UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FUZZY APPROACH FOR CLASSIFICATION AND NOVELTY DETECTION IN DATA STREAMS

ANDRÉ LUIS CRISTIANI

ORIENTADORA: PROFA. DRA. HELOISA DE ARRUDA CAMARGO

São Carlos – SP

Fevereiro, 2022

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FUZZY APPROACH FOR CLASSIFICATION AND NOVELTY DETECTION IN DATA STREAMS

ANDRÉ LUIS CRISTIANI

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: Metodologia e Técnicas da Computação.

Orientadora: Profa. Dra. Heloisa de Arruda Camargo

São Carlos – SP

Fevereiro, 2022

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Defesa de Dissertação de Mestrado do candidato André Luis Cristiani, realizada em 11/02/2022.

Comissão Julgadora:

Profa. Dra. Heloisa de Arruda Camargo (UFSCar)

Prof. Dr. Ricardo Marcondes Marcacini (USP)

Prof. Dr. Ricardo Cerri (UFSCar)

O Relatório de Defesa assinado pelos membros da Comissão Julgadora encontra-se arquivado junto ao Programa de Pós-Graduação em Ciência da Computação.

Dedico este trabalho aos meus pais, irmãos, noiva, familiares, amigos e professores, estes que nunca mediram esforços para que o mesmo fosse realizado com sucesso.

AGRADECIMENTOS

Agradeço primeiramente à Deus, por ter me contemplado com o dom da vida e por ser sempre a minha luz em meio à tanta escuridão.

Aos meus familiares, que nunca mediram esforços para me ajudar, me trazer confiança e incentivo. Especialmente meus pais, Aparecido e Célia, por todos os ensinamentos, amor e carinho, que fizeram com que eu me tornasse o ser humano que sou hoje. Agradeço imensamente todo o apoio dado nos momentos incertos, esse apoio foi fundamental para mais essa conquista. Vocês sempre serão meu porto seguro.

Ao amor da minha vida, minha noiva, Carol, por todo amor, carinho e atenção que sempre me proporciona. Se não fosse você, provavelmente essa conquista não existiria. Obrigado por sempre me apoiar em todas as minhas decisões e, principalmente, me incentivar em trocar nosso tempo livre juntos, por horas de estudos e revisões de textos, tudo isso foi fundamental.

Ao meu fiel companheiro de caminhada, por todos os momentos alegres que me proporcionou ao longo desse percurso. Nos dias difíceis, te levar passear foi sempre um momento de descontração e conforto para mim. Se não fossem esses momentos, com a mente vazia e pensando sozinho, enquanto andávamos, provavelmente a ideia desse trabalho não teria surgido. Mesmo sabendo que você nunca vai ler, sou eternamente grato por tudo, Fera!

À minha orientadora, Profª. Dra. Heloisa de Arruda Camargo, que sempre demonstrou muita atenção, competência e paciência para sanar todas as minhas dúvidas e dificuldades ao longo deste trabalho. Sem o seu auxílio e dedicação, nada disso seria possível.

Ao grupo de pesquisa CIG, que sempre me proporcionou debates incríveis que agregaram muito para minha formação pessoal e profissional. Em especial, ao Tiago Pinho da Silva, por ter dedicado tempo me explicando características e funcionalidades do PFuzzND, etapa que foi fundamental para o desenvolvimento do EFuzzCND.

À todos os professores que passaram por minha vida. Porém, em especial, dois professores que me introduziram no mundo da pesquisa e sempre me incentivaram. Muito obrigado Prof. Dr. Rodolfo Ipólito Meneguette e Prof. Me. Douglas Dias Lieira por terem me ajudado a dar os primeiros passos ainda na graduação e por seguir sendo grandes parceiros de pesquisa até os dias de hoje.

À CAPES, que me possibilitou dedicação exclusiva à preparação deste trabalho ao longo

de 18 meses.

Ao Bangkok (Invillia e PagSeguro), que sempre me deixaram muito à vontade durante meus horários de trabalho, permitindo conciliar perfeitamente trabalho e estudos. Além disso, pela torcida, palavras de motivação e apoio ao longo desse percurso, vocês são feras!

Por fim, a todos os companheiros de pesquisa, profissionais e alunos do Departamento de Computação da UFSCar, pelo convívio que tivemos ao longo de 2019. Obrigado por todas as conversas, troca de ideias e parcerias. Em momentos de desânimo e desespero, sempre aparecia alguém para me trazer conforto e tranquilidade.

"Os rios não bebem sua própria água; as árvores não comem seus próprios frutos. O sol não brilha para si mesmo; e as flores não espalham sua fragrância para si. Viver para os outros é uma regra da natureza. A vida é boa quando você está feliz; mas a vida é muito melhor quando os outros estão felizes por sua causa."

(Papa Francisco)

RESUMO

Aprendizado em fluxo contínuo de dados (FCD) é uma área de pesquisa que busca extrair conhecimentos de uma grande quantidade de dados gerados de maneira contínua em um curto espaço de tempo. A detecção de novidades (DN) é responsável por identificar o surgimento de novos conceitos e alterações em conceitos conhecidos. Os rótulos verdadeiros das instâncias podem ser utilizados para que os algoritmos se adaptem à evolução e mudança de conceito. O tempo entre a classificação de uma instância e a chegada de seu rótulo verdadeiro é denominado latência. Grande parte das aplicações consideram que esses rótulos verdadeiros nunca estarão disponíveis. Outras são mais otimistas e consideram que o rótulo verdadeiro estará disponível logo após a classificação da instância. Outra forma é considerar que, após um certo tempo, os rótulos verdadeiros estarão disponíveis, o que é aplicável em grande parte dos cenários do mundo real. A utilização de conceitos da teoria dos conjuntos fuzzy possibilita tornar o aprendizado adaptável a possíveis imprecisões nos dados. No entanto, poucas abordagens utilizam os conceitos da teoria dos conjuntos fuzzy e consideram latência intermediária para obtenção dos rótulos. Diante disso, este trabalho propõe um método para classificação de DN multiclasse em FCD para cenários de latência intermediária e extrema baseada nos algoritmos ECSMiner e PFuzzND. Os resultados obtidos mostram que o algoritmo proposto obteve boa acurácia na classificação e detecção de novidades multiclasse, classificando outliers que abordagens que utilizam agrupamento crisp não conseguiram classificar. Além disso, foram apresentadas melhorias em relação aos parâmetros de inicialização do algoritmo, que reduzem a complexidade de sua utilização, mantendo bons resultados.

Palavras-chave: fluxo contínuo de dados,detecção de novidades, latência intermediária, teoria dos conjuntos *fuzzy*.

ABSTRACT

Learning in data streams (DS) is a research area that seeks to extract knowledge from a large amount of continuously generated data in a short period of time. The novelty detection (ND) is responsible for identifying the emergence of new concepts and changes in known concepts. The true labels of the instances can be used so that the algorithms adapt to the concept evolution and concept drift. The time between the classification of an instance and the arrival of its true label is called latency. Most applications consider that these true labels will never be available. Others are more optimistic and assume that the true label will be available shortly after the instance has been classified. Another way is to consider that, after a certain time, the true labels will be available, which is applicable in most real-world scenarios. The use of concepts from fuzzy set theory makes it possible to make learning adaptable to possible inaccuracies in the data. However, few approaches use the concepts of fuzzy set theory and consider intermediate latency to obtain the labels. Therefore, this work proposes a method for classifying multiclass ND in DS for intermediate and extreme latency scenarios based on ECSMiner and PFuzzND algorithms. The results obtained show that the proposed algorithm obtained good accuracy in the classification and detection of multiclass novelties, classifying outliers that approaches that use crisp clustering were not able to classify. In addition, improvements were presented in relation to the algorithm initialization parameters, which reduce the complexity of its use, maintaining good results.

Keywords: data streams, novelty detection, intermediate latency, fuzzy set theory.

GLOSSARY

ML	Machine Learning
IOT	Internet of Things
DS	Data Streams
ND	Novelty Detection
SPFMiC	Supervised Possibilistic Micro-Cluster
MC	Micro-Cluster
FCM	Fuzzy C-Means
FRB	Fuzzy C-Means
BLM	Baseline Learning Model
CVC	Cohesion Validation Component
CLAM	Class-based Micro Classifier Ensemble
UnkR	Unknown Rate
NP	Novelty Pattern

LIST OF FIGURES

List of Tables

List of Algorithms

SUMMARY

Chapter 1
INTRODUCTION

For a long time, research and studies in the field of machine learning (ML) have focused on batch learning tasks, using small sized databases (GAMA, 2010). In this type of learning, a dataset is previously generated, considering its stationary distribution, and it is submitted to the induction process of the decision model that will be used to predict situations of this same task.

With the advances in technology and the popularization of the internet of things (IoT), the production of large-scale data has increased, influencing the need to obtain knowledge from this large volume of data. As this type of problem deals with an infinite amount of data, which possibly undergoes evolution over time, the use of batch learning techniques is inappropriate and can directly influence the performance of the classifier, requiring the development of new techniques to deal with the characteristics of this scenario. With that, a new field of studies in ML emerged, called data streams (DS). Several real-world scenarios deal with DS, such as: fraud analysis in credit card transaction, intrusion detection computer networks, data mining of social network, energy consumption prediction, and more.

DS is characterized by a sequence of infinite instances that arrive continuously and at high speed, in such a way that it is impossible to store the data in memory, which is why, it must be analyzed and discarded (BABCOCK et al., 2002). Over time, due to its non-stationary nature, the data may undergo changes in their distribution, phenomena called concept drift and concept evolution.

A concept drift occurs when known data undergoes changes in the values of its attributes, in such a way that a classifier trained with old data is ineffective in classifying new data. Concept evolution occurs when unknown concepts arise or known concepts disappear over time in the DS (MASUD et al., 2011).

An important characteristic for algorithms dealing with DS is to identify concept drift and concept evolution to be able to adapt their decision model. In order to supply this demand, incremental ML algorithms have been used, however, due to the non-stationary nature of the data, incremental algorithms are not enough to deal with this type of task (GAMA, 2010). Therefore, the algorithms must have mechanisms that make it possible to identify the concept evolution and

concept drift and the incorporation of new data, according to the current state of the DS. The ability of an algorithm to identify and distinguish changes in data is called novelty detection (ND) (GAMA, 2010).

The availability of true labels and the use of external feedback is fundamental to define how to update the decision model. The time between the classification of an instance and getting its true label is called latency. According to the literature, there are three types of latencies: when the label will never be available to the system, called extreme latency; when the label is available right after the instance classification, called null latency, and when the true label will be available after a certain time, after the instance classification, called intermediate latency.

1.1 Motivation and Justification

Traditional ML algorithms were developed to handle a pre-defined amount of data, considering that every dataset will be available during the training phase. However, with the advances in technology and the great increase in the volume of data production, ML in DS has become popular, increasing the studies and development of solutions regarding its main characteristics. Among the main fields of research in this area, ND stands out, which is important to deal with mutations in data distribution, emergence and disappearance of new classes, keeping the decision model up to date, according to these changes.

Given the large volume of data, reconciled with the high speed of their arrival, storing all data in memory becomes unfeasible. ML algorithms in DS must process the instance in a single pass and then discard it. For this, several works in the literature (MASUD et al., 2011; AL-KHATEEB et al., 2012; FARIA et al., 2016a; LOPES; CAMARGO, 2017; Costa Júnior et al., 2019) use statistical summaries to store information regarding clusters of data that have similar characteristics, called micro-clusters. Micro-cluster have incremental and additive properties, allowing the addition of a new instance to the micro-cluster and the joining of two different micro-clusters, in addition to allow the calculation of basic measures that characterize a cluster, such as centroid and radius.

The use of *fuzzy* set theory is a way to make learning more, a way that allows groups to be seen with adjustments and a given clusters individually belongs a certain amount to all, can make it more adaptable to phenomena of change and evolution of concepts and the possible inaccuracies that the data may suffer along the DS. In order to apply the fuzzy set theory in the ND task in DS, the concept of fuzzy micro-clusters, called SPFMiC (*Supervised Possibilistic Fuzzy Micro-Cluster*), was introduced in the PFuzzND algorithm (SILVA, 2018), being a way to store statistical summaries that allow the calculation of the relevance fuzzy of new instances for each of the SPFMiC.

Many algorithms were developed considering scenarios in which information about the actual labels of the instances would never be available, updating their decision models based

on each classified instance, recalculating the statistics for each of the clusters. Such feature can reduce the performance of these algorithms in more critical environments or inaccurate data processing.

Another approach taken by various algorithms is to consider that the true label of an instance would be available right after its classification, even before the next instance is classified. Although these algorithms update the decision model with true information, the immediate labeling process is impracticable in most real-world scenarios.

Considering that after a certain amount of time, variable or not, the true label will be available can be a realistic way to update the decision model, since in many scenarios, after a while, the true label is made available as a natural consequence of the process of the specific scenario. Several real-world scenarios work this way and can be used in this approach, such as: forecasting energy consumption, analyzing fraudulent transactions, weather forecasting, etc.

This proposal is based on the ECSMiner (MASUD et al., 2011) algorithm, which is one of the few algorithms that consider intermediate latency to obtain the true labels of the instances, which classifies instances through an ensemble of decision trees. Furthermore, despite performing the ND, for this algorithm, any instance that can not be explained by the decision model is considered novelty, without distinguishing different types of novelty.

Therefore, the purpose of this work is to apply fuzzy set theory techniques in an algorithm originally developed for intermediate latency scenarios, but which can be applied in extreme latency scenarios. For this, two types of classifiers will be used, a primary classifier, which is trained in a supervised way, based on the true labels that arrive over time. And a secondary classifier, trained in an unsupervised way, used to identify instances that belong to the different Novelty Patterns (NP), identified by the algorithm.

Different from ECSMiner, the algorithm proposed in this work also deals with multi-class ND, identifying different novelties that may arise along the DS, such as the change of an existing class or the appearance of a new class.

1.2 Hypothesis and Objectives

The main objective of this work is to develop an algorithm for classification and ND in DS, to work under intermediate latency and also to enable its use under extreme latency, based on the ECSMiner (MASUD et al., 2011) algorithm. Unlike ECSMiner, the proposed algorithm will be developed by applying fuzzy clustering, aiming to provide greater flexibility for the classification and ND tasks.

The proposed algorithm introduces the concept of multi-class ND in environments with intermediate latency by means of identifying different NPs along the DS. In view of the main objective of this work, the central hypothesis can be stated:

- The use of supervised and unsupervised learning techniques in cooperation, together with fuzzy clustering, allow the identification of different NPs in continuous DS, reducing the unknown rate and producing comparable or superior results to methods that use only one learning technique and classical set theory.

We propose the use of two decision models, one supervised and the other unsupervised, to classify the instances. The supervised decision model is the main one and is used to store SPFMiCs that were identified through labeled data. The unsupervised decision model is secondary and is only triggered if the main marks the instance as unknown. It is used to store SPFMiCs identified through the ND method. These SPFMiCs are removed from the decision model when the true label equivalent to them arrives.

The algorithm considers that all classified instances will arrive at some point with the true label, in the same sequence in which they were classified and considering a fixed latency value. The algorithm handles incremental, gradual and recurring concept drift. In case of abrupt concept drift, the algorithm cannot associate with any known class or NP and classifies it as novelty. In case of overlapping classes, the algorithm proved to be sensitive, misidentifying class extensions and classifying some wrong instances.

In the experiments performed, the proposed approach showed comparable and even superior results under intermediate latency and promising results under extreme latency. The results showed that, because of the flexibility granted by the use of fuzzy clustering, the proposed algorithm reduced the unknowns rate, in addition to significantly decreasing the time to start classifying instances belonging to the new class and start lowering the unknowns rate when a novelty comes up.

The main contribution of this work is a fuzzy approach, based on the ECSMiner and PFuzzND methods, which adopts the fuzzy micro-cluster summarization structure, called SPFMiC, to perform multiclass ND in DS. In addition, we reduced the initialization parameters used in the PFuzzND method, which proposed the SPFMiC. Finally, we present a study on the use of our approach under different latency values, showing the impacts that latency can have on algorithm performance.

1.3 Text organization

Chapter 2 - **Classic Machine Learning**: describes the main concepts about classical machine learning and its characteristics. It also presents details about the classical algorithms that were used to develop this proposal.

Chapter 3 - **Learning in Data Stream**: describes the main concepts about learning in DS, its characteristics and challenges, facts that supported the development of approaches in this

domain. Furthermore, it presents the main differences between classical machine learning and DS learning.

Chapter 4 - **Related Works**: presents an overview of the state-of-the-art approaches that deal with ND in DS. The works were divided among binary, multi-class and fuzzy approaches. In addition, each division was also split between approaches that deal with null, extreme and intermediate latency. At the end of this section a comparison of all approaches is presented.

Chapter 5 - **Enhanced Fuzzy Classifier for Multi-class Novelty Detection in Data Streams**: presents the context and motivation for the development of this work. The proposed approach is explained step-by-step in detail.

Chapter 6 - **Experimental Evaluation**: describes the set of experiments that were used to partially evaluate the proposed approach, together with the datasets, algorithms that were used for comparison, metrics of evaluation employed and the results obtained. In addition, it also presents metrics that will be used to evaluate the proposed approach.

Chapter 7 - **Results and analysis of experiments**: describes and comments on the experiments carried out, analyzing the results obtained under intermediate and extreme latency, pointing out the main characteristics of the algorithms used.

Chapter 8 - **Conclusions**: presents the final conclusions obtained with this work, its main contributions, limitations and guidelines for future work.

Chapter 2
CLASSIC MACHINE LEARNING

Machine Learning (ML) is an area of artificial intelligence that studies computational techniques for the construction of systems capable of acquiring knowledge automatically. There are several definitions for ML presented in the literature, one of the main definition is given in MITCHELL (1997), such as: *"The ability to improve performance in performing some task through experience"*.

In ML, systems have the ability to acquire knowledge through data analysis and perform tasks with high efficiency. These systems can analyze large volumes of data and find different types of patterns using statistical methods. Many tasks can be performed using ML, such as: fraud detection, image classification, recommendation systems, spam filtering, among others.

There are several methodologies that can be used to perform ML tasks and create computational models capable of learning and making decisions, such as: deduction, learning by habit, analogy and induction (MITCHELL et al., 2013). Among the methodologies presented, inductive inference is one of the main methodologies used in tasks involving ML (MITCHELL, 1997).

Computers are programmed to learn from a given experience, obtaining general conclusions based on a set of instances. In this context, each instance corresponds to a set of characteristics or attributes, of n-dimensional size, that describes its aspects in a given domain. Inductive inference is used to learn a hypothesis (for example, one or more rules) that is able to reach a correct conclusion, based on acquired prior knowledge (FACELI et al., 2011). Inductive learning can be divided into two approaches: supervised and unsupervised.

2.1 Supervised Learning

Supervised learning is based on the acquisition of knowledge through a set of instances that have an output attribute, also called target variable, whose values can be estimated from the rest of the attributes, called input attributes. The purpose of this type of learning is to extract knowledge from a subset of data, called training data, which is used to create a decision model

capable of relating input attribute values to known output attributes (FACELI et al., 2011). Supervised learning can be divided into two categories:

- **Classification:** Task where a label or class is assigned to an input data. For example: sorting whether the feeling of a textual message is positive or negative.
- **Regression:** Task where the mathematical relationship among dependent (or output variable) variable and independent variables (or input variables) is studied, where the values that are being predicted follow a continuous interval. For example: analyze the linear relationship between the change in wages and unemployment rate.

Several algorithms have been proposed to deal with classification and regression problems, such as: decision trees (QUINLAN, 1986), neural networks (BISHOP, 1995), statistical methods (DUDA, 1973), probabilistic methods (LANGLEY, 1993), optimization based methods (CRISTIANINI et al., 2000), evolutionary approaches (GOLDBERG, 1989) and distance-based (COVER; HART, 1967).

2.2 Unsupervised Learning

Unsupervised learning is based on acquiring knowledge through a set of instances that do not have an output attribute or any information on how those instances should be handled (ZHU; GOLDBERG, 2009). The purpose of this type of learning is to identify similarities in the characteristics of unlabeled instances, looking for patterns that categorize them and place them in similar groups. In this case, the best known algorithms rely on clustering techniques to identify patterns and classify data.

Clustering techniques seek to find structures of groups (clusters) in the dataset, in which the objects belonging to each cluster share similar characteristics for the problem domain (FACELI et al., 2011). The division of the instances in clusters is determined by a proximity measure, which can be either a similarity or dissimilarity measure. The choice of measure must take into account the types and scales of the attributes of the data that will be clustered.

There are several measures that can be used to check how similar or dissimilar the instances are to each other. The most common dissimilarity measure for continuous attributes is the Euclidean distance.

2.2.1 Clustering Techniques

There are several categories of algorithms that have been designed to handle unlabeled data clustering tasks. The main categories can be divided into (FACELI et al., 2011):

- **Hierarchical:** A hierarchical clustering algorithm is capable of generating a sequence of nested partitions, according to a certain criterion. Each of the partitions represents a clustering. An example of an algorithm in this category is OPTICS (Ordering Points To Identify the Clustering Structure) (ANKERST et al., 1999).

- **Partitional:** A partition clustering algorithm consists of creating an initial partition and iteratively move objects from one cluster to another, seeking to improve the clustering, according to a certain criterion. An example of an algorithm in this category is the K-means (MACQUEEN et al., 1967).

- **Density-Based:** A density-based clustering algorithm is able to obtain clusters arbitrarily. These algorithms consider that the clusters are formed by regions with high density of instances, divided by regions with low density. An example of an algorithm in this category is DBSCAN (Density-Based Spatial Clustering of Applications with Noise) (ESTER et al., 1996).

- **Graph-Based:** A graph-based clustering algorithm represents data in a proximity graph. Each node represents an instance and each edge represents the distance between the instances. Clustering methods remove inconsistent edges and create new ones, according to a certain criterion, aiming at the formation of clusters. An example of an algorithm in this category is the HSC (Highly Connected Subgraph) (HARTUV; SHAMIR, 2000).

- **Grid-based:** A grid-based clustering algorithm defines a grid to store the dataset information and performs all operations on that data structure. An example of an algorithm in this category is the CLIQUE (Clustering In QUEst) (AGRAWAL et al., 1998).

Among the categories and clustering algorithms presented, one of them stands out and is widely used: the partitional algorithm k-means (MACQUEEN et al., 1967). This algorithm divides the dataset into k clusters, where k is an user-supplied value. The dissimilarity measure used is the Euclidean distance and each cluster is represented by the centroid of the instances belonging to the cluster.

2.2.1.1 Fuzzy Clustering

According to Bezdek et al. (1984), hard clustering can impose many limitations, since, due to the imprecision of the data that can arise from different sources, it would be more appropriate if the instances could be assigned to more than one cluster.

In this sense, the fuzzy clustering category emerged in which the process gains more flexibility by allwing the instances to belong to different clusters with different membership degrees (BEZDEK, 1981). Bellow we describe the Fuzzy C-Means and Possibilistic-Fuzzy C-Means algorithms, which serve as a basis for the development of this work.

2.2.1.1.1 Fuzzy C-Means

Bezdek (1981) proposed the Fuzzy C-Means (FCM) algorithm as an extension of the k-means algorithm incorporating the concepts of fuzzy set theory. As in K-means, this algorithm seeks to minimize an objective function, defined as follows, to cluster N instances into C clusters:

$$J_m = \sum_{i=1}^{N}\sum_{j=1}^{C} u_{ij}^{m}||x_i - c_j||^2, 1 \leq m < \infty \tag{2.1}$$

where m represents the fuzzification parameter and can be any real number greater than 1, u represents the degree of membership of x_i in the cluster j, x_i represents the *i*-th element of the set of instances, c_j the centroid of the cluster and $||*||$ represents any norm that demonstrates the similarity between the analyzed data and the center of the cluster.

The FCM clustering must perform with the restrictions imposed by the equations 2.2 and 2.3. Equation 2.2 defines that the sum of the memberships of a given element x_i to all clusters must be 1.

$$\sum_{j=1}^{k} u_{ij} = 1 \tag{2.2}$$

Equation 2.3 defines that the sum of the memberships of the elements belonging to a certain j cluster must be smaller than the number of instances in the dataset and must contain at least one element with a degree of membership greater than 0.

$$0 < \sum_{i=1}^{N} u_{ij} < N \tag{2.3}$$

The fuzzy partitioning is performed through an iterative optimization of the Equation 2.1, considering the membership degree u_{ij} and the centroid of the cluster c_j, defined by :

$$u_{ij} = \frac{1}{\sum_{k=1}^{C}\left(\frac{||x_i - c_j||}{||x_i - c_k||}\right)^{\frac{2}{m-1}}} \tag{2.4}$$

$$c_j = \frac{\sum_{i=1}^{N} u_{ij}^{m} * x_i}{\sum_{i=1}^{N} u_{ij}^{m}} \tag{2.5}$$

There are two stopping criteria for the iteration of the FCM Equation 2.1. The first, similar to the k-means algorithm, ends the iteration when no element undergoes cluster change. The second occurs when the condition $max_{ij}(|u_{ij}^{(k+1)} - u_{ij}^{(k)}|) < \epsilon$ is satisfied.

