UNIVERSIDADE DO ESTADO DE SANTA CATARINA - UDESC
CENTRO DE CIÊNCIAS TECNOLÓGICAS - CCT
MESTRADO EM COMPUTAÇÃO APLICADA

JULIA GRANDO

ADAPTAÇÃO AO XGBOOST PARA SUPORTE À REGRESSÃO DE FLUXO DE DADOS NÃO-ESTACIONÁRIOS

JOINVILLE

2023

JULIA GRANDO

ADAPTAÇÃO AO XGBOOST PARA SUPORTE À REGRESSÃO DE FLUXO DE DADOS NÃO-ESTACIONÁRIOS

Dissertação submetida ao Programa de Pós-Graduação em Computação Aplicada do Centro de Ciências Tecnológicas da Universidade do Estado de Santa Catarina, para a obtenção do grau de Mestre em Computação Aplicada.

Orientador: Dr. Fabiano Baldo

JOINVILLE

2023

Ficha catalográfica elaborada pelo programa de geração automática da

Biblioteca Universitária Udesc,

com os dados fornecidos pelo(a) autor(a)

Grando, Julia
ADAPTAÇÃO AO XGBOOST PARA SUPORTE À REGRESSÃO DE FLUXO DE DADOS NÃO-ESTACIONÁRIOS / Julia Grando. -- 2023.
60 p.

Orientador: Fabiano Baldo
Dissertação (mestrado) -- Universidade do Estado de Santa Catarina, Centro de Ciências Tecnológicas, Programa de Pós-Graduação em Computação Aplicada, Joinville, 2023.

1. regressão. 2. árvores de decisão. 3. fluxo de dados. I. Baldo, Fabiano. II. Universidade do Estado de Santa Catarina, Centro de Ciências Tecnológicas, Programa de Pós-Graduação em Computação Aplicada. III. Título.

JULIA GRANDO

ADAPTAÇÃO AO XGBOOST PARA SUPORTE À REGRESSÃO DE FLUXO DE DADOS NÃO-ESTACIONÁRIOS

Esta dissertação foi julgada adequada para a obtenção do título de **Mestre em Computação Aplicada** área de concentração em "Ciência da Computação", e aprovada em sua forma final pelo Curso de Mestrado em Computação Aplicada do Centro de Ciências Tecnológicas da Universidade do Estado de Santa Catarina.

BANCA EXAMINADORA

Dr. Fabiano Baldo (Orientador)
CCT/UDESC

Membros:

Dr. Rafael Stubs Parpinelli
CCT/UDESC

Dr. Jean Paul Barddal
PUC-PR

Joinville, 04 de julho de 2023

AGRADECIMENTOS

Agradeço primeiramente ao professor Fabiano por toda orientação, conversas e ensinamentos durante esses dois anos e meio. Foi uma satisfação desenvolver este trabalho o tendo como orientador.

Gostaria de agradecer também minha família: meu pai Grando, minha mãe Margaret e minha irmã Elisa que sempre me apoiaram e nunca mediram esforços para apoiar meu crescimento.

Agradeço também à WEG por ter facilitado a realização deste trabalho, em especial Fusi, Sieg e Leonardo pelo incentivo que sempre me foi dado.

Por fim, muito obrigada aos colegas Deividy, Yuji, Fernanda e Kawan que de diversas formas contribuíram com este trabalho.

“A questão é: não resistir ao fluxo. Você sobe quando deveria subir e desce quando deveria descer. Quando for subir, encontre a torre mais alta e suba até o topo. Quando for descer, encontre o poço mais fundo e desça até o fundo. Quando não houver fluxo, fique parado. Se você resistir ao fluxo, tudo seca. Se tudo secar, o mundo é escuridão.” — Haruki Murakami, Crônica do Pássaro de Corda

RESUMO

Algoritmos de aprendizado de máquina estão sendo cada vez mais utilizados em diversos problemas, muitos desses que necessitam de uma resposta em tempo real. Além da velocidade de resposta deve-se considerar que o padrão dos dados pode ser dinâmico, ou seja, com o passar do tempo o comportamento do conjunto de dados pode sofrer alterações, um fenômeno chamado mudança de conceito. Para que o algoritmo seja capaz de se adaptar às mudanças de conceito existentes nos dados, além de fornecer um curto tempo de resposta, deve-se trabalhar com o tratamento de fluxo de dados, atualizando o modelo à medida que novos dados se tornam disponíveis. A área de classificação de fluxo de dados vem sendo amplamente investigada nos últimos anos, porém foi percebida uma carência na área de regressão de fluxo de dados. Entre os trabalhos encontrados, é comum o uso de conjuntos de árvores de decisão, utilizando as técnicas de *bagging* e *Random Forest* para a geração de modelos, porém o *boosting* foi pouco explorado. Com o objetivo de executar a regressão de fluxo de dados não estacionários, este trabalho propõe duas versões do LAX-Reg, um algoritmo de regressão adaptativo rápido baseado em XGBoost, que tem como base o *Gradient Boosting* para gerar um conjunto de árvores de decisão. A lógica do algoritmo se baseia na exclusão e criação de novas árvores a cada rodada de treinamento, atualizando constantemente o modelo para a previsão de novos conceitos e evitando que este cresça indefinidamente. O LAX-Reg foi comparado com outros cinco algoritmos do estado da arte de regressão de fluxo de dados, sendo avaliado em 9 conjuntos de dados reais e sintéticos com diferentes tipos de mudança de conceito, comparando seu erro médio quadrático, tempo de teste, tempo de treinamento e memória alocada. Os resultados dos experimentos indicam que o LAX-Reg é um algoritmo eficiente e enxuto para a regressão de fluxo de dados, pois apresentou menor erro que os demais algoritmos, enquanto que sua execução é mais rápida e ocupa menos memória.

Palavras-chaves: regressão, árvores de decisão, fluxo de dados

ABSTRACT

Machine learning algorithms are being increasingly used in various problems, many of which require a real-time response. In addition to response speed, one should consider that the data pattern can be dynamic, that is, over time the behavior of the data can change. In order for the algorithm to be able to adapt to concept changes in the data, in addition to providing a short response time, one must work with data stream treatment, updating the model as new data become available. The area of data stream classification has been widely investigated in recent years, but a lack of data stream regression studies was noticed. Among the studies found, it is common to use ensembles of decision trees, using the techniques of *bagging* and *Random Forest* to generate models, but *boosting* was seldom explored. In order to perform non-stationary data flow regression, this work proposes two versions of LAX-Reg, a fast adaptive regression algorithm using XGBoost, which is based on *Gradient Boosting* to generate an ensembles of decision trees. The algorithm's logic is based on the exclusion and creation of new trees at each training round, constantly updating the model to predict new concepts and preventing it from growing indefinitely. LAX-Reg was compared with five other state-of-the-art data stream regression algorithms, being evaluated on 9 real and synthetic datasets, with different types of concept drifts, comparing its mean squared error, test time, training time and used memory. The results of experiments show that LAX-Reg is an efficient and lean algorithm for data stream regression, because it has a smaller error than the other algorithms, while its execution is faster and it consumes less memory.

Key-words: regression, decision trees, data stream, nonstationary time series

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ADWIN	*Adaptive Windowing*
AM	Aprendizado de Máquina
AMRules	*Adaptive Model Rules*
ARF	*Adaptive Random Forest*
AFXGB	*Adaptive Fast eXtreme Gradient Boosting*
AXGB	*Adaptive eXtreme Gradient Boosting*
CC	*Correlation Coefficient*
CD	*Critical Difference*
DDM	*Drift Detection Method*
FIMT-DD	*Fast and Incremental Model Trees*
IoT	*Internet of Things*
KSWIN	*Kolmogorov-Smirnov Windowing*
LAX	*Lean Adaptive XGBoost*
MAE	*Mean Absolute Error*
MAPE	*Mean Absolute Percentage Error*
ME	*Mean Error*
MSE	*Mean Squared Error*
ORTO	*On-line Regression Trees with Options*
RE	*Relative Error*
RMSE	*Root Mean Squared Error*
RRSE	*Root Relative Squared Error*
XGBoost	*eXtreme Gradient Boosting*

SUMÁRIO

1 INTRODUÇÃO

Devido à difusão na adoção e utilização de dispositivos eletrônicos conectados, dotados de sensores e interatividade social, dados são gerados constantemente e cada vez com maior frequência. Dessa forma, a quantidade de dados disponíveis para análise e auxílio à tomada de decisão tem se tornado cada vez mais abundante. Por consequência, a demanda por aprendizado de máquina está aumentando devido à grande quantidade de dados disponíveis (MAHESH, 2020). Muitos setores aplicam aprendizado de máquina para aprender com os dados e extrair informações relevantes, utilizando-os de diversas maneiras e abordando problemas como a previsão de tráfego de veículos, consumo de energia elétrica, prevenção de fraudes e até mesmo surtos de doenças (IKONOMOVSKA; GAMA; DŽEROSKI, 2015).

No contexto do aprendizado de máquina, grande parte das pesquisas se concentra na construção de modelos estáticos. Esses modelos são primeiramente treinados sobre um conjunto de dados de treinamento e depois são aplicados à análise de novos dados, caracterizando-os como uma abordagem assíncrona de aprendizado de máquina (GAMAGE; PREMARATNE, 2017). Todavia, a aplicação do aprendizado de máquina a fluxos de dados tem crescido em importância nos últimos anos devido à grande quantidade de dados gerados por dispositivos conectados por redes, tais como, telefones celulares, dispositivos inteligentes, sensores de IoT, entre outros. Esses dispositivos geram dados de forma ininterrupta, o que requer sua análise em tempo real. Portanto, um fluxo de dados pode ser definido por uma sequência de dados não estacionários gerados de forma constante e por tempo indeterminado (YU; LU; ZHANG, 2021). Segundo Pinagé et al. (2017), uma sequência de dados não estacionária é caracterizada pela mudança na distribuição dos valores dos seus atributos e/ou da variável alvo ao longo do tempo.

A construção de modelos preditivos a partir de fluxos de dados requer o emprego de algoritmos capazes de aprender de forma incremental. Neste tipo de abordagem, o modelo é atualizado à medida que novos dados se tornam disponíveis ao longo do tempo. Aprender com fluxos de dados apresenta desafios, tais como, a impossibilidade de armazenar todos os dados, pois o fluxo de dados pode ser infinito. Neste cenário, os dados não estão disponíveis em sua totalidade em nenhum momento do processo de aprendizagem, ou seja, é observada apenas uma amostra dos dados de cada vez (KREMPL et al., 2014). Portanto, os modelos de aprendizado para fluxo de dados devem ser capazes de assimilar a característica do conjunto de dados e tomar decisões suportadas por uma única passagem pelos dados de treinamento, sem ter que armazenar nenhuma das instâncias de dados. Além disso, conforme pontuado

por Ikonomovska, Gama e Džeroski (2015), devem processar grandes quantidades de dados em um curto espaço de tempo para que não hajam atrasos nas predições e atualizações dos modelos.

Outro ponto fundamental é a dinâmica de atualização do modelo para lidar com fluxos de dados não estacionários. Segundo Krawczyk et al. (2017), é necessário definir como e quando o modelo será redefinido ou atualizado para refletir as alterações na distribuição de dados. Essas alterações são chamadas de mudanças de conceito e ocorrem quando as propriedades estatísticas da variável alvo mudam ao longo do tempo de maneira imprevista (LIAO; WANG, 2018).

Sobre a variável alvo, ela pode ser discreta ou contínua. Para o caso da variável ser discreta, ela é chamada de classe e o processo de aprendizado é denominado classificação. No caso da variável ser contínua, o processo de aprendizado é chamado de regressão. Segundo Gomes et al. (2018), ainda que anualmente sejam publicadas diversas pesquisas acerca de classificação de fluxo de dados, a regressão de fluxo de dados ainda é pouco estudada.

Com relação ao tipo de modelo utilizado para o aprendizado de fluxos de dados, os algoritmos que produzem vários modelos, chamados de métodos *ensemble* ou conjuntos de modelos, são uma tendência para o aprendizado de máquina tanto para dados estáticos quanto para fluxos de dados (GOMES et al., 2020). Os conjuntos de modelos favorecem uma melhora da predição devido à diversificação e consenso da predição de cada modelo individual, potencializando sua precisão. Portanto, eles permitem alavancar o poder de vários modelos simples em direção a uma mesma previsão, mitigando assim as limitações dos modelos individuais. Sendo assim, são considerados uma das abordagens de aprendizado de máquina mais eficazes na solução de problemas de classificação e regressão (IKONOMOVSKA; GAMA; DŽEROSKI, 2015).

Um dos tipos de algoritmos mais investigado, utilizado para criação de conjuntos, é o baseado em árvores de decisão. Eles comumente utilizam uma estratégia de criação de modelos baseada em uma das seguintes técnicas: *bagging*, *boosting* ou randomização (DIETTERICH, 2000). O uso e eficácia de conjuntos de árvores de decisão para regressão de fluxo de dados pode ser observado nos trabalhos de Ikonomovska, Gama e Džeroski (2015), Sousa e Gama (2017), Gomes et al. (2018), Gomes et al. (2020). No entanto, entre as técnicas de construção de conjuntos de árvores investigadas nesses trabalhos, a técnica de *boosting* é pouco explorada.

Conjuntos baseados em *boosting* são avaliado na regressão de conjuntos estáticos desde Duffy e Helmbold (2002). Um algoritmo de *boosting* com relevância bastante reconhecida em vários desafios e problemas de aprendizado de máquina e mine-

ração de dados é o *XGBoost*. Implementado por Chen et al. (2015), seu nome significa *eXtreme Gradient Boosting*, e é uma otimização do método de *Gradient Boost* (FRIEDMAN, 2001). Os trabalhos que fazem uso do *XGBoost* obtiveram, na maior parte dos casos, resultados superiores em termos de precisão e velocidade de resposta (CHEN; GUESTRIN, 2016; OGUNLEYE; WANG, 2019; QIU et al., 2021; DHALIWAL; NAHID; ABBAS, 2018).

O desempenho do *XGBoost* para a classificação de fluxo de dados foi avaliado por Baldo et al. (2022), onde o algoritmo proposto obteve a mesma acurácia que algoritmos de classificação consolidados na literatura, porém, apresentando velocidade de treinamento e teste consideravelmente superior a eles. Já na tarefa de regressão de fluxos, Souza, Grando e Baldo (2022) portou o algoritmo de Baldo et al. (2022) e também obteve desempenhos notáveis de precisão e tempo de execução. Porém, foi percebido que ambos os algoritmos possuem uma deficiência em sua estratégia de treinamento, que acarreta em uma perca de precisão.

Levando em consideração o desempenho em termos de acurácia e velocidade de processamento obtido pelos trabalhos que utilizam o *XGBoost* como algoritmo base para tratamento de fluxos de dados, e as oportunidades identificadas de aprimorá-lo para a tarefa de regressão, este trabalho apresenta a seguinte pergunta de pesquisa: Como portar o *XGBoost* para suportar a regressão de fluxo de dados não-estacionários de forma rápida, obtendo uma acurácia superior ou compatível com as abordagens presentes na literatura?

A necessidade de processamento rápido é uma premissa básica dos fluxos de dados, pois eles costumam ser originados de fontes compostas por sensores e outros dispositivos com alta frequência de amostragem. Portanto, caso o algoritmo de aprendizado não seja rápido o suficiente para processar todos os dados recebidos pelo fluxo, acarretará na geração de fila e perda de informações por descarte de dados represados na fila.

1.1 OBJETIVO

Este trabalho tem como objetivo construir um algoritmo de regressão de fluxo de dados não-estacionários utilizando como base a biblioteca *XGBoost*, chamado de *Lean Adaptive XGBoost for Regression* (LAX-Reg). Ele deverá alcançar níveis iguais ou menores de erro em comparação aos outros algoritmos de regressão de fluxo de dados presentes na literatura, entretanto, com tempo de treinamento e teste menores, tornando assim o processo de regressão dos fluxos mais rápido.

1.1.1 Objetivos Específicos

Com base no objetivo geral apresentado, os seguintes objetivos específicos são definidos:

1. Projeto da arquitetura base do algoritmo para regressão de fluxos de dados utilizando XGBoost, como uma evolução do trabalho de Souza, Grando e Baldo (2022);
2. Projeto da abordagem de detecção de mudança de conceito;
3. Projeto da abordagem de atualização dos regressores;
4. Implementação do algoritmo projetado em Python;
5. Realização de experimentos e análises comparativas entre o algoritmo proposto e algoritmos estado da arte encontrados na literatura.

1.2 METODOLOGIA DE PESQUISA

Esta seção apresenta a caracterização metodológica da pesquisa, assim como o detalhamento do procedimento metodológico utilizado no desenvolvimento deste trabalho.

1.2.1 Caracterização Metodológica

Baseando-se nas classificações definidas por Wazlawick (2017), esta pesquisa pode ser considerada de natureza primária, por propor a produção de um novo conhecimento por meio da construção e experimentação de um algoritmo de aprendizado de máquina ainda não abordado na literatura para solução do problema de regressão de fluxo de dados.

Em relação ao seu objetivo, esta pesquisa é considerada como explicativa, por apresentar dados acerca dos experimentos e análise do comportamento do algoritmo proposto em diversos conjuntos de dados. Quanto ao procedimento técnico, pode ser classificada como experimental, pois são feitas alterações nos hiperparâmetros do algoritmo proposto e, em seguida, são feitas medições e observações sobre os resultados, com o objetivo de validar as hipóteses de pesquisa definidas, por meio de validações estatísticas.

