Universidade Federal de São Carlos

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

Distributed d-FuzzStream: Agrupamento *Fuzzy* Não Supervisionado Distribuído em Fluxo de Dados

Leonardo Schick

Orientadora: Profa. Dra. Heloisa de Arruda Camargo

São Carlos, SP, Brasil

Fevereiro, 2019

Universidade Federal de São Carlos

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

Distributed d-FuzzStream: Agrupamento *Fuzzy* Não Supervisionado Distribuído em Fluxo de Dados

Leonardo Schick

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: Inteligência Artificial
Orientadora: Profa. Dra. Heloisa de Arruda Camargo

São Carlos, SP, Brasil

Fevereiro, 2019

UNIVERSIDADE FEDERAL DE SÃO CARLOS

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

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a Defesa de Dissertação de Mestrado do candidato Leonardo Schick, realizada em 26/02/2019:

Profa. Dra. Heloisa de Arruda Camargo
UFSCar

Prof. Dr. Helio Crestana Guardia
UFSCar

Prof. Dr. Daniel Furtado Leite
UFLA

Certifico que a defesa realizou-se com a participação à distância do(s) membro(s) Daniel Furtado Leite e, depois das arguições e deliberações realizadas, o(s) participante(s) à distância está(ao) de acordo com o conteúdo do parecer da banca examinadora redigido neste relatório de defesa.

Profa. Dra. Heloisa de Arruda Camargo

Dedicado este trabalho a quem mais amo,
minha família.

Agradecimentos

Agradecimentos à família CIG, e em especial a Heloísa que me acolheu nessa dura jornada que é se tornar Mestre. Agradeço aos colegas e amigos que tive a honra de conhecer durante esse tempo. Suzane, Thiago, Frank, Maykon, Marcela, Pedro, Gobo, Luquinha e Mato. Gratidão!

A um grande amigo, Diego Pedroso. Obrigado pelos aprendizados, pelos cafés e tardes de tédio entre as minhas diversas idas e vindas a São Carlos. Espero um dia ser essa lenda que você sempre diz que sou.

Ao Renan por poder compartilhar meus problemas, angústias, vitórias e derrotas. Sou grato por estar comigo e pelas palavras de conforto, conselhos, revisões e discussões durante essa trajetória.

Agradeço imensamente a Priscilla Lopes, que iluminou o meu caminho e me ajudou a contornar as dificuldades. Obrigado por se tornar minha parceira na ciência. Sem você, nada disso teria acontecido.

Por fim, agradecimentos a Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo apoio e concessão da bolsa de estudos.

Science is not only a disciple of reason,
but, also, one of romance and passion.
Stephen Hawking

Resumo

Na última década cresceu o interesse em gerenciar sequências ilimitadas de dados que são produzidas em altas taxas. Estas sequências, conhecidas como fluxo de dados (FD), possuem diversas aplicações como redes de sensores, análises meteorológicas, análises da bolsa de valores, monitoramento de tráfego de redes, entre outras. Como ocorre com outros domínios de dados, existe interesse na extração de conhecimento útil de FDs a partir de modelos e algoritmos de aprendizado. Entretanto, extrair potenciais informações relevantes nesse domínio exige técnicas que se adaptam às mudanças na distribuição dos dados sem necessitar armazenar todo conjunto de dados. Um dos grandes desafios no aprendizado em FD é a capacidade de se adaptar à velocidade de produção dos dados, evitando o acúmulo de exemplos sem processamento e garantindo respostas em tempo real. Se a velocidade de produção de novos exemplos for maior que a velocidade de processamento, os dados se acumulam continuamente, e começam a ser descartados, tornando o aprendizado cada vez mais defasado. Recentemente, com a popularização de ferramentas de computação distribuída, muitos algoritmos de aprendizado de máquina foram adaptados para serem executados de forma distribuída. Apesar disso, grande parte desses algoritmos são adaptações de algoritmos de aprendizado de máquina tradicionais para lidar com grandes volumes de dados. Assim, existe uma carência de algoritmos para aprendizado em FD que possam ser executados de forma distribuída, principalmente no âmbito de aprendizado não supervisionado. Este trabalho apresenta uma nova abordagem distribuída para agrupamento *fuzzy* em fluxo de dados utilizando o *Framework Online-Offline*. Esta abordagem inclui duas estratégias de processamento diferentes para cada uma das fases do *framework*. A estratégia da fase *online* sumariza os dados distribuidamente em estruturas de sumarização distintas, que são unificadas por uma fase intermediária. Já na fase *offline*, a estratégia de processamento distribuído agrupa o resultado gerado pela estrutura de sumarização unificada com diversos algoritmos e números de grupos diferentes. Foram conduzidos 135 experimentos em seis bases de dados com grande quantidade de exemplos, sendo quatro sintéticas e duas reais, de onde foram coletadas métricas internas, externas e métricas de desempenho. Os resultados mostraram que a abordagem distribuída gerou agrupamentos similares aos gerados sem a abordagem, processando em média 18.200 exemplos por segundo. Isso mostra que a proposta é relevante pois conseguiu gerar agrupamentos similares em apenas 6% do tempo de processamento da versão sequencial.

Palavras-chaves: fluxo de dados, agrupamento de dados, teoria de conjuntos *fuzzy*, *Framework Online-Offline*, sistemas distribuídos, computação distribuída

Abstract

In the last decade, the interest in managing unlimited sequences of data produced at high rates, known as data stream (DS), has grown. Applications of DSs include sensor networks, weather analysis, stock market analysis, network traffic monitoring, among others. As in other data domains, there is an interest in extracting useful knowledge from DS using automatic techniques. However, extracting potentially relevant information in this domain requires techniques that can overcome storage constraints and that can conduct a continuous learning process, adapting to changes in data distribution. One of the great challenges in DS learning is the ability to adapt to the data production rate, avoiding the accumulation of non-processed examples and ensuring real-time responses. If the production of new examples is faster than the processing speed, then the data accumulate continuously and are often discarded, making the learning of new concepts outdated. Recently, with the popularization of distributed computing tools, many machine learning algorithms have been adapted to be executed in a distributed manner. Despite this, most of these algorithms are adaptations of classical machine learning algorithms to handle large volumes of data. Therefore, there is a lack of algorithms for learning from DS that can be executed distributedly, mostly in the context of unsupervised learning. This work presents a new distributed approach to fuzzy clustering in data stream using the Framework Online-Offline. The approach includes two different processing strategies for each of the phases of the framework. The online phase's strategy summarizes data in distinct summary structures, which are unified by an intermediate phase. In the offline phase, the distributed processing strategy groups the result generated by the unified summary structure with several algorithms and different number of groups. A total of 135 experiments were conducted on 6 databases with a large number of examples, including four synthetic and two real data sets, from which internal, external and performance metrics were collected. The results showed that the distributed approach generated clustering results similar to those generated without the approach, processing on average 18,200 examples per second. This shows that the proposal is relevant because it was able to generate similar partitions using only 6% of the processing time compared to the non-distributed version.

Keywords: data stream, data clustering, fuzzy sets theory, Online-Offline Framework, distributed systems, distributed computing

Lista de Ilustrações

Figura 2.1 Padrões de desvio de conceito 25
Figura 2.2 Exemplos ilustrativos de modelos de janela temporal 26
Figura 2.3 Funcionamento do *Framework Online-Offline* 28

Figura 3.1 Fluxo de execução do *MapReduce* 36
Figura 3.2 Exemplo de processamento no padrão *Pipeline* 37
Figura 3.3 Exemplo de processamento no padrão *Farm* 37

Figura 4.1 Exemplo de FMiCs e seus contornos de dispersão 48
Figura 4.2 Estratégia de sumarização local e global 54
Figura 4.3 Estratégia de agrupamento paralelo 55

Figura 5.1 Correlação entre um código em Java e o *Streaming Dataflow* 58
Figura 5.2 Paralelismo de Subtarefas 59
Figura 5.3 Representação dos exemplos de cada conjunto em diferentes momentos 62

Figura 6.1 Estrutura de sumarização e agrupamento da observação 5 do BG_10K 73
Figura 6.2 Estrutura de sumarização e agrupamento da observação 4 do Bench1_11K 73
Figura 6.3 Estrutura de sumarização e agrupamento da observação 2 do RBF4_40K 74
Figura 6.4 Resultados comparativos de Silhueta *Fuzzy* 76

Figura 7.1 Estruturas de sumarização da observação 1 do BG_1M 80
Figura 7.2 Agrupamento final da observação 1 do BG_1M 81
Figura 7.3 Estruturas de sumarização da observação 10 do BG_1M 82
Figura 7.4 Agrupamento final da observação 10 do BG_1M 83
Figura 7.5 Estruturas de sumarização da observação 2 do RBF5_1M 84
Figura 7.6 Agrupamento final da observação 2 do RBF5_1M 85
Figura 7.7 Estruturas de sumarização da observação 3 do RBF5_1M 86
Figura 7.8 Agrupamento final da observação 3 do RBF5_1M 87
Figura 7.9 Estruturas de sumarização da observação 7 do RBF6_400K 88
Figura 7.10 Agrupamento final da observação 7 do RBF6_400K 89

Figura 7.11 Estruturas de sumarização da observação 8 do RBF6_400K 90
Figura 7.12 Agrupamento final da observação 8 do RBF6_400K 91
Figura 7.13 Valores de Silhueta em relação ao grau de paralelismo 93
Figura 7.13 Valores de Silhueta em relação ao grau de paralelismo (cont.) 94
Figura 7.14 Valores de Silhueta Fuzzy em relação ao grau de paralelismo 94
Figura 7.14 Valores de Silhueta Fuzzy em relação ao grau de paralelismo (cont.) . . 95
Figura 7.15 Valores de XieBeni em relação ao grau de paralelismo 95
Figura 7.15 Valores de XieBeni em relação ao grau de paralelismo (cont.) 96
Figura 7.16 Valores de Pureza em relação ao grau de paralelismo 97
Figura 7.17 Valores de índice de Rand em relação ao grau de paralelismo 98
Figura 7.18 *Speedup* da estratégia de sumarização local e global 100
Figura 7.19 Eficiência da estratégia de sumarização local e global 100
Figura 7.20 *Speedup* da estratégia de agrupamento distribuído 102
Figura 7.21 Eficiência da estratégia de agrupamento distribuído 102

Lista de Tabelas

Tabela 4.1 Estrutura de um FMiC no *d-FuzzStream* 49

Tabela 5.1 Parâmetros da estratégia de sumarização local e global 64
Tabela 5.2 Parâmetros da estratégia de agrupamento distribuído 64

Tabela 6.1 Parâmetros do *d-FuzzStream* e *FuzzStream* 72
Tabela 6.2 Resultado das métricas informativas para o BG_10K 75
Tabela 6.3 Resultado das métricas informativas para o Bench1_11K 75
Tabela 6.4 Resultado das métricas informativas para o RBF4_40K 75
Tabela 6.5 Comparativo entre os tempos de execução do *FuzzStream* e *d-FuzzStream* 77

Tabela 7.1 Média do tamanho das estruturas de sumarização das dez observações com diferentes graus de paralelismo 92
Tabela 7.2 Tempos de execução da estratégia de sumarização local e global considerando diferentes graus de paralelismo 99
Tabela 7.3 Tempos de execução da estratégia de agrupamento distribuído considerando diferentes graus de paralelismo 101

Lista de Abreviaturas e Siglas

AFD	Aprendizado em Fluxo de Dados
AM	Aprendizado de Máquina
AP	*Affinity Propagation*
CF	*Clustering Feature*
FCM	*Fuzzy C-Means*
FD	Fluxo de Dados
FMiC	*Fuzzy Micro-Cluster*
FOO	*Framework Online-Offline*
F-EAC	*Fast Evolutionary Algorithm for Clustering*
WKMeans	*Weighted K-Means*
MaC	*Macro-Cluster*
MiC	*Micro-Cluster*
OFCM	*Online Fuzzy C-Means*
SLFCM	*Scalable Literal Fuzzy C-Means*
SPFCM	*Single Pass Fuzzy C-Means*
SRSIO-FCM	*Scalable Random Sampling with Iterative Optimization Fuzzy C-Means*
WFCM	*Weighted Fuzzy C-Means*

Sumário

Introdução **15**

I Revisão Bibliográfica **20**

1 Aprendizado de Máquina Clássico **21**
1.1 Aprendizado Supervisionado 21
1.2 Aprendizado Não Supervisionado 22
1.3 Considerações Finais 22

2 Aprendizado em Fluxo de Dados **23**
2.1 Desvios de Conceito 24
2.2 Janelas Temporais 25
2.3 Classificação em FD 26
2.4 Agrupamento em FD 27
2.5 Estruturas de Sumarização de Exemplos 28
2.5.1 Vetor de Atributos 28
2.5.2 Arranjo de Protótipos 29
2.5.3 Árvore de Coreset 29
2.5.4 Grade de Dados 30
2.6 Técnicas de Agrupamento em FD 30
2.7 Técnicas de Agrupamento *Fuzzy* em FD 32
2.8 Considerações Finais 34

3 Computação Distribuída **35**
3.1 Modelo *MapReduce* 35
3.2 Processamento Distribuído em FD 37
3.3 Ferramentas de Processamento em FD 38
3.3.1 Spark Streaming 38
3.3.2 Storm 38
3.3.3 Flink 39
3.3.4 SAMOA 39
3.4 Técnicas de Agrupamento Distribuído 39

3.5 Técnicas de Agrupamento *Fuzzy* Distribuído 41
3.6 Considerações Finais 42

II Proposta e Experimentos 43

4 Abordagem Distribuída para Agrupamento em Fluxo Contínuo de Dados . 44
4.1 Abordagem Distribuída para Agrupamento em Fluxo de Dados 45
4.2 *d-FuzzStream* 46
4.2.1 Dispersão *fuzzy* 47
4.2.2 Mescla baseada em similaridade *fuzzy* 47
4.2.3 Fase *Online*: Sumarização dos dados 49
4.2.4 Fase *Offline*: Agrupamento da estrutura de sumarização 51
4.3 Estratégia de Sumarização Local e Global 52
4.4 Estratégia de Agrupamento Distribuído 54
4.5 Considerações Finais 56

5 Metodologia para Validação 57
5.1 Ferramentas de Software 57
5.2 Ambiente de execução 60
5.3 Conjunto de Exemplos 60
5.3.1 Conjuntos Sintéticos 61
5.3.2 Conjuntos Reais 61
5.4 Parâmetros e Algoritmos Utilizados 63
5.4.1 Parâmetros da Sumarização Local e Global 63
5.4.2 Parâmetros do Agrupamento Distribuído 64
5.5 Métricas de Avaliação 65
5.5.1 Métricas Internas 65
5.5.2 Métricas Externas 66
5.5.3 Métricas de Desempenho 68
5.6 Considerações Finais 68

III Análises e Resultados 70

6 Resultados do *d-FuzzStream* 71
6.1 Experimentos 71
6.2 Estruturas de Sumarização e Agrupamento 72
6.3 Métricas de Avaliação 74
6.3.1 Métricas Informativas 74
6.3.2 Métrica Interna 76

6.3.3 Métricas de Desempenho 77
6.4 Considerações Finais 77

7 Resultados do *Distributed d-FuzzStream* 78
7.1 Estruturas de Sumarização e Agrupamentos 78
7.2 Métricas de Agrupamento 92
7.2.1 Métricas Internas 92
7.2.2 Métricas Externas 96
7.3 Métricas de Desempenho 99
7.3.1 Desempenho da estratégia de sumarização local e global 99
7.3.2 Desempenho da estratégia de agrupamento distribuído 100
7.4 Considerações Finais 102

Conclusão 104

Referências 107

Introdução

No mundo contemporâneo tem se tornado cada vez mais comum a coleta, a transmissão e o compartilhamento de dados por meio da Internet. A geração cada vez maior de dados produzidos diariamente impõe desafios para pesquisadores com relação à criação de ferramentas e técnicas que permitam processar esses grandes volumes de dados, superando as limitações de recursos computacionais.

Na última década, cresceu o interesse em gerenciar sequências ilimitadas de dados que são produzidas em altas taxas. Estas sequências são conhecidas como fluxo contínuo de dados (AGGARWAL, 2007) (GAMA, 2010) ou, como será referido de maneira simplificada neste documento, fluxo de dados. Os fluxos de dados se diferenciam das bases de dados comuns pois os dados chegam continuamente, têm tamanho possivelmente infinito e possuem distribuição não estacionária (SILVA et al., 2013).

Modelos que lidam com fluxos contínuo de dados se tornaram parte integral de diversos domínios de mercado, permitindo rápida resposta e adaptação aos eventos que ocorrem ao longo do tempo. Esses modelos fazem uso de dados gerados a partir de redes de sensores, análises meteorológicas, análises da bolsa de valores, monitoramento de tráfego de redes, entre outros (SILVA et al., 2013). Assim como ocorre para outros domínios de dados, existe interesse na extração de conhecimento útil de fluxos de dados usando técnicas automáticas.

A investigação de métodos computacionais capazes de adquirir conhecimento de forma automática refere-se a uma área da Inteligência Artificial conhecida como Aprendizado de Máquina. Esses métodos permitem que sistemas computacionais não só aprendam, como otimizem seus desempenhos de forma a torná-los mais precisos. A utilização do aprendizado de máquina tem se tornado crucial em um grande número de aplicações no mercado, permitindo encontrar informações úteis dentro de um conjunto de dados, e assim auxiliando na tomada de decisão.

Técnicas clássicas de aprendizado de máquina assumem que existe uma quantidade finita de dados, gerados por uma distribuição probabilística estacionária. Os dados

estão disponíveis podem ser analisados a todo momento. Entretanto, fluxos produzem dados continuamente, o que torna inviável o armazenamento de todos os dados, seja em memória principal ou até mesmo em memória secundária. Sendo assim, extrair potenciais informações relevantes de fluxos de dados exige técnicas que superam as restrições de armazenamento e que realizam um processo de aprendizado continuamente, adaptando-se às mudanças na distribuição dos dados.

As abordagens mais comuns no aprendizado em fluxo de dados são as abordagens supervisionadas e não supervisionadas. O aprendizado supervisionado realiza a extração de conhecimento a partir de conjuntos de dados que possuem um atributo que representa o conceito que se deseja aprender, denominado classe ou rótulo. Entretanto, esse tipo de aprendizagem parte do princípio que este atributo está disponível, o que pode não ocorrer. Nesses casos, são utilizados métodos não supervisionados, como o agrupamento.

No agrupamento em fluxo de dados existem diversas técnicas relevantes, como extensões incrementais de algoritmos de agrupamentos clássicos, algoritmos incrementais baseados em grânulos de informação, medida de potencial, excentricidade, tipicalidade e medidas de densidade. Neste trabalho destaca-se técnicas baseadas no *Framework Online-Offline* (AGGARWAL et al., 2003), que sumarizam os exemplos continuamente por meio de estruturas de sumarização, e depois realizam o agrupamento, resultando em uma potencial partição do fluxo de dados.

Frequentemente, objetos encontrados no mundo real não possuem um conceito preciso, exigindo que o processo de aprendizado consiga representar essas incertezas inerentes ao mundo real e refletida nos dados. Desta forma, diversas propostas de aprendizado em fluxo de dados também passaram a integrar conceitos da teoria de conjuntos e lógica *fuzzy*.

Nessa teoria, elementos possuem graus de pertinência que variam no intervalo $[0, 1]$, permitindo tratar imprecisões e incertezas em relação às variáveis no domínio a ser estudado. Assim, essa teoria é utilizada com o objetivo de promover maior flexibilidade e adaptabilidade do conhecimento aprendido.

Fluxos de Dados representam um domínio de problemas que tende a aumentar, acompanhando a evolução tecnológica. Com aumento da produção de dados, algoritmos de aprendizado de máquina necessitam ter capacidade de serem escaláveis em relação a velocidade dessa produção, caracterizando assim, um grande desafio a ser tratado.

Motivação e Objetivos

A principal dificuldade que motiva este estudo pode ser colocada como segue. Dados provenientes dos fluxos tipicamente só podem ser armazenados por um curto período,

e o tempo de processamento deve ser compatível com a taxa de produção de novos dados. Caso o tempo de processamento seja maior que o tempo de chegada de um próximo exemplo, os dados se acumulam continuamente, tendo de ser descartados. Assim, informações importantes podem ser perdidas e o aprendizado tende a ficar cada vez mais defasado, descaracterizando o processamento em tempo real.

Para contornar esse problema, muitos algoritmos de aprendizado em fluxo de dados utilizam a técnica de *sampling*, que consiste em escolher um subconjunto representativo de exemplos dentro do conjunto de dados. Mesmo assim, esse subconjunto de exemplos também pode não contemplar informações importantes como mudanças na distribuição dos dados, caso que só seria possível de ser identificado se todos os exemplos fossem processados (GAMA, 2012).

Recentemente, ferramentas de computação distribuída começaram a se popularizar, principalmente para processamento de grandes volumes de dados. Elas trazem como vantagem a execução coordenada de tarefas em diversos computadores, espalhados geograficamente, possibilitando processar maior volume de dados em menor tempo. Com essa popularização, muitos algoritmos de aprendizado de máquina foram adaptados para serem executados de forma distribuída por essas ferramentas (BACKHOFF; NTOUTSI, 2016).

Apesar disso, grande parte desses algoritmos são adaptações de algoritmos de aprendizado de máquina clássico para lidar com grandes volumes de dados. Assim, existe uma carência de algoritmos para aprendizado em fluxo de dados que possam ser executados de forma distribuída, principalmente no âmbito de aprendizado não supervisionado.

Em vista disso, a proposta deste trabalho é uma abordagem de agrupamento em fluxo de dados que utiliza computação distribuída para ser escalável em relação à velocidade do fluxo de dados.

Para tanto, essa proposta utiliza como base o *Framework Online-Offline*, dividindo o aprendizado em duas fases: *online* e *offline*. A fase *online* utiliza uma estratégia de sumarização em estruturas distintas armazenadas localmente, que são posteriormente condensadas em uma única estrutura em uma fase intermediária. Já na fase *offline*, o conjunto de exemplos é replicado e agrupado com diversos números de grupos e por diferentes algoritmos de agrupamento. O algoritmo de agrupamento faz uso do conceito de conjuntos *fuzzy* em todo o processo para prover maior flexibilidade em relação à imprecisão dos dados.

Contribuições

Para a aplicação da abordagem distribuída foi necessária a criação de algoritmo de agrupamento *fuzzy* em fluxo de dados, baseado no *Framework Online-Offline*, que tivesse um mecanismo eficiente para identificação de exemplos *outliers* e similaridades entre os micro-grupos que compõem a estrutura de sumarização. Desta maneira, as maiores contribuições deste trabalho são:

d-FuzzStream - algoritmo de agrupamento *fuzzy* em fluxo de dados que utiliza o *Framework Online-Offline*. Ele é baseado em algoritmos encontrados na literatura e introduziu os conceitos de similaridade e dispersão *fuzzy* na fase *online* do *framework*. Apresentou melhorias no processo de sumarização e diminuiu a complexidade em relação ao algoritmo original no qual foi baseado;

Distributed d-FuzzStream - versão distribuída do *d-FuzzStream* que utiliza a abordagem distribuída para processar fluxos de dados com grande quantidade de dados mais rapidamente.

A análise dos resultados de experimentos produzidos para este trabalho mostra que a abordagem distribuída trouxe ganhos significativos em termos de desempenho, e forneceu conhecimento comparável ao gerado sem o uso da abordagem distribuída.

Organização

O presente trabalho está organizado em sete capítulos e a conclusão, conforme:

Capítulo 1 - **Aprendizado de Máquina Clássico**: descreve conceitos gerais a respeito do aprendizado de máquina clássico, supervisionado e não supervisionado.

Capítulo 2 - **Aprendizado em Fluxos de Dados**: caracteriza o aprendizado em fluxo de dados, apresentando os conceitos gerais sobre a área e outras noções que fundamentam o desenvolvimento de novos métodos de aprendizado. O capítulo foca no aprendizado por meio de agrupamento, apresentando conceitos e técnicas já presentes na literatura.

Capítulo 3 - **Computação Distribuída**: apresenta contextualização quanto à área de Computação Distribuída, com foco em ferramentas de processamento distribuído. Neste capítulo é traçada uma visão geral do estado-da-arte em relação às abordagens distribuídas de agrupamento e agrupamento *fuzzy*.

Capítulo 4 - **Abordagem Distribuída para Agrupamento em Fluxo Contínuo de Dados**: detalhamento geral da proposta e explicação das estratégias

desenvolvidas para aplicação na fase de abstração e agrupamento do *Framework Online-Offline*.

Capítulo 5 - **Metodologia para Validação**: expõe a construção dos experimentos, descrevendo as ferramentas, os conjuntos de dados e métricas de avaliação empregadas.

Capítulo 6 - **Resultados do *d-FuzzStream***: apresenta comentários e análises dos resultados da comparação entre o *d-FuzzStream* e o algoritmo original, ao qual foi baseado, explicitando suas características.

Capítulo 7 - **Resultados do *Distributed d-FuzzStream***: expõe comentários e análises sobre os resultados obtidos nos experimentos, baseados em retratos do processo de agrupamento e valores das métricas avaliadas. Retrata-se as características da abordagem distribuída e suas vantagens em velocidade de processamento.

Parte I

Revisão Bibliográfica

1

Aprendizado de Máquina Clássico

Processos de aprendizagem incluem a aquisição de conhecimento por meio de prática ou instrução, organização de novos conhecimentos em representações gerais e efetivas, e a descoberta de novos fatos e teorias por meio da observação e da experimentação. Desta forma, o Aprendizado de Máquina (AM) visa implantar esses processos de aprendizagem nos computadores, investigando métodos computacionais capazes de aprender e adquirir conhecimento de forma automática (CARBONELL; URBANA; MITCHELL, 1983).

Um dos métodos mais utilizados em AM é o aprendizado indutivo. A ideia deste tipo de aprendizado é obter novos conhecimentos por meio de inferências indutivas sobre um conjunto de exemplos (dados, instâncias ou objetos), que por sua vez são descritos por um conjunto de atributos (MITCHELL, 1997). Portanto, o aprendizado indutivo é um método de aprendizado com o objetivo de encontrar conceitos gerais utilizando exemplos específicos do domínio a ser estudado (MICHALSKI; CARBONELL; MITCHELL, 1986). As duas principais técnicas de aprendizado indutivo são conhecidas como aprendizado supervisionado e não supervisionado.

1.1 Aprendizado Supervisionado

O aprendizado supervisionado tem como objetivo a extração de conhecimento pelo desenvolvimento de um modelo geral baseado em um conjunto de exemplos rotulados. Esses exemplos rotulados têm como característica um atributo especial que representa um conceito do domínio, denominado classe ou rótulo. Posteriormente, o modelo geral desenvolvido por esse tipo de aprendizado é utilizado para prever os valores de rótulo em dados adicionais não rotulados (HAN; KAMBER; PEI, 2011).

As técnicas de aprendizado supervisionado se diferenciam pelo tipo de rótulo presente nos exemplos. Para rótulos com valores nominais são utilizados métodos de classificação, enquanto que para rótulos com valores contínuos são utilizados métodos de regressão. Exemplos desses métodos incluem árvores de decisão (QUINLAN, 1986), re-

des neurais (BISHOP, 1995), métodos estatísticos (DUDA; HART, 1973) e abordagens evolutivas (GOLDBERG, 1989).

1.2 Aprendizado Não Supervisionado

O aprendizado não supervisionado é utilizado em casos onde os exemplos não possuem informações de rótulos, ou seja, não existem conceitos prévios relacionados ao conjunto de exemplos. Assim, o objetivo deste método de aprendizado é encontrar uma estrutura que permita extrair e explicar aspectos básicos desse conjunto.

O agrupamento de exemplos é uma técnica frequentemente utilizada no aprendizado não supervisionado. Essa técnica realiza a divisão dos exemplos em grupos de forma que exemplos semelhantes pertençam a um mesmo grupo e exemplos distintos pertençam a grupos distintos (JAIN; MURTY; FLYNN, 1999). A noção de similaridade entre os exemplos é definida de diferentes maneiras de acordo com a finalidade do processo e as características dos dados. Para exemplos que possuem atributos inteiramente numéricos, a avaliação de similaridade mais comum baseia-se em métricas de distância (TAN; STEINBACH; KUMAR, 2005).

Um dos exemplos de algoritmo de agrupamento mais populares e simples é o algoritmo particional *k-means* (MACQUEEN, 1967). Esse algoritmo tem como objetivo agrupar os dados em k grupos disjuntos, de maneira que a soma das distâncias entre os exemplos pertencentes a um grupo e seu respectivo centro seja mínima. Apesar de sua simplicidade, o *k-means* se destaca por ser amplamente aplicado e utilizado como base para o desenvolvimento de novos algoritmos.

1.3 Considerações Finais

Atualmente, o uso do AM abrange diversos sistemas como: detecção de fraudes em transações bancárias, análise estatísticas da bolsa de valores e até sistemas de recomendação de músicas, vídeos e notícias.

Com a evolução de novas tecnologias e ampliação do acesso à Internet, muitos sistemas passaram a produzir grandes volumes de dados em curtos espaços de tempo. Sendo assim, os conjuntos de dados passaram a ter distribuições estatística mutáveis de acordo com o tempo, tamanho indefinido e potencialmente infinito.

Técnicas clássicas de AM assumem que o conjunto de exemplos é finito, possui distribuição de probabilidade estática e está disponível a qualquer instante durante o processo de aprendizagem. Em vista disso, a busca por novas técnicas que permitam a aquisição de conhecimento de forma automática, sem as limitações das técnicas clássicas, representa um desafio recente dentro da área de AM.

2

Aprendizado em Fluxo de Dados

Um Fluxo de dados (FD) pode ser definido como uma sequência ilimitada de dados que são gerados continuamente em altas taxas (AGGARWAL, 2007) (GAMA; GABER, 2007). Aplicações de FDs incluem mineração de dados em redes de sensores, análises meteorológicas, análise do mercado de ações, monitoramento de tráfego de redes, entre outras (AGGARWAL, 2014)(NGUYEN; WOON; NG, 2015). FDs se diferenciam significativamente dos conjuntos de exemplos ditos tradicionais em diversos aspectos (SILVA et al., 2013):

- Os exemplos chegam de maneira contínua e constante.
- Não há controle sobre a ordem na qual os exemplos chegam para serem processados.
- Os fluxos têm tamanho potencialmente infinito.
- Exemplos devem ser descartados (ou armazenados por um curto período de tempo) após o processamento.
- A distribuição dos dados é possivelmente não estacionária, ou seja, muda com o tempo.

Devido à quantidade de dados provenientes de um fluxo ser indefinida e possivelmente infinita, não é possível armazenar todo esse conjunto de exemplos em memória ou disco, sendo comum que apenas os exemplos mais recentes estejam disponíveis. Portanto, algoritmos de aprendizado em fluxo de dados (AFD) devem ser capazes de incorporar novas informações na velocidade em que elas aparecem. Ao mesmo tempo, eles devem eliminar os efeitos dos dados antigos (ou dados históricos) (NGUYEN; WOON; NG, 2015). Segundo Domingos e Hulten (2001), algoritmos de AFD devem seguir os seguintes critérios:

- O tempo de processamento de um exemplo deve ser curto e aproximadamente constante ao longo do processo de aprendizagem.

- O algoritmo deve considerar um limite de memória a ser utilizada, independentemente do número de exemplos já processados.
- Um modelo deve estar disponível a qualquer momento, e não somente após o processamento de todo o conjunto de dados, uma vez que ele é infinito.
- Quando ocorrer uma mudança nos dados, o modelo deve ser atualizado, incluindo as informações dos exemplos passados que não estão obsoletos, e removendo os efeitos dos dados obsoletos.

Dois contextos distintos podem ser relacionados ao AFD. A aprendizagem baseada em atributos é utilizada para identificar um conjunto de atributos ou FDs correlacionados (e.g, fonte de dados de sensores) que possuem comportamentos semelhantes ao longo do tempo. Um exemplo de aplicação é identificar comportamentos em uma rede de sensores (GAMA; RODRIGUES; LOPES, 2011).

Já no contexto de aprendizagem baseada em exemplos, o objetivo é a identificação de características similares dos exemplos presentes em um FD (SILVA et al., 2013). Nesse contexto, é comum que os exemplos não sejam independentes no tempo, ou não sejam provenientes de uma mesma distribuição, sendo altamente provável que a distribuição desses exemplos sofra mudanças, conhecidas como desvios de conceito.

2.1 Desvios de Conceito

Desvios de conceito se referem à mudança na distribuição dos dados que podem introduzir conceitos novos, nunca identificados anteriormente, e também conceitos previamente identificados que desapareceram e voltaram a aparecer após algum tempo (GAMA et al., 2014). Entretanto, é importante ressaltar que estudos mais recentes no contexto de detecção de novidades passaram a adotar o termo evolução de conceito para a descoberta de novos conceitos (FARIA et al., 2016).

Segundo Gama et al. (2014), os desvios de conceito podem ocorrer de diversas formas, conforme ilustrado na Figura 2.1 e listado abaixo:

- **Abruptamente:** Um conceito muda de forma repentina.
- **Incrementalmente:** A distribuição muda de forma lenta, podendo ser identificados conceitos intermediários.
- **Gradualmente:** O conceito muda repentinamente, mas continua voltando ao conceito anterior durante um tempo.
- **Recorrentemente:** Os conceitos aparecem e desaparecem da distribuição.

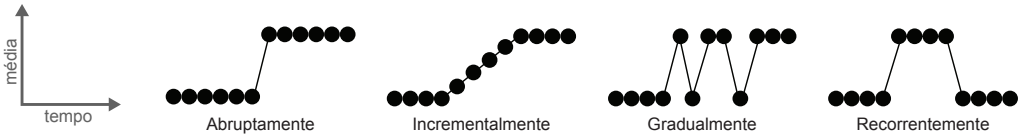


Figura 2.1: Padrões de desvio de conceito

Um dos grandes desafios na identificação de desvios de conceito é a identificação e separação de ruídos ou *outliers*, pois, se forem levados em consideração, podem causar divergências entre o conceito identificado e o real (GAMA et al., 2014)(CHANDOLA; BANERJEE; KUMAR, 2009).

No contexto de AFD, os desvios de conceito demonstram a evolução do FD de acordo com o tempo. Contudo, os dados históricos e os dados recentes não podem ter a mesma importância, pois só assim é possível identificar as características não estacionárias do FD (CHEN; TU, 2007). Uma das abordagens mais comuns adotadas para resolução desse problema é a abordagem de janelas temporais (BARBARÁ, 2002) (BABCOCK et al., 2003) (GAMA, 2010), apresentada na seção a seguir.

2.2 Janelas Temporais

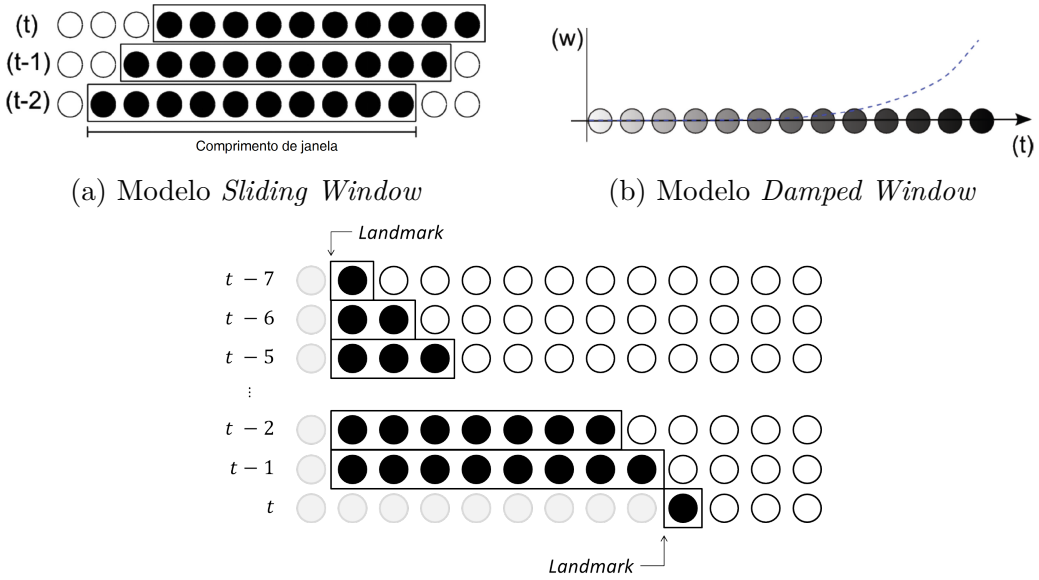
Esta abordagem divide o conjunto de exemplos em subconjuntos (janelas) no decorrer do tempo. Para que isso seja possível, cada novo exemplo é associado a uma marca temporal, o que permite determinar se aquele exemplo pertence ou não a uma determinada janela. Atualmente existem diversos modelos de janelas temporais, mas os três mais relevantes de acordo com Zhu e Shasha (2002) são: *sliding window*, *damped window* e *landmark window*.

Sliding Window: Este modelo considera apenas as informações mais recentes. Por meio de uma estrutura do tipo *first in, first out* (FIFO), um exemplo mais antigo é removido para adicionar o exemplo recém chegado (Figura 2.2a). Diversos algoritmos de agrupamento utilizam esse modelo, por exemplo, os apresentados em (BABCOCK et al., 2003), (ZHOU et al., 2008) e (REN; MA; REN, 2009).

Damped Window: Também conhecido como *time-fading*, este modelo associa pesos aos exemplos pertencentes ao FD. Ao chegar, um exemplo recebe um peso que decai de acordo com o tempo (JIANG; GRUENWALD, 2006). É usualmente adotado em algoritmos baseados em densidade (CAO et al., 2006) (CHEN; TU, 2007) (ISAKSSON; DUNHAM; HAHSLER, 2012). Na figura Figura 2.2b pode ser visualizado um exemplo do modelo, onde o decaimento é representado de acordo com o degradê dos exemplos.

