PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ

PRÓ-REITORIA DE PESQUISA, PÓS-GRADUAÇÃO E INOVAÇÃO
PROGRAMA INSTITUCIONAL DE BOLSAS DE INICIAÇÃO CIENTÍFICA (PIBIC)

RELATÓRIO FINAL DO ESTUDANTE
PIBIC – VIGÊNCIA 2023/2024

RELATÓRIO FINAL

AVALIAÇÃO DO IMPACTO DE TÉCNICAS DE SELEÇÃO DE INSTÂNCIAS EM ENSEMBLES ORIENTADOS A FLUXOS DE DADOS

FABRICIO ENEMBRECK

CURITIBA
2024

VITOR RODRIGUES IZIDORO
FABRICIO ENEMBRECK

CIÊNCIA DA COMPUTAÇÃO - PUCPR
MODALIDADE CNPq

AVALIAÇÃO DO IMPACTO DE TÉCNICAS DE SELEÇÃO DE INSTÂNCIAS EM ENSEMBLES ORIENTADOS A FLUXOS DE DADOS

Relatório Final apresentado ao Programa Institucional de Bolsas de Iniciação Científica (PIBIC), Pró-Reitoria de Pesquisa, Pós-Graduação e Inovação da Pontifícia Universidade Católica do Paraná.

Orientador: Prof. Dr. Fabricio Enembreck

CURITIBA
2024

RESUMO

A geração crescente de dados em tempo real de diversas fontes, dispositivos móveis e sensores realça a importância da mineração de fluxo de dados (Data Stream Mining – DSM). A classificação desses fluxos enfrenta desafios significativos, como conceitos em constante mudança, alta frequência de novos exemplos e recursos computacionais limitados. Algoritmos baseados em Ensembles têm mostrado bons resultados, mas ainda consomem muitos recursos. Na literatura de Ensembles, o algoritmo "Random Forest" destaca-se pela sua alta taxa de acerto. Ele cria várias árvores de decisão usando subconjuntos de atributos selecionados aleatoriamente e combina essas árvores de forma ponderada para a decisão final. Sua eficiência e velocidade de cálculo são notáveis em comparação com outros algoritmos de ensemble. Inspirado pelo "Random Forest", foi desenvolvido o "Adaptive Random Forest" (ARF), um algoritmo de ensemble com capacidade de aprendizagem *online*, o ARF reativa periodicamente os classificadores individuais para lidar com mudanças conceituais nos dados ao longo do tempo, suportando o aprendizado incremental sem reprocessar todo o conjunto de dados original. Outra abordagem de ensemble é o algoritmo "boosting", que treina modelos de aprendizado de máquina sequencialmente, ajustando cada novo modelo para corrigir os erros dos modelos anteriores. O "AdaBoost" é um exemplo popular de boosting, destacando-se por sua capacidade de adaptação aos erros durante o treinamento e sua eficiência na classificação de grandes quantidades de dados, no entanto, a sua taxa de acerto pode ser baixa com poucos dados de treinamento, problema que pode ser mitigado com a adição de novos dados de treinamento. Diante das limitações impostas pelo grande volume de dados em fluxo, esta pesquisa visa estudar como técnicas de Seleção de Instâncias podem mitigar a complexidade computacional envolvida na classificação de fluxos de dados. O objetivo é avaliar e desenvolver técnicas de seleção de instâncias para ensembles orientados a fluxos de dados, viabilizando a aplicação desses algoritmos em cenários de larga escala. Foram estudadas técnicas de seleção de instâncias para ambientes *batch* e *stream*, avaliando o impacto nos algoritmos de classificação de fluxos de dados (ARF e SRP), e desenvolvendo e avaliando uma técnica simples de seleção de instâncias baseada no erro de classificação, com o objetivo de ignorar o treinamento em instâncias cuja classificação obtida é a correta. Ao longo da pesquisa foi possível concluir que a aplicação dos ensembles para cenários de larga escala continuam inviáveis para casos em que a velocidade computacional é a prioridade, mas sendo viável em condições onde a principal prioridade seja a taxa de acerto, também sendo visível que a seleção de instâncias é capaz de reduzir o consumo de recursos computacionais ao sacrificar a taxa de acerto, abrindo caminho para futuras pesquisas serem desenvolvidas com técnicas de seleção de instâncias mais avançadas, em busca de mitigar as limitações encontradas.

Palavras-chave: 1. Mineração de fluxo de dados; 2. Algoritmos de ensemble; 3. Instance-Based Learning; 4. Random Forest; 5 Boosting; 6. Recursos computacionais.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS OU SIGLAS

MOA	- Massive Online Analisys
IS/SI	- Instance selection /seleção de instâncias
ARF	- Adaptive Random Forest
SRP	- Straming Random Patches
NN	- Nearest Neighbour
Obs.	- Observação
IBL	- *Instance-Based Learning*
ATISA	- *Adaptive Threshold-based Instance Selection Algorithm*

SUMÁRIO

1 INTRODUÇÃO

O aumento drástico na geração de dados em tempo real provenientes de diversas fontes, dispositivos móveis e inúmeros sensores, tem destacado a importância da mineração de fluxo de dados (Data Stream Mining – DSM) [13][14][15][16]. Devido a quantidade gigantesca de fluxo de dados, sua classificação enfrenta desafios significativos, incluindo conceitos em mudança, alta frequência de novos exemplos e recursos computacionais limitados. Algoritmos baseados em *ensembles* têm demonstrado excelentes resultados, porém, eles ainda sofrem com o consumo excessivo de recursos.

Uma literatura de suma importância para esta pesquisa foi sobre *técnicas de Instance-Based Learning* (IBL) [5]. Mesmo que este método de seleção e aprendizado não vá ser utilizado diretamente no projeto devido ao processamento elevado necessário, a análise dessas técnicas permite compreender como funcionam as técnicas baseadas em instâncias e a complexidade envolvida ao se tentar classificar ou selecionar dados. Conceitos e técnicas de IBL tem sido utilizado tanto para a classificação de dados quanto para seleção de instâncias.

Pode-se observar na literatura de *ensembles*, que um dos algoritmos mais promissores em taxa de acerto é o Random Forest [23]. Trata-se de um algoritmo classificador que utiliza o método de árvores de decisão. Seu objetivo é criar várias árvores de decisão usando um subconjunto de atributos selecionados aleatoriamente a partir de um conjunto principal. Após a criação dos conjuntos o algoritmo treina as árvores em diferentes instâncias de forma a maximizar o ganho de conhecimento para a resolução do determinado problema. Para isso, ele combina o subconjunto de árvores de forma ponderada para a tomada de decisão final. Este método é muito famoso para classificação de dados devido a sua alta taxa de acerto, além de uma velocidade de cálculo comparado com outros algoritmos *ensembles*, como podemos ver no “Estudo Comparativo entre os algoritmos de Mineração de Dados Random Forest e J48 na tomada de Decisão” [6].