1.2.2 Procedimento Metodológico

O procedimento metodológico desta pesquisa inicia com a revisão bibliográfica de conceitos sobre regressão de dados, conjuntos de árvores de decisão, fluxos de da-

dos, técnicas de detecção de mudança de conceitos e estudo da biblioteca *XGBoost*. A partir da fundamentação dos conceitos base para o desenvolvimento do trabalho, é realizado o projeto da arquitetura base do algoritmo de regressão proposto. Essa arquitetura usa como referência os trabalhos desenvolvidos por Baldo et al. (2022) e Souza, Grando e Baldo (2022), onde o *XGBoost* é utilizado como base para a classificação binária e regressão de fluxos de dados, respectivamente. A partir da criação da estrutura base da arquitetura do algoritmo, serão adicionadas abordagens para a detecção e atualização dos modelos a medida que ocorrerem mudanças de conceito. Finalizada a etapa de projeto, será iniciada a implementação do algoritmo proposta, utilizando a linguagem Python e as bibliotecas Scikit-learn e Scikit-multiflow.

Após concluir a implementação, serão feitas análises sobre o desempenho do algoritmo proposto utilizando métricas, tais como: erro médio quadrático, tempo de treinamento, tempo de teste, tempo total e memória alocada. Será feito uso de um *framework* para simulação dos fluxos de dados obtidos de bases de dados reais e sintéticas, onde o algoritmo proposto será comparado com algoritmos de regressão de fluxo de dados consolidados na literatura, tais como: HATR (BIFET; GAVALDA, 2009), AMRules (DUARTE; GAMA; BIFET, 2016) e ARFReg (GOMES et al., 2018). As bases de dados utilizadas devem apresentar alteração no comportamento da variável objetivo ao passar do tempo, para que seja possível avaliar a capacidade do modelo em se adaptar às mudanças de conceito.

Por fim, será feita a avaliação estatística sobre os resultados obtidos pelo algoritmo proposto em comparação aos dos algoritmos da literatura, a fim de avaliar se o algoritmo apresentou melhoras significativas na acurácia, tempos de treinamento e tempo de teste.

1.3 ESTRUTURA DO TRABALHO

O restante deste trabalho está estruturado da seguinte forma. O Capítulo 2 apresenta a fundamentação teórica da pesquisa, expondo as definições sobre aprendizado de máquina, árvores de regressão, conjuntos de árvores de regressão e fluxo de dados. Também nesta seção é apresentada a revisão dos trabalhos relacionados. O capítulo 3 apresenta o modelo proposto até o momento. O capítulo 4 apresenta os resultados parciais alcançados e por fim, no Capítulo 5 são realizadas as considerações parciais do trabalho e antecipação das próximas etapas.

2 REFERENCIAL TEÓRICO

Neste capítulo é apresentada a fundamentação teórica dos conceitos abordados por este trabalho, que dizem respeito ao aprendizado de máquina utilizando árvores de decisão, particularmente focados na regressão de fluxos de dados, com a ocorrência de mudança de conceito. Ainda, serão descritos os trabalhos relacionados encontrados na literatura, fazendo uma contextualização do estado da arte na regressão de fluxos de dados não estacionários.

2.1 APRENDIZADO DE MÁQUINA

Popularizado por Samuel (1959), o termo Aprendizado de Máquina (AM) é definido como o campo de estudo que dá aos computadores a habilidade de aprender sem serem explicitamente programados. Segundo Muhammad e Yan (2015), o AM pode ser considerado como um subcampo da Inteligência Artificial, uma vez que os algoritmos de AM são utilizados para fazer com que os computadores se comportem de forma mais inteligente.

De acordo com (MAHESH, 2020), esta inteligência se dá pelo fato de que os algoritmos de AM ensinam os computadores a lidar com os dados com mais eficiência, generalizando-os de alguma forma, em vez de apenas armazenar e recuperar dados como sistemas comuns fazem. A função principal dos algoritmos de AM é a de fazer previsões com base em experiências passadas, ou seja, aprendendo com as características dos conjuntos de dados.

Os algoritmos de AM podem ser categorizados de acordo com o seu tipo de aprendizado e sua função objetivo. Essas categorias serão abordadas nas seções 2.1.1 e 2.1.2, respectivamente.

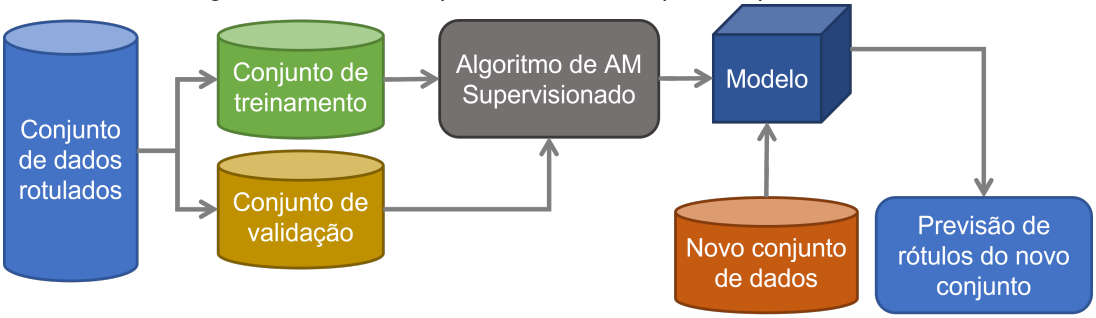
2.1.1 Tipos de Aprendizado

Conforme destacado por Kostopoulos et al. (2018), existem três tipos principais de Aprendizado de Máquina:

- **Supervisionado**: É o tipo de aprendizado em que uma função f é definida utilizando um conjunto de exemplos rotulados $L = \{(\vec{x_1}, y_1), (\vec{x_2}, y_2), ..., (\vec{x_n}, y_n)\}$, tal que $\vec{x_i}$ representa o vetor de atributos do dado e y_i o seu rótulo. Este conjunto rotulado é apresentado a um algoritmo de aprendizado de máquina, responsável por criar a função f, e tem como resultado final a geração de um modelo. Este modelo é capaz de prever os rótulos y para dados ainda não rotulados. A Figura

1 apresenta o fluxograma do processo de aprendizado supervisionado. Destaca-se a separação do conjunto de dados rotulados entre treinamento e validação, pois durante a etapa de aprendizado deve-se garantir que o modelo seja capaz de fazer previsões adequadas, comparando suas previsões $\hat{y}$ com os rótulos y do conjunto de validação.

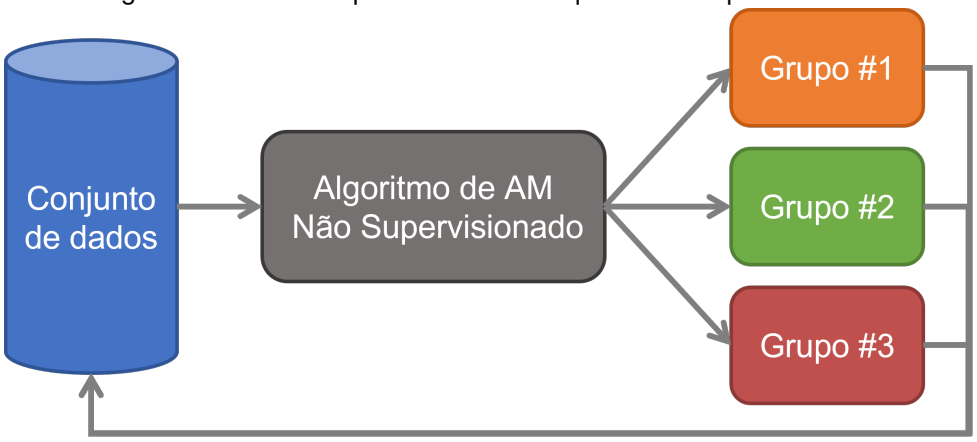
Figura 1 – Fluxo do Aprendizado de Máquina Supervisionado



Fonte: A autora.

- **Não supervisionado**: É o tipo de aprendizado cujo propósito é encontrar semelhanças ou associações dentro do conjunto de dados, agrupando ou relacionando exemplos similares do conjunto, sem que haja intervenção humana. Sendo assim, é capaz de trabalhar com um conjunto de dados não rotulados, pois não precisa possuir conhecimento da saída esperada para cada exemplo do conjunto de entrada. A Figura 2 ilustra um exemplo de fluxo de operação de um algoritmo não-supervisionado. É possível notar que este tipo de AM não gera um modelo, fornecendo como saída apenas o agrupamento dos exemplos do conjunto de entrada.

Figura 2 – Fluxo do Aprendizado de Máquina Não Supervisionado



Fonte: A autora.

- **Semi-supervisionado**: É o tipo de aprendizado que combina o aprendizado supervisionado e o não supervisionado, fazendo uso de dados rotulados e não rotulados para treinamento do modelo. Portanto, ele busca aproveitar ao máximo

os dados não rotulados para aprimorar a eficiência de predição do modelo treinado.

Como pode ser observado pela caracterização de cada tipo de aprendizado, pode-se dizer que o aprendizado supervisionado é adequado para problemas que possuem apenas dados rotulados, o não supervisionado para problemas cujos dados não são rotulados, e o semi-supervisionado para problemas em que estão presentes dados rotulados e não-rotulados.

2.1.2 Função Objetivo

A função objetivo denota o resultado final que se deseja atingir com o modelo de AM. Conforme observado por Kostopoulos et al. (2018), existem dois problemas principais que podem ser solucionados por algoritmos de AM supervisionados e semi-supervisionados, sendo eles:

- **Classificação**: tem por objetivo estimar rótulos de y categóricos por meio de uma função f discreta. Esses rótulos são chamados de classes. É possível medir o erro de previsão do modelo entre a classe predita e a prevista por meio de métricas tais como: acurácia, precisão, cobertura, Kappa e etc.
- **Regressão**: tem como objetivo estimar rótulos de y numéricos utilizando uma função f contínua. Os rótulos assumem valores que representam a média dos valores de y para dados com valores para o vetor de atributos $\vec{x}$ semelhantes. O erro de previsão pode ser medido pela diferença entre o valor predito e o previsto, utilizando métricas tais como: Erro médio absoluto, Erro médio quadrático, Raiz quadrada do erro médio quadrado e etc.

Em resumo, a classificação possui a função de prever um rótulo de classe discreta, enquanto que a regressão possui a função de prever um valor contínuo (MAULUD; ABDULAZEEZ, 2020).

2.2 ÁRVORES DE REGRESSÃO

Uma árvore de regressão é um tipo de algoritmo de AM supervisionado que gera como resultado um modelo do tipo árvore de decisão. Esta árvore possui nós internos e nós folhas conectados por arestas em que cada nó interno representa uma tomada de decisão que guia o caminho até um nó folha que representa a resposta, ou seja, a predição da variável alvo (KOTSIANTIS, 2013). Nas árvores de regressão os nós folha possuem a solução para uma variável alvo contínua. As árvores de decisão

estão entre os algoritmos de AM mais populares devido à sua simplicidade e facilidade de interpretação (WU et al., 2008).

Segundo Witten et al. (2016), a árvore pode ser expressa por um grafo $G = (N, E)$, em que N representa um conjunto de nós $(n_1, n_2, \ldots, n_n)$ e A um conjunto de arestas $(a_1, a_2, \ldots, a_n)$. Cada aresta é composta um par de nós $a_m = (n_i, n_j)$, em que n_i é o nó de origem e n_j é o nó de destino da aresta. Desta forma, as árvores de regressão são consideradas grafos direcionados acíclicos, cujos nós folha apresentam um valor contínuo.

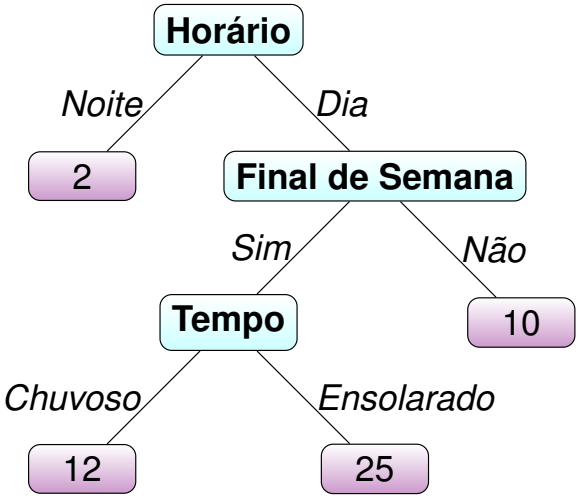
Para predizer o rótulo numérico de uma instância do conjunto de dados, deve-se percorrer o caminho entre o nó raiz da árvore até encontrar um nó folha, avaliando as condições existentes em cada um dos nós internos passados. Portanto, é necessário percorrer os nós consecutivamente, comparando em cada nó o atributo da instância avaliada com a condição de satisfação daquele atributo contido no nó. Para nós que possuem atributos numéricos, os testes consistem em definir um valor para tomar a decisão e avaliar se o valor contido naquele atributo da instância avaliada é “maior que” ou “menor que ” o valor de decisão. Para nós que comparam atributos discretos, é comum que existam tantas arestas quantos forem os valores para o atribuir utilizado para a tomada de decisão no nó.

A Figura 3 ilustra um exemplo de árvore de regressão. Neste cenário o objetivo é prever a quantidade de pessoas em um parque, utilizando os atributos de horário, dia da semana e previsão do tempo. Os nós de decisão estão representados em azul, as arestas em preto e os nós folha em roxo. A raiz da árvore é o atributo **Horário**, que se divide nas arestas de *Dia* e *Noite*. Caso seja noite, obtém-se a resposta de 2 pessoas no parque e a previsão está finalizada, porém caso seja dia deve ser avaliado o nó seguinte. O atributo **Final de semana** é avaliado como *Sim* ou *Não*: se não for final de semana, a resposta obtida é 10, caso contrário deve ser testado o atributo **Tempo**. Para o **Tempo**, caso a previsão seja *Chuvoso*, estima-se um total de 12 pessoas e, no caso de *Ensolarado*, estima-se 25 pessoas presentes no parque.

Conforme apresentado por Loh (2011), o problema de construção de uma árvore de decisão pode ser abstraído nas seguintes instruções:

1. Comece pelo nó raiz;
2. Encontre o atributo do conjunto de dados rotulados que minimiza a soma das impurezas dos nós filhos e crie uma aresta para cada teste relevante sobre o atributo;
3. Se um critério de parada for atingido, finalize. Caso contrário, aplique a etapa para cada nó filho.

Figura 3 – Árvore de regressão para previsão de quantidade de pessoas em um parque.



Fonte: A autora.

O procedimento descrito será repetido recursivamente até que todos os caminhos que saiam da raiz tenham como resultado uma solução para a variável alvo, ou seja, um nó folha. Percebe-se que a árvore é formada pela adição de nós de decisão de forma incremental, utilizando os exemplos rotulados do conjunto para guiar a escolha das comparações efetuadas (KINGSFORD; SALZBERG, 2008). Existem dois desafios principais na construção da árvore: i) qual é o melhor atributo para ser posicionado como nó, e ii) quais testes são relevantes em cada ramo da árvore de forma que ela se torne adequada para a solução do problema, ao mesmo tempo que mantenha a simplicidade e seja a menor árvore possível.

Para encontrar o melhor atributo a servir como critério para a tomada de decisão em cada nó da árvore, é necessário encontrar o atributo que mais diminua a soma das impurezas dos nós filhos. Esse valor é calculado pela redução do desvio padrão, representada na equação 2.1 como SDR, do inglês *Standard Deviation Reduction* (CHOU, 1991). O atributo que apresentar o maior SDR será escolhido como critério do nó da árvore.

$$SDR(S, x) = \sigma_S - \sigma_{xj} \tag{2.1}$$

O cálculo do SDR é realizado sobre o conjunto de dados rotulados $S = \{\vec{s_1}, \vec{s_2}, \ldots, \vec{s_n}\}$, em que cada amostra $\vec{s_i} = \{x_{1i}, x_{2i}, \ldots, x_{mi}, y_i\}$ representa um conjunto de atributos x_{ji} observados e de um valor alvo y_i. O σ_S representa o desvio padrão de todos os valores y_i do conjunto S, em contrapartida o σ_{xj} simboliza o desvio padrão dos subconjuntos, divididos pelos valores possíveis dos atributos x_j. Desta forma, os nós escolhidos são os que separam o conjunto total de dados em subconjuntos que possuem um desvio padrão menor do que o conjunto completo.

O critério de parada do algoritmo pode ser definido como um valor mínimo de desvio padrão do subconjunto ou a quantidade mínima de instâncias no subconjunto. Quando este critério é atingido, é criado um nó folha com a média dos valores y_i do subconjunto e o ramo da árvore é finalizado.

2.2.1 Conjuntos de Árvores de Regressão

Como já mencionado, as árvores de regressão são amplamente utilizadas no campo de aprendizado de máquina, pois têm como características sua fácil interpretação e implementação. Porém, conforme pontuado por Strobl, Malley e Tutz (2009), modelos compostos por uma única árvore apresentam dificuldades em se adaptar a mudanças nos dados, pois se tornam inflexíveis devido à escolha permanente de atributos para a formação de seus nós de decisão. Isto causa uma super especialização da árvore, fazendo com que suas previsões sejam tendenciosas ao conjunto de treinamento.