Landmark Window: Processar o FD baseado no modelo de *Landmark* requer

lidar com porções disjuntas do fluxo, denominadas *chunks*, separadas por exemplos relevantes (*landmarks*). Esses *Landmarks* podem ser definidos tanto em termos de tempo (e.g. dia, mês) quanto em termos de número de elementos observados desde o *landmark* anterior (METWALLY; AGRAWAL; El Abbadi, 2005). Exemplos que chegam após um *landmark* são armazenados em uma estrutura de dados, e removidos inteiramente quando atingem o próximo *landmark*. Algoritmos de agrupamento baseados neste modelo são apresentados em (O'CALLAGHAN et al., 2002) (AGGARWAL et al., 2003). A figura Figura 2.2c apresenta um exemplo deste modelo.



(a) Modelo *Sliding Window*

(b) Modelo *Damped Window*

(c) Modelo *Landmark Window*

Figura 2.2: Exemplos ilustrativos de modelos de janela temporal

Por meio dessa abordagem é possível limitar o volume de dados que será considerado pelo algoritmo de AFD. Isto permite identificar as características não estacionárias inerentes dos FDs.

Propostas de algoritmos de AFD possuem inspiração em métodos clássicos de AM, podendo realizar a aprendizagem de forma supervisionada, não supervisionada ou semissupervisionada. As próximas sessões apresentam conceitos de classificação e agrupamento em FD, com maior foco na aprendizagem não supervisionada.

2.3 Classificação em FD

Na classificação em FD, é comum que os dados não-rotulados sejam usados para fazer previsão, e os dados rotulados são usados para treinar o modelo. Diferentemente dos modelos clássicos, modelos de classificação em FD devem ser continuamente atualizados para incluir as propriedades dos exemplos rotulados que cheguem, adequando-se também aos desvios de conceitos (AGGARWAL, 2014).

Algoritmos de classificação em FD podem prover um aprendizado incremental, onde os novos dados são mais impactantes, ou de forma decremental, onde os dados que são removidos de uma janela do tipo *sliding* possuem maior impacto (AGGARWAL, 2014).

Por ser uma técnica de aprendizado supervisionado, a classificação considera que exemplos do FD sejam devidamente rotulados, o que pode não ser a realidade em certos casos. Este trabalho tem interesse por abordagens de agrupamento não supervisionado em FD, apresentadas na sessão a seguir.

2.4 Agrupamento em FD

O agrupamento em FD é definido como um processo contínuo capaz de agrupar exemplos dentro de restrições de memória e tempo (GAMA, 2010). Assim, o desenvolvimento de novos algoritmos de agrupamento em FD deve idealmente seguir os seguintes requisitos (BABCOCK et al., 2002)(BARBARÁ, 2002) (TASOULIS; ADAMS; HAND, 2006) (BIFET, 2010):

- Prover resultados frequentes por meio do processamento rápido e incremental de exemplos.
- Adaptar-se rapidamente às evoluções do FD, detectando novos grupos e removendo os antigos.
- Prover uma representação ou modelo que não cresça de acordo com o número de exemplos processados.
- Detectar novos comportamentos *outliers* de forma rápida.

Diversos algoritmos clássicos foram estendidos para trabalhar com as restrições e características do AFD. Essas extensões dividem o FD em *chunks* de forma que seja possível o seu armazenamento em memória. Assim, o agrupamento é feito de forma incremental, utilizando os resultados obtidos do *chunk* anterior, como centroides, para o agrupamento do próximo. Entretanto, técnicas deste tipo possuem problemas na identificação de desvios de conceito (NGUYEN; WOON; NG, 2015).

Assim, uma estratégia bastante utilizada no agrupamento em FD é a divisão do processamento em duas fases (CAO et al., 2006) (YANG; ZHOU, 2006) (ZHOU et al., 2008): fase de abstração dos dados (ou fase *online*) e fase de agrupamento (fase *offline*), como demonstrada na figura Figura 2.3. Essa estratégia é também conhecida como *Framework Online-Offline* (FOO) (AGGARWAL et al., 2003).

No agrupamento clássico, todos os exemplos estão disponíveis para acesso e processamento recorrentes, o que não é possível no domínio de FD. Assim, a fase de abstração

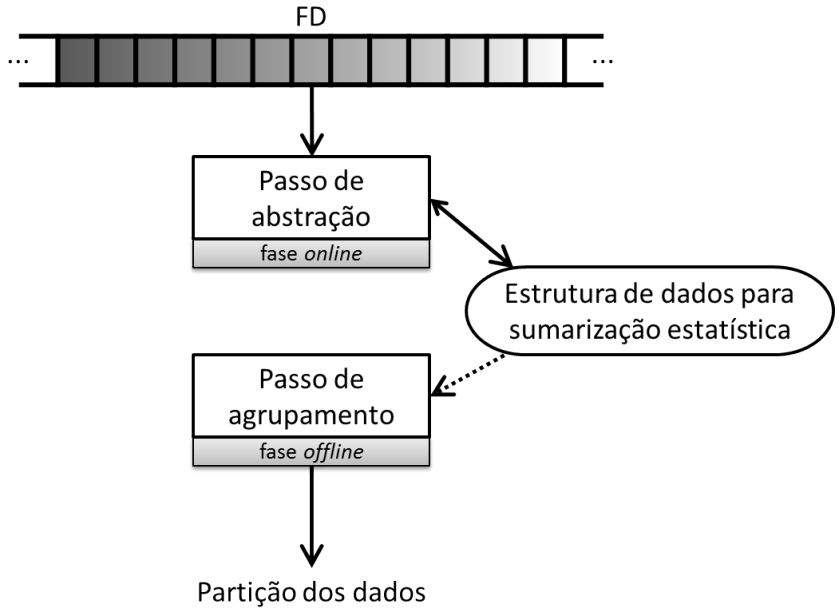


Figura 2.3: Funcionamento do *Framework Online-Offline*

de dados tem como objetivo criar uma representação para os exemplos encontrados no FD até um determinado instante, sem a necessidade de armazená-los. Para que isto ocorra, os exemplos são sumarizados em uma estrutura de dados conhecida como estrutura de sumarização.

Na fase de agrupamento, a estrutura de sumarização é utilizada para gerar a representação do modelo por meio de algoritmos de agrupamento clássicos como *k-means*, *fuzzy c-means*(BEZDEK, 1981) e *DBSCAN* (**??**). Vale ressaltar que a fase *online* é executada continuamente, enquanto que a fase *offline* é apenas executada de tempos em tempos, ou por ordem explícita do usuário.

Como este trabalho utiliza o FOO, a sessão a seguir apresenta estruturas de sumarização de exemplos que são utilizadas na fase de abstração dos dados.

2.5 Estruturas de Sumarização de Exemplos

Considerando-se que armazenar os exemplos provenientes do FD é inviável, diversas estruturas de sumarização podem ser utilizadas para representar esses dados. Nesta seção são apresentadas as quatro estruturas mais empregadas (SILVA et al., 2013): vetor de atributos, arranjo de protótipos, árvores de *coreset* e grade de dados.

2.5.1 Vetor de Atributos

Introduzido inicialmente no algoritmo *BIRCH* (ZHANG; RAMAKRISHNAN; LIVNY, 1996), o vetor de atributos, ou *cluster features* (CF), possui três componentes: a cardi-

nalidade do grupo (N) e duas estruturas n-dimensionais que representam a soma linear dos exemplos do grupo (LS) e a soma quadrática dos exemplos do grupo (SS). Por meio destes três componentes é possível calcular métricas como o centroide (Equação 2.1) e o raio de um grupo (Equação 2.2).

$$centroide = \frac{LS}{N} \tag{2.1}$$

$$raio = \sqrt{\frac{SS}{N} - \left(\frac{LS}{N}\right)^2} \tag{2.2}$$

Além disso, vetores de CF possuem as seguintes propriedades:

- **Incremental:** Um novo exemplo $\mathbf{x^j}$ pode ser inserido no CF pela atualização das estatísticas demonstradas abaixo.

$$LS \leftarrow LS + \mathbf{x^j}$$
$$SS \leftarrow SS + \left(\mathbf{x^j}\right)^2$$
$$N \leftarrow N + 1$$

- **Aditivas:** Dois vetores $CF1$ e $CF2$ podem ser combinados somando os seus componentes, resultando em um único CF.

Posteriormente, o conceito de vetor de CF foi estendido em diversas propostas como o *Micro Cluster* (MiC) no algoritmo *CluStream*, o *Fuzzy Micro Cluster (FMiC)*, proposto no *FuzzStream* (LOPES; CAMARGO, 2017) e o *weighted CF vector*, utilizado pelo *ClusTree* (KRANEN et al., 2011).

2.5.2 Arranjo de Protótipos

O Arranjo de Protótipos é uma estrutura de sumarização simplificada (DOMINGOS; HULTEN, 2001) cujo objetivo é armazenar um conjunto de protótipos (centroides, medoides, etc) que sumarizem a partição dos dados.

Exemplos desses algoritmos são o *Stream* (GUHA et al., 2000) e *Stream LSearch* (O'CALLAGHAN et al., 2002). Neles, o fluxo é dividido em pedaços (*chunks*) que são agrupados em k grupos. Logo após, todos os grupos formados pelos *chunks* são combinados e agrupados novamente formando a mesma quantidade de grupos k.

2.5.3 Árvore de Coreset

Esse tipo de estrutura de sumarização em árvore binária difere significativamente das outras. Cada nó armazena um conjunto de exemplos E_i (apenas nos nós folhas), o

protótipo x^{p_i} de E_i, número de exemplos em E_i e a soma quadrática das distâncias entre os exemplos em E_i e x^{p_i}. A atualização dessa estrutura segue os seguintes passos:

1. O nó raiz N_v armazena $2m$ exemplos (E_v).
2. Um exemplo do nó raiz, x^{p_v} é escolhido aleatoriamente como protótipo.
3. O nó é expandido em outros dois nós folhas N_{v1} e N_{v2}.
4. Um exemplo x^{q_v} de E_v é escolhido aleatoriamente do nó raiz.
5. O conjunto E_v é dividido entre os nós (E_{v1} , E_{v2}) onde:

   $$E_{v1} = \{x^{i_v} \in E_v \quad | \quad \|x^{i_v} - x^{p_v}\| < \|x^{q_v} - x^{p_v}\|\}$$

   $$E_{v2} = E_v - E_{v1}$$

6. As estatísticas N_{v1} e N_{v2} são atualizadas de forma que:

   $$x^{p_{v1}} = x^{p_v}$$

   $$x^{p_{v2}} = x^{q_v}$$

7. Se o número de nós for menor que m, um nó folha é escolhido para ser expandido e volta ao passo 3.

Essa estrutura de sumarização é empregada pelo StreamKM++ (ACKERMANN et al., 2012) e tem como objetivo a geração de m protótipos a partir de $2m$ exemplos.

2.5.4 Grade de Dados

Alguns algoritmos realizam a sumarização dos exemplos por meio de uma estrutura de grade (CAO et al., 2006) (PARK; LEE, 2007) (CHEN; TU, 2007) (GAMA; RODRIGUES; LOPES, 2011), particionando o espaço n-dimensional em células de densidade.

Novos exemplos que chegam são mapeados para uma célula e utilizando mecanismos de esquecimento, como janelas temporais, os exemplos mais antigos são removidos. Esse processo contínuo torna as células densas, com grande número de exemplos, ou esparsas, com poucos exemplos.

2.6 Técnicas de Agrupamento em FD

Nessa sessão, é apresentado o estado da arte dos algoritmos de agrupamento que fazem uso do FOO. Apesar de se basearem em um mesmo modelo, essas técnicas apresentam diferenças na estratégia utilizada nas duas fases do *framework*.

CluStream

O algoritmo *CluStream* (AGGARWAL et al., 2003) reduz o FD a um conjunto de MiCs, que são posteriormente convertidos em exemplos ponderados para a criação de um possível particionamento dos exemplos em um número menor de grupos, chamados *Macro-Clusters* (MaC).

O MiC é uma extensão de CF e é definido por $(\overline{CF2^e}, \overline{CF1^e}, CF2^t, CF1^t, n)$, onde $\overline{CF2^e}$ é a soma quadrática dos exemplos do grupo, $\overline{CF1^e}$ é a soma linear dos exemplos que pertencem ao grupo, $CF2^t$ é a soma quadrática de *timestamps* (tempo de chegada dos exemplos), $CF1^t$ é a soma linear de *timestamps* e n é a cardinalidade do MiC.

Inicialmente é aplicado o algoritmo *k-means* em um conjunto inicial de exemplos para encontrar o número m de MiCs definido pelo usuário. Assim, começa a fase *online* do algoritmo, onde a distância de cada novo exemplo e_j é calculada para cada MiC. Considerando o MiC mais próximo de e_j, é avaliado se o exemplo está dentro de um limiar máximo para o MiC.

Esse limiar é definido proporcionalmente para um parâmetro de entrada para o algoritmo e o desvio padrão estimado para o MiC. Se e_j está dentro desse limiar máximo, então é feita a atribuição ao MiC. Caso contrário ocorre a criação de um novo MiC, que requer a redução da estrutura de sumarização a fim de obter espaço para armazenamento.

Existem duas formas de redução da estrutura de sumarização, sendo a primeira a remoção de um MiC considerado *outlier*. Para determinar se um MiC é *outlier*, uma marca de relevância é calculada para cada MiC, obtida pela média e desvio padrão de *timestamps*. Se a relevância de um MiC estiver abaixo de um limiar, então o MiC é um *outlier* e pode ser eliminado da estrutura. Caso nenhum MiC seja avaliado como *outlier*, dois MiCs mais próximos entre si são combinados em um único MiC.

A criação de MaC ocorre na fase *offline*, convertendo os MiCs em exemplos e aplicando o algoritmo *k-means*. Essa fase só inicia por meio de requisição explícita do usuário, permitindo a flexibilidade de explorar o FD em diferentes instantes de tempo.

ClusTree

O algoritmo *ClusTree* (KRANEN et al., 2011) utiliza uma estrutura de sumarização chamada *weighted CF vector* (vetor CF ponderado), onde os vetores que não são atualizados recentemente perdem importância ao longo do tempo. Essa estrutura é armazenada nos nós de uma estrutura de árvore hierárquica (R-Tree), onde o vetor CF do nó pai é a soma dos vetores dos nós filhos.

Inicialmente um conjunto de exemplos é ordenado, ou agrupados, e utilizados para inicializar a árvore. Quando um novo exemplo chega, o algoritmo percorre o caminho até

o nó folha com a maior similaridade (menor distância Euclidiana) com o exemplo. Se a similaridade entre o exemplo e algum vetor de CF presente no nó estiver acima de um limiar, o exemplo é então adicionado ao vetor de CF mais similar. Caso contrário, um novo vetor de CF é criado e adicionado ao nó. Entretanto, se o nó estiver cheio, ele é dividido de forma que a distância entre os vetores de CF seja mínima. Adicionalmente, o algoritmo pode combinar os CFs mais próximos enquanto não há novos exemplos a serem processados.

Esse algoritmo aplica uma estratégia que tenta se adaptar tanto a FDs lentos quantos FDs que produzem dados rapidamente. Caso um novo exemplo chegue enquanto o processo de adição do exemplo anterior a um nó folha ainda estiver em curso, o exemplo antigo é adicionado a um *buffer* presente na subárvore em que ele se encontra no momento, iniciando o processamento do novo exemplo. Quando um novo exemplo passar pela subárvore onde existem exemplos no *buffer*, esses exemplos são removidos e processados conjuntamente com o novo exemplo. Apesar disso, essa estratégia armazena os exemplos para serem processados depois, o que pode ocasionar um acúmulo ao longo prazo.

D-Stream

O algoritmo *D-Stream* (CHEN; TU, 2007) utiliza uma grade d-dimensional como estrutura de sumarização, dividindo o espaço de d atributos em células. Na fase *online*, os exemplos do FD são mapeados nas células das grades de acordo com os valores de seus atributos.

Na fase *offline* ocorre o particionamento dos exemplos. Cada célula considerada densa é associada a um grupo distinto, e nos intervalos seguintes os grupos são ajustados. Esses ajustes ocorrem por meio da identificação de células densas e esparsas, onde as células relativamente mais densas são mescladas a células vizinhas, formando um grupo.

2.7 Técnicas de Agrupamento *Fuzzy* em FD

Na teoria de conjuntos clássica um dado elemento do domínio pertence ou não pertence a um determinado conjunto. Assim, esses conjuntos apenas podem assumir representações binárias, ou valores $(0, 1)$, e são frequentemente referenciados pelo termo em inglês *crisp*.

Já na teoria dos conjuntos *fuzzy* cada elemento possui um grau de pertinência a um determinado conjunto, permitindo representar matematicamente certos termos linguísticos subjetivos como: “aproximadamente”, “em torno de”, dentre outros.

Formalmente, um conjunto *fuzzy* A para um conjunto de dados X é caracterizado por uma função de pertinência $f_A(x)$, que associa cada elemento x pertencente ao conjunto

X para um valor real no intervalo $[0, 1]$ (ZADEH, 1965).

Assim como muitos domínios de AM, o AFD também pode ser beneficiado pela extensão *fuzzy* de métodos de aprendizado, incorporando características definidas pela teoria de conjuntos e lógica *fuzzy*.

Boa parte dos algoritmos de agrupamento *fuzzy* em FD utilizam como base o algoritmo Fuzzy C-Means (FCM) (BEZDEK, 1981), extensão *fuzzy* do algoritmo *k-means*. Nessa extensão, é inicialmente definido o número de grupos e uma constante de fuzzificação ($m > 1$). Então, cada exemplo está relacionado a k grupos por um grau de pertinência, sendo que a soma das pertinências dos exemplos aos grupos deve ser sempre igual a 1.

Inicialmente, o FCM gera aleatoriamente uma matriz de pertinência com elementos u_{ij} que representam o grau de pertinência de um exemplo e_j a um grupo c_i. Em seguida são gerados os centroides iniciais c, calculados pela Equação 2.3.

$$c_i = \frac{\sum_{j=1}^{n} u_{ij}^m e_j}{\sum_{j=1}^{n} u_{ij}^m} \tag{2.3}$$

Então, a matriz de pertinência é atualizada a partir da distância entre os exemplos e os centroides (Equação 2.4), e são gerados novos centroides a partir da Equação 2.3. Esse processo se repete até que a distância entre os novos centroides e os antigos seja menor que um limiar.

$$u_{ij} = \left[\sum_{l=1}^{k} \left(\frac{\|e_j - c_i\|}{\|e_j - c_l\|} \right)^{\frac{2}{m-1}} \right]^{-1}, \forall i, j \tag{2.4}$$

Por fim, o FCM resulta em um conjunto de centroides finais e uma matriz com os valores de pertinência para cada exemplo e_j em cada grupo final.

A abordagem mais comum utilizada para adaptar o FCM para AFD é a abordagem de *chunks*, onde o FD é dividido em porções disjuntas que são processadas separadamente.

O *Single Pass Fuzzy C-Means* (SPFCM) (HORE; HALL; GOLDGOF, 2007) é um exemplo de algoritmo que utiliza essa abordagem. Seu objetivo é agrupar base de dados de grande dimensionalidade que não podem ser armazenadas em memória.

Para tanto, o agrupamento é realizado aplicando o algoritmo *Weighted Fuzzy C-Means* (WFCM), versão ponderada do FCM (BEZDEK, 1981), em cada *chunk*. Os centroides encontrados em um *chunk* são adicionados ao conjunto de exemplos do próximo *chunk* ponderados de acordo com o grau de pertinência. Esse processo se repete até o processamento do último *chunk*. O SPFCM considera que todo novo exemplo vindo do FD possui peso de valor igual a 1 e o conjunto de centroides que representa o FD é aquele obtido pela aplicação mais recente do WFCM.

Outros algoritmos de agrupamento *fuzzy* baseados em *chunks* incluem o *weighted FCM - Adaptive Cluster* (MOSTAFAVI; AMIRI, 2012) e *Online Fuzzy C-Means* (OFCM)(HORE et al., 2008).

O *FuzzStream* (LOPES, 2016)(LOPES; CAMARGO, 2017) é um algoritmo de agrupamento *fuzzy* baseado no *CluStream*, realizando o processamento de exemplos um-a-um. Ele introduz o *Fuzzy Micro-Cluster* (FMiC), um vetor de CF onde os exemplos são ponderados pelo grau de pertinência. Neste caso, como um exemplo possui um grau de pertinência para todos os FMiCs, o algoritmo faz a atualização de todos os grupos ao invés de apenas um, como ocorre na estratégia *crisp*. Diferentemente do *CluStream*, o *FuzzStream* utiliza um número mínimo de exemplos para inicializar os FMiCs, e aplica o algoritmo FCM na fase *offline*. Uma outra versão do *FuzzStream*, o *decaying FuzzStream* (LOPES, 2016) aplica uma função de decaimento aos FMiCs.

2.8 Considerações Finais

O domínio de AFD introduziu diversas mudanças em relação ao AM clássico. A impossibilidade de armazenar todos os exemplos e a natureza não estacionária dos dados estimulou a criação de técnicas que utilizam mecanismos de esquecimento, como janelas temporais, e sumarização de exemplos, para extrair o comportamento dos dados ao longo do tempo.

Diversos algoritmos de agrupamento utilizam como base o FOO onde o processamento se divide em duas fases: a *online* para representar o conjunto de dados processados até o momento, e a *offline* onde é aplicado algoritmos de AM clássico para encontrar a partição dos dados.

Um dos grandes desafios no AFD é a capacidade de processar FD que produzem exemplos de forma rápida, evitando o acúmulo de exemplos. Apesar de existir a técnica de *sampling*, é difícil encontrar um conjunto de dados no FD que seja representativo do conjunto total de exemplos. Para tanto, o uso da computação distribuída é altamente recomendado, permitindo, por exemplo, processar mais de um exemplo de forma simultânea.

3

Computação Distribuída

Em geral, problemas computacionais complexos são divisíveis em partes menores e distintas que podem ser executadas de forma simultânea (DHAENENS; JOURDAN, 2016). Essa característica permite que esses problemas sejam beneficiados pelo uso de múltiplos recursos computacionais. Entretanto, limitações físicas como a velocidade na transmissão dos dados e na miniaturização dos processadores, dificultam e encarecem a criação de computadores mais rápidos e potentes (MARKOV, 2014).

Com os avanços da tecnologia de redes de computadores, tornou-se possível a interligação mais rápida entre os computadores, permitindo que esses compartilhem seus recursos computacionais para a resolução de problemas. Assim, a computação distribuída ou um sistema distribuído foi definido como uma coleção de computadores independentes conectados por meio da rede capazes de se comportar como um sistema único, provendo um serviço ou resolvendo um problema (TANENBAUM; STEEN, 2006).

Devido à disponibilidade de grandes volumes de dados encontrados atualmente, há uma demanda crescente por poder computacional capaz de processá-los de forma rápida e precisa. Isso estimulou a criação de novos modelos de programação e a implementação de plataformas de processamento distribuído, que permitem a execução coordenada de tarefas em diversos computadores espalhados geograficamente de forma paralela. Um desses modelos de programação que se destaca é o modelo *MapReduce*.

3.1 Modelo *MapReduce*

Baseado no paradigma de programação funcional, o modelo *MapReduce* reduz a resolução de problemas pela execução de duas funções (ou fases), denominadas *Map* e *Reduce*. Essencialmente, a fase do *Map* recebe como entrada um par $< chave, valor >$ e produz um conjunto intermediário de respostas de mesmo formato. Logo após, é automaticamente iniciada a fase intermediária, denominada de *shuffle*, que agrega e ordena as tuplas intermediárias que possuem chaves correspondentes. Por fim, essas novas tuplas

são enviadas à função *Reduce*, que gera novas tuplas como resultado final (LIN; DYER, 2010). A Figura 3.1 demonstra o fluxo de execução do modelo, onde as letras representam as chaves, e os números e símbolos representam os valores.

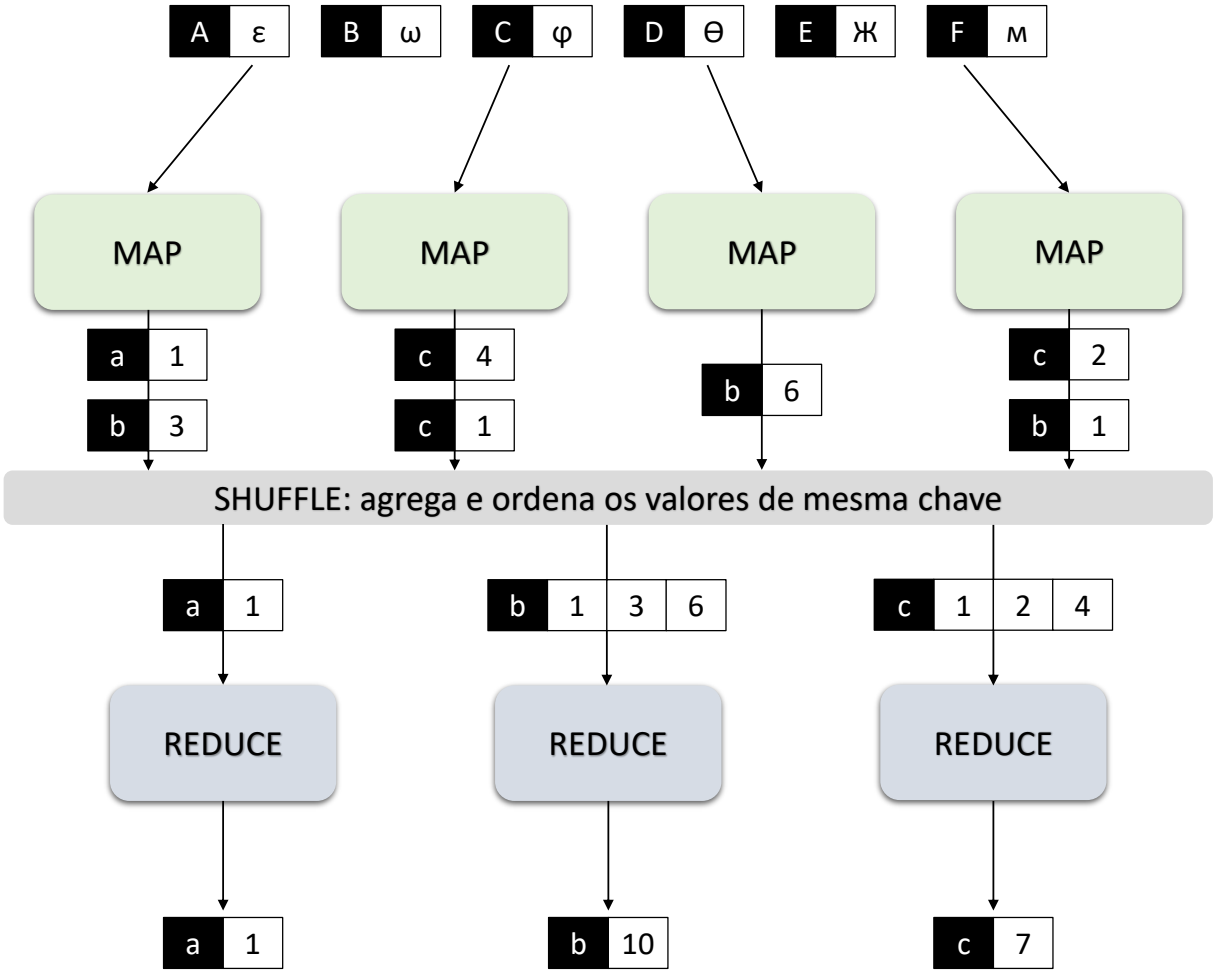


Figura 3.1: Fluxo de execução do *MapReduce*

Além de ser um modelo de programação, o *MapReduce* também é um *framework* de execução que escalona as tarefas para os nós (computadores) em que os dados residem, reduzindo significativamente a comunicação e a transferência de dados na rede. Para tanto, ele considera que a arquitetura é composta de nós que possuem tanto capacidades de processamento quanto armazenamento.

As implementações mais conhecidas do *MapReduce* estão presentes nas plataformas de processamento distribuído Hadoop (WHITE, 2015) e Spark (GULLER, 2015). Essas plataformas se mostraram efetivas no processamento de grandes volumes de dados de forma distribuída, mas consideram que todos os dados já estejam consolidados e disponíveis, finalizando o processamento após a análise de todos os dados. Desta forma, para melhor endereçar as restrições e desafios dos FDs surgiram outras ferramentas mais específicas para esse domínio.

3.2 Processamento Distribuído em FD

Diferentemente do processamento de grandes volumes de dados, em que é possível processar todo o conjunto de dados e chegar a um resultado, o processamento de dados em um FD deve ocorrer de forma contínua. Isso se deve ao fato de um FD poder ser infinito, fazendo com que o processamento nunca acabe. Desta forma, os resultados também devem ser gerados de maneira contínua.

Para conseguir processar um FD de forma contínua e paralela é comumente empregado o padrão *Pipeline*. Nesse padrão, os dados são processados por uma sequência linear de etapas independentes, onde cada etapa processa os dados produzidos pela etapa anterior e envia o resultado para a próxima (F. Dolz et al., 2018). Portanto, considerando um FD $(x_0, ..., x_i)$ e um *Pipeline* com n etapas que computam uma função $f_n : \alpha \rightarrow \beta$, o resultado do processamento de um dado x_j seria $f_n(f_{n-1}(...f_1(x_j)))$.

Essa técnica se mostra eficiente para o processamento paralelo de um FD, uma vez que divide o processamento em etapas independentes. Isso permite que enquanto um dado é processado por uma etapa, outro dado seja processado em outras etapas, como pode ser visto no exemplo da Figura 3.2.

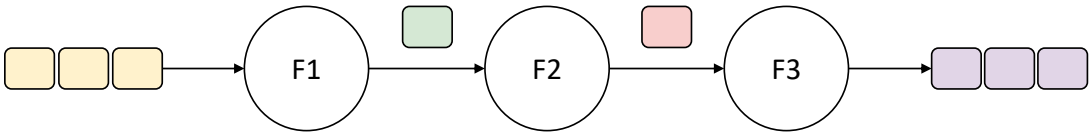


Figura 3.2: Exemplo de processamento no padrão *Pipeline*

Outro padrão utilizado para processamento paralelo em FD é o padrão *Farm*. Nesse padrão, o processamento dos dados é feito de forma paralela em réplicas de uma mesma etapa, como exemplificado pela Figura 3.3. Assim, um dado do FD é processado por uma réplica que estiver disponível no momento. Assume-se ainda que essas réplicas não possuem estado, e possam processar diferentes dados.

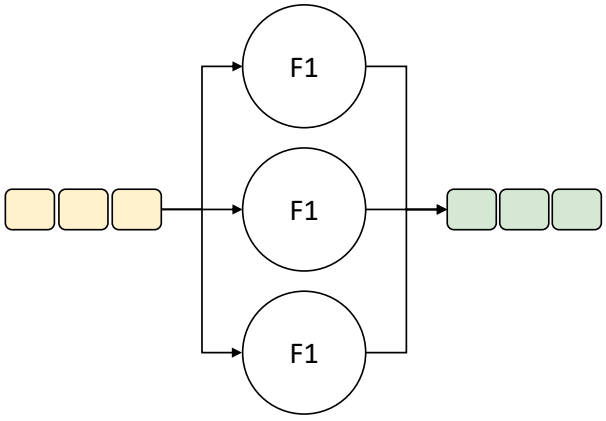


Figura 3.3: Exemplo de processamento no padrão *Farm*

Muitas ferramentas de processamento distribuído em FD combinam esses dois padrões, dividindo o processamento em etapas e permitindo que dados sejam processados de forma paralela em diferentes etapas ou réplicas de uma mesma etapa. Tanto essas réplicas quanto as diferentes etapas podem estar presentes em diversas unidades de processamento, assim processando o FD de forma distribuída.

3.3 Ferramentas de Processamento em FD

Nessa sessão são apresentadas algumas ferramentas de processamento distribuído em FD. Destaca-se que elas têm como base o modelo de programação *MapReduce*, mas realizam o processamento de forma contínua, diferentemente como ocorre no Spark e Hadoop.

3.3.1 Spark Streaming

O Spark Streaming (GULLER, 2015) faz parte de um conjunto de componentes que formam a plataforma Spark. Esse componente modifica a arquitetura de processamento em *batch* para uma arquitetura de *micro-batch*, onde o processamento do FD é tratado como o processamento contínuo de pequenos conjuntos de dados. Inicialmente, os dados são recebidos de diversas fontes e agrupados em pequenos *batches*, executando posteriormente ciclos de *MapReduce* que geram um resultado. Novos *batches* são criados em intervalos regulares, e a cada início do intervalo, os dados que chegam são agrupados e posteriormente processados. Desta forma, o processamento do FD do Spark Streaming é conhecido como *near real-time*, impossibilitando o processamento de um exemplo por vez (MORSHED; RANA; MILRAD, 2016).

3.3.2 Storm

O Storm (LEIBIUSKY; EISBRUCH; SIMONASSI, 2012) é um *framework* que realiza o processamento em tempo real por meio de topologias. Essas topologias são estruturadas como um grafo acíclico direcionado, onde é possível ter uma visão geral dos componentes computacionais e como eles estão conectados. Uma grande vantagem trazida pelo Storm é a possibilidade de programar em diversas linguagens como Java, Python, Scala e Pearl (MORSHED; RANA; MILRAD, 2016).

Inicialmente, um componente chamado *spout* lê os dados do FD e os enviam para outro componente denominado *bolt*. Este componente então realiza um processo de transformação dos dados, podendo tanto persisti-los quanto enviá-los a outro *bolt*. Assim, um programa no Storm pode ser entendido como uma cadeia de componentes *bolt* que realizam transformações dos dados enviados por um *spout*.

3.3.3 Flink

Flink (FRIEDMAN; TZOUMAS, 2016) é uma plataforma tanto para processamento de FDs em tempo real quanto para processamento em *batch* de grandes volumes de dados. O modelo de programação *MapReduce* também forma sua base de modelo de programação, com a adição de funcionalidades como funções de filtro, junção e agregação (como soma, mínimo e máximo), entre outras.

Os dois componentes principais de um programa em Flink são chamados *stream*, que é a abstração do FD, e *transformation*, que é um operador de transformação. A execução do programa é mapeada ao chamado *streaming dataflow*, que se assemelha a um gráfico acíclico direcionado. Ele consiste em fluxos de dados e operadores de transformação. Cada *dataflow* inicia-se com uma ou mais fontes de dados (*sources*) e termina com um ou mais *sinks*, que descartam os dados.

3.3.4 SAMOA

O SAMOA (*Scalable Advanced Massive Online Analysis*) (BIFET; MORALES, 2014) é tanto uma plataforma quanto uma biblioteca específica para AFD de forma distribuída. Como plataforma, o SAMOA permite o desenvolvimento de algoritmos de AM sem a necessidade de lidar com aspectos específicos das ferramentas de processamento distribuído, como as apresentadas anteriormente. Por meio de chamados adaptadores, os algoritmos desenvolvidos no SAMOA são convertidos para serem executados em diversas plataformas, permitindo assim uma maior flexibilidade e reutilização de código.

Já como biblioteca, SAMOA possui implementações de diversos algoritmos de aprendizado em FD como o VHT, para classificação, *CluStream* para agrupamento, e o HAMR, uma versão distribuída de um modelo adaptativo baseado em regras.

Um algoritmo em SAMOA é representado por um grafo direcionado de nós que se comunicam por meio de mensagens. Utilizando a mesma terminologia do Storm, esse grafo é chamado de topologia. Cada nó dessa topologia é chamado de *Processor* e envia mensagens por meio de um fluxo. Um *Processor* é responsável por realizar o processamento dos dados.

3.4 Técnicas de Agrupamento Distribuído

As técnicas de agrupamento distribuído encontradas na literatura se diferenciam pelos seus objetivos. Enquanto a maioria dos trabalhos se concentram no agrupamento de grandes volumes de dados, aqui serão destacados trabalhos que focam no agrupamento em FD baseado em exemplos.

O P-DenStream (LU et al., 2018) é um algoritmo de agrupamento em FD baseado em grade e densidade, implementado no Apache Storm. Ele consiste em dividir uma janela de exemplos entre células em uma grade e sumarizá-las de forma distribuída para formar uma grade denominada global. Assim, células são enviadas a diferentes unidades de processamento que usam o DenStream para gerar um conjunto de vetores de atributos. Logo após, esses vetores são combinados aos outros na grade global. Por fim, os vetores de atributos que tiverem o peso menor que um limiar são considerados antigos e removidos. A grade global é mantida durante todo o processamento do FD, por isso, ela é armazenada em uma base de dados externa.

Em (CâNDIDO et al., 2017) são apresentadas três propostas de agrupamento distribuído em FD utilizando Spark Streaming. A primeira proposta é o algoritmo *Scalable Evolutionary Stream Clustering* (SESC), que sumariza o fluxo de dados em pequenos *chunks* (*micro-batches*), utilizando o *CluStream* em estruturas de sumarização diferentes em paralelo. Então, essas estruturas são posteriormente unidas e agrupadas utilizando o algoritmo *Fast Evolutionary Algorithm for Clustering* (F-EAC). Todo esse processo é feito de uma só vez, sendo que a estrutura de sumarização não é mantida entre o processamento de um *chunk* e outro.

As duas outras propostas, *Incremental Scalable Evolutionary Stream Clustering* (ISESC) e *Incremental Scalable Evolutionary Stream Clustering with Change Detection* (ISESCCD), baseiam-se na ideia de agrupamento incremental, tentando encontrar o número ideal de grupos. Elas utilizam uma janela do tipo *landmark* para remover exemplos antigos, apostando em uma melhor qualidade do agrupamento por utilizar exemplos ao invés de estruturas de sumarização. Por conta disso, os métodos tendem a apresentar pior desempenho por demandarem mais tempo e maior quantidade de memória.

