S.l.: s.n.], 2018. p. 605–611.

PADMAKUMARI, P.; UMAMAKESWARI, A. Task failure prediction using combine bagging ensemble (cbe) classification in cloud workflow. **Wireless Personal Communications**, 2019.

PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. Scikit-learn: Machine learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011.

PRATHIBHA, S. Investigating the performance of machine learning algorithms for improving fault tolerance for large scale workflow applications in cloud computing. In: **2019 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE)**. [S.l.: s.n.], 2019.

RAWAT, A.; BHADORIA, R. S. Accuracy estimation for fault classification in virtual machine using deep learning. In: **2021 2nd International Conference on Secure Cyber Computing and Communications (ICSCCC)**. [S.l.: s.n.], 2021.

RAWAT, A.; SUSHIL, R.; AGARWAL, A.; SIKANDER, A. A new approach for vm failure prediction using stochastic model in cloud. **IETE Journal of Research**, 2021.

REISS, C.; TUMANOV, A.; GANGER, G. R.; KATZ, R. H.; KOZUCH, M. A. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In: **Proceedings of the third ACM symposium on cloud computing**. [S.l.: s.n.], 2012. p. 1–13.

REISS, C.; WILKES, J.; HELLERSTEIN, J. L. **Google cluster-usage traces: format + schema**. Mountain View, CA, USA, 2011. Revised 2014-11-17 for version 2.1. Posted at <https://github.com/google/cluster-data>.

SANTOS, V. A. d.; MANACERO, A.; LOBATO, R. S.; SPOLON, R.; CAVENAGHI, M. A. A systematic review of fault tolerance solutions for communication errors in open source cloud computing. In: **2020 15th Iberian Conference on Information Systems and Technologies (CISTI)**. [S.l.: s.n.], 2020.

SAXENA, D.; SINGH, A. K. Ofp-tm: an online vm failure prediction and tolerance model towards high availability of cloud computing environments. **The Journal of Supercomputing**, 2022.

SAXENA, D.; SINGH, A. K. Vm failure prediction based intelligent resource management model for cloud environments. In: . [S.l.: s.n.], 2022.

SHAO, Y.; ZHANG, Y. A failure prediction method for spacecraft loads based on time series model. In: **2018 12th International Conference on Reliability, Maintainability, and Safety (ICRMS)**. [S.l.: s.n.], 2018.

SHETTY, J.; SAJJAN, R.; G., S. Task resource usage analysis and failure prediction in cloud. In: **2019 9th International Conference on Cloud Computing, Data Science Engineering (Confluence)**. [S.l.: s.n.], 2019.

SINGH, D.; SINGH, B. Investigating the impact of data normalization on classification performance. **Applied Soft Computing**, 2020.

SOUZA, L.; CAMBOIM, K.; ALENCAR, F. A systematic literature review about integrating dependability attributes, performability and sustainability in the implantation of cooling subsystems in data center. **The Journal of Supercomputing**, Springer, v. 78, n. 14, p. 15820–15856, 2022.

STAPIC, Z.; LÓPEZ, E. G.; CABOT, A. G.; ORTEGA, L. de M.; STRAHONJA, V. Performing systematic literature review in software engineering. In: FACULTY OF ORGANIZATION AND INFORMATICS VARAZDIN. **Central European Conference on Information and Intelligent Systems**. [S.l.], 2012. p. 441.

SU, C.-J.; TSAI, L.-C.; HUANG, S.-F.; LI, Y. Deep learning-based real-time failure detection of storage devices. In: **International Conference on Applied Human Factors and Ergonomics**. [S.l.: s.n.], 2019.

TEAM, T. pandas development. **pandas-dev/pandas: Pandas**. Zenodo, 2020. Disponível em: <https://doi.org/10.5281/zenodo.3509134>.

TEHRANI, A. F.; SAFI-ESFAHANI, F. A threshold sensitive failure prediction method using support vector machine. **Multiagent and Grid Systems**, 2017.

VU, D. D.; VU, X. T.; KIM, Y. Deep learning-based fault prediction in cloud system. In: **2021 International Conference on Information and Communication Technology Convergence (ICTC)**. [S.l.: s.n.], 2021.

WASKOM, M. L. seaborn: statistical data visualization. **Journal of Open Source Software**, The Open Journal, v. 6, n. 60, p. 3021, 2021. Disponível em: <https://doi.org/10.21105/joss.03021>.

WILKES, J. **Yet more Google compute cluster trace data**. 2020. Google research blog.

YANG, H.; KIM, Y. Design and implementation of machine learning-based fault prediction system in cloud infrastructure. **Electronics**, 2022.

YETURU, K. Machine learning algorithms, applications, and practices in data science. In: **Handbook of Statistics**. [S.l.]: Elsevier, 2020. v. 43, p. 81–206.

ZHANG, P.; SHU, S.; ZHOU, M. An online fault detection model and strategies based on svm-grid in clouds. **IEEE/CAA Journal of Automatica Sinica**, v. 5, n. 2, p. 445–456, 2018.

