UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

EIFUZZCND: UMA ESTRATÉGIA INCREMENTAL PARA CLASSIFICAÇÃO MULTICLASSE E DETECÇÃO DE NOVIDADES EM FLUXOS DE DADOS

LUCAS RICARDO DUARTE BRUZZONE

ORIENTADORA: PROF. DRA. HELOISA DE ARRUDA CAMARGO

São Carlos – SP

Outubro/2023

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

EIFUZZCND: UMA ESTRATÉGIA INCREMENTAL PARA CLASSIFICAÇÃO MULTICLASSE E DETECÇÃO DE NOVIDADES EM FLUXOS DE DADOS

LUCAS RICARDO DUARTE BRUZZONE

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Ciências da computação

Orientadora: Prof. Dra. Heloisa de Arruda Camargo

São Carlos – SP

Outubro/2023

SUMÁRIO

Capítulo 1

INTRODUÇÃO

Na era dos grandes volumes de dados e fluxos de informações, a capacidade de detectar padrões e anomalias em tornou-se uma necessidade crítica. Essa necessidade é particularmente evidente em domínios como cibersegurança, detecção de fraudes e controle de qualidade industrial. A detecção de novidades, que envolve a identificação de instâncias ou padrões que diferem significativamente das normas estabelecidas, desempenha um papel fundamental na garantia da integridade e segurança de sistemas impulsionados por dados (FARIA et al., 2015).

O advento dos fluxos de dados, caracterizados por sua natureza contínua e alta velocidade, acrescentou complexidade à tarefa de detectar novidades (GAMA, 2010). Métodos tradicionais de processamento em lote muitas vezes não atendem a esse contexto, exigindo o desenvolvimento de algoritmos inovadores capazes de processar dados mantendo alta precisão.

Uma abordagem notável que se mostrou bem-sucedida é a estratégia offline-online. Essa abordagem envolve o pré-processamento dos dados em lote (offline), onde modelos são treinados com um conjunto de dados históricos para identificar padrões. Em seguida, esses modelos são ajustados e adaptados continuamente (online) à medida que novos dados são recebidos. Essa hierarquia de processamento, conforme (FARIA et al., 2016a), inclui etapas treinamento de subconjuntos, agrupamento de micro-grupos e detecção de padrões de novidade.

É importante ressaltar que a área de detecção de novidades está em constante evolução, impulsionada pela crescente necessidade de lidar com fluxos de dados. A pesquisa nesse campo continua ativa, com cientistas e engenheiros explorando novos algoritmos, técnicas de aprendizado de máquina, e abordagens interdisciplinares para lidar com os desafios únicos apresentados por essa demanda. A detecção de novidades ainda enfrenta desafios em aberto, como lidar com a imprevisibilidade dos dados e garantir a escalabilidade e eficiência dos algoritmos em ambientes de alto fluxo de informações. Portanto, é uma área em constante desenvolvimento que

desempenha um papel crítico na garantia da integridade e segurança de sistemas impulsionados por dados.

Um exemplo de algoritmo que foi proposto para lidar com essa tarefa foi o Enchanted Fuzzy Clustering for Novelty Detection (EFuzzCND) (CRISTIANI; CAMARGO, 2021). Apesar das contribuições valiosas que ofereceu, tornou-se evidente que ele possui limitações específicas. Essas limitações incluem sua capacidade limitada de lidar com mudanças súbitas de conceito em fluxos de dados com latência extrema, bem como a necessidade absoluta de rótulos verdadeiros em cenários de latência intermediária. Esses desafios ressaltam a importância de buscar aprimoramentos no algoritmo, visando uma maior adaptabilidade a cenários dinâmicos e uma redução na dependência de rótulos verdadeiros, a fim de torná-lo mais eficaz em uma gama mais ampla de contextos de fluxos de dados.

Este estudo apresenta o algoritmo Enchanted Increemental Fuzzy Clustering for Novelty Detection (EIFuzzCND), uma abordagem para classificação multi classe e detecção de novidades em fluxos de dados, sendo uma melhoria do EFuzzCND. As melhorias propostas foram:

- **Abordagem Incremental**: Implementação de uma abordagem incremental para atualização do modelo de classes conhecidas.
- **Presença de rótulos verdadeiros**: Eliminação ou redução da necessidade de rótulos verdadeiros em cenários de latência intermediária.
- **Matriz de Confusão Incrementa**: Implementação da Matriz de Confusão Incremental proposta por (FARIA et al., 2015) para análise dos resultados.

Para avaliar o desempenho e a versatilidade do algoritmo EIFuzzCND, ele é comparado com algoritmos de referência bem estabelecidos, incluindo EFuzzCND(CRISTIANI; CAMARGO, 2021), ECSMiner (MASUD et al., 2010) e MINAS (FARIA et al., 2016b), em diversos cenários. Métricas-chave de desempenho, como acurácia e taxas de desconhecidos, são examinadas para obter uma compreensão abrangente das capacidades do algoritmo.

Esta pesquisa também implementa uma ferramenta - a matriz de confusão incremental (MCI) proposta em (FARIA et al., 2015) - para aprimorar a análise do comportamento do algoritmo. Ao visualizar o desempenho do algoritmo ao longo do tempo, essa ferramenta fornece insights valiosos sobre como o EIFuzzCND se adapta às mudanças nas distribuições de dados e identifica instâncias novas. A robustez do algoritmo em cenários com sobreposição de classes, desbalanceamento de classes e alta dimensionalidade é examinada usando conjuntos de dados reais e sintéticos.

A avaliação abrange cenários de latência extrema e intermediária, espelhando aplicações do mundo real onde fluxos de dados podem exibir velocidades e características variadas. Além disso, além da funcionalidade incremental, foi proposto um cenário de latência intermediária, onde nem sempre todos os rótulos verdadeiros estão presentes.

Este estudo mostra não somente a eficácia do algoritmo EIFuzzCND, mas também destaca a importância de adaptar algoritmos de detecção de novidades para enfrentar os desafios únicos apresentados por diferentes conjuntos de dados e cenários do mundo real. Ao contribuir para o corpo de conhecimento no campo da análise de fluxos de dados, esta pesquisa abre caminho para avanços adicionais na detecção de novidades em fluxo de dados, com implicações para indústrias que dependem de processos rápidos de tomada de decisões baseados em dados.

A seguir é descrito o conteúdo que foi abordado em cada uma das seções:

- Seção 2 - Conceitos Básicos: Abrange os princípios fundamentais do aprendizado de máquina, incluindo técnicas de agrupamento fuzzy, como Fuzzy C-Means e Possibilistic-Fuzzy C-Means. Também explora a detecção de outliers e novidades em fluxos de dados, com foco nas mudanças de conceito, detecção de novidades e latência de rótulos. Além disso, apresenta diversas técnicas e estruturas utilizadas no contexto do aprendizado de máquina em fluxos de dados, proporcionando uma base sólida para as discussões subsequentes.

- Seção 3 - Revisão Bibliográfica: Concentra-se em uma análise abrangente dos estudos relevantes encontrados na literatura por meio de um mapeamento sistemático. Este processo tem como objetivo identificar, resumir, avaliar e interpretar os estudos publicados na literatura científica. As etapas detalhadas desse processo são descritas neste capítulo, seguidas pela apresentação dos resultados obtidos. Posteriormente, os trabalhos selecionados são minuciosamente discutidos e analisados em relação ao tema de detecção de novidades em fluxos de dados, fornecendo uma visão crítica do estado atual da pesquisa nessa área.

- Seção 4 - Enhanced Incremental Fuzzy Classifier for Multi-Class Novelty Detection in Data Stream: Aborda o problema crítico de detecção de novidades em fluxos de dados, explorando as limitações de abordagens existentes, como o EFuzzCND, e introduzindo o algoritmo proposto, o EIFuzzCND. O EIFuzzCND se destaca por sua capacidade de atualização incremental do modelo de decisão e sua capacidade de lidar com cenários de latência intermediária. Detalha a motivação por trás dessa abordagem e fornece uma visão geral completa do EIFuzzCND, estabelecendo a base para experimentos subsequentes.

- Seção 5 - Experimentos e Métricas de Avaliação: Descreve os aspectos práticos dos experimentos realizados para avaliar o desempenho do EIFuzzCND e de outros algoritmos relevantes. Começa por detalhar os conjuntos de dados utilizados e os parâmetros configurados para os algoritmos, incluindo o EIFuzzCND. Também introduz as métricas de avaliação que serão empregadas para analisar os resultados obtidos.

- Seção 6 - Resultados e Análises dos Experimentos: Apresenta e discute os resultados obtidos por meio dos experimentos detalhados na Seção 5. Os resultados são divididos em três seções distintas, correspondendo aos cenários de latência extrema, latência intermediária e latência intermediária com variação no percentual de rótulos verdadeiros. Cada seção aborda os resultados em termos de métricas de desempenho, fornecendo uma análise crítica do desempenho do EIFuzzCND em comparação com outros algoritmos.

- Seção 7 - Conclusão: Encerra o trabalho com uma síntese dos principais pontos e conclusões alcançadas. Destaca as principais contribuições do estudo, os insights obtidos por meio da pesquisa e as implicações de nossos resultados para a área de detecção de novidades em fluxos de dados. Também define possíveis direções para pesquisas futuras, enfatizando a importância contínua de aprimorar os algoritmos de detecção de novidades em cenários de fluxos de dados em constante evolução.

Capítulo 2

CONCEITOS BÁSICOS

Este capítulo aborda os principais aspectos do aprendizado de máquina clássico, incluindo aprendizado supervisionado e não supervisionado, com foco em técnicas de agrupamento fuzzy, como Fuzzy C-Means e Possibilistic-Fuzzy C-Means. Também é abordada a detecção de outliers e novidades em fluxos de dados, com ênfase na mudança e evolução de conceitos e na latência de rótulos. Por fim, são discutidas técnicas de aprendizado de máquina em fluxos de dados, incluindo classificação por árvore de decisão e agrupamento por micro-grupo, Fuzzy Micro-Cluster, Supervised Fuzzy Micro-Cluster e Supervised Possibilistic Fuzzy Micro-Cluster.

2.1 Aprendizado de máquina clássico

Define-se o aprendizado de máquina (AM) como o estudo de algoritmos que permitem que programas computacionais aprendam e evoluam automaticamente através de experiências (MITCHELL, 1997). Estes algoritmos utilizam fatores relacionados ao aprendizado e a inteligência, possibilitando estes de serem descritos com tamanha precisão que uma máquina possa ser desenvolvida para simula-los (MCCARTHY et al., 2006). Esses fatores relacionados são classificados em diferentes métodos de aprendizado como hábito, instrução, analogia e indução.

O aprendizado indutivo vem sendo usado mais comumente no contexto de AM. Este tipo de aprendizado se caracteriza como uma forma de indução lógica para obter conclusões universais a partir de um conjunto de premissas, esse sendo denominado conjunto de treinamento. O conjunto de treinamento, apresenta diversos exemplos previamente observados, que podem ser representados por um vetor n-dimensional, sendo n o número de características ou *features* que cada exemplo possui. Não necessariamente as hipóteses geradas representam a verdade, porém o aprendizado indutivo é um dos principais métodos utilizados para detectar padrões ou predizer eventos futuros (MITCHELL, 1997). Sendo assim, este aprendizado pode ser categorizado em

supervisionado, não supervisionado.

2.1.1 Aprendizado supervisionado

Este tipo de aprendizado busca algoritmos que raciocinam a partir de instâncias fornecidas externamente para produzir hipóteses gerais, que então fazem projeções sobre instâncias futuras (KOTSIANTIS et al., 2007). O algoritmo cria um modelo que é responsável por fazer previsões futuras, e o desempenho desse modelo pode ser avaliado por várias medidas, uma delas é a acurácia.

No aprendizado supervisionado, o conjunto de dados utilizado para treinar o modelo é chamado de conjunto de treinamento. Cada exemplo nesse conjunto possui um conjunto de características, também conhecido como *features*, e uma saída desejada (rótulo ou valor) correspondente. O modelo então aprende a associar as características às saídas desejadas para fazer previsões precisas sobre novas entradas.

Após o treinamento, o desempenho do modelo é avaliado utilizando um conjunto de dados separado chamado de conjunto de teste. Esse conjunto é composto por exemplos que o modelo nunca viu antes, e o objetivo é avaliar como o modelo generaliza para novos dados.

Existem duas subcategorias no aprendizado supervisionado:

- Classificação: Nessa tarefa, o modelo atribui um rótulo (classe) a uma entrada. Por exemplo, um modelo pode ser treinado para classificar comentários de usuários como "positivo"ou "negativo"com base no texto. O conjunto de treinamento incluiria exemplos de comentários rotulados como "positivo"ou "negativo", e o modelo aprenderia a associar as características do texto com o rótulo correto.

- Regressão: Nessa tarefa, o modelo estuda a relação entre as variáveis dependentes e independentes, onde os valores que estão sendo previstos possuem um aspecto contínuo. Por exemplo, um modelo pode ser treinado para prever a taxa de internação com base na quantidade da população que foi vacinada. O conjunto de treinamento incluiria exemplos com as quantidades de população vacinada e as taxas de internação correspondentes, e o modelo aprenderia a associar as características com os valores numéricos previstos.

Essas subcategorias são adequadas para conjuntos de dados rotulados, onde já se sabe qual é a saída desejada para cada exemplo. No entanto, em muitos casos, os conjuntos de dados não estão rotulados. Para analisar esses conjuntos, existe o aprendizado não supervisionado.

2.1.2 Aprendizado não supervisionado

O aprendizado não supervisionado é uma técnica de análise de dados que utiliza algoritmos para identificar padrões ocultos ou agrupamentos em dados não rotulados, sem a necessidade de intervenção humana (GHAHRAMANI, 2003).

O agrupamento é a tarefa mais comum, e, portanto, será detalhada neste trabalho. Existem outros tipos de algoritmos que utilizam a extração de regras de associação, os quais não serão abordados neste trabalho devido ao nosso foco em agrupamento. Para referências sobre regras de associação, pode-se consultar (KOTSIANTIS; KANELLOPOULOS, 2006).

As técnicas de agrupamento são empregadas para agrupar exemplos observados em conjuntos ou *clusters* que atendam a dois critérios principais:

- Cada grupo é homogêneo, o que significa que os exemplos pertencentes ao mesmo grupo são semelhantes entre si.
- Cada grupo deve ser distinto dos outros grupos, ou seja, os exemplos em um grupo devem ser diferentes dos exemplos em outros grupos.

A representação dos grupos pode variar dependendo da técnica utilizada.

As técnicas de agrupamento são utilizadas para dividir um conjunto de dados em grupos, com o objetivo de encontrar padrões ou estruturas subjacentes. Podemos classificar as técnicas de agrupamento em cinco categorias:

- Particional: Os métodos de particionamento geralmente resultam em um conjunto de N grupos, onde cada objeto pertence a um grupo. Cada grupo pode ser representado por um centroide ou por um representante do grupo, que é algum tipo de elemento típico do grupo. Um exemplo muito conhecido desta categoria é o algoritmo K-Means (MACQUEEN, 1967).
- Hierárquico: Diferentemente dos métodos particionais, em que é necessário definir o número de clusters antecipadamente, os algoritmos hierárquicos constroem grupos de forma gradual e aninhada, de modo que cada nível específico da hierarquia representa uma partição dos dados. (MURTAGH; CONTRERAS, 2012).
- Baseado em densidade: Algoritmos baseados em densidade são capazes de descobrir grupos de formas arbitrárias. Esses algoritmos agrupam objetos de acordo com funções objetivas de densidade específica. Um exemplo de algoritmo que utiliza essa técnica é

o DBSCAN (do inglês, Density-Based Spatial Clustering of Applications with Noise) (ESTER et al., 1996).

- Baseado em grades: Se concentra em dados espaciais, ou seja, os dados que modelam a estrutura geométrica dos objetos no espaço, suas relações, propriedades e operações. Esta técnica quantiza o conjunto de dados, constrói vários níveis hierárquicos de grupos de objetos e a fusão de grades e consequentemente grupos não depende de uma distância medida, mas sim é determinada por um parâmetro predefinido. Um algoritmo utilizado nesta categoria é o CLIQUE (em inglês, Clustering in QUEst) (AGRAWAL et al., 1998).

- Baseado em grafos: No agrupamento baseado em grafos, os dados são representados como um grafo, onde os vértices do grafo representam os objetos de dados e as arestas entre os vértices representam as relações ou similaridades entre esses objetos. A partir dessa representação em grafo, o objetivo do agrupamento baseado em grafos é particionar o grafo em subgrafos de tal forma que as arestas dentro de cada subgrafo tenham pesos altos (indicando alta similaridade entre os objetos representados pelos vértices) e as arestas entre diferentes subgrafos tenham pesos baixos (indicando baixa similaridade entre os objetos representados pelos vértices de diferentes subgrafos). Um exemplo de algoritmo que utiliza essa técnica é o HSC (Highly Connected Subgraph), que busca identificar subgrafos altamente conectados em um grafo de similaridade entre os pontos de dados.

Dentre os algoritmos mais populares, está o K-Means. Este algoritmo visa particionar um conjunto de dados $X = (x_1, \ldots, x_N)$ em M subconjuntos, chamados de grupos $(C_1, \ldots, C_M)$, de forma a otimizar o agrupamento. O critério utilizado é a soma dos quadrados das distâncias euclidianas entre cada ponto de dado x_i e o centroide (centro do cluster) m_k de cada grupo C_k que contenha x_i (LIKAS; VLASSIS; VERBEEK, 2003). O erro de clusterização é descrito pela Equação 2.1, e depende diretamente dos centros dos clusters $m_1, \ldots, m_M$.

$$E(m_1, \ldots, m_M) = \sum_{k=1}^{M} \sum_{i=1}^{N} I(x_i \in C_k) ||x_i - m_k||^2 \tag{2.1}$$

Onde:

- $E(m_1, \ldots, m_M)$: Esta é a função de erro de clusterização, que é o critério utilizado para avaliar o desempenho do algoritmo de K-Means. O objetivo do K-Means é minimizar essa função, ajustando os centros dos clusters.

- N: É o número total de pontos de dados em seu conjunto de dados.

- M: Representa o número de clusters que você deseja criar no processo de agrupamento.

- k: É um índice que varia de 1 a M, indicando o cluster atual em consideração.

- i: É um índice que varia de 1 a N, indicando o ponto de dados atual em consideração.

- $I(x_i \in C_k)$: É uma função indicadora que avalia se o ponto de dados x_i pertence ao cluster atual (C_k) ou não. Ela retorna 1 se x_i pertencer ao cluster C_k e 0 caso contrário.

- $||x_i - m_k||^2$: Representa a distância euclidiana entre o ponto de dados x_i e o centroide (centro do cluster) m_k do cluster C_k que contém x_i. A notação $||\ldots||_2$ representa a norma Euclidiana, que é a distância euclidiana padrão entre dois pontos em um espaço Euclidiano.

A equação 2.1 descreve o erro de clusterização total como a soma das distâncias quadráticas dos pontos de dados para os centros dos clusters apropriados, ponderadas pela função indicadora $I(x_i \in C_k)$. O objetivo do algoritmo K-Means é ajustar os centros dos clusters de forma iterativa para minimizar essa função, o que leva à formação de clusters bem definidos.

As técnicas de agrupamento apresentadas anteriormente podem apresentar limitações, uma vez que, devido à imprecisão dos dados provenientes de diferentes fontes, seria mais apropriado se as instâncias pudessem ser atribuídas a mais de um cluster (BEZDEK; EHRLICH; FULL, 1984). O agrupamento *fuzzy* surge como uma solução mais flexível, permitindo que as instâncias pertençam a diferentes clusters com diferentes graus de associação (BEZDEK, 2013). Nesta seção, serão descritos o *Fuzzy C-Means* e o *Possibilistic-Fuzzy C-Means*, que servem como base para o desenvolvimento deste trabalho.

2.1.2.1 *Fuzzy C-Means*

O algoritmo *Fuzzy C-Means* (FCM) foi proposto como uma extensão do algoritmo *K-Means*, incorporando os conceitos da teoria dos conjuntos *fuzzy* (BEZDEK; EHRLICH; FULL, 1984). Como no *K-Means*, este algoritmo procura minimizar a função objetivo para agrupar N instâncias em C *clusters*, como mostra a equação (2.2):

$$J_m = \sum_{i=1}^{N}\sum_{j=1}^{C} u_{ij}^{m}||x_i - c_j||^2, 1 \leq m < \infty \tag{2.2}$$

onde m representa o parâmetro de fuzzificação e pode ser qualquer número real maior que 1, u representa o grau de pertinência de x_i no cluster j, x_i representa o i-ésimo elemento do conjunto de instâncias, c_j o centroide do cluster e ||*|| representa qualquer norma que demonstre a semelhança entre os dados analisados e o centro do cluster.

O agrupamento FCM deve atender às restrições impostas pelas equações (2.3) e (2.4). A

equação (2.3) define que a soma dos membros de um dado elemento x_i a todos os agrupamentos deve ser 1.

$$\sum_{j=1}^{K} u_{ij} = 1 \tag{2.3}$$

A equação (2.4) define que a soma dos membros dos elementos pertencentes a um determinado cluster j deve ser menor que o número de instâncias no conjunto de dados e deve conter pelo menos um elemento com um grau de pertinência maior que 0.

$$0 < \sum_{i=1}^{N} u_{ij} < N \tag{2.4}$$

O particionamento *fuzzy* é realizado através de uma otimização iterativa da equação (2.2), considerando o grau de pertinência u_{ij} e o centroide do cluster c_j, definidos por:

$$u_{ij} = \frac{1}{\sum_{k=1}^{C} (\frac{||x_i - c_j||}{||x_i - c_k||})^{\frac{2}{m-1}}} \tag{2.5}$$

$$c_j = \frac{\sum_{i=1}^{N} u_{ij}^m * x_i}{\sum_{i=1}^{N} u_{ij}^m} \tag{2.6}$$

Existem dois critérios de parada para a iteração da equação (2.2) do FCM. O primeiro critério, semelhante ao algoritmo *K-Means*, termina a iteração quando nenhum elemento sofre alteração de cluster. O segundo critério ocorre quando a condição $max_{i,j}|u_{ij}^{(k+1)} - u_{ij}^{(k)}|$ é menor ou igual a um determinado limite definido previamente.

2.1.2.2 *Possibilistic-Fuzzy C-Means*

O algoritmo Possibilistic-Fuzzy C-Means (PFCM) foi proposto como uma junção entre o Possibilistic C-Means (PCM) (KRISHNAPURAM; KELLER, 1993) e o Fuzzy C-Means (FCM) (BEZDEK; EHRLICH; FULL, 1984), com o objetivo de resolver problemas como a sensibilidade ao ruído presente no FCM, coincidência de clusters no PCM e eliminar as restrições de soma de linha presentes no Fuzzy-Possibilistic C-Means (FPCM), algoritmo anterior ao PFCM que propõe suprir a necessidade de calcular valores de associação e tipicidade ao agrupar os dados. Os conceitos de pertinência e tipicidades são descritos da seguinte forma:

Pertinência: Refere-se ao grau de pertencimento de um ponto de dados a um cluster em algoritmos de agrupamento fuzzy. Ela indica a probabilidade de um ponto de dados pertencer

a um cluster, variando de 0 (indicando que o ponto não pertence ao cluster) a 1 (indicando pertencimento total).

Tipicidade: Este conceito relacionado mede o grau de tipicidade ou representatividade de um cluster para um ponto de dados. A tipicidade está relacionada à adequação de um ponto ao cluster e também varia de 0 (indicando que o cluster não é típico para o ponto) a 1 (indicando que o cluster é altamente típico para o ponto).

O método PFCM busca minimizar a função objetivo definida na equação 2.7:

$$J = \sum_{i=1}^{r} \sum_{k=1}^{n} [a(u_{ik})^m + b(t_{ik})^\eta] + d_{ik} + \pi_i(1 - t_{ik}^\eta) \tag{2.7}$$

Onde:

- d_{ik} representa a distância euclidiana entre o objeto k e o protótipo do cluster i.

- a e b representam a importância do grau de pertinência e tipicidade para a clusterização, e esses valores são definidos pelo usuário.

- η refere-se à importância de considerar os graus de tipicidade t_{ik} para o agrupamento.

- π_i é usado para equilibrar os dois termos da equação e garantir que o critério seja calculado de forma a maximizar os valores de t_{ik}.

- r é o número de clusters.

- n é o número de objetos.

No entanto, os autores sugerem usar a média ponderada das distâncias definida por:

$$\pi_i = K \frac{\sum_{k=1}^{n} (u_{ik})^m d_{ik}}{\sum_{k=1}^{n} (u_{ik})^m} \tag{2.8}$$

onde se considera que K geralmente assume o valor 1 (PAL et al., 2005).

Para realizar a minimização da função objetivo, são utilizados o grau de relevância e tipicidade. A equação 2.9 basicamente calcula uma média ponderada dos valores dos atributos do ponto de dados k com base nos graus de pertinência (u_{ik}) e tipicidade (t_{ik}). Isso é feito somando os produtos dos valores dos atributos (x_{ik}) pelos termos de grau de pertinência e tipicidade e, em seguida, dividindo essa soma pela soma dos termos de grau de pertinência e tipicidade para normalizar os valores.

$$y_{ij} = \frac{\sum_{k=1}^{n}[a(u_{ik})^m + b(t_{ik})^\eta]x_{ik}}{\sum_{k=1}^{n}[a(u_{ik})^m + b(t_{ik})\eta]} \tag{2.9}$$

O resultado y_{ij} representa o grau de pertinência do ponto de dados x_{ik} ao cluster i, levando em consideração tanto o grau de pertinência quanto o grau de tipicidade, com suas importâncias ponderadas pelos parâmetros a e b, respectivamente. Esse cálculo é uma parte fundamental do algoritmo PFCM, pois ajuda a determinar a associação de cada ponto de dados aos clusters durante o processo de agrupamento.

O grau de pertinência é calculado sob a restrição $\sum_{i=1}^{c} u_{ik} = 1$ e é definido por:

$$u_{ik} = \left[\sum_{h=1}^{c}\left(\frac{d_{ik}}{d_{hk}}\right)^{\frac{2}{m-1}}\right]^{-1} \tag{2.10}$$

Por fim, devido ao fato de não haver limitação de soma, o grau de tipicidade e relevância podem ter valores diferentes, além do valor de b, o que impacta diretamente no valor de t_{ik}. Quanto maior o valor de b, menor o valor de t_{ik}, e consequentemente, o grau de tipicidade diminui em relação à distância. O grau de tipicidade é definido por:

$$t_{ik} = \left[1 + \left(b\frac{d_{ik}}{\pi_i}\right)^{\frac{2}{(n-1)}}\right] \tag{2.11}$$

2.1.3 Definição de *Outlier*, ruído e anomalias

Outlier, ruído e anomalias são termos relacionados e possuem definições muito similares na literatura. Para simplificar o entendimento do trabalho, esses termos serão definidos de acordo com (AGGARWAL, 2017).

Outlier é um exemplo que difere significativamente e de maneira incomum de outros exemplos conhecidos. Isso pode ocorrer devido a uma variabilidade na medição ou um erro experimental, e nesses casos, ele deve ser excluído do conjunto de dados para que não interfira negativamente na análise. O conceito de *outlier* engloba ruídos e anomalias e é encontrado na literatura como um termo mais genérico.

O termo anomalia é um tipo de *outlier* que se distancia dos exemplos já observados, mas traz informações novas para o modelo. Essa informação pode ser vista como intrusos em uma rede de computadores, fraudes em cartões de crédito, entre outros. Em contrapartida, o termo ruído é definido como qualquer exemplo que se desvie do modelo normal, mas não é relevante.

Os exemplos considerados ruidosos devem ser descartados, pois podem diminuir o desempenho do classificador.

Nesse espectro, surge a necessidade de modelos que se adaptam para analisar novas informações que chegam até o modelo. O termo novidade é apresentado na literatura como uma informação discrepante que deve ser analisada, ou seja, uma anomalia, para que esta passe a incorporar o modelo de decisão. O termo que define o processo de detecção e análise para incorporação no modelo é encontrado na literatura como detecção de novidade e será descrito na próxima seção.

2.1.4 Detecção de Novidades

A detecção de novidades, ou *Novelty Detection*, é uma subcategoria de aprendizado de máquina que tem como objetivo identificar padrões ou eventos que diferem do que foi previamente observado (CHANDOLA; BANERJEE; KUMAR, 2009; PIMENTEL et al., 2014). Também é conhecida como detecção de anomalias ou detecção de outliers. Esta técnica é utilizada em diversos cenários, como detecção de fraudes em cartões de crédito, detecção de intrusões em sistemas de segurança, identificação de doenças em exames médicos, entre outros (CHANDOLA; BANERJEE; KUMAR, 2009).

Em um problema de detecção de novidades, é suposto que se tem um conjunto de dados de treinamento que contém apenas exemplos de comportamentos ou eventos normais. A partir desse conjunto de dados, o algoritmo de detecção de novidade constrói um modelo que aprende a representação desses comportamentos normais. Então, quando o algoritmo recebe um novo exemplo, ele calcula a probabilidade do exemplo ser normal ou uma novidade. Se a probabilidade for inferior a um certo limiar, então o exemplo é considerado uma novidade (MARKOU; SINGH, 2003).

Existem duas abordagens principais para detecção de novidades: a abordagem baseada em densidade e a abordagem baseada em distância. Na abordagem baseada em densidade, a ideia é que os dados de novidade geralmente residem em regiões de baixa densidade de dados normais, enquanto que na abordagem baseada em distância, a ideia é que os dados de novidade são geralmente mais distantes dos exemplos de dados normais (CHANDOLA; BANERJEE; KUMAR, 2009; PIMENTEL et al., 2014).

A detecção de novidades é uma técnica útil em muitos cenários, mas também é um desafio, pois é difícil obter conjuntos de dados de treinamento que contenham exemplos de novidade, além disso, muitas vezes as novidades são desconhecidas e inesperadas, o que dificulta a detec-

ção (MARKOU; SINGH, 2003; PIMENTEL et al., 2014).

2.2 Aprendizado de máquina em fluxo de dados

Fluxo de dados (FD) refere-se a dados que fluem continuamente em um sistema. Geralmente são gerados simultaneamente e em alta velocidade por muitas fontes de dados, que podem incluir aplicativos, sensores de IoT, arquivos de *log* e servidores. Esses aplicativos envolvem conjuntos de dados que são grandes demais para caber em memória principal e são normalmente armazenados em um dispositivo de armazenamento secundário (SILVA et al., 2013). Defini-se o FD como uma sequência de exemplos multi-dimensionais infinita $(x_1, x_2 ... x_n ...)$ que flui rápida e continuamente que varia com o tempo $(T_1, T_2 ... T_n ...)$. Cada exemplo é descrito por um vetor n-dimensões (AGGARWAL et al., 2003).

O aprendizado de máquina em FD possui características diferentes em comparação ao aprendizado de máquina clássico. A tabela 2.1 compara as características do aprendizado de máquina clássico com o aprendizado em fluxo de dados (NGUYEN; WOON; NG, 2015).

Características	Aprendizado de máquina clássico	Aprendizado de máquina em FD
Número de passes	Múltiplos	Único
Tempo	Ilimitado	Tempo real
Memória	Ilimitada	Limitada
Resultado	Preciso	Aproximado
Número de conceitos	Único	Múltiplos

Tabela 2.1: Comparações entre aprendizado de máquina tradicional e em fluxo de dados (NGUYEN; WOON; NG, 2015)

Uma das principais diferenças é o número de passes que o algoritmo faz nos dados. Enquanto o aprendizado clássico pode ler os conjuntos de dados repetidamente para melhorar a precisão do modelo, o FD realiza apenas um único passe no conjunto de dados, devido à limitação de tempo e memória.

Outra diferença importante é que o FD lida com o fluxo de dados em tempo real, enquanto o aprendizado de máquina clássico pode levar tempo ilimitado para processar grandes conjuntos de dados. Isso significa que os algoritmos de FD precisam realizar o processamento de dados e a tomada de decisão em tempo real, o que é essencial em aplicações como sistemas hospitalares e previsões no mercado de ações.

A memória também é uma limitação importante no FD, pois ela é limitada em comparação com o aprendizado de máquina clássico. Os algoritmos de FD criam memórias de curto prazo,

chamadas de *buffer*, para armazenar temporariamente um conjunto de dados para processos internos. No entanto, esses dados devem ser descartados posteriormente para dar espaço aos dados recém-chegados, o que significa que o armazenamento estático de dados é impossível no FD.

Além disso, o FD tem o potencial infinito de dados, conhecido como comprimento infinito, o que significa que o número de amostras é desconhecido e pode crescer continuamente. Para lidar com isso, os algoritmos de FD calculam e armazenam apenas um resumo do fluxo de dados, descartando os dados não mais necessários, o que torna resultados aproximados aceitáveis.

