VINICIOS CAINÃ DOS SANTOS COELHO

CLASSIFICAÇÃO DE FLUXOS CONTÍNUOS DE DADOS DESBALANCEADOS USANDO ENSEMBLES HETEROGÊNEOS, SUB E SOBRE-AMOSTRAGEM INVERSA ALEATÓRIA E META-APRENDIZAGEM

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná (PUCPR) como requisito parcial para obtenção do título de Mestre em Informática.

Curitiba
2022

VINICIOS CAINÃ DOS SANTOS COELHO

CLASSIFICAÇÃO DE FLUXOS CONTÍNUOS DE DADOS DESBALANCEADOS USANDO ENSEMBLES HETEROGÊNEOS, SUB E SOBRE-AMOSTRAGEM INVERSA ALEATÓRIA E META-APRENDIZAGEM

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná (PUCPR) como requisito parcial para obtenção do título de Mestre em Informática.

Área de Concentração: Ciência da Computação

Orientador: Jean Paul Barddal
Co-orientador: Alceu de Souza Britto Jr.

Curitiba
2022

Dados da Catalogação na Publicação
Pontifícia Universidade Católica do Paraná
Sistema Integrado de Bibliotecas – SIBI/PUCPR
Biblioteca Central
Edilene de Oliveira dos Santos CRB 9 / 1636

C672c
2022

Coelho, Vinicios Cainã dos Santos
Classificação de fluxos contínuos de dados desbalanceados usando ensembles heterogêneos, sub e sobre-amostragem inversa aleatória e meta-aprendizagem / Vinicios Cainã dos Santos Coelho ; orientador: Jean Paul Barddal ; coorientador: Alceu de Souza Britto Jr. -- 2022
98 f. : il. ; 30 cm

Dissertação (mestrado) – Pontifícia Universidade Católica do Paraná, Curitiba, 2022.
Bibliografia: f. 90-98

1. Informática. 2. Mineração de dados (Computação). 3. Fluxo de dados (Computadores). 4. Algoritmos. 5. Inteligência artificial. I. Barddal, Jean Paul. II. Britto Junior, Alceu de Souza. III. Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Informática. IV. Título

CDD 20. ed. – 004

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

52-2022

DECLARAÇÃO

Declaro para os devidos fins que o aluno **VINICIOS CAINÃ DOS SANTOS COELHO**, defendeu sua dissertação de Mestrado intitulada "**CLASSIFICAÇÃO DE FLUXOS CONTÍNUOS DE DADOS DESBALANCEADOS USANDO ENSEMBLES HETEROGÊNEOS, SUB E SOBRE-AMOSTRAGEM INVERSA ALEATÓRIA E META-APRENDIZAGEM",** na área de concentração Ciência da Computação, no dia 05 de abril de 2022, no qual foi aprovado.

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

Por ser verdade, firmo a presente declaração.

Curitiba, 05 de agosto de 2022.

Prof. Dr. Emerson Cabrera Paraiso
Coordenador do Programa de Pós-Graduação em Informática
Pontifícia Universidade Católica do Paraná

Agradecimentos

Inicialmente, gostaria de agradecer a Deus por essa conquista. Tudo só foi possível graças a sua benção.

Dentre as pessoas que me guiaram por essa jornada, quero começar agradecendo ao meu orientador Jean Paul Barddal. Obrigado por ter me acolhido e me aceitado como seu aluno. Me sinto honrado. Por inúmeras vezes esteve disponível para responder às minhas dúvidas, mesmo quando eu às considerava simples demais para serem ouvidas. Teve toda a paciência do mundo e sempre esteve um passo a frente ao sugerir novas alternativas e opções de exploração ao longo da pesquisa. Mesmo quando eu imaginava que não conseguiria produzir algo, ele estava lá para apoiar e me levar a seguir em frente. Agradeço todo o respeito e companheirismo oferecido. Espero levar para a minha vida toda a integridade, seriedade e profissionalismo que sempre notei em você desde quando o conheci enquanto ainda estava concluindo seu mestrado. Sempre foi e sempre será um exemplo para mim. Lhe desejo todo o sucesso em sua vida.

Agradeço também ao meu co-orientador Alceu de Souza Britto Jr., que não só agregou ao projeto com seu profundo conhecimento, mas também sempre buscou trazer luz à alternativas que não estava ciente que existiam, além de iluminar vários dos meus dias com seu bom humor. Também lhe desejo muito sucesso sempre.

Aos professores Fabricio Enembreck e Paulo Ricardo Lisboa de Almeida, agradeço imensamente pelos comentários sobre este trabalho. Suas considerações foram de imenso valor para a evolução desse projeto e também proverão frutos nos trabalhos futuros.

A todos os meus amigos que sempre me motivaram a seguir em frente por todo o percurso. Em especial ao Edson Monteiro, por ter sido um dos responsáveis por me fazer iniciar essa jornada no momento certo, e ao Lucca Portes, pelo mesmo motivo anteriormente citado, mas também por ter me auxiliado e acompanhado por todo desenvolvimento deste projeto.

A Fundação Araucária e a Pontifícia Universidade Católica do Paraná pelo apoio financeiro cedido para este projeto.

Finalmente, agradeço à minha família. Obrigado do fundo do meu coração por todo

o esforço, sacrifício, carinho e amor que tiveram comigo. Amo todos vocês imensamente e não tenho como expressar em palavras o quanto me sinto agradecido e abençoado por ter vocês em minha vida. Sem vocês eu nunca estaria onde estou hoje. Espero estar me tornando alguém que vocês tenham orgulho.

E por fim, agradeço em especial a uma heroína que não só me deu a vida, me criou e lutou tanto por mim ao longo do caminho, mas que também me salvou no que talvez foi o pior episódio da minha vida. Ela literalmente me deu uma segunda vida colocando a sua em risco. Muitíssimo obrigado mãe. Te amo.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviações e Símbolos

β	*Um fluxo de dados qualquer*
s	*Instância presente no fluxo de dados* β
X	*Vetor de atributos que formam uma instância qualquer*
x	*Valor de algum atributo dentro de* X
d	*Número de dimensões de um dataset*
y	*Classe ou regressão à qual uma instância* X *pertence*
Δ	*Tamanho de um subconjunto temporal de instâncias*
E	*Um evento aleatório qualquer*
e	*Número total de classes que um problema analisado possui*
IR	*Imbalance Ratio*
Maj	*Quantidade de exemplos que pertencem à classe majoritária*
Min	*Quantidade de exemplos que pertencem à classe minoritária*
K	*Número de dobras da validação cruzada*
VP	*Número de votos verdadeiro positivos*
VN	*Número de votos verdadeiro negativos*
FP	*Número de votos falso positivos*
FN	*Número de votos falso negativos*
PP	*Número de votos preditos positivos*
PN	*Número de votos preditos negativos*
RP	*Número de votos reais positivos*
RN	*Reais negativos*
T	*Número de votos totais*
ϵ	*Um ensemble formado por* M *classificadores base*
M	*Número de classificadores membros de um ensemble* ϵ
κ_i	*O valor de Kappa para o* i*-ésimo classificador base de um ensemble*
C^y	*Número de instâncias da classe* y
C	*Quantidade total de instâncias já recebidas por uma stream*

F	*Número de vezes que uma instância é selecionada para fazer parte de um bag*
L	*Quantidade de exemplos na base de dados*
B	*Número de conjuntos de treinamento criados no algoritmo SIRUS*
Z	*Número de instâncias da classe minoritária de um conjunto de treinamento do algoritmo SIRUS*
W	*Número de instâncias da classe majoritária de um conjunto de treinamento do algoritmo SIRUS*
W_i	*Número de instâncias da classe majoritária do i-ésimo conjunto criado através do inverse random undersampling a partir de uma base de treinamento do algoritmo SIRUS*
R	*Número de algoritmos diferentes sendo usados no ensemble das técnicas SIRUS e OnlineSIRUS*
I_y	*Número de instâncias da classe y que já foram usadas para treino pelo OnlineSIRUOS*
I	*Número de instâncias que já foram usadas para treino pelo OnlineSIRUOS*
λ'	*Valor do parâmetro lambda escolhido para o OnlineSIRUOS*
w	*Peso gerado para uma instância*
D	*Número de grupos de modelos criados no algoritmo OnlineSIRUOS*
Ω	*Meta-classificador do OnlineSIRUOS*
U	*Número de instâncias em um experimento*
δ	*Intervalo de instâncias entre amostragens de desempenho de um algoritmo*

Resumo

Com a massiva quantidade de dados gerados todos os dias, a possibilidade de aplicação de inteligência artificial chegou a um patamar histórico. Com a evolução dessas tecnologias, surgiram oportunidades de aplicação de mineração de dados em diversos meios inexplorados. Todavia, a limitação do tempo de resposta disponível para esses sistemas surgiu. Com isso, novas abordagens precisaram ser desenvolvidas para minerar fluxos de dados contínuos sem perder informação, usando recursos computacionais de forma que não fossem sobrecarregados e ao mesmo tempo entregando previsões precisas e rápidas. Este trabalho está focado em fluxos de dados onde a frequência das classes é significativamente díspar, culminando no problema de desbalanceamento. Para tratar fluxos de dados desbalanceados, é proposto o algoritmo OnlineSIRUOS. Este algoritmo é inspirado no algoritmo SIRUS, projetado originalmente para cenários de classificação em lote. Assim como o algoritmo original, o algoritmo proposto, doravante denominado *Online Stacking Inverse Random Under and Over Sampling* (OnlineSIRUOS), combina aprendizagem baseada em *ensembles* e meta-aprendizagem, mas conta com uma variação diferente de subamostragem inversa aleatória. Uma implementação do método proposto foi realizada e comparada com diferentes classificadores projetados para fluxos contínuos de dados, dentre os quais a maioria possui mecanismos para lidar com desbalanceamento. Quando comparado contra métodos existentes da literatura, os resultados obtidos demonstram que o método proposto é competitivo em relação as métricas de desempenho F1-score, tendo resultados até 2,7 pontos percentuais melhores e também se mantendo a frente da maior parte dos algoritmos analisados no custo computacional. Mas ainda há pontos fracos em relação a bases de dados com naturezas mais similares a cenários reais. Finalmente, uma conclusão é feita a respeito dos resultados obtidos e melhorias do estudo são sugeridas para servirem como possíveis bases de estudo em projetos futuros.

Palavras-chave: Mineração de Fluxos de Dados; Desbalanceamento de classes.

Abstract

With the massive amount of data generated every day, the possibility of applying artificial intelligence has reached an all-time high. With the evolution of these technologies, opportunities have arisen for the application of data mining in various unexplored environments. However, the limitation of the response time available to these systems emerged. With this, new approaches needed to be developed to mine continuous data streams without losing information, using computational resources in a way that would not be overloaded while delivering accurate and fast predictions. This work is aimed towards data streams where the frequency of classes is significantly disparate, culminating in the imbalance problem. To handle imbalanced data streams, the OnlineSIRUOS algorithm is proposed. This algorithm is inspired by the SIRUS algorithm, originally designed for batch classification scenarios. Like the original algorithm, the proposed algorithm, henceforth referred to as *Online Stacking Inverse Random Under and Over Sampling* (OnlineSIRUOS), combines learning based on ensembles and meta-learning, but relies on a different variation of random inverse undersampling. An implementation of the proposed method was performed and compared against different classifiers designed for continuous data streams, among which most have mechanisms to deal with imbalance. When compared against existing methods in the literature, the results obtained show that the proposed method is competitive with respect to the F1-score performance metrics, with results up to 2.70 percentage points better and also keeping ahead of most of the analyzed algorithms in computational cost. But there are still weaknesses with respect to databases with natures more similar to real scenarios. Finally, a conclusion is made regarding the results obtained and improvements to the study are suggested to serve as possible bases for study in future projects.

Keywords: Data Stream Mining; Class Imbalance.

Capítulo 1

Introdução

Com a massiva quantidade de dados sendo gerada todos os dias pelos milhões de dispositivos eletrônicos existentes, a possibilidade de aplicação e refinamento de instâncias de inteligência artificial (IA) chegou a um patamar histórico. A evolução e incremento dessas tecnologias gerou oportunidades de desenvolvimento de negócios como nunca vistas. Atualmente, é possível criar aplicações que podem classificar a espécie de uma planta a partir da foto de suas folhas (ARAÚJO et al., 2018), prever a duração das interrupções do sistema de distribuição de energia e indicar as suas possíveis causas e etapas de reparo (JAECH et al., 2019), entre outros exemplos. Contudo, essa riqueza de informação trouxe alguns problemas para o paradigma de execução atual que utilizamos na mineração de dados, os quais têm uma origem única: o tempo de resposta disponível.

Em aplicações reais que envolvem fluxos de dados constantes e que possibilitam a realização de previsões baseadas nos mesmos, como por exemplo um sistema de *e-commerce*, onde é possível tentar prever os prováveis interesses de compra de um usuário baseado no seu histórico e de indivíduos similares, é necessário que o software esteja sempre pronto para atender uma nova requisição. Usando o exemplo citado acima e focando principalmente na tarefa de classificação, é possível elencar algumas condições que determinado sistema deve atender para que funcione da maneira mais correta possível (BIFET et al., 2018):

- O sistema deve sempre estar disponível para prever o que provavelmente seus usuários podem querer comprar.
- O sistema deve sempre estar aprendendo mais sobre seus usuários para melhor lhes recomendar produtos.
- O sistema não deve ser sobrecarregado pelo seu(s) classificador(es), afinal os recursos computacionais são finitos e podem estar sendo compartilhados com outros

componentes da mesma aplicação.

Essas condições podem não ser bem atendidas por algoritmos de mineração de dados tradicionais, também conhecidos como algoritmos *batch*, pois:

- Eles utilizam o máximo da capacidade que lhes é dada para gerar o treinamento de um modelo, sem uma limitação do uso de recursos computacionais.
- Para adaptar o modelo a novos dados, é necessário treiná-lo novamente com todas as instâncias utilizadas anteriormente na sua criação juntamente com os novos dados coletados, gerando assim uma demanda muito grande de armazenamento ao longo do tempo.
- Ao passar do tempo, se torna impossível carregar muitos exemplos em memória para executar o treinamento de um novo modelo pois o fluxo de dados pode não ter fim.
- Tanto o treinamento quanto o processo de previsão desses algoritmos não possuem limitações impostas pelo ambiente onde eles executam.

Devido a cenários com desafios intrínsecos como esse, foi necessária a criação de um novo formato de mineração de dados baseado em fluxos contínuos de dados ou *data stream mining*. Essa abordagem busca encontrar um equilíbrio entre o uso de recursos computacionais e um tempo de resposta baixo, objetivando atender a aplicações que demandam alto desempenho e que condizem com muitos dos problemas atuais de computação. Nesse tipo de cenário, é comum encontrar algoritmos baseados em *ensembles*, pois esse tipo de abordagem permitiu aumentar a performance de classificação de fluxos contínuos de dados, mesmo que abrindo mão de tempo de processamento e uso de memória.

Entretanto, assim como em qualquer cenário de mineração de dados, existem condições especiais que o próprio ambiente pode possuir. Uma delas é o desbalanceamento de classes, que decorre da existência desproporcional de instâncias de diferentes classes. Isso pode acarretar em problemas de classificação, pois os algoritmos tendem a aprender mais sobre as classes de maior frequência. Ademais, os cenários em que o desbalanceamento é considerado como algo importante a ser tratado são os mesmos em que as classes com menos exemplos são as que devem ser tratadas com mais cuidado, buscando sempre aumentar os acertos de classificação sobre essas. Além disso, o desbalanceamento é uma ocorrência comum em fluxos de dados reais. Para lidar com isso, os algoritmos tendem a utilizar estratégias como *sampling*, que busca equilibrar o número de instâncias das diferentes classes de um problema antes de serem usadas para treinar um modelo (CHAWLA et al., 2002), *ensembles*, treinando cada classificador base em uma parte do problema

(HO, 1998), *cost-sensitive*, utilizando os custos dos erros de classificação para melhorar o treinamento dos modelos (SUN, 2007), e modificação de algoritmo, buscando alterar seus mecanismos internos para lidar com o desbalanceamento (XU, 2017).

Dentro dos grupos de opções abordados anteriormente, existem muitas estratégias que podem ser usadas para tratar o desbalanceamento. Entre elas existem as técnicas de *inverse random under- and over-sampling*, *ensembles* heterogêneos e meta-aprendizagem (BIFET et al., 2009). Cada uma dessas já foram usadas separadamente em diversos trabalhos, mas não foi encontrado nenhuma menção do uso das três ao mesmo tempo em um algoritmo de *data stream mining*. Desta forma, ainda não foi possível verificar uma provável sinergia entre essas técnicas, que consiste no seguinte: o uso de *undersampling* é uma alternativa eficiente de *sampling* pois basicamente remove exemplos que seriam usados para treinamento; quando usada em conjunto com *oversampling* sobre fluxos de dados, ainda podem trazer bastante velocidade ao tratar o desbalanceamento, visto que nesse cenário o uso de uma mesma instância para repetidos treinamentos é uma alternativa válida, evitando a necessidade de criação de novos exemplos, e atendendo melhor o caso geral de cenários ao usar ambas as técnicas em conjunto devido as suas particularidades; além disso, as técnicas de *sampling* geram a opção do uso de quaisquer algoritmos para um *ensemble*, visto que assim eles podem lidar com o desbalanceamento sem a necessidade de alteração de seus mecanismos internos de funcionamento; além disso, o uso da estratégia de *stacking* sobre um *ensemble* treinado sobre diferentes parte de um cenário desbalanceado pode trazer ainda mais diversidade para o modelo. Por isso, o presente projeto busca criar um novo método para lidar com o desbalanceamento de fluxos de dados contínuos baseado no uso dessas técnicas em conjunto para avaliar a utilização combinada das mesmas. O algoritmo aqui proposto, denominado *Online Stacking Inverse Random Under and Over Sampling* (OnlineSIRUOS), é apresentado no Capítulo 4.

1.1 Hipóteses

A hipótese alternativa a ser checada neste projeto é definida a seguir: o *ensemble* proposto, OnlineSIRUOS, ao combinar técnicas de heterogeneidade, sub e sobre amostragem inversa aleatória e meta-aprendizagem supera métodos do estado da arte para classificação de fluxos contínuos de dados desbalanceados considerando técnicas específicas para este tipo de cenário.

1.2 Objetivos

O objetivo deste trabalho é propor um novo método baseado em *ensembles* heterogêneos para classificação de data streams que usará em conjunto as técnicas de *inverse random under and over sampling* e meta-aprendizagem para sobrepujar o problema de desbalanceamento em data streams.

Os objetivos específicos incluem:

- Avaliar o método proposto em cenários sintéticos e reais onde o desbalanceamento de classes está presente;
- Conduzir um estudo experimental de diferentes configurações de hiper-parâmetros para definição de uma configuração padrão (*default*);
- Comparar os resultados obtidos com algoritmos existentes e disponíveis do estado da arte.

1.3 Contribuições

O projeto tem como possíveis contribuições:

- A adaptação e combinação das técnicas de *ensembles* heterogêneos, subamostragem inversa aleatória (*inverse random undersampling*) e meta-aprendizado em conjunto para cenários de classificação de *data streams* desbalanceadas, inspirada no SIRUS (ZHANG et al., 2018), algoritmo originalmente construído para ambientes *batch* que utiliza dessas mesmas técnicas;
- A avaliação do uso combinado das técnicas de *ensembles* heterogêneos, *inverse random undersampling* e meta-aprendizado;
- A disponibilização do código-fonte do algoritmo proposto para a comunidade científica, garantindo a reprodutibilidade da pesquisa;
- Uma análise do algoritmo proposto frente ao estado da arte em cenários reais e sintéticos com desbalanceamento de classes perante diferentes métricas de avaliação;
- A publicação do método proposto e seus resultados em artigo científico.

1.4 Visão geral

Este documento está dividido como mostrado a seguir. O Capítulo 2 apresenta uma introdução sobre mineração de fluxos de dados contínuos e ao tema do desbalanceamento. O Capítulo 3 mostra trabalhos relacionados a mineração de fluxos de dados, sendo vários deles preparados para o problema específico de desbalanceamento de dados. O Capítulo 4 detalha o método SIRUS original, assim como a variante proposta para cenários de *data streams*. O Capítulo 5 apresenta os experimentos realizados, assim como discute os resultados obtidos na presente pesquisa. O Capítulo 6 conclui este documento, reportando as conclusões do projeto e elenca trabalhos futuros.

Capítulo 2

Mineração de Fluxos Contínuos de Dados

Minerar cenários onde os dados que estão continuamente surgindo em fluxos possivelmente infinitos não é uma atividade trivial. Algoritmos tradicionais de mineração de dados nem sempre conseguem obter um desempenho suficientemente bom para atuar sobre dados nesse formato, seja por motivos de processamento, memória consumida ou até pela dificuldade em se adaptar a um ambiente tão volátil. A necessidade de criação de novos algoritmos voltados para tais desafios surgiu e muitas técnicas foram construídas, partindo de modificações de algoritmos já existentes ou até mesmo surgindo de ideias completamente novas. Entretanto, assim como em cenários tradicionais, existem muitas dificuldades que podem ser restritivas ao desempenho de qualquer técnica e algumas podem ser mais situacionais e específicas que outras, como o acesso limitado a exemplos de uma classe específica. Sendo assim, muitas áreas de estudo inexploradas ainda existem para serem pesquisadas em meio a esse ambiente específico.

Esse capítulo busca apresentar as diferentes condições, requisitos, restrições, e necessidades que diferentes algoritmos devem possuir para executar apropriadamente sobre ambientes de fluxos de dados contínuos. Descrições sobre como avaliar o desempenho de algoritmos sobre tal ambiente assim como informações sobre uma restrição específica, conhecida como desbalanceamento, que pode facilmente ocorrer em bases de dados reais sobre fluxos de dados contínuos também são abordadas.

2.1 Fluxos de Dados

Um fluxo de dados β pode ser descrito como uma sequência de elementos possivelmente infinita $(s^1, s^2, s^3, ..., s^\infty) \in \beta$, onde cada item s representa uma instância. Essas instâncias são independentes umas das outras e seguem o formato de um par (X, y), dado que X é um conjunto formado por d itens x, sendo $(x^1, x^2, x^3, ..., x^d) \in X$ e $d \in \mathbb{N}^*$, onde d

é a dimensionalidade do espaço de *features* (ou atributos) do presente problema e cada um dos elementos do conjunto representa o valor de uma *feature* da sua instância original, e y o valor da classificação ou regressão à qual o exemplo pertence. Sendo assim, um modelo criado para resolver esse problema deve ser capaz de derivar a função $f(X) = y$. É importante ressaltar que o formato com que dados se tornam disponíveis pode variar. O formato mais comum em artigos científicos segue a notação acima, assumindo que as instâncias chegam uma a uma, além de que y^n se torna disponível logo após a chegada de x^n e antes da chegada de (x^{n+1}). Por outro lado, outro formato de processamento de instâncias assume que as instâncias chegam - ou são processadas - em mini-lotes, ou *mini batches*. Neste formato, instâncias se tornam disponíveis em subconjuntos temporais, isto é, assumindo que um lote possui tamanho Δ, as instâncias s^i até $s^{i+\Delta}$ se tornam disponíveis para teste (apenas x^i até $y^{i+\Delta}$) e posteriormente para treino, isto é, neste momento y^i a $y^{i+\Delta}$ são disponibilizadas. Este processo é repetido infinitamente, sempre avançando o processamento de acordo com Δ: $s^{i+2\Delta}$ a $s^{i+3\Delta}$, $s^{i+3\Delta}$ a $s^{i+4\Delta}$ e assim por diante.

Ambientes de fluxos de dados diferem em muitos aspectos de um contexto tradicional de mineração de dados, principalmente em relação a disponibilidade e quantidade dos dados. Em ambientes comuns, os dados a serem analisados podem ser pré-processados da melhor maneira possível, armazenados por tempo indeterminado, reutilizados quando necessário e processados na forma que for preciso. Isso ocorre porque, em problemas como esse, a maior motivação é maximizar a precisão de um algoritmo, sem tanta necessidade de se ater a outros aspectos. Obviamente, outros fatores podem influenciar a avaliação de algoritmos inseridos nesse contexto. Como exemplo, podemos destacar casos em que dois modelos atingem o mesmo nível de precisão de acertos, mas um deles completa a etapa de treinamento mais rapidamente que o outro. Todavia não há nenhuma limitação externa que é realmente imposta pelo ambiente e que precise ser seguida a todo custo.

Em fluxos de dados, ou também conhecidos como ambientes *online*, todo o estudo que for elaborado sobre esse meio deve levar em consideração algumas restrições específicas já que esse tipo de contexto busca replicar os mesmos desafios reais que aplicações de mineração de dados podem vir a enfrentar em sistemas em produção com alta magnitude de transações de informação. Segundo (BIFET; KIRKBY, 2009), para que um algoritmo seja considerado como apto a trabalhar com fluxos de dados contínuos, ele deve atender a essas restrições ou, nesse caso, a esses requisitos que estão listados abaixo:

1. **Processar uma instância de cada vez e somente uma vez:** Cada instância que chega até o modelo deve ser processada, ou ignorada, independente da ordem em que vier, descartando a mesma após o uso pois outras instâncias também estão

chegando e devem passar pelo mesmo procedimento. Há casos onde é possível refinar o modelo criado repassando os mesmos dados previamente utilizados, porém isso só deve ser feito em momentos que isso for prático e as instâncias devem ser revistas poucas vezes, já que um algoritmo que requer o uso dessa estratégia com pouca moderação pode não ser muito flexível para cenários *online*. É importante lembrar também que não há limitação para que os modelos guardem algumas instâncias para serem usadas em breve, desde quem o uso da memória seja tratado com precaução.

2. **Usar uma quantidade de memória limitada:** Ao trabalhar com fluxos de dados contínuos, pressupõe-se que há uma quantidade massiva de dados a ser processada. Isso pode exaurir facilmente os recursos de memória do sistema computacional sendo usado. Por isso, algoritmos de *data stream mining* são definidos de forma a usarem a memória disponível de maneira consciente. Essa restrição também possui exceções, onde o relaxamento do uso da memória só ocorre se houver algum meio de armazenamento externo, como por exemplo o uso de arquivos temporários. Contudo, independente da estratégia abordada para tal fim, ela deve levar em consideração o próximo requisito.

3. **Usar uma quantidade de tempo limitada:** Para que um modelo qualquer consiga atender em tempo real o número de instâncias que chegam de uma stream de dados, ele deve ser capaz de terminar o processamento de uma delas antes que a próxima esteja disponível. Caso isso não ocorra, há o risco de perda de dados. Portanto, ao projetar um algoritmo nesse cenário, é preciso que exista uma margem que limita o máximo de tempo por instância que deve ser respeitado. Obviamente essa limitação pode não ser tão restritiva em alguns casos, entretanto, é recomendado que ela seja um limiar pequeno porque geralmente cenários assim são mais restritivos e modelos mais rápidos podem ser mais confiáveis em casos mais genéricos.

4. **Estar pronto para predizer novos exemplos a qualquer momento:** A ideia é que o modelo se adapte aos dados observados da melhor maneira possível, sem a necessidade de ser criado e retreinado novamente a partir do zero. O algoritmo ideal deve aproveitar os dados de treinamento de maneira que não haja motivo para reconstruir o que já foi criado, pois ele já deve estar em sua melhor forma e só precisa aprender novos conceitos de acordo com as instâncias que lhe foram apresentadas.

5. **Mudança de Conceito:** Ao criar um modelo para esse tipo de ambiente, é necessário estar atento ao fato de que o fluxo pode ser dinâmico, ou seja, novos conceitos podem surgir e os antigos podem desaparecer. Essa propriedade será explicada

em maiores detalhes na Seção 2.1.2 e está relacionada com o item anterior no que tange a necessidade de adaptação do modelo conforme novas instâncias se tornam disponíveis.

Essas propriedades devem ser seguidas da melhor maneira possível pois um modelo projetado para fluxos de dados deve sempre permitir que os processos de treinamento e previsão executem praticamente de maneira paralela e de forma quase ininterrupta. Todavia, nem sempre o algoritmo irá conseguir paralelizar realmente essas etapas. Portanto, mesmo que a execução delas seja sequencial, as mesmas devem ocorrer em tamanha velocidade que aparentem um paralelismo, mantendo assim as propriedades elencadas acima. Um modelo baseado em ambientes *online* tem como padrão um fluxo de execução como o apresentado na Figura 2.1. A ideia é que o modelo inicie esperando, por um curto período, um novo grupo de instâncias para serem usadas para treino. Caso não existam instâncias para serem utilizadas, a etapa de treinamento dá prosseguimento ao próximo passo. Caso contrário os exemplos são usados para treino, mesmo que seja somente um. Ao terminar essa etapa, o classificador fica disponível, também por pouco tempo, para predizer novos exemplos que possam chegar e logo após o ciclo volta a se repetir. Esse conjunto de passos ocorre por tempo indeterminado e de maneira ininterrupta. Cada etapa deve ser feita de forma que todas elas aparentem estar sendo executadas em paralelo, respeitando cada um dos requisitos apresentados anteriormente, mas dando preferência para a condição elencada em cada etapa. A única exceção é o requisito 5, o qual não aparece elencado já que pode ocorrer em qualquer etapa e a qualquer momento.

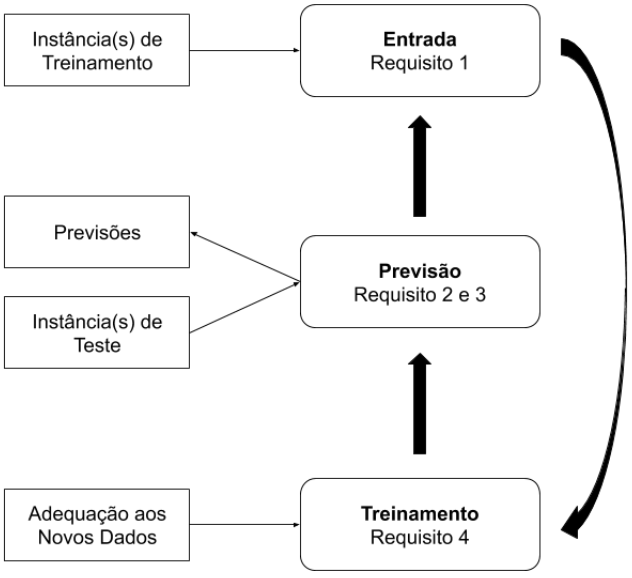

Figura 2.1: Ciclo de Classificação Online, adaptado de (BIFET; KIRKBY, 2009).

2.1.1 Classificadores Base

Os algoritmos a seguir são usados por algumas das técnicas da Seção 3 como base e/ou como parte de seus componentes. Além disso, são estratégias conhecidas da literatura de data stream mining e por isso podem ser usadas como boas referências do quão grande foram os ganhos que novas técnicas tiveram em relação a alguns dos algoritmos que serviram de alicerce para essa área de estudo.

2.1.1.1 Naive Bayes

O *Naive Bayes* (NB) (BISHOP, 2006) é um classificador baseado em análise estatística e utiliza da suposição de que cada atributo das instâncias da *stream* é independente. Muitas vezes um atributo pertencente à dimensionalidade do problema estudado pode ser enviesado por outro. Todavia isso não é considerado pelo algoritmo. Por isso, o Naive Bayes é considerado "naive" (ingênuo), já que o mesmo considera que a probabilidade de dois eventos, E_1 e E_2, acontecerem juntos é matematicamente apresentada como $P(E_1 \cap E_2) = P(E_1) \times P(E_2)$. Ou seja, como dito antes, os eventos (E) são considerados como sendo sempre independentes, não importando o que eles sejam.

Todo o método se baseia no Teorema de Bayes, demonstrado abaixo:

$$P(E_1|E_2) = \frac{P(E_2|E_1) \times P(E_1)}{P(E_2)} \tag{2.1}$$

A ideia do algoritmo é que ao treinar um modelo, o mesmo calcula probabilidades respondendo a seguinte questão: dado um valor de um atributo x^i, qual a probabilidade de que tal valor em uma instância qualquer indique que a mesma pertence à classe y^j, isto é, $P(x^i|y^j)$. Isso é feito para todos os valores de todas as *features* do problema analisado. Outro cálculo é feito também para descobrir qual a probabilidade de uma instância qualquer pertencer à classe y^j, isto é, $P(y^j)$. Todas essas probabilidades são atualizadas a cada rodada de treino e armazenadas dentro do modelo para que as classificações sejam realizadas futuramente seguindo a versão alternativa do Teorema de Bayes que é apresentado na Equação 2.2, onde X representa os valores de atributos de uma instância qualquer, y^k uma classe qualquer existente no problema e e é o número total de classes que o problema analisado possui.

$$P(y^k|X) = \frac{P(y^k) \times \prod_{i=1}^{d} P(x^i|y^k)}{\sum_{j=1}^{e} P(y^j) \times \prod_{i=1}^{d} P(x^i|y^j)} \tag{2.2}$$

A ideia é que ao chegar uma nova instância para predição, os seus valores de

atributos são usados junto com todas as possibilidades de classe do cenário presente na Equação 2.2. Dentre os resultados gerados, o maior valor é escolhido e o valor de classe (y^k) associado ao mesmo é selecionado como a predição do modelo.

2.1.1.2 Hoeffding Tree

A *Hoeffding Tree* (HT) (DOMINGOS; HULTEN, 2000) foi a primeira técnica de árvore de decisão adaptada para o ambiente online. Ela assume que a distribuição dos dados não muda com o passar do tempo e funciona basicamente como qualquer outra árvore de decisão. Todavia, o que faz dessa árvore possível de ser utilizada em cenários de *stream* é o fato da mesma utilizar do limite de Hoeffding ou *Hoeffding Bound*, com o qual é possível demonstrar que as Hoeffding Trees são assintoticamente idênticas às árvores de um algoritmo não incremental, desde que use uma quantidade suficientemente grande de instâncias. Esse limite também proporciona que a árvore expanda seus nós assim que existam evidências estatísticas suficientes para corroborar com o fato de que há um atributo "ótimo" local com ganho de informação suficiente para justificar a transformação de um nó folha em um nó intermediário (de decisão). Além disso, o *Hoeffding Bound* possibilita que essa divisão possa ser verificada com um número relativamente pequeno de instâncias. É importante ressaltar que existem diversas variantes do algoritmo Hoeffding Tree, conforme visto em (CAL; WOŹNIAK, 2013), (BIFET; GAVALDA, 2009) e (MANAPRAGADA; WEBB; SALEHI, 2018). Contudo, somente a versão base é abordada aqui nesse capítulo pois esta implementação é a mais comumente utilizada de forma isolada mas também em *ensembles* (GOMES et al., 2017; CANO; KRAWCZYK, 2020).

