LUCAS LOEZER

MINERAÇÃO DE FLUXOS DE DADOS DESBALANCEADOS: UMA ABORDAGEM BASEADA EM CUSTOS DE CLASSIFICAÇÃO

Curitiba - PR, Brasil

2021

LUCAS LOEZER

MINERAÇÃO DE FLUXOS DE DADOS DESBALANCEADOS: UMA ABORDAGEM BASEADA EM CUSTOS DE CLASSIFICAÇÃO

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

Pontifícia Universidade Católica do Paraná - PUCPR

Programa de Pós-Graduação em Informática - PPGIa

Orientador: FABRÍCIO ENEMBRECK

Curitiba - PR, Brasil

2021

Este trabalho é dedicado às crianças adultas que, quando pequenas, assistiam e se encantavam com os episódios de "O mundo de Beakman", e sonharam em se tornar cientistas.

Agradecimentos

Este trabalho exigiu muito esforço, dedicação e paracetamol para lidar com as dores de cabeça resultantes de esforços prolongados. Cerca de 25kg de café especial foram consumidos durante as horas dedicas à este projeto. Por essa razão, meus primeiros agradecimentos vão para os produtores de café especial que colaboraram de maneira indireta e possibilitaram que eu escolhesse passar café ao invés de passar raiva. Agradeço e muito aos membros do grupo Wakaba Yosakoi Soran, os quais me ofereceram um espaço para me exercitar, socializar, aprender, sorrir, chorar e dançar muito. Sem o apoio de vocês, eu provavelmente teria entrado em depressão e desistido deste trabalho. Devo agradecimentos à minha mãe e minha irmã! Em especial a minha mãe, que até hoje não entende o que eu faço, mas comemora, me apoia e larga um sorriso a cada conquista realizada. A minha melhor amiga e futura companheira de vida, Naomi, que ouviu minhas lamentações e sempre esteve presente para me ajudar. Às amizades construídas ao longo da graduação que me proporcionaram encontros felizes e construtivos. Agradeço também os professores que me orientaram ao longo deste trabalho e ao meu orientador. Por fim, agradeço a mim mesmo, ao Lucas do passado que resolveu iniciar o mestrado para aprofundar seus conhecimentos e dar orgulho ao Lucas do presente. Quanto ao Lucas do futuro, espero que ele esteja orgulhoso do que tenho conquistado até lá.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) - Código de Financiamento 001.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) - Finance Code 001.

"É enfrentando as grandes dificuldades que a humanidade consegue alcançar com mais êxito os grandes objetivos. É do perigo e da insegurança que surge a força que impulsiona a humanidade para novas e grandiosas conquistas."

O Fim da Eternidade - Isaac Asimov

Resumo

Dentre os desafios presentes na classificação em *streams* de dados, reside o desbalanceamento. Esse problema ocorre quando a distribuição de classes é desequilibrada e uma das classes possui significativamente mais exemplos do que outras classes do mesmo problema. Em razão disso, os algoritmos tradicionais de classificação são incapazes de construir modelos que apresentem boa performance em ambas as classes, sendo comum que os modelos construídos classifiquem bem a classe majoritária e apresentem dificuldades ao lidarem com as classes minoritárias, uma vez que trabalham sob a suposição de um problema de classificação balanceado. Ao longo dos anos, a academia apresentou uma variedade de estratégias para minimizar o impacto do desbalanceamento. Nesse trabalho, utiliza-se uma abordagem que emprega custos de classificação durante o estágio de decisão de um classificador. A essência de utilizar uma abordagem sensível a custos em problemas que apresentam desbalanceamento, é fazer com que a decisão de um algoritmo indutor para uma determinada classe, mesmo que outras sejam mais prováveis, possa ser uma ótima decisão. O objetivo do presente trabalho é tornar o algoritmo *Adaptive Random Forest* (ARF) disposto a trabalhar com *streams* de dados desbalanceados, incorporando em sua estrutura estratégias baseadas em custos. Neste trabalho, o método para classificação de *streams* de dados desbalanceadas *Cost-sensitive Adaptive Random Forest* (CSARF) é apresentado. O método emprega mudanças nas características do ARF e induz sensibilidade a custos de classificação utilizando três estratégias. Estudos empíricos com o método evidenciam um ganho na performance de classificação em instâncias minoritárias, com resultados competitivos ao estado da arte, em *streams* reais e sintéticas com diferentes níveis de desbalanceamento.

Palavras-chave: Mineração de fluxos contínuos de dados, desbalanceamento, árvore de decisão, combinação de classificadores, custos de classificação.

Abstract

Among the challenges present in the classification in data streams lies the class imbalance. This problem occurs when the class distribution is imbalanced and one of the classes has significantly more examples than other classes of the same problem. Because of this, traditional classification algorithms are unable to construct models that perform well in both classes. It is common for the constructed models to classify the majority class well and present difficulties in dealing with the minority classes, since they work under the assumption of a balanced classification problem. Over the years, researchers have presented a variety of strategies to minimize the impact of imbalanced data. In this work, the cost-sensitive approach that employs misclassification costs during the decision-making stage of a classifier is used. The essence of using a cost-sensitive approach to imbalanced problems is to make the decision of an inducer algorithm for a given class, even if others are more likely, to be a good decision. The goal of the present work is to make the algorithm Adaptive Random Forest (ARF) prone to work with imbalanced data streams, incorporating in its structure strategies based on misclassification costs. In this work, the method for classifying imbalanced data streams Cost-sensitive Adaptive Random Forest (CSARF) is presented. The method employs changes in ARF individuals and induces sensitivity to misclassification costs using three different strategies. Empirical studies with the method evidenced a gain in classification performance in minority instances, presenting state-of-art competitive results, in real and synthetic data streams with varying levels of imbalance.

Keywords: Data stream mining, class imbalanced, decision tree, ensemble of classifiers, misclassification costs.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AdaBoost	Adaptive Boosting
AdaMEC	Adaptive Boosting with Minimum Expected Cost
ADASYN	Adaptive Synthetic Sampling
AD	Árvore de Decisão
AM	Aprendizagem de Máquina
ARF	Adaptive Random Forest
ARF_{RE}	Adaptive Random Forests with Resampling
AsymADA	Asymmetric AdaBoost
AUC-PR	Area under the Precision-Recall Curve
AUROC	Area under the Receive Operating Characteristics curve
BRF	Balanced Random Forest
CDS	Concept Drift with SMOTE
CSARF	Cost-sensitive Adaptive Random Forest
CSDBN-DE	Cost-Sensitive Deep Belief Network with Differential Evolution
ENN	Edited Nearest Neighborhood
FIFO	First in, First out
FILO	First in, Last out
HT	Hoeffding Tree
IDSL	Imbalance Data Stream Learner
KNN	K-nearest Neighbors
KUE	Kappa Updated Ensemble
MCC	Matthews Correlation Coefficient
MOA	Massive Online Analysis

MWEL	Multi-Windows Based Ensemble Learning
MWMOTE	Majority Weighted Minority Oversampling TEchnique
NCL	Neighborhood Cleaning Rule
NIE	Nonstationary and Imbalanced Environments
NSE	Nonstationary Environments
OACLNIS	Online Adaptive Cost-sensitive one-Layer Neural network for Imbalanced data streams
OB	Online Bagging
OCLNIS	Online Cost-sensitive one-Layer Neural network for Imbalanced data streams
OOB	Oversampling-based Online Bagging
OSS	One-sided Selection
PR	Precision-Recall
RF	Random Forest
ROC	Receive Operating Characteristics curve
RS	Exemplos de risco
SERA	Selectively Recursive Approach
SMOTE	Synthetic Minority Over-sampling Technique
TK	Tomek Links
UOB	Undersampling-based Online Bagging
VDM	Value Difference Metric

Lista de símbolos

S	Fluxo contínuo de dados
S_{min}	Classe minoritária
S_{maj}	Classe majoritária
f_j	Atributo referente ao índice j
y_i	Classe referente ao índice i
x_i	Instância ou exemplo referente ao índice i de um determinado fluxo/base de dados
w	Janela deslizante
w_{size}	Tamanho da janela deslizante w
x^t	Vetor de atributos de uma determinada instância que está disponível no tempo t
Z	Tamanho de um subconjunto de dados
X_{new}	Instância minoritária sintética
X_i	Exemplo pertencente a classe minoritária
$cost$	Matriz de custos
$\dot{p}$	Limite de decisão (*threshold*) considerando as penalidades de classificar uma classe em outra

Sumário

1 Introdução

A produção de dados vem crescendo e se desenvolvendo conforme acompanha a evolução tecnológica e científica. Fatores como o crescimento contínuo do uso da internet, da popularização de smartphones e a queda no preço em produtos tecnológicos que criam, capturam, protegem ou armazenam dados, colaboram com a expansão da quantidade de dados (GANTZ; REINSEL, 2012). Segundo Gantz e Reinsel, a adoção de técnicas e práticas para processar esses dados possibilita a tomada de decisões inteligentes.

De forma a acompanhar o acelerado desenvolvimento de dados, técnicas de Aprendizagem de Máquina (AM) podem ser utilizadas devido a sua rápida identificação de padrões de comportamento. A aplicação dessas técnicas provaram-se de grande valor em diversos domínios de aplicações. Os modelos construídos por meio de AM, além de serem eficientes, são úteis em problemas de classificação em que o ser humano pode não ter conhecimento suficiente para desenvolver um algoritmo com boa performance (por exemplo, como no reconhecimento de faces a partir de imagens) (JORDAN; MITCHELL, 2015). Além disso, o uso de AM possibilita diminuir a chance de um resultado conter viés de um analista que está propenso a cometer erros (KOTSIANTIS; ZAHARAKIS; PINTELAS, 2007).

As técnicas de AM podem ser divididas em duas categorias: supervisionadas e não supervisionadas. No aprendizado não supervisionado, o modelo recebe e analisa um conjunto de dados não rotulados e os categoriza utilizando técnicas de agrupamento ou redução de dimensionalidade (JORDAN; MITCHELL, 2015). O aprendizado supervisionado consiste em treinar um modelo preditivo, também conhecido como algoritmo indutor, em dados rotulados para que ele possa realizar predições em dados não rotulados. Se o rótulo a ser previsto for contínuo, configura-se como uma tarefa de regressão. Caso o rótulo seja discreto, o problema em questão é de classificação. Formalmente, o indutor recebe uma base de dados S que contém m exemplos. Define-se $S = (x_i, y_i)$, $i = 1, \ldots, m$, em que $x_i \in X$ é um exemplo no espaço de características n-dimensional $X = f_1, f_2, \ldots, f_n$ e $y_i \in Y = 1, ..., C$ é o rótulo que classifica o exemplo x_i (HE; GARCIA, 2009).

A fim de processar uma grande base de dados, enfrenta-se o desafio de carregá-la na memória do computador. Esse problema agrava-se ainda mais quando há um limite físico de *hardware* que possa impedir o processamento dessa base. Para lidar com isso, os algoritmos tradicionais de AM, nomeados aqui como *batch*, processam individualmente subdivisões que compõe uma base de dados grande. Porém, quando o conjunto de dados tende a ser um fluxo potencialmente infinito de dados, os algoritmos *batch* apresentam a limitação de não se manterem atualizados conforme novas instâncias chegam. O paradigma de *data streams* surgiu para superar essa limitação (BIFET et al., 2010). Esse paradigma

fundamenta-se na ideia de processar bases de dados maiores que a memória do computador ao interpretá-las como um fluxo contínuo de dados. Não só isso, como também lidar com as características intrísecas dos fluxos de dados, em que, muitas vezes, dados antigos são menos relevantes que dados recentes. O fundamento é que um algoritmo indutor deve ser capaz de inspecionar cada instância uma única vez e depois descartá-la para não ocupar memória. Deve ser capaz de manter seu modelo atualizado de maneira incremental a cada novo exemplo processado. E por conseguinte, o algoritmo indutor deverá ser utilizável a qualquer momento (BIFET et al., 2010).

Devido a variabilidade da natureza dos dados, há fatores que dificultam a tarefa de classificação tanto no cenário de *data streams* quanto no cenário *batch*. Esses fatores incluem problemas relacionados à complexidade, separabilidade, dimensionalidade e desbalanceamento dos dados.

O foco deste trabalho está relacionado com o problema de classificação de *streams* de dados desbalanceadas. O desbalanceamento é caracterizado quando uma classe tem uma quantidade de exemplos significativamente maior do que outra classe. Dessa maneira, é comum que os algoritmos indutores classifiquem corretamente exemplos da classe com maior representatividade e, apresentem dificuldades em compreender as demais classes (GONG; KIM, 2017). Ao processarmos fluxos de dados desbalanceados, a tarefa de classificação se torna ainda mais difícil pois não há como analisarmos o conjunto de dados por completo. Somente os dados do passado podem ser levados em conta e não há garantia de que o comportamento desses contribuam de forma efetiva para o problema. Além das dificuldades citadas, *streams* de dados apresentam mudanças de conceito que podem variar a proporção da distribuição das classes ao longo do tempo (KRAWCZYK, 2016).

Neste trabalho, o método *Cost-sensitive Adaptive Random Forest* (CSARF) é apresentado como uma solução para amenizar o impacto do desbalanceamento. Esse algoritmo foi desenvolvido atribuindo-se sensibilidade a custos de classificação ao *Adaptive Random Forest* (ARF) (GOMES et al., 2017b) para classificação de fluxos contínuos de dados. O algoritmo apresentado supera a performance do ARF em *streams* de dados desbalanceadas em testes preliminares realizados.

1.1 Objetivos

O objetivo principal deste trabalho é tornar o algoritmo *Adaptive Random Forest* disposto a trabalhar com *streams* de dados desbalanceadas. Para cumprir esse objetivo principal, os seguintes objetivos específicos foram definidos:

1. Alterar as particularidades da estrutura do ARF para ambientes desbalanceados.
2. Definir estratégias para atribuição de sensibilidade a custos de classificação.

3. Definir heurísticas para o cálculo de custos de classificação.

4. Comparar o método proposto com o seu método de origem e algoritmos do estado da arte.

O objetivo 1 tem como finalidade realizar alterações na estrutura do ARF para o ambiente de dados desbalanceados. O objetivo 2 diz respeito ao desenvolvimento de um *framework* que aplique penalidades de classificação ao ARF utilizando diferentes estratégias. Seguido pelo objetivo 3, o qual tem relação com a pesquisa, codificação e definição de heurísticas para o cálculo de custos de classificação. Por fim, no objetivo 4 definiu-se que uma avaliação empírica seja realizada.

1.2 Motivação

Os estudos de diversas estratégias para amenizar o impacto de dados desbalanceados nas tarefas de AM estão cada vez mais frequentes e relevantes. A necessidade de resolver um problema tão recorrente como o desbalanceamento — importante para a solução de vários problemas reais, como evitar desastres ambientais e a entender a evolução de doenças (TOPOUZELIS, 2008; KRAWCZYK et al., 2016) — despertou o interesse da academia. A maioria das soluções existentes na literatura são oriundas do ambiente *batch*. No entanto, ao trabalharmos com fluxos de dados potencialmente infinitos e desbalanceados, tais técnicas devem ser adaptadas para suprir as limitações e funcionar corretamente no ambiente de *streams*.

A motivação do presente trabalho é tentar minimizar o impacto do desbalanceamento nos problemas de classificação em *stream* de dados. Para isso, adota-se a estratégia de adaptar um algoritmo com resultados competitivos ao estado da arte, para que assim, o modelo consiga trabalhar com fluxos de dados desbalanceados.

1.3 Hipótese

***Hipótese**. O Cost-sensitive Adaptive Random Forest é capaz de melhorar a performance de classificação na classe minoritária em data streams desbalanceadas empregando custos de classificação, sem que a performance da classe majoritária seja prejudicada de forma significativa.*

A hipótese é fundamentada na ideia de que ao se aplicar penalidades de se classificar uma classe em outra, o algoritmo esteja propenso a realizar decisões ponderadas tendendo a uma melhora na performance de classificação de exemplos da classe minoritária.

1.4 Resultados esperados e contribuição

Espera-se que o presente trabalho contribua para o entendimento do problema de *data streams* desbalanceadas, assim como forneça um algoritmo capaz de diminuir o impacto do desbalanceamento na tarefa de classificação. Por meio de análises empíricas e testes estatísticos, evidencia-se que o método proposto aumenta a performance de classificação em *streams* reais.

1.5 Organização

Esse trabalho está organizado da seguinte maneira: o Capítulo 2 apresenta detalhadamente o assunto sobre mineração de fluxo de dados e discute o problema do desbalanceamento; assim como traz em detalhes as abordagens utilizadas em trabalhos anteriores para mitigar o desbalanceamento no ambiente *batch* e *stream*, e descreve as métricas adequadas para avaliar a performance de modelos preditivos na presença do desbalanceamento. O Capítulo 3 detalha o método de proposto, apresentando um estudo exploratório de suas características e discussões sobre os pontos fortes e fracos do método. O Capítulo 4 apresenta os resultados preliminares a partir de uma série de experimentos conduzidos com o protótipo do método proposto. Nesse capítulo detalha-se o protocolo experimental seguido das discussões sobre os resultados obtidos. Por fim, no Capítulo 5 apresenta-se as conclusões preliminares deste trabalho.

2 Fundamentação Teórica

A tarefa de classificação consiste em prever um valor nominal de um exemplo composto por um vetor de atributos. Essa definição se aplica tanto para o ambiente *batch* quanto para o ambiente de *data streams*. Porém, durante a etapa de treinamento em *batch*, todos os exemplos do conjunto de dados estão disponíveis para o algoritmo indutor. Em contrapartida, no ambiente de *data streams*, os exemplos são apresentados de forma individual compondo um fluxo contínuo de dados.

Os classificadores oriundos de um ambiente de *streams* devem ser capazes de processar um volume de dados potencialmente infinito. Algumas técnicas do ambiente *batch* foram adaptadas para *data streams*, como o *Online Bagging* (OB) (OZA, 2005) procedente do *Bagging* (BREIMAN, 1996), evidenciando que é possível reaproveitar estratégias já existentes. Para isso, nas seções seguintes apresentam-se os conceitos fundamentais sobre desbalanceamento, em ambientes *batch* e *stream*, e os fundamentos da mineração de *stream* de dados. As diferentes estratégias apresentadas no presente capítulo colaboraram para a criação do método proposto.

A fundamentação teórica deste trabalho está organizada da seguinte forma: as seções 2.1 e 2.2 trazem os conceitos sobre mineração de fluxo contínuo de dados e desbalanceamento, respectivamente. Em seguida, as sub-seções 2.2.1, 2.2.2, 2.2.3, 2.2.4 apresentam abordagens para tratamento de desbalanceamento, sendo as duas últimas dedicadas para técnicas baseadas em custos e ensemble de classificadores. Logo após, as seções 2.3, 2.4 e 2.5 apresentam as métricas para avaliação de desempenho, os procedimentos de avaliação e por fim as bases de dados utilizadas.

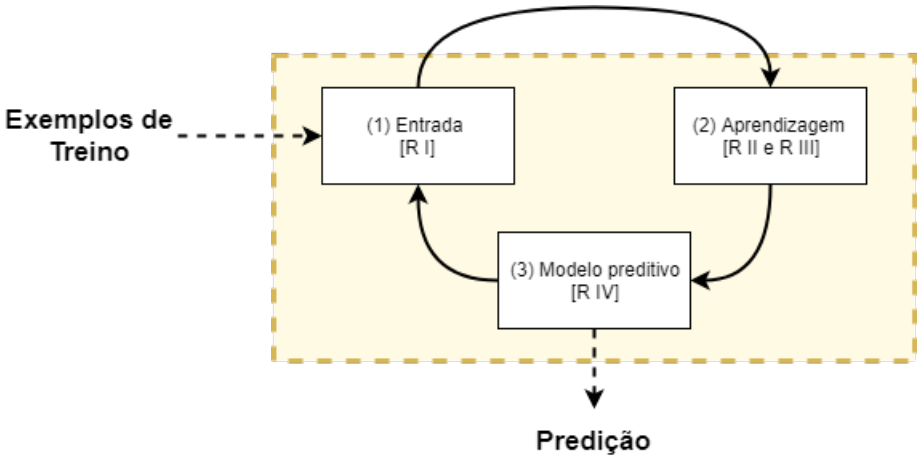
2.1 Mineração de fluxo contínuo de dados

Diferentes abordagens de Mineração de Dados e AM vêm sendo utilizadas para lidar com a grande quantidade de dados produzidos por dispositivos móveis, redes sociais, transações bancárias e outras aplicações de larga escala. O objetivo em questão é extrair conhecimento útil e aplicável a partir desses dados. Tarefas como escoragem de crédito (LAWI; AZIZ; SYARIF, 2017) e detecção de transações fraudulentas (DAL POZZOLO et al., 2015) são resolvidas por métodos de AM diariamente. Para isso, costuma-se extrair uma amostra desses dados, que por sua vez é pré-processada e utilizada para construir um modelo preditivo. Uma das desvantagens presentes nesse procedimento, resume-se na reexecução frequente do mesmo para manter o modelo preditivo atualizado. Em um problema de escoragem de crédito, por exemplo, os atributos que caracterizam um bom ou mal pagador costumam apresentar mudanças ao longo do tempo. Um modelo preditivo

do tipo *batch* não acompanha essa variação e com isso acaba degradando, sendo necessário uma nova operação para reconstrução de um novo modelo em amostras de dados mais recentes. Uma das vantagens de se utilizar um algoritmo indutor oriundo do ambiente de *stream* de dados, sendo adaptativo e incremental, é que ele pode ser atualizado sem que seja necessária a criação do modelo a partir do zero.

A massiva e rápida produção de dados, que pode ser processada como fluxo de dados ou *data streams*, vem se popularizando na comunidade científica. O pipeline de construção de modelos preditivos descrito no parágrafo anterior, consiste no modelo *batch*, o qual pode representar uma amostra de dados retirada de uma *stream* em um intervalo de tempo. Segundo (BIFET et al., 2010), as particularidades presentes na mineração de *streams* de dados que diferem do ambiente de aprendizado em *batch*, requerem que um modelo indutor seja capaz de:

Figura 1 – Ciclo de classificação em *streams*.



Adaptado de (BIFET et al., 2010).

I. Processar um exemplo e analisá-lo uma única vez (na maioria dos casos).

II. Utilizar uma quantidade limitada de memória.

III. Processar uma instância em uma quantidade limitada de tempo.

IV. Estar pronto para prever a qualquer momento.

O processo de aprendizagem e utilização de um modelo preditivo em um *data stream*, pode ser resumido a partir da Figura 1. A etapa 1 ilustra o algoritmo recebendo uma instância disponível do fluxo de dados (Requisito I). Na etapa 2 o indutor processa a instância, o mais rápido possível (Requisito III), de forma a atualizar as estruturas do

modelo sem que o limite de memória seja ultrapassado (Requisito II). E por fim, na etapa 4, o algoritmo está pronto para receber uma nova instância e, se necessário, predizer a classe de exemplos não rotulados (Requisito IV).

Há modelos preditivos que consomem as instâncias uma a uma conforme elas se tornam disponíveis. No entanto, existem classificadores que utilizam janelas (*batches*) de tamanho w_{size} para armazenar temporariamente os exemplos e em seguida atualizar o modelo. Essas janelas podem ser categorizadas como *Slinding window*, *Damped window* e *Adaptive window* (GOMES et al., 2017a). Uma *Slinding window* é semelhante a uma *batch* retirada de um *data stream* em um tempo t, porém os exemplos menos recentes da janela são substituídos um a um à medida que exemplos novos ficam disponíveis. A *Damped window* pondera as instâncias recentes com maiores pesos, fazendo com que o algoritmo de classificação foque nas instâncias mais novas. E por fim, a *Adaptive window* consiste em uma janela comum que tem um w_{size} variável de acordo com as heurísticas definidas para aumentar ou diminuir o tamanho da janela.

Formalmente, define-se que um fluxo contínuo de dados S, teoricamente infinito, é composto por exemplos x^t, em que x^t representa o vetor de atributos de uma determinada instância que chegou no tempo t. Portanto, a tarefa de classificação fundamenta-se em construir um modelo capaz de categorizar uma instância x^t em uma determinada classe (GOMES et al., 2017a).

2.1.1 Mudança de conceito

Eventos oriundos do mundo real costumam apresentar mudanças de conceito, fenômeno conhecido como *Concept Drift*. Um exemplo disso é o mercado de varejo, o qual costuma exibir diferentes volumes de venda em determinadas épocas do ano. O conceito relacionado com os dados produzidos, deriva ao longo do tempo (WANG; ABRAHAM, 2015). Outros exemplos comuns de mudança de conceito são: posicionamento político, mercado financeiro, assuntos populares em redes sociais, dentre outros.