ZHANG, P.; WANG, Y.; MA, X.; XU, Y.; YAO, B.; ZHENG, X.; JIANG, L. Predicting dram-caused node unavailability in hyper-scale clouds. In: . [S.l.: s.n.], 2022.

APÊNDICES

APÊNDICE A – Código utilizado no pré-processamento dos dados

Importação das bibliotecas utilizadas.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import os
import gzip
import shutil
import imblearn
from sklearn.metrics import classification_report
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler
from imblearn.over_sampling import SMOTE
from collections import Counter
from glob import glob
from sklearn.metrics import confusion_matrix,
   ConfusionMatrixDisplay
```

Ler todos os arquivos nas pastas *TaskEvent* e *TaskUsage*.

```
taskevent = glob('TaskEvents/*')
taskusage = glob('TaskUsage/*')
```

Descompactar arquivos na pasta *TaskEvents*.

```
for m in taskevent:
    with gzip.open(m, 'rb') as entrada1:
      with open(m[:34], 'wb') as saida1:
        shutil.copyfileobj(entrada1, saida1)
```

Descompactar arquivos na pasta *TaskUsage*.

```
for n in taskusage:
    with gzip.open(n, 'rb') as entrada2:
      with open(n[:33], 'wb') as saida2:
        shutil.copyfileobj(entrada2, saida2)
```

Definir nomes das colunas para a tabela Eventos de tarefa (*task_ events.*) e recuperar nomes dos arquivos csv no diretório.

```
col_Names=["time", "missing_info", "job_ID", "task_index",
   "machine_ID", "event_type", "user_name", "
   scheduling_class", "priority", "request_CPU_cores", "
   request_RAM", "request_disk", "different-
   machine_constraint"]
files = sorted(glob('TaskEvents/part*.csv'))
```

Definindo variável para o nome do arquivo que será salvo.

```
nome=files[0]
name=(nome[16:30])
name_arq=("Processado-"+name+".csv")
#print ("Processado-"+name+".csv")
print(name_arq)
```

Importa e concatenar cada um dos arquivos csv presentes no diretório.

```
df_task_events=pd.concat((pd.read_csv(file, names=col_Names
   ,sep=",",encoding="ASCII") for file in files),
   ignore_index=True)
```

Excluir a coluna *missing_ info*.

```
df_task_events.drop(['missing_info'], axis=1, inplace=True)
```

Excluir valores *NaN* na tabela Eventos de tarefa.

```
df_task_events.dropna(inplace=True)
```

Filtrar linhas com *timestamp* igual a 0.

```
df_task_events=(df_task_events[df_task_events['time']>0])
```

Filtrar *event_type* somente para as classes 3 (falha) e 4 (finalizado).

```
df_task_events=df_task_events[(df_task_events['event_type']
    == 3) | (df_task_events['event_type'] == 4)]
```

Definir nome das colunas para a tabela Uso de tarefas *task_usage* e recuperar nomes dos arquivos csv no diretório.

```
col_Names1=["start_time", "end_time", "job_ID", "task_index
   ", "machine_ID", "mean_CPU_rate", "canonical_memory", "
   assigned_memory", "unmapped_cache", "total_cache", "
   maximum_memory", "mean_disk_time", "mean_disk_space", "
   maximum_CPU", "maximum_disk_time", "
   cycles_per_instruction", "
   memory_accesses_per_instruction", "sample_portion", "
   aggregation_type", "sampled_CPU_usage"]
files1 = sorted(glob('TaskUsage/part*.csv'))
```

Importa e concatenar cada um dos arquivos csv presentes no diretório.

```
df_usage=pd.concat((pd.read_csv(file1, names=col_Names1,sep
   =",",encoding="ASCII") for file1 in files1),
   ignore_index=True)
```

Excluir linhas duplicadas.

```
df_usage.drop_duplicates(ignore_index=True, inplace=True)
```

Adicionar coluna com cálculo da duração de cada tarefa.

```
df_usage['running_time'] = df_usage['end_time'] - df_usage
    ['start_time']
```

Excluir colunas desnecessárias.

```
df_usage.drop(['start_time', 'end_time', 'machine_ID', '
    sample_portion', 'aggregation_type'], axis=1, inplace=
    True)
```

Excluir valores *NaN* na tabela Uso de tarefa.

```
df_usage.dropna(inplace=True)
```

Agregar todas as características da tabela Uso de tarefa pela média aritmética (a duração da tarefa foi agregada pela soma).

```
df_usage=df_usage.groupby(by=["job_ID", "task_index"], sort
    =False, as_index=False).agg({'running_time':(lambda x: x
    .shift(2).sum()), 'mean_CPU_rate':(lambda x: x.shift(2).
    mean()), 'canonical_memory':(lambda x: x.shift(2).mean()
    ), 'assigned_memory':(lambda x: x.shift(2).mean()), '
    unmapped_cache':(lambda x: x.shift(2).mean()), '
    total_cache':(lambda x: x.shift(2).mean()), '
    maximum_memory':(lambda x: x.shift(2).mean()), '
    mean_disk_space':(lambda x: x.shift(2).mean()), '
    maximum_CPU':(lambda x: x.shift(2).mean()), '
    cycles_per_instruction':(lambda x: x.shift(2).mean()), '
    memory_accesses_per_instruction':(lambda x: x.shift(2).
    mean()), 'sampled_CPU_usage':(lambda x: x.shift(2).mean
    ())})
```

Fazer *merge* das tabelas Eventos de tarefa e Uso de tarefa.