Finalmente, a tabela indica que o número de conceitos abordados no aprendizado de máquina clássico é único, enquanto que no FD, múltiplos conceitos podem ser aprendidos simultaneamente a partir do fluxo de dados. Isso ocorre porque os dados chegam em uma sequência interminável, permitindo que o algoritmo aprenda múltiplos conceitos de forma simultânea.

2.2.1 Classificação em FD

A classificação é uma tarefa de aprendizado supervisionado em que o modelo aprende a partir de um conjunto de treinamento com dados rotulados para que o modelo seja capaz de atribuir um rótulo de classe para novos dados. Existem muitas técnicas de classificação em FD, incluindo árvore de decisão, K-NN, entre outros.

2.2.1.1 Árvore de decisão em FD

O algoritmo de árvore de decisão classifica uma população em segmentos semelhantes a ramos que constroem uma árvore invertida, com nós raiz, nós internos e nós folhas (SONG; YING, 2015). O objetivo da árvore de decisão é que cada nó tenha uma condição que será utilizada para analisar os valores de entrada até que chegue a um nó final que classifica determinada entrada em uma classe. Árvores de decisão tradicionais precisam ler os dados do conjunto de treinamento muitas vezes para seleção dos atributos de divisão, o que é inviável no ambiente de FD.

Para resolver esse problema, o limite de Hoeffding (DOMINGOS; HULTEN, 2000) é utilizado para escolher o melhor atributo de divisão dentro de uma quantidade suficiente de dados. Este algoritmo é incremental, satisfazendo a restrição de passagem única. O algoritmo de árvore de Hoeffding verifica a cada exemplo se o atributo de divisão é confiável o suficiente para criar o próximo nó da árvore. Este algoritmo tem alta precisão e funciona bem com grandes conjuntos de dados. No entanto, ele não consegue lidar com mudanças de conceito em FD, pois seria

necessário a recriação de toda a árvore quando surgisse uma mudança de conceito.

Para lidar com esse problema, foi proposto em (HULTEN; SPENCER; DOMINGOS, 2001) uma adaptação da árvore de Hoeffding que mantém estatísticas suficientes em cada nó da árvore para monitorar a validade de suas decisões anteriores. Assim, quando novos exemplos chegam, o algoritmo atualiza continuamente estas estatísticas, removendo assim o efeito de dados desatualizados. Caso uma mudança de conceito apareça, o algoritmo proposto cria ramificações com o novo atributo e remove ramificações antigas que podem resultar em um modelo menos preciso. Este método é conhecido como ADWIN (*Adaptive Windowing*) e é uma das abordagens mais populares para lidar com mudanças de conceito em FD.

2.2.2 Agrupamento em FD

Agrupamento é uma tarefa de aprendizado não supervisionado, onde o modelo tenta agrupar objetos em diferentes conjuntos, chamados de grupos ou *clusters*. Os algoritmos de agrupamento propostos em fluxo de dados buscam satisfazer as restrições de memória limitada, passagem única, processamento em tempo real e mudança de conceito.

Muitos métodos de agrupamento em FD são adaptações de métodos tradicionais, mas com diferentes abordagens computacionais. Por exemplo, o algoritmo *CluStream* (AGGARWAL et al., 2003) aborda este problema com um aprendizado em duas etapas: a etapa online e a etapa offline.Na etapa online, os microgrupos são criados e atualizados. Na etapa offline, é feito o agrupamento que gera macrogrupos. Essa abordagem permite que o usuário acompanhe o surgimento e junção dos grupos. Além disso, a sumarização das estatísticas ajuda a limitar a memória necessária, pois o modelo requer menos armazenamento para manter as informações.

Outro exemplo de método de agrupamento em FD é o *StreamKM++* (ACKERMANN et al., 2012), que é uma adaptação do método *k-means* para streaming de dados. O método usa amostragem aleatória para selecionar um conjunto representativo de pontos iniciais, que são usados para iniciar o processo de agrupamento. Em seguida, o método atualiza dinamicamente esses centróides à medida que novos dados são adicionados. Esse método também atende às restrições de passagem única e processamento em tempo real, além de ter um baixo custo computacional. No entanto, assim como a maioria dos métodos de agrupamento em FD, ele não consegue lidar com mudanças de conceito.

Outros métodos de agrupamento em FD incluem o *StreamCluster* (ARASU et al., 2003), o *Clustering Data Streams* (AMINI et al., 2011), entre outros. Cada método tem suas próprias abordagens e vantagens, e a escolha do método adequado depende do problema específico em

questão e das restrições do ambiente de FD em que ele é aplicado.

2.2.2.1 Micro-grupo

Um micro-grupo, também conhecido como vetor de características do cluster, é uma estrutura fundamental no contexto de agrupamento em fluxo de dados. Este conceito foi proposto inicialmente por (BIRCH, 1996) e serviu de base para a definição de outros formatos em trabalhos posteriores. A principal função do micro-grupo é salvar informações sumarizadas sobre um grupo de dados, uma vez que os próprios dados não são armazenados durante o processo de criação e atualização do modelo em fluxos de dados em tempo real.

O micro-grupo é responsável por armazenar as seguintes características essenciais:

- Número de Instâncias (N): Representa a contagem de instâncias pertencentes ao grupo. É uma métrica importante para entender o tamanho do cluster.

- Vetor de Soma Linear (LS): Consiste em um vetor da mesma dimensão das instâncias pertencentes ao cluster e armazena a soma linear das N instâncias. Isso fornece informações sobre a tendência central do cluster em relação aos atributos das instâncias.

- Vetor de Soma Quadrática (SS): É um vetor da mesma dimensão das instâncias do cluster e armazena a soma quadrática das N instâncias. Essa medida ajuda a capturar informações sobre a dispersão ou variância dos dados dentro do cluster.

Essa abordagem de micro-grupo permite resumir as principais características do cluster em um formato compacto, facilitando o processamento e a atualização contínua do modelo em cenários de fluxo de dados, onde o armazenamento completo de dados individuais pode não ser viável devido a restrições de memória e processamento em tempo real.

Com estas informações armazenadas, as seguintes propriedades podem ser verificadas nos micro-grupos:

- Incremental: caso um ponto x seja adicionado ao cluster A, as estatísticas armazenadas no vetor devem ser atualizadas da seguinte forma:

$$LS_A = LS_A + x \tag{2.12}$$

$$SS_A = SS_A + x^2 \tag{2.13}$$

$$N_A = N_A + 1 \tag{2.14}$$

- União: Quando os conjuntos de micro-grupos A e B são disjuntos, a operação de união

é equivalente à soma de suas partes. Isso permite a fusão dos micro-grupos da seguinte maneira:

$$LS_C = LS_A + LS_B \tag{2.15}$$

$$SS_C = SS_A + SS_B \tag{2.16}$$

$$N_C = N_A + N_B \tag{2.17}$$

Um micro-grupo tem informações suficientes para calcular as normas e medidas básicas para caracterizar um *cluster*:

$$L_1 = \sum_{i=1}^{N} |x_{a_i} - x_{b_i}| \tag{2.18}$$

$$L_2 = \sqrt{\sum_{i=1}^{N} (x_{a_i} - x_{b_i})^2} \tag{2.19}$$

- Centroide: define o centro do grupo:

$$\vec{X}_0 = \frac{LS}{N} \tag{2.20}$$

- Raio:

$$R = \sqrt{\frac{\sum_{i=1}^{N} (\vec{x}_i - \vec{X}_0)^2}{N}} \tag{2.21}$$

Além das características (N, LS, SS), o *CluStream* (AGGARWAL et al., 2003) adiciona dois atributos: CF_1, que representa a soma dos marcadores de tempo que sinalizam quando o dado foi inserido no modelo, e CF_2, que representa a soma quadrática dos marcadores de tempo. Ambas as características foram adicionais para lidar com os aspectos temporais do fluxo de dados.

2.2.2.2 *Fuzzy Micro-Cluster*

Uma estrutura de sumarização baseada no vetor de atributos que permite trabalhar com agrupamento *fuzzy* é o *Fuzzy Micro-Cluster* (FMiC) (LOPES; CAMARGO, 2017). Este vetor é definido como $(\overline{CF2^e}, \overline{CF1^e}, t, M)$, onde t é o *timestamp* do exemplos mais atual. M é a cardinalidade escalar do micro-grupo, definida como a soma das pertinências μ_j de todos os n exemplos e_j que possuem pertinência maior que zero no micro grupo, definido na equação (2.22).

$$M = \sum_{j=1}^{n} \mu_j^m \tag{2.22}$$

$\overline{CF2^e}$ é a soma quadrática dos exemplos ponderada pelos valores de pertinência μ_j :

$$\overline{CF2^e} = \sum_{j=1}^{n} \mu_j e_j^2 \tag{2.23}$$

$\overline{CF1^e}$ é a soma linear dos exemplos ponderada pelos valores de pertinência μ_j onde e_j e μ_j são respectivamente o j-ésimo exemplo e seu valor de pertinência no micro-grupo:

$$\overline{CF1^e} = \sum_{j=1}^{n} \mu_j e_j \tag{2.24}$$

Esta estrutura de sumarização possibilita a mesclagem de micro-grupos *fuzzy*, bem como a identificação e remoção de micro-grupos antigos.

2.2.2.3 *Supervised Fuzzy Micro-Cluster*

O *Supervised Fuzzy Micro-Cluster* (SFMiC) é uma estrutura de sumarização desenvolvida para uso em ambientes de aprendizado supervisionado. Ele se diferencia do *Fuzzy Micro-Cluster* (FMiC) no sentido de que enquanto o FMiC é utilizado para tarefas de agrupamento, onde não há classes predefinidas, o SFMiC é especialmente projetado para tarefas de classificação, onde se tem conhecimento prévio das classes(SILVA et al., 2018).

O vetor de características do SFMiC $(M, \overline{CF1^e}, \overline{SSD^e}, t,$ Rótulo de Classe$)$ é composto pela soma linear das pertinências dos exemplos no micro-grupo (M), a soma linear dos exemplos ponderada pelos valores de pertinência μ_j ($\overline{CF1^e}$), a soma quadrática das distâncias dos exemplos para o protótipo do micro-grupo elevadas a um fator de fuzzificação m ($\overline{SSD^e}$) representado na equação 2.25, o tempo de chegada do exemplo mais atual associado ao SFMiC (t), e o rótulo da classe associada ao micro-grupo.

$$\overline{SSD^e} = \sum_{j=1}^{n} \mu_j^m |e_j - c|^2 \tag{2.25}$$

O SFMiC utiliza os graus de pertinência para identificar o conjunto de micro-grupos que melhor representa o exemplo, permitindo a classificação do exemplo com o rótulo desses micro-grupos. Além disso, a adição do parâmetro de fuzzificação m permite a criação de uma superfície de decisão mais flexível, tornando o modelo mais tolerante a imprecisão.

2.2.2.4 *Supervised Possibilistic Fuzzy Micro-Cluster*

O *Supervised Possibilistic Fuzzy Micro-Cluster (SPFMiC)* é um vetor de atributos que representa um grupo fuzzy, projetado para ser usado em ambientes de aprendizado supervisionado. O SPFMiC foi proposto por (SILVA et al., 2018) e mantém estatísticas necessárias para o cálculo de tipicidade usado em agrupamento possibilistico fuzzy, o que permite que ele seja mais tolerante a imprecisão.

O SPFMiC armazena parâmetros de fuzzificação α e de tipicidade θ, e é responsável por armazenar um conjunto de estatísticas sumarizadas ($M^e, T^e, \overline{CF1^e_\mu}, \overline{CF1^e_T}$, $SSD^e, N, t, class_id$) de um grupo. Essas estatísticas são calculadas a partir de um conjunto de exemplos $e_1, ..., e_N$, cada um com um grau de pertinência $u_1, ..., u_N$.

O número de exemplos pertencentes ao grupo é representado por N, e t representa o tempo de chegada do exemplo mais atual ao grupo. $class_id$ representa a classe associada ao grupo. M^e é a soma linear dos graus de pertinência dos exemplos no grupo elevados a α. T^e é a soma linear das tipicidades dos exemplos no grupo elevadas a θ.

$\overline{CF1^e_\mu}$ é a soma linear dos exemplos pertencentes ao grupo ponderados por suas pertinências, e $\overline{CF1^e_T}$ é a soma linear dos exemplos ponderados por suas tipicidades. Finalmente, SSD^e é a soma das distâncias dos exemplos para o protótipo do grupo, elevadas a α e ponderadas pelas pertinências de cada exemplo. Com essas estatísticas, o SPFMiC é capaz de manter a tipicidade dos exemplos em relação ao grupo.

2.3 Detecção de novidade em FD

Algoritmos que lidam com detecção de novidade em FD, lidam com a premissa de que a distribuição dos dados varia de acordo com o tempo. Essa variação implica no surgimento de dois tipos de mudanças: mudança de conceito e evolução de conceito.

2.3.1 Mudança de conceito e evolução de conceito

A mudança de conceito é um fenômeno que ocorre quando a relação entre os valores de entrada e a classe-alvo sofrem alteração durante o tempo, criando cenários onde o modelo gerado pelo treinamento não obtém boas respostas quando colocado em produção. Existem diferentes padrões de mudança, conforme definidos por (GAMA, 2010), que são:

- Mudança Abrupta: Ocorre quando um conceito é abruptamente substituído por outro.

Nesse padrão, o conceito antigo desaparece completamente, enquanto o novo conceito surge repentinamente, sem qualquer sobreposição com o conceito anterior.

- Mudança Incremental: Consiste em muitos conceitos intermediários suavemente se alterando, até que a mudança se torne iminente. Nesse padrão, o conceito antigo e o novo coexistem por um certo período de tempo, até que o novo conceito se torne predominante.
- Mudança Gradual: Surge um novo conceito que se assemelha ao antigo e, com o passar do tempo, o antigo não tem mais relevância se comparado ao número de ocorrências, fazendo assim com que o novo tome seu lugar. Nesse padrão, o conceito antigo se dissolve gradualmente em um novo conceito, sem que haja uma mudança abrupta.
- Mudança Recorrente: Um conceito antigo volta a surgir após certo tempo. Nesse padrão, o conceito antigo se repete em um momento posterior, após ter desaparecido por um período de tempo.

A identificação do tipo de mudança é fundamental para a proposição de um algoritmo que consiga distinguir entre uma mudança de conceito e um *outlier*. É importante destacar que a mudança de conceito é diferente da evolução de conceito.

A evolução de conceito se caracteriza pelo surgimento de uma nova classe que não foi identificada no conjunto de treinamento e sua adição ao modelo. Com a intenção de endereçar essas mudanças, diferentes abordagens foram propostas na literatura utilizando aprendizado supervisionado e semi-supervisionado (FARIA et al., 2016b) e aprendizado não supervisionado (HEIGL et al., 2021). Para ter um entendimento mais geral da estrutura dos algoritmos, uma taxonomia em duas etapas foi proposta (FARIA et al., 2016a).

2.3.2 Taxonomia DN em FD

A taxonomia proposta em (FARIA et al., 2016a) descreve como lidar com a tarefa de Detecção de Novidade (DN) em Fluxos de Dados (FD) em duas etapas: *offline* e *online*. Na fase *offline*, um modelo de decisão é induzido a partir de um conjunto de dados rotulados. São definidos três aspectos: o número de classes, o número de classificadores e o tipo de aprendizado. O número de classes pode ser uma única classe (normal) ou multi-classe, com técnicas de classificação específicas para rotular novos exemplos. O número de classificadores pode ser um único classificador, que é atualizado incrementalmente, ou um conjunto de classificadores (*ensemble*), que rotula os dados. Quanto ao tipo de aprendizado, alguns algoritmos não utilizam rótulos no

treinamento e supõem que todos os exemplos pertencem à classe normal, enquanto outros usam uma abordagem supervisionada para classificar diferentes exemplos das classes conhecidas.

Na fase *online*, novos exemplos são classificados em tempo real. Três aspectos são definidos nesta etapa: classificação de novos exemplos, detecção de padrões de novidade e atualização do modelo de decisão. A classificação é realizada imediatamente em alguns algoritmos, enquanto outros armazenam novos exemplos em uma memória secundária para análise posterior. Exemplos não rotulados são armazenados temporariamente para ajudar na identificação de novos padrões. A escolha entre um único classificador e um conjunto de classificadores impacta na atualização do modelo de decisão, que pode ser feita incrementalmente ou substituindo um classificador antigo por um novo. O *feedback* externo é importante para decidir como atualizar o modelo, com alguns algoritmos considerando que os rótulos verdadeiros estarão disponíveis após um período de tempo, enquanto outros não usam feedback externo.

2.3.3 Latência de rótulos

A latência refere-se ao tempo entre a classificação de uma instância e a disponibilidade do rótulo verdadeiro correspondente (SOUZA; PINHO; BATISTA, 2018). Dependendo do ambiente em que o modelo está sendo usado, a latência pode variar em intensidade, o que influencia diretamente na atualização do modelo de decisão. Abaixo estão os tipos de latência:

- Latência Nula: Nesse ambiente, o rótulo verdadeiro está disponível logo após a classificação do modelo, ou seja, não há atraso entre a classificação e a disponibilidade do rótulo. Isso permite que o modelo seja atualizado de forma contínua, já que o rótulo correto sempre estará disponível. Esse é um cenário ideal, mas raramente encontrado na prática.

- Latência Intermediária: Nesse ambiente, o rótulo verdadeiro está disponível após determinado tempo depois da classificação do modelo. Esse tempo pode variar e é influenciado por fatores externos, como a velocidade de transmissão de dados em uma rede. Essa latência intermediária pode ser gerenciada usando estratégias como o armazenamento em *cache* dos rótulos verdadeiros ou a implementação de algoritmos que considerem a latência e a incerteza nas predições.

- Latência Extrema: Nesse ambiente, o rótulo verdadeiro nunca estará disponível após a classificação do modelo. Isso pode ocorrer, por exemplo, quando o modelo é usado para detectar fraudes financeiras e o fraudador nunca admite sua ação. Esse cenário é o mais desafiador para o FD, pois o modelo não pode ser atualizado de forma incremental com

base nos rótulos verdadeiros. Nesses casos, as abordagens baseadas em aprendizagem não supervisionada ou semi-supervisionada podem ser mais adequadas.

É importante que os modelos de aprendizagem de máquina levem em consideração o tipo de latência presente no ambiente em que serão aplicados, para garantir a eficiência na atualização do modelo e, consequentemente, sua capacidade de lidar com mudanças de conceito (GAMA et al., 2014).

Na próxima seção, será realizada a revisão bibliográfica, onde serão apresentados estudos e trabalhos que abordam a detecção de novidades em fluxo de dados. A revisão bibliográfica tem como objetivo identificar os principais métodos e técnicas utilizadas para a detecção de novidades em FD, permitindo uma melhor compreensão das abordagens existentes e destacando as contribuições desses trabalhos para o tema em questão.

Capítulo 3

REVISÃO BIBLIOGRÁFICA

Este capítulo tem como objetivo abordar os principais pontos presentes na literatura que foram obtidos através de um mapeamento sistemático. O mapeamento sistemático tem como objetivo identificar, sumarizar, avaliar e interpretar os estudos publicados na literatura. Existem etapas processuais que auxiliam neste mapeamento e todas as etapas serão descritas neste capítulo. Após apresentação destas etapas e quais foram os resultados, serão discutidos e analisados os trabalhos selecionados.

3.1 Mapeamento Sistemático

O objetivo deste mapeamento foi fazer um levantamento sobre os algoritmos utilizados na literatura para lidar com a detecção de novidades em fluxo de dados e as metodologias/métricas para análise de desempenho desses algoritmos. Para alcançar esse objetivo, foi definido um protocolo que incluiu as seguintes etapas:

- Questões de pesquisa: Definição das principais questões que orientaram a busca pelos estudos relevantes. As questões utilizadas neste trabalho foram:

  Quais são os principais algoritmos que lidam com a detecção de novidades em fluxo de dados?

  Quais são as principais características desses algoritmos, como latência, por exemplo?

  Quais são as principais métricas utilizadas para avaliação de desempenho dos algoritmos?

  Quais foram as vantagens apresentadas pelas diferentes implementações?

- Estratégias de busca: Definição das estratégias de busca que orientaram a pesquisa pelos estudos relevantes. Foram utilizadas a busca inicial por meio de uma string de busca otimizada, a definição dos principais artigos e a técnica de bola de neve. A técnica de bola de neve consiste em utilizar não somente os artigos retornados pela string de busca, mas também as referências bibliográficas que esses artigos utilizaram e analisar os artigos que utilizaram esses artigos como referência. Essa técnica aumenta a possibilidade de encontrar estudos relevantes sem a necessidade de construir uma string de busca muito abrangente.

- Fontes de pesquisa: Definição das fontes de pesquisa que foram utilizadas para a busca pelos estudos relevantes. Foram utilizadas as bases de dados MDPI, Springer, Wiley Online Library, National Library of Medicine (NLM), Bio Med Central (BMC), Hindawi, Plos Org, Research Gate e Wires, além do portal Capes.

- String de Busca: Definição da string de busca otimizada para recuperar os estudos relevantes. A string de busca utilizada, assim como o número de resultados obtidos, estão apresentados na tabela 3.1. Como critérios de busca, foram recuperados somente artigos dos últimos 10 anos e em inglês.

Os termos utilizados na string de busca foram escolhidos com base na leitura de vários artigos. Identificou-se que os termos "Novel Detection", "Novelty Detection"e "Anomaly Detection"são equivalentes em inglês para detecção de novidade. Além disso, foram identificados os termos "Evolving concept", "Concept Evolution"e "Concept Drift"em relação à mudança e evolução de conceitos. Como o foco do trabalho são algoritmos de agrupamento, o termo "Clustering"também foi incluído. Devido à importância do tratamento de ruídos e outliers na detecção de novidades, esses termos também foram adicionados à busca. Após a otimização da string de busca, a última string da tabela foi escolhida e resultou em 29 artigos retornados.

A revisão dos artigos foi realizada considerando dois tipos de critérios: critérios de seleção e critérios de qualidade.

Os critérios de seleção foram definidos para garantir que os artigos selecionados apresentem soluções para tratamento de ruídos e *outliers*, lidem com evolução de conceito utilizando a abordagem em duas etapas (offline e online) e apresentem resultados em conjuntos de dados reais ou sintéticos. Além disso, foram excluídos trabalhos que não abordam a detecção de novidade e algoritmos que lidam com latência nula.

Já os critérios de qualidade foram baseados na leitura dos artigos selecionados. Foram considerados artigos que apresentam uma definição clara e sólida em relação à forma de lidar

String de busca	Resultados obtidos
(Novel OR Novelty) AND (Class OR Classes) AND Detection AND (datastream OR datastreams) AND Time Constraints	300
(Novel OR Novelty OR Anomaly) AND (Class OR Classes) AND Detection AND (datastream OR datastreams)	896
(Novel OR Novelty OR Anomaly) AND (Class OR Classes) AND Detection AND (datastream OR datastreams) AND unsupervised	266
(Novel OR Novelty OR Anomaly) AND (Class OR Classes) AND Detection AND (datastream OR datastreams) AND unsupervised AND Concept Evolution	80
(Novel OR Novelty OR Anomaly) AND (Class OR Classes) AND Detection AND (datastream OR datastreams) AND Unsupervised AND (Concept Evolution OR Concept Drift)	124
(Novel Detection OR Novelty Detection OR Anomaly Detection) AND (Class OR Classes) AND (Datastream OR Datastreams) AND Unsupervised AND (Concept Evolution OR Concept Drift) AND Algorithm	119
(Novel Detection OR Novelty Detection OR Anomaly Detection) AND (Class OR Classes) AND (Datastream OR Datastreams) AND Unsupervised AND (Concept Evolution OR Concept Drift) AND Algorithm AND Clustering	98
(Novel Detection OR Novelty Detection OR Anomaly Detection) AND (Class OR Classes) AND (Datastream OR Datastreams) AND (Concept Evolution OR Concept Drift) AND Algorithm AND Clustering	190
(Novel Detection OR Novelty Detection OR Anomaly Detection) AND (Class OR Classes) AND (Datastream OR Datastreams) AND (Evolving concept OR Concept Evolution OR Concept Drift) AND Algorithm AND Clustering AND (Noise or outlier)	84
(Novel Detection OR Novelty Detection OR Anomaly Detection) AND (Class OR Classes) AND (Datastream OR Datastreams) AND (Evolving concept OR Concept Evolution OR Concept Drift) AND Algorithm AND Clustering AND (Noise or outlier) AND (Offline Phase or Online Phase)	29

Tabela 3.1: Strings de busca utilizadas para busca no Portal Capes

com ruídos e *outliers*, possuem conceitos relacionados à detecção de novidade, principalmente a evolução de conceito, e realizam vasta experimentação utilizando conjuntos de dados reais ou sintéticos.

Após a revisão, foram discutidos os algoritmos encontrados na literatura para latência intermediária e latência extrema. Além disso, foi dada uma atenção especial às abordagens que utilizam conjuntos *fuzzy*.

Título do Estudo	**Autores (Ano)**	**Latência**	**Abordagem "Fuzzy"**
Streaming Autoencoder (SA)	Dong (2018) (DONG; JAPKOWICZ, 2018)	Intermediária	Não
Enhanced Classifier for Data Streams with Novel Class Miner (ECSMiner)	Masud et al. (2010) (MASUD et al., 2010)	Intermediária	Não
Multiclass learning algorithm for DS Active Learning (MINAS-AL)	de Souza et al. (2016) (FARIA et al., 2016b)	Intermediária	Não
Grid-based Clustering for Concept-drifting Data Streams (GC3)	Sethi et al. (2016) (SETHI; KANTARDZIC; HU, 2016)	Intermediária	Não
WiSARD-based Change Detection System (WCDS)	Cardoso et al. (2017) (CARDOSO; FRANÇA; GAMA, 2017)	Extrema	Não
MINAS (Multi-Instance Learning Algorithm from Noisy, Unlabeled Data Streams)	de Souza et al. (2016) (FARIA et al., 2016b)	Extrema	Não
SENNE (Stream-based Evolving Nearest Neighbor)	Cai et al. (2019) (CAI et al., 2019)	Extrema	Não
CluStream	Aggarwal et al. (2003) (AGGARWAL et al., 2003)	Extrema	Não
FuzzND	da Silva et al. (2018) (SILVA et al., 2018)	Extrema	Sim
PFuzzND	da Silva et al. (2018) (SILVA; CAMARGO, 2020)	Extrema	Sim
eClass	Angelov et al. (2008) (ANGELOV; ZHOU, 2008)	Intermediária	Sim
EFuzzCND	Cristiani et al. (2021) (CRISTIANI; CAMARGO, 2021)	Intermediária	Sim

Tabela 3.2: Estudos Selecionados para Detecção de Novidades em Fluxos de Dados

Na próxima seção, descreveremos os trabalhos selecionados de acordo com a seguinte clas-

sificação: latência intermediária, latência extrema e abordagem "fuzzy". Isso facilitará a compreensão das características e foco de cada trabalho selecionado.

3.2 Latência Intermediária

O *Streaming Autoencoder* (SA) é um algoritmo de detecção de novidades que usa uma abordagem baseada em redes neurais(DONG; JAPKOWICZ, 2018). Ele é projetado para detectar anomalias em fluxos de dados (*streaming*) com foco em baixa latência e alta precisão de detecção. O SA é composto por vários *autoencoders*, que são redes neurais especializadas em compressão e reconstrução de dados, cada um treinado em uma janela de tempo específica dos dados do fluxo.

A ideia por trás do SA é que o conjunto de dados de treinamento consiste apenas de amostras da classe normal, sem a necessidade de amostras de anomalias. Isso é possível porque o SA utiliza uma abordagem de detecção binária, em que os *autoencoders* são treinados para reconstruir apenas a classe normal. Como resultado, qualquer amostra que não possa ser reconstruída adequadamente pelo *autoencoder* é considerada uma novidade.

Para lidar com a mudança de conceito, o SA usa uma abordagem baseada em janelas deslizantes, em que os *autoencoders* são treinados em janelas de dados recentes e, em seguida, a janela é deslocada no tempo para incluir novos dados. Além disso, o SA usa um mecanismo de detecção de duas etapas para reduzir o número de falsos positivos. A primeira etapa é baseada na saída de reconstrução do *autoencoder* e a segunda etapa é baseada em um limite de decisão estatístico.

Em resumo, o SA é um algoritmo baseado em redes neurais que se concentra em detectar anomalias em fluxos de dados com alta precisão e baixa latência, usando um conjunto de *autoencoders* treinados em janelas deslizantes de dados normais e um mecanismo de detecção de duas etapas para reduzir os falsos positivos.

O *Enhanced Classifier for Data Streams with Novel Class Miner* (ECSMiner) é um modelo de detecção de novidades (DN) em fluxos de dados (FD) que utiliza uma abordagem baseada em ensemble (MASUD et al., 2010). Ele é projetado para lidar com latência intermediária, ou seja, o atraso entre a classificação de uma instância e a disponibilidade do rótulo verdadeiro, que pode ser influenciado por fatores externos, como a velocidade de transmissão de dados em uma rede. O ECSMiner utiliza o conceito de micro grupo para agrupar instâncias semelhantes em subconjuntos menores de dados.

O ECSMiner possui um componente denominado Novel Class Miner (NCM), que é responsável por detectar novas classes (ou anomalias) no fluxo de dados. O NCM é composto por um conjunto de modelos de classificação binária, cada um treinado em um conjunto de instâncias representativas de uma determinada classe. Quando uma nova classe é detectada pelo NCM, um novo modelo é treinado para detectar a nova classe.

O ECSMiner considera restrições de tempo para classificação de novas instâncias, o que significa que ele busca reduzir a latência e garantir que as instâncias sejam classificadas dentro de um determinado período de tempo. Para isso, ele utiliza uma abordagem de classificação híbrida, que combina o uso de modelos de classificação binária e de regras de classificação para maximizar a precisão de detecção de novidades.

O algoritmo *Multiclass learning algorithm for DS Active Learning* MINAS-AL é uma extensão do algoritmo MINAS, que é um algoritmo de detecção de novidades (DN) em fluxos de dados (FD)(FARIA et al., 2016b). O MINAS-AL é projetado para aproveitar dados rotulados provenientes de *feedback* externo para melhorar a precisão de detecção de novidades.

O MINAS-AL é composto por duas fases: a fase offline e a fase online, que são semelhantes às fases correspondentes do MINAS. Durante a fase offline, o MINAS-AL cria micro grupos de instâncias semelhantes e constrói um modelo para cada micro grupo. Durante a fase online, o MINAS-AL classifica cada nova instância como pertencente a um micro grupo conhecido ou a uma nova classe.

A diferença entre o MINAS-AL e o MINAS está na forma como o modelo é atualizado. No MINAS-AL, os centroides dos micro grupos detectados em um intervalo de tempo t são selecionados como exemplos para serem rotulados. Após a obtenção do rótulo verdadeiro, o modelo é atualizado com o rótulo e os exemplos são removidos do conjunto de instâncias não rotuladas.

O MINAS-AL é capaz de melhorar a precisão de detecção de novidades em fluxos de dados com a ajuda de *feedback* externo. No entanto, ele ainda apresenta algumas limitações, como a dependência de dados rotulados externos, o que pode não estar sempre disponível, e a possibilidade de interferência do rótulo incorreto no processo de atualização do modelo.

O *framework* GC3 é um modelo de aprendizado incremental projetado para classificar dados que apresentam mudanças de conceito (SETHI; KANTARDZIC; HU, 2016). Ele utiliza a técnica de agrupamento baseado em densidade de grade para encontrar regiões densas no espaço dos dados e manter um conjunto de modelos localizados para cada uma dessas regiões. Essa abordagem permite que o GC3 seja altamente adaptável às mudanças nos dados de entrada, pois ele

pode atualizar seus modelos de maneira seletiva em regiões específicas do espaço.

O GC3 é uma estrutura de aprendizado semi-supervisionada, o que significa que ele pode aprender a partir de dados que são parcialmente rotulados. Isso é particularmente útil em um ambiente de FD, onde a rotulagem é uma tarefa custosa e pode não estar disponível em tempo hábil. A capacidade do GC3 de aprender com dados parcialmente rotulados é alcançada por meio de uma técnica chamada aprendizado ativo. Nessa técnica, o modelo seleciona os exemplos de treinamento mais informativos a serem rotulados por um especialista e usa esses rótulos para atualizar seus modelos.

O GC3 é composto por dois estágios principais: o estágio *offline* e o estágio *online*. Durante o estágio *offline*, o modelo é treinado com um conjunto de dados rotulados iniciais e um algoritmo de agrupamento baseado em densidade de grade é aplicado para dividir o espaço dos dados em regiões densas. Em seguida, um conjunto de modelos é criado, cada um responsável por uma região específica do espaço. Durante o estágio *online*, novas instâncias são classificadas de acordo com a região a que pertencem e o modelo correspondente é utilizado para classificar a instância.