Os *data streams* gerados pelos eventos reais refletem as mesmas características de mudança de conceito. O *concept drift* pode ser caracterizado por ocorrer em diferentes velocidades, teores de gravidade, ciclos temporais, etc (WANG; MINKU; YAO, 2018). De acordo com a velocidade, uma mudança de conceito pode ocorrer de forma abrupta, mudando rapidamente a distribuição dos dados no espaço de características. Também pode ser gradual, quando as instâncias de um presente conceito tornam-se menos frequentes enquanto que, exemplos do novo conceito tornam-se predominantes. O estado extremo da mudança gradual, em que o conceito evolui lentamente ao longo do tempo, pode ser caracterizada como incremental. E por fim, temos a mudança recorrente, cuja aparição é caracterizada por ciclos temporais.

A seção 2.1.2 irá abordar um dos algoritmos que compõe o estado da arte na mineração de *data streams*. Esse algoritmo será utilizado como base para a solução proposta nesse projeto.

2.1.2 Adaptive Random Forest

O *Random Forest* (RF), proposto por (BREIMAN, 2001), tornou-se um dos algoritmos de aprendizagem de máquina mais populares graças ao seu desempenho em tarefas de classificação e regressão. A proposta de Breiman consiste em combinar os votos de um conjunto de classificadores do tipo Árvore de Decisão (AD). Durante o processo de aprendizagem, cada AD é treinada com um subconjunto de dados gerado pelo *Bagging* (BREIMAN, 1996) e, além disso, um subconjunto de atributos aleatórios é selecionado para realizar a divisão dos nós das árvores. O uso da AD como algoritmo base foi estratégico, pois essa família de indutores é instável em relação aos dados, isto é, a estrutura da árvore varia de acordo com o conjunto de dados e atributos apresentados a ela. Com isso, Breiman conseguiu construir um conjunto de classificadores homogêneos e diversificados.

Ao aplicarmos o RF em fluxos potencialmente infinitos de dados, surgem algumas limitações. No processo de aprendizagem do RF, é necessário passar pelo conjunto de dados diversas vezes para criar os subconjuntos e treinar os classificadores base. Porém, isso se torna inviável em um ambiente *streaming*, em que o tempo para realizar tal tarefa pode ser maior do que o tempo de chegada de novos exemplos a classificar. Outrossim, a seleção do melhor atributo para a divisão do nó requer também várias iterações com os dados. Para superar essas duas limitações e possibilitar o uso do RF em um ambiente *streaming*, os autores em (GOMES et al., 2017b) propuseram o *Adaptive Random Forest* (ARF).

A primeira limitação, que seria utilizar o método *Bagging* em *data streams*, foi contornada pela abordagem *Online Bagging* proposta em (OZA, 2005). A técnica *Bagging* fundamenta-se em treinar n modelos em subconjuntos de tamanho Z. Os subconjuntos possuem exemplos selecionados aleatoriamente, com reposição, do conjunto original de treino. Essa estratégia permite construir subconjuntos que possuem uma instância do conjunto original N vezes, aonde $P(N = n)$ tende a uma distribuição binomial. Quanto maior o valor de Z, mais a distribuição binomial se aproxima de uma distribuição de $Poisson(\lambda = 1)$. Fundamentado nisso, o Oza propôs uma solução equivalente à estratégia do *Bagging*, atribuindo pesos às instâncias de acordo com a distribuição de $Poisson(\lambda = 1)$. O ARF utiliza o *Online Bagging* para simular a reamostragem resultante da técnica de Breiman, porém os pesos são atribuídos seguindo a distribuição de $Poisson(\lambda = 6)$. Segundo (BIFET; HOLMES; PFAHRINGER, 2010), isso permite aproveitar a reamostragem além de aumentar a probabilidade de atribuir pesos maiores às instâncias durante o treinamento dos modelos.

A segunda limitação é resolvida por meio de uma modificação no indutor base. Assim sendo, o algoritmo delimita um subconjunto aleatório de tamanho f, em que $f < F$ e F é o conjunto total de atributos, para realizar as subdivisões dos nós da AD durante o treinamento.

Algorithm 1 RFTreeTrain. **Symbols**: λ: Parâmetro fixo para distribuição de *Poisson*. GP: *Grace Period* antes de recalcular as heurísticas para dividir um nó.

```
1:  function RFTreeTrain(m, t, x, y)
2:      k ← Poisson(λ = 6)
3:      if k > 0 then
4:          l ← FindLeaf(t, x)
5:          UpdateLeafCounts(l, x, k)
6:          if InstancesSeen(l) ≥ GP then
7:              AttemptSplit(l)
8:              if DidSplit(l) then
9:                  CreateChildren(l, m)
10:             end if
11:         end if
12:     end if
13: end function
```

Fonte: Reprodução de (GOMES et al., 2017b)

O ARF pode ser dividido em dois módulos: um de aprendizagem e o responsável por gerenciar o conjunto de árvores. O primeiro módulo (Algoritmo *RFTreeTrain* 1) resume-se a uma versão modificada do indutor incremental *Hoeffding Tree* (HT). A HT baseia-se na ideia de que é possível decidir a partir de um pequeno conjunto de dados, calculando os limites de *Hoeffding*, quando e a partir de qual atributo uma divisão na folha deve ser feita (DOMINGOS; HULTEN, 2000). Isso permite que o indutor, além de aprender incrementalmente, construa uma estrutura inicial a partir de uma pequena quantidade de dados. As modificações realizadas na HT, presentes no *RFTreeTrain* são:

1. O *RFTreeTrain* não possui poda precoce da árvore.
2. As tentativas de divisão de um novo nó são realizadas somente em um subconjunto aleatório de atributos com tamanho fixo f. (Linha 7, Algoritmo 1)

O segundo módulo (Algoritmo *Adaptive Random Forest* 2) consiste no gerenciamento do conjunto de árvores. Para lidar com a mudança de conceito presente em *data streams*, o ARF dispõe do uso das seguintes estratégias: detecção e suspeita de mudança de conceito; treino de indutores adicionais para substituir as AD base quando houver mudança de conceito e; voto majoritário ponderado segundo o desempenho dos indutores base.

O algoritmo do ARF não está atrelado unicamente a um detector de mudança de conceito, pelo contrário, ele permite a utilização de diferentes detectores. Para isso, esses devem possuir as funções de detecção (Linha 15, Algoritmo 2) e suspeita de mudança (Linha 11, Algoritmo 2). Os parâmetros desses detectores são simplificados pelos autores para um limite permitido para detecção (δ_d) e suspeita (δ_w) de mudança de conceito. Cada algoritmo indutor base do ARF possui seu próprio detector de mudança e quando uma suspeita de mudança é sinalizada, ao invés de reinicializar a árvore que sinalizou a mudança, o ARF inicia o treinamento de uma árvore adicional utilizando somente os dados disponíveis após a sinalização da suspeita (Linhas 12-13, Algoritmo 2). No momento em que a mudança é de fato confirmada pelo indutor, o ARF o substitui pela árvore que estava sendo treinada a partir da suspeita sinalizada (Linha 16, Algoritmo 2). Ao receber um exemplo rotulado, cada árvore do conjunto classifica a instância, antes de treinar, e o ARF computa as métricas de desempenho de cada indutor (Linha 9, Algoritmo 2). A acurácia é utilizada de forma a atribuir pesos às árvores, fazendo com que o voto majoritário seja ponderado.

2.2 Dados desbalanceados

Nos dados do mundo real, pode ocorrer um fenômeno chamado desbalanceamento que é caracterizado quando um número de exemplos pertencente a uma classe supera em muito a quantidade de exemplos de outra classe. A classe minoritária, ou positiva, pode ser descrita como $S_{\text{min}} \in S$ e a classe majoritária como $S_{\text{maj}} \in S$, sendo que $S_{\text{min}} \bigcup S_{\text{maj}}$ representa o conjunto inteiro de dados. É importante ressaltar que o desbalanceamento pode ocorrer tanto em problemas binários, como descrito anteriormente, quanto em problemas multi-classe, em que podem haver múltiplas classes minoritárias e majoritárias. Esse fenômeno acontece em problemas como: detecção de fraude (BHATTACHARYYA et al., 2011), em que há uma pequena quantidade de transações fraudulentas em meio a uma enorme quantidade de transações honestas; detecção de derramamento de óleo (TOPOUZELIS, 2008), em que há poucos casos registrando o ocorrido; detecções de atividades raras ou pouco frequentes (GAO et al., 2016) e; classificação da malignidade do câncer (KRAWCZYK et al., 2016). Na maioria dos casos como os citados, S_{min} tem mais relevância que S_{maj}, porque são exemplos raros e que possuem um grande custo envolvido caso sejam classificados incorretamente.

Dados desbalanceados podem ser categorizados de acordo com as suas características. Se o desbalanceamento é naturalmente encontrado na definição do problema, como por exemplo a detecção de fraude, ele pode ser categorizado como intrínseco. Se o surgimento do fenômeno for abrupto e não originário do problema, ele é categorizado como extrínseco. Ademais, em tarefas de classificação em que uma instância pode pertencer a mais de uma classe (*Multi-label Classification*), algumas classes também podem ser mais frequentes do

Algorithm 2 Adaptive Random Forest. **Symbols**: m: Número máximo de atributos avaliados por divisão; n: Número total de árvores ($n = |T|$); δ_w: *Threshold* de suspeita; δ_d: *Threshold* de mudança; $c(\cdot)$: Detector de *concept drift*; S: *Data stream*; B: Conjunto de indutores adicionais; $W(t)$: Peso da árvore t; $P(\cdot)$: Função de estimação performance.

1: **function** AdaptiveRandomForest(m, n, δ_w, δ_d)
2: $T \leftarrow CreateTrees(n)$
3: $W \leftarrow InitWeights(n)$
4: $B \leftarrow \emptyset$
5: **while** $HasNext(S)$ **do**
6: $(x, y) \leftarrow next(S)$
7: **for all** $t \in T$ **do**
8: $\hat{y} \leftarrow predict(t, x)$
9: $W(t) \leftarrow P(W(t), \hat{y}, y)$
10: $RFTreeTrain(m, t, x, y)$ ▷ Treina t na instância corrente (x, y)
11: **if** $C(\delta_w, t, x, y)$ **then** ▷ Suspeita de *concept drift*?
12: $b \leftarrow CreateTree()$ ▷ Constrói indutor adicional
13: $B(t) \leftarrow b$
14: **end if**
15: **if** $C(\delta_d, t, x, y)$ **then** ▷ *Concept drift* detectado?
16: $t \leftarrow B(t)$
17: **end if**
18: **end for**
19: **for all** $b \in B$ **do** ▷ Treina os indutores adicionais
20: $RFTreeTrain(m, b, x, y)$
21: **end for**
22: **end while**
23: **end function**

Fonte: Reprodução de (GOMES et al., 2017b)

que outras, causando um desbalanceamento entre elas (*inter-class imbalance*). Além disso, pode ocorrer situações em que uma das classes possui um desbalanceamento entre as suas instâncias positivas e negativas, caracterizando assim, um desbalanceamento interno da classe (*inner-class imbalance*) (SPYROMITROS-XIOUFIS et al., 2011). Em ambientes de *data stream*, o fenômeno do desbalanceamento também ocorre, mas combinado com mudança de conceito. Isto é, o relacionamento entre S_{min} e S_{maj} pode variar ao longo do tempo e, novas classes e desbalanceamentos podem vir a surgir (KRAWCZYK et al., 2016).

A academia vem trabalhando para diminuir o impacto desse fenômeno presente na mineração de dados. As soluções existentes envolvem técnicas de amostragem (BATISTA; CARVALHO; MONARD, 2000; HULSE; KHOSHGOFTAAR; NAPOLITANO, 2007) que trabalham manipulando os dados antes de treinar o algoritmo, sendo o objetivo dessas técnicas diminuir o desequilíbrio entre S_{min} e S_{maj} por meio da criação ou replicação de exemplos da classe minoritária, *oversampling*, ou a remoção de instâncias da classe

majoritária, *undersampling*. Também há abordagens baseadas em custos (ZADROZNY; LANGFORD; ABE, 2003; GHAZIKHANI; MONSEFI; YAZDI, 2014) e *ensemble* de classificadores (POZZOLO et al., 2014; LICHTENWALTER; CHAWLA, 2009), visando diminuir a sobre-representação da classe majoritária.

As seguintes sub-seções 2.2.1 e 2.2.2 descrevem técnicas para o tratamento de desbalanceamento de dados utilizando amostragem em ambiente *batch* (os métodos de *oversampling* e *undersampling* para *data streams*, por estarem relacionados à combinação de classificadores, estão descritos na sub-seção 2.2.4). Logo após, a sub-seção 2.2.3 traz abordagens baseadas em custos. Em seguida, na sub-seção 2.2.4 serão descritas as abordagens baseadas em *ensemble* de classificadores para *batch* e *data streams*. E por fim, as seções 2.3, 2.4 e 2.5 abordam as métricas para avaliação de desempenho, o procedimento de avaliação e as *streams* de dados utilizadas, respectivamente.

2.2.1 Oversampling

As técnicas de *oversampling* têm como foco diminuir o desbalanceamento por meio da replicação de exemplos da classe minoritária. A abordagem mais simples consiste em selecionar aleatoriamente instâncias positivas e replicá-las, diminuindo assim, a proporção de exemplos majoritários para minoritários. Partindo da ideia de replicação, os autores (CHAWLA et al., 2002) desenvolveram uma técnica que ao invés de repetir instâncias já existentes, gera instâncias sintéticas vizinhas às instâncias minoritárias. O *Synthetic Minority Over-sampling Technique* (SMOTE) busca nas vizinhanças de cada instância positiva $X_i \in S_{\min}$ os K-vizinhos de mesma classe mais próximos. Com isso, o método seleciona uma quantia de exemplos aleatórios dos K-vizinhos, de acordo com a taxa de *oversampling*, e então, para cada vizinho selecionado ($\hat{X}_i$) calcula-se a distância para X_i e multiplica-se o vetor gerado por um número aleatório $\delta \in [0, 1]$, gerando assim uma instância minoritária sintética X_{new}, localizada entre X_i e o K-vizinho escolhido, conforme a Equação 2.1.

$$X_{\text{new}} = X_i + \delta(\hat{X}_i - X_i) \tag{2.1}$$

Uma das limitações do SMOTE, surge do fato do método não supor que entre $X_{\min}$ e o K-vizinho selecionado podem haver exemplos pertencentes à S_{maj}. Justamente por esse detalhe, as instâncias sintéticas podem ser geradas de forma a sobrepor as instâncias majoritárias já existentes.

Com base na proposta e nas limitações apresentadas pelo SMOTE, diversos métodos de *oversampling* sintéticos foram desenvolvidos. Dentre esses, o SMOTE-Bordeline-1 e SMOTE-Bordeline-2, propostos pelos autores (HAN; WANG; MAO, 2005), que geram instâncias sintéticas na região do espaço de classificação entre S_{maj} e $S_{\min}$ (*Borderline*).

Para isso, o SMOTE-Bordeline-1, diferentemente do tradicional SMOTE, seleciona os K-vizinhos m independente da classe de cada exemplo $X_i \in S_{\min}$ de tamanho j. O número de exemplos majoritários presentes em m é denominado m' $(0 \leq m' < m)$ e caso o $m' = m$, então X_i é considerado ruído pois todos os seus vizinhos são majoritários e os passos a seguir são desconsiderados. Se $(\frac{m}{2} \leq m' < m)$, o número de instâncias negativas vizinhas de X_i é maior que o número de instâncias positivas, então X_i é considerado um exemplo difícil de se classificar e é armazenado no grupo de exemplos de risco (RS). Caso $(0 \leq m' < \frac{m}{2})$, então a probabilidade de X_i ser classificado incorretamente é baixa e os passos a seguir são desconsiderados. As instâncias positivas presentes em RS são considerados como exemplos que pertencem a borda de $S_{\min}$, tal que $RS \in S_{\min}$ e $RS = x'1, x'2, ..., x'n$, com $0 \leq n < j$. Para cada exemplo x'_{i} presente em RS, seleciona-se aleatoriamente s vizinhos mais próximos dos K-vizinhos mais próximos que pertencem a $S_{\min}$. Com isso, calcula-se a diferença $dif_{\mathrm{h}}(h = 1, 2, .., s)$ entre x' e os seus s vizinhos mais próximos e então multiplica-se dif_{h} por um número aleatório $r_{\mathrm{h}}(h = 1, 2, ..., s)$ entre 0 e 1. Por fim, s novas instâncias minoritárias sintéticas são geradas entre x'_{i} e seus vizinhos mais próximos conforme a Equação 2.2, e isso é repetido para cada membro de RS.

$$X_{\text{sintético}} = x'_i + r_h \times dif_h(h = 1, 2, ..., s) \tag{2.2}$$

A segunda versão do SMOTE-Bordeline, além de seguir os mesmos passos citados anteriormente, não somente gera instâncias sintéticas para cada exemplo de RS e seus devidos vizinhos positivos mais próximos, como também gera exemplos para os vizinhos negativos mais próximos. A diferença entre o exemplo selecionado e o majoritário é multiplicada por um número aleatório entre 0 e 0.5, fazendo com que a instância gerada fique mais próxima do exemplo minoritário.

Há também outros algoritmos baseados em cálculo de distância, mas não baseados no SMOTE, que também lidam com o desbalanceamento. Dentre eles, o ADASYN (HE et al., 2008) que utiliza uma abordagem de atribuição de pesos de acordo com a dificuldade de classificação de uma instância minoritária. Isto é, mais exemplos sintéticos são gerados com base nos exemplos que um indutor tem dificuldade de compreender, do que os exemplos minoritários mais fáceis de classificar. O ADASYN gera uma quantidade G de novas instâncias sintéticas com base na Equação 2.3:

$$G = (m_l - m_s) \times \beta \tag{2.3}$$

Onde que m_l e m_s são os tamanhos dos conjuntos S_{maj} e $S_{\min}$ respectivamente, $\beta \in [0, 1]$ é o parâmetro para determinar o nível de balanceamento desejado após a geração de instâncias sintéticas, aonde $\beta = 1$ balanceia inteiramente o conjunto de dados. Para cada exemplo da classe minoritária, o ADASYN computa a taxa de exemplos majoritários

presentes nos K-vizinhos mais próximos. Essa taxa é normalizada de forma a se tornar uma distribuição de densidade (r'). E então, computa-se a quantidade de exemplos sintéticos g_i a serem gerados por instância minoritária (Equação 2.4). Por fim, para cada instância positiva X_i gera-se g_i exemplos sintéticos selecionando-se aleatoriamente g_i instâncias da vizinhança de X_i e aplicando a Equação 2.1.

$$g_i = r'_i \times G \tag{2.4}$$

O método *Majority Weighted Minority Oversampling TEchnique* (MWMOTE) proposto pelos autores (BARUA et al., 2012), surgiu da motivação de superar as limitações geradas pela técnica ADASYN. O parâmetro que define a quantidade de exemplos majoritários entre os K-vizinhos de X_i é inapropriado para atribuir pesos a instâncias minoritárias oriundas da borda de classificação. Além de ser insuficiente para distinguir quais instâncias minoritárias são mais importantes para o treinamento e pode favorecer a geração de ruídos sintéticos. Para resolver isso, o MWMOTE é composto por três etapas: 1) Seleção de um subconjunto de instâncias minoritárias apropriadas para geração de instâncias sintéticas; 2) Atribuição de pesos aos exemplos selecionados de acordo com a sua importância ao conceito do problema e; 3) Utilização de uma abordagem de agrupamento para gerar instâncias sintéticas da classe minoritária.

A primeira etapa do MWMOTE consiste em computar os K-vizinhos para cada instância $X_i \in S_{\text{min}}$. Após isso, um filtro é aplicado no conjunto de K-vizinhos para remover os exemplos positivos que possuem apenas vizinhos negativos em suas vizinhanças, gerando assim o S_{minf}. Para cada exemplo $x_i \in S_{\text{minf}}$, computa-se os vizinhos majoritários mais próximos $N_{\text{maj}}(x_i)$. E então combina-se todos os subconjuntos $N_{\text{maj}}(x_i)$, encontrando-se assim o conjunto de exemplos negativos S_{bmaj} oriundos da borda. Itera-se cada exemplo $b_i \in S_{\text{bmaj}}$, computando os vizinhos minoritários mais próximos $N_{\text{min}}(b_i)$. Dessa forma, constrói-se um conjunto de instâncias minoritárias informativas S_{imin} unindo-se todos os subconjuntos gerados por $N_{\text{min}}(b_i)$. O conjunto S_{imin} contém todas as instâncias com altas probabilidades de serem classificadas erroneamente.

Partindo-se para a segunda etapa do MWMOTE, os autores supõem que os exemplos contidos em S_{imin} possuem diferentes níveis de relevância, e então eles atribuem pesos diferentes para os exemplos de acordo com a sua importância. Quanto maior o peso, maior é o número de instâncias sintéticas que serão geradas perto do exemplo minoritário. O valor do peso de cada instância $S_w(x_i)$ é convertido para um valor de probabilidade de seleção $S_p(x_i)$, de acordo com a Equação 2.5:

$$S_p(x_i) = \frac{S_w(x_i)}{\sum_{z_i \in S_{\text{imin}}} S_w(z_i)} \tag{2.5}$$

Para evitar as limitações citadas anteriormente, a terceira etapa do MWMOTE

seleciona os agrupamentos $LM(L1, L2, \ldots, LM)$ contidos em $S_{\min}$. Por fim, de forma iterativa, seleciona-se um exemplo x_i de S_{imin} de acordo com a sua distribuição de probabilidade $\{S_p(x_i)\}$ e, o agrupamento L_k $(1 \leq k \leq M)$ a qual x_i pertence; a instância sintética é gerada a partir de Equação 2.1, em que $\hat{X}_i$ é um exemplo aleatório selecionado de Lk e δ um número aleatório que pertence ao intervalo [0,1].

2.2.2 Undersampling

Seguindo a ideia de balancear a proporção de instâncias positivas e negativas, os métodos categorizados como *undersampling* procuram eliminar instâncias da classe majoritária. A abordagem mais comum consiste em excluir instâncias escolhidas aleatoriamente de S_{maj}, sem se importar se tais exemplos são importantes ou não na construção de um classificador. Outras abordagens para a redução de exemplos ou ruídos foram desenvolvidas utilizando critérios para a seleção de instâncias. Dentre os métodos de undersampling inteligentes, um dos mais populares é o *Tomek Links* (TK) (TOMEK, 1976). O método exclui pares de instâncias de classes opostas que são classificadas como um TK. Para isso, um par de exemplos $(X_{\min}, X_{\text{maj}})$, sendo $X_{\min} \in S_{\min}$ e $X_{\text{maj}} \in S_{\text{maj}}$, é classificado como TK se não houver um exemplo X_k entre o par selecionado, i.e. a $distancia(X_{\min}, X_k) < distancia(X_{\min}, X_{\text{maj}})$ ou $distancia(X_{\text{maj}}, X_k) < distancia(X_{\min}, X_{\text{maj}})$. Se um par de instâncias é classificado como TK, ele é removido do conjunto de treinamento pois uma das instâncias é considerada um ruído ou o par é oriundo da borda de classificação. A remoção desses exemplos que se sobrepõem permite uma melhor definição dos agrupamentos de cada classe e isso possibilita que o classificador construa regras de classificação melhor definidas. No entanto, se o problema desbalanceado apresentar sobreposição de classes, o uso de TK pode aumentar o desbalanceamento pois removerá instâncias minoritárias.

Para remover instâncias que não possuem muita relevância para o aprendizado da classe majoritária, deve-se seguir algumas heurísticas que são divididas em quatro grupos:

1. Ruídos.
2. Exemplos redundantes.
3. Exemplos perto da borda de classificação (*borderlines*).
4. Casos seguros: nem perto e nem longe da borda de classificação.

Os métodos que removem somente instâncias da classe majoritária são classificados como *One-sided Selection* (OSS) e, têm como objetivo construir um conjunto de treino composto de casos seguros. Isto é, ruídos, redundância e *borderlines* são removidos. Uma derivação do TK foi proposta pelos autores (KUBAT; MATWIN et al., 1997), o qual consiste em selecionar um subconjunto de instâncias negativas representativas. Fazendo com que o

conjunto de treino fique mais balanceado e que as implicações resultantes do *Random Undersampling* sejam amenizadas. Para isso, o algoritmo constrói um subconjunto C, tal que $C \in S$, com todos os exemplos positivos e somente um exemplo negativo é selecionado aleatoriamente. E então, classifica-se S utilizando o *k-nearest neighbors* (KNN) ($k = 1$) nos exemplos contidos em C. Todas as instâncias de S classificadas incorretamente são movidas para C, o qual é tão consistente quanto S mas com um tamanho menor. E por fim, todas as instâncias majoritárias que formam um TK são removidas por serem *borderlines* ou ruídos, constituindo-se assim, um conjunto de treino T que segue as heurísticas citadas anteriormente. Outra solução proposta pelos autores (BATISTA; CARVALHO; MONARD, 2000), também segue os passos anteriores, porém ela utiliza a métrica *Value Difference Metric* (VDM) para calcular a distância entre os atributos simbólicos. A razão disso está na baixa precisão que as métricas possuem para calcular a distância entre atributos simbólicos, isto é, a distância tem valor 1 caso os atributos possuam o mesmo valor e 0 caso sejam diferentes.