```
df = pd.merge(df_task_events, df_usage, how='inner', on=['
    job_ID', 'task_index'])
```

Excluir colunas desnecessárias após o *merge*.

```
df.drop(['time', 'job_ID', 'task_index', 'machine_ID', '
    user_name', 'scheduling_class', 'different-
    machine_constraint'], axis=1, inplace=True)
```

Separar a variável *target* das demais variáveis.

```
X = df[['priority', 'request_CPU_cores', 'request_RAM', '
    request_disk', 'mean_CPU_rate', 'canonical_memory', '
    assigned_memory', 'unmapped_cache', 'total_cache', '
    maximum_memory', 'mean_disk_space', 'maximum_CPU', '
    cycles_per_instruction', '
    memory_accesses_per_instruction', 'sampled_CPU_usage', '
    running_time']]
y = df['event_type']
```

Realizar a amostragem dos dados.

```
undersample = RandomUnderSampler(sampling_strategy=1)
X, y = undersample.fit_resample(X, y)
```

Concatenar as variáveis X e y após amostragem.

```
df_novo=pd.concat([X, y], axis=1, sort=False)
```

Salvar arquivo em um novo csv após amostragem.

```
df_novo.to_csv('Processados/'+name_arq, index = False)
```

APÊNDICE B – Código utilizado na abordagem experimental

Importação das bibliotecas utilizadas.

```
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sns
import numpy as np
import scikitplot as skplt
import imblearn
from sklearn.metrics import classification_report
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler
from imblearn.over_sampling import SMOTE
from collections import Counter
from glob import glob
import warnings
warnings.filterwarnings('ignore')
from sklearn.tree import DecisionTreeClassifier
import time
from sklearn.metrics import confusion_matrix,
   ConfusionMatrixDisplay
pd.set_option('display.max_columns', None)
```

Atribuir dados de treinamento pré-processados a uma variável.

```
arquivos = sorted(glob('ProcessadosTreino/*.csv'))
```

Importar dados de treinamento.

```
dados=pd.concat((pd.read_csv(arquivo,sep=",",encoding="
   ASCII") for arquivo in arquivos), ignore_index=True)
```

Definir as variáveis X e y nos dados de treinamento.

```
X = dados[['request_CPU_cores', 'request_disk','maximum_CPU
    ', 'maximum_disk_time']]
y = dados['Estado da tarefa']
```

Definir e ajustar os parâmetros do classificador.

```
decision_tree = DecisionTreeClassifier(max_depth=6,
    random_state=0, max_features="sqrt")
```

Treinar o modelo com os dados de treinamento.

```
decision_tree.fit(X, y)
```

Atribuir dados de teste a uma variável.

```
teste = sorted(glob('ProcessadosTeste/*.csv'))
```

Importar dados de teste.

```
df_test=pd.concat((pd.read_csv(test,sep=",",encoding="ASCII
    ") for test in teste), ignore_index=Tr
```

Definir as variáveis X e y nos dados de teste.

```
X_test = df_test[['request_CPU_cores','request_disk' , '
    maximum_CPU', 'maximum_disk_time']]
y_test = df_test['event_type']
```

Realizar as previsões nos dados de teste.

```
y_pred = decision_tree.predict(X_test)
```

Verificar as métricas de desempenho do modelo.

```
print (classification_report(y_test, y_pred))
```

Visualizar a matriz de confusão normalizada.

```
cm = confusion_matrix(y_test, y_pred)
cm_normalized = cm.astype('float') / cm.sum()
cm_display = ConfusionMatrixDisplay(cm_normalized,
   display_labels=['Falha', 'Finalizada'])
cm_display.plot(cmap='viridis', values_format='.2%',
   colorbar=False)
cm_display.ax_.set_xlabel('Classes previstas')
cm_display.ax_.set_ylabel('Classes reais')
```

Visualizar a ROC AUC do modelo.

```
y_probas = decision_tree.predict_proba(X_test)
ax=skplt.metrics.plot_roc_curve(y_test, y_probas, curves='
   each_class', title='')
plt.xlabel('Taxa de falsos positivos')
plt.ylabel('Taxa de verdadeiros positivos')
ax.legend(custom_legend, loc='best', fontsize='medium')
plt.show()
```

APÊNDICE C – Modelos por experimento realizado

Experimento 1