O “Random Forest” foi utilizado como inspiração para o desenvolvimento de um algoritmo de *ensemble* com capacidade de aprendizagem *online*, chamado “Adaptive Radom Forest” (ARF) [12]. O algoritmo introduz uma técnica para lidar com as mudanças conceituais que podem surgir nos dados ao longo do tempo, sendo feito através da reativação periódica dos classificadores individuais, além de suportar o modelo de aprendizado incremental, ou seja, treinar o modelo em novos dados sem

ter que reprocessar todo o conjunto de dados originais, se tornando ideal para nossa pesquisa, onde vamos receber dados novos a todo momento.

Outra possibilidade orientada a ensemble é o algoritmo *boosting*, onde treina diversos modelos de aprendizado de máquina de modo sequencial, onde cada novo modelo é reajustado com o objetivo de corrigir os erros cometidos pelos modelos anteriores. Um exemplo bem popular deste método é o AdaBoost (*Adaptative Boosting*) [17], que é uma técnica especifica de *boosting*, tendo como principais características de destaque sua capacidade de se adaptar aos erros cometidos pelos modelos anteriores durante o processo de treinamento, diminuindo a interferência por ruído, também sendo boa para a classificação de grandes quantidades de dados recebidas a todo momento, tendo como principal ponto fraco a possível baixa taxa de acerto quando há poucos dados de treinamento, mas que pode ser contornado com adição de novos dados de treinamento para os modelos iniciais terem uma base solida antes de serem implementados na prática. Assim como *bagging[22]* utilizado no Random Forest, *boosting* também possui adaptações para o ambiente de aprendizagem *online* [17][22].

Tendo em vista as limitações impostas pelo grande volume de dados em fluxo aos algoritmos de *ensemble*, esta pesquisa visa estudar como técnicas de Seleção de Instâncias [7][18][19][20][21] podem ser úteis para mitigar a complexidade computacional envolvida na tarefa de classificação de fluxos de dados.

2 OBJETIVOS

O objetivo desse projeto consiste em avaliar e desenvolver técnicas de seleção de instâncias para *ensembles* orientados a fluxos de dados de forma a viabilizar a aplicação dos algoritmos em cenários de larga escala onde a quantidade de instâncias geradas em fluxo é grande.

2.1 OBJETIVO GERAL

Descobrir as melhores técnicas de seleção de instâncias para *ensembles* orientados a fluxo de dados para a aplicação dos algoritmos em cenários de larga escala.

2.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos do projeto são:

• Estudar técnicas de seleção de instâncias para ambientes *batch* e *stream*;

• Avaliar o impacto das técnicas de seleção de instâncias nos algoritmos estado-da-arte para classificação de fluxos de dados (ARF e SRP);

• Propor e avaliar técnicas simples de seleção de instâncias baseadas em performance de classificação;

3 MATERIAIS E MÉTODO

Neste projeto, foi utilizado um computador rodando o sistema operacional Windows 10 com 16gb de memória RAM, a IDE Eclipse [11] e o aplicativo MOA [12]. O MOA é um software dedicado à mineração de dados em *streaming* que foi desenvolvido para lidar com grandes volumes de dados em tempo real, possuindo uma arquitetura flexível e extensível, permitindo a experimentação e avaliação de algoritmos de aprendizado de máquina, tendo como uma de suas principais características sua capacidade de suportar uma ampla gama de algoritmos de aprendizado de máquina. As principais fontes de pesquisa no momento inicial foram arquivos disponibilizados pelo professor Fabricio Enembreck, após isso a pesquisa prosseguiu por teses publicadas de outros alunos encontradas no Google acadêmico.

3.1 REVISÃO DA LITERATURA

A revisão da literatura foi iniciada com a leitura de artigos relacionados à família de algoritmos *Instance-Based Learning* (IBL) [5], sendo uma ótima introdução ao aprendizado de máquina, explicando em português e de modo simples como os algoritmos de *instance-based learning* funcionam. Verificou-se que esses algoritmos são variações do Nearest Neighbour (NN), sendo que cada uma delas tem objetivos de contornar alguma limitação, tanto do NN quando de alguma versão prévia sua. Os IBL armazenam instâncias de treinamento na memória como pontos em um espaço N-dimensional, sendo dois desafios importantes desse modelo (I) quais pontos

armazenar? E (II) qual métrica de seleção adotar? Tendo respostas diferentes em cada versão do algoritmo, indo do IB1 ao IB5.

Após a leitura dos IBL, analisou-se *surveys* sobre técnicas de seleção de instâncias, sendo que em [8], é explicado que os algoritmos de seleção são divididos em três tipos, sendo eles: (I) *noise filters* (filtros de ruídos), (II) *condesation algorithms* (algoritmos de condensação) e (III) *prototype searching algorithms* (algoritmos de busca de protótipos). Cada um tendo suas funções especificas.

Técnicas de filtros de ruído foram criadas originalmente em 1972 com o “Edited Nearest Neighbor” [8], e têm como objetivo principal reduzir o ruído causado por dados incorretos, removendo as instâncias cuja classe não concorda com a classe majoritária de seus vizinhos próximos.

Algoritmos de condensação tem como objetivo armazenar apenas os dados classificados de modo errado, com o objetivo de aprender com seus erros, um ótimo exemplo se encontra na família IBL, com o IB2.

Algoritmos de busca de protótipos utilizam protótipos para representar o “conhecimento” de forma compactada, transformando o conjunto de treinamento original em diversos protótipos que cobrem áreas especificas de um espaço conhecido como Diagrama de Voronoi [8], onde cada área dele possui características especificas.

Essa diferenciação de algoritmos é de extrema importância para esta pesquisa, pois, quando estes conceitos são aplicados de maneira correta, em alguns casos os conjuntos de treinamento são extremamente reduzidos, podendo atingir até mesmo 1% do tamanho original e mantendo a alta performance para classificação.

Em [7], as técnicas apresentadas têm como objetivo reduzir a quantidade de memória utilizada, removendo as instâncias consideradas “irrelevantes”, classificando as técnicas de redução em três tipos: “*condensation”*, “edition” e “hybrid”. O método “condensation” busca um subconjunto consistente para treiná-lo. Um subconjunto consistente é aquele que, da melhor maneira possível, pode eliminar instâncias supérfluas no conjunto de treinamento de modo que não afete a precisão de classificação do conjunto. O método “edition”, tem o objetivo de remover instâncias que causem ruído na classificação e o “hybrid” combina as características de ambos. O artigo mostra um algoritmo chamado “*Adaptive Threshold-based Instance Selection Algorithm*” (ATISA), que busca selecionar os melhores conjuntos de instâncias e

realizar o modelo “hybrid” para a melhora computacional da seleção de instâncias fornecidas como entrada.