Tal como o *One-sided Selection*, o método *Neighborhood Cleaning Rule* (NCL) remove instâncias da classe majoritária porém considera a qualidade dos dados a serem removidos (LAURIKKALA, 2001). O autor fundamenta-se na hipótese de que a qualidade dos resultados de classificação não está ligada a somente a proporção das classes. Por essa razão, as métricas para a limpeza dos dados são aplicadas antes da construção do subconjunto de treinamento. Para isso, o NCL divide os dados de treino T em dois subconjuntos, o da classe de interesse C e o resto dos dados O, em que $C \bigcup O = T$. A remoção de possíveis ruídos A_1 é realizada no subconjunto O utilizando a técnica *Edited Nearest Neighborhood* (ENN). Essa que elimina os exemplos negativos classificados incorretamente pelos seus três vizinhos mais próximos. Em seguida, os vizinhos das instâncias em C são computados e todos aqueles que pertencem a classe majoritária, são descartados para A_2. A união de A_1 e A_2 é removida de T, construindo-se assim o subconjunto reduzido T_n.

2.2.3 Abordagens baseadas em custos

Além da utilização de técnicas para diminuir a sobre-representação da classe majoritária, há também abordagens que consideram o custo envolvido ao se classificar uma instância erroneamente. Imagine que um dos atributos que caracterizam uma transação fraudulenta é o montante de dinheiro. Se uma instituição financeira não consegue detectar a fraude, ela terá um prejuízo de no mínimo o valor da transação envolvida. Para lidar com isso, é possível incorporar os custos de classificação incorreta na decisão de um classificador (ELKAN, 2001).

A atribuição de penalidades de se classificar uma classe em outra, pode ser utilizada para atribuir sensibilidade a custos em um classificador. Assim sendo, o modelo deverá

prever uma instância considerando os custos de classificação, onde a probabilidade condicional de cada classe e seu devido custo esperado, para predições corretas ou incorretas, influenciam na decisão final. Formalmente, seja $cost(i, j)$ o custo envolvido ao se classificar uma instância x em i quando a classe verdadeira é j. Se $i = j$ então a previsão é correta e o custo é zero, caso contrário, é incorreta e seu custo associado deve ser considerado. Para que o classificador induza uma classe considerando os custos, a melhor previsão para o exemplo x deve ser a classe i que minimiza a soma das probabilidades alternativas para a classe verdadeira de x (Equação 2.6).

$$L(x, i) = \sum_{j} P(j|x)cost(i, j) \tag{2.6}$$

A essência da abordagem sensível a custos é fazer com que a decisão de um indutor para uma classe y, mesmo que outras sejam mais prováveis, possa ser uma ótima decisão. E, por ser uma abordagem que destaca uma determinada classe, ela pode ser empregada para lidar com o desbalanceamento (MALOOF, 2003; LIU; ZHOU, 2006).

A matriz de custos é utilizada para representar as penalidades atribuídas a um classificador. Essa matriz (Tabela 1) possui a seguinte estrutura para um problema binário:

Tabela 1 – Matriz de Custos

	Negativo	Positvo
Previsto Negativo	C(0,0) ou TN	C(0,1) ou FN
Previsto Positivo	C(1,0) ou FP	C(1,1) ou TP

Em ambientes desbalanceados, a classe minoritária tende a ser denominada como classe positiva, enquanto que a majoritária é a classe negativa. Desta forma, o custo $C(0, 1)$ (FN) geralmente é maior do que o custo $C(1, 0)$ (FP). Partindo da hipótese de que o $C(0, 0)$ (TN) e $C(1, 1)$ (TP) são nulos, ao aplicarmos a equação 2.6, uma instância x será categorizada como positiva se e somente se:

$$P(0|x)C(1, 0) \leq P(1|x)C(0, 1) \tag{2.7}$$

A partir da Equação 2.7, e de que $P(0|x) = 1 - P(1|x)$, é possível obter um *threshold* $\dot{p}$ (Equação 2.8) para que um modelo classifique uma instância como positiva, se $P(1|x) \geq \dot{p}$, aonde:

$$\dot{p} = \frac{C(1, 0)}{C(1, 0) + C(0, 1)} = \frac{FP}{FP + FN} \tag{2.8}$$

Os algoritmos indutores que não possuem probabilidades como saída, tendem a prever uma classe com um threshold fixo $\dot{p} = 0.5$. Em (ELKAN, 2001), o autor evidencia

que ao balancearmos o conjunto de treino que foi utilizado para construir os indutores, fazendo amostragem com base no *threshold* estimado, obteremos um algoritmo indutor sensível a custos. Fundamentado nisso, os autores em (LING; SHENG, 2011) agrupam as soluções sensíveis a custo em duas categorias.

A primeira categoria agrupa os *frameworks* capazes de transformar um algoritmo indutor comum em um sensível a custos. Essa categoria engloba as soluções que utilizam o *threshold* para influenciar as decisões dos classificadores e, também as abordagens que empregam o *threshold* para balancear o conjunto de treinamento. Uma das técnicas mais populares para ensemble de classificadores, o *Adaptive Boosting* (AdaBoost) (FREUND; SCHAPIRE, 1997), motivou diversas variações do seu mecanismo de atribuição de pesos para o ambiente sensível a custos. O AdaBoost constrói um conjunto de classificadores individualmente fracos, mas quando combinados compõem um modelo robusto. Para isso, os classificadores base são treinados sucessivamente, de modo que o classificador M_t utilize um conjunto de treino ponderado pelas dificuldades do classificador $M_{\text{t-1}}$. Dessa forma, o modelo final é uma combinação de vários classificadores que focaram em resolver as dificuldades do seu anterior (FREUND; SCHAPIRE, 1997). Logo após, os autores em (SCHAPIRE; SINGER, 1999) atribuiram um coeficiente de confiança para cada classificador gerado. Fazendo com que a predição final (Equação 2.9) de um exemplo x seja dada pelo sinal da soma das previsões do indutor $h_t(x)$ ponderadas pelos seus coeficientes de confiança (α_t).

$$H(x) = sign\left[\sum_{t=1}^{M} \alpha_t h_t(x)\right] \tag{2.9}$$

A atribuição de pesos presente no funcionamento do AdaBoost motivou diversos pesquisadores a adaptar essa solução para o ambiente sensível a custos. Em (NIKOLAOU et al., 2016), os autores realizam um estudo comparativo de 15 versões do AdaBoost sensíveis a custo, em 18 bases de dados com 21 variações de grau de desbalanceamento. Dentre as versões comparadas, os autores evidenciaram que as variações *Adaptive Boosting with Minimum Expected Cost* (AdaMEC) e *Asymmetric AdaBoost* (AsymADA) são as versões mais consistentes com os *frameworks* de comparação utilizados. O AdaMEC (TING, 2000) assume que os votos ponderados são proporcionais às probabilidades de cada classe e, atribui os custos de classificação na predição final de um exemplo. Já o AsymADA (VIOLA; JONES, 2002), pondera as instâncias e busca minimizar uma função de perda assimétrica empregada, favorecendo assim a classe minoritária. Os autores concluíram que, uma vez que essas variações estejam definidamente calibradas, elas apresentam uma performance semelhante e desempenham melhor do que as outras variações estudadas na métrica *Area Under the Brier Curve*.

Além das variações do Adaboost, existem técnicas como o *Costing* (ZADROZNY;

LANGFORD; ABE, 2003) e *MetaCost* (DOMINGOS, 1999) que pertencem à categoria *sampling* e *threshold*, respectivamente. A primeira técnica realiza amostragens no conjunto de treinamento com base na probabilidade de uma instância ocorrer considerando seu custo de classificação. Já a segunda, atribui sensibilidade a custos a um indutor treinando-o em uma base de dados re-rotulada por ensemble de classificadores combinado com uma matriz de custos. A atribuição de pesos às instâncias pode ser considerada uma estratégia de *sampling*, pois instâncias com maiores custos são interpretadas como duplicação. A partir dessa ideia, o autor em (TING, 2002) induz árvores de decisão ponderando as instâncias, de maneira proporcional à base de dados, durante a etapa de aprendizagem.

A segunda categoria aborda os algoritmos que empregam a sensibilidade a custo diretamente em suas estruturas. Duas abordagens comuns são as árvores de decisão e redes neurais adaptadas. A AD pode utilizar os custos de três formas: utilizar o *threshold* $\dot{p}$ (Equação 2.8) para deslocar a decisão final do classificador; utilizar métricas sensíveis a custos como critério de decisão da divisão de um nó e; empregar sensibilidade a custos no mecanismo de poda da árvore. Em (ELKAN, 2001), o autor evidencia que a utilização de técnicas como Gini, Entropia e DKM para encontrar a melhor divisão do nó, apresentam maior sensibilidade aos custos do que utilizar a taxa de erro. Já em (ZHAO et al., 2015), os autores adaptaram o mecanismo de poda da árvore considerando a probabilidade da árvore pós-poda apresentar maiores custos de classificação.

Segundo os autores em (KUKAR; KONONENKO et al., 1998), a estrutura multi-camadas de uma rede neural permite instigar sensibilidade a custos de quatro maneiras: deslocar a estimativa de probabilidade utilizando o *threshold* $\dot{p}$ (Equação 2.8); tornar a saída da rede sensível a custos; empregar mecanismos de custos na taxa de aprendizagem e; adaptar a função de minimização de erro para contabilizar os custos esperados. Ao se aplicar custos na camada de saída de uma rede neural, o processo de aprendizagem prioriza a classe com os maiores custos de classificação. Estudos empíricos apresentados por Kukar e Kononenko demonstram que é possível aumentar a taxa de aprendizagem para os exemplos mais custosos fazendo com que eles impactem mais na mudança de pesos dos neurônios. A última abordagem fundamenta-se em substituir a função de minimização de erro por uma de minimização de custos. Os autores concluem que essa última abordagem apresentou melhores desempenhos do que as anteriores.

O *framework* (Equação 2.6) e seus fundamentos de atribuição de custos a um classificador são adaptáveis para o ambiente de *data streams*. Os autores em (GHAZIKHANI; MONSEFI; YAZDI, 2013) apresentam duas redes neurais sensíveis a custo: OCLNIS e OACLNIS. Ambas incorporaram penalidades de classificar uma classe em outra em sua função objetivo e, suas estruturas são compostas por apenas uma camada com vários neurônios. O OACLNIS ainda incorpora uma matriz de custos adaptativa capaz de mudar os pesos envolvidos quando uma mudança de conceito ocorre. Já em (GHAZIKHANI;

MONSEFI; YAZDI, 2014), uma função de minimização de erro que atribui altos custos aos exemplos positivos é utilizada. O CSDBN-DE, proposto em (ZHANG; TAN; REN, 2016), emprega uma matriz de custos e durante a etapa de treinamento busca otimizar os pesos atribuídos por meio de um algoritmo genético.

A próxima seção aborda técnicas de Ensemble de Classificadores. Serão listados não somente os métodos de combinação de classificadores mas também os que empregam técnicas de amostragem e abordagens baseadas em custos.

2.2.4 Ensemble de classificadores

A ideia de construir um ensemble de classificadores consiste em utilizar múltiplos classificadores fracos que quando combinados, podem formar um classificador robusto. Um classificador fraco pode ser descrito como um modelo induzido por um algoritmo capaz de performar melhor do que uma escolha aleatória. Quando diferentes modelos fracos são construídos a partir do mesmo conjunto de treino, a combinação deles consiste em analisar o mesmo problema com perspectivas diferentes. Isso possibilita que um classificador complemente a opinião do outro (POLIKAR, 2006). Nesta seção, seguindo a taxonomia apresentada pelos autores em (GOMES et al., 2017a), o termo "combinação" irá se referir tanto à arquitetura do ensemble como o método de voto utilizado. O termo "ensemble" vai se referir à maneira na qual os classificadores estão organizados, enquanto que o termo "voto" descreve como as predições dos classificadores são utilizadas para compor a decisão final.

Os métodos que geram algoritmos indutores para trabalhar com tarefas de classificação, trabalham lado a lado com o dilema viés-variância. Isto é, o modelo construído deve ser capaz de generalizar previsões para o futuro com base no conhecimento extraído do passado. Caso o indutor obtenha bons resultados no conjunto de treino mas não consiga generalizar o suficiente para o conjunto de teste, esse modelo possui um viés baixo e uma variância alta. É ideal que se encontre um meio termo entre o viés e a variância, de forma que o modelo consiga obter uma boa performance para os exemplos vistos ou não durante a etapa de treinamento. Técnicas como *Bagging* (BREIMAN, 1996), *Random Forest* (BREIMAN, 2001) e *Boosting* (SCHAPIRE; SINGER, 1999) geralmente obtém sucesso na redução da variância por meio da combinação de classificadores.

Quando um conjunto de dados é extremamente desbalanceado, os métodos de amostragem presentes em ensembles podem construir classificadores que nunca sequer viram um exemplo minoritário. Os autores em (CHEN et al., 2004), apresentam uma adaptação do *Random Forest* de modo a diminuir o impacto desse problema. O *Balanced Random Forest* (BRF) constrói cada AD utilizando um conjunto de treino balanceado composto por exemplos aleatórios de S_{maj} e, um conjunto de instâncias de S_{min}. Outra abordagem semelhante chamada *Easy Ensemble*, proposta em (LIU; WU; ZHOU, 2009), divide o

problema principal em sub-problemas balanceados. Cada subproblema é representado por um conjunto de dados da mesma forma que na BRF e, isso permite que cada classificador tenha conhecimento sobre diferentes aspectos do conjunto original. O classificador resultante da iteração do Adaboost em cada subproblema se torna um membro do *Easy Ensemble*. Em (CHAN; STOLFO, 1998) os autores lidam com a tarefa de detecção de fraude criando subconjuntos com base na proporção de exemplos positivos e negativos presentes no conjunto original. Os classificadores gerados para cada subproblema são combinados por um *meta-learning*. Para isso, os autores constroem um modelo utilizando um conjunto de dados composto pelas predições dos indutores do ensemble em uma base de validação e, a classe verdadeira do exemplo em questão.

Em ambientes *batch* é possível consultar o conjunto de dados inúmeras vezes, fazendo com que as técnicas como as citadas anteriormente, sejam viáveis para lidar com o desbalanceamento. No entanto, para lidar com fluxos de dados potencialmente infinitos e desbalanceados, tais técnicas devem ser adaptadas para respeitar as limitações impostas na seção 2.1.

O *framework* Learn++.NSE (ELWELL; POLIKAR, 2009) criado para lidar com diferentes mudanças de conceito, deu origem a outras duas outras variações que dizem respeito a tarefa de classificação de *data streams* desbalanceados: Learn++.NIE (DITZLER; POLIKAR, 2010) e Learn++.CDS (DITZLER; POLIKAR, 2013). Ambas as variações mantém a característica do Learn++.NSE de interpretar um fluxo de dados como vários batches sucessivos. A primeira variação constrói um sub-ensemble a cada novo batch de dados e, o pondera de acordo com o seu desempenho no *recall* para classes positivas e negativas. O Learn++.NIE visa treinar cada um dos classificadores em subconjuntos balanceados e utiliza os ensembles para diminuir a perda de informação que pode ser causada ao se criar esses subconjuntos. A segunda variação, o Learn++.CDS, avalia o desempenho do ensemble no *batch* atual e atribui pesos às instâncias que foram classificadas incorretamente. Logo após, o SMOTE é utilizado para enriquecer o *batch* recém-chegado criando instâncias sintéticas com base nos exemplos com maiores pesos. Um novo classificador é treinado no *batch* enriquecido pelo SMOTE e cada membro do ensemble é avaliado nesse mesmo *batch*. Caso algum indutor do ensemble não satisfaça a condição de eficiência, ele é substituído pelo novo classificador. Em ambas as variações, Learn++.NIE e Learn++.CDS, a predição final dada pelo ensemble é a combinação ponderada dos votos de cada algoritmo indutor.

Os autores em (GAO et al., 2007) apresentam uma solução que separa os exemplos S_{maj} de S_{min} de um *batch* retirado de um fluxo de dados. O *framework* constrói um ensemble utilizando subconjuntos balanceados por meio de *undersampling* e reaproveitamento de exemplos positivos anteriores. A predição de um novo exemplo é dada pela média das probabilidades estimadas pelos membros do ensemble. Seguindo o mesmo conceito, o *Selectively Recursive Approach* (SERA) reutiliza os exemplos positivos de *batches* anteriores

e dependendo do grau de desbalanceamento, seleciona as instâncias mais próximas do *batch* atual conforme a métrica Mahalanobis (CHEN; HE, 2009). Os membros do *ensemble* são treinados utilizando *Biased Bagging* que motiva o algoritmo indutor a focar nos exemplos positivos. O *Imbalance Data Stream Learner* (IDSL) também reaproveita as instâncias minoritárias e utiliza o KNN para selecionar as mais próximas do *batch* recém-chegado (GODASE; ATTAR, 2012). Os classificadores do IDSL são ponderados de acordo com o desempenho na métrica AUROC e a decisão final é dada por um voto majoritário ponderado. O método *Ensembles Weighted Using Meta-Classification* além de propagar as instâncias minoritárias, também propaga as instâncias majoritárias que não foram classificados corretamente (LICHTENWALTER; CHAWLA, 2009). No *framework* um meta classificador é alimentado com as performances de cada membro do *ensemble* e a similaridade, calculada por meio da combinação da métrica *Hellinger Distance* e ganho de informação, entre o *batch* recorrente e o *batch* utilizado para construir o modelo. Esse meta classificador é empregado para prever o desempenho de cada membro e então atribuir pesos a eles.

A solução proposta em (WANG; MINKU; YAO, 2013) consiste em um *framework* que utiliza um detector de desbalanceamento. Esse módulo permite que o *framework* funcione adequadamente sem conhecimento prévio da distribuição dos dados. Durante a etapa de treinamento, o módulo reconhece todas as potenciais classes majoritárias e minoritárias e, depois às compara em pares para avaliar o desbalanceamento relativo. Se o desbalanceamento é detectado, o *framework* utiliza o *Oversampling-based Online Bagging* (OOB) e o *Undersampling-based Online Bagging* (UOB) para balancear o conjunto de treino. Esses conjuntos são usados para treinar redes neurais que compõem o ensemble.

O *Multi-Windows Based Ensemble Learning* (MWEL) lida com o desbalanceamento utilizando três janelas que armazenam: o *batch* atual, os exemplos positivos e na terceira um *ensemble* de classificadores e os subconjuntos utilizados para treinar esses modelos (LI et al., 2017). A estratégia FILO é utilizada para substituir as instâncias minoritárias mais antigas pelas mais novas e os classificadores mais antigos por classificadores novos se, e somente se, o novo modelo apresentar uma boa performance (*precision*) para ambas as classes. A performance dos classificadores é empregada para detectar desbalanceamento presente na janela deslizante recorrente. Se de fato houver desbalanceamento, o SMOTE é aplicado nas instâncias classificadas incorretamente, minoritárias ou não, e *undersampling* aleatório é utilizado nas instâncias majoritárias classificadas corretamente. A predição final é dada pelo voto majoritário ponderado.

Mais recentemente, uma nova versão do *Adaptive Random Forest* para *data streams* desbalanceados foi apresentada. O *ARF with Resampling* (ARF_{RE}) combina pesos com a distribuição de *Poisson* calculada para uma determinada instância (BOIKO et al., 2019). Dessa forma, o ARF_{RE} altera a chance de um exemplo ser utilizado, durante o

Tabela 2 – Matriz de Confusão

	Previsto Positivo	Previsto Negativo
Positivo	TP	FN
Negativo	FP	TN

treinamento, baseado na distribuição de sua classe (Equação 2.10) e o replica (*sampling*) utilizando a distribuição de *Poisson* com o parâmetro λ. Seja uma classe c, quanto maior a sua incidência (S_c) em um *data stream* com S_n exemplos já observados, menor é o peso atribuído às instâncias pertencentes a essa classe.

$$weight(S_c, S_n, \lambda) = \frac{100 - \frac{S_c \times 100}{S_n}}{100} \times Poisson(\lambda) \tag{2.10}$$

O método *Kappa Updated Ensemble* (KUE) (CANO; KRAWCZYK, 2020) não é exclusivamente orientado a *data streams* desbalanceadas, porém os autores demostraram que ele apresentou ótimos resultados em diferentes níveis de desbalanceamento. O KUE é um *ensemble* que explora as características do *Random Forest* (BREIMAN, 2001) e constrói um conjunto ponderado de classificadores diversificados, isto é, cada indutor explora um subespaço e treina com uma quantidade diferente de dados. Os classificadores base do KUE são ponderados pelo seu desempenho na métrica estatística Kappa e a predição de uma instância é dada pelo voto majoritário ponderado, podendo até haver abstenção de classificadores.

Essa seção apresentou os conceitos envolvidos com o fenômeno desbalanceamento, tal como descreveu as principais abordagens para diminuir o impacto de dados desbalanceados na tarefa de mineração de *data streams*. As próximas seções 2.3 e 2.4 apontam as métricas e os procedimentos de avaliação para modelos preditivos em ambiente *streaming* desbalanceado.

2.3 Métricas para avaliação

Ao avaliarmos classificadores em ambientes desbalanceados, deve-se empregar métricas que consigam expressar o desempenho do modelo ao classificar instâncias minoritárias e majoritárias (HE; GARCIA, 2009). As métricas são geralmente desenvolvidas a partir da matriz de confusão. A Tabela 2 apresenta a matriz de confusão para um problema binário, em que TP (*True Positive*) e TN (*True Negative*) denotam os exemplos positivos e negativos que foram classificados corretamente, enquanto que FP (*False Positive*) e FN (*False Negative*) indicam os exemplos positivos e negativos classificados incorretamente.

A acurácia (Equação 2.11) é a métrica mais comum para avaliar a performance de um classificador em um conjunto de dados. No entanto, em ambientes desbalanceados ela

não informa adequadamente a performance do modelo devido à baixa representatividade da classe minoritária. Por exemplo, um indutor pode demonstrar uma acurácia de 99% somente escolhendo a classe majoritária, se para ele for apresentado um conjunto de dados composto por 990 exemplos negativos e apenas 10 exemplos positivos. Outras métricas como *Precision* (Equação 2.13) e *Recall* (Equação 2.12) oferecem mais informações sobre o percentual de classificações corretas em cada classe.

$$Acurácia = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.11}$$

$$Recall = Sensibilidade = TPRate = \frac{TP}{TP + FN} \tag{2.12}$$

$$Precision = \frac{TP}{TP + FP} \tag{2.13}$$

O *Precision* permite analisar quantos dos exemplos detectados como positivos são verdadeiramente positivos, enquanto que o *Recall* possibilita avaliar quantos exemplos positivos foram detectados em sua totalidade. Apesar de serem métricas mais informativas que a acurácia para ambientes desbalanceados, ambas as métricas avaliam apenas a performance do classificador em uma das classes. Dessa forma, fica inviável contabilizar quantos exemplos são classificados incorretamente como positivos (*Precision*) e, quantos exemplos positivos são incorretamente classificados (*Recall*) (HE; GARCIA, 2009). Uma maneira de se combinar essas duas métricas é utilizar o $F1_{score}$ (Equação 2.14), que é uma média harmônica entre *Precision* e *Recall*.

$$F1_{score} = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{2.14}$$

As métricas *Receive Operating Characteristics curve* (ROC) e *Area under the Receive Operating Characteristics curve* (AUROC) utilizam a sensibilidade (*Recall*) (Equação 2.12) e a especificidade (Equação 2.15). Dessa maneira, é possível visualizar as trocas relativas entre benefícios e custos da classificação (FAWCETT, 2006). Sabe-se que quanto maior a área abaixo da curva do ROC (AUROC), melhor um indutor consegue distinguir entre as classes positivas e negativas. Por essa razão, ambas as métricas são utilizadas em problemas de classificação binária. Porém, em bases de dados fortemente desbalanceadas a métrica *Area under the Precision-Recall Curve* (AUC-PR) apresenta informações mais relevantes sobre a performance do classificador. Isso se deve ao fato de que em um problema desbalanceado, a quantidade de exemplos negativos supera e muito a de exemplos positivos. Consequentemente, uma grande mudança no número de FP pode representar uma pequena variação na especificidade utilizada na métrica AUROC. Em oposição, a métrica AUC-PR utiliza o *precision* (Equação 2.13) que consegue capturar o efeito dessa grande mudança

Figura 2 – Curva ROC vs Curva PR

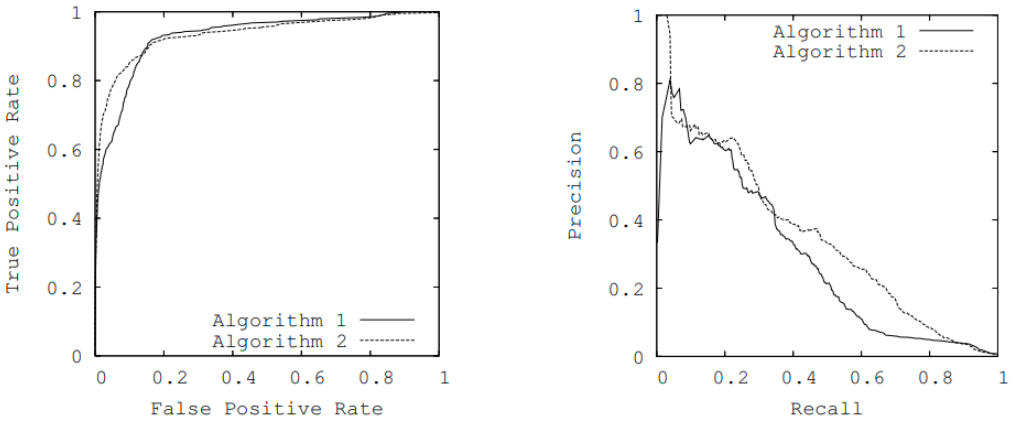


Ilustração gráfica da Curva ROC (esquerda) e Curva PR (direita) para dois diferentes classificadores em um mesmo problema. Reprodução de (DAVIS; GOADRICH, 2006).

por meio da comparação de FP com TP ao invés de FP com TN (DAVIS; GOADRICH, 2006). Essa diferença fica ainda mais evidente ao compararmos as representações gráficas do AUROC e AUC-PR (Figura 2).

$$Especificidade = FPRate = \frac{FP}{TN + FP} \tag{2.15}$$

O AUC-PR é um valor único obtido através do cálculo da área abaixo da curva *Precision-Recall* (PR). Para isso, calcula-se um par de PR para cada limite de decisão entre a classe positiva e negativa e partir disso constrói-se uma curva. Cada par de PR pode representar um classificador com um determinado limite de decisão para classificar um exemplo como positivo. De forma análoga ao AUROC, quanto maior a área abaixo da curva PR, melhor é o indutor em distinguir a classe positiva da negativa.