```
|--- maximum_disk_time <= 0.03
|   |--- request_CPU_cores <= 0.01
|   |   |--- request_disk <= 0.00
|   |   |   |--- maximum_CPU <= 0.02
|   |   |   |   |--- maximum_disk_time <= 0.00
|   |   |   |   |   |--- request_disk <= 0.00
|   |   |   |   |   |   |--- class: 3
|   |   |   |   |   |--- request_disk >  0.00
|   |   |   |   |   |   |--- class: 4
|   |   |   |   |--- maximum_disk_time >  0.00
|   |   |   |   |   |--- request_CPU_cores <= 0.01
|   |   |   |   |   |   |--- class: 3
|   |   |   |   |   |--- request_CPU_cores >  0.01
|   |   |   |   |   |   |--- class: 4
|   |   |   |--- maximum_CPU >  0.02
|   |   |   |   |--- maximum_CPU <= 0.03
|   |   |   |   |   |--- request_CPU_cores <= 0.01
|   |   |   |   |   |   |--- class: 4
|   |   |   |   |   |--- request_CPU_cores >  0.01
|   |   |   |   |   |   |--- class: 4
|   |   |   |   |--- maximum_CPU >  0.03
|   |   |   |   |   |--- request_CPU_cores <= 0.01
|   |   |   |   |   |   |--- class: 4
|   |   |   |   |   |--- request_CPU_cores >  0.01
|   |   |   |   |   |   |--- class: 4
|   |   |--- request_disk >  0.00
|   |   |   |--- request_disk <= 0.00
|   |   |   |   |--- request_CPU_cores <= 0.00
|   |   |   |   |   |--- maximum_disk_time <= 0.00
```

```
|   |   |   |   |   |   |--- class: 4
|   |   |   |   |   |--- maximum_disk_time >  0.00
|   |   |   |   |   |   |--- class: 3
|   |   |   |   |--- request_CPU_cores >  0.00
|   |   |   |   |   |--- maximum_disk_time <= 0.01
|   |   |   |   |   |   |--- class: 4
|   |   |   |   |   |--- maximum_disk_time >  0.01
|   |   |   |   |   |   |--- class: 4
|   |   |   |--- request_disk >  0.00
|   |   |   |   |--- request_disk <= 0.00
|   |   |   |   |   |--- maximum_CPU <= 0.07
|   |   |   |   |   |   |--- class: 4
|   |   |   |   |   |--- maximum_CPU >  0.07
|   |   |   |   |   |   |--- class: 4
|   |   |   |   |--- request_disk >  0.00
|   |   |   |   |   |--- request_CPU_cores <= 0.00
|   |   |   |   |   |   |--- class: 4
|   |   |   |   |   |--- request_CPU_cores >  0.00
|   |   |   |   |   |   |--- class: 4
|   |--- request_CPU_cores >  0.01
|   |   |--- maximum_CPU <= 0.04
|   |   |   |--- request_disk <= 0.00
|   |   |   |   |--- request_disk <= 0.00
|   |   |   |   |   |--- maximum_disk_time <= 0.01
|   |   |   |   |   |   |--- class: 3
|   |   |   |   |   |--- maximum_disk_time >  0.01
|   |   |   |   |   |   |--- class: 4
|   |   |   |   |--- request_disk >  0.00
|   |   |   |   |   |--- request_disk <= 0.00
|   |   |   |   |   |   |--- class: 3
|   |   |   |   |   |--- request_disk >  0.00
|   |   |   |   |   |   |--- class: 3
```

```
|   |   |   |--- request_disk >  0.00
|   |   |   |   |--- request_disk <= 0.00
|   |   |   |   |   |--- request_disk <= 0.00
|   |   |   |   |   |   |--- class: 4
|   |   |   |   |   |--- request_disk >  0.00
|   |   |   |   |   |   |--- class: 4
|   |   |   |   |--- request_disk >  0.00
|   |   |   |   |   |--- request_CPU_cores <= 0.06
|   |   |   |   |   |   |--- class: 3
|   |   |   |   |   |--- request_CPU_cores >  0.06
|   |   |   |   |   |   |--- class: 4
|   |   |--- maximum_CPU >  0.04
|   |   |   |--- maximum_CPU <= 0.04
|   |   |   |   |--- request_CPU_cores <= 0.06
|   |   |   |   |   |--- request_disk <= 0.00
|   |   |   |   |   |   |--- class: 4
|   |   |   |   |   |--- request_disk >  0.00
|   |   |   |   |   |   |--- class: 4
|   |   |   |   |--- request_CPU_cores >  0.06
|   |   |   |   |   |--- request_disk <= 0.00
|   |   |   |   |   |   |--- class: 3
|   |   |   |   |   |--- request_disk >  0.00
|   |   |   |   |   |   |--- class: 4
|   |   |   |--- maximum_CPU >  0.04
|   |   |   |   |--- maximum_CPU <= 0.05
|   |   |   |   |   |--- maximum_disk_time <= 0.02
|   |   |   |   |   |   |--- class: 4
|   |   |   |   |   |--- maximum_disk_time >  0.02
|   |   |   |   |   |   |--- class: 4
|   |   |   |   |--- maximum_CPU >  0.05
|   |   |   |   |   |--- maximum_CPU <= 0.14
|   |   |   |   |   |   |--- class: 4
```