2.2.1.1.2 Possibilistic-Fuzzy C-Means

Aiming to supply the need to calculate association and typicality values when clustering data, Pal et al. (1997) proposed the Fuzzy-Possibilistic C-Means (FPCM) algorithm. After, Pal et al. (2005) proposed the Possibilistic-Fuzzy C-Means (PFCM), being a hybrid algorithm between the Possibilistic C-Means (PCM) (Krishnapuram; Keller, 1993) and the Fuzzy C -Means (FMC), in order to solve some problems, such as: FCM noise sensitivity defect, PCM coincident clusters problem and eliminate FPCM row sum restrictions. The method seeks to minimize the objective function, defined as (Pal et al., 2005):

$$J = \sum_{i=1}^{c}\sum_{k=1}^{n}\{[a(u_{ik})^m + b(t_{ik})^\eta]d_{ik} + \pi_i(1-t_{ik})^\eta\} \tag{2.6}$$

where d_{ij} represents the Euclidean distance between the object k and the prototype of the cluster i, a and b represent the importance of the degree of membership and typicality for the clustering and these values are defined by the user, η refers to the importance of considering the typicality degrees t_{ik} for the clustering, π_i is used to balance the two terms of the equation and ensure that the criterion is calculated in order to maximize the values of t_{ik} and can be entered by the user. However, the authors suggest using the weighted average of the distances defined by:

$$\pi_i = K\frac{\sum_{k=1}^{n}(u_{ik})^m d_{ik}}{\sum_{k=1}^{n}(u_{ik})^m} \tag{2.7}$$

According to Pal et al. (2005), K usually takes the value 1. To perform the minimization of the objective function, the degree of relevance and typicality are used, which are calculated by:

$$v_i = \frac{\sum_{k=1}^{n}[a(u_{ik})^m + b(t_{ik})^\eta]x_{ik}}{\sum_{k=1}^{n}[a(u_{ik})^m + b(t_{ik})^\eta]} \tag{2.8}$$

Where v_i represents the prototype of the cluster i, and is summarized within d_{ik} in Equation 2.6. The membership degree is calculated under the constraint $\sum_{i=1}^{c} u_{ik} = 1$ and is defined by:

$$u_{ik} = [\sum_{h=1}^{c}(\frac{d_{ik}}{d_{hk}})^{\frac{2}{m-1}}]^{-1} \tag{2.9}$$

Finally, due to the fact that there is no sum limitation, the degree of typicality and relevance may have different values, in addition to the value of b, which directly impacts the value of t_{ik}. The greater the value of b, the more the value of t_{ik} will decrease and, consequently, the degree of typicality must decrease in relation to the distance. The degree of typicality is defined by:

$$t_{ik} = [1 + (b\frac{d_{ik}}{\pi_i})^{\frac{2}{(\eta-1)}}] \tag{2.10}$$

Chapter 3
DATA STREAM LEARNING

For many years, a large part of ML applications were intended to solve learning problems in batch mode, training their decision models, considering that the data distribution would be stationary and that the entire dataset would be available in the training phase (GAMA, 2010).

Currently, a large part of the problems are dynamic, generating large volumes of data in a short period of time and being able to change their distribution over time. According to Babcock et al. (2002), there are some main characteristics that differentiate DS learning from conventional learning:

- Instances arrive continuously;
- Data can arrive disorderly to be processed;
- Size is potentially infinite;
- Instances are discarded after being processed and its recovery is not easy, unless it is stored in a temporary memory, however, this memory has a much smaller size than the DS.

Another important feature of the DS is the possible occurrence of concept drift and concept evolution, caused by the alteration that the data distribution can undergo over time. With that known concepts can change, new concepts can emerge and familiar concepts can disappear. The algorithm must be able to keep up to date, always representing the current distribution of data, adapting to new concepts that arrive and forgetting old concepts that are no longer used (FARIA et al., 2016a).

Algorithms dealing with ML in DS must be able to process the data in real time as it becomes available and forget about outdated data. According to Domingos & Hulten (2001), these systems must have the following characteristics:

1. Incrementality;
2. Real-time Learning;

3. Ability to process data in constant time and using limited memory;
4. Limited access to already processed instances;
5. Ability to detect and update the decision model to changes in concept.

Many algorithms have been proposed to deal with the ML task in DS using different types of methodologies, such as: predictive learning (Gama et al. (2003), Jin & Agrawal (2003), Troyano et al. (2005), Masud et al. (2011)), descriptive learning (Zhang et al. (1996), Aggarwal et al. (2004), Rodrigues et al. (2006), Faria et al. (2016a), Silva & Camargo (2020)), association rules (Han et al. (2000), Giannella et al. (2003), Angelov & Zhou (2008)) and dimensionality reduction (SOUSA et al., 2006).

Learning in DS has been the subject of several studies, and has been applied in several areas, such as: fraud detection in credit card transactions (Wang et al. (2003), Wu et al. (2012), Arya & G (2020)), mining clicks in websites (Marin et al. (2013)), network traffic monitoring (Aggarwal & Philip (2008), Cristiani et al. (2020)) and web text mining (Nahar et al. (2014)).

With the great evolution of information technology and the popularization of the internet and connected mobile devices, the volume of generated data is increasing, in addition to emerging new scenarios for DS learning. Therefore, new algorithms must be studied to solve the different problems of these scenarios.

3.1 Data Stream Classification

Classification is an ML task that deals with the discovery of knowledge in datasets through the relationship of their characteristics. The decision model is trained with a set of labeled instances, in such a way that unlabeled instances can be predicted with these labels.

Classification techniques in batch scenarios consider that all class labels will be available during the training stage and that the data distribution will not change over time, which is unfeasible in DS environments.

According to Nguyen et al. (2015), two approaches can be used for the classification task in DS: incremental and two-phase. The incremental approach updates the decision model by computing each incoming labeled instance. Most of these classifiers use complex incremental equations to train and update the decision model (OLIVEIRA, 2016).

The two-phase approach divides the process into two steps (*offline*-online). In the first step, an initial subset of labeled DS data is processed and summarized. In the second step, whenever a user requests, an unlabeled instance can be classified based on the summarized data that was processed in the first step.

3.2 Data Stream Clustering

Clustering techniques in batch scenarios consider that all data will be available in memory, which is unfeasible in DS. The main difficulties in working with DS clustering are the amount of data that has been arriving continuously and infinitely and the urgency of analyzing it in real time. According to Gama (2010), the biggest clustering challenge in DS is maintaining a good cluster, using a small amount of memory.

To deal with the challenges mentioned above, clustering algorithms in DS must be incremental, maintaining cluster structures that allow their evolution along with the DS. Barbará (2002) cites four basic requirements for algorithms that work with clustering in DS: *i)* Representation capacity; *ii)* Fast and incremental processing of new instances; *iii)* Track and adapt to clustering changes and *iv)* Quick and clear identification of outliers.

Aggarwal et al. (2003) divides the clustering process in DS into two steps. The first step is responsible for creating micro-clusters, storing summary statistics for each cluster. The second step is responsible for creating the macro-clusters, using the statistical summaries to provide clusters for the user. This approach is very efficient as it provides the user the ability to track evolutions and aggregations among the clusters.

3.2.1 Feature Vector

According to Gama (2010), a powerful way to store cluster information in DS is the use of feature vectors proposed by Tian et al. (1996). The feature vector, also called micro-cluster (MC) in the literature, represents, in a compact way, a set of instances. A micro-cluster is responsible for storing the following statistics (*N, LS, SS*) of a cluster:

- *N* represents the number of instances belonging to the cluster;
- *LS* represents a vector of the same dimension of instances belonging to the cluster that stores the linear sum of the *N* instances;
- *SS* represents a vector of the same dimension of instances belonging to the cluster that stores the quadratic sum of the *N* instances;

The properties of a MC are (GAMA, 2010):

- **Incrementality**

  If an instance x is added to cluster A, the MC statistics are updated as follows:

$$LS_A \leftarrow LS_A + x$$

$$SS_A \leftarrow SS_A + x^2$$

$$N_A \leftarrow N_A + 1$$

- **Additivity**

  If A and B are separate clusters, joining them results in a cluster C defined by the sum of their parts. Two MCs can be merged as follows:

$$LS_C \leftarrow LS_A + LS_B$$

$$SS_C \leftarrow SS_A + SS_B$$

$$N_C \leftarrow N_A + N_B$$

A MC has enough information to calculate basic measures that characterize a cluster, such as:

- **Centroid**:

$$C = \frac{LS}{N} \tag{3.1}$$

- **Radius**:

$$R = \sqrt{\frac{\sum_1^N (x_i - X0)^2}{N}} \tag{3.2}$$

$$R = \sqrt{\frac{SS}{N} - \frac{LS^2}{N}} \tag{3.3}$$

In the literature, the best known method that is based on clustering and uses feature vectors to keep the information of clusters is CluStream (AGGARWAL et al., 2003). This algorithm defines the attribute vector keeping the information (*N, LS, SS*) of the original method (TIAN et al., 1996), adding two more attributes: $CF1^t$, representing the sum of the timestamps and $CF2^t$, representing the quadratic sum of timestamps. These new added attributes are responsible for temporal aspects of DS.

3.2.2 Supervised Possibilistic Fuzzy Micro-Cluster

The Supervised Possibilistic Fuzzy Micro-Cluster (SPFMiC) is a vector of attributes that represents a fuzzy micro-cluster (SILVA; CAMARGO, 2020). The main difference between SPFMiC and the other fuzzy attribute vectors is the ability to maintain statistics for typicality calculation. An SPFMiC uses the fuzzification parameters m and typicality n and is responsible for storing the following statistics $(M^e, T^e, \overline{CF1^e_\mu}, \overline{CF1^e_T}, SSD^e, N, t, class\_id)$ of a cluster:

- M^e represents the linear sum of the memberships of the instances in the micro-cluster raised to *m*, is defined by:

$$M^e = \sum_{j=1}^{N} \mu_j^m \tag{3.4}$$

- T^e represents the linear sum of the typicalities of the instances in the micro-cluster raised to n, it is defined by:

$$T^e = \sum_{j=1}^{N} T_j^\eta \tag{3.5}$$

- $\overline{CF1_\mu^e}$ represents the linear sum of the instances belonging to the micro-cluster weighted by their membership, is defined by:

$$\overline{CF1_\mu^e} = \sum_{j=1}^{N} \mu_j e_j \tag{3.6}$$

- $\overline{CF1_T^e}$ represents the linear sum of the instances belonging to the micro-cluster weighted by their typicalities, is defined by:

$$\overline{CF1_T^e} = \sum_{j=1}^{N} T_j e_j \tag{3.7}$$

- SSD^e represents the sum of the distances from the instances to the micro-cluster prototype, raised to m and weighted by the pertinence of each instance, is defined by:

$$SSD^e = \sum_{j=1}^{n} \mu_j^m |e_j - c|^2 \tag{3.8}$$

- N represents the number of instances belonging to the micro-cluster;
- t represents the arrival time of the most current instance associated with the micro-cluster;
- *class_id* represents the class associated with the micro-cluster.

This summary structure allows for the merging of micro-clusters, as well as the identification and removal of old micro-clusters.

3.3 Novelty Detection

ND consists of identifying unlabeled data that differs from the current distribution known by the decision model. This is an important problem in the literature (PIMENTEL et al., 2014) and have been addressed by several authors (MASUD et al., 2011; PIMENTEL et al., 2014; FARIA et al., 2016b; SILVA; CAMARGO, 2020; GARCIA et al., 2019).

Many works discuss the ND in batch scenarios, considering that the data will be available for training (FARIA et al., 2016a). However, currently, in the context of DS, a new, much more complex and challenging scenario has emerged to apply ND techniques, being necessary to deal with a large volume of data that arrives in an unlimited manner, which may have its distribution altered over time.

3.3.1 Definition of Outliers, Noise and Anomalies

Outlier, noise and anomaly are co-related terms. The last two have been used in a similar way lately. To simplify the understanding of this work, the definitions adopted for these three terms are the ones proposed by Aggarwal (2017), presented below:

- **Outlier**: is the most generic concept that contains noise and anomalies. It can be defined as an instance that deviates in an unusual way from known data, raising suspicion that it was generated by different mechanisms from the usual ones.
- **Noise**: is defined as any instance that deviates from the normal model. Instances considered noisy should be discarded, as they can decrease the performance of classifier.
- **Anomaly**: has the same definition as noise, however, this type of data has abnormal information about new characteristics of the data generation process that may be relevant for the system.

Both noisy and abnormal data are marked as unknown by classifiers dealing with ND. When the ND process is executed, normally the anomalies represent new classes and are used to identify novelties. During this process, the algorithms have mechanisms that enable the identification and natural discard of noisy data, so that they do not impair the clarity of the novelty found.

The appearance of outliers can be caused by different factors, such as: changes in the environment, malicious activities, instrumentation errors in measuring equipment, human errors, among others. (CHANDOLA et al., 2009). Some examples of outliers in specific environments are (AGGARWAL, 2017):

- **Credit card fraud detection**: Fraudulent credit card transactions may have different patterns, such as purchases in specific locations or exorbitant value transactions, cha-

racteristics that may differ from the usual and represent unauthorized use of the credit card.

- **Intrusion detection systems**: In many computer systems, different types of data are collected from user actions, network traffic, and others. Discrepant data can represent malicious activity.
- **Medical diagnostics**: Many medical applications collect information from a variety of devices, such as: magnetic resonance imaging, electrocardiogram, tomography, among others. Discrepant patterns may represent unhealthy conditions in patients.

In all these applications, outliers are data with different behavior than usual, representing data that deviate from the known normal model. Taking into account the definitions presented, given the proposal of this work, the term novelty can be understood as an anomaly, being an important information that must be analyzed to be incorporated into the decision model.

3.3.2 Concept Evolution and Concept Drift

In numerous real-world applications, DS can change over time due to the non-stationary nature of the data generating environment (WIDMER; KUBAT, 1996). In classification tasks, the label and distribution of class attribute values can mutate over time, such that the attribute values used to define a given class at time t are not the most suitable for define the same class in time $t + k$. Therefore, a classifier that is trained on the initial data will not be efficient if it is not updated according to the mutations of the environment.

Algorithms that deal with traditional ML consider that all classes are previously known. These algorithms assume that the instances belong to at least one class from the set of known classes (PARK; SHIM, 2010). In DS, this same scenario can not be considered, as not all classes are known during the training phase and new classes may emerge over time. The emergence of new classes in DS is called concept evolution.

For Elwell & Polikar (2011), changes in class definitions over time, caused by a change in the distribution in the generating base of the data, represent a concept drift. Figure 1 represents four types of concept drift, along with the time and mean of the data distribution. This figure shows the most commonly found changes in the literature, which are: abrupt, incremental, gradual and recurrent. There are several other types of concept changes already studied that address different features, some can be found in Minku et al. (2010).

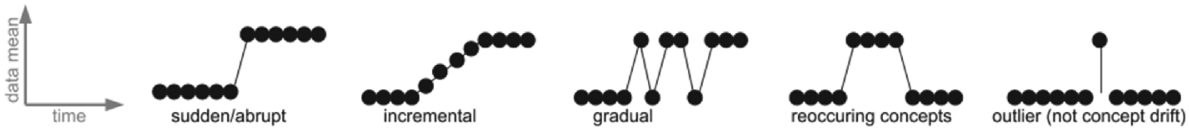


Figure 1 – Patterns of change over time (GAMA et al., 2014).

The four types of concept drift illustrated in Figure 1 can be defined as (GAMA et al., 2014):

- **Abrupt**: one concept is suddenly replaced by another.
- **Incremental**: usually involves more than two concepts with few differences. The change is smooth and becomes eminent after a long period of time.
- **Gradual**: a new concept appears mixing with the older concept, in such a way that both are active for a while. Over time, the occurrence of the older concept decreases, giving place for more occurrences of the new concept.
- **Recurring**: an old concept comes up again after a certain period of time. In this case, it is not possible to know in advance when an old concept will arise again.

Changes in data distributions can occur in different ways and at different times. Identifying the type of change is critical to the adaptation and performance of the classifier.

Algorithms that work with ND must deal with concept evolution, concept drift and noise and outliers detection. These algorithms must identify changes in data and new classes, updating the decision model, in addition to identifying noise and outliers and discarding them, in order to maintain the good performance of the decision model.

3.3.3 Novelty Detection in Data Stream

Faria et al. (2016a) presents a taxonomy, in which ND methods in DS perform the induction of the decision model and the classification of instances in two distinct phases, called *offline* and online.

The *offline* phase addresses the algorithm learning stage. In this phase, the decision model is induced based on a set of labeled data. Three aspects compose this stage (FARIA et al., 2016a):

1. **Number of classes that compose the known concepts**:

   Several algorithms consider that the known concepts are composed of only one class, called the normal class. These algorithms apply techniques with a class to perform the classification of instances. Any instance that do not fit in the normal distribution of the data is considered novelty. Other algorithms consider that known concepts are composed of more than one class and use multi-class classification techniques to label instances.

2. **Number of classifiers to build**:

   Some algorithms consider that only one classifier will be responsible for labeling instances, it will be updated incrementally. Other algorithms use a ensemble of classifiers to classify instances. To induce each classifier, a chunk of instances is used.

3. **Learning task**:

   Some algorithms use the information from the labels of the training set instances to perform the induction of the decision model, making it possible to represent one or more previously known classes of the problem. Other algorithms do not use this information to perform the induction of the classification model, assuming that all training instances belong to the same class, called normal.

   Classifiers using information from labels are often replaced by new trained classifiers with updated data. Classifiers that do not use information from the labels are usually updated incrementally, adding each instance that arrives from the DS to the decision model, when trained through clustering techniques, which enables the evolution of the decision model.

The *online* phase addresses the data classification step. At this stage, unlabeled instances arrive at high speed and sequentially. Three aspects make up this stage (FARIA et al., 2016a):

1. **Classification of new instances**:

   This task checks if the new instance can be explained by the decision model. Approaches using a single classifier and an ensemble of classifiers have been proposed. Some proposals consider the new instance as normal if it is explained by the decision model or as novel when it is not explained by the current decision model. In these proposals, a novelty is identified through the presence of a single instance that is not explained by the model.

   Other proposals do not classify the instance as novelty, in this case, when a new instance is not explained by the decision model, it is marked as unknown and stored in a temporary memory that will be used in the future to identify NPs. In this case, a cohesive and representative set of instances must exist to be considered novelty.

2. **Novelty Pattern Detection**:

   This task uses unlabeled instances to identify novelties. In anomaly detection systems, a single example not explained by the decision model is sufficient to classify it as anomalous. However, NPs can belong to several different classes and must be used to classify instances of that class. Because of this, many algorithms consider that a novelty or class extension is formed by a set of unknown instances that form a cohesive and representative group. If this group has high similarity with an existing group, it is given the same label as the existing group, otherwise it represents a new NP.

3. **Decision Model Update**:

   Algorithms dealing with ND in DS must update their decision model in order to adapt to concept evolution and concept drift. The use of a single classifier and a ensemble of classifiers impacts directly the update of the decision model, as algorithms with a single

classifier update their decision model incrementally, while algorithms that use a ensemble of classifiers train a new classifier to replace an old one.

The use of external feedback is fundamental to define the way to update the decision model. Algorithms that use external feedback assume that after a certain period of time the actual labels of the instances will be available. Algorithms that do not use external feedback update their decision model without information about the actual labels of the instances.

In addition to updating their your decision model, many algorithms use a mechanism to forget about outdated data. Algorithms using ensemble of classifiers replace a classifier whenever a new one is trained. Several forgetting mechanisms have been proposed for algorithms that use only one classifier.

3.4 Label latency

The use of external feedback is essential to define the way to update the decision model. Algorithms that use external feedback consider that, at some point in the DS, the actual labels of the instances will be available and these will be used to update the decision model, incrementally, in the case of only one classifier, or replacing one of the classifiers when it comes to an ensemble. Algorithms that do not use external feedback assume that the actual labels of the instances will never be available.

Some algorithms for DS (SPINOSA et al., 2009; FARIA et al., 2016b; SILVA; CAMARGO, 2020; ABDALLAH et al., 2016; GARCIA et al., 2019) available in the literature do not use external feedback for updating of the decision model, that is, they consider that a subset of labelled instances will be available during the *offline* phase, or that the real labels of the instances will never be available in the *online* phase. These algorithms update their models incrementally, without real information from the instances.

There are several algorithms for DS (MASUD et al., 2011; RUSIECKI, 2012; KRAWCZYK; WOŹNIAK, 2013; AL-KHATEEB et al., 2012; FARID; RAHMAN, 2012; FARID et al., 2013) that handle the classification task and model update using external feedback. For this to be possible, after a certain time of classification of an instance, the true label is made available to the algorithm.

The time between classifying an instance and getting its true label is called latency. Latency time may vary depending on the environment being considered. Below are the three most common environments found in the literature (Souza et al., 2018):

- **Null latency:** environment in which the true instances label are obtained right after there classification, without any delay.

- **Extreme latency:** environment where the true instance label are never made available to the application.
- **Intermediate latency:** environment where the true label instance will be available after a interval time *T* from its classification, where $0 < T < \infty$. Time can be variable or fixed, so labels can arrive disorderly for the application. Intermediate latency can be measured by number of instances (eg latency time is 5 instances, so instance true label x_t will be available after classification instance x_{t+5}) or by measures of time (eg seconds, minutes, hours, days, etc).

Latency plays a fundamental role in updating the decision model in applications that deal with DS learning, in addition to having a direct impact on performance (KREMPL et al., 2014).

Chapter 4
RELATED WORKS

This chapter presents a set of approaches found in the literature review that address novelty detection and are relevant to the contextualization of the approach proposed here. To simplify, the works were divided into three different sections, which are: Binary Approaches, Multi-Class Approaches and Fuzzy Approaches. The works presented in each section are divided among approaches that consider extreme, null and intermediate latency to obtain the true labels of the data coming from the DS. Finally, in order to facilitate the understanding of related works, a table containing a comparative summary is presented.

4.1 Binary Approaches

In this subsection, the works that consider classification and ND in DS as a binary task are described, that is, these works consider that the known concepts represent the normal class and unknown concepts, which differ from the data distribution used for training, represent the abnormal class or novelties. During the bibliographical review, no works in this category were found that consider environments with null latency.

4.1.1 Extreme Latency

Spinosa et al. (2009) presents an approach to ND called OLINDDA (*Online* Novelty and Drift Detection Algorithm). This algorithm uses a single cluster-based classifier to identify new NPs and considers extreme latency. The algorithm starts by analyzing a set of normal instances to create the initial decision model. The K-means clustering algorithm is used to produce the initial k clusters and the normal concept decision model, without distinguishing the classes. The normal model is composed of k hyperspheres, which are the clusters obtained in the clustering process.

Each hypersphere is represented by its center and the radius, which is the Euclidean distance from the center to the furthest instance belonging to the cluster. In the *online* phase, a new instance is classified. If the instance can be explained by previously known model, that is, if

the instance is within the radius of any of the hyperspheres belonging to the decision model, it is a sign that it is part of that dataset. Then the hypersphere statistics are updated and the instance is discarded. If the instance does not belong to any hypersphere, it is marked as unknown and moved to a temporary memory for further analysis.

To perform the ND, every time an instance is marked as unknown, if the temporary memory has the minimum amount of instances, k clusters are made with the instances that are in the temporary memory, looking for valid clusters that represent the emergence of a new concept. The cluster validation criteria considers two basic measures: cohesion and representativeness.

When a valid cluster is found, OLLINDA checks its similarity with the already known normal concept. If the identified cluster has similarities with the normal concept, that is, its center is located within the area of some hypersphere, it is considered an extension of the normal model and its statistics are computed together with those of the hypersphere. Otherwise, this cluster represents a new concept and a new hypersphere is created.

The number of clusters (k) used to clustering unknown instances should adapt automatically to increase the probability of finding valid cluster. Initially, the value of k is user-defined. During the execution of the algorithm, in each iteration to create new clusters with the unknown instances, the algorithm checks what prevents the validation of each cluster, caused by low density or small number of instances. If most failures are due to low density, k will be increased. If few instances are the most frequent cause of failure, k decreases. In this way, OLINDDA performs the adaptation of the decision model incrementally.

Tan et al. (2011) presents an anomaly detection approach called HS-Trees (Streaming Half-Space-Trees). This algorithm consists in an ensemble of HS-Trees that does not use information from the labels to perform the induction or classification of the data, that is, this algorithm deals with extreme latency. Each HS-Tree consists of a set of nodes responsible for capturing the number of instances (mass) within a specific DS space.

First, through the training data, the algorithm learns the mass profile of the data. After the learning step, the algorithm starts the identification of anomalies of new instances that arrive from the DS. Instances belonging to high-mass nodes are considered normal, while instances belonging to low-mass nodes are considered anomaly. Data label information is irrelevant to this algorithm. For learning and classification, only information from the dimensions of the data in space is used.