2.4 Procedimentos de avaliação para *streams*

A avaliação de algoritmos de classificação em ambientes estáticos (*batch*), permite que a utilização de diferentes arranjos de dados para treino e teste contribuam para a análise do modelo. Quando trabalhamos em ambientes dinâmicos (*streams*), tais técnicas deixam de ser utilizadas devido à complexidade e tempo computacional que infringem os requisitos para mineração de *data streams*. Uma solução para isso consiste em utilizar janelas temporais e avaliar o desempenho do modelo perante aos dados selecionados (BIFET et al., 2010).

O procedimento de avaliação fundamenta-se em determinar quais dados serão utilizados na etapa de aprendizagem e, quais serão empregados para avaliar o desempenho

do classificador. A avaliação *hold out*, que é utilizada no ambiente *batch* quando as bases de dados excedem a memória disponível, também pode ser empregada em *data streams*. Esse método consiste em expor o modelo preditivo a conjuntos de teste em intervalos temporais e, computar as devidas métricas de avaliação.

O método *Intervaled Test-Than-Train* ou *Prequential* permite avaliar o desempenho do classificador a cada exemplo subsequente de um fluxo de dados. Para tal, o procedimento consiste em utilizar cada exemplo para teste e em seguida treino do algoritmo de classificação. Dessa maneira, o modelo preditivo é avaliado somente em exemplos não vistos anteriormente e, não é necessário parametrizar o tamanho dos conjuntos de treino e teste como no *holdout* (GAMA; SEBASTIÃO; RODRIGUES, 2009). O erro computado pelo *prequential* embasa-se na soma cumulativa de uma função de erro entre a predição e os valores observados (Equação 2.16):

$$Erro = \sum_{i=1}^{n} L(y_i, \hat{y}_i) \tag{2.16}$$

O *prequential* é vastamente utilizado para avaliar o desempenho de algoritmos em fluxos contínuos de dados, pois oferece uma estimativa atual sobre o desempenho do classificador. Além disso, permite visualizar o desenvolvimento do modelo preditivo conforme a chegada de novos exemplos para treinar (GAMA; SEBASTIÃO; RODRIGUES, 2013).

2.5 *Streams* de dados

Com o objetivo de avaliar o desempenho do método proposto neste projeto, selecionou-se diferentes *streams* de dados disponíveis na literatura, os quais são ordenados em relação ao tempo. Dentre essas, estão presentes conjuntos de dados reais públicos, uma base de dados privada e geradores sintéticos. A Tabela 3 resume a descrição de cada uma das bases selecionadas.

Santander - santd[1]. Composto por exemplos que identificam se um cliente irá realizar uma transação no futuro, independentemente da quantia de dinheiro envolvida. O banco Santander disponibilizou esse conjunto de dados com uma estrutura semelhante aos dados reais utilizados pelo banco. O conjunto contém 76,020 instâncias, 371 atributos e apresenta a seguinte distribuição de instâncias para cada classe $[96.04\%, 3.95\%]$.

Electricity - elect. Contém dados coletados do mercado australiano de eletricidade, em que os preços flutuam conforme a demanda e oferta do mercado a cada 5 minutos. Essa base de dados contém 45,312 instâncias, com 9 atributos, rotuladas com valores que

[1] https://www.kaggle.com/c/santander-customer-transaction-prediction/

indicam se o preço subiu ou desceu. A sua distribuição de classes é de [42.45%, 57.54%] (HARRIES; WALES, 1999).

Airlines - airli[2]. Contém um histórico de vôos e o objetivo de classificação é determinar a probabilidade de um vôo atrasar. Contém 539,383 instâncias, 7 atributos e uma distribuição de [55.46%, 44.54%].

Poker Hands - poker[3]. Os exemplos contidos nesse conjunto de dados são compostos por 11 atributos e representam 5 cartas oriundas de um baralho com 52 cartas. O atributo classe descreve a *"poker hand"* com base na combinação de cartas. Esse problema multiclasse possui a seguinte distribuição: [50.11%, 42.26%, 4.75%, 2.11%, 0.38%, 0.19%, 0.14%, 0.0235%, 0.0013%, 0.0002%] (DUA; GRAFF, 2017).

Give Me Some Credit - gmsc[4]. O conjunto de dados disponibilizado na competição do *Kaggle* tem como objetivo determinar a probabilidade de um cliente se tornar inadimplente nos próximos dois anos. Essa base de dados contém 150,000 instâncias, 11 atributos e apresenta a seguinte distribuição [93.316%, 6.684%].

Forest CoverType - covrt[5]. Composta por dados que representam os tipos de cobertura florestal da Região 2 do Serviço Florestal dos EUA (USFS) obtidos via sensoriamento remoto. Esse conjunto contém 581,012 exemplos e 54 atributos, apresentando a seguinte distribuição de classes [36.46%, 48.75%, 6.15%, 0.47%, 1.63%, 2.98%, 3.53%] (DUA; GRAFF, 2017).

Pozzolo Credit Card Fraud Detection - pozzl[6]. Contém transações realizadas por cartão de crédito oriundos da Europa. Os 284,807 exemplos, com 31 atributos, contém apenas 492 transações fraudulentas, apresentando uma distribuição de [99.82%, 0.17%].

Escoragem de Crédito Privada - privt. Conjunto de dados privados que representam uma escoragem de crédito, possuindo 97,226 instâncias, 130 atributos e com uma distribuição [26.07%, 73.92%].

Agrawal Generator - agrwl. Gerador capaz de sintetizar uma tarefa de classificação binária contendo 9 atributos. Além disso, permite adicionar dados ruidosos entre os atributos conforme uma distribuição aleatória uniforme. O grau de desbalanceamento do *Agrawal Generator* será definido pelo método *ImbalancedStream* presente no *Massive Online Analysis* (MOA) (BIFET et al., 2010), utilizando as versões **agrwl90** ([90%, 10%]), **agrwl95** ([95%, 5%]), **agrwl99** ([99%, 1%]) (AGRAWAL; IMIELINSKI; SWAMI, 1993).

SEA Generator - sea (STREET; KIM, 2001). Assim como o *Agrawal Generator*, o *SEA Generator* é capaz de sintetizar uma tarefa classificação binária contendo 3

[2] http://kt.ijs.si/elena_ikonomovska/data.html
[3] https://archive.ics.uci.edu/ml/datasets/Poker+Hand
[4] https://www.kaggle.com/c/GiveMeSomeCredit/
[5] http://kdd.ics.uci.edu/databases/covertype/covertype.data.html
[6] https://www.kaggle.com/mlg-ulb/creditcardfraud

Tabela 3 – Bases de dados selecionadas

Base de dados	# Instâncias	# Atributos	# Classes	Distribuição
santd	76,020	371	2	[96.04%, 3.95%]
elect	45,312	9	2	[42.45%, 57.54%]
airli	539,383	7	2	[55.46%, 44.54%]
poker	829,201	11	10	Ver 2.5
gmsc	150,000	11	2	[93.31%, 6.68%]
covrt	581,012	54	7	Ver 2.5
pozzl	284,807	31	2	[99.82%, 0.17%]
privt	97,226	130	2	[26.07%, 73.92%]
agrwl90	1,000,000	9	2	[90%, 10%]
agrwl95	-	-	-	[95%, 5%]
agrwl99	-	-	-	[99%, 1%]
sea90	1,000,000	3	2	[90%, 10%]
sea95	-	-	-	[95%, 5%]
sea99	-	-	-	[99%, 1%]

atributos contínuos e empregar um percentual de ruído entre os dados. As versões geradas possuem 20% de ruído e são definidas como: **sea90** ($[90\%, 10\%]$), **sea95** ($[95\%, 5\%]$), **sea99** ($[99\%, 1\%]$).

2.6 Considerações finais

Esse capítulo apresentou os fundamentos relacionados a mineração de *data streams*, assim como cobriu as principais características do fenômeno de desbalanceamento e, os procedimentos e métricas adequados para a avaliação do desempenho de classificadores nesse contexto. Esses conceitos serviram como fundamentação teórica para a atribuição de sensibilidade a custos ao *Adaptive Random Forest* que está sendo proposto neste trabalho. O próximo capítulo descreve o *Cost-sensitive Adaptive Random Forest*, baseado na hipótese de diminuir a sobre-representação da classe majoritária empregando custos de classificação de forma a evidenciar a classe minoritária.

3 Método

O foco do método desenvolvido consiste em utilizar diferentes abordagens de custos de classificação com a finalidade de diminuir o impacto do desbalanceamento na tarefa de classificação de *streams* de dados. A flexibilidade de um *framework* de atribuição de penalidades de classificação pode tornar sensíveis a custos vários tipos de classificadores. De forma resumida, o uso de penalidades de classificação fundamenta-se em fazer com que o algoritmo indutor, ao realizar uma predição, considere as penalidades de se classificar uma classe em outra. Para isso, ele deve induzir uma classe i para um exemplo x que minimiza a soma das probabilidades alternativas para a classe verdadeira de x (Equação 3.1):

$$L(x, i) = \sum_{j} P(j|x)cost(i, j) \tag{3.1}$$

A hipótese do método proposto é de que ao atribuir maiores custos de classificação para a classe minoritária, ocorreria uma melhora na performance de classificação em S_{min}, sem que a performance da classe majoritária seja severamente comprometida. A próxima seção descreve um novo *ensemble* adaptativo sensível a custos baseado em árvores de decisão chamado *Cost-sensitive Adaptive Random Forest*.

3.1 Cost-sensitive Adaptive Random Forest

As características presentes em um *ensemble* permitem a utilização de diferentes abordagens para diminuir o impacto dos dados desbalanceados. Como apresentado na seção 2.2, diferentes abordagens podem ser utilizadas para diminuir a super representação da classe majoritária. Porém, nem todas as estratégias funcionam ou se adaptam em ambientes de data streams devido a alguns fatores, tais como: extensos cálculos computacionais e a necessidade de analisar o conjunto de dados por inteiro. Desta forma, optou-se pelas abordagens baseadas em custos para amenizar o impacto do desbalanceamento na construção de modelos.

O *Adaptive Random Forest* dispõe de diferentes particularidades que podem ser alteradas para o ambiente de dados desbalanceados. O método aqui proposto, *Cost-sensitive Adaptive Random Forest* (Algoritmo 3), atribui as seguintes mudanças na estrutura do ARF:

I. Atribuição de pesos às árvores (Algoritmo 3, Linha 14) utilizando o valor médio de F1_{score} (Equação 2.14) ao invés da acurácia.

II. Modificação no processo de aprendizagem (*sampling*), com o objetivo de garantir que todas as árvores treinem com as instâncias que pertencem às classes minoritárias.

III. Utilização de uma janela deslizante para verificar a distribuição das classes.

O CSARF mantém o voto majoritário ponderado, originalmente presente no ARF, para priorizar as árvores com melhores desempenho de classificação. Formalmente, o voto majoritário ponderado de um *ensemble* de tamanho B para uma instância x é calculado pela combinação de um peso w_j com uma probabilidade de que a instância x seja da classe i estimada pelo classificador j:

$$C(x) = \arg\max_{i} \sum_{j=1}^{B} w_j p_{ij} \tag{3.2}$$

No entanto, o uso da métrica acurácia em ambientes desbalanceados prova-se ineficaz para avaliar o desempenho de um classificador, conforme demonstrado na seção 2.3. Assim sendo, no CSARF os classificadores base serão ponderados conforme a sua performance na métrica F1_{score} médio (Algoritmo 3, Linha 14). A métrica *Matthews Correlation Coefficent* (MCC) (Equação 3.3) (MATTHEWS, 1975) se mostrou boa inicialmente para avaliar as árvores em ambientes desbalanceados, conforme foi proposta originalmente e implementada no protótipo do método. Os autores em (CHICCO, 2017; CHICCO; JURMAN, 2020), evidenciam que o MCC consegue descrever com uma granularidade maior que a do F1_{score} o desempenho de um classificador em bases de dados desbalanceadas. No entanto, testes empíricos realizados analisando diferentes possibilidades para ponderar as árvores do CSARF evidenciaram que o F1_{score} médio apresentava os melhores resultados nas métricas avaliadas. Os detalhes e resultados obtidos desse experimento estão em anexo (Anexo A) no presente documento.

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FN)(TP + FP)(TN + FP)(TN + FN)}} \tag{3.3}$$

A mudança II presente no CSARF diz respeito à utilização do *Online Bagging* (OB) para realizar a reamostragem das instâncias de treinamento. O ARF utiliza o OB para reamostrar as instâncias independente da classe entre as árvores seguindo a distribuição de $Poisson(\lambda = 6)$. Dessa maneira, é possível que para ambientes desbalanceados, a aplicação dessa técnica reforce o desbalanceamento ao invés de mitigá-lo. Durante a etapa de treinamento, o CSARF ameniza esse problema ao verificar se a instância corrente pertence a S_{min} e então, a utiliza mesmo se o valor obtido de k para $Poisson(\lambda = 6)$ for igual a zero (Algoritmo 4, Linha 2).

A terceira mudança fundamenta-se na utilização de uma janela deslizante capaz de reter informações sobre a distribuição de classes localmente. Essa janela consiste em um vetor de tamanho n que armazena as instâncias mais recentes e respeita um comportamento

First In First Out (FIFO). A partir dessa janela (Algoritmo 3, Linha 4), o CSARF é capaz de:

1. Deduzir a classe majoritária para aplicar a mudança II: Isto é, as instâncias da classe majoritária devem aparecer em maior quantidade ao decorrer do fluxo de dados. Dessa forma, as classes detectadas como minoritária, serão aquelas com menos frequência dentro da distribuição local da janela e não do fluxo todo.

2. Atualizar a matriz de custos conforme a distribuição de classes: Isso é ideal para as heurísticas de cálculo de custos que consideram a distribuição de classes em sua estrutura. A subseção 2.2.3 descreve detalhadamente essas heurísticas.

Dessa forma, é possível utilizar o CSARF em ambientes com diferentes níveis e flutuações de desbalanceamento conforme o *data stream* evolui. No algoritmo 3 a função *applyCost* implementa a equação 3.1.

A partir das informações obtidas por meio da janela deslizante, é possível definir os custos de classificação a serem considerados pelo *ensemble*. Diferentes estratégias podem ser utilizadas para definir os custos de classificação com base na distribuição das classes. No entanto, ressalta-se a importância de sempre manter o modelo atualizado. Dessa forma, a distribuição dentro da janela será a mais recente e representativa da situação corrente em que o problema de classificação se encontra. A subseção 2.2.3 aborda as estratégias estudadas para o método proposto.

3.1.1 Heurísticas de custos

O custo de classificação, também denominado como penalidade, é um valor que simboliza o prejuízo ao se classificar uma classe em outra. Em abordagens baseadas em custos, esse valor pode informar ao classificador quantas vezes ele deve treinar com determinada instância (*sampling*) ou, ser utilizado para definir um novo limite de decisão de classificação entre as classes do problema (*threshold*). Para construir uma matriz de custos e definir as penalidades atribuídas para cada classe deve-se adotar uma heurística de custo.

O método mais comum de se construir uma matriz de custos é fixar uma penalidade para cada classe. Os valores podem ser escolhidos por meio de um *grid search* exaustivo, testando-se assim vários custos até atingir o desempenho desejado. Essa estratégia funciona em ambiente *batch*, porém em *data streams* deve-se considerar a variação da distribuição de classes e que se desconhece o universo potencialmente infinito de um fluxo de dados.

As estratégias que utilizam como informação a distribuição das classes são alternativas ao uso de penalidades fixas. Em (PICEK et al., 2018), os autores definem pesos para cada classe conforme a sua distribuição na base de dados (Equação 3.4). Isto é, se

Algorithm 3 Cost-sensitive Adaptive Random Forest. **Symbols**: m: Número máximo de atributos avaliados por divisão; n: Número total de árvores ($n = |T|$); δ_w: *Threshold* de suspeita; δ_d: *Threshold* de mudança; I_w: Tamanho da janela deslizante; C: Matriz de custos; $c(\cdot)$: Detector de *concept drift*; S: *Data stream*; B: Conjunto de indutores adicionais; $W(t)$: Peso da árvore t; I_w: Janela deslizante; C: Matriz de custos; $P(\cdot)$: Função de estimação performance.

```
1: function CSARF(m, n, δ_w, δ_d, I_w, C)
2:     T ← CreateTrees(n)
3:     W ← InitWeights(n)
4:     B ← ∅
5:     while HasNext(S) do
6:         (x, y) ← next(S)
7:         I_w ← (x, y)
8:         for all t ∈ T do
9:             ŷ ← predict(t, x)
10:            if isTrainingCost then
11:                applyCost(cost, ŷ)
12:            W(t) ← P(W(t), ŷ, y)
13:            CSARFTreeTrain(m, t, x, y)
14:            if C(δ_w, t, x, y) then
15:                b ← CreateTree()
16:                B(t) ← b
17:            end if
18:            if C(δ_d, t, x, y) then
19:                t ← B(t)
20:            end if
21:        end for
22:        for all b ∈ B do
23:            CSARFTreeTrain(m, b, x, y)
24:        end for
25:    end while
26: end function
```

uma classe A é n vezes mais frequente do que a classe B, ela terá seu custo dividido por n enquanto que a classe B terá seu peso multiplicado por n. Os autores empregam esse custo calculado para realizar *undersampling* da classe majoritária e *oversampling* da classe minoritária. Apesar de não termos a informação da distribuição de um fluxo de dados, esse método pode ser aplicado na janela deslizante presente no CSARF. Conforme os dados são consumidos, a distribuição de classes pode variar e por consequência as penalidades da matriz de custos também se alteram.

$$Picek = classweight(i) = \frac{\#samples}{\#classes \times samples(i)} \tag{3.4}$$

Em bases de dados desbalanceadas, as probabilidades da classe majoritária tendem

Algorithm 4 CSARFTreeTrain. **Symbols**: λ: Parâmetro fixo para distribuição de *Poisson*. GP: *Grace Period* antes de recalcular as heurísticas para dividir um nó.

```
1: function CSARFTreeTrain(m, t, x, y)
2:     k ← Poisson(λ = 6)
3:     if k > 0 OR isMinority(x, y) then
4:         l ← FindLeaf(t, x)
5:         UpdateLeafCounts(l, x, k)
6:         if InstancesSeen(l) ≥ GP then
7:             AttemptSplit(l)
8:             if DidSplit(l) then
9:                 CreateChildren(l, m)
10:            end if
11:        end if
12:    end if
13: end function
```

a ser maior devido a sobre-representação dela no classificador. Porém, em problemas de classificação em que as classes estão bem separadas, essa situação pode não se repetir. Um exemplo de uma estratégia para o cálculo de custos baseada na dificuldade de classificação é utilizada por Oza e Russel em (OZA, 2005), para adaptar o *Adaboost* (SCHAPIRE; SINGER, 1999) para ambientes de *data streams*. No *Online Boosting* (Algoritmo 5), cada classificador base tem uma variável que contabiliza os acertos (λ_m^{sc}) e outra que contabiliza os erros (λ_m^{sw}). Esses contadores são populados durante a etapa de treinamento, em que cada exemplo recebe um peso inicial $\lambda_d = 1$ (Algoritimo 5, Linha 1). O peso atribuído à instância é utilizado para reamostrar o exemplo durante o treinamento do classificador base conforme $Poisson(\lambda = \lambda_d)$ (Algoritmo 5, Linha 4). E então, iterativamente as variáveis λ_d, λ_m^{sc} e λ_m^{sw} são alteradas conforme a classificação correta ou incorreta de cada indutor base (Algoritmo 5, Linhas 8-13). Dessa forma o peso de cada classificador base é calculado conforme as equações 3.5, 3.6 e 3.7.

$$E_m = \frac{\lambda_m^{sw}}{\lambda_m^{sc} + \lambda_m^{sw}} \tag{3.5}$$

$$\beta_m = \frac{E_m}{1 - E_m} \tag{3.6}$$

$$weight(h_m) = \log \frac{1}{\beta_m} \tag{3.7}$$

Fundamentado na estratégia do *Online Boosting*, propõe-se aqui uma tentativa de criação de uma heurística de custo que incorpora os erros e acertos do *ensemble* e desconsidera a distribuição das classes (Algoritmo 6). Para tal, as variáveis λ_d, λ_m^{sc} e λ_m^{sw} são calculadas para cada classe. Durante a etapa de treinamento do CSARF, cada instância

Algorithm 5 Online Boosting. **Symbols**: h_M: Conjunto de M classificadores base; *OnlineBase*: Algoritmo de aprendizagem incremental; d: Instância atual; λ_m^{sc}: Soma dos valores λ para os exemplos corretamente classificados no estágio m; λ_m^{sw}: Soma dos valores λ para os exemplos incorretamente classificados no estágio m; N: Número de instâncias vistas até então.

function OnlineBoosting(h_M, *OnlineBase*, d)
  $\lambda_d \leftarrow 1$
  for all $h_m \in h_M$ **do**
    $k \leftarrow Poisson(\lambda_d)$
    loop k times
      $h_m = OnlineBase(model_m, d)$
    end loop
    if $h_m(d)$ is correct **then**
      $\lambda_m^{sc} \leftarrow \lambda_m^{sc} + \lambda_d$
      $\lambda_d \leftarrow \lambda_d(\frac{N}{2\lambda_m^{sc}})$
    else
      $\lambda_m^{sw} \leftarrow \lambda_m^{sw} + \lambda_d$
      $\lambda_d \leftarrow \lambda_d(\frac{N}{2\lambda_m^{sw}})$
  end for
end function

Fonte: Adaptação de (OZA, 2005).