```
|   |   |   |   |   |--- maximum_CPU >  0.14
|   |   |   |   |   |   |--- class: 4
|--- maximum_disk_time >  0.03
|   |--- request_CPU_cores <= 0.07
|   |   |--- request_disk <= 0.00
|   |   |   |--- request_CPU_cores <= 0.06
|   |   |   |   |--- maximum_CPU <= 0.10
|   |   |   |   |   |--- request_CPU_cores <= 0.01
|   |   |   |   |   |   |--- class: 4
|   |   |   |   |   |--- request_CPU_cores >  0.01
|   |   |   |   |   |   |--- class: 4
|   |   |   |   |--- maximum_CPU >  0.10
|   |   |   |   |   |--- request_disk <= 0.00
|   |   |   |   |   |   |--- class: 4
|   |   |   |   |   |--- request_disk >  0.00
|   |   |   |   |   |   |--- class: 4
|   |   |   |--- request_CPU_cores >  0.06
|   |   |   |   |--- maximum_disk_time <= 0.04
|   |   |   |   |   |--- maximum_disk_time <= 0.03
|   |   |   |   |   |   |--- class: 4
|   |   |   |   |   |--- maximum_disk_time >  0.03
|   |   |   |   |   |   |--- class: 4
|   |   |   |   |--- maximum_disk_time >  0.04
|   |   |   |   |   |--- request_disk <= 0.00
|   |   |   |   |   |   |--- class: 4
|   |   |   |   |   |--- request_disk >  0.00
|   |   |   |   |   |   |--- class: 4
|   |   |--- request_disk >  0.00
|   |   |   |--- request_disk <= 0.00
|   |   |   |   |--- request_CPU_cores <= 0.05
|   |   |   |   |   |--- request_CPU_cores <= 0.02
|   |   |   |   |   |   |--- class: 3
```

```
|   |   |   |   |   |--- request_CPU_cores >  0.02
|   |   |   |   |   |   |--- class: 3
|   |   |   |   |--- request_CPU_cores >  0.05
|   |   |   |   |   |--- maximum_CPU <= 0.13
|   |   |   |   |   |   |--- class: 4
|   |   |   |   |   |--- maximum_CPU >  0.13
|   |   |   |   |   |   |--- class: 4
|   |   |   |--- request_disk >  0.00
|   |   |   |   |--- maximum_CPU <= 0.07
|   |   |   |   |   |--- maximum_disk_time <= 0.06
|   |   |   |   |   |   |--- class: 3
|   |   |   |   |   |--- maximum_disk_time >  0.06
|   |   |   |   |   |   |--- class: 4
|   |   |   |   |--- maximum_CPU >  0.07
|   |   |   |   |   |--- maximum_CPU <= 0.16
|   |   |   |   |   |   |--- class: 4
|   |   |   |   |   |--- maximum_CPU >  0.16
|   |   |   |   |   |   |--- class: 3
|   |--- request_CPU_cores >  0.07
|   |   |--- maximum_CPU <= 0.18
|   |   |   |--- request_disk <= 0.00
|   |   |   |   |--- maximum_disk_time <= 0.11
|   |   |   |   |   |--- request_CPU_cores <= 0.08
|   |   |   |   |   |   |--- class: 4
|   |   |   |   |   |--- request_CPU_cores >  0.08
|   |   |   |   |   |   |--- class: 4
|   |   |   |   |--- maximum_disk_time >  0.11
|   |   |   |   |   |--- maximum_CPU <= 0.06
|   |   |   |   |   |   |--- class: 3
|   |   |   |   |   |--- maximum_CPU >  0.06
|   |   |   |   |   |   |--- class: 4
|   |   |   |--- request_disk >  0.00
```

```
|   |   |   |   |--- request_CPU_cores <= 0.10
|   |   |   |   |   |--- request_disk <= 0.00
|   |   |   |   |   |   |--- class: 3
|   |   |   |   |   |--- request_disk >  0.00
|   |   |   |   |   |   |--- class: 3
|   |   |   |   |--- request_CPU_cores >  0.10
|   |   |   |   |   |--- maximum_disk_time <= 0.06
|   |   |   |   |   |   |--- class: 3
|   |   |   |   |   |--- maximum_disk_time >  0.06
|   |   |   |   |   |   |--- class: 4
|   |   |--- maximum_CPU >  0.18
|   |   |   |--- request_CPU_cores <= 0.08
|   |   |   |   |--- maximum_CPU <= 0.30
|   |   |   |   |   |--- maximum_disk_time <= 0.03
|   |   |   |   |   |   |--- class: 3
|   |   |   |   |   |--- maximum_disk_time >  0.03
|   |   |   |   |   |   |--- class: 3
|   |   |   |   |--- maximum_CPU >  0.30
|   |   |   |   |   |--- maximum_CPU <= 0.32
|   |   |   |   |   |   |--- class: 4
|   |   |   |   |   |--- maximum_CPU >  0.32
|   |   |   |   |   |   |--- class: 3
|   |   |   |--- request_CPU_cores >  0.08
|   |   |   |   |--- maximum_disk_time <= 0.03
|   |   |   |   |   |--- maximum_disk_time <= 0.03
|   |   |   |   |   |   |--- class: 4
|   |   |   |   |   |--- maximum_disk_time >  0.03
|   |   |   |   |   |   |--- class: 3
|   |   |   |   |--- maximum_disk_time >  0.03
|   |   |   |   |   |--- request_disk <= 0.00
|   |   |   |   |   |   |--- class: 3
|   |   |   |   |   |--- request_disk >  0.00
```