Com o intuito de minimizar este problema, é proposta a utilização de um conjunto de árvores, também chamado de *ensemble*. Conforme definido por Krawczyk et al. (2017), um *ensemble* é a união de modelos preditivos fracos cujas previsões são combinadas para a geração de um modelo mais preciso.

De acordo com Banfield et al. (2007), é possível encontrar na literatura as seguintes técnicas de construção de conjuntos de árvores de decisão:

- ***Bagging*:** Breiman (1996) apresentou a técnica de *Bootstrapped Aggregating*, abreviada como *Bagging*. Ela consiste em treinar diferentes árvores utilizando reamostragem *bootstrap* do conjunto de dados de treinamento original, ou seja, as árvores do conjunto são diversificadas já que são treinadas com diferentes amostras. Esta diversificação auxilia na diminuição do viés do modelo como um todo, e por consequência, aumenta a precisão do modelo.

- ***Random Forest*:** esta técnica, também apresentada por Breiman (2001), é considerada uma evolução do *Bagging* pois, além de utilizar a reamostragem *bootstrap*, os atributos do conjunto que serão utilizados no treinamento de cada árvore são selecionados aleatoriamente. Isto adiciona uma camada extra de diversidade às árvores do conjunto, melhorando ainda mais o desempenho preditivo do modelo.

- ***Boosting*:** a técnica de *boosting*, proposta por Schapire (1990), faz a construção do conjunto em etapas, de forma que as novas árvores são ajustadas em favor das instâncias calculadas incorretamente pelas árvores anteriores. Para isso, são atribuídos pesos às amostras a cada iteração de treinamento das árvores.

Sendo assim, o *boosting* almeja aprimorar o desempenho de cada nova árvore do conjunto de forma iterativa, com base nas árvores anteriores.

- ***Gradient Boosting*:** apresentada por Friedman (2001), é uma expansão do *boosting*, que utiliza o método do gradiente descendente para otimizar a função objetivo. Também faz a criação de novas árvores de forma sequencial, utilizando o erro das anteriores para o treinamento das novas.

Quando se faz uso de conjuntos de árvores, é necessário combinar os resultados das árvores individuais em uma única previsão final. Nas árvores de regressão geralmente isto é feito por meio de média simples, média ponderada ou mediana (STROBL; MALLEY; TUTZ, 2009).

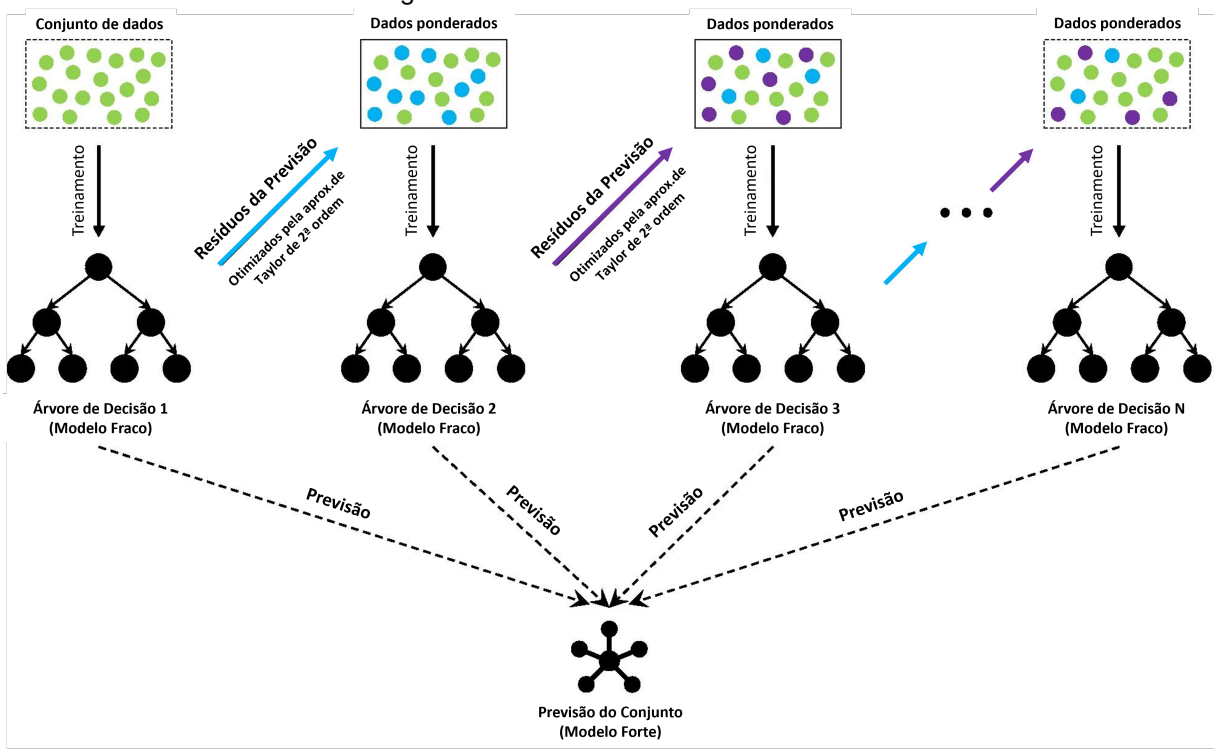
2.2.2 XGBoost

O *eXtreme Gradient Boosting*, abreviado como XGBoost, é um algoritmo proposto por Chen et al. (2015) que se baseia na técnica de *Gradient Boosting*. Ele faz a otimização da função objetivo utilizando uma aproximação de Taylor de segunda ordem, similar ao método de Newthon-Raphson, para ajustar o treinamento das árvores do modelo. Além disso, o XGBoost implementa diversas funcionalidades a fim de obter um alto desempenho, sendo elas a penalização de árvores, encolhimento de nós, implementação em multi-core para paralelização de criação de árvores, e seleção automática de atributos. Sua implementação está disponibilizada em uma biblioteca de código aberto.

A Figura 4 demonstra o funcionamento do XGBoost. Por ser baseado em *boosting*, o treinamento do modelo ocorre de forma sequencial, de modo que os erros da árvore anterior são utilizados para atualizar o peso dos dados do conjunto para o treinamento da árvore seguinte. Estes pesos são ajustados com base na função objetivo, que pode ser por exemplo o erro quadrático, também chamado de resíduo. O erro da previsão de cada instância é otimizado utilizando uma aproximação de Taylor de segunda ordem, que é aplicado ao conjunto, criando um conjunto de dados ponderado. O processo de otimização ocorre a cada treinamento de uma nova árvore, e é encerrado quando o conjunto atinge uma quantidade de árvores máxima, definida por parâmetro.

Após encerrada a etapa de treinamento dos modelos fracos é possível utilizar o conjunto de modelos, considerado um modelo forte, para realizar a previsão de novos dados. Cada modelo fraco executa sua previsão individual sobre o dado, e as previsões de todos os modelos são combinadas através de média ponderada para gerar a previsão final do conjunto.

Figura 4 – Funcionamento do XGBoost



Fonte: Adaptado de Deng et al. (2021).

2.3 FLUXO DE DADOS

O uso do aprendizado de máquina em fluxos de dados tem crescido em importância nos últimos anos devido ao enorme volume de dados gerados em tempo real por *smartphones*, sistemas, sensores e transações de banco de dados, e por isso, podem ser considerados como uma das principais fontes do chamado *Big Data*. De acordo com Sagiroglu e Sinanc (2013), o *Big Data* é composto por conjuntos de dados massivos com estrutura grande e complexa, em que se encontram dificuldades de armazenamento e análise desses dados. Portanto, um fluxo de dados pode ser entendido como uma sequência de dados gerados de forma contínua e por tempo ilimitado (YU; LU; ZHANG, 2021).

De acordo com Krempl et al. (2014), o aprendizado em fluxo de dados distingue-se do processo de aprendizado em conjuntos de dados estáticos devido às seguintes características: volume, velocidade e volatilidade. Essas características se tornam desafios que os algoritmos tradicionais de AM precisam contornar para se tornarem aptos a processar fluxos de dados. A Tabela 1 expõe os desafios e requisitos necessários para os algoritmos.

Conforme destacado por Liao e Wang (2018), atualmente, existe uma vasta literatura com propostas de abordagens para a classificação de fluxos de dados. Entretanto, essa mesma abundância de propostas não é encontrada quando se foca na

Tabela 1 – Desafios e requisitos para o aprendizado de fluxo de dados.

Desafio	**Requisitos do algoritmo**
Os dados não estão disponíveis em sua totalidade em nenhum momento do processo de aprendizagem, ou seja, é observada apenas uma amostra dos dados de cada vez;	Capacidade de aprender de forma incremental, em que o modelo é atualizado à medida que novos dados se tornam disponíveis ao longo do tempo;
Em diversas situações os dados são não estacionários, apresentando mudanças na distribuição dos seus atributos ao longo do tempo;	Deve possuir a característica de adaptação, sendo capaz de adequar-se rapidamente a novos conceitos, mantendo sua precisão;
Impossibilidade de armazenar todos os dados, pois o fluxo de dados pode ser infinito;	Deve ser capaz de assimilar a característica do conjunto de dados apenas com uma única passagem pelos dados de treinamento;
Rapidez do fluxo, grandes quantidades de dados devem ser processadas em um curto espaço de tempo para que não hajam atrasos nas predições;	Apresentar alto desempenho, com grande velocidade no treinamento e previsões do modelo.

Fonte: A autora.

regressão de fluxos de dados. Isso pode ser justificado, pois a área de estudo em análise de fluxos de dados é relativamente recente, sendo que as primeiras propostas abordaram classificação, o que se tornou o primeiro pilar de investigação na aplicação de AM em fluxos de dados. Entretanto, dos poucos trabalhos encontrados para regressão, os mais relacionados ao trabalho em tela são apresentados na seção 2.4.

2.3.1 Mudança de Conceito

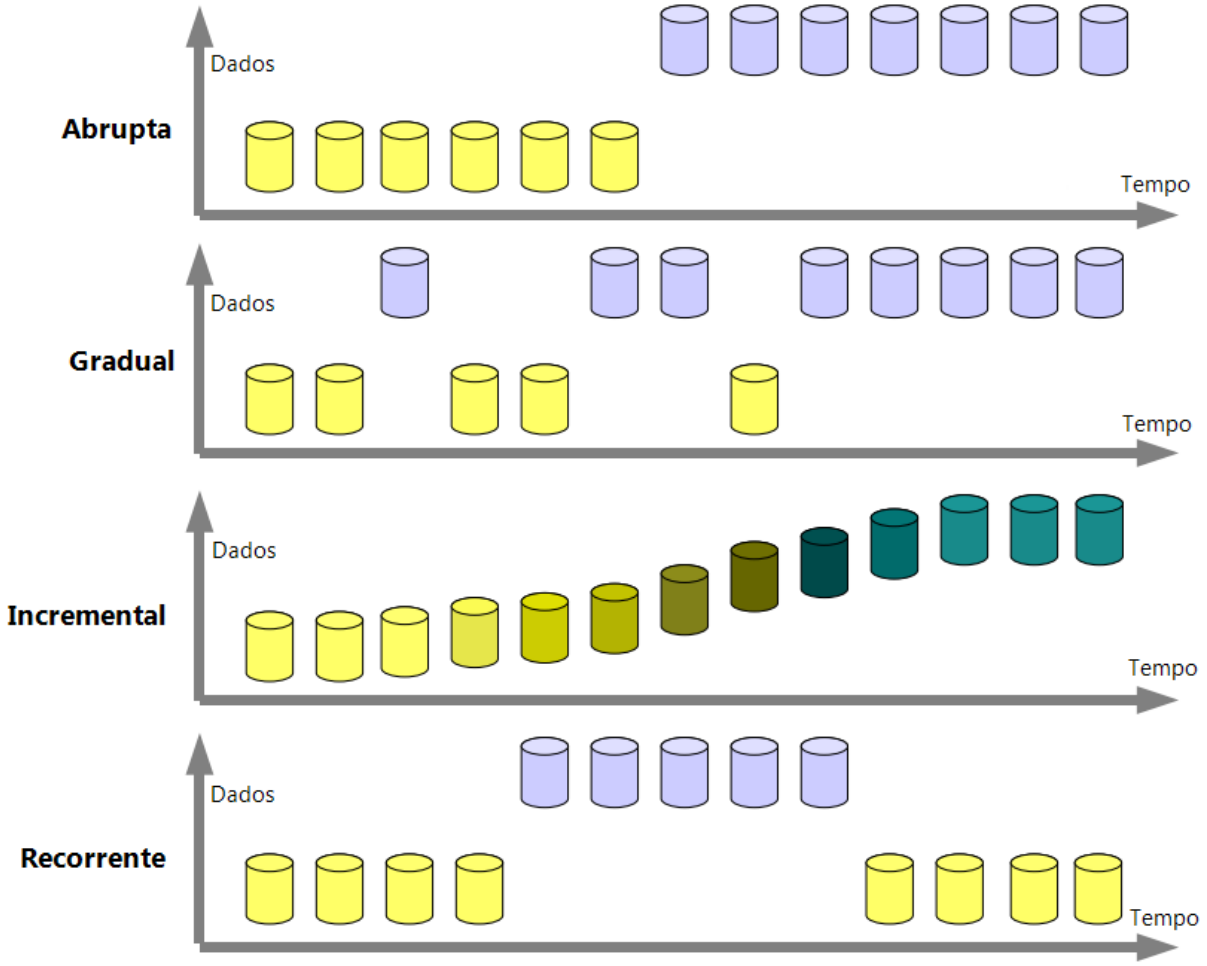
Entre os obstáculos para o AM em fluxos de dados mencionados na seção 2.3 se destaca a chamada mudança de conceito. Ela consiste na variação da distribuição dos dados durante sua geração no fluxo. Segundo Pinagé et al. (2017), ela ocorre de maneira imprevisível, ou seja, não se sabe em que momento ou de que forma irá ocorrer.

Um exemplo de mudança de conceito ocorreu durante a pandemia de COVID-19, que trouxe diversos impactos em áreas como saúde, economia, hábitos de consumo, transporte, turismo, entre outros. Vários padrões de dados que eram considerados duradouros foram alterados inesperadamente, alguns de forma temporária e outros de forma permanente. Outro domínio pertinente para os dias atuais e que sofre de mudanças de conceito é o de *IoT (Internet of Things)*, no qual transformações do ambiente em que os sensores estão situados podem acarretar na alteração da distribuição da variável alvo.

Conforme definido por Lu et al. (2018), existem quatro tipos de mudança de conceito em fluxos de dados. Eles estão ilustrados na Figura 5 e são os seguintes:

- **Abrupta**: o conceito é subitamente alterado, sendo substituído completamente por um novo;
- **Gradual**: o conceito é substituído gradualmente, isto é, durante um período a distribuição se alterna entre o novo conceito e o antigo, até que permaneça somente o novo conceito;
- **Incremental**: a mudança também ocorre de forma gradual, porém de forma mais longa, sem um período de transição bem definido;
- **Recorrente**: é quando um conceito que existia anteriormente, porém não está mais em vigor, ressurge depois de um período.

Figura 5 – Ilustração dos tipos de mudança de conceito.



Fonte: Adaptado de Žliobaitė (2010).

Para que o algoritmo de AM seja capaz de tratar as mudanças de conceito, os dois requisitos necessários são a detecção da mudança o mais breve possível e a adaptação do modelo para realizar predições de acordo com o novo conceito (BOSE et al., 2013). De acordo com Krawczyk et al. (2017), existem duas táticas principais para o tratamento de mudanças de conceito: passiva e ativa. Na abordagem passiva, o algoritmo atualiza o modelo constantemente à medida que novos dados se tornam disponíveis, sem levar em conta se estão ocorrendo mudanças nos dados. Em contrapartida, algoritmos que trabalham com a forma ativa possuem a capacidade de identificar desvios na distribuição dos dados e são atualizados somente quando estas detecções ocorrem (MAHDI et al., 2020).

Diversos algoritmos de AM para fluxo de dados fazem uso do tratamento de mudança de conceito na forma ativa por meio de algoritmos que são explanados na seção 2.3.1.1.

2.3.1.1 Detectores de Mudança de Conceito

Segundo Basseville, Nikiforov et al. (1993), os algoritmos de detecção de mudança de conceito possuem estratégias que identificam a variação na distribuição dos dados por meio da observação de pontos ou períodos de mudança. Estas mudanças são detectadas por dois cálculos principais: a dissimilaridade e o teste de significância estatística.

Durante o cálculo da dissimilaridade é feita uma estimativa de distância entre os dados, medindo a magnitude da mudança de conceito. Posteriormente, estas medidas de dissimilaridade são utilizadas no teste de hipótese, que avalia a significância estatística da mudança. É importante realizar a avaliação estatística da dissimilaridade, pois somente um aumento nesta medida não representa uma mudança de conceito, já que ruídos ou amostras isoladas podem impactar na dissimilaridade (LU et al., 2018).

Conforme observado por Bifet (2017), destacam-se dois algoritmos de detecção de mudança de conceito: teste de Page-Hinckley e ADWIN. O teste de *Page-Hinckley* atua calculando a média dos dados históricos e a média recente dos dados observados. Quando a diferença entre essas médias extrapola um limite definido é disparado um alarme de mudança de conceito na distribuição (BARDDAL, 2019).