é avaliada pelo voto combinado do *ensemble* antes de ser utilizada para treino e então, as variáveis λ_d, λ_m^{sc} e λ_m^{sw} são atualizadas conforme a resposta correta ou incorreta do *ensemble*. Para calcular as penalidades de cada classe, somente a Equação 3.5 é alterada de maneira a gerar maiores pesos às classes mais dificultosas de se classificar, e para as demais mantém-se como são utilizadas no *Online Boosting* (Equações 3.6 e 3.7[1]):

Algorithm 6 Online Boosting Costing. **Symbols**: $CSARF$: Cost-sensitive Adaptive Random Forest; d: Instância atual; *classIndex*: Índice da classe da instância atual; λ_m^{sc}: Soma dos valores λ para os exemplos corretamente classificados para classe m; λ_m^{sw}: Soma dos valores λ para os exemplos incorretamente classificados para classe m; N: Número de instâncias vistas até então.

function OzaCosting($CSARF$, d, *classIndex*)
  $m \leftarrow classIndex$
  if $CSARF(d)$ is correct **then**
    $\lambda_m^{sc} \leftarrow \lambda_m^{sc} + \lambda_d$
    $\lambda_d \leftarrow \lambda_d(\frac{N}{2\lambda_m^{sc}})$
  else
    $\lambda_m^{sw} \leftarrow \lambda_m^{sw} + \lambda_d$
    $\lambda_d \leftarrow \lambda_d(\frac{N}{2\lambda_m^{sw}})$
end function

[1] A chamada da função $weight(h_m)$ tornar-se $weight(class_m)$.

$$E_m = \frac{\lambda_m^{sc}}{\lambda_m^{sc} + \lambda_m^{sw}} \tag{3.8}$$

As seguintes soluções baseadas em *ensembles* para *data streams* desbalanceados, presentes em (WANG; MINKU; YAO, 2013; LI et al., 2017) e detalhadas na seção 2.2.4, empregam um detector de desbalanceamento. Para as duas soluções, as informações obtidas pelo detector servem para iniciar um processo de balanceamento do *batch* corrente utilizando *oversampling* e *undersampling*. Estudou-se a funcionalidade que um detector teria na estrutura do CSARF, visto que esse possui uma janela deslizante capaz de reter informações sobre a distribuição de classes. Todavia, descartou-se a ideia dado que as heurísticas de custos, detalhadas anteriormente, já se adaptam conforme a distribuição de classes ou assertividade do classificador. Do ponto de vista empírico, é possível visualizar a variação do custo da classe minoritária conforme o fluxo de dados evolui na Figura 3. O eixo y dos gráficos representa o valor da penalidade para a classe minoritária, enquanto que o eixo x informa a quantidade de dados processada ao decorrer do fluxo de dados. Para a construção destes gráficos, uma janela deslizante com capacidade para 10,000 instâncias foi escolhida e o valor do custo é adquirido a cada 100 instâncias. Os custos foram calculados utilizando o CSARF no conjunto de dados *santd* (Ver Tabela 3).

Nessa subseção foram apresentadas as heurísticas de custos presentes no CSARF. A subseção 3.1.2 detalha diferentes estratégias para empregar as penalidades de classificação na estrutura do CSARF.

3.1.2 Estratégias de uso do *threshold*

Ao deslocarmos as probabilidades de saída de um classificador, fixa-se uma probabilidade mínima para que um exemplo seja categorizado como minoritário ou majoritário. Em conjuntos de dados desbalanceados, as instâncias majoritárias tendem a possuir maiores probabilidades pois uma maior quantidade de exemplos majoritários foi apresentada ao algoritmo indutor. Quando consideram-se os custos envolvidos ao errar a classificação de um exemplo positivo, um limite mínimo pode ser fixado para se classificar uma classe em outra. Dessa forma, para minimizar os custos de classificação, a classe escolhida é aquela com a maior probabilidade e menor custo envolvido (Equação 3.1).

O voto majoritário ponderado (Equação 3.2) possibilita a aplicação do *framework* $L(x, i)$ (Equação 3.1) de duas maneiras diferentes. Assim sendo, descreve-se a seguir duas estratégias de uso do *threshold* no CSARF:

- **CSARF-local**: Essa abordagem fundamenta-se em empregar os custos de classificação para influenciar a saída dos classificadores base antes da combinação dos votos. Assim sendo, a opinião do *ensemble* é influenciada e isso pode ou não modificar o voto obtido após a combinação. Matematicamente, o CSARF-local calcula o voto

Figura 3 – Variação das heurísticas de custos.

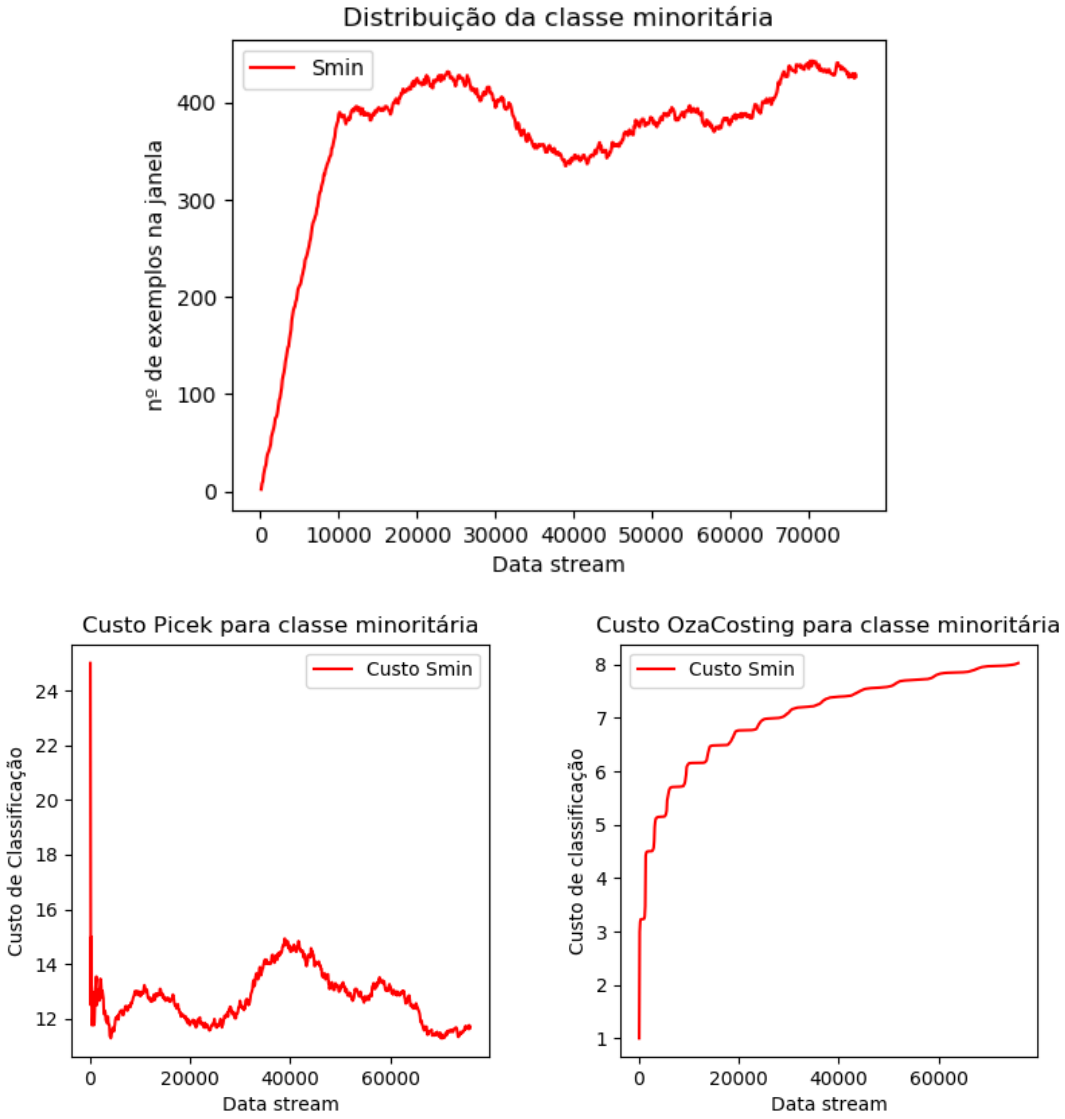


Ilustração da variação das heurísticas de custos conforme a distribuição da classe minoritária (S_{min}) na janela deslizante para base de dados *santd*.

majoritário combinando o peso w_j (performance do classificador base na métrica $F1_{score}$ médio) com a classe i que minimiza a soma das probabilidades alternativas da classe corrrente x (Equação 3.9):

$$C(x) = \arg\max_{i} \sum_{j=1}^{B} w_j \times \left[\arg\min_{i} \sum_{i} L(x,i) \right] \tag{3.9}$$

- **CSARF-global**: A abordagem **global** baseia-se em aplicar as penalidades de classificação após a combinação dos classificadores. Alterando-se assim, não as opiniões individuais de cada indutor base, mas a combinação final de suas opiniões. Matematicamente, o CSARF-global após computar o voto majoritário escolhe a classe i

que minimiza a soma das probabilidades alternativas da classe corrente x (Equação 3.10):

$$C(x) = \arg\min_{i} \sum_{i} \sum_{j} cost(i,j) \times \left[\sum_{k=1}^{B} w_k p_{jk} \right] \tag{3.10}$$

As heurísticas do ARF para ponderar as árvores do *ensemble* possibilitam a atribuição das penalidades a partir de uma terceira abordagem. Durante a etapa de treinamento e atualização do modelo, as probabilidades estimadas por cada classificador são utilizadas para calcular a métrica de ponderamento. A terceira abordagem, denominada como **CSARF-training**, baseia-se na ideia de aplicar o *framework* $L(x,i)$ (Equação 3.1) somente nas probabilidades estimadas durante o treinamento e atualização do modelo (Algoritmo 3, Linha 12). Dessa maneira, a métrica que pondera os classificadores base é alterada pelos custos de classificação e por consequência, a probabilidade estimada para cada classe, durante uma decisão, também será alterada de maneira indireta.

Para saber de fato como essas estratégias influenciam na decisão final do *ensemble*, apresenta-se a seguir uma análise empírica do CSARF. Como exemplo para fins de explicação, as probabilidades de saída do CSARF foram obtidas após a construção do modelo utilizando a base *weather* (QUINLAN, 1993) duplicada, conforme os parâmetros na tabela 4.

Tabela 4 – Parâmetros CSARF

CSARF (*default, training, local e global*)	
ensembleSize	3
imbalancedWindow	10
cost	Picek (Eq. 3.4)

A representação do CSARF e suas tomadas de decisão estão ilustradas na Figura 4. Nessa ilustração, apresentam-se as saídas de quatro configurações do CSARF:

1. *Default*: não utiliza os custos, mas possui as mudanças citadas anteriormente.
2. *Training*: utiliza os custos somente para alterar a avaliação do desempenho da árvore.
3. *Local*: a saída de cada indivíduo do *ensemble* é influenciada pelos custos.
4. *Global*: o *threshold* é utilizado para deslocar as probabilidades após a combinação dos votos dos classificadores base.

A saída de cada classificador consiste em um vetor com duas posições para os valores de classe $[yes, no]$. O rótulo do exemplo avaliado nessa análise tem como valor *no*, e então, para ser corretamente classificado a saída do *ensemble* deve possuir a maior probabilidade

Figura 4 – CSARF e suas variações.

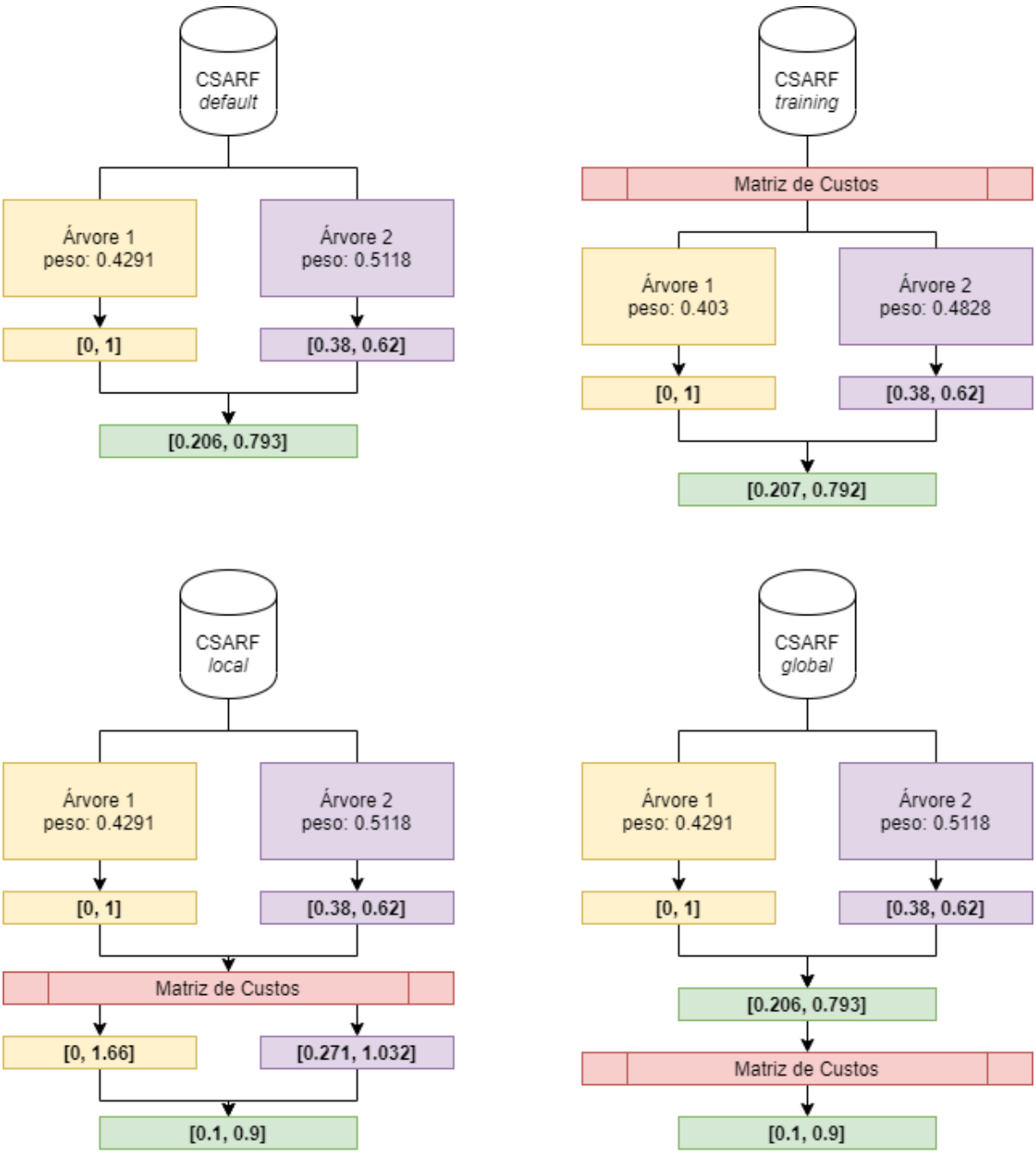


Ilustração do funcionamento das diferentes estratégias para atribuição de sensibilidade à custos no CSARF: *training*, *local* e *global*. As árvores que não contribuíram para os votos não foram representadas.

na segunda posição do vetor. O peso de cada árvore consiste em sua performance na métrica $F1_{score}$ médio. Os votos de cada árvore são ponderados pela métrica e em seguida combinam-se os votos e normalizam-se as probabilidades finais obtidas. As penalidades para erro de classificação de cada classe estão representadas em formato de uma matriz de

custos (Tabela 5).

Tabela 5 – Matriz de Custos Weather

	Actual 'yes'	Actual 'no'
Pred. 'yes'	0	1,666
Pred. 'no'	0,714	0

Observa-se que dentre das estratégias escolhidas, a que menos influenciou no voto final foi a abordagem CSARF-training. Isso deve-se ao fato dos custos serem utilizados somente para mudar o peso atribuído a árvore. Dessa forma, a participação do classificador base pode ser alterada positivamente ou negativamente conforme a avaliação dos seus votos influenciados pelos custos de classificação. No CSARF-training, verifica-se que os custos de classificação influenciaram negativamente no desempenho da métrica $F1_{score}$ médio.

Em outro cenário, o CSARF-local classifica a instância com a classe *no* com maiores valores de probabilidade que o CSARF-default. O desempenho obtido pelos classificadores base é o mesmo presente na versão *default*. Todavia, suas probabilidades de saída são influenciadas pela matriz de custo antes de fazer a combinação dos votos. Dessa forma, cada árvore escolhe a classe que minimiza a soma das penalidades combinadas com probabilidades de classes alternativas.

Por fim, observa-se o comportamento do CSARF-global, que apresenta desempenho das árvores com a métrica $F1_{score}$ médio e votos equivalentes ao CSARF-default. A diferença localiza-se nas probabilidades obtidas após a combinação ponderada de cada classificador. Para esse caso, verifica-se que a matriz de custos apenas reforçou a resposta final do *ensemble* para o exemplo que contém o rótulo *no*.

Para explorar o potencial do CSARF, realizou-se experimentos com os *streams* de dados listadas na subseção 2.5. Para tanto, desenvolveu-se um protocolo experimental com o objetivo de avaliar o desempenho da versão inteiramente implementada do CSARF contra a sua versão de origem, ARF, e outros dois algoritmos: ARF_{RE} e KUE. Os resultados desse experimento encontram-se no próximo capítulo.

3.2 Considerações finais

Esse capítulo apresentou o método proposto *Cost-sensitive Adaptive Random Forest* para lidar com *stream* de dados desbalanceados. Enumerou-se as modificações realizadas na estrutura do método original *Adaptive Random Forest* e, apresentou-se duas heurísticas de custos de classificação, sendo que uma das heurísticas, *OzaCosting*, é uma tentativa de concepção de uma heurística de custo. Conceitualizou-se as três estratégias de atribuição de

sensibilidade a custos do CSARF (*local, training* e *global*) e, apresentou-se o funcionamento de cada uma delas por meio de um protótipo. O próximo capítulo apresenta os resultados e discussões sobre o experimento comparativo realizado entre o CSARF e outros métodos.

4 Resultados

Neste capítulo, serão apresentados os resultados obtidos por meio dos experimentos realizados com o *Cost-sensitive Adaptive Random Forest* (CSARF). O *framework Massive Online Analysis* (MOA) (BIFET et al., 2010) foi utilizado para implementar o método proposto e realizar os demais experimentos. O método proposto foi avaliado em *streams* de dados com diferentes níveis de balanceamento e comparado contra os métodos *Adaptive Random Forest* (ARF) (GOMES et al., 2017b), *ARF with Resampling* (ARF_{RE}) (BOIKO et al., 2019) e *Kappa Updated Ensemble* (KUE) (CANO; KRAWCZYK, 2020). Os experimentos foram conduzidos explorando a variação de parâmetros de modo a se estabelecer a melhor versão de cada método para uma determinada métrica e *stream* de dados escolhido. Os resultados que são apresentados neste capítulo condizem com uma comparação orientada ao mundo real, isto é, buscando os melhores parâmetros de cada método para obter a maior assertividade possível, dentro do espaço de variação explorado, a fim solucionar o problema em cada *stream* de dados.

A próxima seção descreve com detalhes o protocolo experimental conduzido para avaliar a performance dos métodos tanto nos *streams* de dados binários quanto nos problemas multi-classe. Em seguida, uma análise é conduzida para avaliar o potencial das heurísticas de custos *Picek e OzaCosting* (seção 4.2). As seções 4.3 e 4.4 apresentam os resultados obtidos pelo método proposto em comparação ao *baseline* (ARF) e métodos concorrentes (ARF_{RE} e KUE) para o cenário binário e multi-classe, respectivamente. Este capítulo se encerra com a verificação das hipóteses formuladas no protocolo experimental e uma análise do potencial das estratégias de uso do *threshold* do CSARF.

4.1 Protocolo Experimental

Esta seção detalha as etapas conduzidas para a obtenção e avaliação dos resultados. O protocolo experimental consiste em avaliar as características do método proposto e o seu desempenho contra os demais. Para isso, as seguintes hipóteses nulas foram formuladas:

1. Não há diferença significativa entre a heurística de custo *Picek* e *OzaCosting*.
2. Não há diferença significativa entre o CSARF, ARF, ARF_{RE} e KUE para as métricas *recall* médio, F1_{score} médio e AUC-PR.

O resultado da primeira hipótese implica na escolha da melhor heurística de custo para assim conduzir os experimentos que dizem respeito a segunda hipótese. Para verificar as hipóteses formuladas, analisou-se o desempenho dos algoritmos utilizando a estratégia

Prequential ou *Test-than-train* (descrita anteriormente na seção 2.4), garantindo que o modelo gerado primeiro teste e depois treine a cada nova instância do fluxo de dados. Os primeiros 1,000 (mil) exemplos foram utilizados somente para treinar cada um dos modelos. Para cada um dos *streams* de dados (listados na seção 2.5) um *grid search* foi realizado para cada método avaliado e os seguintes parâmetros foram explorados:

- Comum a todos os algoritmos[1]:
  - Tamanho do *ensemble* (**-e** #): (25, 50, 75, 100).
  - Espaço de *features* (**-f** %): (25, 50, 75, 100).
- CSARF:
  - Estratégia de *threshold*: *Training* (**-t**), *Local* (**-l**) e *Global* (**-g**).
  - Heurística de custos: *Picek* e *OzaCosting*.
  - Janela deslizante (*Imbalanced window*) (**-i** #): (1,000, 5,000, 10,000, 20,000).
- KUE:
  - Número de novos classificadores a serem treinados (**-n** % do tamanho do *ensemble*): (10, 20, 30, 40)

Os resultados obtidos são oriundos de uma exploração dos parâmetros de cada algoritmo avaliado para um determinado fluxo de dados. A próxima sub-seção descreve como cada resultado gerado foi avaliado.

4.1.1 Validação

A probabilidade estimada, para cada classe do problema, pelos métodos escolhidos é empregada para calcular o desempenho do modelo preditivo gerado nas métricas de avaliação. Por se tratar de uma tarefa de classificação de dados desbalanceados, o uso da métrica acurácia carece de informações para avaliar o classificador conforme explicado anteriormente na seção 2.3. Portanto, as métricas que consideram o desempenho na classe minoritária, como o AUC-PR (*Area under the precision recall curve*), se destacam (DAVIS; GOADRICH, 2006).

O *precision* (Equação 2.13) e o *recall* (Equação 2.12) são duas métricas construídas a partir da matriz de confusão. A primeira permite analisar quantos dos exemplos positivos detectados são verdadeiramente positivos, enquanto que a segunda possibilita avaliar quantos exemplos positivos foram detectados em sua completude para cada classe do

[1] O algoritmo KUE possui um espaço de features aleatório para cada árvore, impossibilitanto explorar esse parâmetro.

problema. Uma forma de se combinar as duas métricas é por meio do $F1_{score}$ (Equação 2.14), que é uma média harmônica entre as duas métricas. Uma outra forma de aproveitar as informações que essas métricas descrevem é por meio do cálculo de um par de *precision* e *recall* para cada limite de decisão entre a classe positiva e negativa, de um problema de classificação binário, e assim construir uma curva. O valor único obtido através da área abaixo dessa curva (AUC-PR) proporciona visualizar o desempenho de um classificador sem que a performance na classe negativa interfira no resultado obtido (DAVIS; GOADRICH, 2006).

Resumidamente, o protocolo experimental consiste em avaliar o desempenho do método proposto e os demais nas métricas *recall* médio[2], $F1_{score}$ médio e AUC-PR (somente para os fluxos de dados binários). A próxima seção descreve e discute os resultados obtidos com as heurísticas de custos para os *streams* de dados analisados.

4.2 Picek versus OzaCosting

A escolha de uma heurística de custo impacta diretamente na performance de um classificador sensível a custos. Normalmente, realiza-se um *grid search* exaustivo em busca de uma matriz de custos que consiga melhorar a performance desse classificador. Porém, como descrito anteriormente, um custo fixo não é uma boa estratégia para o evolutivo e potencialmente infinito ambiente de fluxo de dados. Pois ele pode piorar a performance do classificador ao longo do tempo por não possuir a mesma representatividade a qual foi atribuído durante a construção do modelo. Por essa razão, as heurísticas de custos empregadas no CSARF são orientadas a fluxos de dados. Isso quer dizer que, uma vez que o *data stream* evolua e varie a distribuição de classes, as heurísticas também acompanharão essas mudanças de alguma maneira desde que o modelo preditivo seja atualizado constantemente.

No presente trabalho, apresentou-se duas heurísticas de custos: *Picek* e *OzaCosting* (subseção 3.1.1), sendo a segunda uma tentativa de criação de uma heurística baseada na dificuldade do classificador. De forma a avaliar a performance de ambas e cobrir a primeira hipótese do protocolo experimental, realizou-se os experimentos previstos no protocolo. Isto é, um *grid search* foi realizado explorando-se os parâmetros do CSARF e, sendo um desses parâmetros a heurística de custo, escolheu-se o melhor resultado obtido por cada heurística para cada métrica em cada *stream* de dados.

Os melhores resultados obtidos para cada heurística podem ser encontrados na tabela 6. Assim como previsto no protocolo, o potencial das heurísticas foi avaliado em três métricas: *recall* médio, $F1_{score}$ médio e AUC-PR. Evidencia-se por meio do gráfico

[2] A média da métrica calculada para as classes do problema.