Gruhl & Tomforde (2021) presents an approach for anomaly detection called OHODIN (Online Extension with Heuristics to the ODIN Limit). OHODIN is based on K-NN and works in two steps. First, the algorithm builds a directed and weighted k-nearest neighbor graph. Each observation is represented by a vertex, which has *k* edges, which correspond to the distance to its nearest neighbors. In the classification stage, for each instance that arrives, the algorithm calculates the nearest neighbors considering all existing vertices in the decision model. Anomaly

detection occurs by comparing the number of edges of each vertex of the k-neighbors with a threshold value *T*. If the number of edges is less than *T*, it is considered an anomaly.

4.1.2 Intermediate Latency

In Rusiecki (2012) an approach that uses neural networks for anomaly detection in DS is proposed, which considers intermediate latency to obtain the true labels. The decision model is composed of two autoregressive neural networks. A robust learning algorithm is used for the induction of a neural network, seeking to minimize errors and the influence of outliers in the training set. The other neural network is trained using the traditional backpropagation algorithm.

The instances used in the neural networks induction process represent the normal concept. New instances arriving at the DS are submitted to neural networks. A user-defined threshold is used to define whether the instance is normal or whether is an anomaly. If the difference between the results of the two neural networks is less than this threshold, the instance is classified as normal, otherwise, as an anomaly. Instances that have true labels are stored in a temporary memory and the weights of the neural networks are often updated.

4.2 Multi-Class Approaches

In this sub-section, the works that consider classification and DN in DS as a multi-class task are described. These works classify the instances in different classes and consider that one or more novelty pattern may arise.

4.2.1 Null Latency

Farid & Rahman (2012) present an approach using a decision tree for ND in DS, considering null latency to obtain the true labels. In the *offline* phase, the algorithm induces a decision tree based on the data available for training, calculates the percentage of the number of instances belonging to each leaf node in the tree, in relation to the training dataset, and performs the clustering of the data at each of the leaf nodes.

In the *online* phase, every new instance that can be explained by the current decision model is considered normal, otherwise it is considered novelty. If the number of instances classified by a leaf node increases above the previously calculated percentage, it may mean the arrival of a new class. Next, it is analyzed which leaf node cluster the instance belongs to. If the instance does not belong to any cluster, it is a sign that something new has arrived. Finally, the new instance is added to the training dataset and the decision tree is rebuilt.

In Farid et al. (2013) an ensemble of three decision trees is proposed, being an extension of the approach proposed by Farid & Rahman (2012). In the *offline* phase, using a Naive Bayes classifier, built on the training dataset, the weight of the resulting probability of each instance

of the training dataset is initialized. The construction of the first tree happens through selection techniques with recomposition. For the construction of the other two trees, a new dataset is created, from the instances that have the highest weights according to their classification, using the first tree that was trained.

The leaf nodes of the decision trees are submitted to a clustering process in order to define decision limits to enable the identification of novelties. In addition, information such as the calculation of the percentage of instances in each leaf node in relation to the training set is stored. In the *online* phase, a new instance is classified according to the majority vote of the classifiers or, as unknown, if it is not part of any leaf node cluster.

When an instance is classified as unknown, it is stored in temporary memory and its novelty flag is assigned a value of 1. If the number of instances classified by a leaf node increases or decreases in relation to the previously calculated value, this new instance with the value 1 in the novelty flag belongs to a novelty class. To adapt to the concept evolution and concept drift, each decision tree has a weight, based on its accuracy obtained in the classification of instances from the training set. A decision tree is replaced by a new one when it has a low weight, indicating that its accuracy is low.

4.2.2 Extreme Latency

Faria et al. (2016b) proposes an algorithm for multi-class ND and classification called MINAS (MultI-class learnNing Algorithm for data Stream), which updates its decision model incrementally, considering extreme latency. The algorithm is divided into two phases, named *offline* and online. The *offline* phase is responsible for inducing the initial decision model, based on a labeled dataset. First, the instances are clustered by classes. For each identified class, a set of k micro-clusters are created using the algorithm K-means or CluStream. Each micro-cluster is represented by a feature vector composed by the number of instances, linear sum of instances, square of the sum of instances, the arrival time of the last instance classified by the micro-cluster and the class label. With these measurements, it is possible to obtain important information, such as the center and radius of the micro-clusters.

In the *online* phase, a new instance is classified, using the decision model. If the instance is explained by the current decision model, statistics of the micro-clusters are updated. If the instance is not explained by the current decision model, it is marked as unknown and is stored in a temporary memory. When a minimum number of instances is stored in this memory, MINAS performs the clustering of these data into k new micro-clusters, using one of the algorithms described above. Each micro-cluster is evaluated and the non-cohesive and non representative ones are removed. To validate cohesion, the simplified silhouette coefficient (VENDRAMIN et al., 2010) is used. If the simplified silhouette coefficient is greater than a user-defined value, the group is valid. A new micro-cluster is representative if it has a minimum number of instances. Valid micro-clusters are analyzed in order to verify if they represent a class extension or a new

pattern of novelties. A micro-cluster is considered an extension of an existing concept when its centroid is at a maximum distance T (user-defined) from the centroids of the already known micro-clusters.

If the distance for all known micro-clusters is greater than T, it is considered a novelty.

To handle recurring contexts, MINAS keeps in a memory all the micro-clusters that have not been used within a user-defined interval. When a new valid micro-cluster is found when clustering the instances in temporary memory, it is compared with the unused micro-clusters. If the distance between the centroid of the new micro-cluster and the centroid of one of the unused micro-clusters is less than T, it is considered a recurrence and the micro-clusters are removed from the recurrence memory are added to the decision model.

In Abdallah et al. (2016) a technique capable of identifying new concepts, normal and abnormal, considering extreme latency is presented. The authors propose an adaptive ensemble approach to multi-class ND called AnyNovel. In the *offline* phase, the initial learning model is created, using the training data. For this, a supervised learning technique is applied to the labeled data, dividing them into groups of classes. Then, an unsupervised learning technique is applied to each of the groups to detect sub-groups and create local models for each class. These groups together with their sub-groups are stored in a data structure called Baseline Learning Model (BLM).

In the *online* phase, each chunk of instances that arrives from the DS is analyzed. To perform the DN, two components called CVC (Cohesion Validation Component) and Observer are used to analyze the chunks. Data that have similar characteristics to one of the known classes are explained by the CVC component, while data that can not be explained are stored in a temporary memory.

The Observer is used for cohesion and separability of the data contained in the temporary memory, comparing with the evolution of the DS. When a new concept appears, a clustering of the data belonging to the new concept is performed and sub-clusters of this clustering are created and added to the BLM. A forgetting mechanism is used to identify concepts that are no longer present in the CDF and mark them as irrelevant. When a new unknown cluster is added to the BLM, a portion of the data is selected to be manually labeled by an expert.

Garcia et al. (2019) proposes an algorithm called Hygia that use extreme latency. This algorithm performs the update of the decision model incrementally, considering the absence of labeled data to perform the ND and the concept evolution. In the *online* phase, the decision model is induced based on a set of labeled data. This model is composed of micro-clusters, equivalent to those used in Faria et al. (2016b), created using the algorithm CluStream.

In the *online* phase, a new instance that arrives can be classified as normal, extended, or unknown. For classification, Higia uses an approach similar to k-nearest neighbors (k-NN), calculating the Euclidean distance between the new instance and the centroid of the closest

K micro-clusters belonging to normal classes . If the new instance has a distance to smaller than a threshold T for some micro-cluster, it receives the micro-cluster label and latter updates its statistics. If the distance for all micro-clusters is greater than T, it is stored in a temporary memory to be analyzed if it is new. Instances stored in temporary memory form micro-clusters. When a micro-cluster receives a minimum amount of instances, defined by the user, a novelty is found and the micro-cluster is added to the decision model.

Costa Júnior et al. (2019) proposed the MINAS-BR algorithm (MultI-label learning Algorithm for Data Streams with Binary Relevance transformation), being an evolution of the MINAS method, created to deal with ND in multi-label DS, considering extreme latency. As its predecessor, MINAS-BR is divided into two phases, called *offline* and online.

In the *offline* phase, a binary relevance transformation method is applied to the problem. As input, a training multi-label dataset is divided into subsets and used for the induction of the initial decision model. Each subset is divided into k micro-clusters, using the k-means algorithm.

The *online* phase is similar to the original MINAS. The algorithm searches for the closest micro-cluster to the instance to be classified and assigns the label of the closest Z micro-clusters, where Z is the cardinality of the dataset, a property of multi-label datasets. If the instance can not be explained by the decision model, it is added to temporary memory. ND occurs when the temporary memory is full. For this, micro-clusters are created from temporary memory. Among these micro-clusters, those with at least three instances and a silhouette value greater than zero are valid. Valid micro-clusters are analyzed to see if they represent a novelty or a class extension. As MINAS, MINAS-BR often removes old instances from temporary memory.

In Cai et al. (2019) the SENNE algorithm is presented, which uses an instance-based ensemble of classifiers per class to identify new emerging classes in DS. Each instance of the training dataset represents a hypersphere, which has a radius, defined by the distance between the instance and its nearest neighbor belonging to the same class. For every instance that arrives, the algorithm checks the distance between its nearest neighbor for each class, if the distance is greater than an user-defined threshold, that instance is marked as novelty and stored in a novelty buffer. Otherwise, this instance receives the label of its nearest neighbor. When the novelty buffer reaches its maximum size, the instances stored in it are used to create a set of hyperspheres that will represent the new class.

4.2.3 Intermediate Latency

Masud et al. (2011) propose an algorithm called ECSMiner (Enhanced Classifier for Data Streams with novel class Miner). This algorithm is based on an ensemble of classifiers for classification and ND in concept drifting DS under time constraints, considering intermediate latency to obtain the true labels. The authors suggest that each instance should be classified within a certain time window, taking into account the possible delay for a true label to become

available to the system, in such a way that the classification time is less than the time to obtain the true label.

ECSMiner is based on a majority vote of the classifiers, where it can be used to classify decision trees or the K-NN algorithm, here we will describe the functioning of ECSMiner using K-NN, which was the approach used in the comparisons with the our proposal. In the *offline* phase, the algorithm divides the training dataset into chunks, the instances belonging to each chunk are separated by classes and grouped using the k-means algorithm. The resulting clusters for each chunk result in a classifier that is added to the ensemble.

During the *online* phase, the algorithm checks if the instance can be classified by the ensemble. Through the majority vote of all classifiers, the instance is labeled. If it is marked as unknown by all classifiers, this instance is buffered from unknown instances. When this buffer reaches its maximum size, the instances that are stored in it go through a clustering process, the valid and representative clusters go through a validation process, by all classifiers in the ensemble, using the q-NSC measure, which allows the classifier define whether the cluster represents a novelty or not. If all classifiers vote for novelty, the instances belonging to the cluster are classified as novelty and discarded by the algorithm. ECSMiner does not differentiate between the types of novelties found and does not use this information to update its decision model. Instances that arrive with a true label are stored in a buffer and used to train new classifiers.

In Al-Khateeb et al. (2012) an algorithm called CLAM (Class-based Micro Classifier Ensemble) is proposed, which considers intermediate latency to obtain the true labels. The authors propose the use of an ensemble of classifiers based on classes, where each known class has a set of M micro-clusters. For each micro-cluster, its radius, center and number of elements are summarized. In total, L classifiers are created, where L is the number of known classes.

CLAM is trained through data chunks, whose size is defined by the user. Each chunk is divided into subsets such that each subset contains instances of one of the classes of the classes present in the chunk. Each subset is clustered using the K-means algorithm to obtain the necessary information to form micro-clusters and to be added to the classifier of their respective classes.

In the *online* phase, a new instance is classified according to the micro-cluster that has the closest center to it, receiving the label of the class of the classifier to which the closest micro-cluster belongs. If the instance is not found next to any of the micro-clusters of the ensemble, it is marked as unknown and stored in a temporary memory. To perform the ND, CLAM performs the clustering of the instances that are in temporary memory as in Masud et al. (2011). This algorithm depends on the arrival of true labels from the data to update its decision model.

The *offline* phase is executed every time a chunk of labeled data is formed, training a new classifier. Whenever a new classifier is trained, it replaces the classifier that has the highest predictive error of its corresponding class from the ensemble classifier.

Zheng et al. (2021) presents a semi-supervised algorithm called ESCR to detect concept evolution and concept drift in DS. First, based on the data available for training, the algorithm prepares the initial decision model, which is composed of an ensemble of cluster-based classifiers. The training dataset is divided into chunks and each chunk is used to train a base classifier.

When a new instance arrives, the algorithm checks whether the instance represents an outlier or not, checking whether the instance is outside the decision boundaries of all classifiers in the ensemble. If not, for each base classifier, the ESCR finds the closest pseudopoint of the instance and calculates the confidence score, based on the Jensen-Shannon divergence, for each pseudopoint, finally, the label is assigned based on the majority vote of the base classifiers, weighted by confidence score.

If the instance represents an outlier, it is stored in a buffer, which is often analyzed. If a set of instances is not in the decision model, this set is considered a novelty and added to the decision model. Eventually, labeled instances arrive and are stored in a buffer and used to train new classifiers.

Din & Shao (2020) present an algorithm called EMC (Evolving Micro-Clusters) dealing with concept drift and concept evolution in DS. First, the algorithm trains the decision model by applying the K-Means algorithm to the initial training data, which does not need to be labeled. The clusters obtained are summarized in micro-clusters, which constitute the initial decision model.

In the *online* phase, a micro-cluster level lazy learning framework technique based on the distance of the cluster structure and the purity of the cluster is used to classify the instances. Instances that have a distance to the center of the nearest micro-cluster, greater than the radius of the respective micro-cluster, are marked as an outlier and stored in a temporary memory.

To identify novelties, for each instance of the buffer, EMC uses the Local Outlier Factor (LOF), which calculates a score based on the average local density of the nearest neighbors and their local density. Instances that represent novelties have high LOF, while normal instances have similar local density to their neighbors. The decision model is updated in a semi-supervised way. A small part of the data is often labeled and made available to be used in the training of new micro-clusters.

4.3 Fuzzy Approaches

In this section, methods that deal with ND in DS and apply concepts of fuzzy set theory are described. During the bibliographic review, no works were found in this category that consider environments with null latency.

4.3.1 Extreme Latency

Silva et al. (2018) present a fuzzy generalization of the MINAS method (FARIA et al., 2016b) for ND in DS called FuzzND (Fuzzy multi-class Novelty Detector for data streams). In the *offline* phase, FuzzND creates the decision model from the labeled data available for training. For each class identified, a set of k micro-clusters, called SFMiC (Supervised Fuzzy Micro-Cluster) (SILVA et al., 2017) is created, using the algorithm Fuzzy C- Means (BEZDEK et al., 1984). The SFMiC structure allows calculating the fuzzy similarity between two micro-clusters.

In the *online* phase, for each instance that arrives, the algorithm calculates its membership to each of the SFMiCs present in the decision model. Afterwards, the memberships of the instance to each one of the SPFMiCs of the same class are added, resulting in the match value of the instance with that class. Which will be used to define if a label of a class will be applied to the instance or if it will be labeled as unknown. When an instance is classified as unknown, it is added to a temporary memory for further analysis. Whenever an instance is added to the temporary memory, FuzzND checks if there is the minimum number of instances to perform the ND. If is true, the algorithm Fuzzy C-Means is used to find k clusters. Each of the clusters found is evaluated, checking the value of the fuzzy silhouette coefficient and the number of instances. FuzzND removes SFMiCs that have not rated instances within a user-entered time range.

Silva & Camargo (2020) present an evolution of the FuzzND algorithm called PFuzzND (Possibilistic Fuzzy Multiclass Novelty Detector for Data Streams). As this algorithm was one of the bases for our work, we will present more details about it. The main differences between FuzzND and PFuzzND are the algorithm used for the data clustering task and the summary structure that represents the micro-clusters. As already mentioned, FuzzND uses the Fuzzy C-Means algorithm, while its successor uses the Possibilistic-Fuzzy C-Means algorithm (PAL et al., 2005). In FuzzND, the structure called SFMiCs is used, in PFuzzND the structure proposed by the author to store the statistics used for all the calculations, including typicality, was called Supervised Possibilistic Fuzzy Micro-Cluster (SPFMiC).

The SPFMiC is a summary structure for possibilistic fuzzy clusters. It is defined as a vector (M^e, T^e, $\overline{CF1^e_\mu}$, $\overline{CF1^e_T}$, SSD^e, N, t, $class\_id$), where M^e represents a linear sum of the membership of instances raised to *m*; T^e a linear sum of the typicalities of instances raised to n; $\overline{CF1^e_\mu}$ is the linear sum of instances weighted by their membership; $\overline{CF1^e_T}$ is the linear sum of instances weighted by their tipicalities; SSD^e the sum of the distances of the instances to the prototype of the micro-cluster, raised to m and weighted by the membership of an instance; N the number of instances associated to the micro-cluster and *class_id* a class associated with the micro-cluster.

The SPFMiC stores information that makes it possible to calculate the centroid, as defined in Equation 4.1 and to calculate the typicality of an instance for that cluster (Equation 4.3) which is a measure of the Possibilistic Fuzzy C-Means algorithm. The value γ defined in

Equation 4.2 is required for the calculation of tipicality. These calculations are necessary during the *online* phase.

$$center_i = \frac{\alpha * \overline{CF1_e^x} + \beta * \overline{CF1_t^x}}{N} \tag{4.1}$$

$$\gamma_i = K * \frac{SSD^e}{M^e} \tag{4.2}$$

$$typicality_i = \frac{1}{1 + (\frac{\beta}{\gamma_i} * |x_k - center_i|^2)^{1/(n-1)}} \tag{4.3}$$

In the *offline* phase, PFuzzND separates the labeled training data into classes and the instances belonging to each class are clustered into K clusters using the FCM algorithm. The resulting clusters form the initial decision model.

In the *online* phase, when a new instance arrives, the algorithm calculates the typicality of this instance for all SPFMiCs that are part of the decision model. If the highest typicality obtained is greater than the average of the typicality values obtained, subtracted from the θ_adapt parameter, which is an initialization parameter, the instance receives the label of the SPFMiC and the summary structure of this SPFMiC is updated to include the instance. As a result, cluster maintenance is done incrementally, making the model adjust smoothly to concept drift.

Otherwise, PFuzzND adds the instance to a temporary memory of unknowns. Whenever an instance is added to this memory, the algorithm checks if the temporary memory of unknowns has the minimum number of instance to run the ND. In the ND process, the algorithm performs the clustering of the instances that are in the temporary memory of unknowns using the FCM algorithm.

Each resulting cluster goes through a validation process, where it is verified if the fuzzy silhouette coefficient (CAMPELLO; HRUSCHKA, 2006) is greater than 0 and if the cluster has a representative number of instances. Invalid cluster are discarded and their instances remain in the temporary memory of unknowns. The valid clusters are summarized and go through a process that verifies if they represent a novelty or a class extension using the fuzzy similarity FR introduced by Xiong et al. (2012) and presented in Equation 4.5 where $FR_{i,j}$ is the fuzzy similarity between clusters i and j. The similarity $FR_{i,j}$ is the ratio between the sum of the fuzzy dispersion of cluster i (dp_i) and cluster j (dp_j) and the distance between the centers of clusters i ($centers_i$) and j ($centers_j$). The fuzzy dispersion is calculated as in Equation 4.4. If the fuzzy similarity measure FR is greater than a pre-defined threshold, the cluster receives the same label as the SPFMiC that had the highest value. Otherwise, a new label of the form $NP\#$ is generated for the cluster. Finally, the SPFMiC that defines this cluster is added to the decision model.

$$dp = \sqrt{\frac{SSD^e}{N}} \tag{4.4}$$

$$FR_{i,j} = \frac{dp_i + dp_j}{|centers_i - centers_j|} \tag{4.5}$$

4.3.2 Intermediate Latency

An algorithm called eClass (Evolving Classifier) is proposed by Angelov & Zhou (2008), which considers intermediate latency for obtaining true labels, for classification and ND in DS based on fuzzy decision rules of the Takagi-Sugeno type. eClass is based on *online* learning, being able to start learning from scratch, that is, fuzzy decision rules do not need to be previously specified for the algorithm. In addition, the algorithm can adapt to the emergence of new classes along the DS, however, to carry out the identification, the algorithm depends on the arrival of instances with true labels.

4.4 Summary of related approaches

This section aims to summarize the algorithms described in the previous section. For that, Table 1 presents a summary of the main characteristics of each one of the presented algorithms.

Algorithm	Number of classes	Classifiers	External Feedback	Type of Latency
OLINDDA (SPINOSA et al., 2009)	Binary	Single	No	Extreme
HS-Trees (TAN et al., 2011)	Binary	Ensemble	No	Extreme
OHODIN (GRUHL; TOMFORDE, 2021)	Binary	Single	No	Extreme
RNAs (RUSIECKI, 2012)	Binary	Ensemble	Yes	Intermediate
Árvore (FARID; RAHMAN, 2012)	Multi-Class	Single	Yes	Null
Árvores (FARID et al., 2013)	Multi-Class	Ensemble	Yes	Null
MINAS (FARIA et al., 2016b)	Multi-Class	Single	No	Extreme
AnyNovel (ABDALLAH et al., 2016)	Multi-Class	Ensemble	No	Extreme
Higia (GARCIA et al., 2019)	Multi-Class	Single	No	Extreme
MINAS-BR (Costa Júnior et al., 2019)	Multi-Class	Single	No	Extreme
SENNE (CAI et al., 2019)	Multi-Class	Ensemble	No	Extreme
ECSMiner (MASUD et al., 2011)	Multi-Class	Ensemble	Yes	Intermediate
CLAM (AL-KHATEEB et al., 2012)	Multi-Class	Ensemble	Yes	Intermediate
ESCR (ZHENG et al., 2021)	Multi-Class	Ensemble	Yes	Intermediate
EMC (DIN; SHAO, 2020)	Multi-Class	Single	Yes	Intermediate
FuzzND (SILVA et al., 2018)	Multi-Class	Single	No	Extreme
PFuzzND (SILVA; CAMARGO, 2020)	Multi-Class	Single	No	Extreme
eClass (Angelov; Zhou, 2008)	Multi-Class	Single	Yes	Intermediate

Table 1 – Summary of related approaches.

As we can see in Table 1, some algorithms available in the literature deal with binary classification (SPINOSA et al., 2009; TAN et al., 2011; RUSIECKI, 2012; GRUHL; TOMFORDE, 2021), that is, these algorithms consider that the normal concept is constituted by only one class, called normal. These algorithms consider that the data used in the training phase

represent normal concepts, while any instance that arrives at the DS and it is different from the known concept represents a novelty (or anomaly).

Other approaches consider the classification task as multi-class, such as (FARID; RAHMAN, 2012; FARID et al., 2013; FARIA et al., 2016a; ABDALLAH et al., 2016; GARCIA et al., 2019; Costa Júnior et al., 2019; MASUD et al., 2011; AL-KHATEEB et al., 2012; SILVA et al., 2018; SILVA; CAMARGO, 2020; Angelov; Zhou, 2008; CAI et al., 2019; ZHENG et al., 2021; DIN; SHAO, 2020). These algorithms consider that the known concepts are constituted by more than one class. Therefore, their decision models use techniques that make it possible to differentiate the classes that compose the problem.

Another characteristic analyzed is the number of classifiers used by each of the algorithms. Approaches such as (SPINOSA et al., 2009; FARID; RAHMAN, 2012; FARIA et al., 2016a; GARCIA et al., 2019; Costa Júnior et al., 2019; SILVA et al., 2018; SILVA; CAMARGO, 2020; Angelov; Zhou, 2008; GRUHL; TOMFORDE, 2021; DIN; SHAO, 2020) use only a single classifier that is incrementally updated, recalculating its summary statistics for each new instance that arrives from DS, with the exception of (FARID; RAHMAN, 2012), which updates its decision model from time to time by training a new decision tree.

Methods presented in (TAN et al., 2011; RUSIECKI, 2012; FARID et al., 2013; ABDALLAH et al., 2016; MASUD et al., 2011; AL-KHATEEB et al., 2012; CAI et al., 2019; ZHENG et al., 2021) use an ensemble of classifiers and update their decision models by training new classifiers through data chunks. Although all the works cited used an ensemble of classifiers, Al-Khateeb et al. (2012) followed a different line and proposed the use of an ensemble of classifiers per class.

Approaches described, such as (SPINOSA et al., 2009; TAN et al., 2011; FARIA et al., 2016a; ABDALLAH et al., 2016; GARCIA et al., 2019; Costa Júnior et al., 2019; SILVA et al., 2018; SILVA; CAMARGO, 2020) do not use external feedback to obtain the labels and, consequently, consider extreme latency. Other algorithms use external feedback and consider null latency to obtain the true labels, like (FARID; RAHMAN, 2012) and (FARID et al., 2013). Methods such as (RUSIECKI, 2012; MASUD et al., 2011; AL-KHATEEB et al., 2012; Angelov; Zhou, 2008) use external feedback and consider scenarios that allow obtaining true labels and using algorithms that deal with intermediate latency.

