Pontifícia Universidade Católica do Paraná
Programa de Pós-Graduação em Informática

ABORDAGENS ADAPTATIVAS DE RAMIFICAÇÃO EM ÁRVORES DE DECISÃO INCREMENTAIS PARA MINERAÇÃO DE FLUXO DE DADOS

Daniel Nowak Assis

Curitiba
2025

Pontifícia Universidade Católica do Paraná
Programa de Pós-Graduação em Informática

ABORDAGENS ADAPTATIVAS DE RAMIFICAÇÃO EM ÁRVORES DE DECISÃO INCREMENTAIS PARA MINERAÇÃO DE FLUXO DE DADOS

Daniel Nowak Assis

Dissertação apresentada ao Programa de Pós-Graduação em Informática como requisito parcial para obtenção do título de Mestre em Informática.

Campo de Concentração: Ciência da Computação

Orientador: Fabrício Enembreck

Co-orientador: Jean Paul Barddal

Curitiba
2025

Dados da Catalogação na Publicação
Pontifícia Universidade Católica do Paraná
Sistema Integrado de Bibliotecas – SIBI/PUCPR
Biblioteca Central
Luci Eduarda Wielganczuk – CRB 9/1118

A848a
2025

Assis, Daniel Nowak
Abordagens adaptavivas de ramificação em árvores de decisão incrementais para mineração de fluxo de dados / Daniel Nowak Assis ; orientador: Fabricio Enembreck ; co-orientador: Jean Paul Barddal. – 2025.
119 f. : il. ; 30 cm

Dissertação (mestrado) – Pontifícia Universidade Católica do Paraná, Curitiba, 2025
Bibliografia: f. 111-119

1. Informática. 2. Mineração de dados. 3. Árvores de decisão. 4. Aprendizado de máquina. I. Enembreck, Fabrício. II. Barddal, Jean Paul. III. Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Informática. IV. Título.

CDD. 20. ed. – 004

Pontifícia Universidade Católica do Paraná
Escola Politécnica
Programa de Pós-Graduação em Informática

Curitiba, 26 de março de 2025.

54-2025

DECLARAÇÃO

Declaro para os devidos fins, que **DANIEL NOWAK ASSIS** defendeu a dissertação de Mestrado intitulada "**ABORDAGENS ADAPTATIVAS DE RAMIFICAÇÃO EM ÁRVORES DE DECISÃO INCREMENTAIS PARA MINERAÇÃO DE FLUXO DE DADOS"**, na área de concentração Ciência da Computação no dia 13 de março de 2025, no qual foi aprovado.

Declaro ainda, que foram feitas todas as alterações solicitadas pela Banca Examinadora, cumprindo todas as normas de formatação definidas pelo Programa.

Por ser verdade, firmo a presente declaração.

_____
Prof. Dr. Emerson Cabrera Paraiso
Coordenador do Programa de Pós-Graduação em Informática

Von einem gewissen Punkt an gibt es keine Rückkehr mehr. Dieser Punkt ist zu erreichen. - Franz Kafka

Resumo

A área de mineração de fluxo de dados estuda o desenvolvimento de algoritmos capazes de extrair padrões em dados produzidos constantemente em alta velocidade em forma de fluxo. Na tarefa de classificação de fluxo de dados, um dos principais e mais utilizados algoritmos é a *Hoeffding Tree*, árvore de decisão projetada para o cenário de fluxo de dados. Para evitar a indução de uma árvore de decisão para cada exemplo que se deseja classificar, a *Hoeffding Tree* mantém estatísticas de instâncias em nós folhas e periodicamente realiza tentativas de ramificação com base no teorema de Hoeffding. Contudo, essa abordagem não considera a evolução dos dados ao longo do tempo. Mesmo que uma mudança evidente ocorra na distribuição dos dados ou na acurácia de um nó folha, uma tentativa de ramificação ocorrerá posteriormente, dado que a verificação de novas ramificações é feita periodicamente. Por outro lado, mesmo quando pouca mudança ocorre em acurácia ou na função de pureza da distribuição, a avaliação gulosa da melhor ramificação com todos os atributos e seus valores ainda é realizada, consumindo tempo de processamento desnecessariamente. Esse trabalho foca no desenvolvimento de árvores de decisão com nós-folha adaptativos que visam contornar as limitações de *Hoeffding Tree*. A adaptabilidade em nós folhas é alcançada por algoritmos de detecção de mudança, que constantemente são atualizados com instâncias que chegam e determinam os momentos em que ramificações irão ocorrer. As árvores de decisão propostas nesse trabalho são avaliadas a nível monolítico e de ensemble, apresentando desempenho superior ao de topologias do estado-da-arte.

Palavras-chave: Mineração de Fluxo de Dados, Árvores de Decisão, Aprendizado de máquina.

Abstract

Data stream mining encompasses the development of algorithms capable of extracting patterns from data that arrives constantly and in high-speed. In the task of data stream classification, one of the main and widely used algorithm is Hoeffding Tree, a decision tree adapted to the data stream scenario. To avoid the induction of a new decision tree for each data sample desired to classify, Hoeffding Trees maintain statistics in leaf nodes and periodically attempt to split based on the Hoeffding theorem. However, this approach does not take into account the evolution of the stream throughout time. Even if a clear change happens in the data distribution or accuracy of a leaf node, a split attempt will occur posteriorly, given that split attempts are done periodically. Or when little change in accuracy or the purity function of the data distribution ensue, a greedy evaluation of the best split with all features and its values will still be done, consuming unnecessary time. This work focuses on the development of decision trees with adaptive leaf nodes that aim to cover the described Hoeffding Trees limitations. The adaptability is reached through change detection algorithms, that are constantly updated with arriving instances which determine when a split should occur. The Decision Trees proposed in this work are evaluated in monolithic and ensemble levels, presenting superior predictive quality in comparison to state-of-the-art algorithms.
Keywords: Data stream mining, Decision Tree, Machine Learning

Agradecimentos

Ao meu orientador, Prof. Fabrício Enembreck, por me orientar desde o segundo período até o fim do PIBIC-Master. Para cada ideia apresentada e todo artigo escrito, o Prof. Fabrício foi fundamental para o refinamento e qualidade de toda pesquisa que realizei nesses últimos 4 anos. O impacto do Prof. Fabrício na minha carreira como pesquisador é imensurável.

Ao meu coorientador, Prof. Jean Paul Barddal, por toda contribuição durante meu mestrado, e desde minha primeira publicação até todas as próximas que teremos pela frente. Todas as discussões foram importantes para os resultados dessa dissertação. Obrigado por todo o esforço e dedicação.

Aos meus pais, tios e avós, que são o pilar de todas as oportunidades que já tive na vida e ainda terei. Obrigado por todo apoio incondicional em todos os momentos.

A Prof. Leilah Paiva, por me ensinar valores importantes que refletem na minha vida profissional e pessoal.

A Sanziâna Dobrovicescu e Mihai Codrea, pela ajuda em momentos de dificuldade. Talvez nunca encontre ninguém com nível técnico e humanístico simultaneamente tão elevados como vocês apresentam.

Aos meus amigos Alessandro, Thiago, José e Gustavo. Obrigado por todos os momentos. Vamos nos encontrar em vários cantos do mundo.

A Prof^a Cleybe H. Vieira e toda equipe da Coordenação de Iniciação Científica, por permitir a realização do meu mestrado simultaneamente com minha graduação através do programa de bolsa PIBIC-Master. É um programa fantástico para formação de novos pesquisadores.

A Deus, por tudo.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Acrônimos

ADWIN Adaptive Windowing,

ARE Adaptive Regularized Ensemble

ARF Adaptive Random Forest

ARTE Adaptive Random Tree Ensemble

DDM Drift Detection Method

EDDM Early Drift Detection Method

EFDT Extremely Fast Decision Tree

HAT Hoeffding Adaptive Tree

HT Hoeffding Tree

HDDM$_A$ Hoeffding Drift Detection Method com média

HDDM$_W$ Hoeffding Drift Detection Method com ponderação de instâncias mais recentes como mais importantes

LAST Local Adaptive Streaming Tree com monitoramento de erro

LAST$_D$ Local Adaptive Streaming Tree com monitoramento de distribuição de dados

MDDM$_A$ McDiarmid Drift Detection Method com estratégia aritmética

MDDM$_E$ McDiarmid Drift Detection Method com estratégia exponencial

MDDM$_G$ McDiarmid Drift Detection Method com estratégia geométrica

RDDM Reactive Drift Detection Method

VFDT Very Fast Decision Tree

SRP Streaming Random Patches

1

Introdução

A área de Aprendizado de Máquina tem como objetivo investigar como computadores podem realizar um processo de aprendizado baseado em dados. Comumente, algoritmos de aprendizado de máquina são projetados para lidar com dados armazenados em memória, conhecidos como lote de dados (HAN; KAMBER; PEI, 2011).

Todavia, a enorme quantidade de dados gerados em tempo real em cenários como transações bancárias (BARDDAL et al., 2020), *Internet-of-Things (IoT)* (MORALES et al., 2016) e mídias sociais (ZHAO et al., 2019) impõe desafios para descoberta de padrões, pela sua natureza potencialmente infinita e de rápida e constante geração.

Destarte, pesquisadores tem dado atenção para o desenvolvimento de algoritmos de mineração de fluxo de dados, que são capazes de treinar em tempo real e incrementalmente assim que uma instância de dado estiver disponível para treino, considerando restrições de memória e tempo de processamento, e que são adaptativos, para que a performance do sistema não seja afetada caso mudanças ocorram nos dados ao longo do tempo (GOMES et al., 2019).

Como em aprendizado de máquina em lote, a mineração de fluxo de dados engloba o desenvolvimento de técnicas onde consideram cenários nos quais as instâncias de dados são totalmente rotuladas (supervisionado), parcialmente rotuladas (semi-supervisionado) e sem qualquer rotulação (não supervisionado). Nesse trabalho é abordada a tarefa supervisionada de classificação, em que um modelo recebe como entrada um conjunto de características com dados nominais ou numéricos, e a partir dessa instância é capaz de atribuir uma classe

discreta a esse exemplo (HAN; KAMBER; PEI, 2011).

Outra problemática que norteia o desenvolvimento da área são as mudanças de conceito (*concept drifts*). Mudanças de conceito são mudanças que ocorrem nas propriedades estatísticas da distribuição e comportamento dos dados ao longo do fluxo, sendo possível que algum ruído esporádico se torne efetivamente o conceito que mapeia os dados (SCHLIMMER; GRANGER, 1986). Essas mudanças podem afetar o desempenho preditivo de classificadores, pois uma vez detectadas, os modelos devem se adaptar às mudanças. Para isso, diversos algoritmos foram criados para detectar mudanças de conceito (LU et al., 2018).

Um dos modelos mais utilizados para o problema de classificação são as árvores de decisão, pela sua utilidade, interpretabilidade, e robustez em vários problemas (COSTA; PEDREIRA, 2022). Árvores de decisão consistem em estruturas com múltiplas separações lineares para produzir um modelo em formato de árvore em que instâncias de uma mesma classe sejam maioria ou únicas em nós folhas. Diversas estratégias de ramificações em árvores foram propostas, como nos algoritmos ID3 (QUINLAN, 1986), C4.5 (QUINLAN, 1992) e CART (BREIMAN, 1984).

Para lidar com *streams* de dados, em Domingos & Hulten (2000), os autores propuseram o algoritmo *Hoeffding Tree*. Esse algoritmo herda mecanismos de ramificação de árvores de decisão de aprendizado de máquina em lote, porém para lidar com os requisitos necessários em um cenário de fluxo de dados, um nó folha acumula informações de instâncias e realiza tentativas de ramificação periodicamente, sendo que a periodicidade em que tentativas de ramificação são feitas é determinada por meio de hiper-parâmetros. Dessa forma, o custo de busca pela melhor ramificação é distribuído ao longo do processamento do fluxo de dados.

Hoeffding Trees são amplamente usadas em tarefas de classificação de fluxo de dados e são o principal *base learner* de *ensembles* estado-da-arte orientados a fluxos de dados (GOMES et al., 2017a). Porém, o algoritmo apresenta inconsistências. Em (RUTKOWSKI et al., 2013), os autores evidenciam que a aplicação do teorema de Hoeffding (HOEFFDING, 1963) como estratégia de tentativa de ramificação é errônea, visto que métricas de pureza como ganho de informação (SHANNON, 1948) e gini index (BREIMAN, 1984) não são médias de valores aleatórios, ou seja, a tentativa de ramificação utilizando o teorema é uma heurística. Além disso, o algoritmo realiza tentativas de ramificações em frequência constante, sendo que cada cenário pode ter um valor ideal distinto. Entretanto, o monitoramento

constante da acurácia ou distribuição dos dados poderia permitir a árvore reagir com maior velocidade e se ajustar mais rapidamente às mudanças, enquanto em situações de estabilidade a quantidade de *splits* pode ser menos frequente.

Esse trabalho apresenta o algoritmo *Local Adaptive Streaming Tree* (LAST), que aplica uma estratégia de ramificação adaptativa que considera o estado da árvore e seus nós folhas. A estratégia consiste em monitorar mudanças em estatísticas (como taxa de acerto e distribuição de classe) referentes aos nós folhas por meio de algoritmos de detecção de mudança (LU et al., 2018), que determinam os momentos de ramificação da árvore. A estratégia de ramificação adaptativa visa monitorar a evolução do fluxo e evitar tempo de processamento desnecessário em avaliação gulosa de ramificações em casos que pouca mudança ocorreu no estado da árvore, dado que a verificação de melhores ramificações é realizada somente quando o algoritmo de detecção de mudança de conceito alertar uma mudança.

1.1 Motivação

Embora *Hoeffding Trees* sejam um método amplamente usado na literatura de fluxo de dados, as suas inconsistências devem ser contornadas com novas propostas ou agregadas aos seus aspectos positivos de forma que a árvore seja capaz de considerar o estado do fluxo e sua performance para se adaptar quando necessário. Além disso, testes estatísticos diferentes de Hoeffding apresentados na literatura por vezes não focam no protocolo experimental e aumento da qualidade preditiva em comparação com *Hoeffding Trees* (RUTKOWSKI et al., 2013).

O presente trabalho visa propor uma solução que resulte em melhoria da qualidade preditiva de árvores de decisão incrementais e também abranger aspectos fundamentais para mineração de fluxo de dados como custo computacional e complexidade do modelo de árvore de decisão.

1.2 Hipótese

Hipótese: Árvores de decisão com mecanismos adaptativos de ramificação em nós folhas obtêm resultados superiores em acurácia comparados ao estado-da-arte de árvores de decisão, enquanto possuem tempo de processamento

equiparável a Hoeffding Trees dado que a verificação da melhor ramificação ocorre somente mediante alerta do detector de mudança e não periodicamente.

1.3 Objetivos

O objetivo geral desse projeto consiste em desenvolver e avaliar novos métodos de árvores de decisão nos cenários monolíticos e *ensemble* que resultem em melhoria da qualidade preditiva sem aumento substancial de custo computacional.

Os objetivos específicos desse projeto consistem em:

1) Desenvolver e avaliar métodos de ramificação e crescimento de árvores incrementais baseados em detectores de mudança como classificadores monolíticos e membros de *ensemble*.
2) Realizar um *ablation* do método proposto e árvores de decisão incrementais.

O objetivo 1 consiste em prover um comparativo da árvore de decisão proposta nesse trabalho com árvores de decisão incrementais estado-da-arte como árvores individuais (ambiente monolítico) e componentes de um conjunto de classificadores (ambiente *ensemble*). O objetivo 2 consiste em demonstrar como o método proposto funciona em cenários fundamentais de árvores de decisão incrementais, como o efeito das ramificações na acurácia ao longo do tempo, comportamento de qualidade preditiva e custo computacional da árvore de decisão ao variar o número de classes a atributos de um problema e reação a mudanças de conceito.

1.4 Organização

Esse trabalho está organizado da seguinte forma: o Capítulo 2 apresenta detalhadamente os métodos e conceitos de mineração de fluxo de dados que embasam o desenvolvimento e resultados da proposta presente nesse trabalho. O Capítulo 3 apresenta o método proposto e técnicas adicionadas ao método como regularização e reavaliação de ramificações. O Capítulo 4 apresenta os resultados obtidos. Por fim, o Capítulo 5 conclui esse trabalho e indica trabalhos futuros.

1.5 Suporte Financeiro

O presente trabalho foi realizado com o apoio da Associação Paranaense de Cultura (APC), através do Programa de Iniciação Científica na modalidade PIBIC Master - Combined Degree , que permite ao estudante a realização do mestrado simultaneamente a graduação. O projeto foi aprovado no Edital 09/2023 e intitulado *Abordagens Adaptativas de Ramificação em Árvores de Decisão Incrementais para Mineração de Fluxo de Dados*.

2

Fundamentação Teórica

A tarefa de mineração de dados é o processo de descoberta de padrões e conhecimento a partir de um conjunto de dados. A mineração de dados engloba várias tarefas como classificação, regressão, agrupamento, dentre outras (HAN; KAMBER; PEI, 2011).

Diferentemente do cenário em lote, conhecido como *batch learning*, em mineração de fluxo de dados as instâncias chegam continuamente e de forma possivelmente ilimitada. Para o cenário de fluxo de dados, vários algoritmos foram desenvolvidos para processamento incremental e eficiente de fluxos.

Essa seção visa apresentar uma fundamentação teórica para a proposta desse trabalho de forma a facilitar o entendimento da proposta. A Seção 2.1 apresenta o problema de classificação de fluxo de dados. A Seção 2.2 discorre a forma de avaliação de classificadores em fluxo de dados. A Seção 2.3 apresenta técnicas de validação de classificadores em fluxo de dados. A Seção 2.4 discorre do problema de mudança de conceito. A Seção 2.5 detalha algoritmos que realizam monitoramento explícito de mudança de conceito. A Seção 2.6 apresenta algoritmos de árvore de decisão incrementais. A Seção 2.7 detalha inconsistências presentes em *Hoeffding Trees*. A Seção 2.8 discute condições de ramificação em árvores de decisão incrementais que propõem alternativas para inconsistências em *Hoeffding Trees*. A Seção 2.9 apresenta algoritmos baseados em *ensemble* e a Seção 2.10 conclui a fundamentação teórica.

2.1 Classificação de Fluxo de Dados

Classificação de fluxo de dados é a tarefa de predição de uma classe discreta a partir de um vetor de características de dados numéricos, nominais ou mistos que representam as instâncias de aprendizagem. As instâncias podem chegar continuamente, de maneira rápida e potencialmente infinita (GAMA, 2010).

Formalmente, aprendizado supervisionado de classificação de fluxo de dados é composto de um conjunto $S = \{(X, Y)\}$, onde $X = \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_t, \ldots, \vec{x}_\infty\}$ é um conjunto de vetores de características de d-dimensões observado no tempo t, e $Y = \{y_1, y_2, y_3, \ldots, y_t, \ldots, y_\infty\}$ é um conjunto de classes discretas de rótulos referentes ao conjunto de vetores e a cardinalidade de X e Y é potencialmente infinita. Classificadores são algoritmos que tem como objetivo, dado como entrada um vetor de características, definir a classe que melhor descreve o vetor de características ($f : \vec{x}_n \rightarrow y_n$).

Para uma mineração de fluxo de dados eficiente, algoritmos devem considerar restrições referentes a natureza rápida e potencialmente infinita (e.g., chegada de centenas de instâncias por segundo (ZHONG; SOUZA; MUEEN, 2022)) do fluxo. Como apontado em (BIFET et al., 2010), um modelo de aprendizado de máquina para minerar fluxo de dados idealmente i) processa uma instância de dado e analisa-o somente uma vez, ii) usa uma quantidade limitada de memória, iii) trabalha o mais rapidamente possível e iv) pode realizar predições a qualquer momento.

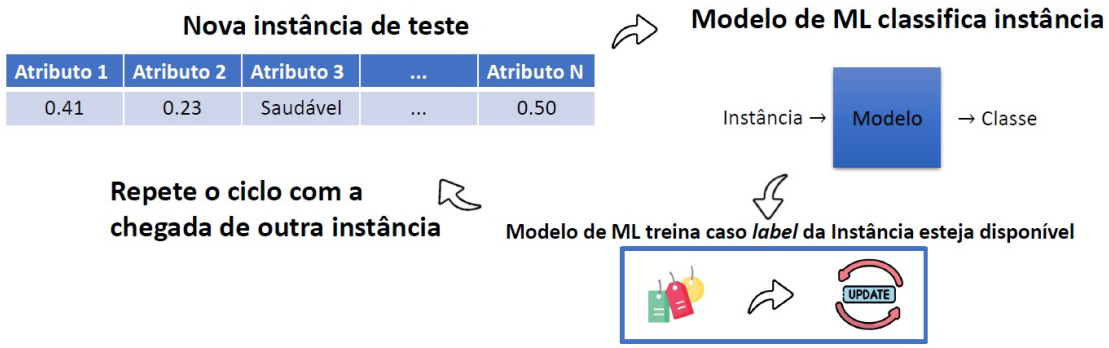

Figura 2.1: Ciclo de classificação de fluxos

Autoria: O Autor (2023)

A Figura 2.1 ilustra o ciclo de classificação de fluxo de dados, onde uma ins-

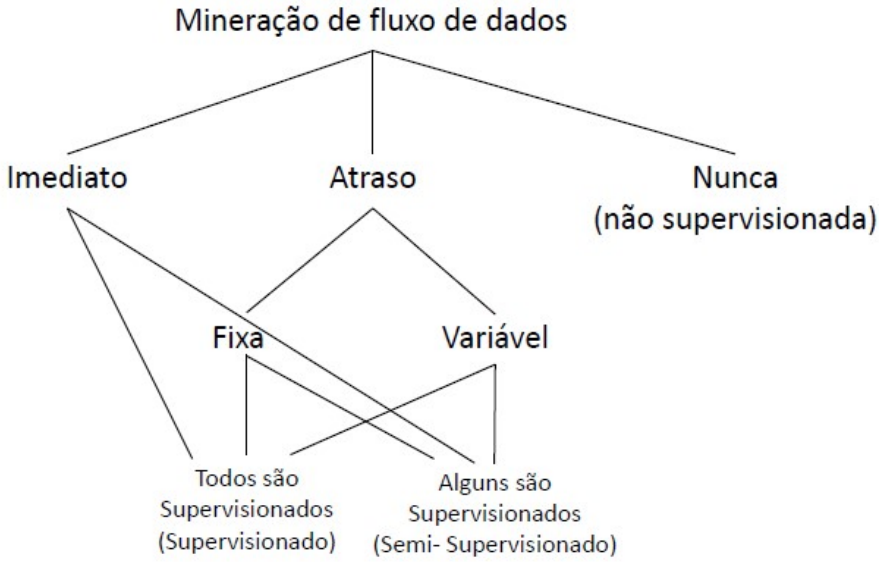

Figura 2.2: Cenários de disponibilidade de rótulos no contexto de fluxo de dados

Autoria: Autor, 2023. Adaptado de (GOMES et al., 2022)

tância chega, o modelo realiza a predição (teste) e realiza treino com a instância caso seu rótulo esteja disponível. Esse processo se repete até não haver mais instâncias disponíveis para realizar o processo de classificação ou *ad eternum*, caso o fluxo seja infinito. Consequentemente, as restrições apontadas em (BIFET et al., 2010) são importantes no processo de classificação de fluxo de dados, pois se o modelo demora para processar uma instância mais do que o tempo de chegada de outra instância (pontos i e iii pelo rápido processamento de instâncias e ponto iv pela disponibilidade do modelo para realizar classificações a qualquer momento), instâncias devem ser armazenadas e muitas instâncias armazenadas podem colapsar o sistema por falta de memória (ponto ii), ou instâncias são descartadas, tornando o modelo desatualizado (GOMES et al., 2017b).

A Figura 2.2 ilustra uma taxonomia sobre diferentes cenários de fluxo de dados considerando a disponibilidade do rótulo, em que há a possibilidade de haver disponibilidade imediata, (i) com atraso fixo (rótulos das instâncias chegam em tempo fixo), (ii) variável (rótulos das instâncias chegam em tempos diferentes), e (iii) nunca disponíveis (Não supervisionado).

O presente trabalho engloba o desenvolvimento de algoritmos para o cenário onde rótulos tem disponibilidade imediata para treino. A maioria dos trabalhos de mineração de fluxo de dados lida com esse cenário. Dessa forma, torna-se

possível comparar os métodos propostos com uma gama maior de abordagens da literatura e outros cenários de fluxo de dados.

2.2 Avaliação de Classificadores

A avaliação de modelos de aprendizagem de máquina é realizada por métricas de avaliação. Para classificação, uma métrica comum de avaliação é a acurácia, sendo a razão entre o número correto de predições p_c e o número total de instâncias n, como apresentado na Equação 2.1.

$$\text{Acurácia} = \frac{p_c}{n} \tag{2.1}$$

A acurácia pode ser uma métrica adequada para avaliar modelos de aprendizado de máquina, porém em alguns cenários podem levar a conclusões equivocadas.

Por exemplo, em cenários com desbalanceamento de classes (i.e. casos em que instâncias de uma classe são mais numerosas que instâncias de outra classe). Em um cenário em que 95% dos exemplos são de uma classe A e 5% são de uma classe B, o modelo pode ter 95% de acurácia, mesmo errando todos os exemplos da classe B.

Uma métrica comumente utilizada em cenários de desbalanceamento é a *Kappa*. Kappa é a concordância entre classificações corretas e a distribuição de classes, corrigindo concordâncias que ocorrem por mera casualidade. A Equação 2.2 apresenta o cálculo do Kappa, onde p_{ran} é a probabilidade do modelo classificar as classes randomicamente.

$$\text{Kappa} = \frac{\text{Acurácia} - p_{ran}}{1 - p_{ran}} \tag{2.2}$$

Nesses cenários, uma análise do número de acertos por classe é preferível. Para o leitor interessado, o trabalho de (HE; MA, 2013) é recomendado para métricas de avaliação em cenários com desbalanceamento.

Em (BIFET et al., 2015a), os autores propõem a métrica *Kappa Majority*. Ao invés de corrigir concordâncias que ocorrem por mera casualidade, *Kappa Majority* realiza o reajuste em comparação com um classificador que realiza predições com a classe majoritária.

Em (ŽLIOBAITė et al., 2015), os autores propõem a métrica *Kappa Temporal*,

em que ao invés do reajuste ser realizado pela acurácia de um classificador randômico, o reajuste é realizado pela acurácia de um classificador persistente, em que a predição de uma instância é realizada pela classe da instância anterior. *Kappa Temporal* visa, então, punir classificadores que não tem vantagem em *datasets* com dependências temporais quando comparados com o classificador persistente.

Como abordado na Seção 2.1, tempo de processamento e memória são componentes importantes para algoritmos de mineração de fluxo de dados e serão avaliados nesse trabalho. Tempo de processamento é avaliado pelo tempo que a unidade central de processamento foi utilizada para processar instruções (*CPU-Time*) e memória é avaliada pela quantidade de espaço de acesso aleatório consumido pelo algoritmo em Gigabytes por hora (*RAM-Hours*).

2.3 Validação de Classificadores em Fluxos de Dados

Em *machine learning* para lote, estratégias de validação cruzada são comumente usadas para validar os modelos. Porém, em cenários de fluxo de dados, onde instâncias não estão disponíveis indefinidamente em memória e chegam continuamente, outros tipos de avaliação são mais adequadas, como a validação *Prequential* (conhecida como teste e treino) (GAMA; SEBASTIAO; RODRIGUES, 2013). Na validação *Prequential*, cada instância que chega é primeiro utilizada para teste, e depois usada para treino. O algoritmo incremental de Welford (WELFORD, 1962) é aplicado para cálculo da acurácia e apresentado no Algoritmo 1 com atualização da acurácia com a função indicação[1] da predição do algoritmo ser igual ao rótulo real.

Em (BIFET et al., 2015b), os autores propõem uma forma de realizar *k-Cross-Validation* em cenários de fluxo de dados. Para avaliação de um modelo de classificação, k classificadores são criados. Os k classificadores recebem instâncias primeiro para teste e depois para treino. Porém, dada a n-ésima instância n e o k-ésimo classificador, se $n \bmod k = 0$, o k-ésimo classificador não realiza treino com a instância. Ao final do processo de classificação, é possível extrair a acurácia média dos classificadores.

[1] A função indicação tem valor 1 caso sua entrada seja verdadeira, e 0 caso falsa

Algoritmo 1 Acurácia *Prequential*

Seja Alg um algoritmo de aprendizado de máquina
Inicialize acurácia $\leftarrow 0$ ▷ inicialização da acurácia
Inicialize $n \leftarrow 0$ ▷ número de instâncias observadas
for cada exemplo $(\vec{x}, y) \in S$ **do**
    $n \leftarrow n + 1$
    $\hat{y} \leftarrow Alg(\vec{x})$ ▷ predição do algoritmo para instância
    acurácia $\leftarrow \frac{\mathbb{1}\{\hat{y}=y\}+(n-1)\cdot\text{acurácia}}{n}$ ▷ atualização da acurácia com função indicação
    Treine Alg com $(\vec{x}, y)$
end for
    return acurácia

2.4 Mudança de Conceito

Mudanças de conceito são definidas como mudanças nas propriedades probabilísticas dos dados, quando novos conceitos que mapeiam os dados emergem, fazendo com que modelos de classificação fiquem desatualizados com informação de conceitos passados (LU et al., 2018), sendo problemas inerentes na literatura de mineração de fluxo de dados.

Formalmente, dadas 2 marcas temporais t e $t+\Delta$, uma mudança de conceito é observável quando $P_t(X, Y) \neq P_{t+\Delta}(X, Y)$, onde P_t se refere a probabilidade conjunta no marco temporal t dado um conjunto de exemplos X e suas respectivas classes Y.

2.4.1 Mudança de Conceito Real e Virtual

Dado que a probabilidade conjunta $P_t(X, Y)$ pode ser decomposta em $P_t(X, Y) = P_t(X)P_t(Y|X)$, três formas de mudança de conceito são denotadas e ilustradas na Figura 2.3, sendo elas:

1. **Forma 1:** $P_t(X) \neq P_{t+\Delta}(X)$ enquanto $P_t(Y|X) = P_{t+\Delta}(Y|X)$. Nessa forma de mudança de conceito, mudanças nos vetores de características são observáveis. Dado que $P_t(X)$ não afeta o mapeamento de classes dado os vetores de características, essa forma de mudança de conceito é conhecida como mudança de conceito virtual (RAMÍREZ-GALLEGO et al., 2017).

2. **Forma 2:** $P_t(X) = P_{t+\Delta}(X)$ enquanto $P_t(Y|X) \neq P_{t+\Delta}(Y|X)$. Essa forma de mudança afeta o mapeamento de classes dado os vetores de características, afetando a performance dos modelos negativamente, e é conhecida como mudança de conceito real.

3. **Forma 3:** $P_t(X) \neq P_{t+\Delta}(X)$ enquanto $P_t(Y|X) \neq P_{t+\Delta}(Y|X)$. Essa forma de mudança é uma combinação das formas 1 e 2.

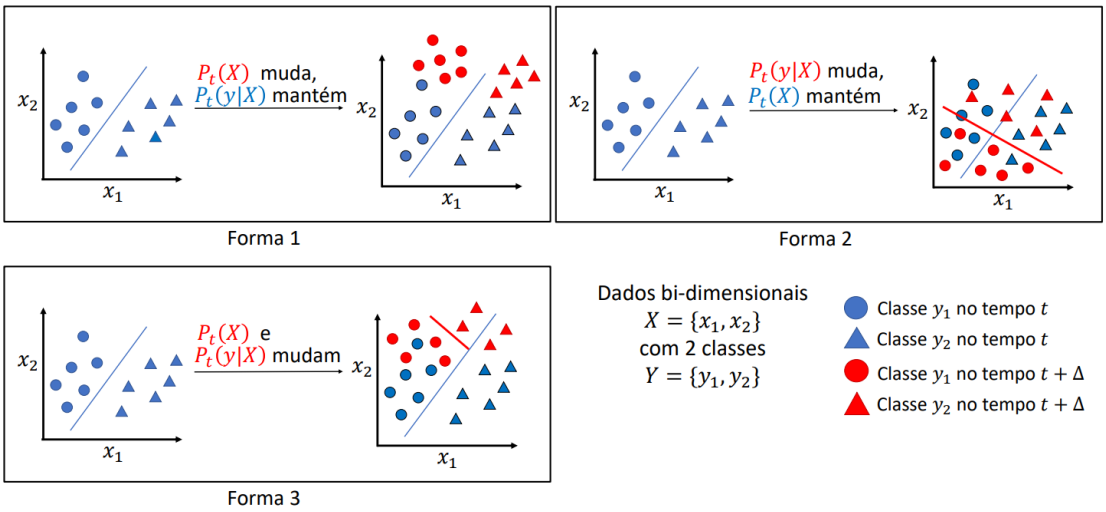

Figura 2.3: Formas de Mudança de Conceito

Autoria: Autor, 2023. Adaptado de (LU et al., 2018)

2.4.2 Taxas de Mudança de Conceito

Mudanças de conceito também podem ser categorizadas pela forma que a alteração ocorreu ao londo do tempo, sendo importante delimitá-las, pois técnicas diferentes podem ser mais eficientes em certas taxas de mudança.

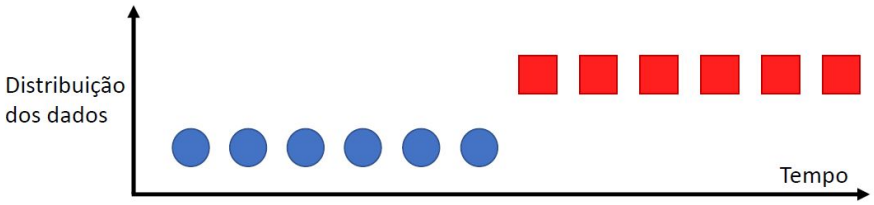

Figura 2.4: Mudança Abrupta de Conceito

Autoria: Autor, 2023. Adaptado de (LU et al., 2018)

A Figura 2.4 ilustra uma mudança de conceito abrupta, onde em um intervalo curto de tempo instâncias de outra instância emergem e são predominantes.

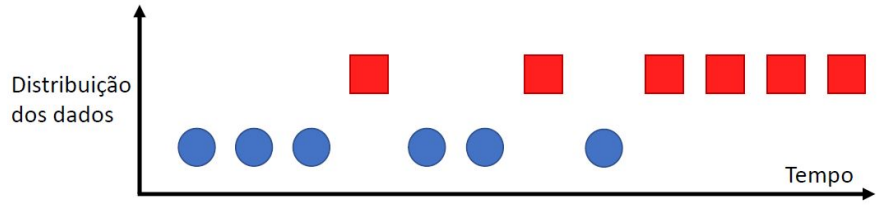

Figura 2.5: Mudança Gradual de Conceito

Autoria: Autor, 2023. Adaptado de (LU et al., 2018)

Diferentemente da mudança abrupta de conceito, mudanças graduais de conceito (ilustradas na Figura 2.5) são definidas como a aparição gradual de instâncias de outro conceito até ser efetivamente predominante.

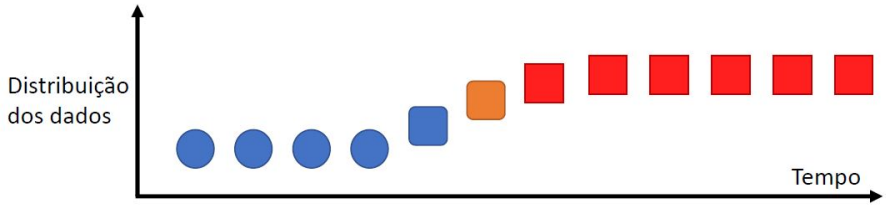

Figura 2.6: Mudança Incremental de Conceito

Autoria: Autor, 2023. Adaptado de (LU et al., 2018)

Mudanças incrementais de conceito (ilustradas na Figura 2.6) são similares a mudanças de conceito graduais, porém a transição entre 2 conceitos é composta de outros conceitos intermediários.

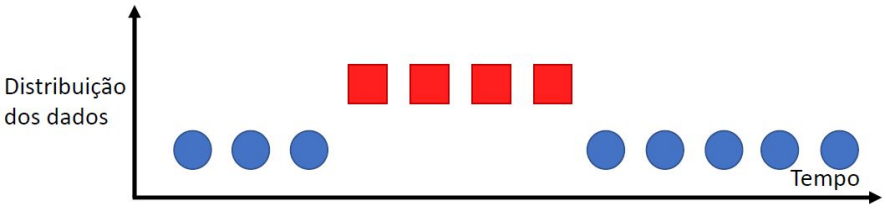

Figura 2.7: Mudança Recorrente de Conceito

Autoria: Autor, 2023. Adaptado de (LU et al., 2018)

Mudanças recorrentes de conceito (ilustradas na Figura 2.7) se referem a mudanças de conceito onde instâncias de um conceito anterior voltam a ser

novamente predominantes.

2.5 Monitoramento Explícito de Mudanças de Conceito

Para lidar com ambientes não estacionários (com mudança de conceito), duas abordagens são comuns na literatura (CAVALCANTE; MINKU; OLIVEIRA, 2016). Uma é a **implícita**, em que os modelos de *machine learning* adaptam sua estrutura para serem resilientes a mudanças de conceito. Modelos como os descritos na Seção 2.6.2 e 2.6.4 lidam com mudanças de conceito de forma implícita. Outra é a **explícita**, em que estatísticas referentes aos dados ou ao modelo de *machine learning* são monitoradas para determinação se em dado momento do fluxo de dados uma mudança de conceito ocorreu e atualizações no modelo são necessárias. As próximas seções são dedicadas à apresentação de alguns dos métodos mais conhecidos na literatura de detecção de mudanças.

2.5.1 *Drift Detection Method (DDM)*

Em (GAMA et al., 2004), os autores propuseram um dos primeiros métodos explícitos para lidar com mudanças de conceito. Esse trabalho tem como suposição que se a qualidade preditiva de classificador decresceu, uma mudança de conceito ocorreu.

O algoritmo recebe entradas binárias (1 caso o classificador erre, e 0 caso o classificador acerte) e monitora métricas como média $\overline{X}$ e desvio padrão $\sigma(X)$ dos valores observados. A cada instante t de tempo, será verificado se $\overline{X}_t + \sigma(X)_t < \overline{X}_{min} + \sigma(X)_{min}$. Caso verdadeiro, $\overline{X}_{min} = \overline{X}_t$ e $\sigma(X)_{min} = \sigma(X)_t$. Como apresentado em (GAMA et al., 2004), a situação de *warning* ocorre quando um *drift* é eminente e a situação de *drift* sinaliza a ocorrência de uma mudança. *Warnings* são alertados caso a Equação 2.3 seja verdadeira e *Drifts* são alertados caso a Equação 2.4 seja verdadeira.

$$\overline{X}_t + \sigma(X)_t \geq \overline{X}_{min} + \delta_{warning} \cdot \sigma(X)_{min} \tag{2.3}$$

$$\overline{X}_t + \sigma(X)_t \geq \overline{X}_{min} + \delta_{drift} \cdot \sigma(X)_{min} \tag{2.4}$$

sendo $\delta_{warning} = 2$ e $\delta_{drift} = 3$ os valores sugeridos em (GAMA et al., 2004). A estratégia do algoritmo DDM consiste em monitorar se a média dos valores observados difere dos valores mínimos de média observados anteriormente no fluxo. Outras alternativas foram propostas na literatura, como utilização de limiares menos restritivos para detecção de *drift* como $\delta_{warning} = 1.2$ e $\delta_{drift} = 1.95$ (BARROS; SANTOS; JÚNIOR, 2016). Como abordado em (BAENA-GARCIA et al., 2006), esse método é ideal para cenários com mudança de conceito abrupta e gradual rápida, porém não é adequado quando a mudança ocorre lentamente.

2.5.2 *Early Drift Detection Method (EDDM)*

Visando um método robusto para mudanças de conceito que ocorrem lentamente, os autores em (BAENA-GARCIA et al., 2006) propõem o algoritmo *Early Drift Detection Method* (EDDM). Ao invés de considerar o erro médio do classificador, ele considera a média da diferença entre 2 momentos t e $t + \Delta$ em que o classificador errou. A cada erro feito pelo classificador, se $(\overline{\Delta}_t + 2 \cdot \sigma(\Delta)_t) > (\overline{\Delta}_{max} + 2 \cdot \sigma(\Delta)_{max})$, $\overline{\Delta}_{max} = \overline{\Delta}_t$ e $\sigma(\Delta)_{max} = \sigma(\Delta)_t$. *Warnings* e *drifts* são alertados pelo algoritmo de acordo com as Equações 2.5 e 2.6, respectivamente.

$$(\overline{\Delta}_t + 2 \cdot \sigma(\Delta)_t)/(\overline{\Delta}_{max} + 2 \cdot \sigma(\Delta)_{max}) \leq \alpha \tag{2.5}$$

$$(\overline{\Delta}_t + 2 \cdot \sigma(\Delta)_t)/(\overline{\Delta}_{max} + 2 \cdot \sigma(\Delta)_{max}) \leq \beta \tag{2.6}$$

Os autores utilizam $\alpha = 0,95$ e $\beta = 0,9$ como s avaliados em (BAENA-GARCIA et al., 2006). A estratégia do algoritmo EDDM consiste em monitorar se a média da distância entre 2 momentos em que o classificador errou diminuiu. Em casos de *drift* que ocorrem lentamente, a acurácia do classificador é afetada lentamente e a diminuição da distância entre 2 erros feitos pelo classificador é mais evidente, tornando *EDDM* um método mais eficaz para *drifts* que ocorrem de forma lenta.

2.5.3 *Adaptive Windowing (ADWIN)*

ADaptive WINdowing (*ADWIN*) (BIFET; GAVALDÀ, 2007) é um dos algoritmos de detecção de mudança mais populares da literatura. O algoritmo mantém

uma janela adaptativa W de valores reais e realiza todas as partições possíveis em duas sub-janelas W_1 e W_2 para detectar *drifts*. Uma mudança é alertada pelo algoritmo quando duas sub-janelas tem médias distintas de acordo com um limite ϵ derivado do teorema de Hoeffding (HOEFFDING, 1963). Após detecção, W_1 é descartado e $W = W_2$.