Para que esse algoritmo funcione, em cada nó folha é importante que existam informações estatísticas suficientes para que seja possível calcular o ganho de informação que cada atributo pode trazer. Essas informações são usadas de acordo com um hiperparâmetro chamado *grace period*, que é o número de instâncias que devem chegar em um determinado nó da árvore nos períodos de treinamento antes de acionar o limite de Hoeffding para calcular o melhor atributo a ser usado por determinado nó para expandi-lo. É importante lembrar que é possível que o atributo escolhido pelo limite seja o "no-split", ou seja, a não expansão do nó também pode ser uma opção recomendável e permite que exista uma pré-poda na árvore. Além do mais, esse algoritmo conta com uma prevenção de divisão enviesada, evitando que uma expansão de um nó aconteça quando um número muito pequeno de instâncias seguirá por um caminho, eliminando, na teoria, a possibilidade de crescimento da árvore sem ganhos justificáveis. Pelo uso do *Hoeffding Bound* a cada certo período e das atualizações das informações do nó enquanto isso não acontece

que o algoritmo de Hoeffding Tree consegue se encaixar nos requisitos que os cenários de fluxos de dados exigem de seus modelos.

2.1.2 Mudança de Conceito

Como mostrado na Seção 2.1, ambientes de *data stream* impõem diversos desafios para qualquer indivíduo que considere a opção de criar uma instância de inteligência artificial capaz de categorizar corretamente qualquer novo elemento aleatório que possa surgir. Todavia, entre esses desafios, um ganha destaque por ser bastante complexo e possuir diversas maneiras de se enfrentar. A mudança de conceito, ou *concept drift*, é uma propriedade dos fluxos de dados contínuos que pode existir ou não, dependendo do problema analisado, e que confere aos mesmos a característica de serem dinâmicos, ou seja, um conceito pode mudar por influência de diversos fatores ao longo do tempo.

Dada uma *stream* de dados β e uma instância genérica $(X, y) \in \beta$, um conceito pode ser descrito como a função que mapeia os valores de X em algum valor de y, ou seja, $f(X) = y$. Entretanto, nem sempre é possível conhecer todas as d dimensões do espaço de atributos de um problema e como consequência nem todos os modelos criados para derivar um conceito serão 100% precisos. A esse evento se dá o nome de *hidden context* (TSYMBAL, 2004) e um possível resultado da ocorrência do mesmo pode ser descrito como na equação abaixo:

$$g(Z, t_1) \neq g(Z, t_2) \tag{2.3}$$

onde $Z \subset X$ e possui dimensionalidade menor do que d, t_1 e t_2 são momentos diferentes no tempo, e $g(Z)$ é a derivação de $f(X)$ por parte de um modelo com acesso somente à Z. A essa possível ocorrência se dá o nome de *concept drift* (SCHLIMMER; GRANGER, 1986).

Uma mudança de conceito pode ocorrer em diversos cenários. Como exemplo, podemos citar o consumo de energia elétrica de uma região do país. Prever a quantidade de energia que será usada em um cenário desses é algo muito complicado pois envolve muitas variáveis que até hoje são difíceis de se obter e/ou mensurar com bastante precisão, como o clima. Desta maneira, um modelo criado para calcular tal gasto deve conseguir se adaptar às diversas alterações de consumo que podem acontecer ao longo do tempo, como mostrado na Figura 2.2, devido a datas festivas, variações de temperatura, desastres naturais, entre outros.

Outro ponto importante é que as mudanças de conceito podem acontecer em conjunto com diversas formas de desafios que o ambiente pode gerar sobre como os dados se

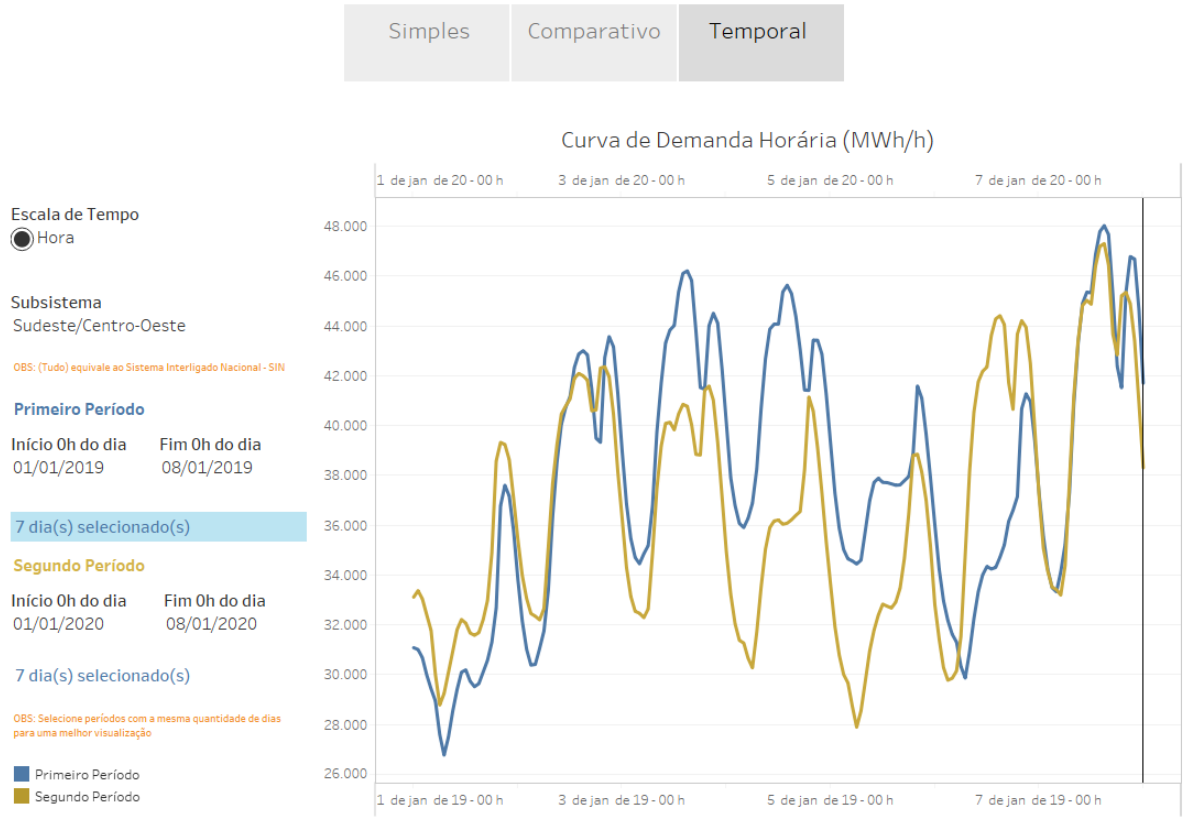

Figura 2.2: Gráfico comparando o consumo de energia elétrica das regiões Sudeste/Centro-Oeste ao longo de dois períodos diferentes de 2019 e 2020. Disponível em <http://www.ons.org.br/Paginas/resultados-da-operacao/historico-da-operacao/curva_carga_horaria.aspx>.

comportam ao longo do tempo. Por isso, é importante que o algoritmo consiga se adaptar a diferentes dificuldades, como as citadas por (CANO; KRAWCZYK, 2020; GAMA et al., 2014; KREMPL et al., 2014; KRAWCZYK et al., 2017; BARDDAL et al., 2017), que incluem:

- **Influência das fronteiras de decisão:** Pode-se destacar que há uma diferença entre *concept drifts* reais e virtuais (GAMA et al., 2014). O primeiro gera mudanças nas fronteiras de classificação, o que pode acarretar o incremento do número de erros de classificação na *stream* analisada. O segundo impacta na mudança da distribuição das *features* ao longo do tempo. Essa mudança não obriga que o classificador seja atualizado, contudo ela pode gerar alarmes falsos para que isso aconteça, gastando tempo e recursos desnecessariamente.

- **Localização dos *drifts*:** Aqui é possível distinguir as mudanças em dois tipos: globais e locais (KREMPL et al., 2014; KRAWCZYK et al., 2017). As globais afetam o fluxo de dados como um todo, e as locais somente um conjunto menor

de classes e/ou uma parte do espaço de atributos. Um classificador que consegue reconhecer isso pode restringir a adaptação do modelo somente à parte que lhe interessar, sem a necessidade de modificar toda a sua estrutura quando for possível.

- ***Concept drifts* abruptos:** Ocorre quando, a partir de uma determinada instância no fluxo de dados o conceito que descrevia o grupo ao qual ela pertence deixa de ser válido e passa a não conseguir mais descrevê-las (GAMA et al., 2014).

- ***Concept drifts* incrementais:** Ocorre quando um conceito vai deixando de ser válido aos poucos enquanto um novo conceito começa a tomar o seu lugar incrementalmente (GAMA et al., 2014).

- ***Concept drifts* graduais:** Ocorre quando ao longo do tempo a validade entre 2 conceitos, um novo e um antigo, oscila. Contudo, quanto mais o tempo passar, maior será a probabilidade do novo conceito ocorrer (GAMA et al., 2014).

- **Recorrência de *drifts*:** Em alguns casos, existe a possibilidade de que um conceito já visto possa reaparecer uma ou mais vezes. A isso se dá o nome de mudança de conceito recorrente (GAMA et al., 2014).

- **Presença de *outliers* e ruído:** Além de identificar e se adaptar a mudanças de conceito, os modelos devem ser capazes de detectar e ignorar dados ruidosos e *outliers* (GAMA et al., 2014). Ruído é todo o dado que pode decorrer do erro de medição ou interferência externa e que não representa a realidade de uma determinada *stream*.

- **Mudança de atributo (*feature drift*):** Existem casos em que um conjunto de atributos deixa de ser relevante e/ou um novo conjunto passa a integrar o espaço de *features*, fazendo com que o modelo tenha que se adaptar ao novo estado do fluxo de dados para evitar que sua taxa de erro cresça (BARDDAL et al., 2017).

2.1.3 Detectores de Mudança

Existem diversos algoritmos utilizados para a detecção das mudanças de conceito em uma *stream*. Eles se encaixam em 4 classificações de funcionamento (DUONG; RAMAMPIARO; NORVRAG, 2018): análise sequencial, controle estatístico do processo, métodos baseados em janelas e abordagens contextuais. Cada um dos detectores existentes podem ser utilizados de diversas formas por diferentes algoritmos, já que são feitos para serem usados em qualquer técnica, independente do formato que eles possuam. Por

isso, um detector pode ser usado não somente para iniciar uma adaptação no modelo ao qual pertence devido a um *concept drift*, mas pode também ser usado para retirar membros de um *ensemble* do processo de predição caso os mesmos estejam inaptos para tal atividade naquele momento.

Uma dessas técnicas de detecção se chama *Adaptive Windowing* (ADWIN) (BIFET; GAVALDA, 2007). Ela consiste em uma análise sobre uma janela de tamanho variável que mantêm as últimas instâncias processadas e as informações de erro para cada uma delas. O tamanho máximo dessa janela é definido durante a execução do próprio algoritmo. Durante o seu funcionamento, essa janela é dividida e se a diferença da média de cada parte for maior do que um limite estabelecido, a parte mais antiga é descartada e um *drift* é detectado.

O *Exponentially Weighted Moving Average for Drift Detection* (ECDD) (ROSS et al., 2012) é também uma técnica baseada em identificar as mudanças na distribuição da taxa de erro, em um ambiente online, para então detectar uma mudança de conceito. A diferença dessa abordagem é que ela compara a média móvel exponencialmente ponderada (EWMA) do erro com a média simples do mesmo para então detectar se um *drift* pode estar ocorrendo.

Por fim, pode-se citar os algoritmos HDDM-A e HDDM-W (FRÍAS-BLANCO et al., 2015). O primeiro se diferencia de outras técnicas de detecção porque utiliza da Desigualdade de Hoeffding, usando a média como estimador do estado do modelo. Já a segunda usa da EWMA como estimador para identificar se o modelo pode estar em mudança de conceito ou não.

2.2 Desbalanceamento

Há situações onde existe uma desproporção entre o número de elementos pertencentes a diferentes classificações. Quando isso acontece de maneira considerável, é dado o nome de desbalanceamento. Geralmente, os cenários onde o desbalanceamento deve ser tratado com cuidado é quando a classe com o menor número de instâncias é a mais importante de ser classificada corretamente, como por exemplo, uma aplicação para detectar nódulos através de imagens, onde é muito mais fácil encontrar fotos de indivíduos saudáveis do que o contrário. A medida que demonstra o nível de desbalanceamento em qualquer cenário é conhecida como *imbalance ratio* (IR). Considerando um problema binário genérico, a classe com a menor porcentagem de representantes é chamada de **minoritária** e a sua contrapartida de **majoritária**. Sendo assim, a IR pode ser calculada

como na fórmula a seguir (ZHU; GUO; XUE, 2020):

$$\text{IR} = \frac{Maj}{Min} \tag{2.4}$$

onde Maj representa a quantidade de exemplos majoritários e Min a quantidade de exemplos minoritários.

Um modelo treinado sobre um ambiente desbalanceado está sujeito a possibilidade de *overfitting* sobre os dados da classe majoritária. Isso pode ocorrer porque a maior parte dos algoritmos convencionais de ML buscam minimizar a taxa geral de erro, tratando o custo de classificar erroneamente ambas as classes igualmente (WANG; PINEAU, 2016). Portanto, considerando um problema onde 95% das instâncias pertencem a uma única classe, atestando um claro desbalanceamento sobre esse ambiente, e um modelo que possui acurácia de 95% sobre tal cenário, o que pode estar ocorrendo é o fato do mesmo estar classificando todas as instâncias de tal cenário como parte da classe majoritária, pois com esse comportamento ele acertará a classificação de 95% das instâncias.

Para lidar com esse tipo de desafio, muitas propostas foram criadas e testadas sob ambientes tradicionais e replicadas para fluxos de dados. Estas técnicas podem ser divididas em quatro grupos (FERNÁNDEZ et al., 2018): *sampling*, *ensembles*, *cost-sensitive* e modificação de algoritmo.

2.2.1 Sampling

A abordagem de *sampling* é conhecida como a mais popular para lidar com o desbalanceamento de dados. Ela consiste na ideia de, entre os dados de treinamento de um modelo, balancear a quantidade de elementos pertencentes a cada classe do problema estudado. Isso é especialmente importante porque muitos dos algoritmos existentes acabam sendo enviesados pelos dados sob os quais são treinados, ou seja, os modelos criados irão aprender sem nenhuma discriminação de qual o nível de desbalanceamento existente e como consequência é possível que ocorra um *overfit* sobre os integrantes da classe majoritária. E como o objetivo dessa estratégia é igualar a quantidade de elementos das classes de um problema para um futuro treinamento, isso gera uma vantagem muito grande pois permite que ela seja usada com qualquer algoritmo existente sem a necessidade de alteração do mesmo, já que o sampling ocorre na etapa de pré-processamento dos dados de treino (NGUYEN; COOPER; KAMEI, 2012).

Dentre as diferentes variações de sampling existentes podemos destacar 3 grandes grupos: *oversampling*, *undersampling* e métodos híbridos. O *oversampling* consiste na ideia de gerar instâncias sintéticas que façam parte da classe minoritária, geralmente a

partir de outras instâncias existentes no conjunto de treinamento, até um ponto em que exista um equilíbrio no desbalanceamento das classes do problema. O *undersampling* também busca equilibrar o IR das classes, mas a estratégia utilizada busca remover instâncias da classe majoritária que não tragam um ganho de informação tão grande. A abordagem híbrida mescla as duas anteriores. O método proposto pelo presente estudo utiliza o *undersampling* como forma de lidar com o balanceamento. Teoricamente essa alternativa pode obter vantagem sobre o *oversampling*, em relação ao tempo de processamento, porque não precisa sintetizar novas instâncias para o conjunto de dados.

Existem diversas variações de sampling que foram criadas e analisadas ao longo do tempo. Todavia, uma das mais conhecidas se trata da técnica SMOTE (CHAWLA et al., 2002). A sua ideia consiste em selecionar uma instância pertencente a classe minoritária aleatoriamente. Em seguida, um dos k vizinhos mais próximos dessa instância é também selecionado. Uma nova instância é então criada em qualquer ponto da linha no espaço de atributos que liga os dois exemplos selecionados anteriormente. Esse processo pode ser usado quantas vezes forem necessárias.

O SMOTE, assim como outras técnicas já criadas, é amplamente utilizado em ambientes tradicionais e de fluxos de dados (GULOWATY; KSIENIEWICZ, 2019; CHEN et al., 2020; LIANG et al., 2020). Todavia, em data stream mining, para todo o processo em que as técnicas de sampling são utilizadas, o mesmo deve sempre estar dentro das restrições que esse cenário apresenta, aumentando a complexidade sobre essas abordagens.

2.2.2 Ensembles

O processo de criação de um modelo busca a aptidão do mesmo em classificar qualquer instância que possa vir do problema estudado. Entretanto, é possível que os dados levem o modelo a aprender muito sobre uma parte do problema e pouco sobre outras. Isso acaba afetando a taxa de acerto e levando a IA a ter pouca generalização em relação ao cenário analisado. Para casos em que é difícil construir uma boa generalização com algoritmos mais tradicionais, existem as estratégias conhecidas como *ensembles*.

Ensembles são conjuntos de modelos que são criados a partir de um mesmo algoritmo base ou de algoritmos diferentes, cada qual podendo ser treinado com partes diferentes dos conjuntos de treinamento. Desta forma, *ensembles* especializam cada um dos seus componentes em uma parte do espaço de classificação do problema estudado, aumentando assim a capacidade de generalização da IA como um todo.

O treinamento de *ensembles* pode ser realizado de diversas maneiras, mas as mais conhecidas são as técnicas de *random subspaces*, *random forests*, *bagging* e *boosting* (HO,

1998; BREIMAN, 2001; OPITZ; MACLIN, 1999). Já a etapa de predição pode usar vários métodos, como voto majoritário, união, entre outros, para unir as classificações individuais de cada membro do *ensemble*, em relação à instância analisada, de maneira a gerar uma única previsão final (ORRITE et al., 2008; KUNCHEVA; RODRÍGUEZ, 2014).

Como a ideia de um *ensemble* é especializar cada um de seus integrantes em cada parte do problema, é possível preparar o mesmo para lidar com o desbalanceamento. Um exemplo genérico seria forçar metade de seus componentes a treinar com mais exemplos da classe minoritária do que o contrário, fazendo com que os votos finais para as classificações sejam mais equilibrados. Como alguns exemplos concretos dessa estratégia, podemos destacar os algoritmos AdaC2 (SUN, 2007) e CSB2 (TING, 2000). Outras estratégias similares e voltadas para data stream mining serão descritas no Capítulo 3.

2.2.3 Cost-Sensitive

Cost-sensitive é uma variação de técnicas usadas em diversos estudos que buscam uma alternativa diferente para lidar com o desbalanceamento ao utilizar os erros de classificação para melhorar a precisão de um algoritmo. O erro gerado por um algoritmo passa a ser usado para criar um custo. Este custo é utilizado como uma maneira de penalizar cada classificador de forma que as classes que tiverem mais erros serão priorizadas no momento do treinamento deles. Como consequência desse processo, uma melhora na capacidade de generalização dos classificadores é ocasionada.

Os custos, ou *misclassification costs* como são conhecidos na literatura, gerados em alguma etapa da execução do modelo, são criados a partir da matriz de custo. Nela, a informação de como criar os custos pode ser dada por um especialista no assunto analisado que irá indicar o método mais confiável de se obter custos mais realistas, por um método heurístico ou através do estudo dos dados de treinamento. Alguns métodos que utilizam dessa estratégia são o CSARF (Seção 3.4), o AdaC2 (SUN, 2007) e o CSB2 (TING, 2000). No Capítulo 3 estão alguns métodos relacionados a data stream mining e que utilizam da abordagem cost-sensitive, incluindo o próprio CSARF, sendo descritos em mais detalhes.

2.2.4 Modificação de Algoritmo

Ao invés de usar estratégias de sampling para balancear o conjunto de treinamento que será usado para treinar um modelo, é possível realizar modificações internas no algoritmo para que ele saiba como lidar com os dados caso eles possuam um grau de desbalanceamento. Para realizar uma alteração desse tipo em um algoritmo, é necessário

que se conheça profundamente os seus mecanismos de aprendizado para que seja possível identificar qual deles deve ser modificado de forma que o mesmo passe a dar preferência à classe minoritária causando um equilíbrio de aprendizado sobre todas as classes de um problema. Por isso, é possível concluir que nenhuma alteração sobre os dados do cenário é feita, mantendo a taxa de desbalanceamento da mesma forma como já estava. Algoritmos adaptados para tratar o desbalanceamento internamente em seu processo de treinamento tendem a ser mais adaptáveis a diversos níveis de desbalanceamento. Contudo a adaptação feita só pode ser aplicada a esse mesmo algoritmo já que os mecanismos internos de cada método podem variar muito. Exemplos de aplicação dessa estratégia são o CSARF e o MMTSSVM (XU, 2017). Tanto o CSARF como outras técnicas relacionadas a data stream mining e que utilizam dessa estratégia de como lidar com desbalanceamento são explicadas no Capítulo 3.

2.3 Avaliação

Todo processo de criação de *software* baseado em inteligência artificial exige que ele seja testado para validar o seu real desempenho. Isso pode ser importante por diversos motivos: verificação da qualidade do modelo gerado, análise dos dados utilizados para o treinamento do modelo, escolha de melhores técnicas de pré e pós processamento, entre outras questões. Contudo, as técnicas existentes para esse fim são muito dependentes do meio no qual elas serão utilizadas.

Em ambientes tradicionais, podemos destacar que as técnicas mais utilizadas para avaliação são: *holdout* e validação cruzada com K dobras (HAN; KAMBER; PEI, 2012). Ambas consistem em utilizar toda a base disponível para a validação das capacidades do modelo construído. Contudo, mesmo que muito parecidos, esses métodos possuem diferenças. O holdout divide a base em duas porções, uma de treinamento e uma de validação, e as usa para adquirir métricas de desempenho sobre a execução do teste. É possível ainda dividir a base em mais uma parte, sendo essa última utilizada em um teste final para averiguar o despenho de um modelo criado após todo o processo de treinamento e validação sobre dados não vistos antes. O modelo de validação cruzada com K dobras funciona de maneira parecida à técnica anterior. Todavia, dado um valor de n, a base disponível é dividida em K partes iguais, sendo $(K-1)$ para treinamento e o restante para teste. A avaliação então é feita novamente K vezes, recriando o modelo a cada rodada, usando sempre uma parte diferente da base original para validação, sem nunca repetir a mesma, e deixando o restante para treinamento. A cada rodada as métricas são aferidas e usadas para realizar uma média dos resultados ao final do processo.

Ambas as técnicas são úteis para o processo de desenvolvimento em machine learning. Entretanto, a escolha de qual abordagem usar deve ser feita conforme o contexto e o objetivo a ser cumprido. Mas em ambientes dinâmicos, essas técnicas não podem ser utilizadas, porque o meio impõe restrições que impossibilitam o armazenamento de todas as instâncias que serão utilizadas para treinar um modelo. Por isso, se faz necessária a utilização da técnica conhecida como prequential e de suas variações (GAMA; SEBASTIAO; RODRIGUES, 2009). Essa técnica consiste em utilizar cada nova instância que chega para avaliar o modelo em execução, atualizando as métricas de desempenho dele ao longo do tempo, e depois para treiná-lo mantendo-o sempre atualizado. Mesmo sendo uma estratégia muito diferente do tradicional, há estudos que comprovam a sua eficácia (GAMA; SEBASTIAO; RODRIGUES, 2013).

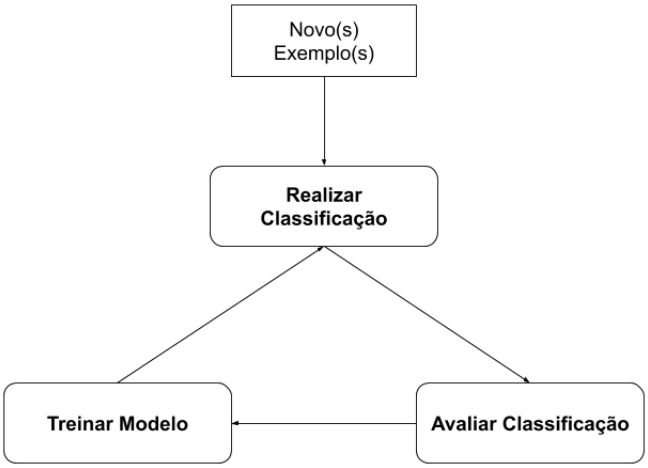

Figura 2.3: Ciclo de execução do Prequential.

Como mostrado na Figura 2.3, o prequential é um processo de validação que funciona em um padrão *test-then-train*, ou seja, cada instância que chegar é usada primeiramente para teste e depois para treino do modelo. A primeira etapa seria esperar novos exemplos chegarem, seja em unidades ou em conjuntos. Depois as classificações seriam feitas para cada instância disponível e em seguida os resultados dessas predições seriam avaliados. Após isso o modelo é treinado com as mesmas instâncias usadas anteriormente.

2.3.1 Métricas

Independente da estratégia escolhida para o processo de validação, o que realmente importa são as mensurações que podem ser obtidas nesse processo, pois eles podem levar às informações de desempenho que um determinado modelo de inteligência artificial possui. Dentre essas informações, podemos destacar a capacidade do modelo de classificar exemplos do problema estudado, o quão eficiente o modelo é em casos de desbalancea-

mento, se o modelo está passando por uma mudança de conceito, entre outras. Contudo, cada métrica geralmente leva a diferentes informações. Portanto, escolher as métricas certas para chegar ao objetivo desejado é algo de suma importância. Por exemplo, se um modelo está errando muito em uma das classes presentes no problema estudado, então o primeiro passo a ser feito é buscar obter as métricas que demonstrem o porquê de isso estar acontecendo e o quão ineficiente o classificador está realmente sendo para esse caso.

Para conhecer um pouco mais sobre métricas é importante conhecer sobre alguns termos utilizados para alguns dados obtidos através de avaliações de modelos. O primeiro, e mais importante de todos, é a matriz de confusão (Tabela 2.1), onde é possível ver um panorama geral de erros e acertos do modelo testado. Dentro dessa matriz podemos obter alguns dados das predições já realizadas que são os números de votos **Verdadeiro Positivos** (**VP**), **Verdadeiro Negativos** (**VN**), **Falso Positivos** (**FP**), **Falso Negativos** (**FN**), **Preditos Positivos** (**PP**), **Preditos Negativos** (**PN**), **Reais Positivos** (**RP**), **Reais Negativos** (**RN**) e **Totais** (**T**) (HAN; KAMBER; PEI, 2012). É importante salientar que positivo e negativo é uma terminologia alternativa usada na literatura e se referem respectivamente à instâncias das classes minoritárias e majoritárias. Além disso, verdadeiro está relacionado a votos corretamente realizados e falso à votos incorretos.

Predito / Real	Classe Positiva	Classe Negativa	Totais Reais
Classe Positiva	VP	FN	RP
Classe Negativa	FP	VN	RN
Totais Preditos	PP	PN	T

Tabela 2.1: Exemplo de matriz de confusão.

É importante lembrar que quase todas as métricas apresentadas nesse trabalho possuem valores entre 0 e 1, sendo 1 o valor que indica a melhor performance. As métricas conhecidas como AUC-PR e AUC-ROC se baseiam em áreas abaixo da curva, as quais são feitas relacionando diferentes métricas (SAITO; REHMSMEIER, 2015), porém também seguem o mesmo princípio de valores finais. As exceções são o MCC e o Kappa (BRZEZINSKI et al., 2020), com seus possíveis valores sendo mostrados a seguir. Abaixo se encontra a formulação matemática, quando existente, e uma respectiva definição das métricas que serão utilizadas e/ou citadas ao longo desse estudo:

- **Acurácia:** Definida como a proximidade do número de predições corretamente realizadas experimentalmente e o número total de exemplos avaliados.

$$\frac{VP + VN}{VP + VN + FP + FN} \tag{2.5}$$

- **Especificidade (*Specificity*):** Indica a proporção dos elementos negativos que foram corretamente classificados.

$$\frac{VN}{VN + FP} \tag{2.6}$$

- **Acurácia Balanceada:** Essencialmente é igual a métrica da Acurácia (Item 2.3.1), porém o cálculo é ponderado pelo número de exemplos de cada classe. Mais especificamente, a Acurácia Balanceada é a média aritmética de Recall (Item 2.3.1) e Specificity (Item 2.3.1).

$$\frac{\frac{VP}{VP+FN} + \frac{VN}{FP+VN}}{2} \tag{2.7}$$

- **Kappa:** Diferencia o custo dos erros no momento da validação. Utiliza os valores da matriz de confusão e de uma matriz de pesos, com valores para cada uma das possíveis posições dentro da matriz de confusão, para realizar os cálculos. Os valores dessa métrica vão de -1 até 1, sendo 0 igual a uma escolha aleatória e quanto mais próximo de 1 melhor o classificador será.

$$\frac{\text{Acurácia} - \frac{1}{T} \times (\frac{RP \times PP}{T} + \frac{RN \times PN}{T})}{1 - \frac{1}{T} \times (\frac{RP \times PP}{T} + \frac{RN \times PN}{T})} \tag{2.8}$$

- **G-mean:** É uma métrica que usa informações de todas as classes e foca no balanço entre performances individuais de classificação de todas as classes. Mais especificamente, essa métrica é a média geométrica de Recall (Item 2.3.1) e Specificity (Item 2.3.1).

$$\sqrt{\frac{VP}{VP + FN} \times \frac{VN}{VN + FP}} \tag{2.9}$$

- **Precisão (*Precision*):** Avalia a fração de instâncias classificadas corretas entre as classificadas como minoritárias.

$$\frac{VP}{VP + FP} \tag{2.10}$$

- **Revocação (*Recall*):** É a fração do total de instâncias positivas corretamente classificadas como minoritárias.

$$\frac{VP}{VP + FN} \tag{2.11}$$

- **F1-score:** Foca na análise das classes minoritárias e busca analisar a relação entre o quão correto e qual é a cobertura na classificação dessas classes. É conhecida também como a média harmônica de Precision (Item 2.3.1) e Recall (Item 2.3.1).

$$2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.12}$$

- **MCC:** Medida criada na bioinformática que é uma adaptação do coeficiente de correlação de Pearson para avaliar a correlação dentro de matrizes de confusão. Os valores vão de -1 (classificação sempre incorreta) até 1 (classificação sempre correta). O valor 0 indica que o classificador é igual a escolha aleatória.

$$\frac{VP \times VN - FP \times FN}{\sqrt{PP \times RP \times PN \times RN}} \tag{2.13}$$

- **AUC-ROC:** É uma medida baseada na curva ROC, a qual é uma curva de probabilidade que associa a taxa de VPs (Recall) (Item 2.3.1) contra a taxa de FPs ($1 - Specificity$) (Item 2.3.1), e que busca mostrar o grau de separação dos eixos dessa curva. A informação obtida através dos resultados dessa medida é o quanto um modelo é capaz de distinguir entre as classes. Quanto maior for o valor dessa métrica, melhor o modelo realiza essa atividade. Os possíveis resultados vão de 0 até 1, sendo 0 o indicativo de que o modelo está sempre errando, 0,5 aponta que o modelo classifica igual a uma escolha aleatória e 1 mostra que o modelo sempre acerta.

- **AUC-PR:** O AUC-PR funciona de maneira similar à métrica de AUC-ROC (Item 2.3.1). A diferença está em quais métricas são usadas para realizar a associação. Para esta medida, o Precision (Item 2.3.1) é associado contra a taxa de VPs (Recall) (Item 2.3.1). Essa métrica é mais indicada que o AUC-ROC para ambientes desbalanceados (SAITO; REHMSMEIER, 2015).

- **Tempo de Processamento:** O Tempo de Processamento é basicamente o tempo em atividade que o CPU é utilizado para as tarefas relativas ao processamento de um modelo.

- **Custo do Modelo:** O Custo do Modelo é a medida que demonstra o custo de memória que um modelo demanda de um sistema computacional para funcionar.

Todas as métricas existentes podem ser usadas em qualquer cenário. Entretanto, algumas delas revelam informações mais valiosas que outras dependendo do contexto. Isso

não é diferente para casos de desbalanceamento, onde medidas mais comumente usadas, como a acurácia, não revelam sozinhas informação o suficiente para indicar que um modelo está desempenhando bem. Por exemplo, considerando um ambiente fictício onde 90% das instâncias pertencem à classe majoritária e os 10% restantes à classe minoritária, se um modelo baseado em um algoritmo tradicional for utilizado nesse ambiente e o mesmo aprender a sempre atribuir todas as predições à classe majoritária, o mesmo terá uma acurácia de 90% e isso não indica que o modelo possui uma boa capacidade de generalização para todas as classes do problema. Além disso, casos em que o desbalanceamento é considerado um problema são geralmente aqueles em que classificar corretamente os integrantes da classe minoritária é mais importante. Portanto, considerando o estudo de (BRZEZINSKI et al., 2020), que analisa algumas métricas amplamente usadas em cenários desbalanceados e o efeito que índices de desbalanceamento diferentes causam sobre as mesmas, que a lista de métricas acima foi selecionada. É importante notar também que não há uma métrica ideal para todos os casos de desbalanceamento e geralmente é necessário definir a mais apropriada para cada cenário de aplicação e suas especificidades.

2.4 Considerações Finais

Esse capítulo buscou apresentar uma introdução a *data stream mining*, bem como alguns dos desafios e limitações que as pesquisas inseridas nesse ambiente devem levar em consideração ao serem realizadas. Além disso, o tópico de desbalanceamento em meio a fluxos de dados contínuos também foi abordado, elencando alternativas de como os algoritmos lidam com ele, métricas para avaliar o quão bem os modelos utilizados sobre esse ambiente podem lidar com o desbalanceamento, entre outros tópicos relativos a esse desafio.

No Capítulo 3, as diferentes técnicas que serão utilizadas para análise e posterior comparação no presente estudo serão apresentadas em detalhes. Todas elas estão inclusas no grupo de algoritmos pertencentes a mineração de fluxos de dados e várias delas são estratégias preparadas para lidar com desbalanceamento.

Capítulo 3

Trabalhos Relacionados

Esse capítulo tem como objetivo apresentar diversas técnicas de data stream mining que serão explicadas e utilizadas em experimentos que servirão como comparativo para os futuros desenvolvimentos do presente trabalho. Grande parte dos algoritmos aqui presentes também possui algum mecanismo para lidar com desbalanceamento em fluxos de dados.

3.1 *Kappa Updated Ensemble*

A técnica conhecida como *Kappa Updated Ensemble* (KUE) (CANO; KRAWCZYK, 2020) é baseada em boas práticas já utilizadas em outros estudos como *random subspaces with varying size*, *online bagging*, *dynamic weighted voting* e *incremental learning*. Mesmo sendo similar a outros algoritmos, como o Adaptive Random Forest (Seção 3.2) e o Accuracy Updated Ensemble (BRZEZIŃSKI; STEFANOWSKI, 2011), o KUE possui algumas peculiaridades em relação a inicialização dos componentes do *ensemble*, o tamanho dos *random subspaces* ser variável para cada classificador base, incorporação de novos classificadores quando necessário, abstenção de classificadores na predição e o uso da métrica Kappa (2.3.1) para atribuir uma medida de "peso" a cada componente do *ensemble*.

Sendo ϵ um *ensemble* formado por M classificadores base, eles são inicializados com a primeira carga de dados que chega da stream β, deixando todos prontos para uso imediato. Mas para que haja diversidade entre os diferentes classificadores é atribuído a cada componente um subespaço de features aleatório de dimensionalidade f, sendo $1 \leq f \leq d$, e também um bagging onde cada nova instância s recebe um peso w de acordo com Poisson(1) e é possível ocorrer a ação de resampling com substituição.