O GC3 é capaz de se adaptar às mudanças nos dados de entrada, pois pode detectar automaticamente mudanças de conceito. Quando uma mudança é detectada, o modelo é atualizado seletivamente apenas nas regiões afetadas pela mudança, mantendo a precisão do modelo em outras regiões do espaço dos dados.

3.3 Latência Extrema

O algoritmo WCDS é uma técnica de detecção de novidades que utiliza um modelo de rede neural artificial chamado WiSARD, que é conhecido por ter uma capacidade de aprendizado incremental (CARDOSO; FRANÇA; GAMA, 2017). Em outras palavras, o modelo pode ser treinado continuamente com novos dados, sem a necessidade de reprocessar todo o conjunto de dados de treinamento. Isso torna o WCDS adequado para lidar com fluxos de dados.

O WCDS utiliza uma abordagem multi-classe para o problema de detecção de novidades, o que significa que o algoritmo é capaz de detectar várias classes de anomalias em um fluxo de dados. Diferentemente do SA, o WCDS lida tanto com a mudança de conceito quanto com a evolução de conceito. A mudança de conceito ocorre quando a distribuição dos dados muda ao longo do tempo, enquanto a evolução de conceito ocorre quando novas classes de anomalias aparecem no fluxo de dados.

Para lidar com a mudança de conceito e a evolução de conceito, o WCDS utiliza um mecanismo para descartar dados expirados e um agrupamento offline com base em uma medida de semelhança de pares. O descarte de dados expirados é importante para garantir que o modelo não fique sobrecarregado com dados antigos que já não são relevantes. O agrupamento offline é utilizado para detectar novas classes de anomalias e agrupá-las em um conjunto de classes conhecidas.

O algoritmo MINAS (FARIA et al., 2016b) é um método de detecção de novidades em fluxos de dados que utiliza uma abordagem baseada em micro grupos para modelar cada classe presente nos dados. O algoritmo consiste em duas fases: fase *offline* e fase *online*.

Na fase de treinamento inicial, o algoritmo utiliza os dados de treinamento para criar micro grupos, que são conjuntos de instâncias de uma mesma classe que possuem características similares. O MINAS utiliza a técnica de agrupamento baseado em densidade para identificar esses micro grupos, e cada micro grupo é modelado por um conjunto de centroides que representam as instâncias do grupo.

O modelo de decisão inicial é a união dos conjuntos dos micro-grupos criados para cada classe. Esse modelo é utilizado para classificar as instâncias do fluxo de dados durante a fase *online*. Quando uma nova instância é apresentada ao modelo durante a fase *online*, ela é comparada com os centroides dos micro grupos existentes. Se a instância não pertencer a nenhum dos micro grupos existentes, ela é considerada como uma possível novidade e um novo micro grupo é criado para modelar a nova classe.

O MINAS é capaz de detectar novidades em fluxos de dados em tempo real, e tem a capacidade de lidar com um grande número de classes e de instâncias. Além disso, o MINAS pode ser utilizado em ambientes de aprendizado incremental, em que novos dados podem ser adicionados ao modelo ao longo do tempo.

O algoritmo SENNE é um método para detecção de novas classes emergentes em um sistema de detecção de mudanças de conceito (DS) (CAI et al., 2019). O algoritmo utiliza um conjunto de classificadores por classe, baseado em instâncias, para identificar novas classes. Cada instância no conjunto de treinamento é representada por uma hiperesfera, com raio definido pela distância para o vizinho mais próximo na mesma classe. Quando uma nova instância chega, o algoritmo verifica sua distância para o vizinho mais próximo em cada classe. Se a distância for maior do que um limite definido pelo usuário, a instância é considerada uma novidade e armazenada em um *buffer* de novidade. Caso contrário, a instância recebe o rótulo da sua classe mais próxima. Quando o *buffer* de novidade atinge seu tamanho máximo, as instâncias armazenadas são usadas para criar um conjunto de hiperesferas que representarão a nova classe.

O algoritmo é capaz de lidar com dados não-rotulados e com a presença de ruídos, sendo uma solução promissora para detecção de novas classes em sistemas de DS.

O CluStream é um algoritmo de aprendizado de máquina em fluxo de dados que foi proposto para lidar com dados de fluxo que podem evoluir com o tempo (AGGARWAL et al., 2003). O algoritmo trabalha em duas etapas principais: offline e online. Na fase offline, o algoritmo cria micro-grupos, que são grupos de dados que compartilham características semelhantes e armazenam estatísticas resumidas, como média e desvio padrão, de cada grupo. A fase offline é executada periodicamente, com o objetivo de atualizar o modelo com os novos dados que chegaram desde a última execução da fase offline.

Na fase online, o algoritmo agrupa os dados que chegam em tempo real em macro-grupos, que são grupos de micro-grupos. Os macro-grupos são criados usando as estatísticas resumidas armazenadas nos micro-grupos. O algoritmo utiliza a distância euclidiana para medir a semelhança entre as estatísticas dos micro-grupos e agrupá-los em macro-grupos. Isso ajuda a limitar o armazenamento de informações, pois apenas as estatísticas dos micro-grupos são armazenadas em vez de todos os dados.

O CluStream é escalável e apresenta bons resultados quando há evolução de dados, uma vez que ele é capaz de se adaptar às mudanças nas estatísticas dos grupos à medida que novos dados chegam. Além disso, a sumarização das estatísticas auxilia na limitação da memória, tornando o algoritmo prático para lidar com grandes volumes de dados.

3.4 Abordagens *Fuzzy*

Esta seção apresenta métodos desenvolvidos com base na teoria dos conjuntos *fuzzy* para a tarefa de detecção de novidades.

3.4.1 Latência Extrema

O algoritmo Fuzzy Multiclass Novelty Detector for Data Streams (FuzzND) utiliza abordagem *fuzzy* para a detecção de novidades em fluxos de dados. O método foi desenvolvido para lidar com a tarefa de detecção de novidades multi-classe em ambientes de fluxo de dados(SILVA et al., 2018). O processo de aprendizagem é dividido em duas fases: offline e online. Na fase offline, um modelo de decisão é criado a partir de exemplos rotulados. Durante a fase online, novos exemplos são classificados de forma incremental em uma das classes conhecidas do modelo ou como desconhecidos. Os exemplos desconhecidos são agrupados em intervalos

regulares para identificar padrões novidade. O algoritmo utiliza a estrutura SFMiC para calcular a similaridade *fuzzy* entre grupos.

O SFMiC é um modelo de grupo que utiliza estatísticas para calcular a similaridade entre grupos. Ele é composto por um conjunto de micro-grupos *fuzzy*, cada um representando uma classe. O modelo utiliza a medida de similaridade *fuzzy* para comparar os micro-grupos com os exemplos desconhecidos, a fim de determinar se um exemplo pertence a uma classe conhecida ou se é uma novidade. Se um exemplo for classificado como uma novidade, ele é agrupado com outros exemplos desconhecidos em um grupo *fuzzy* que representa um padrão novidade. Esses padrões novidade são adicionados ao modelo como novas classes.

O FuzzND é uma generalização *fuzzy* do método MINAS, que utiliza a teoria dos conjuntos *fuzzy* para lidar com a incerteza na tarefa de detecção de novidades em fluxos de dados. Ele é capaz de lidar com a detecção de novidades em ambientes de fluxo de dados com múltiplas classes e é adequado para ambientes onde o número de novidades é desconhecido. Além disso, o FuzzND apresenta bom desempenho na detecção de novidades em relação a outras abordagens *fuzzy* e não *fuzzy* em diferentes conjuntos de dados.

O algoritmo Possibilistic Fuzzy multiclass Novelty Detector for data streams (PFuzzND) proposto por (SILVA; CAMARGO, 2020) é uma extensão do FuzzND, que utiliza a teoria dos conjuntos *fuzzy* para a tarefa de detecção de novidades em fluxos de dados. A principal diferença entre o FuzzND e o PFuzzND é que este último usa a tipicidade para gerar uma melhor descrição dos tipos de exemplos durante a etapa de classificação. O algoritmo PFuzzND utiliza a estrutura SPFMiC, que inclui a tipicidade como uma medida adicional para calcular a similaridade *fuzzy* entre os grupos. O valor de tipicidade é calculado como a distância média entre um novo exemplo e todos os exemplos associados a um micro-grupo.

Ao contrário das pertinências, que são baseadas apenas nas distâncias do novo exemplo para os centroides dos micro-grupos existentes no modelo, os valores de tipicidade gerados pelo PFuzzND podem melhor definir a área de atuação de um micro-grupo. Além disso, esses valores de tipicidade não são fragmentados com o aumento de micro-grupos no modelo, o que pode melhorar a capacidade de generalização do modelo em casos de fluxos de dados com mudanças frequentes de conceito.

O PFuzzND é dividido em duas fases: a fase offline e a fase online. Durante a fase offline, um modelo de decisão é criado a partir de uma porção de exemplos previamente rotulados. Durante a fase online, novos exemplos não rotulados são classificados incrementalmente em uma das classes conhecidas do modelo ou como desconhecidos. Intermitentemente, os exemplos desconhecidos são agrupados para procurar grupos coesos que serão incorporados no modelo

de decisão como padrões novidade. Em resumo, o PFuzzND é uma extensão do FuzzND que utiliza a tipicidade para melhorar a capacidade de generalização do modelo em casos de fluxos de dados com mudanças frequentes de conceito.

3.4.2 Latência Intermediária

O algoritmo *eClass* utiliza uma abordagem baseada em regras *fuzzy* do tipo Takagi-Sugeno para classificação e detecção de novidades em fluxos de dados(ANGELOV; ZHOU, 2008). O método é capaz de aprender continuamente com a chegada de novas instâncias, e pode começar a aprender do zero sem a necessidade de pré-especificação de regras *fuzzy*. O algoritmo opera em duas fases: a fase offline, onde um conjunto de regras *fuzzy* é criado a partir de um conjunto de exemplos rotulados, e a fase online, onde novas instâncias são classificadas usando o conjunto de regras criado na fase offline.

Durante a fase offline, o algoritmo utiliza o algoritmo *GFS-Growing* para construir um conjunto de regras *fuzzy* a partir dos exemplos rotulados. O algoritmo GFS-Growing utiliza um conjunto de seletores para determinar quais regras devem ser incluídas no conjunto de regras. Em seguida, o algoritmo utiliza o algoritmo *GFS-Pruning* para remover as regras redundantes e otimizar o conjunto de regras.

Durante a fase online, o algoritmo recebe novas instâncias e as classifica usando o conjunto de regras criado na fase offline. Se uma nova instância não puder ser classificada por nenhuma regra no conjunto existente, ela é considerada uma novidade. As novidades são armazenadas em um buffer e, periodicamente, são utilizadas para atualizar o conjunto de regras existente.

O algoritmo *eClass* é capaz de lidar com a latência intermediária para obtenção de rótulos verdadeiros, pois depende da chegada de instâncias com rótulos verdadeiros para realizar a identificação de novidades. Além disso, o método é capaz de lidar com o surgimento de novos conceitos, pois é capaz de começar a aprender do zero sem a necessidade de pré-especificação de regras *fuzzy*. No entanto, o método depende do algoritmo *GFS-Growing* para criar o conjunto de regras *fuzzy*, o que pode ser um desafio em problemas com grandes conjuntos de dados.

O algoritmo EFuzzCND é um método para detectar novidades em fluxos de dados multiclasse(CRISTIANI; CAMARGO, 2021). Ele usa dois modelos de decisão, um supervisionado e outro não supervisionado. O modelo supervisionado consiste em micro-grupos *fuzzy* que são obtidos a partir de dados rotulados e é usado para classificar instâncias pertencentes a conceitos conhecidos. O modelo não supervisionado é composto por micro-grupos *fuzzy* obtidos pelo método de detecção de novidades a partir dos dados não rotulados e é usado para classificar

instâncias marcadas como desconhecidas pelo modelo supervisionado.

Durante a fase online, o modelo é atualizado assumindo que todos os rótulos estarão disponíveis após determinado tempo. Quando os rótulos verdadeiros surgem no modelo, eles são armazenados em uma memória temporária e usados para atualizar o modelo supervisionado. Os exemplos que não são classificados pelo modelo supervisionado são considerados novidades e o modelo não supervisionado começa a ser usado. Esse modelo é formado por micro-grupos *fuzzy* cujos rótulos representam diferentes padrões de novidade e são criados ao longo da fase online.

Quando a memória temporária está cheia, as instâncias rotuladas são agrupadas, gerando novos micro-grupos *fuzzy*. Esses novos micro-grupos são analisados para encontrar grupos representativos que correspondem a um novo padrão de novidade ou uma extensão de um padrão já conhecido. Em ambos os casos, esses novos micro-grupos são adicionados ao modelo não supervisionado e usados para classificar novas instâncias não rotuladas que foram marcadas como desconhecidas.

O algoritmo remove micro-grupos *fuzzy* que não classificaram nenhum exemplo em determinado tempo e busca por semelhanças entre os micro-grupos gerados pelos dados rotulados e os micro-grupos do modelo não supervisionado. Se um micro-grupo gerado pelos dados rotulados for semelhante a um micro-grupo do modelo não supervisionado, este micro-grupo é removido do modelo não supervisionado e todos os novos micro-grupos são incluídos no modelo supervisionado.

Neste capítulo, foram apresentados os principais tópicos relacionados ao tema deste trabalho, incluindo um mapeamento sistemático dos estudos sobre detecção de novidade em fluxos de dados, a discussão sobre as latências intermediária e extrema, bem como as abordagens fuzzy utilizadas em trabalhos relacionados à detecção de novidade.

A partir dessa revisão bibliográfica, é possível notar a importância da utilização de algoritmos capazes de lidar com a detecção de novidades em fluxos de dados em tempo real. Dessa forma, na próxima seção, serão apresentados os detalhes do algoritmo *Enhanced Incremental Fuzzy Classifier for Multi-Class Novelty Detection in Data Stream* (EIFuzzCND) proposto neste trabalho, que é capaz de detectar novidades em diferentes padrões de novidade em fluxos de dados.

A seção seguinte apresentará o funcionamento do algoritmo, mostrando como ele lida com a detecção de novidades em fluxos de dados e as principais técnicas utilizadas para essa tarefa.

Capítulo 4

ENHANCED INCREMENTAL FUZZY CLASSIFIER FOR MULTI-CLASS NOVELTY DETECTION IN DATA STREAM

A detecção de novidade em fluxos de dados (DN em FD) é um desafio significativo no campo de aprendizado de máquina, especialmente quando se trata de classificação em múltiplas classes. Várias abordagens têm sido propostas na literatura para abordar esse problema, variando desde o uso de um único classificador até conjuntos de classificadores, levando em consideração diferentes níveis de latência, como extrema, intermediária ou nula, e incorporando o uso de conjuntos fuzzy para lidar com imprecisões nos dados.

No entanto, muitas dessas abordagens apresentam limitações. Por exemplo, algumas não são capazes de se adaptar rapidamente a mudanças de conceito em cenários de latência extrema. Outras dependem fortemente do método de detecção de novidade para identificar mudanças de conceito em modelos de decisão estáticos. Além disso, em cenários de latência intermediária, muitas abordagens requerem uma grande quantidade de dados rotulados, o que pode ser impraticável em algumas situações.

Recentemente um modelo chamado EFuzzCND (Evolving Fuzzy Clustering for Novelty Detection) foi proposto(CRISTIANI; CAMARGO, 2021). Este modelo aborda o problema de detecção de novidade em fluxos de dados de forma abrangente. O EFuzzCND é capaz de realizar classificação multiclasse e detecção de novidade e possui versões tanto para latência extrema quanto para latência intermediária, o que o torna flexível o suficiente para se adaptar a diferentes cenários.

O EFuzzCND assume que, em algum momento após a classificação, todos os rótulos estarão disponíveis, o que permite a identificação de novas classes à medida que esses rótulos

são fornecidos. No entanto, essa abordagem não abrange cenários de latência intermediária, nos quais nem todos os rótulos estarão disponíveis imediatamente após a classificação. Além disso, no cenário de latência extrema, o modelo inicial, obtido na fase offline, nunca é atualizado. Isso pode prejudicar significativamente o desempenho do modelo, pois ele não se adapta às mudanças que ocorrem no fluxo de dados ao longo do tempo. Mesmo no cenário de latência intermediária, o modelo é atualizado somente quando um determinado número de rótulos verdadeiros é observado no fluxo de dados, o que pode resultar em uma deterioração do desempenho, já que ele não consegue lidar eficazmente com mudanças de conceito que ocorrem antes desse ponto de atualização.

Essas limitações destacam a necessidade de melhorias no EFuzzCND, a fim de torná-lo mais adequado para cenários de latência intermediária e extrema, permitindo uma adaptação mais eficiente às mudanças de conceito e acomodando situações em que a disponibilidade de rótulos é limitada ou a atualização do modelo é necessária de forma mais dinâmica.

Nesse contexto, formulamos a hipótese de que a atualização incremental dos modelos de decisão pode tornar a manutenção mais simples e eficaz, permitindo que o modelo se ajuste mais rapidamente a mudanças de conceito. Além disso, a atualização incremental pode ser especialmente útil em cenários de latência extrema, onde a falta de rótulos verdadeiros torna impossível a atualização do modelo com base nesses rótulos. Em cenários de latência intermediária, a consideração de apenas uma quantidade limitada de dados rotulados pode ser uma alternativa viável para se aproximar de um cenário real.

Com base nesta hipótese, propomos o desenvolvimento do Enhanced Incremental Fuzzy Classifier for Multi-Class Novelty Detection in Data Stream (EIFuzzCND), um algoritmo baseado no EFuzzCND. O EIFuzzCND utiliza a atualização incremental do modelo de decisão e leva em consideração uma quantidade limitada de dados rotulados em cenários de latência intermediária. A próxima seção apresenta uma visão geral do algoritmo proposto.

4.1 Visão Geral do Algoritmo

O algoritmo EIFuzzCND tem como objetivo a classificação multiclasse e detecção de novidades em fluxos de dados por meio da abordagem composta por duas etapas, offline e online. Para isso, ele emprega dois modelos: o Modelo de Classes Conhecidas (MCC) e o Modelo de Classes Desconhecidas (MCD).

Na etapa offline, o algoritmo utiliza um conjunto de treinamento rotulado, dividindo seus exemplos por classe e aplicando o algoritmo Fuzzy C-Means (FCM) para criar clusters repre-

sentativos de cada classe. Esses clusters são sumarizados em micro-grupos, que servem para representar as classes , como mostrado na Figura 4.1. Importante notar que, embora o objetivo final seja a classificação, o algoritmo faz uso de técnicas de agrupamento para a criação dos micro-grupos. Isso possibilita ao modelo a detecção de novidades em fluxos de dados, mesmo quando as classes são dinâmicas.

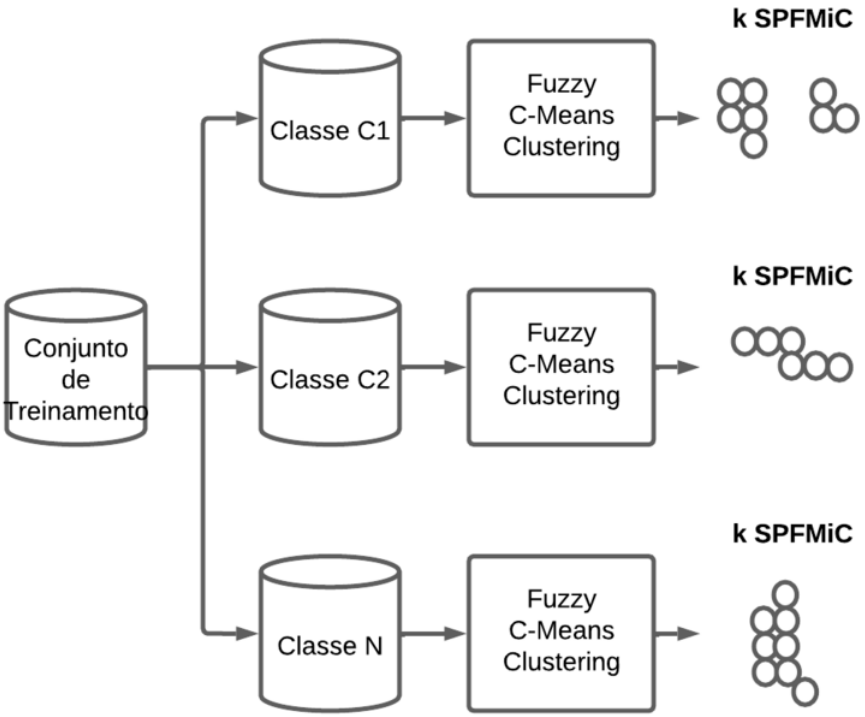


Figura 4.1: Fase Offline - Algoritmo do modelo formado por EIFuzzCND

O MCC é construído na fase offline e é composto por micro-grupos que representam as classes já conhecidas. Por outro lado, o MCD é criado e atualizado durante a fase online. O MCD é formado por micro-grupos que representam as classes identificadas como novidades, ou seja, aquelas que ainda não possuem rótulos conhecidos. Além disso, o MCD também pode incluir exemplos que não puderam ser classificados pelo MCC, mas que apresentam semelhanças com as classes conhecidas.

Na fase online, como mostrado na Figura 4.2 o algoritmo recebe exemplos do fluxo de dados individualmente. No cenário de latência intermediária se um exemplo estiver rotulado, seus dados correspondentes são temporariamente armazenados. Quando a memória temporária acumula um certo número de exemplos, o algoritmo utiliza esses dados para criar novos micro-grupos, seguindo o mesmo processo utilizado na fase offline. Esses novos micro-grupos são incorporados ao MCC. Isso permite ao algoritmo adaptar-se a mudanças no fluxo de dados e

identificar novas classes à medida que surgem.

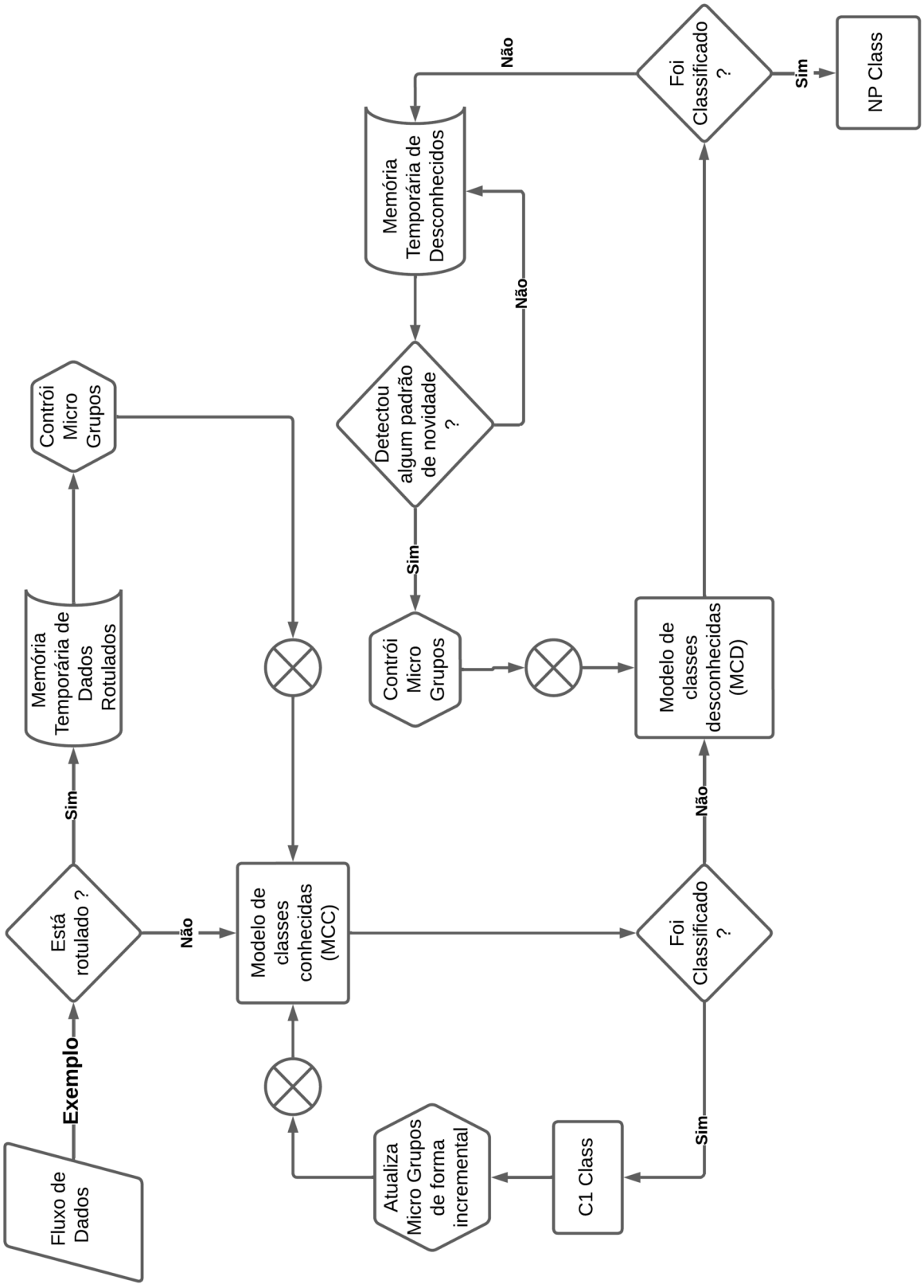


Figura 4.2: Fase Online - Algoritmo EIFuzzND

Caso o exemplo não esteja rotulado, tanto no cenário de latência extrema quanto no cenário de latência intermediária o MCC tenta classifica-los. Se bem-sucedido, o exemplo recebe um rótulo e é incorporado incrementalmente ao modelo, atualizando o micro-grupo ao qual pertence. No entanto, se o MCC não conseguir classificar o exemplo, o MCD assume essa tarefa para verificar se o exemplo pertence a uma das classes novas já identificadas, mas para as quais ainda não se conhece o rótulo verdadeiro. Se o MCD também não conseguir classificá-lo, o exemplo é armazenado temporariamente em uma memória de exemplos desconhecidos.

Quando a memória temporária de exemplos desconhecidos atinge um certo número, o algoritmo tenta identificar padrões de novidade entre esses exemplos. Padrões de novidade são grupos de dados que tenham uma quantidade mínima de exemplos e uma similaridade mínima para se supor que são dados de uma nova classe, desconhecida até aquele momento. Se padrões de novidade são identificados, o algoritmo atribui a esses padrões rótulos fictícios, criando novas classes. Essas novas classes são representadas por micro-grupos e são incorporadas ao MCD. Essas classes são temporárias e fictícias, pois os rótulos reais ainda não são conhecidos. Exemplos não identificados como pertencentes a classes existentes ou padrões de novidade permanecem na memória temporária de desconhecidos, com um registro de tempo de chegada.

A identificação de padrões de novidade ocorre da seguinte forma. Quando um novo exemplo desconhecido é adicionado à memória temporária, o algoritmo o submete a um processo de agrupamento (clustering) usando o algoritmo Fuzzy C-Means (FCM), resultando em vários clusters. Cada cluster passa por um processo de validação para determinar se é um candidato válido a representar um padrão de novidade. Para atribuir um rótulo a um cluster válido, o algoritmo calcula a similaridade fuzzy entre o cluster e os padrões já conhecidos. Se a similaridade atinge um limiar definido pelo usuário, o cluster é rotulado com o mesmo rótulo do padrão mais similar. Caso contrário, um novo rótulo é gerado, representando uma nova classe desconhecida. Essa abordagem permite que o algoritmo identifique e crie novas classes para padrões de novidade no fluxo de dados.

Em resumo, o algoritmo EIFuzzCND utiliza uma abordagem híbrida, combinando técnicas de classificação e agrupamento para detectar novidades em fluxos de dados de forma incremental e em tempo real, permitindo a adaptação a mudanças e a criação de novas classes quando padrões de novidade são identificados. As etapas offline e online e seus respectivos algoritmos serão descritos com maiores detalhes nas seções seguintes.

4.2 Fase Offline

Na fase offline, o algoritmo recebe como entrada um conjunto de treinamento rotulado (trainSet), um valor para o grau de fuzzificação (fuzzification) e um número de clusters (K) que são utilizados no algoritmo Fuzzy C-Means. Armazena também um valor mínimo de peso (minWeight) para representar a quantidade mínima de exemplos que um micro-grupo precisa para ser considerado uma classe, que na fase offline é considerado sempre zero. Os valores de α e θ são utilizados na soma linear dos graus de pertinência e tipicidade dos exemplos no grupo. A saída é o MCC, que utilizamos para classificar novos exemplos como mostra o Algoritmo 1.

Algorithm 1 Algoritmo da Fase Offline

Require: $trainSet$, fuzzification, K, minWeight = 0, α, θ

Ensure: MCC - Initial State

1: $examplesByClass \leftarrow$ separateByClasses($trainSet$)
2: **for** each class **do**
3: $clusters \leftarrow$ FuzzyC-Means($examplesByClass$, K, fuzzification)
4: $SPFMiCs \leftarrow$ SummaryClusters($examplesByClass$, $clusters$, class, α, θ, $minWeight$)
5: $MCC \leftarrow MCC \cup SPFMiCs$
6: **end for**
7: **return** MCC

O algoritmo começa separando os exemplos por classes. Separar os exemplos por classe antes de treinar um algoritmo de agrupamento é importante porque isso permite que o algoritmo identifique padrões mais específicos para cada classe.

Para cada classe, o algoritmo executa o *Fuzzy C-Means*, fornecendo o grau de *fuzzification* e o valor de K como parâmetros, resultando em um conjunto de clusters. Em seguida, são criados os SPFMiCs (*Supervised Possibilistic Fuzzy Micro-Cluster*) para resumir as informações contidas nos clusters. Durante esse processo de resumo, para cada exemplo, o algoritmo atualiza os seguintes valores nos respectivos SPFMiCs:

1. M^e - Representa a soma linear das pertinências dos exemplos no grupo elevadas a α.
2. T^e - Representa a soma linear das tipicidades dos exemplos no grupo elevadas a θ.
3. $\overline{CF1^e_\mu}$ - Representa a soma linear dos exemplos pertencentes ao grupo ponderados por suas pertinências.

4. $\overline{CF1_T^e}$ - Representa a soma linear dos exemplos pertencentes ao grupo ponderados por suas tipicidade.

5. SSD^e - É a soma dos quadrados das distâncias entre os exemplos e o centro do cluster.

6. $class_id$ - Indica a classe à qual o SPFMiC pertence.

Essas variáveis são essenciais para o processo de atualização incremental dos SPFMiCs e desempenham um papel fundamental na representação e na manutenção dos micro-grupos e das informações relacionadas a eles. Ao final, o algoritmo retorna uma lista contendo o MCC em seu estado inicial e assim o algoritmo avança para a fase online.

4.3 Fase Online

Em resumo, a fase online consiste em utilizar o MCC e o MCD para classificar as instâncias de dados e detectar novidades, ou seja, instâncias que não pertencem a nenhuma das classes conhecidas.

O Algoritmo 2 recebe como entrada um fluxo de dados (dataStream), o MCC obtido através da fase offline, o valor de latência (L), o percentual de rótulos verdadeiros que vão retornar ao fluxo (*percentLabeled*), o tamanho da memória temporária de dados desconhecidos (T), o tamanho da memória temporária de dados rotulados (tChunk) e um limite superior para o índice de similaridade definido pelo usuário (*phi*) que é utilizado para o método de detecção de novidade para definir se o micro-grupo é referente a uma classe existente ou a um padrão de novidade.

Durante a fase online, o algoritmo percorre um conjunto em um laço de repetição simulando um fluxo de dados. O MCC é utilizado para classificar cada exemplo. Caso o exemplo seja classificado (Linha 4) , o micro-grupo correspondente ao índice de tipicidade máxima é selecionado para rotular o exemplo e atualizar o modelo de forma incremental. O método de atualização incremental e quais informações são atualizadas serão explicados com mais detalhes na próxima seção.

Caso o rótulo retornado seja -1 (Linha 5), indicando que a instância é desconhecida, o MCD é utilizado para determinar o rótulo da instância (Linha 6). O MCD representa as classes identificadas como novidades, ou seja, aquelas que não possuem rótulos conhecidos. Se a instância for classificada pelo MCD, o rótulo associado a essa classificação é atribuído à instância e a mesma é descartada. Por outro lado, se após a classificação pelo MCD o rótulo ainda for -1, a instância é adicionada a uma memória de instâncias desconhecidas (Linhas 7 e 8).