Em [1], o estudo enfatiza como os métodos de ensemble são uteis para enfrentar a evolução dos conceitos ao longo do tempo, já que, podem se adaptar às mudanças, atribuindo pesos diferentes aos membros dos ensembles de acordo com sua relevância para o conceito atual, destacando que os métodos ensemble são aplicáveis tanto em ambientes de aprendizado de lotes estáticos quanto em fluxos de dados constantes. A pesquisa classifica 60 algoritmos diferentes por meio de uma taxonomia proposta.

O artigo [3] discute sobre a importância do pré-processamento, ou seja, redução de dados no momento do treinamento para diminuir o tempo de processamento, visando a enumeração, classificação e análise completa das contribuições do pré-processamento, mostrando os conceitos de: “data streaming” e “concept drift”, “data reduction” e suas contribuições na redução de dados *on-line*.

Em [4] os autores têm como objetivo utilizar um algoritmo para simular o resultado do efeito das vacinas RV144 em anticorpos e das atividades citotóxicas das células primarias. Esta pesquisa usou uma técnica de mineração de fluxo de dados a fim de simular dados reais, tendo como objetivo identificar quais as instâncias são mais relevantes e produzem os melhores resultados, com menor quantidade de ruído e dados “desnecessários”. Cada conjunto de dados contém 100 amostras, sendo 80 reais e 20 placebos, sendo que o conjunto de dados possui 20 características de anticorpos diferentes.

Em [6] os autores explicam e comparam os algoritmos de mineração de dados ensembles “Random Forest” e “J48”, mostrando que o grau de certeza do “Random Forest” foi maior, com 99% de certeza, levando 37,98 segundos, já o “J48” teve uma taxa de acerto de 88%, levando apenas 1,37 segundos para os experimentos realizados, deixando evidente que é necessário considerar se vale a pena sacrificar parte da taxa de acerto pelo tempo de execução.

Os autores em [9] discutem a tarefa de quantificação no aprendizado de máquina (que tem como objetivo fornecer uma estimativa precisa da distribuição de classes de um conjunto não rotulado) e como ele possui aplicações práticas significativas, destacam a importância da quantificação em cenários de fluxos de dados não estacionários, nos quais os dados chegam continuamente e podem variar ao longo do tempo. Para lidar com essa dificuldade, é proposto o algoritmo SQSI

(Algoritmo de quantificação de *data streams* com mudança de *conceito*), pois ele é resiliente a mudanças, podendo detectar desvios de maneira não supervisionada, podendo ser custoso em termos de recursos, mas ainda ser útil por não necessitar de supervisão humana, o tornando mais eficiente nestes casos.

Em [2] os autores discutem sobre como o aprendizado de máquina se torna cada vez mais utilizado em aplicações do mundo real, como por exemplo prever interesses de usuários em anúncios ou até mesmo filtros de *spam*. Esta pesquisa foca em um novo algoritmo derivado do *Random Forest*, combinando ele com características de algoritmos em lotes com métodos de atualização dinâmica para conseguir lidar com os grandes fluxos de dados em evolução, utilizando um método de reamostragem teoricamente sólido, baseado em *bagging online* [12]. O algoritmo é capaz de monitorar o surgimento de avisos e mudanças drásticas nos dados para que, com o treinamento de novas árvores em segundo plano ela possam substituir a original em caso de necessidade, possuindo também uma configuração experimental que inclui diferentes cenários de teste, como atrasos na disponibilidade de rótulos.

Em [10] os autores incluem detalhamentos dos algoritmos “Random Tree”, “J48” e “Functional Trees”. O “J48” (Java) é a versão *open source* do algoritmo de árvore de decisão “C4.5” [24], que é um classificador estático, baseado no conceito de entropia, onde cada nó do algoritmo determina qual atributo realiza a melhor separação entre classes, gerando um nó univariado (onde cada nó da árvore é baseado em somente um atributo). O “Functional Trees”, é um algoritmo de construção de árvores de decisão obliquas, onde as árvores são multivariadas (cada regra possui uma expressão baseada em um ou mais atributos). O “Random Tree” considera apenas alguns atributos escolhidos aleatoriamente para cada nó da árvore, não realizando a “poda” de galhos para refinamento e, também não permitindo estimativa de probabilidade de classes. Os algoritmos “J48"' e “Functional Tree” obtiveram o índice kappa de 0,76 e exatidão global de 88,31%, já o “Random Tree” teve um índice kappa de 0,75 e taxa de acerto de 86,74% no artigo analisado. Mesmo o “J48” e o “Functional Tree” tendo obtido os mesmos resultados, o algoritmo J48 obteve maior destaque por ser mais intuitivo, proporcionando um maior entendimento de como acontece a separação de classes.

3.2 DESENVOLVIMENTO DO MÉTODO DE SELEÇÃO DE INSTÂNCIAS

O código principal para este projeto foi desenvolvido utilizando como inspiração o código do *DriftDetectionMethodClassifier, .java* pois a estrutura de seleção do *learner*, escolha de arff's externos já havia sido desenvolvida, sendo um código com a seleção de instâncias baseada em erros.

A ideia base para a criação do código era que tal seleção de instâncias teria como objetivo reduzir a quantidade de instâncias que seriam utilizadas no treinamento e aumentar a taxa de acerto dos algoritmos, a partir do momento que o classificador treina apenas as instâncias que foram classificadas do modo incorreto, podemos concentrar o aprendizado nos casos mais difíceis, que por sua vez, melhora a eficiência do treinamento.

TABELA 1 - Pseudocódigo

Classe InstanceSelectedClassifier

variáveis:

classifier: classificador atual;

Função resetLearningImpl():

Inicializa 'classifiers' como novo classificador;

Reseta aprendizagem de 'classifier';

Função trainOnInstanceImpl(instância):

Se classe verdadeira da instância igual classe prevista pelo classificador:

Não faça nada;

Senão:

Treine o classificador na instância;

Função getVotesForInstance(instância):

Retorna os votos do classificador para a instância;

É possível observar na Tabela 1 que o método pode ser caracterizado como um *meta-learner*, podendo ser utilizado para qualquer tipo de algoritmo, desde algoritmos monolíticos até ensembles. O processo de seleção de instâncias pode ser descrito em três etapas principais:

I. **Inicialização e Reset:** A função resetLearningImpl() prepara o classificador para um novo ciclo de aprendizado, garantindo que ele esteja pronto para processar novas instâncias.

II. **Treinamento Seletivo:** A função trainOnInstanceImpl(instância) implementa a lógica central do nosso método. Ao comparar a classe verdadeira da instância com a previsão do classificador, decidimos se a instância deve ser utilizada para treinamento. Isso permite que o classificador se concentre nas instâncias onde cometeu erros, ajustando-se para melhorar sua precisão.

III. **Previsões e Votos:** A função getVotesForInstance(instância) é responsável por retornar a previsão do classificador para uma determinada instância. Esta função pode ser usada para avaliar o desempenho do classificador em novas instâncias.