Ao finalizar o processo de treinamento e avaliação dos componentes do *ensemble*, que ocorre a cada vez que chegar um novo conjunto de dados de treino, as suas respectivas

métricas de Kappa são usadas para a fase de votação, como um peso de confiança no voto de cada um, para cada nova instância que precisar ser avaliada. Isso ocorre porque o Kappa pode ser usado como uma medida de competência do classificador em relação ao estado atual da *stream* β. Ou seja, essa métrica oferece um bom panorama de como o *ensemble* se comporta com variações em β. A equação que representa como o processo de voto do *ensemble* do KUE funciona está a seguir:

$$y = \arg\max \sum_{i=1}^{M} = \begin{cases} \kappa_i \times \text{Predição}(\epsilon, i, X), & \text{se } \kappa_i \geq 0 \\ 0, & \text{caso contrário} \end{cases} \tag{3.1}$$

sendo κ_i o valor de Kappa para o classificador base i do *ensemble* ϵ e Predição(ϵ, i, X) a classificação dada pelo membro i do *ensemble* ϵ para os valores de atributos X da instância avaliada.

Um fator importante sobre o KUE é que ele foi desenvolvido para ser adaptável. A métrica Kappa, além de conferir peso sobre os votos individuais de cada classificador, funciona como um limitador de quais componentes do *ensemble* irão participar da próxima votação de exemplos que chegarem. Ou seja, o KUE possui um mecanismo de abstenção de componentes caso os mesmos possuam um Kappa ≤ 0. Além disso, esse algoritmo possui um outro mecanismo de incorporação de novos membros do *ensemble*, onde ele se encarrega de treinar e substituir membros antigos que não estejam desempenhando bem.

3.2 Adaptive Random Forest

O algoritmo conhecido como *Adaptive Random Forest* ou (ARF) (GOMES et al., 2017) é uma técnica muito conhecida no âmbito de mineração de fluxos de dados. O ARF foi criado como uma adaptação do algoritmo de *Random Forest* (RF) (BREIMAN, 1996) para cenários online, o qual é uma técnica comumente utilizada em ambientes tradicionais.

Em *data streams*, como não é recomendado que uma mesma instância seja analisada muitas vezes, o ARF utiliza uma simulação de amostragem com reposição sobre elas para realizar a criação de um *ensemble* de Hoeffding Trees. Isso é realizado de maneira similar ao que é feito pelo RF, o qual gera o crescimento das árvores que o compõem com diferentes subconjuntos de instâncias que são processadas mais de uma vez. Além disso, a amostragem usada pelo ARF faz parte de um processo de bagging tradicional adaptado ao contexto online. Cada nova instância a ser usada para treinamento recebe, para cada árvore do *ensemble*, um peso w^i que é baseado na distribuição de Poisson$(\lambda = 6)$. Caso o w^i seja maior do que 0, a instância é usada para treinar o respectivo membro do *ensemble*

w^i vezes. Ao usar o valor de $\lambda = 6$, fica garantido que cada instância será usada na média 6 vezes para o treinamento, garantindo uma fixação do conhecimento maior do que ao usar um λ menor e evitando o *overfitting* caso usasse valores maiores desse parâmetro. Todavia, o valor de λ pode ser alterado conforme a necessidade, já que isso pode eventualmente ser interessante em casos específicos. Para o processo de predição, a classificação final se dá pelo voto majoritário ponderado, onde cada classificador do *ensemble* usa a sua acurácia individual como um peso para os seus votos, de maneira que os membros que classificaram corretamente mais instâncias possuirão mais ênfase na decisão final do *ensemble*.

3.3 Adaptive Random Forests with Resampling

A técnica conhecida como *Adaptive Random Forest with Resampling* (ARFRE) (FERREIRA et al., 2019) foi criada como uma adaptação do ARF (Seção 3.2), procurando lidar melhor com os ambientes em que o desbalanceamento de classes ocorre. A única diferença entre os dois algoritmos se dá no momento do treinamento das árvores do *ensemble*, mais especificadamente em como o peso calculado para cada instância é realizado. O ARFRE realiza esta ponderação usando a equação abaixo:

$$\text{Peso}(X, y) = \frac{100 - \frac{C^y \times 100}{C}}{100} \times \text{Poisson}(\lambda) \tag{3.2}$$

onde C^y representa o número de instâncias da classe y e C representa a quantidade total de todas as instâncias que foram recebidas da stream.

Com essa mudança baseada em sampling, as classes minoritárias tendem a ser selecionadas para treino mais vezes em relação ao que ocorre no ARF original, buscando equilibrar o aprendizado do *ensemble*, evitando assim o sobre-ajuste sobre a classe majoritária.

3.4 Cost-sensitive Adaptive Random Forest

Cost-sensitive Adaptive Random Forest (CSARF) (LOEZER et al., 2020), assim como o ARFRE (Seção 3.3), é uma adaptação do ARF (Seção 3.2) criada para lidar com desbalanceamento. A grande diferença aqui é que uma estratégia baseada no custo do erro de classificação é empregada. Todavia, há mais mudanças feitas sobre o ARF para culminar no CSARF. Estas são elencadas a seguir:

- Inclusão de uma janela deslizante sobre o *ensemble* para verificar a distribuição das classes ao longo do tempo.

- Alteração no processo de treinamento para forçar com que todas as árvores do *ensemble* treinem com instâncias da classe positiva.

- A troca da medida de acurácia pela métrica conhecida como *Matthews Correlation Coefficient* (Seção 2.3.1) para ser usada como o peso de cada um dos membros do *ensemble*. Houve uma alteração mais atual no algoritmo, que passou a utilizar o F1-score (Seção 2.3.1) no lugar do MCC, mas ela está no processo para ser publicada.

- A adição de um mecanismo de sensibilidade aos custos de classificações incorretas. Esse mesmo mecanismo conta com duas variações denominadas **local** e **global**.

Para os procedimentos em que é necessário contar com a predição do *ensemble*, os membros do mesmo com o melhor desempenho possuem maior importância ao participarem da classificação final, que é baseada em voto majoritário. Contudo, de acordo com o algoritmo original do ARF, como a performance das árvores é baseada na acurácia delas, isso implica em uma priorização ineficiente dos supostos melhores membros em ambientes desbalanceados. Isso ocorre porque a acurácia por si só não é um bom indicativo de que um classificador está conseguindo realizar predições confiáveis para todas as classes de um problema conforme exemplificado ao final da Seção 2.3.1. Por isso, para cenários com desbalanceamento, o uso do MCC se sobressai, pois, a mesma considera as proporções de cada classe em seu cálculo. Desta forma, os classificadores dentro do CSARF terão prioridade caso possuam uma boa generalização e acerto em todas as classes do problema avaliado. Portanto, o MCC é usado no lugar da acurácia como um peso sobre o processo de voto majoritário, através de duas variações possíveis do presente algoritmo:

- **Local:** Essa opção utiliza o peso para influenciar a predição de cada árvore do *ensemble* individualmente. Portanto, isso pode não ter efeito algum sobre a predição final. Em resumo, o voto majoritário é realizado sobre o resultado da associação do MCC individual de cada membro i do *ensemble* com a classe y_i que minimiza a soma das probabilidades alternativas, ou seja, a probabilidade das outras classes, da instância avaliada. Em outras palavras, cada classificador base tem a sua predição, que é um vetor de probabilidades, associada com os seus respectivos custos, que vem de suas matrizes de custos, antes da utilização da estratégia de voto majoritário ponderado sobre todos os votos.

- **Global:** Essa versão utiliza o peso de uma maneira similar a versão local. Entretanto, a diferença se encontra no fato de que após o cálculo dos votos majoritários, a classe y que minimiza a soma da probabilidade das outras classes da instância avaliada é escolhida como o resultado. Em outras palavras, os votos de cada classificador

base do *ensemble* são combinados através do voto majoritário e após isso a matriz de custos é aplicada sobre o vetor de probabilidades que representa a combinação desses votos.

Os custos de erros de classificação para cada classe, usados principalmente para calcular o MCC, são calculados de maneira simples como mostrado abaixo:

$$\texttt{Peso}(y) = \frac{\#\texttt{Instâncias}}{\#\texttt{Classes} \times \#\texttt{Instâncias de } y} \tag{3.3}$$

Para calcular esses erros é necessário conhecer quantas instâncias existem no problema estudado segundo a Equação 3.3. Contudo, como não é possível conhecer todos os elementos de uma stream de dados, usa-se a janela deslizante do CSARF como base para a realização desses cálculos.

Na etapa de treinamento, o ARF pode gerar casos em que nenhuma instância positiva seja apresentada a alguns de seus membros, pois as classes não são diferenciadas. Por esse motivo, o CSARF conta com uma janela deslizante para poder verificar se uma instância corrente qualquer é positiva dentro da mesma. Com isso, as instâncias da classe minoritária são usadas mesmo se o resultado de Poisson($\lambda = 6$) for igual a 0.

3.5 OzaBag

O *OzaBag* (OZA, 2005) é um *ensemble* que porta a estratégia do Bagging tradicional (BREIMAN, 1996) para ambientes online. A ideia do bagging consiste em no período de treinamento de um *ensemble* ϵ com M membros que são diretamente mapeados a *bags* de instâncias que são gerados. Os pacotes serão compostos por conjuntos de instâncias diferentes. As instâncias para cada um desses conjuntos são selecionadas a partir dos exemplos de treinamento de acordo com a seguinte lógica: cada instância de treino é selecionada F vezes para fazer parte do bag de um dado membro do *ensemble*, sendo que F é um valor resultante de uma variável aleatória $P(F)$ que segue uma distribuição binomial. Essa ideia é executada para cada um dos componentes de ϵ. Portanto, há a possibilidade de ocorrência de reposição e a não existência de algumas instâncias em alguns dos *bags*. Os conjuntos finais gerados são então usados para treinar cada um de seus respectivos classificadores.

Com o *bagging* tradicional, busca-se gerar diversidade dentro do *ensemble* para que cada um de seus componentes consiga se especializar em partes diferentes do problema, aumentando a capacidade de generalização do modelo como um todo. Todavia, em ambientes de fluxos de dados não é possível conhecer toda a base de dados. Consequentemente,

esse algoritmo não pode ser utilizado diretamente. Como adaptação desse processo, se usa a distribuição de $Poisson(\lambda = 1)$, que é estatisticamente muito próxima de uma distribuição binomial. Ao chegar uma nova instância de treinamento, um número aleatório $F = Poisson(\lambda = 1)$ é criado para cada membro do *ensemble*. Então, esse exemplo será usado F vezes para treinar o seu respectivo classificador, seguindo a mesma estratégia do algoritmo original de *bagging*.

3.6 ADWIN Bagging

O *ADWIN Bagging* (BIFET et al., 2009) é uma técnica que busca combinar o OzaBag (Seção 3.5) com o detector de mudança conhecido como ADWIN (Seção 2.1.3). Esse detector mantém uma janela, de tamanho variável definido pelo próprio algoritmo, com informações das últimas instâncias analisadas. O tamanho da janela pode diminuir após o ADWIN realizar a divisão dela em algum ponto e comparar a média estatística coletada das informações armazenadas de cada parte. Com isso, se o valor absoluto da diferença dessas médias for maior que um limite previamente definido, uma mudança de conceito (*concept drift*) é detectado e a parte mais antiga da janela é descartada. Com essa detecção, o classificador do *ensemble* de menor acurácia é então removido e substituído por um novo modelo.

3.7 OzaBoost

O *OzaBoost* (OZA, 2005), publicado no mesmo trabalho que o OzaBag (Seção 3.5), usa o mesmo princípio que este último. A diferença está no fato de que o OzaBoost procura replicar a ideia de um boosting tradicional, que é basicamente um bagging onde após a criação do primeiro bag e treinamento do primeiro membro do *ensemble*, todas as instâncias recebem um peso baseado no erro obtido ao realizar o teste de desempenho que é feito ao fim do processo de treinamento. Erros maiores indicarão pesos maiores. Com isso, o próximo bag é criado para o próximo modelo do *ensemble*. Entretanto, dessa vez as instâncias com os maiores pesos terão mais chances de serem selecionadas para compor o novo bag. Esse processo continua se repetindo até o último classificador, sempre readaptando os pesos das instâncias conforme os novos modelos forem sendo treinados.

Para realizar o processo de boosting, é necessário que todos os exemplos da base de dados estejam disponíveis inicialmente, mas em cenários de data stream isso não é possível. Como adaptação a isso, o OzaBoost usa a mesma estratégia que o OzaBag com

$F \sim Poisson(\lambda = 1)$. Além disso, esse algoritmo também utiliza a estratégia de pesos para cada instância sendo utilizada para treino, onde F tem seu valor modificado pelo peso da sua respectiva instância. Por consequência, isso altera a chance de um exemplo ser usado mais vezes para treino quanto mais alto forem os erros associados a si. Esses pesos são repassados para o próximo modelo do *ensemble* e modificados conforme os resultados do teste desse membro. O processo se repete até o último classificador.

3.8 Leveraging Bagging

O *Leveraging Bagging* (LB) (BIFET; HOLMES; PFAHRINGER, 2010) é conhecido como uma adaptação sobre o ADWIN Bagging (Seção 3.6). Este algoritmo usa do mesmo procedimento de funcionamento que a técnica na qual é baseada, mas usa $Poisson(\lambda = 6)$ para o processo de reamostragem durante o treinamento e códigos de detecção de saída (*output codes*) para aumento de diversidade. Esses códigos funcionam da seguinte maneira: uma matriz de *bits* é construída onde o número de linhas é igual ao número de classificadores no *ensemble* e o número de colunas é igual ao número de classes do problema sendo tratado. A Tabela 3.1 apresenta um exemplo de matriz para três classificadores e três classes. Nesta tabela, associa-se uma sequência de *bits* para cada classificador, sendo que durante a geração das sequências de bits, a quantidade de *bits* de valor 1 por classe deve ser equilibrada.

Tabela 3.1: Exemplo de tabela de códigos de detecção de saída.

	Classe 1	Classe 2	Classe 3
Classificador 1	0	0	1
Classificador 2	0	1	1
Classificador 3	1	0	0

Quando uma nova instância é processada em tempo de treinamento, ela tem sua classe original convertida com o *bit* associado para cada classificador, o que induz diversidade entre os modelos, uma vez que cada classificador dicotomiza as classes existentes de forma diferente. Durante o teste, todos os classificadores realizam as suas respectivas previsões, e a classe original com o maior número de 1s preditos usando a máscara de *bits* é tida como a previsão final do *ensemble*.

3.9 LearnNSE

O *LearnNSE* (ELWELL; POLIKAR, 2011) é um algoritmo criado a partir de uma família de outros algoritmos conhecidos como Learn^{++}. Esses algoritmos são baseados em *ensembles*, treinados incrementalmente sem acesso aos dados anteriormente vistos, e seus classificadores membros são combinados com alguma forma de voto majoritário ponderado. O que geralmente diferencia esse grupo de algoritmos é como a regra de distribuição de dados para treinamento do *ensemble* vai acontecer e de que maneira os pesos dos votos vão ser gerados. O LearnNSE foi feito principalmente para lidar com ambientes não estacionários, ou seja, que possuem *concept drifts*, e por isso possui um mecanismo de detecção passiva para tais eventos. Além disso, é um algoritmo que possui tratamento para cenários com aparecimento e desaparecimento de classes.

Essa técnica inicia com um único membro no *ensemble*. A partir do momento em que novos dados de treinamento chegam, o *ensemble* é avaliado sobre o mesmo. As instâncias que não forem classificadas corretamente pelo *ensemble* são então identificadas e um novo classificador que irá fazer parte do mesmo é treinado com os novos dados de treinamento. Então, todos os membros do *ensemble*, incluindo o recém criado, são reavaliados. Contudo, a penalidade por classificar incorretamente as instâncias que foram identificadas anteriormente é reduzida no cálculo de erro, dando maior importância aos membros que conseguirem classificar esses exemplos. A seguir, o erro de cada classificador é ponderado, dando ênfase para a taxa de erro mais recente, e então o mesmo é utilizado para também atribuir pesos aos votos de cada membro. Um fato importante é que se algum classificador do *ensemble* não desempenhar bem, o mesmo receberá um peso muito baixo, ou até nulo, mas ele não será descartado porque futuramente o mesmo poderá obter um peso maior conforme a stream evoluir. Por isso, pode-se dizer que o LearnNSE somente se esquece temporariamente. Por fim, a predição final é feita por um voto majoritário ponderado pelo peso de cada membro. Para que o *ensemble* não cresça infinitamente existem modos de podá-lo. Um dos métodos funciona de maneira que ao chegar no tamanho máximo do *ensemble*, que é parametrizável, o classificador membro mais velho é descartado. O outro funciona quase da mesma maneira que o anterior, com a diferença de que o membro com o maior erro é descartado.

3.10 OnlineADAC2

O *OnlineAdaC2* (WANG; PINEAU, 2016) é um algoritmo baseado em outro chamado de simplesmente *AdaC2* (SUN, 2007). Ambas as versões são baseadas em boos-

ting e usam os diferentes custos de erro de classificação para atribuir pesos diferentes às instâncias, fortalecendo mais o treinamento sobre instâncias das classes positivas, ou minoritárias. A ideia desse algoritmo adaptado para o ambiente online segue muito do que é feito em outras técnicas, como o OzaBoost (Seção 3.7), e utiliza da distribuição de $Poisson(\lambda = 1)$ para simular um sampling e agrega a isso o uso de pesos para priorizar certas instâncias. A diferença em relação ao OzaBoost está em como o peso é gerado, pois o Online AdaC2 conta com uma série de cálculos matemáticos a mais que levam em consideração os custos de erros de classificação para enfatizar mais o treinamento sobre instâncias minoritárias. Além disso, para identificar mudanças de conceito, o detector ADWIN é empregado, sendo usado um para cada membro do *ensemble*.

3.11 OnlineUnderOverBagging

O *OnlineUnderOverBagging* (OnlineUOB) (WANG; PINEAU, 2016) é uma outra adaptação de um algoritmo de *batch* chamado de UnderOverBagging (WANG; YAO, 2009). Esse algoritmo busca realizar o sampling de duas formas simultaneamente: undersampling sobre a classe negativa e oversampling sobre a classe positiva. Essa técnica também possibilita a variação da taxa de sampling o que pode aumentar a diversidade do *ensemble*. Além disso, ela também usa o ADWIN para identificar mudanças de conceito, utilizando um detector por cada membro do *ensemble*. Essa adaptação do algoritmo de UnderOverBagging de *batch* só foi feita porque foi possível constatar o seguinte:

$$P\left(\frac{V}{L}\right) \approx Poisson\left(\lambda = V\right) \tag{3.4}$$

onde $P\left(\frac{V}{L}\right)$ é uma distribuição binomial, V é a taxa de amostragem e L é a quantidade de exemplos na base de dados do algoritmo de *batch*.

3.12 Considerações Finais

O presente capítulo teve como intuito apresentar diferentes técnicas de data stream mining que abordaram ou não estratégias para lidar com o desbalanceamento que pode existir em cenários de fluxos de dados.

Dentre todas as técnicas, é possível observar que a estratégia presente em todos os algoritmos mostrados é o uso de *ensembles*. Essa é uma opção muito usada em mineração de dados sobre diferentes contextos, pois trás a ideia de "dividir para conquistar" especializando cada um dos seus elementos sobre diferentes áreas do problema a ser classificado.

Isso também auxilia no processo de tratamento do desbalanceamento quando presente. Entretanto, não foi possível encontrar nenhum algoritmo que chegue a utilizar a técnica de *stacking* sobre o seu funcionamento, visto que a maioria usa alguma variação dos votos majoritários simples ou ponderados para o unir os votos dos membros dos *ensembles* em uma classificação final. Outro ponto a ser mencionado é o uso de abordagens *cost-sensitive*, como no KUE e no CSARF. Elas compreendem a maior parte dos algoritmos selecionados. Fica claro que há eficiência nessa técnica visto a sua frequência de uso, fora o resultado dos estudos individuais de cada algoritmo que comprovam o mesmo. Mas mesmo sendo muito comum na maior parte dos algoritmos da literatura estudados, pode trazer uma complexidade de cálculo a mais para o algoritmo que as vezes pode não ser justificável. Estratégias de *sampling* também são comuns em suas mais variadas formas. Porem, não foi identificado nenhum uso de *inverse random under and over sampling* (Seção 4.2) e nem de suas variações em algoritmos de *data stream mining*, mesmo sendo um abordagem que busca retirar o desbalanceamento do treinamento de modelos, evitando a sua super especialização em só um espectro das classes de um problema. Um dos possíveis motivos para isso é a necessidade de incorporar o processo feito por algoritmos como o OzaBag para simular a criação dos conjuntos de treinamento necessários para o processo, que é muito próximo de um *bagging* tradicional. Isso pode não ser muito simples e talvez nem viável de fazer em todos os algoritmos. O mais próximo disso é realizado pelo OnlineUnderOverBagging e portanto ainda não era possível validar a efetividade dessa técnica. Por fim, é importante mencionar que mesmo sendo possível encontrar algoritmos que utilizem *ensembles* heterogêneos e *stacking* na literatura de fluxos de dados, não foi encontrada nenhum algoritmo que unisse ambos com o *inverse random under and over sampling*.

No Capítulo 4 será apresentada a estratégia a ser implementada por esse projeto, demonstrando as opções de estudo existentes, as adaptações a serem feitas para a criação da mesma, entre outras questões necessárias para validar a proposta de pesquisa aqui exposta.

Capítulo 4

Proposta

Existem diversas técnicas que focam no problema de desbalanceamento. Conforme discutido no capítulo anterior, diversas adaptações de algoritmos de ambientes batch já foram realizadas para focar em ambientes de *data streams*. Contudo, um detalhe importante que ocorre nessas adaptações é que nem sempre uma técnica que funciona bem em cenários tradicionais irá desempenhar bem sobre fluxos de dados. Isso pode acontecer devido a diversos motivos, mas principalmente pela natureza volátil que as streams possuem de modificar seu comportamento com o passar do tempo (*concept drift*), do próprio desbalanceamento, da adição de novas *features* ao problema analisado, entre outros motivos.

Dentre as abordagens criadas para lidar com o desbalanceamento entre classes, podemos encontrar várias alternativas dentre os grupos elencados nas subseções do Capítulo 2.2, isto é, métodos baseados em sampling, *ensembles*, cost-sensitive e modificações de algoritmo. As possibilidades de criação de novas estratégias para tratar o desbalanceamento são inúmeras para cada uma dessas opções individuais. Todavia, o que mais comumente se encontra na literatura são variações combinando essas possíveis abordagens. Por exemplo, o algoritmo CSARF (Capítulo 3.4) que utiliza uma abordagem baseada em *ensembles* combinada a uma estratégia *cost-sensitive*.

Existem várias restrições relevantes que devem ser levadas em consideração em qualquer algoritmo que processe dados de uma stream, as quais estão elencadas na Seção 2.1. Esses requisitos devem ser atendidos pois qualquer ambiente baseado em fluxos de dados contínuos irá exigi-los, senão o processamento de seus dados se tornará inviável. Entretanto, dentre cada uma das 5 restrições assinaladas, há dois fatores que se mostram mais recorrentes: a mínima utilização de tempo de processamento e de memória. Nessa proposta, ressaltamos técnicas que auxiliam no atendimento destes fatores focando no problema de desbalanceamento. A primeira é o uso de *undersampling*, que permite

balancear os dados sem a necessidade de sintetizar novos elementos, evitando uma seleção detalhada que pode ter um alto custo computacional. Contudo, é importante notar que nem todas as variações de undersampling irão trazer um baixo custo, como visto na técnica de Tomek Links (TOMEK, 1976), que requer um grande número de cálculos de distância entre instâncias de treinamento. Desta forma, técnicas de undersampling que atuem de forma aleatória se tornam interessantes para ambientes de stream.

Esse capítulo apresenta o método proposto nesse estudo, o qual é baseado em uma adaptação do algoritmo SIRUS (ZHANG et al., 2018) para ambientes online. O SIRUS é uma técnica fundamentada no uso de *ensembles* heterogêneos sobre ambientes *batch* que combina *inverse random undersampling* com meta-aprendizado. Desta forma, apesar de ser um trabalho relacionado, dedica-se a Seção 4.1 à descrição do método SIRUS original, permitindo assim a introdução da abordagem proposta, denominada *Online Stacking Inverse Random Under and Over Sampling* (ou *OnlineSIRUOS*), para cenários de fluxo de dados na Seção 4.2. Finalmente, a Seção 4.3 apresenta as considerações finais deste capítulo.

4.1 SIRUS

O algoritmo *SIRUS* (ZHANG et al., 2018) é uma técnica criada para executar sobre ambientes *batch* que conta com um mecanismo para tratar do desbalanceamento de dados. Além disso, essa estratégia possui uma camada de *meta-learning* que é usada na parte final do processo de predição de um exemplo a ser classificado.

O algoritmo utiliza de uma estratégia conhecida como sub-amostragem inversa aleatória (*inverse random undersampling*) (TAHIR; KITTLER; YAN, 2012) para realizar o treinamento do *ensemble* que forma o SIRUS. Essa técnica consiste em criar um número B de conjuntos de exemplos a partir da base de treinamento, similar a um processo de *bagging* (BREIMAN, 1996), isto é, utilizando *undersampling* randômico sem *replacement* sobre a classe majoritária de maneira que o *imbalance ratio* (IR) de cada conjunto seja o oposto do IR da base original. Cada conjunto é usado para treinar um classificador de cada tipo do *ensemble* e cada classificador irá ser treinado por um conjunto somente. É importante ressaltar que neste processo todas as instâncias da classe minoritária são associados a todos os subconjuntos de treinamentos selecionados para todos os membros do *ensemble*. Formalmente, assume-se que o IR da base de dados original é dado por $\frac{W}{Z}$, sendo que Z e W representam, respectivamente, as cardinalidades dos conjuntos de instâncias minoritárias e majoritárias. Desta forma, o IR de cada amostra de dados gerada para cada grupo de classificadores segue a Equação 4.1, sendo que W_i representa a

quantidade de instâncias da classe majoritária que são aleatoriamente selecionados para o i-ésimo grupo de membros do *ensemble*.

$$\frac{Z}{W_i} \approx \frac{W}{Z} \tag{4.1}$$

A estratégia de *meta-learning* utilizada no SIRUS é chamada de *stacking* (WOLPERT, 1992). Ela funciona de maneira que as previsões de um grupo de classificadores são usadas como dados de entrada (*features*) de outro classificador, que irá dar a classificação final a uma instância qualquer. No SIRUS, um conjunto de classificadores, que não necessariamente precisam ser homogêneos, é treinado com a base de treinamento. Posteriormente, os mesmos classificadores são usados para realizar predições sobre uma base de validação. De forma similar a abordagens seminais para aprendizagem em cenários desbalanceados (LIU; WU; ZHOU, 2009), os resultados das classificações e as reais classes de cada instância avaliada são então utilizados para criar uma nova base de dados de meta-instâncias. Com essa nova base, um meta-classificador é treinado, o qual é integrado no processo de classificação de novos exemplos.

Desta forma, o SIRUS conta com 3 pontos que diferenciam o seu funcionamento de outras técnicas: o uso do *inverse random undersampling* sem reposição de instâncias já selecionadas para a criação de conjuntos que vão ser usados para treinar o seu *ensemble*, o uso de R algoritmos diferentes para compor os membros do *ensemble* base e o uso de *stacking*. A visualização genérica de como os processos de treinamento e classificação ocorrem nesse algoritmo está nas Figuras 4.1 e 4.2, respectivamente.

No processo de treinamento, o *undersampling* empregado gera os B conjuntos de treinamento que irão ser usados para treinar B grupos de classificadores base. Cada grupo possui R tipos de elementos, os quais são baseados em algoritmos diferentes, ou seja, cada grupo nunca terá dois membros de mesmo tipo. Cada um dos classificadores do *ensemble* é então usado para classificar cada instância da base de validação. Os resultados dessas classificações junto com a real classe de cada exemplo processado são então transformados em meta-instâncias, as quais serão utilizadas para treinar o meta-classificador. Já no processo de predição, cada membro do *ensemble* classifica a nova instância e o resultado desse processo é então usado para criar uma meta-instância, feita dos votos dos membros do *ensemble* como suas *features*, a qual é passada para o meta-classificador, que gera a classificação final do SIRUS.

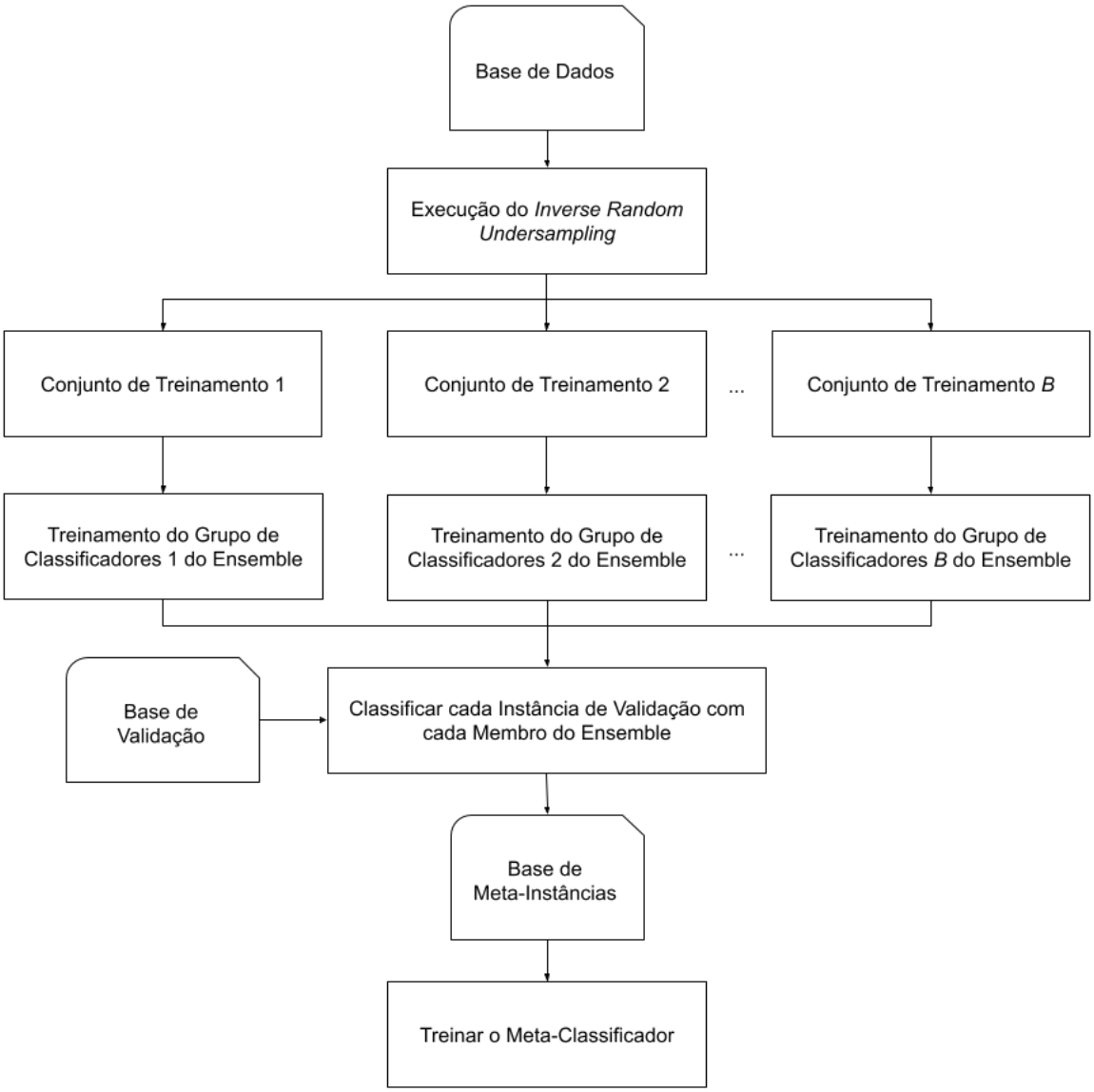

Figura 4.1: Fluxograma do processo de treinamento feito pelo SIRUS.

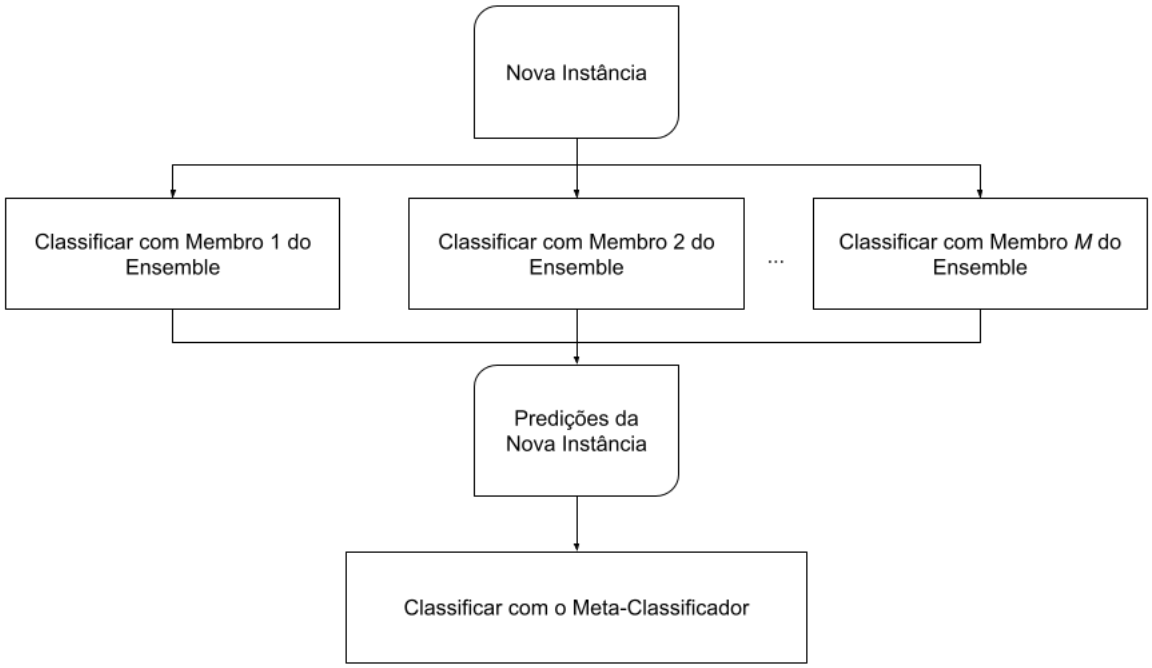

Figura 4.2: Fluxograma do processo de classificação feito pelo SIRUS.

4.2 O Método OnlineSIRUOS

O *OnlineSIRUOS* foi criado usando como inspiração o algoritmo SIRUS (Seção 4.1), contudo, projetado também para considerar com restrições e características de ambientes de fluxos contínuos de dados. Como nem todas as estratégias usadas em cenários tradicionais podem ser utilizadas em fluxos de dados, algumas adaptações tiveram que ser realizadas, especialmente sobre o processo de *inverse random undersampling*, que no caso do presente algoritmo utiliza também o *oversampling*.