O limite ϵ aplicado pelo algoritmo *ADWIN* é explicado a seguir. Dado n_1 and n_2, o número de elementos de W_1 e W_2, respectivamente, n como a soma $n_1 + n_2$, m como a média harmônica de $\{n_1, n_2\}$, σ^2_W a variância dos valores da janela W, δ um nível de confiança, ϵ é apresentado na Equação 2.7.

$$\epsilon = \sqrt{\frac{2}{m}\sigma^2_W \ln\left(\frac{2n}{\delta}\right)} + \frac{2}{3m}\ln\left(\frac{2n}{\delta}\right) \tag{2.7}$$

O pseudocódigo do algoritmo *ADWIN* é apresentado no Algoritmo 2. Essa versão do algoritmo é computacionalmente custosa, visto que o algoritmo deve verificar todas 2 sub-partições da janela W. Para ser um algoritmo eficiente em cenários de fluxo de dados, os autores propõem uma versão com uma técnica de histograma exponencial (DATAR et al., 2002). Essa estrutura de dados permite que o ADWIN mantenha estimativas de estatísticas com compressão de dados e complexidade de memória $O(\log W)$.

Algoritmo 2 *ADWIN* (BIFET; GAVALDÀ, 2007)

Input: W : uma janela adaptativa,
S : uma *Stream* de dados
1: **for** cada valor $v \in S$ **do**
2: $W \leftarrow W \cup \{v\}$ (i.e, adicionar valores a janela W)
3: **for** cada partição de W em W_1 e W_2 **do**
4: **while** $|\overline{W_1} - \overline{W_2}| \geq \epsilon$ **do**
5: retire elementos da cauda de W
6: **end while**
7: **end for**
8: **end for**

2.5.4 *Hoeffding Drift Detection Method (HDDM)*

Em (FRÍAS-BLANCO et al., 2015), os autores propõem a utilização do *bound* de Hoeffding com níveis de confiança similar ao método em (GAMA et al., 2004). Níveis de confiança são aplicados para determinação de momentos de

warning e *drift*, tendo níveis de confiança 95% e 99%, respectivamente. Diferentemente de (BIFET; GAVALDÀ, 2007), onde uma janela adaptativa com uma abordagem de histograma exponencial determina 2 janelas para realização de comparações entre dados antigos e mais recentes para detecção de *drifts*, duas janelas para comparação de médias para determinação de mudança de conceito são atualizadas mediante a observação de novos valores mínimos de média. Denota-se $\overline{X_{cut}}$ e $\overline{Y_{n-cut}}$ como sendo as médias observadas entre observações $\{i, \ldots, cut\}$ e $\{cut+1, \ldots, n\}$, respectivamente, cut o momento de corte determinado pela observação de novos valores mínimos de média mais o *bound* no intervalo $\{i, \ldots, n\}$ $\left(\overline{X_n} + \epsilon_{X_n}, \epsilon_{X_n} = \sqrt{\frac{\log 1/\delta}{2|X_n|}}\right)$. Caso novos valores mínimos de $\overline{X_n} + \epsilon_{X_n}$ sejam observados, Y_{n-cut} é resetado e $X_{cut} = X_n$, similar como ocorre no algoritmo *DDM* (GAMA et al., 2004), que monitora valores mínimos. *Warning* são alertados caso $\overline{X_{cut}} - \overline{Y_{n-cut}} \geq \epsilon_{X_n}$ com nível de confiança $1 - \delta = 0.95$, e *drifts* são alertados caso $\overline{X_{cut}} - \overline{Y_{n-cut}} \geq \epsilon_{X_n}$ com nível de confiança $1 - \delta = 0.99$. Os autores também propõem atribuir uma ponderação das observações, sendo observações mais recentes aquelas com maior peso.

2.5.5 *Reactive Drift Detection Method (RDDM)*

Em (BARROS et al., 2017), os autores propõem o algoritmo *RDDM* (*Reactive Drift Detection Method*), que contorna problemas presentes no algoritmo *DDM* em detecções de mudança de conceito que ocorrem lentamente por 5um conceito ter um grande número de instâncias em estado sem *drift* ou *warning*, visto que valores de média tendem a não ter grandes alterações enquanto o número de amostras aumenta, principalmente quando o algoritmo recebe somente entradas binárias. Caso um conceito tenha um grande número de instâncias (um hiper-parâmetro dado pelo usuário), em que o algoritmo DDM não alertou nenhuma mudança, o recálculo das estatísticas mensuradas pelo algoritmo é realizado com as instâncias mais recentes (hiper-parâmetro dado pelo usuário de mínimo de instâncias que uma mudança de conceito pode ser alertada pelo algoritmo, já presente no DDM). Outra alteração em relação ao DDM é a inclusão de um limite de instâncias que o algoritmo está em período de *warning*. Caso um algoritmo esteja em estado de *warning* por um período de instâncias (hiper-parâmetro definido pelo usuário), um *drift* é alertado pelo algoritmo.

2.5.6 *McDiarmid Drift Detection Method (MDDM)*

Em (PESARANGHADER; VIKTOR; PAQUET, 2018), os autores propõem estratégias de detecção de *drift* baseadas no *bound* de McDiarmid (MCDIARMID, 1989). O algoritmo recebe como entrada valores binários, indicando se um classificador acertou (1) ou errou (0). Valores de média ponderada e valor máximo de média ponderada observada são monitorados em janelas deslizantes, sendo os valores mais recentes considerados com maior peso. Três estratégias são aplicadas para ponderar os valores em janelas. Dado o peso w_i da i-ésima observação:

1. **Aritmética:** $w_i = 1 + (i-1) \cdot d$, em que d é a diferença entre 2 pesos consecutivos.
2. **Geométrica:** $w_i = r^{i-1}$, em que r é o fator de aumento (hiper-parâmetro dado pelo usuário).
3. **Euler:** $w_i = e^{\lambda(i-1)}$, em que λ é um hiper-parâmetro definido pelo usuário.

Com base no modelo PAC (MITCHELL, 1997), a acurácia deve aumentar ou se manter estável. Caso contrário, a possibilidade de *drifts* ocorrerem são aparentes (GAMA et al., 2004). Em outras palavras, a possibilidade de um *drift* ocorrer aumenta caso os valores de média ponderada máxima não é alterada e os valores de média ponderada decrescem. Portanto os autores aplicam o teorema de McDiarmid da seguinte forma para determinar se houve uma mudança de conceito. Dado μ_{max}, a média ponderada máxima observada e μ_t a média ponderada na observação t e nível de confiança δ, uma mudança é alertada caso $\mu_{max} - \mu_t \geq \epsilon$, sendo ϵ definido na Equação 2.8.

$$\epsilon = \sqrt{\frac{\sum_{i=1}^{n} v_i^2}{2} \ln \frac{1}{\delta}} \qquad v_i = \frac{w_i}{\sum_{i=1}^{n} w_i} \tag{2.8}$$

2.6 Árvores de Decisão Incrementais

Árvores de decisão são algoritmos bem conhecidos e amplamente usados em aprendizado de máquina em lote, pela sua utilidade, interpretabilidade, e robustez em vários problemas (COSTA; PEDREIRA, 2022). Esses algoritmos consistem em realizar múltiplas separações lineares que visam separar em nós

folhas instâncias que pertencem a uma mesma classe, i.e., ter nós folhas com maior nível de pureza. Para isso, vários algoritmos foram desenvolvidos que aplicam estratégias para escolher o atributo e seu valor que resultam em ramificação com os nós folhas mais puros possíveis.

Por exemplo, o algoritmo C4.5 (QUINLAN, 1992) aplica estratégias de teoria da informação (SHANNON, 1948) para quantificar o nível de pureza presente em um conjunto de dados em um nó folha (Entropia, Equação 2.9) e um comparativo entre a pureza do nó folha e a pureza dos nós folhas resultantes em caso de ramificação (Ganho de informação, Equação 2.10).

$$\text{Entropia}(N) = -\sum_{y \in N} p(y) \log_2(p(y)) \tag{2.9}$$

$$\text{Ganho de Informação}(N) = \text{Entropia(N)} - \sum_{r \in R} \frac{|r|}{|N|} \text{Entropia}(r) \tag{2.10}$$

onde denota-se N o conjunto de instâncias presentes em um nó folha, $p(y)$ como a proporção de exemplos de uma classe c em relação ao número de exemplos presentes em um nó, R como o conjunto de nós folhas resultantes em caso de ramificação e $|\cdot|$ como a cardinalidade de um conjunto. O algoritmo seleciona o atributo que maximize o ganho de informação em caso de ramificação, visando ter os nós folhas mais puros comparados ao nó folha anterior a ramificação.

Em (BREIMAN, 1984), o nível de impureza de um nó é mensurado por Gini Index (Equação 2.11) e a qualidade de uma ramificação é quantificada por uma soma ponderada da impureza (Equação 2.12).

$$\text{GiniIndex}(N) = 1 - \sum_{y \in N} p(y)^2 \tag{2.11}$$

$$\text{Gini}_{pureza}(N) = 1 - \sum_{r \in R} \frac{|r|}{|N|} \text{GiniIndex}(r) \tag{2.12}$$

Por serem modelados para o ambiente *batch*, esses algoritmos realizam ramificações em um conjunto de treinamento e realizam predições em um conjunto de teste. Para o ambiente *stream*, modelar uma árvore para cada instância que chega pode ser custoso pelo alto número de instâncias e pela verificação de todos os valores observados de atributos para cálculo de métricas de pureza, sendo inviável para mineração de fluxo de dados.

Para contornar esse problema, algoritmos de árvore de decisão com a capacidade de receber instâncias incrementalmente foram propostos na literatura e serão detalhados a seguir.

2.6.1 *Very Fast Decision Tree (VFDT)*

Em (DOMINGOS; HULTEN, 2000), os autores propuseram o algoritmo *Hoeffding Tree* e o sistema *VFDT* (*Very Fast Decision Tree*), que torna *Hoeffding Tree* um algoritmo eficiente para mineração de fluxo de dados. O algoritmo propõe uma estratégia de ramificação baseada no teorema de Hoeffding (HOEFFDING, 1963).

A cada n observações disponíveis em um nó folha (hiper-parâmetro dado pelo usuário conhecido como *Grace Period*, GP), uma ramificação é realizada caso:

$$\Delta G(A_a) - \Delta G(A_b) > \sqrt{\frac{R^2 \log\left(\frac{1}{\delta}\right)}{2n_l}} \tag{2.13}$$

sendo $\Delta G(\cdot)$ uma métrica de pureza como 2.10 ou 2.12, A_a o atributo com o maior valor de métrica de pureza, A_b o atributo com o segundo maior valor de métrica de pureza, R o intervalo da variável (no caso de ganho de informação 2.10 $\log_2(c)$, sendo c o número de classes), δ o nível de confiança de A_a ser um atributo apropriado para realização de uma ramificação no nó e n_l o número de observações contidas no nó.

O Algoritmo 3 apresenta o pseudocódigo do algoritmo *VFDT*. Além de realizar ramificações periodicamente considerando o número de observações em um nó folha (Algoritmo 3, linha 7), outros elementos compõe o sistema *VFDT*.

O *tie threshold* é utilizado quando em um nó folha a diferença entre $\Delta G(A_a)$ e $\Delta G(A_b)$ seja próxima de zero. Nestes casos, muitos exemplos serão necessários para realizar uma ramificação com confiança alta. Para o modelo ser atualizado em situações como essa, o *tie threshold* (valor dado pelo usuário) precisa ser superior à desigualdade de Hoeffding (Algoritmo 3, linha 12). Nestes casos uma ramificação é realizada em A_a.

Dado d o número de atributos, v o maior número de valores por atributo, c o número de classes e l o número de nós folhas, a complexidade de espaço do algoritmo *Hoeffding Tree* é $O(ldcv)$.

Algoritmo 3 *VFDT* (DOMINGOS; HULTEN, 2000)

Input: S: um fluxo de dados,
X: um conjunto de vetores de características,
$\Delta G(\cdot)$: uma métrica de pureza,
δ : um menos a probabilidade desejada para escolha de um atributo para ramificação em qualquer nó folha,
GP: *Grace Period* (frequência de observações presentes em um nó folha para realizar uma tentativa de ramificação)
τ : *Tie threshold*
Output: DT: uma árvore de decisão

Seja DT uma árvore com um único nó l_1 (nó raiz)
Inicialize $n_{l_1} \leftarrow 0$ ▷ número de observações no nó
for cada exemplo $(\vec{x}, y) \in S$ **do**
    $l \leftarrow$ percorra DT com $(\vec{x}, y)$
    Classifique $\vec{x}$ com a classe majoritária presente em l
    Atualize as estatísticas em l com $(\vec{x}, y)$
    $n_l \leftarrow n_l + 1$
    if n_l mod $GP = 0 \wedge \neg$(l contêm somente exemplos de uma classe) **then**
        Compute $\Delta G(A_i)$ para cada $A_i \in A_l$ armazenados em l
        Seja A_a o atributo com maior ΔG
        Seja A_b o atributo com segundo maior ΔG
        Seja $\epsilon \leftarrow \sqrt{\frac{R^2 \log(\frac{1}{\delta})}{2n_l}}$ (2.13)
        if $(\Delta G(A_a) - \Delta G(A_b) > \epsilon \vee \tau > \epsilon) \wedge A_a \neq \emptyset$ **then**
            Integre a l nós folhas que se ramificam em A_a
            for cada nó folha l_i resultante da ramificação em A_a **do**
                Inicialize $n_{l_i} \leftarrow 0$
            end for
        end if
    end if
end for

Outro aspecto importante do algoritmo *VFDT* é o controle de uso de memória, visto que é inviável armazenar potencialmente infinitas instâncias (Algoritmo 3, linha 4). Em (KIRKBY, 2007), os autores propõem uma rotina de gerenciamento de memória para *VFDT* com desativação de nós folhas (i.e. nós folhas não irão realizar mais splits e armazenar instâncias) caso o uso de memória de uma árvore ultrapasse um limiar especificado por um usuário. A verificação do uso de memória é realizado a cada vez que uma ramificação será realizada pelo *VFDT*.

Em (GAMA; ROCHA; MEDAS, 2003), os autores adicionam o suporte de *Naive Bayes* a nós folhas, substituindo a estratégia de predição via classe majoritária (Algoritmo 3, linha 6) pela predição realizada por um *Naive Bayes*. Em (KIRKBY, 2007), os autores aplicam uma estratégia adaptativa de seleção entre predição por classe majoritária ou *Naive Bayes* baseado na maior acurácia observada por essas abordagens no nó folha.

Essas melhorias ao algoritmo *VFDT* são comumente utilizadas na literatura e serão utilizados para experimentações realizadas nesse trabalho.

2.6.2 *Concept-adapting Very Fast Decision Tree (CVFDT)*

Em (HULTEN; SPENCER; DOMINGOS, 2001), os autores propõem estratégias para lidar com esquecimento de instâncias antigas e tratar instâncias de dados mais recentes como mais importantes. A ideia principal do algoritmo é ter uma janela deslizante com as instâncias mais recentes e que as estatísticas referentes à árvore estejam consistentes em comparação com a janela deslizante.

Devido ao crescimento incremental da árvore, há possibilidade que ramificações realizadas anteriormente não sejam adequadas considerando as instâncias mais recentes. Periodicamente, o algoritmo verifica se novos atributos compõe uma ramificação melhor do que aquelas realizadas anteriormente em nós onde a Equação 2.13 (desigualdade de Hoeffding) não é mais verdadeira, construindo sub-árvores a partir desses nós. Caso a sub-árvore construída possua acurácia superior à sub-árvore atual nas próximas instâncias que chegam, a sub-árvore construída substitui a atual.

2.6.3 *Hoeffding Option Tree (HOT)*

Árvores de decisão visam maximizar a pureza em ramificações localmente, sendo que não tem perspectiva de futuras ramificações, podendo resultar em uma decisão que não é a melhor a longo prazo. Isso os caracteriza como um classificador instável, sendo que poucas mudanças nos dados podem ocasionar na indução de uma árvore completamente diferente (KOHAVI; KUNZ, 1998).

Nós opções tem como objetivo reduzir a instabilidade de árvores de decisão ao permitir que uma instância possa percorrer mais de um caminho de decisão (BUNTINE, 1992; KOHAVI; KUNZ, 1998). Por exemplo, se uma instância alcança um nó opção na Figura 2.8, a instância irá percorrer todos os caminhos de seus nós filhos $\{A_3 \leq 0,5, A_5 \leq 0,5\}$. Após as instâncias chegarem em um ou mais nós folhas, o voto é combinado por voto majoritário, como em (KOHAVI; KUNZ, 1998), ou voto ponderado, em que a probabilidade por classe dos nós folhas é somada (BUNTINE, 1992).

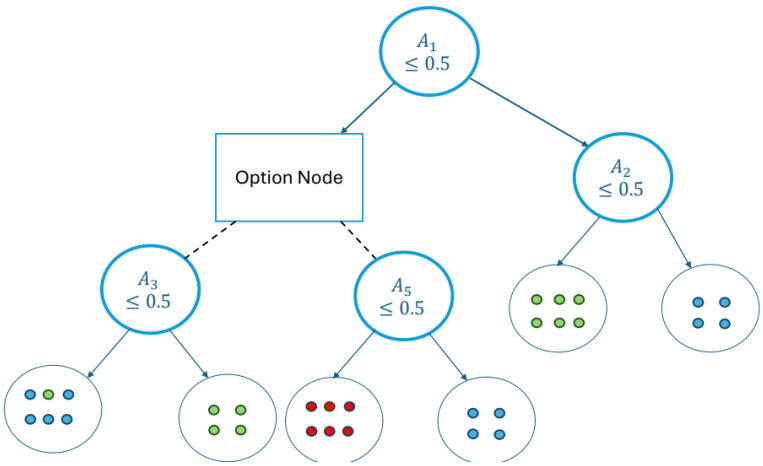

Figura 2.8: Ilustração de uma árvore de decisão com nós opção

Em (PFAHRINGER; HOLMES; KIRKBY, 2007), os autores propõem integrar nós opções em *Hoeffding Tree* (DOMINGOS; HULTEN, 2000). A integração de nós opções é similar ao mecanismo de ramificação, em que periodicamente o algoritmo realiza uma tentativa de adição de nós opções. Em um nó não terminal, a cada GP instâncias uma tentativa de ramificação é realizada considerando o teorema de Hoeffding como na Equação 2.14. Para um nó opção ser adicionado, a diferença entre o mérito da ramificação de um novo nó ($\Delta G(X_{cand})$) em um atributo ainda não utilizado no nó e o maior mérito de um atributo presente no nó ($\Delta G(X_{max})$) deve ser maior que o limiar de Hoeffding, justificando a adição

de um nó.

$$\Delta G(X_{cand}) - \Delta G(X_{max}) \geq \sqrt{\frac{R^2 \log(\frac{1}{\delta'})}{2n_l}} \tag{2.14}$$

Hoeffding Option Tree, como em (KOHAVI; KUNZ, 1998), definem o número máximo de nós filhos de um nó opção, para evitar um custo computacional elevado e *overfitting*, sendo definido como 5, pois não obteve acurácia superior com valores maiores de máximo número de nós filhos.

2.6.4 *Hoeffding Adaptive Tree (HAT)*

Em (BIFET; GAVALDÀ, 2009), os autores propõem o algoritmo *Hoeffding Adaptive Tree* (HAT), que implementa estratégias adaptativas para monitorar a qualidade preditiva da árvore e realizar alterações, caso necessário. Cada nó tem um detector de mudança *ADWIN* (BIFET; GAVALDÀ, 2007), que detecta degradações de performance em instâncias que o perpassam. Originalmente o algoritmo foi idealizado para lidar com cenários na presença de mudança de conceito, abordado na Seção 2.4. Caso o detector alerte uma mudança na qualidade preditiva das instâncias que perpassam os nós, uma nova sub-árvore é construída. Caso a acurácia da sub-árvore seja superior àquela da sub-árvore presente, todo o ramo referente ao nó que detectou a árvore é substituído por uma nova sub-árvore alternativa construída.

2.6.5 *Extremely Fast Decision Tree (EFDT)*

Em (MANAPRAGADA; WEBB; SALEHI, 2018), os autores propõem o algoritmo *EFDT* (*Extremely Fast Decision Tree*), uma extensão ao algoritmo *VFDT*. O termo extremamente rápido se refere a uma convergência que ocorre de forma mais rápida que *Hoeffding Trees* ao resultado de árvores de decisão em *batch*, como os autores alegam. A primeira alteração ao algoritmo *VFDT* é referente ao uso da desigualdade de Hoeffding. Ao invés de comparar métricas de pureza dos dois atributos com os maiores valores de métrica de pureza, a comparação é feita entre o atributo com maior valor de métrica de pureza e o caso onde não há *split* ($\Delta G(A_\emptyset) = 0$). A Equação 2.15 é utilizada como condição para realização de um *split* em A_a.

$$\Delta G(A_a) - \Delta G(A_\emptyset) > \sqrt{\frac{R^2 \log(\frac{1}{\delta})}{2n_l}} \tag{2.15}$$

Outra adição presente no algoritmo é a reavaliação de ramificações. Quanto maior a quantidade de instâncias observadas em um nó, aumenta-se possibilidade de haver um atributo melhor para realizar um split comparado a um split anterior. Por um lado, mais dados devem ser armazenados como o ganho de informação do atributo do split realizado em cada nó, e a reavaliação de *splits* adiciona um custo computacional adicional ao avaliar se algum split é melhor do que o realizado anteriormente. Por outro lado, essa abordagem possui qualidade preditiva superior ao algoritmo *Hoeffding Tree* em inúmeros *datasets* reais e sintéticos. Como nós folhas, a cada n instâncias que perpassam um nó, uma reavaliação de *split* ocorre. Para reavaliação de *splits*, o comparativo é realizado entre o valor da métrica de pureza de A_a em caso de substituição da ramificação da árvore por uma nova ramificação e o valor de métrica de pureza na qual o nó foi realizado o split. Uma *branch* da árvore é substituída por uma nova ramificação caso:

$$\Delta G(A_a) - \Delta G(A_{atual}) > \sqrt{\frac{R^2 \log(\frac{1}{\delta})}{2n_l}} \tag{2.16}$$

em que A_{atual} é o atributo em que uma ramificação foi realizada anteriormente no nó de reavaliação.

2.7 Inconsistência presente em *Hoeffding Tree*

Em (HOEFFDING, 1963), os autores propuseram o teorema a seguir. Se $\phi_1, \phi_2, \ldots, \phi_n$ são variáveis independentes e $a_i \leq \phi_i \leq b_i$ $(i = 1, 2, \ldots, n)$, para todo $t > 0$:

$$P[\overline{\phi} - \mu \geq t] \leq e^{\frac{-2n^2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}} \tag{2.17}$$

que é um limite superior em relação à diferença entre a média $(\overline{\phi})$ e valor esperado (μ) de variáveis independentes.

Dado R o escopo de uma métrica de pureza, os autores em (DOMINGOS; HULTEN, 2000) derivaram, dado um intervalo de confiança $P[\overline{\phi} - \mu \geq t] \leq \delta$,

que:

$$\overline{\phi} - \mu > \sqrt{\frac{R^2 \log(\frac{1}{\delta})}{2n_l}} \tag{2.18}$$

e aplicam métricas de pureza (2.13), sendo $\overline{\phi} = \Delta G(A_a)$ e $\mu = \Delta G(A_b)$.

Contudo, em (RUTKOWSKI et al., 2013), os autores evidenciam que métricas de pureza não são médias (como $\overline{\phi}$) , e como o teorema de Hoeffding é a probabilidade de desvio padrão da média e seu valor esperado, sua aplicação para determinação de um split ser adequado é errônea. Portanto, toda condição de ramificação baseada em (DOMINGOS; HULTEN, 2000) apresenta uma inconsistência. A partir de (RUTKOWSKI et al., 2013), várias alternativas foram propostas na literatura como condição de ramificação em árvores de decisão incrementais, sendo algumas delas apresentadas na Seção 2.8.

2.8 Condições de Ramificação em Árvores de Decisão Incrementais

A Tabela 2.1 apresenta condições de ramificação apresentadas na literatura, sendo ϵ o valor em que a condição $\Delta G(A_a) - \Delta G(A_b) > \epsilon$ deve ser verdadeira para ocorrer uma ramificação. São apresentadas alternativas propostas na literatura para incorreção da aplicação da desigualdade de Hoeffding e extensão incorreta do uso da desigualdade de Hoeffding. Além de (MANAPRAGADA; WEBB; SALEHI, 2018), a mudança em relação ao uso da desigualdade de Hoeffding é em ϵ, ou seja, o lado direito da desigualdade.

Tabela 2.1: Condições de ramificação apresentadas na literatura.

Referência	Ganho de Informação	Gini Index	Possui prova
(DOMINGOS; HULTEN, 2000)	$\epsilon = \sqrt{\frac{R^2 \log(1/\epsilon)}{2n_l}}$	$\epsilon = \sqrt{\frac{R^2 \log(1/\epsilon)}{2n_l}}$	✘
(RUTKOWSKI et al., 2013)	$\epsilon = C_{Gain}(K,n)\sqrt{\frac{\ln(1/\epsilon)}{2n_l}}$ $C_{Gain}(K,n) = 6(K\log_2(en_l) + \log_2(2n_l)) + 2\log_2(K)$	$\epsilon = 8\sqrt{\frac{\log(1/\epsilon)}{2n_l}}$	✓
(ROSA; CESA-BIANCHI, 2015)	$\epsilon = \sqrt{8\ln(4/\delta)}\frac{ln(n_l)}{\sqrt{n_l}}$	-	✓
(ROSA; CESA-BIANCHI, 2015)	$\epsilon = \sqrt{8\ln(4/\delta)}\frac{ln(n_l)}{\sqrt{n_l}} + \frac{4}{n_l}$	-	✓
(JAWORSKI; DUDA; RUTKOWSKI, 2018)	-	$\epsilon = \sqrt{\frac{8\ln(1/\delta)}{n_l}}$	✓
(JAWORSKI; DUDA; RUTKOWSKI, 2018)	-	$\epsilon = 0.5\sqrt{\frac{R^2\log(1/\delta)}{2n_l}}$	✘
(MANAPRAGADA; WEBB; SALEHI, 2018)	$\Delta G(A_a) - \Delta G(A_\emptyset) > \sqrt{\frac{R^2\log(1/\delta)}{2n_l}}$	$\Delta G(A_a) - \Delta G(A_\emptyset) > \sqrt{\frac{R^2\log(1/\delta)}{2n_l}}$	✘

Além de evidenciar o erro do uso da desigualdade de Hoeffding em condições de ramificação, os autores em (RUTKOWSKI et al., 2013) propõem uma alternativa baseada no teorema de McDiarmid (MCDIARMID, 1989). O teorema de McDiarmid é uma generalização do teorema de Hoeffding para funções arbitrárias e (RUTKOWSKI et al., 2013) derivam condições de ramificação tanto para ganho de informação quanto para Gini Index, como apresentado na Tabela 2.1, sendo K o número de classes presente no nó folha no momento de tentativa de ramificação. Porém, como evidenciado pelos próprios autores em (RUTKOWSKI; JAWORSKI; DUDA, 2020), a desigualdade derivada para ganho de informação tem valores muito elevados, não sendo eficiente por ter menos atualizações do modelo comparado a *Hoeffding Trees*.

Para isso, os autores em (ROSA; CESA-BIANCHI, 2015) propõem uma melhora no limite de McDiarmid para ganho de informação. Adicionalmente, os autores propõem alternativas para redução do viés de árvores de decisão, com adição do termo de viés no limite (*bound*) ($\frac{4}{n_l}$, disposto na Tabela 2.1).

Em (JAWORSKI; DUDA; RUTKOWSKI, 2018), os autores propõem melhorias na desigualdade de McDiarmid com uso de Gini Index como métrica de pureza. Além disso, os autores também apresentam heurísticas na desigualdade de Hoeffding para induzir crescimento na árvore mais rapidamente, multiplicando o *bound* por 0.5.

Apesar de todos os trabalhos apresentarem alternativas para o uso errôneo da desigualdade de Hoeffding para realização de ramificações, os trabalhos não têm foco na parte experimental para aumento da qualidade preditiva das árvores de decisão. E como evidenciado em (JAWORSKI; DUDA; RUTKOWSKI, 2018), a maioria dos *bounds* propostos na literatura são da forma $c\sqrt{\frac{\ln(1/\delta)}{n_l}}$, sendo c uma constante, e $c = \frac{R}{\sqrt{2}}$ em (DOMINGOS; HULTEN, 2000), $c = \frac{8}{\sqrt{2}}$ em (RUTKOWSKI; JAWORSKI; DUDA, 2020), entre outros. Destarte, os trabalhos utilizam estratégias similares para avaliação de ramificações e a Seção 3 apresenta novas estratégias de ramificação.

2.9 *Ensembles* para Mineração de Fluxo de Dados

Algoritmos baseados em *ensembles* englobam o desenvolvimento de técnicas que combinam a predição de dois ou mais classificadores monolíticos, e a diversidade desses classificadores é um fator importante para o *ensemble* ter boa

qualidade preditiva (KUNCHEVA; WHITAKER, 2003).

Ensembles homogêneas induzem a diversidade de seus classificadores ao criar vários modelos do mesmo tipo que são treinados em diferentes subconjuntos de treinamento, constituindo o estado-da-arte para o problema de classificação de dados tabulares em fluxo de dados (KRAWCZYK et al., 2017).

A maioria dos métodos de *ensembles* modelados para classificação de fluxo de dados são adaptações de *ensembles* projetados para mineração de lotes de dados, sendo apresentados a seguir.

2.9.1 *Online Bagging*

O algoritmo *Online Bagging* (OZA; RUSSELL, 2001) é um algoritmo que adapta o algoritmo *Bagging* (*Bootstrap Aggregation*) (BREIMAN, 1996) para o cenário de fluxo de dados. Na fase de treinamento, *Bagging* realiza amostragem com reposição para cada classificador da *ensemble*. *Online Bagging* simula amostragem com reposição incrementalmente, em que o algoritmo determina o número de cópias com as quais cada classificador deve treinar cada instância. Cada classificador recebe k cópias de uma instância, em que k é uma variável que segue uma distribuição de $Poisson(\lambda = 1)$. Destarte, um classificador recebe para treino uma ou mais cópias de uma instância aproximadamente 63% das vezes, dado que $P[k > 0] = 1 - P[k = 0] = 1 - \frac{1}{e \times k!} = 1 - \frac{1}{e \times 0!} \approx 63\%$. A predição final da *ensemble* é composta pelo voto majoritário simples, dependendo de qual classe maioria dos *base learners* realizaram predição.

2.9.2 *Online Boosting*

Os autores em (OZA; RUSSELL, 2001) também adaptam o algoritmo *AdaBoost* (SCHAPIRE, 1999) para o ambiente *stream*. *AdaBoost* é um algoritmo meta-heurístico onde a taxa de erro de um classificador influencia na criação de outros classificadores, favorecendo o treino de instâncias incorretamente preditas pelos classificadores.

Online Boosting difere em alguns pontos do algoritmo *AdaBoost*. O tamanho da ensemble é fixo, i.e., nenhum classificador adicional é integrado na *ensemble* e os classificadores influenciam o treino de outros classificadores por instância ao invés um conjunto de treinamento. Na chegada de uma instância, λ tem valor inicial 1 e o classificador c_i é treinado com $k = Poisson(\lambda)$ cópias da

instância. Caso c_i acerte a classificação da instância, o valor de λ para treino do classificador c_{i+1} decresce. Caso o classificador acerte a classificação da instância, o valor de λ para treino do classificador c_{i+1} aumenta. Assim, o algoritmo favorece o treinamento de instâncias em que os classificadores mais erram, dado que $\mathbb{E}[Poisson(\lambda)] = \lambda$. Para cada classificador, o algoritmo mantém taxas de erro e acerto que influenciam o quanto λ irá aumentar ou decrescer. O leitor interessado em como o aumento e declínio dos valores de λ ocorrem é referenciado a (OZA; RUSSELL, 2001). A predição final da *ensemble* é a soma ponderada das probabilidades por classe dos *base learners* pela taxa de erro do classificador.

2.9.3 Integração do *ADWIN* em *Ensembles* para Detecção de *Concept Drift*

Em (BIFET et al., 2009) os autores propõem a integração do algoritmo *ADWIN* para as *ensembles* apresentadas em (OZA; RUSSELL, 2001) serem adaptativas a cenários com mudança de conceito. Para cada classificador da *ensemble*, um detector *ADWIN* é criado e monitora taxas de erro dos algoritmos. Caso algum detector dos classificadores alerte uma mudança de conceito, o classificador com pior taxa de erro (estimado pelo algoritmo *ADWIN*) é reinicializado. A predição da *ensemble* é realizada com voto majoritário simples, como em *OzaBag*.

2.9.4 *Leveraging Bagging*

Em (BIFET; HOLMES; PFAHRINGER, 2010), os autores propõem o algoritmo *Leveraging Bagging*, que propõe melhorias ao algoritmo *Online Bagging* com *ADWIN* (BIFET et al., 2009). O mecanismo para adaptação a *concept drift* é o mesmo que em (BIFET et al., 2009), porém os autores realizam amostragem com reposição via $Poisson(\lambda = 6)$ ao invés de $Poisson(\lambda = 1)$. Dado que $P[k > 0|\lambda = 6] \approx 99.7$, há uma grande probabilidade dos classificadores receberem pelo menos uma cópia para treino. A predição da *ensemble* é realizada com voto majoritário simples, como em *OzaBag*.

Isso faz com que os classificadores fiquem mais especializados dado que o número de cópias que um classificador recebe para voto é maior, porém isso causa um custo adicional computacional a *ensemble*.

Os autores também propõem utilizar *error output codes* juntamente ao *Leve-*

raging Bagging, em que classificadores binários por classe são criados, e a fusão de voto final da *ensemble* é a soma dos resultados dos classificadores por classe. Esse componente adiciona um custo, visto que o número de classificadores da *ensemble* é multiplicado pelo número de classes, mas apresenta em alguns casos acurácia superior ao *Leveraging Bagging* padrão, mesmo que o *Leveraging Bagging* padrão tenha apresentado melhor *ranking* em acurácia comparado ao *Leveraging Bagging* com *error output codes* no estudo original (BIFET; HOLMES; PFAHRINGER, 2010).

2.9.5 *BOLE*

Estendendo o algoritmo *Online Boosting*, em (BARROS; SANTOS; JÚNIOR, 2016), os autores propõem o algoritmo *Boosting-like Online Learning Ensemble* (*BOLE*). O treinamento começa com $\lambda = 1$ e os classificadores são ordenados pela taxa de predições corretas (considerando pesos de classificação corretos e incorretos (OZA; RUSSELL, 2001)). O classificador com a pior taxa de predições corretas é treinado primeiro. Se esse classificador obtiver uma classificação correta, o melhor classificador ainda não treinado (taxa de predições corretas) recebe um λ alterado para treinamento. Caso contrário, o pior classificador ainda não treinado recebe um λ alterado para treinamento. O *BOLE* aumenta significativamente λ quando muitos *base learners* classificam incorretamente as instâncias, pois os piores *base learners* influenciam uns aos outros até que um *base learners* faça a predição correta. O desempenho da *ensemble* em geral influencia os valores de λ. Como em *OzaBoost*, a predição é o voto ponderado pela taxa de erro dos *base learners*.

2.9.6 *Adaptive Random Forest*

Em (GOMES et al., 2017b), os autores propõem o algoritmo *Adaptive Random Forests*, que adapta o algoritmo *Random Forests* (BREIMAN, 2001) para mineração de fluxo de dados. A cada tentativa de *split* realizada por uma *Hoeffding Tree*, somente um conjunto aleatório de atributos é considerado. E como em (BIFET; HOLMES; PFAHRINGER, 2010), amostragem com reposição é realizada com $Poisson(\lambda = 6)$. Para lidar com ambientes com *Concept Drift*, cada classificador da *ensemble* tem dois detectores de mudança: o primeiro para *warnings* e o segundo para *drifts*. Se o detector de *warnings* detecta uma mudança, um novo

classificador é criado e treinado em *background*. Caso o detector de *drifts* alerte uma mudança, o classificador treinado em *background* substitui o classificador base. Além disso, os votos dos classificadores são ponderados por acurácia, sendo que esta é calculada desde o momento de criação da árvore.

2.9.7 *Streaming Random Patches*

Em (GOMES; READ; BIFET, 2019) os autores propõem uma versão do algoritmo *Random Patches* (LOUPPE; GEURTS, 2012) para o cenário de fluxo de dados. O método *Random Patches* combina amostragem com reposição de instâncias com seleção de atributos aleatório para cada classificador. Em (GOMES; READ; BIFET, 2019), os autores utilizam os mesmos mecanismos que em (GOMES et al., 2017b) para lidar com *Concept Drift*, amostragem com reposição e pesagem dos votos. Em (GOMES et al., 2017b), cada tentativa de *split* é realizada com atributos aleatórios definidos por nó, enquanto em (GOMES; READ; BIFET, 2019) todas as tentativas de *split* avaliadas são com um conjunto de atributos aleatórios definido na criação do classificador. Como em *Adaptive Random Forest*, os votos dos classificadores são ponderados por acurácia, sendo que esta é calculada desde o momento de criação da árvore.

2.9.8 *Adaptive Random Tree Ensemble*

Em (PAIM; ENEMBRECK, 2024a) os autores propõem o algoritmo *Adaptive Random Tree Ensemble* (ARTE) para classificação de fluxo de dados. Como a ensemble *Adaptive Random Forest* (ARF), ARTE possui um mecanismo de seleção de atributos para ramificação a nível de nó folha. Porém, diferentemente do ARF, a porcentagem de atributos selecionados randomicamente por nó folha não é fixo, sendo que a porcentagem de atributos selecionados randomicamente por nó folha em todas as árvores segue uma distribuição normal. Os autores também incorporam a seleção do melhor atributo para ramificação por ponto de corte aleatório, como proposto em (GEURTS; ERNST; WEHENKEL, 2005). Dado um atributo numérico, um valor aleatório é selecionado entre o valor máximo e mínimo do atributo para realizar a ramificação, sendo o mérito da ramificação avaliado a partir do corte aleatório. O ARTE também possui um mecanismo de seleção dinâmica de classificadores, sendo que somente os classificadores que possuem acurácia melhor que a acurácia média dos classificadores nas instâncias

mais recentes realizam voto. Os classificadores de ARTE possuem detectores de mudança de conceito explícitos, que ao alertarem uma mudança de conceito, provocam a criação de uma nova árvore de decisão que substitui a árvore ativa. Como em *Adaptive Random Forest*, os votos dos classificadores são ponderados por acurácia, sendo que esta é calculada desde o momento de criação da árvore.

2.9.9 *Adaptive Regularized Ensemble*

Adaptive Regularized Ensemble (ARE) (PAIM; ENEMBRECK, 2024b) apresenta uma extensão de *Adaptive Random Forest*. ARE possui um mecanismo de seleção de instâncias, em que as árvores realizam treinamento com instâncias que realizaram predição incorreta. Porém, para evitar que o algoritmo seja treinado majoritariamente com instâncias com ruído (dado que são as instâncias que mais serão classificadas incorretamente), um fator de rejeição por classe é aplicado para treinamento de instâncias que foram classificadas corretamente. Dado um fator de rejeição ζ, ζ instâncias de uma classe não incorporam o treinamento de um classificador (instâncias são rejeitadas) até que uma instância dessa classe seja utilizada para treino. Como ARTE, ARE possui o mesmo mecanismo de seleção dinâmica de classificadores. Como em *Adaptive Random Forest*, os votos dos classificadores são ponderados por acurácia, sendo que esta é calculada desde o momento de criação da árvore.

2.10 Considerações Finais

Essa seção apresentou conceitos fundamentais de mineração de fluxo de dados, com foco especial em árvores de decisão incrementais, para a compreensão das próximas seções desse documento. Além do uso errôneo da desigualdade de Hoeffding em árvores de decisão incrementais, a verificação periódica de ramificações por modelos baseados na desigualdade de Hoeffding e McDiarmid não considera a evolução do fluxo ao longo do tempo. Mesmo quando mudanças evidentes na distribuição dos dados ou acurácia dos nós folhas ocorrem, uma ramificação ocorre somente quando o número de observações no nó atinge o *Grace Period*. Por outro lado, quando o nó folha está em período de estabilidade, o algoritmo continua periodicamente avaliando ramificações, resultando possivelmente em ramificações desnecessárias. O capítulo também apresentou uma breve revisão sobre algoritmos de *ensembles* estado-da-arte para classificação de

fluxos de dados.

O próximo capítulo descreve a proposta de um novo método que visa ter árvores de decisão incrementais com estratégias adaptativas de ramificação.

3

Método

Árvores baseadas no teorema de Hoeffding e McDiarmid utilizam um mecanismo estático e periódico de avaliação de ramificações. A árvore não é capaz de monitorar o estado do fluxo constantemente, sendo que mudanças nos dados podem ocorrer nos intervalos entre tentativas de ramificações. Além disso, mesmo quando pouca mudança ocorre, a avaliação gulosa da melhor ramificação com todos os atributos e seus valores ainda será efetuada, resultando em tempo de processamento desnecessário e pouca mudança na estrutura da árvore.

Destarte, nesse trabalho é proposto uma nova árvore de decisão incremental que constantemente avalia estatísticas de nós folhas para reagir a mudanças no fluxo somente quando necessário.

A Figura 3.1 ilustra como abordagens adaptativas de ramificação podem ser adequadas para árvores de decisão incrementais. Nesse exemplo, uma mudança na distribuição dos dados ocorre antes de atingir o *Grace Period* da *Hoeffing Tree* ao se observar que mais instâncias da classe verde mudaram a distribuição dos dados majoritariamente compostas de instâncias da classe azul para um equilíbrio entre as classes.

Os métodos de detecção de mudança de conceito mais usuais (*concept drift detectors*) monitoram aumentos significativos em métricas ($\{z|z \in \mathbb{R}\}$) de impureza e taxas de erro, fornecendo alertas quando essas variações são significativas estatisticamente. Neste trabalho, detectores de mudanças são utilizados para monitorar nós-folha de árvores de decisão incrementais, de forma que, em caso de alerta de mudança, uma tentativa de ramificação seja realizada e as taxas de impureza e erro observadas sejam mitigadas.