As vantagens dessa abordagem incluem:

- **Redução de Dados de Treinamento:** Ao treinar apenas em instâncias onde o classificador errou, reduzimos a quantidade de dados necessários, economizando tempo e recursos computacionais.
- **Melhoria da Precisão:** Focando no aprendizado em casos mais difíceis, esperamos melhorar a precisão geral do classificador, uma vez que ele será mais adaptado para lidar com instâncias que inicialmente apresentou dificuldade.

3.3 DESENVOLVIMENTO DOS EXPERIMENTOS

O primeiro passo para iniciar este projeto foi a instalação do aplicativo MOA [12] e da IDE Eclipse [11]. A parte mais desafiadora foi a instalação e configuração do ambiente do MOA [12], que exigiu a reconfiguração do *build path* do Java e a compreensão do método correto para utilizar a variável *javaagent*.

Em seguida, o objetivo foi definir as métricas a serem avaliadas durante o treinamento dos algoritmos. As métricas escolhidas foram:

- Taxa de classificações corretas: Calculada com base na quantidade de previsões corretas dividida pela quantidade total de previsões.
- Tempo: Calculado pelo próprio MOA em segundos.
- Memória utilizada: Também calculada pelo próprio MOA.

Com as métricas definidas, o próximo passo foi determinar a seleção de instâncias. Optou-se pela seleção baseada nos erros, e o código foi criado conforme os requisitos estabelecidos. Além disso, decidiu-se pelo uso do método *prequential*, esse método realiza a testagem primeiramente e, em seguida, treina o modelo com base nos resultados obtidos. O processo é realizado instância por instância de maneira contínua, proporcionando uma maneira robusta de manter e monitorar o desempenho do modelo ao longo do tempo. Após isso, iniciou-se a aprendizagem da interface do MOA para aprender as noções básicas de configuração para o treinamento dos modelos e começar a aprender como fazer criações de baterias de testes com códigos .bat. A seguir é apresentado um pseudocódigo na Tabela 2 para melhor explicação (nele as anotações serão feitas utilizando * para não ocorrer confusão na leitura):

TABELA 2 - Pseudocódigo

@echo off cd caminho_para_lib_do_moa\lib set MEMORY=10000m *quantidade de memória *ram* disponibilizada para o teste em megabytes* java -Xmx%MEMORY% *inicia o java e recupera a variável que armazena a quantidade de memória a ser utilizada* -cp moaPIBIC.jar -javaagent:sizeofag-1.0.4.jar *Este parâmetro especifica um agente Java para medir o tamanho dos objetos na memória* moa.DoTask "EvaluatePrequential -l (drift.InstanceSelectedClassifier -l *define se vai ou não utilizar seleção de instâncias* (meta.nome_do_algoritmo_de_ensemble -s 100*quantidade de subclassificadores*)) -s (ArffFileStream -f C:\caminho_do_dataset/dataset.arff) -e BasicClassificationPerformanceEvaluator -f 10000000*número de instâncias a serem processadas* -q 4500*intervalo de relatório dos resultados*" > C:\onde_será_ser_salvo_o_resultado\nome_do_txt.txt

Após os treinamentos das baterias com os algoritmos de: Levering, Adaptive Random Forest, OzaBoost e Streaming Random Patches (todos com e sem seleção de instâncias), foi desenvolvido um código em Python para separar com mais facilidade os dados importantes para esta pesquisa em um arquivo do Excel. Depois dos dados mais importantes serem separados, foram criadas três tabelas, a tabela de classificação (com os dados de taxa de acerto), a tabela “Time” (com o tempo de treinamento), “*modelCost”* (com o custo de memória), cada uma delas também teve uma tabela de *ranking* para melhor visualização, e uma tabela com os dados de cada *dataset*. Após isso iniciou-se a escrita da análise e do relatório.

4 RESULTADOS

TABELA 3 - Análise dos *Datasets*

Dataset	Tipo	Atributos	instancias	Distribuição de classes	numérico	categórico
airlines	Real	7	539383	6,40/8,46/2,13/3,92/11,30/3,36 /1,03/9,32/3,84/2,34/5,19/ 5,77/2,54/5,12/6,79/3,86/1,20/17,45	3	4
kdd99	Real	41	489831	0,044974/0,019986/0,000184/ 21,88/0,000204/0,000082/0,324430/ 0,000612/0,000612/0,000429/0,000041/0,254796/ 0,000143/57,32/0,005389/0,000061/0,020823 /0,047280/ 0,000245/0,000408/0,212578/19,86/0,001082	34	7
kddcup	Real	41	494021	19,69/0,445932/0,006073/0,001619/ 0,010728/0,002429/0,252418/0,004251/ 0,001822/0,001417/21,70/0,046759/ 0,000607/0,000810/0,053439/0,210517/ 0,002024/0,321646/56,84/0,000405/ 0,198170/0,206469/0,004048	34	7
keystroke	Real	10	1600	25/25/25/25	10	0
luxembourg	Real	31	1901	51,39/48,61	16	15
NOAA	Real	8	1859	68,62/31,38	8	0
nomao	Real	118	34465	28,56/71,44	89	29
outdoor	Real	21	4000	2,50*39	21	0
ozone	Real	72	2534	93,69/6,31	72	0
poker	Real	10	829201	50,11/42,26/4,76/2,12/0,388929/0,199831/0, 143029/0,023517 /0,001327/0,000241	5	5
rialto	Real	27	82250	10,00*9	27	0

0bs: o atributo *class* foi desconsiderado.

Ao analisarmos a Tabela 3, é observado que toda a tabela consiste em *datasets* reais, podemos observar que o *dataset* “keystroke” possui a menor quantidade de instâncias e “kdd99” possui a maior.

TABELA 4 - Tabela de Classificação.

Dataset	levering	IS-levering	ARF	IS-ARF	OzaBoost	IS-OzaBoost	SRP	IS-SRP
airlines	63,506	59,752	66,720	61,788	64,930	61,032	**68,565**	62,444
kdd99	99,958	99,969	99,973	99,976	99,913	99,972	99,981	**99,982**
kddcup	99,809	99,828	99,857	99,634	99,764	99,802	**99,890**	99,886
keystroke	87,438	81,000	91,813	88,375	82,563	82,250	**94,688**	89,250
luxembourg	99,421	98,317	**99,684**	99,527	98,211	97,317	99,421	99,369
NOAA	78,556	74,211	**79,465**	74,018	74,844	72,779	79,239	73,930
nomao	96,347	96,779	97,229	96,968	93,698	95,813	**97,383**	96,953
outdoor	57,650	63,975	64,325	67,925	55,900	61,150	68,475	**70,300**
ozone	**93,646**	91,910	**93,646**	91,160	93,528	90,963	93,607	91,594
poker	**95,473**	91,389	88,780	93,124	83,280	77,499	89,798	94,696
rialto	61,066	67,606	72,119	74,338	30,500	33,945	**80,010**	77,364