Como o SIRUS é um algoritmo feito para executar em ambientes de *batch*, o mesmo tem a base de dados inteira à sua disposição quando for necessário. Isso é algo inviável em ambientes online, onde as restrições impostas pelo ambiente impossibilitam boa parte das estratégias que necessitam de acesso recorrente à informações já processadas. Desta forma, a criação de conjuntos de treinamento se torna inviável. Todavia, a ideia de utilizar valores da distribuição de Poisson para simular um *bagging* em fluxos de dados já é usada por outras técnicas como o OzaBag (Seção 3.5), o ARF (Seção 3.2), entre outros, com o intuito de gerar diversidade no *ensemble*. Portanto, o método proposto buscou utilizar uma estratégia de adaptação ao *inverse random under and over sampling*, usando o inverso da razão entre o número de instâncias já treinadas e que são da mesma classe que a instância analisada (I_y) sobre o total de instâncias já treinadas (I), combinado com o valor de Poisson($\lambda = \lambda'$). O valor de λ' pode ser parametrizado, mas tem como padrão o número 6, seguindo o exemplo do que já é feito no ARF. O valor resultante da adaptação é então usado como o número de vezes que a instância vai ser utilizada para treinar cada membro de um grupo do *ensemble*. Esse valor é conhecido como peso (w) e seu cálculo é apresentado na Equação 4.2.

$$w = \text{getWeight}(\lambda', y) = \left\lceil \left(1 - \frac{I_y}{I}\right) \times \text{Poisson}\left(\lambda = \lambda'\right) \right\rceil \tag{4.2}$$

Este cálculo é igual ao realizado no ARFRE 3.3, com a diferença que ocorre um arredondamento do resultado para valores acima. Também é importante ressaltar que essa equação deve ser recalculada cada vez que uma instância for ser utilizada para treinar um grupo de classificadores membros do *ensemble*, com a intenção de gerar diversidade. Consequentemente, ela acaba sendo recalculada menos vezes do que em relação ao ARFRE, pois o processo de *sampling* é realizado apenas para as instâncias da classe majoritária no algoritmo *batch*, enquanto o processo de amostragem proposto para o OnlineSIRUOS é aplicado para todas as instâncias. O objetivo é evitar um *overfitting* sobre a classe minoritária e um *undersampling* exagerado sobre a classe majoritária, já que o algoritmo tem que ser capaz de classificar ambas as classes corretamente, mesmo dando um foco

maior à classe minoritária. Além disso, como cada instância pode ser utilizada mais de uma vez para treinamento, não é possível garantir que não ocorra *resampling*. Outro ponto notável é que o *ensemble* utilizado é heterogêneo, assim como o algoritmo SIRUS original. Por isso, esse *ensemble* é formado por D grupos compostos por R modelos. Cada grupo é formado por uma instância de cada um dos R algoritmos de *data stream mining* selecionados para compor o *ensemble*. Ou seja, deve haver D modelos para cada um dos R algoritmos do *ensemble* e haverá um modelo de cada algoritmo em cada um dos D grupos. O racional por trás da criação de um *ensemble* heterogêneo é que classificadores base diferentes possuem vieses diferentes e, consequentemente, representações de conceitos diferentes que podem ser complementares para a classificação.

Com o treinamento do *ensemble* base do algoritmo seguindo a adaptação do *inverse random under and over sampling*, um processo básico de *stacking* para meta-aprendizagem é realizado em conjunto. Utilizando cada instância que chega para o algoritmo, após serem usadas para treinamento, cada membro do *ensemble* é aplicado para prever o rótulo da instância. O resultado de cada predição é combinado com a real classe de tal instância, formando um novo exemplo que vai ser usado para treinar um meta-classificador (Ω). O motivo disso é para que Ω aprenda com os erros e acertos do estado atual do *ensemble*. Se esse processo de aprendizado do meta-classificador fosse realizado antes do treinamento do *ensemble* com a nova instância, o mesmo estaria a par de como interagir com as informações de um *ensemble* defasado, fazendo com que ao final do treinamento, o meta-classificador tenha um funcionamento menos sinérgico com o *ensemble* e possivelmente menos correto. Por fim, esse mesmo meta-classificador é combinado ao processo de classificação assim como também é feito no SIRUS, onde o mesmo será responsável por dar a classificação final de uma nova instância no processo de predição de acordo com as informações que o *ensemble* lhe disponibilizar. O processo de classificação e treinamento do OnlineSIRUOS está descrito nos Algoritmos 1 e 2, respectivamente.

O processo de predição reportado no Algoritmo 1 inicia ao chegar um novo exemplo que precisa ser classificado. Os classificadores base do *ensemble* do OnlineSIRUOS são usados individualmente para predizer qual a possível classe da nova instância (linha 5-6). Essas classificações são então armazenadas e futuramente utilizadas como *features* para montar uma meta-instância (linha 6). Essa por sua vez é então classificada pelo meta-classificador e o seu resultado de classificação é a saída final, indicando a classe dada para a nova instância pelo OnlineSIRUOS (linhas 10-11).

Já o processo de treinamento apresentado no Algoritmo 2, que também recebe como entrada uma instância, desta vez rotulada; inicia atualizando a contagem de instâncias de cada classe que já foram usadas para treinamento, adicionando mais uma unidade

Algoritmo 1: Método de Predição do OnlineSIRUOS

Input : X: instância a ser classificada
Input : ϵ: *ensemble* base
Input : Ω: meta-classificador
Output: y: a classificação dada a X

```
[1] Y ← ∅;
[2] z ← 1;
[3] for i ← 1 to D do
[4] |   for j ← 1 to R do
[5] |   |   ε^{i,j} ← ε.get(i, j);
[6] |   |   Y_z ← ε^{i,j}.predict(X);
[7] |   |   z ← z + 1;
[8] |   end
[9] end
[10] y ← Ω.predict(Y);
[11] return y;
```

Algoritmo 2: Método de Treinamento do OnlineSIRUOS

Input : (X, y): instância rotulada para treinamento
Input : ϵ: *ensemble* base
Input : Ω: meta-classificador
Input : λ': valor de lambda

```
[1] Y ← ∅;
[2] z ← 1;
[3] updateClassesCounts(y);
[4] for i ← 1 to D do
[5] |   w ← getWeight(λ', y);
[6] |   for j ← 1 to R do
[7] |   |   ε^{i,j} ← ε.get(i, j);
[8] |   |   ε^{i,j}.train(X, y, w);
[9] |   |   Y_z ← ε^{i,j}.predict(X);
[10] |  |   z ← z + 1;
[11] |  end
[12] end
[13] Ω.train(Y, y);
```

à classe do novo exemplo de treinamento (linha 3). Com isso, a nova instância recebe um peso de acordo com a Equação 4.2 (linha 5), antes de ser usada para treinar todos os classificadores de um mesmo grupo no mesmo número de vezes que o peso recebido (linhas 7-8). Ao fim do treinamento, o classificador é usado para classificar a mesma instância usada para treiná-lo e o resultado dessa predição é guardado (linha 9). O processo se repete para cada um dos D grupos de classificadores do *ensemble*, conforme descrito no laço de repetição das linhas 4-12. Ao finalizar essa etapa, com os resultados das predições guardadas, uma meta-instância é criada de maneira que cada predição é usada como uma

feature e a real classificação do exemplo de treino é usada como a classe da meta-instância. Por fim, essa meta-instância é usada para treinar o meta-classificador Ω do OnlineSIRUOS (linha 13).

Há também uma opção de não uso de meta-classificador para o OnlineSIRUOS. Nesta configuração, Ao invés de gerar uma meta-instância através dos votos do *ensemble* e utilizá-la para treinar um meta-classificador baseado em algum algoritmo específico, os votos são utilizados em um processo de voto majoritário simples (SM) para as etapas de predição, que consiste basicamente em determinar que a classe com mais votos do *ensemble* será definida como a classificação dada para a instância sendo avaliada pelo OnlineSIRUOS. Nas etapas de treinamento, somente o *ensemble* realizará uma atualização, visto que não há um modelo para ser treinado no lugar do meta-classificador. Esta opção foi definida para que fosse possível verificar se existe algum benefício em utilizar o processo de *stacking* neste algoritmo.

4.3 Considerações Finais

Este capítulo apresentou o método proposto para o desenvolvimento deste estudo. Este método é inspirado em uma técnica proposta para ambientes batch chamada SIRUS que combina aprendizagem usando *ensembles* heterogêneos, *inverse random undersampling* e meta-aprendizagem. Ressalta-se que o processo de *inverse random undersampling*, por se tratar de uma seleção aleatória, apresenta custo computacional baixo. Por outro lado, o processo de meta-aprendizagem usando *stacking* deve ser ponderado pois fatalmente induzirá custo computacional elevado quando comparado ao treinamento de um *ensemble* que realize voto majoritário como processo de combinação. Desta forma, uma análise sobre o desempenho deste algoritmo, principalmente no que tange métricas de avaliação de classificação serão discutidas no próximo capítulo. No Capítulo 5 serão apresentados e discutidos os resultados experimentais obtidos sobre cada um dos algoritmos elencados no Capítulo 3 e sobre a implementação da técnica proposta neste estudo.

Capítulo 5

Experimentação

Esse capítulo apresenta inicialmente o protocolo experimental usado para testar e avaliar os algoritmos do estado da arte elencados no Capítulo 3 contra o algoritmo proposto no Capítulo 4. Além disso, os resultados obtidos são discutidos e uma análise é feita para apontar informações importantes, a respeito das técnicas e sobre os dados de avaliação obtidos ao final do processo de teste.

5.1 Protocolo Experimental

Neste capítulo é apresentada a metodologia empregada para obter dados de desempenho das técnicas analisadas e a discussão dos resultados obtidos. Todos os algoritmos utilizados, incluindo geradores de dados sintéticos, classificadores baseados em fluxos de dados, métodos de avaliação de classificadores e cálculos de métricas de desempenho foram obtidos a partir do *Massive Online Analysis* (MOA) (BIFET et al., 2010). O MOA é um *framework* escrito em Java, que pode ser usado isoladamente como um programa de *benchmark*, e dispõe de diversas técnicas utilizadas na mineração de fluxos de dados já implementadas, validadas e prontas para uso. Todo e qualquer parâmetro de qualquer algoritmo usado que não esteja definido explicitamente em alguma das tabelas de parâmetros ao longo dessa seção está sendo usado com seu valor padrão, definido no próprio *framework* MOA.

No total foram executados mais de 450 experimentos. Os resultados foram organizados e as métricas de desempenho de cada algoritmo foram processadas através da combinação dos testes de **Friedman** e **Nemenyi** de acordo com o protocolo descrito em (DEMŠAR, 2006), buscando encontrar diferenças significativas de desempenho entre as técnicas analisadas para chegar a uma conclusão mais precisa. Além disso, os dados foram apresentados através de outros gráficos e testes para uma melhor visualização dos

resultados de cada um dos algoritmos, como através dos *Radar Charts* e dos *Sign Tests* (JAPKOWICZ; SHAH, 2014), trazendo assim novas perspectivas dos mesmos resultados.

5.1.1 Prequential

O método usado como esquema de validação dos classificadores testados foi o *Prequential* (GAMA; SEBASTIAO; RODRIGUES, 2009). Esse método é o mais utilizado para avaliar modelos em ambientes de fluxos contínuos de dados. O processo de Prequential consiste em a cada nova instância apresentada ao modelo para treino, ela primeiro é testada, comparando a classificação dada pelo próprio modelo com a real classe dessa instância e com isso atualizando as métricas de desempenho relativas ao próprio classificador, e posteriormente usada para treinar o modelo. Isso configura um formato de teste-então-treino, ou *test-then-train*. Conforme o artigo original (GAMA; SEBASTIAO; RODRIGUES, 2009), o método Prequential é uma opção válida para avaliar classificadores em fluxos de dados pois há garantias teóricas de que a análise Prequential assintoticamente converge para um *holdout*, uma vez que os dados são usados para teste antes do treinamento.

O Prequential possui dois formatos de execução, isto é, o formato "básico", onde os resultados de todas as instâncias são acumulados desde o início do processamento, e o formato "janelado", onde define-se o intervalo (quantidade) de instâncias que são analisadas a partir de uma janela deslizante para que as parciais da avaliação dos classificadores sejam calculadas e apresentadas. O formato usado nas etapas de experimentação desse trabalho foi o janelado. Desta forma, assumindo um experimento com tamanho U e uma janela de avaliação de tamanho δ, uma avaliação *Prequential* atuará no formato teste-então-treino, descrito no parágrafo anterior, e o resultado da avaliação obtido a cada δ instâncias é reportado na interface e usado para comparação dos modelos. Nos testes apresentados a seguir, os experimentos sintéticos usaram $U = 1.000.000$ e $\delta = 10.000$. Para os experimentos com bases reais, todo o padrão utilizado para os experimentos sintéticos foi seguido, com a exceção dos valores de U e δ. Cada experimento realizado contou com 10 checagens de métricas durante as suas execuções. Esse valor foi definido pois entre as bases citadas na Seção 5.1.3 há diversas variedades de tamanhos. Devido a essa diferença na quantidade de instâncias em cada base, foi decidido que uma quantidade pequena mas razoável de checagens que poderiam ser feitas de forma que entre elas houvesse o processamento de no mínimo 1000 exemplos, para que assim pudesse ser observada as diferenças com mais garantias de que haveriam mudanças de desempenho, ou seja, $\frac{U}{\delta} \geq 1000$.

5.1.2 Geradores de Dados

Neste estudo, 4 geradores de dados encontrados nativamente no MOA foram utilizados e todos eles são definidos nas subseções a seguir. Eles foram escolhidos por serem usados comumente na literatura e por serem geradores de dados sintéticos, o que implica em uma fácil utilização, flexibilidade e replicação dos experimentos propostos. Todos estes geradores foram modificados para que utilizem diferentes configurações com diferentes níveis de desbalanceamento (IR). Aos experimentos em que não há uma mudança no IR ao longo de sua execução se deu o nome de **desbalanceamento estático**. Os parâmetros utilizados para configurar cada um dos cenários de desbalanceamento estático com cada um dos geradores estão descritos na Tabela 5.1.

Tabela 5.1: Parâmetros usados para cada gerador de dados sintéticos em cada experimento com desbalanceamento estático.

Parâmetro	**Valor**
U	1.000.000
IR	50%–50%
	70%–30%
	80%–20%
	90%–10%
	95%–5%
	99%–1%
	99,5%–0,5%

Cada gerador de dados foi configurado para ter sempre o mesmo número U de instâncias em cada um dos experimentos executados. Além disso, cada execução dos mesmos contou com um IR diferente. Ou seja, o primeiro experimento criado com algum dos geradores contaria com 1.000.000 de instâncias e IR = 50%–50%, o segundo contaria com a mesma quantidade de instâncias mas com um IR = 70%–30% e assim por diante. Cada um desses experimentos foi utilizado para testar todos os algoritmos presentes nas análises a seguir.

5.1.2.1 *Asset Negotiation Generator*

O *Asset Negotiation Generator* (ANG) (ENEMBRECK et al., 2007) simula modelos gerando propostas de negociação. As classes alvo podem ser duas: interessante ou não interessante. Os atributos presentes nesse gerador estão listados na Tabela 5.2 e o mesmo conta com 5 tipos de simulações para gerar mudanças de conceito (*concept drifts*) em meio a sua execução.

5.1.2.2 *Agrawal Generator*

O AGRAWAL (AG) (AGRAWAL; IMIELINSKI; SWAMI, 1993) divide instâncias em dois grupos (classes). Existem 10 funções que determinam a qual classe um conjunto de atributos gerados aleatoriamente irá pertencer e *concept drifts* podem ser criados ao alternar entre essas funções. Esse gerador também conta com um fator de perturbação que é adicionado a todos os atributos contínuos, com o intuito de criar ruído sobre os dados visando manter uma aleatoriedade na criação dos dados. Os atributos presentes nesse gerador estão listados na Tabela 5.3.

5.1.2.3 *Random Tree Generator*

O *Random Tree Generator* (RTG) (DOMINGOS; HULTEN, 2000) conta com uma funcionalidade que permite inserir o número de atributos contínuos, categóricos e de classe. Seu mecanismo interno funciona da seguinte maneira: uma árvore de decisão é criada baseada em divisões aleatórias nos atributos, com nós folha sendo definidos em alturas pré-estabelecidas pelo usuário. As mudanças de conceito são simuladas recriando as árvores com novas *seeds* aleatórias. A partir dessas árvores, instâncias são geradas criando atributos aleatórios para cada atributo necessário até chegar em um nó folha, definindo assim a classe desse exemplo.

5.1.2.4 *Streaming Ensemble Algorithm*

O *Streaming Ensemble Algorithm* (SEAG) (STREET; KIM, 2001) é um gerador de dados com perfil relativamente simples. Para cada instância é criado um vetor de valores de atributo (x^1, x^2, x^3), onde somente dois dos valores são valores reais de feature, enquanto que o último serve para a criação de ruído. Feito isso, a classe dessa instância é definida segundo um limite linear, definido por $x^1 + x^2 > \zeta$, sendo ζ de valor passível de mudança. Ao mudar esse valor ao longo da execução do gerador, uma mudança de

Tabela 5.2: Atributos do gerador de dados ANG adaptado de (ENEMBRECK et al., 2007).

Atributo	**Possíveis Valores**
Pagamento	[0, 30, 60, 90, 120, 150, 180, 210, 240]
Quantidade	[very low, low, normal, high, very high, quite high, enormous, non ensured]
Atraso de Entrega	[very low, low, normal, high, very high]
Cor	[black, blue, cyan, brown, red, green, yellow, magenta]
Preço	[very low, low, normal, high, very high, quite high, enormous, non salable]

Tabela 5.3: Atributos do gerador de dados AG adaptado de (AGRAWAL; IMIELINSKI; SWAMI, 1993).

Atributo	**Possíveis Valores**
Elevel	[0, 1, 2, 3, 4]
Carro	[1, 2, ..., 20]
Código Postal	$[C_0, C_1, \ldots, C_8]$
Salário	Aleatório entre 20.000 e 150.000
Comissão	0 se Salário $\geq$ 75.000 senão Aleatório entre 10.000 e 75.000
Ano	Aleatório entre 20 e 80
Empréstimo	Aleatório entre 0 e 500.000
Hvalue	Aleatório entre $\frac{i}{2} \times 100.000$ e $\frac{3i}{2} \times 100.000$ para i = Índice do Código Postal
Hyears	Aleatório entre 1 e 30

conceito ocorre.

5.1.3 Bases Reais

Junto aos dados sintéticos utilizados, bases de dados reais foram utilizadas para avaliar os classificadores testados. Todas as bases selecionadas possuem desbalanceamento e por isso foram consideradas. Cada uma delas possui as suas próprias características e por serem diferentes do que as bases sintéticas apresentam, podem trazer informações relevantes a respeito de como os algoritmos analisados se comportam em ambientes mais próximos ainda de um cenário real. Uma informação importante a respeito dessas bases é que é possível encontrar algumas delas disponíveis no repositório UCI (DUA; GRAFF, 2017). A descrição de cada uma das bases reais usadas nos experimentos se dará nas próximas subseções.

5.1.3.1 *Bank Marketing*

A base conhecida como *Bank Marketing* (MORO; LAUREANO; CORTEZ, 2011; MORO; CORTEZ; RITA, 2014) é formada por dados de campanhas de marketing direto de uma instituição bancária portuguesa baseadas em ligações telefônicas. Mais de uma ligação ao mesmo cliente foi necessária em vários casos para avaliar se o depósito bancário a prazo seria assinado ou não. Classificar se um cliente iria ou não assinar um depósito a prazo é o objetivo dessa base.

Essa base conta com 45.211 instâncias e 16 atributos, dentre os quais 7 são numéricos, 3 são binários e 6 são categóricos. O atributo de classe também é binário e a classe majoritária ocupa 39.922 exemplos, atestando um claro cenário de alto desbalanceamento.

5.1.3.2 *Forest Covertype*

Os dados da base *Forest Covertype* são comumente usados para tarefas de classificação (BIFET et al., 2013a; BIFET et al., 2013b; KOSINA; GAMA, 2012) e já foram utilizados em diversos trabalhos de *data stream mining*. A ideia é que um classificador tente predizer o tipo de vegetação florestal baseando-se em variáveis cartográficas dentre um conjunto de células de 900m^2 obtidas da US Forest Service Region 2 Resource Information System (RIS) (KOSINA; GAMA, 2012).

Essa base conta com 581.012 instâncias e 54 atributos, dentre os quais 10 são numéricos e 44 são binários. O atributo de classe não é binário e duas classes ocupam a maior parte dos exemplos existentes (211.840 e 283.301 instâncias).

5.1.3.3 CSDS

As bases CSDS são 3 conjuntos de dados reais relativos a cenários de empréstimos a clientes que devem ser analisados de forma que um determinado modelo deve prever quais devedores vão pagar as suas dívidas ou se ficarão inadimplentes (BARDDAL et al., 2020).

A CSDS-1 conta com informações de 315.000 pedidos de empréstimos que foram analisados e concedidos. A diferença dessa base em relação às outras é que os modelos criados com ela devem identificar se cada cliente pagou ou não as suas dívidas em um período de 6 meses. Além disso, os modelos preditivos criados devem ter taxas de predição competitivas quando comparado com as duas principais agências de pontuação de crédito do Brasil. Em termos das propriedades dessa base, ela possui 315.539 instâncias e 176 atributos, dentre os quais todos são numéricos. O atributo de classe é binário e a classe majoritária ocupa cerca de 274.519 exemplos, o que corresponde a mais de 85% das instâncias, demonstrando um cenário desbalanceado.

A CSDS-2 difere da primeira pois conta com cerca de 50.000 empréstimos dados por uma empresa de financiamento de carros. Além disso, os modelos criados devem identificar se cada cliente pagou as suas dívidas ou não, mas em um períodos de 2 meses. O objetivo final dessa base é o de criar modelos que avaliem se os atributos disponíveis nessa empresa resultariam em taxas preditivas interessantes. Em relação às propriedades da base, ela possui 50.401 instâncias e 37 atributo, todos numéricos. A classe do problema é binária e os elementos da classe majoritária contam por cerca de 49.393 exemplos, fazendo dessa base um cenário extremamente desbalanceado.

Já a CSDS-3 conta com dados a respeito de 97.000 empréstimos que foram conce-

didos por um grande banco no Brasil. Assim como nas outras bases, a ideia é identificar se o cliente ficou inadimplente ou não em um período de 3 meses. O objetivo é o de criar um modelo com taxas preditivas competitivas em relação ao modelo já usado pelo banco em questão. Essa base conta com 97.226 instâncias e 152 atributos, todos também numéricos. A classe possui valores binários e os elementos da classe majoritária correspondem a 71.947 exemplos, caracterizando em um cenário menos desbalanceado que as outras bases mas ainda sim interessante para o estudo do desbalanceamento.

5.1.3.4 *Give me Loan*

Essa base de dados constitui um desafio de classificação de mais de 10 anos que foi colocado no site Kaggle, que é uma plataforma de desafios de machine learning aberta para os usuários disponibilizarem as suas soluções e concorrerem a prêmios, onde o objetivo é melhorar o estado da arte de pontuação de créditos predizendo a probabilidade de algum cliente sofrer com problemas financeiros nos próximos 2 anos. Com isso, os modelos criados podem ser usados para ajudar os recebedores dos auxílios a tomarem as melhores decisões financeiras.

A base conta com 150.000 instâncias e 10 atributos, todos numéricos. A classe comporta valores binários e os elementos da classe majoritária ocupam cerca de 139.974 instâncias. Com isso é possível notar que essa base representa um cenário altamente desbalanceado.

5.1.3.5 IntelLabSensors

Os dados dessa base foram coletados de 54 sensores implantados no laboratório Intel Berkeley Research entre 28 de fevereiro e 5 de abril de 2004 (BODIK et al., 2004). Todos eles captaram dados de topologia com marcação de data e hora, juntamente com valores de umidade, temperatura, luz e tensão a cada 31 segundos.

A mesma base foi dividida em 5 bases diferentes, como o que foi feito no artigo do algoritmo KUE (Seção 3.1). Todas elas contam com os mesmos atributos e exemplos. A única diferença está relacionada aos possíveis valores de atributo de classe que essas bases possuem. Essas bases possuem 2.313.153 instâncias e 5 atributos, todos numéricos. Para a base principal, o atributo de classe não é binário e possui 58 possíveis resultados, todos numéricos de 1 a 58. Para as outras bases, o atributo de classe é binário, unindo diferentes classes da base original em 2 grupos diferentes, que por sua vez servem como seus atributos de classe. Os grupos das bases, com exceção da base original, estão descritos na Tabela

5.4.

5.1.3.6 *KDD99 Binary*

O *KDD99* corresponde a um problema de detecção automática de ciberataques e que possui a característica de ser uma base evolucionária (AGGARWAL et al., 2003; CAO et al., 2006) porque se trata de um problema que por definição é *online*, onde as instâncias são apresentadas em série, em que evoluções de conceito acontecem devido ao aparecimento e desaparecimento de tipos de ataques (AGGARWAL et al., 2003). Diferentemente da base de dados original em que os rótulos das instâncias eram divididos em classes (tipos de ataques), neste experimento, foi utilizada uma versão binária da base de dados onde instâncias foram divididas apenas entre ataques e não-ataques. Essa base conta com 4.898.431 instâncias e 41 atributos, dentre os quais 34 são numéricos, 4 são binários e 3 são categóricos. O atributo de classe é binário e a classe majoritária constitui 3.925.650 exemplos.

5.1.4 Configuração dos Classificadores

A parametrização dos classificadores seguiu a configuração padrão proposta nos artigos originais, com a exceção de alguns parâmetros específicos. Apesar da adoção de uma parametrização padrão seja subótima, ressalta-se que a realização do *tuning* de parâmetro por experimento se tornou inviável pelo tempo disposto para este trabalho. Ainda, a realização de *tuning* em cenários de fluxos contínuos de dados não é trivial uma vez que não há garantia que uma parametrização ótima para uma parte do fluxo de dados seja também ótima para outras partes por conta de mudanças de conceito. Sendo assim, é importante ressaltar que o *tuning* de parâmetros de classificadores para *data streams* é um tópico de pesquisa por si só, uma vez que a configuração ótima de um classificador pode mudar de acordo com a distribuição dos dados e *concept drifts* (VELOSO; GAMA; MALHEIRO, 2018; GOMES et al., 2019). As mudanças realizadas incluíram os tipos dos classificadores base nos ensembles usados (ϵ) e o número de elementos do ensemble

Tabela 5.4: Grupos de valores das classes das bases IntelLabSensors adicionais.

Nome da Base	**Classe 1**	**Classe 2**
IntelLabSensors-1vsAll	[1]	[2, 3, ..., 58]
IntelLabSensors-1-2-3vsAll	[1, 2, 3]	[4, 5, ..., 58]
IntelLabSensors-1-2-3-4-5vsAll	[1, 2, 3, 4, 5]	[6, 7, ..., 58]
IntelLabSensors-1-2-3-4-5-6-7-8-9vsAll	[1, 2, 3, 4, 5, 6, 7, 8, 9]	[10, 11, ..., 58]

Tabela 5.5: Parâmetros usados para cada classificador em todos os experimentos.

Algoritmo	**Parâmetro**	**Valor**
ARF (3.2)	M	100
ARFRE (3.3)	M	100
KUE (3.1)	M	100
LeveragingBag (3.8)	M	100
CSARF (3.4)	M	100
OnlineAdaC2 (3.10)	M	100
	ϵ	HoeffdingTrees
OnlineUOB (3.11)	M	100
	ϵ	HoeffdingTrees

$M = 100$ sempre que possível.

Todos os classificadores apresentados nos Capítulos 3 e 4 foram utilizados nos experimentos com os geradores de dados escolhidos (Seção 5.1.2) sobre o esquema de avaliação Prequential (Seção 5.1.1). Além dos classificadores baseados em *ensembles*, os algoritmos Naive Bayes e Hoeffding Tree (DOMINGOS; HULTEN, 2000) foram adicionados à esta experimentação inicial. O objetivo desta adição neste primeiro momento foi verificar se os métodos mais complexos apresentavam performance próxima ou até superior a estes classificadores base em relação às métricas utilizadas. O resultado foi positivo e por isso foi possível prosseguir para as próximas etapas de análise das próximas seções.

Para a análise das seções a seguir, as mesmas configurações previamente mencionadas foram utilizadas, com a diferença de que somente alguns dos algoritmos relacionados à *data streams* foram considerados, visto que um dos objetivos deste trabalho é verificar o quão eficiente o OnlineSIRUOS é quando comparado a outros algoritmos similares em relação a cenários de fluxos de dados contínuos desbalanceados. Usar algoritmos não preparados minimamente para tais ambientes poderia ser uma comparação injusta. A lista com os algoritmos usados e seus parâmetros que foram modificados em relação aos seus valores padrão estão listados na Tabela 5.5.

Um ponto importante a ser mencionado é o fato de que nem todos os algoritmos utilizados estão relacionados à *data streams* desbalanceadas. Isso aconteceu porque os algoritmos ARF e LeveragingBag obtiveram resultados interessantes em relação aos outros algoritmos.

Como o algoritmo OnlineSIRUOS proposto nesse estudo pode ser parametrizado de diferentes maneiras, as variações testadas são compostas de mudanças no *ensemble*, no meta-classificador e no valor de λ'. As escolhas que foram utilizadas estão na Tabela 5.6. É importante lembrar que todas as versões dessa tabela utilizaram por padrão um *ensemble* com $D = 100$ grupos, o que indica que $M = D \times R$, sendo M o número de

classificadores membros do *ensemble*, D o número de grupos de modelos criados no *ensemble* e R o número de algoritmos diferentes sendo usados no *ensemble*. Entretanto, houve algumas variações que utilizaram valores de D diferentes. Além disso, como informado na Seção 4.2, SM refere-se à alternativa de voto majoritário simples no lugar de um meta-classificador no OnlineSIRUOS. Já NB e HT referem-se respectivamente aos algoritmos de Naive Bayes e Hoeffding Tree (DOMINGOS; HULTEN, 2000).

Por facilidade, conveniência e para evitar a elaboração de análises demasiadamente extensas - e que poderiam levar a resultados confusos, foram selecionadas 2 versões dentre as abordadas na Tabela 5.6.

A escolha foi feita de maneira que uma das versões foi selecionada devido ao melhor desempenho de F1-score entre as variações de OnlineSIRUOS usadas e a outra foi escolhida devido ao melhor equilíbrio entre resultados de F1-score e custo computacional entre essas mesmas variações. Esse processo de seleção é descrito em mais detalhes na Seção 5.2 e as versões selecionadas são usadas como base para os comparativos feitos nas seções seguintes.

Por fim, há alguns pontos relevantes para reprodutibilidade que devem ser mencionados aqui a respeito dos classificadores utilizados. O primeiro é que o algoritmo CSARF originalmente utiliza a métrica MCC no lugar da acurácia, a qual era a métrica utilizada pelo ARF para atribuir pesos aos membros do ensemble. Contudo, após contato direto e sugestão dos autores do algoritmo em questão, o CSARF foi alterado para utilizar a métrica F1-score ao invés do MCC, apesar desta configuração não ser a reportada na publicação original.

5.1.5 Métricas de Avaliação

Para avaliar todos os algoritmos, foi utilizado um subconjunto de métricas disponível no MOA e também apresentados anteriormente na Seção 2.3.1. As métricas escolhidas foram: F1-score, Tempo de Processamento e Custo do Modelo. Essas métricas mostram diferentes perspectivas sobre o desempenho de cada algoritmo. Para uma análise extensiva e completa de um algoritmo, o ideal seria utilizar diferentes métricas para a avaliação do desempenho de classificação, devido às diferentes informações que cada métrica pode apresentar. Entretanto, utilizar todas essas métricas poderia resultar em uma análise complexa e demasiadamente extensa. Portanto, foi feita a escolha de utilizar somente o F1-score como métrica principal de classificação, pois seu resultado é dependente e sumariza as métricas de precisão (*Precision*) e revocação (*Recall*), trazendo uma boa relação entre ambas, fazendo do F1-score uma boa fonte de informação a respeito dos algoritmos

Tabela 5.6: Parâmetros usados para cada variação do OnlineSIRUOS testada.

Variação	Parâmetro	Valor
OnlineSIRUOS	ϵ	HT
	Ω	SM
	λ'	6
OnlineSIRUOS1	ϵ	HT
	Ω	SM
	λ'	1
OnlineSIRUOS2	ϵ	HT
	Ω	HT
	λ'	1
OnlineSIRUOS3	ϵ	HT
	Ω	NB
	λ'	1
OnlineSIRUOS4	ϵ	HT
	Ω	SM
	λ'	10
OnlineSIRUOS5	ϵ	HT
	Ω	HT
	λ'	10
OnlineSIRUOS6	ϵ	HT
	Ω	NB
	λ'	10
OnlineSIRUOS7	ϵ	HT
	Ω	HT
	λ'	6
OnlineSIRUOS8	ϵ	HT
	Ω	NB
	λ'	6
OnlineSIRUOS9	ϵ	HT, NB
	Ω	SM
	λ'	6
OnlineSIRUOS10	ϵ	HT, NB
	Ω	SM
	λ'	1
OnlineSIRUOS11	ϵ	HT, NB
	Ω	HT
	λ'	1
OnlineSIRUOS12	ϵ	HT, NB
	Ω	NB
	λ'	1
OnlineSIRUOS13	ϵ	HT, NB
	Ω	SM
	λ'	10
OnlineSIRUOS14	ϵ	HT, NB
	Ω	HT
	λ'	10
OnlineSIRUOS15	ϵ	HT, NB
	Ω	NB
	λ'	10
OnlineSIRUOS16	ϵ	HT, NB
	Ω	HT
	λ'	6
OnlineSIRUOS17	ϵ	HT, NB
	Ω	NB
	λ'	6
OnlineSIRUOS18	ϵ	HT, NB
	Ω	SM
	λ'	6
	D	50
OnlineSIRUOS19	ϵ	HT, NB
	Ω	SM
	λ'	1
	D	50
OnlineSIRUOS20	ϵ	HT, NB
	Ω	HT
	λ'	1
	D	50
OnlineSIRUOS21	ϵ	HT, NB
	Ω	NB
	λ'	1
	D	50
OnlineSIRUOS22	ϵ	HT, NB
	Ω	SM
	λ'	10
	D	50
OnlineSIRUOS23	ϵ	HT, NB
	Ω	HT
	λ'	10
	D	50
OnlineSIRUOS24	ϵ	HT, NB
	Ω	NB
	λ'	10
	D	50
OnlineSIRUOS25	ϵ	HT, NB
	Ω	HT
	λ'	6
	D	50
OnlineSIRUOS26	ϵ	HT, NB
	Ω	NB
	λ'	6
	D	50
OnlineSIRUOS27	ϵ	NB
	Ω	SM
	λ'	6
OnlineSIRUOS28	ϵ	NB
	Ω	SM
	λ'	1
OnlineSIRUOS29	ϵ	NB
	Ω	HT
	λ'	1
OnlineSIRUOS30	ϵ	NB
	Ω	NB
	λ'	1
OnlineSIRUOS31	ϵ	NB
	Ω	SM
	λ'	10
OnlineSIRUOS32	ϵ	NB
	Ω	HT
	λ'	10
OnlineSIRUOS33	ϵ	NB
	Ω	NB
	λ'	10
OnlineSIRUOS34	ϵ	NB
	Ω	HT
	λ'	6
OnlineSIRUOS35	ϵ	NB
	Ω	NB
	λ'	6

avaliados. Apesar da análise ser enviesada para a métrica de F1-score, demais métricas apresentadas na Seção 2.3.1 também são apresentadas e discutidas.