Chapter 5

Enhanced Fuzzy Classifier for Multi-class Novelty Detection in Data Streams

With advances in technology and the popularization of the internet, sensor networks and mobile devices, large-scale data production has increased and, consequently, new ways of acquiring knowledge through these large volumes of data have emerged. This large-scale production, also known as DS, refers to the amount of data that is produced sequentially, over time, in such a way that it is impossible to store it in an environment that can change its generating system (GAMA, 2010).

The characteristics of DS can lead to unexpected situations for the learning process, especially when it comes to noisy data, due to their non-stationary distribution. The use of fuzzy set theory is a way to make the learning process more flexible, allowing the algorithm to adapt better to possible inaccuracies in the data.

Latency has a fundamental role in algorithms dealing with classification in DS, especially when dealing with non-stationary data, and directly impacts the results (Souza et al., 2018). Most of the methods proposed in the literature consider extreme latency, they update their decision models without the information on the true labels, incrementally, recalculating the statistics of their decision model for each classified instance.

An optimistic way of addressing the availability of true labels, adopted by some authors, is null latency. In this case, the methods update their decision models considering that the real label of the instances will be available right after the classification, even before the classification of the next instance.

Although the mentioned approaches are widely used, they may not correspond with such efficient results in practice in certain scenarios. Methods that use extreme latency can be degraded in more critical environments where small inaccuracies can affect performance. Methods that use null latency may be unfeasible in most real-world scenarios, as the immediate labeling process is very costly.

Considering intermediate latency to obtain the true labels can be a deft way, in many real-world scenarios, to adapt to concept evolution and concept drift. In many cases it is common that, with a certain amount of time after classification, the actual label is made available such as: medical image analysis; credit card fraud detection; electricity market where prices vary according to demand; weather forecast; and others.

Among the most representative approaches in this context, Masud et al. (2011) proposed a method called ECSMiner which is an enemble-based model for multi-class classification and ND in DS, considering intermediate latency to obtain the true labels and time constraints for classifying new instances. Despite its good performance, this algorithm considers the ND as a binary classification problem, not identifying the different NPs that can arise over time.

Following another context, the PFuzzND (SILVA; CAMARGO, 2020) uses fuzzy clustering and considers ND as a multi-class classification problem. However, this algorithm considers the extreme latency of the labels, that is, the real labels will never be available for updating the decision model.

Unlike the method proposed by Masud et al. (2011), which performs the ND without distinguishing different NPs, this proposal aims to identify and classify different NPs. Our approach was originally developed to work in environments that consider intermediate latency, however, it also allows the use in extreme latency environments. Furthermore, our proposal uses fuzzy clustering techniques based on the PFuzzND algorithm (SILVA; CAMARGO, 2020).

In this work, we propose a method called Enhaced Fuzzy Classifier for Classification and Multi-class Novelty Detection for Data Streams (EFuzzCND), inspired by these two approaches just mentioned, which aims to perform multi-class classification and novelty detection in data streams with intermediate and extreme latency, identifying different novelty patterns. Therefore, two decision models, one supervised and the other unsupervised, are combined. The supervised model performs the multi-class classification of unlabeled instances as they arrive from the DS. The unsupervised decision model takes care of maintaining the structure that identifies novelties and separates them into different patterns when an instance is not classified by the supervised model.

In the case of intermediate latency, we consider that the sorted data reappears in the DS after a certain time, in the same order in which they appear the first time, but with the true labels. This data is stored in a temporary memory until there is enough data to generate new micro-clusters with labeled data. The new micro-clusters are used to update the supervised model and eventually label novelty patterns that were in the unsupervised model, which will then be eliminated from the unsupervised model. The next sections present details of the proposed method.

5.1 Algorithm Overview

The approach proposed in this work uses, in cooperation, a supervised decision model and an unsupervised decision model. The supervised decision model consists of SPFMiCs obtained from labeled data, and it is used to classify instances belonging to known concepts. The unsupervised decision model is composed of SPFMiCs obtained by the ND method, and it is used to classify instances marked as unknown by the supervised decision model in different novelty patterns.

The algorithm consists of two phases: *offline* and *online*, following the proposal presented by Faria et al. (2016a). Figures 2 and 3 show these phases. In the *offline* phase, the algorithm builds the initial decision model. For this, the instances belonging to the labeled dataset for training are separated by classes and each class is clustered using FCM to form SPFMiCs that will be part of the initial decision model.

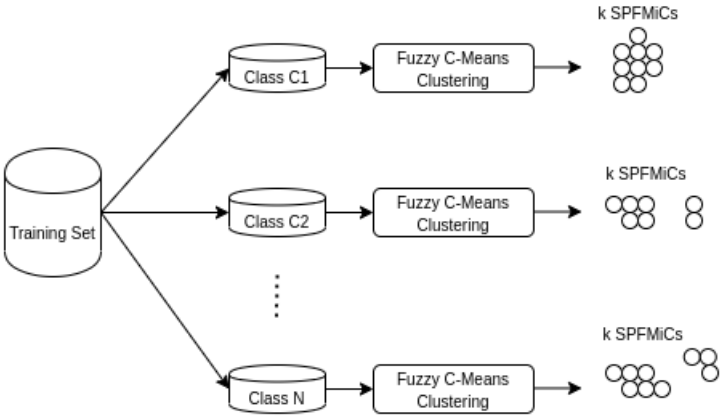


Figure 2 – *offline* phase overview

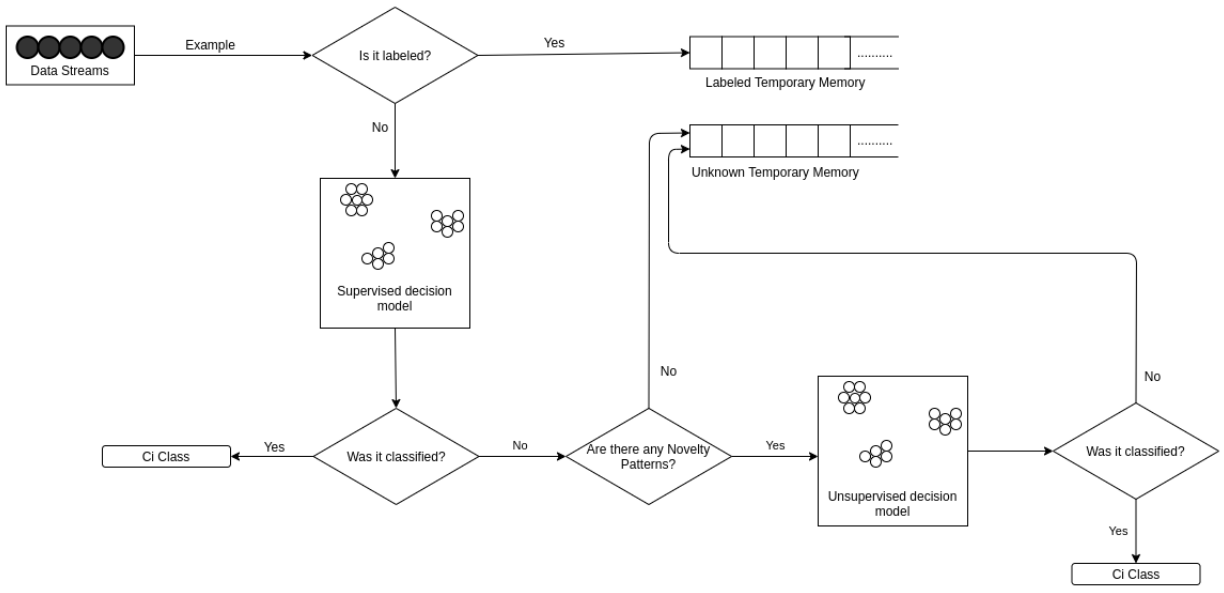


Figure 3 – *Online* phase overview

In the *online* phase, the algorithm classifies new unlabeled instances that arrive at the DS, detecting new NPs and updating the decision models. When a new instance arrives, the algorithm checks if it has a label. Instances that have labels are the ones that have been classified by the model some time earlier and, due to intermediate latency, when their true labels are available, return in the same DS interleaved with the unlabeled instances. They are stored in the labeled temporary memory and will be used to update the supervised decision model. If the new instance has no label, the algorithm tries to classify it.

Some instances may not be classified by the supervised decision model. When this happens, the algorithm assumes that the instance is a novelty and starts using the unsupervised model. The unsupervised model is also formed by SPFMiCs whose labels represent different NPs and are created along the online phase. It is checked if the instance can be considered as part of any SPFMiC representing NPs and, if it is, it receives the NP label. However, if the new instance does not belong to any NP, it is tagged as unknown and stored in the unknown temporary memory for further analysis.

When the unknown temporary memory reaches a predefined number of instances, its instances are clustered, generating new SPFMiCs, which are analyzed to find cohesive and representative clusters that correspond to a new NP or an extension of an already known NP. In both cases, these SPFMiCs are added to the unsupervised decision model and used to classify new unlabeled instances that were marked as unknown.

When the labeled temporary memory is full, the labeled instances are clustered, generating SPFMiCs, which are analyzed looking for similarities with the NPs in the unsupervised model. If an SPFMiC generated by the labeled data is similar to a NP, this NP is removed from the unsupervised classifier and all the new SPFMiCs are included in the supervised model.

The algorithm removes SPFMiCs that have not classified any instances for a certain amount of time from the supervised decision model.

5.2 *Offline* Phase

The *offline* phase, detailed in Algorithm 1, is performed only once, being responsible for training the initial supervised decision model. The *offline* phase receives as input parameters the labeled dataset ($initPoints$), the fuzzification parameter (m) and the number of clusters per class (K).

Algorithm 1 *Offline* step

Input: $initPoints, m, K$
Output: $model$
1: $model \leftarrow \varnothing$
2: **for all** Class C_i in $initPoints$ **do**
3: $cluster\_li \leftarrow$ FCM$(model, m, K, C_i)$
4: $SFMiC\_Ci \leftarrow$ SUMMARIZE$(cluster_li)$
5: $model \leftarrow model \cup SFMiC\_Ci$
6: **end for**
7: **return** $model$

In Algorithm 1, instances of the same class are clustered in K clusters using FCM algorithm (step 3). The clusters found are summarized in SPFMiCs (SILVA; CAMARGO, 2020) (step 4) and incorporated into the decision model (step 5). Finally, the initial supervised decision model $model$ is returned (step 7).

As the proposed algorithm does not update the decision model incrementally, several of the SPFMiC components used in PFuzzND algorithm are not necessary. For this reason, the SPFMiC structure used here has a different structure compared to the one used in the PFuzzND algorithm. Our version of SPFMiC is defined as a vector ($centroid$, $gamma$, dp, N, t, $class\_id$), where $centroid$ is the center of the micro-cluster obtained by the FCM clustering; $gamma$ is a value defined by the equation 4.2 used to calculate the typicality; dp is the value of the fuzzy dispersion of the micro-cluster defined according to Equation 4.4; N is the number of instances associated with the micro-cluster; t is a parameter that indicates the last time the micro-cluster classified an instance and $class\_id$ is the class associated with the micro-cluster.

5.3 *Online* Phase

The *online* phase is responsible for three operations: classifying unlabeled instances, detecting novelties and updating decision models. Algorithm 2 presents details of this phase, which receives as input parameters the data stream (DS), the initial supervised decision model ($model$), the fuzzification parameter (m), values necessary for the calculation of typicality (α, β, K, n), the number of clusters (k_short) for the ND, the minimum number of instances to run the ND (T), the number of instances to generate new SPFMiCs in a supervised manner (C), the time limit (ts) for removing SPFMiCs and old instances, the minimum number of instances in a valid SPFMiC (min_weight), the multiplying factor (F) used to define the maximum decision boundaries of a SPFMiC, and the latency time (lat_val) (in number of instances).

The decision models, either supervised or unsupervised, classify instances using the concept of decision boundary of its micro-clusters. The fuzzy dispersion measure dp, stored in the SPFMiCs, is the fuzzy radius of the micro-clusters. The maximum decision boundary of a micro-cluster is defined by the fuzzy radius multiplied by a F factor. An instance is classified by

Algorithm 2 *Online* phase

Input: $DS, model, m, \alpha, \beta, K, n, k\_short, T, C, ts, min\_weight, F, lat\_val$

1: $unk\_mem \leftarrow \varnothing$
2: $lab\_mem \leftarrow \varnothing$
3: $model\_uns \leftarrow \varnothing$
4: **while** !ISEMPTY(DS) **do**
5: $x \leftarrow$ NEXT(DS)
6: **if** ISNULL($x.class$) **then**
7: **if** !ISFOUTLIER($model, x$) **then**
8: $x.class \leftarrow$ CLASSIFY($model, x, F$)
9: **else if** !ISFOUTLIER($model\_uns, x$) **then**
10: $x.class \leftarrow$ CLASSIFY($model\_uns, x, m, n, K, \beta$)
11: **else**
12: $unk\_mem \leftarrow unk\_mem \cup x$
13: **if** $|unk\_mem| \geq T$ **then**
14: $model \leftarrow$ ND($unk\_mem, m, k\_short, model$)
15: **end if**
16: **end if**
17: $ts \leftarrow x.time$
18: $model \leftarrow$ REMOLDSPFMIC($model, ts + lat\_val$)
19: $unknown\_mem \leftarrow$ REMUNK(unk_mem, ts)
20: **else**
21: $lab\_mem \leftarrow lab\_mem \cup x$
22: **if** $|lab\_mem| \geq C$ **then** UPDATE($lab\_mem, model, model\_uns, m, K$)
23: **end if**
24: **end if**
25: **end while**

a model if it is inside the maximum decision boundary of at least one of the SPFMiCs of models, that is, the distance between the instance and the centroid of some SPFMiC is less than the SPFMiC maximum decision boundary. When an instance is not classified by a decision model, it is called a filtered outlier (Foutlier) for this model.

In Algorithm 2, each instance that arrives is analyzed to see if it is a training (labeled) or classification (unlabeled) instance (step 6). If the instance has no label, the algorithm checks whether this instance represents a filtered outlier (Foutlier) for the supervised decision model (step 7). If the instance does not represent a Foutlier for the supervised decision model, it is classified (step 8), receiving the label of the SPFMiC to which the instance has the highest typicality. If the instance represents a Foutlier for the supervised decision model, it is submitted for classification using the unsupervised decision model that maintains information regarding the NPs found so far.

If the instance does not represent a Foutlier for the unsupervised model (step 9), it receives the label of the SPFMiC in the unsupervised model to which it has the highest typicality (step 10). Note that, at this time, the label of the SPFMiC represents a NP. Whenever an unlabeled

instance is marked as Foutlier by the two decision models, it is added to the unknown memory (step 12). After processing each instance, the algorithm checks if the unknown memory has the minimum number of instances to carry out the ND process (step 13), described in section 5.3.2.

After these steps, the algorithm updates the time counter (step 17), removes old SPFMiCs from the supervised decision model, which are older than $(ts + lat\_val)$ (step 18), and the unknown instances that have been inserted in the temporary memory of unknown instances before $(ts - current\_time)$ time units (step 19).

If the instance is labeled, it is stored in the labeled temporary memory (step 21) to be used to update the decision model (step 22), described in section 5.3.2.

5.3.1 Novelty Detection

Whenever a new unknown instance is added to the temporary memory ($unknown\_mem$), the algorithm checks if it has a minimum number of instances to start the ND process. Algorithm 3 represents details of this step, where it receives as input parameters the instances stored in the temporary memory of unknowns ($unknown\_mem$), the fuzzification parameter (m), the number of clusters (k_short), the supervised decision model ($model$) and the unsupervised decision model ($model\_uns$).

Algorithm 3 Novelty Detection Step

Input: $unknown\_mem, m, k\_short, model, model\_uns$
1: $temp\_clustes \leftarrow$ FCM$(unknown_mem, m, k_short)$
2: $all\_clustes \leftarrow model \cup model\_uns$
3: **for all** Cluster $cluster\_li$ in $temp\_clustes$ **do**
4: **if** VALIDATE$(cluster_li)$ **then**
5: $SFMiC_{NP} \leftarrow$ SUMMARIZE$(cluster_li)$
6: **for all** $SFMiC_{class=NP\#}NP_i \in all\_clustes$ **do**
7: $FR \leftarrow$ SIMILARITY$(NP_i, cluster_li)$;
8: $all\_FR \leftarrow all\_FR \cup FR$;
9: **end for**
10: $(label_{max}, max\_FR) \leftarrow$ MAX(all_FR);
11: **if** $max\_FR \geq \sigma$ **then**
12: $SFMiC_{NP}.class \leftarrow label_{max}$;
13: $model\_uns \leftarrow model\_uns \cup SFMiC_{NP}$;
14: **else**
15: $label_{new} \leftarrow$ NEW_NP_LABEL$(model_uns)$;
16: $SFMiC_{NP}.class \leftarrow label_{new}$;
17: $model\_uns \leftarrow model\_uns \cup SFMiC_{NP}$;
18: **end if**
19: **end if**
20: **end for**

In Algorithm 3, firstly, the instances stored in the temporary memory unk_mem are subjected to a clustering process using the FCM algorithm, resulting in k_short clusters (step 1). Each resulting cluster is subjected to a validation process, which consists of analyzing whether the fuzzy silhouette coefficient (CAMPELLO; HRUSCHKA, 2006) is greater than 0 and whether the cluster has a minimum number of instances (min_weiht) (step 4) .

To assign a label to a valid cluster, the algorithm calculates the fuzzy similarity FR (XIONG et al., 2012), defined by Equation 4.5, for all SPFMiCs of the decision model (steps 6, 7 and 8). If the highest FR between the validated cluster and an SPFMiC is greater than a user-defined threshold σ, the valid cluster represents an extension of a known NP, therefore, the cluster receives the label of the SPFMiC with the greatest similarity with it (steps 11 , 12 and 13). Otherwise, a new label of the form $NP\#$ is generated for the valid cluster and it is added to the unsupervised decision model $model\_uns$ (steps 15, 16, and 17) representing a new NP.

5.3.2 Updating decision models

Whenever a labeled instance arrives, it is added to the labeled temporary memory (*labeled_men*) and the algorithm checks if it has the minimum number of instances to perform the decision models update. Algorithm 4 presents details of this step, which takes as input parameters the labeled instances stored in the buffer of labeled instances ($labeled\_mem$), the fuzzification parameter (m), the number of clusters (K), the supervised decision model ($model$) and the unsupervised decision model ($model\_uns$).

Algorithm 4 Updating the decision model

Require: $labeled\_mem, m, K, model, model\_uns$
1: $chunk\_clustes \leftarrow \varnothing$
2: **for all** Class li in $labeled\_mem$ **do**
3: $cluster\_li \leftarrow$ FCM$(model, m, K, li)$
4: $SFMiC_{NP} \leftarrow$ SUMMARIZE$(cluster_li)$
5: $chunk\_clusters \leftarrow chunk\_clusters \cup SFMiC_{NP}$
6: **for all** $SFMiC_{class=NP\#} NP_i \in model\_uns$ **do**
7: $FR \leftarrow$ SIMILARITY$(NP_i, SFMiC_{NP})$;
8: $all\_FR \leftarrow all\_FR \cup FR$;
9: **end for**
10: $(index_{max}, max\_FR) \leftarrow$ MAX(all_FR);
11: **if** $max\_FR \geq \sigma$ **then**
12: $model\_uns \leftarrow$ REMSPFMIC$(model_uns, index_{max})$
13: **end if**
14: **end for**
15: $model \leftarrow model \cup chunk\_clustes$

In Algorithm 4, first, for each class of the labeled temporary memory ($labeled\_mem$), its corresponding instances are clustered by the FCM algorithm (step 4), resulting in K clusters by class. The clusters found are summarized in SPFMiCs (step 5) and each one of them is compared to SPFMiCs belonging to the unsupervised decision model (steps 6, 7, 8 and 9). If there are micro-clusters in the unsupervised model whose FR similarities between them and the new micro-cluster are greater than a threshold σ, the one with the highest FR is removed from the unsupervised model (steps 10 and 11). Thus, for each micro-cluster found from the labeled data, it is possible to eliminate up to one micro-cluster from the NPs model. This elimination is

based on the assumption that, if a micro-cluster formed from recently received labeled data is sufficiently similar to a micro-cluster of a NP, the examples previously identified as novelty are, in fact, examples of one of the classes that were discovered with this update process. Finally, the generated SPFMiCs are incorporated into the supervised decision model (step 14).

Chapter 6
EXPERIMENTAL EVALUATION

This chapter describes the experiments performed to evaluate the algorithm described in this work. The experiments were defined based on available software tools, synthetic and real datasets and algorithms available in the literature.

In the next subsections, it will be presented the details of the experiments performed, the synthetic and real datasets used, the algorithms selected for comparison and the settings and parameters used in each algorithm. The results of the experiments will be presented and discussed in Chapter 7.

6.1 Datasets

To carry out the experiments, three synthetic datasets and two real datasets were used. The description of each dataset is presented below:

- *MOA3* (FARIA et al., 2016b): Dataset generated using a function available in the *Massive Online Analysis* framework [2] (MOA). This dataset is formed by non-stationary hyperspherical clusters (concept drift). Also, for every 30,000 instances, new classes appear and old classes disappear (concept evolution).
- *RBF* (SILVA; CAMARGO, 2020): Dataset generated using a random radial function available in the MOA tool. This dataset shows only the disappearance and appearance of new classes (concept evolution) for every 10,000 instances.
- *SynEDC* (MASUD et al., 2011): Dataset generated using Gaussian distribution available in the MOA tool, using means between -5 and 5 and variance per class from 0.5 to 5. This dataset shows the appearance and disappearance of classes (concept evolution) and changes in the distribution of attributes over time (concept drift).
- *KDD99* (CUP, 1999): Dataset containing real traffic data in computer networks. The original dataset has 4 million instances, 41 attributes and 23 labels. In our experiments, we

[2] Available at: https://moa. cms.waikato.ac.nz/

used a reduced version, with 10% of the data, containing 490 thousand instances. With the pre-processing performed, the dataset used has 34 numerical attributes and a class attribute (with 5 possible values).

- *CoverType* (UCI, 1998): Dataset containing real data on the type of forest cover in an area. The dataset contains 581,000 instances, 54 attributes and 7 labels.

Table 2 presents the details of each dataset. Column #Instances represents the total number of instances, column #Instances (Offline) represents the number of instances used for training in the *offline* phase and #Attributes indicates the total number of attributes for each dataset. The total number of classes, available during the *online* phase, is presented in column #Classes and the number of classes available during the training of the initial decision model in the *offline* phase is presented in the column #Classes (Offline). The difference between the values of the last two column in the table is the number of classes that are not seen during training and are supposed to be identified as novelties in the *online* phase.

Table 2 – Datasets used in the experiments.

Identifier	#Instances	#Instances (Offline)	#Attributes	#Classes	#Classes (Offline)
MOA3	100,000	10,000	4	4	2
RBF	48,588	2,000	2	5	3
SynEDC	400,000	30,000	54	20	6
KDD	490,000	48,791	34	5	2
CoverType	581,000	47,045	54	7	2

6.2 Comparison algorithms and experimental setup

The algorithm proposed in this work for ND in DS, called EFuzzCND, is based on ECSMiner and PFuzzND methods. The method ECSMiner and the one proposed here deal with intermediate latency and the main difference between them is the use of fuzzy clustering in EFuzzCND and hard clustering in ECSMiner. Considering that ECSMiner is one of the main algorithms available for ND in DS in the literature, it was selected to be used as a basis for comparison and evaluation of our proposal considering intermediate latency.

In addition, we present an evaluation of our proposal considering extreme latency. In this set of experiments, we use as basis for comparison and evaluation the MINAS and PFuzzND algorithms, both originally developed to deal with extreme latency.

For the methods used in the comparison, we sought to apply the parameters described in their respective proposals. Tables 3, 5, 7, 6 show the parameters used in each method and their respective values. The values presented in this section were defined empirically, according to results obtained in preliminary experiments. Each experiment was run 5 times in order to obtain an average of the metrics considered for evaluation.

Table 3 – Parameters used in the experiments for the EfuzzCND algorithm.

EFuzzCND			
Step	Parameter	Parameter description	Value
offline	m	fuzzification parameter used in the FCM method	2
	K	number of clusters per class	*
Online	m	fuzzification parameter used in the FCM method	2
	α	parameter used in centroid calculation	1
		parameter used in the calculation of typicality and centroid	1
	n	parameter used in typicality calculation	2
	K	number of clusters	*
	T	minimum number of instances to run ND	*
	C	minimum number of instances to update the supervised decision model	500
	ts	timeout for removing old micro-clusters and instances	200
	F	micro-cluster radius multiplication factor	*
	lat_val	latency time (in number of instances)	*
DN	k_short	number of clusters	*
	min_weight	minimum number of instances in a valid cluster	*
	m	fuzzification parameter used in the FCM method	2
	ϕ	fuzzy similarity threshold	0,5

The analysis of the results of preliminary experiments showed that some parameters had greater sensitivity for different datasets. Table 4 presents the values of the parameters K, k_short, T, F and min_weight, separated by dataset, for the EFuzzCND algorithm.