A Seção 3.1 introduz o algoritmo *Local Adaptive Streaming Tree (LAST)* para classificação de fluxo de dados. A subseção 3.1.1 discute as limitações da aplicação da árvore proposta na sua aplicação como *base learner* de *ensembles* e a subseção 3.1.2 propõe novos métodos para contornar as limitações descritas.

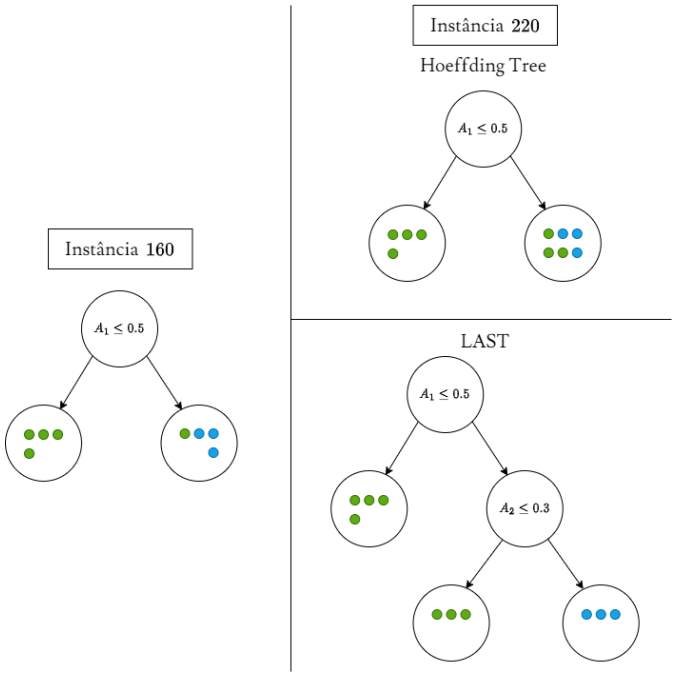

Figura 3.1: Ilustração da adaptabilidade do método proposto em comparação com *Hoeffding Trees* com *GP*= 160

Autoria: Autor, 2024.

3.1 *Local Adaptive Streaming Trees*

O algoritmo *Local Adaptive Streaming Trees* introduz um mecanismo adaptativo de ramificação de nós folhas, em que ramificações ocorrem em nós folhas da árvore afetadas por decaimento de acurácia ou mudança na distribuição dos dados (sendo escolha do usuário a utilização de detecção de decaimento de acurácia ou mudança na distribuição dos dados).

O Algoritmo 4 apresenta o algoritmo proposto *Local Adaptive Streaming Trees* (*LAST*). *LAST* cria e mantêm detectores de mudança em nós folhas (Algoritmo 4, linhas 3 e 21). No nó folha, o detector de mudança pode monitorar tanto pureza da distribuição dos dados (Algoritmo 4, linha 10) quanto taxa de erro

Algoritmo 4 *LAST*

Input: S: um fluxo de dados,
X: um conjunto de vetores de características,
$\Delta G(\cdot)$: uma métrica de pureza,
$H(\cdot)$: Função de pureza de um único conjunto de dados análogo a $\Delta G(\cdot)$,
ψ : Um algoritmo de detecção de mudança,
monitorar_erro : variável booleana. Se verdadeira, ψ tem entrada 1 para classificações incorretas e 0 para classificações corretas, caso contrário $H(\cdot)$.
Output: DT: uma árvore de decisão

1: Seja DT uma árvore com um único nó l_1 (nó raiz)
2: Inicialize $n_{l_1} \leftarrow 0$ ▷ número de observações no nó
3: Inicialize $l_{1\psi} \leftarrow \psi$ ▷ cria um detector de mudança no nó folha
4: **for** cada exemplo $(\vec{x}, y) \in S$ **do**
5: $l \leftarrow$ percorra DT com $(\vec{x}, y)$
6: Classifique $\vec{x}$ com a classe majoritária presente em l
7: Atualize as estatísticas em l com $(\vec{x}, y)$
8: $n_l \leftarrow n_l + 1$
9: **if** monitorar_erro **then**
10: Atualize l_ψ com $\mathbb{1}\{DT(\vec{x}) \neq y\}$
11: **else**
12: Atualize l_ψ com $H(l)$
13: **end if**
14: **if** l_ψ detectou mudança $\wedge \neg$(l contêm somente exemplos de uma classe) **then**
15: Compute $\Delta G(A_i)$ para cada $A_i \in A_l$ armazenados em l
16: Seja A_a o atributo com maior ΔG
17: **if** $\Delta G(A_a) > 0 \wedge A_a \neq \emptyset$ **then**
18: Integre a l nós folhas que se ramificam em A_a
19: **for** cada nó folha l_i resultante da ramificação em A_a **do**
20: Inicialize $n_{l_i} \leftarrow 0$
21: $l_{i\psi} \leftarrow \psi$ ▷ cria um detector de mudança em cada nó folha
22: **end for**
23: **end if**
24: **end if**
25: **end for**

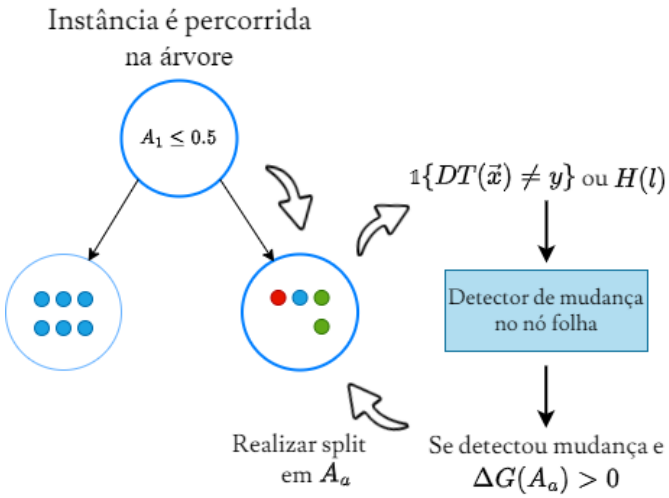

Figura 3.2: Ilustração do treinamento de uma instância do algoritmo LAST

Autoria: Autor, 2024.

(Algoritmo 4, linha 12). Se um detector de mudança alerta uma mudança, uma ramificação ocorrerá se $\Delta G(X_a) > 0$ (Algoritmo 4, linha 17). A Figura 3.2 ilustra esse processo, realizado no treinamento do algoritmo.

Ao considerar somente a pureza da distribuição dos dados para monitoramento, o algoritmo favorece a estratégia de voto majoritário (referida como LAST_D nesse trabalho). Enquanto a estratégia por taxa de erro considera o voto adaptativo que pode ser realizado com voto majoritário ou *Naive Bayes* nas folhas.

Os detectores de mudança são constantemente atualizados com instâncias que chegam nos nós-folha, indicando que os detectores de mudança acompanham como o fluxo evolui.

O algoritmo *LAST* aplica a condição de ramificação menos restritiva possível ($\Delta G(A_a) > 0$), sendo os algoritmos de detecção de mudança aqueles que ditam como a árvore cresce.

Uma vantagem do algoritmo é a do usuário não precisar especificar hiper-parâmetros como *Grace Period* e τ para *tie threshold*, sendo que para cada cenário o valor ótimo desses hiper-parâmetros pode variar. Detectores de mudança também possuem hiper-parâmetros, mas alguns deles são fáceis de interpretar como a confiança estatística desejada. Além disso, alguns detectores de mudança sequer possuem hiper-parâmetros, tornando o método proposto mais fácil de

utilizar e menos dependente de conhecimento do domínio a partir do qual os dados foram gerados.

LAST demanda $O(\psi ldcv)$ de complexidade de memória, sendo ψ a complexidade de memória do algoritmo de mudança de conceito aplicado, dado que cada nó folha instancia um detector de mudança.

O código-fonte da proposta está disponível em frameworks modernos de mineração de fluxo de dados como Massive Online Analysis (MOA)[1] (BIFET et al., 2010) para linguagem de programação JAVA e River[2] (MONTIEL et al., 2021) em Python.

3.1.1 Limitações da aplicação de Árvores de Decisão Adaptativas como *base learners* de *ensembles*

Uma das principais componentes de *ensembles* é a distribuição de um total diferente de cópias de uma instância aos seus *base learners* em fase de treinamento, como por amostragem aleatória com reposição (Bagging (BREIMAN, 1996)) ou ponderação da instância (Boosting (SCHAPIRE, 1999)) para induzir diversidade (KUNCHEVA; WHITAKER, 2003). Contudo, detectores de mudança lidam com instâncias monoliticamente, e múltiplas atualizações de um detector podem causar elevado *overhead* computacional dependendo de sua complexidade de memória. O treino de *LAST* então resultaria em *base learners* muito similares pois todos tratariam instâncias de forma monolítica, os tempos de ramificação entre árvores serão similares entre as árvores e, portanto, podem ocasionar baixa diversidade da *ensemble*. Esse efeito é observável nos experimentos realizados e discutidos na subseção 4.3.2.

3.1.2 HLAST e EFLAST: Combinação de Árvores de Decisão Adaptativas e Árvores baseadas no Teorema de Hoeffding

Árvores de Decisão baseadas no Teorema de Hoeffding, como *Hoeffding Trees* e EFDT, são sensíveis ao número de cópias que recebem de uma instância para o momento de uma ramificação, dado que uma tentativa de ramificação está

[1] <https://github.com/Waikato/moa/blob/master/moa/src/main/java/moa/classifiers/trees/LAST.java>

[2] <https://riverml.xyz/latest/api/tree/LASTClassifier/>

baseada no número de observações no nó folha. Para contornar as limitações presentes em árvores de decisão adaptativas como *base learners* de *ensembles*, esse trabalho também apresenta a proposta de combinar árvores de decisão adaptativas e baseadas no teorema de Hoeffding, aumentando a sensitividade ao número de cópias da instância e induzindo diversidade para *ensembles*, e adaptabilidade na ramificação.

O Algoritmo 5 apresenta a combinação dos mecanismos de ramificação adaptativos e aplicados em HT (referido nesse trabalho como HLAST) ou EFDT (referido nesse trabalho como EFDT). A estratégia de ramificação combinada utiliza a seguinte heurística: se o total de instâncias presentes no nó folha atinge um número divisível pelo *Grace Period*, uma tentativa de ramificação é realizada baseada no limite de Hoeffding, ou caso um detector alerte que uma mudança ocorreu, uma ramificação ocorre em A_a caso $\Delta G(A_a) > 0$. Caso o algoritmo seja HLAST, a condição de ramificação periódica é a mesma de HT, sendo $\Delta G(A_a) - \Delta G(A_b) > \epsilon$ (Algoritmo 5, linha 8). Caso contrário, o algoritmo é EFLAST e a condição de ramificação periódica é a mesma de EFDT, sendo $\Delta G(A_a) > \epsilon$ (Algoritmo 5, linha 8).

O código das combinações entre Árvores de Decisão Adaptativas e baseadas no Teorema de Hoeffding está disponível em página própria[3] acoplável com o *framework* MOA (BIFET et al., 2010) 24.01.

3.2 Considerações finais

Essa seção apresentou os métodos propostos nesse trabalho. A Seção apresentou o método *Local Adaptive Splitting Trees (LAST)* e suas vantagens em comparação com métodos que realizam avaliação periódica de ramificações baseadas no teorema de Hoeffding. Também foram apresentadas as limitações do método em sua aplicação como *base learners* de *ensembles*, dado que detectores de mudança lidam com instâncias de forma monolítica, resultando em *base learners* similares entre si e baixa diversidade, que é uma das principais características que tornam *ensembles* métodos efetivos de aprendizagem (KUNCHEVA; WHITAKER, 2003). Foram também propostas novas árvores (HLAST e EFLAST) que combinam o mecanismo estático de ramificação, que é sensível ao número

[3] <https://sites.google.com/view/last-ensemble/home>

Algoritmo 5 HLAST e EFLAST: Estratégia de ramificação em combinação de árvores adaptativas e HT ou EFDT

Input: S: um fluxo de dados,
X: um conjunto de vetores de características,
$\Delta G(\cdot)$: uma métrica de pureza,
ψ : Um algoritmo de detecção de mudança
GP: *Grace Period* (frequência de observações presentes em um nó folha para realizar uma tentativa de ramificação).
tipo_ramificação : Seleção do mecanismo de ramificação entre HT ou EFDT (gerando HLAST ou EFLAST, respectivamente)

$n_l \leftarrow n_l + 1$
if $\neg$(l contêm somente exemplos de uma classe) **then**
  if n_l mod $GP = 0$ **then**
    Compute $\Delta G(A_i)$ para cada $A_i \in A_l$ armazenados em l
    Seja A_a o atributo com maior ΔG
    Seja A_b o atributo com segundo maior ΔG
    Seja $\epsilon \leftarrow \sqrt{\frac{R^2 \log(\frac{1}{\delta})}{2n_l}}$ (2.13)
    Seja $cond \leftarrow \Delta G(A_a) - \Delta G(A_b) > \epsilon$ se tipo_ramificação = HT (HLAST), senão $\leftarrow \Delta G(A_a) > \epsilon$ (tipo_ramificação = EFDT, EFLAST)
    if $(cond \vee \tau > \epsilon) \wedge A_a \neq \emptyset$ **then**
      Integre a l nós folhas que se ramificam em A_a
      for cada nó folha l_i resultante da ramificação em A_a **do**
        Inicialize $n_{l_i} \leftarrow 0$
        $l_{i\psi} \leftarrow \psi$ ▷ cria um detector de mudança em cada nó folha
      end for
    end if
  else if l_ψ detectou mudança **then**
    Compute $\Delta G(A_i)$ para cada $A_i \in A_l$ armazenados em l
    Seja A_a o atributo com maior ΔG
    if $\Delta G(A_a) > 0 \wedge A_a \neq \emptyset$ **then**
      Integre a l nós folhas que se ramificam em A_a
      for cada nó folha l_i resultante da ramificação em A_a **do**
        Inicialize $n_{l_i} \leftarrow 0$
        $l_{i\psi} \leftarrow \psi$ ▷ cria um detector de mudança em cada nó folha
      end for
    end if
  end if
end if

de cópias que o *base learner* recebe para voto para ramificar e gerar diversidade na *ensemble*, e o mecanismo adaptativo de ramificação que reage a mudanças e considera a evolução do fluxo.

A próxima seção é dedicada a experimentos para atestar que os métodos propostos produzem resultados de estado-da-arte e elenca as vantagens e desvantagens dos métodos propostos em diferentes domínios.

4

Experimentos e Resultados

Essa seção tem como objetivo atestar a qualidade preditiva e de custo computacional dos métodos propostos em relação ao estado-da-arte no respectivo cenário.

A Seção 4.1 apresenta os *datasets* que serão utilizados para comparar a performance dos algoritmos em diferentes domínios. A Seção 4.2 apresenta os resultados para o ambiente monolítico para o algoritmo LAST. A Seção 4.3 apresenta os resultados para o ambiente *ensemble* para o algoritmo LAST e as combinações propostas.

4.1 *Datasets*

Essa seção abrange um descritivo de *datasets* utilizados em experimentos realizados. A Subseção 4.1.1 apresenta um descritivo de *datasets* reais, a Subseção 4.1.2 apresenta um descritivo de *datasets* sintético, a Subseção 4.1.3 detalha como mudanças de conceito são simuladas em *datasets* sintéticos e a Subseção 4.1.4 descreve um resumo das características dos conjuntos de dados.

4.1.1 *Datasets* Reais

Os *datasets* reais utilizados nesse trabalho foram extraídos do repositório de Souza et al. (2020) e essa seção apresenta um descritivo do cenário dos *datasets* e características do problema.

***Airlines*: (IKONOMOVSKA; GAMA; DžEROSKI, 2011)** A tarefa desse *dataset* é prever se um vôo irá atrasar baseado em informações de partida. É uma tarefa de classificação binária, em que os rótulos possíveis são atrasado ou não atrasado. O *dataset* possui 539.383 instâncias e 7 atributos.

Elec (HARRIES; WALES et al., 1999): O *dataset electricity* contempla dados de preço de contas de eletricidade na cidade de *New South Wales* na Austrália, em que o preço da conta não é fixo, são influenciados pela oferta e demanda, sendo ajustados a cada 5 minutos. Esse *dataset* contêm 45.312 instâncias, em que os rótulos possíveis para atribuir a instância são se o preço da eletricidade aumentou comparado a uma média móvel das últimas 24 horas. Esse *dataset* contêm dependências temporais.

Forest Covertype (BLACKARD; DEAN, 1999): esse *dataset* contêm dados florestais em células de 30x30 metros, extraídos do Sistema de Informação de Recursos (*Resource Information System - RIS*) da Região 2 do serviço florestal dos Estados Unidos (*US Forest Service - USFS*). Esse conjunto possui 581.012 instâncias e a tarefa consiste em prever o tipo de cobertura florestal, sendo 7 classes possíveis, a partir de 54 atributos cartográficos, sendo 10 numéricos e 44 binários. Esses atributos são referentes a dados de elevação, declividade, tipo de solo, entre outros. O *dataset* contêm dependências temporais pela relação geográfica dos exemplos, pois árvores da mesma espécie estão próximas umas das outras.

INSECTS (SOUZA et al., 2020): O *dataset* INSECTS é um problema de saúde pública, que envolve a identificação de insetos transmissores de doenças utilizando um sensor óptico responsável por medir a frequência da batida de asa do inseto. A frequência da batida de asa é similar a um áudio, e os autores extraem características relacionadas a esse sinal, como soma da energia de picos de frequência e harmônicos. No processo de coleta de dados, os autores alteram a temperatura do ambiente para induzir mudança de conceito, sendo 3 versões do *dataset* com mudanças de conceito abrupta, gradual e incremental utilizadas para experimento.

NOAA (ELWELL; POLIKAR, 2011): O *dataset* foi cedido pela Administração Nacional Oceânica e Atmosférica do Departamento de Comércio dos Estados

Unidos (*National Oceanic and Atmospheric Administration - NOAA*) e utilizado pela primeira vez em (ELWELL; POLIKAR, 2011). O *dataset* possui 18.154 instâncias de medições metereológicas diárias em mais de 7.000 estações metereológicas ao redor do mundo, em um período de 50 anos (1949-1999). A tarefa do *dataset* é prever se o dia foi chuvoso, com dados como temperatura, ponto de orvalho, pressão a nível do mar, visibilidade e velocidade média do vento.

Nomao (CANDILLIER; LEMAIRE, 2012): O *dataset* Nomao contêm dados de 2 lugares, onde a tarefa consiste em detectar se 2 lugares se referem ao mesmo local. Foi uma competição realizada pela conferência ECML-PKDD 2012. O *dataset* contêm 34.465 instâncias e 119 atributos.

Outdoor (LOSING; HAMMER; WERSING, 2015): A tarefa desse *dataset* consiste em classificar objetos em imagens capturadas por um celular em um jardim. O *dataset* contêm 4000 instâncias e 40 classes, e os atributos são constituídos por um histograma RGB normalizado com 21 dimensões.

Poker (CATTRAL; OPPACHER, 2007): cada instância desse *dataset* é uma mão de Poker que consiste em 5 cartas de um baralho de 52 cartas. Cada carta é descrita por 2 atributos (tipo e *ranking*), totalizando em 10 atributos. Esse *dataset* contêm 10 classes, sendo 10 tipos distintos de mão no jogo e 829.201 instâncias.

Rialto (LOSING; HAMMER; WERSING, 2016): esse *dataset* contêm imagens de dez edifícios adjacentes a ponte de Rialto em Veneza. As imagens foram capturadas por uma *web-cam* em posição fixa. O *dataset* contêm 82.250 instâncias, e 27 atributos que advêm de um histograma RGB normalizado. O prédio referente a classe (entre dez possíveis) foi mascarada manualmente pelos autores na imagem para se referir a classe desejada.

***LADPU* (SOUZA et al., 2021):** A tarefa desse *dataset* consiste em diferenciar entre 10 residências distintas baseado no consumo de energia elétrica medido por um *smart meter* pelo departamento de utilidades públicas de Los Alamos (LADPU), no Novo México, Estados Unidos. Os dados foram coletados a cada 15 minutos, tendo 96 observações por dia que compõem o espaço de atributos do *dataset* e 10 classes.

***Asfault* (SOUZA, 2018)** : esse *dataset* consiste em um sistema de monitoramento da qualidade de asfalto. Os dados foram coletados com um acelerômetro triaxial de um *smartphone* presente no interior de um veículo, que monitora a vibração em diferentes condições de asfalto. O *dataset* contêm 62 atributos, sendo as observações do sensor, e 5 classes, que indica a qualidade da rua (boa, média, aceitável, ruim e obstáculos).

4.1.2 *Datasets* Sintéticos

Os geradores de *datasets* sintéticos utilizados nesse trabalho estão presentes no *framework* MOA (BIFET et al., 2010) e são descritos a seguir.

***Agrawal* (AGRAWAL; IMIELINSKI; SWAMI, 1993):** O *dataset* simula a decisão de concessão de crédito a um cliente bancário. A classe de uma instância é determinada por uma das dez funções de empréstimos pré-definidas que mapeiam duas classes. Dos 9 atributos gerados pseudo-aleatoriamente, sendo 6 numéricos e 3 categóricos, tendo 10% de deviação do valor original, na qual simula ruído nos dados. Mudanças de conceito são simuladas ao se trocar a função que mapeiam as classes.

***LED* (BREIMAN, 1984):** Esse *dataset* simula a previsão de um dígito apresentado em um display de LED de 7 segmentos, tendo 24 atributos, sendo 17 deles irrelevantes. Mudanças de conceito são simuladas ao se definir o número de atributos que são relevantes.

***RBF* (BIFET et al., 2010):** esse *dataset* produz 10 atributos 5 classes. Os dados são gerados baseado na função de base radial (RBF). *Centroids* são gerados em posições pseudo-randômicas e mapeados com um desvio padrão, um peso e uma classe. Nesse *dataset*, a mudança de conceito incremental é simulada ao se alterar as posições dos *centroids* continuamente. É possível alterar o número de classes e atributos nesse *dataset*. 3 versões desse *dataset* foram utilizadas, tendo taxa de mudança de 10^{-4} (mudança devagar, referido como RBF_s), 10^{-3} (mudança moderada, referido como RBF_m), 10^{-2} (mudança rápida, referido como RBF_f).

***RTG* (DOMINGOS; HULTEN, 2000):** o gerador *Random Tree Generator* gera uma árvore de decisão com seleção randômica de atributos para ramificação e atribuição de uma classe randômica para cada nó folha. Após a árvore ser montada, novas instâncias são geradas ao atribuir valores que seguem uma distribuição normal para cada atributo. A classe dessa instância é determinada depois de percorrer a árvore e atribuir a classe presente nesse nó folha. É possível alterar o número de classes e atributos nesse *dataset*.

***Rotating Hyperplane (HYPER)* (HULTEN; SPENCER; DOMINGOS, 2001):** Um hiperplano é um subconjunto com $(n-1)$ dimensões que dividem o espaço de decisão em 2 ou mais partes desconectadas. Mudanças de conceito são simuladas incrementalmente ao mudar o limiar de decisão aplicado. 3 versões desse *dataset* foram utilizadas, tendo taxa de mudança de 10^{-4} (mudança devagar, referido como HYPER_s), 10^{-3} (mudança moderada, referido como HYPER_m), 10^{-2} (mudança rápida, referido como HYPER_f).

***SEA* (STREET; KIM, 2001):** esse gerador produz 3 atributos numéricos pseudo-aleatórios (f_1, f_2, f_3). Se $f_1 + f_2 \leq \theta$, a instância é atribuída classe 1, caso contrário 0. Nesse *dataset*, mudanças de conceito são simuladas ao mudar os valores de θ. Mudança de conceito são simuladas ao se mudar os valores de θ.

4.1.3 Simulação de Mudanças de Conceito em Bases Sintéticas

A simulação de mudanças de conceito abruptas e graduais em *datasets* sintéticos consiste na geração intercalar de instâncias de um conceito C_1 ou C_2. Então, é definido um período de mudança que determina a probabilidade de geração de uma instância do conceito C_1 ou C_2.

Esse trabalho utiliza a função probabilística por *sigmoid* introduzida em (BIFET et al., 2009). Dada uma instância I, sendo gerada em um tempo t_n e a mudança tendo início no tempo t_d com duração de w instâncias, a probabilidade que essa instância pertença ao conceito C_1, ou $P[I \in C_1]$, é definida na Equação 4.1.

$$P[I \in C_1] = \frac{1}{e^{-w \cdot (t_n - t_d)}} \tag{4.1}$$

Nota-se que $P[I \in C_1] = 1 - P[I \in C_2]$. Para geração de mudanças de conceito abruptas, o valor de w deve ser baixo em relação ao número total de instâncias

do fluxo, enquanto para mudanças de conceito graduais, o valor de w deve ser alto.

Mudanças de conceito recorrente consistem na volta de conceitos anteriores como predominantes, podendo ser de forma abrupta ou gradual.

Mudanças de conceito incrementais nesse trabalho são inerentes ao problema do *dataset*, como no *dataset* sintético RBF, em que a mudança de posição dos *centroids* muda ao longo do tempo e simula conceitos intermediários.

4.1.4 Resumo dos Conjuntos de Dados

A Tabela 4.1 apresenta as propriedades dos *datasets* apresentados nas subseções 4.1.1 e 4.1.2, com número de atributos, número de classes, a porcentagem de instâncias que pertencem à classe majoritária e o tipo de mudança de conceito. Para *datasets* sintéticos, é indicado também a ordem das funções que mapeiam a classe entre momentos de mudança de conceito. Para *datasets* reais, desconsiderando o *dataset* INSECTS, o tipo de mudança de conceito não foi rotulada e é desconhecida.

Para experimentos que variam o número de classes e atributos, foram utilizados os *datasets* RBF e RTG com 50.000 instâncias. O número de classes teve valores {2,10,50,100,200} e o número de atributos teve valores {10, 50, 100, 200, 300, 400, 500}.

4.2 Resultados para o Ambiente Monolítico

Os experimentos nessa Seção tem como objetivo atestar a performance do método proposto na Seção 3.1 (*Local Adaptive Streaming Tree* (LAST)) em relação à qualidade preditiva e custo computacional comparados com métodos estado-da-arte de árvore de decisão incrementais em diversos cenários, sendo que o protocolo experimental para os experimentos é descrito na Seção 4.2.1. Para isso, foi realizada uma análise sobre o efeito dos diferentes detectores de mudança (descritos na Seção 2.4) na qualidade preditiva e custo computacional do método proposto (abordado na Seção 4.2.2). Uma análise dos momentos de ramificação dos classificadores foi realizada para compreender o momento em que o método proposto tem qualidade superior ou inferior a árvores de decisão baseadas no teorema de Hoeffding (abordado na Seção 4.2.3). Em *datasets* sintéticos, foi verificado o efeito do número de classes e atributos para árvores de

Tabela 4.1: Propriedades dos *datasets* para a tarefa de classificação

Dataset	# Instâncias	# Atributos	# Classes	Classe majoritária (%)	Mudança de conceito
AGR-f(7,8,9,10,9,8,7)$_{ra}$	1.000.000	9	2	52,83	Recorrente-Abrupta
AGR-f(7,8,9,10,9,8,7)$_{rg}$	1.000.000	9	2	52,83	Recorrente-Gradual
AGR-f(1,2,3,4,5,6,7,8,9,10)$_{a}$	1.000.000	9	2	52,83	Abrupta
AGR-f(10,9,8,7,6,5,4,3,2,1)$_{a}$	1.000.000	9	2	52,83	Abrupta
AGR-f(1,2,3,4,5,6,7,8,9,10)$_{g}$	1.000.000	9	2	52,83	Gradual
AGR-f(10,9,8,7,6,5,4,3,2,1)$_{g}$	1.000.000	9	2	52,83	Gradual
SEA-f(1,2,3,4,3,2,1)$_{ra}$	1.000.000	3	2	59,91	Recorrente-Abrupta
SEA-f(1,2,3,4,3,2,1)$_{rg}$	1.000.000	3	2	59,91	Recorrente-Gradual
SEA-f(1,2,3,4)$_{a}$	1.000.000	3	2	59,91	Abrupta
SEA-f(1,2,3,4)$_{g}$	1.000.000	3	2	59,91	Gradual
SEA-f(4,3,2,1)$_{a}$	1.000.000	3	2	59,91	Abrupta
SEA-f(4,3,2,1)$_{g}$	1.000.000	3	2	59,91	Gradual
LED-f(7,5,3,1,3,5,7)$_{ra}$	1.000.000	24	10	10,28	Recorrente-Abrupta
LED-f(7,5,3,1,3,5,7)$_{rg}$	1.000.000	24	10	10,28	Recorrente-Gradual
LED-f(1,3,5,7)$_{a}$	1.000.000	24	10	10,28	Abrupta
LED-f(1,3,5,7)$_{g}$	1.000.000	24	10	10,28	Gradual
LED-f(7,5,3,1)$_{a}$	1.000.000	24	10	10,28	Abrupta
LED-f(7,5,3,1)$_{g}$	1.000.000	24	10	10,28	Gradual
RBF$_{s}$	1.000.000	10	5	30,01	Incremental
RBF$_{m}$	1.000.000	10	5	30,01	Incremental
RBF$_{f}$	1.000.000	10	5	30,01	Incremental
HYPER$_{s}$	1.000.000	10	2	50	Incremental
HYPER$_{m}$	1.000.000	10	2	50	Incremental
HYPER$_{f}$	1.000.000	10	2	50	Incremental
Outdoor	4.000	21	40	4,11	Desconhecido
Elec	45.312	8	2	57,41	Desconhecido
Rialto	82.250	27	10	10,00	Desconhecido
Airlines	539.383	7	2	55,47	Desconhecido
CoverType	581.012	54	7	48,75	Desconhecido
Nomao	34.465	119	2	71,44	Desconhecido
Poker	829.201	10	10	47,78	Desconhecido
NOAA	18.158	8	2	69,74	Desconhecido
INSECTS$_{a}$	52.848	33	6	16,07	Abrupta
INSECTS$_{i}$	57.018	33	6	11,56	Incremental
INSECTS$_{g}$	24.150	33	6	15,76	Gradual
Asfault	8.066	62	5	55,59	Desconhecido
LADPU	22.950	96	10	10	Desconhecido

decisão incrementais (abordado na Seção 4.2.4). Também foi desenvolvida uma análise sobre o efeito na qualidade preditiva dos métodos em relação a mudanças de conceito (tema abordado na Seção 4.2.5). Na Seção 4.2.6 é averiguado se combinações de mecanismos de ramificação periódicos e adaptativos obtêm resultados estado-da-arte.

Para abordar estes objetivos, os experimentos respondem as seguintes perguntas:

P1: Qual o efeito dos detectores de mudança na qualidade preditiva e custo computacional de LAST?

P2: Qual o efeito de ramificações na qualidade preditiva dos algoritmos?

P3: Como o número de classes e atributos afeta a qualidade preditiva e custo computacional do LAST?

P4: Qual o efeito de mudanças de conceito na qualidade preditiva do LAST e outras árvores de decisão?

P5: Qual o efeito de combinação de mecanismos de ramificação periódicos e adaptativos?

4.2.1 Protocolo Experimental para o Ambiente Monolítico

A qualidade preditiva dos algoritmos é avaliada a partir da acurácia, *Kappa Majority* e *Kappa Temporal* médio ao longo do fluxo dos modelos (conforme descrito na Seção 2.2) através da estratégia de validação *prequential* (discutido na Seção 2.3) extraídos de 20 pontos do fluxo em intervalos de tamanho igual, dado que não houve alteração significativa no *ranking* ao se dividir o fluxo em mais pontos. Essa estratégia evita que classificadores que desempenham melhor somente no fim do fluxo tenham *ranking* superior a métodos que performaram melhor durante todo o fluxo e tiveram uma queda de rendimento no final. Todos os experimentos para o ambiente monolítico foram realizados no *framework* MOA 23.01 (BIFET et al., 2010) com uma máquina Intel(R) Xeon(R) CPU E5649 @ 2.53GHz com 32 GB de RAM. As árvores baseadas no teorema de Hoeffding foram avaliadas com hiper-parâmetros padrão do *framework* MOA (*Grace Period* = 200, $\delta = 10^{-7}$, sendo o ganho de informação utilizado como função de pureza). Os detectores de mudança para o LAST assumiram hiper-parâmetros padrão do *framework* MOA 23.01 (BIFET et al., 2010) como apresentado na Tabela 4.2.

Testes de Wilcoxon com *ranking* e postos sinalizados unicaudal por par de algoritmo foram realizados, e uma correção de Holm forma os grupos não estatisticamente diferentes, como feito em (MIDDLEHURST PATRICK SCHäFER, 2024; GARCíA; HERRERA, 2008; BENAVOLI; CORANI; MANGILI, 2016). A ilustração gerada é similar a feita em (DEMŠAR, 2006), na qual uma linha numérica contêm o *ranking* dos algoritmos e grupos não estatisticamente diferentes estão conectados por uma barra.

4.2.2 Efeito dos Detectores de Mudança na Qualidade Preditiva e Custo Computacional de Árvores de Decisão Incrementais

Para abordar a pergunta **P1**, essa seção apresenta uma análise do efeito de diferentes detectores de mudança na qualidade preditiva e custo computacional do algoritmo LAST. O algoritmo LAST com monitoramento de taxa de erro é referido como LAST, enquanto o algoritmo LAST com monitoramento de distribuição dos dados é referido como LAST_D.

A Figura 4.1 apresenta um teste de Wilcoxon para acurácia média ao longo do fluxo. O algoritmo LAST, independente do detector, se enquadrou entre os

Detector	parâmetros
ADWIN	$\delta = 0,002$ clock = 32 max buckets = 5
DDM	$\delta_{warning} = 2$ $\delta_{drift} = 3$ min num instances = 30
EDDM	$\alpha = 0,95$ $\beta = 0,9$ min num instances = 30
HDDM$_A$	$\delta_{warning} = 0,001$ $\delta_{drift} = 0,005$ tipo = unicaudal
HDDM$_W$	$\delta_{warning} = 0,001$ $\delta_{drift} = 0,005$ $\lambda = 0,05$ tipo = unicaudal
RDDM	$\delta_{warning} = 1,773$ $\delta_{drift} = 2,258$ min num instances = 129 max concept size = 40000 min stable size = 7000 warning limit = 1400
MDDM$_A$	sliding window size = 100 diff = 0,01 conf = 0,000001
MDDM$_E$	sliding window size = 100 $\lambda = 0,01$ conf = 0,000001
MDDM$_G$	sliding window size = 100 ratio = 1,01 conf = 0,000001

Tabela 4.2: Hiper-parâmetros *default* dos detectores de mudança no framework MOA 23.01

melhores resultados. Com exceção dos detectores $MDDM_E$ e $HDDM_W$, todos os detectores obtiveram resultados estatisticamente diferentes de EFDT, enquanto HAT não obteve. Os detectores $HDDM_A$ e $MDDM_A$ obtiveram os melhores resultados. O algoritmo $LAST_D$ não obteve resultados superiores a EFDT em *ranking*, com exceção do detector ADWIN, porém não obteve resultados estatisticamente diferentes de EFDT. Ramificações em momentos de aumento de erro em nós folhas de árvores de decisão se mostraram efetivas para produzir árvores com boa qualidade preditiva, enquanto monitoramento da pureza da distribuição dos dados não, tendo 2 causas são possíveis. Ou os detectores não obtêm bons resultados de detecção em mudanças em variáveis numéricas, ou não há correlação entre as mudanças na distribuição dos dados e momentos de ramificação que resultem em aumento da qualidade preditiva.

A Figura 4.2 apresenta um teste de Wilcoxon para tamanho da árvore (número de nós). LAST com os detectores MDDM, HDDM e DDM não apresentaram resultados estatisticamente diferentes de HT, EFDT e HAT, enquanto LAST com o detector ADWIN foi o único que apresentou resultados estatisticamente diferentes de EFDT. Não necessariamente um tamanho de árvore maior implica que o algoritmo demande maior custo computacional (e.g, a árvore ter mecanismos de reavaliação de ramificações, resultar em uma árvore menor, porém necessitar armazenar dados em todos os nós da árvore, demandando maior uso de memória). Um tamanho de árvore maior também pode justificar maior qualidade preditiva. Nesse caso, interessantemente LAST não apresenta resultados estatisticamente diferentes de HT, EFDT e HAT em tamanho da árvore e simultaneamente tem resultados estatisticamente diferentes em acurácia.

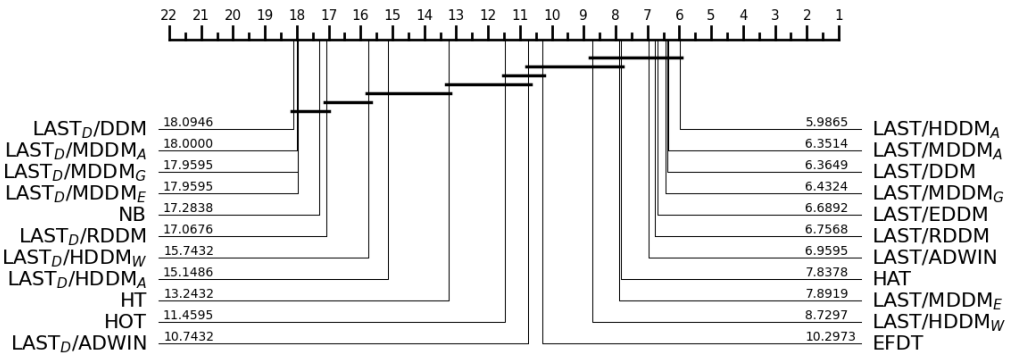

Figura 4.1: Ranking da Acurácia (em %) média ao longo do fluxo para o *baseline*, LAST e $LAST_D$ com diversos detectores

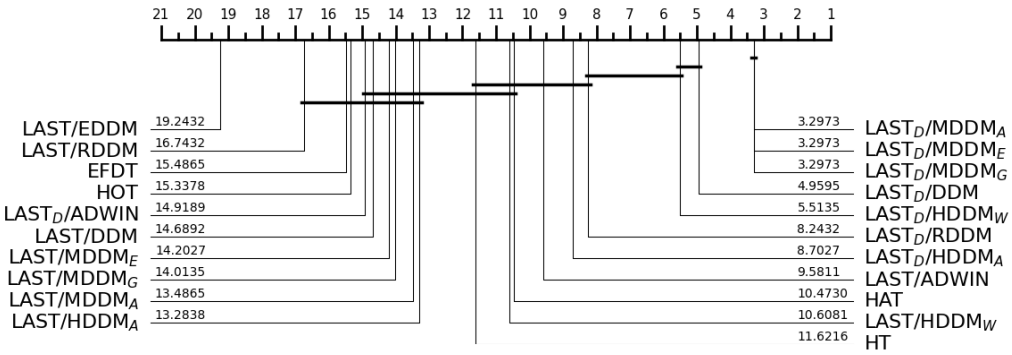

Figura 4.2: Ranking de tamanho de árvore do *baseline*, LAST e LAST$_D$ com diversos detectores

A Tabela 4.3 apresenta o *ranking* médio das métricas acurácia, Kappa$_M$, Kappa$_T$, número de nós, *CPU-Time* e *RAM-Hours* para os classificadores monolíticos avaliados nesse trabalho. LAST com os detectores RDDM, EDDM e DDM apresentaram os melhores resultados em acurácia, Kappa$_M$ e Kappa$_T$ em *datasets* reais, porém não tem os melhores resultados em *datasets* sintéticos, mostrando uma disparidade, e apresentam os piores resultados em número de nós, *CPU-Time* e *RAM-Hours* tanto em *datasets* reais e sintéticos. O comportamento homogêneo dos detectores RDDM, EDDM e DDM pode ser explicado pelos mecanismos similares de detecção de mudança utilizados por eles, ao identificar desvio do padrão com estatística descritiva.

HDDM$_A$ apresentou custo computacional inferior em comparação a RDDM, EDDM e DDM e obteve segundo melhor ranking acurácia média tanto em *datasets* reais e sintéticos. MDDM obteve acurácia, Kappa$_M$ e Kappa$_T$ similares a HDDM, enquanto obteve custo computacional inferior. Todas as variantes de MDDM obtiveram *ranking* de custo computational inferior a HDDM, enquanto apresenta bom *ranking* em qualidade preditiva em *datasets* reais e sintéticos. MDDM$_A$ apresentou os melhores resultados entre as variantes e MDDM. HDDM$_A$ e MDDM obtiveram resultados superiores a EFDT em qualidade preditiva, enquanto apresentam *CPU-Time* e *RAM-Hours* inferior em *datasets* reais, e *CPU-Time* inferior em *datasets* sintéticos. ADWIN e HDDM$_W$ apresentaram-se como métodos com bom custo-benefício, sendo que possuem qualidade preditiva média superior a EFDT com custo computacional equiparável a *Hoeffding Tree*. ADWIN apresentou tempo total de processamento de todos os *datasets* mais próximo de HT. Dado que LAST depende de mudanças em acurácia ocorrerem em nós folhas, é esperado que LAST não aprenda em cenários com pouca ou ne-

nhuma mudança. Porém, em *datasets* reais, nenhum caso de pouca atualização do modelo é observável.

ADWIN, HDDM e MDDM possuem condições de detecção de mudança mais estritas e baseadas nos teoremas de Hoeffding e McDiarmid em comparação com RDDM, EDDM e DDM, e um *trade-off* entre qualidade preditiva e custo computacional é observável entre esses 2 tipos de métodos. *Trade off* entre custo computacional e acurácia em métodos de detecção baseados em teste estatístico (mais restritivo) e desvio do padrão com estatística descritiva (menos restritivo) se dá pelo número de ramificações dos métodos, sendo que LAST com os detectores RDDM, EDDM e DDM apresentaram tamanho de árvore médio superior comparado a LAST com os detectores ADWIN, HDDM e MDDM.