Tabela 6 – *Picek* vs *OzaCosting*

	AUC-PR		$F1_{score}$ médio		*Recall* médio	
	Picek	OzaCosting	Picek	OzaCosting	Picek	OzaCosting
santd	**0.148555**	0.144724	**0.584470**	0.524876	**0.723490**	0.545991
elect	0.375969	**0.386288**	**0.898619**	0.896815	**0.898598**	0.893845
airli	**0.713641**	0.712705	**0.666812**	0.666560	**0.667175**	0.666972
gmsc	0.370925	**0.370954**	**0.686781**	0.646573	**0.773720**	0.617911
pozzl	**0.852199**	0.852028	**0.882988**	0.874480	**0.937490**	0.879882
privt	**0.591701**	0.588605	**0.712386**	0.671275	**0.729307**	0.678582
agrwl90	**0.932365**	0.932326	0.914581	**0.914583**	**0.941482**	0.890655
agrwl95	**0.894362**	0.894320	**0.898669**	0.898503	**0.938778**	0.857077
agrwl99	0.704411	**0.705634**	**0.570327**	0.529681	**0.904496**	0.520902
sea90	**0.349223**	0.349061	**0.703458**	0.613027	**0.876815**	0.599683
sea95	**0.202945**	0.202658	**0.618571**	0.537584	**0.875424**	0.530720
sea99	**0.046399**	0.045848	**0.516973**	0.500230	**0.813124**	0.501348
covrt	-	-	**0.904954**	0.904775	**0.923028**	0.899078
poker	-	-	**0.582578**	0.501422	**0.626085**	0.436843

Tabela 7 – Teste de Wilcoxon - *Picek* vs *OzaCosting*

AUC-PR		$F1_{score}$ médio		*Recall* médio	
T	p-value	T	p-value	T	p-value
22	0.1823	1	**0.0012**	0	**0.0009**

(Figura 5) que a heurística de custo *Picek* (Equação 3.4) apresentou os melhores resultados para as métricas em *stream* de dados.

Ao analisarmos as performances das heurísticas (Tabela 6) para a métrica AUC-PR, nota-se que a diferença entre *Picek* e *OzaCosting* não é tão grande. Isso fica evidente quando observado a diferença nos resultados alcançados para as bases de dados *agrwl90*, *agrwl95*, *agrwl99*, *gmsc*, *pozzl*, *sea90*, *sea95* e *sea99* onde a diferença está na terceira para quarta casa decimal. Tendo como base o método de comparação de algoritmos pelo autor em (DEMŠAR, 2006), realizou-se o teste de *Wilcoxon* para verificar se a diferença entre os resultados obtidos é de fato estatisticamente significativa. Para a métrica AUC-PR, o teste de hipótese não paramétrico de *Wilcoxon* aponta que não há diferença estatisticamente significativa com 95% de confiança (Tabela 7).

Para a métrica $F1_{score}$ médio nota-se novamente que a heurística *Picek* dominou quase todos os melhores resultados, tanto para os *streams* binários quanto para os multi-classe. Porém, neste caso observa-se que a diferença da performance obtida entre *Picek* e *OzaCosting* é mais visível, isto é, está presente nas primeiras casas decimais. Para de fato saber se essa diferença é significativa, também realizou-se o teste de *Wilcoxon* que apontou uma diferença estatisticamente significativa com 95% de confiança (Tabela 7).

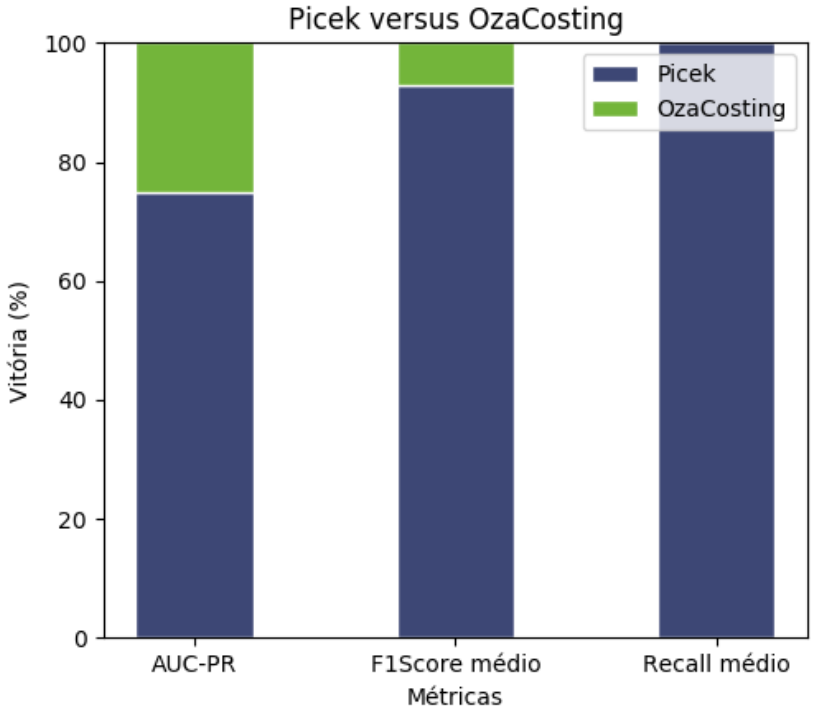


Figura 5 – Percentual de vitória das heurísticas de custo *Picek* e *OzaCosting* em cada métrica para todas as *streams* de dados.

A métrica *recall* médio consiste na média do *recall* obtido em todas as classes do problema. Para esse caso, a heurística *Picek* demonstrou-se a melhor em todos os *stream* de dados analisados. As únicas performances que apresentam baixa diferença entre si estão presentes nas bases com baixo nível de desbalanceamento: *elect* e *airli* (Tabela 6). Por meio do teste de *Wilcoxon*, confirma-se que há diferença estatisticamente significativa com 95% de confiança (Tabela 7).

A diferença de performance nas métricas $F1_{score}$ médio e *recall* médio entre as heurísticas *Picek* e *OzaCosting* pode estar em sua natureza. A heurística *Picek* é baseada na distribuição de classes de uma base de dados e no CSARF ela utiliza uma janela deslizante para calcular dinamicamente os custos conforme o *data stream* evolui e o modelo atualizado. Como demonstrado na seção 3.1.1, essa heurística é sensível à variação de classes que possa ocorrer na janela deslizante. Em contrapartida, o *OzaCosting* baseia-se nos acertos e erros do *ensemble* para cada classe e a sua variação de custo é baixíssima. O custo tende a se estabilizar se os acertos e erros do classificador apresentarem um comportamento com pouca variação, isto é, quando a taxa de erro do classificador é quase estática não variando positiva ou negativamente. De qualquer forma, a hipótese nula de que não há diferença estatisticamente significativa entre as métricas foi rejeitada e portanto, escolheu-se a heurística *Picek* para comparar o CSARF com os demais algoritmos.

Um dos parâmetros do CSARF diz respeito ao tamanho da janela deslizante

(*imbalanced window*). Como descrito no capítulo 3, a janela permite que o método identifique qual é a classe majoritária na amostra contida na janela e também possibilita calcular as penalidades de classificação utilizando a heurística *Picek*. O anexo B traz em detalhes uma análise sobre a exploração desse parâmetro nos experimentos conduzidos.

A seção seguinte descreve os experimentos conduzidos com o CSARF para os *streams* de dados binários. Nela se encontram em detalhes os melhores resultados obtidos pelo método proposto, utilizando a heurística *Picek*, em comparação com o ARF (*baseline*), ARF_{RE} e KUE para cada *data stream* binário analisado.

4.3 Resultados para fluxo de dados binários

Esta seção engloba os resultados e discussões sobre os experimentos realizados nos *streams* de dados binários. Esses conjuntos de dados têm como característica a ordenação temporal, sendo possível interpretá-los como um fluxo de dados, e diferentes níveis de desbalanceamento. Dentre os *streams* de dados descritos na seção 2.5, os seguintes serão abordados na presente seção: *Santander* (*santd*), *Electricity* (*elect*), *Airlines* (*airli*), *Give me some credit* (*gmsc*), *Pozzolo Credit Card Fraud Detection* (*pozzl*), Escoragem de crédito privada (*privt*), *Agrawal Generator* (*agrwl90*, *agrwl95* e *agrwl99*) e *SEA Generator* (*sea90*, *sea95* e *sea99*).

As subseções seguintes descrevem os melhores resultados obtidos pelo CSARF, utilizando a heurística *Picek*, em comparação com o ARF (*baseline*), ARF_{RE} e KUE para cada *data stream* binário. Os parâmetros utilizados em cada método também estão listados e eles seguem as configurações descritas na seção 4.1. Por exemplo, os parâmetros do algoritmo ARF-e100-f75 são equivalentes a utilizar o ARF com um *ensemble* de 100 classificadores e um espaço de *features* de 75%.

4.3.1 Santander - *santd*

Este *stream* de dados está entre os conjuntos de dados reais binários com maior grau de desbalanceamento e contém o maior conjunto de atributos entre os *streams* analisados. O *baseline* alcançado pelo ARF é de 0.144 para AUC-PR, 0.496 para F1_{score} médio e 0.5 para *recall* médio. Os demais resultados e parâmetros de cada método estão descritos nas tabelas 8 e 9, respectivamente.

Ao analisarmos a performance dos algoritmos para a métrica AUC-PR, observa-se que o *baseline* (ARF) apresenta melhores resultados (Tabela 8) que o ARF_{RE} e KUE, sendo o ARF_{RE} construído para lidar com desbalanceamento. Isso evidencia que o ARF consegue apresentar boas performances em conjuntos desbalanceados se considerarmos diferentes limites de decisão para detectar a classe positiva (AUC-PR). No entanto, o método proposto

Tabela 8 – Santander - Resultados

	AUC-PR	$F1_{score}$ médio	*Recall* médio
ARF (*baseline*)	0.144178	0.496995	0.50085
CSARF	**0.148555**	**0.58447**	**0.72349**
ARF_{RE}	0.061419	0.49132	0.566098
KUE	0.083294	0.498441	0.49842

Tabela 9 – Santander - Configurações utilizadas

	AUC-PR	$F1_{score}$ médio	*Recall* médio
ARF	-e100-f75	-e100-f50	-e25-f25
CSARF	-l-e100-f75-i20000	-l-e25-f25-i1000	-g-e75-f25-i1000
ARF_{RE}	-e75-f50	-e25-f25	-e75-f50
KUE	-e50-n40	-e25-n30	-e25-n30

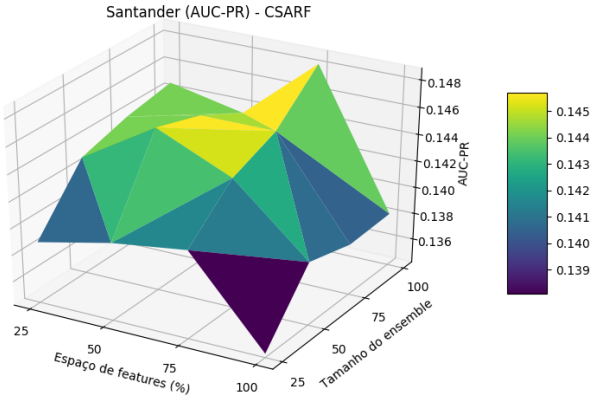
(CSARF) conseguiu superar o resultado apresentado pelo *baseline*. Obtendo-se assim, a melhor performance para esse *data stream*.

As métricas $F1_{score}$ médio e *recall* médio são calculadas considerando o limite de decisão de um classificador em 0.5 (50%). Isso quer dizer que a classe escolhida pelo algoritmo indutor é aquela que possui uma probabilidade acima de 50%. Para ambas as métricas, o CSARF superou o *baseline* e foi ranqueado em primeiro lugar com os melhores resultados (Tabela 8). Observa-se que para a métrica $F1_{score}$ médio, os métodos concorrentes ARF, ARF_{RE} e KUE apresentam performances similares entre eles, isto é, a diferença está na terceira casa decimal. Um cenário destoante pode ser verificado com os resultados obtidos na mesma métrica pelo CSARF e os demais. O método proposto coloca-se a frente nos resultados para a métrica *recall* médio com aproximadamente 44% de ganho sobre o *baseline*.

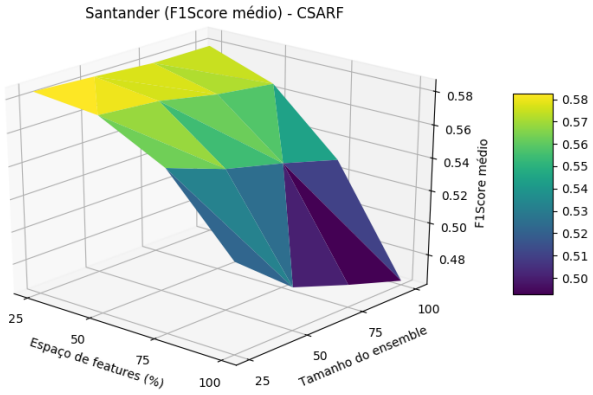
Para um *stream* de dados com cerca de 300 atributos, os métodos CSARF, ARF e ARF_{RE} apresentaram os melhores resultados para AUC-PR explorando o maior número de indutores no *ensemble* (-e 75-100) e o maior espaço de atributos (-f 25%-75%) por indutor (Tabela 9). Essas configurações são diferentes quando analisamos a métrica $F1_{score}$ médio em que os algoritmos CSARF, ARF_{RE} e KUE apresentaram melhores performances com um número reduzido de indutores (-e). A figura 6 ilustra como o tamanho do *ensemble* e espaço de *features* do CSARF influenciam a performance nas métricas AUC-PR (Figura 6a), $F1_{score}$ médio (Figura 6b) e *recall* médio (Figura 6c).

4.3.2 Electricity - *elect*

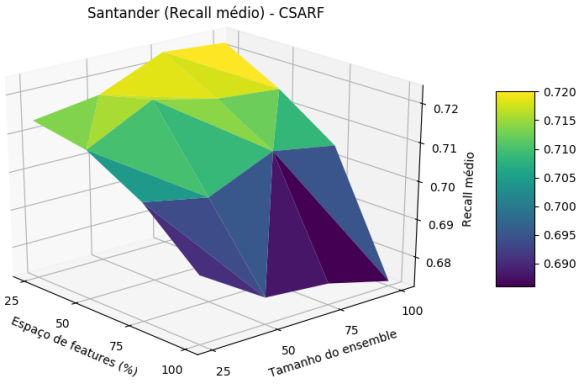
O *stream* de dados *Electricity* apresenta um baixo grau de desbalanceamento, tendo a classe majoritária cerca de 57.54% das instâncias que compõem esse conjunto de dados. O *baseline* resultante dos experimentos com o ARF é de 0.376 para AUC-PR, 0.895 para



(a) AUC-PR



(b) $F1_{score}$ médio



(c) *Recall* médio.

Figura 6 – Resultados obtidos pelo *grid search* realizado na base Santander (*santd*).

$F1_{score}$ médio e 0.892 para *recall* médio. Os demais resultados e parâmetros das melhores performances alcançadas estão detalhados nas tabelas 10 e 11, respectivamente.

A performance do CSARF na métrica AUC-PR para esse conjunto de dados ficou abaixo do *baseline* (Tabela 10). O melhor resultado obtido pelo CSARF, utilizando a heurística *Picek*, ficou 0.1% aquém do *baseline* enquanto que o KUE apresentou um resultado 8.3% acima do ARF. Neste caso, os dois métodos orientados a desbalanceamento, CSARF e ARF_{RE}, apresentaram as piores performances mas ainda assim muito próximas do baseline.

Ao analisarmos as performances nas métricas $F1_{score}$ e *recall* médio, nota-se que ambos os métodos orientados ao desbalanceamento disputam os primeiros lugares. Apesar da pequena diferença acima do *baseline*, o método proposto ranqueia em segundo e primeiro lugar nas métricas $F1_{score}$ e *recall* médio, respectivamente. Observa-se que o KUE e o ARF, os quais haviam apresentado os melhores resultados para métrica AUC-PR, ocuparam os últimos lugares para essas duas métricas avaliadas.

Por mais que o CSARF tenha apresentado uma performance abaixo do *baseline* para esse conjunto de dados, a diferença foi pequena e evidencia que o método proposto também pode trabalhar em *streams* com baixo grau de desbalanceamento. Os parâmetros dos

Tabela 10 – Electricity - Resultados

	AUC-PR	$F1_{score}$ médio	*Recall* médio
ARF (*baseline*)	0.376280	0.895220	0.892158
CSARF	0.375970	0.898620	**0.898599**
ARF_{RE}	0.375208	**0.900129**	0.898343
KUE	**0.407486**	0.767987	0.763885

Tabela 11 – Electricity - Configurações utilizadas

	AUC-PR	$F1_{score}$ médio	*Recall* médio
ARF	-e100-f100	-e50-f100	-e50-f100
CSARF	-g-e100-f100-i5000	-l-e50-f100-i1000	-g-e50-f100-i1000
ARF_{RE}	-e100-f75	-e50-f100	-e50-f100
KUE	-e25-n40	-e25-n30	-e25-n30

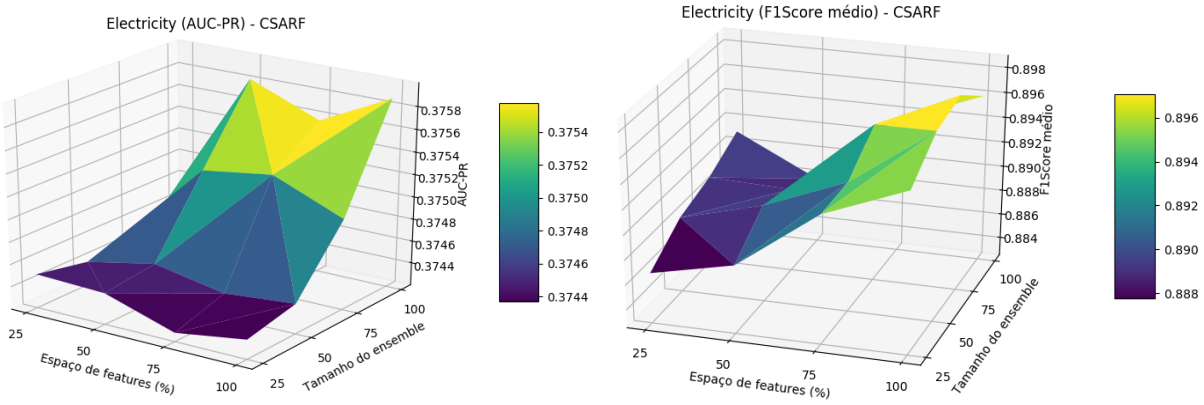
métodos (Tabela 11), oriundos do ARF, que obtiveram os melhores resultados utilizaram o máximo possível do espaço de *features* por indutor (-f 100%). Observa-se que nos gráficos explorando os parâmetros do CSARF (Figura 7) que quanto maior o número de *features* e o tamanho do *ensemble*, melhor é o resultado alcançado em cada uma das métricas. Para esse conjunto de dados com 9 atributos, construir a diversidade dos classificadores base do *ensemble* a partir do subespaço de *features* não resultou nas melhores performances.

4.3.3 Airlines - *airli*

Este conjunto de dados, assim como *elect*, possui um baixo grau de desbalanceamento com uma distribuição de [55.46%, 44.54%]. Os melhores resultados obtidos pelo ARF neste *stream* de dados são utilizados como *baseline*, então temos 0.712 em AUC-PR, 0.666 em $F1_{score}$ médio e 0.667 em *recall* médio. Os resultados obtidos estão detalhados na tabela 12 e os parâmetros utilizados em cada um dos algoritmos estão descritos na tabela 13.

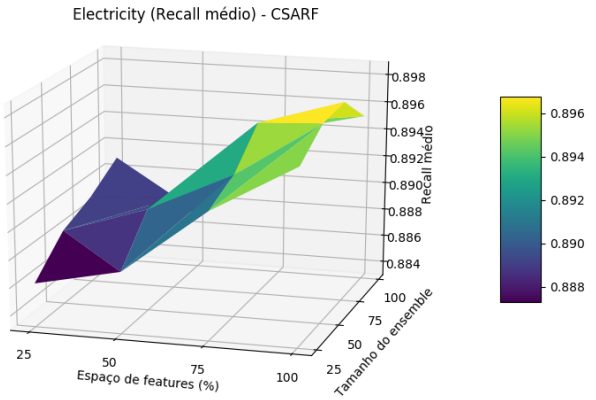
Dentre os resultados obtidos para a métrica AUC-PR, o CSARF apresenta a melhor performance dentre os métodos comparados (Tabela 12). Ressalta-se que assim como no *stream* de dados *elect*, as performances resultantes dos algoritmos orientados a desbalanceamento (CSARF e ARF_{RE}) estão bem próximas (acima ou abaixo) do *baseline*. Somente o método KUE denota uma diferença acima de 5% para pior em relação ao ARF.

Os experimentos conduzidos com o método proposto não apresentaram resultados acima do *baseline* tanto para a métrica $F1_{score}$ médio quanto para o *recall* médio (Tabela 12). No entanto, a diferença entre o CSARF e o *baseline* é de 0.02% e 0.03% para as métricas $F1_{score}$ e *recall* médio, respectivamente. Em ambientes que têm como característica um baixo nível de desbalanceamento, o custo envolvido ao se errar a classe minoritária tende a ser pequeno. Por essa razão, a diferença entre o método proposto e o *baseline*,



(a) AUC-PR

(b) $F1_{score}$ médio



(c) *Recall* médio

Figura 7 – Resultados obtidos pelo *grid search* realizado na base *Electricity* (*elect*).

Tabela 12 – Airlines - Resultados

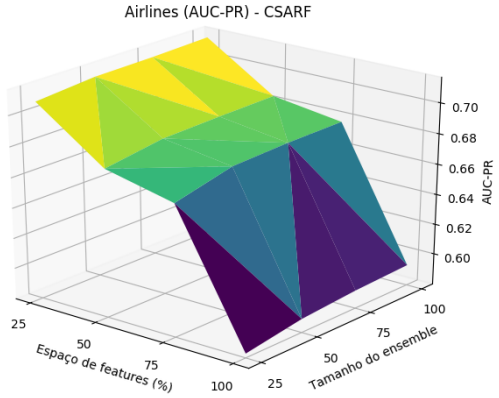
	AUC-PR	$F1_{score}$ médio	*Recall* médio
ARF (*baseline*)	0.712789	0.666931	0.667347
CSARF	**0.713641**	0.666812	0.667175
ARF_{RE}	0.711073	0.675104	**0.674751**
KUE	0.667052	**0.644636**	0.644402

para esse conjunto de dados, inclina-se a ser pouco significativa.

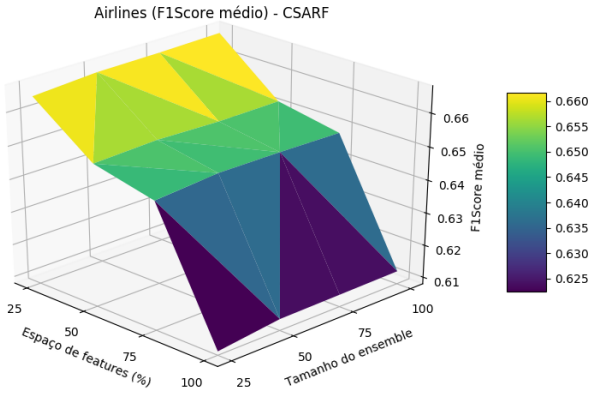
Apesar do *stream* de dados *airli* apresentar um número de atributos e grau de desbalanceamento semelhante a *elect*, as melhores configurações geradas pelo CSARF não apresentam o mesmo comportamento que as listadas na seção anterior. Observa-se que para esse conjunto de dados, a configuração do CSARF com o maior número de indutores (-e 100) combinado com menor número de *features* (-f 25%) denota o melhor resultado para as três métricas (Figura 8). E isso pode ser notado também no método ARF_{RE} (Tabela 13), que apresenta configurações semelhantes com os melhores resultados nas métricas $F1_{score}$ e *recall* médio (Tabela 12).

Tabela 13 – Airlines - Configurações utilizadas

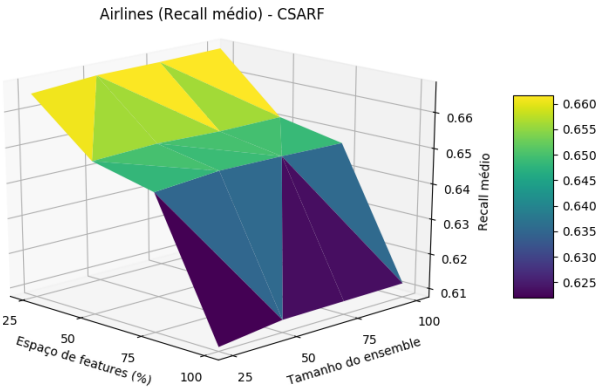
	AUC-PR	$F1_{score}$ médio	*Recall* médio
ARF	-e75-f25	-e75-f25	-e75-f25
CSARF	-t-e100-f25-i1000	-t-e100-f25-i1000	-t-e100-f25-i1000
ARF_{RE}	-e100-f25	-e100-f25	-e100-f25
KUE	-e75-n10	-e25-n10	-e25-n10



(a) AUC-PR



(b) $F1_{score}$ médio



(c) *Recall* médio

Figura 8 – Resultados obtidos pelo *grid search* realizado na base *Airlines* (*airli*).