The variation on the number of micro-clusters was due to the high dimensionality and complexity of some datasets. Simpler datasets, with little overlapping classes and little noise, do not need many micro-clusters. However, complex datasets, with a lot of overlapping classes and noisy data require a larger number of micro-clusters. Furthermore, in these cases, these micro-clusters should cover a smaller and more specific amount of data, as this can directly influence the classification of noisy data.

Table 4 – Variable parameters, separated by dataset, used in the experiments for the EfuzzCND algorithm.

Identifier	**K**	**k_short**	**T**	**F**	**min_weight**
MOA3	4	4	40	4	25
RBF	4	4	40	4	25
SynEDC	8	8	80	4	25
KDD99	8	8	80	1.5	20
CoverType	8	8	80	4	25

For the ECSMiner algorithm, the values defined in (MASUD et al., 2011) were used, with the exception of Tc, which was defined as 500. The choice of K-NN as classifier was made to make the approach used by ECSMiner closer to the functioning of the EFuzzCND.

Table 5 – Parameters used in the experiments for the ECSMiner algorithm.

ECSMiner			
Step	Parameter	Parameter description	Value
offline	classifier	classifier that will be used	K-NN
	K	number of cluster	50
Online	classifier	classifier that will be used	K-NN
	Tc	maximum time to classify a new instance	500
	Tl	latency time to get the label of an instance	*
	num_classifiers	maximum number of classifiers in the ensemble	6
	minPts	number of instances to run ND process	50
	chunk_size	number of instances to train a new decision model	1,000
ND	q	used in calculating the q-NSC of the ND process	50

The parameters lat_val (EFuzzCND) and Tl (ECSMiner) represent the latency time that the algorithm will take to receive instances with true labels. As we consider different latency values to evaluate these algorithms, these parameters do not have fixed values, varying according to the experiment. In this case, we considered 2,000, 5,000 and 10,000 latency values.

In the extreme latency experiments, only the EFuzzCND was evaluated. In this case, the value of lat_val was always defined as the number of instances of the dataset + 1.

The parameter values used for the MINAS method are presented in Table 6 and follow the values proposed in the experiments in (FARIA et al., 2016b).

Table 6 – Parameters used in the experiments for the MINAS algorithm.

MINAS			
Step	Parameter	Parameter description	Value
offline	algOff	clustering algorithm	K-Means
	k_class	number of groups per class	100
Online	algOn	clustering algorithm	Clustream
	threshold	threshold for identifying novelty patterns and class extensions	1.1
	thresholdStrategy	strategy used to calculate the threshold	TV1
	T	minimum number of instances to run ND	2000
	P	micro-cluster deletion time threshold	4000
	ts	temporal threshold for removing unknown instances from temporary memory	4000
	window_size	interval where the removals mentioned above occur	4000
DN	k_short	number of clusters	100
	numMinExCluster	minimum number of instances in a valid cluster	*

The parameter values used for the PFuzzND method are presented in Table 7 and follow the values proposed in the experiments described in (SILVA; CAMARGO, 2020). As with EFuzzCND, PFuzzND has parameters that are sensitive to the dataset and had to be adapted for

each dataset. These parameters are described in Table 8.

Table 7 – Parameters used in the experiments for the PfuzzND algorithm.

PFuzzND			
Step	Parameter	Parameter description	Value
offline	m	fuzzification parameter used in the FCM method	2
	k_class	number of clusters per class	*
Online	m	fuzzification parameter used in the FCM method	2
	n	parameter used in typicality calculation	2
	α	parameter used in centroid calculation	1
		parameter used in the calculation of typicality and centroid	1
	$init\_\theta$	initial classification threshold	*
	θ_adapt	adaptation threshold used in the classification process	*
	θ_class	adaptation threshold used in the classification process	
	T	minimum number and instances to run ND	40
	P	temporal threshold of removal of micro-clusters	500
	ts	temporal threshold of removal of unknown instances from temporary memory	*
	window_size	interval where the removals mentioned above occur	*
ND	k_short	number of clusters	*
	min_weight	minimum number of instances in a valid cluster	*
	m	fuzzification parameter used in the FCM method	2
	ϕ	fuzzy similarity threshold	0,5

Table 8 – Variable parameters, separated by dataset, used in the experiments for the PFuzzND algorithm.

	$init\_\theta$	θ_class	θ_adapt	k_class	k_short	ts	min_weight
MOA	0.95	0.00	0.90	4	4	200	25
RBF	0.95	0.00	0.90	4	4	200	25
SynEDC	0.90	0.00	0.58	8	8	200	25
KDD99	0.99	0.06	0.90	8	8	1000	10
CoverType	0.99	0.00	0.90	8	8	1000	10

In our proposal, we present improvements in relation to the initialization parameters of the PFuzzND using the fuzzy dispersion to define the decision boundaries of the SPFMiCs, with this it was possible to remove the parameters θ_class and θ_adapt, avoiding the highest frequency for using the resource.

6.3 Evaluation Metrics

To evaluate the methods mentioned above in relation to EFuzzCND, the incremental rectangular confusion matrix proposed by Faria et al. (2016a) was used in the experiments. The incremental rectangular confusion matrix adapts to each new classified instance and the appearance and disappearance of classes, being ideal for carrying out ND evaluations in DS.

The incremental rectangular confusion matrix requires that each NPs be associated with a class. To define the class that is associated to a NP, the number of instances of each class in the NP is counted and, the class with the largest number of instances is selected. Details of the calculations to generate the incremental rectangular confusion matrix can be found in (FARIA et al., 2016a). Note that the same class can be associated with more than one NP. As the EFuzzCND makes this association between NPs and classes during the decision model maintenance, when labeled data arrive, that is, under intermediate latency, there was no need to perform any previous analysis on the results to prepare the matrix.

However, in experiments where we consider extreme latency, the algorithm does not update decision models based on labeled data because there is no external feedback. In this case, when assembling the incremental confusion matrix, it is necessary to define the class in which a certain NP will be associated. For this, the number of instances that the NP has for each class of the problem is verified. The predominant class will be associated with the NP and considered in the assessment of the accuracy of that NP.

To evaluate the performance of the algorithm over time, the evaluation metrics were calculated every time a sequence of 1000 instances from the DS have been processed. These moments of metrics calculation are called evaluation moments. For each evaluation moment, the accuracy was calculated, considering the instances classified as one of the known classes or one of the NPs. The number of instances not classified by the decision models, that is, marked as unknown, was evaluated using the measure Unknown Rate (UnkR), proposed by Faria et al. (2016a) and defined by:

$$Unkr = \frac{1}{\#C}(\sum_{i=1}^{\#C} \frac{\#UnkR_i}{\#Exc_i}) \tag{6.1}$$

where $\#C$ represents the total number of classes, $\#UnkR_i$ the number of instances belonging to class C_i classified as unknown and Exc_i the total number of instances belonging the class C_i. With this metric, it is possible to understand the behavior of the algorithm at times when a new class appears or the concept-drift occurs.

Chapter 7

RESULTS AND ANALYSIS OF EXPERIMENTS

This chapter presents the results obtained from the experiments performed based on the guidelines presented in the previous chapter. The analysis of the results is divided into two sections: the first one discusses the results obtained under intermediate latency and the second one discusses the results obtained under extreme latency.

7.1 Intermediate latency

In this section, a comparative analysis of the results obtained by the EFuzzCND and ECSMiner algorithms will be presented, considering intermediate latency, applying three different latency values: 2,000, 5,000 and 10,000. The results presented by the EFuzzCND algorithm are related to the fuzzy cluster characteristics. The results presented by the ECSMiner algorithm demonstrate results related to hard clustering characteristics. As they present similar classification and ND strategies, the comparison between these two algorithms can help us understand the advantages and disadvantages of using fuzzy and hard clustering in ND in DS.

7.1.1 MOA3

Figures 4, 5 and 6 show the results of accuracy and UnkR for the 90 moments of evaluation of the MOA3 dataset, considering different values of intermediate latency, for the EFuzzCND and ECSMiner algorithms, respectively. The black and dashed vertical lines represent the evaluation moments when a novelty arrived and the gray vertical lines represent the evaluation moments when the algorithm identified a NP. This applies to all graphics presented in this chapter.

The MOA3 dataset shows the emergence of novelties in two evaluation moments, being 25 and 60, represented by the black dashed vertical line. Figure 4a shows the EFuzzCND results considering a latency of 2,000. It is possible to notice that the algorithm correctly classified all instances and there was a sudden increase in UnkR at moments 25 and 60, when the novelty emerged. The algorithm managed to identify the new classes that emerged fast, since UnkR is reduced to almost zero at the next evaluation moments, 26 and 61. The UnkR measure had peaks

that reached 36% and 29% when the novelties appeared, but afterwards they started to decrease and remained stable, close to 0%.

Figure 4b presents the results of the algorithm for the latency value 2,000. It is possible to notice that ECSMiner identified NPs more times, compared to EFuzzCND, with occurrences in the evaluation moments: 25, 26, 27, 28, 60, 61, 62, 63 and 64. However, before the emergence of the first NP in the evaluation moment 25, ECSMiner kept the UnkR measure at zero, having two peaks of 34% and 27% and keeping the rate stable close to 5% during the vast majority of evaluation moments.

(a) EFuzzCND **(b) ECSMiner**

Figure 4 – Accuracy and UnkRs of the EFuzzCND and ECSMiner algorithms, for the MOA3 dataset and 2,000 latency.

Figure 5a shows the EFuzzCND results for the MOA3 dataset, considering latency of 5,000. It is notable that the behavior of the algorithm for this latency value was similar to the one of latency 2,000, identifying the novelties at the same evaluation moments. However, it took a bit longer for the algorithm to adjust to the appearance of the first novelty at moment 25, since the UnkR decreased more slowly until reach its minimum value at 5%.

Figure 5b presents the results of the algorithm considering the latency value 5,000. It is possible to notice that the accuracy of ECSMiner had some oscillations, which indicates that the algorithm was not able to classify correctly all the instances. In addition, the number of identified NPs increased, with occurrences at the moments of evaluation: 25, 26, 27, 28, 29, 30, 31, 60, 61, 62, 63, 64, 65, 66 and 67, with 6 occurrences more than in latency 2,000. The UnkR measure had an increase, peaking at 34% and 29% and keeping the rate stable between 8% and 20% during the most part of the evaluation moments.

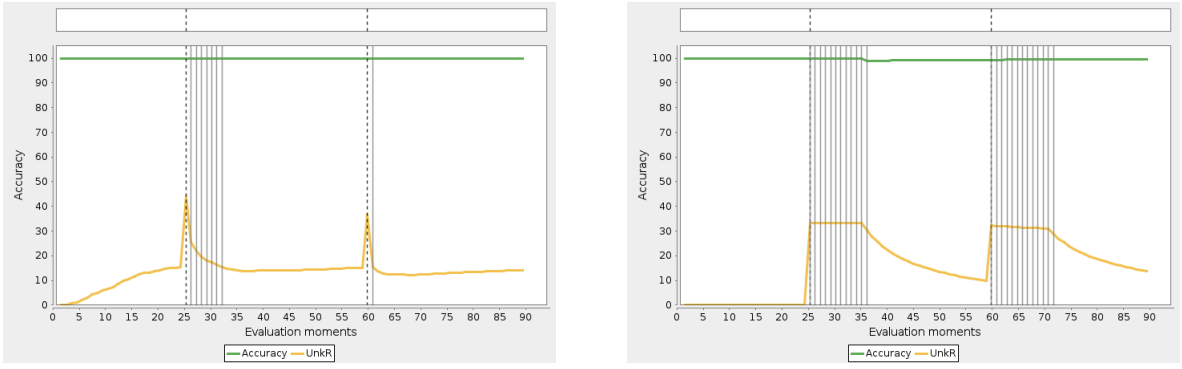
Figure 6a shows the EFuzzCND results for the MOA3 dataset, considering latency of 10,000. It is remarkable an increase in the UnkR measure in this scenario, reaching 45% and 38% with the emergence of novelties in evaluation moments 25 and 60 and remaining stable between 15% and 20% during most of the evaluation moments. This happened because of the longer time to obtain the true label, increasing the number of instances classified as unknown. It is possible to notice, by the higher number of vertical gray bars in the graph, that the number of identified NPs

(a) EFuzzCND **(b) ECSMiner**

Figure 5 – Accuracy and UnkRs of the EFuzzCND and ECSMiner algorithms, for the MOA3 dataset and 5,000 latency.

increased significantly. This is a reflection of the strategy used to update the decision models, which when the classes undergo concept evolution. The only way for the algorithm to adapt to the concept evolution is executing the ND, where a PN is identified and, through similarity, linked to a PN or an existing class. Because of this, the number of identified PNs and the UnkR increases.

ECSMiner behaved similarly when we increased the latency value to 10,000 (Figure 6b). The accuracy had some oscillations and the number of identified NPs increased, identifying a total of 22 NPs, 7 more compared to running with 5,000 latency and 14 more than running with 2,000 latency. In this scenario, the increase in the UnkR measure was even more significant, reaching peaks of 35% and remaining at high values, above 13%. This was mainly due to the delay that ECSMiner takes to start decreasing the UnkR after the arrival of a new class.



(a) EFuzzCND **(b) ECSMiner**

Figure 6 – Accuracy and UnkRs of the EFuzzCND and ECSMiner algorithms, for the MOA3 dataset and 10,000 latency.

Making a comparative analysis between the two algorithms for the MOA3 dataset, it is possible to notice that the EFuzzCND adapted better to the different latency values. For all novelties that arrived, the EFuzzCND took only one evaluation moment to identify the NP that emerged and start to reduce the UnkR regardless of the latency value, while the ECSMiner took

2 evaluation moments for the 2,000 latency, 6 evaluation moments for the 5,000 latency and 11 evaluation moments for latency of 10,000, a behavior that causes a large increase of UnkR.

Furthermore, EFuzzCND always kept the accuracy at 100%, showing that the algorithm got right all the instances it could classify, while ECSMiner had small oscillations, showing that the algorithm missed some instances it tried to classify. However, the UnkR of EFuzzCND was higher, maintaining a stable value throughout the DS, while ECSMiner kept at zero until novelties appeared.

Figure 7 presents a comparison of UnkR for EFuzzCND and ECSMiner in the evaluation moments close to the arrival of the first novelty. This comparison makes it easy to analyze the behavior of this metric between the two algorithms. The latency value of 10,000 was selected because it is what the algorithms had the greatest impact on the considered metric. As already reported, it is possible to notice that EFuzzCND obtains a higher value of UnkR when a novelty appears, however, it quickly identifies the novelty and starts to classify instances of this novelty at the next evaluation moment, while ECSMiner takes much longer time to start reducing the UnkR.

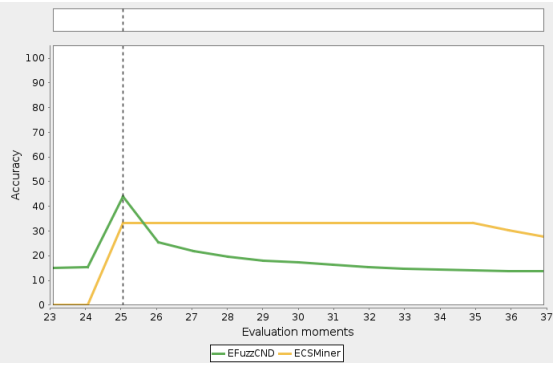


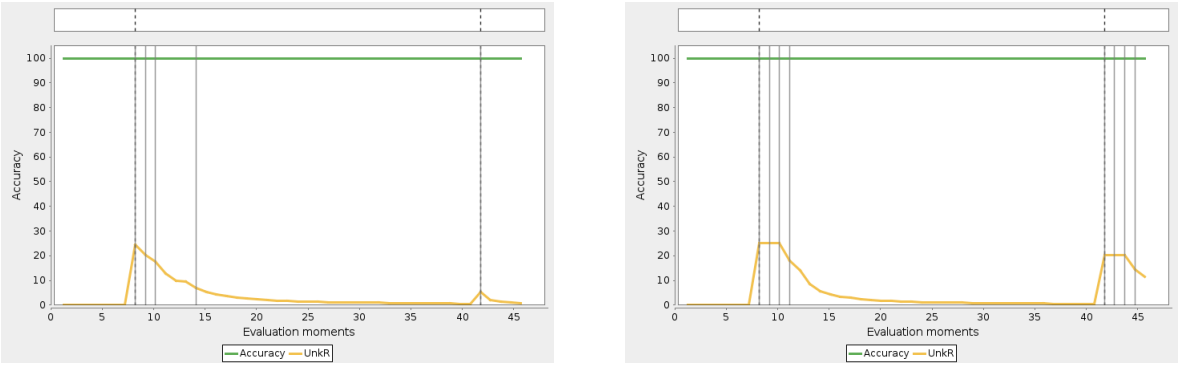
Figure 7 – UnkR comparison of the EFuzzCND and ECSMiner algorithms, for a latency of 10,000, in the evaluation moments close to the first novelty that appeared in the MOA3 dataset.

7.1.2 RBF

Figures 8, 9 and 10 show the results of accuracy and UnkR for the 46 evaluation moments of the RBF dataset, considering different values of intermediate latency, for the EFuzzCND and ECSMiner algorithms, respectively. This dataset has the emergence of two novelties, at evaluation times 8 and 42.

Figure 8a presents the results obtained by the EFuzzCND considering the 2,000 latency value. It is possible to notice that the algorithm successfully identified the novelties that emerged. NPs were identified in the evaluation moments 8, 9 10, 14 and 42. As for the UnkR measure, the algorithm kept the value zeroed until the appearance of the novelty and after that, had two peaks, when the novelties appeared, of 28% and 7% and in general, it kept below 5%.

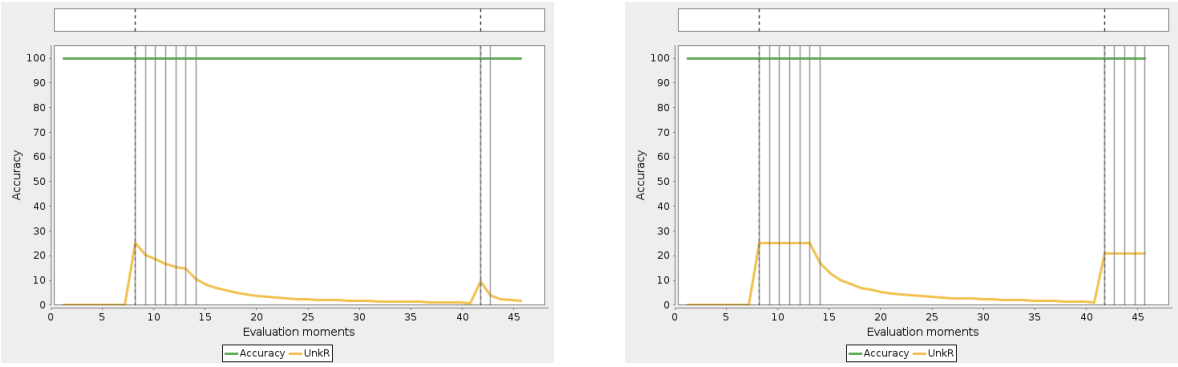
Figure 8b presents the results of the ECSMiner algorithm considering 2,000 latency. Its possible to notice that the algorithm identified 8 NPs, in the evaluation moments: 8, 9, 10, 11, 42, 43, 44 and 45. Furthermore, the algorithm always kept the UnkR stable, rising only when novelties appeared. However, when the novelties appeared, the UnkR measure reached 25% in the first novelty and 21% in the second, in addition to taking two moments of evaluation to start decreasing.



(a) EFuzzCND **(b) ECSMiner**

Figure 8 – Accuracy and UnkRs of the EFuzzCND and ECSMiner algorithms, for the RBF dataset and 2,000 latency.

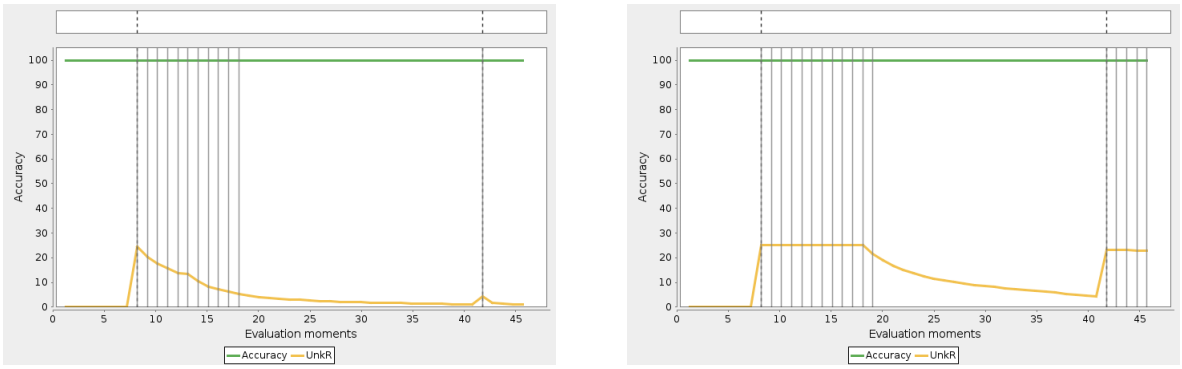
Figure 9a presents the results obtained by the EFuzzCND considering the 5,000 latency value. It is possible to notice that with increasing latency, the number of identified NPs also increased, as well as the rate of unknowns. This delay in identifying the novelty makes the UnkR remain high for more evaluation moments. Again, the UnkR measure had two peaks, reaching 29% and 10%. Between evaluation moments 8 and 13, the UnkR measure decreased slightly, remaining between 28% and 17%. From evaluation moment 14 onwards, the rate started to decrease until below 5%.



(a) EFuzzCND **(b) ECSMiner**

Figure 9 – Accuracy and UnkRs of the EFuzzCND and ECSMiner algorithms, for the RBF dataset and 5,000 latency.

Figure 10a presents the results obtained by the EFuzzCND algorithm considering 10,000 latency. It is possible to notice that the number of identified NPs increased considerably in



(a) EFuzzCND **(b) ECSMiner**

Figure 10 – Accuracy and UnkRs of the EFuzzCND and ECSMiner algorithms, for the RBF dataset and 10,000 latency.

relation to the other latency values. Furthermore, the UnkR measure remained very similar to the 5,000 latency value.

Figures 9b and 10b show the ECSMiner results for 5,000 and 10,000 latency values, respectively. ECSMiner maintained a pattern in the detections of NPs when a novelty class appears: the higher the latency value, the greater the number of NPs identified. For the latency value of 5,000, ECSMiner identified NPs at evaluation times 8, 9, 10, 11, 12, 13, 14, 42, 43, 44, 45 and 46. For the latency value 10,000, NPs were identified in the evaluation times 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 42, 43, 44, 45 and 46. In total, 8 NPs were identified for the 2,000 latency value , 12 NPs for 5,000 latency value and 17 NPs for 10,000 latency value. The UnkR was also affected directly and proportionally according to the latency value, mainly because ECSMiner took a few moments of evaluation to actually stop classifying novelty instances as unknown.

Making a comparative analysis between the two algorithms for the RBF dataset, it is possible to notice that both had 100% accuracy in all evaluation moments, showing that all the instances they could classify were correctly classified.

As for the UnkR measure, the two algorithms behaved diferently. in all the analyses, EFuzzCND identified the novelties at the evaluation moment in which they appeared and, in the following evaluation moments, the rate began to decrease. ECSMiner have also identified the novelties at the evaluation moment in which they appeared, but it took several evaluation moments to start having a decrease in the UnkR. It is possible to notice that this behavior gets worse with the increase of the latency value, showing that ECSMiner does not have as much flexibility in classifying outliers and noisy data. This resulted in an increase in the UnkR in ECSMiner results.

Figure 11 presents a comparative analysis of the UnkR obtained by the two algorithms, considering the latency value of 10,000, for the evaluation moments close to the emergence of the first novelty. It is possible to notice that until the emergence of the novelty the behavior of

the algorithms is the same. When the news hits, they both have a boost in UnkR. Again, the ability of EFuzzCND to quickly adapt its decision model to the new arrivals is clear, since at the moment of evaluation following the emergence of the novelty, the algorithm already starts to classify instances belonging to this novelty and decrease the UnkR, different from ECSMiner, which keeps the metric high for a long time.