Os dois algoritmos se baseiam na ideia de um modelo de agrupamento global. Inicialmente um agrupamento é gerado por meio do SF-EAC (versão distribuída do F-EAC). Assim, para cada exemplo dos *chunks* que chegam, é calculada a distância entre ele e o centróide do agrupamento gerado. Caso a distância entre todos os exemplos esteja dentro de um limiar, o agrupamento ainda é considerado atual e é apenas atualizado a partir de iterações do algoritmo *k-Means*. Caso contrário, todo agrupamento é realizado novamente pelo SF-EAC.

Em (BACKHOFF; NTOUTSI, 2016) é apresentada uma versão do algoritmo CluStream adaptada para a arquitetura de *micro-batch* do Spark Streaming. Nesta plataforma, quando um *micro-batch* é criado, a fase *online* recebe um conjunto de exemplos que é distribuído entre os nós, que se encarregam de encontrar o MiC mais próximo para cada exemplo recebido. Caso o exemplo esteja dentro do raio de algum MiC, o ponto e o ID do respectivo MiC são enviados para a próxima fase, onde os MiCs são atualizados. Caso contrário, o exemplo é considerado um *outlier* e enviado a fase de processamento de *ou-*

tliers, que realiza o processo de redução da estrutura de sumarização para a criação de novos MiCs. Como nessa fase um conjunto de *outliers* é recebido, o algoritmo deve então verificar se os próximos *outliers* no conjunto não pertencem aos MiCs recém-criados.

Ao fim do processo de sumarização, na fase *offline*, o algoritmo utiliza a versão do *k-means* presente no próprio Spark, apenas modificando a inicialização do algoritmo para adicionar pesos aos exemplos.

O *StreamAP* (HALKIDI; KOUTSOPOULOS, 2011) é um algoritmo que utiliza como base o algoritmo *Affinity propagation* (AP) (FREY; DUECK, 2007). O agrupamento é realizado em *chunks* que chegam para os diversos nós. Cada um desses nós executa o algoritmo AP paralelamente encontrando um conjunto de pontos que seja representativo dos pontos recebidos no *chunk*. Posteriormente, todos os resultados são enviados a um nó central que executa a fase de agrupamento global. Nela, o AP é novamente executado utilizando os conjuntos de pontos recebidos pelos nós, gerando a partição dos dados. Por fim, o algoritmo adiciona o conjunto de pontos encontrados na fase de agrupamento global junto aos próximos *chunks* recebidos pelos nós.

Outras técnicas apresentadas em (BAHMANI et al., 2012), (WANG et al., 2016), (SHAHRIVARI; JALILI, 2016) e (ZHANG; TANGWONGSAN; TIRTHAPURA, 2017) (ZHANG; TANGWONGSAN; TIRTHAPURA, 2017) propõem versões distribuídas do algoritmo *k-means* com foco em desempenho e processamento de grandes volumes de dados.

Atualmente, bibliotecas como Mahout (GUPTA, 2015) para o Hadoop, e MLib (GULLER, 2015) para Spark possuem versões distribuídas do *k-Means* (entre outros algoritmos de AM) para processamento de grandes volumes de dados. Uma versão distribuída do CluStream também está presente no SAMOA e no StreamDM (biblioteca de AM para o Spark Streaming) (BIFET et al., 2016), entretanto, essas versões são incompletas, pois não implementam a fase *offline* do algoritmo (BACKHOFF; NTOUTSI, 2016).

3.5 Técnicas de Agrupamento *Fuzzy* Distribuído

A maioria das propostas de agrupamento *fuzzy* distribuído, como as apresentadas em (SABIT; AL-ANBUKY; GHOLAM-HOSSEINI, 2009) (KALANTARIAN; MASHAYEKHI; ABDOSHAHI, 2014) (ZHOU et al., 2014), focam no processamento de dados distribuídos em redes de sensores, com o objetivo de reduzir a comunicação entre os nós pela geração de protótipos locais.

Em (BHARILL; TIWARI; MALVIYA, 2016) são apresentados o *Scalable Random Sampling with Iterative Optimization Fuzzy C-Means* (SRSIO-FCM), um algoritmo de agrupamento *fuzzy* voltado para processamento de grandes volumes de dados, e o *Scalable*

Literal Fuzzy C-Means (SLFCM), implementação distribuída do *Literal Fuzzy C-Means*. Inicialmente, o SRSIO-FCM divide o conjunto de dados em subconjuntos menores, e aplica o SLFCM no primeiro subconjunto. Posteriormente, os protótipos encontrados são utilizados como protótipos iniciais do SLFCM no processamento do segundo subconjunto. Para os próximos subconjuntos, os protótipos iniciais são calculados somando as matrizes de pertinência do subconjunto $n-1$ e $n-2$, onde n é o subconjunto atual.

O *SLFCM*, assim como versões distribuídas baseadas no *k-means* e FCM (MATHEW; CHANDRAN, 2015) (ZHOU et al., 2015), utiliza uma estratégia de paralelismo similar. Inicialmente o conjunto de exemplos é dividido e os centroides iniciais são gerados. Então, cada nó recebe um subconjunto de exemplos e o conjunto total de centroides.

Os nós calculam paralelamente a distância e pertinência (no caso do SLFCM e FCM) entre os exemplos e os centroides, e enviam os resultados a um nó central. Por fim, o nó central atualiza os centroides e verifica se houve convergência. Caso não houver, os centroides atualizados são retransmitidos para todos os nós, e o processamento se repete.

3.6 Considerações Finais

Este capítulo apresentou conceitos de computação distribuída com foco em ferramentas de processamento distribuído. A utilização dessas ferramentas no AFD permite que algoritmos possam processar mais dados em menor tempo, produzindo resultados de forma mais rápida e acompanhando a velocidade de produção de novos dados. Isso é importante pois torna possível a transformação dos dados em informações úteis em tempo real, o que pode garantir maior tempo e agilidade em um processo de tomada de decisões.

Também foi traçada uma visão geral do estado-da-arte em relação às abordagens distribuídas de agrupamento e agrupamento *fuzzy*, sendo, a maioria delas focada em processamento de grandes volumes de dados. Assim, observa-se que existe uma carência de algoritmos para agrupamento em FD que realizam o processamento de maneira distribuída, principalmente no âmbito de agrupamento *fuzzy*.

Assim sendo, é proposta uma abordagem de computação distribuída para o aprendizado não supervisionado em FD baseado em agrupamento *fuzzy*. O Capítulo 4 traz maiores detalhes dessa proposta.

Parte II

Proposta e Experimentos

4

Abordagem Distribuída para Agrupamento em Fluxo Contínuo de Dados

Os FDs se diferenciam das bases de dados comuns pois os dados chegam continuamente, têm tamanho possivelmente infinito e possuem distribuição não estacionária. Essas características particulares não são atendidas por métodos clássicos de aprendizado, sendo necessários mecanismos diferenciados que delimitem a porção dos dados que será considerada a cada passo e a maneira como essa porção deve ser atualizada.

Dentre os métodos utilizados no AFD, este trabalho tem foco no agrupamento. Nesse método, destacam-se técnicas baseadas no *Framework Online-Offline* (FOO) (AGGARWAL et al., 2003), que divide o processo de aprendizado em duas fases. A primeira fase, denominada fase *online*, sumariza os exemplos continuamente por meio de estruturas de sumarização, criando uma representação dos dados sem a necessidade de armazenar o conjunto de exemplos. Por fim, a fase *offline* realiza o agrupamento da estrutura de sumarização por meio de um algoritmo de agrupamento clássico, resultando em uma potencial partição do FD.

Como no aprendizado de máquina clássico, a integração de conceitos da teoria de conjuntos *fuzzy* no AFD é utilizada com o intuito de tornar o aprendizado mais adaptável e tolerante à imprecisão, pois não se baseia em definições únicas ou binárias de determinadas características. A maioria das abordagens, como SPFC e OFCM apresentadas no Capítulo 2, realizam o processamento considerando que o FD está dividido em *chunks*, o que prejudica a compreensão da evolução do FD já que impede a identificação de alguns tipos de desvio de conceito.

Dentre as abordagens que utilizam como base algoritmos de agrupamento *fuzzy* e o FOO, destacam-se o *FuzzStream* (LOPES; CAMARGO, 2017) e o *decaying FuzzStream* (LOPES, 2016). Ambas se mostraram flexíveis no processo de aprendizagem não supervisionada utilizando conceitos da teoria de conjuntos *fuzzy* para construção e gerenciamento de uma estrutura de sumarização no *Framework Online-Offline*. Entretanto, elas não atendem a um importante requisito no AFD que é a capacidade de produzir

respostas em tempo real.

Como os dados de um FD chegam continuamente e mudanças podem ocorrer em sua distribuição, é importante que algoritmos que realizam o AFD apresentem resultados rápidos. Caso contrário, pode ocorrer o acúmulo e descarte dos dados devido à incapacidade de armazenamento. Assim, informações relevantes podem ser perdidas e o aprendizado tende a ficar cada vez mais defasado, já que os dados processados podem não mais condizer com o estado atual do FD.

Tendo em vista essa problemática e o aumento cada vez maior da quantidade de dados produzidos diariamente, é desejável que técnicas possam se adaptar ao volume de dados, e que ainda assim, gerem resultados rápidos e precisos. Uma das técnicas bastante utilizadas é a técnica de *sampling*, que consiste em encontrar um subconjunto representativo de exemplos dentro de um conjunto de exemplos que chegaram em um determinado período de tempo. Entretanto, nem sempre é possível extrair, por amostragem, um conjunto representativo dos dados originais, pois informações importantes, como desvios de conceito, podem não ser contempladas.

Assim, uma alternativa interessante é a execução dos algoritmos em ambientes de computação distribuída. Por meio de ferramentas de processamento distribuído é possível a execução coordenada de tarefas em diversos computadores de forma paralela, possibilitando processar um maior volume de dados em menor tempo. Deste modo, algoritmos de AFD distribuídos podem se adaptar à velocidade de produção dos dados do FD, produzindo resultados em tempo real sem a necessidade de descartar dados.

Nesse contexto, a proposta desse trabalho é o desenvolvimento de uma abordagem distribuída para aprendizado não supervisionado em FD, baseado no FOO e na teoria de conjuntos *fuzzy*, com o objetivo de ser escalável em relação à produção de novos exemplos, produzindo resultados em tempo real e que melhor representem conceitos imprecisos.

4.1 Abordagem Distribuída para Agrupamento em Fluxo de Dados

A proposta apresentada a seguir envolve dois temas: o algoritmo de agrupamento *fuzzy* que é efetivamente utilizado na abordagem distribuída e a abordagem distribuída propriamente dita.

O algoritmo de agrupamento que foi desenvolvido para a abordagem distribuída é o *d-FuzzStream* (SCHICK; LOPES; CAMARGO, 2018), e tem como inspiração o algoritmo *FuzzStream*. Esse algoritmo permitiu melhorar o processo de sumarização de dados, apresentando menor complexidade computacional e retendo maior quantidade de informações em relação ao algoritmo em que se baseia.

Já a abordagem distribuída consiste em duas diferentes estratégias para paraleli-

zação do processamento e que são únicas para cada fase do FOO. Isso se dá por conta das fases do FOO serem diferentes e possuírem objetivos diferentes. A fase *online* processa exemplos um-a-um sumarizando-os em uma estrutura de sumarização, e caracteriza-se pelo processamento contínuo e intermitente de dados em forma de fluxo. Já a fase *offline* tem como objetivo agrupar uma estrutura de sumarização, que é um pequeno conjunto de dados, caracterizando-se assim, pelo processamento em *batch*.

Desta forma, a abordagem distribuída desenvolvida neste trabalho utiliza duas diferentes estratégias que se adequam à proposta das duas fases do FOO. A fase *online* sumariza o FD distribuidamente utilizando diferentes estruturas independentes, e a fase *offline* agrupa a estrutura de sumarização utilizando diferentes algoritmos com diferentes parâmetros. Como o resultado da fase *online* são diversas estruturas de sumarização, uma nova fase denominada intermediária foi desenvolvida para transformar essas diversas estruturas, denominadas locais, em uma única estrutura, a sumarização global.

Desta forma, as duas estratégias desenvolvidas nessa proposta de abordagem distribuída são a estratégia de sumarização local e global na fase *online* e intermediária, e a estratégia de agrupamento em paralelo da estrutura de sumarização, utilizando diferentes algoritmos com diferentes parâmetros na fase *offline*. Assim, nas sessões a seguir serão apresentados e discutidos os seguintes tópicos:

- ***d-FuzzStream*:** evolução do *FuzzStream* que proporciona um melhor processo de sumarização dos dados.
- **Estratégia de sumarização local e global na fase *online* e fase *intermediária*:** viabiliza o processamento simultâneo de diversos exemplos em estruturas de sumarização diferentes que são posteriormente condensadas.
- **Estratégia de agrupamento paralelo da estrutura de sumarização na fase *offline*:** replica o conjunto de exemplos e agrupa as réplicas utilizando diversos algoritmos e números de grupos de maneira distribuída.

4.2 *d-FuzzStream*

O *d-FuzzStream* (SCHICK; LOPES; CAMARGO, 2018) é um algoritmo de agrupamento desenvolvido com base no *FuzzStream* (LOPES, 2016)(LOPES; CAMARGO, 2017). Como é baseado no FOO, o algoritmo divide o processo de agrupamento em duas fases: sumarização dos dados utilizando FMiCs e agrupamento da estrutura de sumarização.

Entretanto, o *d-FuzzStream* introduz o uso dos conceitos de similaridade e dispersão *fuzzy*, melhorando o processo de sumarização e diminuindo sua complexidade em

relação ao *FuzzStream*. As sessões a seguir apresentam esses dois conceitos, assim como todo o funcionamento do algoritmo *d-FuzzStream*. O resultado detalhado da comparação entre os dois algoritmos é exposto no Capítulo 6.

4.2.1 Dispersão *fuzzy*

Na etapa de sumarização dos dados, os exemplos são sumarizados por meio de uma estrutura de sumarização, os FMiCs. Um grupo inicial de FMiCs é criado a partir de exemplos tal que os próximos exemplos são atribuídos aos FMiCs caso sejam suficientemente similares, ou se tornam novos FMiCs caso sejam considerados *outliers*. A condição de similaridade, entre um FMiC e um exemplo, empregada pelo algoritmo é baseada no conceito de dispersão *fuzzy*.

A dispersão *fuzzy* é uma medida proposta em (XIONG; CHAN; TAN, 2004) para representar o raio de grupo *fuzzy*. Como a fase *online* do *d-FuzzStream* utiliza FMiCs ao invés de grupos, esse conceito foi generalizado para representar o raio de um FMiC.

Assim sendo, seja x_j um exemplo do FD $(x_1, ..., x_n)$, $(F_1, ..., F_k)$ um conjunto de k FMiCs e N_i o número de exemplos atribuídos a um FMiC F_i. A dispersão fuzzy dp_i para o FMiC F_i é calculada como mostrado na Equação 4.1.

$$dp_i = \sqrt{\frac{\sum_{x_j \in F_i} \mu_j^m \left\| x_j - f_i \right\|^2}{N_i}} \tag{4.1}$$

Onde μ_j^m é o valor da associação do exemplo j para o cluster F_i, f_i é o protótipo de F_i e $\|x_j - f_i\|$ representa a distância entre o exemplo x_j e o protótipo f_i.

Com esse conceito de dispersão *fuzzy*, é possível determinar se um exemplo pertence ou não a um determinado FMiC, ou se ele é um *outlier*. Para tanto, se a distância entre um exemplo x_j e o centróide de um FMiC for menor que a sua dispersão, ele é absorvido pela estrutura de sumarização. Caso contrário, este exemplo é considerado um *outlier* e um novo FMiC deve é criado.

Por fim, a dispersão *fuzzy* também é empregada para calcular uma matriz de similaridade *fuzzy* entre todos os FMiCs da estrutura de sumarização, apresentada na sessão a seguir.

4.2.2 Mescla baseada em similaridade *fuzzy*

A mescla é uma operação que ocorre durante a etapa de sumarização dos dados, e que consiste em unir dois FMiCs baseados em uma métrica de similaridade entre eles.

A matriz de similaridade *fuzzy* FR, também introduzida em (XIONG; CHAN; TAN, 2004), foi generalizada para representar a similaridade entre os FMiCs. Cada célula

da matriz $FR = \{FR_{ij}, (i,j) = 1, 2, ..., k\}$ reflete a razão da soma da dispersão *fuzzy* de dois FMiCs para a distância entre seus protótipos (Equação 4.2).

$$FR_{ij} = \frac{dp_i + dp_j}{\|f_i - f_j\|} \tag{4.2}$$

O *Critério de mescla baseada em similaridade* (XIONG; CHAN; TAN, 2004) usa a matriz FR e um limiar τ (tau) para identificar grupos sobrepostos. Portanto, também foi utilizado para definir se dois FMiCs são suficientemente similares para serem mesclados. Se o valor FR_{ij} for maior que τ, os dois FMiCs poderão ser mesclados, ou caso contrário, os dois FMiCs serão considerados completamente separados e não semelhantes o suficiente para serem mesclados.

A Figura 4.1 mostra uma estrutura de sumarização, com seis FMiCs (A, B, C, D, E e F) representados por seus protótipos (cruzes vermelhas) e sua dispersão *fuzzy* (contornos azuis). Quando não há interseção entre dois contornos de dispersão ($FR > 1$), os dois FMiCs são considerados separados um do outro.

Como mostrado em Figura 4.1, o grupo C é completamente separado dos demais. Se houver uma interseção entre os contornos de dispersão de dois grupos ($FR < 1$), os dois FMiCs são considerados sobrepostos (grupos D e F se sobrepõem ao E). No entanto, se dois contornos de dispersão estiverem tangentes ($FR = 1$), pode-se considerar que esses dois FMiCs (A e B) estão separados.

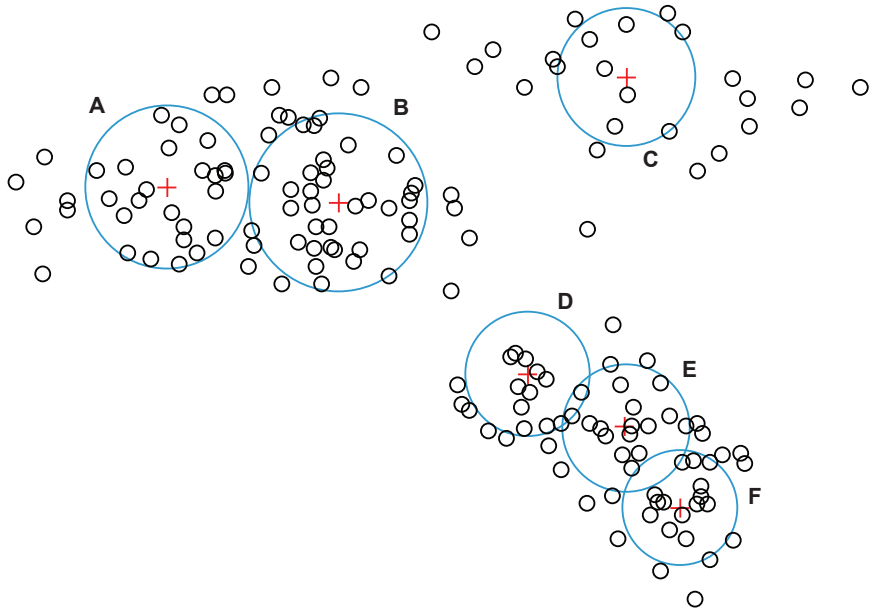


Figura 4.1: Exemplo de FMiCs e seus contornos de dispersão

O limiar τ é um parâmetro do algoritmo utilizado para definir o quão sobrepostos os FMiCs devem estar para que sejam mesclados. Este parâmetro é definido pelo usuário

no início do processo e não sofre alteração durante o processamento.

Para $\tau < 1$, o *d-FuzzStream* considera os FMiCs não sobrepostos, $\tau = 1$ considera quaisquer FMiCs sobrepostos e $\tau > 1$ faz com que o algoritmo apenas considere FMiCs com maior sobreposição. Vale ressaltar que quanto maior o valor de τ, mais sobrepostos os FMiCs necessitam ser para que uma mescla possa ocorrer. Por outro lado, quanto menor o valor de τ, menos semelhantes os FMiCs devem ser para que ocorra uma mescla.

4.2.3 Fase *Online*: Sumarização dos dados

Como mencionado anteriormente, o *d-FuzzStream* sumariza os dados utilizando uma extensão de vetor de atributos, denomidada FMiC. Entretanto, para que fosse possível o cálculo de dispersão *fuzzy*, a estrutura do FMiC foi modificada em relação à proposta original. A Tabela 4.1 apresenta todos os componentes presentes na nova estrutura.

Tabela 4.1: Estrutura de um FMiC no *d-FuzzStream*

Componente	**Definição**
$\overline{CF^x}$	soma linear dos exemplos ponderados pelos valores de pertinência
SSD	soma quadrática das distâncias entre os exemplos e o protótipo do FMiC ponderados pelos valores de pertinência
N	número de exemplos atribuídos ao FMiC
t	*timestamp* do último exemplo que entrou no raio do FMiC
M	soma dos valores de pertinência dos exemplos atribuídos ao FMiC

Nessa nova estrutura, são armazenadas duas informações diferentes, SSD e N, que são necessárias para o cálculo da dispersão *fuzzy*. Como SSD representa a soma das distâncias ao quadrado ponderada pela pertinência, o cálculo de dispersão apresentado na Equação 4.1, pode ser realizado conforme mostrado na Equação 4.3.

$$dp_i = \sqrt{\frac{SSD_i}{N_i}} \tag{4.3}$$

A manutenção para um conjunto desse novo FMiC é apresentada no Algoritmo 4.1, onde as entradas necessárias são o Fluxo de Dados (FD), o parâmetro de *fuzzificação* (m), os números mínimo e máximo de FMiCs permitidos na estrutura ($minFMiC, maxFMiC$) e o limiar de mescla (τ).

A estrutura de sumarização (F) é inicializada vazia. Para cada exemplo em FD, o algoritmo primeiro verifica se a estrutura tem pelo menos $minMiC$ FMiCs, caso contrário o exemplo necessariamente se torna um novo FMiC (linha 5). Se a estrutura de sumarização já contiver o número mínimo de FMiCs, o algoritmo avalia se o exemplo é um *outlier* ou não. Para tanto, as distâncias euclidianas entre o exemplo e todos os FMiCs são medidas (linha 8), e o raio para cada FMiC é calculado (linhas 11-17). Se um FMiC

Algoritmo 4.1: Manutenção dos FMiCs no *d-FuzzStream*

```
   Entrada: FD, minFMiC, maxFMiC, τ, m
   Saída: F
1  início
2      F = ∅;
3      para cada exemplo e em FD faça
4          se tamanho(F) < minFMiC então
5              adicionaNovoFMiC(e);
6          fim
7          senão
8              D = distancias(e, F);
9              isOutlier = VERDADEIRO;
10             para cada FMiC f em F faça
11                 raio = 0;
12                 se f.N == 1 então
13                     #Caso Especial: FMiC recém criado;
14                     raio = distanciaMinima(f, F);
15                 fim
16                 senão
17                     raio = dispersaoFuzzy(f);
18                 fim
19                 se D[f] <= raio então
20                     isOutlier = FALSO;
21                     f.timestamp = e.timestamp;
22                 fim
23             fim
24             se isOutlier então
25                 se tamanho(F) = maxFMiC então
26                     removeMaisAntigo(F);
27                 fim
28                 adicionaNovoFMiC(e);
29             fim
30             senão
31                 U = pertinencias(e, F, m);
32                 atualizaEstrutura(F, D, U);
33             fim
34             mescla(F, τ);
35         fim
36     fim
37 fim
```

tiver mais de um exemplo atribuído a ele ($N > 1$), o raio se tornará a dispersão *fuzzy* do FMiC, calculada usando Equação 4.3 (linha 17); Senão, $N = 1$, não é possível realizar o cálculo da dispersão *fuzzy*, logo, o algoritmo entra no caso especial (linha 14), onde o raio se torna a menor distância entre este protótipo do FMiC e todos os outros protótipos dos FMiCs.

Quando um exemplo está dentro do raio de algum FMiC, esse exemplo não é mais considerado um *outlier* e o FMiC em questão tem o seu *timestamp* atualizado. A atualização do *timestamp* só ocorre para os FMiCs nos quais o exemplo está dentro do raio, evitando com que todos os FMiCs tenham o mesmo *timestamp*.

Se o exemplo não for um *outlier*, a matriz de pertinência entre o exemplo e todos os FMiCs é calculada (linha 31), assim como ocorre no FCM, e todos os FMiCs serão atualizados. Caso contrário, um novo FMiC deve ser criado e adicionado à estrutura de sumarização. Nesse caso, se a estrutura estiver cheia, o FMiC mais antigo será substituído pelo novo.

Finalmente, a etapa de mescla é iniciada pela função `mescla()`. Uma matriz de similaridade *fuzzy* entre todos os FMiCs é calculada e pares de FMiCs com a maior semelhança *fuzzy* entre aqueles com semelhança maior que o parâmetro τ são mesclados.

Diferentemente de como ocorre no *FuzzStream*, no *d-FuzzStream* não é necessário calcular a matriz de pertinência para identificar quando um exemplo é um *outlier*. Além disso, o algoritmo elimina completamente a necessidade de calcular a matriz de pertinência entre todos os FMiCs presentes na estrutura de sumarização. Isso reduz significativamente a complexidade do algoritmo e, por consequência, o tempo de processamento.

Assim, os conceitos de dispersão e similaridade *fuzzy* são apenas utilizados na fase *online* para identificação de *outliers* e possíveis mesclas de FMiCs. Esses conceitos permitiram com que o *d-FuzzStream* melhorasse o processo de sumarização em relação ao *FuzzStream*. Da mesma forma, esses conceitos removeram a necessidade de realizar cálculos onerosos de matrizes de pertinência.

4.2.4 Fase *Offline*: Agrupamento da estrutura de sumarização

A etapa *offline* é semelhante àquela do algoritmo *FuzzStream*. Primeiro, o conjunto de protótipos dos FMiCs é transformado em um conjunto de exemplos ponderados e posteriormente agrupado usando o algoritmo *Weighted Fuzzy C-Means* (WFCM) para gerar uma partição *fuzzy*.

Para cada FMiC na estrutura de sumarização, um protótipo é obtido dividindo $\overline{CF^x}$ por M e seu peso é definido como o valor de M. Os protótipos iniciais dos grupos da etapa *offline* podem ser definidos tanto de forma aleatória, quanto a partir dos exemplos com maiores pesos.

O Algoritmo 4.2 apresenta o funcionamento do WFCM, onde as entradas são a base de dados (E), o número de grupos (k), o fator de fuzzificação (m), o limiar de parada (ξ), o peso dos exemplos (w) e o número máximo de iterações ($maxIter$).

Algoritmo 4.2: *Weighted Fuzzy C-Means* (WFCM) (BEZDEK, 1981)

Entrada: E, k, m, ξ, w, $maxIter$
Saída: U, C

```
1  início
2      U = geraMatrizPertinênciaAleatória();
3      C = geraCentróidesIniciais(E, U);
4      ϵ = ∞;
5      iterações = 1;
6      enquanto ϵ > ξ ou iterações > maxIter faça
7          U = atualizarMatrizPertinência(E, C);
8          C' = C;
9          C = atualizarCentróides(E, U);
10         ϵ = max_{1≤i≤k}{‖c_i − c'_i‖²};
11         iterações = iterações + 1;
12     fim
13 fim
```

A inicialização do algoritmo começa pela geração de uma matriz de pertinência aleatória para os k grupos, que é posteriormente utilizada para calcular os centróides iniciais dos grupos. Em seguida, o processo de agrupamento se repete enquanto a diferença entre a posição dos centróides da iteração atual e da anterior for maior que o limiar ξ, e o número de iterações for menor que $maxIter$ (linhas 7-11).

O processo de agrupamento consiste em atualizar a matrix de pertinência (linha 7), exatamente como no FCM, e atualizar os centróides (linha 9). Essa atualização se diferencia do FCM pela utilização do peso dos exemplos no cálculo dos centróides. A Equação 4.4 demonstra como é realizado este cálculo, onde c_i é o centróide do grupo i, w_j é o peso do exemplo j, u_{ij}^m é a pertinência do exemplo j para o grupo i, e e_j é o próprio exemplo j.

$$c_i = \frac{\sum_{j=1}^{n} w_j u_{ij}^m e_j}{\sum_{j=1}^{n} w_j u_{ij}^m} \tag{4.4}$$

Logo após a atualização dos centróides, é feita uma comparação entre a posição dos centróides antigos e a posição dos centróides novos (linha 10) para encontrar a maior diferença. Por fim, o resultado do algoritmo é a matriz de pertinência (U) e o conjunto de centróides (C).

4.3 Estratégia de Sumarização Local e Global

O processo de manutenção dos FMiCs na fase *online* adiciona alguns desafios para o processamento distribuído. Como cada exemplo possui graus de pertinência para todos os FMiCs, é necessário realizar operações de leitura e escrita em todos os FMiCs presentes na

estrutura de sumarização. Por consequência, sumarizar diversos pontos simultaneamente em uma única estrutura de sumarização exigiria acesso simultâneo a uma versão atualizada dessa estrutura, o que demandaria um processo de sincronização custoso. Por exemplo, o processamento de novos exemplos teria de ser interrompido toda vez que houvesse remoção e mescla de FMiCs, algo que ocorre de maneira frequente.

A partir dessa dificuldade, é proposta uma estratégia de sumarização distribuída em estruturas de sumarização distintas, que são posteriormente mescladas em uma única estrutura em uma nova fase denominada fase intermediária.

Para essa estratégia funcionar, o processamento precisa manter controle da quantidade de exemplos que serão processados antes que a fase intermediária seja iniciada por meio de janelas temporais. Assim, os dados provenientes do fluxo de dados são armazenados até atingir a janela definida, e são então divididos em n conjuntos de dados menores, onde n representa o paralelismo da tarefa de sumarização local (fase *online*).

Nesta etapa, esses pequenos conjuntos de dados são sumarizados como no *d-FuzzStream* em uma estrutura de sumarização armazenada localmente. Quando o processamento desse conjunto termina, toda a estrutura de sumarização é enviada para a fase de sumarização global. Vale ressaltar que a estrutura de sumarização armazenada localmente não é removida após o processamento de um conjunto de dados, sendo utilizada para sumarizar os próximos conjuntos.

A etapa de sumarização global também é precedida de uma janela temporal. Isso permite que a etapa de sumarização global apenas se inicie quando todas as estruturas de sumarização locais estiverem disponíveis para processamento. Nesta fase *intermediária* é executada uma função de mescla que recebe de duas em duas as estruturas de sumarização, transformando-as em uma única. Essa função é uma modificação da função de mescla presente no *d-FuzzStream*, comparando exemplos similares entre as duas estruturas. Assim, FMiCs que são similares entre as duas estruturas são mesclados e todos os outros são simplesmente agregados em uma única estrutura.

As duas fases, *online* e intermediária, são análogas a um ciclo de *MapReduce*, e ao final geram uma observação do fluxo de dados. Essa observação é então salva para ser posteriormente utilizada para o agrupamento na fase *offline*. A Figura 4.2 demonstra um esquema do funcionamento da estratégia de sumarização local e global no processamento de seis eventos ($e1$ a $e6$).

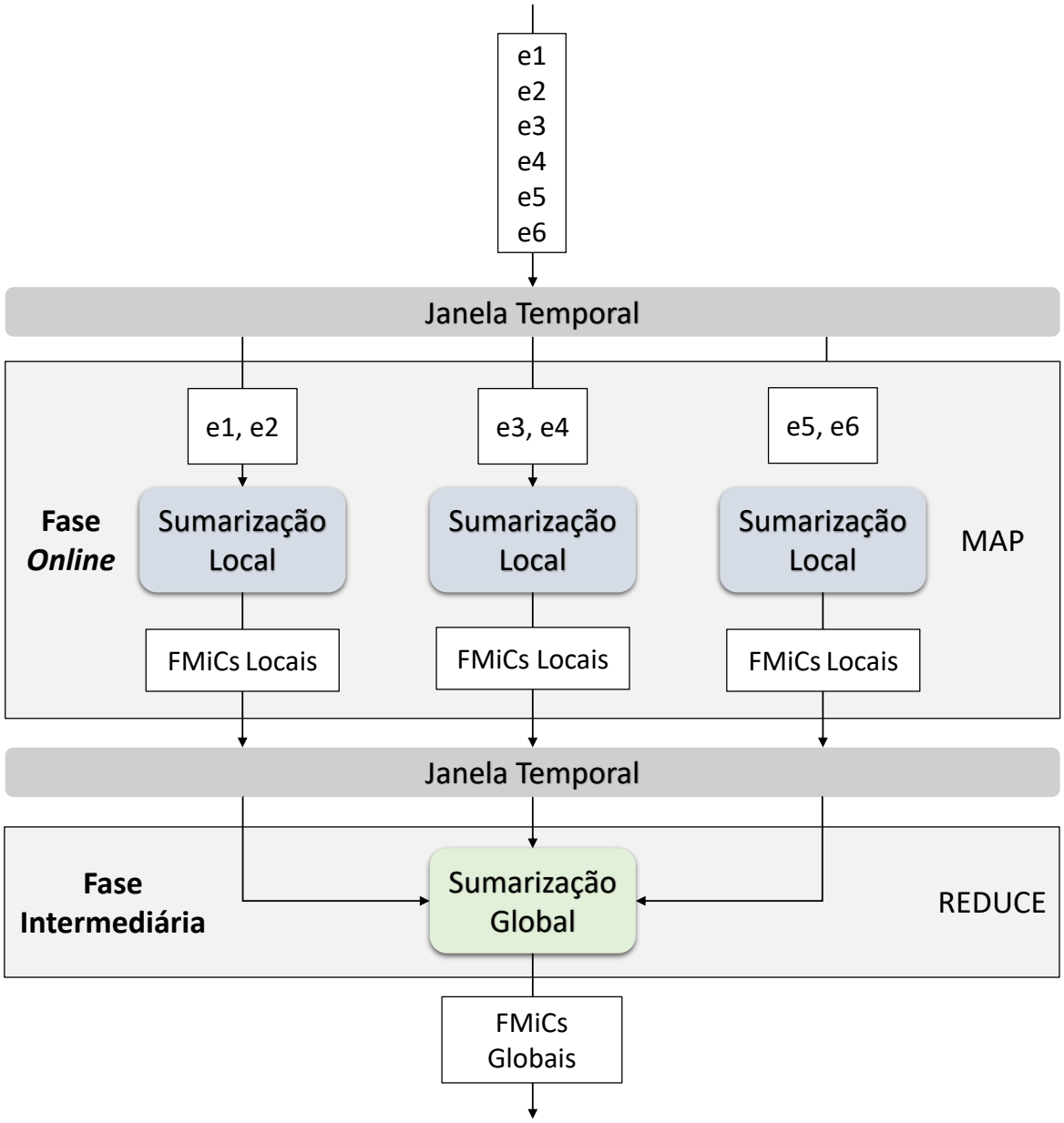


Figura 4.2: Estratégia de sumarização local e global

4.4 Estratégia de Agrupamento Distribuído

Devido à natureza não-estacionária dos FDs, pode ser difícil estimar a quantidade de grupos ideal para cada sumarização de um FD gerada após a fase *intermediária*.

Desta forma, a estratégia desenvolvida na fase *offline* foi a de agrupar distribuidamente a estrutura de sumarização utilizando diferentes algoritmos de agrupamento e com diferentes números de grupos. Assim, esta estratégia permite gerar o resultado de diversos agrupamentos simultaneamente.

O processamento da fase *offline* é feito em *batch*, uma vez que o agrupamento é realizado em cima da estrutura de sumarização. Assim, inicialmente a estrutura gerada pela fase *intermediária* é replicada para diversas unidades de processamento que realizaram o

processo de agrupamento, junto com o algoritmo de agrupamento e o número de grupos. A figura Figura 4.3 apresenta um esquema do agrupamento da fase *offline* utilizando os algoritmos *Weighted Fuzzy C-Means* (WFCM) e *K-Means* para dois números de grupos diferentes.