Já o ADWIN, abreviação do inglês *Adaptive Sliding Window*, monitora uma sequência de dados oriundos do fluxo fazendo uso de janelas deslizantes (BARDDAL, 2019). O ADWIN mantém uma janela de dados recentes e corta esta janela, a dividindo em duas. Esta divisão ocorre em diferentes pontos e são analisadas as médias dos erros destas duas subjanelas. É então calculada a diferença entre as médias das

subjanelas de dados e, após isso, o teste estatístico utilizado pelo ADWIN verifica se esta diferença é maior do que o limite de corte que, caso positivo, indica uma detecção de mudança de conceito (BIFET; GAVALDA, 2007). Este método é capaz de detectar de forma eficaz desvios de conceito graduais, pois a janela deslizante cresce a longo prazo, comportando uma grande quantidade de exemplos históricos (YANG; MANIAS; SHAMI, 2021).

2.3.2 Métricas de Avaliação de Erro

Para determinar se um modelo gerado por um algoritmo de AM é adequado para a realização de previsões, é necessário realizar a avaliação do seu desempenho por meio de métricas avaliação. Nos métodos tradicionais de AM estas métricas são calculadas sobre um conjunto de dados de validação, que é diferente do conjunto utilizando no treinamento. Já no AM de fluxo de dados a métrica é calculada sobre as novas amostras provenientes do fluxo, e posteriormente essas amostras são usadas para treinamento do modelo.

Em ambos os casos a avaliação faz a comparação das previsões do modelo ($\hat{y}$) com relação aos rótulos já conhecidos do conjunto (y). Na regressão, elas podem ser chamadas de medidas de erro, pois a variável alvo é um valor contínuo e a comparação é feita medindo a distância absoluta entre y e $\hat{y}$, ou seja, por meio do cálculo de $|y - \hat{y}|$.

Conforme observado por Botchkarev (2019), as principais métricas para regressão são as seguintes:

- **Erro Médio Absoluto ou *Mean Absolute Error* (MAE)**: é calculado pela soma dos erros, dividido pela quantidade de dados avaliados n:

$$MAE = \frac{1}{n}\sum_{i-1}^{n}|y - \hat{y}| \tag{2.2}$$

- **Erro Quadrático Médio ou *Mean Squared Error* (MSE)**: similar ao MAE, porém o erro é elevado ao quadrado, aumentando o peso dos erros maiores:

$$MSE = \frac{1}{n}\sum_{i-1}^{n}|y - \hat{y}|^2 \tag{2.3}$$

- **Raiz do Erro Quadrático Médio ou *Root Mean Squared Error* (RMSE)**: uma normalização do MSE, aplicando a raiz quadrada, isto é, $RMSE = \sqrt{MSE}$:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i-1}^{n}|y - \hat{y}|^2} \tag{2.4}$$

- **Erro Médio Absoluto Percentual ou *Mean Absolute Percentage Error* (MAPE)**: é uma medida percentual da precisão, não influenciada pela magnitude da variável alvo. Isto é obtido através da divisão do erro pelo valor do rótulo:

$$MAPE = \frac{1}{n}\sum_{i-1}^{n}|\frac{y-\hat{y}}{y}| * 100\% \tag{2.5}$$

2.4 TRABALHOS RELACIONADOS

Nesta seção são apresentadas as referências de trabalhos relacionados a esta dissertação. As pesquisas apresentadas compreendem algoritmos de regressão de fluxo de dados que fazem uso de modelos baseados em árvores e trabalhos que utilizam o XGBoost para tratamento de fluxos de dados.

Um dos primeiros algoritmos desenvolvidos para construção de árvores incrementais de regressão de fluxo de dados foi o *Fast and Incremental Model Trees with Drift Detection* (FIMT-DD), proposto por Ikonomovska, Gama e Džeroski (2011). Baseado no algoritmo de Hoeffding Tree, o FIMT-DD mantém as estatísticas dos dados, fazendo os exemplos provenientes do fluxo percorrer a árvore até chegar em um nó folha e atualizando as estatísticas relevantes durante o percurso. Ele também inclui uma rotina de detecção de mudanças de conceito, que periodicamente adapta ramos das árvores em que se percebe um aumento da variância. No trabalho de Ikonomovska et al. (2011) foi proposto o chamado *On-line Regression Trees with Options* (ORTO), algoritmo que é um aprimoramento do FIMT-DD, incluindo nós de "opção" para tratar atributos que possuem o mesmo ganho, acelerando o processo de treinamento do modelo. Estes nós de opção fazem com que os exemplos provenientes do fluxo percorram mais ramos disponíveis da árvore, consequentemente intensificando a atualização do modelo.

Com base em *Decision Rules*, o algoritmo *Adaptive Model Rules* (AMRules), desenvolvido por Duarte, Gama e Bifet (2016), embasa seu funcionamento na geração de regras a partir dos exemplos observados no fluxo. Estas regras tem como objetivo encontrar regularidades nos dados que podem ser expressas na forma de uma regra Se-Então (FÜRNKRANZ; KLIEGR, 2015). Para suportar a mudança de conceito é aplicado um detector *Page-Hinkley* para cada regra, e as regras são eliminadas quando o detector as percebe como incoerentes com o conceito vigente do fluxo de dados. Em comparação com o FIMT-DD, o AMRules apresentou menores erros absolutos e médios quadráticos.

Explorando o uso de conjuntos de árvores para regressão de fluxos de dados, Ikonomovska, Gama e Džeroski (2015) propuseram os algoritmos *Online Random Forest* (ORF) e *Online Bagging* (OBag), que utilizam o FIMT-DD como modelo base. Os

autores compararam estes modelos com os desenvolvidos previamente e concluíram que o ORTO manteve os menores erros em suas previsões. Também fazendo uso de *Random Forest*, Gomes et al. (2018) apresentaram o *Adaptive Random Forest Regressor* (ARF-Reg), uma adaptação do algoritmo já consolidado para classificação de fluxo de dados de mesmo nome. O ARF-Reg também utiliza FIMT-DD como base, entretanto ele faz uso do detector de mudança de conceito ADWIN para cada árvore, eliminando as que estão com baixa precisão em comparação ao conceito vigente do fluxo. O ARF-Reg e suas variações apresentaram resultados promissores, demonstrando um menor erro médio quadrático em comparação com o ORTO e AMRules.

Com relação aos trabalhos que utilizam o XGBoost como modelo base, Montiel et al. (2020) propuseram o *Adaptive XGBoost* (AXGB), um algoritmo de classificação binária que utiliza um conjunto de modelos XGBoost. No AXGB os modelos XGBoost mais antigos são substituídos por modelos novos, treinados com dados mais recentes do fluxo, mantendo um número máximo de modelos no conjunto. Para treinar os classificadores é utilizada uma estratégia de janela deslizante adaptativa, uma espécie de *buffer* que armazena os dados recebidos do fluxo na forma de uma fila, criando lotes com os dados mais antigos para treinamento dos classificadores e depois descartando-os. Esta janela deslizante é adaptativa pois é definido um tamanho mínimo e máximo, e a cada rodada de treinamento o tamanho da janela é atualizado, crescendo exponencialmente até atingir o seu valor máximo. Portanto, no início do treinamento o tamanho da janela é pequeno para que os classificadores possam começar a serem treinados rapidamente. Entretanto, a medida que os treinamentos acontecem a janela aumenta para diminuir a frequência de criação de classificadores. Apesar do AXGB tratar a mudança de conceito passivamente pela renovação dos classificadores mais antigos do conjunto, para acelerar a adaptação do modelo a novos conceitos foi também implementado o detector ADWIN. Quando o ADWIN identifica uma mudança de conceito no fluxo de dados, a janela deslizante é redefinida para seu menor valor, acelerando a renovação dos modelos do conjunto pelo treinamento de novos modelos XGBoost.

Baseando-se no AXGB, Baldo et al. (2022) propuseram uma adaptação a ele para classificação de fluxos de dados utilizando apenas dois modelos XGBoost, ao invés de um conjunto de modelos XGBoost. O processo de treinamento consiste na atualização de um modelo XGBoost a cada janela deslizante processada, até o momento em que ele comece a se tornar lento e pesado. A partir deste momento, um novo modelo XGBoost é criado e substitui o modelo anterior. Essa proposta teve como objetivo diminuir o tempo de treinamento e teste dos modelos, já que o AXGB, por utilizar um conjunto de XGBoost, que é um conjunto de árvores de decisão, possui elevado tempo de processamento. Entretanto, ela apresenta situações de queda abrupta de

acurácia quando acontece a substituição do modelo atual pelo que iniciou seu treinamento. O uso de janela deslizante e ADWIN foi mantido neste trabalho, assim como estava presente no AXGB.

Também utilizando XGBoost, Souza, Grando e Baldo (2022) portaram o trabalho de Baldo et al. (2022) para a regressão de fluxos de dados. A lógica de treinamento do modelo manteve-se a mesma, apenas trocando o objetivo de classificação para regressão. Entretanto, para detecção de mudança de conceito, em vez de utilizar somente o ADWIN, foi proposto um vetor de detectores que eram utilizados para melhor identificar uma mudança de conceito no fluxo. Os detectores utilizados neste trabalho foram o ADWIN, o KSWIN e o DDM. Este trabalho mostrou uma precisão equivalente aos algoritmos estado da arte da literatura para a regressão de fluxos de dados, entretanto, com tempo de processamento consideravelmente inferior a eles. Porém, assim como no trabalho de Baldo et al. (2022), ele também apresenta queda acentuada na precisão quando o modelo atual é substituído pelo que iniciou seu treinamento.

Como pode ser observado, entre os trabalhos relacionados revisados, apenas três utilizavam o XGBoost para análise de fluxos de dados, sendo que apenas um faz especificamente regressão de fluxo de dados.

2.4.1 Considerações sobre os trabalhos relacionados

Dado o levantamento dos trabalhos relacionados, a Tabela 2 sintetiza os trabalhos descritos nesta seção. A partir da análise da tabela, é possível perceber um crescimento de pesquisas na área de regressão de fluxos de dados na última década.

O trabalho de Ikonomovska, Gama e Džeroski (2011), que apresenta o FIMT-DD, serve como base para diversos outros trabalhos, conforme observado na coluna “Modelo Base” da Tabela 2. Estes trabalhos propõem evoluções e melhorias ao FIMT-DD, com destaque para os trabalhos que implementam conjuntos de modelos. Com relação à abordagens de conjuntos, elas incluem principalmente técnicas de *Random Forest* e *Bagging*, com o *boosting* visto somente no trabalho de Souza, Grando e Baldo (2022), que utiliza o XGBoost como modelo base para regressão de fluxo de dados.

Em termos de desempenho, em Gomes et al. (2020) os algoritmos foram comparados e constatou-se um menor erro médio quadrático do ARF-Reg de Gomes et al. (2018). Porém, no trabalho de Souza, Grando e Baldo (2022) foi possível observar que o algoritmo deles equiparou-se ao ARF-Reg em precisão, enquanto que obteve um tempo de treinamento e de teste consideravelmente menor.

Apesar do trabalho de Souza, Grando e Baldo (2022) ter apresentado bons resultados, percebe-se que ainda há oportunidades de melhoria, já que a lógica do

algoritmo manteve-se a mesma que foi proposta por Baldo et al. (2022), idealizada para o objetivo de classificação. As principais limitações identificadas em sua implementação foram a perda de precisão causada pela troca de modelos. Portanto, para este trabalho, serão exploradas estratégias para mitigar esse problema apresentado no algoritmo de Souza, Grando e Baldo (2022), particularmente focadas na não substituição de modelos.

Com relação às métricas de avaliação, na coluna “Medidas Observadas” da Tabela 2 percebe-se que grande parte dos trabalhos utilizam o MSE ou o RMSE para medir a precisão do modelo, enquanto que somente alguns avaliam o tempo de execução e memória alocada.

Tabela 2 – Resumo dos trabalhos relacionados

Trabalho	Objetivo	Modelo Base	Detecção de Mudança de Conceito	Medidas Observadas
Ikonomovska, Gama e Džeroski (2011)	Regressão	Hoeffding Trees	Passiva	ME, MSE, RE, RRSE, CC
Ikonomovska et al. (2011)	Regressão	FIMT-DD	Passiva	MSE, Tempo, Memória alocada
Duarte, Gama e Bifet (2016)	Regressão	Decision Rules	Page-Hinckley	MAE, RMSE
Ikonomovska, Gama e Džeroski (2015)	Regressão	Random Forest, Bagging, FIMT-DD	Passiva	MSE, Tempo, Memória alocada
Gomes et al. (2018)	Regressão	Random Forest, FIMT-DD	Page-Hinckley, ADWIN	RMSE
Montiel et al. (2020)	Classificação	XGBoost	Passiva, ADWIN	Acurácia
Baldo et al. (2022)	Classificação	XGBoost	Passiva	Acurácia, Tempo
Souza, Grando e Baldo (2022)	Regressão	XGBoost	Passiva, ADWIN DDM, KSWIN	Acurácia, Tempo
Trabalho em tela	Regressão	XGBoost	Passiva, ADWIN	MSE, Tempo, Memória alocada

Fonte: A autora.

3 MÉTODO PROPOSTO

Como já mencionado, este trabalho tem por objetivo desenvolver um algoritmo de aprendizado de máquina para regressão de fluxos de dados não-estacionários utilizando como base o XGBoost. O XGBoost é uma biblioteca que utiliza a estratégia de *boosting* para a construção de conjuntos de árvores de decisão. Entretanto, o XGBoost não é um algoritmo desenvolvido para o tratamento de fluxos de dados não estacionários, portanto, ele apresenta limitações no tratamento das seguintes características dos fluxos: i) indisponibilidade do conjunto completo de dados no início do treinamento, ii) crescimento ilimitado do modelo devido ao fluxo poder ser infinito, iii) impossibilidade de se adaptar à mudanças de conceito.

Uma das poucas iniciativas já desenvolvidas que utiliza o XGBoost para a regressão é a proposta por Souza, Grando e Baldo (2022). Nela, foi desenvolvida uma adaptação sobre o trabalho de Baldo et al. (2022), com o objetivo de portar o algoritmo para suportar a regressão. Entretanto, por utilizar a mesma estratégia proposta por Baldo et al. (2022), seu trabalho apresenta a mesma limitação de aumento no erro sempre que acontece a troca de modelos.

O trabalho em tela tem o intuito de mitigar esse problema do crescimento do erro na substituição dos modelos por meio da aplicação de uma abordagem que não realiza a substituição de modelos, mas apenas atualização do modelo XGBoost existente. Portanto, este trabalho usa como solução para o problema i) a aplicação da mesma estratégia de janelas dinâmicas deslizante de dados proposta por Montiel et al. (2020), em que o fluxo é dividido em lotes de dados de tamanho variável para treinamento do modelo. Já as limitações ii) e iii) são mitigadas neste trabalho pela atualização de um único modelo XGBoost ao longo do decorrer do tempo. O algoritmo faz a exclusão de árvores, atualização das árvores remanescentes e criação de novas árvores, impedindo que o modelo cresça indefinidamente e renovando seus conceitos para refletir a característica dos dados mais atual, suportando assim a mudança de conceito. São propostas duas abordagens para a exclusão de árvores, as quais chamamos de FIFO e *Target*. Na abordagem FIFO as árvores mais antigas são excluídas, podendo ou não ser utilizado um detector de mudança de conceito para fazer o *reset* da janela. Já na abordagem *Target* é atribuído um detector de mudança de conceito individual para cada árvore do conjunto XGBoost e a exclusão das árvores é feita de forma pontual.

As seções 3.1 e 3.2 apresentam as duas versões do algoritmo proposto neste trabalho e descrevem os hiperparâmetros necessários para executá-los, respectiva-

mente. Por fim, a seção 3.3 apresenta os principais aspectos sobre sua implementação.

3.1 ALGORITMO

O algoritmo proposto neste trabalho, chamado LAX-Reg – *Lean Adaptive XGBoost Regressor*, usa a estratégia de treinamento e atualização de forma incremental de um único modelo XGBoost. Portanto, assim que novos dados chegam ao algoritmo na forma de fluxo, uma janela deslizante armazena os dados até que ela encha a sua capacidade, ou seja, até que a quantidade de dados seja igual ou maior que o tamanho máximo da janela. Quando a janela está completa, inicia-se o processo de treinamento. No primeiro treinamento, apenas são criadas novas árvores, já que o modelo se encontra vazio. A partir dos próximos treinamentos uma parte das árvores é eliminada em duas estratégias distintas, FIFO ou *Target*.

Fazendo uso dos dados correntes da janela deslizante de treinamento, as árvores remanescentes são atualizadas e são criadas novas árvores para repor as árvores eliminadas. Caso o modelo ainda não esteja com a quantidade máxima de árvores no conjunto, além de repor as árvores eliminadas são criadas mais árvores novas por rodada de treinamento, com a finalidade de ampliar o conjunto. Quando o modelo atinge seu tamanho máximo o modelo deixa de crescer e são criadas novas árvores apenas com o propósito de repor as árvores eliminadas.

As seções 3.1.1 e 3.1.2 apresentam respectivamente as estratégias de eliminação de árvores e os pseudocódigos do LAX-Reg FIFO e LAX-Reg *Target*.