```
|   |   |   |   |   |   |--- class: 4
```

Experimento 2

```
    |--- maximum_CPU <= 0.04
|   |--- running_time <= 970500000.00
|   |   |--- canonical_memory <= 0.00
|   |   |   |--- running_time <= 156500000.00
|   |   |   |   |--- class: 3
|   |   |   |--- running_time >  156500000.00
|   |   |   |   |--- class: 3
|   |   |--- canonical_memory >  0.00
|   |   |   |--- canonical_memory <= 0.00
|   |   |   |   |--- class: 4
|   |   |   |--- canonical_memory >  0.00
|   |   |   |   |--- class: 4
|   |--- running_time >  970500000.00
|   |   |--- running_time <= 18821499904.00
|   |   |   |--- maximum_CPU <= 0.01
|   |   |   |   |--- class: 3
|   |   |   |--- maximum_CPU >  0.01
|   |   |   |   |--- class: 3
|   |   |--- running_time >  18821499904.00
|   |   |   |--- maximum_CPU <= 0.01
|   |   |   |   |--- class: 3
|   |   |   |--- maximum_CPU >  0.01
|   |   |   |   |--- class: 3
|--- maximum_CPU >  0.04
|   |--- canonical_memory <= 0.02
|   |   |--- running_time <= 12074500096.00
|   |   |   |--- running_time <= 1923500032.00
|   |   |   |   |--- class: 4
|   |   |   |--- running_time >  1923500032.00
|   |   |   |   |--- class: 4
```

```
|   |   |--- running_time >  12074500096.00
|   |   |   |--- canonical_memory <= 0.00
|   |   |   |   |--- class: 3
|   |   |   |--- canonical_memory >  0.00
|   |   |   |   |--- class: 3
|   |--- canonical_memory >  0.02
|   |   |--- maximum_CPU <= 0.08
|   |   |   |--- canonical_memory <= 0.06
|   |   |   |   |--- class: 4
|   |   |   |--- canonical_memory >  0.06
|   |   |   |   |--- class: 4
|   |   |--- maximum_CPU >  0.08
|   |   |   |--- canonical_memory <= 0.05
|   |   |   |   |--- class: 4
|   |   |   |--- canonical_memory >  0.05
|   |   |   |   |--- class: 4
```

Experimento 3

```
    |--- priority <= 2.96
|   |--- running_time <= 10524549937.02
|   |   |--- running_time <= 7877811717.06
|   |   |   |--- running_time <= 1873868892.82
|   |   |   |   |--- running_time <= 248667111.37
|   |   |   |   |   |--- class: 4
|   |   |   |   |--- running_time >  248667111.37
|   |   |   |   |   |--- class: 4
|   |   |   |--- running_time >  1873868892.82
|   |   |   |   |--- priority <= 0.99
|   |   |   |   |   |--- class: 4
|   |   |   |   |--- priority >  0.99
|   |   |   |   |   |--- class: 4
|   |   |--- running_time >  7877811717.06
```

```
|   |   |   |--- priority <= 1.80
|   |   |   |   |--- running_time <= 7997619143.09
|   |   |   |   |   |--- class: 4
|   |   |   |   |--- running_time >  7997619143.09
|   |   |   |   |   |--- class: 3
|   |   |   |--- priority >  1.80
|   |   |   |   |--- canonical_memory <= 0.03
|   |   |   |   |   |--- class: 4
|   |   |   |   |--- canonical_memory >  0.03
|   |   |   |   |   |--- class: 4
|   |--- running_time >  10524549937.02
|   |   |--- running_time <= 33353560951.32
|   |   |   |--- canonical_memory <= 0.07
|   |   |   |   |--- maximum_CPU <= 0.06
|   |   |   |   |   |--- class: 3
|   |   |   |   |--- maximum_CPU >  0.06
|   |   |   |   |   |--- class: 3
|   |   |   |--- canonical_memory >  0.07
|   |   |   |   |--- priority <= 0.95
|   |   |   |   |   |--- class: 4
|   |   |   |   |--- priority >  0.95
|   |   |   |   |   |--- class: 3
|   |   |--- running_time >  33353560951.32
|   |   |   |--- running_time <= 38874872991.43
|   |   |   |   |--- running_time <= 38637800001.91
|   |   |   |   |   |--- class: 3
|   |   |   |   |--- running_time >  38637800001.91
|   |   |   |   |   |--- class: 3
|   |   |   |--- running_time >  38874872991.43
|   |   |   |   |--- priority <= 0.74
|   |   |   |   |   |--- class: 3
|   |   |   |   |--- priority >  0.74
```