Ao realizar os experimentos sintéticos, cada uma das métricas foi captada a cada 10.000 instâncias processadas, seguindo os parâmetros configurados para o Prequential (Seção 5.1.1) e os geradores de dados (Seção 5.1.2) utilizados. No caso dos experimentos com bases reais, essas mesmas métricas foram calculadas a cada 10% da base de dados. Isso se deve ao fato de cada uma delas possuírem tamanhos diferentes e na maior parte serem muito menores do que os experimentos sintéticos.

5.2 Seleção de OnlineSIRUOS

A primeira etapa para a avaliação do desempenho do OnlineSIRUOS foi feita de maneira a selecionar uma versão dentre às apontadas na Tabela 5.6 de modo que uma configuração padrão do novo método pudesse ser usada para os comparativos com os demais algoritmos da literatura, sem a necessidade de gerar uma análise muito extensa e potencialmente confusa sobre os dados e evitando criar um comparativo enviesado ao método proposto. A versão foi escolhida de acordo com a melhor relação entre o melhor resultado de F1-score possível dentro de um intervalo de confiança estatístico enquanto mantêm os menores valores de tempo de processamento e custo do modelo. Essa versão foi denominada OnlineSIRUOSΔ.

A análise para a escolha dessa versão foi feita através dos gráficos de Nemenyi (Figuras 5.1, 5.2, 5.3, 5.4 e 5.5, 5.6), comparando os resultados de cada variação de OnlineSIRUOS utilizada nos cenários com dados reais e sintéticos separadamente. Considerando esses gráficos, foram criadas as Tabelas 5.7 e 5.8, onde são apresentados valores relacionados às métricas de F1-score, Tempo de Processamento e Custo Computacional para cada cenário analisado. Esses valores indicam a posição que determinada variação do OnlineSIRUOS obteve dentro dos gráficos. Por exemplo, nos gráficos sobre os dados reais, a variação OnlineSIRUOS4 ficou em 2º lugar em relação F1 Score, 18º lugar em relação ao Tempo de Processamento e 22º lugar em relação ao Custo do Modelo. Um fato importante a se lembrar é que somente as variações que estão dentro da linha de diferença crítica mais alta de F1-score, ou seja, de maior desempenho, que obtiveram valores de posicionamento. As variações que obtiveram o caractere "-" como posição estão fora dessa linha.

Ao observar as tabelas e considerando somente a métrica de F1-score, é possível reconhecer que a variação OnlineSIRUOS6 consegue a 1ª posição nos experimentos com bases sintéticas e a 3ª posição nos experimentos com bases reais, fazendo dessa variação

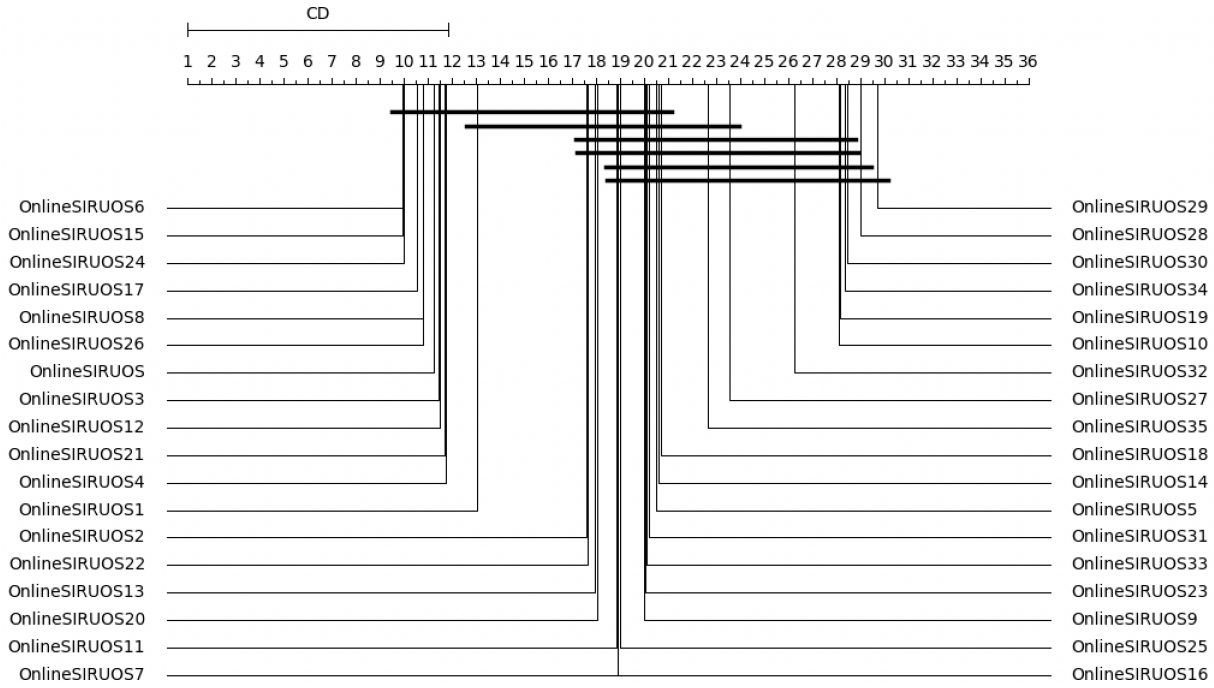

Figura 5.1: Gráfico de Nemenyi sobre variações do OnlineSIRUOS com resultados de F1-Score sobre cenários sintéticos desbalanceados.

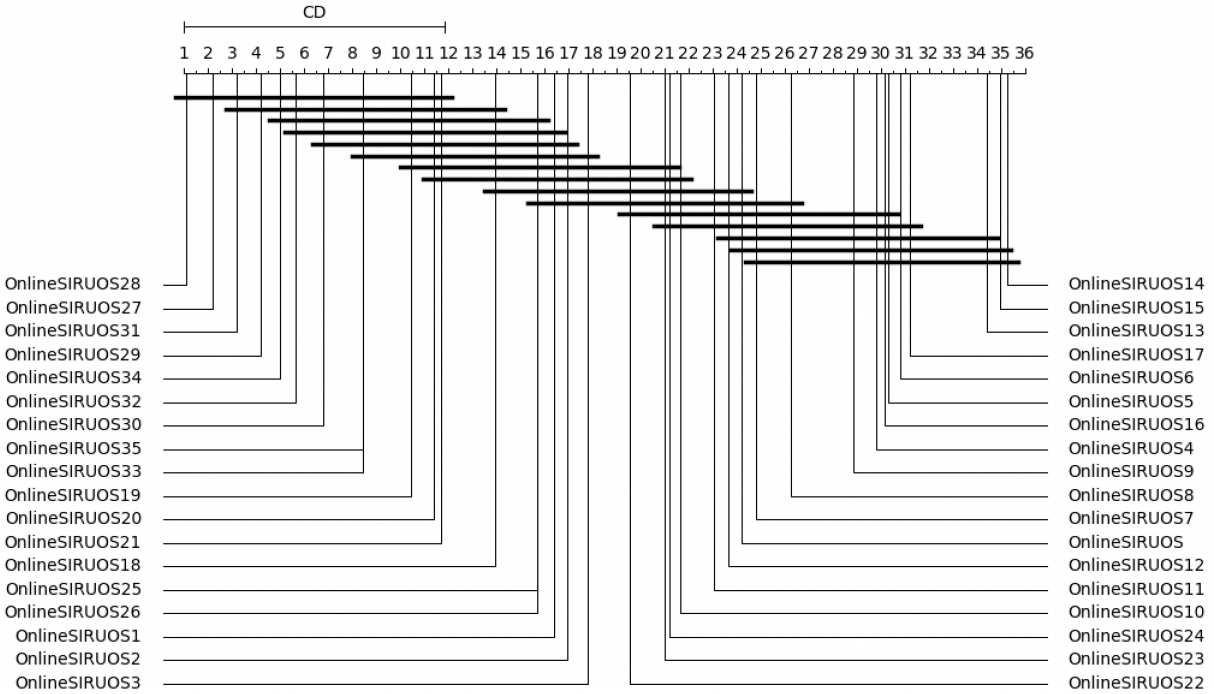

Figura 5.2: Gráfico de Nemenyi sobre variações do OnlineSIRUOS com resultados de Tempo de Processamento sobre cenários sintéticos desbalanceados.

a que melhor obteve resultados de F1-score. Entretanto, não necessariamente essa variação é a mais equilibrada em relação a precisão de classificação e custo computacional. Por isso, considerando a relação entre F1-score, Tempo de Processamento e Custo do Modelo, a escolha se fez da seguinte forma: primeiramente só variações dentro da linha

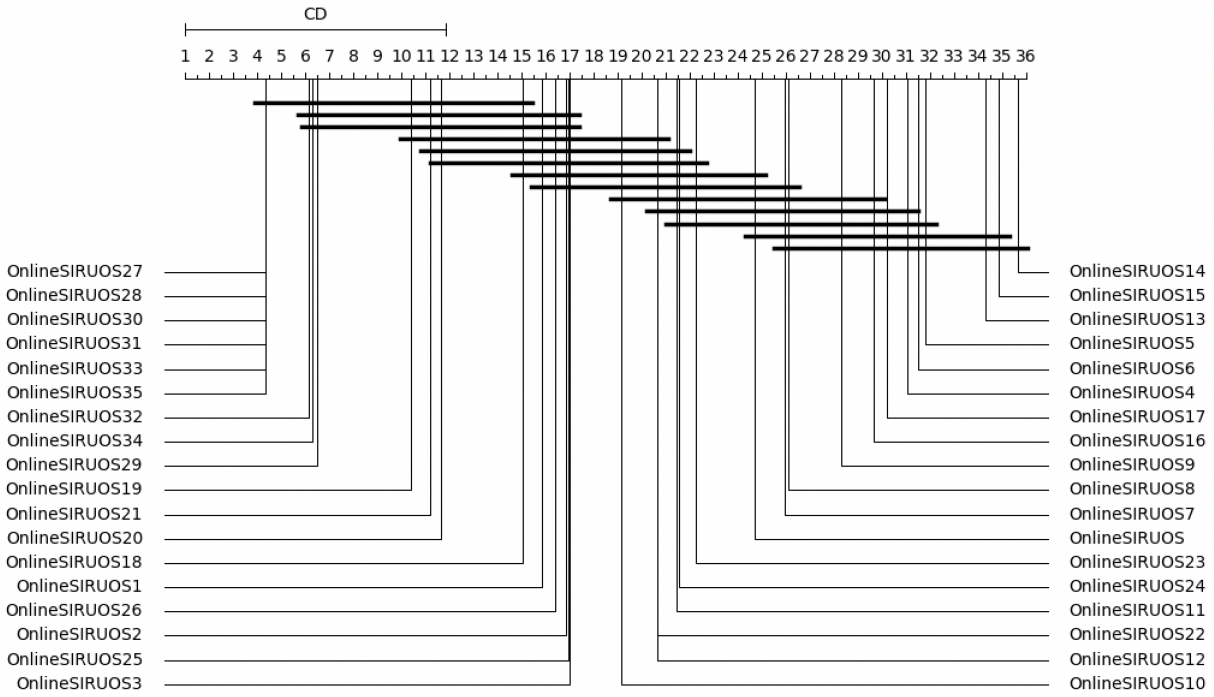

Figura 5.3: Gráfico de Nemenyi sobre variações do OnlineSIRUOS com resultados de Custo de Modelo sobre cenários sintéticos desbalanceados.

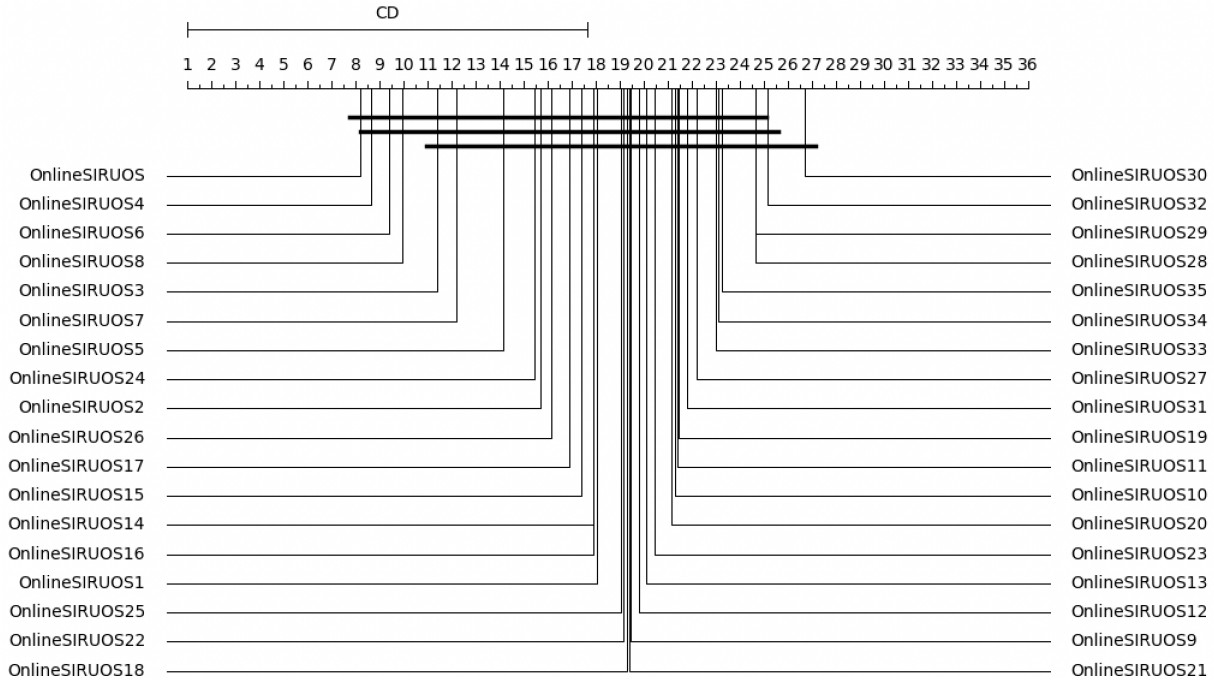

Figura 5.4: Gráfico de Nemenyi sobre variações do OnlineSIRUOS com resultados de F1-Score sobre cenários reais.

de diferença crítica mais alta para F1-score foram consideradas, o que também inclui o OnlineSIRUOS6; em segundo foram somados os valores das posições das 3 métricas para cada variação e o resultado foi colocado em uma nova coluna referida como **Total** nas Tabelas 5.7 e 5.8; por fim, foi avaliado qual variação obteve o menor resultado ao somar

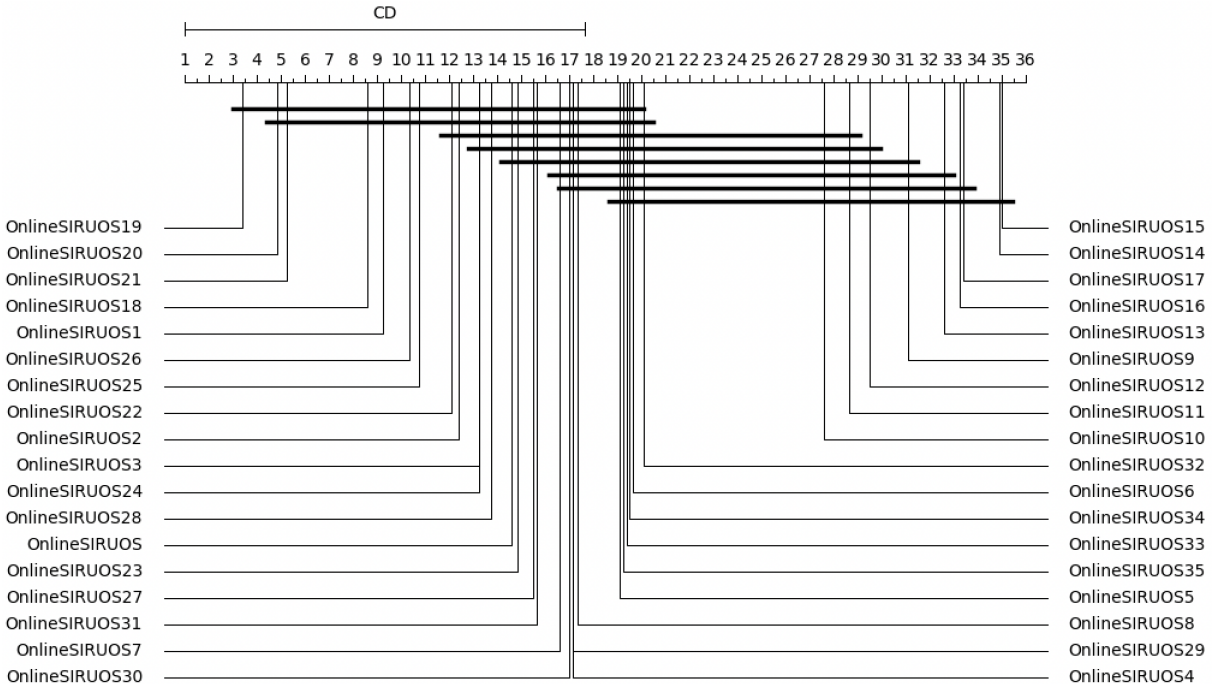

Figura 5.5: Gráfico de Nemenyi sobre variações do OnlineSIRUOS com resultados de Tempo de Processamento sobre cenários reais.

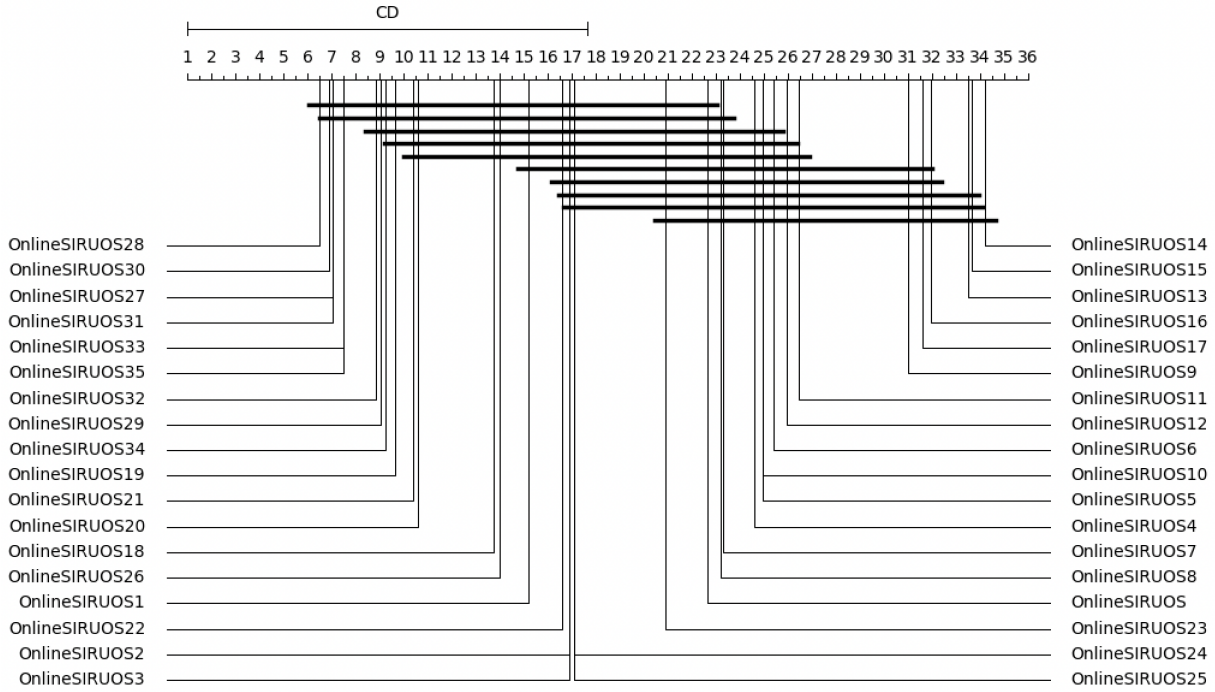

Figura 5.6: Gráfico de Nemenyi sobre variações do OnlineSIRUOS com resultados de Custo de Modelo sobre cenários reais.

os valores dessas colunas. Com isso, é possível observar que o OnlineSIRUOS21, com a 1ª posição nos experimentos sintéticos e a 3ª posição nos experimentos reais, obteve o melhor desempenho segundo esse critério de avaliação, somando 55 pontos. Todavia, é possível observar que a variação OnlineSIRUOS26 ficou logo atrás com 56 pontos, mas

Tabela 5.7: Ranking das variações de OnlineSIRUOS sobre os dados sintéticos.

Variação	**Tempo de Processamento**	**Custo de Modelo**	**F1-score**	**Total**
OnlineSIRUOS28	1	1	-	2
OnlineSIRUOS27	2	1	-	3
OnlineSIRUOS29	4	4	-	8
OnlineSIRUOS30	7	1	-	8
OnlineSIRUOS32	6	2	-	8
OnlineSIRUOS34	5	3	-	8
OnlineSIRUOS35	8	1	-	9
OnlineSIRUOS19	9	5	-	14
OnlineSIRUOS21	11	6	8	25
OnlineSIRUOS31	3	1	21	25
OnlineSIRUOS26	13	10	5	28
OnlineSIRUOS33	8	1	20	29
OnlineSIRUOS20	10	7	14	31
OnlineSIRUOS1	14	9	10	33
OnlineSIRUOS10	20	13	-	33
OnlineSIRUOS3	16	12	7	35
OnlineSIRUOS2	15	11	11	37
OnlineSIRUOS24	19	16	2	37
OnlineSIRUOS25	13	12	17	42
OnlineSIRUOS12	22	14	7	43
OnlineSIRUOS22	17	14	12	43
OnlineSIRUOS18	12	8	24	44
OnlineSIRUOS	23	18	6	47
OnlineSIRUOS8	25	20	4	49
OnlineSIRUOS11	21	15	15	51
OnlineSIRUOS23	18	17	19	54
OnlineSIRUOS6	30	25	1	56
OnlineSIRUOS17	31	23	3	57
OnlineSIRUOS7	24	19	16	59
OnlineSIRUOS4	27	24	9	60
OnlineSIRUOS15	33	28	1	62
OnlineSIRUOS9	26	21	18	65
OnlineSIRUOS16	28	22	16	66
OnlineSIRUOS13	32	27	13	72
OnlineSIRUOS5	29	26	22	77
OnlineSIRUOS14	34	29	23	86

conta com posições de F1-score razoavelmente mais altas enquanto mantém posições um pouco mais baixas segundo as outras métricas quando comparadas com os resultados do OnlineSIRUOS21. Portanto, como o F1-score possui uma leve prioridade sobre as outras duas métricas, o OnlineSIRUOS26 foi considerado como o OnlineSIRUOSΔ.

Tabela 5.8: Ranking das variações de OnlineSIRUOS sobre os dados reais.

Variação	**Tempo de Processamento**	**Custo de Modelo**	**F1-score**	**Total**
OnlineSIRUOS30	17	2	-	19
OnlineSIRUOS26	6	12	10	28
OnlineSIRUOS3	10	15	5	30
OnlineSIRUOS21	3	9	18	30
OnlineSIRUOS32	25	5	-	30
OnlineSIRUOS	12	19	1	32
OnlineSIRUOS1	5	13	14	32
OnlineSIRUOS18	4	11	17	32
OnlineSIRUOS2	9	15	9	33
OnlineSIRUOS24	10	17	8	35
OnlineSIRUOS20	2	10	23	35
OnlineSIRUOS19	1	8	26	35
OnlineSIRUOS25	7	16	15	38
OnlineSIRUOS22	8	14	16	38
OnlineSIRUOS4	18	22	2	42
OnlineSIRUOS8	19	20	4	43
OnlineSIRUOS7	16	21	6	43
OnlineSIRUOS28	11	1	32	44
OnlineSIRUOS31	15	3	27	45
OnlineSIRUOS27	14	3	28	45
OnlineSIRUOS5	20	23	7	50
OnlineSIRUOS6	24	24	3	51
OnlineSIRUOS23	13	18	22	53
OnlineSIRUOS33	22	4	29	55
OnlineSIRUOS35	21	4	31	56
OnlineSIRUOS29	18	6	32	56
OnlineSIRUOS34	23	7	30	60
OnlineSIRUOS17	32	28	11	71
OnlineSIRUOS16	31	29	13	73
OnlineSIRUOS12	28	25	20	73
OnlineSIRUOS10	26	23	24	73
OnlineSIRUOS9	29	27	19	75
OnlineSIRUOS15	34	31	12	77
OnlineSIRUOS14	33	32	13	78
OnlineSIRUOS11	27	26	25	78
OnlineSIRUOS13	30	30	21	81

5.3 Gráficos de Radar

Apesar de o foco do presente trabalho ser na análise das métricas de F1-score, custo de modelo e tempo de processamento, demais métricas relevantes para cenários desbalanceados de classificação foram obtidas. O objetivo é apresentar uma visão geral da viabilidade do algoritmo OnlineSIRUOS, visto que de nada adianta ser excelente em uma perspectiva de análise e completamente disfuncional em outras. Essa etapa contou com uma análise de resultados das métricas de Precision, Recall e F1-score, elaborados sob o formato de gráficos de radar, visando observar o comportamento que cada uma dessas métricas obteve nos grupos de experimentos usados. Foram considerados tanto os valores completos das métricas como os seus valores parciais para cada classe. Ou seja, **"F1-score - 0"** se refere aos valores dessa métrica em relação à classe majoritária e **"F1-score - 1"** aos valores da classe minoritária.

A avaliação se deu sobre as médias gerais dos experimentos sintéticos e reais, para considerar uma visão abrangente dos testes realizados, e também incluiu análises sobre as médias dos experimentos sintéticos que contavam com níveis de desbalanceamento de 95%-5%, 99%-1% e 99,5%-0,5%, visando isolar os casos mais complexos de serem classificados e também de maior interesse para esse projeto. Contudo, cabe ressaltar que a discussão reportada não é exaustiva dada a extensão que o documento teria. Consequentemente, a presente etapa de avaliação serve somente para demonstrar que a técnica OnlineSIRUOS possui uma fundação sólida de desempenho no caso geral e principalmente sobre os casos com desbalanceamento mais acentuado.

5.3.1 Caso Geral dos Cenários Sintéticos

Nessa etapa, os resultados das métricas aqui analisadas são uma média dos valores obtidos através de experimentos com bases de dados sintéticas (Seção 5.1.2). Esses valores estão elencados na Figura 5.7. As Tabelas 5.9, 5.10 e 5.11 mostram esses dados de maneira numérica, sendo ordenados de forma decrescente pela primeira coluna.

Os valores de Precision, Recall e F1-score da variação selecionada do OnlineSIRUOS ficaram entre os mais altos. No geral, todos os resultados parciais das classes minoritárias de todas as métricas mostraram valores mais baixos que suas contrapartidas das classes majoritárias, mas isso já era esperado devido ao desbalanceamento existente em vários dos experimentos sintéticos.

Em relação à métrica de F1-score, o OnlineSIRUOSΔ liderou nas primeiras posições. A diferença do OnlineSIRUOSΔ para o próximo algoritmo, que foi o CSARF, foi

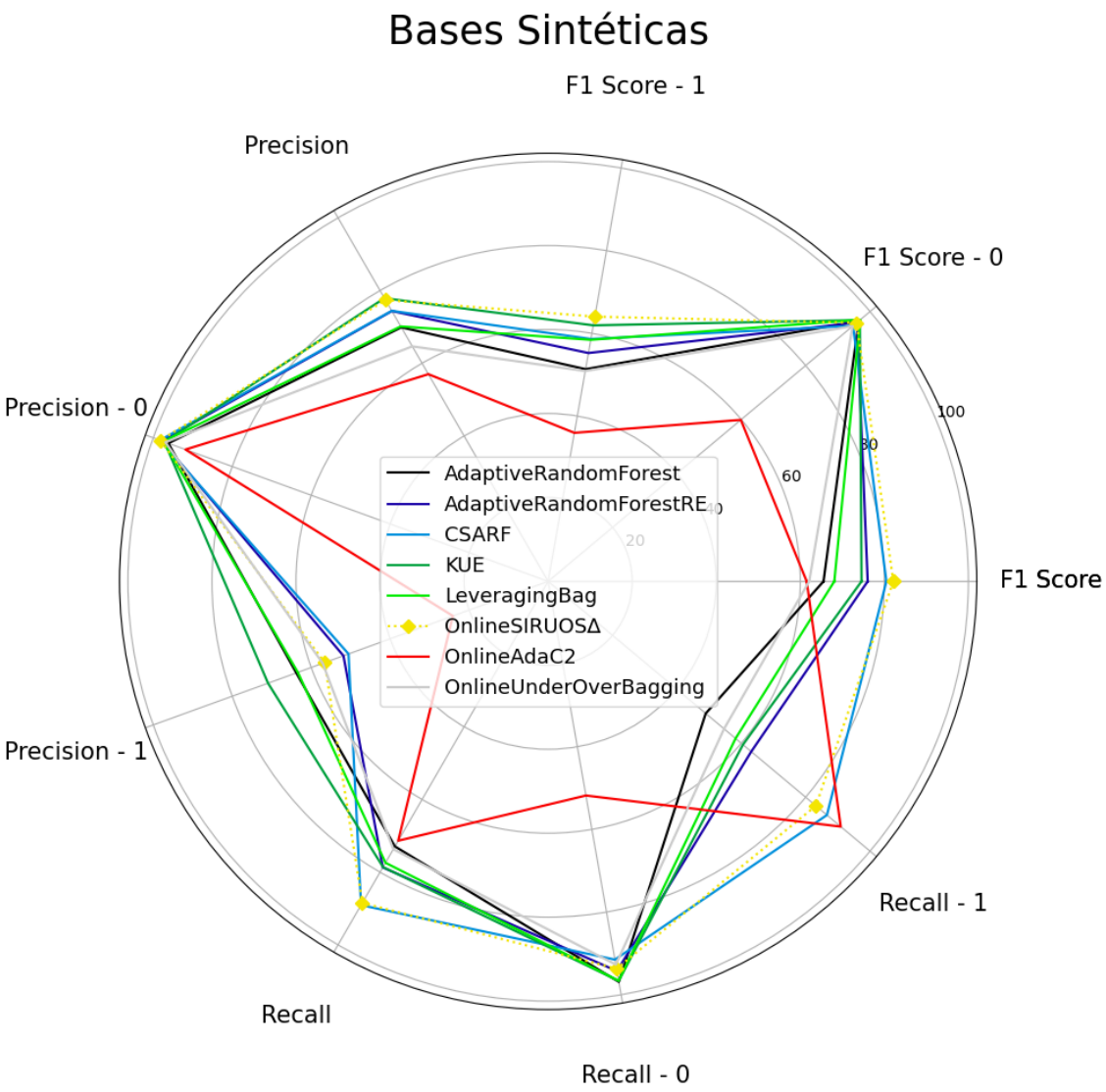

Figura 5.7: Gráfico de Radar mostrando a média dos resultados dos experimentos baseados em bases sintéticas.

Tabela 5.9: Dados de F1-score dos gráficos de radar sobre bases sintéticas em ordem decrescente.

Algoritmo	**F1-score**	**F1-score - 0**	**F1-score - 1**
OnlineSIRUOSΔ	82,14	95,81	64,04
CSARF	80,49	94,71	58,63
AdaptiveRandomForestRE	76,03	95,95	55,23
KUE	74,57	96,84	61,93
LeveragingBag	68,09	96,94	58,43
AdaptiveRandomForest	65,52	96,47	51,32
OnlineUnderOverBagging	62,12	94,58	50,86
OnlineAdaC2	61,52	59,85	35,93

Tabela 5.10: Dados de Precision dos gráficos de radar sobre bases sintéticas em ordem decrescente.

Algoritmo	**Precision**	**Precision - 0**	**Precision - 1**
KUE	77,90	97,18	70,93
OnlineSIRUOSΔ	77,40	98,22	56,59
CSARF	74,41	98,29	50,54
AdaptiveRandomForestRE	74,36	97,89	51,83
LeveragingBag	70,13	97,45	63,38
AdaptiveRandomForest	69,88	96,17	63,77
OnlineUnderOverBagging	64,65	96,94	57,09
OnlineAdaC2	56,99	91,83	24,11

Tabela 5.11: Dados de Recall dos gráficos de radar sobre bases sintéticas em ordem decrescente.

Algoritmo	**Recall**	**Recall - 0**	**Recall - 1**
CSARF	89,05	91,50	86,60
OnlineSIRUOSΔ	88,43	93,69	83,16
AdaptiveRandomForestRE	78,68	94,27	63,09
KUE	78,58	96,65	60,51
LeveragingBag	77,41	96,60	58,22
OnlineUnderOverBagging	73,68	92,71	54,64
AdaptiveRandomForest	72,91	96,88	48,95
OnlineAdaC2	71,34	51,80	90,89

de cerca de 2,01%.

Olhando para o Precision, o KUE ficou na liderança principalmente por seu ótimo desempenho em relação a parcial Precision - 1, que ficou 20,22% acima do segundo colocado. Mas ao analisar a métrica final, o KUE obteve uma vantagem em relação ao segundo colocado de 0,64%, o qual é o OnlineSIRUOSΔ. Observando essas medidas, é possível concluir que não houve uma diferença grande entre essas colocações. Mas ao comparar o OnlineSIRUOSΔ e o CSARF, que foi o terceiro colocado, há uma diferença de mais de 3,65%.

Por fim, em termos de Recall, o CSARF ficou em primeiro lugar do *ranking*, seguido pelo OnlineSIRUOSΔ. As diferenças entre o primeiro e o segundo foi de quase 0,8%. Comparando o segundo com o terceiro colocado, que foi o ARFRE, há uma diferença muito maior do que a vista anteriormente para essa métrica, de cerca de 11,03%.

Observando a amplitude dos resultados disponíveis, é possível observar que o método proposto, OnlineSIRUOSΔ, possui desempenho competitivo quando comparado a todas as outras técnicas analisadas. Visto que há métricas adicionais de fora do escopo da análise principal e sobre as mesmas os resultados se mantêm promissores nos cenários sintéticos, é possível afirmar que pelo menos há um indício de relevância da técnica criada

nesse projeto. As próximas seções discutem os cenários sintéticos de acordo com percentuais de desbalanceamento isolados, o que permitirá vislumbrar eventuais diferenças entre os métodos avaliados nestes cenários.

5.3.1.1 Análise do Cenário Sintético 95% - 5%

A Figura 5.8 apresenta os resultados médios obtidos por diferentes classificadores apenas em experimentos sintéticos com desbalanceamento de 95% - 5%. As Tabelas 5.12, 5.13 e 5.14 mostram esses dados de maneira numérica, sendo ordenados de forma decrescente pela primeira coluna.