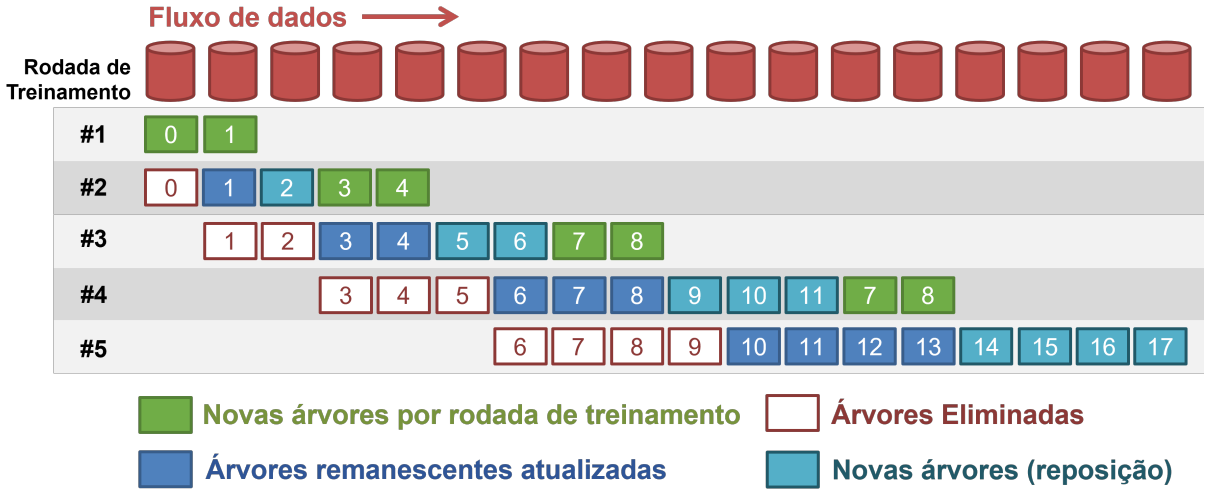
3.1.1 LAX-Reg FIFO

A Figura 6 representa o modelo de treinamento FIFO, onde é eliminada uma porcentagem das árvores mais antigas do modelo a cada rodada de treinamento. A cada rodada de treinamento estão destacadas as árvores eliminadas, remanescentes e as novas árvores criadas para reposição. Até que o modelo atinja seu tamanho máximo são criadas novas árvores, porém quando este número é alcançado, como no Treinamento #5 da Figura 6, são criadas apenas árvores para reposição.

A quantidade de árvores eliminadas, e por consequência, a quantidade de árvores novas criadas a cada rodada de treinamento é definida por parâmetro, assim como a quantidade máxima de árvores do conjunto e o tamanho mínimo e máximo da janela deslizante. O pseudocódigo do algoritmo proposto está apresentado no Algoritmo 1.

O algoritmo recebe como entrada as instâncias do fluxo de dados x e y, sendo que x representa o conjunto de atributos e y o valor da variável alvo da regressão.

Figura 6 – Exemplo de treinamento do modelo adaptativo de XGBoost - LAX-Reg FIFO.



Fonte: A autora.

Essas instâncias são adicionadas à janela deslizante (linha 2) e quando ela se torna cheia (linha 3), o algoritmo verifica se já existe um modelo de regressão XGBoost treinado (Linha 4). Caso não exista, ou seja, o modelo será treinado pela primeira vez, um novo regressor XGBoost é gerado (linha 16). Caso já exista um modelo treinado, é armazenado o número de árvores do modelo (linha 5) e calculada, de acordo com a porcentagem de atualização, a quantidade de árvores que permanecerão no modelo (linha 6).

Após isso, é feita a eliminação das árvores antigas, mantendo o percentual de árvores que serão atualizadas (linha 7). Destaca-se que o conjunto de árvores do modelo funciona como uma fila, onde as árvores mais novas são adicionadas ao final, portanto as árvores sempre estão ordenadas da mais antigas para a mais novas. Sendo assim, nota-se que o modelo trabalha com o método FIFO (*First In First Out*), eliminando sempre as árvores mais antigas. Depois da eliminação, as árvores remanescentes são atualizadas com os dados da janela W (linha 8), alterando os pesos dos atributos de cada nó e valores finais de predição, fazendo com que o modelo seja aprimorado para a regressão do conjunto de dados mais recente.

A partir deste ponto, inicia-se a rotina de cálculo da quantidade de árvores novas que serão criadas. Na linha 9 é verificado se a quantidade de árvores do modelo antes da exclusão das árvores já havia atingido o número máximo de árvores do conjunto. Caso tenha atingido, são criadas árvores somente para repor as árvores eliminadas, valor descoberto fazendo a conta de quantidade de árvores do modelo original menos a quantidade de árvores do modelo após a eliminação das árvores (linha 10). Se o modelo ainda não tiver o número máximo de árvores, então, além de repor as árvores eliminadas, é adicionado um número de novas árvores por rodada de treinamento (linha 12), com a finalidade de fazer o conjunto crescer gradualmente

Algorithm 1 Lean Adaptive XGBoost Regressor FIFO

Entrada: $(x, y) \in$ Fluxo de dados

1: **função** TREINAMENTO LAX-REG-FIFO(W = janela de amostras, `window_size` = tamanho da janela, `M` = modelo de regressão XGBoost, `max_trees` = número máximo de árvores, `trees_train` = árvores extras, `percent_up_tree` = porcentagem de árvores atualizadas, `detect_drift` = detecção de mudança de conceito, `min_window` = tamanho mínimo da janela)
2: Adiciona (x, y) na janela (W)
3: **se** $|W| \geq$ `window_size` **então**
4: **se** `M` $\neq$ `NULL` **então**
5: `prev_n_trees` $\leftarrow n\_trees($`M`$)$ ▷ Salva o número de árvores de M
6: `trees_to_update` $\leftarrow$ `prev_n_trees` $\times (1 -$ `percent_up_trees`$)$
7: `M` $\leftarrow$ `M[trees_to_update:]` ▷ Corte para eliminar árvores antigas
8: `M` $\leftarrow$ `M`$.update(W)$ ▷ Atualiza árvores restantes com dados da janela
9: **se** `prev_n_trees` $\geq$ `max_trees` **então**
10: `new_trees` $\leftarrow$ `prev_n_trees` $- n\_trees($`M`$)$ ▷ Repõe árvores eliminadas
11: **senão**
12: `new_trees` $=$ `prev_n_trees` $- n\_trees($`M`$) +$ `trees_train` ▷ Soma árvores novas por treinamento
13: **fim se**
14: `M` $\leftarrow$ `M`$.train(W,$ `new_trees`$)$ ▷ Treina novas árvores com dados da janela
15: **fim se**
16: `M` $\leftarrow train(W,$ `trees_train`$)$ ▷ Treina novo modelo
17: **fim se**
18: $W \leftarrow 0$ ▷ Limpa janela de dados
19: `window_size` $\leftarrow$ `window_size'` ▷ Ajusta tamanho da janela
20: **se** `detect_drift` $=$ `True` **então**
21: `error` $\leftarrow abs($`M`$.predict(x) - y)$ ▷ Calcula o erro absoluto
22: `ADWIN`$.add($`error`$)$ ▷ Adiciona erro ao ADWIN
23: **se** `ADWIN`$.drift\_detection =$ `True` **então**
24: `window_size` $\leftarrow$ `min_window` ▷ Tamanho da janela resetado
25: **fim se**
26: **fim se**
27: **devolve** `M`
28: **fim função**

até atingir o número máximo de árvores definido como parâmetro. Após definido o número de novas árvores, o modelo é incrementado com as árvores treinadas utilizando os dados da janela W (linha 14).

Em seguida, os dados da janela W são apagados (linha 18) e o tamanho dinâmico da janela é ajustado (linha 19), dobrando de tamanho caso ainda não tenha atingido o tamanho máximo definido. Logo depois, se inicia a rotina de detecção de mudança de conceito. Caso a detecção esteja ativa (linha 20), o algoritmo calcula o erro absoluto do modelo, comparando os resultados da previsão do regressor M utilizando os atributos x com os rótulos y e adiciona o erro ao ADWIN (linhas 21 e 22).

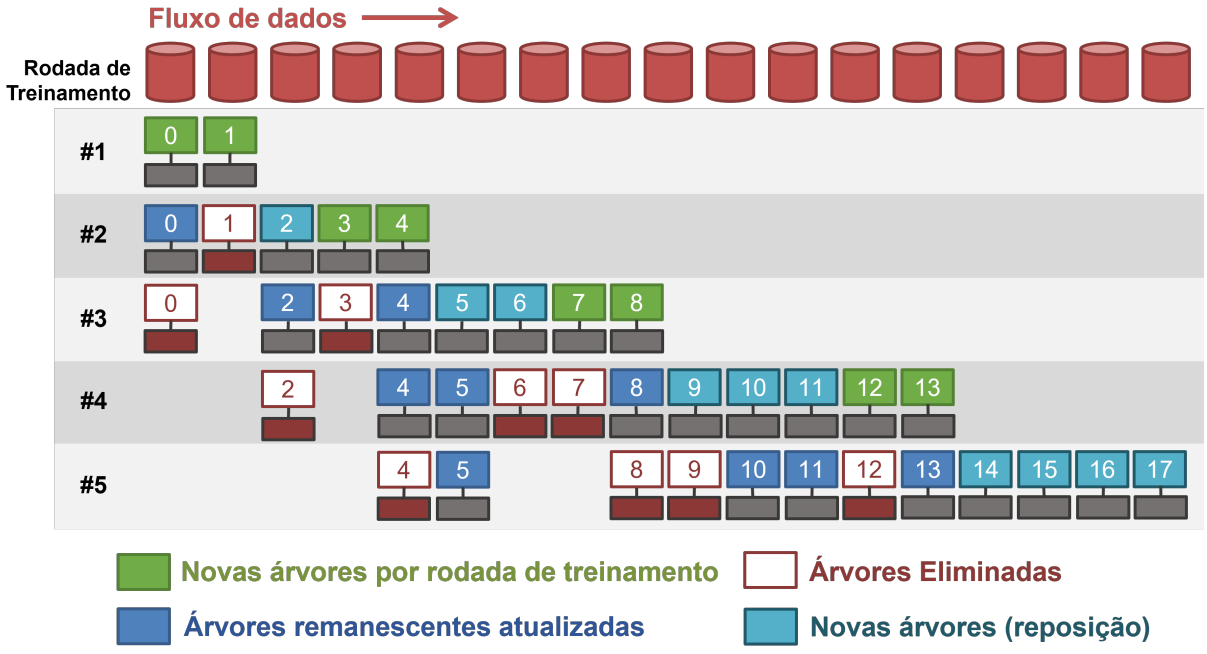
Se o ADWIN detectar uma mudança de conceito, baseada no erro calculado (linha 23), então o tamanho da janela deslizante será resetado ao seu tamanho mínimo definido (linha 24). Por fim, o modelo atualizado é retornado (linha 27) e encerra-se a função de treinamento.

3.1.2 LAX-Reg *Target*

O funcionamento do modelo de treinamento Target é bastante similar ao do modelo FIFO, apenas alterando o método de eliminação de árvores. Enquanto que no FIFO uma porcentagem das árvores mais antigas eram eliminadas, no modelo Target é utilizado um detector de mudança de conceito para cada árvore. Quando o detector sinaliza uma mudança, a árvore é eliminada e é adicionada uma nova árvore ao conjunto XGBoost.

A Figura 7 ilustra o modelo de treinamento Target, com cada ADWIN relacionado a uma árvore do modelo. Nota-se que durante as rodadas de treinamento nem sempre as árvores mais antigas (de menor número) são eliminadas, porém as novas árvores são sempre criadas e anexadas ao final do modelo. Da mesma maneira que no treinamento FIFO, são criadas árvores de reposição e além destas são criadas novas árvores caso o modelo ainda não tenha atingido seu número máximo de árvores do conjunto.

Figura 7 – Exemplo de treinamento do modelo adaptativo de XGBoost - LAX-Reg FIFO.



Fonte: A autora.

O pseudocódigo do algoritmo Target está apresentado no Algoritmo 2. Percebe-se que o algoritmo Target possui menos hiperparâmetros do que o FIFO, pois não é

necessário informar a quantidade de árvores que serão atualizadas e eliminadas a cada rodada de treinamento, já que este valor depende da detecção de mudança de conceito por parte do ADWIN.

Algorithm 2 Lean Adaptive XGBoost Regressor Target

Entrada: $(x, y) \in$ Fluxo de dados

1: **função** TREINAMENTO LAX-REG-T(W = janela de amostras, `window_size` = tamanho da janela, `M` = modelo de regressão XGBoost, `max_trees` = número máximo de árvores, `trees_train` = árvores extras, `min_window` = tamanho mínimo da janela)
2: Adiciona (x, y) na janela (W)
3: **se** $|W| \geq$ `window_size` **então**
4: **se** `M` $\neq$ `NULL` **então**
5: `prev_n_trees` $\leftarrow n\_trees($`M`$)$ ▷ Salva o número de árvores de M
6: **para cada** `tree` em `M` **faça**
7: `error` $\leftarrow abs($`tree`$.predict(x) - y)$ ▷ Calcula o erro absoluto
8: `drift_detectors[index(tree)]`$.add($`error`$)$ ▷ Adiciona erro ao ADWIN correspondente
9: **se** `drift_detectors[index(tree)]`$.drift\_detection =$ `True` **então**
10: $del($`M[tree]`$)$ ▷ Remove a árvore
11: $del($`drift_detectors[index(tree)]`$)$ ▷ Remove o ADWIN correspondente
12: **fim se**
13: **fim para**
14: `M` $\leftarrow$ `M`$.update(W)$ ▷ Atualiza árvores restantes com dados da janela
15: **se** `prev_n_trees` $\geq$ `max_trees` **então**
16: `new_trees` $\leftarrow$ `prev_n_trees` $- n\_trees($`M`$)$ ▷ Repõe árvores eliminadas
17: **senão**
18: `new_trees` $=$ `prev_n_trees` $- n\_trees($`M`$) +$ `trees_train` ▷ Soma árvores novas por treinamento
19: **fim se**
20: `M` $\leftarrow$ `M`$.train(W,$ `new_trees`$)$ ▷ Treina novas árvores com dados da janela
21: `drift_detectors`$.append($`ADWIN * new_trees`$)$ ▷ Anexa ADWINs ao vetor
22: **fim se**
23: `M` $\leftarrow train(W,$ `trees_train`$)$ ▷ Treina novo modelo
24: `drift_detectors` $=$ `drift_detectors[ADWIN * trees_train]` ▷ Cria vetor de ADWINs
25: **fim se**
26: $W \leftarrow 0$ ▷ Limpa janela de dados
27: `window_size` $\leftarrow$ `window_size'` ▷ Ajusta tamanho da janela
28: **devolve** `M`
29: **fim função**

Assim como no modelo FIFO as instâncias de dados são adicionadas à janela deslizante (linha 2) e quando ela se torna cheia (linha 3), o algoritmo verifica se já existe um modelo de regressão XGBoost treinado (Linha 4). Caso não exista, ou seja, o modelo será treinado pela primeira vez, um novo regressor XGBoost é gerado (linha

23) e além deste é criado o vetor de detecção de mudança de conceito, com um ADWIN para cada árvore do conjunto XGBoost (linha 24). Caso já exista um modelo treinado, é armazenado o número de árvores do modelo (linha 5) e para cada árvore do modelo XGBoost é calculado o erro absoluto e o erro é adicionado ao ADWIN correspondente a esta árvore (linhas 6-8).

A eliminação de árvores acontece caso o ADWIN sinalize uma mudança de conceito (linha 9). Neste caso a árvore correspondente ao ADWIN é eliminada do conjunto (linha 10) e o ADWIN também é removido do vetor de detecção de mudança de conceito (linha 11). Esta rotina é executada para todas as árvores do modelo e quando esta é encerrada as árvores remanescentes são atualizadas com os dados da janela W (linha 14), assim como no modelo FIFO.

Após isso faz-se o cálculo da quantidade de árvores novas que serão criadas (linhas 15-19), da mesma forma que no modelo FIFO. Após definido o número de novas árvores, o modelo é incrementado com as árvores treinadas utilizando os dados da janela W (linha 20) e são anexados ao vetor de detecção de mudança de conceito ADWINs, na mesma quantidade que árvores novas criadas.

Em seguida, os dados da janela W são apagados (linha 26) e o tamanho dinâmico da janela é ajustado (linha 27), dobrando de tamanho caso ainda não tenha atingido o tamanho máximo definido. Por fim, o modelo atualizado é retornado (linha 28) e encerra-se a função de treinamento.

3.2 HIPERPARÂMETROS

O XGBoost possui por padrão diversos hiperparâmetros que podem ser configurados para obter uma melhor precisão do modelo de acordo com cada cenário de uso. O algoritmo proposto neste trabalho, LAX-Reg, utiliza alguns dos hiperparâmetros padrão do XGBoost e acrescenta outros próprios, listados e explanados a seguir:

- Próprios do XGBoost:
  - **Taxa de aprendizado (**$learning\_rate$**)**: O XGBoost usa a técnica de *boosting*, que compreende a criação de árvores iterativamente, sendo que a nova árvore tenta minimizar o erro da árvore anterior. Portanto, a taxa de aprendizagem define quão intensa é a correção aplicada. Ou seja, para valores próximos a 0 os ajustes são pequenos e para próximos a 1 são grandes.
  - **Profundidade máxima (**max_depth**)**: Define a profundidade máxima das árvores do modelo, ou seja, o número máximo de nós da raiz as folhas da árvore. Um valor muito alto implica em um modelo complexo e, possivelmente, superespecializado, além de ocupar mais espaço de memória.