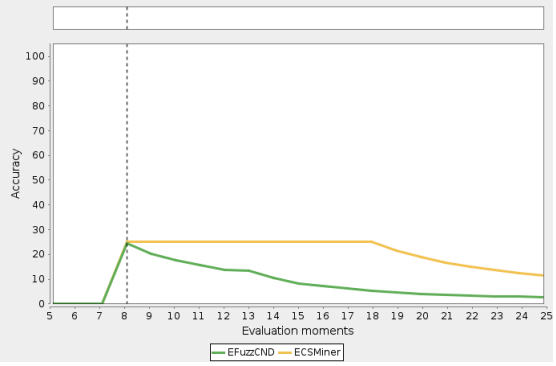
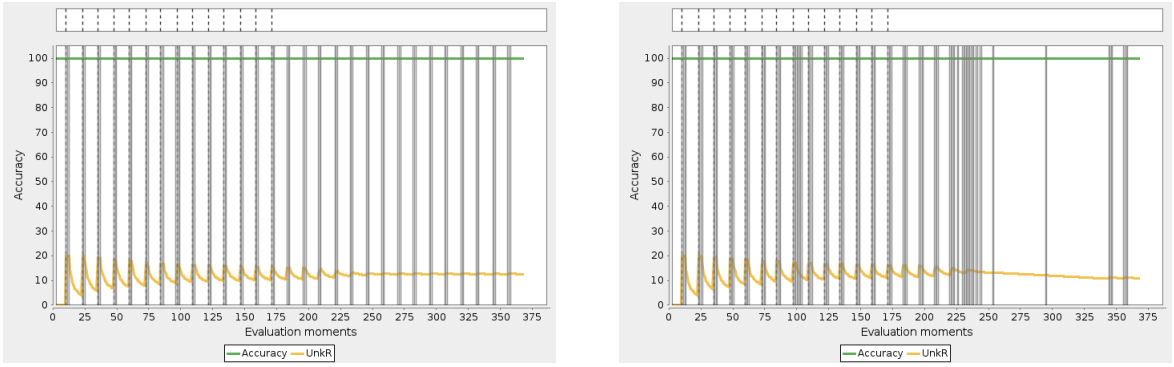


Figure 11 – UnkR comparison of the EFuzzCND and ECSMiner algorithms, for a latency of 10,000, in the evaluation moments close to the first novelty that appeared in the RBF dataset.

7.1.3 SynEDC

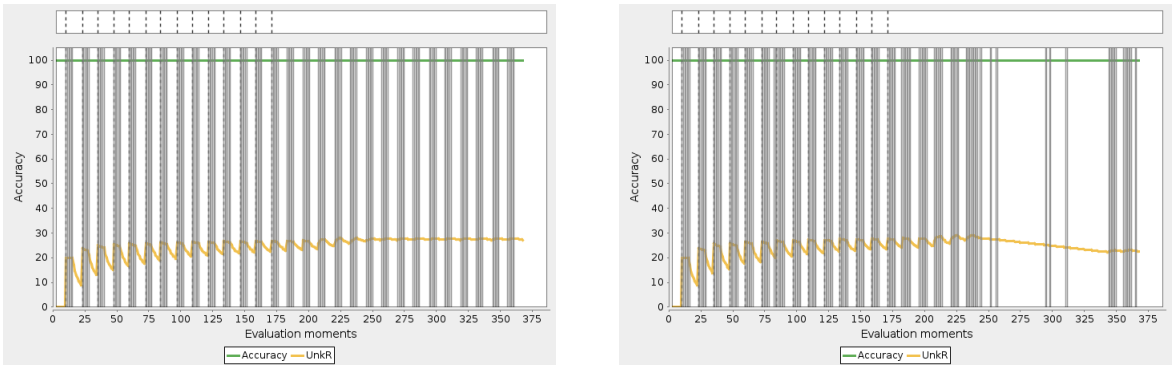
Figures 12, 13 and 14 present the results of accuracy and UnkR for the 370 evaluation moments of the SynEDC dataset, considering different values of intermediate latency, for the EFuzzCND and ECSMiner algorithms, respectively.

The SynEDC dataset has a non-stationary characteristic, that is, known classes change over time. In addition, this dataset has the emergence of new classes at evaluation times: 8, 21, 33, 46, 58, 71, 83, 96, 108, 121, 133, 146, 158 and 171. Some of these classes may reappear over time, but it is important to emphasize that EFuzzCND and ECSMiner do not treat recurrent classes. When a class that has appeared before emerges again, it is treated as a different novelty.



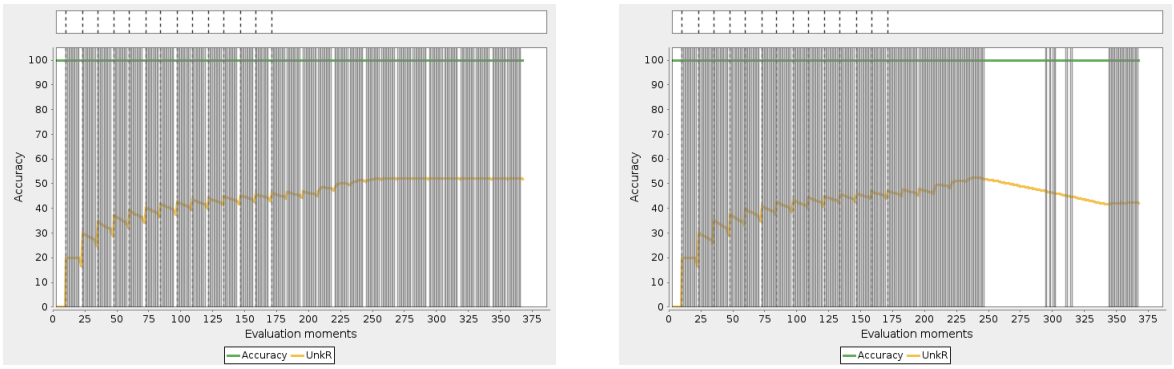
(a) EFuzzCND **(b) ECSMiner**

Figure 12 – Accuracy and UnkRs of the EFuzzCND and ECSMiner algorithms, for the SynEDC dataset and 2,000 latency.



(a) EFuzzCND **(b) ECSMiner**

Figure 13 – Accuracy and UnkRs of the EFuzzCND and ECSMiner algorithms, for the SynEDC dataset and 5,000 latency.



(a) EFuzzCND **(b) ECSMiner**

Figure 14 – Accuracy and UnkRs of the EFuzzCND and ECSMiner algorithms, for the SynEDC dataset and 10,000 latency.

For the SynEDC dataset, EFuzzCND and ECSMiner had similar results for all latency values. Both kept the accuracy always at 100%, showing that they got all the instances they tried to explain right.

Furthermore, both had similar behaviors in the UnkR measure, identifying the novelty at the same time of evaluation in which it appeared and starting to decrease the UnkR in the moments of subsequent evaluations. At this point, it is worth noting that ECSMiner identified much fewer NPs during the 250 and 340 evaluation moments. In addition, from the 250 evaluation moment the ECSMiner UnkR measure started to decrease, while the EFuzzCND remained stable.

Figure 11 presents a comparative analysis of the UnkR obtained by the algorithms, in moments of evaluation close to the emergence of the first three novelties, under a latency of 10,000 for the SynEDC dataset. It is possible to notice that for this dataset, until the emergence of the second novelty, the results were similar. From this, EfuzzCND showed a slight reduction in UnkR compared to ECSMiner.

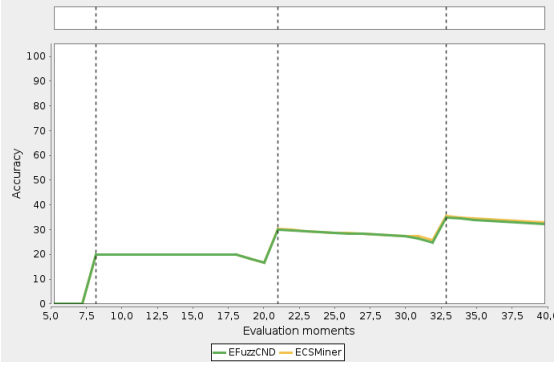
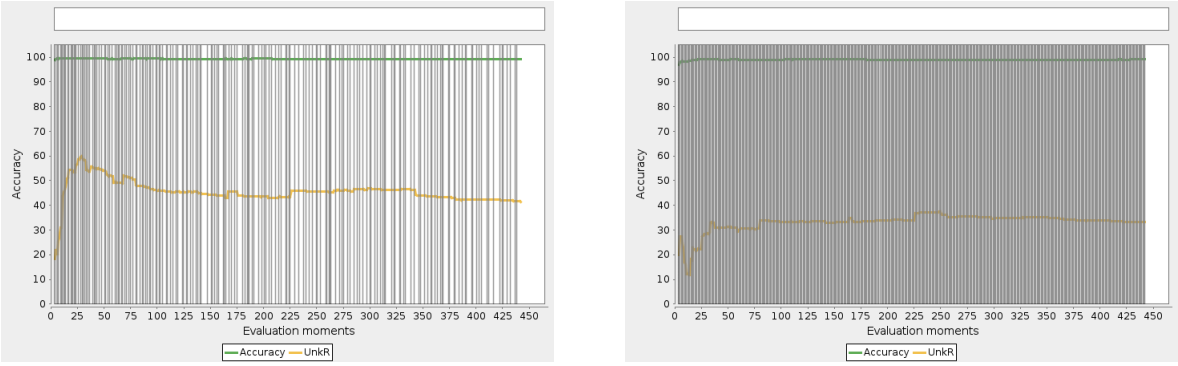


Figure 15 – UnkR comparison of the EFuzzCND and ECSMiner algorithms, for a latency of 10,000, in the evaluation moments close to the first three novelties that appeared in the SynEDC dataset.



(a) EFuzzCND **(b) ECSMiner**

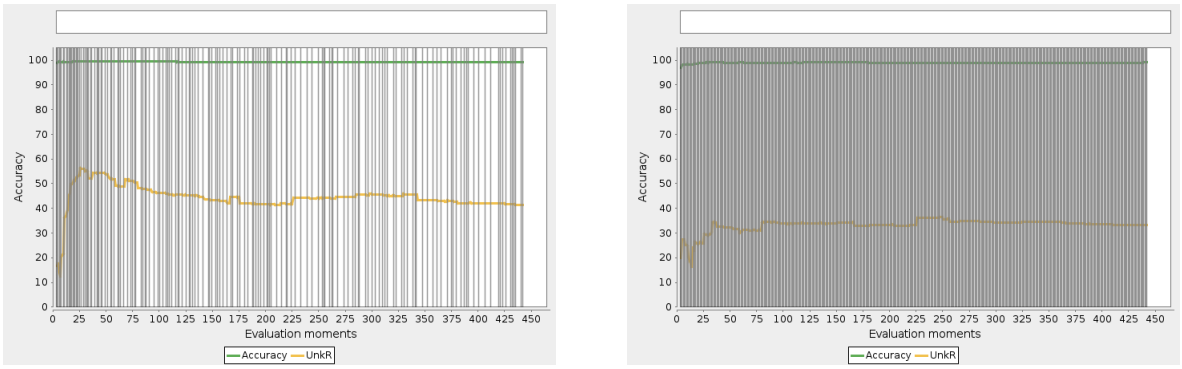
Figure 16 – Accuracy and UnkRs of the EFuzzCND and ECSMiner algorithms, for the KDD99 dataset and 2,000 latency.

7.1.4 KDD99

Figures 16, 17 and 18 present the accuracy and UnkR results for the 442 evaluation moments of the KDD99 dataset, considering different values of intermediate latency, for the EFuzzCND and ECSMiner algorithms, respectively. As already mentioned, KDD99 is a real dataset, with high dimensionality and complexity in data distribution. This and the other real datasets do not have dashed vertical lines, which indicate the real appearance of novelties, as their identification would demand a high cost of time.

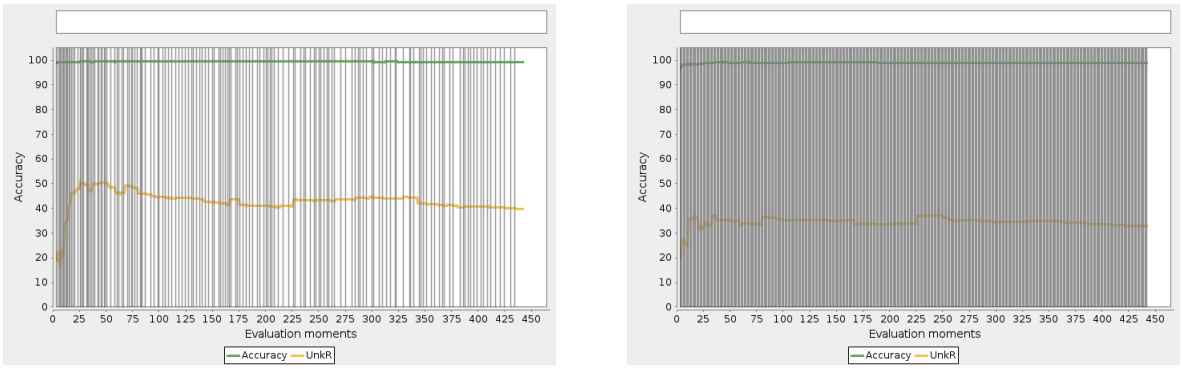
It is remarkable that for the KDD99 dataset the algorithms also obtained similar results. Both maintained the accuracy for the three latency values, close to 100%, showing that the rate of correct answers for the instances they tried to explain was very high.

Analyzing the UnkR measure, it is possible to notice that the EFuzzCND reached 60% under 2,000 latency, 58% under 5,000 latency and 50% under 10,000 latency. However, the algorithm kept the UnkR, in most of the evaluation moments, between 40% and 50%. ECSMiner managed to reduce this measure, reaching 38% under 2,000 latency, 40% under 5,000 latency, and 39% under 10,000 latency. Overall, in most evaluation moments, ECSMiner kept the UnkR



(a) EFuzzCND **(b) ECSMiner**

Figure 17 – Accuracy and UnkRs of the EFuzzCND and ECSMiner algorithms, for the KDD99 dataset and 5,000 latency.



(a) EFuzzCND **(b) ECSMiner**

Figure 18 – Accuracy and UnkRs of the EFuzzCND and ECSMiner algorithms, for the KDD99 dataset and 10,000 latency.

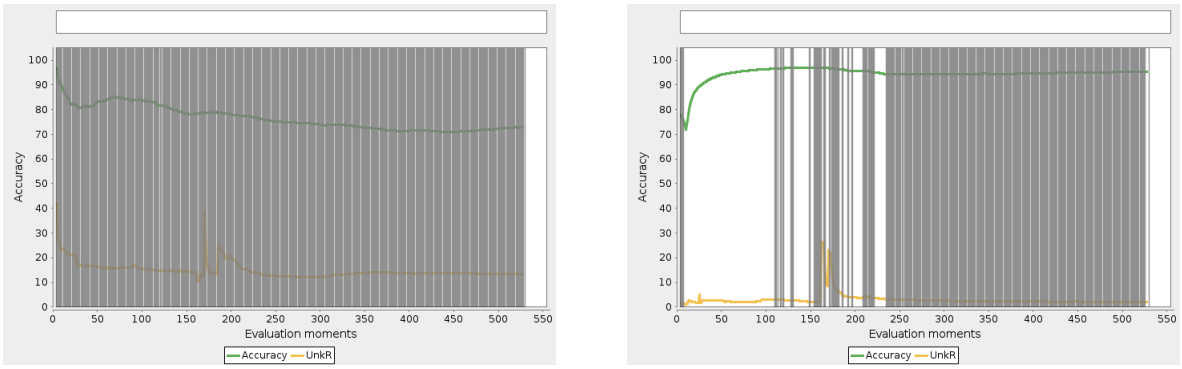
measure between 30% and 40%.

In all the analyses, it is possible to notice that the EFuzzCND identified a much lower amount of NPs, while the ECSMiner identified it in almost all evaluation moments. This explains the increase in the UnkR measure in relation to ECSMiner. For this dataset, due to the high complexity, overlapping classes and noisy data, in the ND process, the EFuzzCND needed more instances in the clusters to meet the predetermined criteria for a micro-cluster to be considered a NP.

7.1.5 CoverType

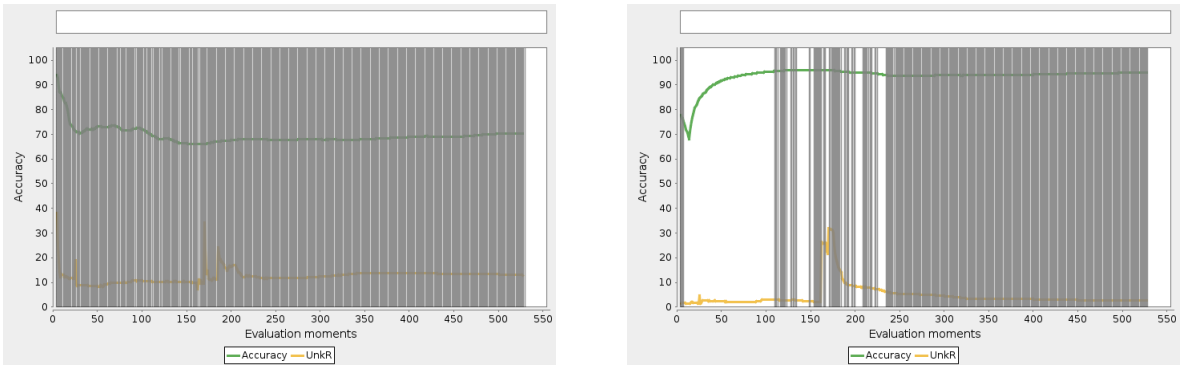
Figures 19, 20 and 21 present the accuracy and UnkR results for the 534 evaluation moments of the CoverType dataset, considering different values of intermediate latency, for the EFuzzCND and ECSMiner algorithms, respectively. CoverType is a real dataset that presents high complexity, being the most elaborated dataset selected to evaluate our proposal.

For this dataset the algorithms showed greater divergence in the results. The accuracy of



(a) EFuzzCND **(b) ECSMiner**

Figure 19 – Accuracy and UnkRs of the EFuzzCND and ECSMiner algorithms, for the CoverType dataset and 2,000 latency.



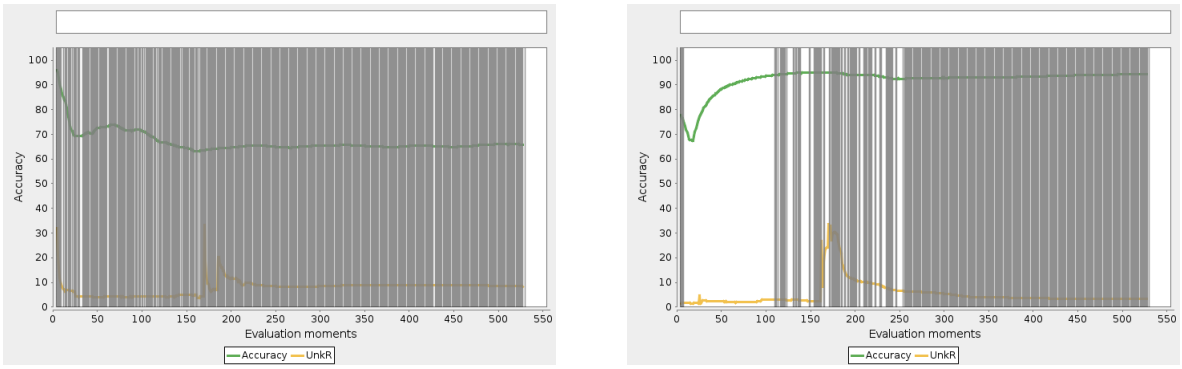
(a) EFuzzCND **(b) ECSMiner**

Figure 20 – Accuracy and UnkRs of the EFuzzCND and ECSMiner algorithms, for the CoverType dataset and 5,000 latency.

the EFuzzCND was similar for the three latency values, starting with high values, above 95% and after dropping, stabilizing at around 80% for the latency of 2,000, 70% for the latency of 5,000 and 65% for latency of 10,000. In this parameter, ECSMiner was more stable, starting at around 80% and then stabilizing around 90%, obtaining similar results for the three latency values.

Regarding UnkR, EFuzzCND marked more instances as unknown, always keeping the rate close to 20%, for latency of 2,000, 10% for latency of 5,000 and 5% for latency of 10,000, with some increases between evaluation moments 150 and 200 for all latency values.

Differently, ECSMiner kept the UnkR measure always below 5%, with the exception of the evaluation moments between 150 and 200, where the algorithm also had several moments where the rate increased. In addition, EFuzzCND identified more NPs, having occurrences in almost all evaluation moments, while ECSMiner significantly reduced the number of identified NPs, remaining unidentified between evaluation moments 6 and 110. This result was also also obtained thanks to the low values of UnkR.



(a) EFuzzCND **(b) ECSMiner**

Figure 21 – Accuracy and UnkRs of the EFuzzCND and ECSMiner algorithms, for the CoverType dataset and 10,000 latency.

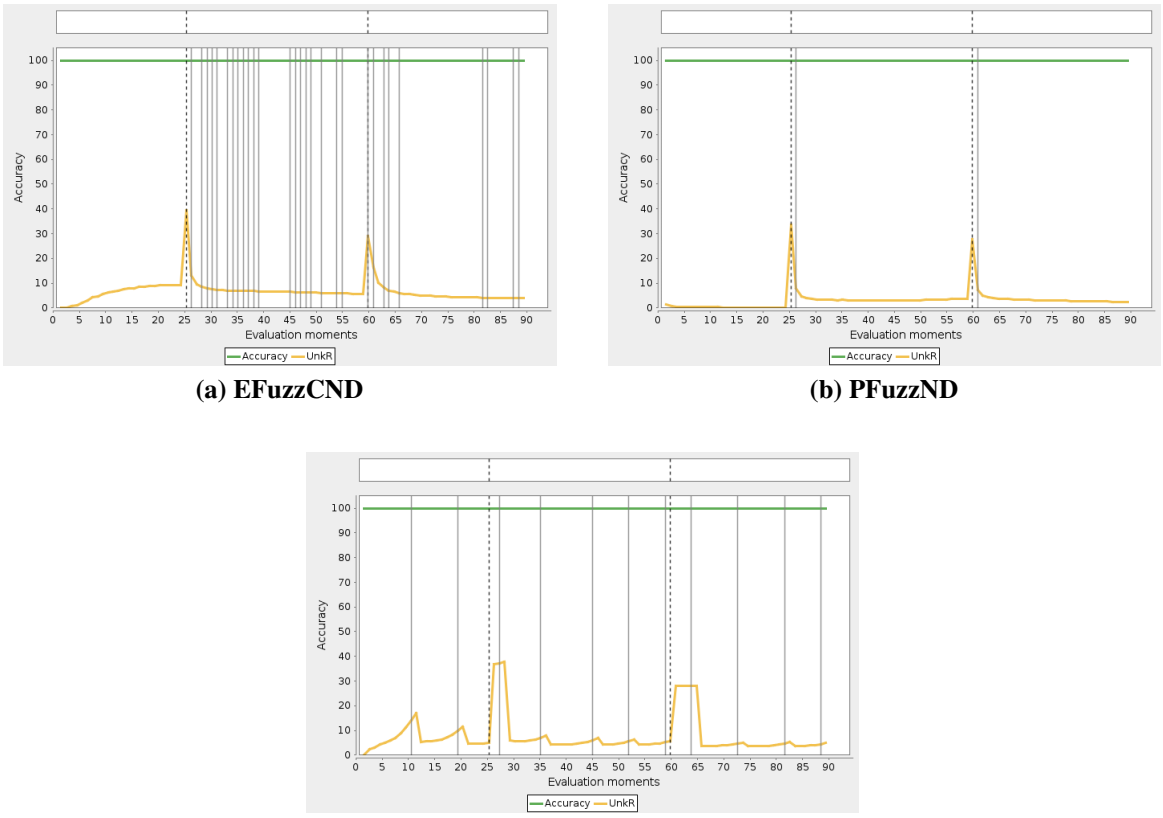
7.2 Extreme Latency

In this section, a comparative analysis of the results obtained by the EFuzzCND, PFuzzND and MINAS algorithms will be presented, under extreme latency, that is, without considering external feedback for the maintenance of decision models.

The EFuzzCND algorithm was built based on PFuzzzND, presenting some new features and improvements. The main difference between EFuzzCND and PFuzzND is the use of two decision models, which allows the use of external feedback to update them as a means to address concept evolution and concept drift. Furthermore, EFuzzCND presented a strategy in which it calculates the fuzzy radius, based on the instances that are part of the SPFMiC, to define the decision boundaries of each SPFMiC. With that, it was possible to eliminate PFuzzND initialization parameters that were defined in an empirical way, for each dataset, which makes its use difficult. In this way, with the obtained results, it will be possible to evaluate the proposed improvements in relation to the PFuzzND and an analysis of the fuzzy clustering (EFuzzCND and PFuzzND) in relation to the hard clustering (MINAS) in ND in DS.