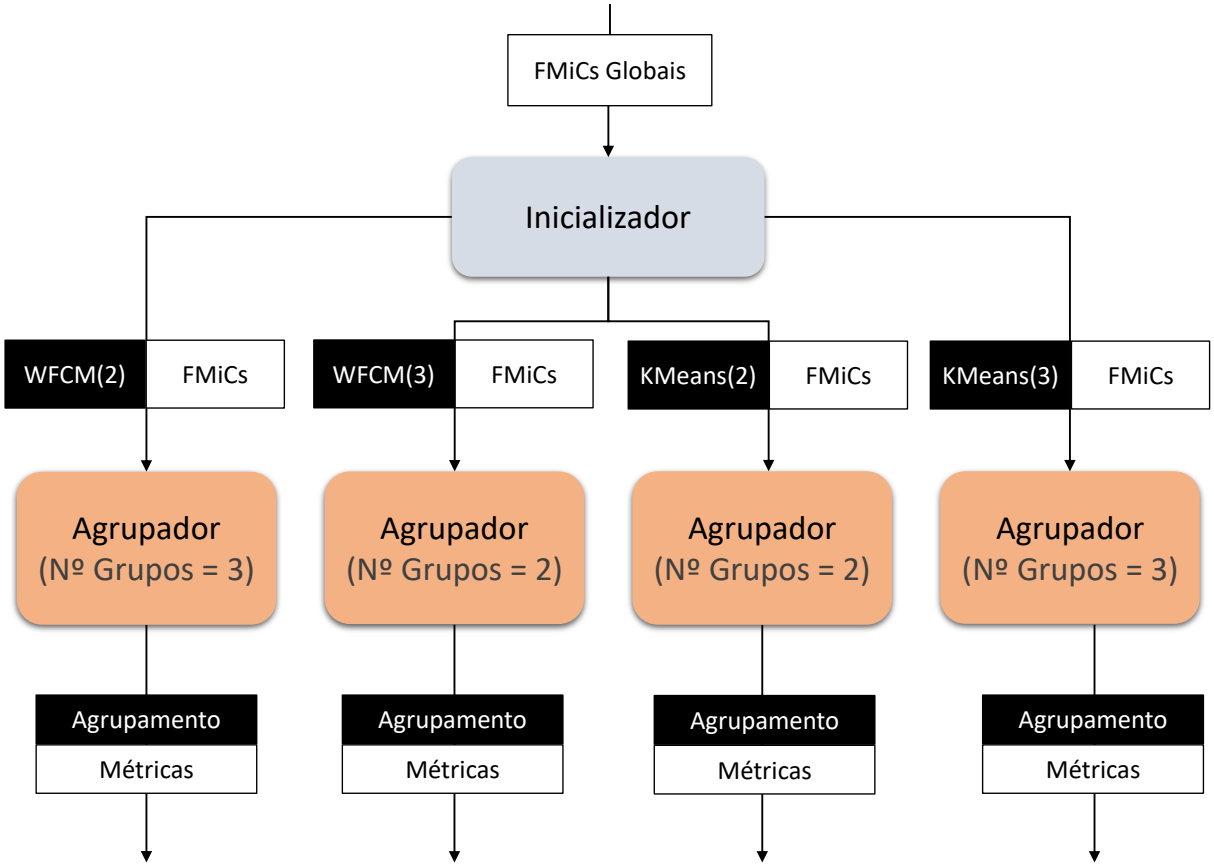


Figura 4.3: Estratégia de agrupamento paralelo

O **Inicializador** inicia os algoritmos de agrupamento com o número de grupos dentro de uma extensão escolhida pelo usuário. Logo após, o conjunto de FMiCs é replicado, e cada **Agrupador** recebe um dado no formato chave e valor, onde a chave é o algoritmo de agrupamento e o valor é o conjunto de FMiCs.

Logo após, os **Agrupadores** realizam n agrupamentos, onde n é um parâmetro passado pelo usuário. Os protótipos iniciais desses n agrupamentos são gerados de forma aleatória, o que pode gerar diferentes agrupamentos. Por fim, métricas internas e externas de avaliação de agrupamento são calculadas para os n resultados produzidos. Essas métricas serão descritas com mais detalhes no Capítulo 5.

Assim, o resultado final é composto do resultado de um dos n agrupamentos e um objeto informativo **Métricas**, que possui a média e o desvio padrão de cada uma das métricas. Este único agrupamento que compõe o resultado final foi escolhido como o agrupamento com maior valor de silhueta (métrica interna) dentre os n gerados.

4.5 Considerações Finais

Este capítulo apresentou uma abordagem que alia a computação distribuída e o aprendizado não supervisionado em FD baseado em agrupamento, com o objetivo de prover escalabilidade e respostas em tempo real que melhor representem conceitos imprecisos. Assim, essa abordagem permite agrupar FDs que produzam grande quantidade de dados em curto período de tempo de maneira rápida, produzindo resultados em tempo real sem a necessidade de utilizar técnicas como de amostragem, que descartam dados em um *chunk* .

As principais contribuições deste trabalho são: o algoritmo *d-FuzzStream*, evolução do *FuzzStream* que melhora o processo de sumarização; e as estratégias para sumarização dos dados e agrupamento de forma distribuída. Estas permitem o processamento mais rápido dos dados do FD, e geram resultados de diversos agrupamentos de uma única vez. A proposta da abordagem distribuída utilizando o *d-FuzzStream* configura o algoritmo *Distributed d-FuzzStream*.

O Capítulo 5 apresenta a metodologia para validação da abordagem como um todo, detalhando as ferramentas utilizadas, assim como os conjuntos de exemplos e métricas de avaliação aplicadas. Os resultados e a metodologia utilizada para avaliar o *d-FuzzStream* são expostos no Capítulo 6.

5

Metodologia para Validação

A avaliação da proposta apresentada no Capítulo 4 foi realizada pela execução e avaliação de uma série de experimentos. Esses experimentos foram conduzidos utilizando a ferramenta Apache Flink sobre conjuntos de exemplos com comportamentos específicos.

Este capítulo apresenta as ferramentas que foram utilizadas para a execução dos experimentos, assim como descreve os conjuntos de exemplos utilizados, os algoritmos empregados, a configuração de parâmetros e as métricas aplicadas para validação da proposta.

5.1 Ferramentas de Software

A ferramenta escolhida para o desenvolvimento e avaliação da proposta foi a plataforma *open source* Apache Flink. O modelo de programação da ferramenta tem como base o *MapReduce*, com a adição de funcionalidades como funções de filtro, junção e agregação (como soma, mínimo e máximo), entre outras (FRIEDMAN; TZOUMAS, 2016). A plataforma é escrita nas linguagens de programação Java e Scala e processa o FD de forma distribuída em *pipeline*.

Os dois componentes principais de um programa em Flink são chamados *stream*, que é a abstração do FD, e *transformation*, que é um operador de transformação. O Flink também possui suporte ao conceito de janelas que são diferenciadas como *tumbling windows*, *sliding windows*, e *session windows* (tempo de inatividade).

A execução do programa é mapeada ao chamado *streaming dataflow*, que se assemelha a um gráfico acíclico direcionado. Ele é composto de operadores de leitura de uma fonte de dados (*source*), transformação (*transformation*) e descarte dos dados (*sink*) (Apache Flink, 2018). Na Figura 5.1[1] pode ser vista a correlação entre um código, escrito em Java, que conta as palavras de um FD e uma esquematização do *streaming dataflow*.

[1] Adaptado de https://ci.apache.org/projects/flink/flink-docs-release-1.6/concepts/programming-model.html

```
DataStream<String> linhas = env.readTextFile("ARQUIVO_DE_TEXTO.txt");                    Source

DataStream<Tuple2<String, Integer>> palavras = linhas.flatMap(new Tokenizer());          Transformation

DataStream<Tuple2<String, Integer>> contagem = words
                                                .keyBy(0)
                                                .timeWindow(Time.seconds(10));           Transformation
                                                .sum(1);

contagem.addSink(new PrintSinkFunction());                                               Sink
```

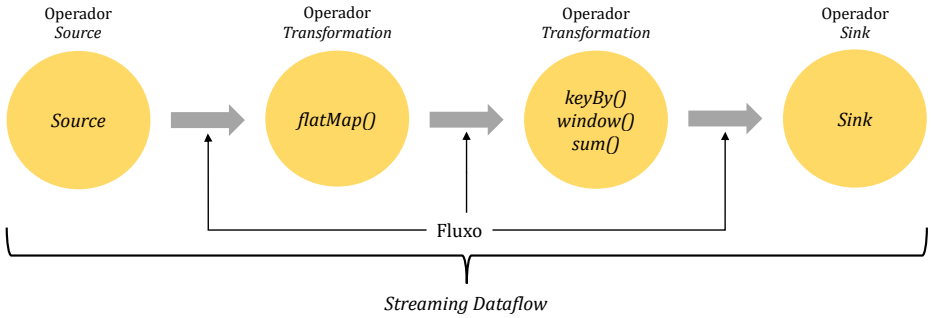


Figura 5.1: Correlação entre um código em Java e o *Streaming Dataflow*

Nesse exemplo, o código se inicia pela criação de um FD a partir de uma fonte de dados. Por meio de operador de leitura é gerado um FD de linhas de um arquivo de texto. Todas as palavras de cada linha são então separadas por um operador de transformação do tipo *flatMap*, formando um novo FD no formato chave-valor, onde a chave passa a ser a palavra e o valor é 1.

A função `keyBy()` agrupa esse novo fluxo a partir de palavras que sejam iguais, ou seja, agrupa os dados pela chave. Após uma janela de tempo de dez segundos (`timeWindow`) é aplicada uma função de agregação (`sum`) que soma a quantidade de vezes que uma palavra foi encontrada. Um novo FD é gerado agora com o par chave-valor `<P, V>`, onde P é a palavra e V é o número de vezes que esta apareceu nos últimos 10 segundos. Esse novo FD é por fim enviado a uma função de *sink*, que mostra os resultados.

Devido à natureza paralela do Flink, cada operador pode ser divido em subtarefas (*subtasks*) que são independentes uma das outras e executadas em diferentes unidades de processamento. Assim, o FD é particionado a partir da quantidade de subtarefas do operador a ser executado, caracterizando o nível de paralelismo. Como pode ser visto na Figura 5.2. [2], nota-se que diferentes operadores podem ter diferentes níveis de paralelismo.

O envio dos dados do FD entre duas subtarefas pode ser feito de diferentes formas (Apache Flink, 2018):

[2] Adaptado de https://ci.apache.org/projects/flink/flink-docs-release-1.6/concepts/programming-model.html

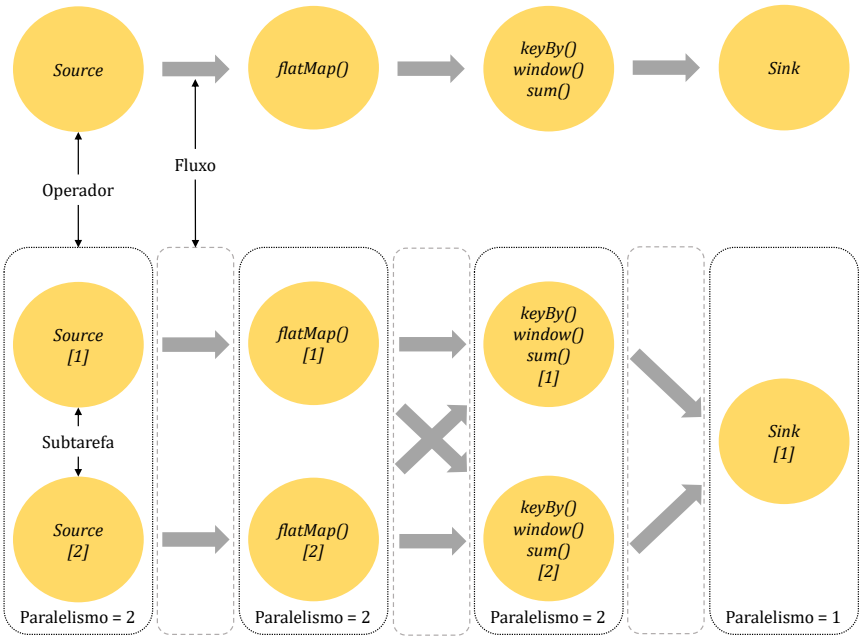


Figura 5.2: Paralelismo de Subtarefas

- **Fluxos um-a-um:** o fluxo é particionado e enviado mantendo a ordem em que os dados foram produzidos pela subtarefa anterior.
- **Fluxos redistribuídos:** o fluxo é particionado dependendo da transformação a ser realizada. Uma transformação do tipo *KeyBy()*, por exemplo, reparte o fluxo a partir dos dados de mesma chave.

Apesar de ser construído para o processamento de FDs, o Flink também executa processamento em *batch*. Isso é feito pela ferramenta como um caso especial de FD, onde o conjunto de dados é internamente tratado com um fluxo que possui um número finito de elementos.

Um outro aspecto relevante do Apache Flink é a sua capacidade de processar FDs mantendo estados (*Stateful Processing*) (HUESKE; KALAVRI, 2018). Por meio de *checkpoints* que são criados automaticamente, a ferramenta consegue reiniciar o estado do fluxo de dados de forma automática, garantindo que todos os dados serão processados. Outra importante funcionalidade, que distingue a ferramenta de outras, é a possibilidade de criar *savepoints*. Esses *savepoints* permitem salvar um estado (por exemplo uma estrutura de dados) de diferentes formas, como, por exemplo, em uma base de dados externa ou um arquivo. Esses estados podem ser consultados e/ou utilizados para reiniciar a aplicação no estado em que ela se encontrava.

Esses dois aspectos mencionados anteriormente, possibilidade de processamento em *batch* e processamento em FD mantendo estados, fizeram com que o Apache Flink fosse escolhido em detrimento de outras ferramentas de processamento distribuído.

Na estratégia de sumarização local e glocal, as estruturas de sumarização armazenadas localmente representam estados que devem se manter ao longo do processamento do fluxo. Já a capacidade de processamento em *batch* permite o desenvolvimento da estratégia de agrupamento distribuído que deve ser processada em *batch*.

Por fim, para a geração das bases de dados sintéticas e visualização dos resultados foi utilizada a plataforma R. Essa plataforma gratuita, engloba a linguagem R e um ambiente multiplataforma que possui uma variedade de técnicas para manipulação, análise e visualização de dados, além de ser extensível por meio de pacotes (R Core Team, 2018).

Os pacotes R específicos utilizados para a geração das bases de dados foram:

`stream` é um *framework* para representação e processamento de FDs a fim de facilitar o desenvolvimento, testes e a comparação entre algoritmos de aprendizado em FD na linguagem R, além de permitir a integração de técnicas modeladas em linguagens para as quais haja uma interface R, como C/C++, Java, Python. (HAHSLER; BOLANOS; FORREST, 2018a).

`streamMOA` é uma extensão do pacote `stream` que faz a interface entre algoritmos de agrupamento já disponíveis na ferramenta *Massive Online Analysis* MOA. (HAHSLER; BOLANOS; FORREST, 2018b).

`BBmisc` é um pacote que possui diversas funções de apoio, incluindo operações sobre matrizes e conjunto de dados, como por exemplo, normalização.

5.2 Ambiente de execução

O Apache Flink foi executado em ambiente local, sendo assim, apenas um nó foi utilizado. Este nó possuia um *hardware* com uma CPU de 4.2GHz com quatro núcleos físicos e oito virtuais, 16GB de memória RAM e em ambiente Linux.

Os conjuntos de dados foram lidos diretamente de arquivos do disco rígido, sem nenhum tipo de cache. Além disso, os experimentos não contemplaram o custo de comunicação de rede, que ocorreria caso mais nós fossem utilizados.

5.3 Conjunto de Exemplos

Os experimentos foram executados sobre conjuntos sintéticos e conjuntos reais de exemplos. Conjuntos de pequena dimensionalidade não possuem ganhos em relação à abordagem distribuída, uma vez que podem ser processados rapidamente de forma sequencial. Assim, os conjuntos de exemplos escolhidos e gerados apresentam grande volume de exemplos.

5.3.1 Conjuntos Sintéticos

Os conjuntos de exemplos sintéticos são estacionários, que possuem distribuição estática; e não estacionários, com distribuição mutável de acordo com o tempo. Esses conjuntos foram produzidos a partir de funções disponíveis nos pacotes `stream` e `streamMOA`, sendo eles:

BG_1M - conjunto estacionário de 1.000.000 exemplos gerado pela função em R `DSD_BarsAndGaussians()`. Possui duas dimensões com quatro grupos de diferentes densidades (Figura 5.3a): (a) gaussiana de maior densidade (círculos na cor preta); (b) gaussiana de menor densidade (triângulos na cor vermelha); (c) barra de maior densidade (cruzes na cor verde); (d) barra de menor densidade (xis na cor azul).

RBF5_1M - conjunto não-estacionário de 1.000.000 exemplos gerado pela função em R `DSD_RandomRBFGeneratorEvents()`. Possui no máximo 8 grupos que se movimentam em direções variadas no espaço bidimensional, sem ruído, e sem ocorrência de eventos de mescla ou divisão de grupos (Figura 5.3b).

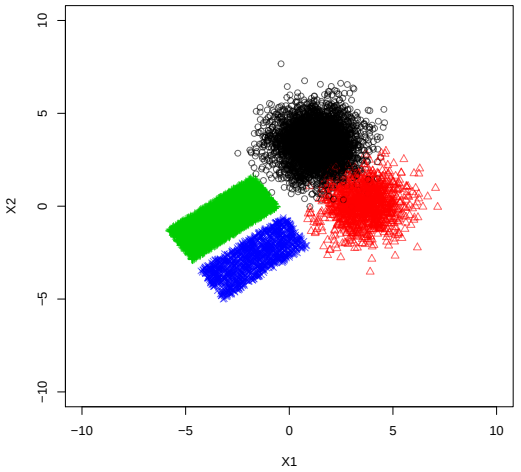
RBF6_400k - conjunto não-estacionário de 400.000 exemplos gerado pela função em R `DSD_RandomRBFGeneratorEvents()`. Possui até 6 grupos que se movimentam no espaço bidimensional em direções variadas, com ruído (círculos cheios em cinza, no canto inferior esquerdo), ocorrendo evento de divisão ou mescla de grupos a cada 50.000 exemplos Figura 5.3c.

SynEDC - conjunto *Synthetic data with concept-drift and novel-class* (MASUD et al., 2011)(AL-KHATEEB et al., 2012a)(AL-KHATEEB et al., 2012b) com 400.000 exemplos, 40 atributos numéricos no intervalo $[0, 1]$ e um atributo classe com 20 valores possíveis. Este conjunto contém mudanças de conceito e surgimento de novas classes ao longo do tempo.

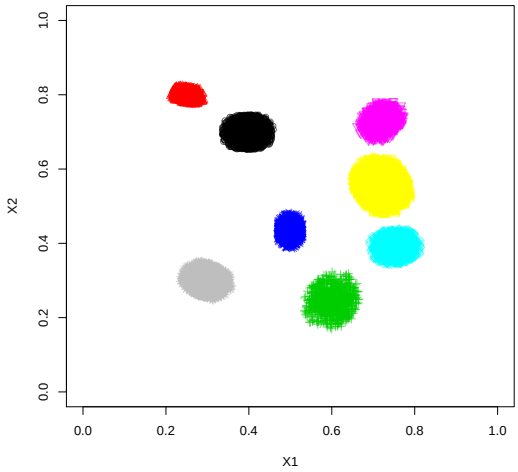
5.3.2 Conjuntos Reais

Para os experimentos com exemplos de domínios reais foram escolhidos dois conjuntos disponíveis no repositório UCI (DHEERU; TANISKIDOU, 2017). Tipicamente utilizados para avaliação de técnicas de classificação, eles são utilizados para avaliação de técnicas de AFD em geral, devido ao elevado número de exemplos e alta dimensionalidade de atributos.

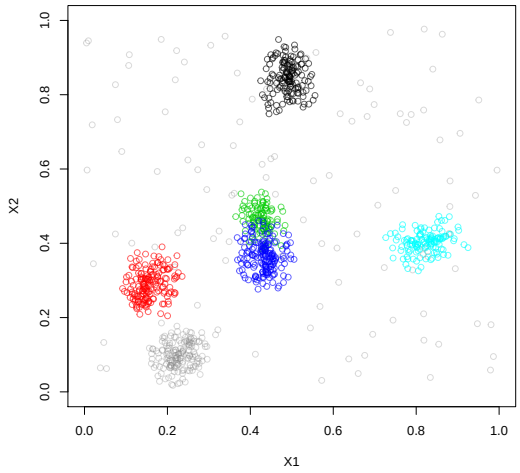
Foram aplicadas técnicas de pré-processamento nos conjuntos de referência, para remoção de atributos de valores nominais e normalização dos valores de atributos contínuos por meio da função `normalize` presente no pacote `BBmisc` do R.



(a) BG_1M



(b) RBF5_1M



(c) RBF6_400k

Figura 5.3: Representação dos exemplos de cada conjunto em diferentes momentos

Covertype - associado à tarefa de classificação do tipo de cobertura florestal de uma área. O conjunto utilizado foi o original com 581.012 exemplos, divididos em 7 classes, e representados por 54 atributos. Após o pré-processamento o conjunto teve todos os seus atributos re-escalados para o intervalo $[0, 1]$.

KDD99 - conjunto de dados que corresponde a um problema real de detecção automática e em tempo real de diversos tipos de ataques a rede mundial de computadores. O conjunto utilizado foi o original com 4.898.431 exemplos, descritos por 41 atributos e um atributo classe (com 23 valores possíveis). Após o pré-processamento, o conjunto teve sua dimensão reduzida para 34 atributos numéricos, todos eles re-escalados para o intervalo $[0, 1]$, e um atributo classe (com 23 valores possíveis).

5.4 Parâmetros e Algoritmos Utilizados

As estratégias presentes na abordagem distribuída, apresentadas no Capítulo 4, são diferentes entre si e também possuem objetivos que se diferenciam. Por conta disso, cada uma dessas estratégias emprega algoritmos diferentes e possuem parâmetros únicos. A seguir são apresentados cada um dos algoritmos e parâmetros que foram definidos para os experimentos para cada estratégia.

5.4.1 Parâmetros da Sumarização Local e Global

Os parâmetros da estratégia de sumarização local e global são compostos pelos parâmetros da fase *online* do *d-FuzzStream*, e pelo número de sumarizações locais.

Um aspecto importante na sumarização realizada pelo *d-FuzzStream* é a definição do tamanho máximo da estrutura de sumarização. Como FMiCs antigos apenas são removidos quando a estrutura estiver cheia, um número elevado de FMiCs pode gerar uma sumarização com exemplos antigos e que não representam a atual situação do FD. Da mesma forma, um número baixo de FMiCs pode prejudicar a sumarização, contemplando poucos exemplos.

Na estratégia de sumarização local e global, múltiplas estruturas de sumarização são utilizadas para sumarizar o FD. Com o aumento da quantidade de sumarizações locais, é possível que o tamanho da estrutura de sumarização global gerada pela fase intermediária também aumente. Assim, foram conduzidos experimentos preliminares para encontrar o tamanho ideal das estruturas de sumarização local para as bases propostas.

Em um experimento inicial foi notado que manter a mesma quantidade de FMiCs da proposta original (200) para todas as estruturas de sumarização locais, faz com que a sumarização global fique muito grande e contemple exemplos antigos. Outros experimentos foram conduzidos reduzindo o tamanho de cada estrutura uma vez que o número de sumarizações locais dobrasse.

Pelos experimentos, diminuir pela metade o tamanho de cada estrutura ao dobrar o número de sumarizações locais gerou resultados insatisfatórios, uma vez que a estrutura fica reduzida quando o número de sumarizações locais é maior do que quatro. O melhor resultado encontrado, foi quando o tamanho máximo de cada estrutura de sumarização local foi reduzido em 25% a cada vez que o número de sumarizações locais dobrava. Esse resultado aproximado se mostrou satisfatório com até 16 sumarizações locais, em que a sumarização global poderia contemplar até 1008 FMiCs.

Essa descoberta também serviu para definir o número máximo de sumarizações locais utilizado nos experimentos. Definiu-se, então, que o número de sumarizações locais utilizado para avaliar a estratégia de sumarização local e global deveria variar de 1 a

16, sendo que o número de FMiCs totais crescem 50% a cada vez que o paralelismo dobra. Sendo assim, o número máximo de FMiCs que a estrutura de sumarização global pode conter é de 200, 300, 448, 672 e 1008, com 1, 2, 4, 8 e 16 sumarizações locais, respectivamente.

Os outros parâmetros do *d-FuzzStream* foram os mesmos utilizados em (SCHICK; LOPES; CAMARGO, 2018) e são mantidos os mesmos independente do grau de paralelismo. Os parâmetros podem ser visualizados na Tabela 5.1.

Tabela 5.1: Parâmetros da estratégia de sumarização local e global

Parâmetro	**Descrição**	**Valor**
minFMiCs	Tamanho mínimo da estrutura de sumarização local	5
maxFMiCs	Tamanho máximo da estrutura de sumarização local	200 150 112 84 63
τ	Limiar de similaridade para mescla	1
m	Fator de fuzzificação	2
janela	Número de exemplos até a fase intermediária	10% do tamanho da base de dados
paralelismo	Grau de paralelismo da sumarização local	1 2 4 8 16

5.4.2 Parâmetros do Agrupamento Distribuído

Os algoritmos utilizados na estratégia de agrupamento distribuído foram o WFCM e o WKMeans com seus valores padrões. O número de grupos varia de 2 até o número máximo de classes que cada base de dados apresenta.

Assim como na estratégia de sumarização local e global, o grau de paralelismo da subtarefa de agrupamento distribuído também variou de 2 a 16. Todos os parâmetros podem ser visualizados na Tabela 5.2.

Atenta-se que não há variação no número de grupos na base BG_1M. Assim, para esta base o número de grupos foi fixado em 4 e o grau de paralelismo foi até 2, uma vez que no máximo 2 agrupamentos estariam acontecento em paralelo.

Tabela 5.2: Parâmetros da estratégia de agrupamento distribuído

Parâmetro	**Descrição**	**WFCM**	**WKMeans**
k	Número de grupos	2 $\sim$ número de classes	2 $\sim$ número de classes
m	Fator de fuzzificação	2	-
ϵ	Limiar de erro para parada	0.001	0.001
iterações	Número máximo de iterações	200	200
paralelismo	Grau de paralelismo do agrupador	1 2 4 8 16	1 2 4 8 16
agrupamentos	Número de agrupamentos realizados por cada agrupador	50	50

5.5 Métricas de Avaliação

As métricas de avaliação utilizadas se dividem em três grupos: internas, externas e de desempenho. As métricas internas e externas foram aplicadas aos resultados da fase offline enquanto que as métricas de desempenho foram aplicadas em ambas as fases. Atenta-se para o fato de que algumas métricas disponíveis não consideram os valores da pertinência *fuzzy* em seus cálculos. A seguir estão as definições desses três grupos e de suas métricas.

5.5.1 Métricas Internas

As métricas internas de avaliação objetivam quantificar o quanto os grupos produzidos pelo agrupamento são compactos e bem separados.

As métricas selecionadas para a avaliação interna dos particionamentos produzidos nos momentos de aplicação da fase *offline* são o coeficiente de silhueta ponderado (*Weighted Silhouette*), o coeficiente de silhueta fuzzy ponderado (*Weighted Fuzzy Silhouette*) e Xie-Beni.

Weighted Silhouette

A silhueta (*silhouette*) é uma métrica que define a qualidade de um agrupamento com base na proximidade entre os exemplos de um determinado grupo e na proximidade desses mesmos exemplos para o grupo mais próximo (ROUSSEEUW, 1987). Já silhueta ponderada (*Weighted Fuzzy Sil*) é uma modificação da silhueta original adicionando os pesos dos exemplos no cálculo.

Para chegar ao valor de silhueta ponderada, primeiramente é calculado o valor de silhueta (s_j) de cada um dos N exemplos a partir da Equação 5.1, onde a_{ij} representa a distância média entre um exemplo j em um cluster i para todos os outros exemplos pertencentes ao mesmo grupo; b_{pj} é a menor distância média entre o exemplo j e todos os outros exemplos de um cluster p, para $p \neq i$; a função $max(a_{ij}, b_{pj})$ corresponde ao maior valor entre a_{ij} e b_{pj}.

$$s_j = \frac{b_{pj} - a_{ij}}{\max(a_{ij}, b_{pj})} \tag{5.1}$$

Por fim, a métrica da silhueta ponderada WS é a razão entre a soma dos valores de s_j multiplicados pelos pesos de cada exemplo j e a soma total dos pesos w_j (Equação 5.2).

$$WS = \frac{\sum_{j=1}^{N} s_j w_j}{\sum_{j=1}^{N} w_j} \tag{5.2}$$

Valores de silhueta podem variar de -1 a 1, onde valores mais próximos de 1 representam uma maior qualidade do agrupamento, ou seja, grupos mais coesos e distantes entre si.

Weighted Fuzzy Silhouette

Baseada no mesmo princípio da silhueta original, a silhueta *fuzzy* é uma versão modificada proposta em (CAMPELLO; HRUSCHKA, 2006), que utiliza os valores de pertinência no seu cálculo. Já a silhueta *fuzzy* ponderada (*Weighted Fuzzy Silhouette*) foi proposta em (SCHICK; LOPES; CAMARGO, 2018) adicionando também o peso de cada exemplo ponderado pelo valor das pertinências.

Assim, a métrica é calculada como na Equação 5.3, onde μ_{ij} e μ_{pj} são respectivamente o maior e o menor valor de pertinência do exemplo j para os grupos, α é um fator de fuzzificação, s_j é o valor de silhueta como demonstrado na Equação 5.1 e w_j é o peso de um exemplo j.

$$WFS = \frac{\sum_{j=1}^{N}(\mu_{ij} - \mu_{pj})^{\alpha} s_j w_j}{\sum_{j=1}^{N}(\mu_{ij} - \mu_{pj})^{\alpha} w_j} \tag{5.3}$$

Xie-Beni

O índice Xie-Beni (XIE; BENI, 1991) é uma métrica para agrupamento *fuzzy* definida como o quociente entre a média do erro quadrático (*MSE*) e a mínima distância quadrática entre os centróides. A métrica pode ser calculada conforme a Equação 5.4, onde μ representa os valores de pertinência, k representa o número de clusters, c os centróides e N representa o número de exemplos.

$$XB = \frac{\sum_{j=1}^{k}\sum_{i=1}^{N}\mu_{ij}^{2}||x_i - c_j||^2}{N * \min_{j \neq l}||c_j - c_l||^2} \tag{5.4}$$

Os valores dessa métrica são sempre maiores que 0, onde menores valores representam melhores agrupamentos.

5.5.2 Métricas Externas

Apesar do agrupamento ser uma técnica de aprendizado não-supervisionado, é possível que os dados possuam informações de classe. Assim, métricas externas podem ser utilizadas para avaliar até que ponto os grupos representam a distribuição de rótulos de um conjunto de exemplos. As métricas selecionadas para a avaliação externa dos particionamentos produzidos nos momentos de aplicação da fase *offline* são pureza (*Purity*) e índice Rand (*Rand Index*).

Purity

A Pureza (*Purity*) é uma métrica que indica a distribuição de classes dentro de cada grupo. Pode assumir valores de 0 a 1, onde maiores valores de pureza indicam melhor distribuição de classes.

Dado um conjunto de grupos $G = \{g_1, g_2, ..., g_k\}$ e um conjunto de classes $T = \{t_1, t_2, ..., t_j\}$, a pureza de um grupo g_i é a razão entre a maior quantidade de exemplos que possuem a mesma classe e o número de exemplos presentes no grupo. Já a pureza do agrupamento, é caracterizada pela média entre a pureza de todos os grupos. Assim, a Equação 5.5 demonstra como este cálculo é realizado de maneira simplificada, onde N é o número total de exemplos e $|g_i \cap t_j|$ caracteriza o número de exemplos no grupo g_i que possuem a classe t_j, com j diferente de i.

$$Pur = \frac{1}{N} \sum_{i=1}^{k} \max_{j \neq i} |g_i \cap t_j| \tag{5.5}$$

Rand Index

O índice Rand (*Rand Index*), proposto por (RAND, 1971), avalia a assertividade quanto à distribuição de classes por meio do particionamento produzido pelo agrupamento. Assumindo que cada grupo possui uma classe definida ou a correta, o índice Rand pode ser calculado pela Equação 5.6.

$$RI = \frac{VP + VN}{VP + VN + FP + FN} \tag{5.6}$$

O cálculo é feito em pares onde os valores da equação representam a quantidade de pares de exemplos que:

- **Verdadeiros Positivos (VP):** possuem a mesma classe, estão no mesmo grupo e no grupo correto.
- **Verdadeiros Negativos (VN):** possuem classes diferentes, estão em grupos diferentes e nos grupos corretos.
- **Falsos Positivos (FP):** possuem a mesma classe, estão no mesmo grupo porém no grupo incorreto.
- **Falsos Negativos (FN):** possuem classes diferentes, estão no mesmo grupo e no grupo incorreto.

Essa métrica pode assumir valores entre 0 e 1, sendo que valores maiores indicam melhor assertividade na distribuição das classes.

5.5.3 Métricas de Desempenho

As métricas de desempenho aqui definidas são métricas que permitem avaliar a melhoria em velocidade na execução de uma mesma tarefa em duas arquiteturas com diferentes recursos. Essas métricas foram utilizadas para avaliar o ganho em desempenho da abordagem distribuída em relação à sequencial. As métricas selecionadas para essa avaliação foram: tempo de execução, *Speedup* e eficiência.

Tempo de Execução

O tempo de execução é o tempo em que uma tarefa computacional permanece em execução. Pode ser simplesmente calculada pela diferença de tempo entre o começo e o final do processamento.

Speedup

O *Speedup* é uma medida baseada na lei de Amdahl (AMDAHL, 1967) útil para analisar o ganho de desempenho quando uma parte da execução de uma tarefa é melhorada. Na computação distribuída essa métrica visa analisar o ganho de desempenho quando o paralelismo de execução de uma tarefa é aumentado. Assim, o *speedup* compara os tempos de execução da tarefa de forma sequencial e paralela. A Equação 5.7 demonstra como a métrica pode ser calculada.

$$Speedup = \frac{Tempo\ de\ execução\ da\ tarefa\ de\ forma\ sequencial}{Tempo\ de\ execução\ da\ tarefa\ de\ forma\ paralela} \tag{5.7}$$

Eficiência

A métrica eficiência é uma métrica que calcula o ganho de desempenho em relação ao consumo de recursos computacionais empregados na execução paralela (EAGER; ZAHORJAN; LAZOWSKA, 1989). Ela é a razão entre o *speedup* e o recurso computacional a ser avaliado, que, no caso da abordagem distribuída, é o número de processadores. A Equação 5.8 apresenta como a métrica é calculada.

$$Eficiência = \frac{Speedup}{Número\ de\ processadores} \tag{5.8}$$

5.6 Considerações Finais

Os experimentos detalhados neste capítulo foram estabelecidos com a finalidade de identificar e avaliar o comportamento da abordagem distribuída para o processo de agrupamento não-supervisionado *fuzzy* em FD, comparada à abordagem sequencial.

Essa análise comportamental objetivou entender o ganho de desempenho das duas estratégias empregadas na abordagem distribuída e o impacto da qualidade do agrupamento, principalmente da estratégia de sumarização local e global. A análise comentada dos resultados é apresentada no Capítulo 7.

Parte III

Análises e Resultados

6

Resultados do *d-FuzzStream*

Como apresentado no Capítulo 4, o *d-FuzzStream* é uma evolução do *FuzzStream* que introduz os conceitos de similaridade e dispersão *fuzzy*, propostos em (XIONG; CHAN; TAN, 2004). Assim, este capítulo apresenta os resultados da comparação entre os dois algoritmos encontrados em Schick, Lopes e Camargo (2018).

A metodologia utilizada não segue diretamente a metodologia da abordagem distribuída, apresentada no Capítulo 5. Além disso, as análises realizadas não contemplaram nenhum tipo de paralelismo. As sessões a seguir apresentam como foram realizados os experimentos e os resultados obtidos.

6.1 Experimentos

A validação e avaliação do *FuzzStream* e *d-FuzzStream* para agrupamento em FD foram conduzidas por meio da execução de experimentos utilizando conjuntos de dados sintéticos. Esses conjuntos foram gerados usando os pacotes *stream* e *streamMOA* do *R* e simulados como fluxos de dados. Mais informações sobre esses dados estão disponíveis em Computational Intelligence Group (CIG) (2017). Os conjuntos de dados utilizados foram:

BG_10k - conjunto estacionário de 10.000 exemplos gerado pela função em R `DSD_BarsAndGaussians()`. Possui duas dimensões com quatro grupos de diferentes densidades.

Bench1_11k - conjunto não-estacionário de 11.000 exemplos gerado pela chamada `DSD_Benchmark(i = 1)`. Possui dois grupos que se movem diagonalmente no espaço bidimensional (com ruído).

RBF4_400K - conjunto não-estacionário de 40.000 exemplos gerado pela função em R `DSD_RandomRBFGeneratorEvents()`. Possui no máximo 7 grupos que se movimen-

tam no espaço bidimensional em direções variadas, com ruído, e com ocorrência de eventos de mescla e divisão de grupos.

Foram conduzidos 6 experimentos utilizando os parâmetros dos dois algoritmos seguindo a proposta original. Para o agrupamento da fase *offline* foi utilizado o algoritmo FCM. Todos os parâmetros utilizados nos experimentos podem ser visualizados na Tabela 6.1, separados pelas fases *online* e *offline*.

Tabela 6.1: Parâmetros do *d-FuzzStream* e *FuzzStream*

Fase	**Parâmetro**	***FuzzStream***	***d-FuzzStream***
Online	$MinFMiC$	5	5
	$MaxFMiC$	200	200
	θ	0.8	-
	Θ	0.8	-
	τ	-	1
	m	2	2
	$janela$	20% do tamanho da base de dados	20% do tamanho da base de dados
Offline	k	Número de classes	Número de classes
	m	2	2
	ϵ	0.001	0.001
	$iterações$	200	200

Nota-se que os parâmetros *theta* minúsculo (θ) e *theta* maiúsculo (Θ) são exclusivos do *FuzzStream*. Eles são limiares de pertinência que o algoritmo utiliza para que um exemplo não seja considerado um *outlier* (θ) ou para que dois FMiCs sejam mesclados (Θ).