ADWIN foi o único detector com o qual o algoritmo LAST_D obteve resultados equiparáveis aos melhores métodos, porém apresenta maior custo computacional e qualidade preditiva inferior comparado com o algoritmo LAST com detector ADWIN. Possivelmente mais detecções desnecessárias foram alertadas ao se monitorar distribuição das classes. Outra explicação para a performance inferior do LAST_D seria a baixa correlação entre alterações na distribuição de classes e momentos apropriados de ramificação.

Para uma análise mais profunda das demais questões relacionadas aos experimentos, 3 métodos foram selecionados, sendo LAST com os detectores HDDM_A e MDDM_A, por obter os melhores resultados em *ranking* e custo computacional inferior aos detectores EDDM, DDM e RDDM, e com o detector ADWIN pelo bom custo-benefício apresentado e diferença estatística de EFDT como descrito anteriormente.

A Figura 4.3 apresenta um *scatter plot* da acurácia média entre HAT e as árvores adaptativas selecionadas. Em *datasets* reais, as árvores adaptativas selecionadas obtiveram ganho maior ou igual a 11 dos 13 *datasets*. Todas as árvores adaptativas mostram diferença estatística em um teste de *wilcoxon* em relação a HAT em *datasets* reais com p-valor inferior a 0,05. Em *datasets* sintéticos, HAT apresentou mais ganhos que as árvores adaptativas selecionadas. Nenhuma das árvores adaptativas mostrou diferença estatística em um teste de *wilcoxon* em relação a HAT em *datasets* sintéticos com p-valor superior a 0,05. A Seção 4.2.5 discute a disparidade entre qualidade preditiva em *datasets* reais e sintéticos pela característica das mudanças de conceito simuladas.

A Figura 4.4 apresenta um *scatter plot* da acurácia média entre EFDT e as árvores adaptativas selecionadas. Em *datasets* reais, MDDM_A foi a árvore com

Tabela 4.3: Ranking médio por métrica e tempo total para processar todos os *datasets* em cenários reais e sintéticos.

Tipo do *Dataset*		NB	HT	HOT	EFDT	HAT
Real	Acc	17.62(19)	15.38(16)	12.77(12)	9.85(11)	14.54(14)
	$Kappa_M$	17,54(19)	15,38(16)	14,38(13)	9,92(11)	14,77(14)
	$Kappa_T$	17,62(19)	15,46(16)	12,77(12)	10,00(11)	14,85(14)
	Size	-	7,38(7)	10,88(10)	13,77(13)	6,85(6)
	CPU-Time	**1,08(1)**	9,27(8)	15,85(16)	18,46(20)	18,69(21)
	Total-Time	**54,45(1)**	123,83(8)	149,81(12)	207,50(21)	182,33(17)
	Ram-Hours(10^{-6})	9,08(8)	7,85(6)	12,00(10)	16,77(19)	15,08(16)
Sintético	Acc	17.10(18)	12.08(14)	10.75(12)	10.54(11)	**1.00(1)**
	$Kappa_M$	17,10(18)	12,08(14)	10,79(12)	10,50(11)	**4,21(1)**
	$Kappa_T$	17,10(18)	12,08(14)	10,79(12)	10,54(11)	**4,21(1)**
	Size	-	13,92(15)	17,75(20)	16,42(19)	12,58(11)
	CPU-Time	**1,00(1)**	14,08(14)	20,75(22)	19,38(20)	20,50(21)
	Total-Time	**95,30(1)**	254,80(11)	649,55(22)	422,56(19)	580,05(21)
	Ram-Hours(10^{-6})	6,21(6)	14,42(15)	19,67(22)	17,62(19)	19,08(21)

Tabela 4.3 : Continuada.

Tipo do *Dataset*		LAST ADWIN	DDM	EDDM	$HDDM_A$	$HDDM_W$	RDDM
Real	Acc	7.31(9)	6.46(7)	6.00(3)	5.62(2)	6.85(8)	**1.00(1)**
	$Kappa_M$	7,15(9)	6,38(7)	6,23(6)	5,31(2)	6,62(8)	**5,23(1)**
	$Kappa_T$	7,23(9)	6,46(7)	6,31(6)	5,38(2)	6,69(8)	**5,31(1)**
	Size	10,81(9)	16,69(19)	20,23(21)	15,88(18)	11,62(12)	18,27(20)
	CPU-Time	10,31(9)	16,15(18)	20,38(22)	15,85(16)	10,50(10)	17,31(19)
	Total-Time	125,67(9)	197,38(19)	266,02(22)	182,50(18)	109,50(7)	204,66(20)
	Ram-Hours(10^{-6})	10,77(9)	16,85(20)	21,00(22)	15,54(17)	12,00(10)	18,62(21)
Sintético	Acc	6.77(6)	6.31(3)	7.06(7)	6.19(2)	9.75(10)	7.54(8)
	$Kappa_M$	6,81(6)	6,44(4)	7,06(7)	6,23(2)	9,62(10)	7,67(8)
	$Kappa_T$	6,81(6)	6,31(3)	7,10(7)	6,19(2)	9,67(10)	7,62(8)
	Size	8,92(7)	13,60(13)	18,71(21)	11,88(10)	10,06(9)	15,92(18)
	CPU-Time	11,69(10)	12,62(12)	17,00(19)	12,17(11)	9,54(9)	15,75(18)
	Total-Time	241,89(10)	308,30(18)	427,34(20)	286,64(13)	221,50(9)	304,94(17)
	Ram-Hours(10^{-6})	9,83(9)	13,62(13)	18,50(20)	12,08(11)	10,21(10)	16,25(18)

Tabela 4.3 : Continuada.

Tipo do *Dataset*		LAST $MDDM_A$	$MDDM_E$	$MDDM_G$	$LAST_D$ ADWIN	DDM
Real	Acc	6.23(6)	6.23(6)	6.15(4)	9.69(10)	16.85(17)
	$Kappa_M$	6,08(4)	6,15(5)	6,08(4)	9,62(10)	16,54(17)
	$Kappa_T$	6,23(4)	6,23(4)	6,15(3)	9,69(10)	16,54(17)
	Size	14,00(14)	14,62(16)	14,54(15)	15,23(17)	5,50(5)
	CPU-Time	13,73(14)	13,54(12)	13,62(13)	14,15(15)	4,88(4)
	Total-Time	171,12(14)	176,70(15)	177,98(16)	154,70(13)	100,66(6)
	Ram-Hours(10^{-6})	13,92(14)	14,23(15)	13,92(14)	15,69(18)	4,69(4)
Sintético	Acc	6.42(4)	8.79(9)	6.58(5)	11.31(13)	18.77(22)
	$Kappa_M$	6,42(3)	8,75(9)	6,54(5)	11,27(13)	18,77(22)
	$Kappa_T$	6,46(4)	8,79(9)	6,58(5)	11,23(13)	18,77(22)
	Size	13,21(12)	13,98(16)	13,73(14)	14,75(17)	4,65(5)
	CPU-Time	14,10(15)	14,31(16)	13,00(13)	15,54(17)	4,15(3)
	Total-Time	287,59(15)	286,92(14)	283,80(12)	292,45(16)	159,45(2)
	Ram-Hours(10^{-6})	13,67(14)	14,58(16)	13,58(12)	15,75(17)	3,92(2)

Tabela 4.3 : Continuada.

Tipo do *Dataset*		$LAST_D$ $HDDM_A$	$HDDM_W$	RDDM	$MDDM_A$	$MDDM_E$	$MDDM_G$
Real	Acc	13.77(13)	14.85(15)	17.42(18)	18.12(22)	18.00(20)	18.00(20)
	$Kappa_M$	13,62(12)	15,00(15)	17,35(18)	17,96(22)	17,85(20)	17,85(20)
	$Kappa_T$	13,69(13)	15,08(15)	17,42(18)	18,04(22)	17,92(20)	17,92(20)
	Size	11,23(11)	8,04(8)	5,42(4)	3,35(2)	3,35(2)	3,35(2)
	CPU-Time	10,92(11)	7,77(7)	6,08(6)	5,50(5)	4,54(3)	4,42(2)
	Total-Time	136,17(11)	72,95(4)	136,09(10)	73,22(5)	72,92(3)	72,91(2)
	Ram-Hours(10^{-6})	12,69(12)	8,08(7)	4,73(5)	3,92(3)	2,85(2)	**2,73(1)**
Sintético	Acc	15.90(15)	16.23(16)	16.88(17)	17.94(20)	17.94(20)	17.94(20)
	$Kappa_M$	15,98(15)	16,10(16)	16,83(17)	17,94(20)	17,94(20)	17,94(20)
	$Kappa_T$	15,90(15)	16,19(16)	16,83(17)	17,94(20)	17,94(20)	17,94(20)
	Size	7,31(6)	4,12(4)	9,75(8)	3,25(2)	3,25(2)	3,25(2)
	CPU-Time	8,19(7)	4,12(2)	9,02(8)	4,98(4)	6,00(6)	5,10(5)
	Total-Time	207,94(8)	164,95(6)	196,84(7)	160,92(3)	162,06(5)	161,12(4)
	Ram-Hours(10^{-6})	7,42(7)	**3,83(1)**	9,81(8)	3,92(2)	4,92(5)	4,10(4)

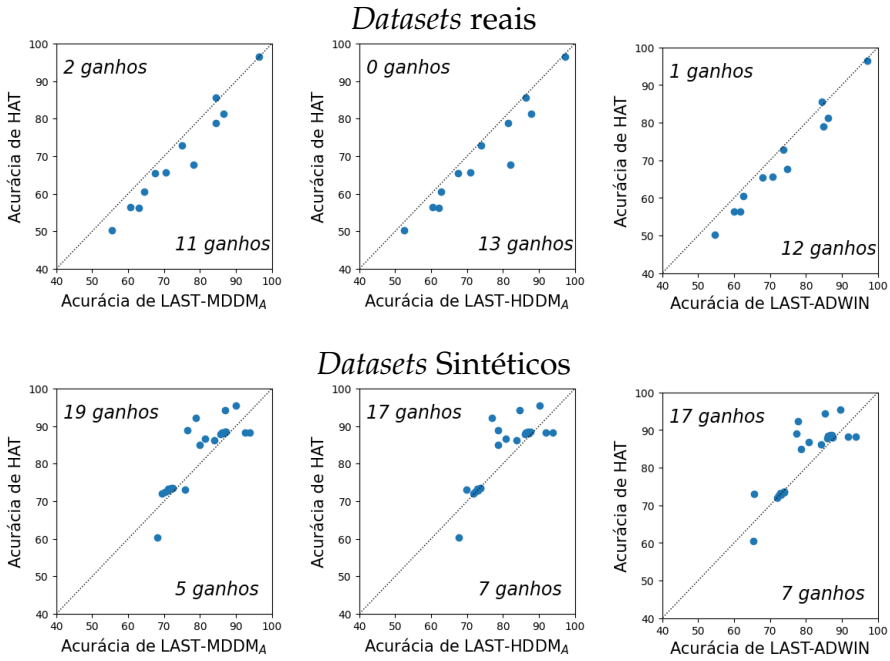

Figura 4.3: Comparação entre a acurácia (em %) média ao longo do fluxo de *HAT* e árvores adaptativas em *datasets* reais e sintéticos. Cada ponto representa o resultado em um *dataset*.

melhor razão entre ganhos e perdas em comparação com EFDT, com 12 ganhos e 1 perda. LAST com os detectores HDDM$_A$ e MDDM$_A$ mostraram diferença estatística em um teste de *wilcoxon* em relação a EFDT em *datasets* reais com p-valor inferior a 0,05. Em *datasets* sintéticos, as árvores adaptativas selecionadas obtiveram ganho maior ou igual a 17 dos 24 *datasets* sintéticos. Todas as árvores adaptativas selecionadas mostraram diferença estatística em um teste de *Wilcoxon* em relação a EFDT em *datasets* sintéticos com p-valor inferior a 0,05.

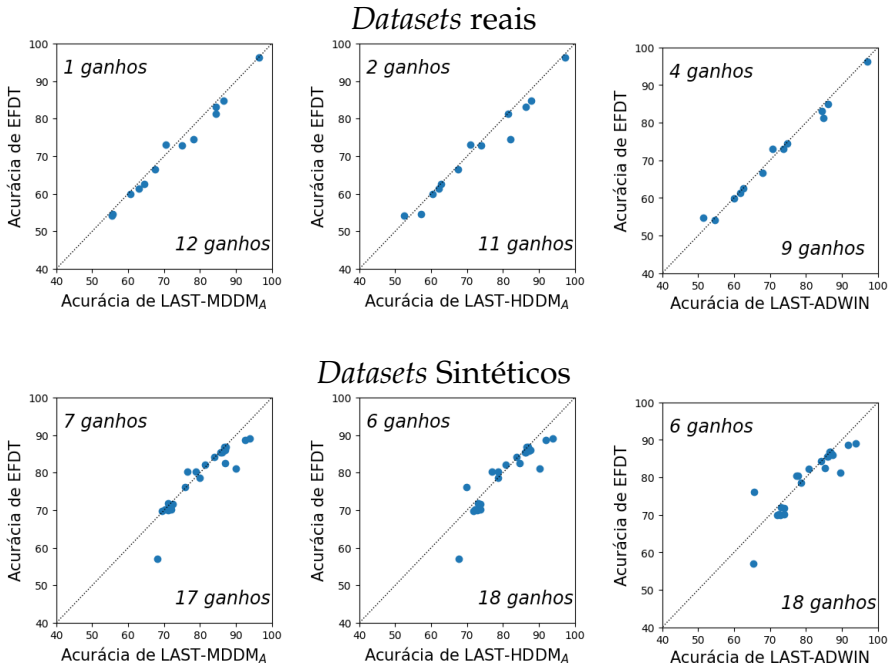

Figura 4.4: Comparação entre entre a acurácia (em %) média ao longo do fluxo de *EFDT* e Árvores Adaptativas em *datasets* reais e sintéticos. Cada ponto representa o resultado em um *dataset*.

Em conclusão, LAST com monitoramento de erro se mostrou um eficiente classificador independente do detector empregado. LAST$_D$ com o detector ADWIN foi o único que se apresentou entre os melhores resultados em *ranking*, porém apresenta custo computacional médio superior em comparação com LAST com o detector ADWIN e qualidade preditiva média e *ranking* inferior. Foi observado que os métodos baseados em limite estatístico (HDDM, MDDM, e ADWIN) são computacionalmente mais baratos que métodos baseados em estatística descritiva (DDM, EDDM e RDDM) devido a forma com a qual esses 2 tipos de detectores alertam uma mudança, sendo que apresentam qualidade preditiva equiparável. ADWIN se apresentou como um detector com bom custo-benefício entre qualidade preditiva e custo computacional, tendo acurácia média superior a EFDT com custo computacional equiparável com *Hoeffding Tree*. Assim, respondendo à pergunta **P1**.

4.2.3 Ramificações em Árvores de Decisão Incrementais

Para abordar a pergunta **P2**, essa seção apresenta o efeito de ramificações na acurácia dos algoritmos. Para isso, é apresentada a acurácia ao longo do

tempo para *datasets* selecionados, em que linhas verticais indicam momentos de ramificação da árvore na sua respectiva cor.

Foram selecionados *datasets* que permitam a visualização da acurácia ao longo do tempo sem interferência do número de ramificações sobre a figura. Para alguns *datasets*, foi limitado o número de instâncias para 15000 por motivos de visualização. Para LAST, apresentamos resultados somente para o detector ADWIN, visto que produz árvores menores que HDDM e MDDM.

A Figura 4.5 apresenta a acurácia ao longo do tempo com pontos de ramificação para os *datasets Asfault*, *Outdoor* e *NOAA*. Esses cenários demonstram que árvores baseadas no teorema de Hoeffding são invariantes à performance da árvore, e que ramificações em caso de degradação da performance tornam LAST superior a árvores baseadas no teorema de Hoeffding quando estabilizam e fazem a acurácia aumentar.

No *dataset Asfault*, HT apresenta mesma performance que LAST com detector ADWIN tendo mesmo estado (Naive Bayes adaptativo), até que a performance da *Hoeffding Tree* decai abruptamente enquanto LAST conseguiu reagir ao decaimento em acurácia com uma ramificação e manter performance estável. EFDT apresentou muitas ramificações desnecessárias, principalmente em momentos iniciais do fluxo.

Nos *datasets Outdoor* e *NOAA*, LAST com o detector ADWIN apresentou comportamento similar, em que ramificações em caso de degradação na acurácia, principalmente em momentos iniciais do fluxo, estabilizaram ou aumentaram a acurácia. No *dataset Outdoor*, a quarta ramificação do EFDT apresentou um momento de ramificação que superou a acurácia de LAST, que ramificou após o decaimento ser observado. Porém, mais adiante no fluxo, LAST apresentou uma ramificação em momento de decaimento que superou a acurácia de EFDT. No *dataset NOAA*, EFDT também apresenta um momento de ramificação próximo da instância 15000 que aproximou sua performance à do LAST.

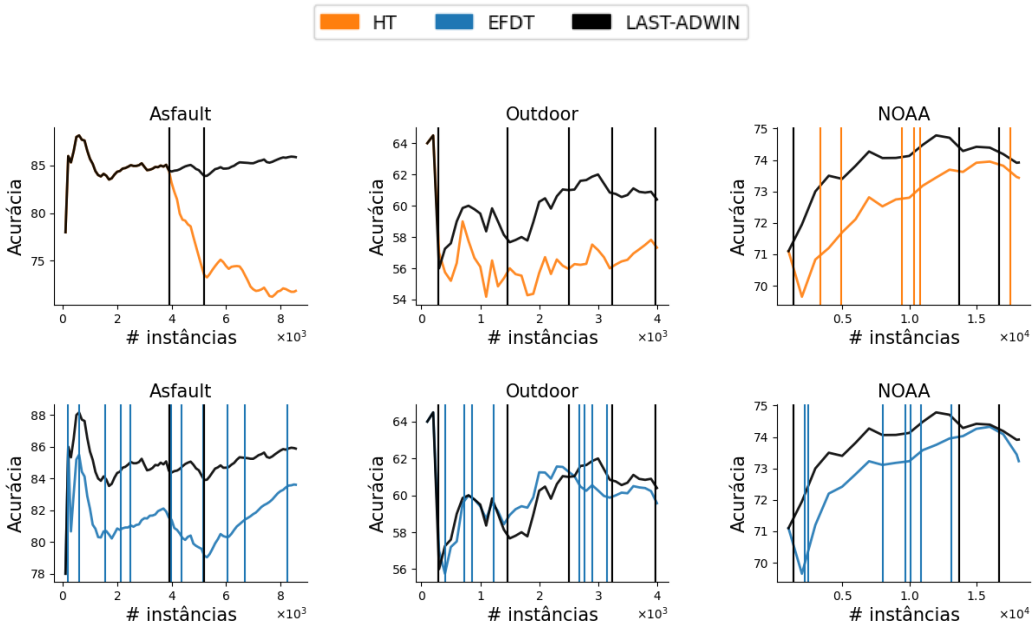

Figura 4.5: Acurácia (em %) ao longo do tempo para Hoeffding Trees, EFDT e LAST com detector ADWIN nos *datasets* Asfault, Outdoor e NOAA.

A Figura 4.6 apresenta a acurácia ao longo do tempo com pontos de ramificação para os *datasets Elec*, *Poker* e *Nomao*. Os *datasets Elec* e *Poker* mostram alguns exemplos em que ramificações não afetam a performance das árvores, dado que a diferença de acurácia entre as árvores permanece similar ao longo do tempo, como entre as instâncias 7500 e 12500 no *dataset Poker*, porém, também é observável que LAST apresenta maior estabilidade em momentos de decaimento de performance, como na quinta ramificação no *dataset Poker* e possuí performance superior em casos que LAST ramifica em momentos de decaimento de performance. No *dataset Nomao*, LAST apresenta performance superior por detectar decaimento no inicio do fluxo e entre as instâncias 12500 e 15000.

Em conclusão, LAST apresenta performance superior a outras árvores de decisão, reagindo rapidamente a decaimento na acurácia com ramificações. Também foram apresentados casos em que EFDT realizou ramificação antes do decaimento da acurácia, como nos *datasets Outdoor* e *NOAA*, porém, sem a devida eficácia. Em geral, as ramificações do LAST em casos de decaimento de acurácia apresentou resultados superiores, respondendo à pergunta **P2**.

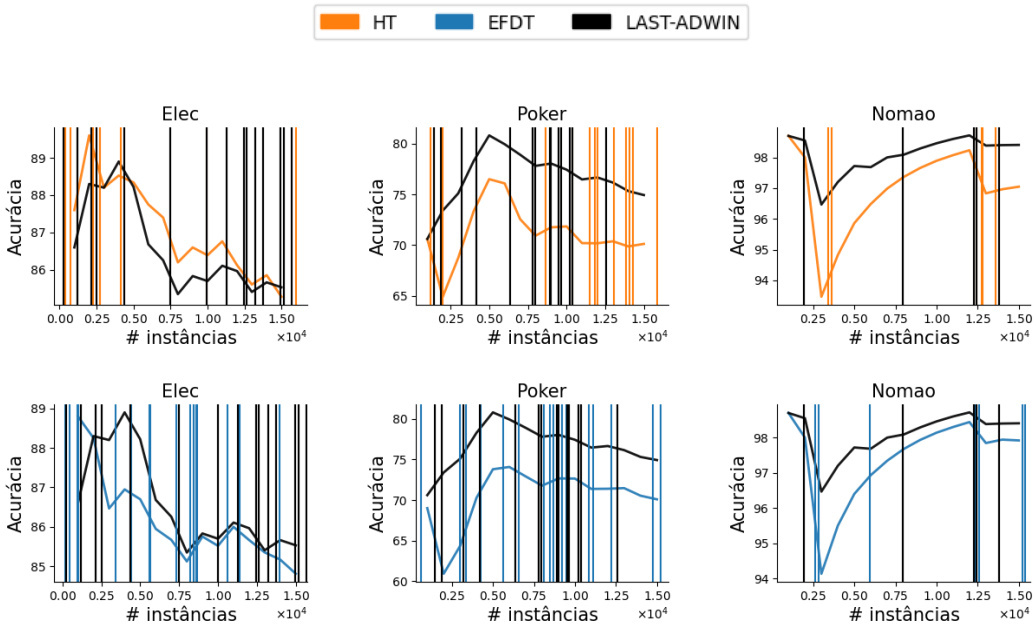

Figura 4.6: Acurácia (em %) ao longo do tempo para Hoeffding Trees, EFDT e LAST com detector ADWIN nos *datasets* Elec, Poker e Nomao

4.2.4 Efeito do Número de Classes e Atributos em Árvores de Decisão Adaptativas

Para abordar a pergunta **P3**, essa seção apresenta resultados para os *datasets* RTG e RBF, em que é possível alterar o número de classes e atributos. Para cada *plot* em três dimensões, é garantido que resultados superiores vão sobrepujar resultados inferiores. A Figura 4.7 apresenta a acurácia ao se variar o número de classes e atributos para uma *Hoeffding Tree*. Em todas as árvores observou-se decaimento na acurácia quando há aumento de atributos e classes. Esse comportamento é esperado, dado que árvores de decisão são afetadas pelo *curse of dimensionality* (BENGIO; DELALLEAU; SIMARD, 2010).

A Figura 4.8 apresenta um comparativo entre HT e LAST com o detector ADWIN para as métricas acurácia (em %), número de nós da árvore, *CPU-Time* e *RAM-Hours* para os *datasets* RBF e RTG. Em acurácia, LAST apresenta acurácia superior na maioria dos casos. HT apresenta tamanho de árvore superior comparado a LAST, principalmente com 10 atributos que obteve o maior tamanho de árvore (entre 30 e 40 para o *dataset* RBF e entre 25 e 30 para o *dataset* RTG). É possível notar que o limite de Hoeffding é mais conservador quando o número de atributos aumenta, sendo que a maioria dos *datasets* em que árvores

de decisão *estado-da-arte* são avaliados tem poucos atributos. Para *CPU-Time* e *RAM-Hours*, LAST e HT obtiveram resultados similares.

A Figura 4.9 apresenta um comparativo entre EFDT e LAST com o detector ADWIN para as métricas acurácia (em %), número de nós da árvore, *CPU-Time* e *RAM-Hours* para os *datasets* RBF e RTG. Em acurácia, LAST apresenta acurácia superior em maioria dos casos. Em número de nós, é observável que o limite de Hoeffding usado em EFDT é menos conservador comparado ao utilizado em (DOMINGOS; HULTEN, 2000), e no *dataset* RTG, EFDT teve maior número de nós que LAST entre 200 e 400 atributos. Para *CPU-Time*, LAST e EFDT obtiveram resultados similares. Para *RAM-Hours*, EFDT apresentou maior uso de memória quanto maior o número de atributos e classes, sendo relativo ao aumento do número de nós.

A Figura 4.10 apresenta um comparativo entre HAT e LAST com o detector ADWIN para as métricas acurácia (em %), número de nós da árvore, *CPU-Time* e *RAM-Hours* para os *datasets* RBF e RTG. Em acurácia, LAST apresenta acurácia superior em maioria dos casos. Em número de nós, HAT apresenta número de nós inferior a LAST pelo limite de Hoeffding ser mais conservador a medida que o número de atributos aumenta, como descrito anteriormente. Para *CPU-Time*, LAST e HAT obtiveram resultados similares. Para *RAM-Hours*, HAT apresentou maior uso de memória comparado a LAST, como no *dataset* RBF, em que quanto maior o número de classes em um cenário com 10 atributos, maior foi o custo de memória pelo HAT. Pela árvore crescer mais, mais detectores são criados e aumentam o custo do HAT em memória.

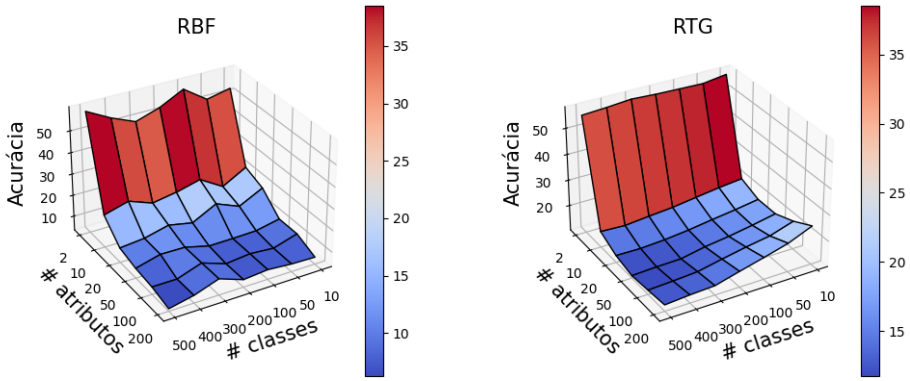

Figura 4.7: Acurácia (em %) de *Hoeffding Tree* com variação do número de classes e atributos nos *datasets* RBF e RTG

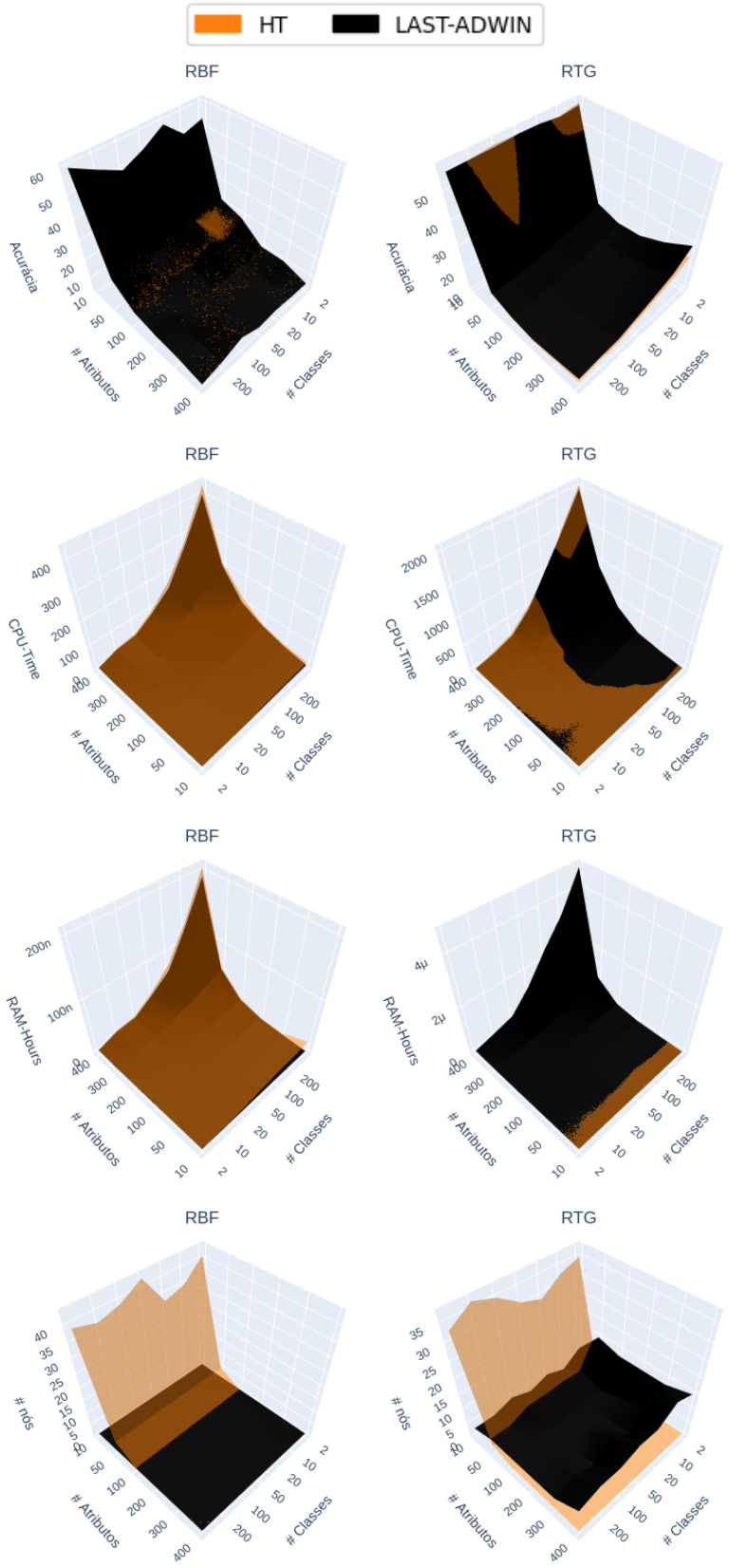

Figura 4.8: Acurácia (em %), *CPU-Time* (em segundos), *RAM-Hours* (em GB/h) e número de nós das árvores Hoeffding Tree e LAST com detector ADWIN nos *datasets* RBF e RTG

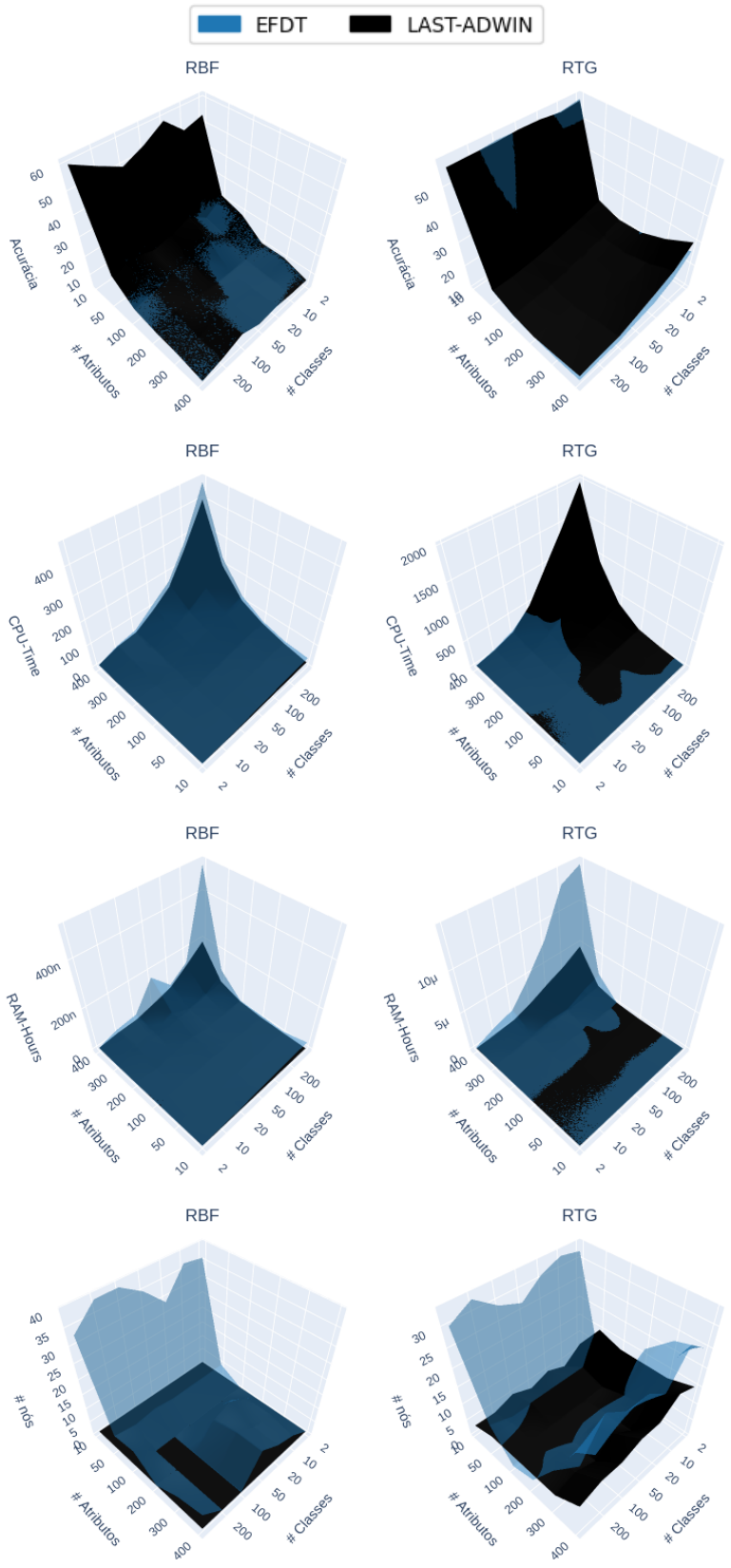

Figura 4.9: Acurácia (em %), *CPU-Time* (em segundos), *RAM-Hours* (em GB/h) e número de nós das árvores EFDT e LAST com detector ADWIN nos *datasets* RBF e RTG

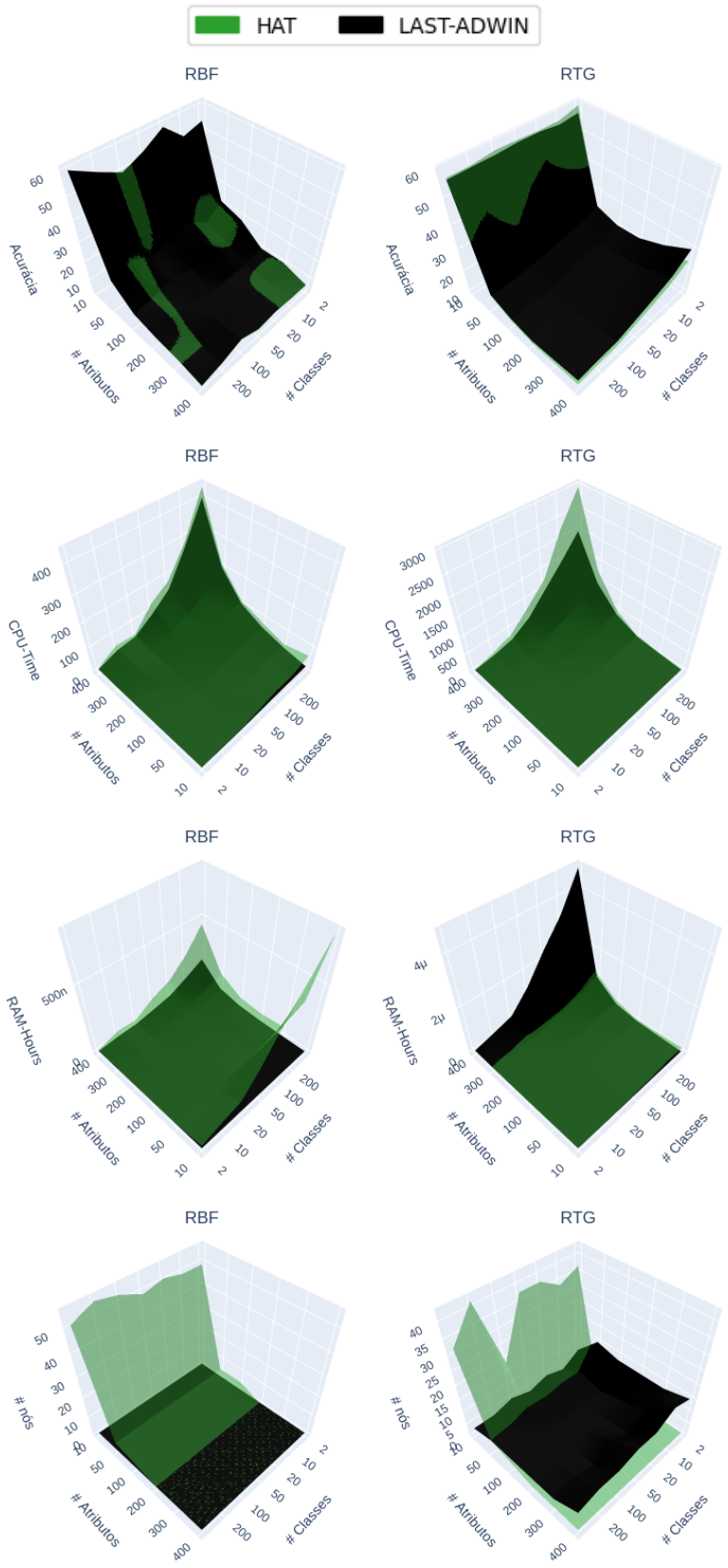

Figura 4.10: Acurácia (em %), *CPU-Time* (em segundos), *RAM-Hours* (em GB/h) e número de nós das árvores *HAT* e LAST com detector ADWIN nos *datasets* RBF e RTG

A Figura 4.11 apresenta a distribuição de *CPU-Time* ao se variar o número de atributos nos *datasets* RBF e RTG. Os algoritmos são ordenados por mediana, e o estrato de atributos considera todos os casos de número de classes. Em geral, os métodos apresentaram distribuição similar, porém, HAT E EFDT se enquadraram entre os métodos com maior quartil inferior, mediana e quartil superior.

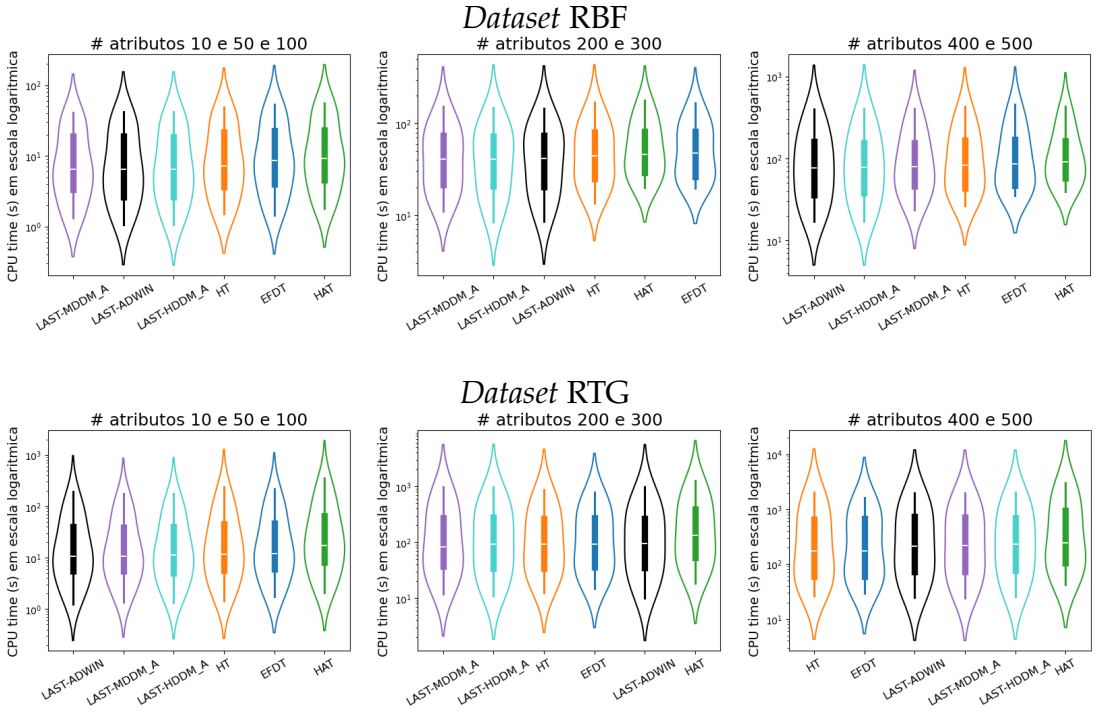

Figura 4.11: Distribuição de CPU-Time (em segundos e escala logaritmica) dos *datasets* RBF e RTG

A Figura 4.12 apresenta a distribuição de *RAM-Hours* ao se variar o número de atributos nos *datasets* RBF e RTG. Os algoritmos são ordenados por mediana, e o estrato de atributos considera todos os casos de número de classes. Diferentemente de CPU-Time, HAT apresentou quartil inferior maior que a mediana de todos os métodos, sendo o método mais custoso em termos de memória. EFDT apresenta mediana e quartil superior maior que Hoeffding Tree. LAST, independente do detector, apresentou uso de memória similar com Hoeffding Trees.

Em resposta à pergunta **P3**, é possível afirmar que o *bound* de *Hoeffding* se mostra mais conservador conforme o número de atributos aumenta, enquanto LAST não apresenta muita alteração em número de nós. Os métodos possuem

CPU-Time similar, mas métodos com reavaliação de ramificações, como EFDT e HAT, apresentaram custo superior comparado a LAST e Hoeffding Tree por terem que armazenar estatísticas das instâncias em nós não terminais. Em especial, HAT apresenta maior custo em memória, pois além de armazenar estatísticas das instâncias em nós não terminais também instancia detectores de mudança em nós não terminais.