4.3.4 Give me some credit - *gmsc*

Este *stream* de dados descreve o problema de classificar clientes que podem se tornar inadimplentes dentro de dois anos e é por natureza um problema desbalanceado. O *baseline* alcançado pelo ARF para esse conjunto de dados foi de 0.37 na métrica AUC-PR, 0.587 para $F1_{score}$ médio e 0.561 para *recall* médio. Os resultados dos experimentos conduzidos com os outros métodos estão detalhados na tabela 14 e os parâmetros das configurações que obtiveram as melhores performances estão descritas na tabela 15.

Para esse conjunto de dados o CSARF e KUE obtiveram as melhores performances na métrica AUC-PR, acima do *baseline* (Tabela 14). Além do mais, o método proposto apresentou bons resultados para as métricas $F1_{score}$ e *recall* médio com um ganho de 16.8% e 37.8% sobre o *baseline*. O método proposto não só foi quase o melhor em todas as

Tabela 14 – GMSC - Resultados

	AUC-PR	$F1_{score}$ médio	*Recall* médio
ARF (*baseline*)	0.370618	0.587893	0.561478
CSARF	0.370926	**0.686781**	**0.773720**
ARF_{RE}	0.341033	0.671809	0.757321
KUE	**0.370939**	0.601865	0.571240

Tabela 15 – GMSC - Configurações utilizadas

	AUC-PR	$F1_{score}$ médio	*Recall* médio
ARF	-e100-f75	-e25-f100	-e25-f100
CSARF	-t-e100-f75-i5000	-l-e50-f50-i5000	-g-e100-f50-i20000
ARF_{RE}	-e100-f100	-e100-f50	-e50-f100
KUE	-e100-n20	-e25-n40	-e25-n40

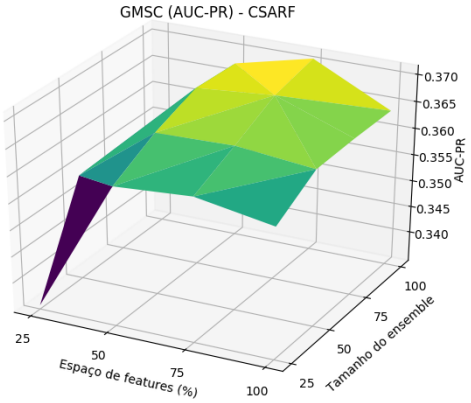
métricas analisadas, como também apresentou ganhos expressivos sobre o *baseline* ARF. Isso significa que, em um problema como o presente no *stream* de dados *gmsc*, as decisões do classificador podem resultar no aumento da entrada de bons clientes e uma redução de inadimplentes.

Os parâmetros dos métodos (Tabela 15) que geraram os resultados descritos na tabela 14 evidenciam que há uma convergência entre os algoritmos baseados no ARF. Nota-se que esses métodos tendem a utilizar o maior número de indutores e porcentagem do espaço de atributos para a métrica AUC-PR. Isso fica evidente quando analisamos o gráfico de exploração dos parâmetros do CSARF (Figura 9a). Já para as métricas $F1_{score}$ médio e *recall* médio, não parece haver uma convergência nos parâmetros entre os algoritmos avaliados (Figuras 9b e 9c).

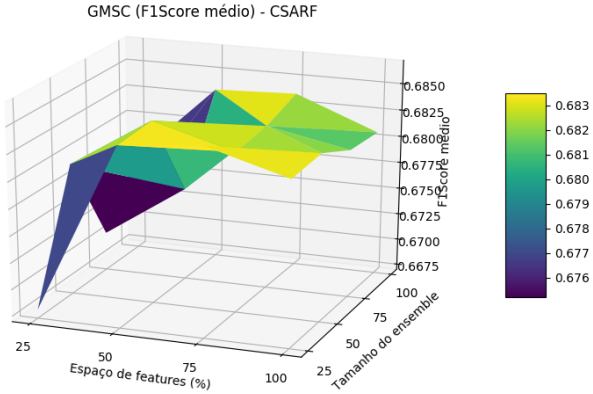
4.3.5 Pozzolo credit card fraud detection - *pozzl*

O stream de dados pozzl é o que possui o maior grau de desbalanceamento dentre todos os conjunto de dados analisados. O *baseline* para esse *data stream* é de 0.853 na métrica AUC-PR, 0.874 para $F1_{score}$ médio e 0.879 para *recall* médio. Os resultados estão descritos em detalhes na tabela 16 e os parâmetros de cada método que gerou os melhores resultados estão na tabela 17.

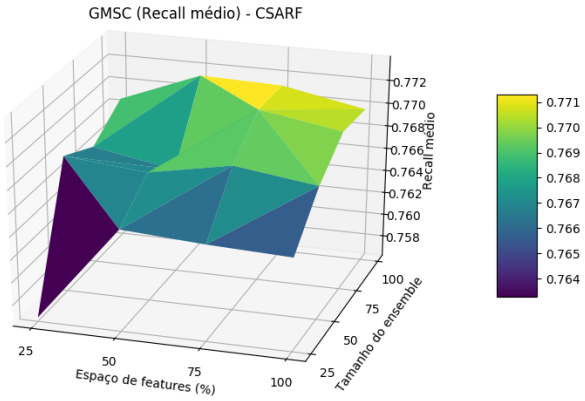
Mesmo apresentando uma distribuição em que a classe majoritária engloba 99.8% das instâncias, o *baseline* disputa o primeiro lugar com o CSARF em quase todas as métricas avaliadas. De fato, o método proposto não ultrapassa o *baseline* na métrica AUC-PR mas apresenta ganhos de 15.18% e 30.4% em relação ao ARF_{RE} e KUE (Tabela 16), respectivamente. Já para as métricas $F1_{score}$ e *recall* médio, o CSARF detém os melhores resultados. Observa-se que ao analisar o método proposto contra o ARF_{RE} na métrica $F1_{score}$ médio, o ganho do CSARF chega a ser de 10.7%. O ARF_{RE} só ultrapassou



(a) AUC-PR



(b) $F1_{score}$ médio



(c) *Recall* médio

Figura 9 – Resultados obtidos pelo *grid search* realizado na base *Give me some credit* (*gmsc*).

Tabela 16 – Pozzl - Resultados

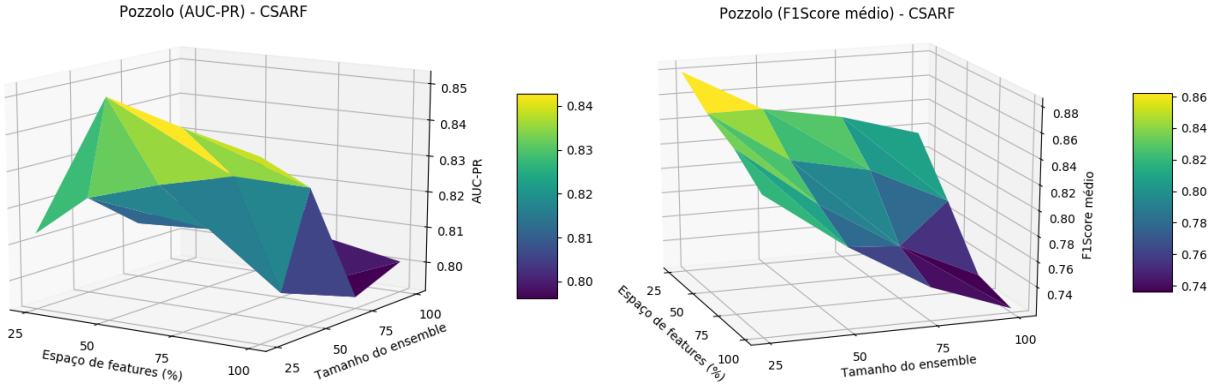
	AUC-PR	$F1_{score}$ médio	*Recall* médio
ARF (*baseline*)	**0.853598**	0.874336	0.879836
CSARF	0.852200	**0.882989**	**0.937490**
ARF_{RE}	0.739876	0.797364	0.904755
KUE	0.653455	0.541690	0.786909

o *baseline* na métrica *recall* médio enquanto que o KUE não conseguiu lidar com o severo desbalanceamento e foi o pior método em todas as métricas avaliadas para esse *data stream*.

Para as métricas AUC-PR e $F1_{score}$ médio, os métodos derivados do ARF convergiram sobre o tamanho do *ensemble* (-e), variando entre 25 e 75 indutores, e o tamanho espaço de atributos utilizado por indutor (-f), variando entre 25% a 50% de um total de 31 atributos (Tabela 17). Ao analisarmos os gráficos do *grid search* realizado no CSARF (Figura 10), nota-se que para as métricas AUC-PR (Figura 10a) e $F1_{score}$ médio (Figura 10b) os experimentos com menor número de indutores apresentam os melhores resultados.

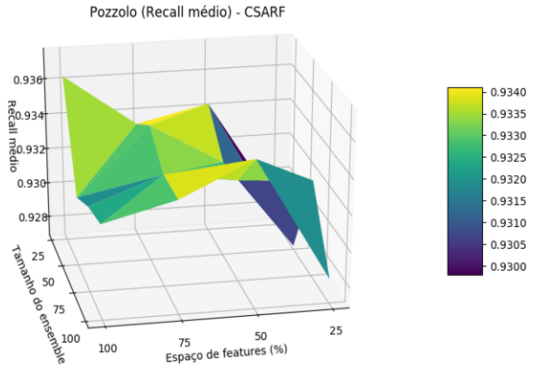
Tabela 17 – Pozzl - Configurações utilizadas

	AUC-PR	$F1_{score}$ médio	*Recall* médio
ARF	-e25-f50	-e25-f25	-e75-f75
CSARF	-t-e25-f50-i10000	-l-e25-f25-i10000	-g-e100-f50-i20000
ARF_{RE}	-e75-f50	-e25-f50	-e25-f25
KUE	-e100-n20	-e100-n20	-e75-n20



(a) AUC-PR

(b) $F1_{score}$ médio



(c) *Recall* médio

Figura 10 – Resultados obtidos pelo *grid search* realizado na base *Pozzolo credit card fraud detection* (*pozzl*).

4.3.6 Escoragem de crédito privada - *privt*

Este *stream* de dados possui dados privados e o seu desbalanceamento é intrínseco, isto é, faz parte da definição do problema o qual ele descreve. Os experimentos conduzidos resultaram nos valores 0.587 para AUC-PR, 0.672 para $F1_{score}$ médio e 0.653 para *recall* médio como *baseline*. A tabela 18 resume os resultados obtidos pelos modelos gerados com os parâmetros descritos na tabela 19.

O método proposto ultrapassou o *baseline* e destacou-se como o melhor resultado dentre os demais na métrica AUC-PR. O ganho sobre o *baseline*, que ficou em segundo lugar, é de aproximadamente 0.6%. Observa-se que todos os resultados na métrica AUC-PR estão próximos um dos outros, isso evidencia que tanto o *baseline* quanto os demais

Tabela 18 – Privt - Resultados

	AUC-PR	$F1_{score}$ médio	*Recall* médio
ARF (*baseline*)	0.587673	0.672860	0.653421
CSARF	**0.591702**	0.712387	**0.729307**
ARF_{RE}	0.586735	**0.716933**	0.719444
KUE	0.581394	0.643993	0.627597

Tabela 19 – Privt - Configurações utilizadas

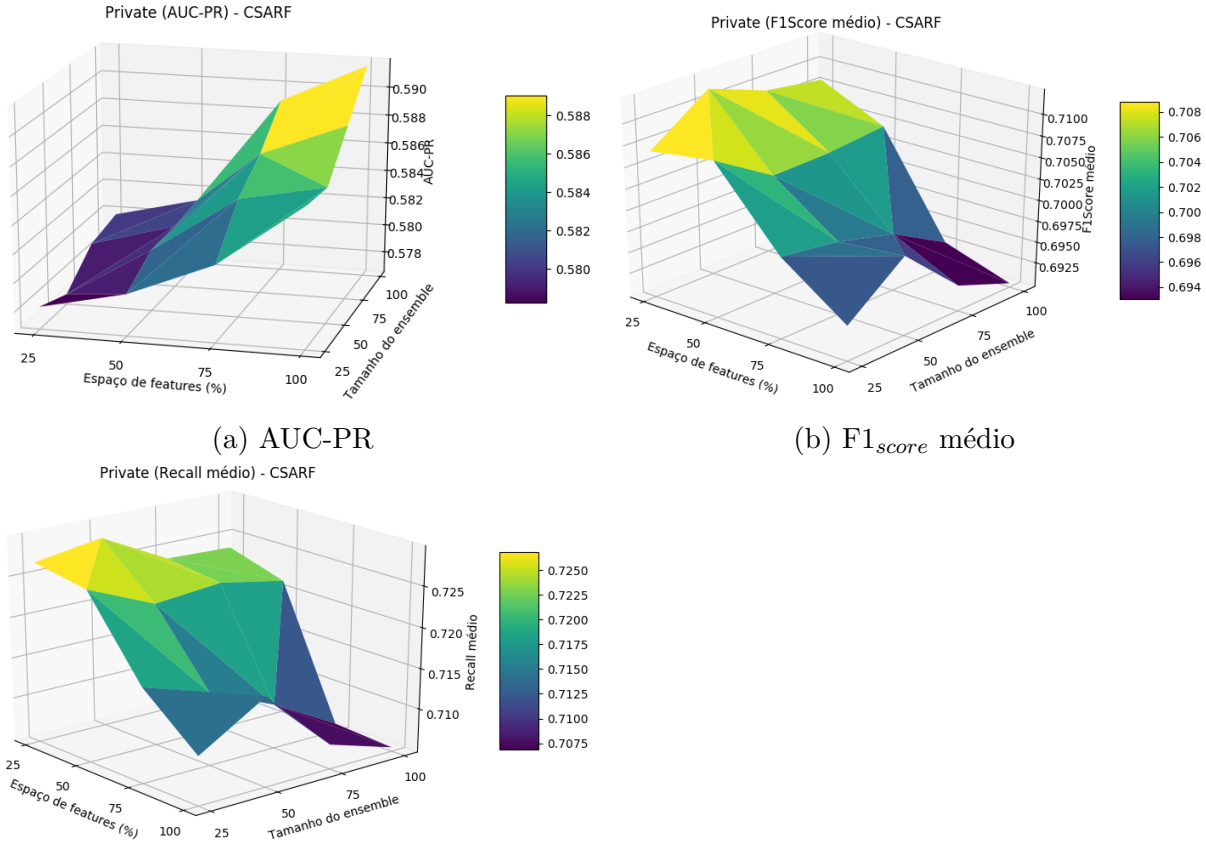
	AUC-PR	$F1_{score}$ médio	*Recall* médio
ARF	-e100-f100	-e25-f25	-e25-f25
CSARF	-l-e100-f100-i1000	-l-e50-f25-i20000	-g-e25-f25-i20000
ARF_{RE}	-e100-f100	-e75-f25	-e75-f25
KUE	-e25-n40	-e25-n30	-e25-n30

métodos avaliados também possuem um potencial para alguns ambientes desbalanceados (Tabela 18). Porém, se compararmos o CSARF na métrica $F1_{score}$ médio o ganho sobre o *baseline* é de 5.8% e a diferença do método proposto para o primeiro lugar, que também é um algoritmo orientado a data *streams* desbalanceados, é de apenas 0.6%. Esse conjunto de dados é mais um em que o método proposto quase dominou os melhores resultados em todas as métricas. O CSARF apresentou um ganho de 11.6% sobre o *baseline* na métrica *recall* médio e conquistou o melhor resultado dentre os algoritmos avaliados.

Para o fluxo de dados *privt*, nota-se que as configurações que obtiveram os melhores resultados do CSARF exploraram as três estratégias de uso do *threshold* (Tabela 19). E novamente, os métodos baseados no ARF convergem nos parâmetros para a obtenção da melhor performance na métrica AUC-PR. Observa-se que esse comportamento ocorre tanto em *data streams* desbalanceados como o *privt* e outros com baixo grau de desbalanceamento como o *elect* (Tabela 11). A figura 11a ilustra como o aumento do tamanho do *ensemble* e espaço de *features* impacta na métrica AUC-PR e como o inverso ocorre nas métricas $F1_{score}$ médio (Figura 11b) e *recall* médio (Figura 11c).

4.3.7 Agrawal Generator - *agrwl*

Este gerador sintético é capaz de construir instâncias com nove atributos que definem uma das duas classes pertencentes ao problema. Por estar implementado no MOA, pode-se induzir um grau de desbalanceamento e ruído. Dessa forma, é possível visualizar como o CSARF lida com o mesmo conjunto de dados apresentando diferentes níveis de desbalanceamento. Para os experimentos, três configurações foram definidas: *agrwl90*, *agrwl95* e *agrwl99*. Na primeira configuração a distribuição das classes foi definida como 90% pertencendo à classe majoritária e as seguintes 95% e 99%, respectivamente, com os outros parâmetros nas configurações *default*. Os resultados alcançados pelo *baseline* e



(a) AUC-PR

(b) $F1_{score}$ médio

(c) *Recall* médio

Figura 11 – Resultados obtidos pelo *grid search* realizado na base Escoragem de crédito privada (*privt*).

os demais métodos estão descritos na tabela 20 e todos os parâmetros de cada melhor resultado obtido nas métricas avaliadas estão detalhados na tabela 21.

Os experimentos com o gerador *agrwl* demonstram que o método proposto superou o *baseline* em todas as métricas analisadas (Tabela 20). Para a métrica AUC-PR, nota-se que o CSARF apresentou melhores performances quando o desbalanceamento foi intensificado (*agrwl95* e *agrwl99*) e, se posicionou a frente do ARF_{RE} com ganhos expressivos. Porém, destaca-se nesse experimento o algoritmo KUE que detém as melhores performances em AUC-PR e $F1_{score}$ médio para esse gerador sintético. Quando avaliamos o poder de discriminação dos métodos pela métrica $F1_{score}$ médio, observa-se que o CSARF apresenta melhores resultados acima do *baseline* e do ARF_{RE}, com um ganho de até 13% na configuração com o maior grau de desbalanceamento. A estratégia de utilizar reamostragem baseada na distribuição de classes impacta de forma positiva quando avaliamos os métodos pela métrica *recall* médio. O ARF_{RE} apresenta resultados com ganhos de 0.27% e 0.22% sob o método proposto para as configurações *agrwl90* e *agrwl95*, respectivamente. No entanto, o CSARF obtém uma performance melhor, com até 75.6% e 79.9% acima do ARF_{RE} e do *baseline* na métrica *recall* médio, respectivamente, quando o grau de desbalanceamento é intensificado (*agrwl99*).

Tabela 20 – Agrwl - Resultados

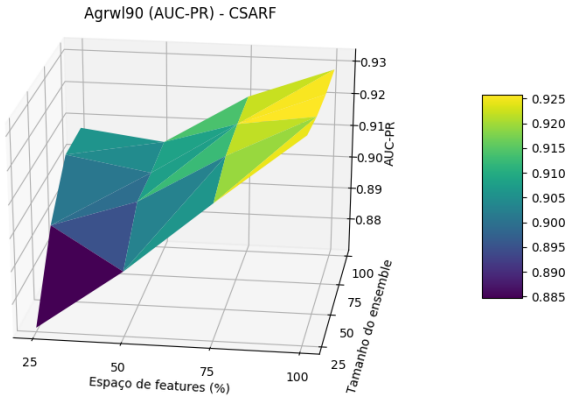
agrwl90			
Algoritmos	AUC-PR	$F1_{score}$ médio	*Recall* médio
ARF (*baseline*)	0.929712	0.912158	0.886931
CSARF	0.932366	0.914581	0.941483
ARF_{RE}	0.935965	0.887804	**0.944065**
KUE	**0.948379**	**0.922401**	0.892025
agrwl95			
ARF (*baseline*)	0.892025	0.896260	0.856869
CSARF	0.894363	0.898670	0.938778
ARF_{RE}	0.865450	0.825333	**0.940927**
KUE	**0.915787**	**0.912873**	0.883858
agrwl99			
ARF (*baseline*)	0.700724	0.503172	0.502679
CSARF	0.704412	0.570327	**0.904496**
ARF_{RE}	0.031940	0.519080	0.515065
KUE	**0.823487**	**0.891223**	0.841767

Para esse *stream* de dados e suas diferentes configurações, nota-se que os métodos oriundos do ARF (CSARF e ARF_{RE}) não convergiram na escolha de parâmetros para maximizar o ganho em todas as métricas analisadas (Tabela 21). No *stream* de dados *agrwl99*, o CSARF atingiu o seu melhor resultado na métrica AUC-PR utilizando o maior tamanho de *ensemble* (-e 100) e o menor percentual do espaço de atributos (-f 25%). O mesmo pode ser observado no seu método de origem, o ARF. Porém, ao analisarmos as configurações para a métrica $F1_{score}$ médio, nota-se que os resultados obtidos pelo CSARF detém parâmetros que buscam explorar mais o espaço de atributos com uma quantidade reduzida de membros no *ensemble* como ilustram as figuras 12b, 13b, e 14b. Isso evidencia que dependendo da métrica objetivo a ser maximizada, os parâmetros de um mesmo método podem variar bastante em um mesmo conjunto de dados. Neste experimento, somente o KUE apresentou configurações semelhantes nas métricas em que se posicionou em primeiro lugar (AUC-PR e $F1_{score}$ médio).

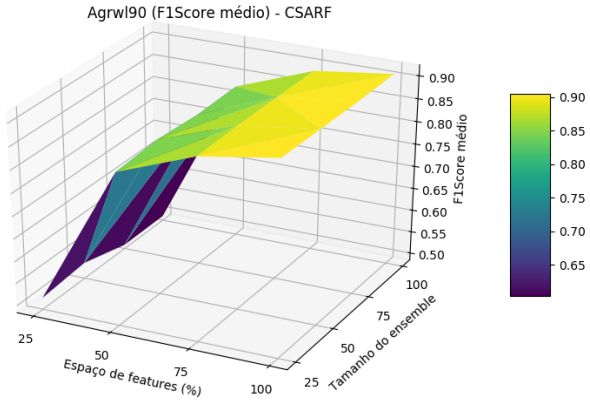
A estratégia de se utilizar um *grid search* em cada *stream* de dados analisado não só proporciona alcançar os melhores resultados de um método, como também possibilita visualizar como a variação dos parâmetros impacta no desempenho do classificador (Figuras 12, 13 e 14). Observa-se que o método proposto apresenta o mesmo comportamento em cada métrica para os *streams* de dados *agrwl90* (Figura 12) e *agrlw95* (Figura 13). Somente no *data stream* mais desbalanceado (Figura 14) que a busca pela melhor performance se comporta de maneira diferente.