```
|   |   |   |   |   |--- class: 3
|--- priority >  2.96
|   |--- priority <= 9.59
|   |   |--- running_time <= 91765721783.04
|   |   |   |--- canonical_memory <= 0.01
|   |   |   |   |--- priority <= 7.69
|   |   |   |   |   |--- class: 4
|   |   |   |   |--- priority >  7.69
|   |   |   |   |   |--- class: 3
|   |   |   |--- canonical_memory >  0.01
|   |   |   |   |--- running_time <= 32684370182.18
|   |   |   |   |   |--- class: 4
|   |   |   |   |--- running_time >  32684370182.18
|   |   |   |   |   |--- class: 3
|   |   |--- running_time >  91765721783.04
|   |   |   |--- class: 3
|   |--- priority >  9.59
|   |   |--- class: 3
```

Experimento 4

```
    |--- canonical_memory <= 0.00
|   |--- running_time <= 156500000.00
|   |   |--- canonical_memory <= 0.00
|   |   |   |--- canonical_memory <= 0.00
|   |   |   |   |--- running_time <= 10500000.00
|   |   |   |   |   |--- class: 3
|   |   |   |   |--- running_time >  10500000.00
|   |   |   |   |   |--- class: 3
|   |   |   |--- canonical_memory >  0.00
|   |   |   |   |--- cycles_per_instruction <= 9.79
|   |   |   |   |   |--- class: 3
|   |   |   |   |--- cycles_per_instruction >  9.79
```

```
|   |   |   |   |   |--- class: 4
|   |   |--- canonical_memory >  0.00
|   |   |   |--- request_RAM <= 0.03
|   |   |   |   |--- cycles_per_instruction <= 1.91
|   |   |   |   |   |--- class: 4
|   |   |   |   |--- cycles_per_instruction >  1.91
|   |   |   |   |   |--- class: 4
|   |   |   |--- request_RAM >  0.03
|   |   |   |   |--- request_RAM <= 0.04
|   |   |   |   |   |--- class: 3
|   |   |   |   |--- request_RAM >  0.04
|   |   |   |   |   |--- class: 3
|   |--- running_time >  156500000.00
|   |   |--- request_RAM <= 0.01
|   |   |   |--- canonical_memory <= 0.00
|   |   |   |   |--- canonical_memory <= 0.00
|   |   |   |   |   |--- class: 3
|   |   |   |   |--- canonical_memory >  0.00
|   |   |   |   |   |--- class: 3
|   |   |   |--- canonical_memory >  0.00
|   |   |   |   |--- running_time <= 16160000000.00
|   |   |   |   |   |--- class: 4
|   |   |   |   |--- running_time >  16160000000.00
|   |   |   |   |   |--- class: 3
|   |   |--- request_RAM >  0.01
|   |   |   |--- running_time <= 494500000.00
|   |   |   |   |--- canonical_memory <= 0.00
|   |   |   |   |   |--- class: 3
|   |   |   |   |--- canonical_memory >  0.00
|   |   |   |   |   |--- class: 3
|   |   |   |--- running_time >  494500000.00
|   |   |   |   |--- running_time <= 1925499968.00
```

```
|   |   |   |   |   |--- class: 3
|   |   |   |   |--- running_time >  1925499968.00
|   |   |   |   |   |--- class: 3
|--- canonical_memory >  0.00
|   |--- running_time <= 1920499968.00
|   |   |--- cycles_per_instruction <= 7.19
|   |   |   |--- request_RAM <= 0.04
|   |   |   |   |--- canonical_memory <= 0.00
|   |   |   |   |   |--- class: 4
|   |   |   |   |--- canonical_memory >  0.00
|   |   |   |   |   |--- class: 4
|   |   |   |--- request_RAM >  0.04
|   |   |   |   |--- canonical_memory <= 0.00
|   |   |   |   |   |--- class: 3
|   |   |   |   |--- canonical_memory >  0.00
|   |   |   |   |   |--- class: 4
|   |   |--- cycles_per_instruction >  7.19
|   |   |   |--- running_time <= 363500000.00
|   |   |   |   |--- canonical_memory <= 0.01
|   |   |   |   |   |--- class: 4
|   |   |   |   |--- canonical_memory >  0.01
|   |   |   |   |   |--- class: 3
|   |   |   |--- running_time >  363500000.00
|   |   |   |   |--- running_time <= 559500000.00
|   |   |   |   |   |--- class: 3
|   |   |   |   |--- running_time >  559500000.00
|   |   |   |   |   |--- class: 4
|   |--- running_time >  1920499968.00
|   |   |--- running_time <= 12037499904.00
|   |   |   |--- cycles_per_instruction <= 7.56
|   |   |   |   |--- request_RAM <= 0.06
|   |   |   |   |   |--- class: 4
```