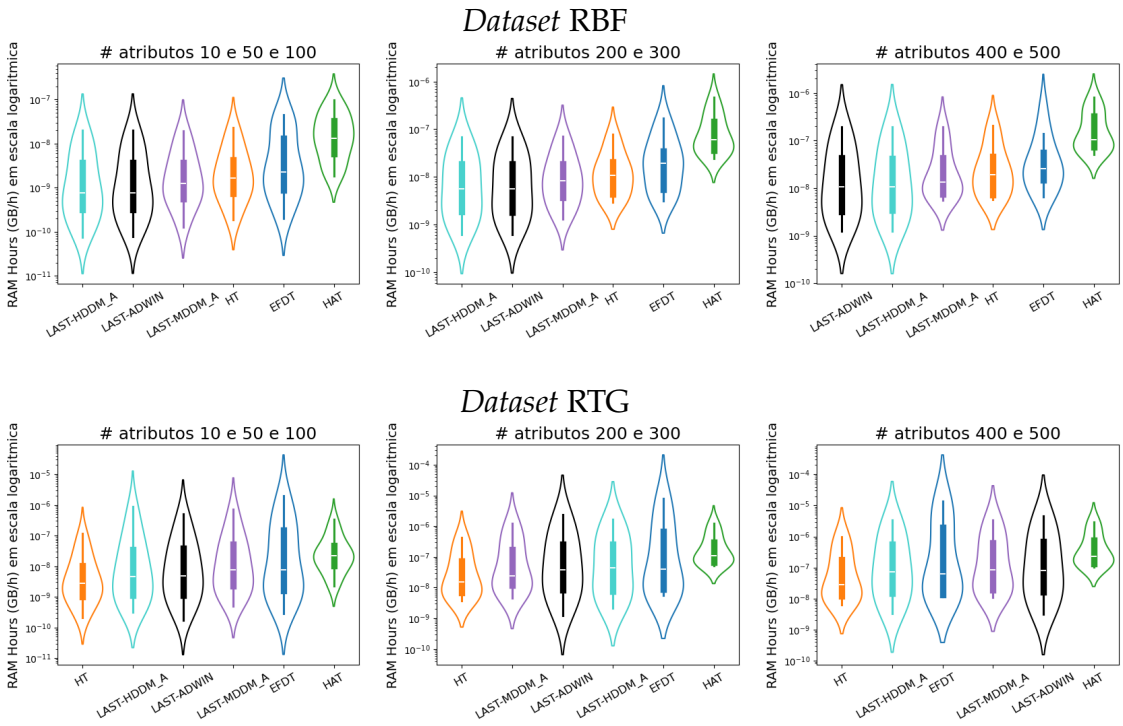

Figura 4.12: Distribuição de RAM-Hours (em GB/hora e escala logaritmica) dos *datasets* RBF e RTG

4.2.5 Efeito de Mudanças de Conceito em Árvores de Decisão Incrementais

Para abordar a pergunta **P4**, essa seção apresenta acurácia dos métodos ao longo do tempo para *datasets* com mudança de conceito. Em todas as ilustrações, linhas verticais em vermelho indicam o momento de mudança de conceito no *dataset*.

Em *datasets* com mudança abruta, seja ela de forma recorrente (Figura 4.13) ou não (Figura 4.15), LAST, independente do detector, performou melhor que HT e EFDT na maioria dos momentos ao longo dos fluxos. Enquanto em *datasets* com mudança gradual, seja ela de forma recorrente (Figura 4.14) ou não (Figura 4.16),

a diferença entre performance de HT, EFDT e LAST é menor, especialmente nos *datasets* AGR e SEA. LAST com os detectores selecionados se mostraram mais eficientes em *datasets* com mudança abrupta de conceito, em que o método se adapta melhor quando o conceito se torna estável rapidamente e as ramificações são suficientes para se adaptar a mudança. Para detectores projetados para lidar melhor com mudanças graduais, como EDDM, as Tabelas 6.1 e 6.2 mostram que o detector performa melhor em *datasets* com mudanças graduais tanto em cenários reais e sintéticos.

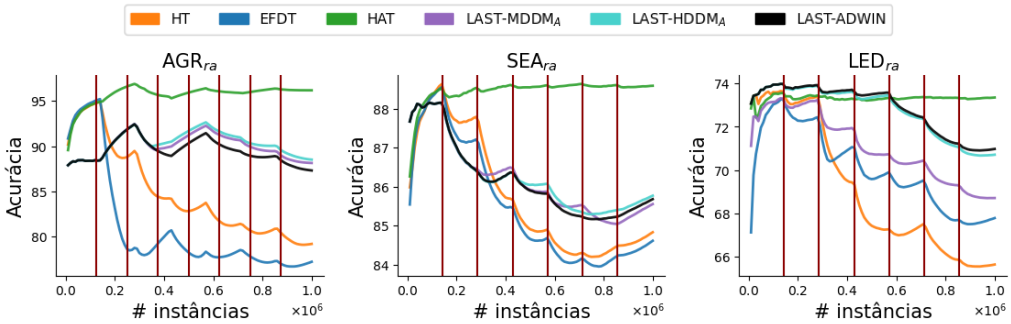

Figura 4.13: Acurácia (em %) de árvores de decisão em *datasets* que simulam mudanças de conceito recorrentes e mudança abrupta entre conceitos

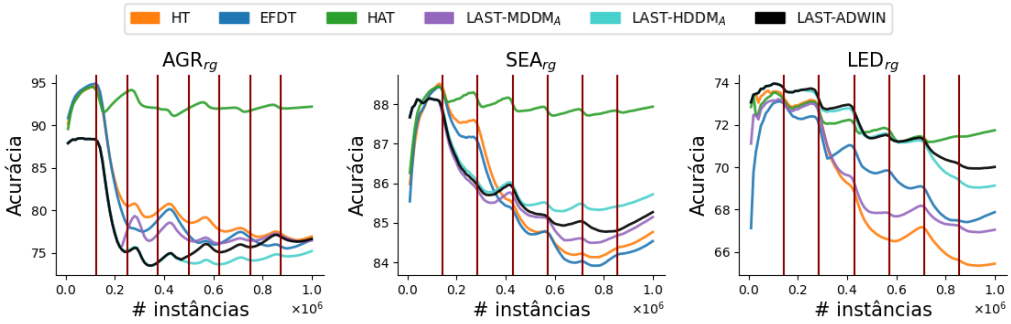

Figura 4.14: Acurácia (em %) de árvores de decisão em *datasets* que simulam mudanças de conceito recorrentes e mudança gradual entre conceitos

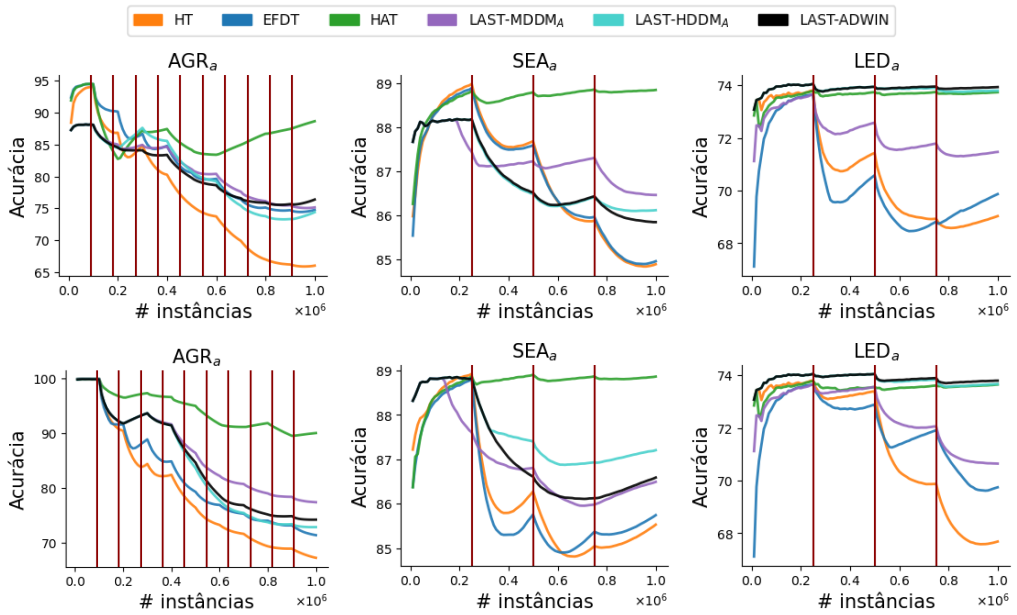

Figura 4.15: Acurácia (em %) de árvores de decisão em *datasets* que simulam mudanças de conceito abruptas

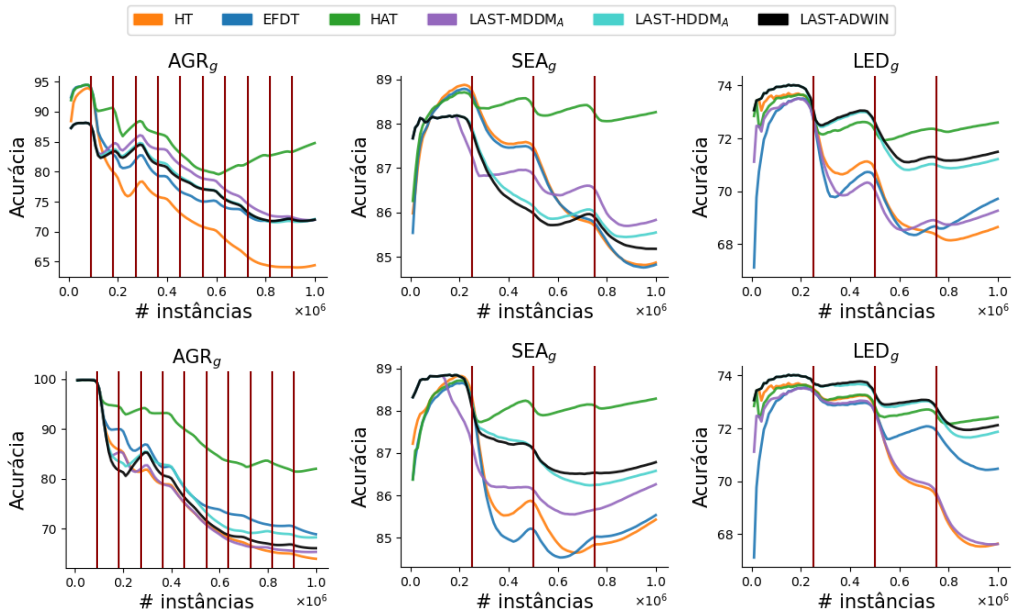

Figura 4.16: Acurácia (em %) de árvores de decisão em *datasets* que simulam mudanças de conceito graduais

Em *datasets* com mudança incremental (Figuras 4.17 e 4.18), EFDT e LAST com o detector MDDM$_A$ apresentou os melhores resultados no *dataset* RBF com mudança lenta e LAST com o detector HDDM$_A$ e ADWIN obtiveram os piores

resultados, enquanto LAST teve os melhores resultados no *dataset* HYPER com mudança lenta. Nos *datasets* RBF e HYPER com mudança moderada, LAST, independente do detector, teve resultados superiores comparados a HT, EFDT e HAT, e HT e EFDT foram os métodos com maior decaimento de acurácia. No *dataset* RBF com mudança rápida, LAST com os detectores MDDM$_A$ e HDDM$_A$ obteve os melhores resultados, enquanto HAT ultrapassou os resultados de LAST com o detector ADWIN na instância 400000, e HT e EFDT obteve os piores resultados ao longo do fluxo. No *dataset* HYPER com mudança rápida, HAT e EFDT alcançaram os melhores resultados enquanto HT e LAST obtiveram os piores.

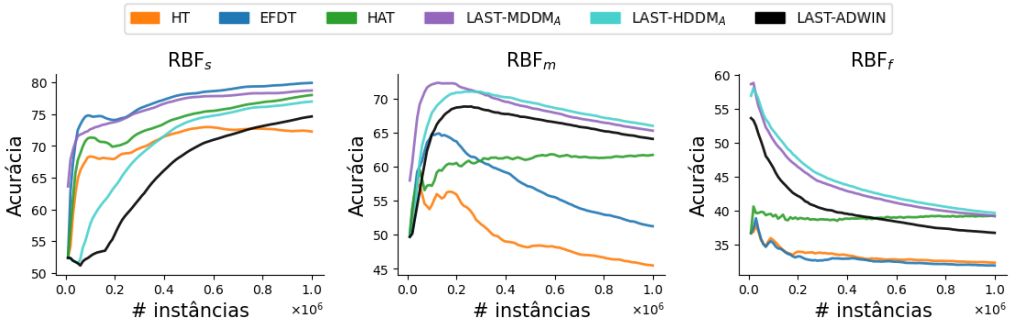

Figura 4.17: Acurácia (em %) de árvores de decisão no *dataset RBF*, que simulam mudanças de conceito incremental

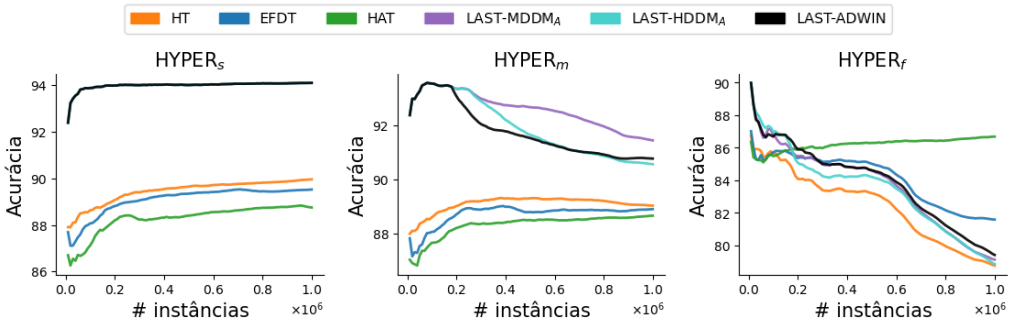

Figura 4.18: Acurácia (em %) de árvores de decisão no *dataset HYPER*, que simulam mudanças de conceito incremental

Nos *datasets* INSECTS (Figura 4.19), fica evidente a disparidade entre os resultados de HAT em *datasets* reais com mudanças de conceito e *datasets* sintéticos que simulam mudança de conceito. Mudanças de conceito em *datasets* sintéticos comumente usadas na literatura simulam mudanças aditivas ao conceito (adição ou remoção de atributos relevantes como em LED, ou adição de conceitos

como em AGR) e similares ao original (mudança no parâmetro θ em SEA). As mudanças empregadas em INSECTS são a combinação entre mudanças reais e virtuais (explicado na Seção 2.4), sendo que a alteração na temperatura resulta em mudanças em $P_t(X)$ e $P_t(y|X)$ uma vez que as observações extraídas pelo Sensor alteram com a temperatura e simulam mudanças mais desafiadoras quando comparadas àquelas simuladas em *datasets* sintéticos da literatura.

No *dataset* INSECTS com mudança abrupta, EFDT começou com os melhores resultados, até as mudanças de conceito afetarem sua performance, finalizando o fluxo entre os piores métodos. LAST com o detector $MDDM_A$ obteve os melhores resultados e LAST com o detector ADWIN e $HDDM_A$ obteve o segundo melhor resultado. HAT teve decaimento de acurácia (segundo maior, sendo HT com o maior) após a primeira mudança, enquanto isso não é observável em *datasets* sintéticos.

No *dataset* INSECTS com mudança gradual, LAST, independente do detector, teve o menor decaimento de acurácia após a mudança de conceito. Enquanto após a mudança, EFDT foi a árvore com os melhores resultados, tendo performance similar a LAST. HT e HAT obtiveram a pior acurácia em todos os momentos do fluxo.

No *dataset* INSECTS com mudança incremental, HAT e HT obtiveram os piores resultados após a instância 7000, enquanto outros métodos não apresentaram decaimento de performance.

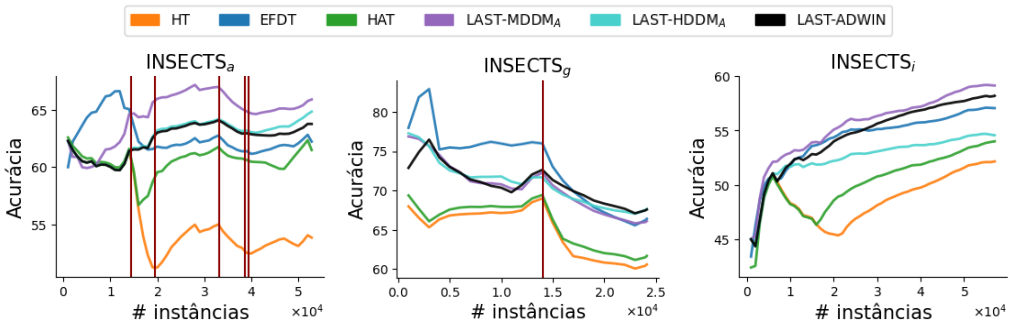

Figura 4.19: Acurácia (em %) de árvores de decisão no *dataset INSECTS*

Em resposta à pergunta **P4**, LAST apresenta resultados superiores em *datasets* com mudança de conceito abrupta, enquanto HT e EFDT conseguem aproximar-se aos resultados de LAST em mudanças de conceito graduais. LAST com detectores projetados para lidar melhor com mudanças graduais apresentaram acurácia superior. Em *datasets* com mudanças incrementais, LAST performa

melhor em *datasets* com mudança moderada. Além desse padrão, a performance varia de método em método para os outros cenários com mudança incremental. Também foi apontado que HAT performa bem em mudanças simuladas na literatura, porém demonstra um caso mais desafiador de mudança em *dataset* real.

4.2.6 Árvores de Decisão com Mecanismos Periódicos e Adaptativos

Para abordar a pergunta **P5**, essa seção apresenta um comparativo entre árvores de decisão com mecanismo de ramificação periódico, adaptativos e a combinação dos dois métodos materializados em HLAST e EFLAST.

A Figura 4.20 apresenta um teste de Wilcoxon para acurácia média ao longo do fluxo. LAST ainda apresenta o melhor *ranking* em acurácia, não apresentando resultados estatisticamente diferentes de EFLAST. HLAST e EFLAST$_D$ apresentam resultados superiores estatisticamente diferentes de EFDT, porém apresentam resultados inferiores e estatisticamente diferentes comparados com LAST.

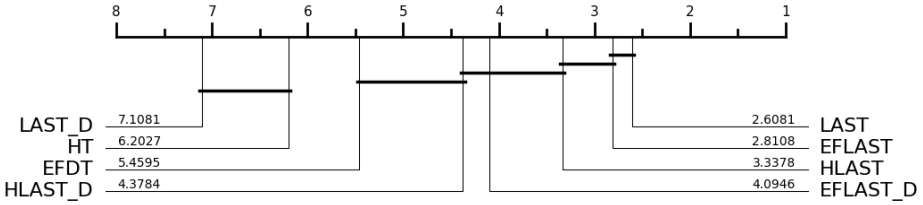

Figura 4.20: Ranking da Acurácia (em %) média ao longo do fluxo para árvores periódicas, adaptativas e combinação dos dois métodos

A Figura 4.21 apresenta um teste de Wilcoxon para tamanho da árvore (número de nós). LAST ainda apresenta resultados não estatisticamente diferentes de EFDT. HLAST é o único método que não apresenta diferença estatística comparado ao LAST, enquanto HLAST$_D$, EFLAST e EFLAST$_D$ apresentam resultados inferiores com diferença estatística comparados com LAST.

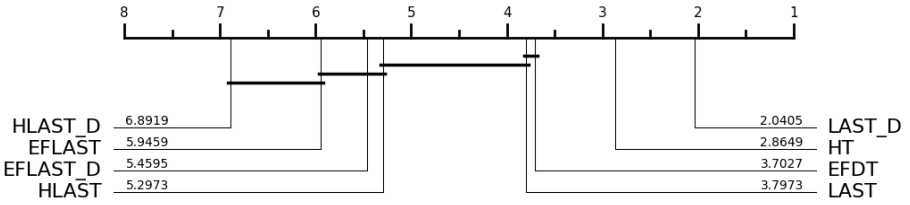

Figura 4.21: Ranking de tamanho de árvore de árvores periódicas, adaptativas e combinação dos dois métodos

A Tabela 4.4 apresenta o *ranking* médio das métricas acurácia, Kappa$_M$, Kappa$_T$, número de nós, *CPU-Time* e *RAM-Hours* de árvores de decisão periódicas, adaptativas e combinação dos 2 métodos. HLAST, em *datasets* reais, obteve acurácia, Kappa$_M$ e Kappa$_T$ superior comparados a HT, porém inferiores a LAST. Porém, apresenta custo computacional inferior a LAST e superior a HT. Com a distribuição de mais detectores por mais nós, a detecção que anteriormente seria realizada por LAST não é realizada pois o *input* que um detector receberia é distribuído por outros detectores em mais nós folhas. Nesse caso, HLAST mostra um *trade-off* entre acurácia e custo computacional. Em *datasets* sintéticos, HLAST também obteve acurácia, Kappa$_M$ e Kappa$_T$ superior a HT e inferior a LAST, porém obteve tamanho de árvore superior a LAST e HT. Em cenários sintéticos com mudança de conceito, principalmente com mudanças aditivas (como explicado na Seção 4.2.5), instâncias de conceito diferente são propagadas por mais nós, ocasionando em mais detecções em nós folhas e aumento do tamanho da árvore.

HLAST$_D$, em *datasets* reais, obteve acurácia, Kappa$_M$ e Kappa$_T$ similar a HLAST e obteve custo computacional superior a HLAST. O crescimento da árvore por ramificações periódicas ocasiona em mais alterações na distribuição de classes nos nós folhas ao longo do tempo, ocasionando em maior crescimento da árvore. Em *datasets* sintéticos, HLAST$_D$ obteve qualidade preditiva inferior a HLAST, custo computacional superior a HLAST e o pior tamanho de árvore entre todas as árvores. Além da alteração da distribuição por ramificações periódicas, como em *datasets* reais, instâncias de conceito diferente são propagadas por mais nós, ocasionando em mais detecções em nós folhas e aumento do tamanho da árvore como em HLAST.

EFLAST apresentou *ranking* superior em acurácia em *datasets* comparados com LAST, porém obteve empate em Kappa$_M$ e Kappa$_T$, o que indica que não

houve superioridade em relação em casos de classe minoritária e classificação por classe da instância anterior. Todavia, obteve os piores custos computacionais e maiores tamanhos de árvores entre as árvores de decisão avaliadas. Em *datasets* sintéticos, apesar de apresentar o segundo melhor *ranking* em acurácia, Kappa$_M$ e Kappa$_T$, apresentou resultados inferiores a LAST. Em custo computacional, também apresentou entre os piores resultados. Custo computacional alto de EFLAST é esperado dado que as ramificações periódicas de EFDT tem condições de ramificações menos restritas que Hoeffding Trees, que ocasionam o aumento do tamanho da árvore.

EFLAST$_D$ apresentou resultados inferiores tanto em qualidade preditiva e custo computacional, tanto em *datasets* reais e sintéticos.

Tabela 4.4: *Ranking* médio de métricas de árvores de decisão periódicas, adaptativas e a combinação dos 2 métodos.

Tipo do *Dataset*		HT	EFDT	LAST	LAST$_D$
Real	Acc	7,00(8)	5,15(6)	2,85(2)	6,88(7)
	Kappa$_M$	7,08(8)	5,15(6)	2,77(2)	6,81(7)
	Kappa$_T$	7,08(8)	5,08(6)	2,77(2)	6,81(7)
	Size	**1,38(1)**	3,92(4)	5,69(6)	3,19(2)
	CPU-Time	2,65(2)	5,85(6)	3,92(4)	2,88(3)
	Total-Time	66,66(2)	93,95(6)	82,25(5)	**64,02(1)**
	Ram-Hours(10^{-6})	**1,38(1)**	4,92(5)	4,85(4)	3,38(3)
Sintético	Acc	5,77(7)	5,62(6)	**1,00(1)**	7,23(8)
	Kappa$_M$	5,77(7)	5,62(6)	**2,52(1)**	7,23(8)
	Kappa$_T$	5,77(7)	5,67(6)	**2,48(1)**	7,23(8)
	Size	3,67(4)	3,58(3)	2,77(2)	**1,42(1)**
	CPU-Time	3,92(4)	5,04(6)	1,92(2)	**1,67(1)**
	Total-Time	153,16(3)	191,53(6)	133,06(2)	**107,48(1)**
	Ram-Hours(10^{-6})	3,50(3)	3,71(4)	2,25(2)	**1,46(1)**

Tabela 4.4: Continuada.

Tipo do *Dataset*		HLAST	HLAST$_D$	EFLAST	EFLAST$_D$
	Acc	3,77(4)	3,62(3)	**1,00(1)**	4,12(5)
	Kappa$_M$	3,69(4)	3,69(4)	2,77(2)	4,04(5)
	Kappa$_T$	3,69(3)	3,77(4)	2,77(2)	4,04(5)
Real	Size	3,35(3)	6,69(8)	6,08(7)	5,69(6)
	CPU-Time	**2,50(1)**	4,73(5)	6,46(7)	7,00(8)
	Total-Time	70,52(3)	80,77(4)	108,19(7)	110,86(8)
	Ram-Hours(10^{-6})	2,69(2)	6,00(6)	6,46(8)	6,31(7)
	Acc	3,10(3)	4,79(5)	2,92(2)	4,08(4)
	Kappa$_M$	3,02(3)	4,79(5)	2,96(2)	4,08(4)
	Kappa$_T$	3,06(3)	4,79(5)	2,92(2)	4,08(4)
Sintético	Size	6,35(7)	7,00(8)	5,88(6)	5,33(5)
	CPU-Time	3,85(3)	4,75(5)	7,54(8)	7,31(7)
	Total-Time	165,16(4)	179,27(5)	252,22(8)	231,81(7)
	Ram-Hours(10^{-6})	5,75(5)	6,54(7)	6,67(8)	6,12(6)

A Figura 4.22 apresenta a distribuição de *CPU-Time* e *RAM-Hours* para as árvores avaliadas nessa Seção, sendo que as árvores estão ordenas pelo valor da mediana. Em *CPU-Time*, tanto em *datasets* reais e sintéticos, HLAST apresentou quartil inferior, mediana e quartil superior menor que EFDT, enquanto EFLAST, EFLAST$_D$ se apresentam como os métodos que mais exigiram tempo de processamento.

Para *RAM-Hours*, em *datasets* reais, HLAST demonstra quartil inferior, mediana e quartil superior menor que LAST e maior que HT, enquanto HLAST$_D$ demonstra resultados similares a EFDT. Em *datasets* sintéticos, EFDT apresenta *RAM-Hours* inferior a HLAST, porém difere de HLAST$_D$, EFLAST e EFLAST$_D$, que apresentam maior densidade entre 10^{-6} e 10^{-5}.

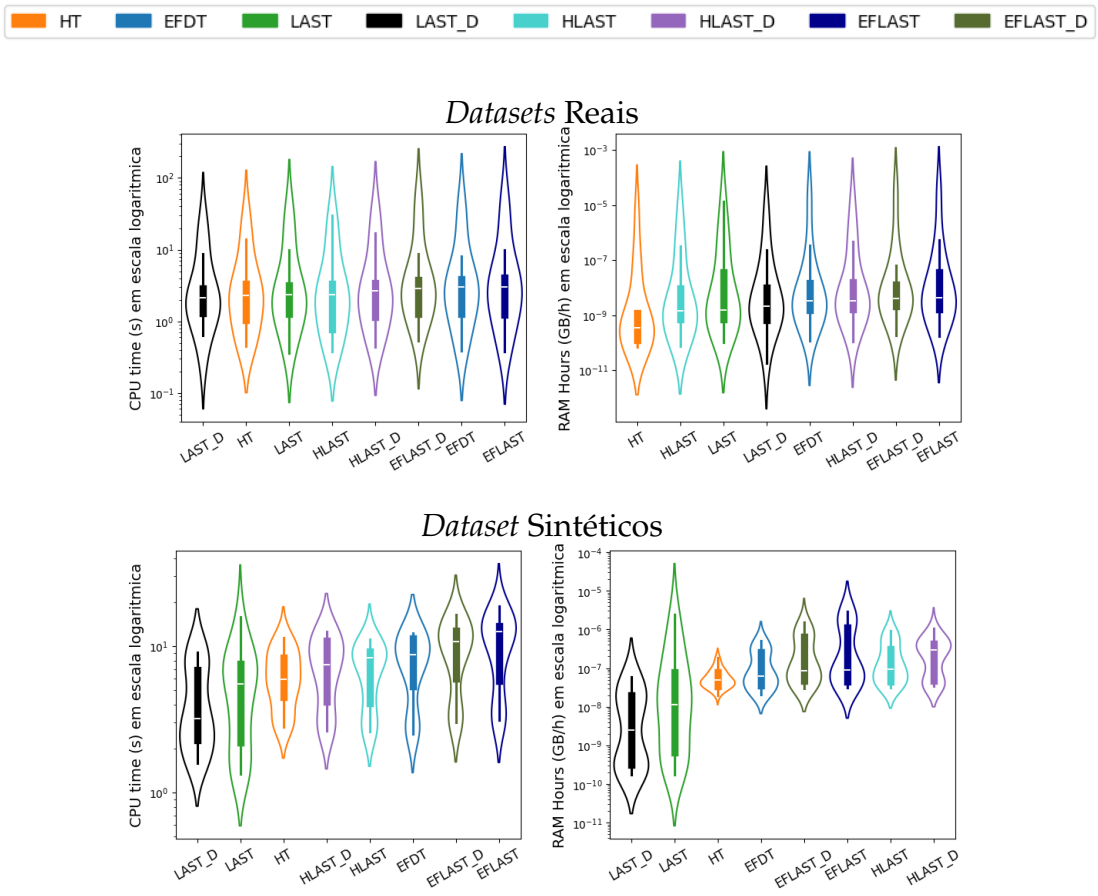

Figura 4.22: Distribuição de CPU-Time (esquerda) e RAM-Hours (direita) de árvores periódicas, adaptativas e combinação dos dois métodos em *datasets* reais e sintéticos

A Figura 4.23 apresenta a acurácia ao longo do tempo das árvores de decisão avaliadas nessa Seção em *datasets* com mudança de conceito. Em datasets sinté-

ticos, HLAST e HLAST$_D$ obteve acurácia similar ao longo do tempo, enquanto EFLAST e EFLAST$_D$ obtiveram resultados similares a EFDT e superiores a árvores com mecanismo periódico de Hoeffding de (DOMINGOS; HULTEN, 2000). Nos *datasets* reais com mudança de conceito INSECTS, HT e LAST$_D$ apresentou os piores resultados, enquanto os outros métodos foram capazes de obter resultados superiores e curva de acurácia similar ao longo do tempo.

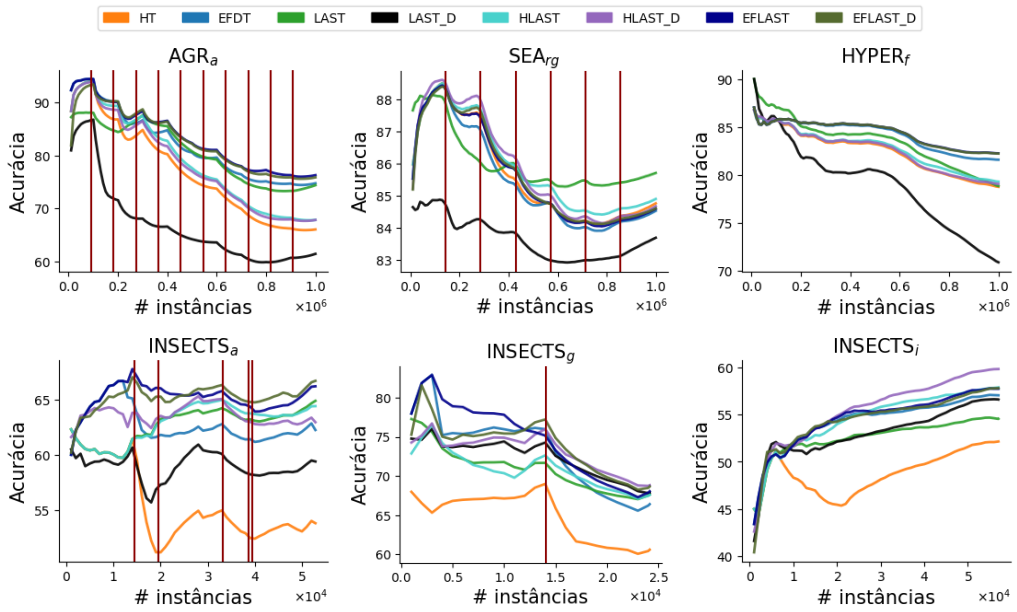

Figura 4.23: Acurácia (em %) de árvores periódicas, adaptativas e combinação dos dois métodos em *datasets* com mudanças de conceito

Em resposta à pergunta **P5**, HLAST se apresentou como um intermédio entre HT e LAST, apresentando qualidade preditiva e custo computacional superior a HT e qualidade preditiva e custo computacional inferior (maioria dos casos) a LAST. HLAST apresenta também diferença estatística com *ranking* superior a EFDT simultaneamente com diferença estatística e *ranking* superior em tamanho de árvore. Em *datasets* sintéticos, foi observado que por obter mais nós, árvores com combinação dos métodos de ramificação periódico e adaptativo tendem a crescer mais, dado que instâncias de um conceito diferente atingem mais nós e ocasionam na detecção de mudança por mais nós folhas. Em árvores que monitoram a distribuição de classes, foi apontado que mais alterações ocorrem na distribuição de classes com mais nós folhas na árvore. EFLAST e EFLAST$_D$ obtiveram o maior custo computacional e tamanho de árvore. Nos cenários de mudança de conceito apresentados, HLAST e HLAST$_D$ obtiveram resultados similares a HT em datasets sintéticos, enquanto obteve resultados inferiores

a EFDT, EFLAST e EFLAST$_D$, que por sua vez obtiveram resultados similares. Nos *datasets* reais com mudança de conceito INSECTS, HT e LAST$_D$ obtiveram os piores resultados, enquanto os outros métodos obtiveram performance superior a HT e similar entre si.

4.3 Resultados para o Ambiente *Ensemble*

Essa seção tem como objetivo atestar a qualidade preditiva do algoritmo LAST como *base learner* de *ensembles*. Os algoritmos avaliados são LAST e as propostas que mitigam a limitação do LAST como *base learners* de *ensembles* (detalhadas na Seção 3.1.1), considerando qualidade preditiva, custo computacional e reação a mudança de conceito.

A Subseção 4.3.1 apresenta o protocolo experimental para os experimentos dessa seção. As subseções 4.3.2 - 4.3.5 apresentam os resultados obtidos para os algoritmos propostos. A Subseção 4.3.6 apresenta um comparativo entre ensembles com o *base learner* respectivo que obteve o melhor resultado.

4.3.1 Protocolo Experimental para o Ambiente *Ensemble*

A qualidade preditiva dos algoritmos é avaliada a partir da acurácia dos modelos (descrito na Seção 2.2) através da estratégia de validação *prequential* (descrito na Seção 2.3). Todos os experimentos para o ambiente ensemble foram realizados no *framework* MOA 24.01 (BIFET et al., 2010) com uma máquina Intel(R) Xeon(R) CPU E5649 @ 2.53GHz com 32 GB de RAM.

Em (MANAPRAGADA et al., 2022), os autores realizam uma análise comparativa entre *EFDT* e *HT* como *base learners* de *ensembles*. Esta Seção tem um objetivo similar, e analisa como as árvores propostas nesse trabalho podem contribuir para o desenvolvimento de ensembles mais robustas e eficientes.

Todas as *ensembles* foram configuradas com 100 *base learners*, como utilizado em *ensembles* estado-da-arte recentes (GOMES et al., 2017b; GOMES; READ; BIFET, 2019). Todos os hiper-parâmetros das *ensembles* foram configuradas como padrão do *framework* MOA (BIFET et al., 2010), até mesmo os hiper-parâmetros adotados nos *base learners* de cada *ensemble*. *Streaming Random Patches*(SRP) e *Streaming Random Subspaces* (SRS) (GOMES; READ; BIFET, 2019) foram adicionados às experimentações, não tendo sido realizado experimentos com essas ensembles em (MANAPRAGADA et al., 2022). Em (MANAPRAGADA et al.,

2022), os autores também não apresentam resultados referentes a custo de memória e tempo de processamento, mesmo sendo aspectos fundamentais em mineração de fluxo de dados (como descrito na Seção 2.1), sendo esses aspectos incluídos nesse trabalho.

Dessa forma, os experimentos contemplam as principais *ensembles* estado-da-arte (PAIM; ENEMBRECK, 2024a), sendo eles: Adaptive Random Forest (ARF), Streaming Random Patches (SRP), Adaptive Regularized Ensemble (ARE) e Adaptive Random Tree Ensemble (ARTE). Streaming Gradient Boosting Trees (GUNASEKARA et al., 2024) é um método proposto recentemente, análogo ao método de estado-da-arte para o ambiente *batch* (CHEN; GUESTRIN, 2016), porém devido ao alto custo computacional e não ser capaz de executar diversos *datasets* por falta de memória, foi retirado das experimentações desse trabalho.

A combinação de LAST com Hoeffding Trees é denotada como HLAST, enquanto a combinação de EFDT com LAST é denotada como EFLAST. Todas as árvores utilizaram o detector HDDM_A, por demonstrar os melhores resultados na Seção 4.2.

Os resultados são apresentados na seguinte ordem. Primeiro são apresentados testes de Wilcoxon com *rankings* da acurácia e tamanho de nós das árvores, seguido de uma tabela com o *ranking* médio das métricas acurácia, Kappa_M, Kappa_T, tamanho de árvore, *CPU-Time*, *Total-Time* (tempo total para processamento de todos os *datasets*) e *RAM-Hours*. Posteriormente, são apresentadas as distribuições de *CPU-Time* e *RAM-Hours* via um *violin plot* em *datasets* reais e sintéticos, seguidas de uma ilustração da acurácia ao longo do fluxo de 6 *datasets* com mudança de conceito, sendo eles AGR_a com 10 mudanças de conceito, SEA_{rg} com 6 mudanças de conceito, Hyper_f, INSECTS_a, INSECTS_g e INSECTS_i.

4.3.2 *Adaptive Random Forest*

A Figura 4.24 apresenta um teste de Wilcoxon para acurácia média ao longo do fluxo. LAST e LAST_D apresentaram os piores resultados em *ranking*, dado que induzem baixa diversidade na *ensemble*, como detalhado na Subseção 3.1.1, e será demonstrado nessa Subseção. HLAST_D foi o único método que mostrou diferença estatística comparado a EFDT, apesar não possuir diferença estatística de HT. Os outros métodos não apresentaram diferença estatística entre sí.

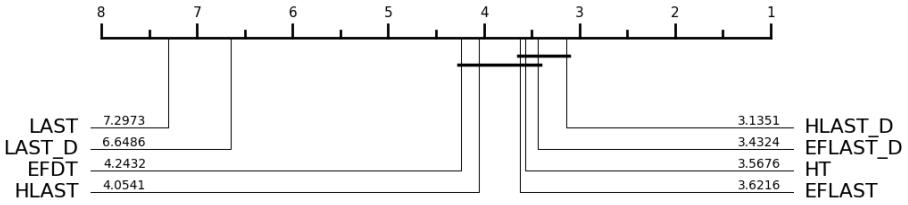

Figura 4.24: Ranking da Acurácia (em %) média ao longo do fluxo de árvores periódicas, adaptativas e combinação dos dois métodos como *base learners* da *ensemble Adaptive Random Forest*.

A Figura 4.25 apresenta um teste de Wilcoxon para tamanho da árvore (número de nós). LAST e LAST$_D$ apresentaram o menor tamanho de árvore. HLAST e HLAST$_D$ apresentaram maior tamanho de árvore, e diferença estatística comparado a HT. EFLAST e EFLAST$_D$ apresentaram *ranking* de tamanho de árvore superior a HT com diferença estatística, porém apresentaram *ranking* de tamanho de árvore inferior a EFDT com diferença estatística.

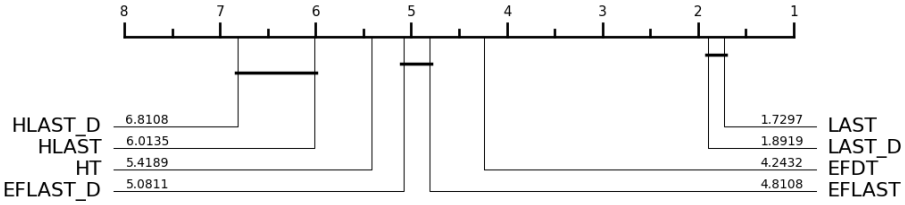

Figura 4.25: Ranking de tamanho de árvores periódicas, adaptativas e combinação dos dois métodos como *base learners* da *ensemble Adaptive Random Forest*.

A Tabela 4.5 apresenta a média e desvio padrão de tamanho de árvore de árvores periódicas em *datasets* reais e sintéticos, adaptativas e combinação dos dois métodos como *base learners* da *ensemble Adaptive Random Forest*. LAST e LAST$_D$ são os únicos métodos com desvio padrão inferior a média do tamanho da árvore, demonstrando menor diversidade em comparação com outros *base learners*.

A Tabela 4.6 apresenta o *ranking* médio das métricas acurácia, Kappa$_M$, Kappa$_T$, número de nós, *CPU-Time* e *RAM-Hours* de *base learners* da *ensemble Adaptive Random Forest*. HLAST$_D$ apresentou os melhores resultados em qualidade preditiva em *datasets* reais e segundo melhor *ranking* em *datasets* sintéticos,

Tabela 4.5: Média e desvio padrão de tamanho de árvore de *base learners* da *ensemble Adaptive Random Forest*

HT	EFDT	LAST	$LAST_D$
683,91(3836,14)	344,20(717,44)	40,70(22,00)	10,27(9,22)
HLAST	$HLAST_D$	EFLAST	$EFLAST_D$
1062,42(8062,94)	1106,77(8109,61)	347,15(812,37)	359,33(782,68)

enquanto apresenta custo computacional inferior a EFLAST e $EFLAST_D$ e apresenta o maior tamanho de árvore. Como abordado em (GOMES et al., 2017b), árvores mais profundas são aceitáveis, dado que mesmo que algumas árvores individuais tenham *overfitting*, a redução de variância na construção de vários *base learners* mitiga o *overfitting* da *ensemble* inteira. HLAST apresentou *ranking* de custo computacional similar a $HLAST_D$, porém com qualidade preditiva inferior e tamanho de árvore inferior.