7.2.1 MOA3

Figure 22 shows the results of the three algorithms for the MOA3 dataset. Figure 22a presents the results obtained by EFuzzCND algorithm. It is possible to notice that the algorithm managed to classify correctly all the instances. In addition, the algorithm identified correctly the novelties that emerged at evaluation moments 25 and 60, and it took only one evaluation moment, after the appearance of the novelty, to start reducing the UnkR. As it is an algorithm developed to use external feedback and not update SPFMiCs, the only way to update is through the identification of novelties and extensions of novelties, which explains the high number of identified NPs, when compared to other algorithms. The UnkR measure had a good result, being between 0% and 10%, with exceptions when the novelties appeared, when the measure reached 40% and 29%.



(a) EFuzzCND

(b) PFuzzND

(c) MINAS

Figure 22 – Algorithm accuracy and UnkRs, considering extreme latency, for the MOA dataset.

Figure 22b presents the results obtained by PFuzzND algorithm. The algorithm obtained good results, classifying all the instances that it could classify correctly and keeping the UnkR low, increasing only in moments when some novelty appeared. The functioning of PFuzzND was similar to EFuzzCND when a novelty arrived, the algorithm took only a moment of evaluation to identify it and then began to reduce the UnkR measure.

Furthermore, PFuzzND identified only two NPs, exactly the ones that refer to the novelties that arrived, that is, with the first instances that arrived, the algorithm was able to classify all the other instances correctly. This is due to the ability of the algorithm to adapt to changes incrementally. The initialization parameters of this algorithm are initialized empirically and favor directly the decision of the algorithm to classify an instance or mark it as unknown.

Figure 22c presents the results obtained by MINAS algorithm. It is possible to notice that the algorithm classified correctly all the instances it could classify. The UnkR measure increases in several moments throughout the DS. Furthermore, the algorithm identified several NPs, even before a novelty actually emerged, which may indicate a reaction to the class extensions or NPs identified in the ND process.

7.2.2 RBF

Figure 23 shows the results of the three algorithms for the RBF dataset. In Figure 23a the results obtained by EFuzzCND algorithm are presented, and it is notable that for this dataset, the algorithm also managed to classify correctly the instances it could classify. The UnkR measure started with a value of zero and only increased when the first novelty appeared, at the evaluation moment 7, reaching 26%. After the identification of the first novelty, it decreased until it stabilized close to 5% and rose again when the second novelty appeared, at the evaluation moment 42. In both cases where the novelty appeared, the algorithm took just a moment of evaluation to identify the novelty and start to reduce the UnkR. The algorithm identified several NPs after the appearance of the first novelty, between evaluation times 8 and 38, showing that the ND process was executed several times to identify class extensions and/or novelty extensions. Class extensions and novelty extensions are micro-clusters that have a high similarity with micro-clusters that belong to classes or novelty patterns that already exist in one of the models.

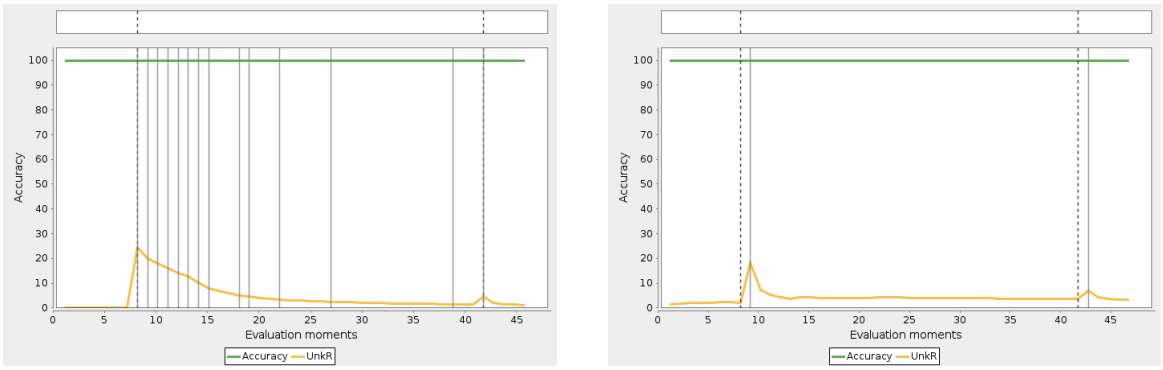
Figure 23b presents the results obtained by PFuzzND algorithm. The behavior of PFuzzND was similar to EFuzzCND, also keeping the accuracy at 100%, showing that it classified correctly all instances. The UnkR was also similar, PFuzzND showed two peaks, also when novelties appear, of 19% and 8%, respectively, and maintained, in most of the evaluation moments, the UnkR measure close to 5 %. When the algorithm identified the novelties, at the next evaluation moment, the UnkR measure had already started to decrease.

Figure 23c presents the results obtained by MINAS algorithm, which also managed to classify all the instances it could. The algorithm showed high UnkR peaks, reaching 35% during evaluation moments 9 and 11 and 30% during evaluation moments 43 and 46. In addition to maintaining this measure, during most of the evaluation moments, above 15%. The data show that MINAS was able to identify the novelties, but with a delay, in addition to taking a long time to start reducing the UnkR after identifying the NPs.

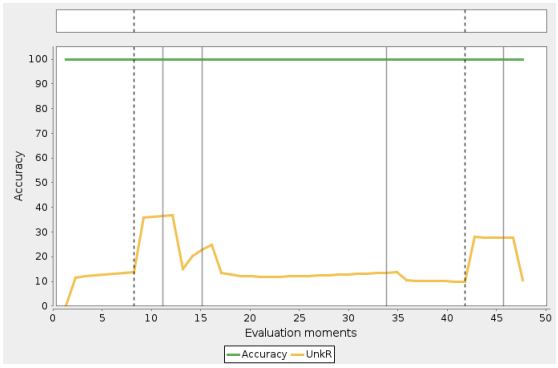
7.2.3 SynEDC

Figure 24 shows the results of the three algorithms for the SynEDC dataset. Figure 24a presents the results obtained by EFuzzCND algorithm, showing that the algorithm classified correctly all the instances it could classify. However, it is possible to notice that, for this dataset, which despite being synthetic, is complex and has concept evolution and concept drift, EFuzzCND did not perform well in the UnkR measure. Despite identifying the novelties, the UnkR increased with each novelty that arrived, reaching about 75% in the rate and then stabilizing at 68%. The results obtained indicate that this was caused by the dispersion of data, which also caused the high number of identified NPs, present in almost all evaluation moments.

Figure 24b presents the results obtained by PFuzzND algorithm, showing that the algorithm classified correctly all instances it could classify. Unlike EFuzzCND, the algorithm



(a) EFuzzCND **(b) PFuzzND**



(c) MINAS

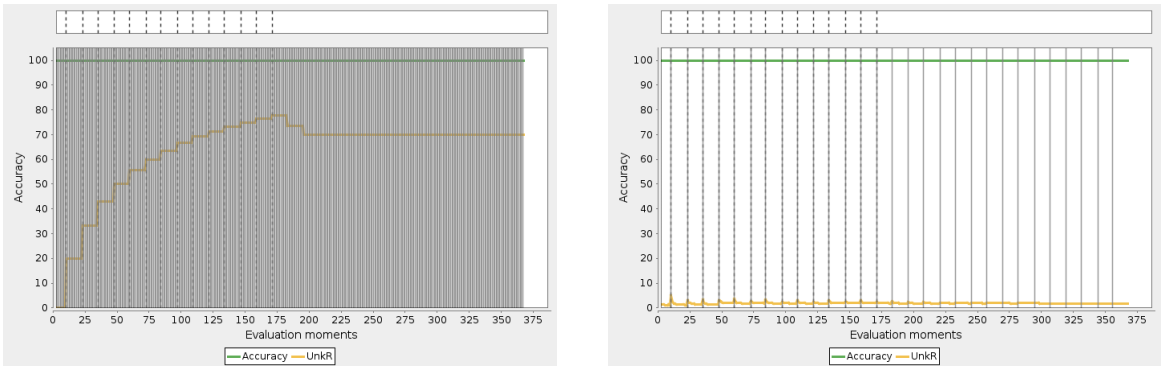
Figure 23 – Algorithm accuracy and UnkRs, considering extreme latency, for the RBF dataset.

had a good performance in the UnkR, filtering exactly the necessary instances to form new NPs that would classify instances belonging to the new class. With this behavior, the UnkR always remained below 5% and when novelties appeared, in just one moment of evaluation, the rates started to drop.

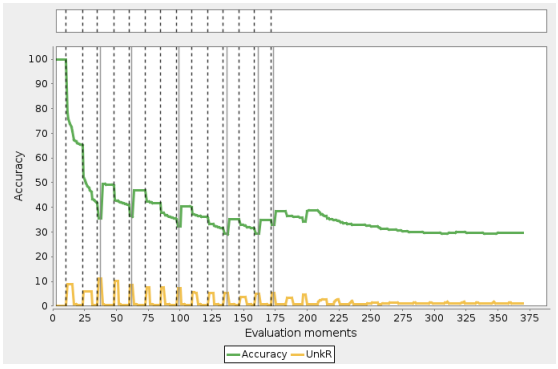
Figure 24c presents the results obtained by MINAS algorithm. It is notable that MINAs was not effective in the classification task, missing the great majority of the instances that tried to classify, and from the moment of evaluation 26 until the end, the algorithm maintained a accuracy rate lower than 50%. The UnkR obtained by MINAS was low, always remaining below 10%, however, when a novelty arrived, in some cases, the algorithm took a long time to identify it and after identification, in all cases, the algorithm took a few moments of evaluation to start lowering down this metric.

7.2.4 KDD99

Figure 25 shows the results of the three algorithms for the KDD99 dataset. Figure 25a presents the results obtained by EFuzzCND algorithm. It is notable that the accuracy was always kept at 100% and the UnkR between 40 and 50% during most of the evaluation moments. Furthermore, EFuzzCND identified NPs at different evaluation moments, which is an expected



(a) EFuzzCND **(b) PFuzzND**



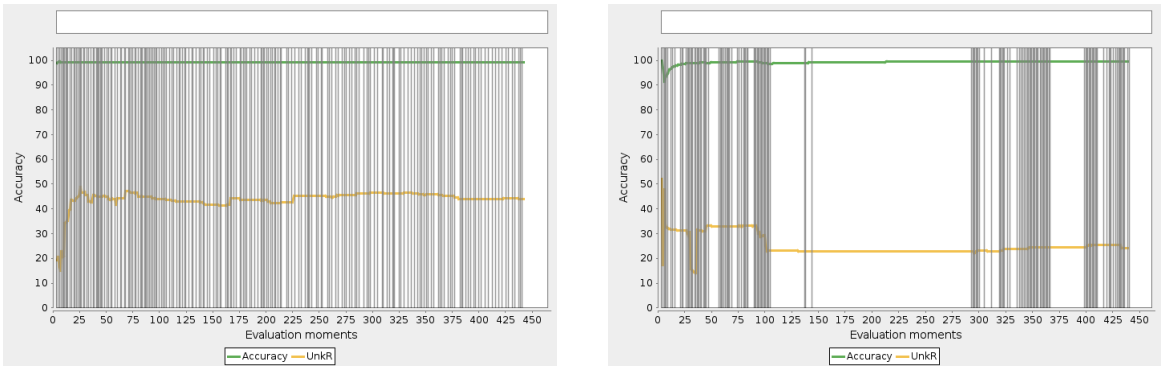
(c) MINAS

Figure 24 – Algorithm accuracy and UnkRs, considering extreme latency, for the SynEDC dataset.

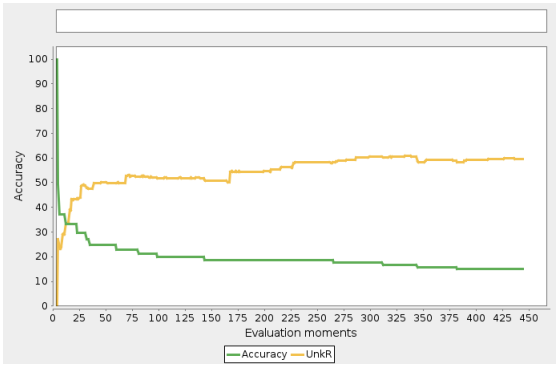
behavior, since it is the way the algorithm adapts to concept drift and novelties that arise.

Figure 25b presents the results obtained by PFuzzND algorithm. The algorithm was not so efficient in the classification, having some changes in the accuracy, reaching 92%. The measure UnkR, started with a high value, reaching close to 55%, however, it soon stabilized and was close to 23%. Furthermore, PFuzzND did not identify NPs at all times, between the 144 and 290 evaluation moments, no NP was identified. This shows that updating the decision model incrementally helps to adapt the decision model to the concept drift, reducing the number of NPs identified by the algorithm.

Figure 25c presents the results obtained by MINAS algorithm. Which did not perform well, with accuracy below 40% during most of the evaluation moments. Furthermore, the UnkR measure was above 50% during the vast majority of evaluation moments, which is not good for the algorithm. This may have happened because of the initialization parameters used in MINAS algorithm. The authors propose several ways to define the threshold for the identification of strangers and, probably, the selected threshold was not ideal for this dataset.



(a) EFuzzCND **(b) PFuzzND**



(c) MINAS

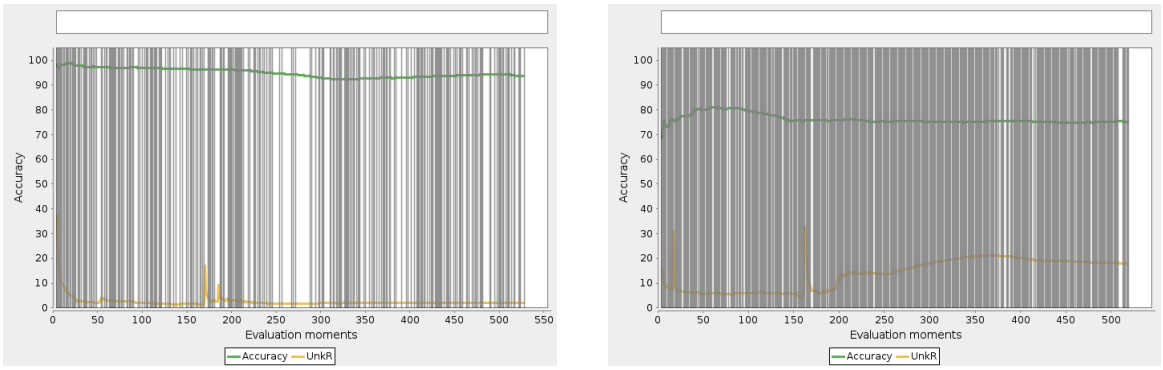
Figure 25 – Algorithm accuracy and UnkRs, considering extreme latency, for the KDD99 dataset.

7.2.5 CoverType

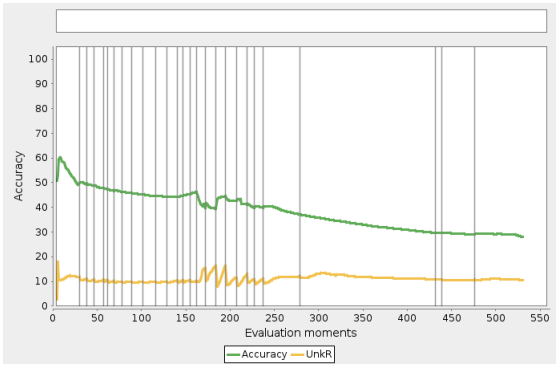
Figure 26 presents the results obtained by the three algorithms for the CoverType dataset. Figure 26a presents the results obtained by EFuzzCND algorithm. Which had a good performance, keeping the accuracy between 90% and 100%. In addition, UnkR started with a high value, around 20%, however, it decreased and remained below 5%, with the exception of some moments, where the values raised, between the evaluation moments 150 and 200.

Figure 26b presents the results obtained by PFuzzND algorithm. Which also maintained a good performance, although inferior to EFuzzCND. Its accuracy reached 80%, however it was close to 70% during most of the evaluation moments. The algorithm also showed a disadvantage in the UnkR measure, having high values between the evaluation moments 150 and 200 and remaining between 15% and 20% after that. In addition, due to the increase in UnkR , the number of identified PNs has increased considerably.

Figure 26c presents the results obtained by the MINAS algorithm. The performance of this algorithm was much lower when compared to the others. The accuracy started above 50% and reached 60% in the first 10 moments of evaluation, however, after that it began to decrease, reaching a level of 30% from the evaluation moment 400. The UnkR measure did not have such a high value, as in most evaluation moments it was between 10% and 20%. The algorithm



(a) EFuzzCND **(b) PFuzzND**



(c) MINAS

Figure 26 – Algorithm accuracy and UnkRs, considering extreme latency, for the CoverType dataset.

identified few NPs , only in a few isolated only a few isolated points along the DS. These factors may have influenced the identification and classification of novelties, which reduces considerably the performance of the algorithm.

Unbalance and class overlap make this dataset complex and significantly influenced the results obtained by the three algorithms. The overlapping of classes directly interferes in the typicality calculation that is used in EFuzzCND and also in PFuzzND. Overlapping classes generate similar values of typicality, which can cause errors in the classification of instances by algorithms that adopt this strategy.

7.2.6 General analysis of the results obtained

The results obtained under intermediate latency showed that both algorithms have good performances in classification and ND in this scenario. However, some advantages and disadvantages have been identified and they will be discussed in this sub-section.

EFuzzCND proved to be effective in classifying and ND, especially in simpler datasets, where overlapping classes do not occur so often, which is the case of datasets: MOA3, RBF and SynEDC. The use of fuzzy clustering brought greater flexibility in the decision boundaries of SPFMiCs, allowing an SPFMiC to classify possible outliers. However, ECSMiner, which uses

hard clustering, was unable to classify these outliers. This influenced directly the UnkR measure and was essential for the SPFMiCs, identified through the ND method, to classify correctly the instances.

These SPFMiCs have few instances, however, the flexibility adopted made it possible to classify new instances of this new identified concept. Because of this, EFuzzCND was able to identify the NPs that represent the new classes and to classify the instances correctly, at the same evaluation moment in which the novelties appeared or in the next evaluation moment, regardless of the latency value, unlike ECSMiner . This caused a significant reduction in the UnkR measure compared to the ECSMiner algorithm.

Analyzing the more complex datasets, which have many features and there are several class overlaps, which is the case of KDD99 and CoverType, it was possible to identify a decrease in the performance of EFuzzCND. In the case of KDD99, the UnkR measure had very high values, a fact that happened in both algorithms, due to the high number of outliers and scattered instances in this dataset.

In CoverType it is possible to notice a drop in the performance of EFuzzCND. This happened due to the strategy used in the classification, which applies the calculation of the typicality of the instances for the SPFMiCs in which they are within the decision boundaries. In the case of overlapping classes, the typicality calculation ends up resulting in very close values, which increases the chance of algorithm errors and that is what happened for this dataset.

The strategy used in the EFuzzCND algorithm to update the decision models ended up directly influencing the UnkR measure. The SPFMiCs contained by the decision models are static, that is, they do not update over time. In this way, the updating of decision models for the evolution of the concept is done through the ND process, which brings an increase in UnkR and in the number of identified NPs. This behavior becomes clear when comparing the different latency values, since the higher the latency value, the greater the UnkR and the number of identified NPs.

Even though the algorithm was originally developed for intermediate latency, executions under extreme latency showed promising results. The results showed that the latency value highly reflects on the number of identified NPs and on the UnkR measure. When compared to other algorithms that were originally designed for extreme latency. In most cases, the EFuzzCND had results comparable to the PFuzzND algorithm, which also uses fuzzy clustering and obtained superior results to the MINAS algorithm, which uses hard clustering.

Under extreme latency, EFuzzCND proved to be very sensitive to concept evolution and this behavior was reflected for the SynEDC dataset (Figure 24a). In this case, because they are static SPFMiCs, the algorithm showed difficulty in following the evolution of the classes, having to execute the ND to identify the extensions that appeared over time, increasing significantly the UnkR.

Overall, the experiments showed that EFuzzCND has promising results, considering different latency values. When compared to approaches that use crisp clustering, it was possible to notice that the flexibility given to EFuzzCND, due to the use of fuzzy clustering, caused the number of instances classified as unknown to decrease considerably, since instances that represented outliers for the SPFMiCs were classified.

When compared to another approach that uses fuzzy clustering, the EFuzzCND showed comparable results, always maintaining the accuracy equivalent to or higher than the comparison method and having a slight increase in the UnkR measure (with the exception of datasets that present very rapid and frequent concept evolution). PFuzzND has initialization parameters used to classify instances as unknown or not, which were empirically selected to generate the results presented. However, our approach makes this identification automatically, depending only on the fuzzy ratio multiplication factor, which brings a huge gain in relation to the comparison algorithm.

Chapter 8
CONCLUSIONS

DS is an area of machine learning that has been gaining notoriety in recent years, being the subject of research in many works. The challenges imposed on algorithms that deal with DS make new approaches emerge, seeking to bring gains to this scenario. One of the novelties that emerged in recent years was the application of fuzzy set theory, seeking to provide greater flexibility and adaptability of learned knowledge.

In this context, several approaches available in the literature do not explore the gains from using fuzzy set theory. Furthermore, we did not find any work that has been applied under intermediate latency or that has delved into different types of latency.

8.0.1 Contributions

This work presents a new approach for multi-class ND in DS, applicable under intermediate and extreme latency, which introduces greater flexibility in the learning process through the use of fuzzy set theory, based on the *Offline-Online* framework.

The main contributions of the work are described below:

- **Development of a fuzzy approach based on the ECSMiner method:** We present the EFuzzCND method, which was developed for intermediate latency, based on ECSMiner, which presented an evolution in the classification of outliers, significantly reducing the unknown rate, making the results equivalent and even superior to the original method.
- **Execution of the ND method as a multi-class task:** Unlike ECSMiner, which classifies the instances not classified by the decision model only as a novelty, the approach proposed in this work considers the ND as a multi-class task, that is, different classes of novelty may emerge over time. For this, different NPs are created to represent the different classes that may arise along the DS. This way, the algorithm must distinguish the instances among different NPs, which, as far as we could see, it is the first multi-class approach evaluated under intermediate latency.

- **Presentation of a study under different latency values in DS:** We present an analysis of the behavior of the approaches (EFuzzCND and ECSMiner) under different intermediate latency values, as well as an analysis of the proposed method also in extreme latency. In our bibliographic review, we did not find any work that explores analyzes under different latency values. Our analysis helps to understand the behavior of algorithms and the impact of latency value on their performance.

- **Reduction in the quantity and complexity of initialization parameters:** PFuzzND was used as a basis for applying the fuzzy set theory in the proposed method. PFuzzND uses several parameters, which must be defined empirically, to classify the instances arriving from the DS. We modified the classification procedure used in PFuzzND using fuzzy dispersion measure (XIONG et al., 2012) to define the boudaries of SPFMiCs, which allows the elimination of these parameters and makes the use of the algorithm simpler.

The experiments showed that the results obtained by our proposal are comparable or superior to the method used as a basis for comparison under intermediate latency. Furthermore, even though it was not originally developed for extreme latency, our proposal obtained promising results in the analyzes and comparisons made in this scenario.

The results obtained show that the proposal brings contributions to future researches and development of new techniques that apply the fuzzy set theory to ND in DS.

In this way, the proposed fuzzy approach, based on the ECSMiner method, contributes to the advancement of data stream learning research communities, especially those focused on providing more flexible learning, through the use of fuzzy techniques, or those focused on studying different types of latency.

8.0.2 Published and submitted papers

This work led to the submission of four articles, three of them accepted and published and one being under review. Are they:

- CRISTIANI, André Luis; DA SILVA, Tiago Pinho; DE ARRUDA CAMARGO, Heloisa. A Fuzzy Approach for Classification and Novelty Detection in Data Streams Under Intermediate Latency. In: Brazilian Conference on Intelligent Systems. Springer, Cham, 2020. p. 171-186.

- CRISTIANI, André L. et al. A Fuzzy Intrusion Detection System for Identifying Cyber-Attacks on IoT Networks. In: 2020 IEEE Latin-American Conference on Communications (LATINCOM). IEEE, 2020. p. 1-6.

- CRISTIANI, André Luis; DE ARRUDA CAMARGO, Heloisa. A Fuzzy Multi-class Novelty Detector for Data Streams Under Intermediate Latency. In: 2021 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE). IEEE, 2021. p. 1-6.
- CRISTIANI, André Luis; LIEIRA, Douglas Dias; Meneguette, Rodolfo Ipolito; DE ARRUDA CAMARGO, Heloisa. A Fuzzy Intrusion Detection System for Identifying Cyberattacks in Non-stationary IoT networks. (Under review)

8.0.3 Limitations and future work

The results obtained demonstrate that the EFuzzCND has several advantages when compared to non-fuzzy methods, as well as excellent results when compared to another fuzzy method, eliminating the use of initialization parameters, used in the classification of instances, that require a lot of time to ajust.

However, some points for improvement were identified and can be used as guidelines for future research. These points are listed below.