Algorithm 2 Algoritmo da Fase Online

Require: *dataStream*, *MCC*, *L*, *tChunk*, T, *phi*, *percentLabeled*
Ensure: *LabeledData*

```
1:  unknownInstances ← empty list
2:  labeledInstances ← empty list
3:  for each instance inst in dataStream do
4:      label ← MCC.predict(inst)
5:      if label = −1 then
6:          label ← MCD.predict(inst)
7:          if label = −1 then
8:              add inst to unknownInstances
9:              if size of unknownInstances ≥ T then
10:                 unknownInstances ← multiClassNoveltyDetection(unknownInstances)
11:             end if
12:         end if
13:     end if
14:     if currentTime ≥ latencyDelay then
15:         if Math.random() < percentLabeled OR labeledMem is empty then
16:             labeledMem ← addToList(labeledExample)
17:         end if
18:         if size(labeledMem) ≥ tChunk then
19:             MCC ← trainModel(labeledMem, tempo)
20:             clear(labeledMem)
21:         end if
22:     end if
23: end for
24: return testSetLabeled
```

Quando o tamanho dessa memória atinge um limite T, uma função de detecção de novidades é utilizada para identificar quais instâncias são novidades. O método multiClassNoveltyDetection realiza um agrupamento Fuzzy C-means na lista de exemplos desconhecidos, obtendo uma lista de centroides e coeficientes de pertinência. Em seguida, os exemplos são separados por cluster, e o Modelo de Classes Conhecidas (MCC) atribui rótulos para cada cluster (Linhas 9 e 10).

O MCC é responsável por atribuir rótulos aos clusters, mas o motivo pelo qual o MCC não atribuiu rótulos a esses exemplos anteriormente pode estar relacionado a sua capacidade de generalização, que pode ser limitada em relação a padrões muito diferentes do que foi visto durante o treinamento. Se o MCC for capaz de atribuir rótulos ao cluster, o rótulo conhecido mais próximo é atribuído ao cluster. Por outro lado, se a distãncia entre o centro de um cluster novo e o centro dos clusters rotulados do MCC estiver acima de um limiar (ϕ), o cluster é rotulado como uma novidade.

Rotular como novidade significa atribuir a classe a um novo rótulo que representa uma classe desconhecida, que ainda não foi reconhecida pelo modelo. Essas novas classes são temporárias e fictícias, permitindo que o algoritmo continue a classificar e acompanhar essas classes à medida que novos exemplos pertencentes a elas aparecem no fluxo de dados. Os exemplos rotulados como novidade são adicionados ao MCD e, ao mesmo tempo, são removidos da lista de exemplos desconhecidos, uma vez que agora têm um rótulo associado.

Além disso, o algoritmo mantém uma memória de exemplos rotulados, que é atualizada com uma certa probabilidade após os rótulos verdadeiros começarem a surgir no fluxo. Quando a memória de exemplos rotulados atinge um limite, o modelo supervisionado é treinado com esses exemplos e a memória é limpa para dar espaço a novos exemplos rotulados.

O código apresentado entre as linhas 14 e 22 ocorre apenas no cenário de latência intermediária.

Inicialmente verifica-se se o tempo atual (currentTime) é maior ou igual ao atraso de latência (latencyDelay). Caso verdadeiro, isso indica que já se passou tempo suficiente para que o rótulo verdadeiro comece a surgir no fluxo de dados. Quando isso acontecer, os exemplos rotulados serão armazenados na memória temporária de dados rotulados.(Linhas 14 a 17) . A porcentagem(percentLabeled) indica quantos rótulos estão disponíveis.

Nas linhas 18 a 21 se o tamanho da memória de exemplos rotulados (labeledMem) atingir ou ultrapassar um certo limite (tChunk), esses dados serão utilizados para definir novos SPFMiCs que serão adicionados ao MCC . Com base nesses exemplos rotulados até o momento. Após o

treinamento, a memória de exemplos rotulados é limpa para liberar espaço para novos exemplos rotulados.

4.3.1 Atualização incremental

A atualização incremental acontece somente no MCC. O método MCC.predict() presente na Linha 4 do algoritmo 2 calcula a distância euclidiana entre o exemplo e o objeto SPFMiC. Se a distância for menor ou igual ao dispersão fuzzy do SPFMiC , a tipicidade e pertinência são calculadas e adicionadas em listas auxiliares, juntamente com o próprio SPFMiC.

Se nenhum SPFMiC for selecionado a função retorna -1. Caso contrário, a função encontra o valor máximo da tipicidade e da pertinência e obtém os índices correspondentes nas respectivas listas. Em seguida, a função seleciona o SPFMiC correspondente à maior tipicidade, atualiza o SPFMiC com o exemplo e retorna o rótulo do SPFMiC selecionado.

A dispersão fuzz(DF) do SPFMiC é uma medida que reflete a dispersão dos dados dentro de um micro-grupo e é usado para determinar a tipicidade e pertinência de um exemplo a uma classe conhecida. Ele é calculado usando a seguinte fórmula:

$$\mathrm{DF} = \sqrt{\frac{\mathrm{SSD}}{\mathrm{N}}} \times \text{fator de escala} \tag{4.1}$$

Onde:

- SSD^e: Representa a soma dos quadrados das distâncias entre os pontos de dados no micro-grupo e seu centroide.

- N: Número de pontos de dados no micro-grupo.

- Fator de escala: É usado para ajustar a sensibilidade da medida de DF.

Por exemplo, quando o fator de escala é maior, o DF é maior, o que significa que os exemplos precisam estar mais próximos do centroide para serem considerados como pertencentes à classe representada pelo micro-grupo. Isso torna a classificação mais restrita e pode aumentar o grau de confiança na atribuição de rótulos.

No EIFuzzCND, onde há um MCC e um MCD, o uso de diferentes fatores de escala pode refletir diferentes níveis de confiança nas classificações desses dois modelos. O MCC (Modelo de Classes Conhecidas) deve ter um fator de escala maior, tornando sua classificação mais restrita e, portanto, mais confiável. Por outro lado, o MCD pode ter um fator de escala menor, tornando sua classificação menos restrita e, portanto, menos confiável.

Algorithm 3 Algoritmo da Atualização Incremental

Require: MCC, exemplo, classes
for each SPFMiC in MCC **do**
  $distancia \leftarrow$ calculaDistanciaEuclidiana(SPFMiC.centroide, exemplo)
  if distancia <= raio com peso do objeto SPFMiC **then**
    $tipicidade \leftarrow$ calculaTipicidade(objetoSPFMiC.centroide, exemplo)
    $pertinencia \leftarrow$ calculaPertinencia(objetoSPFMiC.centroide, exemplo)
    $listaObjetosSelecionados$.append(objetoSPFMiC)
  end if
end for
if $listaObjetosSelecionados$ is empty **then**
  return -1
else
  $ObjetoSelecionado \leftarrow$ seleciona(listaObjetosSelecionados, tipicidade)
  $SPFMiCAtualizado \leftarrow$ atualizaSPFMiC(objetoSelecionado, exemplo, Pertinencia, Tipicidade)
  return *MCC*
end if

A atualização do SPFMiC recebe o exemplo, sua pertinência e tipicidade em relação ao cluster e, em seguida, atualiza as variáveis de cálculo do centroide do cluster, incluindo a distância euclidiana do exemplo para o centroide, o número de exemplos no cluster, os somatórios de pertinência e tipicidade, e a soma dos quadrados das distâncias ponderadas dos exemplos ao centroide conforme apresentado no Algoritmo 4.

Algorithm 4 Método atualizaSPFMiC

Input: $exemplo$, $pertinencia$, $tipicidade$
$dist \leftarrow$ calculaDistanciaEuclidiana(exemplo, centroide)
$N \leftarrow N + 1$
$Me \leftarrow Me + pertinencia^{\alpha}$
$Te \leftarrow Te + tipicidade^{\theta}$
$SSDe \leftarrow SSDe + dist^2 \times pertinencia$
$atualizaCF1pertinencia(exemplo(i), pertinencia)$
$atualizaCF1tipicidades(exemplo(i), tipicidades)$
atualizaCentroide()

Finalmente, atualiza o centroide do SPFMiC com base nas pertinências e tipicidades acumuladas dos exemplos atribuídos a ele. Ele percorre cada atributo do centroide e calcula uma média ponderada dos valores de pertinência e tipicidade para aquele atributo. Essa média ponderada é dividida pela soma ponderada dos valores de pertinência e tipicidade para todos os atributos, a fim de normalizar os valores e obter a média ponderada final para cada atributo do centroide.

Esse processo é útil para lidar com novos dados e ajustar o modelo para melhorar a precisão

Algorithm 5 Método atualizaCentroide

1: $nAtributos \leftarrow$ length of CF1pertinencias
2: $centroide \leftarrow$ array of size $nAtributos$
3: **for** $i \leftarrow 0$ **to** $nAtributos - 1$ **do**
4: $\quad centroide[i] \leftarrow \frac{\alpha \times CF1pertinencias[i] + \theta \times CF1tipicidades[i]}{\alpha \times Te + \theta \times Me}$
5: **end for**

das previsões. A atualização incremental permite que o modelo se adapte gradualmente a novos dados e melhore continuamente sua precisão.

Na próxima seção deste trabalho, serão apresentadas as especificações do experimento realizado para avaliar o desempenho do EIFuzzCND, bem como as métricas utilizadas para avaliar sua eficácia.

Capítulo 5

EXPERIMENTOS E MÉTRICAS DE AVALIAÇÃO

Nesta seção, serão apresentados detalhes sobre os datasets utilizados nos experimentos, bem como os parâmetros adotados tanto para o algoritmo proposto (EIFuzzCND) quanto para os algoritmos apresentados na literatura usados nas comparações. Finalmente, serão descritas as métricas de avaliação utilizadas para analisar os resultados obtidos.

5.1 Datasets

O algoritmo proposto para classificação e detecção de novidades foi avaliado em diferentes cenários de teste usando cinco datasets distintos: MOA3 (FARIA et al., 2016b), RBF (SILVA; CAMARGO, 2020), SynEDC (MASUD et al., 2010), KDD99 (CUP, 1999) e CoverType (UCI, 1998). Segue abaixo uma breve descrição de cada um:

- **MOA3**: Conjunto de dados gerado utilizando uma função disponível no framework Massive Online Analysis (MOA) (BIFET et al., 2010). Este conjunto de dados é formado por agrupamentos hipersféricos não-estacionários (mudança de conceito). A cada 30.000 instâncias, novas classes aparecem e as antigas desaparecem (evolução de conceito).

- **RBF**: Conjunto de dados gerado usando uma função radial aleatória disponível no framework MOA. Este conjunto de dados mostra apenas o desaparecimento e aparecimento de novas classes (evolução de conceito) a cada 10.000 instâncias.

- **SynEDC**: Conjunto de dados gerado usando distribuição gaussiana disponível na ferramenta MOA, utilizando médias entre -5 e 5 e variância por classe de 0,5 a 5. Este conjunto de dados mostra o aparecimento e desaparecimento de classes (evolução do conceito) e mudanças na distribuição de atributos ao longo do tempo (mudança de conceito).

- **KDD99**: Conjunto de dados contendo dados reais de tráfego em redes de computadores (CUP, 1999). O conjunto de dados original possui 4 milhões de instâncias, 41 atributos e 23 rótulos. Em nossos experimentos, utilizamos uma versão reduzida, com 10% dos dados, contendo 490 mil instâncias. Com o pré-processamento realizado, o conjunto de dados utilizado possui 34 atributos numéricos e um atributo de classe (com 5 valores possíveis).

- **CoverType**: conjunto de dados que contém informações sobre o tipo de cobertura vegetal em uma determinada área geográfica (UCI, 1998). É amplamente utilizado em problemas de classificação, sendo composto por diversas características da vegetação.

A tabela 5.1 apresenta de forma resumida as informações dos datasets utilizados na avaliação dos algoritmos. Cada linha da tabela corresponde a um dataset específico e inclui os seguintes atributos:

- Dataset: O nome do dataset.
- Instâncias: O número total de instâncias no dataset.
- Instâncias (Offline): O número de instâncias usado durante a fase offline do algoritmo.
- Atributos: O número de atributos ou características presentes em cada dataset.
- Classes: O número total de classes ou categorias existentes no dataset durante a fase online.
- Classes (Offline): O número de classes ou categorias existentes no dataset durante a fase offline.
- Sintético: Indica se o dataset é sintético (Sim) ou não (Não).

Dataset	Instâncias	Instâncias(Offline)	Atributos	Classes	Classes(Offline)	Sintético
MOA3	100,000	10,000	4	4	2	Sim
RBF	48,588	2,000	2	5	3	Sim
SynEDC	400,000	30,000	54	20	6	Sim
KDD	490,000	48,791	34	5	4	Não
CoverType	581,000	47,045	54	7	7	Não

Tabela 5.1: Características dos datasets utilizados na avaliação dos algoritmos.

A diferença entre Classes Offline e Classes pode ser explicada da seguinte forma:

- Classes (Offline): Refere-se ao número de classes ou categorias que foram conhecidas e usadas durante a fase offline do algoritmo. Durante a fase offline, o algoritmo trabalha apenas com um subconjunto das classes totais que podem estar presentes na fase online.

- Classes: Representa o número total de classes ou categorias que podem estar presentes no dataset em qualquer momento durante a fase online. Isso significa que, durante a fase online, o algoritmo está preparado para lidar com até esse número de classes, incluindo novas classes que podem surgir como novidades.

5.2 Algoritmos e configurações experimentais

Nesta seção, serão apresentados os algoritmos selecionados para a detecção de novidades nos conjuntos de dados descritos anteriormente. Esses algoritmos serão avaliados e comparados com o algoritmo proposto. As técnicas escolhidas foram o EFuzzCND (CRISTIANI; CAMARGO, 2021) , ECSMiner (MASUD et al., 2010) e MINAS (FARIA et al., 2016b). Além disso, serão apresentadas as configurações experimentais utilizadas, tais como os parâmetros de cada algoritmo.

A tabela 5.2 exibe a configuração dos parâmetros utilizados no EIFuzzCND e EFuzzCND. É importante ressaltar que esses valores foram mantidos iguais em ambos os algoritmos utilizados para garantir uma comparação justa entre o algoritmo proposto e o EFuzzCND. Esta comparação é crucial para a análise dos resultados.

EIFuzzCND e EFuzzCND			
Fase	Parametro	Descrição	Valor
Offline	fuzzification	Grau de fuzzificação	2
	alpha (α)	Variável para cálculo de pertinência	2
	theta (θ)	Variável para cálculo de tipicidade	1
	K	Número de clusters por classe	*
	minWeight	Peso Mínimo para considerar um cluster	*
Online	latencia	Tempo de Latência	*
	tChunk	Tamanho máximo da memória temporária de dados rotulados	2000
	T	Número mínimo de instâncias para executar a DN	*
	kShort	Número de clusters	*
	phi (ϕ)	Limite superior para o índice de similaridade	0.5
	TS	Tempo limite para remover microclusters e intâncias antigas	*
	percentedLabel	Percentual de rótulos verdadeiros retornados ao fluxo	*
	F	Fator de multiplicação do raio do micro-grupo	*

Tabela 5.2: Parâmetros utilizados pelo algoritmo EIFuzzCND e EFuzzCND, separados em fases Offline e Online.

Os valores marcados com '*' na tabela são variáveis e podem mudar dependendo do dataset específico utilizado e do cenário de execução do algoritmo. Estes estão sendo definidos na Tabela 5.3

DataSet	K	k_short	T	F	min_weight
MOA3	4	4	40	2	15
RBF	4	4	40	2	15
SynEDC	8	8	80	4	30
KDD99	8	8	80	1.5	30
CoverType	8	8	80	2	30

Tabela 5.3: Valores utilizados durante os experimentos para os algoritmos EIFuzzCND e EFuzzCND.

Além disso, é importante comparar o algoritmo proposto com o ECSMiner, que utiliza agrupamento clássico para atribuir cada instância a um único grupo, enquanto o agrupamento fuzzy permite que uma instância seja atribuída a mais de um grupo por meio do cálculo de pertinência e/ou tipicidade.

O algoritmo ECSMiner utilizou os valores definidos em (MASUD et al., 2010), com exceção de Tc, que foi definido como 500. A escolha do classificador K-NN foi feita para aproximar a abordagem utilizada pelo ECSMiner ao funcionamento do EIFuzzCND como mostra a Tabela 5.4.

ECSMiner			
Fase	Parametro	Descrição	Valor
Offline	classificador	Algoritmo de agrupamento	K-NN
	K	Número de Clusteres	50
Online	classificador	Algoritmo de agrupamento	K-NN
	Tc	Tempo máximo para classificar uma nova instância	500
	TI	Tempo de latência	*
	num_classificadores	Número máximo de classificadores no ensemble	6
	minPts	Número mínimo de instâncias para executar o ND	50
	chunck_size	Instâncias de treinamento para novo modelo de decisão.	1000
	q	Usado para calcular o NCM na detecção de novidade	50

Tabela 5.4: Valores utilizados durante os experimentos para o algoritmos ECSMiner.

Por fim, o MINAS foi escolhido para comparação, pois é amplamente utilizado na literatura para a detecção de novidades em fluxos de dados. Os valores dos parâmetros utilizados no mé-

todo MINAS estão apresentados na Tabela 5.5 e seguem os valores propostos nos experimentos realizados em (FARIA et al., 2016b).

MINAS			
Fase	Parametro	Descrição	Valor
Offline	algOff	Algoritmo de agrupamento	K-Means
	k_class	Número de grupos por classe	100
Online	algOn	Algoritmo de agrupamento	CluStream
	threshold	Limiar para identificar padrões de novidade	1.1
	thresholdStrategy	Estratégia usada para calcular o limiar	TV1
	T	Número mínimo de instâncias para executar o ND	2000
	P	Limiar de tempo de exclusão de micro-grupo	4000
	ts	Tempo limite para remover intâncias antigas	4000
	window_size	Intervalo em que as remoções ocorrem	4000
	k_short	Número de clusters	100
	numMinExCluster	Número mínimo de instâncias em um cluster válido	*

Tabela 5.5: Valores utilizados durante os experimentos para o algoritmo MINAS.

Com base nos algoritmos escolhidos e nas configurações definidas, é necessário estabelecer quais métricas serão utilizadas para avaliação.

5.3 Métricas de avaliação

A avaliação dos algoritmos de detecção de novidade é uma etapa crucial para verificar a eficácia de cada modelo. Para isso, algumas métricas podem ser utilizadas para medir o desempenho de cada algoritmo. Nesta seção, serão apresentadas as métricas de avaliação escolhidas para este trabalho.

A primeira métrica a ser discutida é a Matriz de Confusão Incremental (MCI), que é uma extensão da matriz de confusão clássica (FARIA et al., 2015). A MCI permite avaliar o desempenho do algoritmo ao longo do tempo, considerando o número de verdadeiros positivos (VP), falsos positivos (FP), verdadeiros negativos (VN) e falsos negativos (FN) que ocorreram durante a detecção de novidade. A MCI neste trabalho é utilizada para calcular a acurácia do algoritmo. A acurácia é uma medida amplamente utilizada para avaliar o desempenho dos algoritmos de detecção de novidade. A acurácia mede a proporção de instâncias classificadas corretamente pelo algoritmo em relação ao total de instâncias como demonstrado na equação 5.1:

$$\text{Acurácia} = \frac{\text{VP} + \text{VN}}{\text{total de instâncias}} \tag{5.1}$$

Outra métrica importante é a Taxa de Desconhecidos (UnkR), proposta por (FARIA et al., 2015), que avalia o número de instâncias não classificadas pelos modelos de decisão, ou seja, marcadas como desconhecidas. Ela é definida pela equação 5.2:

$$UnkR = \frac{1}{C}\left(\sum_{i=1}^{C} \frac{UnkR_i}{Exci}\right) \tag{5.2}$$

Onde C representa o número total de classes, UnkRi representa o número de instâncias pertencentes à classe Ci classificadas como desconhecidas e Exci representa o número total de instâncias pertencentes à classe Ci. Com esta métrica, é possível compreender o comportamento do algoritmo quando uma nova classe aparece ou ocorre uma mudança de conceito.

Portanto, a utilização das métricas MCI, acurácia e UnkR é essencial para avaliar o desempenho dos algoritmos de detecção de novidade. A MCI permite uma análise mais detalhada do desempenho do algoritmo ao longo do tempo, enquanto a acurácia mede a proporção de instâncias classificadas corretamente e a UnkR avalia o número de instâncias não classificadas pelos modelos de decisão. Juntas, essas métricas proporcionam uma avaliação abrangente e precisa do desempenho dos algoritmos de detecção de novidade.

Com os algoritmos selecionados e as configurações definidas, a próxima etapa é a realização dos experimentos e a avaliação dos resultados obtidos. Os resultados serão apresentados na seção seguinte, permitindo uma análise comparativa entre os algoritmos testados e possibilitando a identificação do melhor modelo para a detecção de novidade em fluxos de dados.

Capítulo 6

RESULTADOS E ANÁLISES DOS EXPERIMENTOS

Nesta seção, apresentaremos e discutiremos os resultados obtidos a partir dos experimentos realizados para avaliar o desempenho do algoritmo proposto. Os resultados serão apresentados e discutidos em três seções distintas, seguindo a ordem de latência extrema, latência intermediária e latência intermediária com variação no percentual de presença dos rótulos verdadeiros.

6.1 Latência extrema

Nesta seção, serão apresentados e discutidos os resultados obtidos na avaliação do algoritmo EIFuzzCND em relação aos algoritmos existentes na literatura que foram escolhidos para o cenário de latência extrema, que se refere à situação em que não há presença dos rótulos verdadeiros no fluxo de dados.

Em cenários de latência extrema, abordagens incrementais podem ter melhor desempenho do que abordagens tradicionais de aprendizado de máquina. Isso se deve ao fato de que as abordagens incrementais não necessitam obrigatoriamente da presença do rótulo verdadeiro no fluxo para atualizarem o modelo.

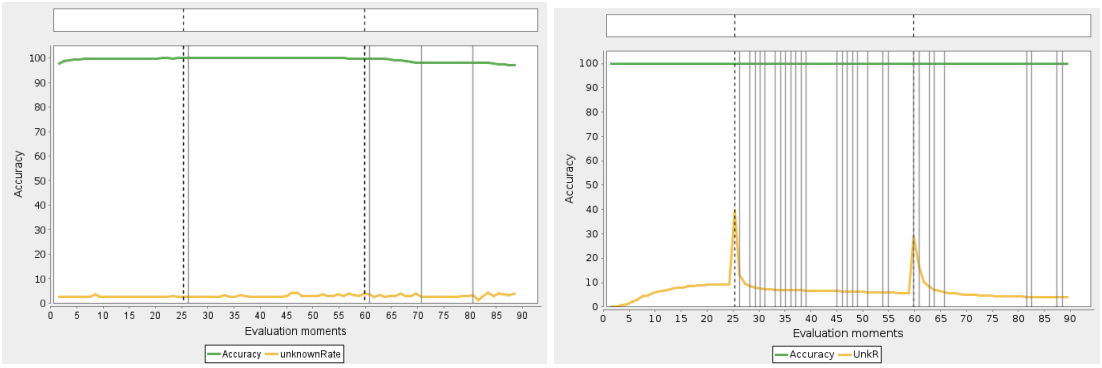
6.1.1 MOA3

A Figura 6.1 apresenta os resultados de acurácia e taxa de desconhecidos (UnkR) em 90 momentos de avaliação do conjunto de dados MOA3. Essas avaliações ocorrem a cada intervalo de 1000 exemplos processados no fluxo de dados.

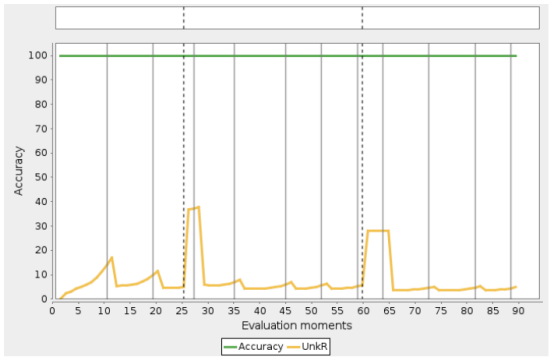
As linhas verticais cinzas e tracejadas indicam os momentos em que uma novidade foi introduzida no fluxo de dados. Isso significa que, nesses momentos, o fluxo de dados apresentou

um exemplo de uma classe que não estava presente na fase offline do modelo, ou seja, é a primeira vez que essa classe aparece. Consequentemente, o modelo de classes conhecidas (MCC) não possui informações sobre essa nova classe e não consegue classificar o exemplo adequadamente, resultando na detecção de uma novidade. Já as linhas verticais cinzas nos gráficos indicam que o modelo identificou um padrão de novidade.

Portanto, esses momentos representam a introdução de padrões de novidade no fluxo de dados, nos quais o modelo de classes conhecidas não possui conhecimento prévio.



(a) EIFuzzCND **(b) EFuzzCND**



(c) MINAS

Figura 6.1: Acurácia e taxa de desconhecidos dos algoritmos, considerando latência extrema, para o conjunto de dados MOA3.

Para melhorar a visualização e obter informações adicionais na análise dos resultados, aplicamos a técnica de redução de dimensionalidade chamada Análise de Componentes Principais (PCA, na sigla em inglês) ao conjunto de dados. O PCA é uma técnica estatística que nos permite reduzir a complexidade de conjuntos de dados de alta dimensionalidade, mantendo as informações mais relevantes.

O conjunto de treinamento do dataset MOA3 usado na fase offline apresenta uma distribuição equilibrada entre suas duas classes como apresenta a Figura 6.2, o que auxilia o modelo a definir com maior precisão os micro-grupos na fase offline. Como resultado, nos primeiros momentos de avaliação, os três algoritmos analisados apresentaram uma alta taxa de acurácia.

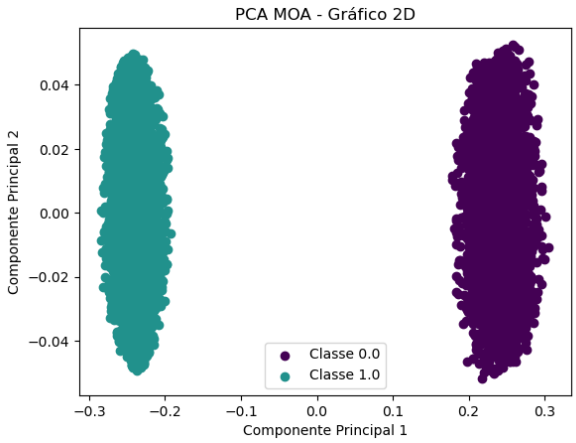


Figura 6.2: Distribuição dos dados no conjunto de treinamento MOA após PCA

No entanto, na fase online, surgem duas novas classes em momentos específicos, nos momentos de avaliação 25 e 60. A Figura 6.1a apresenta que o algoritmo EIFuzzCND mantém uma pequena taxa de desconhecidos durante toda a sua execução, mesmo com o surgimento de novas classes, enquanto o EFuzzCND (Figura 6.1b) e o MINAS (Figura 6.1c) apresentam um crescimento significativo na taxa de desconhecidos com o surgimento de padrões de novidade. Isso ocorre porque abordagens incrementais, como o EIFuzzCND, são mais efetivas em se adaptar ao surgimento de novidades.

As Figuras 6.1a, 6.1b e 6.1c apresentam algumas diferenças notáveis, principalmente em relação à detecção de novos padrões de novidade. Essa detecção está relacionada às linhas verticais cinzas nos gráficos, as quais indicam momentos em que o algoritmo identificou um padrão que não pertencia a nenhuma das classes conhecidas, ou seja, um padrão de novidade (NP).

No caso do algoritmo EFuzzCND, observamos a ocorrência frequente de padrões de novidade ao longo da execução. Isso ocorre porque o EFuzzCND enfrenta desafios na adaptação a mudanças de conceito sem a disponibilidade dos rótulos verdadeiros já que os micro-grupos não são atualizados de forma incremental. Nesse contexto, o alto número de padrões de novidade pode indicar que o EFuzzCND está tentando lidar com as mudanças no fluxo de dados sem informações precisas sobre as novas classes.

O MINAS também apresenta um número considerável de detecções de novos padrões de novidade, semelhante ao EFuzzCND. Isso sugere que o MINAS também enfrenta dificuldades na adaptação a mudanças de conceito sem os rótulos verdadeiros.

A inclusão da Matriz de Confusão Incremental (MCI) junto com o algoritmo EIFuzzCND permitiu uma análise mais abrangente do desempenho do algoritmo. A Tabela 6.1a mostra a

MCI no momento 25.000, destacando o surgimento do primeiro exemplo da classe 2.0. Por outro lado, a Tabela 6.1b representa a MCI no momento 25066, onde é possível observar a detecção de um padrão de novidade ou seja foi criado um padrão novidade no modelo de classes desconhecidas, já que o algoritmo conseguiu identificar que começaram a chegar dados de uma nova classe, apesar deles não estarem rotulados.

Classes	0.0	1.0	UnkMem	2.0
0.0	12565	0	13	0
1.0	0	12410	11	0
UnkMem	0	0	0	0
2.0	0	0	1	0

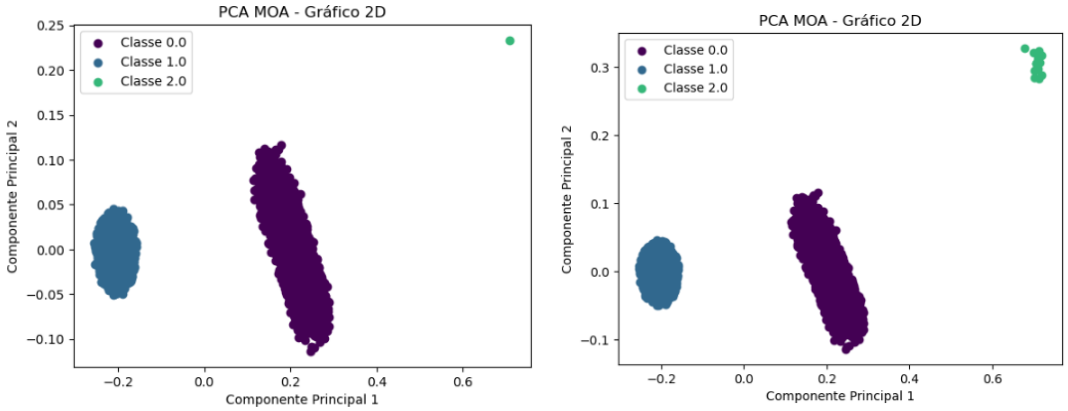
(a) Surgimento do primeiro exemplo da classe 2.0 no momento 25.000

Classes	0.0	1.0	UnkMem	2.0	NP1
0.0	12595	0	13	0	0
1.0	0	12431	11	0	0
UnkMem	0	0	0	0	0
2.0	0	0	0	0	16

(b) Detecção do padrão de novidade no momento 25066

Tabela 6.1: Matriz de confusão incremental (MCI) representando o surgimento e a detecção de um padrão de novidade.

Com base nas Figuras 6.3a e 6.3b, podemos observar a distribuição dos dados no conjunto de testes MOA durante a fase Online nos momentos de avaliação 25.000 e 25.066, respectivamente.



(a) Momento de avaliação 25.000 **(b) Momento de avaliação 25.066**

Figura 6.3: Distribuição dos dados no conjunto de treinamento MOA após PCA nos momentos de avaliação 25.000 e 25.0066

Uma análise dos gráficos revela que a nova classe apresenta um grupo bem distribuído e distinto dos grupos já presentes. Isso significa que as instâncias pertencentes à nova classe formam um conjunto compacto e separado dos demais grupos, o que facilita a detecção de um padrão de novidade. Essa separação clara no espaço de características permite que o algoritmo identifique e classifique adequadamente as novas instâncias pertencentes à nova classe.

O resultado observado, que inclui a detecção frequente de novos padrões de novidade mesmo em uma abordagem fuzzy incremental para o agrupamento, pode ser explicado por algumas características dessa abordagem.

Primeiramente, a abordagem fuzzy permite que uma instância pertença a múltiplos grupos com diferentes graus de pertinência. Isso significa que uma instância pode ser associada a um novo grupo de forma flexível, mesmo que ela também tenha alguma pertinência a grupos já existentes. Em outras palavras, a flexibilidade do modelo fuzzy permite que uma instância seja considerada como parte de uma nova classe emergente, mesmo que compartilhe algumas características com classes já conhecidas.

Isso contrasta com abordagens de agrupamento rígidas, onde uma instância normalmente é atribuída a apenas um grupo, o que pode tornar a inclusão de instâncias em novas classes mais desafiadora. Portanto, a abordagem fuzzy oferece uma maior adaptabilidade para lidar com padrões de novidade, permitindo que instâncias sejam associadas a novas classes emergentes de forma menos restritiva.

Além disso, a abordagem incremental permite que o algoritmo se adapte e aprenda com novas instâncias ao longo do tempo. Dessa forma, à medida que as novas instâncias da nova classe vão chegando, o algoritmo atualiza seus modelos e ajusta os limites de decisão para incorporar as características específicas desse grupo. Isso possibilita que o algoritmo identifique corretamente as instâncias pertencentes à nova classe, mesmo que ela tenha características distintas dos grupos existentes.