Tabela 4.6: *Ranking* médio de métricas de árvores periódicas, adaptativas e combinação dos dois métodos como *base learners* da *ensemble Adaptive Random Forest*

Tipo do *Dataset*		HT	EFDT	LAST	$LAST_D$
Real	Acc	4,77(6)	3,38(2)	6,62(8)	5,69(7)
	$Kappa_M$	4,77(6)	3,38(2)	6,62(8)	5,69(7)
	$Kappa_T$	4,77(6)	3,38(2)	6,62(8)	5,69(7)
	Size	4,81(4)	5,31(5)	**2,15(1)**	2,62(2)
	CPU-Time	3,77(3)	5,31(5)	1,62(2)	**1,46(1)**
	Total-Time	9790,20(3)	11802,97(4)	3940,89(2)	**3573,25(1)**
	Ram-Hours(10^{-6})	3,46(3)	5,15(5)	1,85(2)	**1,31(1)**
Sintético	Acc	**1,00(1)**	4,71(6)	7,67(8)	7,17(7)
	$Kappa_M$	**2,92(1)**	4,67(6)	7,67(8)	7,17(7)
	$Kappa_T$	**2,92(1)**	4,67(6)	7,67(8)	7,17(7)
	Size	5,75(6)	3,67(3)	1,50(2)	1,50(2)
	CPU-Time	5,54(5)	4,33(3)	1,54(2)	**1,46(1)**
	Total-Time	41099,27(3)	49940,81(6)	**12517,27(1)**	13530,61(2)
	Ram-Hours(10^{-6})	4,46(4)	4,12(3)	**1,46(1)**	1,54(2)

Tabela 4.6: Continuada

Tipo do *Dataset*		HLAST	HLAST$_D$	EFLAST	EFLAST$_D$
	Acc	4,46(5)	**1,00(1)**	3,85(3)	3,92(4)
	Kappa$_M$	4,46(5)	**3,31(1)**	3,92(4)	3,85(3)
	Kappa$_T$	4,46(5)	**3,31(1)**	3,92(4)	3,85(3)
Real	Size	4,42(3)	5,62(8)	5,46(6)	5,62(8)
	CPU-Time	5,77(6)	4,23(4)	7,00(8)	6,85(7)
	Total-Time	12490,95(6)	12229,44(5)	13949,69(8)	13886,06(7)
	Ram-Hours(10^{-6})	5,46(6)	4,38(4)	7,15(7)	7,23(8)
	Acc	3,83(5)	3,04(2)	3,50(4)	3,17(3)
	Kappa$_M$	3,83(5)	3,08(2)	3,50(4)	3,17(3)
	Kappa$_T$	3,83(5)	3,08(2)	3,50(4)	3,17(3)
Sintético	Size	6,88(7)	7,46(8)	4,46(4)	4,79(5)
	CPU-Time	5,79(6)	6,12(7)	6,25(8)	4,96(4)
	Total-Time	49712,75(5)	49682,75(4)	55230,11(8)	53146,81(7)
	Ram-Hours(10^{-6})	6,21(7)	6,12(6)	6,50(8)	5,58(5)

A Figura 4.26 apresenta a distribuição de *CPU-Time* e *RAM-Hours* para as árvores avaliadas nessa Seção, sendo que as árvores estão ordenas pelo valor da mediana. Tanto para *CPU-Time* e *RAM-Hours*, em *datasets* reais, HLAST e HLAST$_D$ apresentam mediana inferior a EFDT, EFLAST e EFLAST$_D$, sendo também notável na distribuição. Em *CPU-Time* em *datasets* sintéticos, EFDT, EFLAST e EFLAST$_D$ apresentam 25% de casos abaixo de 10^{-3} (abaixo do quartil inferior), sendo que HLAST e HLAST$_D$ não apresentam casos abaixo de 10^3. Todavia, EFDT, EFLAST e EFLAST$_D$ apresenta quartil superior maior que HLAST e HLAST$_D$. Em *RAM-Hours* em *datasets* sintéticos, EFDT, EFLAST e EFLAST$_D$ também apresentaram quartil inferior menor e quartil superior maior que HT, HLAST e HLAST$_D$.

A Figura 4.23 apresenta a acurácia ao longo do tempo das árvores de decisão como *base learners* da *ensemble Adaptive Random Forest* em *datasets* com mudança de conceito. Desconsiderando LAST e LAST$_D$, todos os métodos apresentaram acurácia similar, apesar de alguns casos uma árvore apresentar performance inferior a outras árvores.

Em conclusão, HLAST$_D$ é o único método que apresenta diferença estatística comparado a EFDT em acurácia. Também apresenta melhor *ranking* em qualidade preditiva em *datasets* reais e segundo melhor *ranking* em *datasets* sintéticos, enquanto apresenta custo computacional inferior a EFLAST e EFLAST$_D$ e similar a EFDT. Em cenários de mudança de conceito, não foram observadas diferenças entre maioria dos métodos, apesar de em alguns casos um método apresentar performance inferior.

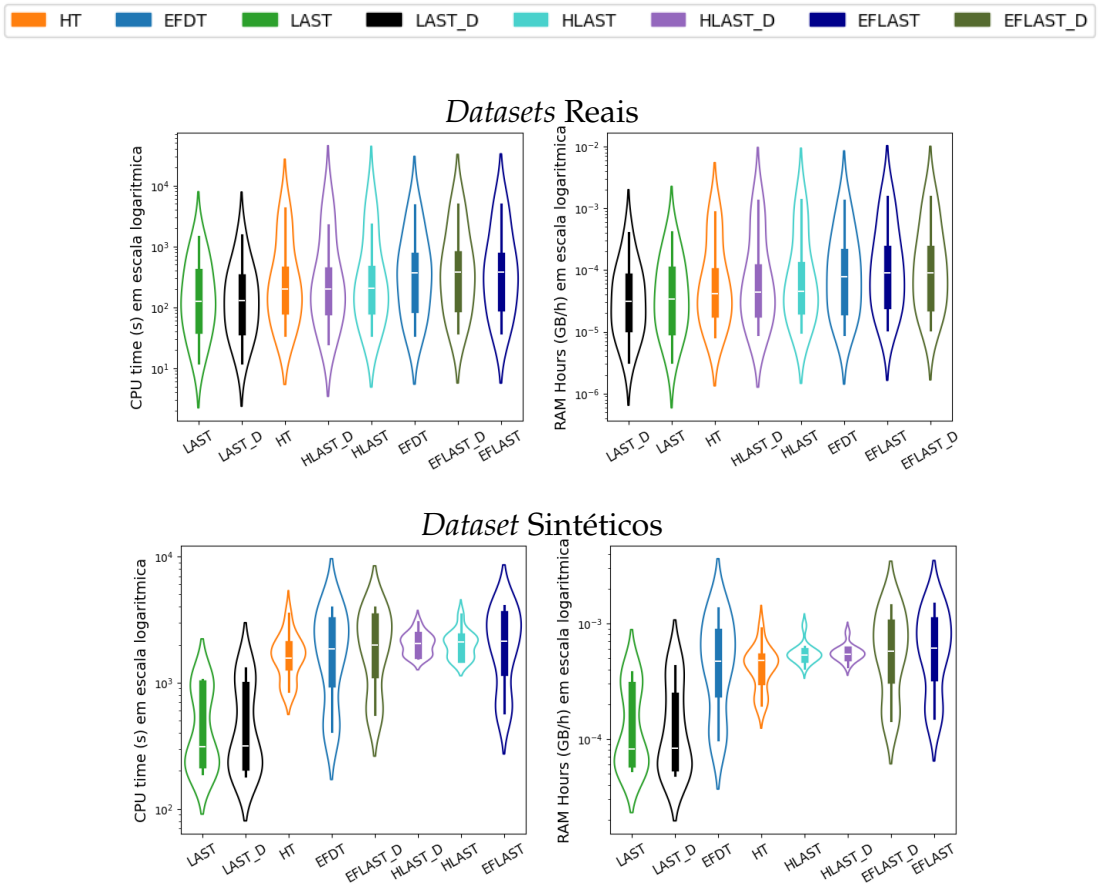

Figura 4.26: Distribuição de CPU-Time (esquerda) e RAM-Hours (direita) de árvores periódicas, adaptativas e combinação dos dois métodos como *base learners* da *ensemble Adaptive Random Forest* em *datasets* reais e sintéticos

4.3.3 *Streaming Random Patches*

A Figura 4.28 apresenta o *ranking* de acurácia dos *base learners* da *ensemble Streaming Random Patches*. LAST com monitoramento de erro apresenta melhores resultados em *Streaming Random Patches* comparado a outras *ensembles*. A seleção de atributos randômica global pode ocasionar em árvores treinadas com atributos menos importantes a crescerem mais que outras árvores treinadas com atributos mais importantes para o problema, gerando diversidade na *ensemble*, enquanto LAST_D apresentam piores resultados, sendo que não há muita alteração na distribuição de dados nos *datasets*. Diferentemente do comportamento de LAST e LAST_D, combinações de LAST com HT e EFDT resultam em piores resultados, enquanto combinações de LAST_D com HT e EFDT resultam em melhores resultados. Como discutido anteriormente, o argumento para cresci-

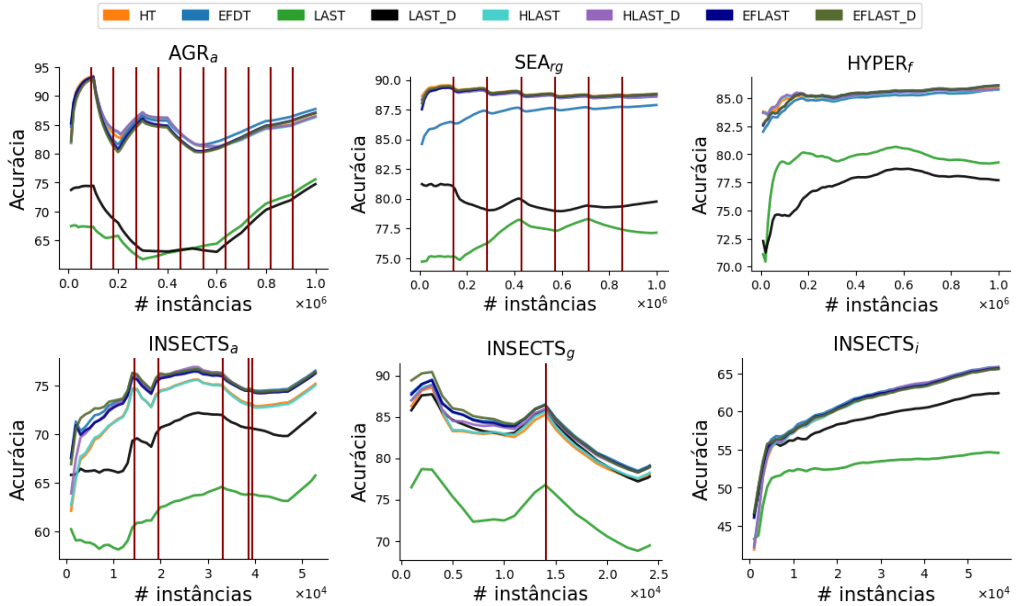

Figura 4.27: Acurácia (em %) de árvores periódicas, adaptativas e combinação dos dois métodos como *base learners* da *ensemble Adaptive Random Forest* em *datasets* com mudanças de conceito

mento maior de combinações de LAST$_D$ com HT e EFDT e maior diversidade induzida entre *learners* é justificado por ramificações periódicas ocasionarem maior mudança na distribuição dos dados, sendo mais adequado para *ensemble Streaming Random Patches*. Todavia, para combinações de LAST com HT e EFDT, pode ocorrer um equilíbrio entre ramificações periódicas e monitoramento de erro que não resultem em muito crescimento das árvores. LAST, HT, EFDT, combinações de LAST$_D$ com HT e EFDT, e EFLAST não apresentaram diferença estatística, porém HT apresentou o melhor *ranking*.

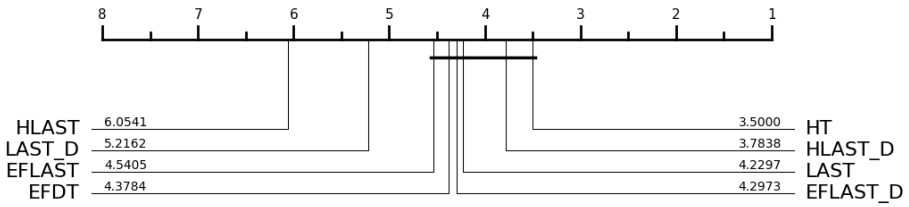

Figura 4.28: Ranking da Acurácia (em %) média ao longo do fluxo de árvores periódicas, adaptativas e combinação dos dois métodos como *base learners* da *ensemble Streaming Random Patches*.

A Figura 4.25 apresenta um teste de Wilcoxon para tamanho da árvore (nú-

mero de nós). Combinações de LAST com EFDT e EFDT não apresentaram diferença estatística, enquanto apresentam *rakning superior* a HT e LAST. HT e LAST não apresentaram diferença estatística entre si, e apresentaram *ranking* superior a HLAST$_D$ com diferença estatística.

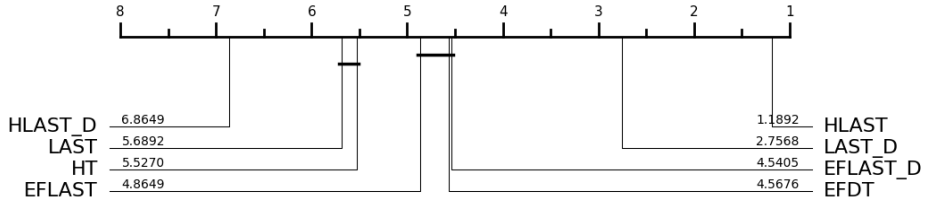

Figura 4.29: Ranking de tamanho de árvores periódicas, adaptativas e combinação dos dois métodos como *base learners* da *ensemble Streaming Random Patches*.

A Tabela 4.7 apresenta o *ranking* médio de métricas de qualidade preditiva e custo computacional de *Streaming Random Patches* por *base learner*. Em *datasets* reais, HLAST$_D$ apresentou *ranking* de acurácia superior a HT, apesar de obter *ranking* em Kappa$_M$ e Kappa$_T$ similares. Em *datasets* sintéticos, HLAST$_D$ obteve *ranking* de acurácia inferior a HT e *ranking* de Kappa$_M$ e Kappa$_T$ similares. EFDT apresentou menor *ranking* de custo computacional que HT em *datasets* sintéticos e *ranking* de qualidade preditiva empatada com HLAST$_D$. Em *datasets* reais, EFDT obteve custo computacional similar a HT e *ranking* de qualidade preditiva inferior. EFLAST e EFLAST$_D$ obtiveram *ranking* de qualidade preditiva superior a EFDT (porém não melhor que HLAST$_D$ e HT) em *datasets* reais, e obtiveram *ranking* de qualidade preditiva inferior a EFDT em *datasets* sintéticos. Em ambos os tipos de *datasets*, EFLAST e EFLAST$_D$ apresentaram menor *ranking* de memória e *ranking* de CPU-Time similar.

A Figura 4.30 apresenta a distribuição de *CPU-Time* e *RAM-Hours* para as árvores avaliadas nessa seção, sendo que as árvores estão ordenas pelo valor da mediana. Em *datasets* reais, tanto para *CPU-Time* e *RAM-Hours*, HT, LAST e HLAST$_D$ apresentaram mediana e quartil superior inferiores a EFDT, EFLAST, EFLAST$_D$. Em *datasets* sintéticos, tanto para *CPU-Time* e *RAM-Hours*, HT, LAST e HLAST$_D$ apresentaram quartil inferior, mediana e quartil superior maior que EFDT, EFLAST e EFLAST$_D$.

A Figura 4.31 apresenta a acurácia ao longo do tempo das árvores de decisão como *base learners* da *ensemble Streaming Random Patches* em *datasets* com mu-

Tabela 4.7: *Ranking* médio de métricas de árvores periódicas, adaptativas e combinação dos dois métodos como *base learners* da *ensemble Streaming Random Patches*

Tipo do *Dataset*		HT	EFDT	LAST	$LAST_D$
	Acc	3,50(2)	4,77(6)	4,04(3)	6,08(8)
	$Kappa_M$	3,50(2)	4,62(6)	4,04(3)	6,08(8)
	$Kappa_T$	3,42(2)	4,85(6)	3,96(3)	6,08(8)
Real	Size	4,35(3)	5,54(6)	4,81(4)	2,85(2)
	CPU-Time	5,15(4)	5,15(4)	5,54(6)	1,54(2)
	Total-Time	11996,25(6)	11213,17(3)	14181,64(8)	5062,22(2)
	Ram-Hours(10^{-6})	4,54(3)	4,69(4)	5,77(6)	**1,46(1)**
	Acc	**1,00(1)**	4,17(2)	4,33(5)	4,75(7)
	$Kappa_M$	**3,58(1)**	4,17(2)	4,38(5)	4,75(7)
	$Kappa_T$	**3,58(1)**	4,17(2)	4,33(5)	4,75(7)
Sintético	Size	6,17(6)	4,04(4)	6,17(6)	2,71(2)
	CPU-Time	4,88(4)	4,54(3)	6,33(7)	1,96(2)
	Total-Time	67061,08(6)	58853,53(3)	80358,31(7)	22372,72(2)
	Ram-Hours(10^{-6})	4,88(4)	4,08(3)	6,33(7)	1,92(2)

Tabela 4.7: Continuada

Tipo do *Dataset*		HLAST	$HLAST_D$	EFLAST	$EFLAST_D$
	Acc	5,85(7)	**1,00(1)**	4,38(5)	4,31(4)
	$Kappa_M$	5,85(7)	**3,08(1)**	4,46(5)	4,38(4)
	$Kappa_T$	5,85(7)	**3,08(1)**	4,38(4)	4,38(4)
Real	Size	**1,54(1)**	5,92(8)	5,46(5)	5,54(6)
	CPU-Time	**1,46(1)**	5,23(5)	5,85(7)	6,08(8)
	Total-Time	**4550,94(1)**	13606,19(7)	11274,84(4)	11404,50(5)
	Ram-Hours(10^{-6})	1,54(2)	5,54(5)	6,23(8)	6,23(8)
	Acc	6,17(8)	4,17(2)	4,62(6)	4,29(4)
	$Kappa_M$	6,17(8)	4,17(2)	4,54(6)	4,25(4)
	$Kappa_T$	6,17(8)	4,17(2)	4,58(6)	4,25(4)
Sintético	Size	**1,00(1)**	7,38(8)	4,54(5)	4,00(3)
	CPU-Time	**1,04(1)**	6,42(8)	5,21(5)	5,62(6)
	Total-Time	**18977,59(1)**	81845,69(8)	59145,97(4)	59963,56(5)
	Ram-Hours(10^{-6})	**1,08(1)**	6,46(8)	5,54(5)	5,71(6)

dança de conceito. As curvas de acurácia são similares a outras *ensembles* nesse trabalho, apesar de HLAST apresentar performance inferior.

Em conclusão, *Streaming Random Patches* apresentou comportamento de acurácia diferente de outras *ensembles*. LAST apresentou resultados melhores comparados a outras *ensembles*, sendo que a seleção global de atributos randômica pode ocasionar em árvores treinadas com atributos menos importantes a crescerem mais que outras árvores treinadas com atributos mais importantes para o problema. $LAST_D$ apresentou os piores resultados, sendo que não há muita alteração na distribuição dos dados nos *datasets*. Todavia, combinações de LAST com HT e EFDT apresentam resultados entre os piores reportados, enquanto combinações de $LAST_D$ com HT e EFDT apresentaram melhores resultados, apesar de obterem *ranking* inferior a HT, não apresentam diferença estatística. Em *datasets* reais, HT, LAST e $HLAST_D$ apresentaram menor custo computa-

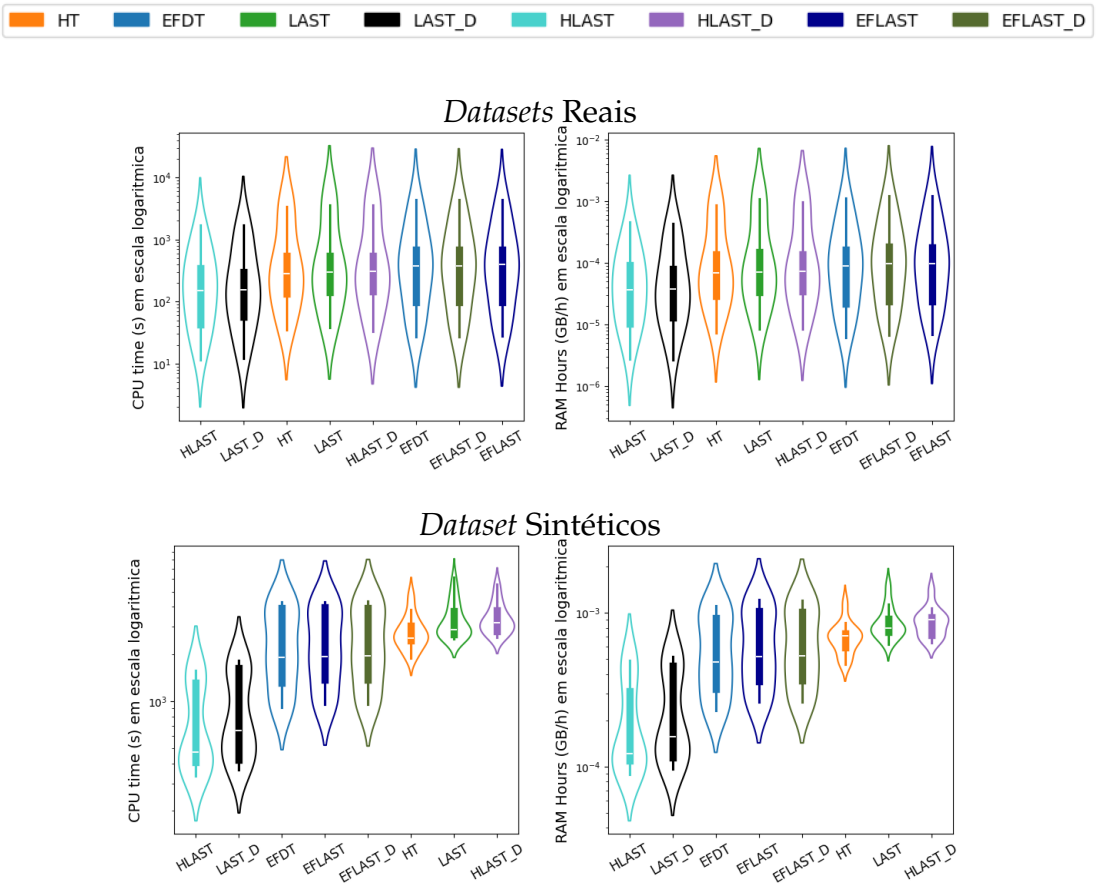

Figura 4.30: Distribuição de CPU-Time (esquerda) e RAM-Hours (direita) de árvores periódicas, adaptativas e combinação dos dois métodos como *base learners* da *ensemble Streaming Random Patches* em *datasets* reais e sintéticos

cional que EFDT, EFLAST e EFLAST$_D$, enquanto em *datasets* sintéticos EFDT, EFLAST e EFLAST$_D$ apresentaram menor custo computacional que HT, LAST e HLAST$_D$.

4.3.4 *Adaptive Regularized Ensemble*

A Figura 4.32 apresenta o *ranking* de acurácia dos *base learners* da *ensemble Adaptive Regularized Ensemble*. EFDT e variantes de LAST combinadas com EFDT apresentaram os melhores resultados, sendo que EFLAST$_D$ possui *ranking* superior a EFLAST e resultados estatisticamente diferentes, e EFLAST possui *ranking* superior a EFDT e resultados estatisticamente diferentes. HT e variantes de LAST combinadas com HT não tem diferença estatística entre si, e apresentam uma diferença de *ranking* de 3 posições comparados a EFDT.

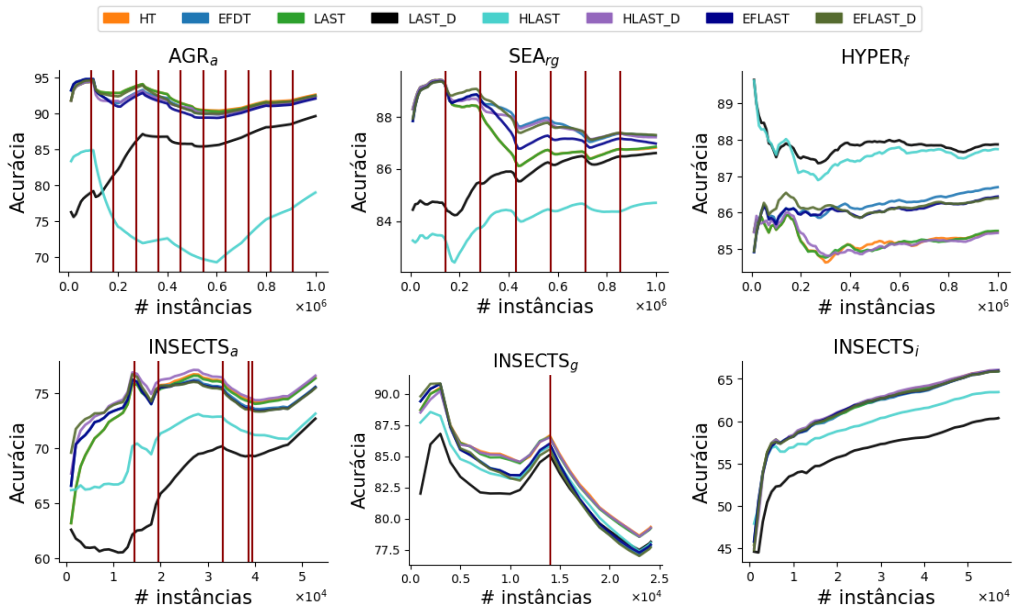

Figura 4.31: Acurácia (em %) de árvores periódicas, adaptativas e combinação dos dois métodos como *base learners* da *ensemble Streaming Random Patches* em *datasets* com mudanças de conceito

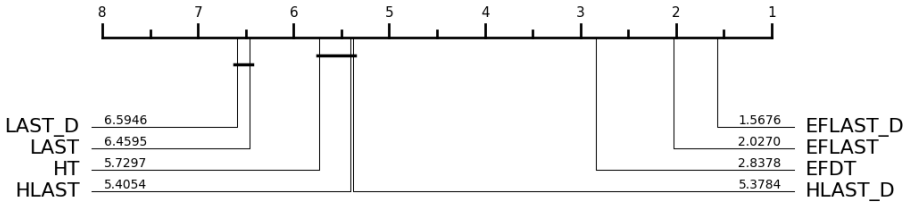

Figura 4.32: Ranking da Acurácia (em %) média ao longo do fluxo de árvores periódicas, adaptativas e combinação dos dois métodos como *base learners* da *ensemble Adaptive Regularized Ensemble*.

A Figura 4.33 apresenta o *ranking* de tamanho de árvore dos *base learners* da *ensemble Adaptive Regularized Ensemble*. EFDT e variantes de LAST combinadas com EFDT apresentaram os maiores *rankings* em tamanho de árvore, porém não possuem diferença estatística comparados com HLAST e HLAST$_D$. Dado o menor número de instâncias para treino, é esperado que o mecanismo de reavaliação de ramificações de EFDT tenha menor incidência, produzindo árvores mais profundas que HT. Diferentemente de *ensembles* como *Adaptive Random Forest* e *Adaptive Random Tree Ensemble*, que possuem alta amostragem e reavaliações de ramificações ocorrem com maior ocorrência.

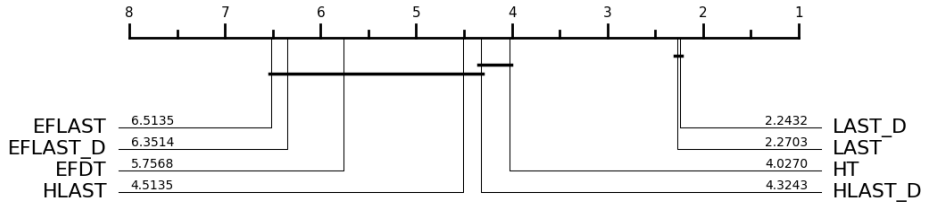

Figura 4.33: Ranking de tamanho de árvores periódicas, adaptativas e combinação dos dois métodos como *base learners* da *ensemble Adaptive Regularized Ensemble*.

A Tabela 4.8 apresenta o *ranking* médio de métricas de qualidade preditiva e custo computacional de *Adaptive Regularized Ensemble* por *base learner*. EFLAST$_D$ apresenta os melhores resultados em acurácia, Kappa$_M$ e Kappa$_T$, tanto em *datasets* sintéticos e reais, EFLAST o segundo melhor e EFDT o terceiro melhor resultado. Apesar de EFDT e variantes de LAST combinadas com EFDT apresentarem o maior custo computacional, ainda apresentam tempo de processamento total de todos os *datasets* muito inferiores comparados com outras *ensembles* estado-da-arte avaliadas nesse trabalho. Um comparativo em custo computacional entre *ensembles* é apresentado na seção 4.3.6.

Tabela 4.8: *Ranking* médio de métricas de árvores periódicas, adaptativas e combinação dos dois métodos como *base learners* da *ensemble Adaptive Regularized Ensemble*

Tipo do *Dataset*		HT	EFDT	LAST	LAST$_D$
Real	Acc	5,62(6)	2,92(3)	6,46(8)	6,00(7)
	Kappa$_M$	5,62(6)	2,92(3)	6,46(8)	6,00(7)
	Kappa$_T$	5,62(6)	2,92(3)	6,46(8)	6,00(7)
	Size	3,46(3)	6,08(6)	**2,15(1)**	2,38(2)
	CPU-Time	2,77(3)	5,77(6)	**1,85(1)**	2,00(2)
	Total-Time	4543,50(5)	3623,86(3)	1386,42(2)	**1235,48(1)**
	Ram-Hours(10^{-6})	2,38(3)	5,62(6)	**1,85(1)**	2,08(2)
Sintético	Acc	5,79(6)	2,79(3)	6,46(7)	6,92(8)
	Kappa$_M$	5,79(6)	2,79(3)	6,46(7)	6,92(8)
	Kappa$_T$	5,79(6)	2,79(3)	6,46(7)	6,92(8)
	Size	4,33(4)	5,58(6)	2,33(2)	**2,17(1)**
	CPU-Time	3,12(3)	4,67(4)	**1,96(1)**	2,04(2)
	Total-Time	5738,55(3)	15857,14(6)	2365,22(2)	**2264,70(1)**
	Ram-Hours(10^{-6})	2,71(3)	4,58(4)	**1,96(1)**	2,04(2)

A Figura 4.34 apresenta a distribuição de *CPU-Time* e *RAM-Hours* para os *base learners* da *ensemble Adaptive Regularized Ensemble*, sendo que as árvores estão ordenas pelo valor da mediana. Os resultados seguem o que já foi discutido anteriormente, sendo que EFDT e variantes de LAST combinadas com EFDT apresentam maior custo computacional que os outros métodos, apresen-

Tabela 4.8: Continuada

Tipo do *Dataset*		HLAST	HLAST$_D$	EFLAST	EFLAST$_D$
Real	Acc	5,54(5)	5,15(4)	2,38(2)	**1,00(1)**
	Kappa$_M$	5,46(5)	5,15(4)	2,46(2)	**1,92(1)**
	Kappa$_T$	5,46(5)	5,15(4)	2,46(2)	**1,92(1)**
	Size	4,15(4)	4,38(5)	6,23(7)	7,15(8)
	CPU-Time	4,54(4)	5,23(5)	6,69(7)	7,15(8)
	Total-Time	5033,22(7)	5100,62(8)	4257,09(4)	4565,53(6)
	Ram-Hours(10^{-6})	4,62(4)	5,31(5)	6,92(7)	7,23(8)
Sintético	Acc	5,33(4)	5,50(5)	1,83(2)	**1,00(1)**
	Kappa$_M$	5,38(4)	5,46(5)	1,83(2)	**1,38(1)**
	Kappa$_T$	5,38(4)	5,46(5)	1,83(2)	**1,38(1)**
	Size	4,71(5)	4,29(3)	6,67(8)	5,92(7)
	CPU-Time	5,67(5)	6,33(8)	6,00(6)	6,21(7)
	Total-Time	9509,56(4)	9652,62(5)	17684,86(7)	26209,34(8)
	Ram-Hours(10^{-6})	5,67(5)	6,33(7)	6,25(6)	6,46(8)

tando maior quartil inferior, mediana e quartil superior, tanto em *datasets* reais e sintéticos.

A Figura 4.35 apresenta a acurácia ao longo do tempo das árvores de decisão como *base learners* da *ensemble Adaptive Regularized Ensemble* em *datasets* com mudança de conceito. A diferença entre acurácia de HT e variantes de LAST combinadas com HT, e EFDT e variantes de LAST combinadas com EFDT é notável. HT e variantes de LAST combinadas com HT apresentam performance muito inferior à acurácia no final da *ensemble*, sendo que o protocolo experimental de média da acurácia em 20 pontos equidistantes do fluxo retratou essa disparidade. Em momentos iniciais do fluxo, a rejeição de instâncias se mostra como prejudicial à performance da *ensemble* com HT, pelo baixo número de ramificações realizadas e baixa atualização do modelo. Para os *base learners* EFDT e variantes de LAST combinadas com EFDT, a curva de acurácia é similar a outras *ensembles*.

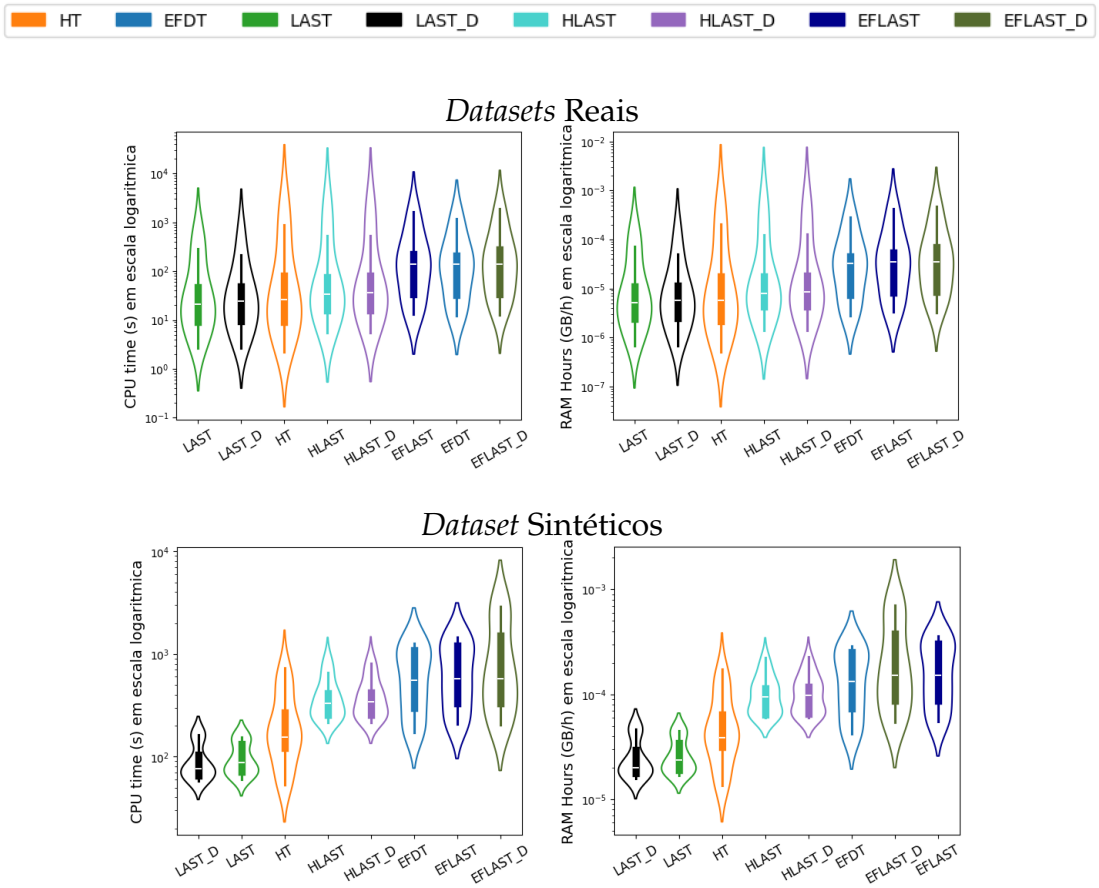

Figura 4.34: Distribuição de CPU-Time (esquerda) e RAM-Hours (direita) de árvores periódicas, adaptativas e combinação dos dois métodos como *base learners* da *ensemble Adaptive Regularized Ensemble* em *datasets* reais e sintéticos

.

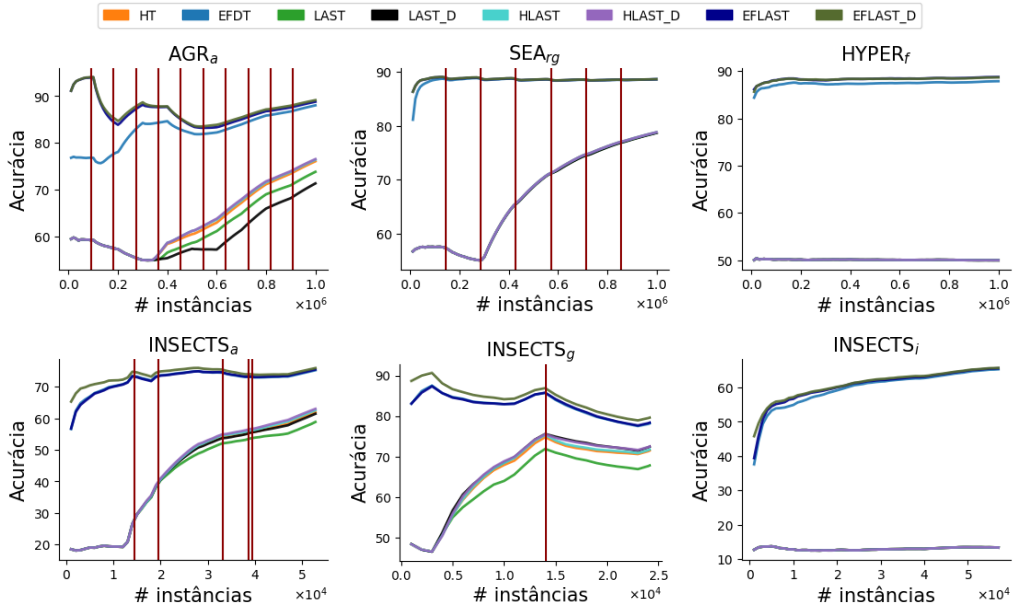

Figura 4.35: Acurácia (em %) de árvores periódicas, adaptativas e combinação dos dois métodos como *base learners* da *ensemble Adaptive Regularized Ensemble* em *datasets* com mudanças de conceito

Em conclusão, EFDT e variantes de LAST combinadas com EFDT apresentaram os melhores resultados. Devido a baixa amostragem procada pelaa rejeição de instâncias, EFDT, EFLAST e EFLAST$_D$ apresentam maior tamanho de árvore e custo computacional comparado a HT, HLAST e HLAST$_D$, sendo que o mecanismo de reavaliação de instâncias de EFDT realiza menos tentativas de reavaliação de instâncias e a condição de ramificação de EFDT menos restritiva faz com que as árvores ramifiquem mais que HT. Apesar de EFDT, EFLAST e EFLAST$_D$ apresentarem custo computacional superior a HT, HLAST e HLAST$_D$, o tempo total de processamento dos *datasets* por esses *base learners* é aproximadamente a metade do tempo requerido por *Adaptive Random Forest* com HT como *base learner*. Resultados referentes a acurácia ao longo do tempo mostram que HT, HLAST e HLAST$_D$ mostram uma disparidade de acurácia grande entre momentos iniciais do fluxo e o final do fluxo, evidenciando que o mecanismo de rejeição de instâncias afeta a performance inicial da ensemble no fluxo pelo baixo número de ramificações realizadas nos *base learners* e baixa atualização do modelo. Todavia, EFDT e EFLAST e EFLAST$_D$ apresentam performance e curva de acurácia ao longo do tempo similar a outras *ensembles* estado-da-arte, dado que a condição de ramificação de EFDT menos restrita comparada com HT faz com que mais ramificações ocorram em EFDT, EFLAST e EFLAST$_D$.

4.3.5 *Adaptive Random Tree Ensemble*

A Figura 4.36 apresenta o *ranking* médio de acurácia de *Adaptive Random Tree Ensemble* por *base learner*. HT, EFDT e variantes de LAST combinados com HT, e EFDT, não apresentaram resultados estatisticamente diferentes. Porém, HLAST$_D$ apresentou *ranking* superior a HT.

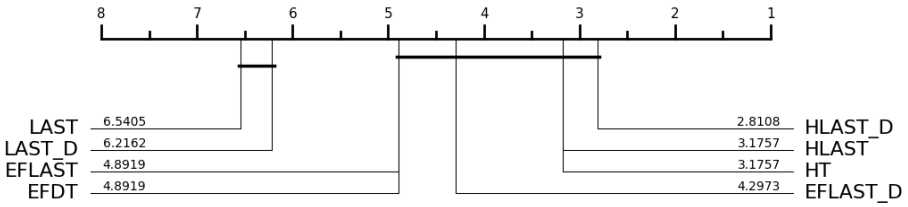

Figura 4.36: Ranking da Acurácia (em %) média ao longo do fluxo de árvores periódicas, adaptativas e combinação dos dois métodos como *base learners* da *ensemble Adaptive Random Tree Ensemble*.