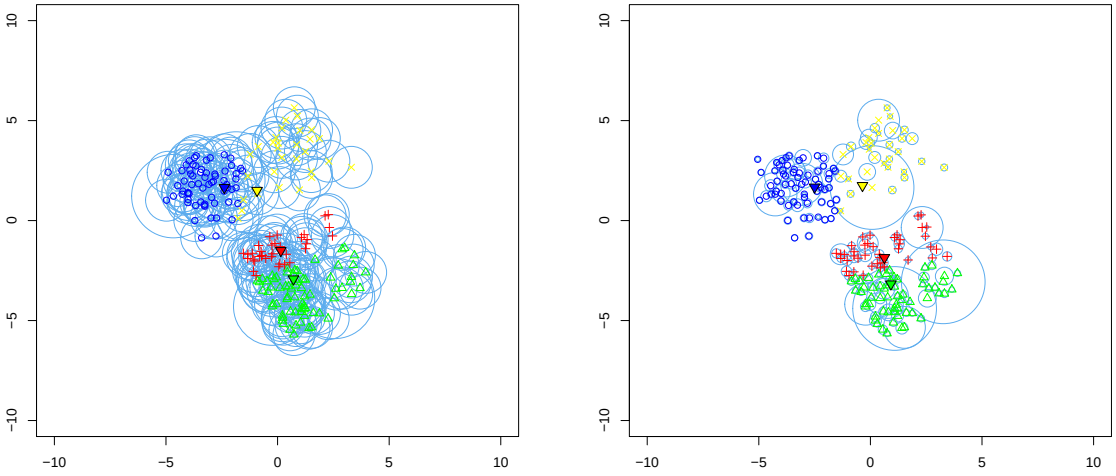
Os experimentos de comparação entre os dois algoritmos visaram analisar como o *d-FuzzStream* afetou o processo de sumarização e como isso se refletiu no agrupamento final. Para tanto, foram utilizadas métricas informativas, uma métrica de agrupamento e uma métrica de desempenho. Também foram analisadas graficamente as estruturas de sumarização e os agrupamentos finais.

Os resultados foram baseados em observações dos conjuntos de dados, que representam as estruturas de sumarização e os seus agrupamentos após o fim de uma janela temporal.

6.2 Estruturas de Sumarização e Agrupamento

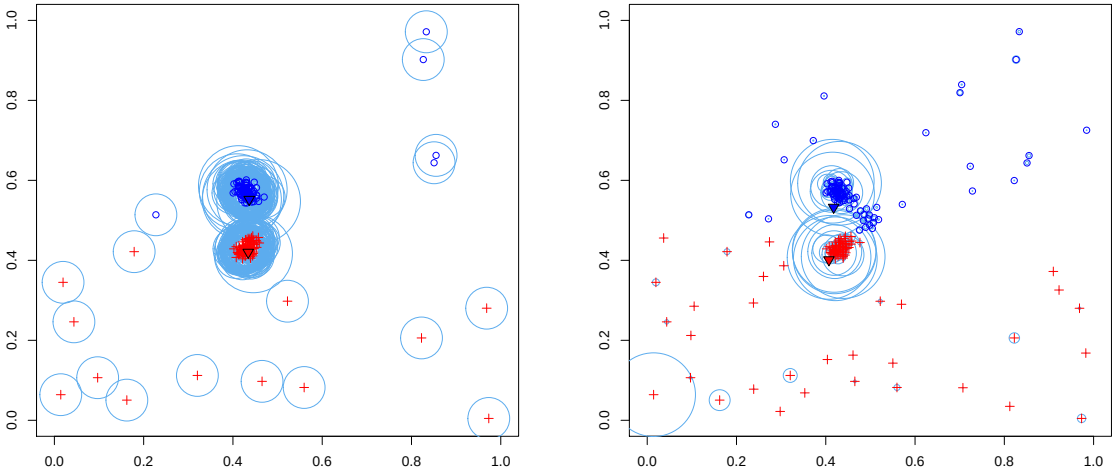
As figuras 6.1, 6.2 e 6.3 apresentam as estruturas de sumarização e o agrupamento final de uma observação de cada conjunto de dados. Nas figuras, o agrupamento final é indicado pelos protótipos gerados na etapa *offline*, representados por um triângulo invertido. Os outros pontos representam os FMiCs presentes na estrutura de sumarização

naquele dado momento. Todo FMiC possui um raio associado (contorno azul) que representa o seu peso. Os raios não seguem nenhuma escala e suas diferenças de tamanho representam unicamente a diferença de peso entre os FMiCs.



(a) *FuzzStream* (b) *d-FuzzStream*

Figura 6.1: Estrutura de sumarização e agrupamento da observação 5 do BG_10K

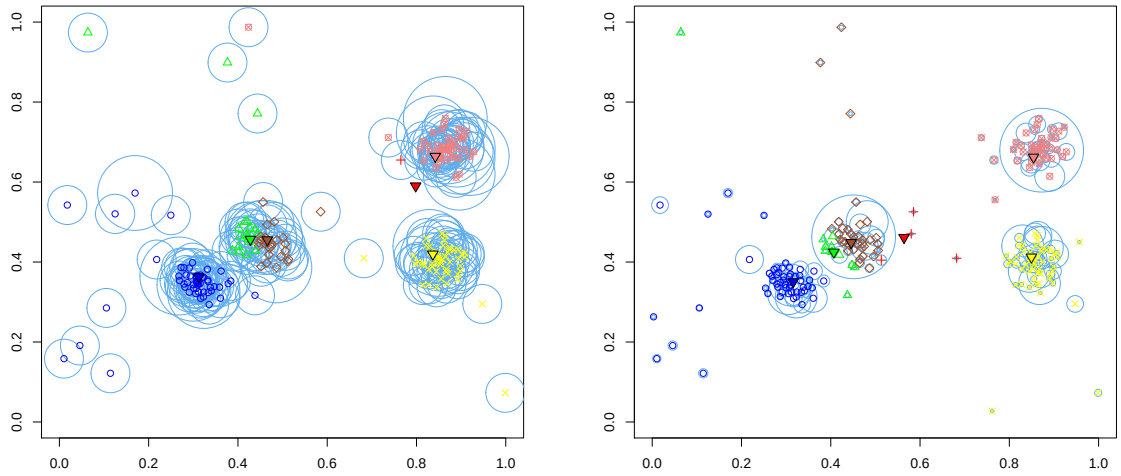


(a) *FuzzStream* (b) *d-FuzzStream*

Figura 6.2: Estrutura de sumarização e agrupamento da observação 4 do Bench1_11K

Como pode ser notado, os raios dos FMiCs no *FuzzStream* (6.1a, 6.2a e 6.3a) possuem tamanhos bastante semelhantes entre si. Isso se deve ao fato de os raios possuirem o mesmo valor, mostrando que o processo de sumarização remove e cria muitos FMiCs, retendo poucas informações dos exemplos passados.

Por outro lado, o *d-FuzzStream* gerou FMiCs com pesos bastante distintos, mostrando melhor retenção de informação. Essa diferença nos pesos também torna mais clara a diferença entre os FMiCs que representam exemplos e os que representam *outliers*. Como pode ser visto nos resultados do *d-FuzzStream* (6.1b, 6.2b e 6.3b), os FMiCs que representam *outliers* têm um raio menor quando comparado com aqueles FMiCs que representam



(a) *FuzzStream* (b) *d-FuzzStream*

Figura 6.3: Estrutura de sumarização e agrupamento da observação 2 do RBF4_40K

a distribuição real de exemplos. Nota-se que o raios dos *outliers* são muito pequenos e pouco visíveis.

Quanto aos resultados do agrupamento final, o *d-FuzzStream* gerou protótipos muito próximos daqueles gerados pelo *FuzzStream*, razão pela qual podemos inferir que as mudanças implementadas na etapa *online* do *d-FuzzStream* não prejudicam os resultados da fase *offline*.

6.3 Métricas de Avaliação

Os resultados das métricas de avaliação foram divididos entre métricas informativas, métrica interna e métricas de desempenho. São apresentados os resultados relativos aos agrupamentos com o número de grupos igual ao número de classes presentes no conjunto de dados.

6.3.1 Métricas Informativas

As métricas informativas utilizadas na avaliação dizem respeito às quatro operações que são realizadas no processo de sumarização. Essas operações incluem: absorção de um exemplo, mescla de dois FMiCs, criação e remoção de um FMiC.

Um exemplo atribuído a todos os FMiCs foi considerado como uma absorção, um FMiC criado é considerado como uma criação, e o mesmo princípio é aplicado quando um FMiC é removido e dois FMiCs são mesclados. As Tabelas 6.2, 6.3 e 6.4 mostram o resultados dessas métricas.

Como esperado, o algoritmo *FuzzStream* mostrou taxas de absorção e mescla muito baixas. Consequentemente, os números de criações e remoções de FMiCs são altos e

semelhantes entre si e também ao tamanho do conjunto de dados, para todos os três conjuntos de dados. Isso significa que os FMiCs no algoritmo quase sempre representam um único exemplo em vez de um grupo de exemplos.

Tabela 6.2: Resultado das métricas informativas para o BG_10K

	FuzzStream	**d-FuzzStream**	**Diferença**
Criações	10.541	8.747	-17%
Remoções	10.198	3.748	-63%
Absorções	459	2.253	+391%
Mesclas	143	4.860	+3.299%

Tabela 6.3: Resultado das métricas informativas para o Bench1_11K

	FuzzStream	**d-FuzzStream**	**Diferença**
Criações	9.529	8.884	-7%
Remoções	9.195	6.191	-33%
Absorções	471	1.116	+137%
Mesclas	134	2.496	+1.763%

Tabela 6.4: Resultado das métricas informativas para o RBF4_40K

	FuzzStream	**d-FuzzStream**	**Diferença**
Criações	38.205	34.187	-11%
Remoções	37.411	20.388	-46%
Absorções	1.795	5.813	+224%
Mesclas	594	13.606	+2.191%

É notável que o número de criações de FMiCs tem uma pequena diferença em ambas as versões. O *d-FuzzStream* foi capaz de criar menos FMiC durante o processo de aprendizado, diferenças variando de -17% a -7% comparado ao *FuzzStream*. Isso indica que o *d-FuzzStream* favorece a absorção de novos exemplos, como visto nos resultados.

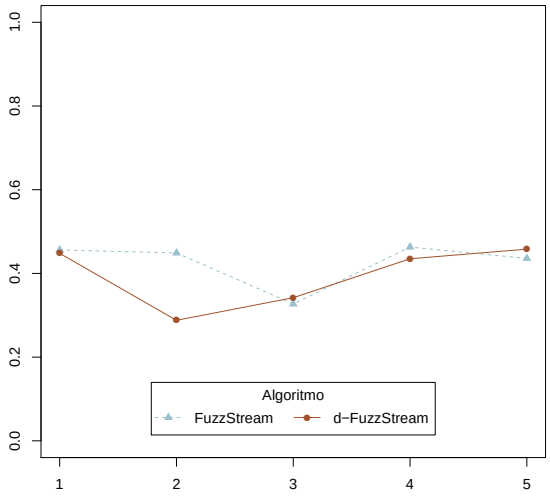
A taxa de mescla teve a maior melhoria para todos os conjuntos de dados, atingindo quase 3.300% de aumento, para um dos conjuntos de dados. De fato, o número de mesclas se tornou, em média, três vezes maior que o número de absorções, enquanto que no *FuzzStream* é o oposto. Esses resultados demonstram que *d-FuzzStream* reteve mais informações sobre os exemplos e que seu processo de sumarização é baseado principalmente em mescla.

Embora o *d-FuzzStream* tenha retido mais informações, o número de remoções ainda é maior ou comparável ao número de absorções e mesclagens em todos os bancos de dados. Isso prova que o algoritmo ainda tem a capacidade de esquecer conceitos antigos removendo FMiCs antigos.

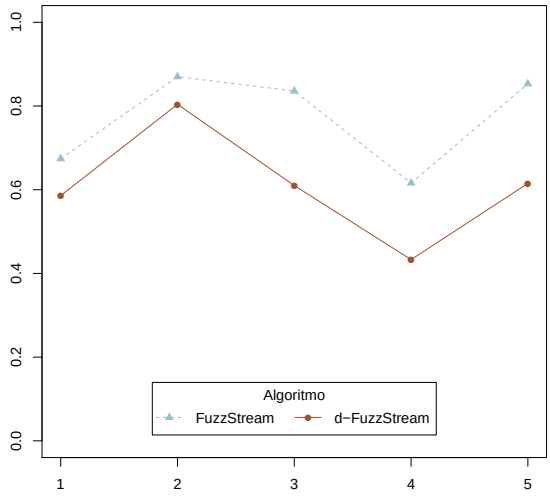
6.3.2 Métrica Interna

Para avaliação do agrupamento, foi utilizada a silhueta *fuzzy* ponderada como métrica interna de avaliação. Esta métrica varia no intervalo [-1,1], onde maiores valores representam um melhor agrupamento. A Figura 6.4 mostra os valores dessa métrica para os conjuntos de dados utilizados, onde o eixo X representa os momentos de avaliação, e o eixo Y representa o valor da métrica.

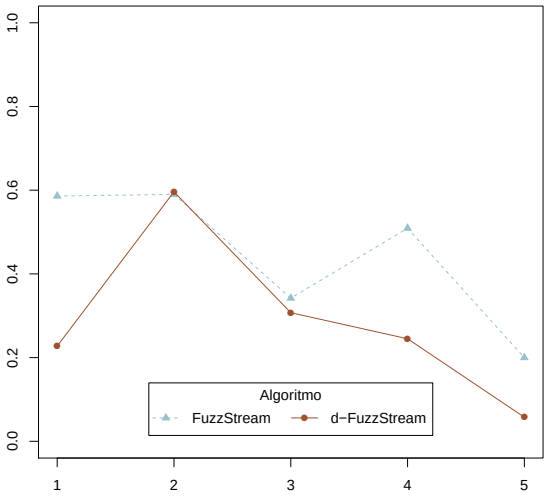
O *d-FuzzStream* apresentou valores mais baixos ou semelhantes de silhueta *fuzzy* em relação ao *FuzzStream*. Isso se deve ao fato de que ambos os algoritmos produzem diferentes FMiCs e os resultados da fase *online* do *d-FuzzStream* contêm um número maior de FMiCs que representam *outliers*. No entanto, ambos os algoritmos possuem comportamentos semelhantes em termos de silhueta. Ao analisar um conjunto de dados sem ruído, como BG_10K, percebe-se que *d-FuzzStream* produz os mesmos ou até melhores resultados de silhueta que o *FuzzStream*.



(a) BG_10K



(b) Bench1_11K



(c) RBF4_40K

Figura 6.4: Resultados comparativos de Silhueta *Fuzzy*

6.3.3 Métricas de Desempenho

Os conceitos de similaridade e dispersão *fuzzy* empregados no *d-FuzzStream* removeram completamente a necessidade de calcular a matriz de pertinência entre os FMiCs no processo de mescla, como ocorre no *FuzzStream*. Isso diminuiu a complexidade do algoritmo e reduziu 98% do tempo de processamento total, como pode ser observado na Tabela 6.5.

Tabela 6.5: Comparativo entre os tempos de execução do *FuzzStream* e *d-FuzzStream*

	FuzzStream	**d-FuzzStream**
BG_10k	6m26s	6s
Bench1_11k	6m55s	5s
RBF4_40k	25m35s	23s

6.4 Considerações Finais

Os conceitos de dispersão e similaridade *fuzzy* usados pelo *d-FuzzStream* melhoraram a sumarização de dados mantendo mais informações sobre os exemplos. Isto é devido a uma maior taxa de absorção de exemplos e mescla de FMiCs, assim como uma menor taxa de criação e remoção de FMiCs. Assim, o algoritmo gera FMiCs com maior representatividade e com pesos mais distintos, tornando mais clara a análise de quais FMiCs representam exemplos reais de distribuição ou outliers.

Em termos de complexidade, o *d-FuzzStream* elimina a necessidade de calcular matrizes de pertinência para todos os FMiCs na etapa de mescla e também elimina a necessidade de calcular a matriz de pertinência do exemplo para toda a estrutura de sumarização quando o exemplo é identificado como um *outlier*. Isso contribui para a produção de resultados de agrupamento similares 98% mais rápido que o *FuzzStream*.

7

Resultados do *Distributed d-FuzzStream*

Este capítulo tem como objetivo apresentar o resultados dos 135 experimentos executados sobre os 6 conjuntos de dados apresentados no Capítulo 5. Esses resultados foram baseados em observações do fluxo de dados, que são compostas pela estrutura de sumarização e os agrupamentos após o fim de uma janela temporal.

Em consequência do grande volume do conjunto total de resultados, foram selecionadas duas observações diferentes de cada base de dados com o objetivo de esclarecer os comportamentos e decorrências da utilização da abordagem distribuída descritos no Capítulo 4.

A análise e discussão dos resultados está divida em quatro seções: iniciando pela apresentação das estruturas de sumarização e agrupamentos e seguido pelas métricas selecionadas para validação.

7.1 Estruturas de Sumarização e Agrupamentos

Nesta seção são apresentados os resultados das sumarizações e agrupamentos de três das seis bases de dados utilizadas, sendo elas as bases BG_1M, RBF5_1M e RBF6_400K. Somente essas bases possuem dados bidimensionais, o que permite visualização gráfica.

As Figuras 7.1, 7.3, 7.5, 7.7, 7.9 e 7.11 apresentam as estruturas de sumarização obtidas após a fase *online* e intermediária com diferentes graus de paralelismo, enquanto que as Figuras 7.2, 7.4, 7.6, 7.8, 7.10 e 7.12 mostram os agrupamentos finais gerados pela fase *offline*. Destaca-se que o grau de paralelismo é apenas relacionado à fase *online*, uma vez que o paralelismo na fase *offline* não impacta o resultado do agrupamento final.

Cada FMiC nas estruturas de sumarização possui uma cor e símbolo associados que representam a classe à qual ele foi associado, e um raio (contorno azul), que representa o seu peso. Esses raios não seguem nenhuma escala e suas diferenças de tamanho representam unicamente a diferença de peso entre os FMiCs.

No agrupamento final, os triângulos invertidos representam os centróides de cada grupo e cada exemplo possui uma cor e símbolo associados que representam o grupo ao qual pertencem. Devido ao grande volume de resultados, serão apenas apresentados os resultados dos agrupamentos gerados pelo WFCM.

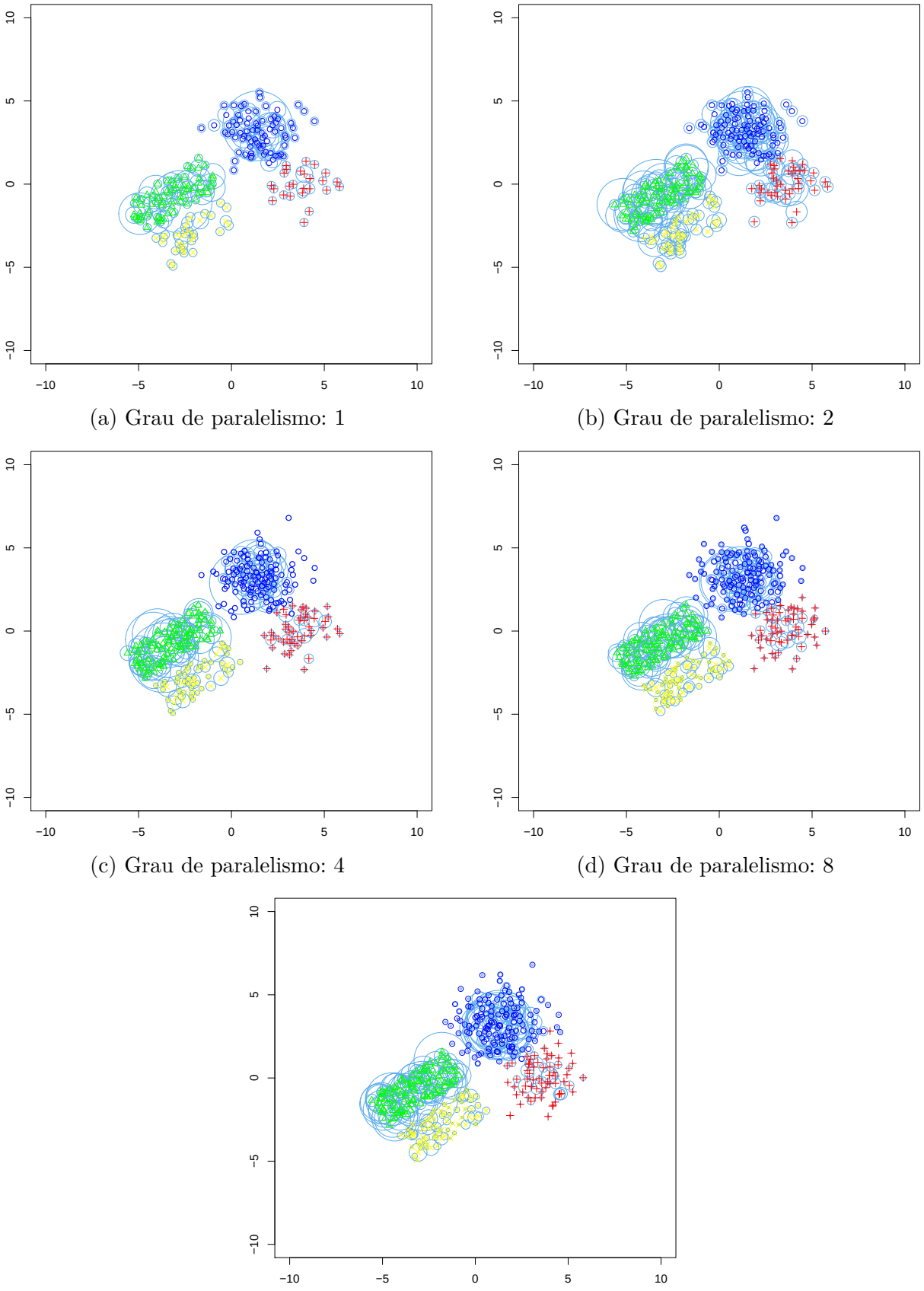
Na base estacionária BG_1M, o aumento do grau de paralelismo permitiu representar mais exemplos do fluxo, uma vez que o número de FMiCs subiu de 200 (sem paralelismo) para 377 (com grau de paralelismo 16) na primeira observação (Figura 7.1), e de 200 para 387 na décima observação (Figura 7.3). É possível constatar que isso também afetou o peso dos FMiCs na estrutura de sumarização, que ficaram mais discrepantes nas classes com maior densidade verde e azul.

Já no agrupamento final, visivelmente há pouca diferença entre as posições dos centróides e dos exemplos que estão em cada grupo (Figura 7.2 e Figura 7.4). Ressalta-se ainda que os dois grupos representados por barras foram separados ao meio de maneira irregular, uma vez que os algoritmos WFCM e WKMeans representam apenas grupos elípticos.

A base não-estacionária RBF5_1M também apresentou comportamento similar. O aumento no número de FMiCs de 200 para 367 na segunda observação (Figura 7.5), e de 200 para 390 na terceira observação (Figura 7.7), permitiu representar mais exemplos, o que também modificou os pesos dos exemplos. Na fase *offline*, o número de grupos escolhido foi igual ao número de classes (Figura 7.6 e Figura 7.8), e mais uma vez o aumento do grau de paralelismo não trouxe mudanças visíveis. Nota-se que existem duas classes sobrepostas em ambas as observações. Na observação 2, os grupos marrom e verde (losango e triângulo) estão sobrepostos, e na observação 3 a sobreposição ocorre entre os grupos rosa e azul escuro (triângulo invertido e asterisco).

Por fim, a base RBF6_400K também apresentou comportamento semelhante até oito graus de paralelismo. Já no resultado com dezesseis graus (Figura 7.9e e Figura 7.11e), o maior tamanho da estrutura de sumarização permitiu que FMiCs que representavam exemplos muito antigos não fossem removidos.

Na Figura 7.9e é possível visualizar que FMiCs representando a classe vermelha (cruz) e azul escuro (asterisco) formam dois grupos diferentes e distantes, e na Figura 7.11e além da classe vermelha (cruz) ser encontrada em dois lugares distintos, existem FMiCs representando a classe amarela (xis). Esta última não está mais presente na situação atual do FD. Desta maneira, o agrupamento final apresentado nas Figuras 7.10e e 7.12e foi afetado de maneira significativa, demonstrando que a sumarização realizada pelo *d-FuzzStream* possui sensibilidade em relação ao tamanho da estrutura de sumarização.



(a) Grau de paralelismo: 1

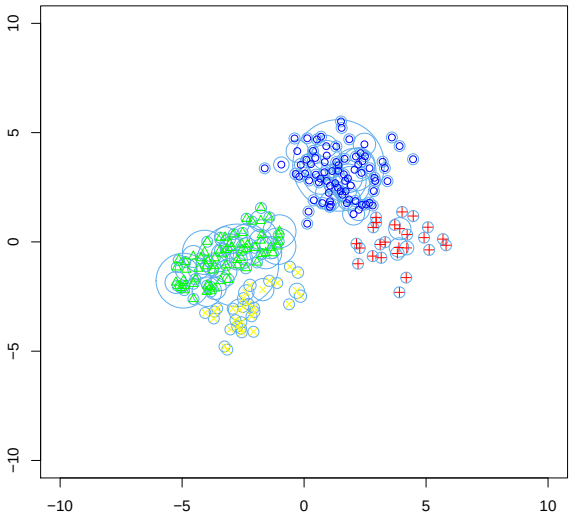
(b) Grau de paralelismo: 2

(c) Grau de paralelismo: 4

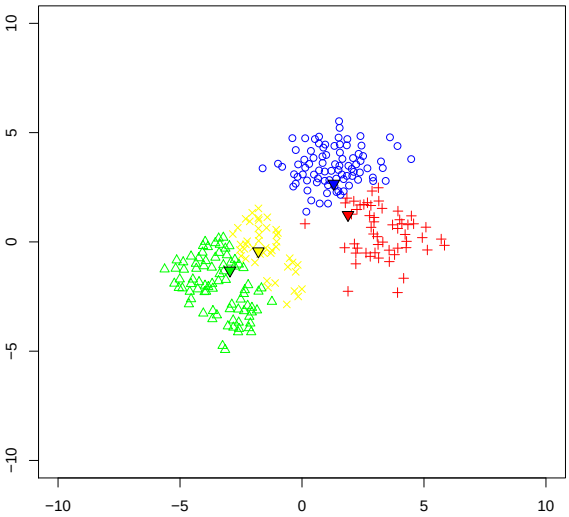
(d) Grau de paralelismo: 8

(e) Grau de paralelismo: 16

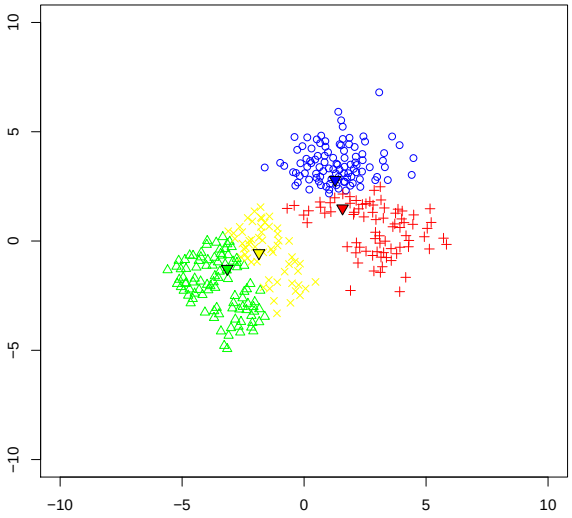
Figura 7.1: Estruturas de sumarização da observação 1 do BG_1M



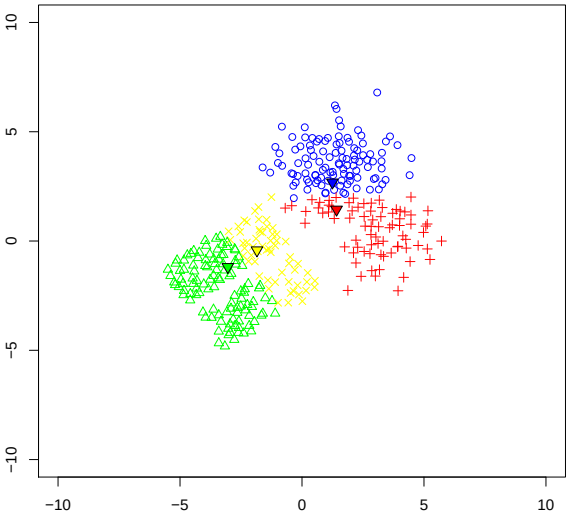
(a) Grau de paralelismo: 1



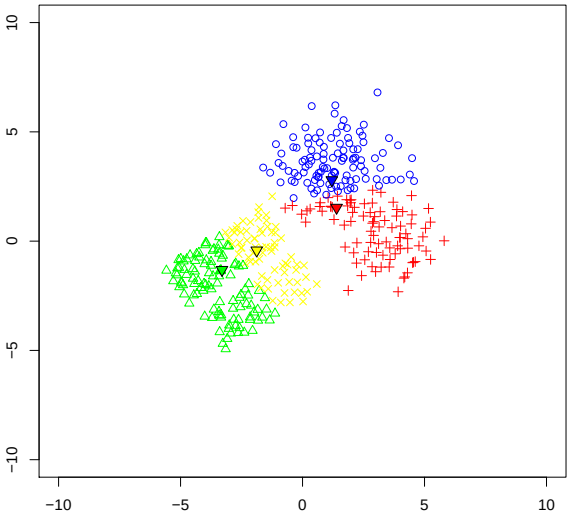
(b) Grau de paralelismo: 2



(c) Grau de paralelismo: 4

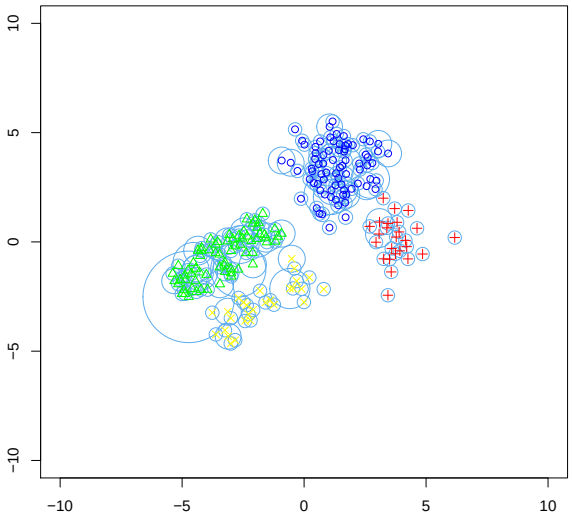


(d) Grau de paralelismo: 8

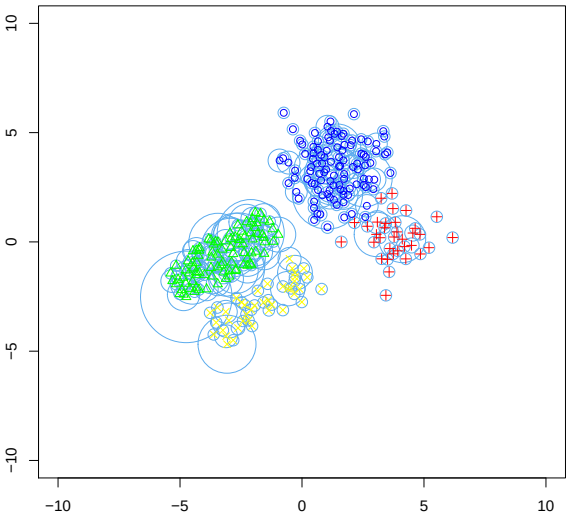


(e) Grau de paralelismo: 16

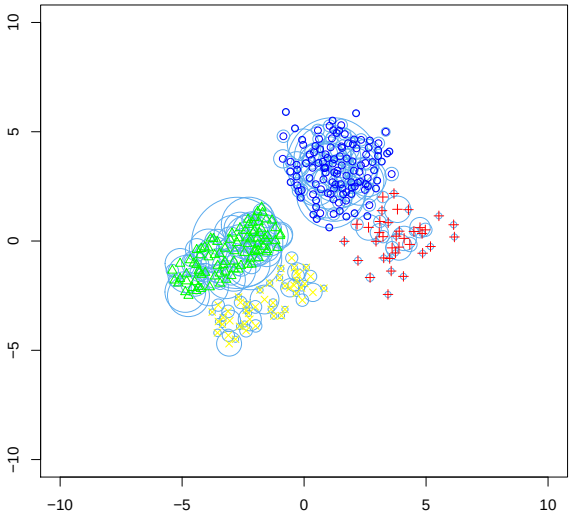
Figura 7.2: Agrupamento final da observação 1 do BG_1M



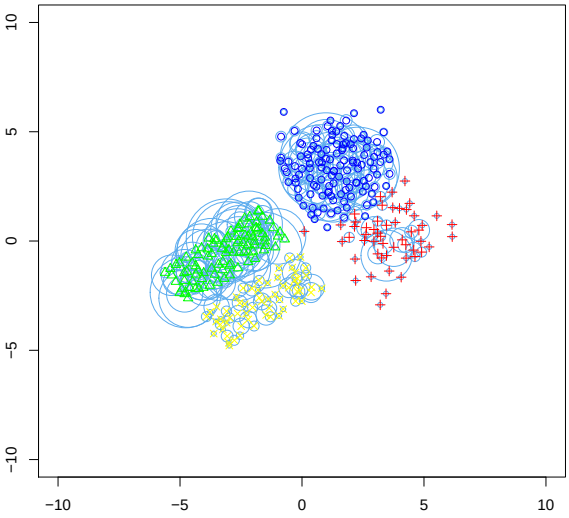
(a) Grau de paralelismo: 1



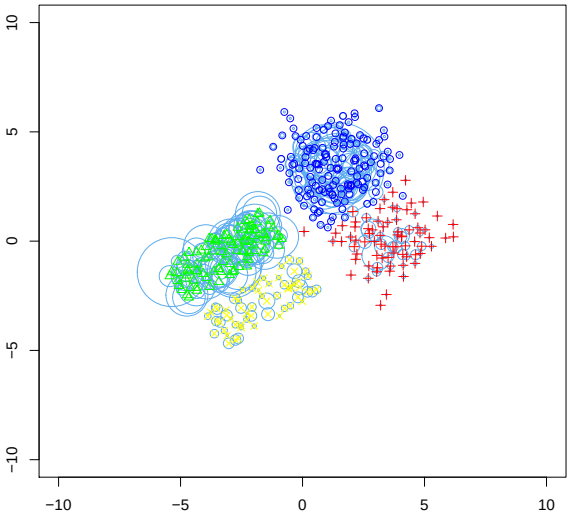
(b) Grau de paralelismo: 2



(c) Grau de paralelismo: 4

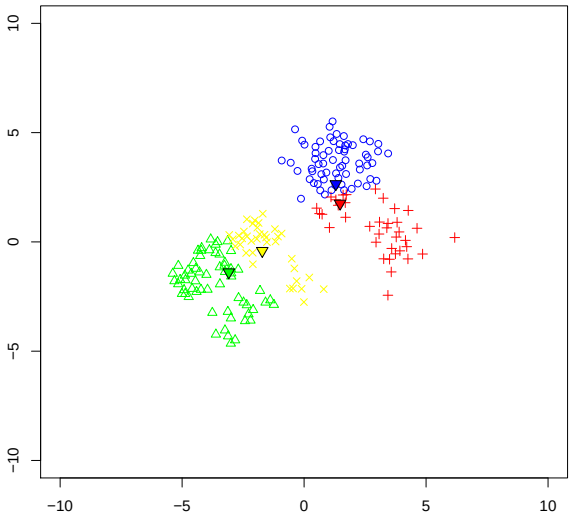


(d) Grau de paralelismo: 8

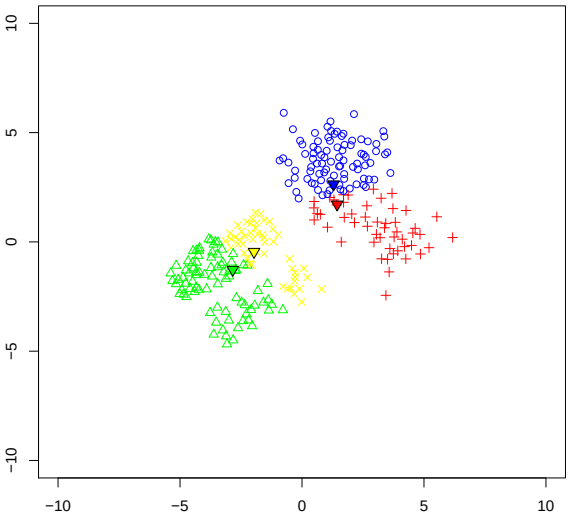


(e) Grau de paralelismo: 16

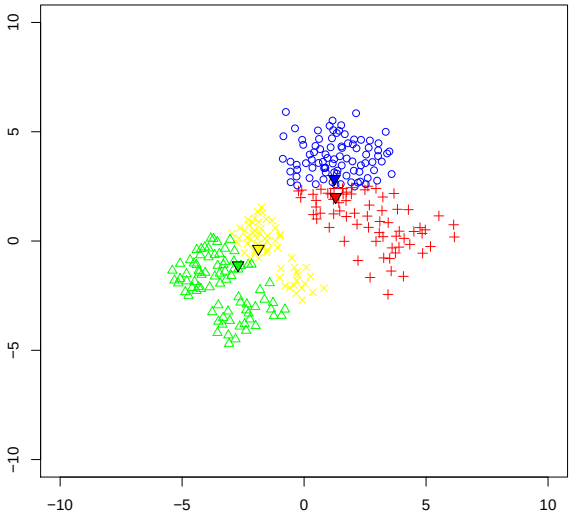
Figura 7.3: Estruturas de sumarização da observação 10 do BG_1M