- **Algorithm parameterization:** Even with several improvements in this regard, when compared to its base fuzzy algorithm, the initialization parameters remain as one of the main limitations, as well as for the algorithms in the literature that perform ND in DS. Many parameters must be configured, such as: number of clusters, thresholds for cluster validation and PN identification, size of instance storage windows and values used in the typicality calculation. Despite the simplifications presented, selecting these values is a difficult task, which directly influences the performance of the classifier.
- **Model based on hyperspheres:** Even using techniques from fuzzy set theory, which make learning more flexible, the decision model used, built in hyperspherical formats, showed lower performance in datasets that do not have these characteristics, especially in situations where classes overlap. Future research should investigate ways to train and perform maintenance on classifiers, especially considering class overlapping.
- **Updating decision models incrementally:** Updating decision models statically made the algorithm depend on the ND method to identify class extensions, which causes an increase in UnkR. Updating the decision model incrementally makes it adjust faster to the concept drift. This can make the maintenance of decision models simpler and more effective, especially in extreme latency scenarios.
- **Treatment of concept drift:** The technique used to identify concept drift is the same to identify novelties, the difference is to compare the SPFMiCs found with the known concepts of the decision model. If the similarity is high, it represents an extension, that is, a concept drift. The technique used, despite being more costly to the algorithm, presented

interesting results, with the exception of abrupt changes. In this case, the algorithm does not have similar SPFMiCs in the decision model to assimilate these changes and ends up misclassifying as novelty.

REFERENCES

ABDALLAH, Z. S.; GABER, M. M.; SRINIVASAN, B.; KRISHNASWAMY, S. Anynovel: detection of novel concepts in evolving data streams. *Evolving Systems*, v. 7, n. 2, p. 73–93, Jun 2016. ISSN 1868-6486. Disponível em: <https://doi.org/10.1007/s12530-016-9147-7>. Citado 4 vezes nas páginas 36, 42, 48 e 49.

AGGARWAL, C. C. An introduction to outlier analysis. In: ____. *Outlier Analysis*. Cham: Springer International Publishing, 2017. p. 1–34. ISBN 978-3-319-47578-3. Disponível em: <https://doi.org/10.1007/978-3-319-47578-3_1>. Citado na página 32.

AGGARWAL, C. C.; HAN, J.; WANG, J.; YU, P. S. A framework for projected clustering of high dimensional data streams. In: *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*. [S.l.]: VLDB Endowment, 2004. (VLDB '04), p. 852–863. ISBN 0120884690. Citado na página 28.

AGGARWAL, C. C.; PHILIP, S. Y. A framework for clustering uncertain data streams. In: IEEE. *2008 IEEE 24th International Conference on Data Engineering*. [S.l.], 2008. p. 150–159. Citado na página 28.

AGGARWAL, C. C.; YU, P. S.; HAN, J.; WANG, J. - a framework for clustering evolving data streams. In: FREYTAG, J.-C.; LOCKEMANN, P.; ABITEBOUL, S.; CAREY, M.; SELINGER, P.; HEUER, A. (Ed.). *Proceedings 2003 VLDB Conference*. San Francisco: Morgan Kaufmann, 2003. p. 81 – 92. ISBN 978-0-12-722442-8. Disponível em: <http://www.sciencedirect.com/science/article/pii/B9780127224428500161>. Citado 2 vezes nas páginas 29 e 30.

AGRAWAL, R.; GEHRKE, J.; GUNOPULOS, D.; RAGHAVAN, P. Automatic subspace clustering of high dimensional data for data mining applications. In: *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: Association for Computing Machinery, 1998. (SIGMOD '98), p. 94–105. ISBN 0897919955. Disponível em: <https://doi.org/10.1145/276304.276314>. Citado na página 23.

AL-KHATEEB, T.; MASUD, M. M.; KHAN, L.; AGGARWAL, C.; HAN, J.; THURAISINGHAM, B. Stream classification with recurring and novel class detection using class-based ensemble. In: IEEE. *2012 IEEE 12th International Conference on Data Mining*. [S.l.], 2012. p. 31–40. Citado 5 vezes nas páginas 17, 36, 44, 48 e 49.

Angelov, P. P.; Zhou, X. Evolving fuzzy-rule-based classifiers from data streams. *IEEE Transactions on Fuzzy Systems*, v. 16, n. 6, p. 1462–1475, Dec 2008. ISSN 1941-0034. Citado 3 vezes nas páginas 28, 48 e 49.

ANKERST, M.; BREUNIG, M. M.; KRIEGEL, H.-P.; SANDER, J. Optics: Ordering points to identify the clustering structure. *SIGMOD Rec.*, Association for Computing Machinery, New York, NY, USA, v. 28, n. 2, p. 49–60, jun. 1999. ISSN 0163-5808. Disponível em: <https://doi.org/10.1145/304181.304187>. Citado na página 23.

ARYA, M.; G, H. S. *Smart Science*, Taylor Francis, v. 8, n. 2, p. 71–83, 2020. Disponível em: <https://doi.org/10.1080/23080477.2020.1783491>. Citado na página 28.

BABCOCK, B.; BABU, S.; DATAR, M.; MOTWANI, R.; WIDOM, J. Models and issues in data stream systems. In: *Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. New York, NY, USA: Association for Computing Machinery, 2002. (PODS '02), p. 1–16. ISBN 1581135076. Disponível em: <https://doi.org/10.1145/543613.543615>. Citado 2 vezes nas páginas 16 e 27.

BARBARá, D. Requirements for clustering data streams. *SIGKDD Explor. Newsl.*, Association for Computing Machinery, New York, NY, USA, v. 3, n. 2, p. 23–27, jan. 2002. ISSN 1931-0145. Disponível em: <https://doi.org/10.1145/507515.507519>. Citado na página 29.

BEZDEK, J. C. Objective function clustering. In: ____. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Boston, MA: Springer US, 1981. p. 43–93. ISBN 978-1-4757-0450-1. Disponível em: <https://doi.org/10.1007/978-1-4757-0450-1_3>. Citado 2 vezes nas páginas 23 e 24.

BEZDEK, J. C.; EHRLICH, R.; FULL, W. et al. The fuzzy c-means clustering algorithm computers & geosciences. *Science Direct*, v. 10, n. 2-3, p. 13, 1984. Citado 2 vezes nas páginas 23 e 46.

BISHOP, C. M. *Neural Networks for Pattern Recognition*. USA: Oxford University Press, Inc., 1995. ISBN 0198538642. Citado na página 22.

CAI, X.-Q.; ZHAO, P.; TING, K.-M.; MU, X.; JIANG, Y. Nearest neighbor ensembles: An effective method for difficult problems in streaming classification with emerging new classes. In: *2019 IEEE International Conference on Data Mining (ICDM)*. [S.l.: s.n.], 2019. p. 970–975. Citado 3 vezes nas páginas 43, 48 e 49.

CAMPELLO, R. J.; HRUSCHKA, E. R. A fuzzy extension of the silhouette width criterion for cluster analysis. *Fuzzy Sets and Systems*, Elsevier, v. 157, n. 21, p. 2858–2875, 2006. Citado 2 vezes nas páginas 47 e 56.

CHANDOLA, V.; BANERJEE, A.; KUMAR, V. Anomaly detection: A survey. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 41, n. 3, jul. 2009. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/1541880.1541882>. Citado na página 32.

Costa Júnior, J. D.; Faria, E. R.; Silva, J. A.; Gama, J.; Cerri, R. Novelty detection for multi-label stream classification. In: *2019 8th Brazilian Conference on Intelligent Systems (BRACIS)*. [S.l.: s.n.], 2019. p. 144–149. ISSN 2643-6256. Citado 4 vezes nas páginas 17, 43, 48 e 49.

COVER, T.; HART, P. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, v. 13, n. 1, p. 21–27, January 1967. ISSN 1557-9654. Citado na página 22.

CRISTIANI, A. L.; LIEIRA, D. D.; MENEGUETTE, R. I.; CAMARGO, H. A. A fuzzy intrusion detection system for identifying cyber-attacks on iot networks. In: *2020 IEEE Latin-American Conference on Communications (LATINCOM)*. [S.l.: s.n.], 2020. p. 1–6. Citado na página 28.

CRISTIANINI, N.; SHAWE-TAYLOR, J. et al. *An introduction to support vector machines and other kernel-based learning methods*. [S.l.]: Cambridge university press, 2000. Citado na página 22.

CUP, K. http://kdd. ics. uci. edu/databases/kddcup99/kddcup99. html. *The UCI KDD Archive*, 1999. Citado na página 59.

DIN, S. U.; SHAO, J. Exploiting evolving micro-clusters for data stream classification with emerging class detection. *Information Sciences*, v. 507, p. 404–420, 2020. ISSN 0020-0255. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0020025519307960>. Citado 3 vezes nas páginas 45, 48 e 49.

DOMINGOS, P.; HULTEN, G. A general method for scaling up machine learning algorithms and its application to clustering. In: *ICML*. [S.l.: s.n.], 2001. v. 1, p. 106–113. Citado na página 27.

DUDA, R. p. hart. *Pattern Classification and Scene Analysis*, p. 289–292, 1973. Citado na página 22.

Elwell, R.; Polikar, R. Incremental learning of concept drift in nonstationary environments. *IEEE Transactions on Neural Networks*, v. 22, n. 10, p. 1517–1531, Oct 2011. ISSN 1941-0093. Citado na página 33.

ESTER, M.; KRIEGEL, H.-P.; SANDER, J.; XU, X. A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise. In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. [S.l.]: AAAI Press, 1996. (KDD'96), p. 226–231. Citado na página 23.

FACELI, K.; LORENA, A. C.; GAMA, J.; CARVALHO, A. C. P. d. L. F. d. *Inteligência artificial: uma abordagem de aprendizado de máquina*. [S.l.]: LTC, 2011. Citado 2 vezes nas páginas 21 e 22.

FARIA, E. R.; GONÇALVES, I. J. C. R.; CARVALHO, A. C. P. L. F. de; GAMA, J. Novelty detection in data streams. *Artificial Intelligence Review*, v. 45, n. 2, p. 235–269, Feb 2016. ISSN 1573-7462. Disponível em: <https://doi.org/10.1007/s10462-015-9444-8>. Citado 10 vezes nas páginas 17, 27, 28, 32, 34, 35, 49, 52, 63 e 64.

FARIA, E. R. de; CARVALHO, A. C. Ponce de L. F.; GAMA, J. Minas: multiclass learning algorithm for novelty detection in data streams. *Data Mining and Knowledge Discovery*, v. 30, n. 3, p. 640–680, May 2016. ISSN 1573-756X. Disponível em: <https://doi.org/10.1007/s10618-015-0433-y>. Citado 8 vezes nas páginas 32, 36, 41, 42, 46, 48, 59 e 62.

FARID, D. M.; RAHMAN, C. M. Novel class detection in concept-drifting data stream mining employing decision tree. In: IEEE. *2012 7th International Conference on Electrical and Computer Engineering*. [S.l.], 2012. p. 630–633. Citado 4 vezes nas páginas 36, 40, 48 e 49.

FARID, D. M.; ZHANG, L.; HOSSAIN, A.; RAHMAN, C. M.; STRACHAN, R.; SEXTON, G.; DAHAL, K. An adaptive ensemble classifier for mining concept drifting data streams. *Expert Systems with Applications*, v. 40, n. 15, p. 5895 – 5906, 2013. ISSN 0957-4174. Disponível em: <http://www.sciencedirect.com/science/article/pii/S095741741300287X>. Citado 4 vezes nas páginas 36, 40, 48 e 49.

GAMA, J. *Knowledge discovery from data streams*. [S.l.]: Chapman and Hall/CRC, 2010. Citado 5 vezes nas páginas 16, 17, 27, 29 e 50.

GAMA, J. a.; ROCHA, R.; MEDAS, P. Accurate decision trees for mining high-speed data streams. In: *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: Association for Computing Machinery, 2003. (KDD '03), p. 523–528. ISBN 1581137370. Disponível em: <https://doi.org/10.1145/956750.956813>. Citado na página 28.

GAMA, J. a.; ZLIOBAITUNDEFINED, I.; BIFET, A.; PECHENIZKIY, M.; BOUCHACHIA, A. A survey on concept drift adaptation. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 46, n. 4, mar. 2014. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/2523813>. Citado 3 vezes nas páginas 10, 33 e 34.

GARCIA, K. D.; POEL, M.; KOK, J. N.; CARVALHO, A. C. de. Online clustering for novelty detection and concept drift in data streams. In: SPRINGER. *EPIA Conference on Artificial Intelligence*. [S.l.], 2019. p. 448–459. Citado 5 vezes nas páginas 32, 36, 42, 48 e 49.

GIANNELLA, C.; HAN, J.; PEI, J.; YAN, X.; YU, P. S. Mining frequent patterns in data streams at multiple time granularities. *Next generation data mining*, AAAI/MIT, v. 212, p. 191–212, 2003. Citado na página 28.

GOLDBERG, D. E. Genetic algorithms in search, optimization, and machine learning, addison wesley, reading, ma. *SUMMARY THE APPLICATIONS OF GA-GENETIC ALGORITHM FOR DEALING WITH SOME OPTIMAL CALCULATIONS IN ECONOMICS*, 1989. Citado na página 22.

GRUHL, C.; TOMFORDE, S. Ohodin – online anomaly detection for data streams. In: *2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*. [S.l.: s.n.], 2021. p. 193–197. Citado 3 vezes nas páginas 39, 48 e 49.

HAN, J.; PEI, J.; YIN, Y. Mining frequent patterns without candidate generation. *SIGMOD Rec.*, Association for Computing Machinery, New York, NY, USA, v. 29, n. 2, p. 1–12, maio 2000. ISSN 0163-5808. Disponível em: <https://doi.org/10.1145/335191.335372>. Citado na página 28.

HARTUV, E.; SHAMIR, R. A clustering algorithm based on graph connectivity. *Information Processing Letters*, v. 76, p. 175–181, 12 2000. Citado na página 23.

JIN, R.; AGRAWAL, G. Efficient decision tree construction on streaming data. In: *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: Association for Computing Machinery, 2003. (KDD '03), p. 571–576. ISBN 1581137370. Disponível em: <https://doi.org/10.1145/956750.956821>. Citado na página 28.

KRAWCZYK, B.; WOŹNIAK, M. Incremental learning and forgetting in one-class classifiers for data streams. In: BURDUK, R.; JACKOWSKI, K.; KURZYNSKI, M.; WOZNIAK, M.; ZOLNIEREK, A. (Ed.). *Proceedings of the 8th International Conference on Computer Recognition Systems CORES 2013*. Heidelberg: Springer International Publishing, 2013. p. 319–328. ISBN 978-3-319-00969-8. Citado na página 36.

KREMPL, G.; ZLIOBAITE, I.; BRZEZIUNDEFINEDSKI, D.; HüLLERMEIER, E.; LAST, M.; LEMAIRE, V.; NOACK, T.; SHAKER, A.; SIEVI, S.; SPILIOPOULOU, M.; AL. et. Open challenges for data stream mining research. *SIGKDD Explor. Newsl.*, Association for Computing Machinery, New York, NY, USA, v. 16, n. 1, p. 1–10, set. 2014. ISSN 1931-0145. Disponível em: <https://doi.org/10.1145/2674026.2674028>. Citado na página 37.

Krishnapuram, R.; Keller, J. M. A possibilistic approach to clustering. *IEEE Transactions on Fuzzy Systems*, v. 1, n. 2, p. 98–110, May 1993. ISSN 1941-0034. Citado na página 25.

LANGLEY, P. Induction of recursive bayesian classifiers. In: SPRINGER. *European Conference on Machine Learning*. [S.l.], 1993. p. 153–164. Citado na página 22.

LOPES, P. de A.; CAMARGO, H. de A. Fuzzstream: Fuzzy data stream clustering based on the online-offline framework. In: *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. [S.l.: s.n.], 2017. p. 1–6. Citado na página 17.

MACQUEEN, J. et al. Some methods for classification and analysis of multivariate observations. In: OAKLAND, CA, USA. *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. [S.l.], 1967. v. 1, n. 14, p. 281–297. Citado na página 23.

MARIN, L.; ISERN, D.; MORENO, A.; VALLS, A. On-line dynamic adaptation of fuzzy preferences. *Information Sciences*, Elsevier, v. 220, p. 5–21, 2013. Citado na página 28.

MASUD, M.; GAO, J.; KHAN, L.; HAN, J.; THURAISINGHAM, B. M. 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, 2011. Citado 13 vezes nas páginas 16, 17, 18, 28, 32, 36, 43, 44, 48, 49, 51, 59 e 61.

Minku, L. L.; White, A. P.; Yao, X. The impact of diversity on online ensemble learning in the presence of concept drift. *IEEE Transactions on Knowledge and Data Engineering*, v. 22, n. 5, p. 730–742, May 2010. ISSN 2326-3865. Citado na página 33.

MITCHELL, R.; MICHALSKI, J.; CARBONELL, T. *An artificial intelligence approach*. [S.l.]: Springer, 2013. Citado na página 21.

MITCHELL, T. M. *Machine learning. WCB*. [S.l.]: McGraw-Hill Boston, MA:, 1997. Citado na página 21.

NAHAR, V.; AL-MASKARI, S.; LI, X.; PANG, C. Semi-supervised learning for cyberbullying detection in social networks. In: WANG, H.; SHARAF, M. A. (Ed.). *Databases Theory and Applications*. Cham: Springer International Publishing, 2014. p. 160–171. ISBN 978-3-319-08608-8. Citado na página 28.

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. Citado na página 28.

OLIVEIRA, L. S. *Classificação de fluxos de dados não estacionários com algoritmos incrementais baseados no modelo de misturas gaussianas*. Tese (Doutorado) — Universidade de São Paulo, 2016. Citado na página 28.

Pal, N. R.; Pal, K.; Bezdek, J. C. A mixed c-means clustering model. In: *Proceedings of 6th International Fuzzy Systems Conference*. [S.l.: s.n.], 1997. v. 1, p. 11–21 vol.1. ISSN null. Citado na página 25.

Pal, N. R.; Pal, K.; Keller, J. M.; Bezdek, J. C. A possibilistic fuzzy c-means clustering algorithm. *IEEE Transactions on Fuzzy Systems*, v. 13, n. 4, p. 517–530, Aug 2005. ISSN 1941-0034. Citado na página 25.

PAL, N. R.; PAL, K.; KELLER, J. M.; BEZDEK, J. C. A possibilistic fuzzy c-means clustering algorithm. *IEEE transactions on fuzzy systems*, IEEE, v. 13, n. 4, p. 517–530, 2005. Citado na página 46.

PARK, C. H.; SHIM, H. Detection of an emerging new class using statistical hypothesis testing and density estimation. *International journal of pattern recognition and artificial intelligence*, World Scientific, v. 24, n. 01, p. 1–14, 2010. Citado na página 33.

PIMENTEL, M. A.; CLIFTON, D. A.; CLIFTON, L.; TARASSENKO, L. A review of novelty detection. *Signal Processing*, Elsevier, v. 99, p. 215–249, 2014. Citado na página 32.

QUINLAN, J. R. Induction of decision trees. *Machine Learning*, v. 1, n. 1, p. 81–106, Mar 1986. ISSN 1573-0565. Disponível em: <https://doi.org/10.1007/BF00116251>. Citado na página 22.

RODRIGUES, P. P.; GAMA, J.; PEDROSO, J. P. Odac: Hierarchical clustering of time series data streams. In: SIAM. *Proceedings of the 2006 SIAM International Conference on Data Mining*. [S.l.], 2006. p. 499–503. Citado na página 28.

RUSIECKI, A. Robust neural network for novelty detection on data streams. In: RUTKOWSKI, L.; KORYTKOWSKI, M.; SCHERER, R.; TADEUSIEWICZ, R.; ZADEH, L. A.; ZURADA, J. M. (Ed.). *Artificial Intelligence and Soft Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 178–186. ISBN 978-3-642-29347-4. Citado 4 vezes nas páginas 36, 40, 48 e 49.

SILVA, T. P. da. *Abordagem Fuzzy para Detecção de Novidade em Fluxo Contínuo de Dados*. 100 p. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal de São Carlos, São Carlos, 2018. Citado na página 17.

SILVA, T. P. da; CAMARGO, H. de A. Possibilistic approach for novelty detection in data streams. In: *2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. [S.l.: s.n.], 2020. p. 1–8. Citado 11 vezes nas páginas 28, 30, 32, 36, 46, 48, 49, 51, 54, 59 e 62.

SILVA, T. P. da; SCHICK, L.; LOPES, P. de A.; CAMARGO, H. de A. 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. Citado 3 vezes nas páginas 46, 48 e 49.

SILVA, T. P. da; URBAN, G. A.; LOPES, P. de A.; CAMARGO, H. de A. A fuzzy variant for on-demand data stream classification. In: IEEE. *2017 Brazilian Conference on Intelligent Systems (BRACIS)*. [S.l.], 2017. p. 67–72. Citado na página 46.

SOUSA, E. P. M. de; TRAINA, A. J. M.; TRAINA, C. J.; FALOUTSOS, C. Evaluating the intrinsic dimension of evolving data streams. In: *Proceedings of the 2006 ACM Symposium on Applied Computing*. New York, NY, USA: Association for Computing Machinery, 2006. (SAC '06), p. 643–648. ISBN 1595931082. Disponível em: <https://doi.org/10.1145/1141277.1141426>. Citado na página 28.

Souza, V.; Pinho, T.; Batista, G. Evaluating stream classifiers with delayed labels information. In: *2018 7th Brazilian Conference on Intelligent Systems (BRACIS)*. [S.l.: s.n.], 2018. p. 408–413. ISSN null. Citado 2 vezes nas páginas 36 e 50.

SPINOSA, E. J.; CARVALHO, A. P. d. L. de; GAMA, J. et al. Novelty detection with application to data streams. *Intelligent Data Analysis*, IOS Press, v. 13, n. 3, p. 405–422, 2009. Citado 4 vezes nas páginas 36, 38, 48 e 49.

TAN, S. C.; TING, K. M.; LIU, T. F. Fast anomaly detection for streaming data. In: *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*. [S.l.: s.n.], 2011. v. 22, n. 1, p. 1511. Citado 3 vezes nas páginas 39, 48 e 49.

TIAN, Z.; RAMAKRISHNAN, R.; BIRCH, L. M. An efficient data clustering method for very large databases. In: *Proc of the ACM SIGMOD International Conference on Management of Data. Montre—al, Canada*. [S.l.: s.n.], 1996. p. 103–114. Citado 2 vezes nas páginas 29 e 30.

TROYANO, F. J. F.; RUIZ, J. S. A.; SANTOS, J. C. R. Incremental rule learning and border examples selection from numerical data streams. *Journal of Universal Computer Science, 11 (8), 1426-1439.*, Graz University of Technology, Institut für Informationssysteme und Computer . . . , 2005. Citado na página 28.

UCI. https://archive.ics.uci.edu/ml/datasets/covertype. *UCI Machine Learning Repository: Covertype Data Set.*, 1998. Citado na página 60.

VENDRAMIN, L.; CAMPELLO, R. J.; HRUSCHKA, E. R. Relative clustering validity criteria: A comparative overview. *Statistical analysis and data mining: the ASA data science journal*, Wiley Online Library, v. 3, n. 4, p. 209–235, 2010. Citado na página 41.

WANG, H.; FAN, W.; YU, P. S.; HAN, J. Mining concept-drifting data streams using ensemble classifiers. In: *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: Association for Computing Machinery, 2003. (KDD '03), p. 226–235. ISBN 1581137370. Disponível em: <https://doi.org/10.1145/956750.956778>. Citado na página 28.

WIDMER, G.; KUBAT, M. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, v. 23, n. 1, p. 69–101, Apr 1996. ISSN 1573-0565. Disponível em: <https://doi.org/10.1023/A:1018046501280>. Citado na página 33.

WU, X.; LI, P.; HU, X. Learning from concept drifting data streams with unlabeled data. *Neurocomputing*, Elsevier, v. 92, p. 145–155, 2012. Citado na página 28.

XIONG, X.; CHAN, K.; TAN, K. L. Similarity-driven cluster merging method for unsupervised fuzzy clustering. *arXiv preprint arXiv:1207.4155*, 2012. Citado 3 vezes nas páginas 47, 57 e 86.

ZHANG, T.; RAMAKRISHNAN, R.; LIVNY, M. Birch: An efficient data clustering method for very large databases. In: *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: Association for Computing Machinery, 1996. (SIGMOD '96), p. 103–114. ISBN 0897917944. Disponível em: <https://doi.org/10.1145/233269.233324>. Citado na página 28.

ZHENG, X.; LI, P.; HU, X.; YU, K. Semi-supervised classification on data streams with recurring concept drift and concept evolution. *Knowledge-Based Systems*, v. 215, p. 106749, 2021. ISSN 0950-7051. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0950705121000125>. Citado 3 vezes nas páginas 45, 48 e 49.

ZHU, X.; GOLDBERG, A. B. Introduction to semi-supervised learning (synthesis lectures on artificial intelligence and machine learning). *Morgan and Claypool Publishers*, v. 14, 2009. Citado na página 22.