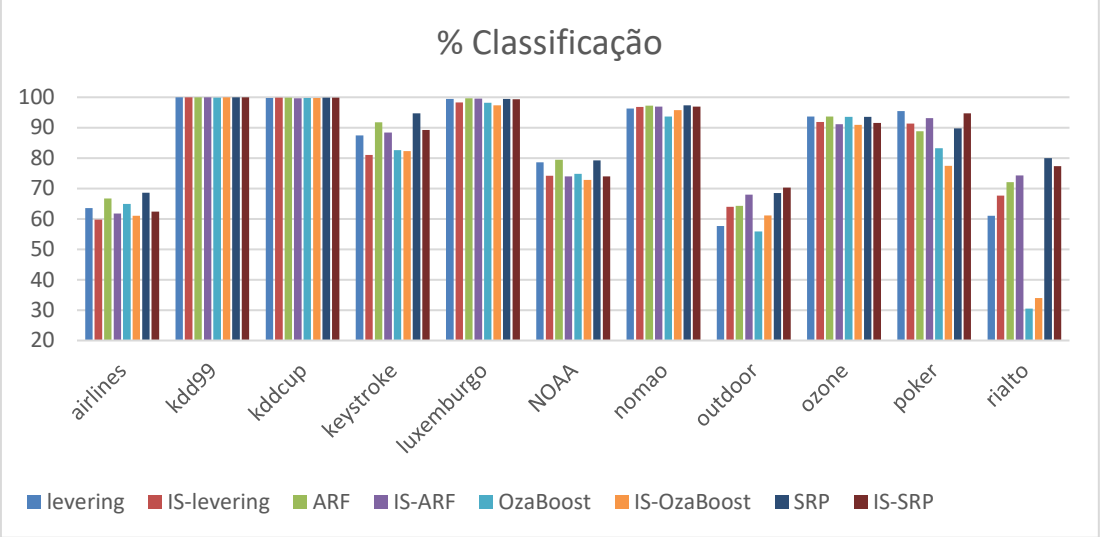
Analisando a Tabela 4 podemos concluir que, na maior parte das bases de dados a taxa de acerto se mantém alta, uma taxa de acerto maior que 80%, com exceção da base “outdoor”, “NOAA” e uma exceção em “keystroke”, sendo provável tal exceção por ser o *dataset* com o menor número de instâncias. Um fato a ser ressaltado é a diferença entre os algoritmos com e sem a seleção de instâncias, onde eles obtiveram os resultados opostos do esperado. Esperava-se que os algoritmos com a seleção de instâncias obtivessem um maior percentual de acerto, fato que ocorreu na menor parte dos testes. Na maior parte das vezes os algoritmos sem a seleção obtiveram um maior percentual de acerto. Outro fator a se ressaltar foram algumas das diferenças bruscas que tivemos entre os algoritmos com e sem a seleção de instâncias, como por exemplo no teste com o *dataset* “Airlines” do algoritmo Streaming Random Patches, onde, ao adicionarmos a seleção de instâncias, ele obteve uma queda de precisão de 6%, ou nos testes com a base de dados “keystroke”, onde obtivemos uma queda de 13% em sua taxa de precisão. Foram poucos os casos em que obtivemos uma melhor taxa de acerto ao adicionarmos a seleção de instância, sendo que em nenhum momento o ganho de precisão foi extremamente significativo, não passando do ganho de precisão de 5,4%.

TABELA 5 - Tabela de Ranking da Classificação.

Dataset	levering	IS-levering	ARF	IS-ARF	OzaBoost	IS-OzaBoost	SRP	IS-SRP
airlines	4	8	2	6	3	7	**1**	5
kdd99	7	6	4	3	8	5	2	**1**
kddcup	5	4	3	8	7	6	**1**	2
keystroke	5	8	2	4	6	7	**1**	3
luxembourg	4	6	**1**	2	7	8	3	5
NOAA	3	5	**1**	6	4	8	2	7
nomao	6	5	2	3	8	7	**1**	4
outdoor	7	5	4	3	8	6	2	**1**
ozone	2	5	**1**	7	4	8	3	6
poker	**1**	4	6	3	7	8	5	2
rialto	6	5	4	3	8	7	**1**	2
raking médio:	4,545	5,545	2,727	4,364	6,364	7,000	2,000	3,455

Analisando a Tabela 5 com ranking, se pode confirmar mais facilmente que, em todos os casos, a média geral de acerto dos algoritmos sem a seleção de instâncias é maior em todos os casos analisados, é também possível analisar que o algoritmo OzaBoost, tanto com quanto sem a seleção de instâncias possui a menor taxa de acerto, mantendo o padrão em que a seleção de instâncias contribui para uma taxa menor de acerto, sendo o Streaming Random Patches sem seleção de instâncias o algoritmo com a maior taxa de acerto.

FIGURA 1 - Gráfico da Classificação



Analisando a Figura 1, é possível observar que os *datasets* com maior taxa de acerto são o “kdd99” e “kddcup” respectivamente. Um detalhe interessante a ser ressaltado é que ambos foram criados para uma competição anual de mineração de dados e aprendizado de máquina, sendo “kddcup” feita para a primeira edição em 1997 e “kdd99” para a competição de 1999. Podendo também ser analisado que a menor taxa de acerto pertence ao algoritmo OzaBoost, tanto com quanto sem seleção de instâncias, com o *dataset “*Rialto”.

TABELA 6 - Tabela do Tempo.

Dataset	levering	IS-levering	ARF	IS-ARF	OzaBoost	IS-OzaBoost	SRP	IS-SRP
airlines	2439,125	1235,469	2297,031	1008,047	19,391	13,328	2503,484	1054,016
kdd99	426,984	619,266	4929,875	6450,297	296,094	874,797	7663,547	6231,063
kddcup	802,031	808,297	686,578	568,953	66,109	87,313	883,156	707,031
keystroke	2,984	0,891	4,453	1,547	0,297	0,156	4,500	1,844
luxembourg	2,375	1,188	2,234	1,203	0,500	0,406	7,703	1,875
NOAA	14,609	5,141	39,922	10,563	0,938	0,578	43,969	12,672
nomao	198,828	17,016	267,109	31,359	8,922	3,172	426,156	58,828
outdoor	52,047	24,047	39,203	28,391	2,578	3,016	54,094	30,375
ozone	12,641	2,234	25,219	5,391	0,750	0,594	23,594	6,031
poker	634,188	232,641	2311,234	278,906	27,172	22,109	4024,000	502,609
rialto	232,063	105,844	341,063	106,203	12,906	7,766	591,953	207,000