No segundo momento de detecção de novidade, no momento 60.000, o algoritmo EI-FuzzCND demonstra um comportamento similar, detectando um novo padrão de novidade e mantendo uma alta taxa de acurácia. Essa capacidade de detecção de novos padrões de novidade reforça a eficácia do algoritmo na identificação de classes não conhecidas anteriormente.

Em momentos específicos, mais precisamente nos momentos 71 e 81, foram identificados padrões de novidades que já existiam anteriormente. Nessas ocasiões, constatou-se uma diminuição na taxa de acurácia, embora de forma sutil. Essa redução pode ser atribuída à classificação inadequada de alguns exemplos, conforme evidenciado pelas tabela 6.2. Esses exemplos

mal classificados podem comprometer a precisão geral do algoritmo, bem como afetar a detecção de novidades.

Classes	0.0	1.0	UnkMem	2.0	3.0	NP1	NP2	NP3	NP4
0.0	26228	96	14	0	0	0	0	2286	15
1.0	110	28414	11	0	0	0	0	0	0
-1.0	0	0	0	0	0	0	0	0	0
2.0	0	0	0	0	0	15914	0	0	0
3.0	0	0	0	0	0	0	6558	0	0

Tabela 6.2: MCI representando o último momento de avaliação no dataset MOA3

6.1.2 RBF

O conjunto de dados RBF consiste em instâncias geradas aleatoriamente em um espaço de alta dimensão, no qual as classes são distribuídas radialmente ao redor de centros de classe gerados aleatoriamente. É um conjunto de dados desafiador para algoritmos de detecção de novidades, uma vez que as classes de novidade podem ser difíceis de distinguir das classes de base. Na Figura 6.4 podemos observar a distribuição dos dados usados na fase offline.

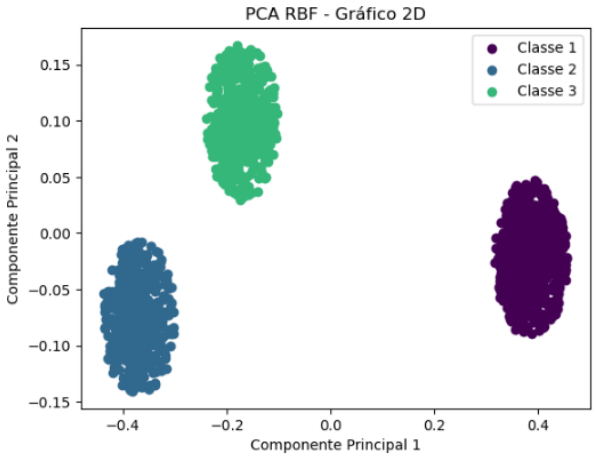
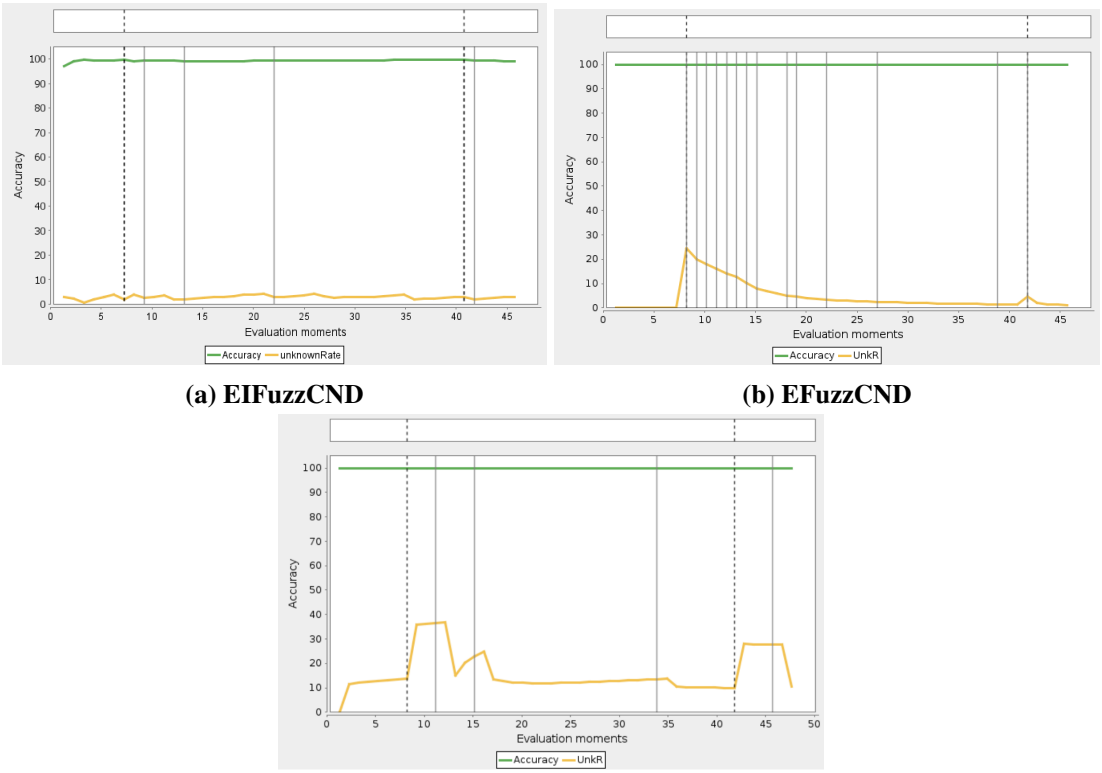


Figura 6.4: Distribuição dos dados no conjunto de treinamento RBF após PCA

A análise do conjunto de treinamento revela a presença de três classes que exibem uma distribuição homogênea, o que facilita o processo de agrupamento. Essa estrutura bem definida do conjunto de treinamento contribui para a obtenção de altas taxas de acurácia nos estágios iniciais de avaliação. Esses resultados promissores podem ser observados nas Figuras 6.5, que ilustram graficamente a performance dos algoritmo.



(a) EIFuzzCND **(b) EFuzzCND**

(c) MINAS

Figura 6.5: Acurácia e taxa de desconhecidos dos algoritmos, considerando latência extrema, para o conjunto de dados RBF.

A Figura 6.5a ilustra o desempenho do algoritmo proposto em um cenário de latência extrema. Como se trata de uma abordagem incremental, o algoritmo apresenta uma menor detecção de padrões de novidade em comparação com o EFuzzCND (Figura 6.5b) e o MINAS (Figura 6.5c).

No conjunto de dados RBF, o MINAS e o EIFuzzCND apresentam um comportamento semelhante em relação ao número de detecções de novidades. O MINAS detecta padrões de novidade nos momentos de avaliação 11, 15, 34 e 46. Por outro lado, o EIFuzzCND detecta padrões nos momentos de avaliação 8, 12, 21 e 42. É importante destacar que as novas classes surgem nos momentos 7 e 41. Esses resultados evidenciam que, além de manter uma baixa taxa de desconhecidos, o algoritmo EIFuzzCND é capaz de detectar padrões de novidade de forma mais rápida.

Essas observações ressaltam a capacidade do algoritmo EIFuzzCND de lidar com a detecção de novidades em cenários de latência extrema. Embora o número de detecções possa ser menor devido à natureza incremental do algoritmo, ele mantém uma taxa de desconhecidos baixa e é eficiente na identificação de novos padrões de novidade. Essa eficiência na detecção precoce de novidades é um fator crucial para a capacidade do algoritmo em lidar com fluxos de

dados em tempo real, onde a velocidade e a precisão são fundamentais.

A Tabela 6.3 apresenta uma análise nos momentos 8 e 12, onde mesmo sem a presença explícita de uma nova classe, o algoritmo EIFuzzCND conseguiu identificar um PN.

Classes	0.0	UnkMem	1.0	2.0	6.0	NP1
0.0	3293	13	0	0	0	0
UnkMem	0	0	0	0	0	0
1.0	0	0	2288	0	0	0
2.0	0	7	0	2371	0	0
6.0	0	0	0	36	0	20

(a) Identificação do padrão de novidade (NP1) referente ao surgimento da classe 6.0

Classes	0.0	UnkMem	1.0	2.0	6.0	NP1	NP2
0.0	4863	13	0	0	0	0	0
UnkMem	0	0	0	0	0	0	0
1.0	0	0	3481	0	0	0	0
2.0	0	7	0	3547	0	0	0
6.0	0	0	0	114	0	438	21

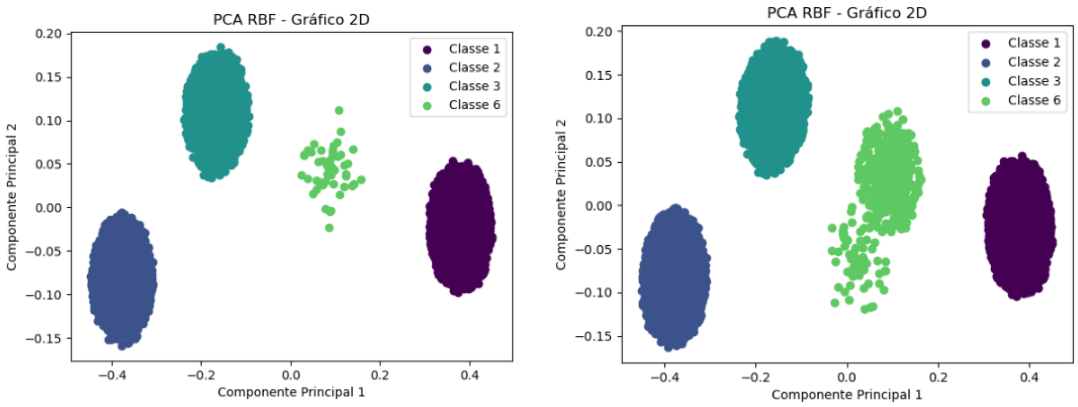
(b) Identificação do padrão de novidade (NP2) no momento em que não há a presença explícita de novas classes

Tabela 6.3: Matriz de confusão incremental (MCI) representando o surgimento e a detecção de um padrão de novidade no dataset RBF.

Na Tabela 6.3a, observamos que no momento 8 ocorre o surgimento da classe "6.0"no fluxo de dados. O algoritmo EIFuzzCND classifica corretamente essa classe como "NP1"(padrão de novidade 1). Isso indica que o algoritmo reconhece essa classe como um padrão de novidade distinto das classes previamente conhecidas.

Por outro lado, na Tabela 6.3b, no momento 12, não há o surgimento de nenhuma nova classe explicitamente. No entanto, o algoritmo EIFuzzCND detecta um padrão de novidade, identificado como "NP2"(padrão de novidade 2). Isso sugere que o algoritmo reconhece um padrão desconhecido ou não correspondente às classes existentes, mesmo na ausência de uma nova classe específica.

Podemos obter uma compreensão mais clara ao analisar os gráficos na Figura 6.6. Nesses gráficos, é possível visualizar a distribuição dos exemplos pertencentes à classe "6.0". Essa representação visual nos permite identificar claramente a presença dessa classe e sua distribuição no conjunto de dados.



(a) Momento de avaliação 8.000 **(b) Momento de avaliação 12.000**

Figura 6.6: Distribuição dos dados no conjunto de treinamento MOA após PCA nos momentos de avaliação 8.000 e 12.000

Ao observar os gráficos, podemos identificar visualmente a existência de exemplos que podem ser agrupados em dois grupos distintos, mesmo pertencendo à mesma classe. Essa observação revela que, apesar de serem da mesma classe, esses exemplos não estão muito próximos uns dos outros em termos de características. Essa falta de proximidade entre os exemplos da mesma classe levou o algoritmo EIFuzzCND a detectar um novo padrão de novidade.

No caso do MINAS, observamos que ele manteve uma taxa de desconhecidos relativamente alta por um período prolongado. Isso sugere que o MINAS pode ser mais conservador na atribuição de exemplos a classes conhecidas, o que pode resultar em uma taxa mais alta de exemplos considerados desconhecidos por um tempo maior.

Por outro lado, o EFuzzCND detectou muitos padrões de novidade devido à sua natureza não incremental, o que significa que ele pode ser mais sensível a mudanças nos dados, especialmente quando não possui informações prévias sobre novos exemplos. Isso pode resultar em uma detecção mais frequente de novos padrões de novidade.

6.1.3 KDD99

O conjunto de dados KDD99 apresenta um desafio significativo para os algoritmos de detecção de novidades devido à sua natureza realista e complexa. Composto por uma variedade de tipos de tráfego de rede, incluindo tráfego normal e atividades maliciosas, como ataques de negação de serviço (DoS), tentativas de acesso não autorizado e varreduras de portas.

O conjunto de dados KDD99 é composto por instâncias que representam uma ampla gama de atividades de rede e são distribuídas em várias classes. Essas classes podem apresentar padrões e comportamentos distintos, o que torna desafiador distinguir as classes de novidade das

classes de base. No entanto, a análise da distribuição dos dados no conjunto de treinamento, conforme ilustrado na Figura 6.7, mostra que a complexidade do conjunto de dados KDD99 persiste mesmo após a aplicação do PCA.

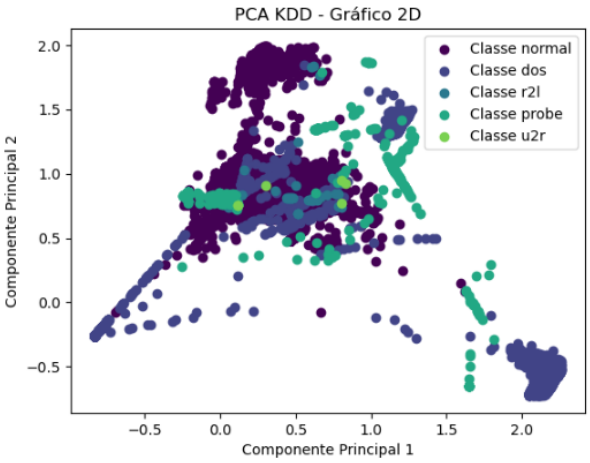
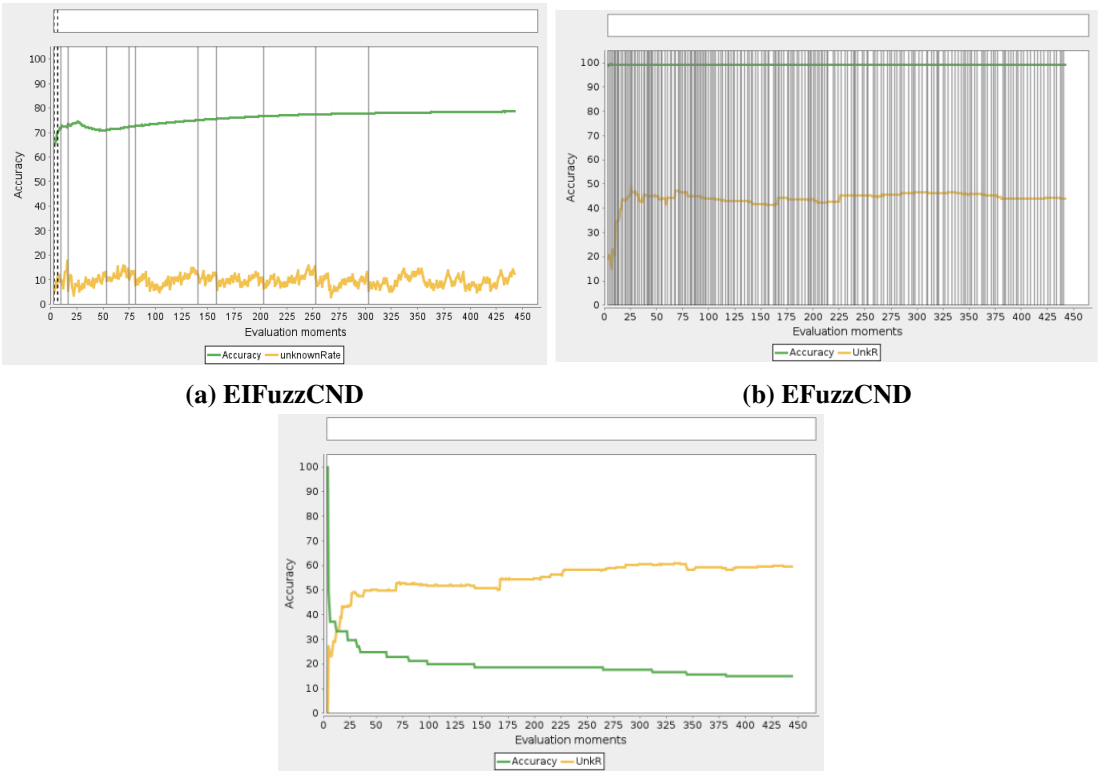


Figura 6.7: Distribuição dos dados no conjunto de treinamento KDD99 após PCA

Ao lidar com o conjunto de dados KDD99, é importante notar que ele possui um maior número de atributos em comparação com conjuntos de dados como MOA e RBF. Com 34 atributos, a visualização clara da distribuição das classes após a aplicação do PCA pode ser desafiadora. No entanto, mesmo com essa complexidade, o algoritmo EIFuzzCND demonstra uma taxa de acurácia inicialmente variando entre 0,64 e 0,71 durante as primeiras avaliações.

Durante a fase online, o algoritmo proposto demonstrou um desempenho significativo no contexto de latência extrema ao lidar com o dataset KDD99 como pode-se observar na Figura 6.8a. Ao comparar sua taxa de acurácia com o algoritmo MINAS, apresentado na Figura 6.8c, observou-se que o algoritmo proposto obteve uma taxa de acurácia maior. Isso indica que o algoritmo proposto foi mais eficiente na classificação correta das instâncias.

Além disso, a taxa de acurácia do algoritmo proposto mostrou um crescimento progressivo à medida que mais exemplos chegavam ao fluxo de dados. Essa capacidade de melhoria contínua na precisão da classificação é uma característica valiosa do algoritmo, pois permite que ele se adapte e refine suas decisões à medida que mais informações se tornam disponíveis. É importante observar que por mais que na Figura 6.5b o valor de acurácia se manteve sempre em 100%, a taxa de desconhecidos se manteve superior a 40% e o número de novidades detectadas é muito alto.



(a) EIFuzzCND **(b) EFuzzCND**

(c) MINAS

Figura 6.8: Acurácia e taxa de desconhecidos dos algoritmos, considerando latência extrema, para o conjunto de dados KDD99.

No que diz respeito à taxa de desconhecidos, embora tenha havido variações ao longo do tempo, ela não sofreu grandes alterações significativas em comparação com o algoritmo EFuzzCND e o MINAS. Isso indica que o algoritmo proposto conseguiu manter uma taxa de desconhecidos estável.

Durante a avaliação no instante 303, ocorre a detecção final do padrão de novidade. Ao analisarmos a Tabela 6.4, podemos identificar a origem dessa classe e quantificar a quantidade de exemplos que foram rotulados erroneamente.

Ao analisar a tabela 6.4, podemos observar algumas informações relevantes sobre as classes "Normal"e "DoS"e suas instâncias corretamente classificadas.

A classe "Normal"possui um total de 59.833 instâncias, das quais 27.662 foram classificadas corretamente. No entanto, é importante destacar que um número considerável de instâncias da classe "Normal"foi classificado erroneamente como pertencente à classe "r2l", totalizando 18.337 instâncias. Essa má classificação pode indicar uma sobreposição de características entre as classes "Normal"e "r2l", resultando em confusão por parte do algoritmo de classificação.

Em relação à classe "DoS", temos um total de 240.556 instâncias, das quais 238.472 fo-

Classes	Normal	u2r	r2l	DoS	UnkMem	probe	NP1	NP2	NP3	NP4	NP5	NP6	NP7	NP8	NP9	NP10	NP11
Normal	22404	0	15383	8030	66	2625	331	181	115	74	194	0	64	163	10	30	4
u2r	14	0	1	0	0	8	0	1	0	0	0	0	0	0	0	0	0
r2l	103	0	202	0	5	276	0	3	0	0	1	0	0	0	0	0	0
DoS	739	0	742	198040	11	40	0	0	1	159	0	0	0	0	2	1	0
UnkMem	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
probe	258	0	58	17	43	1509	3	6	0	11	0	88	0	4	22	0	26

(a) Matriz de confusão incremental (MCI) antes do surgimento do padrão de novidade (NP12)

Classes	Normal	DoS	UnkMem	u2r	r2l	probe	NP1	NP2	NP3	NP4	NP5	NP6	NP7	NP8	NP9	NP10	NP11	NP12
Normal	27662	9217	25	0	18337	3102	394	214	139	93	237	0	92	247	10	30	4	30
DoS	917	238472	5	0	904	56	0	0	1	197	0	0	0	0	2	1	0	1
UnkMem	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
u2r	19	0	0	0	1	10	0	1	0	0	0	0	0	0	0	0	0	0
r2l	127	0	0	0	244	324	0	3	0	0	1	0	0	0	0	0	0	0
probe	335	17	16	0	64	1874	3	6	0	13	0	96	0	4	27	0	32	0

(b) Matriz de confusão incremental (MCI) após do surgimento do padrão de novidade (NP12)

Tabela 6.4: Matriz de confusão incremental (MCI) representando o momento antes e depois do surgimento do padrão de novidade (NP12) no dataset KDD99.

ram corretamente classificadas. Houve um número relativamente baixo de instâncias da classe "DoS"classificadas erroneamente como "Normal", totalizando 917 instâncias. Também ocorreu uma má classificação de 917 instâncias da classe "DoS"como pertencentes à classe "r2l".

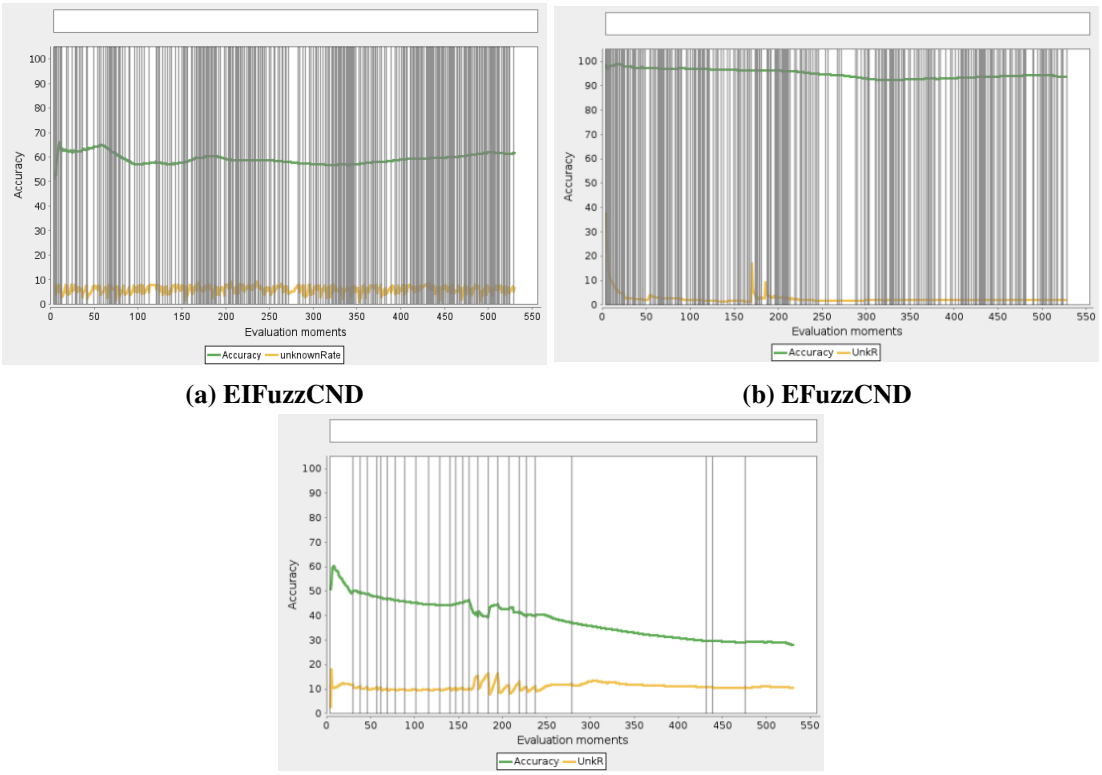
No contexto de uma situação real, essas instâncias classificadas erroneamente podem não ter um impacto significativo. O ataque de negação de serviço (DoS) geralmente requer uma grande quantidade de acessos maliciosos para sobrecarregar um sistema ou rede. Portanto, o fato de algumas instâncias da classe "DoS"serem classificadas incorretamente como "Normal"ou "r2l"pode ter uma relevância menor em termos de impacto real.

6.1.4 CoverType

Antes de iniciar a análise, é importante destacar os desafios apresentados pelo dataset CoverType. Trata-se de um conjunto de dados complexo, com alta dimensionalidade e distribuição heterogênea. Essas características podem dificultar a correta classificação das instâncias e requerer o uso de técnicas avançadas de aprendizado de máquina.

Ao comparar o desempenho do algoritmo EIFuzzCND representado na Figura 6.9a com o algoritmo EFuzzCND representado na Figura 6.9b, observamos que o EIFuzzCND obteve uma taxa de acurácia inferior. Isso indica que o algoritmo proposto não alcançou a mesma precisão na classificação das instâncias em comparação com o algoritmo EFuzzCND.

No entanto, é importante notar que a taxa de detecção de novidades foi similar entre os dois algoritmos. Isso indica que ambos possuem uma boa capacidade de identificar padrões de novidade, o que é uma característica essencial para algoritmos de detecção de anomalias.



(a) EIFuzzCND **(b) EFuzzCND**

(c) MINAS

Figura 6.9: Acurácia e taxa de desconhecidos dos algoritmos, considerando latência extrema, para o conjunto de dados Cover.

Ao compararmos o desempenho do algoritmo EIFuzzCND com o algoritmo MINAS no dataset CoverType, observamos que o EIFuzzCND apresentou um desempenho superior. Embora a taxa de acurácia do EIFuzzCND seja inferior à do EFuzzCND, o algoritmo proposto demonstrou uma maior precisão na classificação e detecção de novidades em relação ao MINAS. Esses resultados ressaltam a promessa do EIFuzzCND como uma abordagem avançada para lidar com o dataset CoverType, superando uma abordagem tradicional como o MINAS.

É relevante destacar que, devido à natureza incremental e fuzzy do algoritmo EIFuzzCND, a presença de sobreposição entre as classes pode representar um desafio. A sobreposição de padrões dificulta a separação clara entre as classes e pode afetar a precisão da classificação. Portanto, é importante considerar essa dificuldade ao aplicar o EIFuzzCND em conjuntos de dados com sobreposição de classes. É necessário desenvolver estratégias adequadas para lidar com essa sobreposição e explorar abordagens complementares que possam melhorar ainda mais o desempenho do EIFuzzCND nessas situações desafiadoras. Essa análise ressalta a importância de considerar as características específicas do dataset e adaptar os algoritmos de detecção de novidades de acordo com as demandas e peculiaridades do problema em questão.

6.1.5 SynEDC

O dataset SynEDC apresenta desafios específicos devido à sua complexidade e desbalanceamento. As classes desse conjunto de dados possuem distribuições diferentes, o que pode ser observado já no conjunto de treinamento, como ilustrado na Figura 6.10. Esse desbalanceamento pode ter impacto nos resultados dos algoritmos de detecção de novidades, uma vez que o número de instâncias de uma classe pode ser significativamente maior do que o de outras classes. É importante considerar essa desigualdade na distribuição das classes ao avaliar o desempenho e interpretar os resultados dos algoritmos aplicados a esse conjunto de dados.

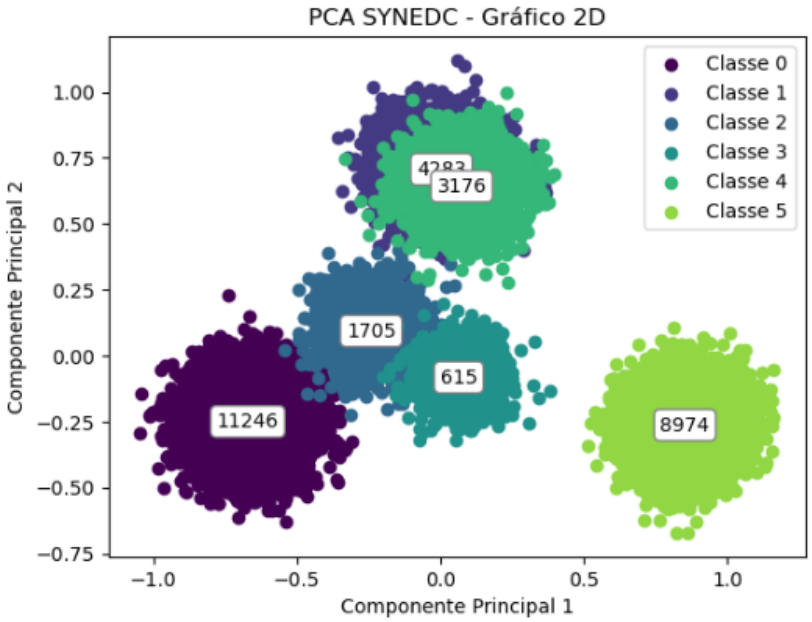


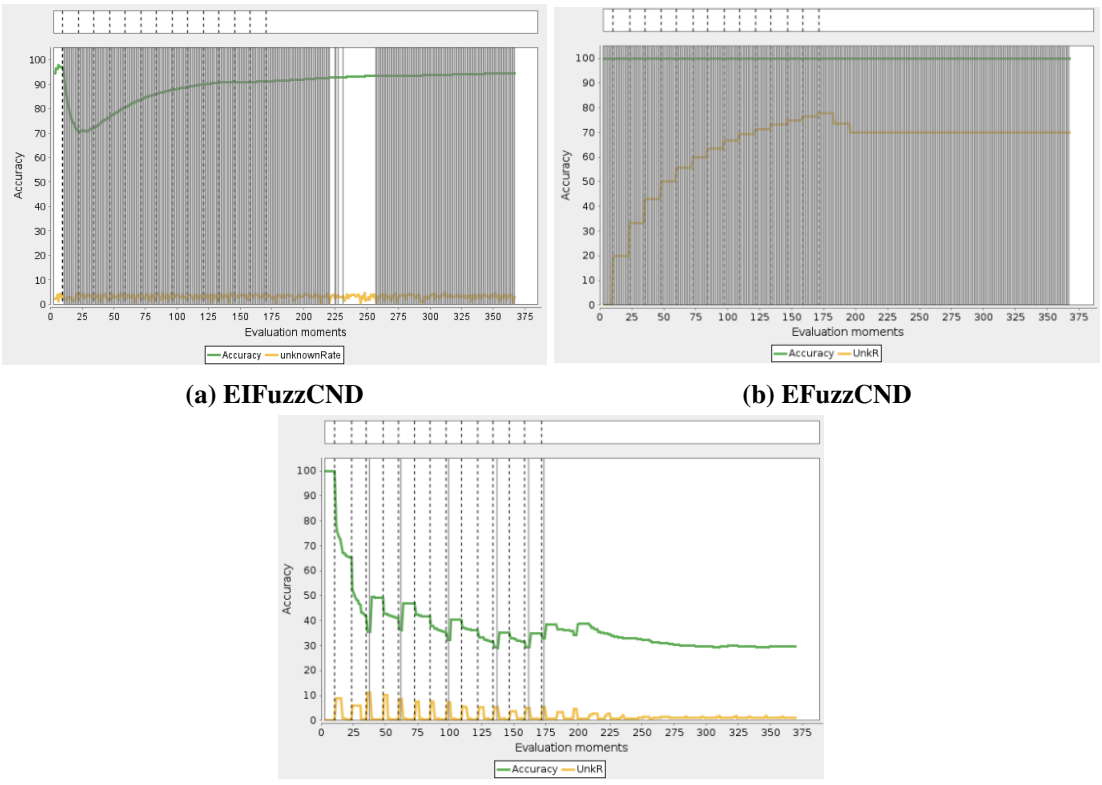
Figura 6.10: Distribuição dos dados no conjunto de treinamento SYNEDC após PCA

Além do desbalanceamento das classes, é possível observar na Figura 6.10 uma sobreposição entre as classes 1 e 4. Essa sobreposição ocorre quando as instâncias dessas duas classes possuem características semelhantes, tornando-as mais difíceis de serem distintas apenas com base nos atributos utilizados pelo algoritmo de detecção de novidades.

Essa sobreposição entre as classes 1 e 4 pode impactar o desempenho do algoritmo de várias maneiras. Primeiramente, a sobreposição aumenta a ambiguidade na classificação dessas instâncias, levando a uma maior taxa de erros ou classificações incorretas. Além disso, a presença de sobreposição pode dificultar a identificação de padrões de novidade específicos para cada classe, uma vez que as características distintivas podem se sobrepor, levando a uma detecção menos precisa.

As Figuras 6.11 apresentam o desempenho dos algoritmos EFuzzCND, EIFuzzCND e MI-

NAS na fase Online, considerando o conjunto de dados SYNEDC.