A Figura 4.37 apresenta o *ranking* de tamanho de árvore dos *base learners* da *ensemble Adaptive Random Tree Ensemble*. HT, HLAST e HLAST$_D$ não apresentam resultados estatisticamente diferentes em tamanho de árvore, e EFDT, EFLAST e EFLAST$_D$ apresentam *ranking* de tamanho de árvore superior e estatisticamente diferente de HT, HLAST e HLAST$_D$, tendo comportamento similar a *ensemble Adaptive Random Forest*, pela similaridade das duas arquiteturas em amostragem com reposição e seleção de atributos randômica local em nós folhas. Apesar que menor tamanho de árvore de EFDT, EFLAST e EFLAST$_D$ não implicam em custo computacional inferior a HT, HLAST e HLAST$_D$, por armazenar dados em todos os nós da árvore devido ao mecanismo de reavaliação de ramificações.

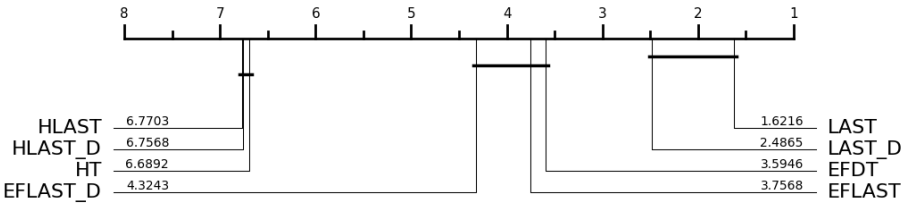

Figura 4.37: Ranking de tamanho de árvores periódicas, adaptativas e combinação dos dois métodos como *base learners* da *ensemble Adaptive Random Tree Ensemble*.

A Tabela 4.9 apresenta o *ranking* médio das métricas acurácia, Kappa$_M$, Kappa$_T$, número de nós, *CPU-Time* e *RAM-Hours* de *base learners* da *ensemble Adaptive Random Tree Ensemble*. HLAST$_D$ apresentou *ranking* maior em acurácia, Kappa$_M$ e Kappa$_T$ comparado a HT em *datasets reais*, e *ranking* inferior a HT em *datasets* sintéticos. Porém, em *datasets sintéticos*, apesar de haver grande diferença de *ranking* em acurácia (1 de HT e 3,21 de HLAST$_D$), não há muita diferença de *ranking* em Kappa$_M$ e Kappa$_T$ (3,06 de HT e 3,21 de HLAST$_D$), indicando não haver muita vantagem em relação a casos de classe minoritária e classificação por classe da instância anterior. HLAST apresenta custo computacional similar a HLAST$_D$ e resultados inferiores em qualidade preditiva. EFDT, EFLAST e EFLAST$_D$ apresentam qualidade preditiva inferior a HT, enquanto apresenta custo computacional superior.

Tabela 4.9: *Ranking* médio de métricas de árvores periódicas, adaptativas e combinação dos dois métodos como *base learners* da *ensemble Adaptive Regularized Ensemble*

Tipo do *Dataset*		HT	EFDT	LAST	LAST$_D$
	Acc	3,38(3)	5,19(5)	6,46(8)	5,92(7)
	Kappa$_M$	3,46(3)	5,35(6)	6,23(8)	5,92(7)
	Kappa$_T$	3,46(3)	5,35(6)	6,23(8)	5,92(7)
Real	Size	6,42(7)	3,27(3)	**2,23(1)**	2,54(2)
	CPU-Time	4,46(3)	6,31(8)	**1,69(1)**	2,00(2)
	Total-Time	8028,27(6)	7000,06(3)	**2622,70(1)**	2724,47(2)
	Ram-Hours(10^{-6})	3,69(3)	5,08(5)	**1,77(1)**	2,00(2)
	Acc	**1,00(1)**	4,73(6)	6,58(8)	6,38(7)
	Kappa$_M$	**3,06(1)**	4,73(6)	6,58(8)	6,38(7)
	Kappa$_T$	**3,06(1)**	4,73(6)	6,58(8)	6,38(7)
Sintético	Size	6,85(7)	3,77(3)	**1,29(1)**	2,46(2)
	CPU-Time	4,46(3)	4,54(4)	**1,25(1)**	2,38(2)
	Total-Time	25400,05(3)	32594,42(7)	**8153,47(1)**	10154,95(2)
	Ram-Hours(10^{-6})	4,42(4)	4,29(3)	**1,33(1)**	2,17(2)

Tabela 4.x: Continuada

Tipo do *Dataset*		HLAST	HLAST$_D$	EFLAST	EFLAST$_D$
	Acc	2,85(2)	**1,00(1)**	5,27(6)	4,85(4)
	Kappa$_M$	2,92(2)	**1,92(1)**	5,19(5)	5,00(4)
	Kappa$_T$	2,92(2)	**1,92(1)**	5,19(5)	5,00(4)
Real	Size	6,35(6)	6,69(8)	3,65(4)	4,85(5)
	CPU-Time	4,62(4)	5,00(5)	5,69(6)	6,23(7)
	Total-Time	9660,44(8)	9619,95(7)	7423,30(4)	7470,08(5)
	Ram-Hours(10^{-6})	5,00(4)	5,31(6)	6,46(7)	6,69(8)
	Acc	3,35(3)	3,21(2)	4,69(5)	4,00(4)
	Kappa$_M$	3,35(3)	3,21(2)	4,65(5)	4,04(4)
	Kappa$_T$	3,35(3)	3,21(2)	4,65(5)	4,04(4)
Sintético	Size	6,98(8)	6,79(6)	3,81(4)	4,04(5)
	CPU-Time	6,00(7)	6,33(8)	5,38(5)	5,67(6)
	Total-Time	28191,56(4)	28617,34(5)	32255,25(6)	33268,36(8)
	Ram-Hours(10^{-6})	6,08(7)	6,38(8)	5,46(5)	5,88(6)

A Figura 4.38 apresenta a distribuição de *CPU-Time* e *RAM-Hours* para as

árvores avaliadas nessa Seção, sendo que as árvores estão ordenas pelo valor da mediana. Diferentemente da *ensemble Adaptive Random Forest*, tanto em *CPU-Time* e *RAM-Hours* em *datasets* sintéticos e reais, HLAST e HLAST$_D$ apresentam quartil inferior, mediana e quartil superior similar a HT, e EFLAST e EFLAST$_D$ apresentam quartil inferior, mediana e quartil superior similar a EFDT, por não instanciar *base learners background* caso o detector de mudança explícito do classificador alerte uma mudança, e substituir a árvore por uma nova sem treinamento. Em *datasets* reais, para *CPU-Time* e *RAM-Hours*, classificadores baseados em EFDT apresentam mediana e quartil superior maior que classificadores baseados em HT, enquanto apresentam quartil inferior similar. Em *datasets* sintéticos, para *CPU-Time* e *RAM-Hours*, classificadores baseados em EFDT quartil inferior menor que classificadores baseados em HT, porém apresentam mediana e quartil superior maior que classificadores baseados em HT.

A Figura 4.39 apresenta a acurácia ao longo do tempo das árvores de decisão como *base learners* da *ensemble Adaptive Random Tree Ensemble* em *datasets* com mudança de conceito. As curvas de acurácia são similares a *ensemble Adaptive Random Forest*, e HLAST$_D$ apresentou maior acurácia comparado com outros classificadores em alguns casos como em SEA$_{rg}$ e INSECTS$_a$.

Em conclusão, HT, combinações de LAST com HT, EFDT e combinações de LAST com EFDT não apresentam diferença estatística em acurácia, porém HLAST$_D$ apresentou o maior ranking, seguido de HT. Em tamanho de árvore, *Adaptive Random Tree Ensemble* apresentou comportamento de tamanho de árvore similar a *Adaptive Random Forest*. EFDT e combinações de LAST com EFDT não apresentaram diferença estatística entre si e *ranking* superior comparados a HT e combinações de LAST com HT. Porém, menor tamanho e árvore de EFDT e combinações de LAST com EFDT não implica que o custo computacional é inferior a HT e combinações de LAST com HT, pois o mecanismo de reavaliação de ramificações de EFDT exige que dados sejam armazenados em todos os nós da árvore, em oposição a HT, que necessita armazenar dados em nós folhas. HLAST$_D$ apresentou qualidade preditiva (acurácia, Kappa$_M$ e Kappa$_T$) superior a HT, e em *datasets* sintéticos apresentou acurácia inferior a HT, porém obteve *ranking* similar em Kappa$_M$ e Kappa$_T$. O mecanismo de substituição por uma nova árvore em caso de detecção de mudança na *Adaptive Random Tree Ensemble* fez com que HT e combinações de LAST com HT obtivessem custo computacional similar, sendo que na ensemble *Adaptive Random Forest* combinações de

LAST com HT apresentaram custo computacional superior a HT por manterem *background learners* em caso de alerta de mudança de conceito pelo detector de mudança explícito. Nos *datasets* com mudança de conceito, as curvas de acurácia obtidas foram similares a outras *ensembles* avaliadas, sendo que HLAST$_D$ apresentou maior acurácia em alguns casos.

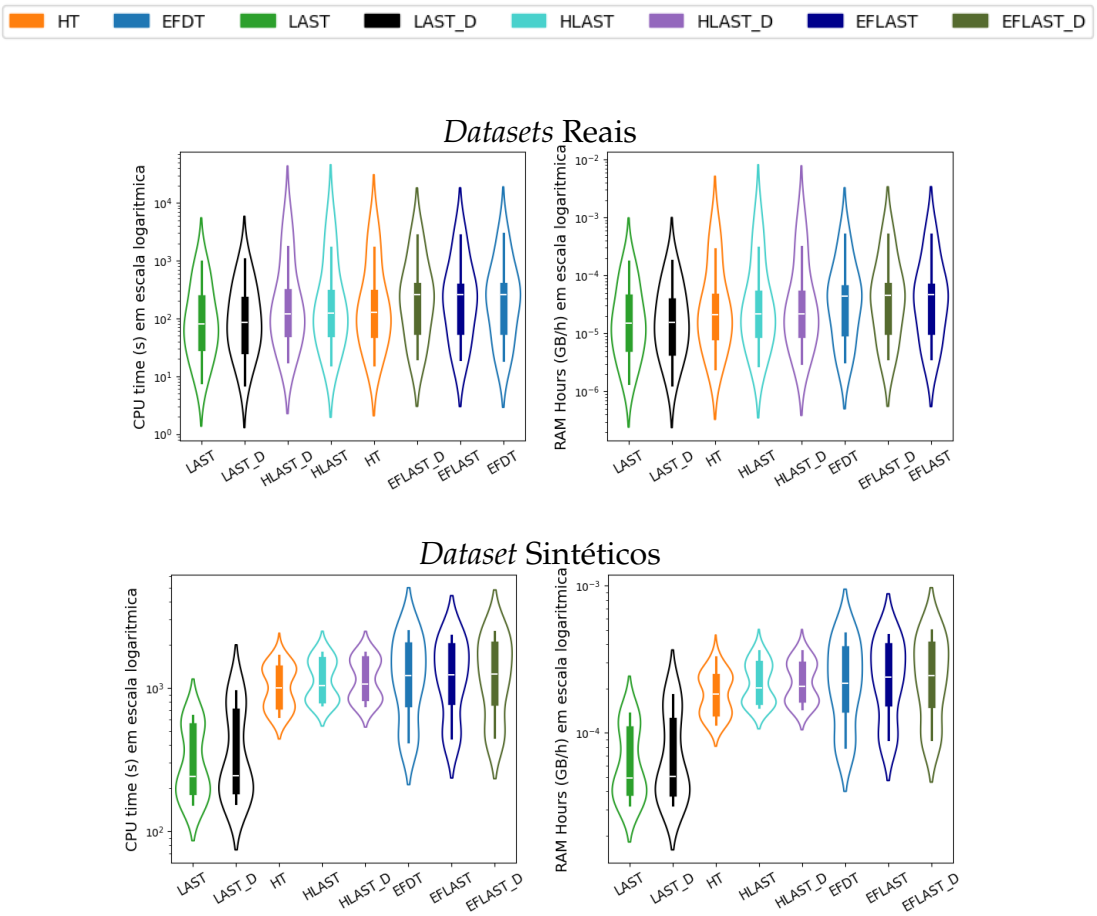

Figura 4.38: Distribuição de CPU-Time (esquerda) e RAM-Hours (direita) de árvores periódicas, adaptativas e combinação dos dois métodos como *base learners* da *ensemble Adaptive Random Tree Ensemble* em *datasets* reais e sintéticos

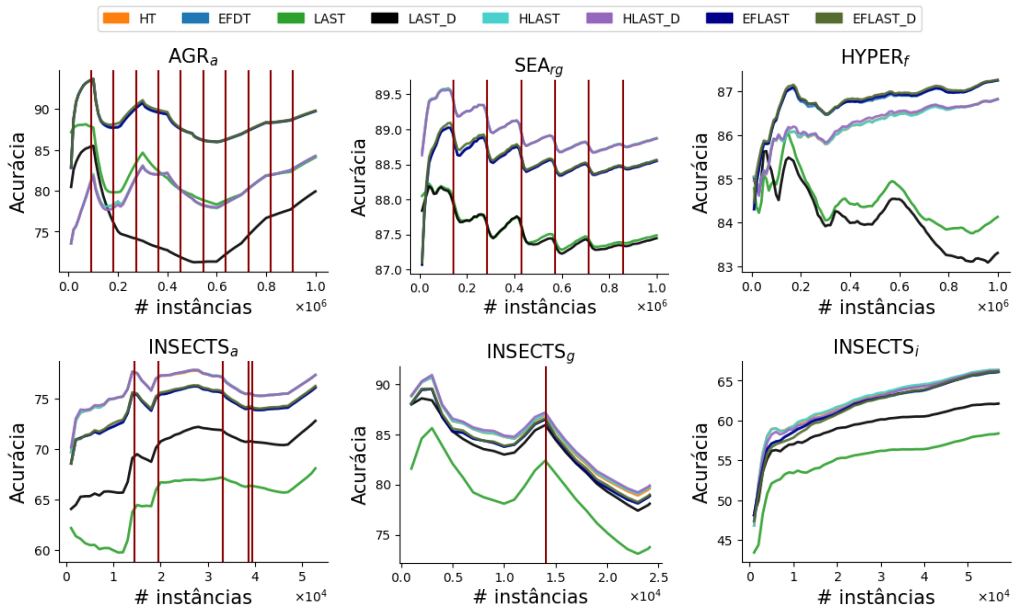

Figura 4.39: Acurácia (em %) de árvores periódicas, adaptativas e combinação dos dois métodos como *base learners* da *ensemble Adaptive Random Tree Ensemble* em *datasets* com mudanças de conceito

4.3.6 Comparativo entre *Ensembles*

Para o comparativo entre *ensembles* avaliados nesse trabalho, foram selecionados os *base learners* com maior *ranking* em acurácia e HT como *baseline*. Para *ensemble Streaming Random Patches*, foi selecionado somente HT dado que obteve os melhores resultados.

A Figura 4.40 apresenta o *ranking* médio de acurácia das *ensembles* avaliadas nesse trabalho com HT e o *base learner* com melhor resultado reportado de acurácia. *Adaptive Random Tree Ensemble* (ARTE) com o *base learner* HLAST$_D$ apresenta os melhores resultados em acurácia, e juntamente com ARTE com HT, tem melhores resultados que SRP com HT e apresentam resultados estatisticamente diferentes de *Adaptive Random Forest* (ARF) com HLAST$_D$. ARF com HLAST$_D$ e *Adaptive Regularized Ensemble* (ARE) com *base learner* HLAST$_D$ não apresentaram diferença estatística de ARF com HT, apesar de apresentarem diferença estatística de ARE com HT, que apresentou os piores resultados e com diferença estatística dos demais métodos.

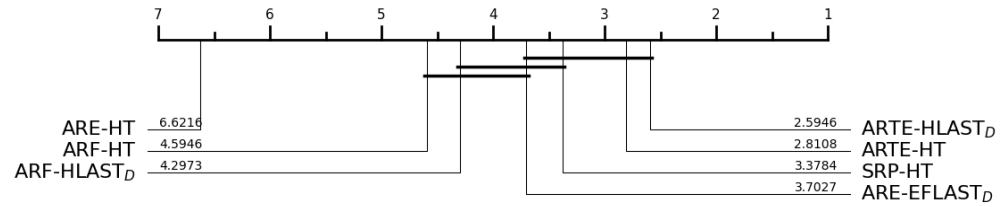

Figura 4.40: Ranking da Acurácia (em %) média ao longo do fluxo das *ensembles* avaliadas nesse trabalho com HT e o *base learner* com melhor resultado reportado de acurácia

A Figura 4.41 apresenta o *ranking* de tamanho de árvore das *ensembles* avaliadas nesse trabalho com HT e o *base learner* com melhor resultado reportado de acurácia. ARTE com HT e HLAST$_D$ (segundo e terceiro em *ranking*, respectivamente) não apresentaram resultados estatisticamente diferentes de ARF com HT, que apresentou o melhor *ranking* de tamanho de árvore. ARTE com HLAST$_D$ não apresentou diferença estatística de ARE com HT e ARF com HLAST$_D$, diferentemente de ARTE com HT. Porém ARTE com HLAST$_D$ ainda apresenta diferença estatística de SRP com HT, diferentemente dos outros métodos.

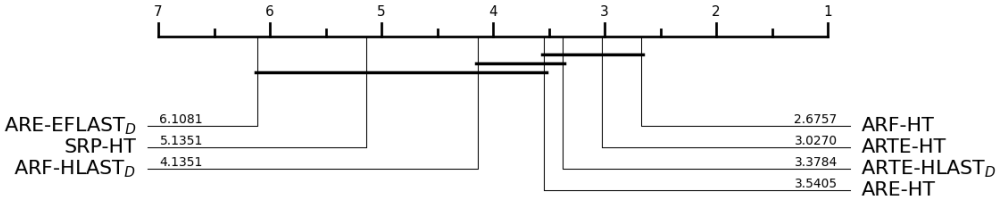

Figura 4.41: Ranking de tamanho de árvores médio das *ensembles* avaliadas nesse trabalho com HT e o *base learner* com melhor resultado reportado de acurácia

A Tabela 4.10 apresenta o *ranking* médio das métricas acurácia, Kappa$_M$, Kappa$_T$, número de nós, *CPU-Time* e *RAM-Hours* das *ensembles* avaliadas nesse trabalho com HT e o *base learner* com melhor resultado reportado de acurácia. Em *datasets* reais, ARTE com HLAST$_D$ apresentou *ranking* de qualidade preditiva superior e custo computacional similar a ARTE com HT, mesmo comparado com outras *ensembles*. Enquanto em *datasets* sintéticos, apesar de ARTE com HLAST$_D$ apresentar *ranking* inferior em acurácia, apresentou *ranking* similar de Kappa$_M$ e Kappa$_T$ e custo computacional similar. Tanto em *datasets* reais e sintéticos, SRP com HT apresenta *ranking* de qualidade preditiva inferior a ARTE

com HT e com HLAST$_D$ e é a *ensemble* mais custosa entre as avaliadas. ARE com EFLAST$_D$ apresentou *ranking* inferior a ARTE com HT e *ranking* em custo computacional similar. ARF com HT e HLAST$_D$ só apresentou resultados superiores a HT em qualidade preditiva, enquanto apresenta um dos maiores *rankings* em custo computacional entre ensembles. ARE apresentou os piores *rankings* em qualidade preditiva, enquanto apresenta o menor custo computacional entre *ensembles*.

Tabela 4.10: *Ranking* médio de métricas das *ensembles* avaliadas nesse trabalho com HT e o *base learner* com melhor resultado reportado de acurácia

Tipo do *Dataset*		ARF-HT	ARF-HLAST$_D$	SRP-HT	ARTE-HT
Real	Acc	5,23(6)	4,69(5)	3,38(3)	2,62(2)
	Kappa$_M$	5,31(6)	4,69(5)	3,31(3)	2,77(2)
	Kappa$_T$	5,31(6)	4,77(5)	3,31(3)	2,77(2)
	Size	**2,38(1)**	3,23(3)	4,77(6)	3,08(2)
	CPU-Time	5,46(5)	5,77(6)	6,54(7)	2,92(3)
	Total-Time	9790,20(5)	12229,44(7)	11996,25(6)	8028,27(3)
	Ram-Hours(10^{-6})	5,38(5)	5,85(6)	6,62(7)	2,38(2)
Sintético	Acc	4,25(6)	4,08(5)	3,38(3)	**1,00(1)**
	Kappa$_M$	4,25(6)	4,08(5)	3,38(3)	**2,92(1)**
	Kappa$_T$	4,25(6)	4,08(5)	3,38(3)	**2,92(1)**
	Size	**2,83(1)**	4,62(5)	5,33(6)	3,00(2)
	CPU-Time	5,00(5)	5,71(6)	6,54(7)	2,79(2)
	Total-Time	41099,27(5)	49682,75(6)	67061,08(7)	25400,05(2)
	Ram-Hours(10^{-6})	4,88(5)	5,88(6)	6,75(7)	2,62(2)

Tabela 4.10: Continuada

Tipo do *Dataset*		ARTE-HLAST$_D$	ARE-HT	ARE-EFLAST$_D$
Real	Acc	**1,00(1)**	6,54(7)	3,77(4)
	Kappa$_M$	**1,77(1)**	6,38(7)	3,77(4)
	Kappa$_T$	**1,77(1)**	6,31(7)	3,77(4)
	Size	3,46(4)	4,15(5)	6,92(7)
	CPU-Time	3,46(4)	**1,15(1)**	2,69(2)
	Total-Time	9619,95(4)	**4543,50(1)**	4565,53(2)
	Ram-Hours(10^{-6})	3,62(4)	**1,08(1)**	3,08(3)
Sintético	Acc	3,04(2)	6,67(7)	3,67(4)
	Kappa$_M$	3,04(2)	6,67(7)	3,67(4)
	Kappa$_T$	3,04(2)	6,67(7)	3,67(4)
	Size	3,33(4)	3,21(3)	5,67(7)
	CPU-Time	3,71(4)	**1,33(1)**	2,92(3)
	Total-Time	28617,34(4)	**5738,55(1)**	26209,34(3)
	Ram-Hours(10^{-6})	3,67(4)	**1,12(1)**	3,08(3)

A Figura 4.42 apresenta a distribuição de *CPU-Time* e *RAM-Hours* de *ensembles* avaliadas nesse trabalho com HT e o *base learner* com melhor resultado reportado de acurácia, sendo que as árvores estão ordenas pelo valor da mediana. Em *datasets* reais, tanto para *CPU-Time* e *RAM-Hours*, ARE com HT apresenta o menor quartil inferior, mediana e quartil superior. ARTE com HT e ARTE com HLAST$_D$ e ARE com EFLAST$_D$ apresentaram resultados similares, enquanto

apresentam quartil inferior, mediana e quartil superior menor que ARF com HT e HLAST$_D$. SRP apresentou o maior quartil inferior, mediana e quartil superior. Em *datasets* reais, tanto para *CPU-Time* e *RAM-Hours*, ARE com HT também apresenta o menor quartil inferior, mediana e quartil superior. ARE com EFLAST$_D$ apresentou quartil inferior e mediana menor que as outras *ensembles*, porém apresenta quartil superior maior que ARTE com HT e HLAST$_D$. ARTE com HT e ARTE com HLAST$_D$ apresentaram resultados similares, enquanto apresentam quartil inferior, mediana e quartil superior menor que ARF com HT e HLAST$_D$. SRP com HT também apresentou os piores resultados.

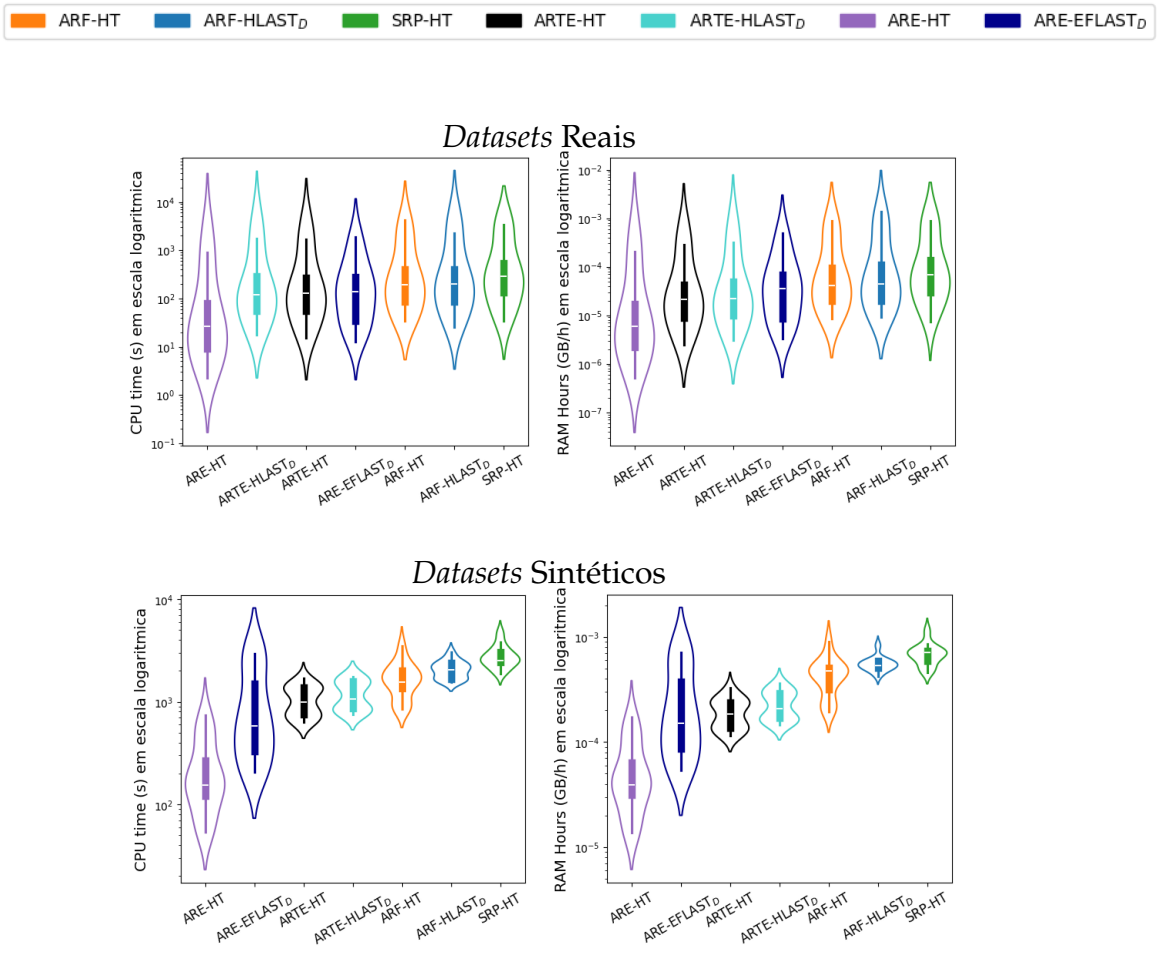

Figura 4.42: Distribuição de CPU-Time (esquerda) e RAM-Hours (direita) de *ensembles* avaliadas nesse trabalho com HT e o *base learner* com melhor resultado reportado de acurácia

Para resultados de acurácia ao longo do tempo em *datasets* com mudança de conceito, não são apresentados resultados para ARE com HT devido aos resul-

tados inferiores em momentos iniciais do fluxo (como descrito na Seção 4.3.4), sendo que os resultados inferiores influenciam na visualização dos resultados de *ensembles* com performance superior, em que resultados parecerem similares pela escala do gráfico, porém tem diferença de 3 a 5% de acurácia.

As *ensembles* em geral, obtiveram curvas de acurácia similares, sendo que os *base learners* tem o mesmo detector explícito de mudança de conceito (ADWIN), sendo que o que diferencia a performance das *ensembles* é a sua estratégia de diversidade e seu efeito em diferentes tipos de mudança de conceito e nas características do *dataset* presente, como mostrado nos resultados das Figuras 4.43 - 4.49.

No *dataset* Agrawal, ARTE apresenta os piores resultados, dado que os conceitos de Agrawal constituem pontos de corte bem definidos, sendo que a estratégia de ponto de corte aleatório não é a ideal para esse *dataset*. SRP mostrou os melhores resultados, pois ponderamento de acurácia no voto combinado com seleção randômica de atributos globais é ideal no contexto de Agrawal, em que atributos são adicionados ao conceito e árvores que treinam com atributos mais relevantes vão ter maior acurácia e influência no voto da *ensemble*. ARE apresentou os melhores resultados em *datasets* com mudanças recorrentes, enquanto apresenta performance similar a ARF em mudanças abruptas e graduais. O único mecanismo condizente de ARE que justifica a performance superior em *datasets* com mudança recorrente é a seleção de classificadores. Os conceitos na mudança recorrente são similares, sendo que afetam menos a acurácia da *ensemble*, diferentemente de outras mudanças de conceito que tem conceitos mais distintos entre os conceitos 1 a 6 comparados com os conceitos 7 a 10.

No *dataset* SEA, SRP apresentou os piores resultados, pelo baixo número de atributos (3), sendo um irrelevante. Maioria das árvores serão idênticas pelo mecanismo de seleção de atributos, gerando baixa diversidade. Outras *ensembles* apresentaram performance similar. Nos casos de mudança gradual (recorrentes ou não), maior decréscimo de acurácia é observável em comparação com casos de mudanças abruptas, uma vez que mudanças mais lentas de conceito resultam em *base learners* treinados com mais instâncias de 2 conceitos simultaneamente.

No *dataset* LED, ARE apresenta os piores resultados, mesmo apresentando curva de acurácia similar. Outras *ensembles* apresentaram performance similar. Nos casos de mudança gradual (recorrentes ou não), maior decréscimo de acurácia também é observável em comparação com casos de mudanças abruptas.

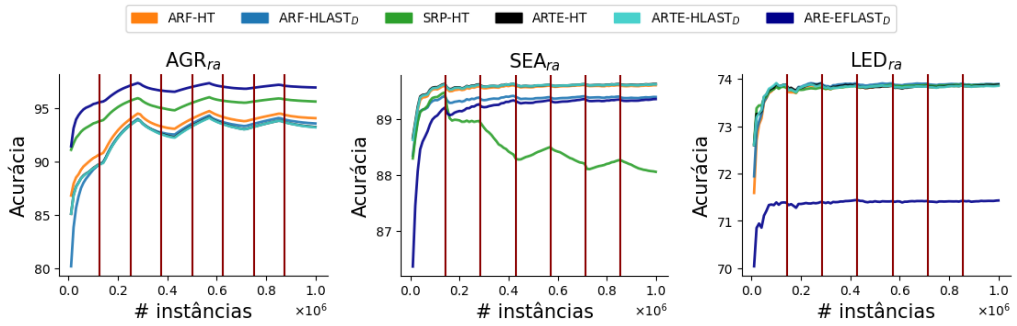

Figura 4.43: Acurácia (em %) de *ensembles* avaliadas nesse trabalho em *datasets* que simulam mudanças de conceito recorrentes e mudança abrupta entre conceitos

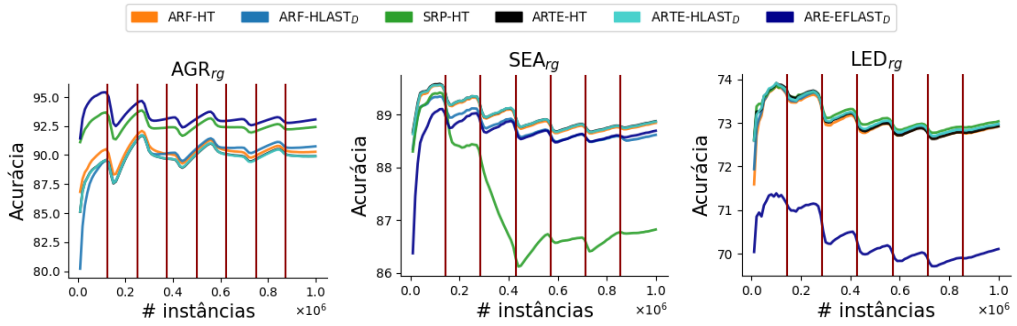

Figura 4.44: Acurácia (em %) de *ensembles* avaliadas nesse trabalho em *datasets* que simulam mudanças de conceito recorrentes e mudança gradual entre conceitos

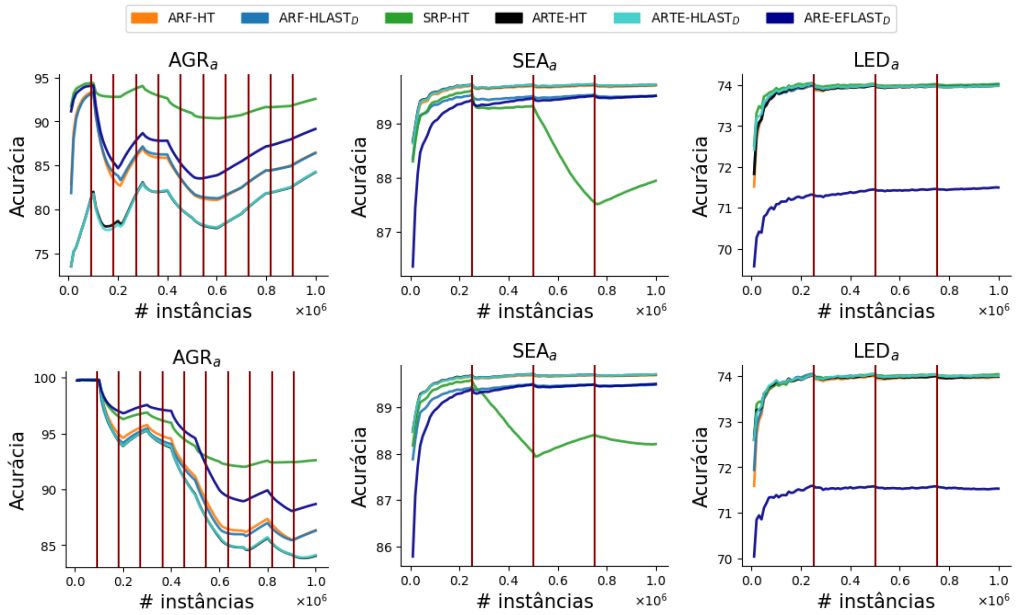

Figura 4.45: Acurácia (em %) de *ensembles* avaliadas nesse trabalho em *datasets* que simulam mudanças de conceito abruptas

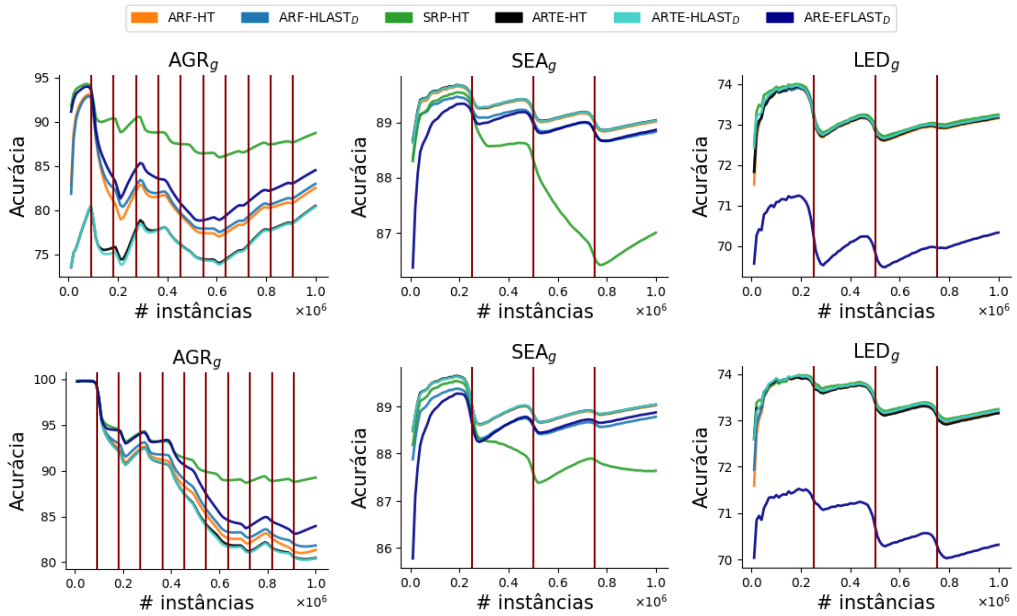

Figura 4.46: Acurácia (em %) de *ensembles* avaliadas nesse trabalho em *datasets* que simulam mudanças de conceito graduais

Para o *dataset* RBF, ARE apresentou os melhores resultados, dado que conceitos intermediários vão ser as instâncias que os classificadores mais vão errar e treinar, tendo importância maior para os classificadores. SRP apresentou os

piores resultados, pela mudança constante dos *centroids*, limitar *base learners* a um conjunto de atributos não se mostrou efetivo, sendo que a importância dos atributos pode mudar constantemente.

Para o *dataset* HYPER, SRP começa a apresentar resultados piores conforme a velocidade das mudanças incrementais aumenta. No caso desse *dataset*, ele possui atributos dependentes, e árvores que não possuem os atributos dependentes afetam a performance da *ensemble*, especialmente em cenários com altas taxas de mudança, sendo complexo se adaptar as mudanças. ARE apresentou os melhores resultados em todas as velocidades de mudança incremental, como no *dataset* RBF.

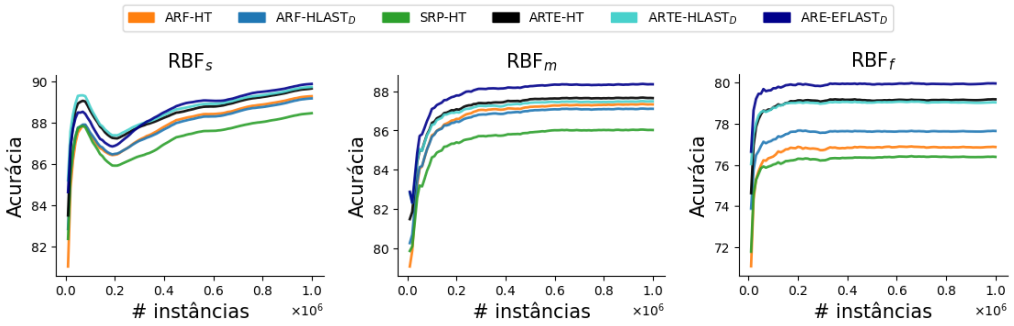

Figura 4.47: Acurácia (em %) de *ensembles* avaliadas nesse trabalho no *dataset RBF*, que simulam mudanças de conceito incremental

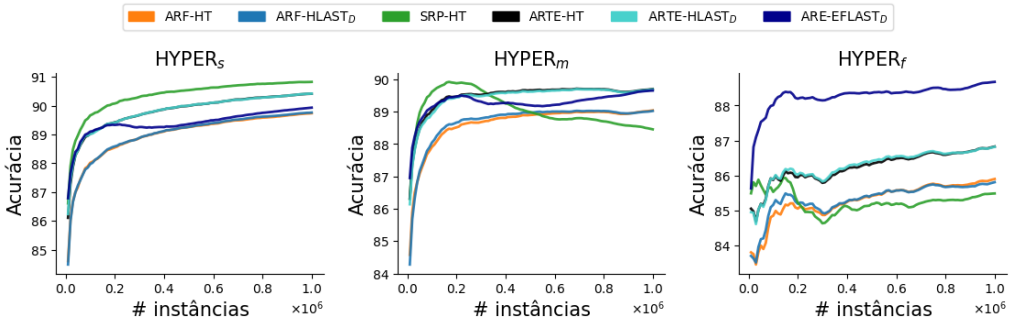

Figura 4.48: Acurácia (em %) de *ensembles* avaliadas nesse trabalho no *dataset HYPER*, que simulam mudanças de conceito incremental

No *dataset* INSECTS, *ensembles* apresentaram performance similar, sendo que ARF apresentou os piores resultados.

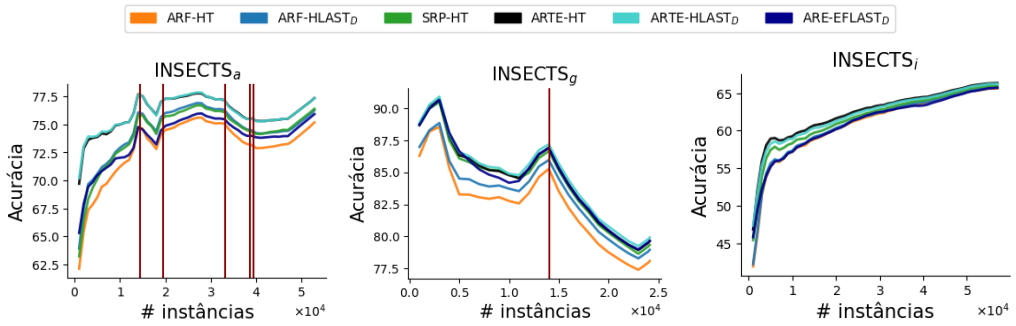

Figura 4.49: Acurácia (em %) de *ensembles* avaliadas nesse trabalho no *dataset* *INSECTS*

Em conclusão, ARTE apresentou os melhores resultados em qualidade preditiva e custo computacional inferior a ARF, também obtendo melhores resultados que SRP. O SRP apresentou o segundo melhor resultado em qualidade preditiva e o maior custo computacional reportado. HLAST$_D$ foi capaz de melhorar a qualidade preditiva de ARTE com HT em *datasets* reais e apresentar resultados similares em *datasets* sintéticos, com custo computacional similar. O melhor desempenho se deve por ele não criar *background base learners* em caso de alerta de *warning*, diferentemente do *ensemble* ARF. Em *datasets* com mudança de conceito, as *ensembles* apresentaram performance similar, sendo que em alguns casos foram apontadas limitações da arquitetura da *ensemble* pelo tipo de problema e mudança de conceito presente no *dataset*.