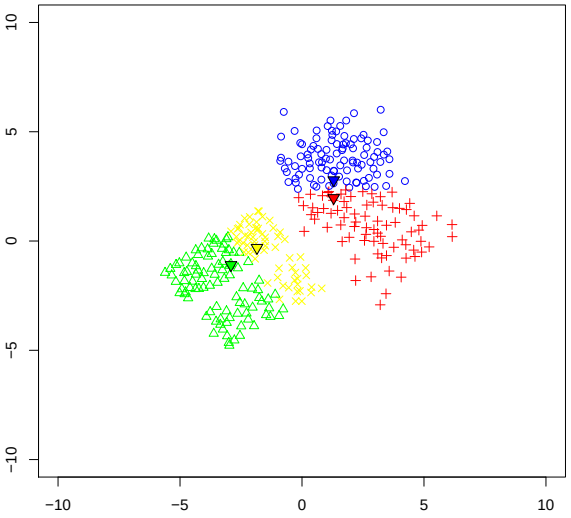
(a) Grau de paralelismo: 1



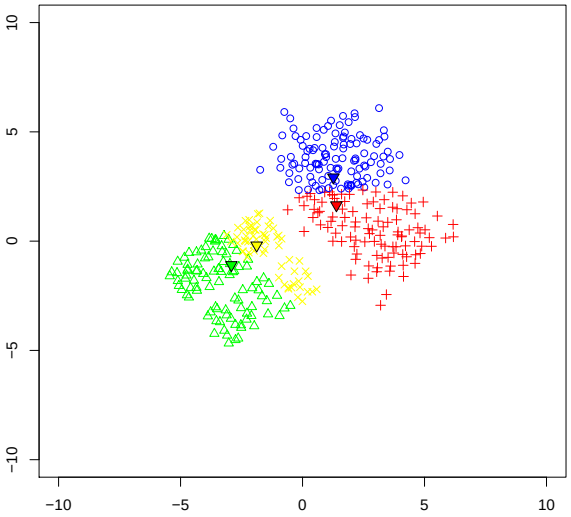
(b) Grau de paralelismo: 2



(c) Grau de paralelismo: 4

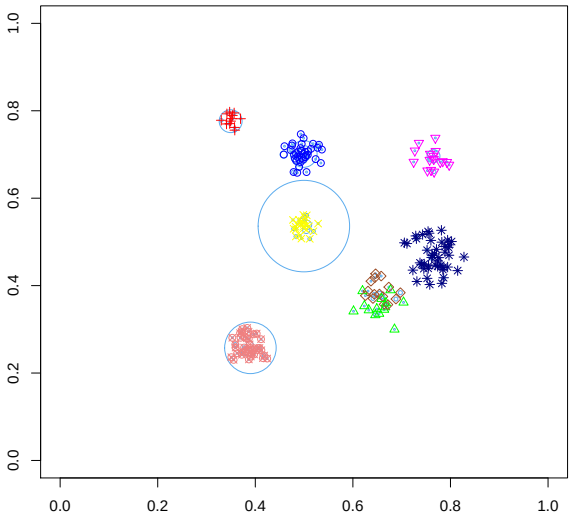


(d) Grau de paralelismo: 8

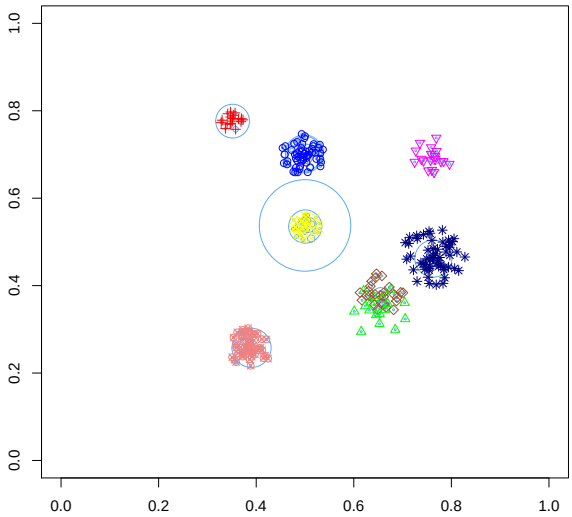


(e) Grau de paralelismo: 16

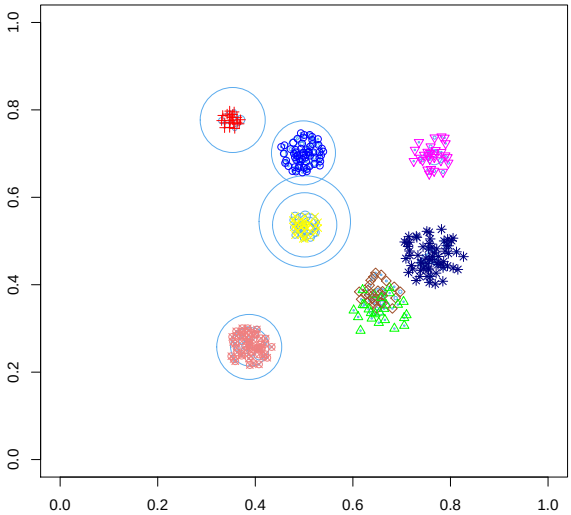
Figura 7.4: Agrupamento final da observação 10 do BG_1M



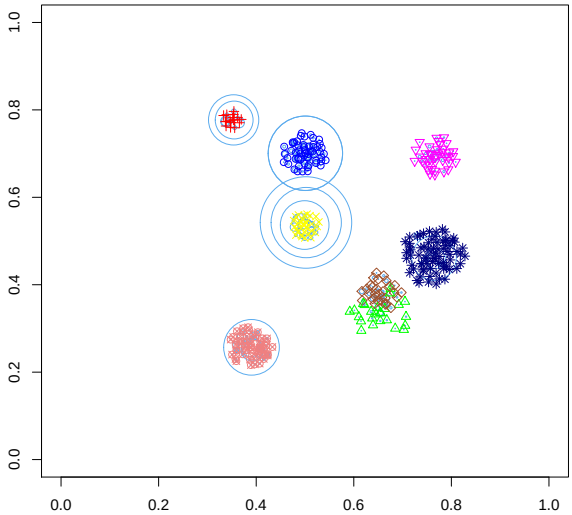
(a) Grau de paralelismo: 1



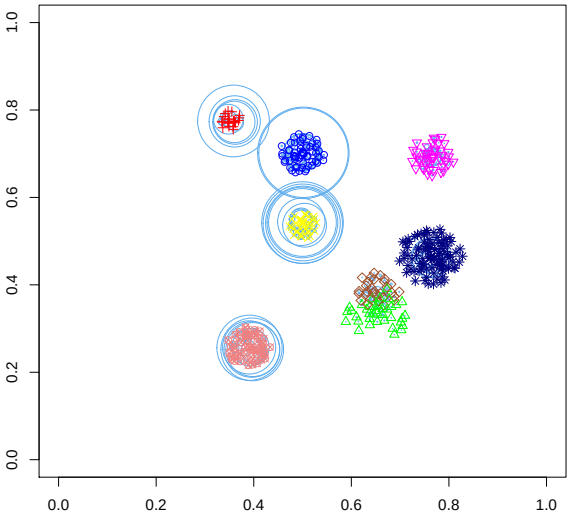
(b) Grau de paralelismo: 2



(c) Grau de paralelismo: 4

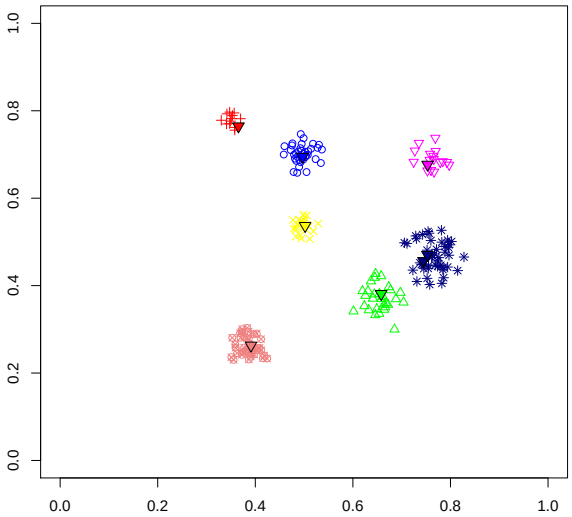


(d) Grau de paralelismo: 8

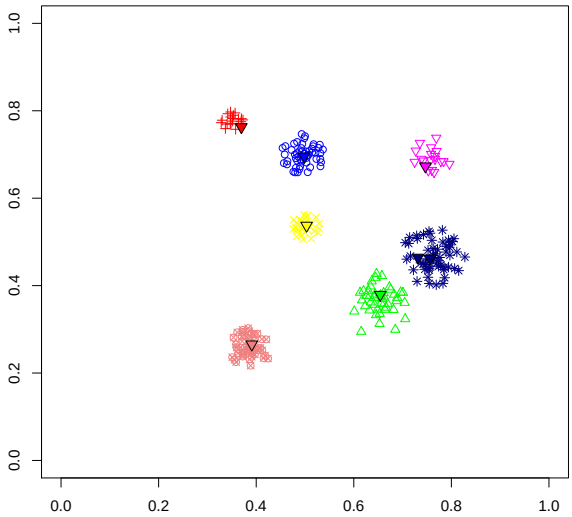


(e) Grau de paralelismo: 16

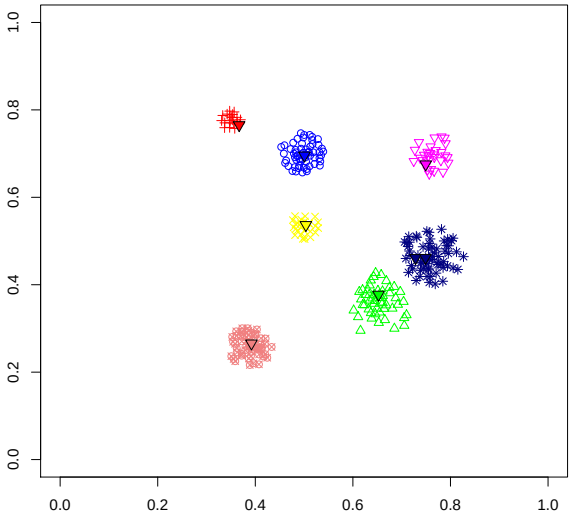
Figura 7.5: Estruturas de sumarização da observação 2 do RBF5_1M



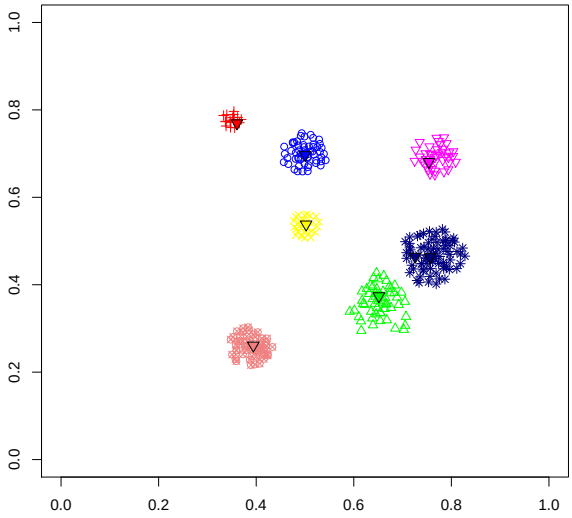
(a) Grau de paralelismo: 1



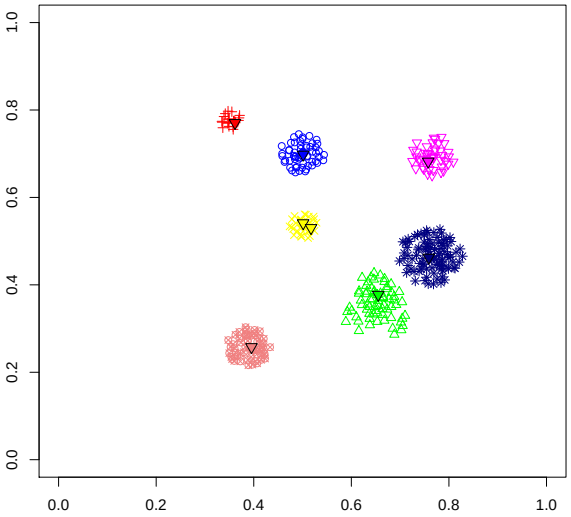
(b) Grau de paralelismo: 2



(c) Grau de paralelismo: 4

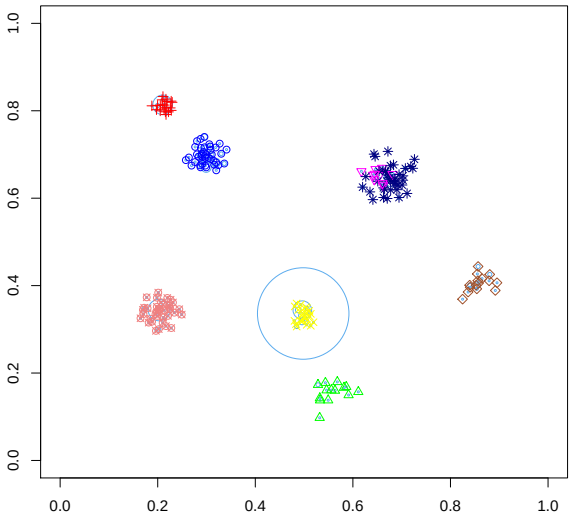


(d) Grau de paralelismo: 8

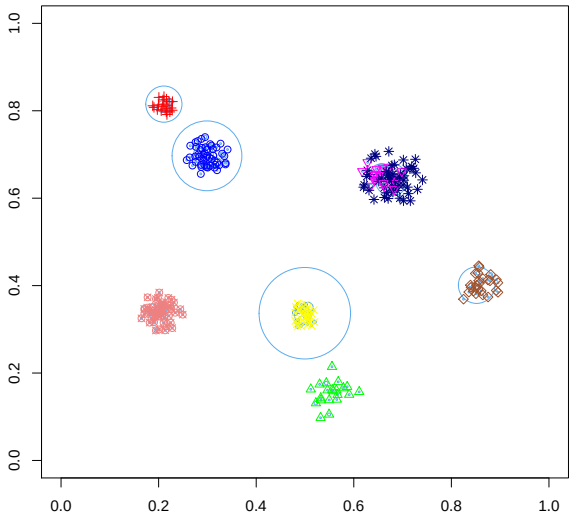


(e) Grau de paralelismo: 16

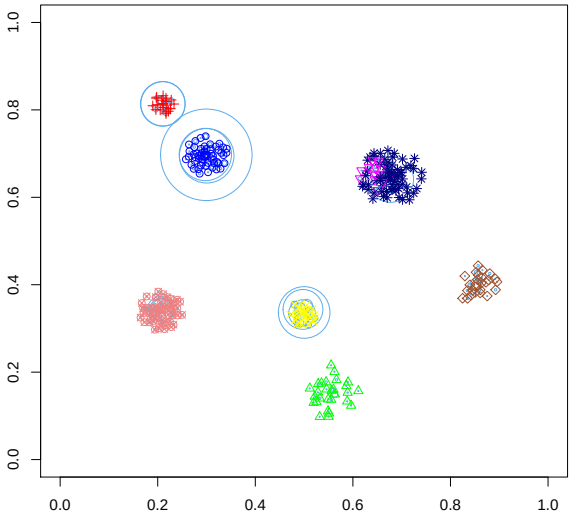
Figura 7.6: Agrupamento final da observação 2 do RBF5_1M



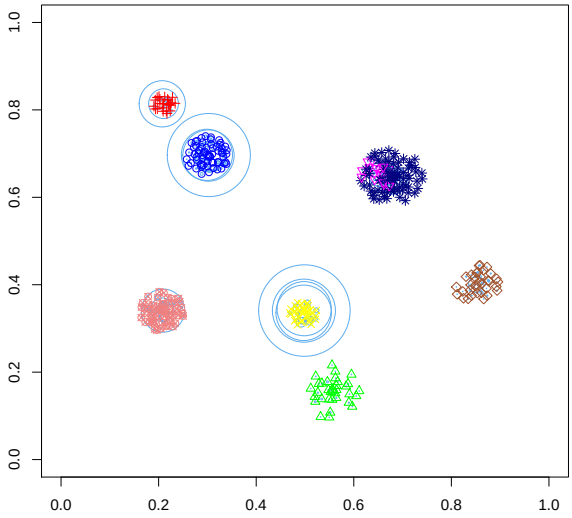
(a) Grau de paralelismo: 1



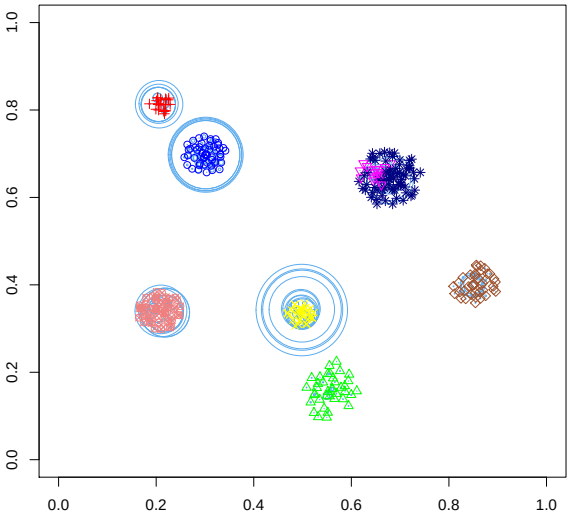
(b) Grau de paralelismo: 2



(c) Grau de paralelismo: 4

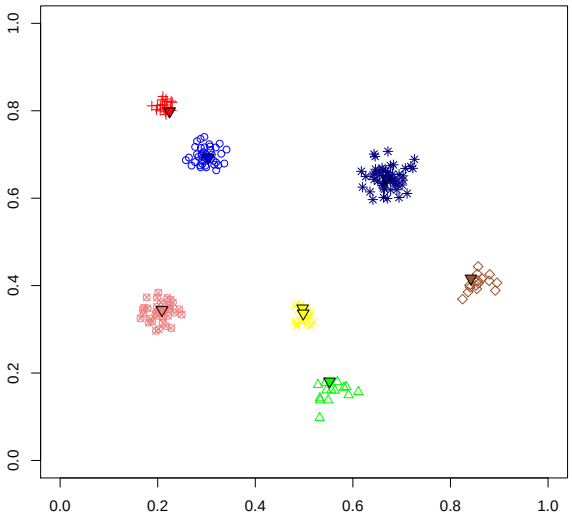


(d) Grau de paralelismo: 8

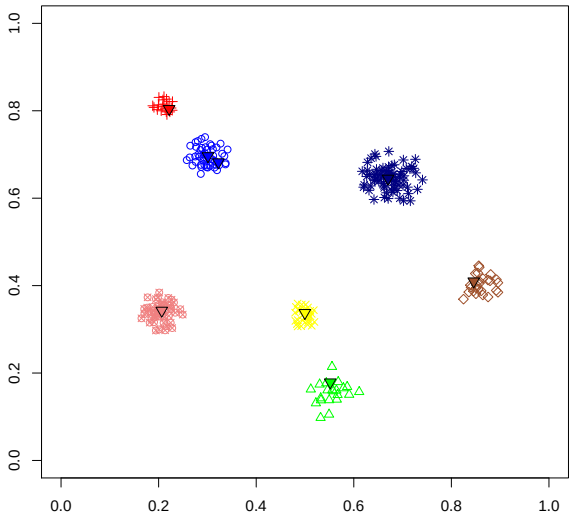


(e) Grau de paralelismo: 16

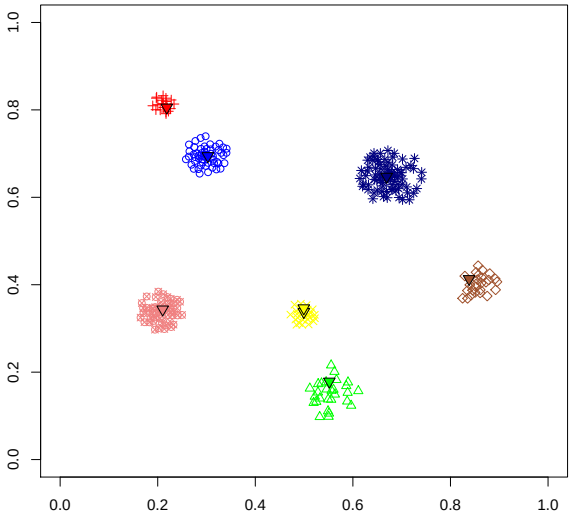
Figura 7.7: Estruturas de sumarização da observação 3 do RBF5_1M



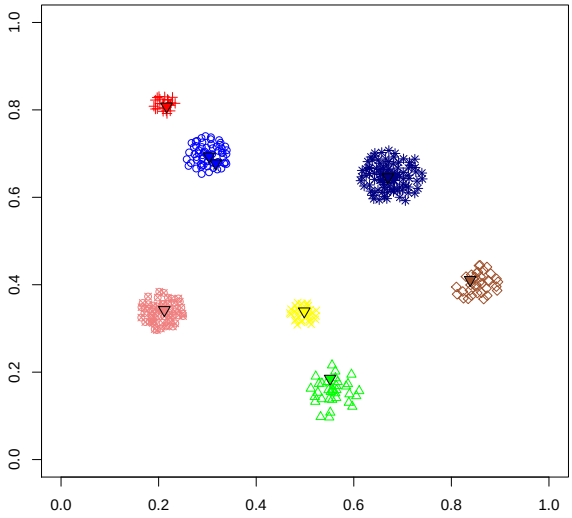
(a) Grau de paralelismo: 1



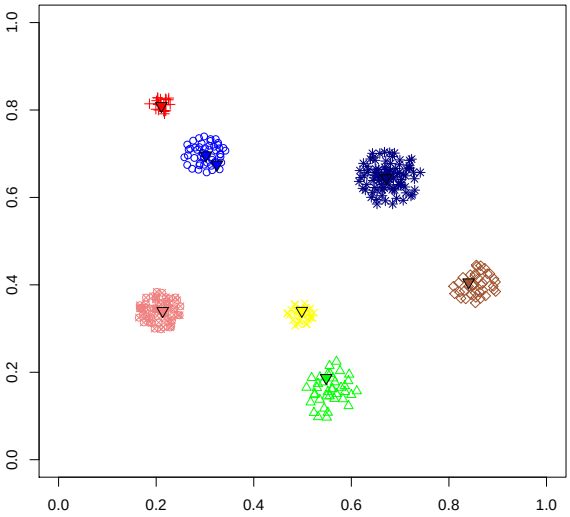
(b) Grau de paralelismo: 2



(c) Grau de paralelismo: 4

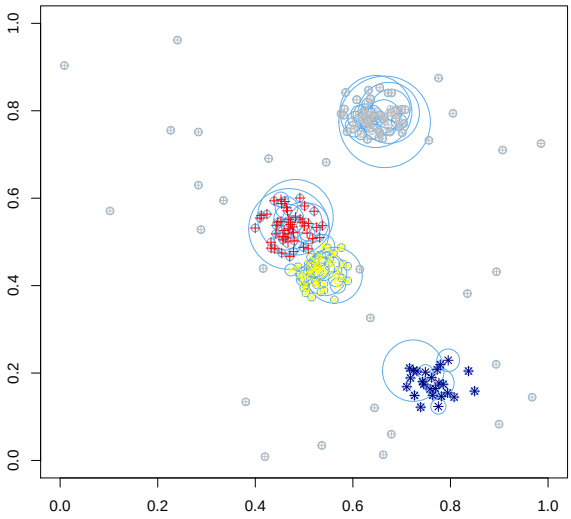


(d) Grau de paralelismo: 8

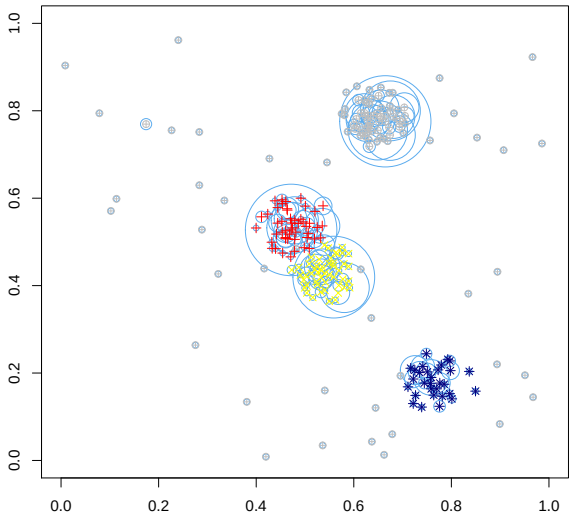


(e) Grau de paralelismo: 16

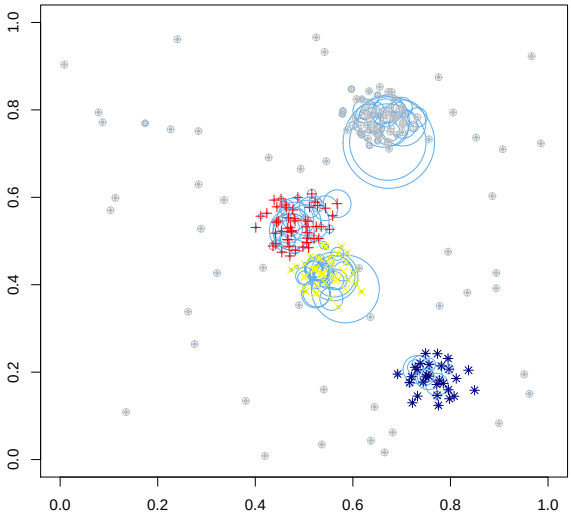
Figura 7.8: Agrupamento final da observação 3 do RBF5_1M



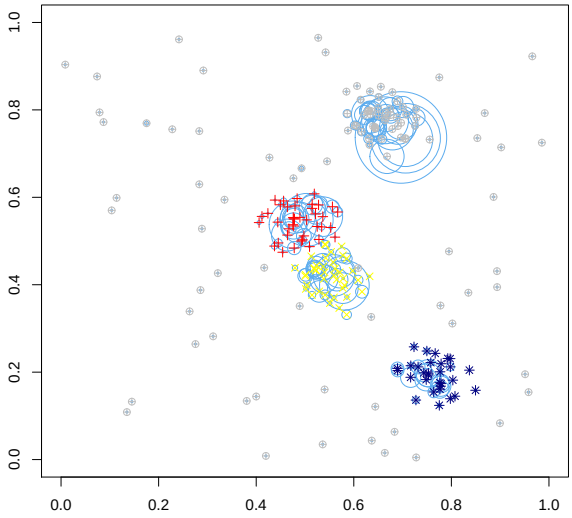
(a) Grau de paralelismo: 1



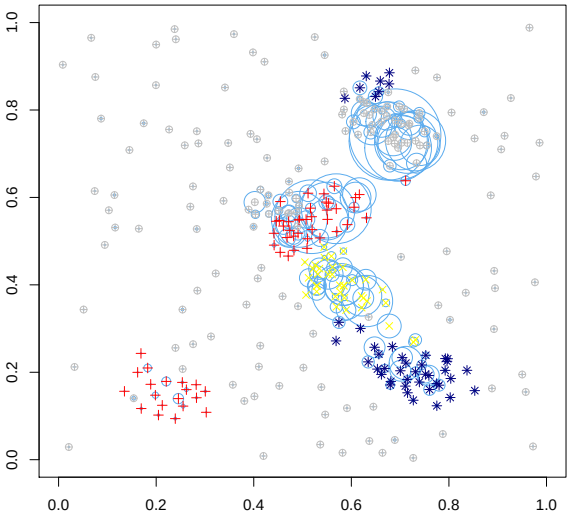
(b) Grau de paralelismo: 2



(c) Grau de paralelismo: 4

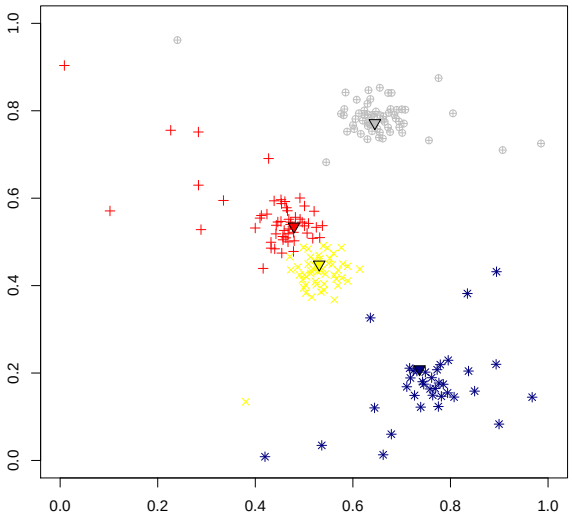


(d) Grau de paralelismo: 8

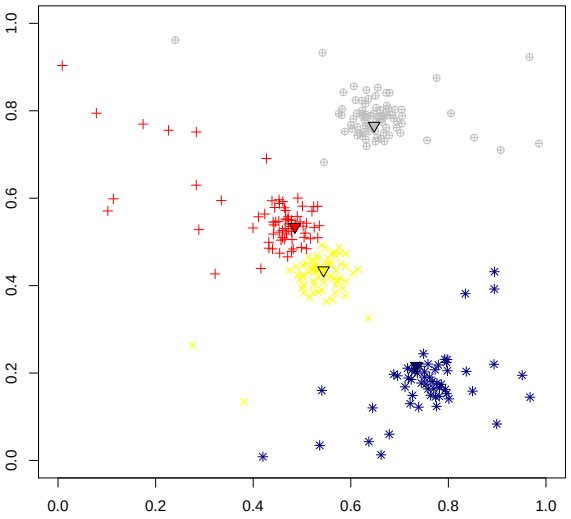


(e) Grau de paralelismo: 16

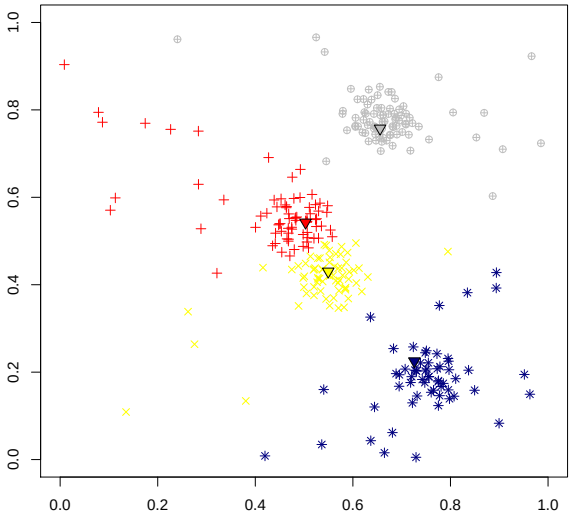
Figura 7.9: Estruturas de sumarização da observação 7 do RBF6_400K



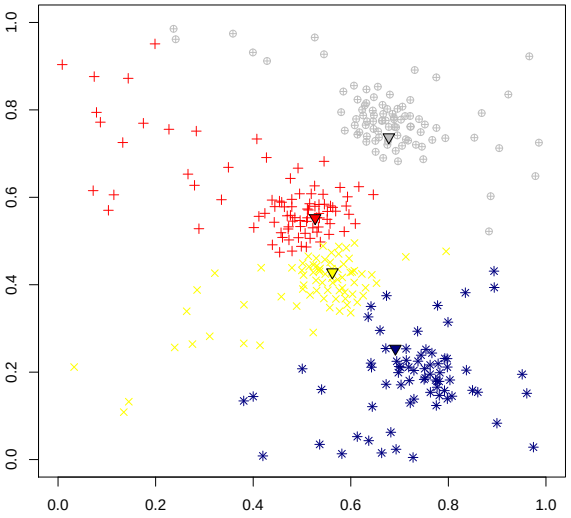
(a) Grau de paralelismo: 1



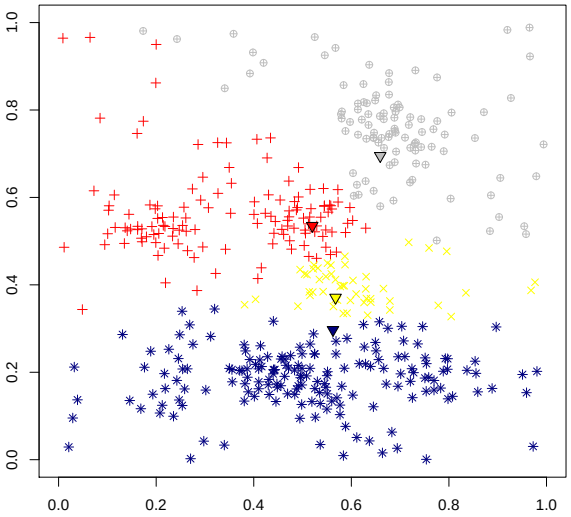
(b) Grau de paralelismo: 2



(c) Grau de paralelismo: 4

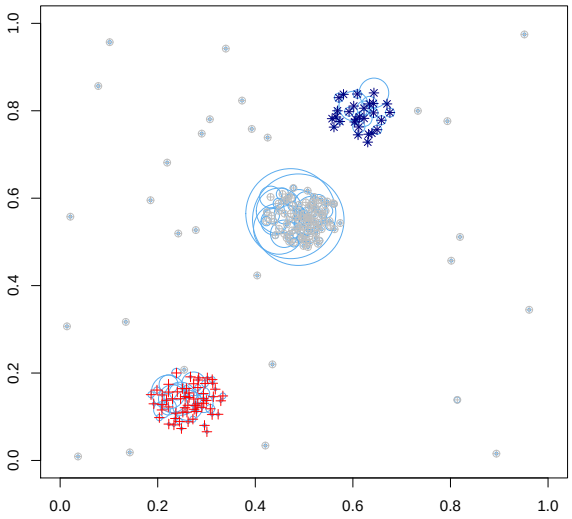


(d) Grau de paralelismo: 8

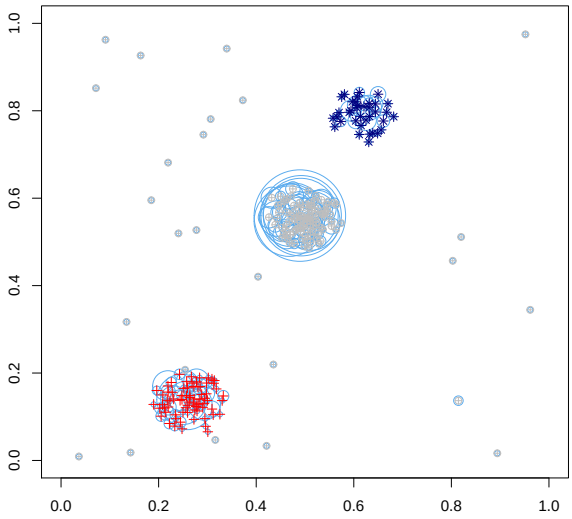


(e) Grau de paralelismo: 16

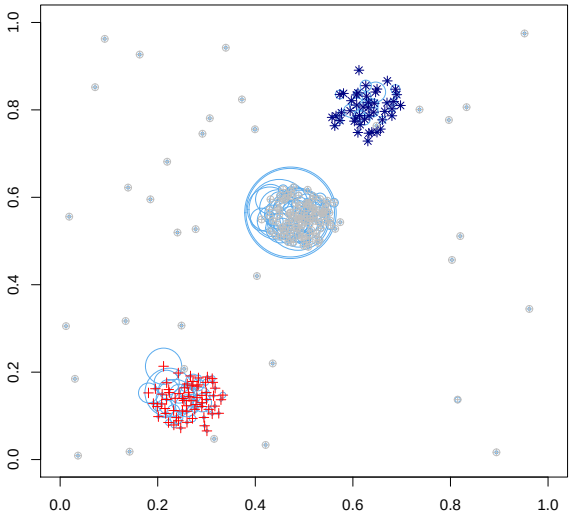
Figura 7.10: Agrupamento final da observação 7 do RBF6_400K



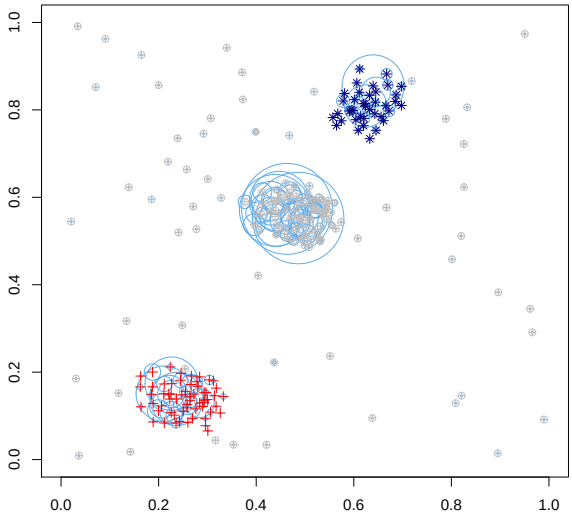
(a) Grau de paralelismo: 1



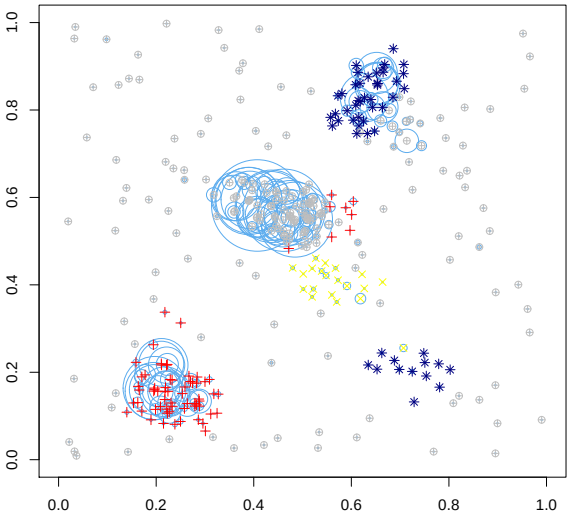
(b) Grau de paralelismo: 2



(c) Grau de paralelismo: 4

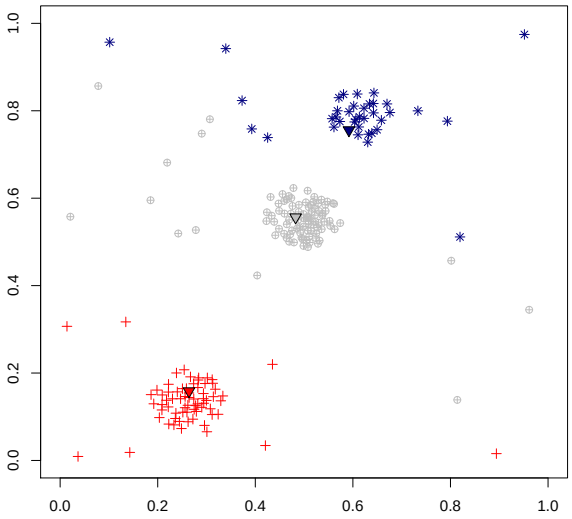


(d) Grau de paralelismo: 8

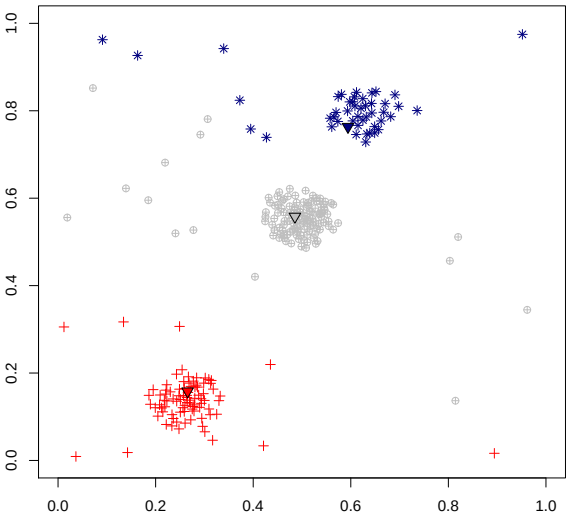


(e) Grau de paralelismo: 16

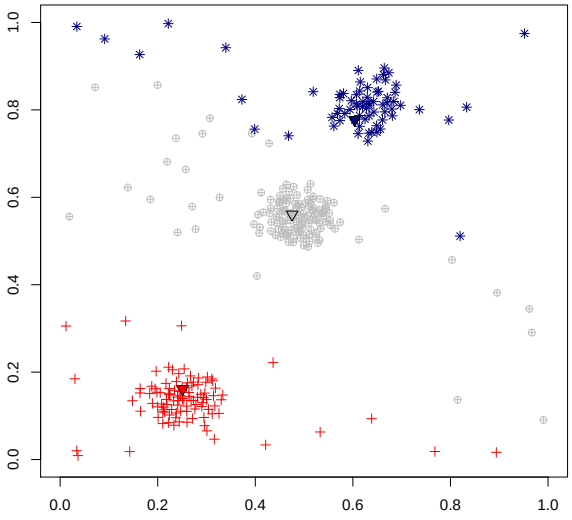
Figura 7.11: Estruturas de sumarização da observação 8 do RBF6_400K



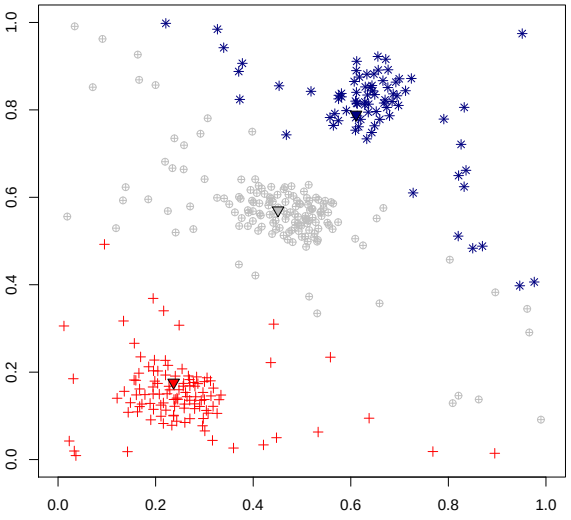
(a) Grau de paralelismo: 1



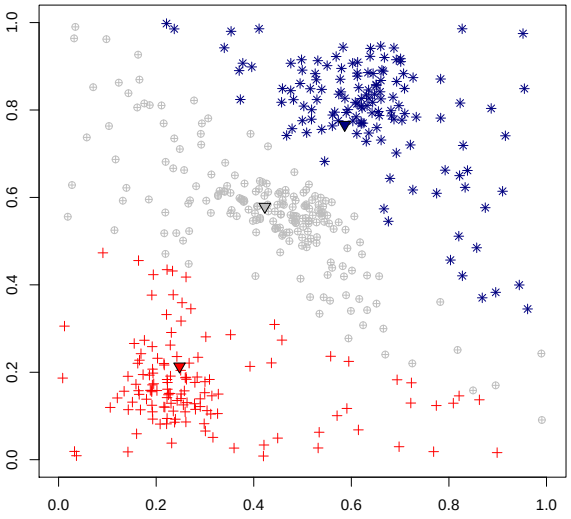
(b) Grau de paralelismo: 2



(c) Grau de paralelismo: 4



(d) Grau de paralelismo: 8



(e) Grau de paralelismo: 16

Figura 7.12: Agrupamento final da observação 8 do RBF6_400K

A Tabela 7.1 mostra a média da quantidade de FMiCs contidos nas estruturas de sumarização das dez observações de cada base de dados em cada grau de paralelismo. Em todas as bases o número de FMiCs aumentou progressivamente mas, com exceção da base SynEDC, esse aumento não acompanhou o crescimento do tamanho máximo da estrutura.