(a) EIFuzzCND **(b) EFuzzCND**

(c) MINAS

Figura 6.11: Acurácia e taxa de desconhecidos dos algoritmos, considerando latência extrema, para o conjunto de dados SYNEDC.

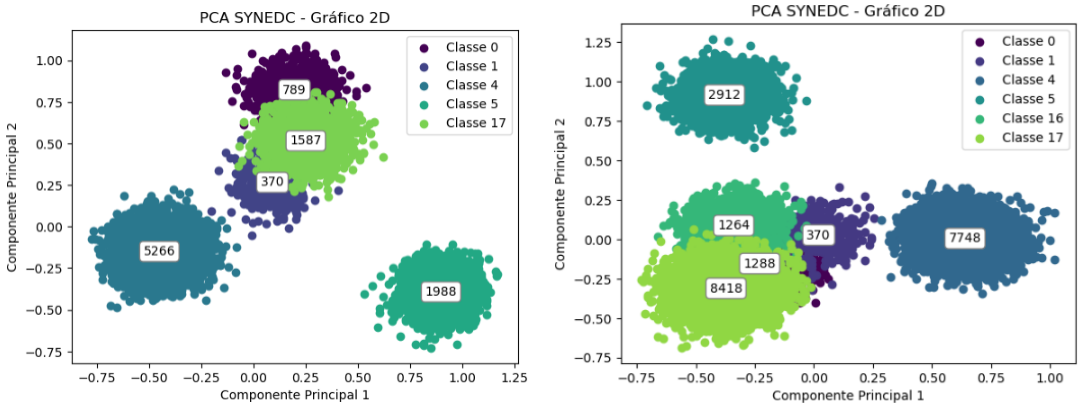
O algoritmo MINAS apresenta um padrão diferente dos deemais algoritmos em relação à taxa de desconhecidos e à acurácia. Inicialmente, o MINAS mantém uma taxa de desconhecidos relativamente alta em comparação com o EFuzzCND. Entretanto, ao longo do fluxo de dados, o MINAS gradualmente reduz sua taxa de desconhecidos e piora sua acurácia. Isso sugere que o algoritmo está se adaptando ao longo do tempo, porém identificando erroneamente as instâncias pertencentes às classes conhecidas e diminuindo a taxa de instâncias desconhecidas.

O EFuzzCND, por outro lado, exibe uma taxa de desconhecidos inicialmente alta, atingindo até 70%. Isso pode ser atribuído à natureza não incremental do algoritmo, que pode ser mais sensível a mudanças nos dados, especialmente quando não possui informações prévias sobre novos exemplos.

No entanto, o EFuzzCND mantém uma taxa de acurácia relativamente estável ao longo do fluxo de dados porém se considerarmos os desconhecidos no cálculo de acurácia assim como o EIFuzzCND, podemos verificar que o algoritmo não está conseguindo detectar novos padrões.

É essencial analisar os momentos específicos no gráfico para compreender melhor o desem-

penho do algoritmo. Uma variação significativa na taxa de acurácia ocorre entre os momentos de avaliação 10 e 22. No momento 10, a acurácia é de 0,86, enquanto no momento 22 é de 0,70. Essa queda na acurácia pode ser atribuída a diversos fatores, como a introdução de novas classes ou a presença de sobreposição entre as classes.



(a) Momento de avaliação 10.000 **(b) Momento de avaliação 22.000**

Figura 6.12: Distribuição dos dados no conjunto de treinamento SYNEDC após PCA nos momentos de avaliação 10.000 e 22.000

No momento de avaliação 7, uma nova classe, a classe 17, é adicionada ao conjunto de dados. Essa classe é identificada pelo algoritmo no momento de avaliação 8. No entanto, no momento de avaliação 10, a Figura 6.12a mostra uma sobreposição entre as classes 0, 1 e 17. Posteriormente, entre o momento 10 e o momento 22, surge uma nova classe, a classe 16. Essa sobreposição entre as classes 0, 1, 16 e 17 dificulta o processo de classificação do algoritmo, mesmo com o uso de técnicas fuzzy. A sobreposição de características e padrões entre essas classes pode levar a ambiguidades na classificação das instâncias, resultando em quedas na acurácia.

Portanto, a presença de sobreposição entre as classes no conjunto de dados SYNEDC representa um desafio para o desempenho do algoritmo de detecção de novidades. A sobreposição dificulta a distinção adequada das instâncias e pode levar a erros de classificação.

Em relação ao número de detecções de novidades, os dois algoritmos apresentaram valores similares. No entanto, foram observados comportamentos diferentes entre os momentos de avaliação 225 e 260. É difícil definir com precisão o que realmente acontece nesse intervalo de tempo devido à complexidade do dataset SynEDC. Tanto a análise gráfica quanto a análise da MCI se tornam muito complexas nessa região.

No entanto, podemos supor que entre esses momentos de avaliação, os novos exemplos apresentados já se encaixam nos grupos e padrões de novidades previamente identificados pe-

los algoritmos. Dessa forma, não são identificados novos padrões de novidade no gráfico e na análise da MCI nesse intervalo específico. Essa suposição sugere que os algoritmos tenham aprendido e se adaptado aos padrões de novidade existentes até aquele ponto, tornando a detecção de novidades menos proeminente nessa região do fluxo de dados.

6.2 Latência Intermediária

Nesta seção, apresentaremos e discutiremos os resultados obtidos na avaliação do algoritmo EIFuzzCND em comparação com os algoritmos existentes na literatura para o cenário de latência intermediária. O cenário de latência intermediária refere-se a situações em que há um atraso moderado na disponibilidade dos rótulos verdadeiros no fluxo de dados.

Em cenários de latência intermediária, espera-se que abordagens incrementais, como o algoritmo EIFuzzCND, ofereçam benefícios em termos de desempenho em comparação com abordagens tradicionais de aprendizado de máquina. Acredita-se que o EIFuzzCND possa apresentar resultados semelhantes ao EFuzzCND em latências menores, devido à sua capacidade de lidar com a falta imediata dos rótulos verdadeiros no fluxo de dados. No entanto, espera-se que o EIFuzzCND supere o EFuzzCND em latências mais altas, graças à sua característica incremental, que permite uma adaptação mais eficiente a mudanças de conceito ao longo do tempo. Essa característica pode proporcionar um desempenho superior na atualização do modelo e na detecção de novidades em cenários de latência intermediária mais prolongada.

A análise de latência intermediária será conduzida considerando três valores de latência: 2000, 5000 e 10000. Nosso objetivo é avaliar o desempenho do algoritmo proposto, EIFuzzCND, em comparação com os algoritmos EFuzzCND e ECSMiner, nesses diferentes cenários de latência.

6.2.1 MOA3

No conjunto de treinamento do dataset MOA3, é observada uma distribuição equilibrada entre as duas classes, o que favorece o modelo na definição precisa dos micro-grupos durante a fase offline. Como a fase offline não depende do valor da latência, o modelo é capaz de aproveitar a distribuição equilibrada das classes para realizar uma análise precisa e estabelecer os micro-grupos de forma eficiente contribuindo para um desempenho similar ao da latência extrema nessa fase.

Ao comparar o desempenho do EIFuzzCND com o EFuzzCND presente na Figura 6.13,

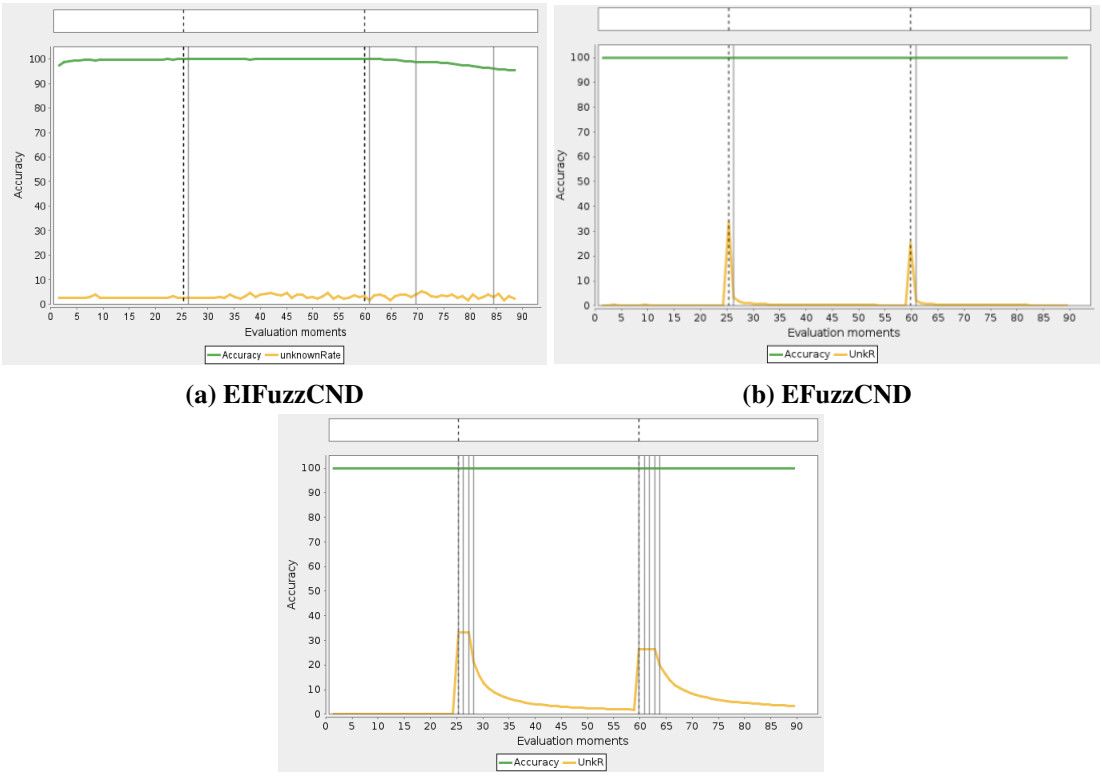
observamos que o algoritmo proposto apresentou uma taxa de acurácia menor. Essa diferença pode ser atribuída à abordagem incremental do EIFuzzCND, que permite uma adaptação mais flexível a mudanças de conceito, mas pode resultar em um comprometimento da acurácia em comparação com algoritmos que dependem dos rótulos verdadeiros.

Isso acontece porque o EIFuzzCND adota uma abordagem incremental que lhe permite se adaptar mais flexivelmente a mudanças de conceito, incorporando novos exemplos e atualizando seus modelos conforme novas informações são recebidas. Essa flexibilidade é uma vantagem quando o ambiente de fluxo de dados é dinâmico e as classes estão sujeitas a evoluções ao longo do tempo.

Quando o EIFuzzCND lida com instâncias desconhecidas ou novas classes que não foram previamente observadas durante a fase offline, pode cometer erros de classificação, levando a uma redução na acurácia. Por outro lado, o EFuzzCND, que não é incremental e depende dos rótulos verdadeiros para a classificação, pode manter uma acurácia mais alta quando confrontado com novidades, uma vez que não toma decisões de classificação até que os rótulos reais estejam disponíveis.

Em relação ao ECSMiner, pode-se destacar que o EIFuzzCND manteve uma baixa taxa de instâncias classificadas como desconhecidas, demonstrando sua eficiência em lidar com a incerteza. Essa característica é crucial para garantir um bom desempenho na detecção de novidades, uma vez que minimiza a ocorrência de falsos desconhecidos e maximiza a capacidade do algoritmo em identificar corretamente instâncias anômalas.

No que diz respeito ao comportamento do algoritmo no cenário de latência intermediária, é importante observar que o EIFuzzCND apresentou um padrão semelhante ao cenário de latência extrema. Apesar de ter uma taxa de acurácia inferior ao EFuzzCND, o EIFuzzCND manteve uma tendência consistente com o desempenho observado na latência extrema.



(a) EIFuzzCND **(b) EFuzzCND**

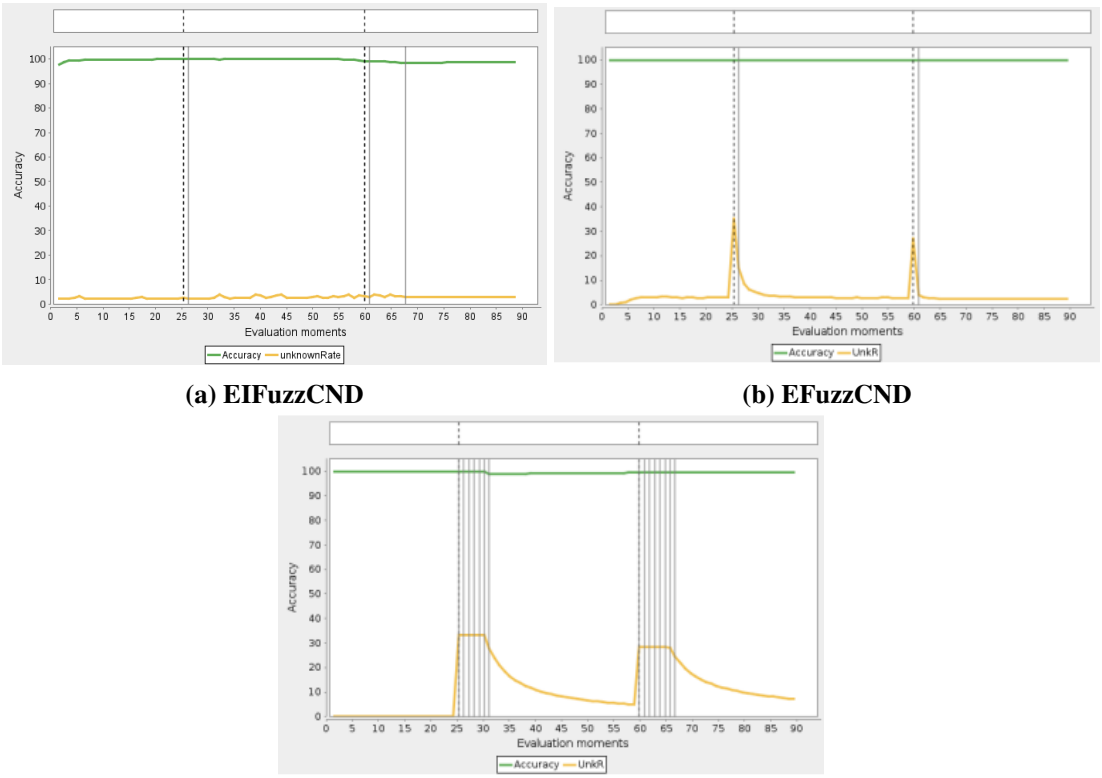
(c) ECSMiner

Figura 6.13: Acurácia e taxa de desconhecidos dos algoritmos, considerando latência intermediária de 2000, para o conjunto de dados MOA3.

Observando a Figura 6.14 no cenário de latência intermediária com valor de 5000, podemos analisar alguns pontos relevantes. Em relação à taxa de acurácia, é observado que o algoritmo EIFuzzCND apresenta um desempenho inferior em comparação ao EFuzzCND, como já constatado anteriormente. No entanto, é interessante destacar que, em comparação com o valor de latência de 2000, o EIFuzzCND começa a demonstrar uma melhoria gradual em seus resultados.

Essa melhoria progressiva era esperada, uma vez que o algoritmo EIFuzzCND possui uma característica incremental que o permite adaptar-se melhor a mudanças de conceito ao longo do tempo. Em cenários de latência intermediária, em que há um atraso moderado na disponibilidade dos rótulos verdadeiros no fluxo de dados, o EIFuzzCND tem a oportunidade de ajustar e atualizar seu modelo de forma mais eficiente.

Ao comparar os gráficos em relação ao valor de latência de 2000, podemos observar que o EIFuzzCND demonstra uma tendência de desempenho mais promissor à medida que o valor de latência aumenta porque ouve uma redução no número de padrões de detecção de novidade. Isso indica que, conforme o atraso na disponibilidade dos rótulos verdadeiros aumenta, o EIFuzzCND é capaz de lidar melhor com essa situação e obter resultados mais satisfatórios.



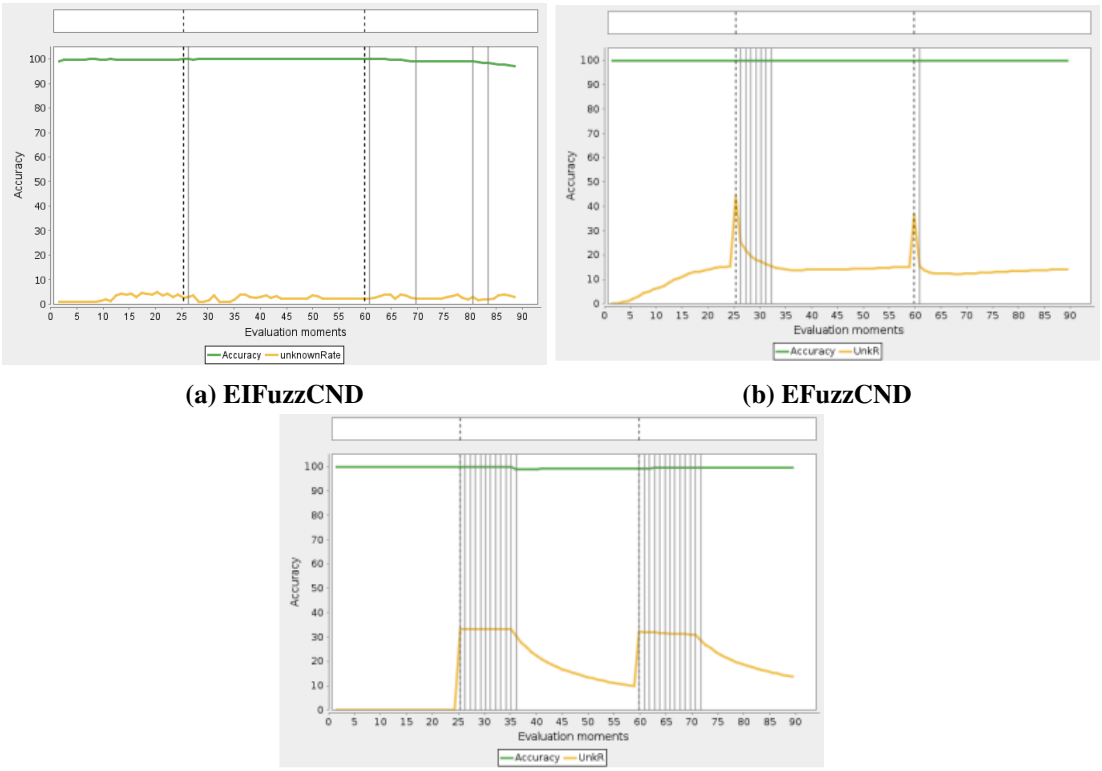
(a) EIFuzzCND **(b) EFuzzCND**

(c) ECSMiner

Figura 6.14: Acurácia e taxa de desconhecidos dos algoritmos, considerando latência intermediária de 5000, para o conjunto de dados MOA3.

Ao analisarmos a Figura 6.15 no cenário de latência intermediária com valor de 10000, podemos destacar que o algoritmo EIFuzzCND apresentou um desempenho superior em relação ao EFuzzCND. Essa diferença de desempenho é evidente ao observarmos tanto o número de novidades detectadas quanto a taxa de desconhecidos.

Esses resultados evidenciam a robustez e a eficácia do EIFuzzCND em lidar com os desafios impostos pela latência intermediária, proporcionando uma detecção mais precisa e confiável de eventos anômalos.



(a) EIFuzzCND **(b) EFuzzCND**

(c) ECSMiner

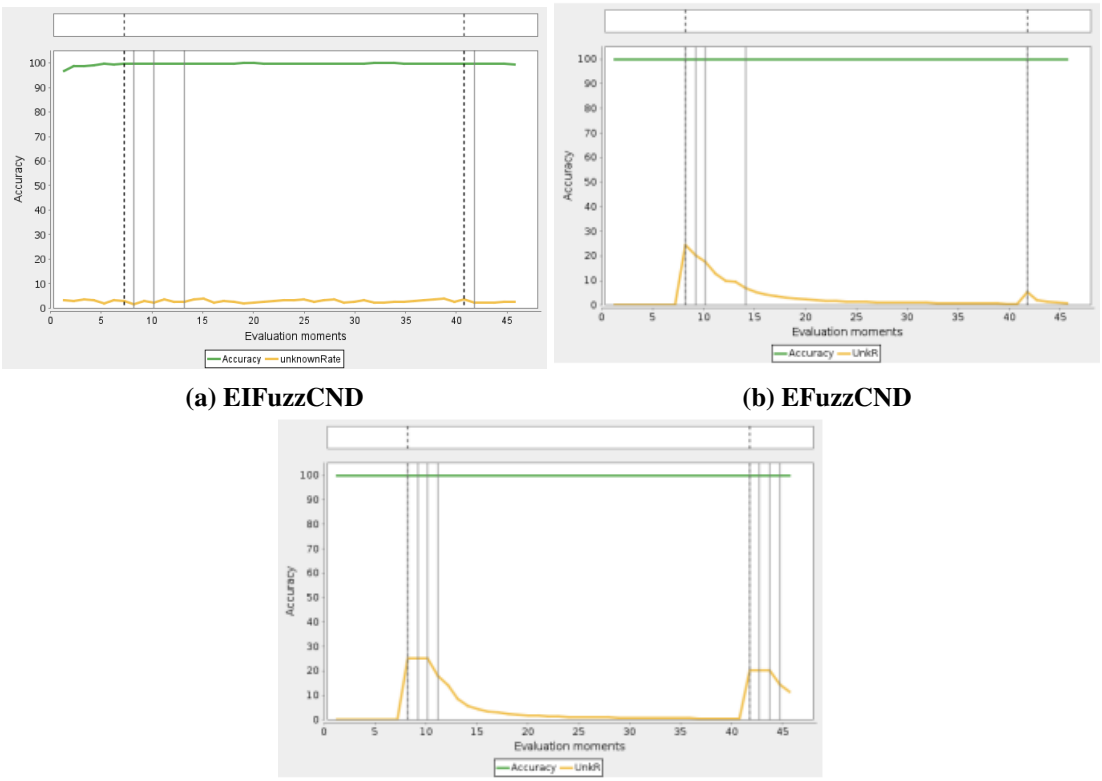
Figura 6.15: Acurácia e taxa de desconhecidos dos algoritmos, considerando latência intermediária de 10000, para o conjunto de dados MOA3.

o ECSMiner mostra um desempenho consistente em termos de acurácia, mas com uma taxa de desconhecidos relativamente alta, principalmente quando acontece o surgimento de uma nova classe. Além disso o modelo detecta uma grande quantidade de padrões de novidade sem que necessariamente surja uma nova classe.

Logo resultados obtidos indicam que, à medida que o valor da latência aumenta, o EIFuzzCND se destaca como uma solução mais eficiente e confiável em comparação com o EFuzzCND. A capacidade de identificar novidades de forma mais assertiva e manter uma baixa taxa de desconhecidos são fatores cruciais que contribuem para a eficácia do EIFuzzCND nesse contexto.

6.2.2 RBF

Ao analisarmos os gráficos presentes na Figura 6.16 no cenário de latência intermediária para o dataset RBF, podemos observar resultados significativos em relação ao desempenho do algoritmo proposto, em comparação com o EFuzzCND e o ECSMiner.



(a) EIFuzzCND **(b) EFuzzCND**

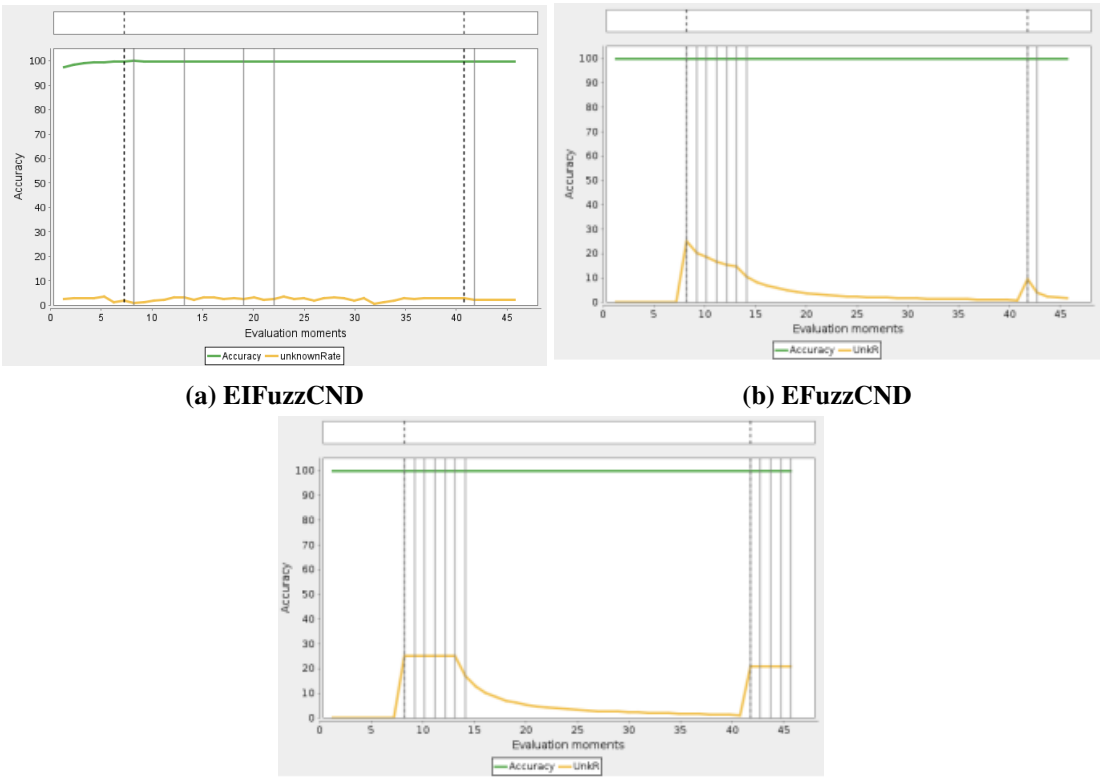
(c) ECSMiner

Figura 6.16: Acurácia e taxa de desconhecidos dos algoritmos, considerando latência intermediária de 2000, para o conjunto de dados RBF.

Em primeiro lugar, é importante ressaltar que, diferentemente do conjunto de dados MOA, no conjunto de dados RBF, o algoritmo proposto apresentou um desempenho superior em relação ao EFuzzCND e ao ECSMiner a partir da latência 2000. Essa diferença de desempenho é um indicativo do impacto do conjunto de dados específico no comportamento dos algoritmos e ressalta a importância de avaliar diferentes conjuntos de dados em estudos comparativos.

O algoritmo proposto apresentou uma baixa taxa de instâncias classificadas como desconhecidas, indicando sua capacidade de lidar com incerteza e desconhecimento nos dados. Além disso, superou o EFuzzCND na detecção de novidades, evidenciando sua capacidade de adaptar-se a mudanças de conceito e reconhecer padrões de novidade no fluxo de dados.

O mesmo padrão de resultados pode ser observado nas latências 5000 e 10000, presentes na Figura 6.17 e 6.18. Essas características se mostraram consistentes em diferentes níveis de latência, evidenciando a confiabilidade e eficiência do algoritmo proposto no cenário de latência intermediária.

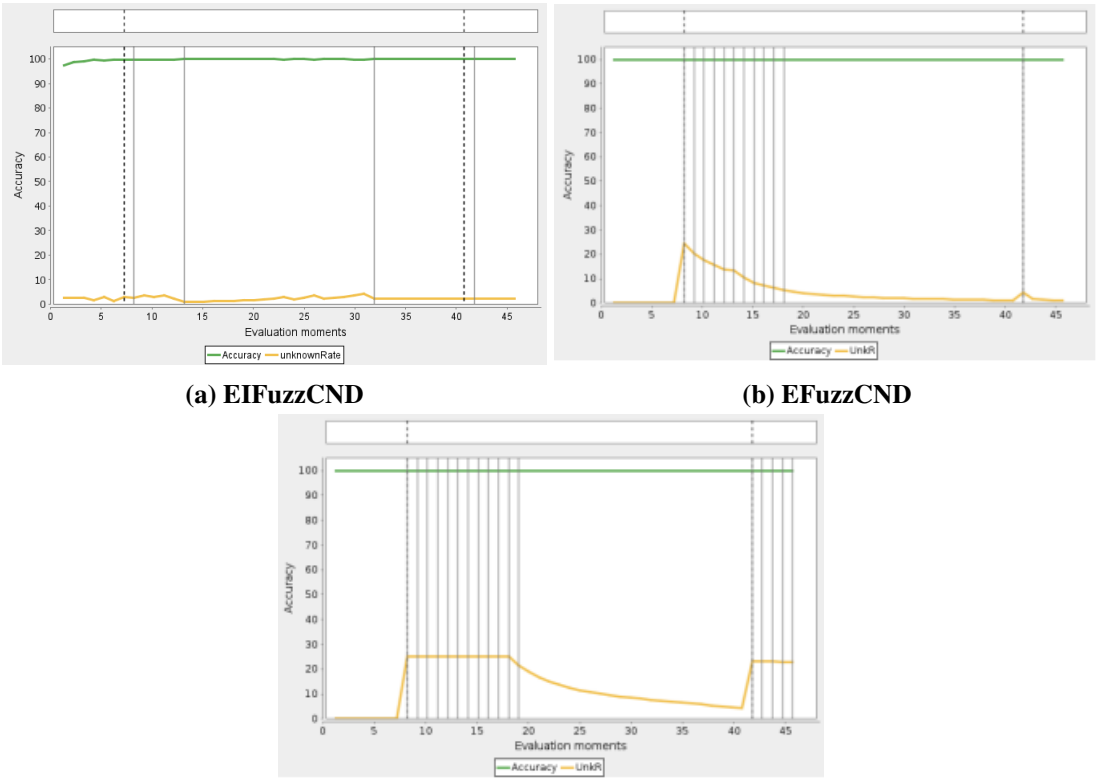


(a) EIFuzzCND **(b) EFuzzCND**

(c) ECSMiner

Figura 6.17: Acurácia e taxa de desconhecidos dos algoritmos, considerando latência intermediária de 5000, para o conjunto de dados RBF.

No cenário de latência 5.000, conforme demonstrado na Figura 6.17c, podemos observar que o EFuzzCND foi capaz de identificar o segundo padrão de novidade. Essa descoberta é especialmente relevante, pois difere do comportamento observado na latência 2.000. No entanto, no cenário de latência 10.000, o EFuzzCND novamente não conseguiu identificar o segundo padrão de novidade, enquanto o EIFuzzCND e o ECSMiner foram capazes de fazê-lo. Essa variação no desempenho do EFuzzCND em diferentes valores de latência ressalta a importância de investigar a influência da latência na capacidade de detecção de novidades dos algoritmos e enfatiza a necessidade de compreender melhor os fatores que afetam sua adaptabilidade em cenários de latência intermediária.



(a) EIFuzzCND **(b) EFuzzCND**

(c) ECSMiner

Figura 6.18: Acurácia e taxa de desconhecidos dos algoritmos, considerando latência intermediária de 10000, para o conjunto de dados RBF.

Uma observação relevante em relação ao comportamento do algoritmo proposto é sua capacidade adaptativa, especialmente em cenários com maior valor de latência. Ao comparar os resultados nos três cenários de latência, pode-se notar um aumento no espaçamento entre os padrões de novidade identificados no intervalo entre o surgimento real de uma nova classe. Isso sugere que o algoritmo é capaz de se ajustar às condições de latência e realizar detecções de novidades de forma mais espaçada, acompanhando a taxa de disponibilidade dos rótulos verdadeiros no fluxo de dados. Essa capacidade adaptativa do algoritmo é justificada pela sua característica incremental, permitindo uma atualização contínua do modelo e uma resposta adequada às mudanças de conceito ao longo do tempo.

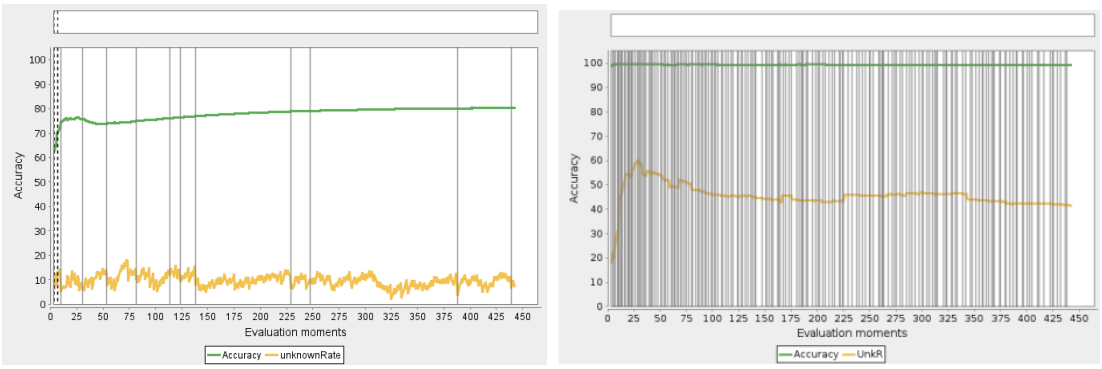
6.2.3 KDD99

O desempenho superior do algoritmo EIFUZZCND em relação ao EFuzzCND e ECSMiner no contexto de latência intermediária pode ser atribuído a várias razões, incluindo as medidas de acurácia, taxa de desconhecidos e o número de padrões de novidade detectados. Vamos analisar cada um desses fatores:

- Acurácia mais alta: O EIFUZZCND tende a apresentar uma acurácia mais alta em comparação com o EFuzzCND e o ECSMiner, conforme evidenciado nas Figuras 6.19, 6.20 e 6.21. Isso significa que o EIFUZZCND classifica corretamente um maior número de instâncias, incluindo aquelas que pertencem às classes conhecidas.