4.4 Considerações Finais

A Seção 4.2 apresentou uma discussão geral sobre os resultados obtidos, que não somente atesta a qualidade do método proposto na seção 3.1, LAST, mas também apresenta um *ablation* sobre a performance de árvores de decisão estado-da-arte em diversos cenários. A Seção 4.2.2 apresentou a performance do LAST com diversos detectores em comparação com o estado-da-arte, em que se atestou que LAST teve os melhores resultados independente do detector. Um *trade-off* de custo computacional e qualidade preditiva é observável entre métodos de detecção que utilizam de estatística descritiva e teste estatístico. A Seção 4.2.3 apresentou evidências de como ramificações afetam a performance de árvores de decisão estado-da-arte. Foi observado que na maioria das vezes LAST obteve performance superior quando realizou ramificações demonstrando capacidade de recuperação quando a performance decai. EFDT apresentou alguns

poucos casos que ramificou antes da performance decair, porém são bem menos significativos comparados a casos em que LAST reage com êxito ao decaimento de performance. A Seção 4.2.4 analisou o comportamento dos algoritmos em cenários onde o número de classes e atributos é elevado. LAST obteve performance superior a HT, EFDT e HAT, enquanto não houve muita diferença em custo computacional. HAT e EFDT apresentaram os piores resultados em uso de memória. A Seção 4.2.5 apresentou o efeito de mudanças de conceito na performance das árvores de decisão. LAST obteve resultados superiores comparado a HT e EFDT em *datasets* com mudança abrupta de conceito, enquanto em *datasets* com mudança gradual de conceito, HT e EFDT obtiveram performance similar a LAST, indicando que ramificações não melhoram a performance quando um nó folha recebe instâncias de 2 conceitos simultaneamente. Destaca-se a disparidade de performance da HAT em *datasets* reais e sintéticos. Em *datasets* sintéticos são simuladas somente mudanças reais de conceito e a performance de HAT é excelente. Entretanto, HAT não obteve resultados superiores a LAST em cenários reais com mudanças de conceito e alteração em $P_t(X)$ e $P_t(y|X)$ simultaneamente.

A Seção 4.3 apresenta resultados para o ambiente *Ensemble*, para avaliar a qualidade preditiva e custo computacional de árvores propostas na Seção 3.1.1 como *base learner* de *ensemble*, que aborda limitações do algoritmo LAST como *base learners* de *ensembles* ao combinar estratégias periódicas de ramificação de Hoeffding Tree com a estratégia adaptativa de LAST. As seções 4.3.2- 4.3.5 apresentam os resultados para cada *ensemble*. Para as *ensembles* ARF, ARE e ARTE, as árvores propostas obtiveram o melhor *ranking* em acurácia geral. Combinações de LAST_D e HT e EFDT obtiveram os melhores resultados, tendo resultados superiores a combinações de LAST com monitoramento de erro e HT e EFDT. As ramificações periódicas causam maior mudança na distribuição dos dados, resultando em árvores com maior tamanho e maior diversidade da *ensemble*. SRP foi a única *ensemble* que HT obteve o melhor *ranking* em acurácia. Porém, como abordado na Seção 4.3.6, que apresenta um comparativo entre *ensembles* avaliadas nesse trabalho com o *base learner* com melhor resultado reportado e HT como *baseline*, ARTE supera os resultados de SRP tendo custo computacional inferior a ARF. Nesse contexto, HLAST_D conseguiu melhorar os resultados de ARTE com HT em qualidade preditiva em *datasets* reais, enquanto apresenta custo computacional similar a ARTE com HT.

5

Conclusão

O estado-da-arte de árvores de decisão para o problema de classificação de fluxo de dados é composto majoritariamente por avaliações periódicas de ramificação baseadas no teorema de *Hoeffding* (DOMINGOS; HULTEN, 2000).

Esse trabalho apontou como avaliações periódicas não consideram a evolução do fluxo em nós folhas, seja na distribuição dos dados ou qualidade preditiva. O mecanismo periódico de avaliação de ramificações em *Hoeffding Trees* não somente é incapaz de reagir a mudanças locais presentes nos nós folhas, mas também realiza busca gulosa de melhor ramificação mesmo quando poucas alterações são observáveis na distribuição dos dados ou qualidade preditiva. Nestes casos as ramificações possuem pouca eficácia e produzem um custo computacional evitável. Também foram apontadas inconsistências teóricas com o uso do teorema de *Hoeffding* (HOEFFDING, 1963) já conhecidas na literatura (RUTKOWSKI; JAWORSKI; DUDA, 2020) para atestar se uma ramificação em um atributo A_a é realmente apropriada, sendo então uma heurística para determinação de uma ramificação.

Esse trabalho contemplou a proposta de uma árvore de decisão para mineração de fluxo de dados, sendo ela materializada com o algoritmo *Local Adaptive Streaming Tree* (*LAST*). O *LAST* mantêm algoritmos de detecção de mudança em nós folhas, que monitoram constantemente a qualidade preditiva ou distribuição dos dados. Caso o algoritmo alerte que uma mudança ocorreu, uma ramificação ocorre caso $\Delta G(A_a) > 0$, sendo a restrição de ramificação menos restritiva possível e os algoritmos de detecção de mudança aqueles que ditam como a árvore cresce.

LAST com monitoramento de erro apresentou os melhores resultados no ambiente monolítico, obtendo resultados superiores ao estado-da-arte de árvores de decisão, independente do detector de mudança utilizado. Também foi apresentada uma análise detalhada do método, que apontou comportamentos relevantes: (i) que ramificação em decaimento de acurácia resulta em qualidade preditiva superior a HT, (ii) que LAST apresenta custo computacional similar a HT e inferior a EFDT e HAT em um cenário que é possível variar o número de classes e atributos, e (iii) LAST apresenta resultados superiores em *datasets* com mudança abrupta de conceito, sendo que boa performance em *datasets* com mudanças graduais de conceito dependem de detectores de mudança de conceito modelados para lidar melhor com esse tipo de mudança.

As árvores propostas também tem limitações referentes a sua aplicação como *base learners* de *ensembles*. Um dos principais componentes de *ensembles* para obter maior qualidade preditiva é a sua diversidade (KUNCHEVA; WHITAKER, 2003), ao treinar classificadores com diferentes cópias das instâncias. Detectores de mudança lidam com instâncias de forma monolítica, e múltiplas atualizações do modelo podem ocasionar em elevada complexidade computacional. Portanto, as árvores teriam tempos de ramificação similares e baixa diversidade entre si. Árvores com mecanismo estático, por sua vez, são sensíveis ao número de cópias para gerar ramificações e induzem diversidade na *ensemble*. As limitações de árvores com mecanismos adaptativos foram contornadas com a proposta de combinação do mecanismo estático presente em *Hoeffding Trees* e adaptativo. Caso o número de exemplos no nó folha seja divisível pelo *Grace Period*, uma tentativa de ramificação é feita como proposto em (DOMINGOS; HULTEN, 2000) (algoritmo HLAST) ou (MANAPRAGADA; WEBB; SALEHI, 2018) (algoritmo EFLAST). Caso o algoritmo de mudança de conceito alerte uma mudança, uma ramificação ocorre caso $\Delta G(A_a) > 0$.

A combinação de LAST com HT (HLAST) e EFDT (EFLAST) apresentaram os melhores resultados, em especial na *ensemble* ARTE, em que HLAST_D apresentou qualidade preditiva superior à ARTE com HT e todas as outras *ensembles*, enquanto apresentou custo computacional inferior a HT e muito inferior a SRP com HT, que apresentou os segundos melhores resultados em qualidade preditiva.

5.1 Trabalhos Futuros

Outros cenários são possíveis de se explorar com a proposta de LAST, como por exemplo:

1. Técnicas de regularização (BARDDAL; ENEMBRECK, 2019), que visam diminuir o tamanho de árvore sem afetar a qualidade preditiva significamente.
2. Técnicas de reavaliação de ramificações (BIFET; GAVALDÀ, 2009), que visam podar a árvore em nós que tem decaimento de acurácia.
3. Cenário de regressão, em que a saída do modelo é uma predição de um valor real ao invés de uma classe discreta, sendo que o objetivo do modelo é realizar predições que mais se aproximem do rótulo original da instância.

5.2 Publicações

Essa dissertação contemplou a publicação dos seguintes artigos:

- Daniel Nowak Assis, Jean Paul Barddal, and Fabrício Enembreck. 2024. ***Just Change on Change: Adaptive Splitting Time for Decision Trees in Data Stream Classification***. *In Proceedings of ACM SAC Conference (SAC'24)*. Avila, Spain.
- Daniel Nowak Assis, Jean Paul Barddal, and Fabrício Enembreck. 2025. ***Behavioral insights of adaptive splitting decision trees in evolving data stream classification*** . *Knowledge and Information Systems* .

6

Apêndice

Tabela 6.1: Acurácia (em %) média ao longo do fluxo de árvores de decisão em *datasets* reais

						LAST	
Dataset	NB	HT	HOT	EFDT	HAT	ADWIN	DDM
Outdoor	56,84	56,72	56,72	59,98	56,42	59,97	61,23
Elec	76,19	83,18	86,17	83,07	85,59	84,47	85,92
Rialto	24,55	28,37	27,16	54,70	29,34	51,33	58,89
Airlines	67,57	66,99	**69,41**	66,56	65,44	67,98	67,38
CovType	66,82	81,11	84,12	84,88	81,25	86,05	87,36
Nomao	92,73	95,13	95,59	96,23	96,53	97,10	97,20
Poker	59,57	74,21	74,38	74,45	67,76	74,76	79,62
NOAA	69,48	72,63	72,66	72,99	72,86	73,81	**76,55**
INSECTS_a	53,66	55,51	55,70	62,59	60,56	62,47	61,78
INSECTS_i	46,64	48,80	48,80	54,17	50,26	54,57	52,38
INSECTS_g	63,40	64,71	64,71	73,04	65,62	70,76	74,94
LADPU	56,33	56,33	56,33	61,37	56,31	61,63	60,50
Asfault	79,02	79,02	79,02	81,30	78,89	**84,77**	80,86
Avg. Rank Real	17,62	15,38	12,77	9,85	14,54	7,31	6,46

Tabela 4.3 : Continuada

	LAST						
Dataset	EDDM	HDDM$_A$	HDDM$_W$	RDDM	MDDM$_A$	MDDM$_E$	MDDM$_G$
Outdoor	61,63	60,51	**61,79**	57,93	60,64	60,37	60,37
Elec	85,57	**86,35**	84,82	85,12	84,44	84,66	84,66
Rialto	**61,08**	57,19	48,58	59,79	55,70	55,50	55,51
Airlines	66,21	67,61	67,96	67,13	67,52	67,53	67,53
CovType	**87,90**	87,85	85,50	87,64	86,54	86,54	86,51
Nomao	**97,24**	97,19	96,14	96,65	96,51	96,55	96,55
Poker	85,74	82,00	**88,08**	81,58	78,32	79,77	79,78
NOAA	76,21	73,98	75,40	76,53	75,09	74,74	74,74
INSECTS$_a$	60,54	62,71	61,66	63,90	**64,58**	64,45	64,45
INSECTS$_i$	52,22	52,52	53,18	55,48	55,48	55,50	**55,50**
INSECTS$_g$	**77,72**	70,83	70,78	75,93	70,40	70,41	70,41
LADPU	60,65	62,25	60,68	62,94	**62,95**	62,85	62,85
Asfault	83,11	81,49	84,52	82,60	84,34	84,29	84,29
Avg. Rank Real	6,00	5,62	6,85	**5,31**	6,23	6,23	6,15

Tabela 4.3 : Continuada

	LAST$_D$							
Dataset	ADWIN	DDM	HDDM$_A$	HDDM$_W$	RDDM	MDDM$_A$	MDDM$_E$	MDDM$_G$
Outdoor	58,83	57,50	54,50	58,33	56,72	56,72	56,72	56,72
Elec	80,13	76,19	81,27	76,19	82,43	76,19	76,19	76,19
Rialto	53,89	24,54	51,54	40,82	24,54	24,54	24,54	24,54
Airlines	67,33	68,05	66,78	67,64	67,51	67,64	67,64	67,64
CovType	85,21	67,94	80,05	82,27	66,82	66,82	66,82	66,82
Nomao	95,72	96,20	94,57	94,18	96,30	93,98	93,98	93,98
Poker	76,02	59,57	74,32	61,71	59,57	59,57	59,57	59,57
NOAA	71,23	70,05	70,05	70,05	70,05	70,05	70,05	70,05
INSECTS$_a$	63,55	53,65	59,01	56,52	53,65	53,65	53,65	53,65
INSECTS$_i$	55,47	46,63	52,97	48,65	46,63	46,63	46,63	46,63
INSECTS$_g$	72,99	63,40	72,21	67,74	63,40	63,40	63,40	63,40
LADPU	62,77	56,33	60,95	57,78	56,33	56,33	56,33	56,33
Asfault	81,73	78,68	80,83	78,16	79,02	79,02	79,02	79,02
Avg. Rank Real	9,69	16,85	13,77	14,85	17,42	18,12	18,00	18,00

Tabela 6.2: Acurácia média ao longo do fluxo de árvores de decisão em *datasets* sintéticos

						LAST	
Dataset	NB	HT	HOT	EFDT	HAT	ADWIN	DDM
AGR-f(7,8,9,10,9,8,7)$_{ra}$	78,11	85,00	83,68	81,15	**95,52**	89,48	88,25
AGR-f(7,8,9,10,9,8,7)$_{rg}$	78,10	81,33	81,55	80,37	**92,25**	77,81	79,28
AGR-f(1,2,3,4,5,6,7,8,9,10)$_{a}$	65,02	77,34	77,30	82,25	**86,71**	80,77	80,08
AGR-f(10,9,8,7,6,5,4,3,2,1)$_{a}$	72,90	79,78	80,47	82,50	**94,33**	85,29	80,57
AGR-f(1,2,3,4,5,6,7,8,9,10)$_{g}$	65,03	73,82	73,89	78,56	**85,04**	78,59	76,09
AGR-f(10,9,8,7,6,5,4,3,2,1)$_{g}$	72,86	76,55	77,65	80,34	**88,99**	77,29	76,94
SEA-f(1,2,3,4,3,2,1)$_{ra}$	85,17	85,74	85,72	85,48	**88,39**	86,20	86,23
SEA-f(1,2,3,4,3,2,1)$_{rg}$	85,17	85,66	85,65	85,47	**87,87**	85,83	85,90
SEA-f(1,2,3,4)$_{a}$	86,22	86,87	86,88	86,84	**88,56**	86,89	87,26
SEA-f(1,2,3,4)$_{g}$	86,21	86,83	86,84	86,78	**88,22**	86,49	87,05
SEA-f(4,3,2,1)$_{a}$	85,98	86,28	86,35	86,15	**88,61**	87,19	86,99
SEA-f(4,3,2,1)$_{g}$	85,95	86,13	86,23	85,92	**88,04**	87,34	87,49
LED-f(7,5,3,1,3,5,7)$_{a}$	63,99	69,24	69,45	69,94	**73,30**	72,89	72,79
LED-f(7,5,3,1,3,5,7)$_{g}$	63,99	69,00	69,17	69,88	72,06	71,99	**72,08**
LED-f(1,3,5,7)$_{a}$	64,73	70,79	71,74	70,14	73,59	**73,85**	73,73
LED-f(1,3,5,7)$_{g}$	64,71	70,55	71,06	70,14	**72,58**	72,33	72,34
LED-f(7,5,3,1)$_{a}$	68,32	71,43	71,74	71,70	73,49	**73,83**	73,65
LED-f(7,5,3,1)$_{g}$	68,28	71,35	71,40	71,94	72,86	**73,15**	73,11
RBF$_{s}$	42,67	70,14	73,01	76,18	73,08	65,53	69,99
RBF$_{m}$	33,32	49,78	59,99	57,05	60,40	65,39	66,66
RBF$_{f}$	29,87	33,40	34,85	32,94	38,88	40,50	43,28
HYPER$_{s}$	93,93	89,41	89,43	89,06	88,24	93,93	93,93
HYPER$_{m}$	92,58	89,08	89,13	88,71	88,30	91,80	91,68
HYPER$_{f}$	79,44	82,61	83,21	84,26	**86,21**	84,13	83,35
Avg. Rank Synth	17,10	12,08	10,75	10,54	**4,21**	6,77	6,31

Table 4.4: Continuada

				LAST			
Dataset	EDDM	HDDM$_{A}$	HDDM$_{W}$	RDDM	MDDM$_{A}$	MDDM$_{E}$	MDDM$_{G}$
AGR-f(7,8,9,10,9,8,7)$_{ra}$	85,55	90,28	88,10	87,80	90,00	90,03	90,03
AGR-f(7,8,9,10,9,8,7)$_{rg}$	77,77	76,98	83,40	81,97	78,94	78,92	78,92
AGR-f(1,2,3,4,5,6,7,8,9,10)$_{a}$	81,02	80,84	78,59	82,00	81,34	81,27	81,28
AGR-f(10,9,8,7,6,5,4,3,2,1)$_{a}$	78,09	84,63	87,26	80,03	87,07	87,16	87,17
AGR-f(1,2,3,4,5,6,7,8,9,10)$_{g}$	76,02	78,69	75,88	75,92	79,83	79,70	79,70
AGR-f(10,9,8,7,6,5,4,3,2,1)$_{g}$	80,35	78,71	76,90	76,94	76,59	76,41	76,41
SEA-f(1,2,3,4,3,2,1)$_{ra}$	86,30	86,27	85,51	86,07	86,22	85,78	86,39
SEA-f(1,2,3,4,3,2,1)$_{rg}$	86,21	86,08	85,52	85,62	85,67	85,07	85,90
SEA-f(1,2,3,4)$_{a}$	87,41	86,91	86,40	87,10	87,25	86,17	87,26
SEA-f(1,2,3,4)$_{g}$	87,29	86,61	86,25	86,70	86,84	85,96	86,88
SEA-f(4,3,2,1)$_{a}$	86,56	87,64	87,70	87,35	86,97	86,99	86,99
SEA-f(4,3,2,1)$_{g}$	86,61	87,27	87,27	86,86	86,60	86,60	86,60
LED-f(7,5,3,1,3,5,7)$_{a}$	71,29	72,79	68,77	71,94	71,07	70,99	70,99
LED-f(7,5,3,1,3,5,7)$_{g}$	71,30	71,77	68,91	71,04	69,52	69,42	69,42
LED-f(1,3,5,7)$_{a}$	71,39	73,81	70,05	73,29	72,09	71,78	71,80
LED-f(1,3,5,7)$_{g}$	71,79	72,19	69,81	72,03	70,32	70,25	70,25
LED-f(7,5,3,1)$_{a}$	72,31	73,79	71,04	73,17	72,38	72,26	72,26
LED-f(7,5,3,1)$_{g}$	72,24	73,07	71,43	72,19	71,20	71,27	71,27
RBF$_{s}$	**76,79**	69,79	73,20	73,30	75,87	76,03	76,02
RBF$_{m}$	**69,17**	67,62	56,81	66,64	68,24	69,00	69,00
RBF$_{f}$	**45,68**	44,55	35,59	45,18	43,85	43,85	43,85
HYPER$_{s}$	93,93	93,93	93,93	91,43	93,93	93,93	93,93
HYPER$_{m}$	92,08	91,95	92,60	91,06	92,60	92,60	92,60
HYPER$_{f}$	80,93	83,74	83,21	83,31	83,91	83,85	83,85
Avg. Rank Synth	7,06	6,19	9,75	7,54	6,42	8,79	6,58

Table 4.4: Continuada

	LAST$_D$							
Dataset	ADWIN	DDM	HDDM$_A$	HDDM$_W$	RDDM	MDDM$_A$	MDDM$_E$	MDDM$_G$
AGR-f(7,8,9,10,9,8,7)$_{ra}$	80,85	78,11	78,84	78,11	86,62	78,11	78,11	78,11
AGR-f(7,8,9,10,9,8,7)$_{rg}$	81,42	78,10	78,70	78,10	83,80	78,10	78,10	78,10
AGR-f(1,2,3,4,5,6,7,8,9,10)$_a$	78,91	65,02	67,05	65,02	76,59	65,02	65,02	65,02
AGR-f(10,9,8,7,6,5,4,3,2,1)$_a$	80,96	77,25	79,32	77,12	77,95	74,49	74,51	74,51
AGR-f(1,2,3,4,5,6,7,8,9,10)$_g$	76,10	65,03	66,99	65,03	75,54	65,03	65,03	65,03
AGR-f(10,9,8,7,6,5,4,3,2,1)$_g$	77,74	75,41	76,55	76,58	75,57	73,60	73,58	73,58
SEA-f(1,2,3,4,3,2,1)$_{ra}$	85,46	84,07	83,67	85,17	84,83	85,17	85,17	85,17
SEA-f(1,2,3,4,3,2,1)$_{rg}$	85,42	84,02	83,67	85,17	84,80	85,17	85,17	85,17
SEA-f(1,2,3,4)$_a$	86,41	84,49	84,25	86,22	85,80	86,22	86,22	86,22
SEA-f(1,2,3,4)$_g$	86,33	84,47	84,24	86,21	85,77	86,21	86,21	86,21
SEA-f(4,3,2,1)$_a$	86,34	85,52	85,83	85,98	86,05	85,98	85,98	85,98
SEA-f(4,3,2,1)$_g$	86,02	85,48	85,78	85,95	85,96	85,95	85,95	85,95
LED-f(7,5,3,1,3,5,7)$_a$	69,25	63,99	68,50	67,64	63,99	63,99	63,99	63,99
LED-f(7,5,3,1,3,5,7)$_g$	68,69	63,99	68,38	67,68	63,99	63,99	63,99	63,99
LED-f(1,3,5,7)$_a$	70,11	64,73	70,74	65,16	64,73	64,73	64,73	64,73
LED-f(1,3,5,7)$_g$	69,48	64,71	70,29	65,13	64,71	64,71	64,71	64,71
LED-f(7,5,3,1)$_a$	71,11	68,32	70,58	70,38	68,32	68,32	68,32	68,32
LED-f(7,5,3,1)$_g$	70,89	68,28	70,43	70,33	68,28	68,28	68,28	68,28
RBF$_s$	74,41	42,67	62,41	48,08	42,67	42,67	42,67	42,67
RBF$_m$	66,16	33,32	42,96	35,05	33,32	33,32	33,32	33,32
RBF$_f$	36,35	30,43	30,95	30,69	30,43	30,43	30,43	30,43
HYPER$_s$	93,93	93,93	93,93	93,93	90,15	93,93	93,93	93,93
HYPER$_m$	92,58	92,58	92,58	92,58	89,88	92,58	92,58	92,58
HYPER$_f$	79,44	79,44	79,44	79,44	82,70	79,44	79,44	79,44
Avg. Rank Synth	11,31	18,77	15,90	16,23	16,88	17,94	17,94	17,94

Referências

AGRAWAL, R.; IMIELINSKI, T.; SWAMI, A. Database mining: A performance perspective. *IEEE transactions on knowledge and data engineering*, IEEE, v. 5, n. 6, p. 914–925, 1993.

BAENA-GARCIA, M.; CAMPO-AVILA, J. D.; FIDALGO, R.; BIFET, A.; GAVALDA, R.; MORALES-BUENO, R. Early drift detection method. In: *In Fourth International Workshop on Knowledge Discovery from Data Streams*. [S.l.: s.n.], 2006.

BARDDAL, J. P.; ENEMBRECK, F. Learning regularized hoeffding trees from data streams. In: *Proceedings of the 34rd Annual ACM Symposium on Applied Computing, SAC 2019, Limassol, Cyprus, April 08-12, 2019*. [S.l.: s.n.], 2019.

BARDDAL, J. P.; LOEZER, L.; ENEMBRECK, F.; LANZUOLO, R. Lessons learned from data stream classification applied to credit scoring. *Expert Systems with Applications*, v. 162, p. 113899, 08 2020.

BARROS, R. S.; CABRAL, D. R.; GONçALVES, P. M.; SANTOS, S. G. Rddm: Reactive drift detection method. *Expert Systems with Applications*, v. 90, p. 344–355, 2017. ISSN 0957-4174. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0957417417305614>.

BARROS, R. S. M. d.; SANTOS, S. Garrido T. de C.; JÚNIOR, P. M. G. A boosting-like online learning ensemble. In: *2016 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2016. p. 1871–1878.

BENAVOLI, A.; CORANI, G.; MANGILI, F. Should we really use post-hoc tests based on mean-ranks? *Journal of Machine Learning Research*, v. 17, n. 5, p. 1–10, 2016. Disponível em: <http://jmlr.org/papers/v17/benavoli16a.html>.

BENGIO, Y.; DELALLEAU, O.; SIMARD, C. Decision trees do not generalize to new variations. *Computational Intelligence*, v. 26, n. 4, p. 449–467, 2010.

BIFET, A.; GAVALDÀ, R. Learning from time-changing data with adaptive windowing. In: . [S.l.: s.n.], 2007. v. 7.

BIFET, A.; GAVALDÀ, R. Adaptive learning from evolving data streams. In: ADAMS, N. M.; ROBARDET, C.; SIEBES, A.; BOULICAUT, J.-F. (Ed.). *Advances*

in Intelligent Data Analysis VIII. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 249–260. ISBN 978-3-642-03915-7.

BIFET, A.; HOLMES, G.; PFAHRINGER, B. Leveraging bagging for evolving data streams. In: *Machine Learning and Knowledge Discovery in Databases*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 135–150. ISBN 978-3-642-15880-3.

BIFET, A.; HOLMES, G.; PFAHRINGER, B.; KIRKBY, R.; GAVALDà, R. New ensemble methods for evolving data streams. In: . New York, NY, USA: Association for Computing Machinery, 2009. (KDD '09), p. 139–148. ISBN 9781605584959. Disponível em: <https://doi.org/10.1145/1557019.1557041>.

BIFET, A.; HOLMES, G.; PFAHRINGER, B.; KRANEN, P.; KREMER, H.; JANSEN, T.; SEIDL, T. Moa: Massive online analysis, a framework for stream classification and clustering. In: PMLR. *Proceedings of the first workshop on applications of pattern analysis*. [S.l.], 2010. p. 44–50.

BIFET, A.; MORALES, G. de F.; READ, J.; HOLMES, G.; PFAHRINGER, B. Efficient online evaluation of big data stream classifiers. In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: Association for Computing Machinery, 2015. (KDD '15), p. 59–68. ISBN 9781450336642. Disponível em: <https://doi.org/10.1145/2783258.2783372>.

BIFET, A.; MORALES, G. de F.; READ, J.; HOLMES, G.; PFAHRINGER, B. Efficient online evaluation of big data stream classifiers. In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: Association for Computing Machinery, 2015. p. 59–68. ISBN 9781450336642. Disponível em: <https://doi.org/10.1145/2783258.2783372>.

BLACKARD, J. A.; DEAN, D. J. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and electronics in agriculture*, Elsevier, v. 24, n. 3, p. 131–151, 1999.

BREIMAN, L. Classification and regression trees. In: . Wadsworth, Belmont, CA,: Wadsworth Statistics, 1984.

BREIMAN, L. Bagging predictors. In: SPRINGER. *Machine Learning*. [S.l.], 1996. v. 24, p. 123–140.

BREIMAN, L. Random forests. *Mach. Learn.*, Kluwer Academic Publishers, USA, v. 45, n. 1, p. 5–32, oct 2001. ISSN 0885-6125. Disponível em: <https://doi.org/10.1023/A:1010933404324>.

BUNTINE, W. Learning classification trees. *Statistics and Computing*, v. 2, p. 63–73, 1992.

CANDILLIER, L.; LEMAIRE, V. *Nomao*. 2012. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C53G79.

CATTRAL, R.; OPPACHER, F. *Poker Hand*. 2007. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5KW38.

CAVALCANTE, R. C.; MINKU, L. L.; OLIVEIRA, A. L. I. Fedd: Feature extraction for explicit concept drift detection in time series. In: *2016 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2016. p. 740–747.

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>.

COSTA, V. G.; PEDREIRA, C. E. Recent advances in decision trees: an updated survey. *Artif. Intell. Rev.*, Kluwer Academic Publishers, USA, v. 56, n. 5, p. 4765–4800, oct 2022. ISSN 0269-2821. Disponível em: <https://doi.org/10.1007/s10462-022-10275-5>.

DATAR, M.; GIONIS, A.; INDYK, P.; MOTWANI, R. Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, v. 31, n. 6, p. 1794–1813, 2002.

DEMŠAR, J. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine learning research*, JMLR. org, v. 7, p. 1–30, 2006.

DOMINGOS, P.; HULTEN, G. Mining high-speed data streams. In: *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: Association for Computing Machinery, 2000. (KDD '00), p. 71–80. ISBN 1581132336. Disponível em: <https://doi.org/10.1145/347090.347107>.

ELWELL, R.; POLIKAR, R. Incremental learning of concept drift in nonstationary environments. *IEEE Transactions on Neural Networks*, IEEE, v. 22, n. 10, p. 1517–1531, 2011.

FRÍAS-BLANCO, I.; CAMPO-ÁVILA, J. d.; RAMOS-JIMéNEZ, G.; MORALES-BUENO, R.; ORTIZ-DíAZ, A.; CABALLERO-MOTA, Y. Online and non-parametric drift detection methods based on hoeffding's bounds. *IEEE Transactions on Knowledge and Data Engineering*, v. 27, n. 3, p. 810–823, 2015.

GAMA, J. *Knowledge Discovery from Data Streams*. 1st. ed. [S.l.]: Chapman & Hall/CRC, 2010. ISBN 1439826110.

GAMA, J.; MEDAS, P.; CASTILLO, G.; RODRIGUES, P. Learning with drift detection. In: SPRINGER. *Advances in Artificial Intelligence–SBIA 2004: 17th Brazilian Symposium on Artificial Intelligence, Sao Luis, Maranhao, Brazil, September 29-Ocotber 1, 2004. Proceedings 17*. [S.l.], 2004. p. 286–295.

GAMA, J.; SEBASTIAO, R.; RODRIGUES, P. On evaluating stream learning algorithms. *Machine Learning*, Springer, v. 90, p. 317–346, 2013.

GAMA, J. a.; ROCHA, R.; MEDAS, P. Accurate decision trees for mining high-speed data streams. In: . New York, NY, USA: Association for Computing Machinery, 2003. (KDD '03), p. 523–528. ISBN 1581137370. Disponível em: <https://doi.org/10.1145/956750.956813>.

GARCíA, S.; HERRERA, F. An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. *Journal of Machine Learning Research*, v. 9, n. 89, p. 2677–2694, 2008. Disponível em: <http://jmlr.org/papers/v9/garcia08a.html>.

GEURTS, P.; ERNST, D.; WEHENKEL, L. Extremely randomized trees. *Machine Learning*, Springer, v. 63, p. 3–42, 2005.

GOMES, H. M.; BARDDAL, J. P.; ENEMBRECK, F.; BIFET, A. A survey on ensemble learning for data stream classification. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 50, n. 2, mar 2017. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3054925>.

GOMES, H. M.; BIFET, A.; READ, J.; BARDDAL, J. P.; ENEMBRECK, F.; PFHARINGER, B.; HOLMES, G.; ABDESSALEM, T. Adaptive random forests for evolving data stream classification. *Machine Learning*, Springer, v. 106, p. 1469–1495, 2017.

GOMES, H. M.; GRZENDA, M.; MELLO, R.; READ, J.; NGUYEN, M. H. L.; BIFET, A. A survey on semi-supervised learning for delayed partially labelled data streams. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 55, n. 4, nov 2022. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3523055>.

GOMES, H. M.; READ, J.; BIFET, A. Streaming random patches for evolving data stream classification. In: *2019 IEEE International Conference on Data Mining (ICDM)*. [S.l.: s.n.], 2019. p. 240–249.

GOMES, H. M.; READ, J.; BIFET, A.; BARDDAL, J. P.; GAMA, J. a. Machine learning for streaming data: State of the art, challenges, and opportunities. *SIGKDD Explor. Newsl.*, Association for Computing Machinery, New York, NY, USA, v. 21, n. 2, p. 6–22, nov 2019. ISSN 1931-0145. Disponível em: <https://doi.org/10.1145/3373464.3373470>.

GUNASEKARA, N.; PFAHRINGER, B.; GOMES, H.; BIFET, A. Gradient boosted trees for evolving data streams. *Machine Learning*, Springer, v. 113, p. 3325–3352, 2024.

HAN, J.; KAMBER, M.; PEI, J. *Data Mining: Concepts and Techniques*. Radarweg 29, 1043 NX Amsterdam, The Netherlands: Elsevier, 2011.

HARRIES, M.; WALES, N. S. et al. Splice-2 comparative evaluation: Electricity pricing. University of New South Wales, School of Computer Science and Engineering, 1999.

HE, H.; MA, Y. Assessment metrics for imbalanced learning. In: ____. *Imbalanced Learning: Foundations, Algorithms, and Applications*. [S.l.: s.n.], 2013. p. 187–206.

HOEFFDING, W. Probability inequalities for sums of bounded random variables. *The collected works of Wassily Hoeffding*, Springer, p. 409–426, 1963.

HULTEN, G.; SPENCER, L.; DOMINGOS, P. Mining time-changing data streams. In: . New York, NY, USA: Association for Computing Machinery, 2001. (KDD '01), p. 97–106. ISBN 158113391X. Disponível em: <https://doi.org/10.1145/502512.502529>.

IKONOMOVSKA, E.; GAMA, J.; DŽEROSKI, S. Learning model trees from evolving data streams. In: *Data Min Knowl Disc*. [S.l.: s.n.], 2011. v. 23, p. 128–168.

JAWORSKI, M.; DUDA, P.; RUTKOWSKI, L. New splitting criteria for decision trees in stationary data streams. *IEEE Transactions on Neural Networks and Learning Systems*, v. 29, n. 6, p. 2516–2529, 2018.

KIRKBY, R. *Improving hoeffding trees*. Tese (Doutorado) — The University of Waikato, 2007.

KOHAVI, R.; KUNZ, C. Option decision trees with majority votes. *Proceedings of the Fourteenth International Conference on Machine Learning*, 1998.

KRAWCZYK, B.; MINKU, L. L.; GAMA, J.; STEFANOWSKI, J.; WOźNIAK, M. Ensemble learning for data stream analysis: A survey. *Information Fusion*, v. 37, p. 132–156, 2017. ISSN 1566-2535. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1566253516302329>.

KUNCHEVA, L. I.; WHITAKER, C. J. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. In: SPRINGER. *Machine Learning*. [S.l.], 2003. v. 51, p. 181–207.

LOSING, V.; HAMMER, B.; WERSING, H. Interactive online learning for obstacle classification on a mobile robot. In: *2015 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2015. p. 1–8.

LOSING, V.; HAMMER, B.; WERSING, H. Knn classifier with self adjusting memory for heterogeneous concept drift. In: IEEE. *2016 IEEE 16th international conference on data mining (ICDM)*. [S.l.], 2016. p. 291–300.

LOUPPE, G.; GEURTS, P. Ensembles on random patches. In: FLACH, P. A.; BIE, T. D.; CRISTIANINI, N. (Ed.). *Machine Learning and Knowledge Discovery in Databases*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 346–361. ISBN 978-3-642-33460-3.

LU, J.; LIU, A.; DONG, F.; GU, F.; GAMA, J.; ZHANG, G. Learning under concept drift: A review. *IEEE transactions on knowledge and data engineering*, IEEE, v. 31, n. 12, p. 2346–2363, 2018.

MANAPRAGADA, C.; GOMES, H.; SALEHI, M.; BIFET, A.; WEBB, G. I. An eager splitting strategy for online decision trees in ensembles. In: *Data Min Knowl Disc*. [S.l.: s.n.], 2022. v. 36, p. 566–619.

MANAPRAGADA, C.; WEBB, G. I.; SALEHI, M. Extremely fast decision tree. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. New York, NY, USA: Association for Computing Machinery, 2018. (KDD '18), p. 1953–1962. ISBN 9781450355520. Disponível em: <https://doi.org/10.1145/3219819.3220005>.

MCDIARMID, C. On the method of bounded differences. In: *Surveys in combinatorics, 1989 (Norwich, 1989)*. [S.l.]: Cambridge Univ. Press, Cambridge, 1989, (London Math. Soc. Lecture Note Ser., v. 141). p. 148–188.

MIDDLEHURST PATRICK SCHäFER, A. B. M. Bake off redux: a review and experimental evaluation of recent time series classification algorithms. *Data Mining and Knowledge Discovery*, Springer, v. 38, p. 1958—-2031, 2024.

MITCHELL, T. M. Machine learning. In: *McGraw-Hill Education*. [S.l.: s.n.], 1997.

MONTIEL, J.; HALFORD, M.; MASTELINI, S. M.; BOLMIER, G.; SOURTY, R.; VAYSSE, R.; ZOUITINE, A.; GOMES, H. M.; READ, J.; ABDESSALEM, T.; BIFET, A. River: Machine learning for streaming data in python. JMLR.org, v. 22, n. 1, jan 2021. ISSN 1532-4435.

MORALES, G. D. F.; BIFET, A.; KHAN, L.; GAMA, J.; FAN, W. Iot big data stream mining. 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. 2119–2120. ISBN 9781450342322. Disponível em: <https://doi.org/10.1145/2939672.2945385>.

OZA, N. C.; RUSSELL, S. J. Online bagging and boosting. In: RICHARDSON, T. S.; JAAKKOLA, T. S. (Ed.). *Proceedings of the Eighth International Workshop on Artificial Intelligence and Statistics*. PMLR, 2001. (Proceedings of Machine Learning Research, R3), p. 229–236. Reissued by PMLR on 31 March 2021. Disponível em: <https://proceedings.mlr.press/r3/oza01a.html>.

PAIM, A. M.; ENEMBRECK, F. Adaptive random tree ensemble for evolving data stream classification. *Knowledge-Based Systems*, v. 309, p. 112830, 2024. ISSN 0950-7051. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0950705124014643>.

PAIM, A. M.; ENEMBRECK, F. Adaptive regularized ensemble for evolving data stream classification. *Pattern Recognition Letters*, v. 180, p. 55–61, 2024. ISSN

0167-8655. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167865524000576>.

PESARANGHADER, A.; VIKTOR, H. L.; PAQUET, E. Mcdiarmid drift detection methods for evolving data streams. In: *2018 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2018. p. 1–9.

PFAHRINGER, B.; HOLMES, G.; KIRKBY, R. New options for hoeffding trees. In: ORGUN, M. A.; THORNTON, J. (Ed.). *AI 2007: Advances in Artificial Intelligence*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. p. 90–99.

QUINLAN, J. Induction of decision trees. In: SPRINGER. *Machine Learning*. [S.l.], 1986. v. 1, p. 81–106.

QUINLAN, J. C4.5: Programs for machine. In: . 340 Pine Street, 6th Floor San Francisco, CA 94104 USA: Morgan Kaufmann Publishers, 1992.

RAMÍREZ-GALLEGO, S.; KRAWCZYK, B.; GARCíA, S.; WOŹNIAK, M.; HERRERA, F. A survey on data preprocessing for data stream mining: Current status and future directions. *Neurocomputing*, v. 239, p. 39–57, 2017. ISSN 0925-2312. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0925231217302631>.

ROSA, R. D.; CESA-BIANCHI, N. Splitting with confidence in decision trees with application to stream mining. In: *2015 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2015. p. 1–8.

RUTKOWSKI, L.; JAWORSKI, M.; DUDA, P. *Stream Data Mining: Algorithms and Their Probabilistic Properties*. [S.l.: s.n.], 2020. ISBN 978-3-030-13961-2.

RUTKOWSKI, L.; PIETRUCZUK, L.; DUDA, P.; JAWORSKI, M. Decision trees for mining data streams based on the mcdiarmid's bound. *IEEE Transactions on Knowledge and Data Engineering*, v. 25, n. 6, p. 1272–1279, 2013.

SCHAPIRE, R. E. A brief introduction to boosting. In: *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999. (IJCAI'99), p. 1401–1406.

SCHLIMMER, J. C.; GRANGER, R. H. Incremental learning from noisy data. *Machine Learning*, v. 1, p. 317–354, 1986.

SHANNON, C. E. A mathematical theory of communication. *The Bell System Technical Journal*, v. 27, p. 379–423, 1948.

SOUZA, V. M. Asphalt pavement classification using smartphone accelerometer and complexity invariant distance. *Engineering Applications of Artificial Intelligence*, v. 74, p. 198–211, 2018. ISSN 0952-1976. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0952197618301349>.

SOUZA, V. M.; PARMEZAN, A. R.; CHOWDHURY, F. A.; MUEEN, A. Efficient unsupervised drift detector for fast and high-dimensional data streams. *Knowledge and Information Systems*, Springer, v. 63, p. 1497–1527, 2021.

SOUZA, V. M.; REIS, D. M. dos; MALETZKE, A. G.; BATISTA, G. E. Challenges in benchmarking stream learning algorithms with real-world data. *Data Mining and Knowledge Discovery*, Springer, v. 34, p. 1805–1858, 2020.

STREET, W. N.; KIM, Y. A streaming ensemble algorithm (sea) for large-scale classification. In: *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: Association for Computing Machinery, 2001. (KDD '01), p. 377–382. ISBN 158113391X. Disponível em: <https://doi.org/10.1145/502512.502568>.

WELFORD, B. P. Note on a method for calculating corrected sums of squares and products. *Technometrics*, Taylor & Francis, v. 4, n. 3, p. 419–420, 1962.

ZHAO, Z.; HONG, L.; WEI, L.; CHEN, J.; NATH, A.; ANDREWS, S.; KUMTHEKAR, A.; SATHIAMOORTHY, M.; YI, X.; CHI, E. Recommending what video to watch next: a multitask ranking system. In: *Proceedings of the 13th ACM Conference on Recommender Systems*. New York, NY, USA: Association for Computing Machinery, 2019. (RecSys '19), p. 43–51. ISBN 9781450362436. Disponível em: <https://doi.org/10.1145/3298689.3346997>.

ZHONG, S.; SOUZA, V. M.; MUEEN, A. Combining filtering and cross-correlation efficiently for streaming time series. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, ACM New York, NY, v. 16, n. 5, p. 1–24, 2022.

ŽLIOBAITė, I.; BIFET, A.; READ, J.; PFAHRINGER, B.; HOLMES, G. Evaluation methods and decision theory for classification of streaming data with temporal dependence. *Machine Learning*, Springer, v. 98, p. 455–482, 2015.