	Sequencial	**2 graus**	**4 graus**	**8 graus**	**16 graus**
BG_1M	199.6	261.6	311.9	346.6	380.2
RBF5_1M	199.5	283.6	313.4	356.6	391.1
RBF6_400K	197.4	227.8	226.8	233.6	328.3
SynEDC	200	300	448	672	1008
Covertype	197.5	213.4	241.7	274.9	389.3
KDD99Cup	176.7	276.9	354.8	408.9	452.9

Tabela 7.1: Média do tamanho das estruturas de sumarização das dez observações com diferentes graus de paralelismo

Um maior número de FMiCs permite com que a estrutura de sumarização consiga representar uma maior quantidade de exemplos provenientes do FD. Entretanto, como pôde ser constatado nos agrupamentos da base RBF6_400K, uma maior representação do FD nem sempre é benéfica, uma vez que dados que não representam mais o estado do fluxo podem ser contemplados.

O mecanismo de esquecimento no algoritmo de sumarização, apesar de eficiente, poderia ser melhorado para reduzir o crescimento da tamanho da estrutura de sumarização. Uma sugestão seria a introdução de um mecanismo de decaimento, onde os FMiCs perdem sua expressividade a medida que não recebem novos exemplos.

7.2 Métricas de Agrupamento

Os resultados das métricas de agrupamento foram divididos entre métricas internas e externas. Os valores apresentados nas figuras a seguir representam a média do valor de cada uma das métricas nos 50 agrupamentos realizados por cada agrupador. Além disso, cada ponto no eixo X possui uma reta para cima e para baixo representando o desvio padrão da métrica em relação à média.

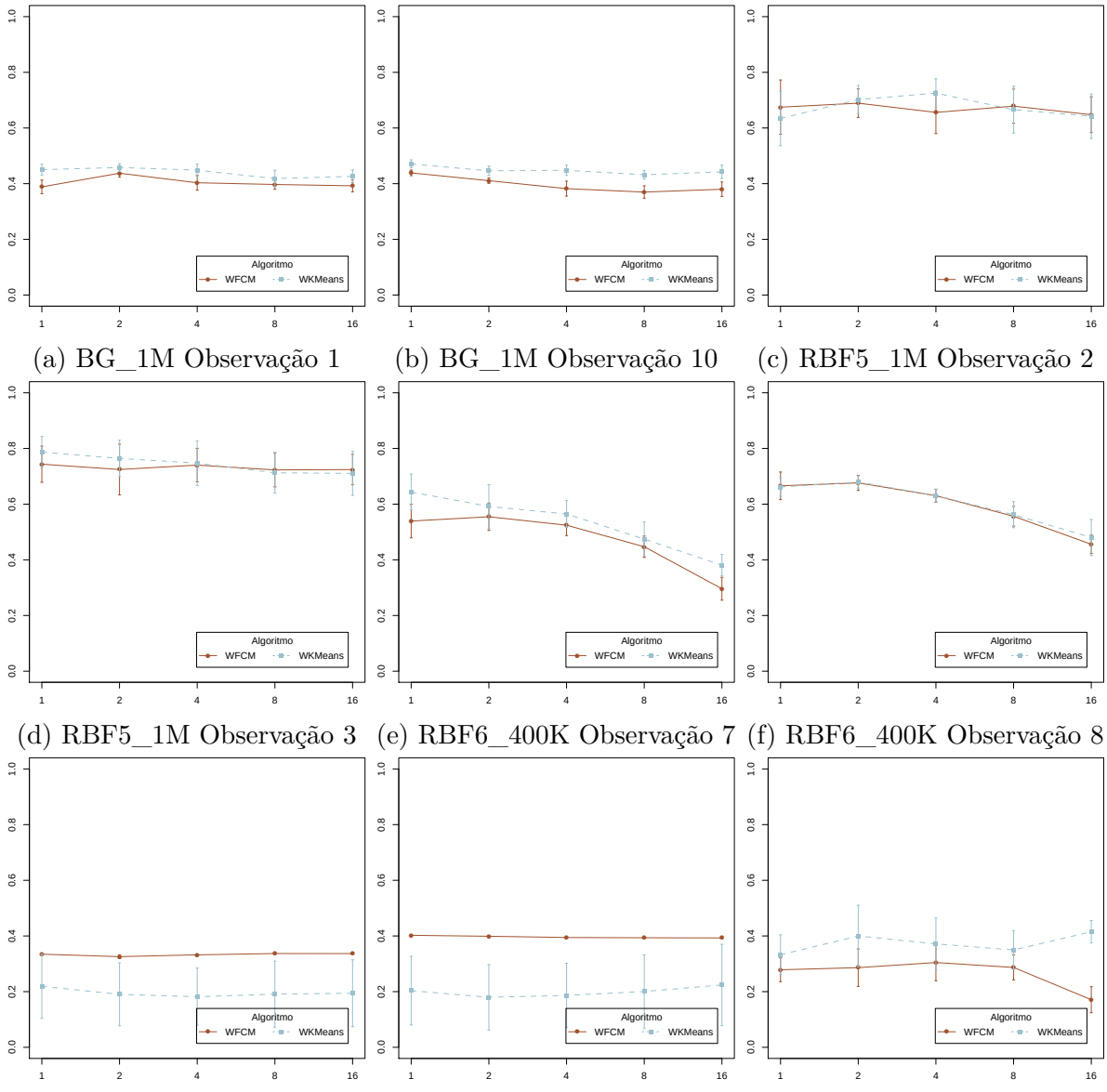
São apresentados os resultados das métricas relativas aos agrupamentos com o número de grupos igual ao número de classes presented na estrutura de sumarização observada.

7.2.1 Métricas Internas

Os gráficos nas figuras 7.13, 7.14 e 7.15 ilustram os valores de silhueta, silhueta *fuzzy* e XieBeni, respectivamente, onde o eixo Y representa a média da métrica dos 50

agrupamentos, e o eixo X representa o paralelismo da sumarização. Cada um dos pontos em X possui uma reta que representa o desvio padrão. Como a métrica XieBeni não possui valor máximo, os gráficos que a representam possuem escalas diferentes para facilitar a visualização.

Nos resultados de silhueta podem ser observados principalmente dois comportamentos com o aumento do paralelismo da sumarização local. Nas bases BG_1M, RBF5_1M e SynEDC, os valores de silhueta se mantêm semelhantes e nas bases RBF6_400K, Covertype e KDD99Cup é possível observar uma queda gradual com o aumento de sumarizações locais.

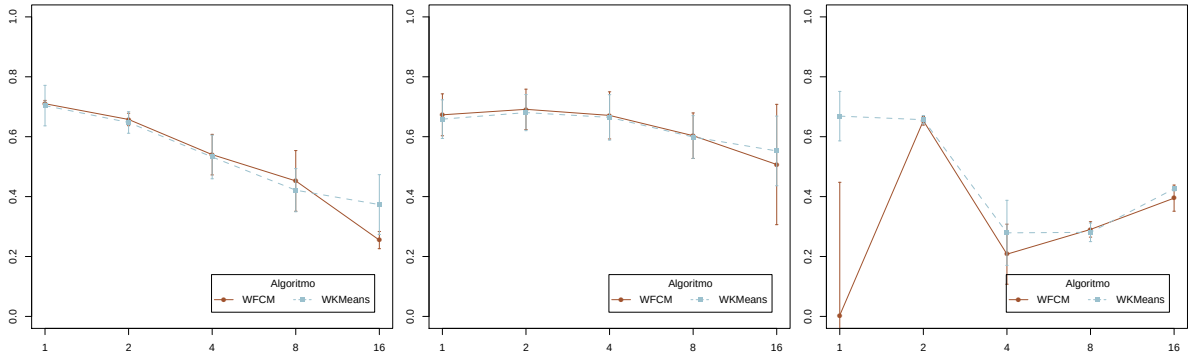


(a) BG_1M Observação 1 (b) BG_1M Observação 10 (c) RBF5_1M Observação 2

(d) RBF5_1M Observação 3 (e) RBF6_400K Observação 7 (f) RBF6_400K Observação 8

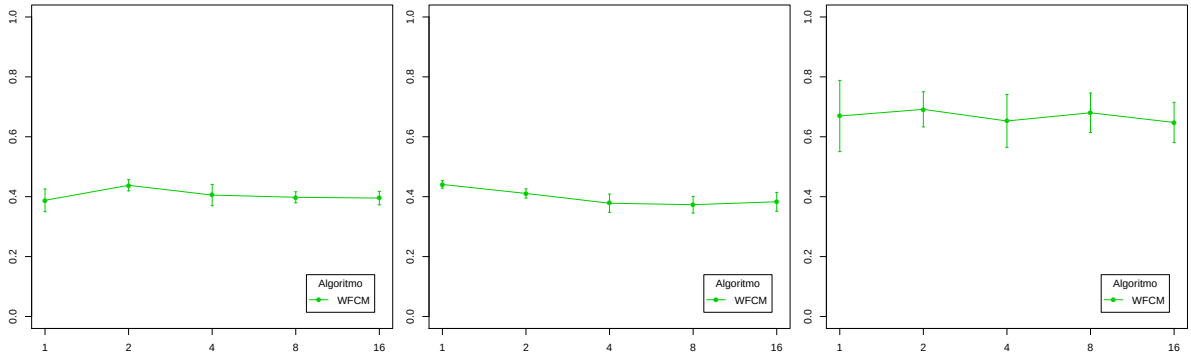
(g) SynEDC Observação 6 (h) SynEDC Observação 9 (i) Covertype Observação 5

Figura 7.13: Valores de Silhueta em relação ao grau de paralelismo

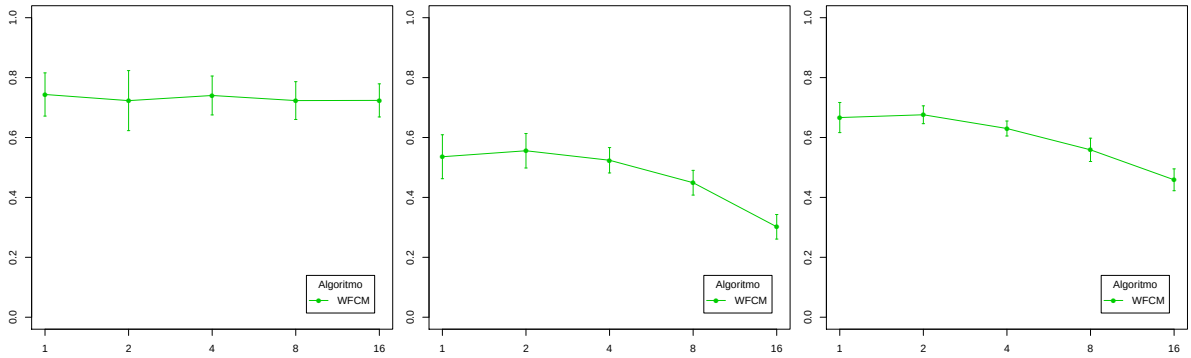


(j) Covertype Observação 10 (k) KDD99Cup Observação 1 (l) KDD99Cup Observação 3

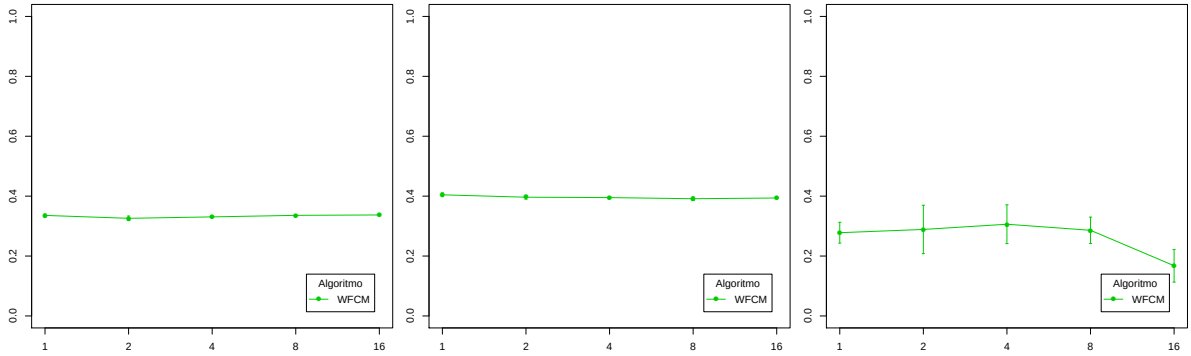
Figura 7.13: Valores de Silhueta em relação ao grau de paralelismo (cont.)



(a) BG_1M Observação 1 (b) BG_1M Observação 10 (c) RBF5_1M Observação 2

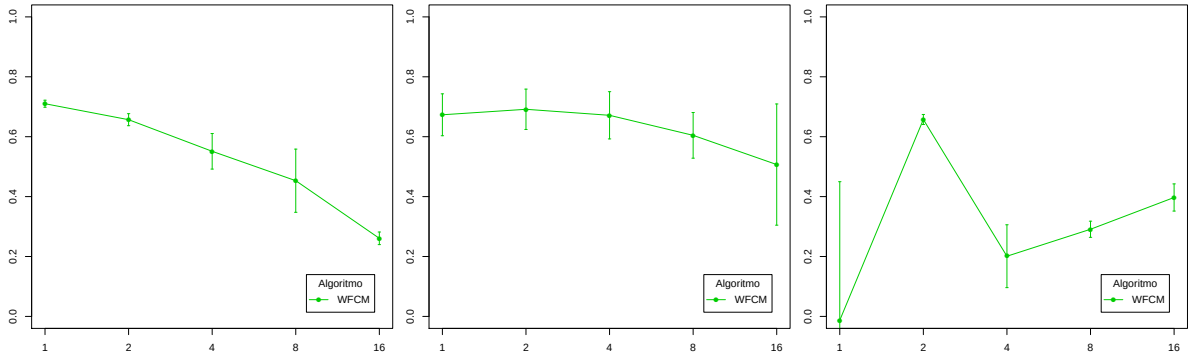


(d) RBF5_1M Observação 3 (e) RBF6_400K Observação 7 (f) RBF6_400K Observação 8



(g) SynEDC Observação 6 (h) SynEDC Observação 9 (i) Covertype Observação 5

Figura 7.14: Valores de Silhueta Fuzzy em relação ao grau de paralelismo



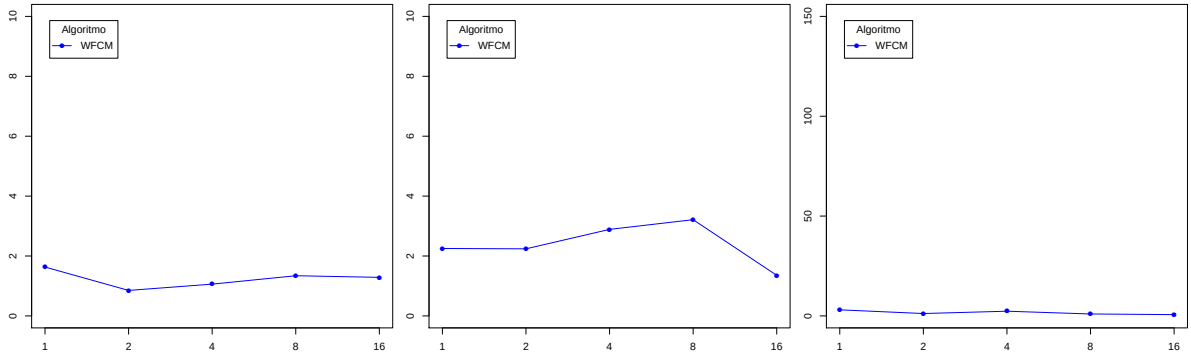
(j) Covertype Observação 10 (k) KDD99Cup Observação 1 (l) KDD99Cup Observação 3

Figura 7.14: Valores de Silhueta Fuzzy em relação ao grau de paralelismo (cont.)

Uma das principais causas dessa queda é a sensibilidade da métrica a ruídos presentes na base de dados. O aumento da estrutura de sumarização final permitiu a criação de mais FMiCs que representavam ruídos, o que afeta as métricas internas de agrupamento utilizadas, uma vez que elas não fazem distinção entre os tipos de exemplos.

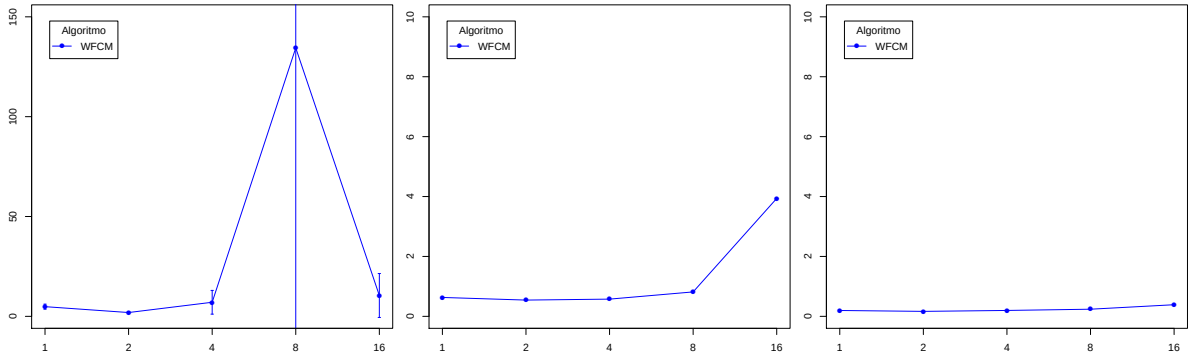
Nos resultados de XieBeni é possível constatar uma estabilidade em relação à métrica em bases de dados BG_1M, RBF6_400K, Covertype e KDD99Cup. Também podem ser observados alguns comportamentos atípicos em algumas observações, como ocorre nas figuras 7.15d, 7.15g, 7.15h e 7.15j. O valor da métrica de XieBeni se mostrou bastante sensível à- inicialização dos centróides o que gerou uma variação discrepante entre os 50 agrupamentos, constatado pelo alto desvio padrão.

É possível inferir que o uso da abordagem distribuída pode ter um impacto negativo na qualidade do agrupamento, principalmente em bases com ruído. Apesar disso, esse impacto não comprometeu o agrupamento.

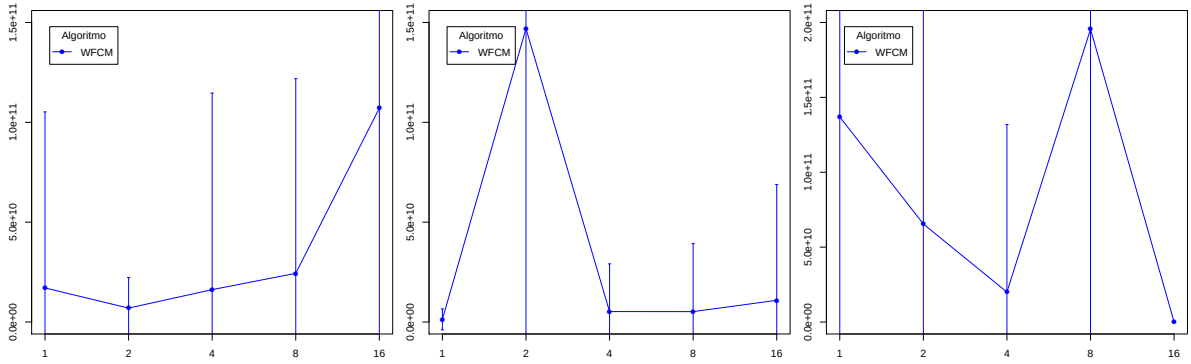


(a) BG_1M Observação 1 (b) BG_1M Observação 10 (c) RBF5_1M Observação 2

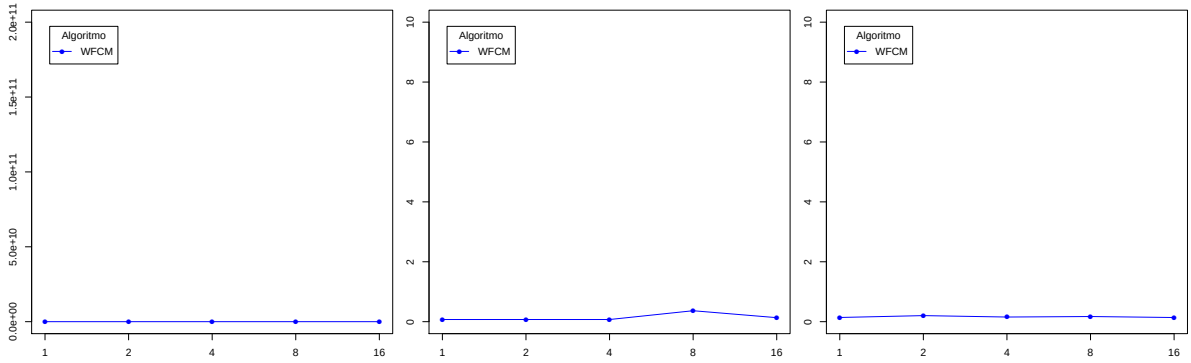
Figura 7.15: Valores de XieBeni em relação ao grau de paralelismo



(d) RBF5_1M Observação 3 (e) RBF6_400K Observação 7 (f) RBF6_400K Observação 8



(g) SynEDC Observação 6 (h) SynEDC Observação 9 (i) Covertype Observação 5



(j) Covertype Observação 10 (k) KDD99Cup Observação 1 (l) KDD99Cup Observação 3

Figura 7.15: Valores de XieBeni em relação ao grau de paralelismo (cont.)

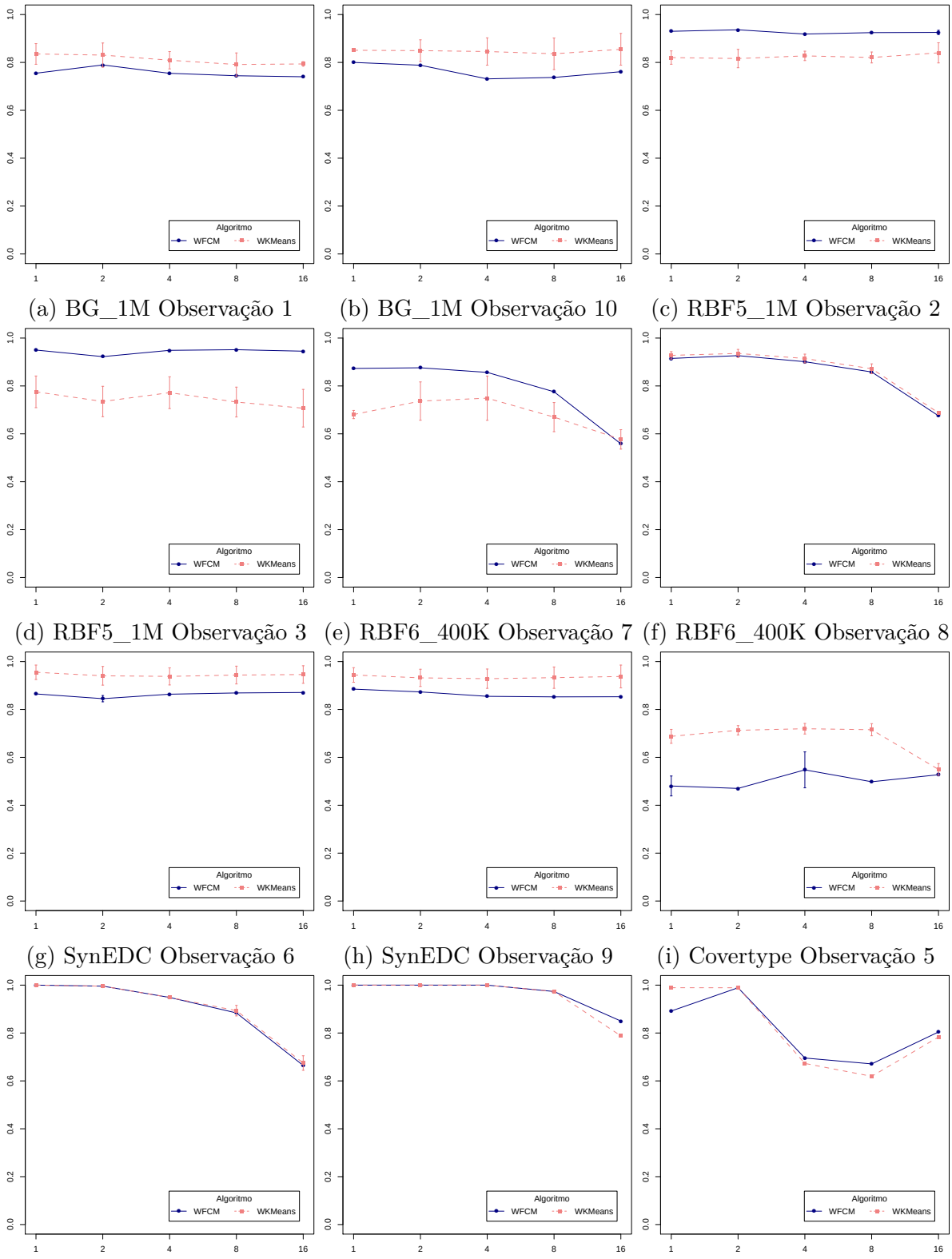
7.2.2 Métricas Externas

As métricas externas necessitam de informações prévias de classe de cada um dos exemplos. Como todas as bases de dados utilizadas possuiam informações de classe, um FMiC foi definido com a classe dos exemplos com maior pertinência a ele.

Os gráficos nas Figuras 7.16 e 7.17 apresentam os resultados de pureza e índice Rand, onde o eixo Y representa a média do valor das métricas, e o eixo X representa o paralelismo da sumarização.

Os resultados de pureza possuem comportamentos parecidos com os resultados de silhueta. As bases BG_1M, RBF5_1M e SynEDC mantiveram os valores de pureza estáveis, enquanto que nas outras bases houve uma queda gradual dos valores. Já os

índices de Rand mantiveram um mesmo padrão com o aumento do paralelismo.



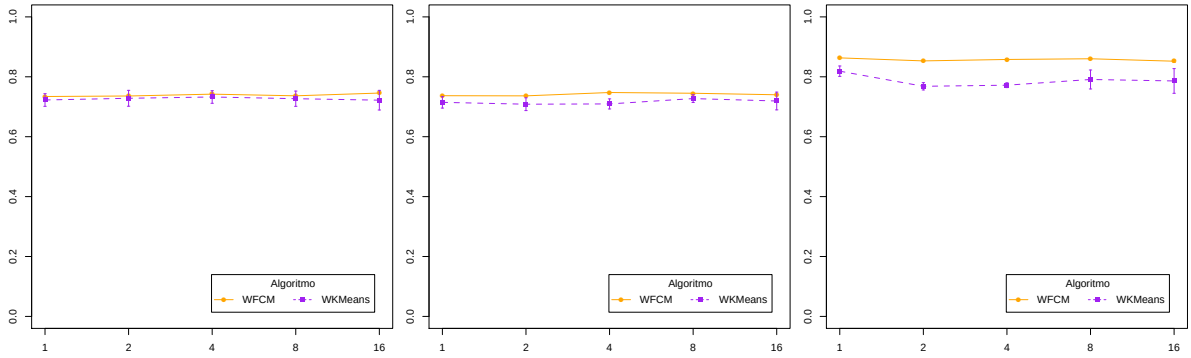
(a) BG_1M Observação 1 (b) BG_1M Observação 10 (c) RBF5_1M Observação 2

(d) RBF5_1M Observação 3 (e) RBF6_400K Observação 7 (f) RBF6_400K Observação 8

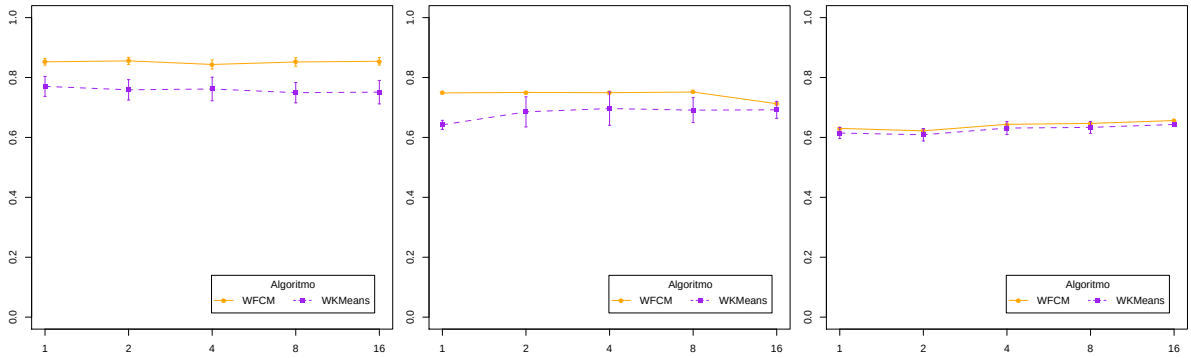
(g) SynEDC Observação 6 (h) SynEDC Observação 9 (i) Covertype Observação 5

(j) Covertype Observação 10 (k) KDD99Cup Observação 1 (l) KDD99Cup Observação 3

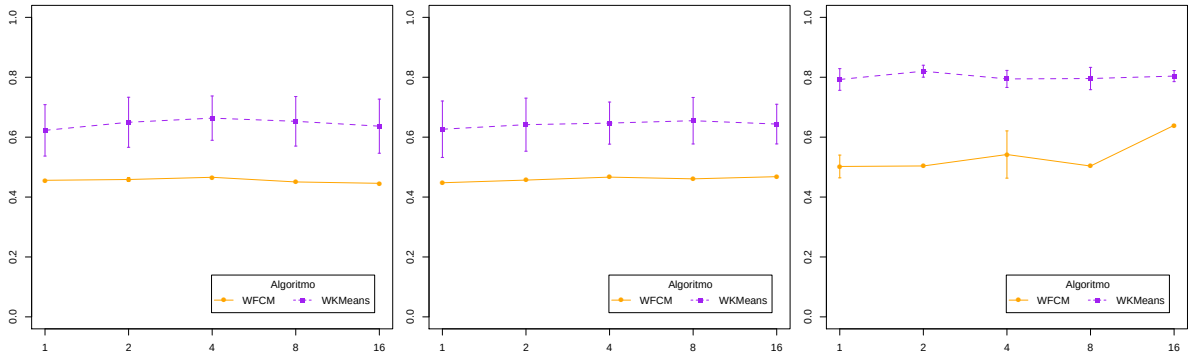
Figura 7.16: Valores de Pureza em relação ao grau de paralelismo



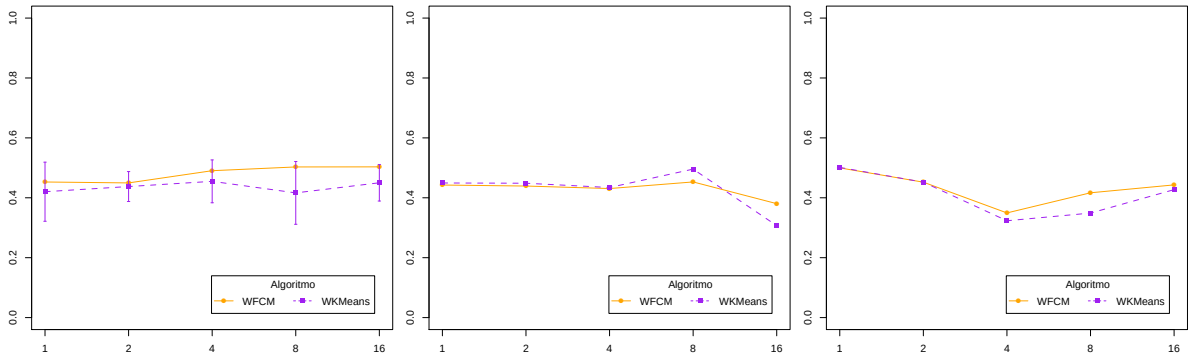
(a) BG_1M Observação 1 (b) BG_1M Observação 10 (c) RBF5_1M Observação 2



(d) RBF5_1M Observação 3 (e) RBF6_400K Observação 7 (f) RBF6_400K Observação 8



(g) SynEDC Observação 6 (h) SynEDC Observação 9 (i) Covertype Observação 5



(j) Covertype Observação 10 (k) KDD99Cup Observação 1 (l) KDD99Cup Observação 3

Figura 7.17: Valores de índice de Rand em relação ao grau de paralelismo

7.3 Métricas de Desempenho

As métricas de desempenho foram calculadas separadamente para as estratégias de sumarização local e global e de agrupamento distribuído. Os tempos de execução foram calculados pela própria ferramenta Apache Flink e compreendem desde a inicialização até a finalização do processo.