- Taxa de desconhecidos menor: O EIFUZZCND geralmente mantém uma taxa de desconhecidos mais baixa em comparação com o EFuzzCND e o ECSMiner. A taxa de desconhecidos indica a proporção de instâncias que o algoritmo não consegue classificar ou considera como desconhecidas. Um valor menor nessa métrica é desejável, pois indica que o algoritmo é mais assertivo na classificação das instâncias.

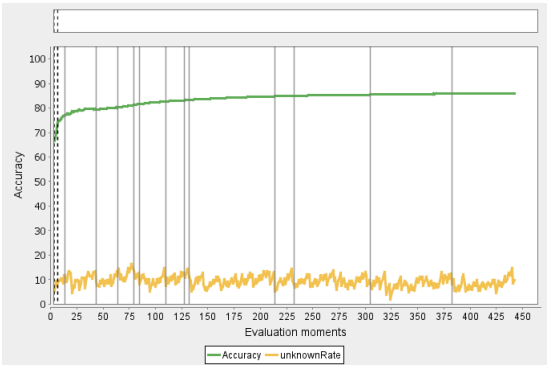
- Menor número de padrões de novidade: Embora a detecção de novidades seja importante, um número excessivamente alto de padrões de novidade pode ser problemático, pois aumenta a complexidade da tarefa de detecção e pode levar a um maior número de falsos positivos. O EIFUZZCND pode ser mais eficiente na identificação de padrões de novidade relevantes e significativos, evitando a identificação excessiva de novidades que não são verdadeiras anomalias.



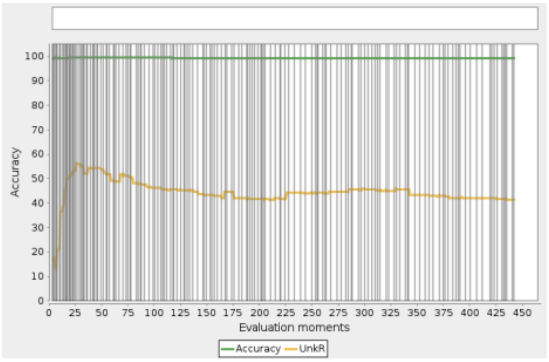
(a) EIFuzzCND **(b) EFuzzCND**

(c) ECSMiner

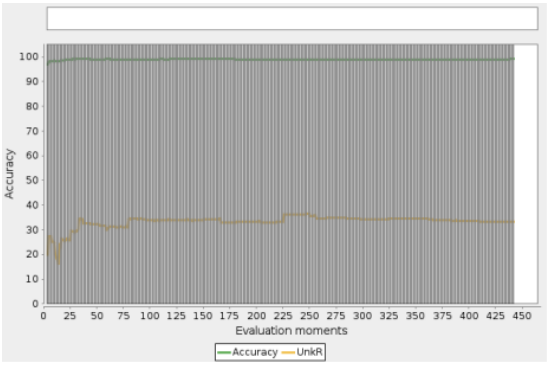
Figura 6.19: Acurácia e taxa de desconhecidos dos algoritmos, considerando latência intermediária de 2000, para o conjunto de dados KDD.



(a) EIFuzzCND

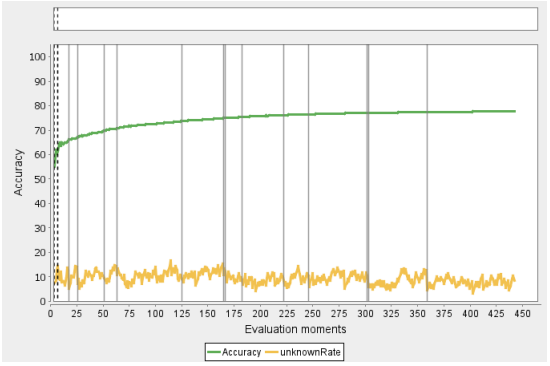


(b) EFuzzCND

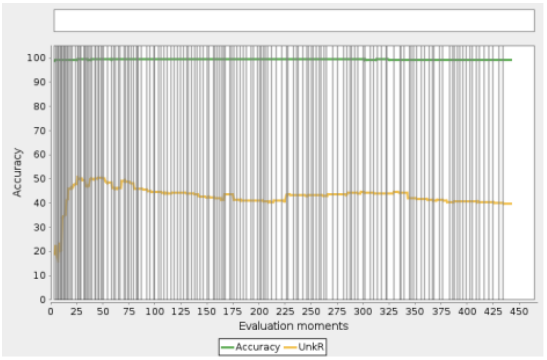


(c) ECSMiner

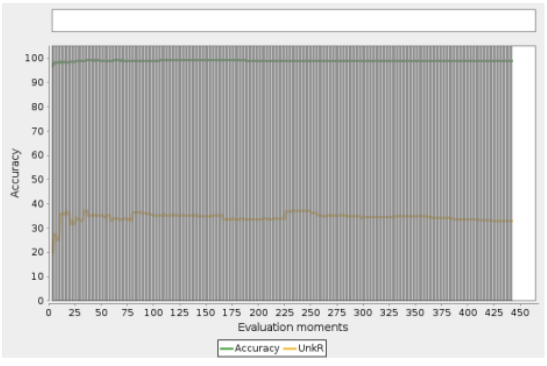
Figura 6.20: Acurácia e taxa de desconhecidos dos algoritmos, considerando latência intermediária de 5000, para o conjunto de dados KDD.



(a) EIFuzzCND



(b) EFuzzCND



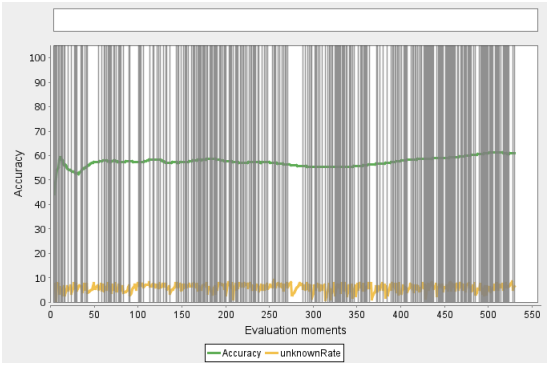
(c) ECSMiner

Figura 6.21: Acurácia e taxa de desconhecidos dos algoritmos, considerando latência intermediária de 10000, para o conjunto de dados KDD.

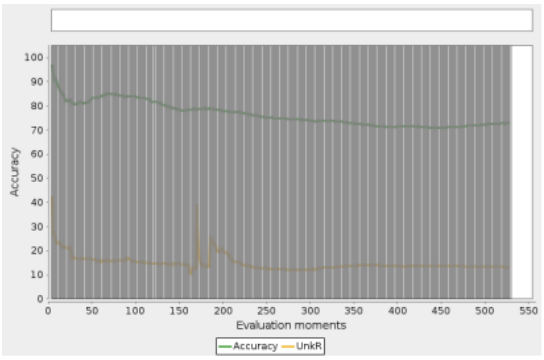
Esses fatores combinados indicam que o EIFUZZCND é capaz de equilibrar a detecção de novidades com uma classificação precisa de instâncias conhecidas, o que é fundamental em cenários de latência intermediária. Sua abordagem incremental e sua capacidade de lidar com a incerteza permitem uma adaptação eficaz às mudanças no fluxo de dados e uma detecção confiável de novidades, resultando em um desempenho superior em termos de acurácia e taxa de desconhecidos.

6.2.4 CoverType

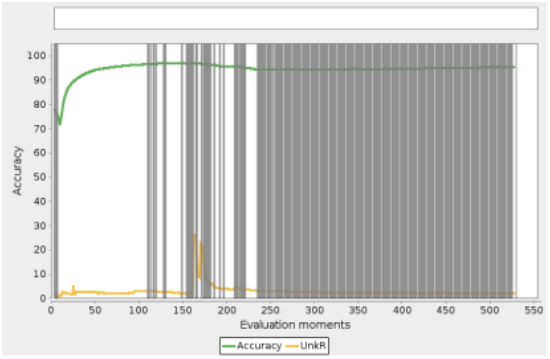
Ao analisarmos os resultados do dataset CoverType no cenário de latência intermediária, observamos um comportamento similar em relação ao desempenho do algoritmo no cenário de latência extrema. Diferentemente dos outros experimentos, nos quais o EIFuzzCND apresentou resultados superiores, neste caso específico, o algoritmo mostrou uma menor taxa de acurácia. Assim como dito na seção anterior, essa discrepância pode ser atribuída a características intrínsecas do próprio dataset, que podem ter impactado a capacidade de generalização do EIFuzzCND.



(a) EIFuzzCND



(b) EFuzzCND

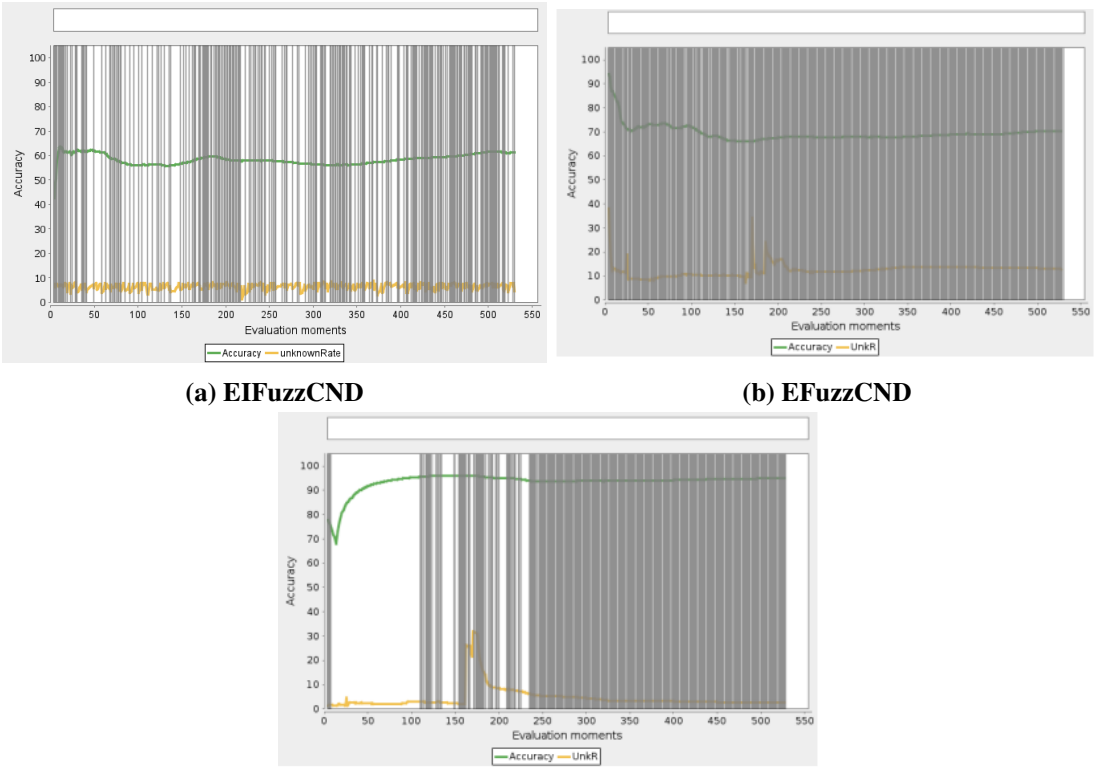


(c) ECSMiner

Figura 6.22: Acurácia e taxa de desconhecidos dos algoritmos, considerando latência intermediária de 2000, para o conjunto de dados CoverType.

A baixa taxa de acurácia do EIFuzzCND no dataset Cover em cenários de latência interme-

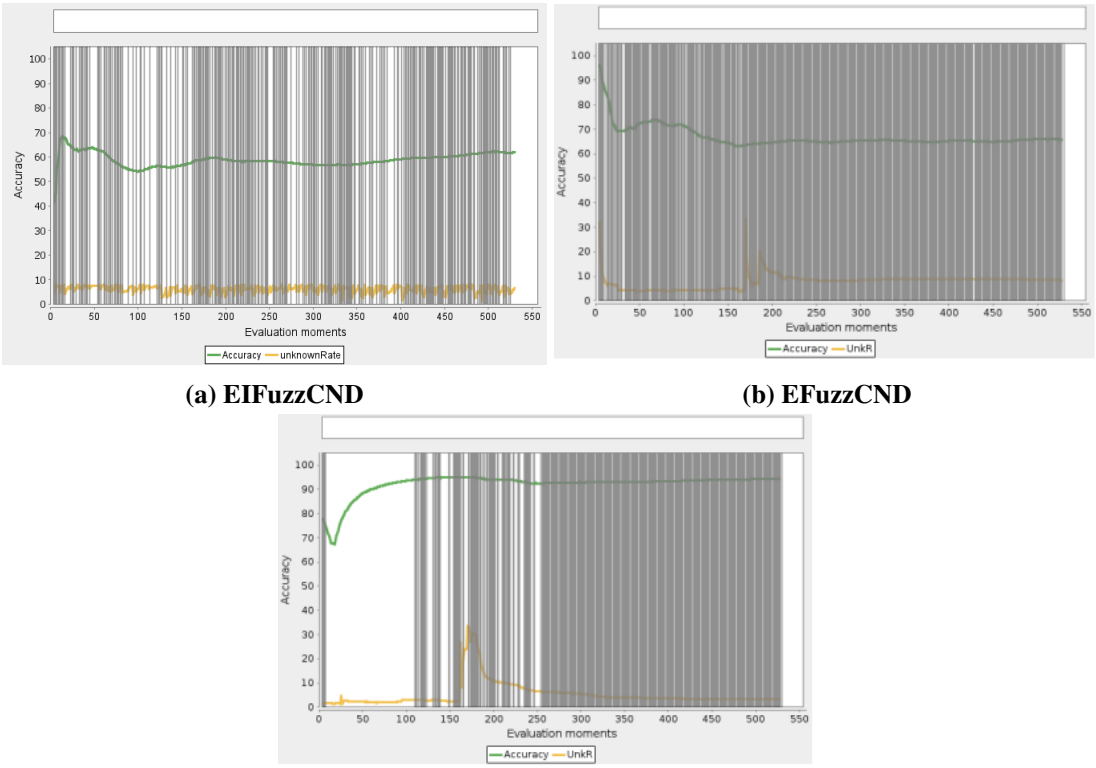
diária indica que o algoritmo pode ter dificuldades em capturar as sutilezas e complexidades das classes nesse contexto específico. Isso ressalta a importância de considerar as características do dataset e a natureza dos dados ao aplicar algoritmos de detecção de novidades.



(a) EIFuzzCND **(b) EFuzzCND**

(c) ECSMiner

Figura 6.23: Acurácia e taxa de desconhecidos dos algoritmos, considerando latência intermediária de 5000, para o conjunto de dados CoverType.



(a) EIFuzzCND **(b) EFuzzCND**

(c) ECSMiner

Figura 6.24: Acurácia e taxa de desconhecidos dos algoritmos, considerando latência intermediária de 10000, para o conjunto de dados CoverType.

Ao analisarmos as Figuras 6.22, 6.23 e 6.24, referentes aos experimentos de latência intermediária no dataset CoverType, podemos observar que o algoritmo ECSMiner obteve resultados consistentes e satisfatórios para os três valores propostos. Essa performance pode ser atribuída à utilização de uma abordagem de clusterização normal pelo ECSMiner, em contraste com os algoritmos EIFuzzCND e EFuzzCND, que utilizam clusterização fuzzy.

A clusterização normal adotada pelo ECSMiner pode ter contribuído para uma melhor adaptação aos padrões específicos presentes no dataset CoverType. Ao atribuir cada instância a um único cluster, essa abordagem pode ter proporcionado uma separação mais clara entre as classes e um ajuste mais preciso do modelo aos dados do fluxo.

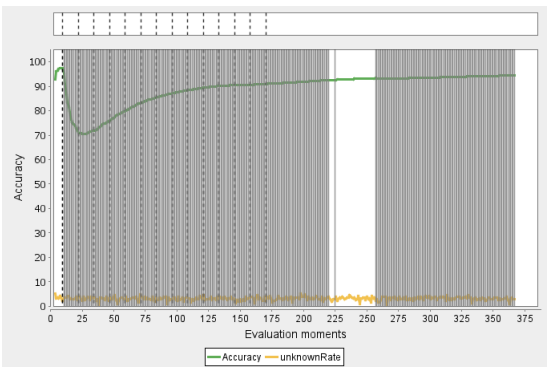
Por outro lado, a clusterização fuzzy utilizada pelos algoritmos EIFuzzCND e EFuzzCND, ao considerar a incerteza e a ambiguidade nas atribuições de instâncias aos clusters, pode ter gerado resultados menos precisos ou menos adaptados aos padrões do dataset CoverType. Essa característica da clusterização fuzzy pode ter afetado a taxa de acurácia e o desempenho na detecção de novidades desses algoritmos em comparação ao ECSMiner nos cenários de latência intermediária. Dependendo da dimensionalidade e da esparsidade do dataset, a clusterização fuzzy pode enfrentar desafios adicionais. Se o dataset for de alta dimensionalidade ou se as

instâncias estiverem espalhadas de forma desigual no espaço de características, a clusterização fuzzy pode ter dificuldades em criar clusters significativos e precisos. Além disso cada dataset tem suas próprias características. O dataset CoverType pode simplesmente não se beneficiar da clusterização fuzzy devido à natureza específica de suas classes e distribuição de dados.

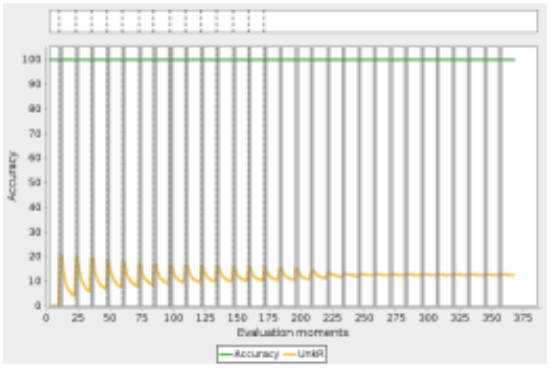
Dessa forma, a escolha da abordagem de clusterização desempenha um papel importante nos resultados obtidos em diferentes cenários de latência. No caso do dataset CoverType, a clusterização normal utilizada pelo ECSMiner se mostrou mais adequada para lidar com os padrões específicos dos dados, resultando em um desempenho superior em termos de taxa de acurácia e detecção de novidades em relação aos algoritmos EIFuzzCND e EFuzzCND.

6.2.5 SynEDC

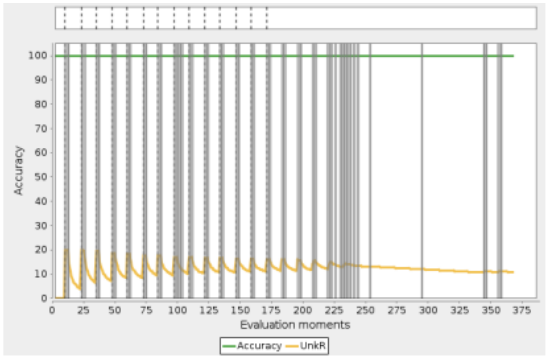
Durante os experimentos de latência intermediária no dataset SynEDC, foram observados resultados distintos entre os algoritmos avaliados. Em particular, o algoritmo EFuzzCND se destacou ao apresentar um desempenho consideravelmente superior nesse cenário. Ele foi capaz de detectar quase exclusivamente os padrões de novidade necessários, mantendo uma taxa de acurácia satisfatória. No entanto, à medida que o valor de latência aumenta, uma preocupação emerge: a taxa de instâncias classificadas como desconhecidas cresce consideravelmente. Isso sugere que o EFuzzCND pode enfrentar dificuldades em lidar com o aumento da latência, resultando em um maior número de instâncias classificadas como desconhecidas.



(a) EIFuzzCND

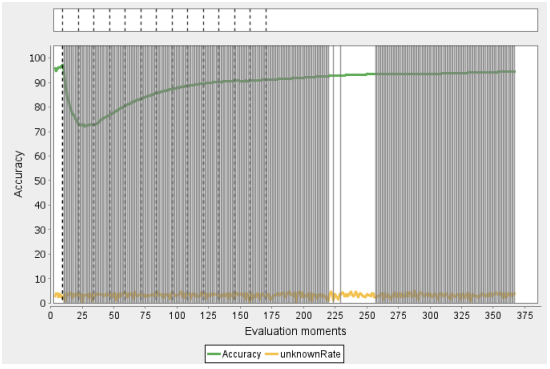


(b) EFuzzCND

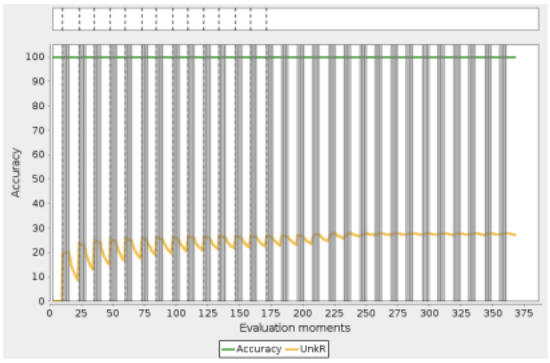


(c) ECSMiner

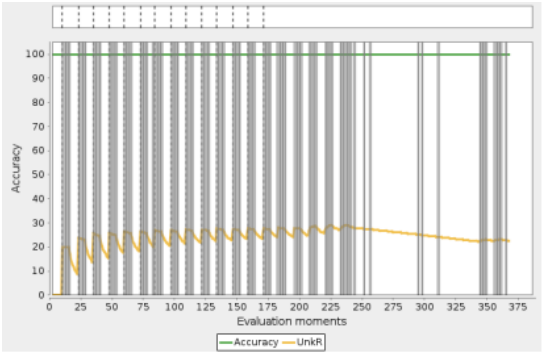
Figura 6.25: Acurácia e taxa de desconhecidos dos algoritmos, considerando latência intermediária de 2000, para o conjunto de dados SynEDC.



(a) EIFuzzCND

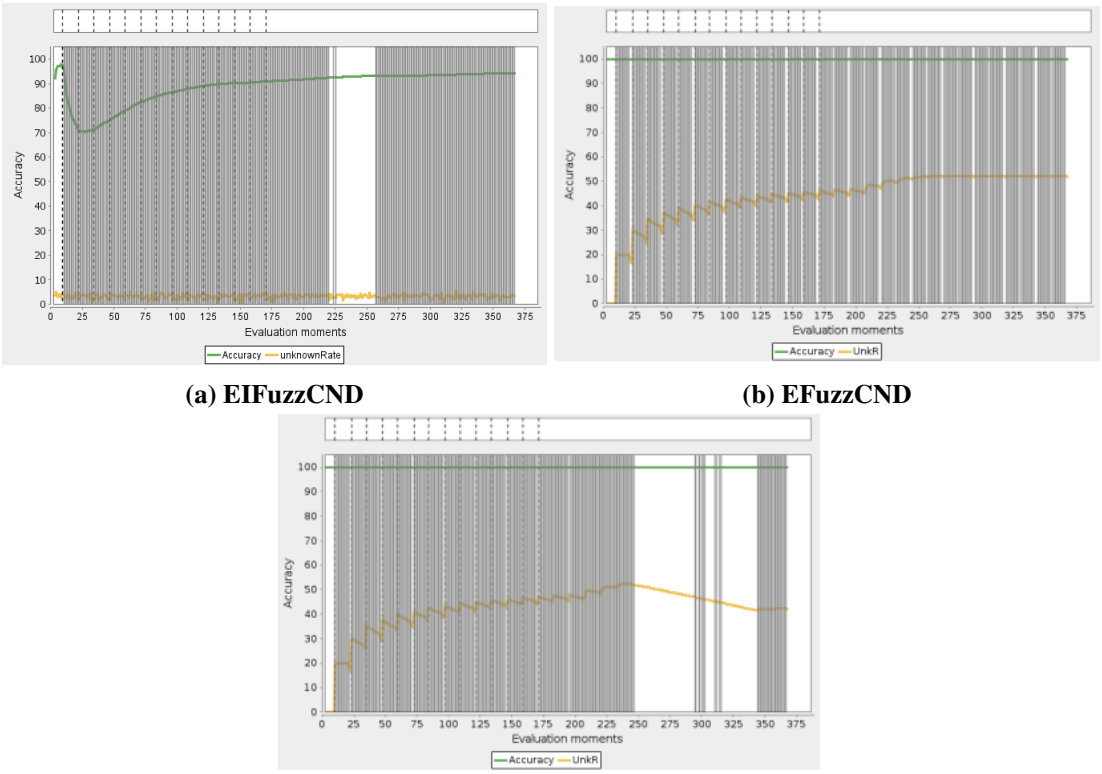


(b) EFuzzCND



(c) ECSMiner

Figura 6.26: Acurácia e taxa de desconhecidos dos algoritmos, considerando latência intermediária de 5000, para o conjunto de dados SynEDC.



(a) EIFuzzCND **(b) EFuzzCND**

(c) ECSMiner

Figura 6.27: Acurácia e taxa de desconhecidos dos algoritmos, considerando latência intermediária de 10000, para o conjunto de dados SynEDC.

Em relação ao algoritmo ECSMiner, também foi possível observar um padrão semelhante. Embora tenha detectado um número maior de novidades do que o necessário, o ECSMiner enfrentou desafios em relação à taxa de desconhecidos à medida que a latência aumentava. Esse comportamento indica que, mesmo que o algoritmo tenha a capacidade de identificar novos padrões, ele pode não ser tão eficiente em lidar com latências mais altas, resultando em uma maior taxa de instâncias desconhecidas.

Por fim o algoritmo EIFuzzCND demonstrou consistência em seus resultados tanto nos cenários de latência intermediária quanto nos cenários de latência extrema. Essa característica é vantajosa, pois indica que o algoritmo é robusto e eficiente em diferentes níveis de latência. Independentemente das variações na latência do fluxo de dados, o EIFuzzCND foi capaz de detectar novidades de maneira precisa e manter uma taxa de desconhecidos relativamente baixa.

Concluindo a análise dos cenários de latência intermediária, observamos que o algoritmo EIFuzzCND demonstrou um desempenho consistente e estável, superando os outros algoritmos em termos de taxa de acurácia, detecção de novidades e taxa de desconhecidos em quase todos os cenários, com exceção do Cover. Sua capacidade adaptativa e incremental o torna uma escolha promissora para aplicações práticas, simplificando a seleção e implementação do algoritmo,

uma vez que não é necessário ajustar seus parâmetros de acordo com variações na latência.

A próxima seção terá como foco os cenários de latência intermediária com variação na presença dos rótulos verdadeiros. Essa análise permitirá verificar se o EIFuzzCND mantém sua estabilidade e desempenho mesmo diante de mudanças dos valores percentuais e presença no fluxo. Através dessa avaliação, será possível compreender se a característica incremental do algoritmo continua a fornecer resultados similares e confiáveis, tornando-o uma opção robusta para a detecção de novidades em fluxos de dados.

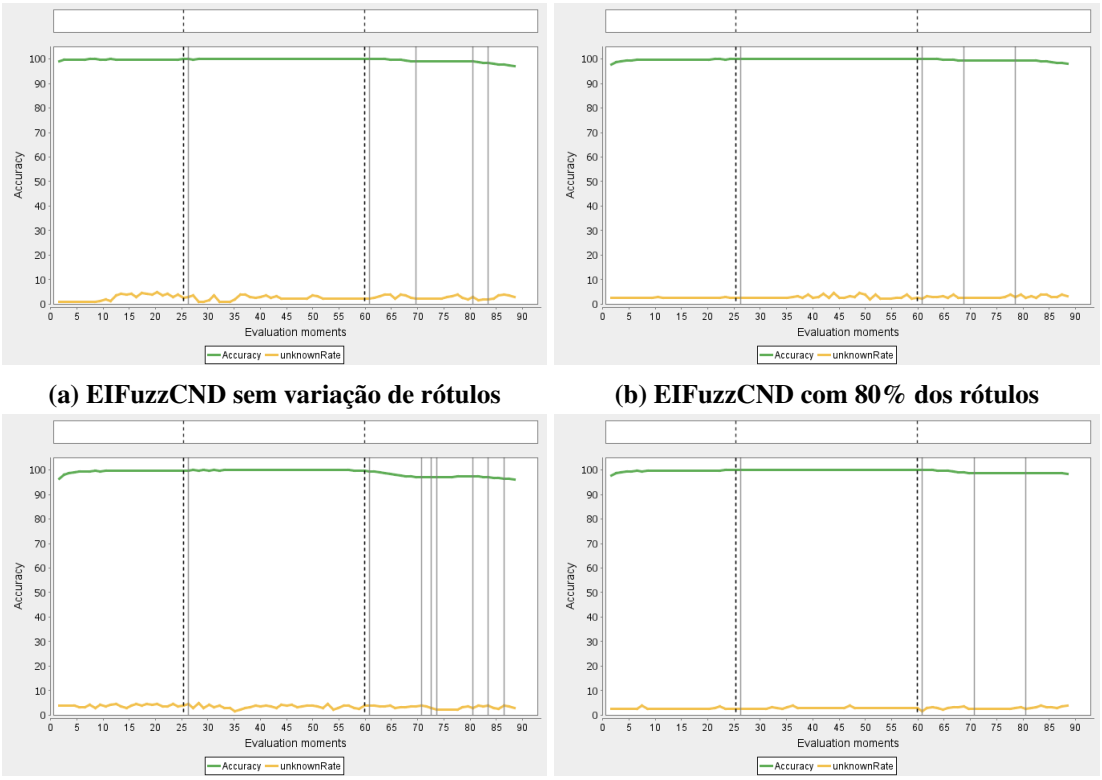
6.3 Latência Intermediária com variação no percentual de presença dos rótulos verdadeiros

Nesse cenário, iremos simular variações nos valores percentuais de presença dos rótulos no fluxo de dados. Essas variações refletem mudanças na proporção de ocorrência da presença dos rótulos verdadeiros ao longo do tempo. Dessa forma, vamos avaliar a capacidade do algoritmo EIFuzzCND em se adaptar a essas variações, mantendo seu desempenho e precisão na detecção de novidades, além de sua habilidade em lidar com a incerteza e o desconhecimento resultantes dessas mudanças nos valores percentuais de presença dos rótulos.

Para realizar a experimentação no cenário de latência intermediária com variação dos valores percentuais de presença dos rótulos, foram selecionados os valores de 20%, 50% e 80%, além do valor já analisado anteriormente de 100%. O valor de latência escolhido para essa análise foi de 10.000, uma vez que, como observado na seção anterior, esse valor proporcionou melhores resultados em termos de desempenho e acurácia para a maioria dos datasets.

6.3.1 MOA3

Ao analisarmos a Figura 6.28 referente ao dataset MOA no cenário de latência intermediária com variação dos valores percentuais de presença dos rótulos, podemos observar um comportamento consistente do algoritmo EIFuzzCND. Ao longo dos diferentes valores percentuais (20%, 50%, 80% e 100%), notamos que o desempenho do EIFuzzCND se manteve relativamente estável. Isso significa que o algoritmo conseguiu lidar de maneira eficiente com as variações nos valores percentuais, mantendo um bom desempenho na detecção de novidades.