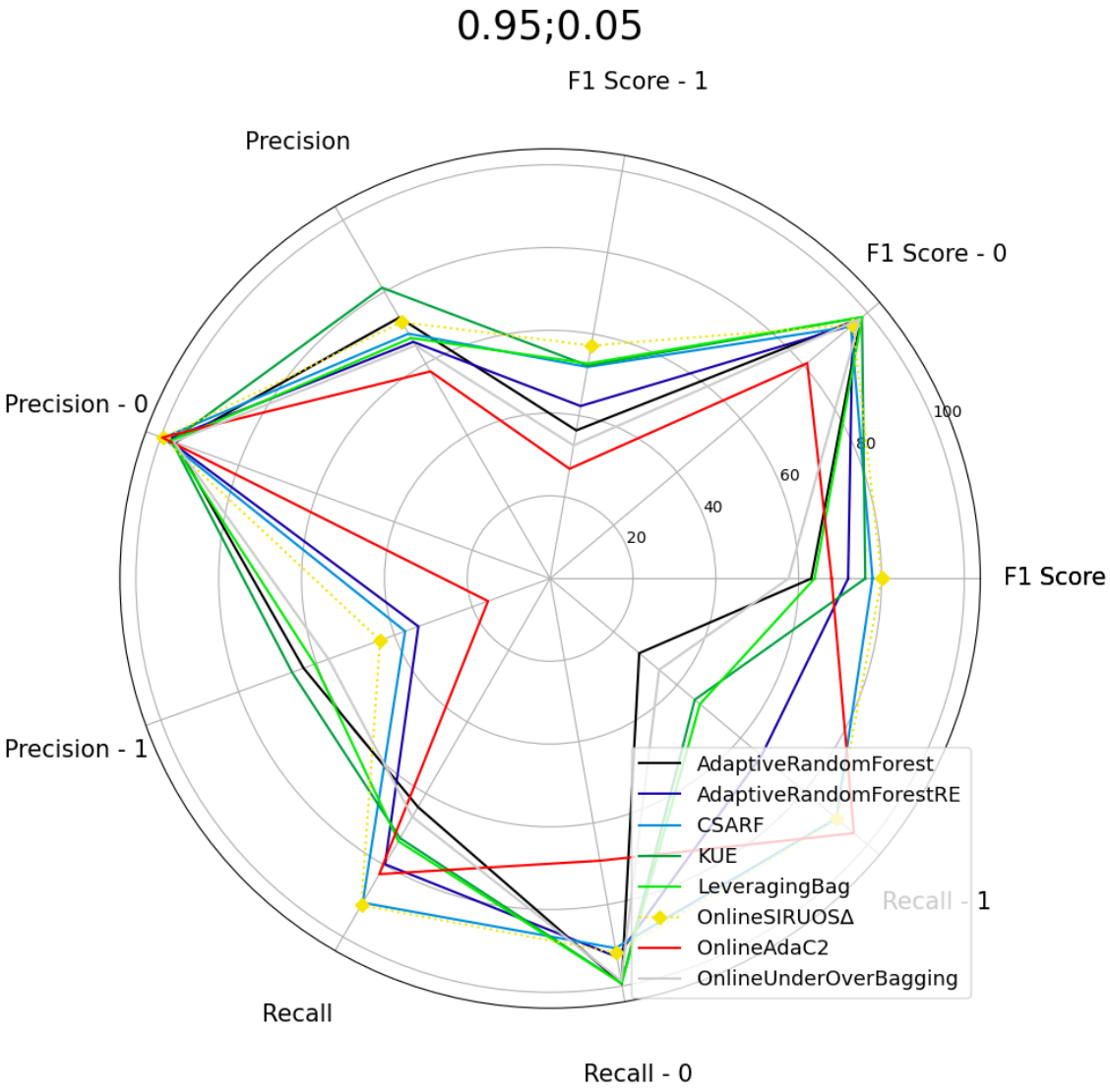

Figura 5.8: Gráfico de Radar mostrando a média dos resultados dos experimentos baseados em bases sintéticas com desbalanceamento de 95% - 5%.

Em relação à análise geral de resultados das bases sintéticas, aqui é possível notar

Tabela 5.12: Dados de F1-score dos gráficos de radar sobre bases sintéticas com desbalanceamento de 95% - 5% em ordem decrescente.

Algoritmo	**F1-score**	**F1-score - 0**	**F1-score - 1**
OnlineSIRUOSΔ	80,04	95,41	57,16
CSARF	77,82	94,80	51,92
KUE	76,07	98,26	52,25
AdaptiveRandomForestRE	71,93	95,32	42,33
OnlineAdaC2	67,95	81,02	26,92
LeveragingBag	63,82	98,32	52,77
AdaptiveRandomForest	63,04	97,88	36,35
OnlineUnderOverBagging	57,53	97,63	32,67

Tabela 5.13: Dados de Precision dos gráficos de radar sobre bases sintéticas com desbalanceamento de 95% - 5% em ordem decrescente.

Algoritmo	**Precision**	**Precision - 0**	**Precision - 1**
KUE	81,16	97,22	66,28
AdaptiveRandomForest	72,83	96,35	63,33
OnlineSIRUOSΔ	71,56	99,44	43,68
CSARF	68,33	99,42	37,24
LeveragingBag	67,13	97,30	60,47
AdaptiveRandomForestRE	66,03	98,23	33,83
OnlineUnderOverBagging	65,05	96,68	56,93
OnlineAdaC2	57,80	99,66	15,94

Tabela 5.14: Dados de Recall dos gráficos de radar sobre bases sintéticas com desbalanceamento de 95% - 5% em ordem decrescente.

Algoritmo	**Recall**	**Recall - 0**	**Recall - 1**
OnlineSIRUOSΔ	91,12	91,86	90,38
CSARF	90,46	90,70	90,21
OnlineAdaC2	82,48	69,24	95,73
AdaptiveRandomForestRE	79,72	92,85	66,59
LeveragingBag	73,29	99,37	47,21
KUE	72,45	99,34	45,57
OnlineUnderOverBagging	66,51	98,68	34,34
AdaptiveRandomForest	63,77	99,48	28,06

uma grande mudança no padrão observado: tendência de acertos maiores sobre as métricas parciais relativas às classes majoritárias, diferença de desempenho entre os algoritmos analisados se torna mais nítida, entre outros pontos. Isso obviamente não foi regra para todos os algoritmos e em todas as métricas, como pode ser visto ao comparar os dados de Recall dessa etapa de análise com os obtidos na Seção 5.7. Mas no geral, isso acontece especialmente porque a taxa de desbalanceamento se aproxima de valores altos (acima de 90% - 10%) e a dificuldade de classificar exemplos das classes minoritárias se torna cada

vez maior, particularmente para algoritmos menos preparados para lidar com desbalanceamento. Todavia, os algoritmos ARF e LeveragingBag, mesmo sendo dessa categoria, conseguiram resultados competitivos quando comparados às outras técnicas. Um outro caso atípico foi o do OnlineAdaC2, que teve nitidamente mais dificuldade no F1-score - 0 e Recall - 0 do que todos os outros algoritmos, mesmo sendo uma técnica com preparo para lidar com desbalanceamento.

Ao isolarmos a métrica de F1-score, é possível observar que a variação de OnlineSIRUOS fica na primeira posição. Ao comparar o OnlineSIRUOSΔ com o terceiro colocado que foi o CSARF, há uma diferença maior, chegando a 2,77%. Aqui, a variação de OnlineSIRUOS selecionada também obteve os maiores valores da parcial de F1-score -1, indicando seu melhor desempenho nas classes minoritárias nesse caso.

Em relação ao Precision, o algoritmo KUE ficou na primeira posição isolado, com 10,26% de diferença para o ARFRE, que foi o segundo colocado. O terceiro colocado foi o OnlineSIRUOSΔ, com diferença de 11,83% em relação ao primeiro colocado em termos de Precision. É importante notar que aqui o KUE foi um *outlier* devido ao seu desempenho em ambas as parciais dessa métrica, o que não necessariamente indica que os outros algoritmos devem ser ignorados. A diferença do segundo para o terceiro colocado foi de somente 1,74%.

Finalmente, ao analisar o Recall, em primeiro lugar está o OnlineSIRUOSΔ. A diferença do primeiro com o segundo algoritmo com melhores resultados, que foi o CSARF, é de cerca de 0,72%.

Considerando todos os valores apresentados, é possível avaliar que o OnlineSIRUOSΔ é competitivo segundo Precision, Recall e principalmente em relação a F1-score, onde nitidamente se sai melhor que todos os outros algoritmos. Mesmo o KUE obtendo uma ótima vantagem em relação ao Precision, o mesmo não se manteve bem em relação ao Recall, levando a valores menores de F1-score, que é dependente dos resultados dessas duas métricas.

5.3.1.2 Análise do Cenário Sintético 99% - 1%

Na Figura 5.9, os dados são baseados na média dos resultados dos experimentos sintéticos com desbalanceamento de 99% - 1%. As Tabelas 5.15, 5.16 e 5.17 mostram esses dados de maneira numérica, sendo ordenados de forma decrescente pela primeira coluna.

Conforme o que foi mostrado na Seção 5.3.1.1 e ao contrário do que se esperaria com o aumento do desbalanceamento, aqui há uma diminuição da diferença de desempenho de várias técnicas. Claramente é possível observar que há bons resultados para a variação de

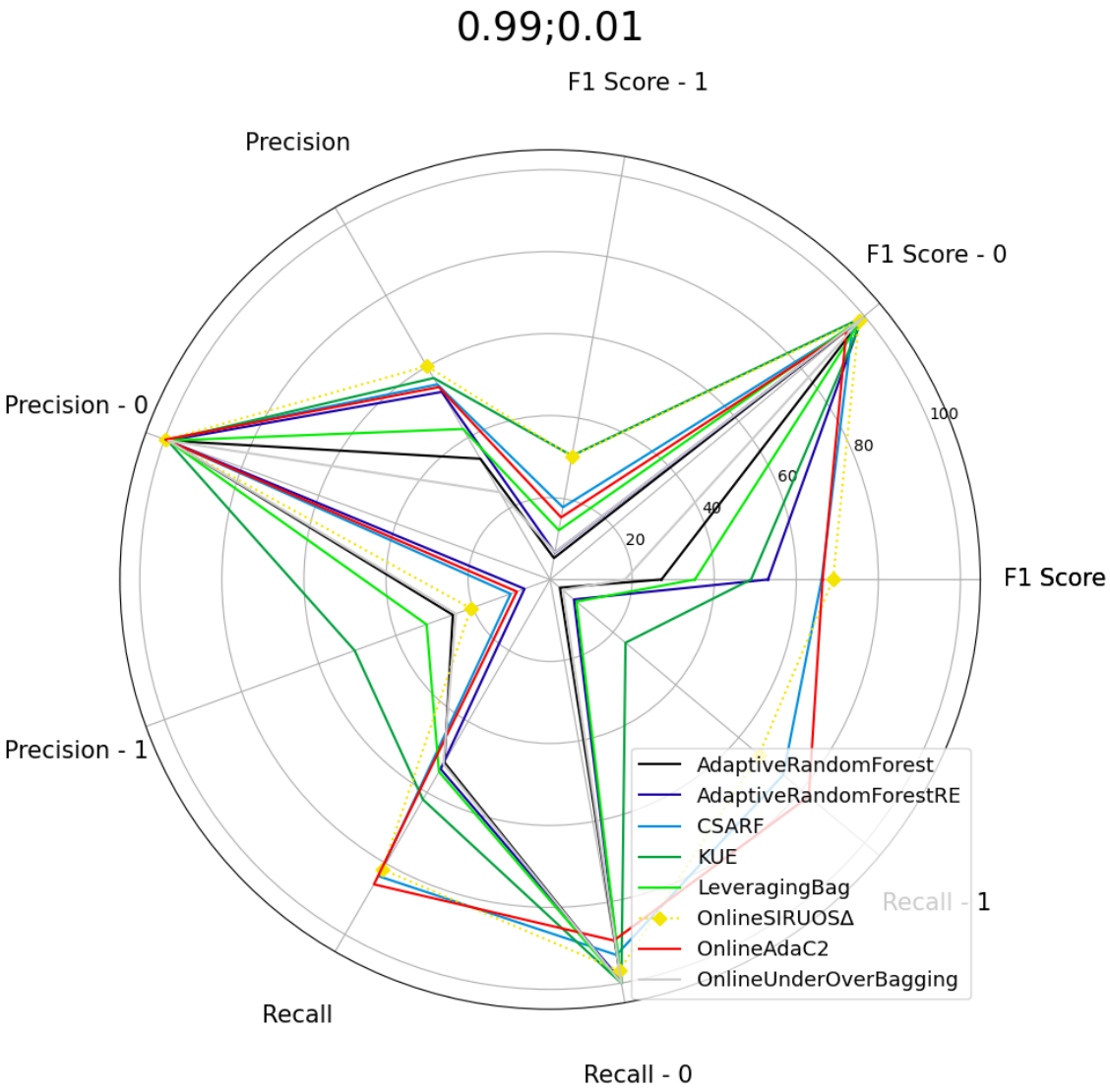

Figura 5.9: Gráfico de Radar mostrando a média dos resultados dos experimentos baseados em bases sintéticas com desbalanceamento de 99% - 1%.

Tabela 5.15: Dados de F1-score dos gráficos de radar sobre bases sintéticas com desbalanceamento de 99% - 1% em ordem decrescente.

Algoritmo	**F1-score**	**F1-score - 0**	**F1-score - 1**
OnlineSIRUOSΔ	68,95	98,26	30,52
OnlineAdaC2	66,41	94,25	15,39
CSARF	66,17	96,24	17,90
AdaptiveRandomForestRE	53,09	99,13	6,69
KUE	48,96	99,62	30,67
LeveragingBag	35,26	99,54	12,19
AdaptiveRandomForest	27,14	99,51	5,32
OnlineUnderOverBagging	18,22	99,52	6,77

Tabela 5.16: Dados de Precision dos gráficos de radar sobre bases sintéticas com desbalanceamento de 99% - 1% em ordem decrescente.

Algoritmo	**Precision**	**Precision - 0**	**Precision - 1**
OnlineSIRUOSΔ	60,11	99,65	20,58
KUE	56,77	99,25	50,68
CSARF	54,97	99,72	10,23
OnlineAdaC2	54,21	99,80	8,61
AdaptiveRandomForestRE	52,86	99,08	6,64
LeveragingBag	42,42	99,09	32,04
AdaptiveRandomForest	34,04	99,04	25,23
OnlineUnderOverBagging	24,62	99,05	24,46

Tabela 5.17: Dados de Recall dos gráficos de radar sobre bases sintéticas com desbalanceamento de 99% - 1% em ordem decrescente.

Algoritmo	**Recall**	**Recall - 0**	**Recall - 1**
OnlineAdaC2	85,87	89,44	82,31
CSARF	83,58	93,01	74,16
OnlineSIRUOSΔ	81,69	96,91	66,47
KUE	61,99	99,99	23,98
LeveragingBag	54,18	99,99	8,37
AdaptiveRandomForestRE	53,37	99,17	7,57
OnlineUnderOverBagging	52,00	99,99	4,01
AdaptiveRandomForest	51,54	99,99	3,10

OnlineSIRUOS, particularmente em relação ao F1-score. O primeiro e o segundo colocados ficaram com o OnlineSIRUOSΔ e o OnlineAdaC2 respectivamente, com uma diferença de um pouco mais de 2,75%. A diferença do OnlineAdaC2 para o terceiro colocado que foi o CSARF foi de 0,36%. Esses são valores a serem lembrados, principalmente ao isolarmos os resultados em relação à métrica parcial F1-score - 1. Somente o OnlineSIRUOS e o KUE se encontram com os valores mais altos. Outra observação é que notoriamente, com o desbalanceamento mais alto, está ficando mais difícil para todos os algoritmos manterem resultados tão bons quanto os já alcançados nas seções anteriores e isso acontece para todas as métricas como é mostrado abaixo.

No Precision, o primeiro lugar também ficou com a variação do OnlineSIRUOS. A diferença do primeiro para o segundo algoritmo foi de 5,88%, o qual foi o KUE. Mesmo assim é importante notar o bom desempenho do KUE sobre a métrica parcial de Precision - 1, que ultrapassa os resultados de todos os outros algoritmos.

Já em relação ao Recall, os OnlineSIRUOS ainda demonstrou valores altos, mas passou a ficar abaixo de outros algoritmos, como o OnlineAdaC2, que agora obtiveram resultados muito melhores com um desbalanceamento mais alto. A diferença do OnlineAdaC2 para o segundo e o terceiro colocados, que foram o CSARF e o OnlineSIRUOSΔ,

foi de 2,67% e 4,87% respectivamente.

Por fim, é possível concluir que o OnlineSIRUOS possui resultados competitivos e a relevância do mesmo se mostra cada vez maior conforme o desbalanceamento aumenta, particularmente em relação ao F1-score.

5.3.1.3 Análise do Cenário Sintético 99,5% - 0,5%

A Figura 5.10 apresenta os resultados médios obtidos, desta vez, para experimentos sintéticos com desbalanceamento de 99,5% - 0,5%. As Tabelas 5.18, 5.19 e 5.20 mostram esses dados de maneira numérica, sendo ordenados de forma decrescente pela primeira coluna.

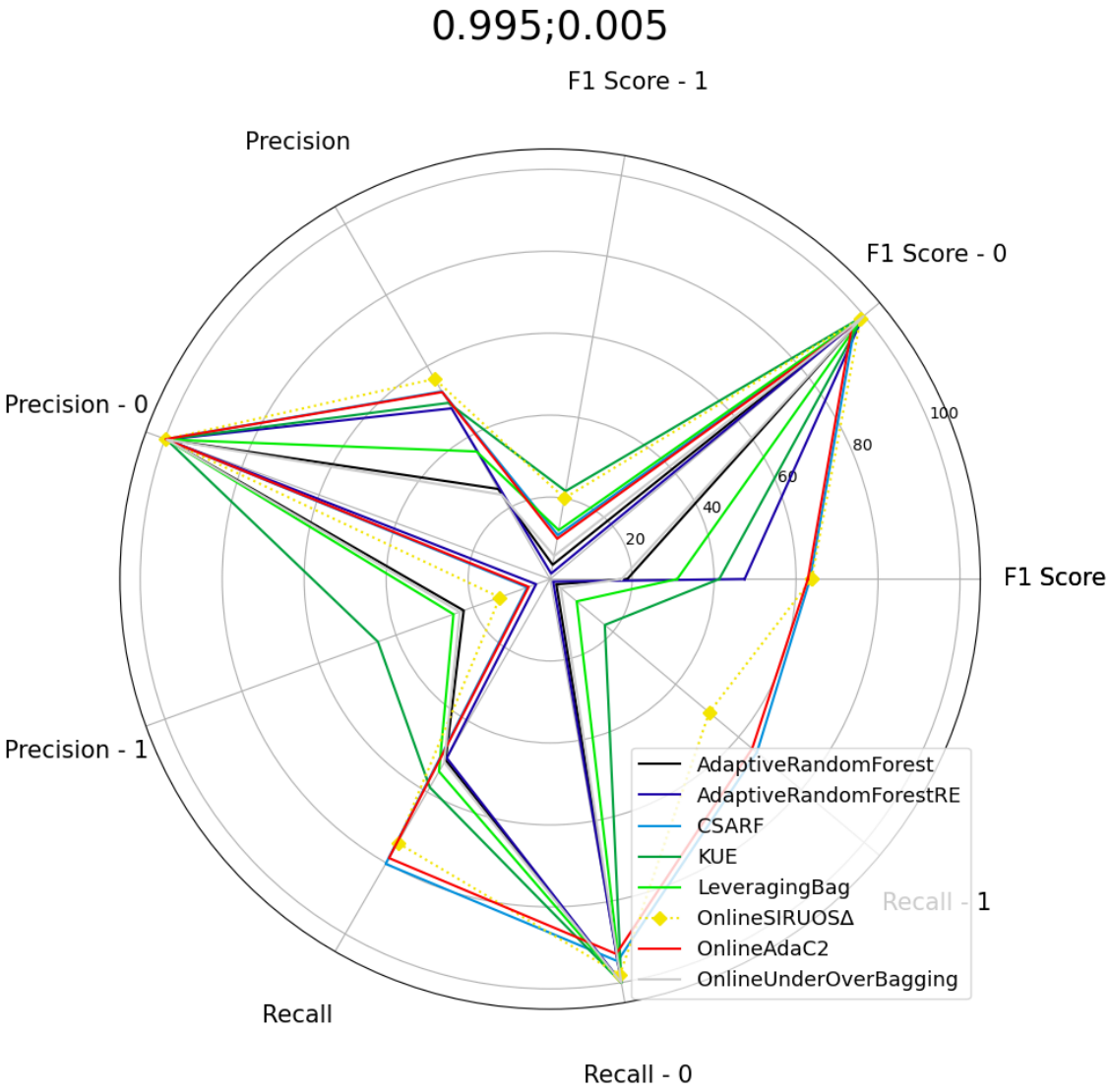

Figura 5.10: Gráfico de Radar mostrando a média dos resultados dos experimentos baseados em bases sintéticas com desbalanceamento de 99,5% - 0,5%.

Tabela 5.18: Dados de F1-score dos gráficos de radar sobre bases sintéticas com desbalanceamento de 99,5% - 0,5% em ordem decrescente.

Algoritmo	**F1-score**	**F1-score - 0**	**F1-score - 1**
OnlineSIRUOSΔ	63,80	98,88	20,06
CSARF	63,43	97,18	10,87
OnlineAdaC2	62,88	96,15	9,99
AdaptiveRandomForestRE	47,43	99,70	1,36
KUE	41,25	99,79	21,79
LeveragingBag	30,99	99,77	12,11
AdaptiveRandomForest	18,81	99,75	3,53
OnlineUnderOverBagging	17,43	99,76	5,72

Tabela 5.19: Dados de Precision dos gráficos de radar sobre bases sintéticas com desbalanceamento de 99,5% - 0,5% em ordem decrescente.

Algoritmo	**Precision**	**Precision - 0**	**Precision - 1**
OnlineSIRUOSΔ	56,40	99,75	13,06
CSARF	52,88	99,82	5,94
OnlineAdaC2	52,68	99,81	5,55
KUE	49,63	99,59	44,71
AdaptiveRandomForestRE	48,09	99,51	3,64
LeveragingBag	35,95	99,55	25,09
AdaptiveRandomForest	25,40	99,51	22,43
OnlineUnderOverBagging	23,80	99,52	23,45

Tabela 5.20: Dados de Recall dos gráficos de radar sobre bases sintéticas com desbalanceamento de 99,5% - 0,5% em ordem decrescente.

Algoritmo	**Recall**	**Recall - 0**	**Recall - 1**
CSARF	80,25	94,67	65,83
OnlineAdaC2	78,61	92,89	64,34
OnlineSIRUOSΔ	74,35	98,04	50,65
KUE	58,70	99,99	17,41
LeveragingBag	54,23	99,99	8,47
OnlineUnderOverBagging	51,67	99,99	3,35
AdaptiveRandomForest	50,99	99,99	1,98
AdaptiveRandomForestRE	50,42	99,89	0,95

Como observado na Seção 5.3.1.1, com o aumento do desbalanceamento dos casos em análise, a dificuldade de avaliação que os algoritmos devem enfrentar também aumenta. Por isso, os resultados apresentados se mostram muito similares aos da Seção 5.3.1.3, mas com valores um pouco inferiores. A variante selecionada do OnlineSIRUOS apresenta as mesmas dificuldades mostradas na seção anterior, como a dificuldade de obter resultados melhores em relação a métrica parcial relativa a classe minoritária Precision - 1, todavia, ainda sim consegue resultados competitivos quando comparada às outras

técnicas da literatura. É possível também notar que técnicas como o CSARF e o OnlineAdaC2 conseguiram ter um decremento de desempenho menor do que o que houve com o OnlineSIRUOS quando comparamos os resultados das Seções 5.3.1.3 e 5.3.1.1.

No caso da métrica F1-score, o OnlineSIRUOSΔ ficou em primeiro lugar, seguido do CSARF e do OnlineAdaC2. A diferença entre o primeiro e os próximos colocados é de 0,58% e 1,46%. É importante considerar que em cenários de alto desbalanceamento, é bem provável que as diferenças dos algoritmos seriam muito pequenas devido ao nível de dificuldade que os cenários possuem para o processo de classificação.

Já em relação ao Precision, há um pouco mais de diferença, mas nada tão significante como o que foi encontrado em algumas métricas das seções anteriores. O OnlineSIRUOSΔ ficou em primeiro mesmo tendo maiores dificuldades para conseguir bons resultados de Precision.

Finalmente, em relação ao Recall, o primeiro colocado foi o CSARF. Esse algoritmo consegue obter bons resultados de Recall em bases desbalanceadas e se destaca quanto mais acentuado é esse desbalanceamento. Em seguida, como vem acontecendo nas seções anteriores, o OnlineAdaC2 também conseguiu obter um bom desempenho e por isso ficou em segundo. A diferença do primeiro para o segundo é de 2,04%. Já a diferença desse último para o terceiro lugar, que foi o OnlineSIRUOSΔ, foi de 5,42%, denotando agora um desempenho consideravelmente menor do OnlineSIRUOS em relação às melhores técnicas.

Ao analisar o caso geral, as conclusões dessa etapa de avaliação vão ao encontro das conclusões definidas nas seções anteriores. A diferença está no fato de que o OnlineSIRUOSΔ obteve resultados bons em relação ao F1-score que só não foram melhores devido aos valores de Recall obtidos.

5.3.2 Cenários Reais

A Figura 5.11 mostra a média dos resultados das métricas de F1-score, Precision e Recall sobre os experimentos baseados em cenários reais apresentados no Capítulo 5.1.3. Nele também se encontram os valores parciais de cada métrica para cada classe. As Tabelas 5.21, 5.22 e 5.23 mostram esses dados de maneira numérica, sendo ordenados de forma decrescente pela primeira coluna.

No comparativo do F1-score, o CSARF ficou na primeira posição. Contudo a diferença para o segundo e o terceiro colocado, que foi o OnlineSIRUOSΔ e o ARFRE, foi de 1,11% e 4,08% respectivamente, fazendo dessas as menores diferenças encontradas entre os primeiros colocados de cada métrica completa desse cenário analisado.

Já na relação das métricas de Precision, os valores das métricas de ambas as parciais

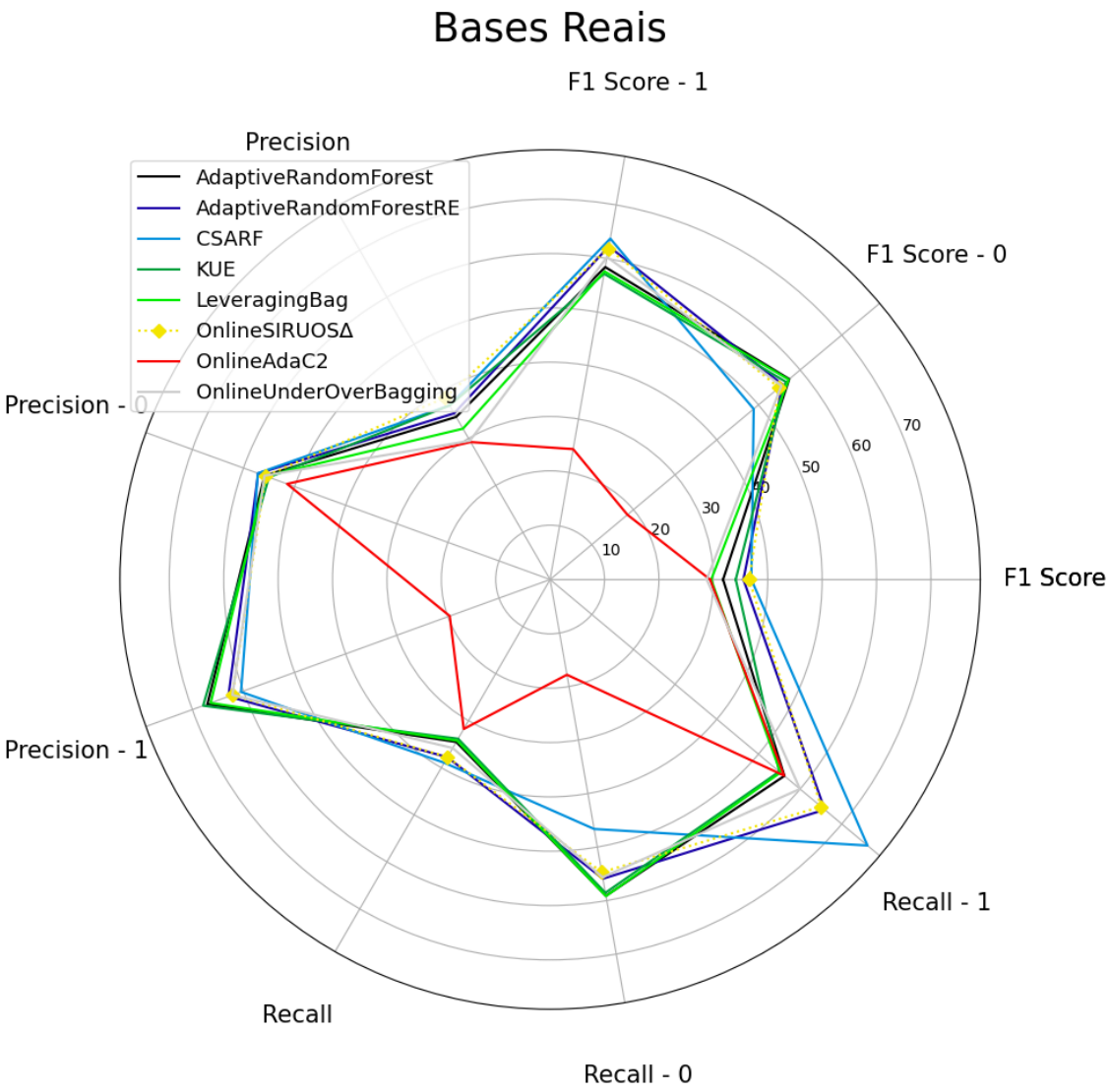

Figura 5.11: Gráfico de Radar mostrando a média dos resultados dos experimentos baseados em bases reais.

Tabela 5.21: Dados de F1-score dos gráficos de radar sobre bases reais em ordem decrescente.

Algoritmo	**F1-score**	**F1-score - 0**	**F1-score - 1**
CSARF	36,98	48,84	63,71
OnlineSIRUOSΔ	36,57	54,92	61,85
AdaptiveRandomForestRE	35,53	55,79	62,27
KUE	34,12	56,59	57,14
AdaptiveRandomForest	31,79	57,39	58,38
LeveragingBag	29,61	57,25	57,58
OnlineAdaC2	29,40	18,54	24,37
OnlineUnderOverBagging	28,78	55,49	60,27

Tabela 5.22: Dados de Precision dos gráficos de radar sobre bases reais em ordem decrescente.

Algoritmo	**Precision**	**Precision - 0**	**Precision - 1**
OnlineSIRUOSΔ	38,64	55,70	62,16
CSARF	36,99	57,18	60,47
KUE	36,97	55,01	67,88
AdaptiveRandomForestRE	35,30	57,11	62,94
AdaptiveRandomForest	34,56	55,88	67,04
LeveragingBag	32,05	55,64	66,40
OnlineUnderOverBagging	29,49	55,88	62,14
OnlineAdaC2	29,20	51,45	19,63

Tabela 5.23: Dados de Recall dos gráficos de radar sobre bases reais em ordem decrescente.

Algoritmo	**Recall**	**Recall - 0**	**Recall - 1**
CSARF	38,93	46,61	76,16
AdaptiveRandomForestRE	37,76	55,91	65,89
OnlineSIRUOSΔ	37,71	54,70	65,02
OnlineUnderOverBagging	35,81	55,64	59,96
AdaptiveRandomForest	34,51	59,10	56,22
LeveragingBag	34,22	59,16	55,27
KUE	33,83	58,54	54,98
OnlineAdaC2	31,75	17,79	56,14

ficaram muito próximas. O único resultado destoante foi o Precision - 1 do OnlineAdaC2, assim como ocorreu com o seu resultado na métrica Recall - 0. O melhor colocado em relação à métrica completa de Precision foi o OnlineSIRUOSΔ, com vantagem de 4,27% sobre o segundo colocado CSARF e 4,32% sobre o terceiro colocado que foi o KUE.

Ao analisar a métrica de Recall e suas parciais, o valor positivo mais significativo está em relação à métrica Recall - 1, onde o CSARF se destaca com cerca de 15,58% de diferença do ARFRE que tem o segundo maior valor, o qual vem a frente do OnlineSIRUOSΔ. O impacto disso é pequeno na métrica de Recall, visto que os resultados se equilibram com os valores de Recall - 0, mantendo a diferença da métrica completa menos impactante. Em termos da métrica completa, o CSARF ficou na liderança com 3% de diferença do segundo colocado, que foi o ARFRE, seguido pelo OnlineSIRUOSΔ, com 3,13% de diferença, todos em relação ao primeiro colocado.

Portanto em termos gerais, quase todas as técnicas apresentam desempenhos próximos, similar ao encontrado nas Seções 5.3.1.1, 5.3.1.2 e 5.3.1.3 em muitos casos. Isso ocorreu porque na média, quase todas as bases usadas possuem um desbalanceamento de no mínimo 70% - 30% e a maior parte não chega perto dos valores encontrados nas Seções apontadas anteriormente. Outro fenômeno importante é o fato de que a variação do On-

lineSIRUOS se mostrou competitiva em relação às 3 métricas analisadas. Considerando os valores de Precision, o OnlineSIRUOSΔ até mesmo se destacou a frente das outras técnicas, fato que só foi alcançado em experimentos sintéticos com níveis mais altos de desbalanceamento.

Por fim, fica visível que o OnlineSIRUOS não demonstra ter uma diferença de desempenho abrupta nesse ambiente, se mantendo até similar aos experimentos sintéticos em grande parte do que foi obtido. Além disso, quando comparados com os outros algoritmos, o mesmo padrão também se segue e há ainda mais indícios da competitividade do OnlineSIRUOS.

5.4 Gráficos de Nemenyi

Mesmo avaliando os gráficos de radar na Seção 5.3, não há comprovação suficiente da viabilidade do algoritmo proposto nesse trabalho. Para criar uma fundamentação mais concreta, é importante utilizar testes estatísticos. Os gráficos que serão mostrados nessa etapa são baseados nos testes de Friedman e Nemenyi sobre os resultados dos testes nas bases reais e cenários sintéticos.

Para cada experimento realizado, foi selecionada a média dos valores de F1-score obtidos. A diferença para as métricas de tempo de processamento e custo do modelo foi que ao invés de utilizar a média dos valores por experimento, o maior valor obtido é que foi utilizado, visto que somente interessa o tempo máximo de processamento e o pico de uso de memória para avaliar essas métricas.