- Próprios do LAX-Reg:
  - **Tamanho mínimo da janela (**min_window_size**)** e **Tamanho máximo da janela (**max_window_size**)**: Tamanhos mínimo e máximo do *buffer* que armazena os dados provenientes do fluxo. O tamanho da janela é dinâmico e é atualizado de acordo com esses valores, crescendo exponencialmente do valor mínimo até o valor máximo.
  - **Tamanho do conjunto (**max_trees**)**: Número máximo de árvores do conjunto XGBoost. Uma quantidade muito grande de árvores aumenta a complexidade do modelo, resultando em um maior tempo de treinamento, teste e espaço ocupado de memória.
  - **Árvores criadas por rodada de treinamento (**$trees\_train$**)**: Número de árvores novas que serão criadas a cada rodada de treinamento. Este valor reflete a velocidade com que o conjunto irá crescer até atingir o número máximo de árvores do conjunto.
- Específicos do LAX-Reg-FIFO:
  - **Porcentagem de árvores atualizadas (**$percent\_up\_trees$**)**: Percentual de árvores que serão mantidas no modelo e atualizadas durante a rodada de treinamento. As árvores que não são atualizadas são eliminadas do modelo, portanto, este valor também determina a quantidade de árvores que são eliminadas e, por consequência, que são criadas a cada rodada de treinamento.
  - **Detecção de mudança de conceito (**$detect\_drift$**)**: Valor booleano que define se o método ativo de detecção de mudança de conceito ADWIN será utilizado durante o processo de treinamento.

3.3 IMPLEMENTAÇÃO DA TECNOLOGIA

O desenvolvimento do algoritmo proposto foi realizado utilizando a linguagem de programação Python3. Ela foi escolhida pelos diversos pacotes de aprendizado de máquina existentes e pela facilidade de implementação de algoritmos desse tipo nela. Foram utilizadas as seguintes bibliotecas:

- `xgboost`: técnica de AM usada como base para a implementação do algoritmo proposto. Para o treinamento utilizou-se a função `train`, em que o objetivo foi configurado como `reg:squarederror` e métrica de avaliação `rmse`. Na etapa de atualização das árvores do modelo também utiliza-se a função `train`, porém alterando os parâmetros de processo para `update` e atualizador para `refresh`. As previsões são realizadas utilizando a função de `inplace_predict`.

- `numpy`: pacote de funções auxiliares de tratamento de vetores (criar, deletar, concatenar);
- `scikit-multiflow`: fornece o detector de mudança de conceito ADWIN.

O desenvolvimento do algoritmo proposto foi realizado em etapas e de forma incremental, sendo que em cada etapa foram avaliadas as deficiências do algoritmo e as formas de melhoria, principalmente no que diz respeito à precisão e à velocidade de processamento. Diversos testes foram realizados a cada funcionalidade adicionada para medir o impacto das melhorias propostas e a consequência nas mudanças dos valores dos hiperparâmetros (*tuning*).

Por fim, os experimentos e comparações com os algoritmos de regressão de fluxo de dados da literatura são descritos no Capítulo 4.

4 RESULTADOS PARCIAIS

Este capítulo apresenta a metodologia de avaliação utilizada, bem como os resultados obtidos até o momento. Portanto, a seção 4.1 apresenta a metodologia, que contempla a caracterização da sistemática de simulação, a descrição dos algoritmos utilizados como *benchmarks* de comparação, os conjuntos de dados utilizados nos experimentos e as métricas avaliadas. Após isso, na seção 4.2, são apresentados e discutidos os resultados obtidos pelo algoritmo proposto em comparação com os *benchmarks*. É importante destacar que este é um trabalho em andamento, portanto os resultados aqui apresentados são parciais.

4.1 METODOLOGIA

A sistemática de simulação utiliza o método de avaliação chamada *Evaluate Prequential*, disponível na biblioteca `scikit-multiflow` do Python3 por meio da invocação da função `skmultiflow.evaluation.EvaluatePrequential`. Esse método intercala o teste e posterior treinamento de cada instância de dado recebido no fluxo. Ele é específico para avaliações de fluxo de dados, pois as amostras são analisadas consecutivamente, por ordem de chegada, e são posteriormente descartadas. Neste método, o modelo é avaliado por dados que ainda não foram utilizados no seu treinamento, uma vez que cada dado é usada para testar o modelo, ou seja, fazer a predição, e somente após isso o dado é utilizado para treinamento do modelo. O *Evaluate Prequential* foi configurado para fornecer o MSE, tempo de treinamento, tempo de teste, e a memória alocada pelo modelo.

Foram feitas 15 execuções para cada algoritmo em cada conjunto de dados, devido à característica não-determinística de alguns algoritmos. Utilizou-se uma máquina com as seguintes configurações: Intel® Core™ i7-11700 (8 cores, 16 MB cache, até 4.9 GHz), 32GB (2 de 16 GB) de memória (DDR4, UDIMM, memória non-ECC), armazenamento 512 GB de M.2 Class 40 SSD e sistema operacional Ubuntu 20.04.4 LTS (CLI Server); Linux 5.4.0-113-generic. Foi utilizado paralelismo inerente aos algoritmos, sem desenvolvimentos de programação paralela.

Os algoritmos avaliados e conjuntos de dados utilizados são apresentados nas seções 4.1.1 e 4.1.2, respectivamente.

4.1.1 *Benchmarks*

Foram selecionados cinco algoritmos de regressão de fluxos de dados estabelecidos na literatura para comparação com três versões do LAX-Reg, explanados

abaixo, juntamente com seus hiperparâmetros:

- LAX (*Lean Adaptive XGBoost Regressor FIFO*): primeiro algoritmo proposto neste trabalho (modelo de treinamento FIFO) e que não utiliza detecção ativa de mudança de conceito. Hiperparâmetros: $learning\_rate = 0.3$, $max\_depth = 6$, $min\_window\_size = 100$, $max\_window\_size = 1000$, $max\_trees = 50$, $trees\_train = 5$, $percent\_up\_trees = 0.825$, $detect\_drift = false$.
- LAX-A (*Lean Adaptive XGBoost Regressor FIFO with Drift Detection*): primeiro algoritmo proposto neste trabalho (modelo de treinamento FIFO) e que faz uso do ADWIN para detecção ativa de mudança de conceito baseado no erro total do conjunto XGBoost. Foram utilizados os mesmos hiperparâmetros do LAX-Reg, exceto $detect\_drift = true$.
- LAX-T (*Lean Adaptive XGBoost Regressor Target*): segundo algoritmo proposto neste trabalho e que faz uso do ADWIN para detecção ativa de mudança de conceito, porém aplicado individualmente para cada árvore do conjunto XGBoost. Foram utilizados os mesmos hiperparâmetros do LAX-Reg, exceto os que não fazem parte deste algoritmo. Hiperparâmetros: $learning\_rate = 0.3$, $max\_depth = 6$, $min\_window\_size = 100$, $max\_window\_size = 1000$, $max\_trees = 50$, $trees\_train = 5$.
- AFXGB (*Adaptive Fast XGBoost Regressor*): algoritmo proposto por Souza, Grando e Baldo (2022) que faz substituição de modelos XGBoost e utiliza um vetor de detectores de mudança de conceito com: ADWIN, KSWIN e DDM. Hiperparâmetros: $learning\_rate = 0.3$, $max\_depth = 6$, $min\_window\_size = 0$, $max\_window\_size = 1000$, $life\_time = 25$, $pre\_train = 15$, $reset\_switch = true$.
- ARFReg (*Adaptive Random Forest Regressor*): algoritmo para regressão feito por Gomes et al. (2018), baseado no algoritmo de mesmo nome utilizado para classificação, que utiliza, por padrão, o ADWIN para detecção de mudança de conceito. Hiperparâmetros: $n\_estimators = 50$, restante padrão da biblioteca `scikit-multiflow`.
- AMRules (*Adaptive Model Rules*): algoritmo proposto por Duarte, Gama e Bifet (2016) baseado em regras, em que cada regra possui um detector de mudança de conceito. Hiperparâmetros: $drift\_detector = ADWIN()$, restante padrão da biblioteca `river`.
- HTR (*Hoeffding Tree Regressor*): versão para regressão do algoritmo de árvore incremental *Hoeffding Tree* utilizado para classificação, proposto por Bifet e Gavalda (2009). Hiperparâmetros: padrão da biblioteca `scikit-multiflow`.

- HATR (*Hoeffding Adaptive Tree Regressor*): evolução do HTR, também baseado no trabalho de Bifet e Gavalda (2009), que faz uso da rede neural PERCEPTRON para executar suas previsões e de ADWIN para detectar mudanças de conceito. Hiperparâmetros: padrão da biblioteca `scikit-multiflow`.

Destaca-se que os algoritmos AMRules, HTR e HATR geram modelos de uma árvore só, enquanto que o restante faz uso de conjuntos de árvores. O total de árvores dos modelos LAX e ARFReg é de 50 e do AFXGB é de 40 árvores.

4.1.2 Conjuntos de dados

Foram utilizados 9 *datasets*, sendo 5 reais e 4 sintéticos. Os conjuntos de dados sintéticos foram gerados a partir das funções `make_friedman1`, `RegressionGenerator` e `ConceptDriftStream` das bibliotecas `scikit-multiflow` e `scikit-learn`, implementadas por Montiel et al. (2018). Com a geração de dados sintéticos é possível criar conjuntos com grande volume de dados, assim como representar diferentes tipos de mudança de conceito, propiciando uma avaliação mais controlada dos modelos em diferentes cenários.

A Tabela 3 apresenta as características dos *datasets* avaliados, separados por reais e sintéticos. Os conjuntos sintéticos foram gerados com dois tipos de mudança de conceito, abrupta (A) e gradual (G), em que as mudanças ocorrem a cada 50.000 amostras. Para os casos de mudança gradual, o conceito é alterado ao longo de 10.000 amostras. A coluna "Unidade" indica a unidade de medida de tempo de geração de cada instância do *dataset* e na coluna "# Conceitos" constam a quantidade de conceitos presentes nos *datasets*.

Tabela 3 – Características dos conjuntos de dados. Mudanças de conceito: Abrupta (A) e Gradual (G).

Dataset	**# Amostras**	**# Atributos**	**# Conceitos**	**Unidade**	**Fonte**
metro	48.204	11	1	Dia	Dua e Graff (2017)
waze	307.800	6	1	Hora	Cardoso (2018)
bikes	17.379	17	2	Hora	Fanaee-T e Gama (2013)
barra	74.305	42	3	10 min	Braz et al. (2022)
costa	74.305	42	2	10 min	Braz et al. (2022)
fried1 (A)	500.000	10	10	Segundo	A autora
fried1 (G)	500.000	10	10	Segundo	A autora
reg (A)	500.000	10	10	Segundo	A autora
reg (G)	500.000	10	10	Segundo	A autora

Fonte: A autora.

Como o objetivo é avaliar o erro dos modelos ao longo do tempo, as amostras dos *datasets* foram agrupadas em blocos que formam uma unidade de tempo maior do que de sua geração. Estes blocos são utilizados para o treinamento e predições dos modelos, onde as predições ocorrem a cada bloco único, porém o treinamento é

feito com vários blocos para fornecer mais dados aos modelos. A Tabela 4 apresenta a granularidade das unidades de tempo utilizadas na avaliação dos modelos (campo “Unidade Predição”), a quantidade de amostras no bloco (campo “# Amostras Bloco”), a quantidade de blocos utilizados no treinamento (campo “# Blocos Treinamento”) e a quantidade total de amostras utilizadas no treinamento dos modelos (campo “# Amostras Treinamento”). A quantidade total de amostras utilizadas no treinamento é representada pela multiplicação entre a quantidade de blocos de treinamento e a quantidade de amostras no bloco.

Tabela 4 – Unidades de tempo avaliadas e quantidade de amostras por bloco dos conjuntos de dados.

Dataset	**Unidade**	**Unidade Predição**	**# Amostras Bloco**	**# Blocos Treinamento**	**# Amostras Treinamento**
metro	Dia	Semana	180	36	6.480
waze	Hora	Dia	2.400	30	50.400
bikes	Hora	Dia	24	90	2.160
barra	10 min	Dia	144	90	12.960
costa	10 min	Dia	144	90	12.960
fried1 (A)	Segundo	Hora	3.600	24	86.400
fried1 (G)	Segundo	Hora	3.600	24	86.400
reg (A)	Segundo	Hora	3.600	24	86.400
reg (G)	Segundo	Hora	3.600	24	86.400

Fonte: A autora.

4.2 RESULTADOS

Nesta seção são apresentados e discutidos os resultados obtidos durante os testes realizados, confrontando o desempenho do algoritmo proposto neste trabalho com os demais apresentados na seção 4.1.1. A avaliação se inicia com a comparação dos valores de MSE, após isso são comparados os tempos de execução e, por fim, da memória alocada.

4.2.1 Erro Médio Quadrático (MSE)

A Tabela 5 apresenta o MSE médio das 15 execuções dos algoritmos avaliados para cada conjunto de dados. Como o MSE representa o erro médio quadrático, valores menores de MSE indicam uma precisão maior do algoritmo. O menor MSE é destacado em negrito e o ranking correspondente do modelo para esse conjunto de dados fica ao lado do MSE. O ranking dos algoritmos é definido ordenando o MSE de todos eles para cada conjunto de dados do menor para o maior, o que significa que quanto menor for o ranking do modelo, menor é o seu MSE. Ao final, a tabela mostra o ranking médio dos algoritmos entre todos os conjuntos de dados, bem como o desvio padrão do ranking médio.

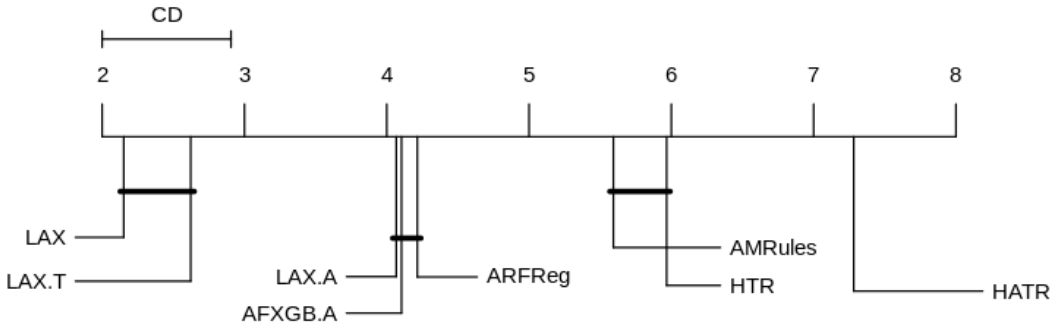
Tabela 5 – MSE e ranking médio das 15 execuções dos algoritmos para cada conjunto de dados.

Dataset	LAX	LAX-T	LAX-A	AFXGB	ARFReg	AMRules	HTR	HATR
metro	1738735 (2.1)	**1536842** (1)	2885753 (4.3)	2009869 (3.2)	17394287 (5.2)	3932086 (5.4)	146820404 (7.6)	110866329 (7.2)
waze	2.81 (2.3)	2.89 (3.3)	**2.79** (1.2)	3.35 (6.7)	3.16 (5.7)	2.94 (4.5)	3.8 (4.9)	4.42 (7.5)
bikes	589 (2)	732 (3)	2683 (5)	**360** (1)	1979 (4)	11766 (8)	4649 (6.4)	5035 (6.6)
barra	351 (3)	355 (4)	439 (6.1)	594 (7.1)	340 (2)	**133** (1)	361 (5)	3664 (7.9)
costa	163 (2)	172 (4)	207 (6.1)	316 (7.3)	166 (3)	**49** (1)	173 (5)	680 (7.7)
fried1 (A)	78 (3)	**33** (1)	175 (5)	49 (2)	131.4 (4)	1822 (7)	303 (6)	5480 (8)
fried1 (G)	82 (3)	**36** (1)	135 (5)	57 (2)	131 (4)	2018 (7.7)	414 (6.1)	1616 (7.3)
reg (A)	**3987** (1)	5022 (3.3)	4728 (2)	5208 (3.7)	8421 (5)	37215 (8)	11658 (6)	14408 (7)
reg (G)	**4173** (1)	5194 (3)	4906 (2)	5520 (4)	8977 (5)	42264 (7.8)	31123 (6.7)	25288 (6.5)
Rank	**2.2** ± 0.8	2.6 ± 1.2	4.07 ± 1.8	4.1 ± 2.3	4.2 ± 1.2	5.6 ± 2.7	6 ± 1.3	7.3 ± 0.8

Fonte: A autora.

Para realizar a comparação da precisão obtida entre os algoritmos é necessária a utilização de testes de hipóteses, pois somente a observação do MSE não é suficiente para inferir conclusões sobre os resultados (CALVO; RODRIGO, 2016). Primeiro, aplicou-se o teste de Friedman com significância de 95%, resultando em um valor p de 2.2×10^{-16}, o que rejeitou a hipótese nula. Isso significa que foram encontradas diferenças significativas e, portanto, é possível aplicar um teste par a par *post-hoc* para comparações múltiplas entre os resultados alcançados pelos algoritmos. O teste *post-hoc* aplicado foi o de Nemenyi, usando o ranking médio dos algoritmos. Este teste determina a Diferença Crítica (CD), o que significa que dois algoritmos cuja diferença entre os rankings seja maior que o CD são considerados significativamente diferentes. O Nemenyi resultou em um CD de 0.9 e as comparações pareadas são mostradas na Figura 8.