(a) EIFuzzCND sem variação de rótulos

(b) EIFuzzCND com 80% dos rótulos

(c) EIFuzzCND com 50% dos rótulos

(d) EIFuzzCND com 20% dos rótulos

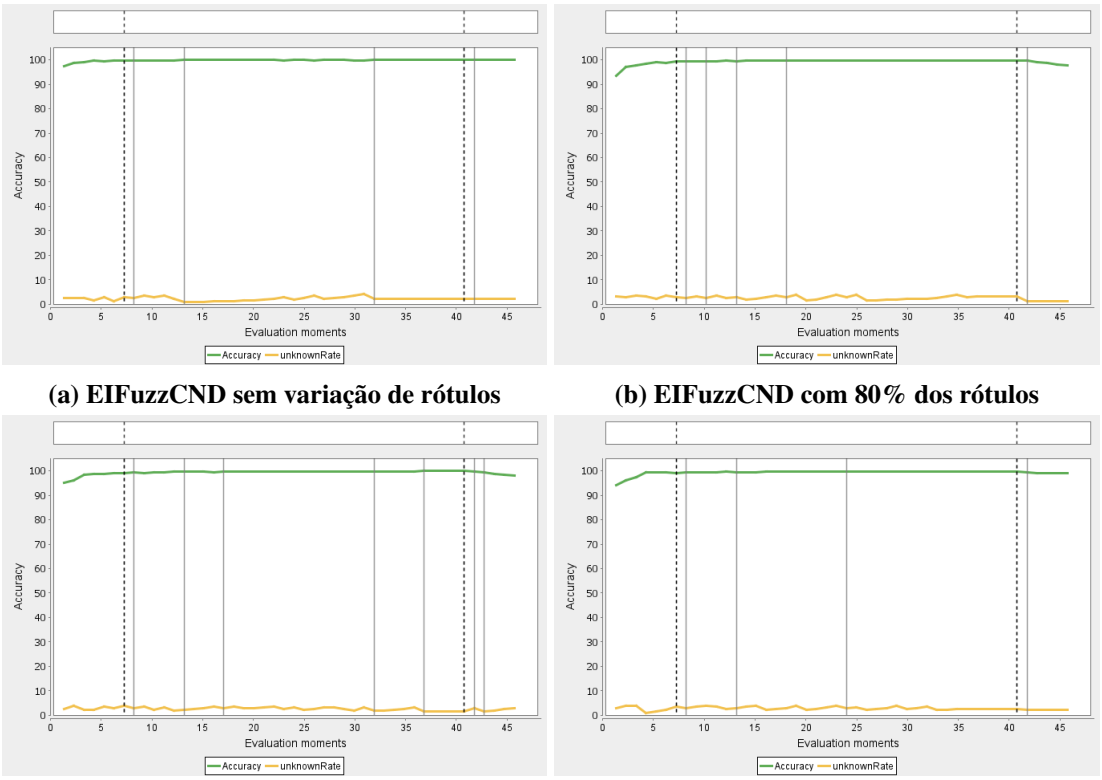
Figura 6.28: Acurácia e taxa de desconhecidos do EIFuzzCND, considerando latência intermediária de 10000 com variação de rótulos, para o conjunto de dados MOA.

No entanto, é importante destacar que houve uma exceção no valor percentual de 50% presente na Figura 6.28c. Nesse caso, observamos um maior número de detecção de padrões de novidade além do necessário em comparação aos outros valores percentuais. Essa discrepância pode ser atribuída à ausência dos rótulos que ajudam a identificar a quantidade de padrões de novidades presentes. Sem essa informação, o algoritmo pode ter interpretado erroneamente alguns padrões como novidades, resultando em uma taxa mais alta de detecção de padrões de novidade. Portanto, a falta de rótulos pode ter influenciado nessa diferença de comportamento do EIFuzzCND em relação aos diferentes valores percentuais.

Em resumo, o EIFuzzCND demonstrou ser robusto e adaptável aos diferentes valores percentuais de presença dos rótulos no dataset MOA. No entanto, a ausência de rótulos pode afetar a capacidade do algoritmo em distinguir corretamente os padrões de novidade, o que pode levar a um maior número de detecções mesmo que isso não afete diretamente a acurácia.

6.3.2 RBF

Ao analisarmos a Figura 6.29 referente ao dataset RBF no cenário de latência intermediária com variação dos valores percentuais de presença dos rótulos, podemos observar um comportamento consistente do algoritmo EIFuzzCND. Assim como no dataset MOA, o EIFuzzCND apresentou estabilidade em relação aos diferentes valores percentuais (20%, 50%, 80% e 100%) de presença dos rótulos. Essa estabilidade pode ser observada no valor de acurácia e taxa desconhecidos semelhantes independente da presença de rótulos.



(a) EIFuzzCND sem variação de rótulos

(b) EIFuzzCND com 80% dos rótulos

(c) EIFuzzCND com 50% dos rótulos

(d) EIFuzzCND com 20% dos rótulos

Figura 6.29: Acurácia e taxa de desconhecidos do EIFuzzCND, considerando latência intermediária de 10000 com variação de rótulos, para o conjunto de dados RBF.

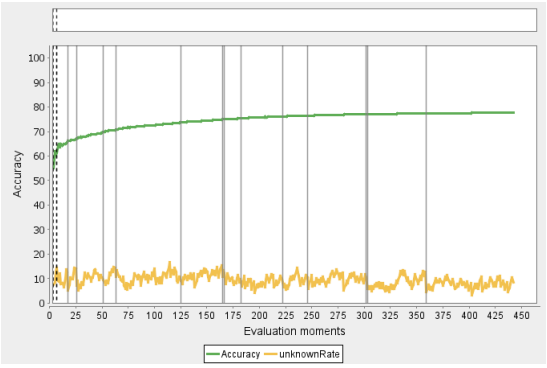
Ao analisar o comportamento do algoritmo EIFuzzCND no dataset RBF em diferentes valores percentuais de presença dos rótulos, observamos uma peculiaridade interessante. O algoritmo apresentou um desempenho superior nos dois extremos: quando todos os rótulos verdadeiros estavam presentes 100% e quando apenas 20% dos rótulos estavam presentes. No entanto, nos cenários intermediários (50% e 80%), o desempenho do algoritmo foi menos consistente.

Uma possível explicação para esse padrão de desempenho é que, com a total presença dos rótulos verdadeiros, o algoritmo possui informações mais precisas e completas sobre as classes

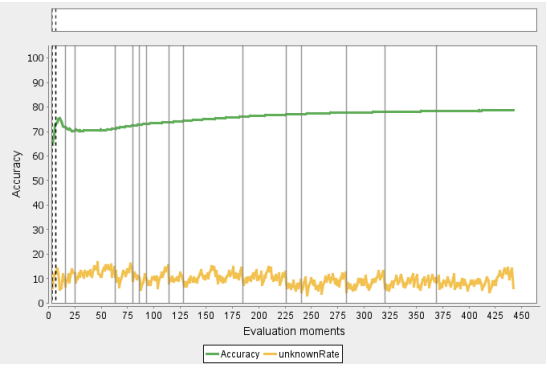
conhecidas, o que facilita a identificação dos padrões normais e reduz a detecção de falsos positivos. Por outro lado, quando apenas 20% dos rótulos estão presentes, o algoritmo pode se concentrar em identificar as instâncias mais representativas e distintas, o que pode resultar em um melhor desempenho na detecção de novidades.

6.3.3 KDD99

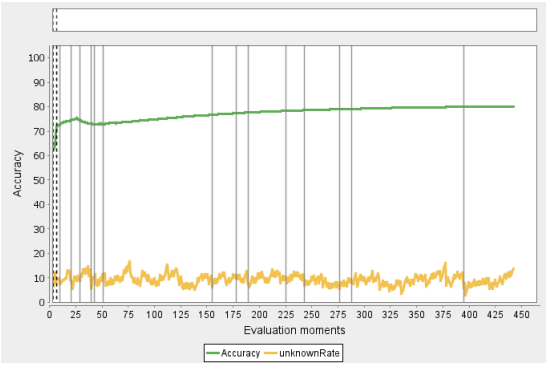
Ao analisar o comportamento do algoritmo EIFuzzCND no dataset KDD em diferentes valores percentuais de presença dos rótulos,presente na Figura 6.30 podemos observar algumas características. Nos cenários com 20% e 80% de presença dos rótulos, houve uma pequena variação na acurácia nos primeiros momentos de avaliação. Isso pode ser explicado pelo fato de que, nesses cenários, o algoritmo precisa adaptar-se às mudanças na distribuição dos rótulos e ajustar seus parâmetros para acomodar as novas instâncias. Essa adaptação inicial pode levar a uma pequena redução temporária na acurácia até que o algoritmo se ajuste adequadamente aos padrões de novidade.



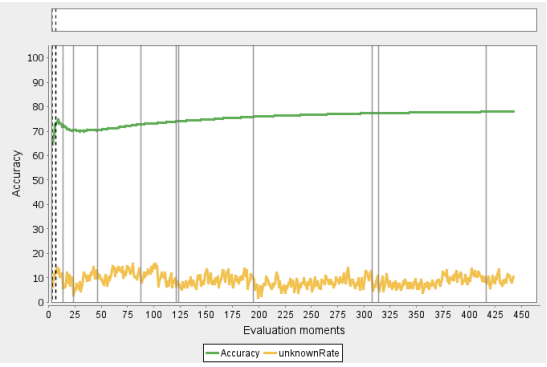
(a) EIFuzzCND sem variação de rótulos



(b) EIFuzzCND com 80% dos rótulos



(c) EIFuzzCND com 50% dos rótulos



(d) EIFuzzCND com 20% dos rótulos

Figura 6.30: Acurácia e taxa de desconhecidos do EIFuzzCND, considerando latência intermediária de 10000 com variação de rótulos, para o conjunto de dados KDD.

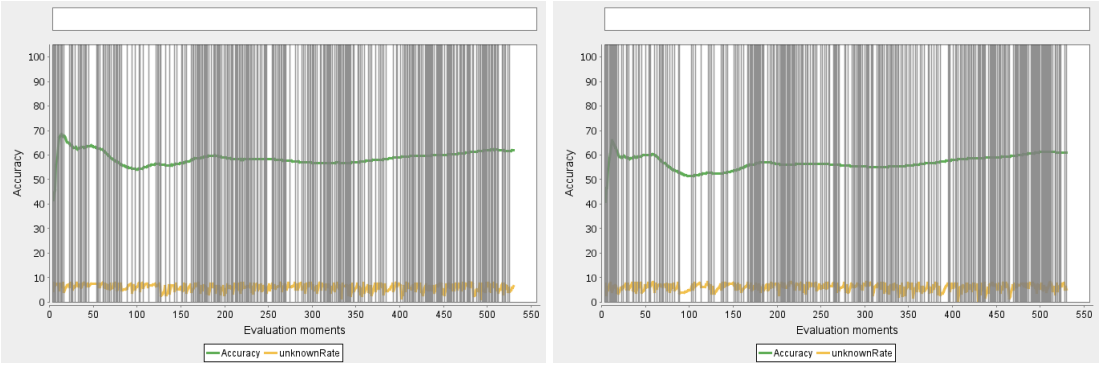
No cenário com 50% de presença dos rótulos representado na Figura 6.30d, também obser-

vamos uma pequena variação nos primeiros momentos de avaliação. No entanto, é importante destacar que esse cenário apresentou a acurácia mais estável ao longo do tempo. Isso indica que o algoritmo conseguiu lidar de forma mais eficiente com as variações nos rótulos, mantendo um desempenho consistente na classificação correta das instâncias.

Além disso, notamos diferenças nos momentos de detecção do padrão de novidade entre os diferentes cenários. Essa diferença nos momentos de detecção reflete a sensibilidade do algoritmo à variação na presença dos rótulos e destaca a importância de ajustar corretamente os parâmetros do EIFuzzCND para cada cenário específico.

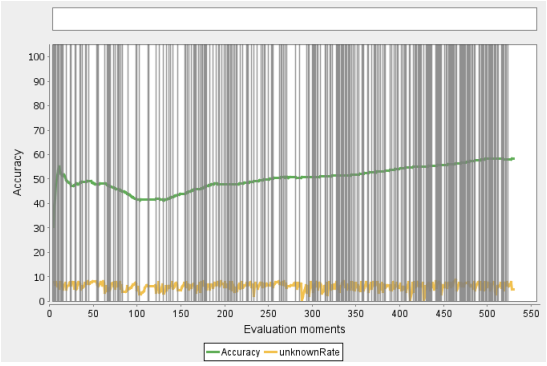
6.3.4 CoverType

Ao analisarmos o desempenho do algoritmo EIFuzzCND no dataset CoverType representado na Figura 6.31, confirmamos as observações feitas anteriormente de que ele não apresentou um bom desempenho em relação a esse conjunto de dados específico. No cenário de 50% de presença dos rótulos, notamos que a acurácia do algoritmo chegou a valores abaixo de 50%, o que indica um desempenho insatisfatório na classificação correta das instâncias.

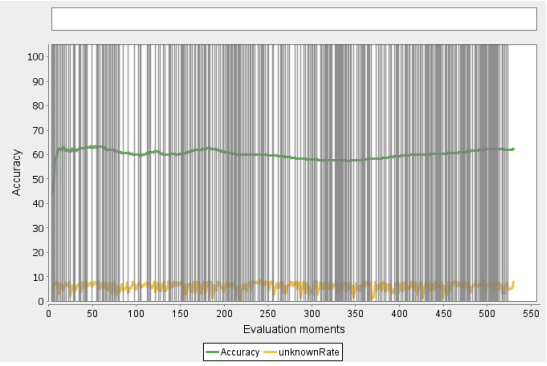


(a) EIFuzzCND sem variação de rótulos

(b) EIFuzzCND com 80% dos rótulos



(c) EIFuzzCND com 50% dos rótulos



(d) EIFuzzCND com 20% dos rótulos

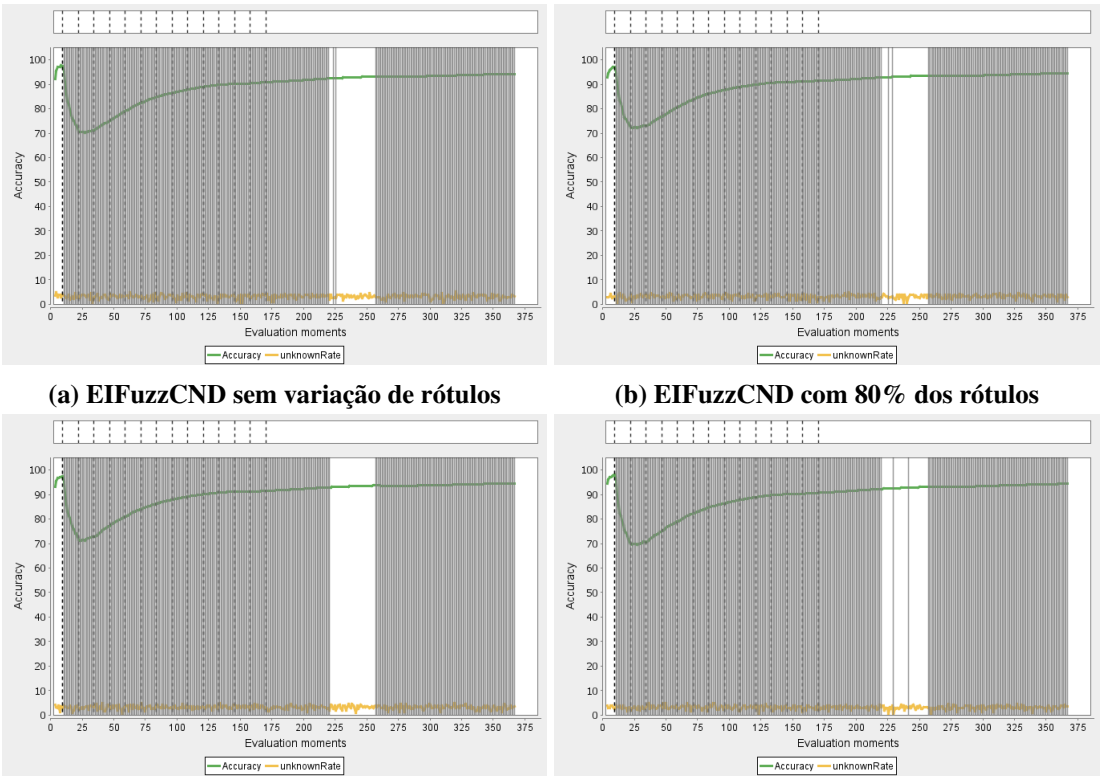
Figura 6.31: Acurácia e taxa de desconhecidos do EIFuzzCND, considerando latência intermediária de 10000 com variação de rótulos, para o conjunto de dados CoverType.

Uma explicação possível é que as características intrínsecas desse conjunto de dados não se adéquem bem à abordagem de clusterização fuzzy utilizada pelo algoritmo. Pode haver uma falta de representatividade dos padrões de novidade no espaço de atributos considerado ou uma dificuldade em definir limites claros entre as diferentes classes e as instâncias desconhecidas.

Para melhorar o desempenho do EIFuzzCND no dataset Cover, algumas estratégias podem ser adotadas. Uma opção seria explorar técnicas de pré-processamento dos dados, como a seleção ou extração de características relevantes, de forma a melhorar a separação entre as classes conhecidas e as instâncias desconhecidas.

6.3.5 SynEDC

Ao analisarmos o desempenho do algoritmo EIFuzzCND no dataset SynEDC representado na Figura 6.32, observamos uma consistência nos resultados em relação aos diferentes valores de variação dos rótulos. Em todas as variações, tanto na taxa de acurácia quanto na taxa de desconhecidos e no número de detecção de padrões de novidade, o dataset SynEDC apresentou resultados muito similares.



(a) EIFuzzCND sem variação de rótulos

(b) EIFuzzCND com 80% dos rótulos

(c) EIFuzzCND com 50% dos rótulos

(d) EIFuzzCND com 20% dos rótulos

Figura 6.32: Acurácia e taxa de desconhecidos do EIFuzzCND, considerando latência intermediária de 10000 com variação de rótulos, para o conjunto de dados SynEDC.

Embora o algoritmo não tenha alcançado um comportamento ideal no que diz respeito ao número de detecção de padrões de novidade, identificando mais do que o necessário, é importante ressaltar que ele obteve uma boa taxa de acurácia.

Em resumo, o dataset SynEDC apresentou resultados consistentes e similares em todas as variações testadas, com boa taxa de acurácia, apesar de um maior número de detecção de padrões de novidade do que o necessário. Esses resultados fornecem insights valiosos para aprimorar o desempenho do algoritmo EIFuzzCND.

Capítulo 7

CONCLUSÃO

No último capítulo, concluímos nossa análise e avaliação do algoritmo proposto EIFuzzCND com os algoritmos de detecção de novidades aplicados a diferentes conjuntos de dados e em diferentes cenários.

7.1 Contribuições

A principal contribuição deste trabalho foi a implementação e avaliação do algoritmo EIFuzzCND, um algoritmo de classificação multiclasse e detecção de novidades incremental baseado em agrupamento fuzzy e possibilístico fuzzy. A atualização incremental da estrutura de sumarização é uma forma de tratar a mudança de conceito e permite que o modelo se adapte rapidamente a essas mudanças no fluxo de dados.

A implementação desse algoritmo permitiu explorar suas capacidades de detecção de novidades em diferentes conjuntos de dados e cenários.

Ao comparar o desempenho do EIFuzzCND com outros algoritmos de detecção de novidades, como MINAS e EFuzzCND, observamos resultados promissores. Em cenários de latência extrema, o EIFuzzCND superou o MINAS e o EFuzzCND em métricas como taxa de acurácia e taxa de desconhecidos. No cenário de latência intermediária, também obtivemos bons resultados ao comparar com o ECSMiner e EFuzzCND.

7.1.1 Cenário de Latência Extrema

- **Taxa de Acurácia**: O EIFuzzCND tende a superar o MINAS e o EFuzzCND em termos de taxa de acurácia em cenários de latência extrema. Isso significa que, quando há um grande atraso

na disponibilidade dos rótulos verdadeiros, o EIFuzzCND mantém uma taxa de classificação correta mais alta em comparação com esses algoritmos.

- **Taxa de Desconhecidos**: O EIFuzzCND também demonstrou um bom desempenho na gestão de instâncias desconhecidas, mantendo uma taxa de desconhecidos relativamente controlada em cenários de latência extrema. Isso é importante porque minimiza a ocorrência de falsos desconhecidos e maximiza a capacidade do algoritmo em identificar corretamente instâncias anômalas.

7.1.2 Cenário de Latência Intermediária

- **Detecção de Novidades**: No cenário de latência intermediária, o EIFuzzCND demonstrou uma capacidade promissora de detecção de novidades, especialmente quando comparado com o ECSMiner e o EFuzzCND. Isso significa que o EIFuzzCND é eficaz em identificar padrões novos ou desconhecidos no fluxo de dados, mesmo quando há um atraso moderado na disponibilidade dos rótulos verdadeiros.

- **Estabilidade**: O EIFuzzCND mostrou estabilidade no desempenho, mesmo quando os valores percentuais de presença dos rótulos variavam. Isso é indicativo de sua capacidade de lidar com diferentes níveis de disponibilidade de rótulos e manter um desempenho consistente ao longo do tempo.

Em resumo, o EIFuzzCND destaca-se por sua capacidade de adaptação rápida a mudanças de conceito, sua eficácia na detecção de novidades e sua estabilidade em diferentes cenários, especialmente em cenários de latência extrema e latência intermediária. Suas principais vantagens incluem taxas de acurácia competitivas e uma boa gestão de instâncias desconhecidas, tornando-o uma opção promissora para a detecção de novidades em fluxos de dados complexos e dinâmicos.

Uma das contribuições adicionais deste trabalho foi a implementação da matriz de confusão incremental (MCI). Essa matriz permitiu uma análise mais detalhada e um melhor entendimento do desempenho e funcionamento do algoritmo EIFuzzCND. Através da MCI, foi possível identificar padrões de classificação e novidades ao longo do fluxo de dados, fornecendo insights valiosos para a melhoria do algoritmo e a interpretação dos resultados.

Além disso, este trabalho destacou os desafios enfrentados pelos algoritmos de detecção de novidades, como a presença de sobreposição entre classes, desbalanceamento e alta dimensionalidade nos conjuntos de dados analisados. Essas características específicas dos conjuntos de dados podem afetar a precisão e o desempenho dos algoritmos e, portanto, requerem estratégias

adaptativas e abordagens complementares para lidar com esses desafios.

Outra contribuição importante deste estudo foi a análise do desempenho do EIFuzzCND em cenários de latência intermediária com variação dos valores percentuais de presença dos rótulos (20%, 50%, 80% e 100%). Isso permitiu avaliar a robustez e a estabilidade do algoritmo em diferentes condições de disponibilidade de rótulos, fornecendo insights adicionais sobre sua capacidade de adaptação a diferentes contextos de fluxo de dados.

7.2 Considerações Finais

Com base nos resultados e análises realizadas, fica evidente que o algoritmo EIFuzzCND possui potencial para a detecção de novidades em fluxos de dados complexos. No entanto, há várias oportunidades de pesquisa que podem ser exploradas como trabalhos futuros:

1. **Melhorias no Tratamento de Classes Desbalanceadas:** Uma área de pesquisa interessante seria investigar estratégias específicas para lidar com conjuntos de dados desbalanceados, que são comuns em cenários reais. Isso poderia envolver o desenvolvimento de técnicas de adaptação dinâmica de limiar ou o uso de abordagens de amostragem para equilibrar a detecção de novidades em diferentes classes.

2. **Métricas Específicas para Detecção de Novidades:** A criação de métricas de avaliação específicas para a detecção de novidades em fluxos de dados poderia fornecer uma maneira mais precisa de avaliar o desempenho do EIFuzzCND e de outros algoritmos nesse contexto. Isso poderia incluir métricas que consideram a sensibilidade à taxa de falsos positivos, a capacidade de detecção de novidades raras e a adaptação a mudanças de conceito.

3. **Estudo de Casos em Aplicações Práticas:** Realizar estudos de casos em aplicações do mundo real, como detecção de fraudes financeiras, monitoramento de redes sociais ou sistemas de segurança cibernética, permitiria avaliar o desempenho do EIFuzzCND em cenários mais específicos. Isso também abriria a possibilidade de ajustar o algoritmo para atender às necessidades de domínios específicos.

4. **Comparação com Outros Algoritmos de Referência:** Realizar comparações mais abrangentes com outros algoritmos de detecção de novidades em fluxos de dados, incluindo abordagens baseadas em deep learning e métodos ensemble, poderia ajudar a posicionar o EIFuzzCND em relação ao estado da arte e identificar suas vantagens e limitações.

5. **Avaliação em Outras Configurações de Fluxo de Dados:** Além dos cenários de latência extrema e intermediária, seria interessante avaliar o desempenho do EIFuzzCND em configurações de fluxo de dados com diferentes características, como cenários de latência nula.

6. **Exploração de Estratégias de Adaptação de Parâmetros:** Investigar estratégias de adaptação de parâmetros do EIFuzzCND, como a índice de similaridade (ϕ), de acordo com as condições do fluxo de dados, pode ajudar a otimizar ainda mais seu desempenho em cenários dinâmicos.

Espera-se que essas sugestões de trabalhos futuros contribuam para o avanço da pesquisa em detecção de novidades em fluxos de dados e para o desenvolvimento contínuo do EIFuzzCND como uma ferramenta eficaz para lidar com problemas desafiadores nesse campo.

REFERÊNCIAS

ACKERMANN, M. R. et al. Streamkm++ a clustering algorithm for data streams. *Journal of Experimental Algorithmics (JEA)*, ACM New York, NY, USA, v. 17, p. 2–1, 2012.

AGGARWAL, C. C. An introduction to outlier analysis. In: *Outlier analysis*. [S.l.]: Springer, 2017. p. 1–34.

AGGARWAL, C. C. et al. A framework for clustering evolving data streams. In: ELSEVIER. *Proceedings 2003 VLDB conference*. [S.l.], 2003. p. 81–92.

AGRAWAL, R. et al. Automatic subspace clustering of high dimensional data for data mining applications. In: *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*. [S.l.: s.n.], 1998. p. 94–105.

AMINI, A. et al. A study of density-grid based clustering algorithms on data streams. In: IEEE. *2011 Eighth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*. [S.l.], 2011. v. 3, p. 1652–1656.

ANGELOV, P. P.; ZHOU, X. Evolving fuzzy-rule-based classifiers from data streams. *Ieee transactions on fuzzy systems*, IEEE, v. 16, n. 6, p. 1462–1475, 2008.

ARASU, A. et al. Stream: the stanford stream data manager (demonstration description). In: *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. [S.l.: s.n.], 2003. p. 665–665.

BEZDEK, J. C. *Pattern recognition with fuzzy objective function algorithms*. [S.l.]: Springer Science & Business Media, 2013.

BEZDEK, J. C.; EHRLICH, R.; FULL, W. Fcm: The fuzzy c-means clustering algorithm. *Computers & geosciences*, Elsevier, v. 10, n. 2-3, p. 191–203, 1984.

BIFET, A. et al. Moa: Massive online analysis, a framework for stream classification and clustering. In: PMLR. *Proceedings of the first workshop on applications of pattern analysis*. 2010. p. 44–50. Disponível em: <https://github.com/Waikato/moa>.

BIRCH, Z. An efficient data clustering method for very large databases. In: *Proceedings of the 1996 ACM SIGMOD international conference on management of data (SIGMOD'96). ACM, New York*. [S.l.: s.n.], 1996. p. 103–114.

CAI, X.-Q. et al. Nearest neighbor ensembles: An effective method for difficult problems in streaming classification with emerging new classes. In: IEEE. *2019 IEEE International Conference on Data Mining (ICDM)*. [S.l.], 2019. p. 970–975.

CARDOSO, D. O.; FRANÇA, F. M.; GAMA, J. Wcds: A two-phase weightless neural system for data stream clustering. *New Generation Computing*, Springer, v. 35, n. 4, p. 391–416, 2017.

CHANDOLA, V.; BANERJEE, A.; KUMAR, V. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, ACM New York, NY, USA, v. 41, n. 3, p. 1–58, 2009.

CRISTIANI, A. L.; CAMARGO, H. de A. A fuzzy multi-class novelty detector for data streams under intermediate latency. In: IEEE. *2021 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. [S.l.], 2021. p. 1–6.

CUP, K. *KDD Cup 1999 Data*. 1999. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

DOMINGOS, P.; HULTEN, G. Mining high-speed data streams. In: *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. [S.l.: s.n.], 2000. p. 71–80.

DONG, Y.; JAPKOWICZ, N. Threaded ensembles of autoencoders for stream learning. *Computational Intelligence*, Wiley Online Library, v. 34, n. 1, p. 261–281, 2018.

ESTER, M. et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In: *kdd*. [S.l.: s.n.], 1996. v. 96, n. 34, p. 226–231.

FARIA, E. R. et al. Novelty detection in data streams. *Artificial Intelligence Review*, Springer, v. 45, n. 2, p. 235–269, 2016.

FARIA, E. R. de et al. Minas: multiclass learning algorithm for novelty detection in data streams. *Data mining and knowledge discovery*, Springer, v. 30, n. 3, p. 640–680, 2016.

FARIA, E. R. de et al. Evaluation of multiclass novelty detection algorithms for data streams. *IEEE Transactions on Knowledge and Data Engineering*, IEEE, v. 27, n. 11, p. 2961–2973, 2015.

GAMA, J. *Knowledge discovery from data streams*. [S.l.]: CRC Press, 2010.

GAMA, J. et al. A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, ACM New York, NY, USA, v. 46, n. 4, p. 1–37, 2014.

GHAHRAMANI, Z. Unsupervised learning. In: SPRINGER. *Summer school on machine learning*. [S.l.], 2003. p. 72–112.

HEIGL, M. et al. Unsupervised feature selection for outlier detection on streaming data to enhance network security. *Applied Sciences*, MDPI, v. 11, n. 24, p. 12073, 2021.

HULTEN, G.; SPENCER, L.; DOMINGOS, P. Mining time-changing data streams. In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. [S.l.: s.n.], 2001. p. 97–106.

KOTSIANTIS, S.; KANELLOPOULOS, D. Association rules mining: A recent overview. *GESTS International Transactions on Computer Science and Engineering*, v. 32, n. 1, p. 71–82, 2006.

KOTSIANTIS, S. B. et al. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, Amsterdam, v. 160, n. 1, p. 3–24, 2007.

KRISHNAPURAM, R.; KELLER, J. M. A possibilistic approach to clustering. *IEEE transactions on fuzzy systems*, IEEE, v. 1, n. 2, p. 98–110, 1993.

LIKAS, A.; VLASSIS, N.; VERBEEK, J. J. The global k-means clustering algorithm. *Pattern recognition*, Elsevier, v. 36, n. 2, p. 451–461, 2003.

LOPES, P. de A.; CAMARGO, H. de A. Fuzzstream: Fuzzy data stream clustering based on the online-offline framework. In: IEEE. *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. [S.l.], 2017. p. 1–6.

MARKOU, M.; SINGH, S. Novelty detection: a review—part 1: statistical approaches. *Signal processing*, Elsevier, v. 83, n. 12, p. 2481–2497, 2003.

MASUD, M. et al. Classification and novel class detection in concept-drifting data streams under time constraints. *IEEE Transactions on Knowledge and Data Engineering*, IEEE, v. 23, n. 6, p. 859–874, 2010.

MCCARTHY, J. et al. A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955. *AI magazine*, v. 27, n. 4, p. 12–12, 2006.

MITCHELL, T. M. *Machine Learning*. New York: McGraw-Hill, 1997. ISBN 978-0-07-042807-2.

MURTAGH, F.; CONTRERAS, P. Algorithms for hierarchical clustering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, Wiley Online Library, v. 2, n. 1, p. 86–97, 2012.

NGUYEN, H.-L.; WOON, Y.-K.; NG, W.-K. A survey on data stream clustering and classification. *Knowledge and information systems*, Springer, v. 45, n. 3, p. 535–569, 2015.

PAL, N. R. et al. A possibilistic fuzzy c-means clustering algorithm. *IEEE transactions on fuzzy systems*, IEEE, v. 13, n. 4, p. 517–530, 2005.

PIMENTEL, M. A. et al. A review of novelty detection. *Signal processing*, Elsevier, v. 99, p. 215–249, 2014.

SETHI, T. S.; KANTARDZIC, M.; HU, H. A grid density based framework for classifying streaming data in the presence of concept drift. *Journal of Intelligent Information Systems*, Springer, v. 46, n. 1, p. 179–211, 2016.

SILVA, J. A. et al. Data stream clustering: A survey. *ACM Computing Surveys (CSUR)*, ACM New York, NY, USA, v. 46, n. 1, p. 1–31, 2013.

SILVA, T. P. da; CAMARGO, H. de A. Possibilistic approach for novelty detection in data streams. In: IEEE. *2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. [S.l.], 2020. p. 1–8.

SILVA, T. P. da et al. A fuzzy multiclass novelty detector for data streams. In: IEEE. *2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. [S.l.], 2018. p. 1–8.

SONG, Y.-Y.; YING, L. Decision tree methods: applications for classification and prediction. *Shanghai archives of psychiatry*, Shanghai Mental Health Center, v. 27, n. 2, p. 130, 2015.

SOUZA, V.; PINHO, T.; BATISTA, G. Evaluating stream classifiers with delayed labels information. In: IEEE. *2018 7th Brazilian Conference on Intelligent Systems (BRACIS)*. [S.l.], 2018. p. 408–413.

UCI. *UCI Machine Learning Repository: Covertype Data Set*. 1998. https://archive.ics.uci.edu/ml/datasets/covertype.