5.4.1 Avaliação dos Cenários Sintéticos

Na Figura 5.12 são apresentados os dados de F1-score relativos aos experimentos com as bases sintéticas desbalanceadas. Os dados apresentados mostram um resultado interessante: a variação de OnlineSIRUOS mostra um bom desempenho quando comparado com as outras técnicas da literatura. Entretanto, não há comprovação estatística suficiente que denote o seu favoritismo. Outro fato interessante é que entre as técnicas empatadas na liderança de desempenho, uma delas é o LeveragingBag, o qual é um algoritmo não preparado para lidar com desbalanceamento. Isso provavelmente se deve ao fato de que este algoritmo conseguiu um bom desempenho em cenários com desbalanceamento menos acentuado, compensando eventuais dificuldades que ele veio a enfrentar com o desbalanceamento aumentando como mostrado nas Seções 5.3.1.1, 5.3.1.2 e 5.3.1.3.

A Figura 5.13 contêm os dados relacionados ao Tempo de Processamento que

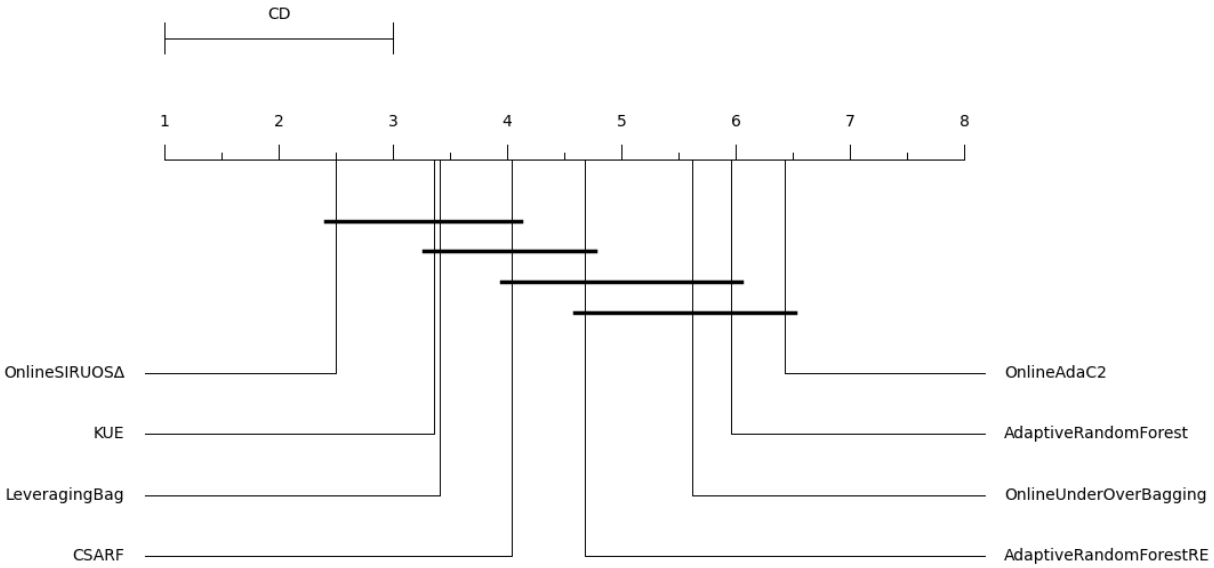

Figura 5.12: Gráfico de Nemenyi comparando algoritmos da literatura e os OnlineSIRUOS sobre cenários sintéticos desbalanceados baseados nos valores de F1-score.

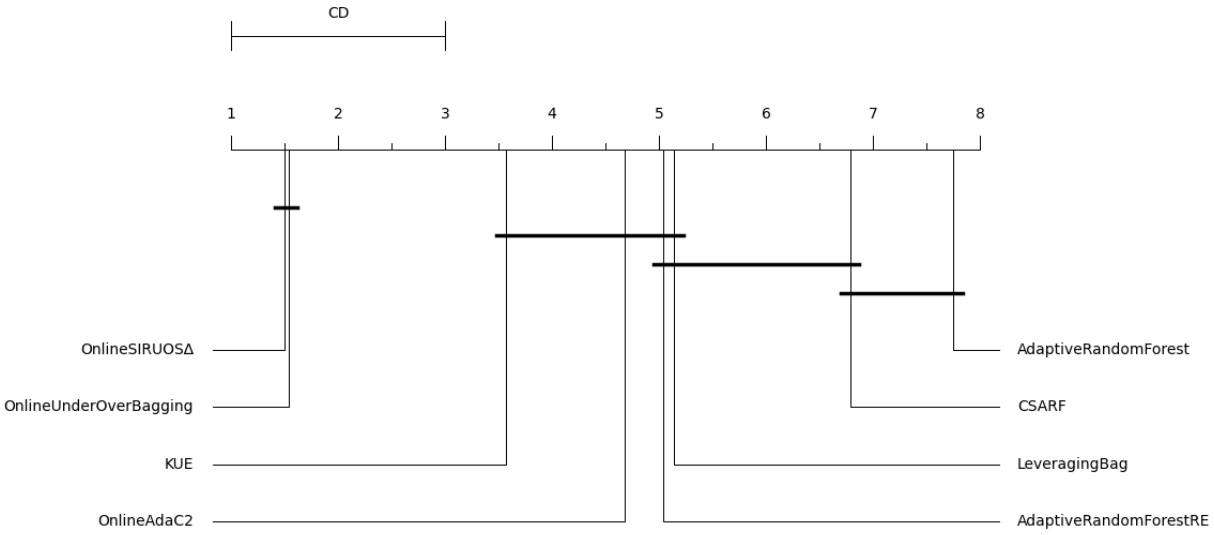

Figura 5.13: Gráfico de Nemenyi comparando algoritmos da literatura e os OnlineSIRUOS sobre cenários sintéticos desbalanceados baseados nos valores de Tempo de Processamento.

cada um dos algoritmos levou para processar os cenários sintéticos criados. Note que os algoritmos mais a esquerda são os que obtiveram os menores resultados, que nesse caso também significavam os melhores resultados. Aqui há evidência estatística suficiente que a variação OnlineSIRUOSΔ é mais rápida que a maior parte dos algoritmos avaliados, ficando empatada somente com o OnlineUOB. O KUE fica logo atrás na próxima linha de diferença crítica, empatando com outros algoritmos. Mas é notável a diferença de desempenho das técnicas da segunda linha comparadas às da primeira.

Por fim, na Figura 5.14 estão os dados de Custo de Modelo. Como na figura 5.13, os dados mais a esquerda são os menores valores obtidos. Aqui também se encontram

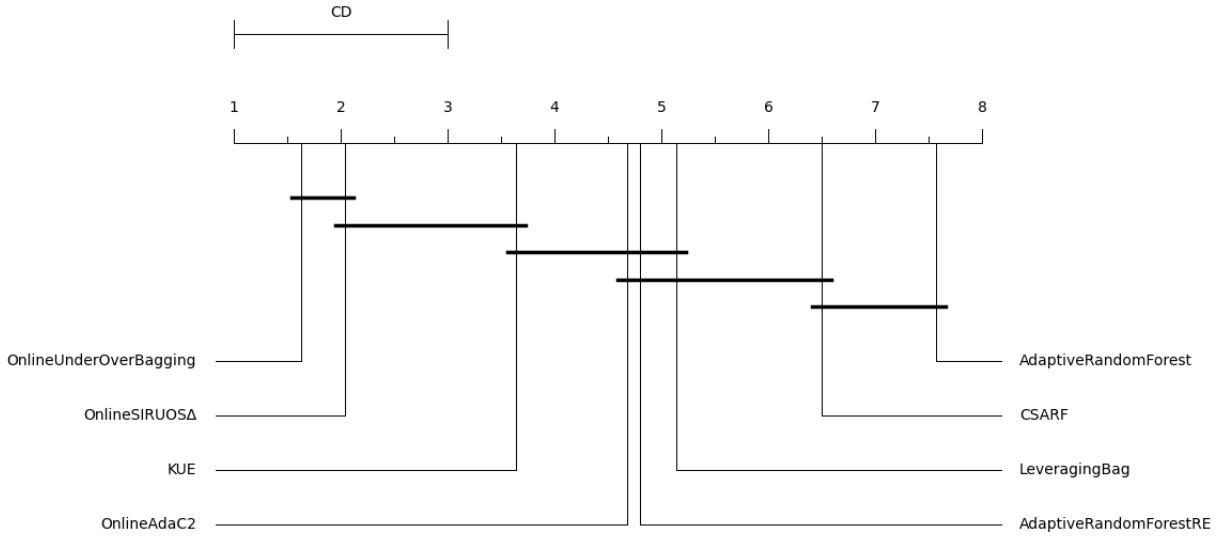

Figura 5.14: Gráfico de Nemenyi comparando algoritmos da literatura e os OnlineSIRUOS sobre cenários sintéticos desbalanceados baseados nos valores de Custo de Modelo.

resultados similares aos obtidos na figura anterior: OnlineSIRUOSΔ e OnlineUOB se isolando na liderança com evidência estatística suficiente e o KUE ficando um pouco mais atrás no quesito de consumo de memória. A diferença desse cenário foi que o KUE e o OnlineSIRUOSΔ também não tiveram diferença estatística suficiente em relação ao outro. Mas mesmo assim, a variação do OnlineSIRUOS tem vantagem atestada pelo próprio gráfico.

Olhando os dados dos experimentos sobre cenários sintéticos, foi possível observar que há resultados relevantes para o OnlineSIRUOSΔ. Não há diferença estatística suficiente que ateste um desempenho pior do que nenhum dos algoritmos da literatura, o que dá uma pista da possível relevância dessa nova técnica.

5.4.2 Avaliação dos Cenários Reais

A Figura 5.15 trás os valores de F1-score coletados dos experimentos com as bases reais utilizadas (Seção 5.1.3). A variação de OnlineSIRUOS não demonstra resultados tão altos quanto os vistos na Seção 5.4, mas ainda mantêm uma boa competência, já que estão dentro da linha de diferença crítica dos algoritmos com os melhores resultados. Ou seja, mesmo com o ARFRE claramente obtendo resultados graficamente melhores, não há comprovação estatística suficiente que denote que o OnlineSIRUOS tenha desempenho pior ou melhor.

Na Figura 5.16, os resultados apresentados são relacionados ao Tempo de Processamento. Assim como na Seção 5.4.1, os resultados mais a esquerda sempre serão os menores obtidos. O OnlineSIRUOS continua dentro da região de diferença crítica dos al-

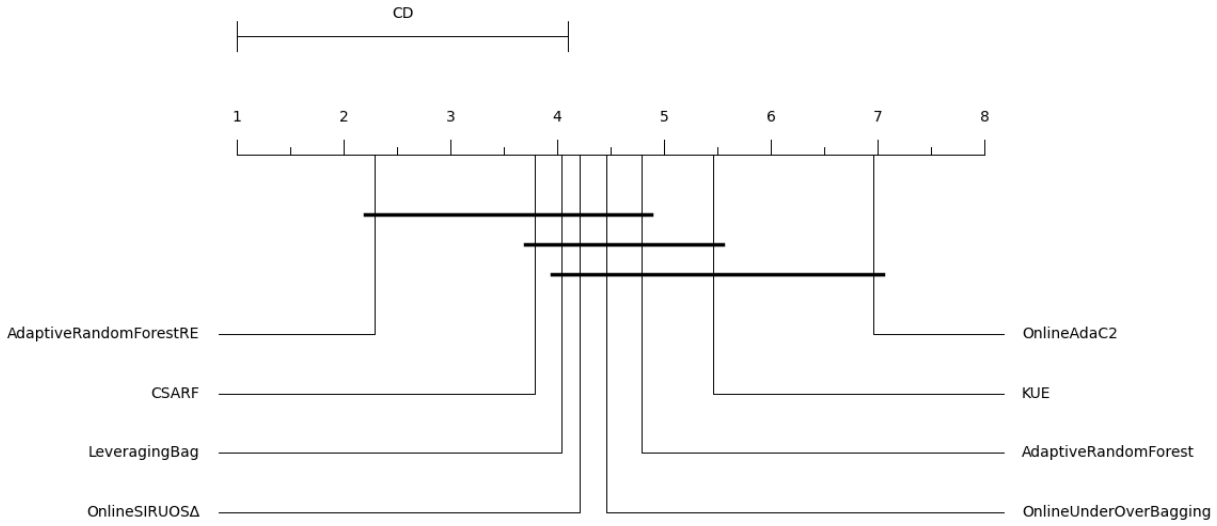

Figura 5.15: Gráfico de Nemenyi comparando algoritmos da literatura e os OnlineSIRUOS sobre cenários reais baseados nos valores de F1-score.

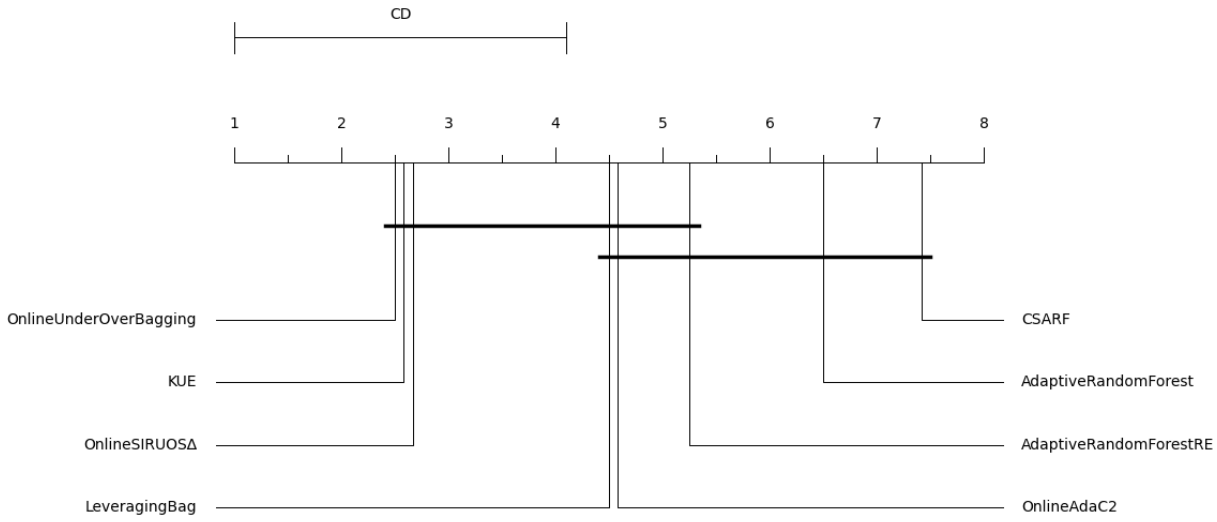

Figura 5.16: Gráfico de Nemenyi comparando algoritmos da literatura e os OnlineSIRUOS sobre cenários reais baseados nos valores de Tempo de Processamento.

goritmos com os melhores resultados, porém fica claro que o OnlineSIRUOSΔ fica muito próximo dos resultados do KUE e do OnlineUOB. As outras técnicas acabam ficando um pouco mais distantes.

Por fim, a Figura 5.17 contêm o dados relacionados ao Custo de Modelo e assim como na seção anterior, os valores mais a esquerda são os menores. Portanto, é possível observar que aqui também fica claro a competitividade do OnlineSIRUOS. Ele se mantêm dentro da região de diferença crítica dos melhores algoritmos, mesmo o OnlineUOB tendo uma leve vantagem nos resultados elencados no gráfico.

Aqui ficou claro que não foi possível atestar uma diferença estatística clara de desempenho do OnlineSIRUOS para todas as outras técnicas ao observar as métricas de

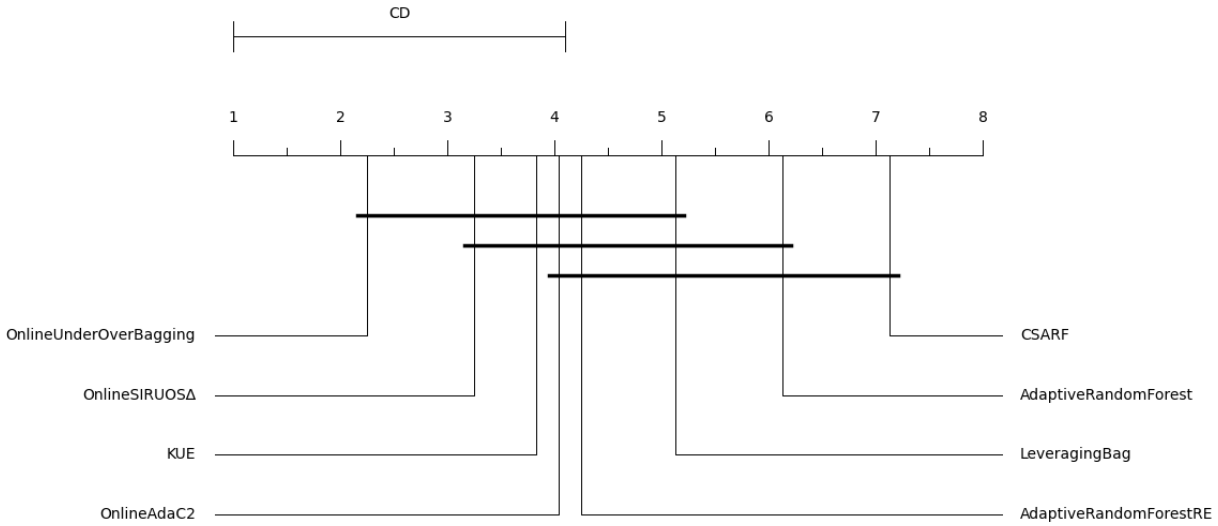

Figura 5.17: Gráfico de Nemenyi comparando algoritmos da literatura e os OnlineSIRUOS sobre cenários reais baseados nos valores de Custo de Modelo.

F1-score, Tempo de Processamento e Custo de Modelo, o que dá pistas da sua possível relevância em relação à literatura.

5.5 Gráficos de Custo Computacional

Para explorar em profundidade a questão do custo computacional, gráficos específicos foram criados para mostrar a média dos resultados combinados de ambientes específicos. Assim como o que foi feito na Seção 5.3, esta etapa da análise conta com avaliações isoladas para os cenários reais, cenários sintéticos gerais e cenários sintéticos com desbalanceamento de 95%-5%, 99%-1% e 99,5%-0,5% separadamente, para acompanhar quais são as mudanças que ocorrem conforme o processo de classificação se torna mais difícil. Ou seja, cada um desses casos contará com a média dos maiores valores de Tempo de Processamento e Custo de Modelo por cenário utilizado de cada conjunto de experimentos analisado. Os valores de Tempo de Processamento estão medidos em **segundos** e o Custo de Modelo em **Gigabytes por hora**.

5.5.1 Caso Geral dos Cenários Sintéticos

A Figura 5.18 mostra a média dos maiores valores de Tempo de Processamento e Custo de Modelo por experimento dentro de todos os cenários sintéticos utilizados.

É possível observar que o OnlineSIRUOS fica entre os 2 melhores valores obtidos para os cenários sintéticos, sendo que o OnlineSIRUOSΔ quase ultrapassa os melhores valores de Tempo de Processamento e Custo de Modelo obtidos pelo OnlineUOB, que por

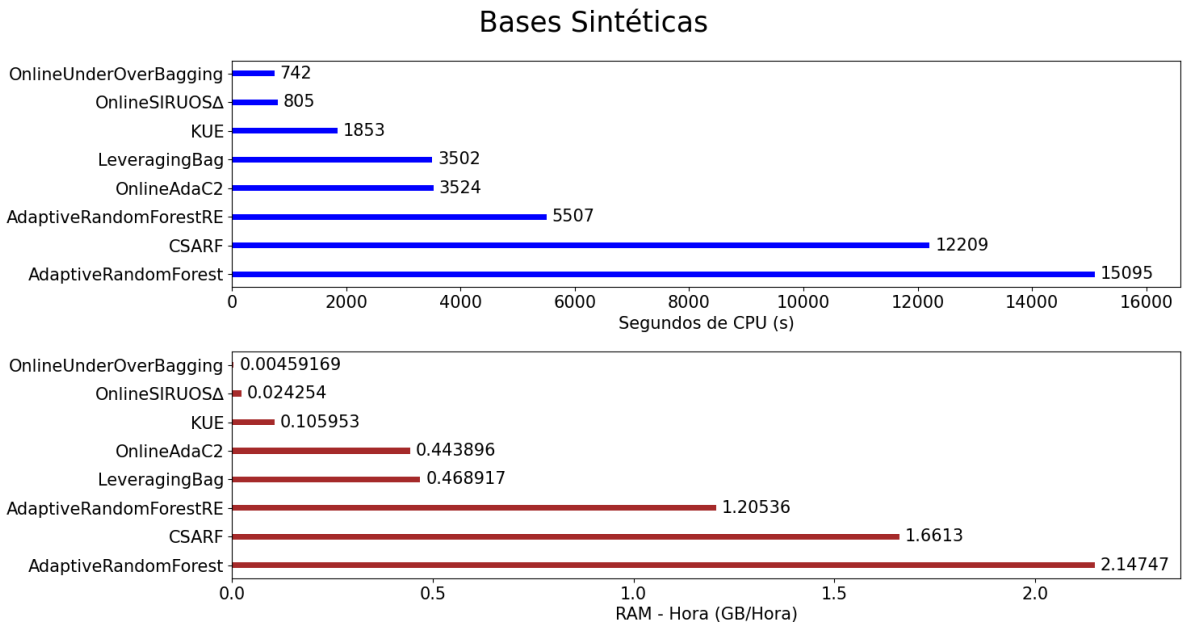

Figura 5.18: Gráfico comparando a média dos maiores resultados por experimento de Tempo de Processamento e Custo Computacional sobre todos os cenários sintéticos utilizados.

sua vez conquistou os melhores resultados dessas métricas.

Contudo, observar somente o caso geral dos experimentos sintéticos considera cenários em que o desbalanceamento não é tão alto, podendo causar uma visão incorreta sobre o desempenho de custo computacional dos algoritmos analisados. Por isso as Seções 5.5.1.1, 5.5.1.2 e 5.5.1.3 foram definidas para averiguar qual seria o comportamento dos algoritmos quando confrontados somente por experimentos de alto desbalanceamento.

5.5.1.1 Análise do Cenário Sintético 95% - 5%

A Figura 5.19 mostra a média dos maiores valores de Tempo de Processamento e Custo de Modelo por experimento dentro dos cenários sintéticos com desbalanceamento de 95% - 5% utilizados.

Nesse caso em específico, é notável que o desempenho do OnlineSIRUOS ficou melhor nas comparações do que no caso geral dos experimentos sintéticos da Seção 5.5.1. O OnlineSIRUOSΔ chegou a obter o melhor desempenho de todos em relação ao Tempo de Processamento, seguido pelo OnlineUOB e pelo KUE. Já em relação ao Custo de Modelo, o OnlineSIRUOSΔ ficou em 2º enquanto que o KUE ficou em 3º, ficando somente atrás do OnlineUOB na primeira posição.

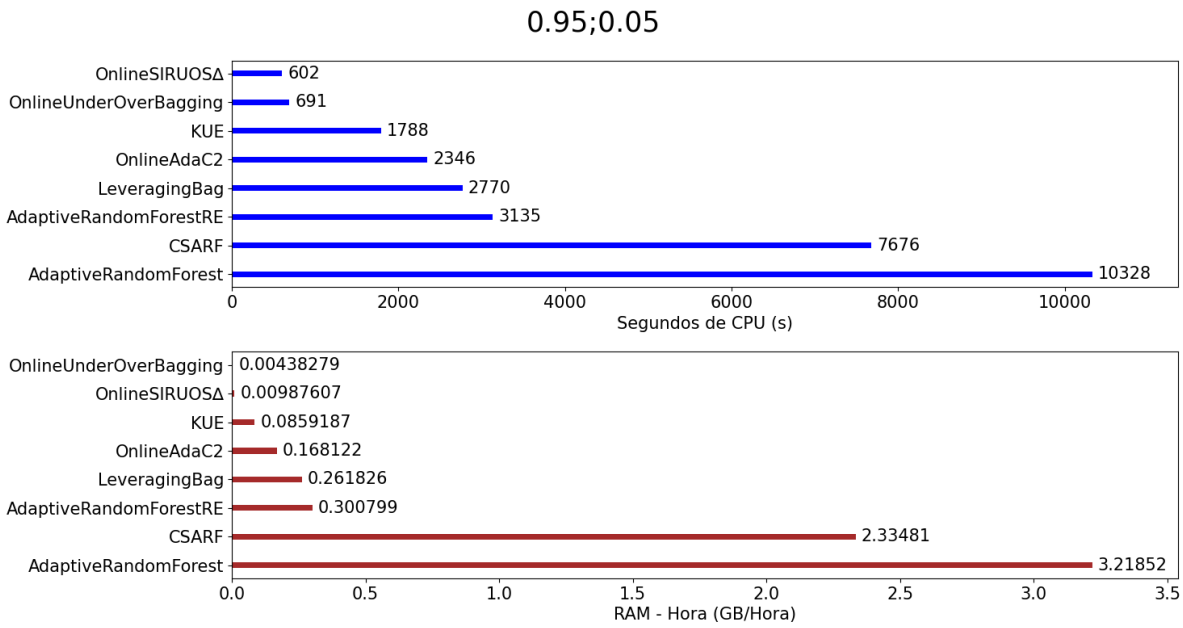

Figura 5.19: Gráfico comparando a média dos maiores resultados por experimento de Tempo de Processamento e Custo Computacional sobre os cenários sintéticos com desbalanceamento de 95% - 5%.

5.5.1.2 Análise do Cenário Sintético 99% - 1%

A Figura 5.20 mostra a média dos maiores valores de Tempo de Processamento e Custo de Modelo por experimento dentro dos cenários sintéticos com desbalanceamento de 99% - 1% utilizados.

Houveram muitas mudanças no desempenho da maior parte dos algoritmos quando comparados aos resultados da Seção 5.5.1.1. O OnlineSIRUOS ainda se mostra competitivo, mas houve uma melhora de desempenho muito grande no ARFRE nesse cenário que o fez despontar entre os melhores valores. Em relação ao Tempo de Processamento, o OnlineSIRUOSΔ continuou em 1º. Já em relação ao Custo de Modelo, o OnlineSIRUOSΔ passou para a 3ª posição.

5.5.1.3 Análise do Cenário Sintético 99,5% - 0,5%

A Figura 5.21 mostra a média dos maiores valores de Tempo de Processamento e Custo de Modelo por experimento dentro dos cenários sintéticos com desbalanceamento de 99,5% - 0,5% utilizados.

Com o aumento do desbalanceamento para casos mais extremos, algumas mudanças muito relevantes ocorreram. Primeiramente, em relação ao Tempo de Processamento, o OnlineSIRUOSΔ ainda continua liderando. O OnlineAdaC2 ficou em 4º lugar, logo atrás

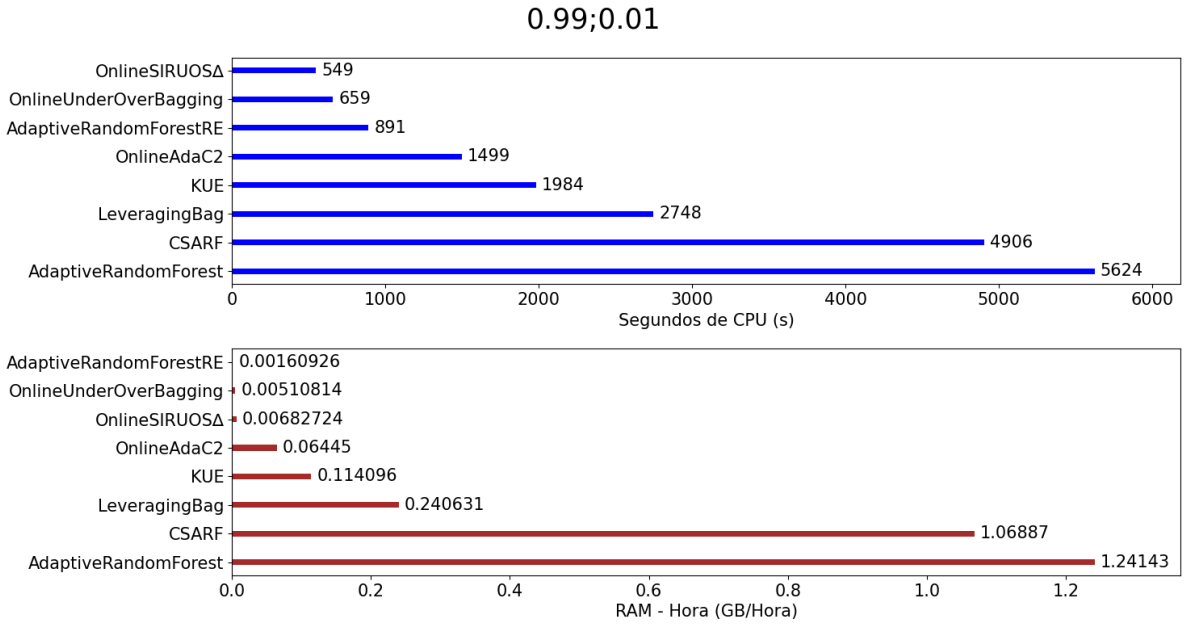

Figura 5.20: Gráfico comparando a média dos maiores resultados por experimento de Tempo de Processamento e Custo Computacional sobre os cenários sintéticos com desbalanceamento de 99% - 1%.

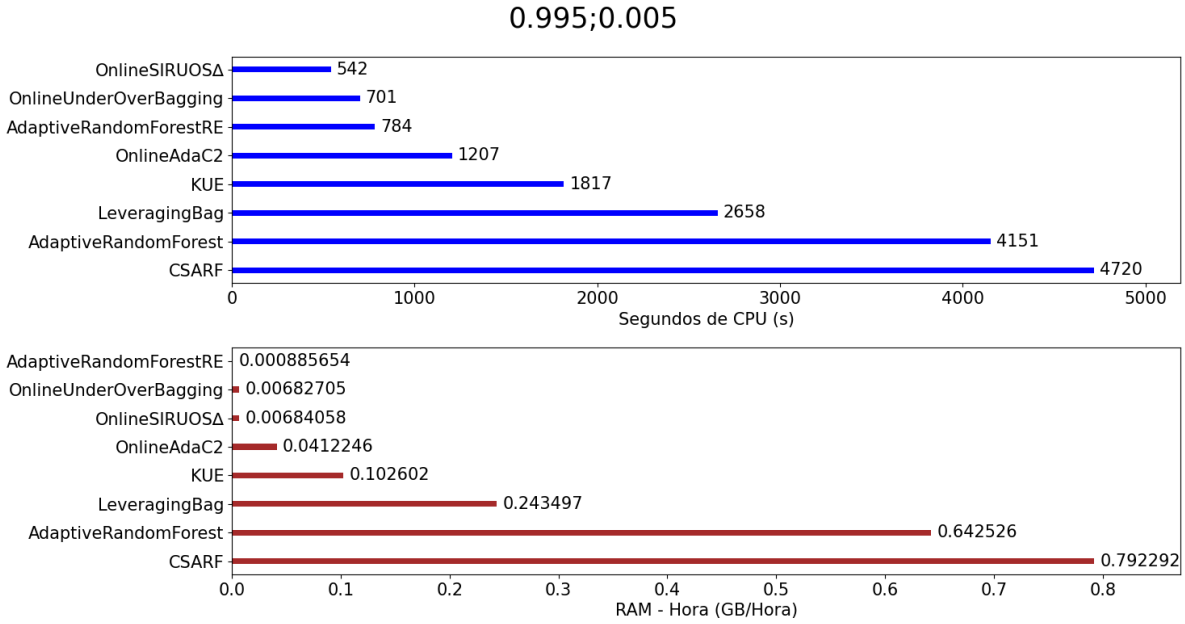

Figura 5.21: Gráfico comparando a média dos maiores resultados por experimento de Tempo de Processamento e Custo Computacional sobre os cenários sintéticos com desbalanceamento de 99,5% - 0,5%.

do ARFRE, que por sua vez está na sequência do OnlineUOB. Verificando os valores de Custo de Modelo, o OnlineSIRUOSΔ ficou em 3º, enquanto que o OnlineAdaC2 ficou em 4º. A diferença mais notória em relação aos resultados encontrados na Seção 5.5.1.2 está no fato de que o OnlineUOB ficou muito próximo dos valores do OnlineSIRUOSΔ, mas

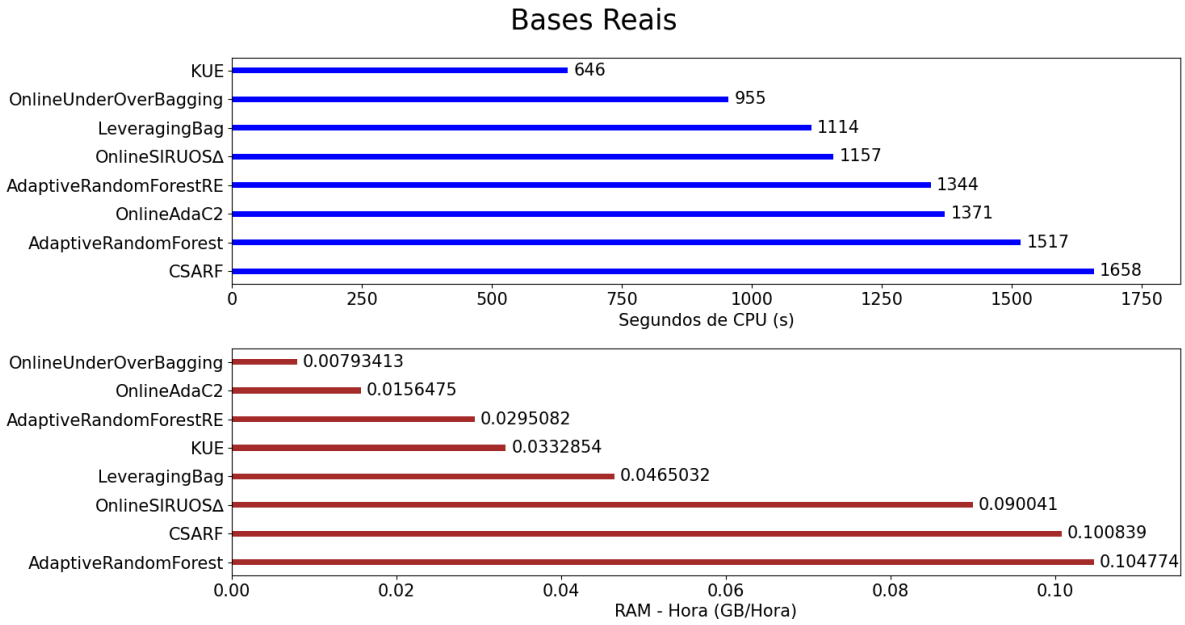

Figura 5.22: Gráfico comparando a média dos maiores resultados por experimento de Tempo de Processamento e Custo Computacional sobre os cenários reais.

ainda sim obteve um melhor desempenho.

5.5.2 Caso Geral dos Cenários Reais

A Figura 5.22 mostra a média dos maiores valores de Tempo de Processamento e Custo de Modelo por experimento dentro dos cenários reais.