7.3.1 Desempenho da estratégia de sumarização local e global

Como pode ser observado na Tabela 7.2, os tempos de execução caíram de forma significativa com o aumento do grau de paralelismo. A mudança mais notável foi na base de KDD99Cup que passou de mais de duas horas sumarizada de forma sequencial, para apenas sete minutos com dezesseis graus de paralelismo. A Figura 7.18 apresenta os resultados de *speedup* e a Figura 7.19 os resultados de eficiência para todas as bases de dados. O eixo X representa o grau de paralelismo e o eixo Y representa o valor da métrica.

BG_1M		
Grau	**Tempo em ms**	**Tempo**
1	760268	12m40s
2	257551	4m17s
4	94648	1m34s
8	53441	0m53s
16	40154	0m40s

RBF5_1M		
Grau	**Tempo em ms**	**Tempo**
1	1028341	17m8s
2	368119	6m8s
4	133285	2m13s
8	81086	1m21s
16	61540	1m1s

RBF6_400K		
Grau	**Tempo em ms**	**Tempo**
1	428454	7m8s
2	150521	2m30s
4	52662	0m52s
8	28949	0m28s
16	20013	0m20s

SynEDC		
Grau	**Tempo em ms**	**Tempo**
1	374109	6m14s
2	115516	1m55s
4	36916	0m36s
8	18771	0m18s
16	13199	0m13s

Covertype		
Grau	**Tempo em ms**	**Tempo**
1	1763790	29m23s
2	715302	11m55s
4	292784	4m52s
8	173538	2m53s
16	104648	1m44s

KDD99Cup		
Grau	**Tempo em ms**	**Tempo**
1	7343618	2h2m23s
2	2522484	42m2s
4	938617	15m38s
8	543659	9m3s
16	427693	7m7s

Tabela 7.2: Tempos de execução da estratégia de sumarização local e global considerando diferentes graus de paralelismo

Por meio dos resultados é possível notar que o tempo de execução foi diminuído em mais da metade em alguns casos, o que caracteriza o crescimento quase exponencial do *speedup*. Este comportamento conhecido como *super-linear speedup* ocorre porque o ganho

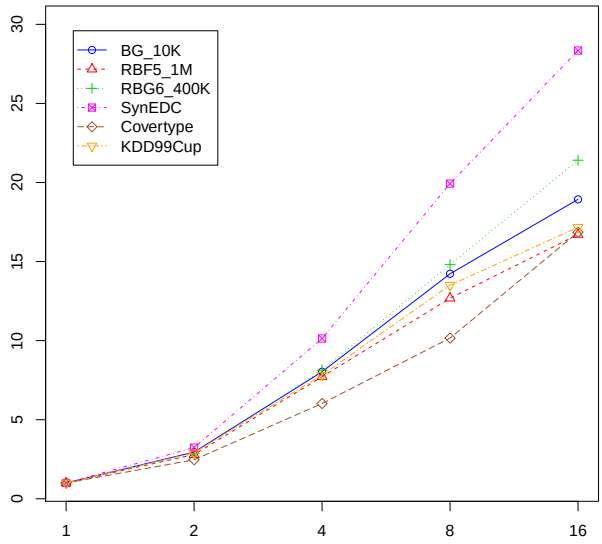


Figura 7.18: *Speedup* da estratégia de sumarização local e global

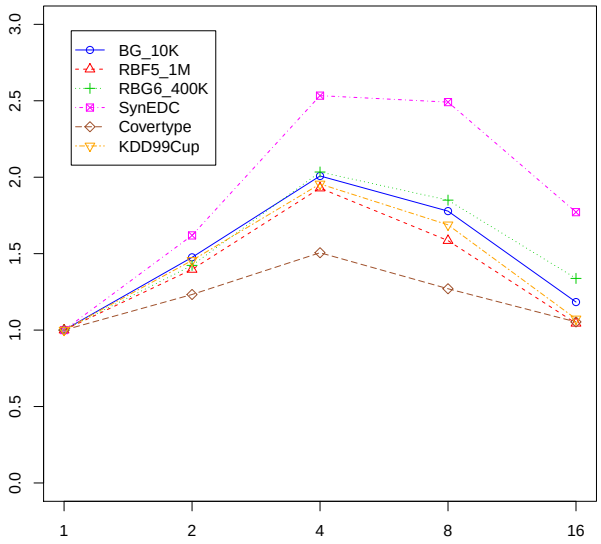


Figura 7.19: Eficiência da estratégia de sumarização local e global

de desempenho não está apenas vinculado ao processamento distribuído, mas também a diminuição dos FMiCs em cada sumarização local, que tornou o processo de sumarização mais rápido.

Por outro lado, a estratégia de sumarização local e global introduziu uma nova fase intermediária. O aumento do número de sumarizações locais fez com que a fase intermediária demandasse mais tempo para unificar todas as estruturas locais na estrutura final. Isso, aliado ao fato de que a quantidade de FMiCs em cada sumarização local decresce cada vez menos, explica o decrescimento da eficiência da estratégia com mais de quatro graus de paralelismo e o menor crescimento do *speedup* ao longo do tempo.

Mesmo apresentando maior eficiência com quatro graus de paralelismo, a estratégia de sumarização local e global ainda se mostra relevante, reduzindo significativamente o tempo necessário para sumarizar o FD.

7.3.2 Desempenho da estratégia de agrupamento distribuído

Na fase *offline*, cada observação é agrupada separadamente, e não de forma contínua. Assim, os resultados dos tempos de execução apresentados foram somados para todas as observações. Além disso, os resultados de desempenho aqui apresentados são relativos a parte *offline* aplicada a estrutura de sumarização gerada pela sumarização com dezesseis graus de paralelismo, pois contém maior número de FMiCs.

A Tabela 7.3 apresenta os tempos de execução enquanto que as Figuras 7.20 e 7.21 ilustram os resultados de *speedup* e eficiência (eixo X representa o grau de paralelismo e o eixo Y representa o valor da métrica). Observa-se que a base BG_1M não foi agrupada com mais de dois graus de paralelismo devido ao seu baixo número de grupos.

BG_1M		
Grau	**Tempo em ms**	**Tempo**
1	5109	0m5s
2	3402	0m3s
4	X	X
8	X	X
16	X	X

RBF5_1M		
Grau	**Tempo em ms**	**Tempo**
1	32330	0m32s
2	21163	0m21s
4	14462	0m14s
8	12522	0m12s
16	11906	0m11s

RBF6_400K		
Grau	**Tempo em ms**	**Tempo**
1	34135	0m34s
2	23077	0m23s
4	16638	0m16s
8	14080	0m14s
16	12501	0m12s

SynEDC		
Grau	**Tempo em ms**	**Tempo**
1	1692283	28m12s
2	849870	14m9s
4	451690	7m31s
8	323459	5m23s
16	334757	5m34s

Covertype		
Grau	**Tempo em ms**	**Tempo**
1	156421	2m36s
2	108514	1m48s
4	61917	1m1s
8	48557	0m48s
16	41777	0m41s

KDD99Cup		
Grau	**Tempo em ms**	**Tempo**
1	897989	14m57s
2	653788	10m53s
4	344499	5m44s
8	223751	3m43s
16	193397	3m13s

Tabela 7.3: Tempos de execução da estratégia de agrupamento distribuído considerando diferentes graus de paralelismo

É notável que o ganho de desempenho do agrupamento distribuído é menor que o da sumarização. Um ponto a se considerar é que a fase *offline* agrupa a estrutura de sumarização, que já é uma redução do fluxo de dados. Assim, esta fase não lida com um grande volume de informações, o que justifica um menor ganho de desempenho. Acredita-se que utilizar essa estratégia teria maior ganho caso um número maior de algoritmos de agrupamento fossem utilizados.

Como pode ser observado nos resultados, a estratégia se mostrou pouco eficiente, sendo o melhor caso com quatro níveis de paralelismo. Isso significa que aumentar o grau de paralelismo após esse ponto, gastaria mais recursos computacionais com cada vez menores ou até nenhum ganho de desempenho.

Ainda assim, se forem comparados os resultados das duas estratégias da abordagem distribuída, é possível notar que ambas possuem tempos de execução compatíveis. Isso demonstra a importância do agrupamento distribuído pois somente utilizando as duas estratégias é possível gerar o resultado do agrupamento de um FD em tempo real.

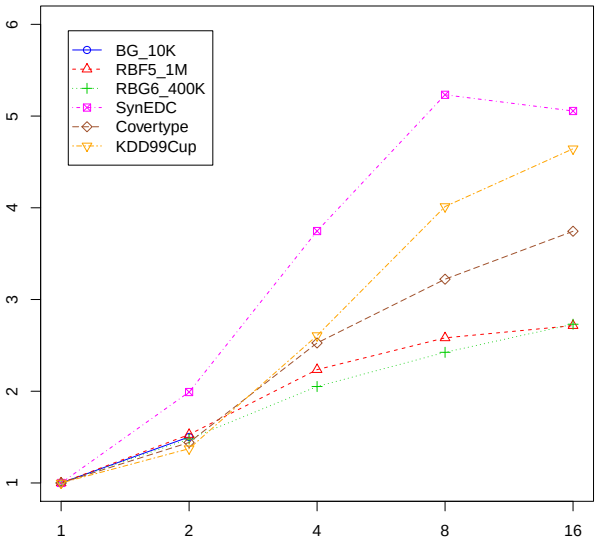


Figura 7.20: *Speedup* da estratégia de agrupamento distribuído

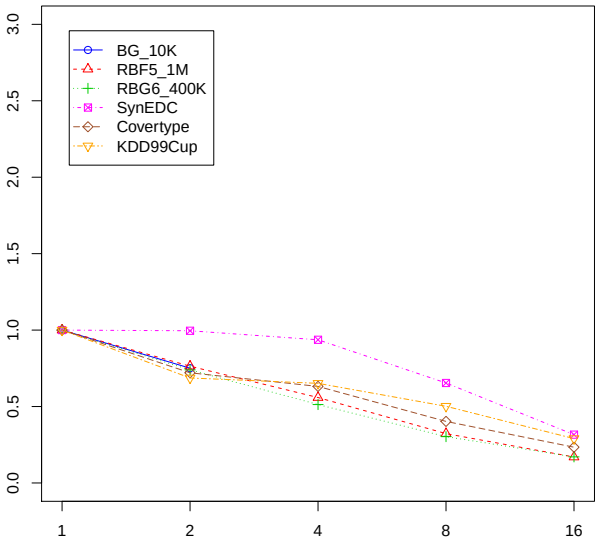


Figura 7.21: Eficiência da estratégia de agrupamento distribuído

7.4 Considerações Finais

O corpo de resultados comentados neste capítulo configura um subconjunto do total obtido pela execução dos 135 experimentos aos 6 conjuntos de exemplos selecionados. Apesar do conjunto reduzido de resultados apresentados neste capítulo, destaques importantes são expostos. Esses destaques dizem respeito a um levantamento de características de comportamento do aumento do grau de paralelismo em relação à qualidade do agrupamento e ao desempenho.

Ressalta-se que a comparação dos resultados das métricas de agrupamento é feita a partir de estruturas de dados diferentes, uma vez que para cada grau de paralelismo a estrutura de sumarização gerada é diferente. Mesmo assim, os resultados dessas métricas mostram que existe um padrão com o uso da estratégia distribuída. É perceptível que o número de FMiCs na estrutura de sumarização cresceu de acordo com o aumento do grau de paralelismo, o que já era esperado.

Esse aumento do número de FMiCs alterou drasticamente o agrupamento da base RBF6_400K com dezesseis graus de paralelismo. Alguns FMiCs passaram a representar dados antigos e que não pertenciam mais ao fluxo de dados. Isso poderia ser evitado, caso a sumarização do d-FuzzStream tivesse um melhor mecanismo de esquecimento.

Em relação às métricas internas e externas, o maior tamanho da estrutura de sumarização não afetou a qualidade do agrupamento em bases sem ruídos, mas impactou negativamente outras bases devido, principalmente, ao maior número de FMiCs que representam ruídos. Apesar disso, a magnitude desse impacto não foi grande o bastante para comprometer o agrupamento.

Por fim, a desempenho da abordagem distribuída trouxe ganhos significativos em

termos de desempenho. Na estratégia de sumarização local e global reduziu até 95% do tempo de execução, diminuido de horas para poucos minutos em alguns casos. Já a estratégia de agrupamento distribuído teve menor ganho em tempo de execução no geral. Mesmo assim, esse ganho permitiu gerar agrupamentos em tempos iguais ou até menores que os tempos da sumarização.

Conclusão

Era comum que as análises dos dados fossem realizadas algumas horas ou dias após a geração das informações. Com o aparecimento cada vez mais expressivo de fontes de dados no formato de Fluxos de Dados, cresceu também a necessidade de entregar análises e predições em tempo real. Assim, estratégias para extração de conhecimento a partir desses fluxos são uma tendência dentro das pesquisas na área de Aprendizado de Máquina.

As características imprevisíveis dos domínios de Fluxo de Dados encorajam a busca por aprendizado flexível, por exemplo, pela integração de conceitos da teoria de conjuntos *fuzzy*. Propostas baseadas em conceitos *fuzzy* têm o objetivo de colaborar para flexibilidade e adaptabilidade do conhecimento aprendido.

Entretanto, com a produção constante e cada vez maior de dados, é comum que algoritmos de aprendizado em fluxo de dados apresentem dificuldades em gerar conhecimento em tempo real. Para atingir esse objetivo, algoritmos recentes encontrados na literatura combinam o aprendizado de máquina e a computação distribuída em detrimento de métodos de amostragem.

Dentro de aprendizado por agrupamento em Fluxo de Dados, a maioria das propostas que utilizam a programação distribuída são baseadas em algoritmos *crisp* que dividem o FD em partes (*chunks*) e aplicam o agrupamento desejado em cada parte. Essa técnica resulta na redução do Fluxo de Dados a um arranjo de protótipos.

Contribuições

Este trabalho apresenta um novo algoritmo e uma nova abordagem distribuída para agrupamento *fuzzy* em fluxo de dados utilizando o *Framework Online-Offline*.

O algoritmo *d-FuzzStream* é uma evolução do algoritmo *FuzzStream*, que introduziu os conceitos de similaridade e dispersão *fuzzy* na fase *online*. Esses dois conceitos permitiram que os FMiCs retessem mais informações sobre os exemplos, melhorando, as-

sim, a sumarização. Além disso, o algoritmo possui menor complexidade em relação ao algoritmo em que foi baseado, pois realiza menos cálculos de matrizes de pertinência.

A abordagem distribuída inclui duas estratégias de processamento diferentes para cada uma das fases do *framework*. A estratégia da fase *online* sumariza os dados distribuidamente em estruturas de sumarização distintas, que são unificadas por uma fase intermediária. Diferentemente de outros trabalhos similares encontrados na literatura, as estruturas de sumarização não são removidas após o processamento de uma janela de exemplos. Em relação a essa estratégia vale ressaltar que, para os experimentos:

- Reduziu em até 94% o tempo de processamento;
- Processou em média 18.200 exemplos de dimensão 2 a 54 por segundo;
- De maneira geral, gerou agrupamentos similares aos gerados sem a abordagem distribuída;
- Os valores obtidos para as métricas internas e externas, aplicadas aos particionamentos gerados, são comparáveis aos obtidos sem a abordagem distribuída.

Em bases com ruídos, o aumento do paralelismo gerou uma queda progressiva nas métricas de agrupamento, mas que não prejudicou o agrupamento. Um dos principais motivos disso foi a maior quantidade de FMiCs presentes na estrutura de sumarização, que passou a representar mais ruídos.

Mesmo assim, experimentos mostraram que mesmo com esse aumento de tamanho da estrutura de sumarização, o máximo possível não é necessariamente atingido, indicando que o algoritmo ainda possui flexibilidade para aumentar ou diminuir a estrutura dentro de um limite.

A estratégia de sumarização local e global não se mostrou altamente escalável, principalmente pela adição da fase intermediária. Entretanto, a estratégia trouxe ganhos significativos de desempenho que a tornam relevantes para aplicação em domínios reais.

Na fase *offline*, a estratégia de processamento distribuído agrupa o resultado gerado pela estrutura de sumarização unificada com diversos algoritmos e números de grupos diferentes. Ao fim dessa fase, são gerados diversos agrupamentos e o resultado das métricas de avaliação. Sobre essa estratégia ressalta-se:

- Tempos de execução ficaram menores ou comparáveis aos da estratégia de sumarização local e global.
- Gera diversos agrupamentos e métricas de uma só vez, o que auxilia na tomada de decisão de qual o melhor algoritmo e número de grupos para aquela partição.

A estratégia de agrupamento distribuído apesar de se mostrar ineficiente do ponto de vista de desempenho, se mostrou complementar a estratégia de sumarização local e global. Sem essa estratégia, não seria possível gerar resultados em tempo real, pois a fase de agrupamento demoraria mais que a fase de sumarização.

Os resultados obtidos são um indicativo de que a proposta é significativa e contribui para o futuro desenvolvimento de extensões e novas técnicas fundamentadas nas propostas apresentadas neste trabalho.

Limitações e Trabalhos Futuros

Os resultados produzidos pela abordagem distribuída proposta sugerem que o conhecimento gerado é comparável ao conhecimento sem utilizar a abordagem.

Todavia, são identificados alguns pontos de melhoria e de extensão para as propostas apresentadas neste documento.

Melhorar o mecanismo de esquecimento da sumarização - desenvolver uma nova proposta para remoção de FMiCs antigos. Isso melhoraria a qualidade do agrupamento e diminuiria o tamanho da estrutura.

Mecanismos adequados para divisão dos exemplos na fase *online* - a divisão dos exemplos entre as sumarizações locais é feita de maneira aleatória. Isto é aceitável em análises de dados como de redes sociais, onde os dados não possuem correlação. Entretanto, este tipo de estratégia não pode ser aplicada em identificação de sistemas dinâmicos ou séries temporais.

Comparação com técnicas de amostragem - algumas abordagens presentes na literatura realizam amostragem dos dados provenientes do FD para lidar com FDs que produzem dados em ritmo acelerado. Como essas abordagens desprezam informações em detrimento do desempenho, é fundamental realizar comparações com a abordagem distribuída para medir o ganho de informação que a abordagem distribuída pode propiciar.

Comparar com outros algoritmos distribuídos em FD - comparação com outras abordagens de agrupamento distribuído em FD, para analisar o desempenho e a qualidade dos agrupamentos gerados.

Mecanismo automático para obter o melhor agrupamento na fase *offline* - a estratégia de agrupamento distribuído gera diversos resultados de agrupamentos com algoritmos e números de grupos diferentes. Um enorme ganho para essa estratégia seria um mecanismo que encontrasse o melhor agrupamento dentre aqueles realizados, gerando assim, apenas um resultado final.

Referências

ACKERMANN, M. R. et al. StreamKM ++ : A Clustering Algorithm for Data Streams. *J. Exp. Algorithmics*, v. 17, 2012.

AGGARWAL, C. C. *Data streams: models and algorithms*. 1. ed. [S.l.]: Springer US, 2007. 354 p. ISBN 978-0-387-47534-9.

AGGARWAL, C. C. A Survey on Stream Classification Algorithm. In: *Data Classification: Algorithms and Applications*. [S.l.]: CRC Press, 2014. p. 245–273. ISBN 9781441960450.

AGGARWAL, C. C. et al. A Framework for Clustering Evolving Data Streams. In: *Proceedings of the 29th International Conference on Very Large Data bases*. [S.l.: s.n.], 2003. v. 29, p. 81–92.

AL-KHATEEB, T. et al. Stream classification with recurring and novel class detection using class-based ensemble. In: *2012 IEEE 12th International Conference on Data Mining*. [S.l.: s.n.], 2012. p. 31–40.

AL-KHATEEB, T. M. et al. Cloud guided stream classification using class-based ensemble. In: *2012 IEEE Fifth International Conference on Cloud Computing*. [S.l.: s.n.], 2012. p. 694–701.

AMDAHL, G. M. Validity of the single processor approach to achieving large scale computing capabilities. In: *AFIPS Conference Proceedings*. [S.l.: s.n.], 1967. p. 225–231.

Apache Flink. *Dataflow Programming Model*. [S.l.], 2018. Disponível em: <https://ci-.apache.org/projects/flink/flink-docs-release-1.6/concepts/programming-model.html>.

BABCOCK, B. et al. 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, New York, EUA: ACM, 2002. p. 1–16.

BABCOCK, B. et al. Maintaining Variance and k – Medians over Data Stream Windows. In: *22th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. San Diego, Califórnia, EUA: ACM Press, 2003. p. 234–243.

BACKHOFF, O.; NTOUTSI, E. Scalable Online-Offline Stream Clustering in Apache Spark. In: *16th International Conference on Data Mining Workshops*. [S.l.: s.n.], 2016. p. 37–44.

BAHMANI, B. et al. Scalable k-means++. In: *Proceedings of the Very Large Data Bases Endowment*. [S.l.: s.n.], 2012. v. 5, n. 7, p. 622–633.

BARBARÁ, D. Requirements for Clustering Data Streams. *SIGKDD Explor. Newsl.*, v. 3, n. 2, p. 23–27, 2002.

BEZDEK, J. C. *Pattern Recognition with Fuzzy Objective Function Algorithms*. [S.l.]: Kluwer Academic Publishers, 1981. 256 p. ISBN 0306406713.

BHARILL, N.; TIWARI, A.; MALVIYA, A. Fuzzy Based Clustering Algorithms to Handle Big Data with Implementation on Apache Spark. In: *2016 IEEE Second International Conference on Big Data Computing Service and Applications (BigDataService)*. [S.l.]: IEEE, 2016. p. 95–104.

BIFET, A. Adaptive Stream Mining: Pattern Learning and Mining from Evolving Data Streams. In: *Proceedings of the 2010 Conference on Adaptive Stream Mining: Pattern Learning and Mining from Evolving Data Streams*. [S.l.]: IOS Press, 2010. p. 1–212.

BIFET, A. et al. StreamDM: Advanced Data Mining in Spark Streaming. In: *Proceedings - 15th IEEE International Conference on Data Mining Workshop, ICDMW 2015*. [S.l.]: IEEE, 2016. p. 1608–1611.

BIFET, A.; MORALES, G. D. F. Big Data Stream Learning with SAMOA. In: *2014 IEEE International Conference on Data Mining Workshop*. [S.l.]: IEEE, 2014. p. 1199–1202.

BISHOP, C. M. *Neural Networks for Pattern Recognition*. 1. ed. [S.l.]: Oxford University Press, 1995. 482 p. ISBN 0198538642.

CAMPELLO, R. J. G. B.; HRUSCHKA, E. R. A Fuzzy Extension of the Silhouette Width Criterion for Cluster Analysis. *Fuzzy Sets and Systems*, v. 157, p. 2858–2875, 2006.

CAO, F. et al. Density-Based Clustering over an Evolving Data Stream with Noise. In: *Proceedings of the 6th SIAM International Conference on Data Mining*. [S.l.: s.n.], 2006. p. 328–339.

CARBONELL, J. G.; URBANA, C.; MITCHELL, T. M. Machine Learning : A Historical and Methodological. *AI Magazine*, v. 4, n. 3, p. 70–79, 1983.

CHANDOLA, V.; BANERJEE, A.; KUMAR, V. Anomaly Detection: A Survey. *ACM Comput. Surv.*, v. 41, n. 3, p. 1–58, 2009.

CHEN, Y.; TU, L. Density-based clustering for real-time stream data. In: *13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. San Jose, Califórnia, EUA: ACM, 2007. p. 133–142.

Computational Intelligence Group (CIG). Data stream repository. In: . Department of Computing, Universidade Federal de São Carlos - UFSCar, 2017. Disponível em: <http://github.com/CIG-UFSCar/DS_Datasets>.

CâNDIDO, P. L. et al. Scalable data stream clustering with k estimation. In: *2017 Brazilian Conference on Intelligent Systems (BRACIS)*. [S.l.: s.n.], 2017. p. 336–341.

DHAENENS, C.; JOURDAN, L. Metaheuristics and Parallel Optimization. In: *Metaheuristics for Big Data*. 1. ed. [S.l.]: John Wiley & Sons, Inc., 2016. cap. 3, p. 53–61. ISBN 9781119347569.

DHEERU, D.; TANISKIDOU, E. K. *UCI Machine Learning Repository*. 2017. Disponível em: <http://archive.ics.uci.edu/ml>.

DOMINGOS, P.; HULTEN, G. A General Method for Scaling Up Machine Learning Algorithms and its Application to Clustering. In: *Proceedings of the Eighteenth International Conference on Machine Learning*. [S.l.: s.n.], 2001. p. 106–113.

DUDA, R. O.; HART, P. E. *Pattern classification and scene analysis*. 1. ed. [S.l.]: John Wiley and Sons, 1973. 512 p. ISBN 0471223611.

EAGER, D. L.; ZAHORJAN, J.; LAZOWSKA, E. D. Speedup versus efficiency in parallel systems. *IEEE Transactions on Computers*, v. 38, n. 3, p. 408–423, March 1989. ISSN 0018-9340.

F. Dolz, M. et al. Towards Automatic Parallelization of Stream Processing Applications. *IEEE Access*, v. 6, p. 39944–39961, 2018.

FARIA, E. R. et al. Novelty detection in data streams. *Artificial Intelligence Review*, Springer Netherlands, v. 45, n. 2, p. 235–269, 2016. ISSN 1573-7462.

FREY, B. J.; DUECK, D. Clustering by Passing Messages Between Data Points. *Science*, n. February, p. 972–976, 2007.

FRIEDMAN, E.; TZOUMAS, K. *Introduction to Apache Flink*. 1. ed. Sebastopol, CA: O'Reilly Media, Inc, 2016. 110 p. ISBN 9781491973936.

GAMA, J. *Knowledge Discovery from Data Streams*. 1. ed. [S.l.]: Chapman and Hall, 2010. 255 p. ISBN 9781439826119.

GAMA, J. A survey on learning from data streams: current and future trends. *Progress in Artificial Intelligence*, Springer-Verlag, v. 1, n. 1, p. 45–55, apr 2012. ISSN 2192-6352.

GAMA, J.; GABER, M. M. *Learning from Data Streams: Processing Techniques in Sensor Networks*. 1. ed. [S.l.]: Springer-Verlag Berlin Heidelberg, 2007. 244 p. ISBN 978-3-540-73679-0.

GAMA, J.; RODRIGUES, P. P.; LOPES, L. Clustering Distributed Sensor Data Streams Using Local Processing and Reduced Communication. *Intell. Data Anal.*, v. 15, n. 1, p. 3–28, 2011.

GAMA, J. et al. A Survey on Concept Drift Adaptation. *ACM Computing Surveys*, v. 46, n. 4, p. 1–37, oct 2014.

GOLDBERG, D. *Genetic Algorithms in Search, Optimization, and Machine Learning*. 1. ed. [S.l.]: Addison-Wesley, 1989. 432 p. ISBN 0201157675.

GUHA, S. et al. Clustering Data Streams. In: *Proceedings of the IEEE Symposium on Foundations of Computer Science*. [S.l.: s.n.], 2000. p. 359–366.

GULLER, M. *Big Data Analytics with Spark*. 1. ed. [S.l.]: O'Reilly, 2015. 274 p. ISBN 978-1-4842-0965-3.

GUPTA, A. *Apache Mahout Clustering Designs*. 1. ed. [S.l.]: Packt Publishing, 2015. 130 p. ISBN 9781783284436.

HAHSLER, M.; BOLANOS, M.; FORREST, J. *stream: Infrastructure for Data Stream Mining*. [S.l.], 2018. R package version 1.3. Disponível em: <https://CRAN.R-project-.org/package=stream>.

HAHSLER, M.; BOLANOS, M.; FORREST, J. *streamMOA: Interface for MOA Stream Clustering Algorithms*. [S.l.], 2018. R package version 1.1-4. Disponível em: <https://CRAN.R-project.org/package=streamMOA>.

HALKIDI, M.; KOUTSOPOULOS, I. Online Clustering of Distributed Streaming Data Using Belief Propagation Techniques. In: *2011 IEEE 12th International Conference on Mobile Data Management*. [S.l.]: IEEE, 2011. v. 1, p. 216–225.

HAN, J.; KAMBER, M.; PEI, J. *Data Mining: Concepts and Techniques*. 3. ed. [S.l.]: Morgan Kaufmann, 2011. 744 p. ISBN 9780123814791.

HORE, P. et al. Online fuzzy c means. In: *NAFIPS 2008 - 2008 Annual Meeting of the North American Fuzzy Information Processing Society*. [S.l.]: IEEE, 2008. p. 1–5.

HORE, P.; HALL, L. O.; GOLDGOF, D. B. Creating Streaming Iterative Soft Clustering Algorithms. In: *NAFIPS 2007 - 2007 Annual Meeting of the North American Fuzzy Information Processing Society*. [S.l.]: IEEE, 2007. p. 484–488.

HUESKE, F.; KALAVRI, V. *Stream Processing with Apache Flink*. First edition. [S.l.]: O'Reilly Media, Inc., 2018. 250 p. ISBN 9781491974285.

ISAKSSON, C.; DUNHAM, M. H.; HAHSLER, M. SOStream: Self Organizing Density-Based Clustering over Data Stream. In: *8th International Conference on Machine Learning and Data Mining in Pattern Recognition*. [S.l.: s.n.], 2012. p. 264–278.

JAIN, A. K.; MURTY, M. N.; FLYNN, P. J. Data clustering: a review. *ACM Computing Surveys*, v. 31, n. 3, p. 264–323, 1999.

JIANG, N.; GRUENWALD, L. Research Issues in Data Stream Association Rule Mining. *SIGMOD Rec.*, v. 35, n. 1, p. 14–19, 2006.

KALANTARIAN, Z. S.; MASHAYEKHI, H.; ABDOSHAHI, A. GFCM: A gossip-based fuzzy C-means algorithm. In: *2014 6th Conference on Information and Knowledge Technology (IKT)*. [S.l.: s.n.], 2014. p. 152–157.

KRANEN, P. et al. The ClusTree: Indexing Micro-clusters for Anytime Stream Mining. *Knowl. Inf. Syst.*, v. 29, n. 2, p. 249–272, 2011.

LEIBIUSKY, J.; EISBRUCH, G.; SIMONASSI, D. *Getting started with Storm*. 1. ed. [S.l.]: O'Reilly Media, Inc, 2012. 106 p. ISBN 9781449324018.

LIN, J.; DYER, C. *Data-Intensive Text Processing with MapReduce*. 1. ed. [S.l.: s.n.], 2010. 178 p. ISBN 1608453421.

LOPES, P. A.; CAMARGO, H. A. FuzzStream: Fuzzy Data Stream Clustering Based on the Online-Offline Framework. In: *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. [S.l.]: IEEE, 2017. p. 6.

LOPES, P. d. A. *FMiC e dFMiC: Manutenção de Micro-Grupos Fuzzy para Agrupamento Flexível em Fluxo de Dados baseado no Framework Online-Offline*. 116 p. Tese (Doutorado) — Universidade Federal de São Carlos - UFSCar, 2016.

LU, J. et al. A parallel approach on clustering traffic data stream based on the density. In: *2018 Sixth International Conference on Advanced Cloud and Big Data (CBD)*. [S.l.: s.n.], 2018. p. 281–286.

MACQUEEN, J. B. Some Methods for Classification and Analysis of MultiVariate Observations. In: *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*. [S.l.]: University of California Press, 1967. p. 281–297.

MARKOV, I. L. Limits on Fundamental Limits to Computation. *Nature*, p. 147–154, aug 2014.

MASUD, M. M. et al. Detecting recurring and novel classes in concept-drifting data streams. In: *2011 IEEE 11th International Conference on Data Mining*. [S.l.: s.n.], 2011. p. 1176–1181.

MATHEW, J. M.; CHANDRAN, L. P. Parallel Implementation of Fuzzy Clustering Algorithm Based on MapReduce Computing Model of Hadoop – A Detailed Survey. *International Journal of Computer Science and Information Technologies*, v. 6, n. 5, p. 4740–4744, 2015.

METWALLY, A.; AGRAWAL, D.; El Abbadi, A. Duplicate Detection in Click Streams. In: *Proceedings of the 14th International Conference on World Wide Web*. [S.l.]: ACM, 2005. p. 12–21.

MICHALSKI, R. S.; CARBONELL, J. G.; MITCHELL, T. M. *Machine Learning an Artificial Intelligence Approach Volume II*. 1. ed. [S.l.]: Morgan Kaufmann, 1986. 738 p. ISBN 0-934613-00-1.

MITCHELL, T. *Machine Learning*. 1. ed. [S.l.]: McGraw-Hill Education, 1997. 432 p. ISBN 0070428077.

MORSHED, S. J.; RANA, J.; MILRAD, M. Open Source Initiatives and Frameworks Addressing Distributed Real-Time Data Analytics. In: *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. [S.l.]: IEEE, 2016. p. 1481–1484.

MOSTAFAVI, S.; AMIRI, A. Extending fuzzy c-means to clustering data streams. In: *20th Iranian Conference on Electrical Engineering (ICEE2012)*. [S.l.]: IEEE, 2012. p. 726–729.

NGUYEN, H.-L.; WOON, Y.-K.; NG, W.-K. A survey on data stream clustering and classification. *Knowledge and Information Systems*, Springer London, v. 45, n. 3, p. 535–569, dec 2015. ISSN 0219-1377.

O'CALLAGHAN, L. et al. Streaming data algorithms for high-quality clustering. In: *Proceedings of the 18th International Conference on Data Engineering*. [S.l.: s.n.], 2002. p. 685–694.

PARK, N. H.; LEE, W. S. Cell Trees: An Adaptive Synopsis Structure for Clustering Multi-dimensional On-line Data Streams. *Data Knowl. Eng.*, v. 63, n. 2, p. 528–549, 2007.

QUINLAN, J. R. Induction of decision trees. *Machine Learning*, v. 1, n. 1, p. 81–106, 1986.

R Core Team. *R: A Language and Environment for Statistical Computing*. Vienna, Austria, 2018. Disponível em: <https://www.R-project.org/>.

RAND, W. M. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, [American Statistical Association, Taylor & Francis, Ltd.], v. 66, n. 336, p. 846–850, 1971.

REN, J.; MA, R.; REN, J. Density-based Data Streams Clustering over Sliding Windows. In: *6th International Conference on Fuzzy Systems and Knowledge Discovery*. [S.l.: s.n.], 2009. v. 5, p. 248–252.

ROUSSEEUW, P. J. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, v. 20, p. 53–65, 1987.

SABIT, H.; AL-ANBUKY, A.; GHOLAM-HOSSEINI, H. Distributed WSN Data Stream Mining Based on Fuzzy Clustering. In: *2009 Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing*. [S.l.: s.n.], 2009. p. 395–400.

SCHICK, L.; LOPES, P. d. A.; CAMARGO, H. A. d-FuzzStream: A Dispersion-Based Fuzzy Data Stream Clustering. In: *IEEE International Conference on Fuzzy Systems*. Rio de Janeiro: IEEE, 2018. p. 135–142.

SHAHRIVARI, S.; JALILI, S. Single-pass and linear-time k-means clustering based on MapReduce. *Information Systems*, Elsevier, v. 60, p. 1–12, 2016.

SILVA, J. a. et al. Data stream clustering. *ACM Computing Surveys*, v. 46, n. 1, p. 1–31, 2013.

TAN, P.-N.; STEINBACH, M.; KUMAR, V. *Introduction to Data Mining, (First Edition)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005. ISBN 0321321367.

TANENBAUM, A. S.; STEEN, M. V. *Distributed Systems: Principles and Paradigms*. 3. ed. [S.l.]: Pearson, 2006. 704 p. ISBN 0132392275.

TASOULIS, D. K.; ADAMS, N. M.; HAND, D. J. Unsupervised Clustering In Streaming Data. In: *Proceedings of the Sixth IEEE International Conference on Data Mining - Workshops*. [S.l.]: IEEE, 2006. p. 638–642.

WANG, B. et al. Parallelizing K-means-based Clustering on Spark. In: *International Conference on Advanced Cloud and Big Data*. [S.l.]: IEEE, 2016. p. 31–36.

WHITE, T. *Hadoop: The Definitive Guide*. 4. ed. [S.l.]: O'Reilly, 2015. 688 p. ISBN 9781491901632.

XIE, X. L.; BENI, G. A validity measure for fuzzy clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, n. 8, p. 841–847, 1991.

XIONG, X.; CHAN, K. L.; TAN, K. L. Similarity-driven cluster merging method for unsupervised fuzzy clustering. In: *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*. [S.l.]: AUAI Press, 2004. p. 611–618.

YANG, C.; ZHOU, J. HClustream: A Novel Approach for Clustering Evolving Heterogeneous Data Stream. In: *Proceedings of the 6th IEEE International Conference on Data Mining - Workshops (ICDMW'06)*. [S.l.]: IEEE, 2006.

ZADEH, L. A. Fuzzy Sets. *Information and Control*, v. 8, n. 3, p. 338–353, 1965.

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*. [S.l.: s.n.], 1996. p. 103–114.

ZHANG, Y.; TANGWONGSAN, K.; TIRTHAPURA, S. Streaming k-Means Clustering with Fast Queries. In: *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. [S.l.]: IEEE, 2017. p. 449–460.

ZHOU, A. et al. Tracking Clusters in Evolving Data Streams over Sliding Windows. *Knowl. Inf. Syst.*, v. 15, n. 2, p. 181–214, 2008.

ZHOU, J. et al. A Collaborative Fuzzy Clustering Algorithm in Distributed Network Environments. *IEEE Transactions on Fuzzy Systems*, v. 22, n. 6, p. 1443–1456, 2014.

ZHOU, J. et al. A distributed K-means clustering algorithm in wireless sensor networks. In: *2015 International Conference on Informative and Cybernetics for Computational Social Systems (ICCSS)*. [S.l.]: IEEE, 2015. p. 26–30.

ZHU, Y.; SHASHA, D. StatStream: Statistical monitoring of thousands of data streams in real time. In: *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB'02)*. [S.l.]: VLBD Endowment, 2002. p. 358–369.