```
|   |   |   |   |--- request_RAM >  0.06
|   |   |   |   |   |--- class: 3
|   |   |   |--- cycles_per_instruction >  7.56
|   |   |   |   |--- request_RAM <= 0.02
|   |   |   |   |   |--- class: 4
|   |   |   |   |--- request_RAM >  0.02
|   |   |   |   |   |--- class: 3
|   |   |--- running_time >  12037499904.00
|   |   |   |--- cycles_per_instruction <= 5.40
|   |   |   |   |--- running_time <= 46558500864.00
|   |   |   |   |   |--- class: 3
|   |   |   |   |--- running_time >  46558500864.00
|   |   |   |   |   |--- class: 3
|   |   |   |--- cycles_per_instruction >  5.40
|   |   |   |   |--- cycles_per_instruction <= 8.22
|   |   |   |   |   |--- class: 3
|   |   |   |   |--- cycles_per_instruction >  8.22
|   |   |   |   |   |--- class: 3
```

ANEXOS

ANEXO A – Exemplo da Tabela Eventos de Tarefa

Tabela 12 – Tabela Eventos de Tarefa.

Tempo	Info ausente	ID trabalho	Índice tarefa	ID máquina	Tipo evento	Usuário[1]	Classe agendamento	Prioridade	CPU solicitada	RAM solicitada	Disco solicitado	Restrição máquina
0	2.0	3418309	0	4.155527e+09	0	*	3	9	NaN	NaN	NaN	NaN
0	2.0	3418309	1	3.291507e+08	0	*	3	9	NaN	NaN	NaN	NaN
0	NaN	3418314	0	3.938719e+09	0	*	3	9	0.125	0.07446	0.000424	0.0
0	NaN	3418314	1	3.516186e+08	0	*	3	9	0.125	0.07446	0.000424	0.0
0	2.0	3418319	0	4.310529e+08	0	*	3	9	NaN	NaN	NaN	NaN

Fonte: O autor.

[1] O identificador de usuário foi omitido para permitir a exibição da tabela.

ANEXO B – Exemplo da Tabela Uso de Tarefa

Tabela 13 – Tabela Uso de Tarefa.

Tempo inicial	Tempo final	ID trabalho	Índice tarefa	ID máquina	CPU média	Memória canônica	Memória atribuída	Cache não mapeado	Cache total	Memória máxima	Tempo médio disco	Espaço médio disco	CPU máxima	Tempo máximo disco	Ciclos instrução	Memória acessada instrução	Proporção amostra	Tipo agregação	Amostra CPU
600000000	900000000	3418309	0	4155527081	0.001562	0.06787	0.07568	0.001156	0.001503	0.06787	0.000003	0.000187	0.039670	0.000357	2.445	0.007243	0	1	0.053410
600000000	900000000	3418309	1	329150663	0.001568	0.06787	0.07556	0.000320	0.000700	0.06787	0.000006	0.000188	0.033020	0.000929	2.100	0.005791	0	1	0.040590
600000000	900000000	3418314	0	3938719206	0.000307	0.08044	0.09521	0.000282	0.000670	0.08044	0.000005	0.000184	0.023770	0.000786	5.588	0.020800	0	1	0.062440
600000000	900000000	3418314	1	351618647	0.000300	0.08044	0.09521	0.000537	0.000870	0.08044	0.000010	0.000183	0.007919	0.002285	5.198	0.020380	0	1	0.005768
600000000	900000000	3418319	0	431052910	0.000461	0.07715	0.08740	0.000625	0.000852	0.07715	0.000002	0.000204	0.005112	0.000215	2.937	0.009449	0	1	0.009277

Fonte: O autor.

ANEXO C – Exemplo do Dataset Processado

Tabela 14 – Fragmento do conjunto de dados processado.

Prioridade	CPU solicitada	RAM solicitada	Disco solicitado	CPU média	Memória canônica	Memória atribuída	Cache não mapeado	Cache total	Memória máxima	Tempo médio disco	Espaço médio disco	CPU máxima	Tempo máximo disco	Ciclos instrução	Memória acessada instrução	Amostra CPU	Duração tarefa	Tipo evento
4	0.062500	0.03180	0.000772	0.029300	0.002819	0.004044	0.000879	0.001520	0.004753	0.020630	6.866600e-06	0.072510	0.090330	0.666900	0.000354	0.000796	3.100000e+07	3
0	0.009369	0.03107	0.000038	0.036301	0.006938	0.007957	0.000006	0.000630	0.008355	0.000148	3.815000e-06	0.071184	0.002984	1.196979	0.001665	0.037582	4.582000e+09	3
4	0.062500	0.03180	0.000772	0.028470	0.003048	0.004089	0.001040	0.001722	0.005104	0.019350	4.768500e-07	0.066280	0.056760	1.257000	0.001971	0.002075	3.200000e+07	4
4	0.062500	0.03180	0.000772	0.009354	0.001593	0.002728	0.000813	0.001286	0.003223	0.029970	6.866650e-06	0.082520	0.072020	1.736000	0.003877	0.002338	2.200000e+07	3
0	0.062534	0.04663	0.001131	0.044400	0.005342	0.006533	0.001147	0.001478	0.006072	0.001363	6.866700e-06	0.074776	0.025249	0.622978	0.000238	0.047744	1.000000e+09	3

Fonte: O autor.