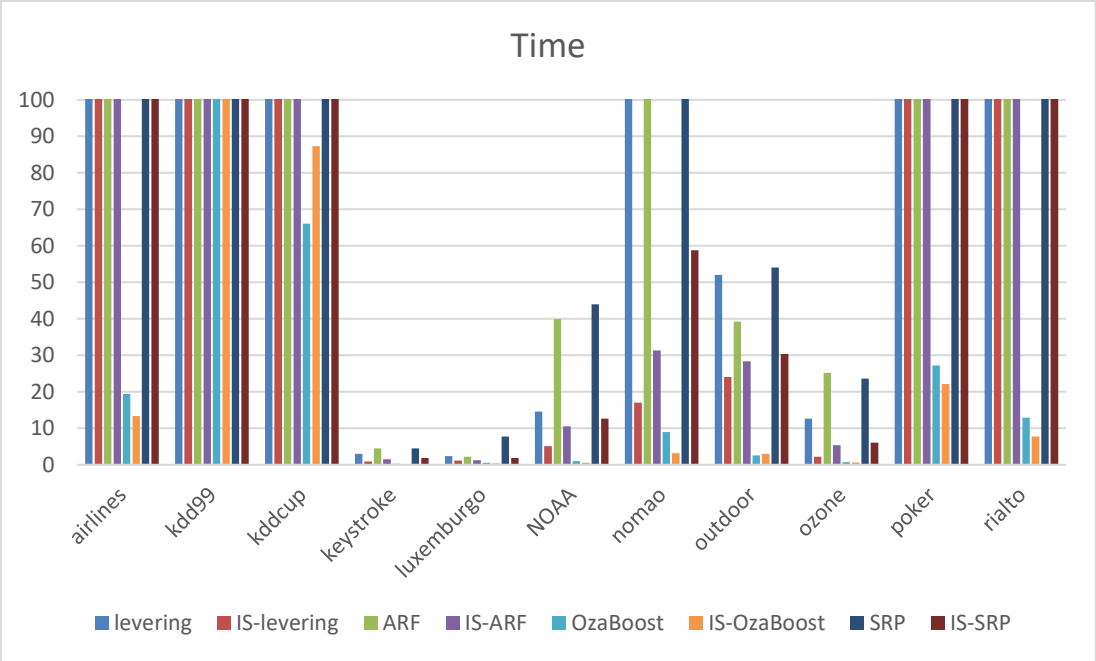
Ao analisarmos os dados de tempo de processamento representados na Tabela 6, reparamos que os algoritmos com seleção de instância são mais rápidos na maior parte das vezes. O único *dataset* onde os algoritmos com seleção de instâncias foram mais lentos 100% das vezes foi no “kdd99”. Os algoritmos com a seleção de instâncias obtiveram redução significativa de tempo em relação aos sem seleção, ressaltando especialmente algoritmo Adaptive Random Forest com o *dataset* “poker”, onde foi obtida uma redução de tempo de 87,93% e o algoritmo Streaming Random Patches, onde a utilização da seleção de instâncias no *dataset* “poker” obteve uma redução no tempo de 87,51%.

TABELA 7 - Tabela Ranking do Tempo.

Dataset	levering	IS-levering	ARF	IS-ARF	OzaBoost	IS-OzaBoost	SRP	IS-SRP
airlines	7	5	6	3	2	1	8	4
kdd99	2	3	5	7	1	4	8	6
kddcup	6	7	4	3	1	2	8	5
keystroke	6	3	7	4	2	1	8	5
luxembourg	7	3	6	4	2	1	8	5
NOAA	6	3	7	4	2	1	8	5
nomao	6	3	7	4	2	1	8	5
outdoor	7	3	6	4	1	2	8	5
ozone	7	3	6	4	2	1	8	5
poker	6	3	7	4	2	1	8	5
rialto	6	3	7	4	2	1	8	5
ranking médio:	6	3,5455	6,1818	4,0909	1,7273	1,4545	8	5

Analisando a Tabela 7 com ranking de tempo, conseguimos confirmar mais facilmente que, em todos os casos, o ranking de tempo dos algoritmos com a seleção de instâncias é melhor em todos os casos analisados, também facilitando a análise do *dataset* “kdd99”, onde é possível identificar que, apenas nele, parte predominante dos algoritmos sem a seleção de instância foram mais rápidos, da mesma forma sendo possível verificar que o Streaming Random Patches sem a seleção de instâncias foi o algoritmo mais lento na média geral e o OzaBoost com seleção de instâncias é o mais rápido.

FIGURA 2 – Gráfico do Tempo.



Na Figura 2 podemos analisar com mais facilidade a diferença do tempo levado em cada *dataset,* ressaltando o “keystroke” e “Luxembourg”, que foram os menores e o *dataset* “nomao”, onde foi obtida a maior variação de tempo entre os algoritmos.

TABELA 8 - Tabela do Custo de Memória.

Dataset	levering	IS-levering	ARF	IS-ARF	OzaBoost	IS-OzaBoost	SRP	IS-SRP
airlines	0,02662	0,02988	0,04495	0,02014	0,00001	0,00003	0,05422	0,02355
kdd99	0,05200	0,14799	0,18297	0,19728	0,00023	0,00183	0,26704	0,17254
kddcup	0,01162	0,01905	0,02391	0,01803	0,00005	0,00018	0,03081	0,01961
keystroke	0,00003	0,00002	0,00009	0,00003	0,00000	0,00000	0,00009	0,00003
luxembourg	0,00003	0,00003	0,00004	0,00002	0,00000	0,00000	0,00015	0,00003
NOAA	0,00016	0,00013	0,00084	0,00021	0,00000	0,00000	0,00093	0,00026
nomao	0,00211	0,00040	0,00552	0,00059	0,00001	0,00001	0,00982	0,00122
outdoor	0,00054	0,00058	0,00138	0,00108	0,00000	0,00001	0,00198	0,00114
ozone	0,00013	0,00005	0,00060	0,00009	0,00000	0,00000	0,00048	0,00011
poker	0,00719	0,00570	0,06453	0,00739	0,00002	0,00004	0,10560	0,01452
rialto	0,00230	0,00250	0,00745	0,00260	0,00001	0,00002	0,01245	0,00550

Obs.: os resultados da tabela foram multiplicados por 100.