Figura 8 – Teste de Nemenyi utilizado o ranking médio dos modelos.



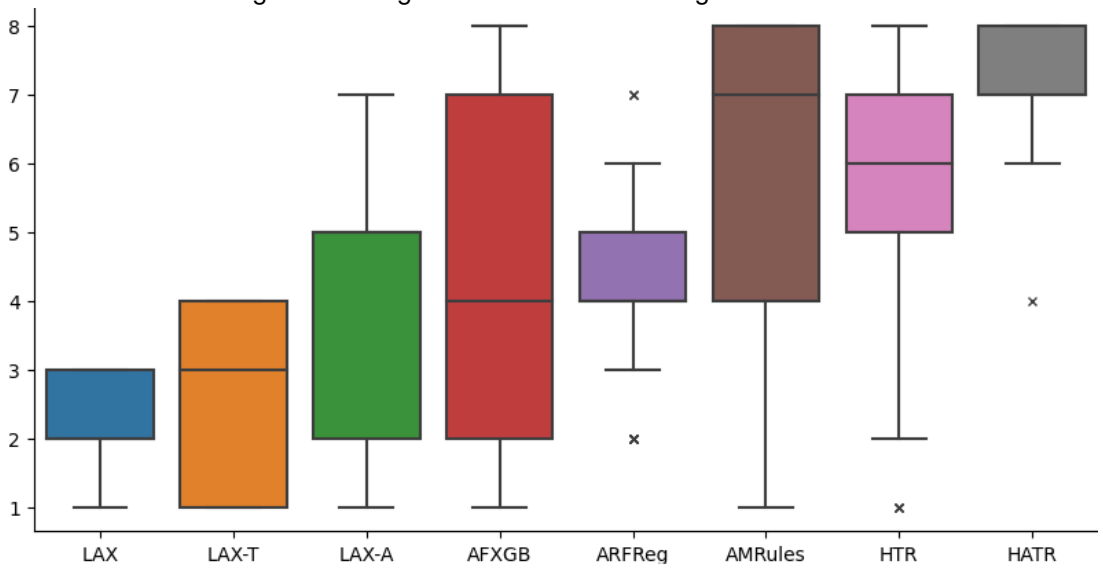
Fonte: A autora.

A partir desta análise, é possível observar que o LAX e LAX-T, ambos propostos neste trabalho, obtiveram os melhores rankings em comparação com os demais algoritmos e demonstram desempenho equivalente entre si. O grupo com o segundo melhor desempenho é composto pelo LAX-A, AFXGB e ARFReg, que tiveram uma média de rankings muito próxima, porém ao analisar seu desvio padrão ilustrado na Figura 9 percebe-se uma consistência maior do desempenho do ARFReg, já que o desvio padrão deste é menor. Destaca-se que o uso do ADWIN para reset da janela no LAX estratégia FIFO não favoreceu o deu desempenho, já que o LAX-A apresenta

uma diferença considerável de ranking, aproximadamente 2 pontos, em comparação com o LAX.

Já entre os de pior desempenho estão o HTR e AMRules, e por fim o HATR. Nota-se na Figura 9 que o AMRules possui uma variação de desempenho muito grande, pois ele obteve o melhor ranking para os *datasets* barra e costa, porém não para os demais *datasets*.

Figura 9 – Diagrama de caixa do ranking dos modelos.



Fonte: A autora.

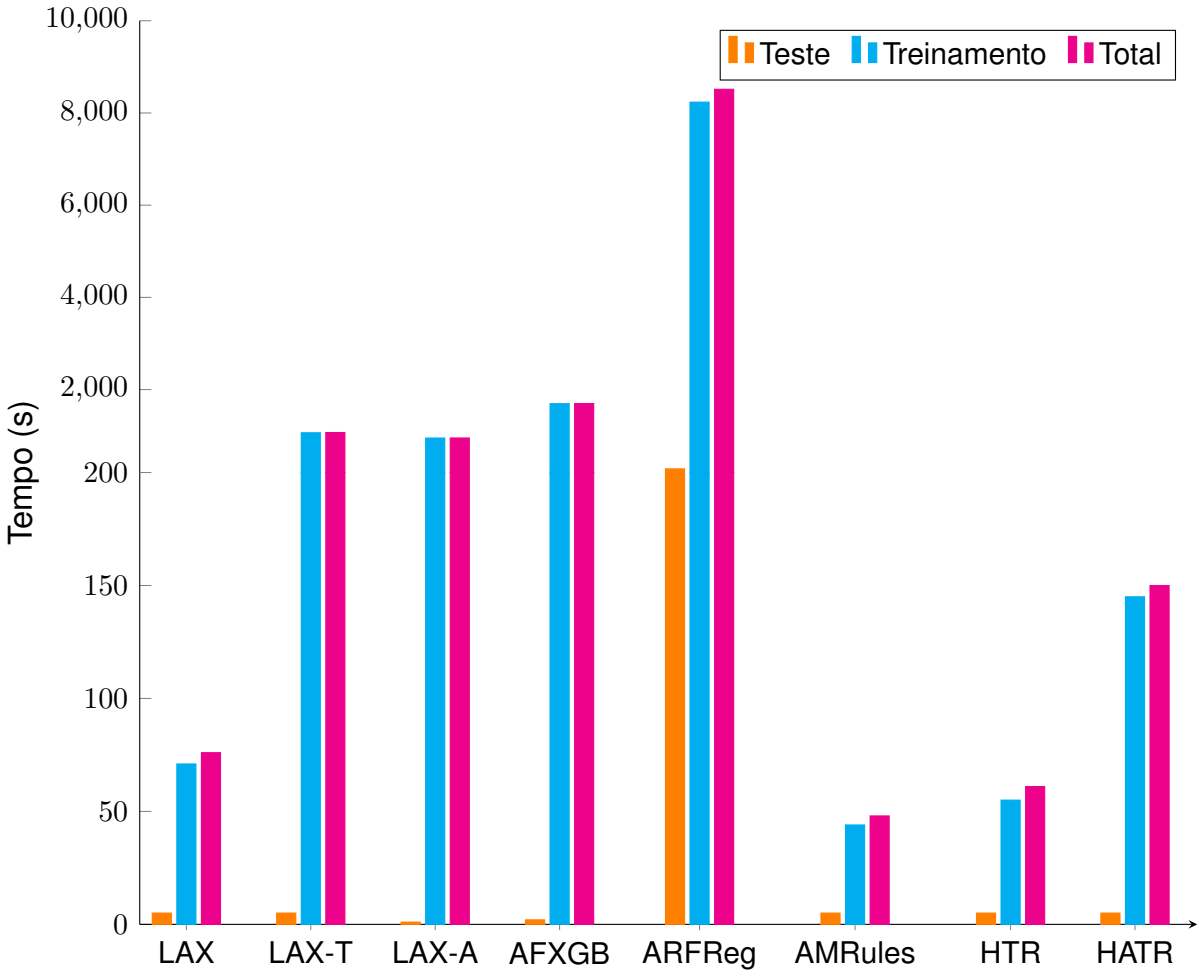
A partir desta análise conclui-se que o LAX com a estratégia de treinamento FIFO sem uso do ADWIN e com a estratégia de treinamento *Target* obtiveram menores erros em comparação aos demais algoritmos avaliados, comprovando seu desempenho superior.

4.2.2 Tempos de execução

Foram registrados os tempos de teste e de treinamento de cada algoritmo, assim como o tempo total que é o resultado da soma deles. A Figura 10 apresenta os tempos médios entre todas as simulações, em segundos, dos algoritmos avaliados.

Observa-se que entre os modelos que apresentaram menor MSE, o LAX que utiliza estratégia FIFO sem detecção de mudança de conceito possui os menores tempo de treinamento e tempo total, com as médias de 71 e 76 segundos, respectivamente. Por outro lado, o LAX-A e LAX-T apresentaram um tempo maior de treinamento, pois é durante esta etapa que se faz uso do detector de mudança de conceito ADWIN, comportamento que também é percebido no HATR e AFXGB, que também fazem uso do mesmo detector de mudança de conceito.

Figura 10 – Tempos médios de teste, treinamento e total de execução dos algoritmos avaliados.



Fonte: A autora.

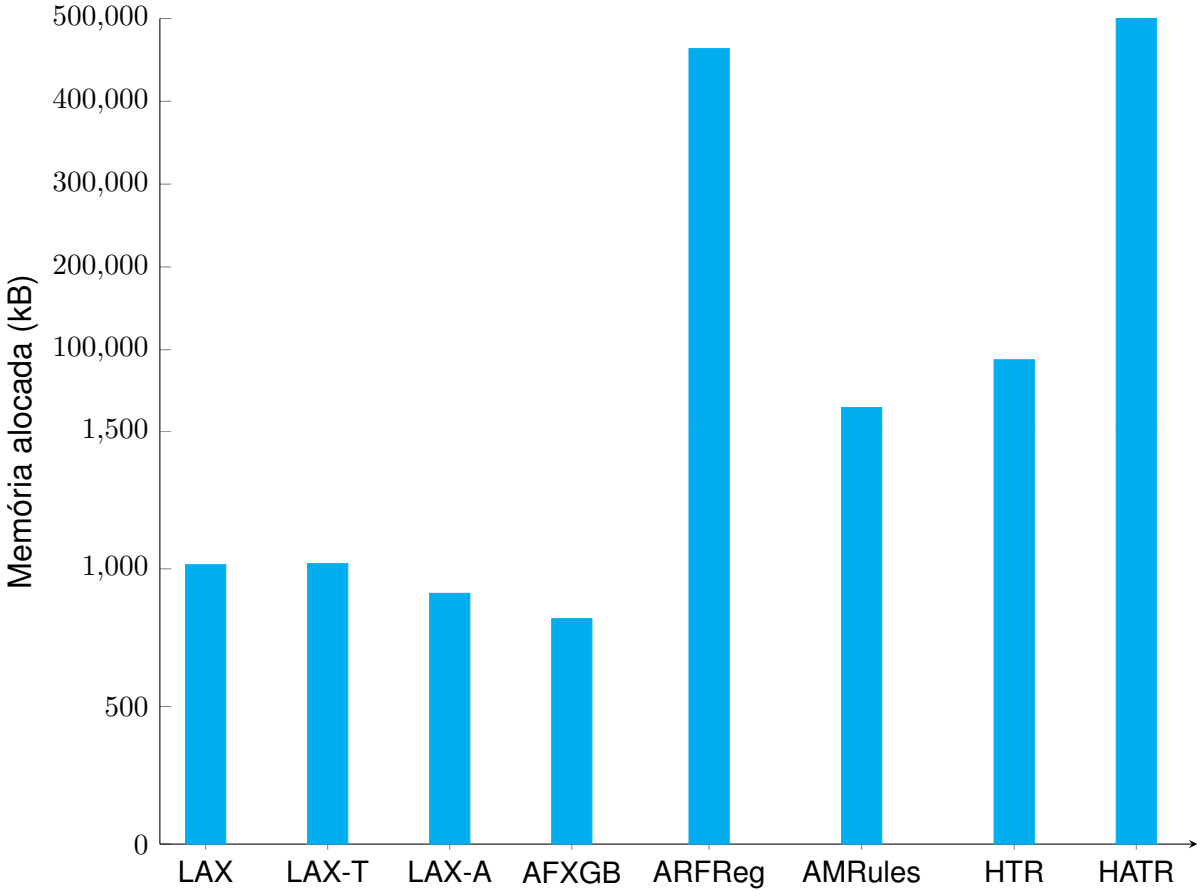
Com relação aos tempos de teste, a maioria dos modelos obteve o tempo médio de 5 segundos, exceto o LAX-A, AFXGB e ARFReg, sendo que os dois primeiros apresentaram um tempo menor, de 1 e 2 segundos respectivamente. Este tempo de teste reduzido do LAX-A e AFXGB é explicado devido ao reset da janela deslizante, que faz com que menos instâncias sejam utilizadas para a criação das árvores destes modelos, o que por consequência as torna menores e mais rápidas no momento de realizar a predição. Já o ARFReg apresentou um tempo de teste de 280 segundos, consideravelmente maior que todos os outros modelos.

4.2.3 Memória alocada

A memória alocada foi registrada medindo o tamanho do modelo em quilobytes (kB) a cada predição e foi identificado seu valor máximo para cada simulação. A Figura 11 apresenta um gráfico da média da memória máxima alocada entre todas as simulações para cada algoritmo avaliado. Os modelos propostos neste trabalho, LAX, LAX-T e LAX-A alocam em média 1015 kB, 1019 kB e 911 kB, respectivamente. Já o

AFXGB aloca em média 819 kB e o ARFReg atinge uma média de 0,4 GB.

Figura 11 – Memória alocada dos modelos gerados pelos algoritmos avaliados.



Fonte: A autora.

Nota-se que assim como no caso do tempo de teste, o LAX-A e AFXGB também apresentam uma menor memória alocada, pelo mesmo motivo de suas árvores serem menores, já que recebem menos dados para treinamento nos momentos de reset da janela. Além disso, percebe-se que o detector de mudança de conceito não apresenta grandes impactos na memória alocada, pois mesmo no caso do LAX-T em que é utilizado um ADWIN para cada árvore, a memória deste foi apenas 4 kB maior que no caso do LAX que não utiliza nenhum tipo de detector.

Com relação aos demais modelos, observa-se que o AMRules, HTR e HATR, mesmo sendo modelos de árvore única, ocupam mais memória do que os modelos baseados em XGBoost. Com isso, observa-se que o XGBoost é uma biblioteca eficiente não só no tempo de execução, mas também com relação ao consumo de memória.

4.2.4 Considerações sobre os Resultados

O algoritmo LAX-Reg, proposto neste trabalho, se mostrou eficiente para a tarefa de regressão de fluxos de dados em todas as suas versões apresentadas. O LAX-Reg na estratégia FIFO sem uso do ADWIN e na estratégia *Target* tiveram um

desempenho em termos de MSE que supera o ARFReg, um dos algoritmos mais consolidados da literatura, enquanto que sua velocidade de treinamento e testes se mostrou superior. Isto ocorre pois o LAX-Reg foi construído utilizando como base o XGBoost, que é um algoritmo reconhecido por sua rapidez de treinamento e predição em comparação com outros algoritmos de AM. Já o ARFReg foi construído sobre o *Random Forest*, que é um algoritmo que apresenta um processo de treinamento mais demorado devido a sua abordagem de construção do conjunto de modelos. Estas mesmas estratégias do LAX-Reg também superaram o desempenho do AFXGB-Reg, modelo base deste trabalho, apresentando menores erros e menor tempo de execução.

Além disso, também foi observado que todas as versões do LAX-Reg apresentam um baixo consumo de memória no processo de construção do conjunto de modelos, alocando no máximo 1019 kB, enquanto que a maioria dos outros algoritmos utilizados na comparação apresentam grandezas na ordem dos megabytes ou gigabytes.

Portanto, como pode ser observado nas análises realizadas sobre os testes executados, o algoritmo LAX-Reg proposto apresenta resultados competitivos em relação aos algoritmos encontrados no estado da arte da literatura sobre regressão de séries temporais.

5 CONCLUSÕES

Esse trabalho teve como objetivo a proposta de um algoritmo de regressão de fluxo de dados não estacionários utilizando a biblioteca XGBoost. No cenário de fluxos de dados, são encontrados diversos trabalhos relacionados ao objetivo de classificação, porém são escassos os trabalhos de regressão. Entre os algoritmos de regressão de fluxo de dados, os conjuntos de modelos se mostram habilidosos em termos de precisão das suas previsões, com destaque para o uso de árvores de decisão no formato de *bagging* e *Random Forest*. Porém, nota-se a carência de trabalhos que explorem a técnica de *boosting*.

Um dos algoritmos mais relevantes que utiliza a técnica de *boosting*, por meio da aplicação do *Gradient Boosting*, é o XGBoost. Ele tem desempenho compatível ou superior em termos de precisão e velocidade em relação as *benchmarks* da literatura comprovado em diversos trabalhos para o aprendizado de dados estáticos. Este algoritmo também teve sua eficiência aferida para regressão de fluxo de dados em Souza, Grando e Baldo (2022), porém, foram encontradas oportunidades de melhoria na estratégia de treinamento de modelos alternados utilizada por Souza, Grando e Baldo (2022). Neste trabalho é proposto o algoritmo LAX-Reg, que utiliza um modelo único que é atualizado ao longo do tempo, solucionando o problema da perca de precisão na substituição de modelos. Para a remoção de árvores durante o treinamento do modelo foram propostas duas estratégias, o LAX-Reg-FIFO e o Lax-Reg-Target.

Foram realizados experimentos para comparar as versões do LAX-Reg com outros algoritmos da literatura, simulando seu uso em *datasets* reais e sintéticos, que apresentavam diferentes tipos de mudança de conceito. As métricas avaliadas foram o MSE, tempo de treinamento, tempo de teste, tempo total de execução e memória alocada. Os resultados indicam que as versões do LAX-Reg com estratégia FIFO e sem uso do ADWIN e com a estratégia *Target* conseguem atingir os menores erros entre todos os algoritmos, demonstrando desempenho superior. Comparando o tempo total de execução e a memória alocada desses algoritmos com um dos algoritmos mais consolidados da literatura, o ARFReg, nota-se que a execução do LAX-Reg-FIFO sem ADWIN é 112 vezes mais rápida e o consumo de memória é 456 vezes menor. Sendo assim, valida-se o LAX-Reg como um algoritmo preciso, rápido e leve.