Os experimentos com bases reais foram nitidamente mais custosos para o OnlineSIRUOS. Como apresentado nas Seções 5.19, 5.20 e 5.21, o algoritmo OnlineSIRUOS em sua variação selecionada aparentou ter um custo computacional menor em ambientes com desbalanceamento mais alto. Como a maior parte das bases reais utilizadas não possuem um desbalanceamento tão acentuado, é possível que isso tenha causado dificuldades para esse algoritmo em conjunto com a natureza dos dados, que é muito diferente dos dados sintéticos criados pelos geradores da Seção 5.1.2. Os Tempos de Processamento não ficaram muito longe do que os outros algoritmos também conseguiram, com exceção do KUE que desempenhou muito bem.

5.6 Gráficos de *Sign Tests*

Para as últimas etapas de análise, foram utilizados os testes de sinal, ou *Sign Tests*, o qual define um número mínimo de experimentos, dentro de um conjunto definido, que um algoritmo deve se sair melhor que o outro frente uma determinada métrica para se atestar

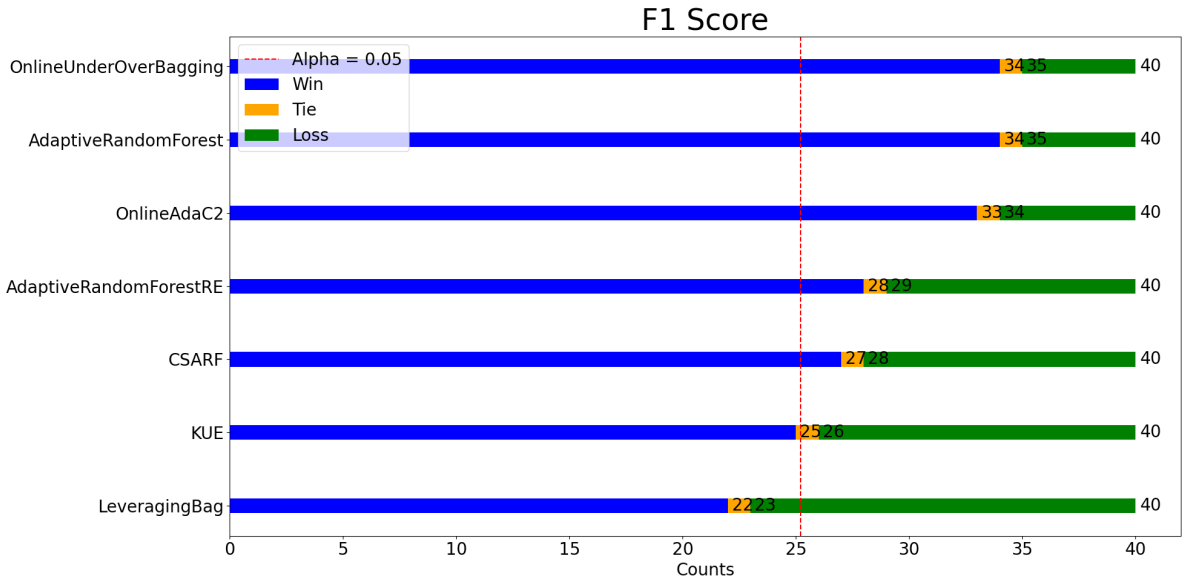

Figura 5.23: Gráfico de *Sign Test* comparando os resultados de F1-score do OnlineSIRUOSΔ com os algoritmos da literatura.

que há uma diferença significativa entre os algoritmos comparados. O parâmetro que define o tamanho da diferença significativa necessária para afirmar que há uma diferença entre os algoritmos foi definido como $\alpha = 5\%$ e todos os testes realizados foram incluídos no comparativo.

Nos gráficos são mostrados os número de vitórias e derrotas que determinado algoritmo teve em comparação com cada um dos outros. Também é possível ver o número de empates que ocorreram. Para os comparativos dessa seção, foi utilizado o OnlineSIRUOSΔ como base para as análises.

5.6.1 Comparativo com o OnlineSIRUOSΔ

A primeira Figura 5.23 mostra os resultados do teste de sinal para a métrica F1-score, usando o OnlineSIRUOSΔ como base. A segunda e terceira Figuras 5.24 e 5.25 seguem a mesma lógica, mas para os resultados de Tempo de Processamento e Custo de Modelo respectivamente.

Não era inesperado que o OnlineSIRUOSΔ não conseguisse um desempenho superior que todos os algoritmos avaliados em relação ao F1-score, visto que de acordo com a seleção da Seção 5.2 o mesmo foi selecionado devido ao seu bom equilíbrio entre custo computacional e classificações precisas. Mas mesmo possuindo resultados levemente inferiores, o OnlineSIRUOSΔ mostra superioridade em relação à maior parte dos algoritmos testados. Somente em relação ao KUE e o LeveragingBag que o OnlineSIRUOSΔ não

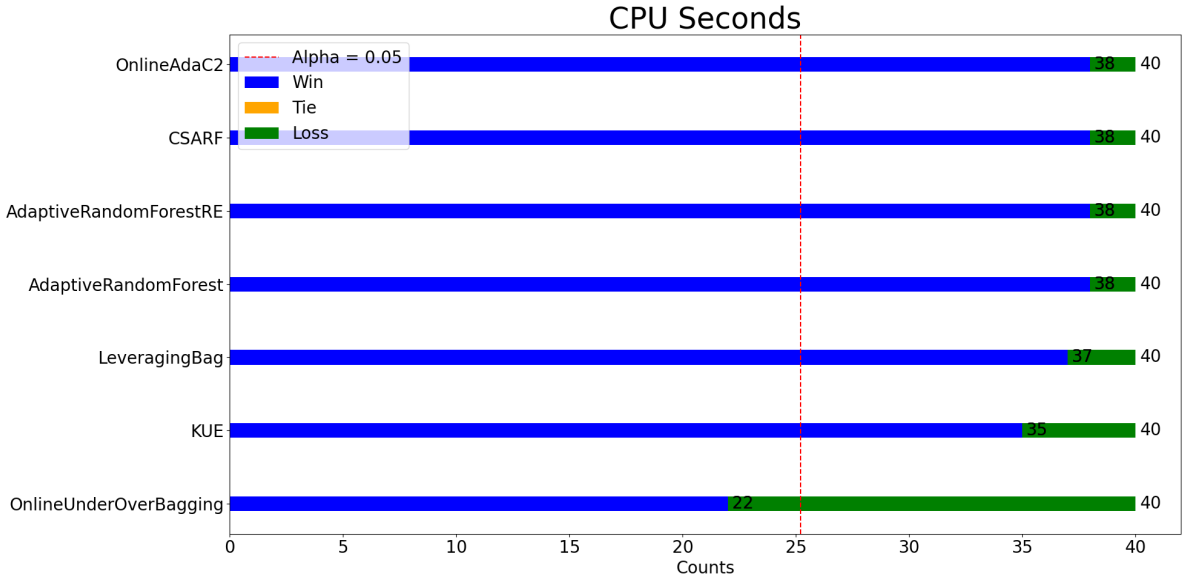

Figura 5.24: Gráfico de *Sign Test* comparando os resultados de Tempo de Processamento do OnlineSIRUOSΔ com os algoritmos da literatura.

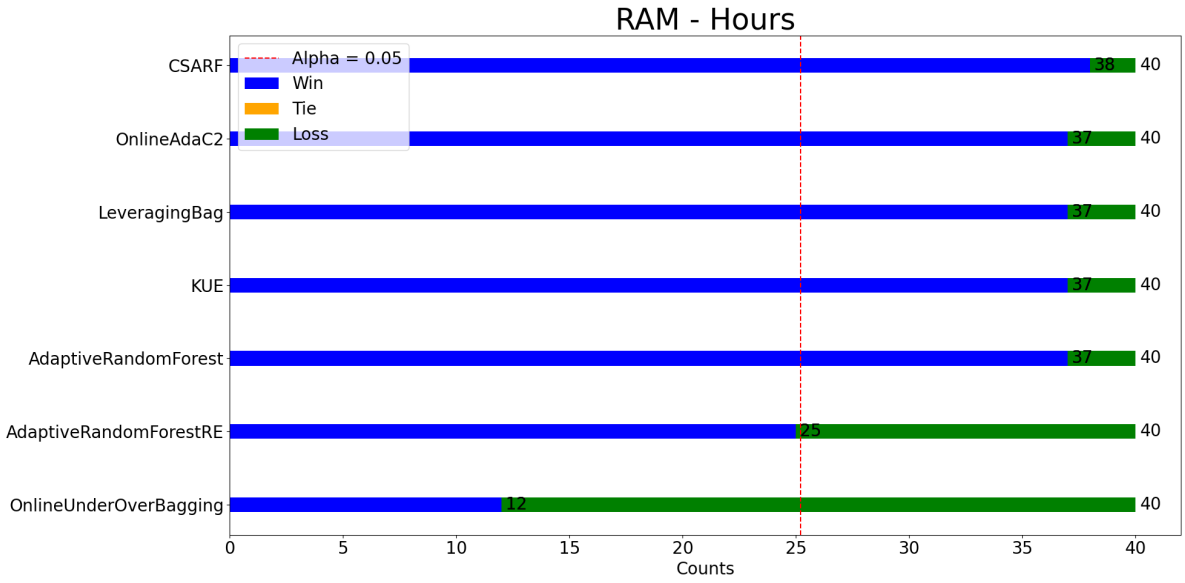

Figura 5.25: Gráfico de *Sign Test* comparando os resultados de Custo de Modelo do OnlineSIRUOSΔ com os algoritmos da literatura.

demonstrou superioridade. Entretanto, os resultados chegaram muito próximo disso.

Analisando a Figura 5.24 é possível observar que o OnlineSIRUOSΔ conseguiu um alto desempenho de Tempo de Processamento, ficando atrás somente do OnlineUOB. Isso também não era totalmente inesperado devido ao motivo da seleção do OnlineSIRUOSΔ (Seção 5.2), que visou também a escolha de uma opção com bom equilíbrio de uso computacional. Mas mesmo com essa premissa, os resultados obtidos foram interessantes pois demonstram a capacidade de rápido processamento que o OnlineSIRUOS possui.

Por fim, ao analisar a Figura 5.25, fica nítida a eficiência em relação a memória que o OnlineSIRUOSΔ consegue quando comparado com os outros algoritmos avaliados. Pelo mesmo motivo anteriormente citado, relacionado ao porque essa variação foi selecionada, não era inesperado que ela tenha obtido valores animadores. Mesmo assim, impressiona o fato de que o OnlineSIRUOSΔ somente não conseguiu se destacar frente ao ARFRE e o OnlineUOB.

5.7 Considerações Finais

Esse capítulo apresentou os resultados das diferentes técnicas dos Capítulos 3 e 4 que foram selecionadas para serem avaliadas sobre cenários com níveis de desbalanceamento variados, sendo eles reais ou sintéticos. O número e os tipos de experimentos realizados trazem robustez para os resultados. Além disso, uma discussão sobre os resultados foi realizada, apresentando pontos importantes que devem ser levados em consideração para a proposta do presente documento e para a conclusão desse projeto.

Em termos gerais, o OnlineSIRUOS se mostrou competitivo frente aos algoritmos da literatura. Como técnicas de *data stream mining* não devem ser avaliadas somente pela perspectiva de acurácia e precisão de predição, mas também através do seu custo computacional, foi possível mostrar uma outra perspectiva dos algoritmos avaliados, mostrando que muitos deles conseguem ser competentes em ambas as medidas. Da perspectiva das métricas escolhidas para análise na Seção 5.1.5, a variação do OnlineSIRUOS conseguiu bons resultados, se mantendo em igualdade ou até superando muitos dos algoritmos avaliados. Isso particularmente ocorreu com mais enfase em cenários com o desbalanceamento mais acentuado, onde foi possível perceber um decréscimo do custo computacional e uma constância de resultados de F1-score superiores que a maior parte das técnicas. Um ponto de extrema importância foi o fato de que a variação OnlineSIRUOSΔ, escolhida na Seção 5.2 para ser a variação da Tabela 5.6 que melhor atende o equilíbrio entre F1-score e custo computacional, foi avaliada como capaz de conseguir se manter competitiva com outros algoritmos no aspecto de precisão de classificação e ainda assim consumir menos recursos computacionais que vários deles. E pelo fato da técnica selecionada para análise com os algoritmos da literatura utilizarem *stacking*, isso só demonstra o benéfico do uso da meta-aprendizagem. Como complemento a esse argumento, é possível ver que há muitas variações também usuárias dessa estratégia que ficaram logo atrás da variação escolhida para análise, trazendo consigo os seus pontos portes e fracos. Obviamente não se pode dizer que essa técnica é a razão final do bom desempenho do OnlineSIRUOS, visto que o real motivo do seu sucesso é o uso conjunto de *ensembles* heterogêneos, *inverse random under and over sampling* e o próprio meta-aprendizado. Contudo, ainda assim é claro que há benefícios em usar essa estratégia. Além disso, foi possível avaliar brevemente o OnlineSIRUOS sobre os resultados das métricas de Precision e Recall, trazendo uma visão diferente do algoritmo e provando a sua capacidade por meio de outras perspectivas.

No Capítulo 6 são apresentadas as conclusões finais deste estudo, incluindo interpretações sobre os resultados obtidos, possíveis trabalhos futuros, pontos de melhora, entre outros.

Capítulo 6

Conclusões

O desbalanceamento é um desafio relevante para a mineração de fluxos de dados. Existem diversas lacunas a serem preenchidas devido ao comportamento variado e complexo de fluxos de dados quando há quantidades desiguais de exemplos de diferentes classes. Apesar de existirem algoritmos criados especificamente para tratar este problema, ainda há espaço para a proposta de novos métodos e até mesmo a combinação de estratégias existentes. Seguindo este raciocínio, o presente estudo propõe uma adaptação para ambientes online inspirada no algoritmo SIRUS, conhecida como OnlineSIRUOS, que conta com o uso de aprendizagem baseada em comitês (*ensembles*), uma variação e adaptação do processo de amostragem, i.e., *inverse random undersampling* para fluxos contínuos de dados e meta-aprendizagem.

A experimentação contemplou cenários sintéticos e reais para avaliação algoritmo OnlineSIRUOS e algoritmos do estado da arte. Sumarizando os resultados de F1-score, Tempo de Processamento e Custo de Modelo, foi possível observar que o OnlineSIRUOS obteve resultados competitivos com os algoritmos da literatura. Tendo em vista os resultados disponíveis nas Seções 5.3, 5.3.1, 5.5 e 5.6, foi possível observar diferentes perspectivas do algoritmo proposto, incluindo análises extras para verificar que o mesmo é eficiente não só sobre o foco principal de avaliação desse projeto. Considerando os testes estatísticos de Friedman, Nemenyi e os testes de sinal (*Sign Tests*), fica atestada a competitividade da variação escolhida. Combinando a análise dos gráficos de radar com os de custo computacional, é notado que alguns algoritmos como o KUE e o ARFRE fazem frente ao OnlineSIRUOS. Entretanto, ainda assim há um equilíbrio de desempenho, pois em alguns casos, o algoritmo aqui proposto tem vantagem na perspectiva de precisão de classificação e em outros em relação ao custo computacional, o que também demonstra a relevância do OnlineSIRUOS. Mesmo ao isolar a análise sobre a métrica de classificação de F1-score, é possível observar que na média, o OnlineSIRUOS conseguiu vantagens de cerca de 2,70%

sobre outros algoritmos quando esse método estava na primeira posição de resultados. E quando não estavam, na média, a diferença não passou dos 2% do melhor resultado. Fora isso, nas métricas de custo computacional, a variação de OnlineSIRUOS foi muito competitiva, particularmente porque conseguiu agregar um resultado de F1-score satisfatório com um custo computacional menor do que outros algoritmos que elencavam os melhores resultados de F1-score. Finalmente, dentre os resultados obtidos e analisados, observamos que a técnica criada nesse projeto possui resultados relevantes em casos desbalanceados acima de 95% - 5%.

6.1 Trabalhos Futuros

Apesar dos resultados obtidos, o vislumbra-se os seguintes tópicos para análises futuras.

- A experimentação realizada carece de análises mais profundas em suas componentes e parametrizações, dado que mesmo contando com muitas variações sendo utilizadas (Tabela 5.6) elas pouco exploram todo o potencial de customização do algoritmo. Além disso, caso uma análise como essa seja feita e como dito anteriormente neste documento, uma exploração dos parâmetros ótimos para todos os algoritmos utilizados também deve ser feita para que uma comparação sobre níveis justos possa ser realizada.

- Os resultados obtidos nos testes com bases reais foram claramente piores para todas as técnicas, provavelmente devido a natureza que os dados possuem em casos reais, gerando uma maior dificuldade de classificação. Por isso, fica a dúvida de como seria o desempenho do OnlineSIRUOS e de outras técnicas testadas em mais bases reais com mais desbalanceamento e de diferentes naturezas?

- Quão impactante são às técnicas de meta-aprendizado e *inverse random under and over sampling* isoladamente? Naturalmente, *stacking* induz custo computacional mais elevado e o *inverse random under and over sampling* não é facilmente adaptado a todos os algoritmos. Contudo, trazer resultados isolados de cada técnica sendo aplicada em outros algoritmos pode levantar respostas sobre o impacto que cada uma pode causar sozinha.

- É necessária a utilização de experimentos, sintéticos e reais, que simulem ambientes em que o desbalanceamento seja dinâmico. Por exemplo, um cenário em que a taxa de desbalanceamento inicie em 50% - 50%, vá até 90% - 10% e depois retorne ao

estado inicial. Isso deve ser feito para averiguar a capacidade de adaptação dos algoritmos em ambientes mais passíveis de mudanças.

- Uso de paralelismo no funcionamento interno do OnlineSIRUOS, visando diminuir ainda mais o tempo de processamento das operações do mesmo, pode realmente trazer benefícios? E quão impactante eles seriam?

Referências Bibliográficas

AGGARWAL, C. C.; HAN, J.; WANG, J.; YU, P. S. A framework for clustering evolving data streams. In: *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*. [S.l.]: VLDB Endowment, 2003. (VLDB '03), p. 81–92. ISBN 0127224424.

AGRAWAL, R.; IMIELINSKI, T.; SWAMI, A. Database mining: a performance perspective. *IEEE Transactions on Knowledge and Data Engineering*, v. 5, n. 6, p. 914–925, Dec 1993. ISSN 1558-2191.

ARAÚJO, V. M.; BRITTO, A. S.; BRUN, A. L.; KOERICH, A. L.; OLIVEIRA, L. E. S. Fine-grained hierarchical classification of plant leaf images using fusion of deep models. In: *2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI)*. [S.l.: s.n.], 2018. p. 1–5. ISSN 1082-3409.

BARDDAL, J. P.; GOMES, H. M.; ENEMBRECK, F.; PFAHRINGER, B. A survey on feature drift adaptation: Definition, benchmark, challenges and future directions. *Journal of Systems and Software*, v. 127, p. 278–294, 2017. ISSN 0164-1212. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0164121216301030>.

BARDDAL, J. P.; LOEZER, L.; ENEMBRECK, F.; LANZUOLO, R. Lessons learned from data stream classification applied to credit scoring. *Expert Systems with Applications*, v. 162, p. 113899, 2020. ISSN 0957-4174. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0957417420306928>.

BIFET, A.; GAVALDA, R. Learning from time-changing data with adaptive windowing. In: *Proceedings of the Seventh SIAM International Conference on Data Mining, April 26-28, 2007, Minneapolis, Minnesota, USA*. SIAM, 2007. p. 443–448. Disponível em: <https://doi.org/10.1137/1.9781611972771.42>.

BIFET, A.; GAVALDA, R. Adaptive learning from evolving data streams. In: *Proceedings of the 8th International Symposium on Intelligent Data Analysis: Advances in Intelligent Data Analysis VIII*. Berlin, Heidelberg: Springer-Verlag, 2009. (IDA '09), p. 249–260.

ISBN 9783642039140. Disponível em: <https://doi.org/10.1007/978-3-642-03915-7_22>.

BIFET, A.; GAVALDÀ, R.; HOLMES, G.; PFAHRINGER, B. *Machine Learning for Data Streams with Practical Examples in MOA*. [S.l.]: MIT Press, 2018. <https://moa.cms.waikato.ac.nz/book/>.

BIFET, A.; HOLMES, G.; KIRKBY, R.; PFAHRINGER, B. MOA: massive online analysis. *J. Mach. Learn. Res.*, v. 11, p. 1601–1604, 2010. Disponível em: <http://portal.acm.org/citation.cfm?id=1859903>.

BIFET, A.; HOLMES, G.; PFAHRINGER, B. Leveraging bagging for evolving data streams. In: BALCÁZAR, J. L.; BONCHI, F.; GIONIS, A.; SEBAG, M. (Ed.). *Machine Learning and Knowledge Discovery in Databases*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 135–150. ISBN 978-3-642-15880-3.

BIFET, A.; HOLMES, G.; PFAHRINGER, B.; KIRKBY, R.; GAVALDA, R. New ensemble methods for evolving data streams. In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: Association for Computing Machinery, 2009. (KDD '09), p. 139–148. ISBN 9781605584959. Disponível em: <https://doi.org/10.1145/1557019.1557041>.

BIFET, A.; KIRKBY, R. Data stream mining a practical approach. In: . [S.l.: s.n.], 2009.

BIFET, A.; PFAHRINGER, B.; READ, J.; HOLMES, G. Efficient data stream classification via probabilistic adaptive windows. In: *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. New York, NY, USA: Association for Computing Machinery, 2013. (SAC '13), p. 801–806. ISBN 9781450316569. Disponível em: <https://doi.org/10.1145/2480362.2480516>.

BIFET, A.; READ, J.; ŽLIOBAITĖ, I.; PFAHRINGER, B.; HOLMES, G. Pitfalls in benchmarking data stream classification and how to avoid them. In: BLOCKEEL, H.; KERSTING, K.; NIJSSEN, S.; ŽELEZNÝ, F. (Ed.). *Machine Learning and Knowledge Discovery in Databases*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 465–479. ISBN 978-3-642-40988-2.

BISHOP, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN 0387310738.

BODIK, P.; THIBAUX, R.; PASKIN, M.; MADDEN, S.; GUESTRIN, C.; HONG, W. Intel Research Berkeley Lab, 2004. Disponível em: <http://db.csail.mit.edu/labdata/labdata.html>.

BREIMAN, L. *Bias, variance, and arcing classifiers*. [S.l.], 1996.

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

BRZEZIŃSKI, D.; STEFANOWSKI, J. Accuracy updated ensemble for data streams with concept drift. In: CORCHADO, E.; KURZYŃSKI, M.; WOŹNIAK, M. (Ed.). *Hybrid Artificial Intelligent Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. p. 155–163. ISBN 978-3-642-21222-2.

BRZEZINSKI, D.; STEFANOWSKI, J.; SUSMAGA, R.; SZCZECH, I. On the dynamics of classification measures for imbalanced and streaming data. *IEEE Transactions on Neural Networks and Learning Systems*, v. 31, n. 8, p. 2868–2878, 2020.

CAL, P.; WOŹNIAK, M. Parallel hoeffding decision tree for streaming data. In: OMATU, S.; NEVES, J.; RODRIGUEZ, J. M. C.; SANTANA, J. F. P.; GONZALEZ, S. R. (Ed.). *Distributed Computing and Artificial Intelligence*. Cham: Springer International Publishing, 2013. p. 27–35. ISBN 978-3-319-00551-5.

CANO, A.; KRAWCZYK, B. Kappa updated ensemble for drifting data stream mining. *Machine Learning*, v. 109, n. 1, p. 175–218, Jan 2020. ISSN 1573-0565. Disponível em: <https://doi.org/10.1007/s10994-019-05840-z>.

CAO, F.; ESTER, M.; QIAN, W.; ZHOU, A. Density-based clustering over an evolving data stream with noise. In: *SDM*. [S.l.: s.n.], 2006.

CHAWLA, N. V.; BOWYER, K. W.; HALL, L. O.; KEGELMEYER, W. P. Smote: Synthetic minority over-sampling technique. *J. Artif. Int. Res.*, AI Access Foundation, El Segundo, CA, USA, v. 16, n. 1, p. 321–357, jun. 2002. ISSN 1076-9757.

CHEN, B.; XIA, S.; CHEN, Z.; WANG, B.; WANG, G. Rsmote: A self-adaptive robust smote for imbalanced problems with label noise. *Information Sciences*, 2020. ISSN 0020-0255. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0020025520310045>.

DEMŠAR, J. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, v. 7, n. 1, p. 1–30, 2006. Disponível em: <http://jmlr.org/papers/v7/demsar06a.html>.

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

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

DUONG, Q.-H.; RAMAMPIARO, H.; NORVRAG, K. Applying temporal dependence to detect changes in streaming data. *Applied Intelligence*, Kluwer Academic Publishers, USA, v. 48, n. 12, p. 4805–4823, dez. 2018. ISSN 0924-669X. Disponível em: <https://doi.org/10.1007/s10489-018-1254-7>.

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

ENEMBRECK, F.; ÁVILA, B. C.; SCALABRIN, E. E.; BARTHÈS, J.-P. Learning drifting negotiations. *Appl. Artif. Intell.*, Taylor & Francis, Inc., USA, v. 21, n. 9, p. 861–881, out. 2007. ISSN 0883-9514. Disponível em: <https://doi.org/10.1080/08839510701526954>.

FERNÁNDEZ, A.; GARCÍA, S.; GALAR, M.; PRATI, R. C.; KRAWCZYK, B.; HERRERA, F. Foundations on imbalanced classification. In: _____. *Learning from Imbalanced Data Sets*. Cham: Springer International Publishing, 2018. p. 19–46. ISBN 978-3-319-98074-4. Disponível em: <https://doi.org/10.1007/978-3-319-98074-4_2>.

FERREIRA, L. E. B.; GOMES, H. M.; BIFET, A.; OLIVEIRA, L. S. Adaptive random forests with resampling for imbalanced data streams. In: *2019 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2019. p. 1–6. ISSN 2161-4407.

FRÍAS-BLANCO, I.; CAMPO-ÁVILA, J. D.; RAMOS-JIMÉNEZ, G.; MORALES-BUENO, R.; ORTIZ-DÍAZ, A.; CABALLERO-MOTA, Y. Online and non-parametric drift detection methods based on hoeffding's bounds. *IEEE Transactions on Knowledge and Data Engineering*, v. 27, n. 3, p. 810–823, March 2015. ISSN 1558-2191.

GAMA, J.; SEBASTIAO, R.; RODRIGUES, P. On evaluating stream learning algorithms. *Machine Learning*, v. 90, p. 317–346, 10 2013.

GAMA, J.; SEBASTIAO, R.; RODRIGUES, P. P. Issues in evaluation of stream learning algorithms. In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: Association for Computing Machinery, 2009. (KDD '09), p. 329–338. ISBN 9781605584959. Disponível em: <https://doi.org/10.1145/1557019.1557060>.

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

GOMES, H. M.; BIFET, A.; READ, J.; BARDDAL, J. P.; ENEMBRECK, F.; PFHARINGER, B.; HOLMES, G.; ABDESSALEM, T. Adaptive random forests for evolving data stream classification. *Mach. Learn.*, Kluwer Academic Publishers, USA, v. 106, n. 9–10, p. 1469–1495, out. 2017. ISSN 0885-6125. Disponível em: <https://doi.org/10.1007/s10994-017-5642-8>.

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

GULOWATY, B.; KSIENIEWICZ, P. Smote algorithm variations in balancing data streams. In: YIN, H.; CAMACHO, D.; TINO, P.; TALLÓN-BALLESTEROS, A. J.; MENEZES, R.; ALLMENDINGER, R. (Ed.). *Intelligent Data Engineering and Automated Learning – IDEAL 2019*. Cham: Springer International Publishing, 2019. p. 305–312. ISBN 978-3-030-33617-2.

HAN, J.; KAMBER, M.; PEI, J. 3 - data preprocessing. In: HAN, J.; KAMBER, M.; PEI, J. (Ed.). *Data Mining (Third Edition)*. Third edition. Boston: Morgan Kaufmann, 2012, (The Morgan Kaufmann Series in Data Management Systems). p. 83–124. ISBN 978-0-12-381479-1. Disponível em: <https://www.sciencedirect.com/science/article/pii/B9780123814791000034>.

HO, T. K. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 20, n. 8, p. 832–844, Aug 1998. ISSN 1939-3539.

JAECH, A.; ZHANG, B.; OSTENDORF, M.; KIRSCHEN, D. S. Real-time prediction of the duration of distribution system outages. *IEEE Transactions on Power Systems*, v. 34, n. 1, p. 773–781, Jan 2019. ISSN 1558-0679.

JAPKOWICZ, N.; SHAH, M. *Evaluating Learning Algorithms: A Classification Perspective*. USA: Cambridge University Press, 2014. ISBN 1107653118.

KOSINA, P.; GAMA, J. Very fast decision rules for multi-class problems. In: *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. New York, NY, USA: Association for Computing Machinery, 2012. (SAC '12), p. 795–800. ISBN 9781450308571. Disponível em: <https://doi.org/10.1145/2245276.2245431>.

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

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

KUNCHEVA, L. I.; RODRÍGUEZ, J. J. A weighted voting framework for classifiers ensembles. *Knowledge and Information Systems*, v. 38, n. 2, p. 259–275, Feb 2014. ISSN 0219-3116. Disponível em: <https://doi.org/10.1007/s10115-012-0586-6>.

LIANG, X.; JIANG, A.; LI, T.; XUE, Y.; WANG, G. Lr-smote — an improved unbalanced data set oversampling based on k-means and svm. *Knowledge-Based Systems*, v. 196, p. 105845, 2020. ISSN 0950-7051. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0950705120302148>.

LIU, X.-Y.; WU, J.; ZHOU, Z.-H. Exploratory undersampling for class-imbalance learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, v. 39, n. 2, p. 539–550, 2009.

LOEZER, L.; ENEMBRECK, F.; BARDDAL, J. P.; BRITTO, A. de S. Cost-sensitive learning for imbalanced data streams. In: *Proceedings of the 35th Annual ACM Symposium on Applied Computing*. New York, NY, USA: Association for Computing Machinery, 2020.

(SAC '20), p. 498–504. ISBN 9781450368667. Disponível em: <https://doi.org/10.1145/3341105.3373949>.

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

MORO, S.; CORTEZ, P.; RITA, P. A data-driven approach to predict the success of bank telemarketing. *Decision Support Systems*, v. 62, p. 22–31, 2014. ISSN 0167-9236. Disponível em: <https://www.sciencedirect.com/science/article/pii/S016792361400061X>.

MORO, S.; LAUREANO, R.; CORTEZ, P. Using data mining for bank direct marketing: An application of the crisp-dm methodology. In: AL., P. N. et (Ed.). *Proceedings of the European Simulation and Modelling Conference - ESM'2011*. Guimaraes, Portugal: EUROSIS, 2011. p. 117–121.

NGUYEN, H. M.; COOPER, E. W.; KAMEI, K. A comparative study on sampling techniques for handling class imbalance in streaming data. In: *The 6th International Conference on Soft Computing and Intelligent Systems, and The 13th International Symposium on Advanced Intelligence Systems*. [S.l.: s.n.], 2012. p. 1762–1767.

OPITZ, D.; MACLIN, R. Popular ensemble methods: An empirical study. *J. Artif. Int. Res.*, AI Access Foundation, El Segundo, CA, USA, v. 11, n. 1, p. 169–198, jul. 1999. ISSN 1076-9757.

ORRITE, C.; RODRÍGUEZ, M.; MARTÍNEZ, F.; FAIRHURST, M. Classifier ensemble generation for the majority vote rule. In: RUIZ-SHULCLOPER, J.; KROPATSCH, W. G. (Ed.). *Progress in Pattern Recognition, Image Analysis and Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. p. 340–347. ISBN 978-3-540-85920-8.

OZA, N. C. Online bagging and boosting. In: *2005 IEEE International Conference on Systems, Man and Cybernetics*. [S.l.: s.n.], 2005. v. 3, p. 2340–2345 Vol. 3.

ROSS, G. J.; ADAMS, N. M.; TASOULIS, D. K.; HAND, D. J. Exponentially weighted moving average charts for detecting concept drift. *CoRR*, abs/1212.6018, 2012. Disponível em: <http://arxiv.org/abs/1212.6018>.

SAITO, T.; REHMSMEIER, M. The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. *PLOS ONE*, Public

Library of Science (PLoS), v. 10, n. 3, p. e0118432, mar. 2015. Disponível em: <https://doi.org/10.1371/journal.pone.0118432>.

SCHLIMMER, J. C.; GRANGER, R. H. Incremental learning from noisy data. *Mach. Learn.*, Kluwer Academic Publishers, USA, v. 1, n. 3, p. 317–354, mar. 1986. ISSN 0885-6125. Disponível em: <https://doi.org/10.1023/A:1022810614389>.

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

SUN, Y. *Cost-Sensitive Boosting for Classification of Imbalanced Data*. Tese (Doutorado), CAN, 2007. AAINR34548.

TAHIR, M. A.; KITTLER, J.; YAN, F. Inverse random under sampling for class imbalance problem and its application to multi-label classification. *Pattern Recognition*, v. 45, n. 10, p. 3738–3750, 2012. ISSN 0031-3203. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0031320312001471>.

TING, K. M. A comparative study of cost-sensitive boosting algorithms. In: *Proceedings of the Seventeenth International Conference on Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000. (ICML '00), p. 983–990. ISBN 1558607072.

TOMEK, I. Two modifications of cnn. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-6, n. 11, p. 769–772, 1976.

TSYMBAL, A. The problem of concept drift: definitions and related work. In: . [S.l.: s.n.], 2004.

VELOSO, B.; GAMA, J.; MALHEIRO, B. Self hyper-parameter tuning for data streams. In: SOLDATOVA, L.; VANSCHOREN, J.; PAPADOPOULOS, G.; CECI, M. (Ed.). *Discovery Science*. Cham: Springer International Publishing, 2018. p. 241–255. ISBN 978-3-030-01771-2.

WANG, B.; PINEAU, J. Online bagging and boosting for imbalanced data streams. *IEEE Transactions on Knowledge and Data Engineering*, v. 28, n. 12, p. 3353–3366, 2016.

WANG, S.; YAO, X. Diversity analysis on imbalanced data sets by using ensemble models. In: *2009 IEEE Symposium on Computational Intelligence and Data Mining*. [S.l.: s.n.], 2009. p. 324–331.

WOLPERT, D. H. Stacked generalization. *Neural Networks*, v. 5, n. 2, p. 241–259, 1992. ISSN 0893-6080. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0893608005800231>.

XU, Y. Maximum margin of twin spheres support vector machine for imbalanced data classification. *IEEE Transactions on Cybernetics*, v. 47, n. 6, p. 1540–1550, June 2017. ISSN 2168-2275.

ZHANG, Y.; LIU, G.; LUAN, W.; YAN, C.; JIANG, C. An approach to class imbalance problem based on stacking and inverse random under sampling methods. In: *2018 IEEE 15th International Conference on Networking, Sensing and Control (ICNSC)*. [S.l.: s.n.], 2018. p. 1–6.

ZHU, R.; GUO, Y.; XUE, J.-H. Adjusting the imbalance ratio by the dimensionality of imbalanced data. *Pattern Recognition Letters*, v. 133, p. 217–223, 2020. ISSN 0167-8655. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167865520300829>.