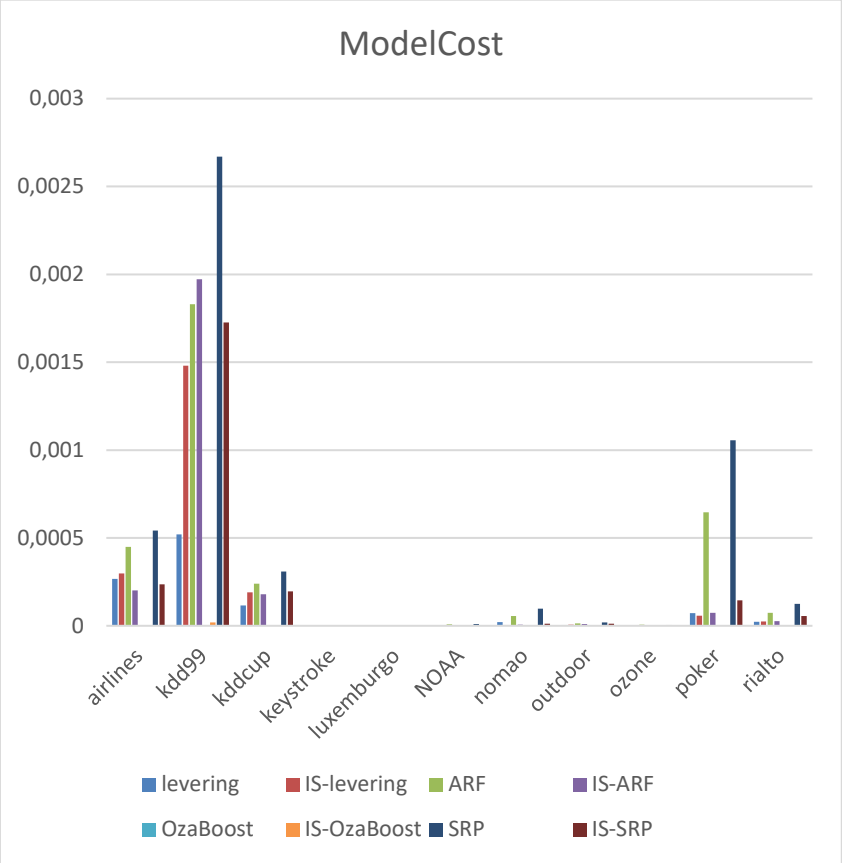
Analisando a tabela 8, podemos reparar que os valores muito baixos acabam sendo difíceis de serem analisados, podemos reparar que o OzaBoost tem um custo de memória mais eficiente que os outros algoritmos. Um caso interessante a se ressaltar é que o OzaBoost foi o único algoritmo em que, sua versão sem a seleção de instâncias se mostrou superior a versão com seleção.

TABELA 9 - Tabela de Ranking do Custo de Memória.

Dataset	levering	IS-levering	ARF	IS-ARF	OzaBoost	IS-OzaBoost	SRP	IS-SRP
airlines	5	6	7	3	1	2	8	4
kdd99	3	4	5	6	1	2	7	4
kddcup	3	5	7	4	1	2	8	6
keystroke	5	3	7	4	1	2	8	6
luxembourg	4	5	7	3	1	2	8	6
NOAA	4	3	7	5	1	2	8	6
nomao	6	3	7	4	2	1	8	5
outdoor	3	4	7	5	1	2	8	6
ozone	6	3	8	4	1	2	7	5
poker	4	3	7	5	1	2	8	6
rialto	3	4	7	5	1	2	8	6
ranking médio:	4,182	3,909	6,909	4,364	1,091	1,909	7,818	5,455

Já analisando esta Tabela 9 com o ranking, conseguimos confirmar mais facilmente que a média geral do custo computacional foi melhor com a seleção de instâncias tendo como única exceção o algoritmo OzaBoost. Outro fato interessante para analisarmos é que, em quase 90,91% dos casos, o OzaBoost, sem a seleção de instâncias, obteve o menor custo computacional, tendo apenas uma exceção, no *dataset* “nomao”, onde o melhor desempenho foi do OzaBoost com a seleção de instâncias, enquanto o algoritmo que mais possui consumo computacional é “Streaming Random Patches” sem a seleção de instâncias.

FIGURA 3 - Gráfico do Custo de Memória.



Analisando a Figura 3, podemos ressaltar o “kdd99”, sendo este o *dataset* com maior custo de memória, fato que faz sentido levando em consideração ele possuir uma maior quantidade de instâncias que todos os outros.

5 DISCUSSÃO

No decorrer das análises, podemos observar que a seleção de instâncias reduziu a taxa de acerto em todos os algoritmos testados, melhorando seu desempenho e custo computacional. Podemos reparar também que o algoritmo com maior taxa de acertos foi o “Adaptive Random Forest” sem a seleção de instâncias, sendo uma grandeza diretamente proporcional a sua grande demora e custo computacional em relação aos outros algoritmos. É possível observar que da mesma forma, o algoritmo “OzaBoost” com a seleção de instâncias obteve a pior porcentagem de acerto e melhor tempo dentre todos os algoritmos, mas não

sendo o melhor em custo computacional, ficando atrás apenas o “OzaBoost” sem a seleção de instâncias.

Como resultado desta análise, podemos concluir que, nos casos testados, a seleção de instâncias sacrifica o percentual de acerto em troca de uma melhoria no custo computacional e na velocidade de treinamento. A escolha de realizar a seleção de instâncias depende muito da necessidade de cada projeto: se for necessário um processamento mais rápido e com menor custo computacional, a seleção de instâncias seria uma solução viável. Por outro lado, se a precisão for uma prioridade e o custo computacional não for uma preocupação, é mais sensato manter o algoritmo sem a seleção de instâncias. Essa análise evidencia a necessidade de mais pesquisas sobre técnicas de seleção de instâncias que possam mitigar a queda significativa na precisão dos algoritmos.

6 CONCLUSÃO

O uso de ensembles para a aplicação de cenários em larga escala ainda parece inviável na maior parte dos casos, pois são algoritmos muito pesados e lentos, mas sendo ótimos em taxa de acerto. Podem ser facilmente aplicados em situações em que a velocidade não é um fator crucial, e sim a taxa de acerto. Por outro lado, as técnicas de seleção de instâncias se mostraram eficazes para reduzir o consumo de recursos computacionais, mas introduz um *trade-off* em relação às taxas de acerto. Dessa forma, conclui-se que mais pesquisas devem ser desenvolvidas sobre técnicas de seleção de instâncias capazes de mitigar as limitações encontradas, que podem ter sido potencializadas pela simplicidade da técnica de seleção avaliada, que treina o modelo apenas com instâncias onde há erro de classificação, supondo ingenuamente que o treinamento não é necessário quando o modelo acerta a classificação de uma instância. Técnicas mais elaboradas de seleção de instâncias podem ser utilizadas ao avaliar-se a complexidade de classificação de uma instância (*instance hardness*) avaliando-se, por exemplo, a entropia existente entre as decisões dos membros do ensemble, as probabilidades de saída geradas, dentre outras estratégias. Essas novas abordagens devem fazer parte da continuidade da presente pesquisa.