Em comparação com o algoritmo AFXGB-Reg de Souza, Grando e Baldo (2022), que serviu como base para este trabalho, observa-se ganhos tanto em termos de MSE quanto em tempo de execução. Os ganhos em tempo são justificados pelo uso de um único modelo, já que não são mais treinados dois modelos em paralelo,

fazendo com que o LAX-Reg seja mais enxuto. Com relação à precisão, nota-se que principalmente nos *datasets* reais o AFXGB-Reg apresenta os maiores erros, estes que não haviam sido avaliados em seu trabalho original.

Analisando as variações do LAX-Reg, percebe-se que o modelo LAX-Reg-A, que faz uso da estratégia FIFO e utiliza o detector de mudança de conceito ADWIN para reset da janela, obteve precisão menor do que o modelo FIFO que não faz uso deste. Na lógica do algoritmo proposto, o uso do ADWIN é feito utilizando o erro da resposta do conjunto completo de árvores para identificar a mudança de conceito, e quando sinaliza uma mudança é feito o *reset* do tamanho da janela de treinamento. Com isso, aumenta-se a frequência de exclusão, atualização e criação de árvores novas, possivelmente prejudicando o modelo, pois muitas árvores são excluídas em um curto período. Já na estratégia *Target* o ADWIN é aplicado para cada árvore individualmente, o que se mostrou efetivo em termos de diminuição do erro, porém esta estratégia ainda não foi capaz de obter menores erros que a estratégia FIFO sem ADWIN e além disso apresentou maiores custos no tempo de treinamento, já que faz diversas predições para cálculo de erro e trabalha com um vetor de ADWINs durante esta fase.

5.1 TRABALHOS PUBLICADOS

Durante o desenvolvimento da dissertação foram publicados dois trabalhos:

- Adaptive fast XGBoost for Binary classification (BALDO et al., 2022) no XXXVII Simpósio Brasileiro de Bancos de Dados
- Adaptive fast XGBoost for Regression no Intelligent Systems: 11th Brazilian Conference (SOUZA; GRANDO; BALDO, 2022)

5.2 TRABALHOS FUTUROS

Foram observadas algumas oportunidades de melhoria do algoritmo proposto. Dentre elas, foi percebido que as versões do LAX-Reg que utilizam o ADWIN para detectar mudanças de conceitos obtiveram um maior erro. Entretanto, essa não parece ser a situação esperada na utilização dessa abordagem, pois com a adição da detecção ativa era esperado que o algoritmo pudesse se adaptar ainda melhor às mudanças de conceito. Portanto, como um dos trabalhos futuros, pode-se explorar a utilização de abordagens diferentes do ADWIN na detecção de mudanças de conceito, bem como outros detectores, como o Page-Hinckley e DDM.

Além disso, pode-se realizar estudos mais extensivos e profundos sobre a aplicação do algoritmo LAX-Reg, e consecutiva comparação com os algoritmos da litera-

tura, em fluxos com mudança de conceito recorrente. Esse tipo particular de mudança de conceito é encontrada em fluxos que apresentam sazonalidade na ocorrência de mudanças na estatísticas de distribuição dos valores de seus atributos.

Por fim, pode-se realizar avaliações mais extensivas sobre o impacto dos atributos temporais e de sua granularidade nas predições em *datasets* reais e sintéticos. Essa avaliação com um número maior de *datasets* pode confirmar a relevância ou não da utilização desse tipo de atributo no desempenho dos modelos de predição gerados.

REFERÊNCIAS

BALDO, F. et al. Adaptive fast XGBoost for binary classification. In: SBC. **Anais do XXXVII Simpósio Brasileiro de Bancos de Dados**. [S.l.], 2022. p. 13–25.

BANFIELD, R. E. et al. A comparison of decision tree ensemble creation techniques. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 29, n. 1, p. 173–180, 2007.

BARDDAL, J. P. Vertical and horizontal partitioning in data stream regression ensembles. In: IEEE. **2019 International Joint Conference on Neural Networks (IJCNN)**. Curitiba, BR, 2019. p. 1–8.

BASSEVILLE, M.; NIKIFOROV, I. V. et al. **Detection of abrupt changes: theory and application**. France: prentice Hall Englewood Cliffs, 1993. v. 104.

BIFET, A. Classifier concept drift detection and the illusion of progress. In: SPRINGER. **International conference on artificial intelligence and soft computing**. [S.l.], 2017. p. 715–725.

BIFET, A.; GAVALDA, R. Learning from time-changing data with adaptive windowing. In: SIAM. **Proceedings of the 2007 SIAM international conference on data mining**. Catalunha, Espanha, 2007. p. 443–448.

BIFET, A.; GAVALDA, R. Adaptive learning from evolving data streams. In: SPRINGER. **International Symposium on Intelligent Data Analysis**. [S.l.], 2009. p. 249–260.

BOSE, R. J. C. et al. Dealing with concept drifts in process mining. **IEEE transactions on neural networks and learning systems**, IEEE, v. 25, n. 1, p. 154–171, 2013.

BOTCHKAREV, A. A new typology design of performance metrics to measure errors in machine learning regression algorithms. **Interdisciplinary Journal of Information, Knowledge, and Management**, Informing Science Institute, v. 14, p. 045–076, 2019. Disponível em: <https://doi.org/10.28945%2F4184>.

BRAZ, F. J. et al. Road traffic forecast based on meteorological information through deep learning methods. **Sensors**, v. 22, n. 12, 2022. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/22/12/4485>.

BREIMAN, L. Bagging predictors. **Machine learning**, Springer, v. 24, n. 2, p. 123–140, 1996.

BREIMAN, L. Random forests. **Machine learning**, Springer, v. 45, n. 1, p. 5–32, 2001.

CALVO, B.; RODRIGO, G. S. scmamp: Statistical comparison of multiple algorithms in multiple problems. **The R Journal, Vol. 8/1, Aug. 2016**, The R Foundation, 2016.

CARDOSO, J. de O. **Coletânea de Trabalhos de Conclusão de Curso - Graduação em Ciência da Computação**. 2018. Trabalhos de Conclusão de Curso (Graduação)-Universidade do Estado de Santa Catarina, Centro de Ciências Tecnológicas, Curso de Ciência da Computação, Joinville. Disponível em: <http://sistemabu.udesc.br/pergamumweb/vinculos/000060/000060eb.pdf>.

CHEN, T.; GUESTRIN, C. Xgboost: A scalable tree boosting system. In: **Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. New York, NY, USA: Association for Computing Machinery, 2016. (KDD '16), p. 785–794. ISBN 9781450342322. Disponível em: <https://doi.org/10.1145/2939672.2939785>.

CHEN, T. et al. Xgboost: extreme gradient boosting. **R package version 0.4-2**, v. 1, n. 4, p. 1–4, 2015.

CHOU, P. A. Optimal partitioning for classification and regression trees. **IEEE Transactions on Pattern Analysis & Machine Intelligence**, IEEE Computer Society, v. 13, n. 04, p. 340–354, 1991.

DENG, H. et al. Ensemble learning for the early prediction of neonatal jaundice with genetic features. **BMC Medical Informatics and Decision Making**, v. 21, n. 1, p. 338, Dec 2021. ISSN 1472-6947. Disponível em: <https://doi.org/10.1186/s12911-021-01701-9>.

DHALIWAL, S. S.; NAHID, A.-A.; ABBAS, R. Effective intrusion detection system using xgboost. **Information**, MDPI, v. 9, n. 7, p. 149, 2018.

DIETTERICH, T. G. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. **Machine Learning**, v. 40, n. 2, p. 139–157, Aug 2000. ISSN 1573-0565. Disponível em: <https://doi.org/10.1023/A:1007607513941>.

DUA, D.; GRAFF, C. **UCI Machine Learning Repository**. 2017. Disponível em: <http://archive.ics.uci.edu/ml>.

DUARTE, J. a.; GAMA, J. a.; BIFET, A. Adaptive model rules from high-speed data streams. **ACM Trans. Knowl. Discov. Data**, Association for Computing Machinery, New York, NY, USA, v. 10, n. 3, jan 2016. ISSN 1556-4681. Disponível em: <https://doi.org/10.1145/2829955>.

DUFFY, N.; HELMBOLD, D. Boosting methods for regression. **Machine Learning**, v. 47, n. 2, p. 153–200, May 2002. ISSN 1573-0565. Disponível em: <https://doi.org/10.1023/A:1013685603443>.

FANAEE-T, H.; GAMA, J. Event labeling combining ensemble detectors and background knowledge. **Progress in Artificial Intelligence**, Springer Berlin Heidelberg, p. 1–15, 2013. ISSN 2192-6352. Disponível em: <[WebLink]>.

FRIEDMAN, J. H. Greedy function approximation: a gradient boosting machine. **Annals of statistics**, JSTOR, p. 1189–1232, 2001.

FÜRNKRANZ, J.; KLIEGR, T. A brief overview of rule learning. In: BASSILIADES, N. et al. (Ed.). **Rule Technologies: Foundations, Tools, and Applications**. Cham: Springer International Publishing, 2015. p. 54–69. ISBN 978-3-319-21542-6.

GAMAGE, S.; PREMARATNE, U. Detecting and adapting to concept drift in continually evolving stochastic processes. In: **Proceedings of the International Conference on Big Data and Internet of Thing**. New York, NY, USA: Association for Computing

Machinery, 2017. (BDIOT2017), p. 109–114. ISBN 9781450354301. Disponível em: <https://doi.org/10.1145/3175684.3175723>.

GOMES, H. M. et al. Adaptive random forests for data stream regression. In: **ESANN**. [S.l.: s.n.], 2018.

GOMES, H. M. et al. On ensemble techniques for data stream regression. In: IEEE. **2020 International Joint Conference on Neural Networks (IJCNN)**. [S.l.], 2020. p. 1–8.

IKONOMOVSKA, E.; GAMA, J.; DŽEROSKI, S. Learning model trees from evolving data streams. **Data Mining and Knowledge Discovery**, v. 23, n. 1, p. 128–168, Jul 2011. ISSN 1573-756X. Disponível em: <https://doi.org/10.1007/s10618-010-0201-y>.

IKONOMOVSKA, E.; GAMA, J.; DŽEROSKI, S. Online tree-based ensembles and option trees for regression on evolving data streams. **Neurocomputing**, Elsevier, v. 150, p. 458–470, 2015.

IKONOMOVSKA, E.; GAMA, J.; DžEROSKI, S. Online tree-based ensembles and option trees for regression on evolving data streams. **Neurocomputing**, v. 150, p. 458–470, 2015. ISSN 0925-2312. Special Issue on Information Processing and Machine Learning for Applications of Engineering Solving Complex Machine Learning Problems with Ensemble Methods Visual Analytics using Multidimensional Projections. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0925231214012338>.

IKONOMOVSKA, E. et al. Speeding up hoeffding-based regression trees with options. In: **Proceedings of the 28th International Conference on International Conference on Machine Learning**. Madison, WI, USA: Omnipress, 2011. (ICML'11), p. 537–544. ISBN 9781450306195.

KINGSFORD, C.; SALZBERG, S. L. What are decision trees? **Nature Biotechnology**, v. 26, n. 9, p. 1011–1013, Sep 2008. ISSN 1546-1696. Disponível em: <https://doi.org/10.1038/nbt0908-1011>.

KOSTOPOULOS, G. et al. Semi-supervised regression: A recent review. **Journal of Intelligent & Fuzzy Systems**, IOS Press, v. 35, n. 2, p. 1483–1500, 2018.

KOTSIANTIS, S. B. Decision trees: a recent overview. **Artificial Intelligence Review**, v. 39, n. 4, p. 261–283, Apr 2013. ISSN 1573-7462. Disponível em: <https://doi.org/10.1007/s10462-011-9272-4>.

KRAWCZYK, B. et al. Ensemble learning for data stream analysis: A survey. **Information Fusion**, Elsevier, v. 37, p. 132–156, 2017.

KREMPL, G. et al. Open challenges for data stream mining research. **SIGKDD Explor. Newsl.**, Association for Computing Machinery, New York, NY, USA, v. 16, n. 1, p. 1–10, sep 2014. ISSN 1931-0145. Disponível em: <https://doi.org/10.1145/2674026.2674028>.

LIAO, Z.; WANG, Y. Rival learner algorithm with drift adaptation for online data stream regression. In: **Proceedings of the 2018 International Conference on Algorithms, Computing and Artificial Intelligence**. New York, NY, USA: Association for Computing Machinery, 2018. (ACAI 2018). ISBN 9781450366250. Disponível em: <https://doi.org/10.1145/3302425.3302475>.

LOH, W.-Y. Classification and regression trees. **Wiley interdisciplinary reviews: data mining and knowledge discovery**, Wiley Online Library, v. 1, n. 1, p. 14–23, 2011.

Lu, J. et al. Learning under concept drift: A review. **IEEE Transactions on Knowledge and Data Engineering**, v. 31, n. 12, p. 2346–2363, 2018.

LU, J. et al. Learning under concept drift: A review. **IEEE Transactions on Knowledge and Data Engineering**, IEEE, v. 31, n. 12, p. 2346–2363, 2018.

MAHDI, O. A. et al. Fast reaction to sudden concept drift in the absence of class labels. **Applied Sciences**, Multidisciplinary Digital Publishing Institute, v. 10, n. 2, p. 606, 2020.

MAHESH, B. Machine learning algorithms-a review. **International Journal of Science and Research (IJSR).[Internet]**, v. 9, p. 381–386, 2020.

MAULUD, D.; ABDULAZEEZ, A. M. A review on linear regression comprehensive in machine learning. **Journal of Applied Science and Technology Trends**, v. 1, n. 4, p. 140–147, 2020.

MONTIEL, J. et al. Adaptive XGBoost for evolving data streams. In: **2020 International Joint Conference on Neural Networks (IJCNN)**. [S.l.: s.n.], 2020. p. 1–8.

MONTIEL, J. et al. Scikit-multiflow: A multi-output streaming framework. **Journal of Machine Learning Research**, v. 19, n. 72, p. 1–5, 2018. Disponível em: <http://jmlr.org/papers/v19/18-251.html>.

MUHAMMAD, I.; YAN, Z. Supervised machine learning approaches: A survey. **ICTACT Journal on Soft Computing**, v. 5, n. 3, 2015.

OGUNLEYE, A.; WANG, Q.-G. Xgboost model for chronic kidney disease diagnosis. **IEEE/ACM transactions on computational biology and bioinformatics**, IEEE, v. 17, n. 6, p. 2131–2140, 2019.

PINAGÉ, F. A. et al. Handling concept drift based on data similarity and dynamic classifier selection. Universidade Federal do Amazonas, 2017.

QIU, Y. et al. Performance evaluation of hybrid woa-xgboost, gwo-xgboost and bo-xgboost models to predict blast-induced ground vibration. **Engineering with Computers**, Springer, p. 1–18, 2021.

SAGIROGLU, S.; SINANC, D. Big data: A review. In: **2013 International Conference on Collaboration Technologies and Systems (CTS)**. [S.l.: s.n.], 2013. p. 42–47.

SAMUEL, A. L. Some studies in machine learning using the game of checkers. **IBM Journal of Research and Development**, v. 3, n. 3, p. 210–229, 1959.

SCHAPIRE, R. E. The strength of weak learnability. **Machine learning**, Springer, v. 5, n. 2, p. 197–227, 1990.

SOUSA, R.; GAMA, J. Co-training semi-supervised learning for single-target regression in data streams using amrules. In: SPRINGER. **International Symposium on Methodologies for Intelligent Systems**. [S.l.], 2017. p. 499–508.

SOUZA, F. M. de; GRANDO, J.; BALDO, F. Adaptive fast XGBoost for regression. In: XAVIER-JUNIOR, J. C.; RIOS, R. A. (Ed.). **Intelligent Systems**. Cham: Springer International Publishing, 2022. p. 92–106. ISBN 978-3-031-21686-2.

STROBL, C.; MALLEY, J.; TUTZ, G. An introduction to recursive partitioning: rationale, application, and characteristics of classification and regression trees, bagging, and random forests. **Psychological methods**, American Psychological Association, v. 14, n. 4, p. 323, 2009.

WAZLAWICK, R. **Metodologia de pesquisa para ciência da computação**. [S.l.]: Elsevier Brasil, 2017. v. 2.

WITTEN, I. H. et al. **Data Mining: Practical machine learning tools and techniques**. [S.l.]: Morgan Kaufmann, 2016.

WU, X. et al. Top 10 algorithms in data mining. **Knowledge and information systems**, Springer, v. 14, n. 1, p. 1–37, 2008.

YANG, L.; MANIAS, D. M.; SHAMI, A. Pwpae: An ensemble framework for concept drift adaptation in iot data streams. **arXiv preprint arXiv:2109.05013**, 2021.

YU, H.; LU, J.; ZHANG, G. Morstreaming: A multioutput regression system for streaming data. **IEEE Transactions on Systems, Man, and Cybernetics: Systems**, p. 1–13, 2021.

ŽLIOBAITĖ, I. Learning under concept drift: an overview. **arXiv preprint arXiv:1010.4784**, 2010.