7 USO DE INTELIGÊNCIA ARTICIFIAL GENERATIVA

PERGUNTAS SOBRE O USO DE IA GENERATIVA
1) Para escrita deste relatório, alguma ferramenta de inteligência artificial generativa foi utilizada? Sim (☒) Não (☐)
2) Qual(is) ferramenta(s) de IA generativa você utilizou? Não se aplica (☐) Se sim, cite quais: ChatGPT
3) Indique quais os usos de IA generativa foram aplicadas no neste relatório. Não se aplica (☐) Correção gramatical (ortografia e concordância) (☐) Formatação das referências (☐) Gerar partes do texto escrito (ex: frases, parágrafos, conceitos) (☐) Gerar a totalidade do texto escrito (☐) Gerar citações (☐) Criação/edição das imagens e gráficos (☐) Correção/auxílio na formatação final dos códigos estatísticos (ou outro software) (☐) Outros usos – especificar: utilizei tanto para auxiliar com o aprendizado das baterias de testes em .bat quanto para me ajudar a resolver problemas com configurações do MOA

REFERÊNCIAS

[1] Heitor Murilo Gomes, Jean Paul Barddal, Fabrício Enembreck, and Albert Bifet. 2017. A Survey on Ensemble Learning for Data Stream Classification. ACM Comput. Surv. 50, 2, Article 23 (March 2017), 36 pages. DOI: https://doi.org/10.1145/3054925

[2] Heitor Murilo Gomes, Albert Bifet, Jesse Read, Jean Paul Barddal, Fabricio Enembreck, Bernhard Pfharinger, Geoff Holmes, Talel Abdessalem. Adaptive random forests for evolving data stream classification. In Machine Learning, DOI: 10.1007/s10994-017-5642-8, Springer, 2017.

[3] Sergio Ramírez-Gallego, Bartosz Krawczyk, Salvador García, Michał Woźniak, Francisco Herrera, A survey on data preprocessing for data stream mining: Current status and future directions, Neurocomputing, Volume 239, 2017, Pages 39-57, ISSN 0925-2312, https://doi.org/10.1016/j.neucom.2017.01.078.

[4] Sarac, Ferdi & Seker, Huseyin. (2016). An instance selection framework for mining data streams to predict antibody-feature function relationships on RV144 HIV vaccine recipients. 003356-003361. 10.1109/SMC.2016.7844752.

[5] Nicoletti, Maria & Santos, Flávia. A Familia de Algoritmos Instance-Based Learning (IBL). Universidade Federal de São Carlos (UFSCar).

[6] Lorenzett, Cassio & Telöken, Alex. Estudo Comparativo entre os algoritmos de Mineração de Dados Random Forest e J48 na tomada de Decisão. Curso de Ciência da Computação– Universidade De Cruz Alta (UNICRUZ). Disponível em < https://web.archive.org/web/20180424102752id_/http://revistaeletronica.unicruz.edu.br/index.php/computacao/article/viewFile/4023/737>.

[7] Cavalcanti, George; Ren Tsang & Pereira, Cesar. ATISA: Adaptative Threshold-base Instance Selection Algorithm. Editor-in-Chief Binshan Lin: ScienceDirect. Disponível em < https://www.sciencedirect.com/science/article/abs/pii/S095741741300448X >.

[8] Jankowski, Nobert & Grochowski, Marek. Comparison of Instances Selections I. Algorithms Survey. Nicholaus Copernicus University: Department of Informatics.

[9] Maletzke, André; dos Reis, Denis & Batista, Gustavo. Combining instance selection and self-training to improve data stream quantification. Journal of the Brazilian Computer Society. Disponível em < https://doi.org/10.1186/s13173-018-0076-0>

[10] Debastiani, Aline; Borges dos Santos, Marcielli; Da Silva, Ricardo; et al. ÁRVORE DE DECISÃO APLICADA NA IDENTIFICAÇÃO DE ÁREAS DE RISCO E DESLIZAMENTO: COMPARAÇÃO DOS METODOS J48,

FUNCTIONAL TREES E RANDOM TREE. Acadêmicos do curso de Engenharia Florestal, Dois Vizinhos: Universidade Tecnológica Federal do Paraná. 2013. Disponível em < v https://cbcg.ufpr.br/home/wp-content/uploads/2013/11/F021_CBCG13.pdf>

[11] Eclipse Foundation. Home-Page da IDE Eclipse. Disponível em: https://eclipseide.org/

[12] MOA. Home-Page do aplicativo MOA. Disponível em: https://moa.cms.waikato.ac.nz/documentation/

[13] BARDDAL, JEAN PAUL ; GOMES, HEITOR MURILO ; Enembreck, Fabrício . SNCStream. In: the 30th Annual ACM Symposium, 2015, Salamanca. Proceedings of the 30th Annual ACM Symposium on Applied Computing - SAC '15. v. 1. p. 935-940.

[14] BARDDAL, JEAN PAUL ; GOMES, HEITOR MURILO ; Enembreck, Fabrício . A Survey on Feature Drift Adaptation. In: IEEE 27th International Conference on Tools with Artificial Intelligence, 2015, Vietri sul Mare. Proceedings of the IEEE 27th International Conference on Tools with Artificial Intelligence. New York: IEEE Computer Society, 2015. v. 1. p. 1053-1060.

[15] 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: ACM, 2000. (KDD ’00), p. 71–80. ISBN 1-58113-233-6. Disponível em: < http://doi.acm.org/10.1145/347090.347107>.

[16] Heitor Murilo Gomes, Jean Paul Barddal, Fabrício Enembreck, and Albert Bifet. 2017. A Survey on Ensemble Learning for Data Stream PIBIC 2023/24 – Classification. ACM Comput. Surv. 50, 2, Article 23 (March 2017), 36 pages. DOI: <https://doi.org/10.1145/3054925>

[17] Fernandes, Guilherme & Lulio, Luciano. AdaBoost. Disponível em: < https://www.academia.edu/download/30703269/ApresentacaoAdaBoost1.pdf>

[18] Sergio Ramírez-Gallego, Bartosz Krawczyk, Salvador García, Michał Woźniak, Francisco Herrera, A survey on data preprocessing for data stream mining: Current status and future directions, Neurocomputing, Volume 239, 2017, Pages 39-57, ISSN 0925-2312, https://doi.org/10.1016/j.neucom.2017.01.078.

[19] Galan, Magdiel & Liu, Huan & Torkkola, Kari. (2005). Intelligent instance selection of data streams for smart sensor applications. 10.1117/12.605855.

[20] Byczkowska-Lipińska, Liliana. (2017). Instance Selection Techniques in Reduction of Data Streams Derived from Medical Devices. PRZEGLĄD ELEKTROTECHNICZNY. 1. 117-120. 10.15199/48.2017.12.29.

[21] Sarac, Ferdi & Seker, Huseyin. (2016). An instance selection framework for mining data streams to predict antibody-feature function relationships on RV144 HIV vaccine recipients. 003356-003361. 10.1109/SMC.2016.7844752.

[22] Breiman, L. Bagging predictors. Mach Learn 24, 123–140 (1996). https://doi.org/10.1007/BF00058655

[23] Breiman, L. Bagging predictors. Mach Learn 24, 123–140 (1996). https://doi.org/10.1007/BF00058655

[24] Salzberg, S.L. C4.5: Programs for Machine Learning by J. Ross Quinlan. Morgan Kaufmann Publishers, Inc., 1993. Mach Learn 16, 235–240 (1994). https://doi.org/10.1007/BF00993309