Tabela 21 – Agrwl - Configurações utilizadas

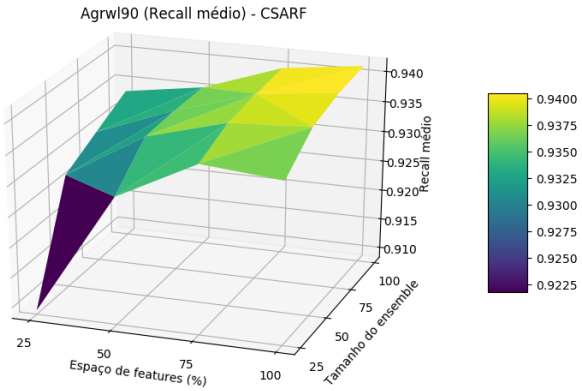
agrwl90			
Algoritmos	AUC-PR	$F1_{score}$ médio	*Recall* médio
ARF	-e50-f100	-e25-f100	-e25-f100
CSARF	-t-e25-f100-i1000	-t-e25-f100-i20000	-g-e100-f100-i20000
ARF_{RE}	-e50-f100	-e100-f50	-e100-f100
KUE	-e100-n30	-e100-n40	-e75-n10
agrwl95			
ARF	-e75-f100	-e75-f100	-e75-f100
CSARF	-g-e75-f100-i20000	-t-e25-f100-i1000	-g-e100-f100-i20000
ARF_{RE}	-e100-f100	-e100-f75	-e50-f100
KUE	-e100-n30	-e50-n40	-e50-n40
agrwl99			
ARF	-e100-f25	-e75-f100	-e75-f100
CSARF	-t-e100-f25-i1000	-g-e25-f50-i20000	-g-e100-f25-i5000
ARF_{RE}	-e75-f100	-e25-f100	-e25-f100
KUE	-e100-n20	-e50-n20	-e100-n20



(a) AUC-PR

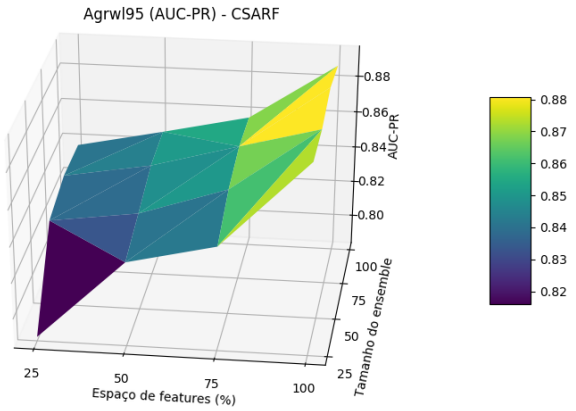


(b) $F1_{score}$ médio

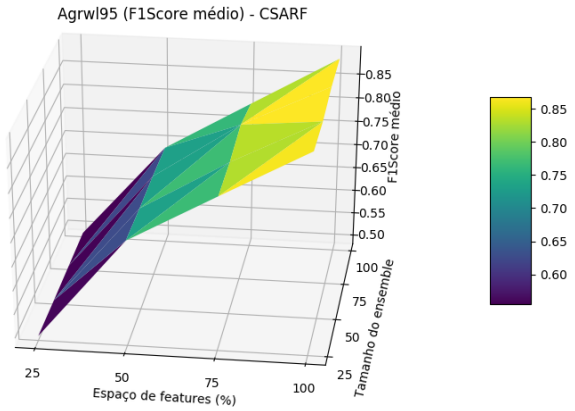


(c) *Recall* médio

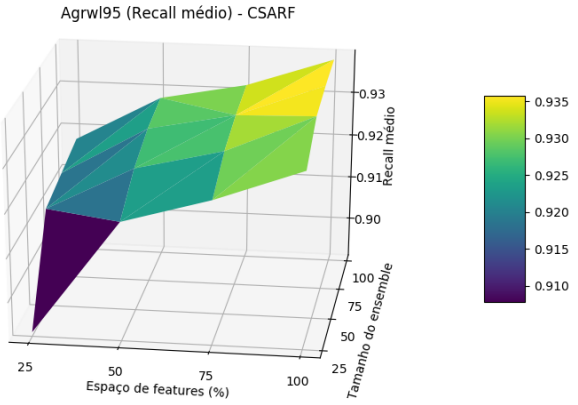
Figura 12 – Resultados obtidos pelo *grid search* realizado na base *Agrawal Generator* (*agrwl90*).



(a) AUC-PR



(b) $F1_{score}$ médio



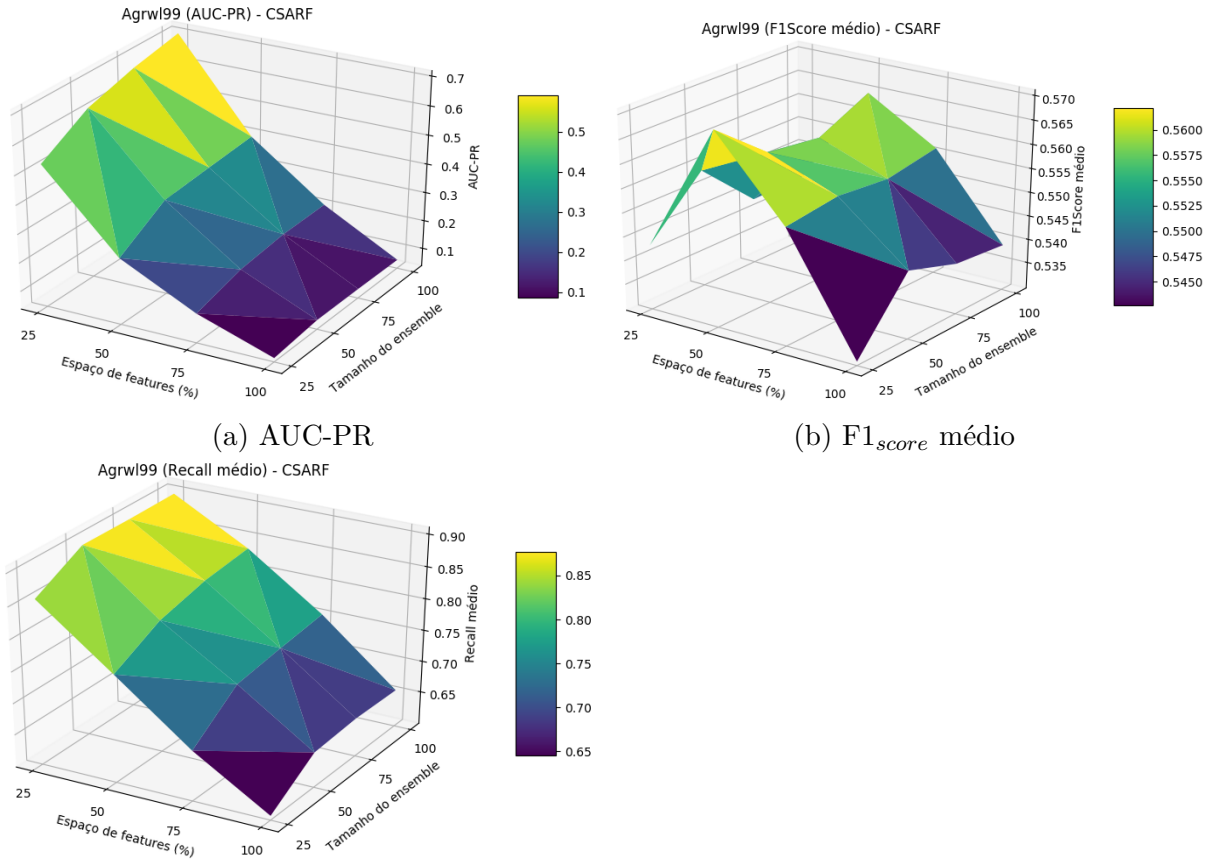
(c) *Recall* médio

Figura 13 – Resultados obtidos pelo *grid search* realizado na base *Agrawal Generator* (*agrwl95*).

4.3.8 SEA Generator - *sea*

O *SEA* é um gerador apto a produzir *streams* com instâncias caracterizadas com três atributos contínuos, sendo um deles irrelevante para classificação. Para avaliar o desempenho do CSARF, construiu-se três configurações semelhantes às descritas na subseção anterior. Os métodos serão avaliados em três *streams* de dados com níveis de desbalanceamento de 90% a 99% das instâncias pertencendo a classe majoritária: *sea90*, *sea95* e *sea99*. O experimento conduzido gerou resultados que estão detalhados na tabela 22 e os parâmetros de cada um dos métodos que gerou o melhor resultado estão descritos na tabela 23.

O método proposto ultrapassou o *baseline* em quase todas as métricas e níveis de desbalanceamento, ficando apenas atrás, por menos de 0.001%, no *stream sea95* para a métrica AUC-PR (Tabela 22). Diferentemente do gerador anterior, o algoritmo KUE ficou entre os piores colocados em todas as configurações e métricas avaliadas. Uma razão por trás disso está na própria natureza do método, o gerador *sea* possui um número de *features* pequeno e isso reduz a diversidade do *ensemble* quando o KUE propõem ao selecionar uma dimensão aleatória de espaço de atributos para cada árvore. Ao analisarmos as



(a) AUC-PR

(b) $F1_{score}$ médio

(c) *Recall* médio

Figura 14 – Resultados obtidos pelo *grid search* realizado na base *Agrawal Generator* (*agrwl99*).

performances dos métodos na métrica AUC-PR, nota-se que todos apresentam dificuldades e que as performances obtidas estão bem próximas, com uma diferença de 0.04% entre o primeiro e último colocado para o *data stream sea90*. Já nas métricas $F1_{score}$ médio e *recall* médio, os algoritmos orientados a desbalanceamento (CSARF e ARF_{RE}) destacam-se com os primeiros lugares. O método proposto detém resultados com ganhos de até 38% sobre o *baseline* na métrica $F1_{score}$ médio (*sea90*). No entanto, essa diferença diminui quando intensifica-se o desbalanceamento (*sea99*). Observa-se que o método ARF_{RE} conquistou majoritariamente o primeiro lugar na métrica $F1_{score}$ médio mas o seu ganho sobre o CSARF é de 0.09%, 0.01% e 1.7% para os *streams sea90*, *sea95* e *sea99*, respectivamente. Em contrapartida, o inverso ocorre quando analisamos as performances na métrica *recall* médio. Nesse caso, o método proposto apresentou os melhores resultados em todas as configurações e manteve um ganho de 34.8% sob o segundo lugar (ARF_{RE}) no *stream* com maior grau de desbalanceamento (*sea99*).

A utilização de um tamanho reduzido do *ensemble* para maximizar a performance na métrica AUC-PR só é observável no método proposto (Tabela 23). Os demais algoritmos optaram pelo maior número de indutores possíveis inclusive quando o desbalanceamento

Tabela 22 – Sea - Resultados

Algoritmos	AUC-PR	$F1_{score}$ médio	*Recall* médio
sea90			
ARF (*baseline*)	0.348509	0.509086	0.517692
CSARF	**0.349223**	0.703459	**0.876816**
ARF_{RE}	0.348759	**0.704117**	0.875295
KUE	0.347856	0.486684	0.505441
sea95			
ARF (*baseline*)	0.203097	0.494348	0.503386
CSARF	0.202945	0.618572	**0.875425**
ARF_{RE}	**0.203688**	**0.618673**	0.873595
KUE	0.196592	0.503428	0.504209
sea99			
ARF (*baseline*)	0.046017	0.499960	0.501252
CSARF	**0.046400**	0.516973	**0.813124**
ARF_{RE}	0.041893	**0.525844**	0.603045
KUE	0.041568	0.500454	0.502485

intensifica (*sea95* e *sea99*). Esse comportamento é observado no ARF em todas as métricas avaliadas.

Ao analisarmos os gráficos gerados pelo *grid search* do método proposto (Figuras 15, 16 e 17), nota-se que somente na configuração *sea99* houve mais variação na performance obtida ao variar os parâmetros explorados (Figura 17). Isso fica evidente nas figuras em todas os *streams* de dados para a métrica AUC-PR e $F1_{score}$ médio. Por outro lado, a métrica *recall* médio, que é a média entre o *recall* da classe majoritária e minoritária, é mais sensível à alteração dos parâmetros conforme os gráficos ilustram (Figura 15c, 16c, 17c). Essa estabilidade na performance mesmo com a variação de parâmetros evidencia que os resultados obtidos podem representar o limite máximo do método proposto para os *streams* de dados *sea90* e *sea95*.

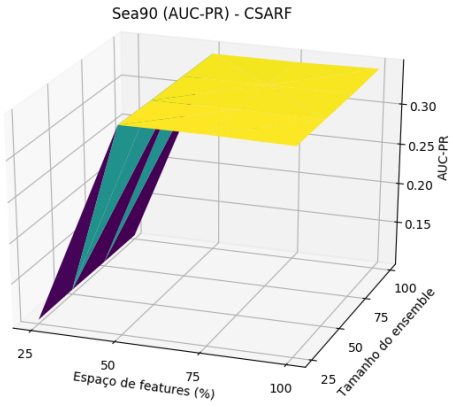
A próxima seção descreve em detalhes os experimentos conduzidos com o método proposto e demais algoritmos nos *streams* de dados multi-classe. Em seguida, na seção 4.5 realiza-se uma análise estatística com todos os resultados obtidos e verifica-se as hipóteses formuladas no protocolo experimental.

4.4 Resultados para fluxo de dados multi-classe

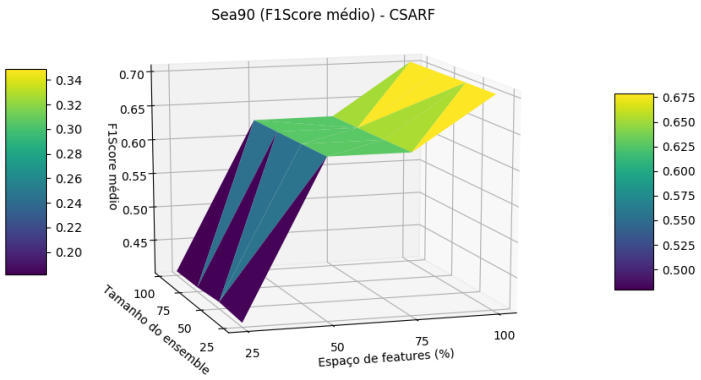
A presente seção descreve em detalhes os resultados e discussões sobre os experimentos conduzidos com os fluxos de dados multi-classe. Assim sendo, dentre os streams de dados listados na seção 2.5, os seguintes serão abordados nesta seção: *Poker Hands*

Tabela 23 – Sea - Configurações utilizadas

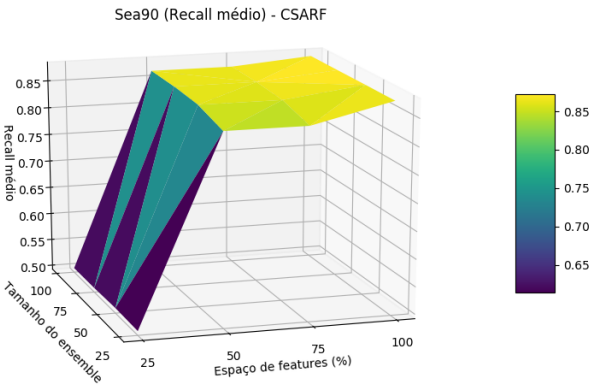
Algoritmos	AUC-PR	$F1_{score}$ médio	*Recall* médio
		sea90	
ARF	-e25-f50	-e25-f50	-e25-f50
CSARF	-l-e25-f50-i5000	-l-e25-f100-i20000	-g-e50-f100-i20000
ARF_{RE}	-e25-f100	-e100-f100	-e75-f100
KUE	-e50-n20	-e25-n20	-e25-n20
		sea95	
ARF	-e25-f100	-e100-f50	-e100-f50
CSARF	-g-e25-f100-i5000	-l-e75-f100-i5000	-g-e100-f100-i10000
ARF_{RE}	-e50-f50	-e25-f50	-e25-f100
KUE	-e100-n40	-e25-n10	-e25-n10
		sea99	
ARF	-e75-f100	-e100-f100	-e100-f50
CSARF	-l-e25-f100-i20000	-l-e25-f100-i1000	-g-e100-f100-i20000
ARF_{RE}	-e100-f100	-e75-f100	-e25-f100
KUE	-e100-n30	-e75-n20	-e50-n20



(a) AUC-PR

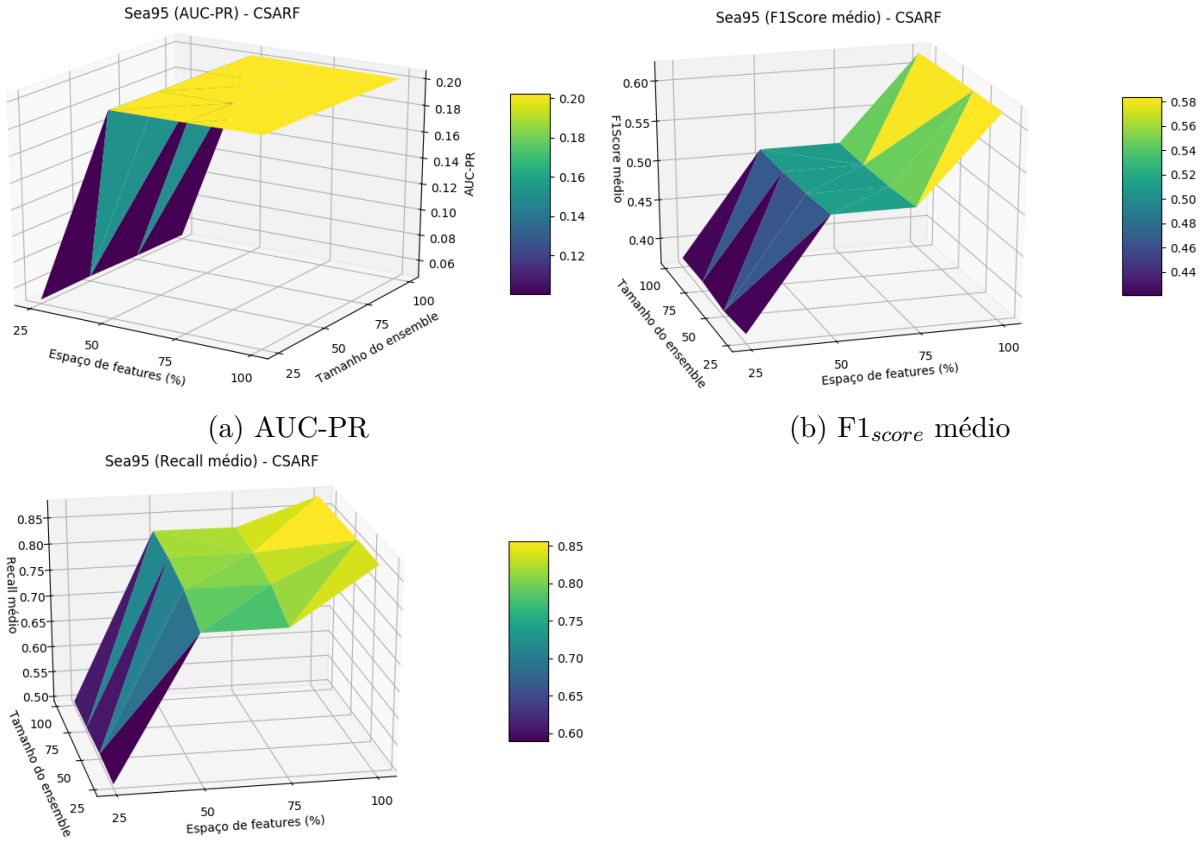


(b) $F1_{score}$ médio



(c) *Recall* médio

Figura 15 – Resultados obtidos pelo *grid search* realizado na base *Sea Generator* (*sea90*).



(a) AUC-PR

(b) $F1_{score}$ médio

(c) *Recall* médio

Figura 16 – Resultados obtidos pelo *grid search* realizado na base *Sea Generator* (*sea95*).

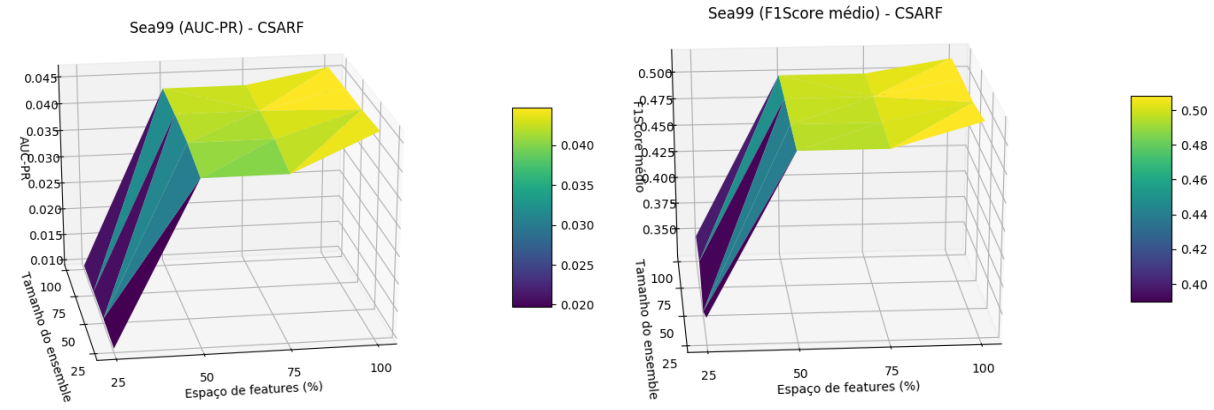
(*poker*) e *Forest CoverType* (*covrt*).

As subseções seguintes descrevem os melhores resultados obtidos pelo CSARF, utilizando a heurística *Picek*, em comparação com o ARF (*baseline*), ARF_{RE} e KUE para cada *data stream* multi-classe desbalanceado.

4.4.1 Pokerhands - *poker*

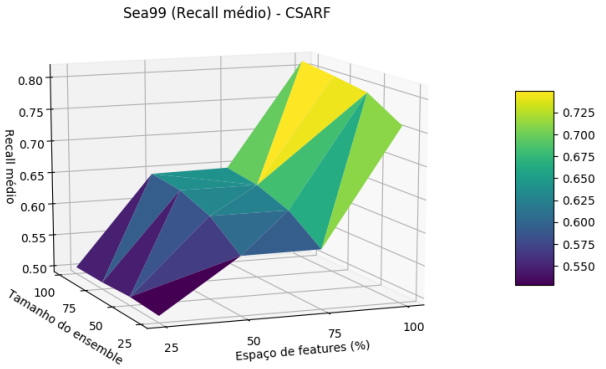
Este *stream* de dados contém um total de 10 classes das quais são caracterizadas por 11 atributos. O *grid search* conduzido resultou em valores de *baseline*, utilizando o ARF, de 0.541 para $F1_{score}$ médio e 0.465 para *recall* médio. A tabela 24 resume as performances alcançadas por todos os algoritmos parametrizados com as configurações listadas na tabela 25.

O método proposto ultrapassou o *baseline* em todas as métricas avaliadas. Com uma performance 7.6% acima do ARF, o CSARF se posiciona em segundo lugar atrás do ARF_{RE} na métrica $F1_{score}$ médio (Tabela 24). Observa-se que o inverso ocorre quando analisamos a métrica *recall* médio, em que dessa vez o método proposto ranqueia em primeiro e o ARF_{RE} em segundo lugar. O CSARF se destaca com uma performance com ganho de 34.3% sob o *baseline* e 20.4% contra o segundo lugar em *recall* médio. Para ambientes



(a) AUC-PR

(b) F1$_{score}$ médio



(c) *Recall* médio

Figura 17 – Resultados obtidos pelo *grid search* realizado na base *Sea Generator* (*sea99*).

Tabela 24 – Poker - Resultados

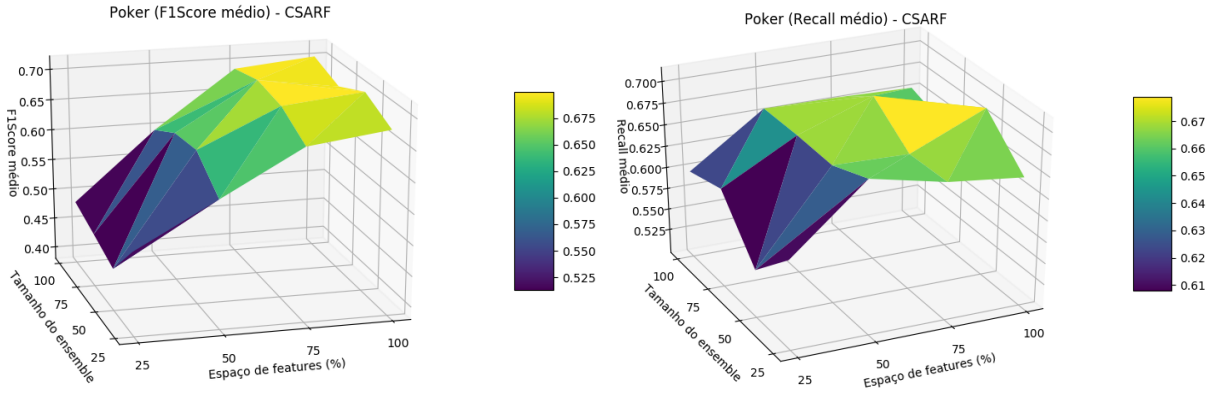
	F1$_{score}$ médio	*Recall* médio
ARF (*baseline*)	0.541600	0.465963
CSARF	0.582579	**0.626085**
ARF$_{RE}$	**0.624851**	0.519960
KUE	0.473570	0.437673

multi-classe desbalanceados, nota-se que o KUE não apresenta boas performances em nenhuma das métricas e se posiciona em último lugar com os piores resultados obtidos no experimento.

Para ambas as métricas avaliadas a mesma configuração do CSARF foi utilizada. Isto é, os melhores resultados obtidos com o método proposto em ambas as métricas pertencem ao mesmo conjunto de parâmetros (Tabela 25). Observa-se que somente o ARF e ARF$_{RE}$ optaram pelo maior tamanho de *ensemble* (-e) enquanto que os resultados obtidos pelo CSARF e KUE utilizam apenas 50 indutores em ambas as métricas. A figura 18 ilustra a exploração dos parâmetros do CSARF e a busca da melhor configuração para as métricas F1$_{score}$ médio (Figura 18a) e *recall* médio (Figura 18b).

Tabela 25 – Poker - Configurações utilizadas

	$F1_{score}$ médio	*Recall* médio
ARF	-e100-f100	-e100-f100
CSARF	-l-e50-f75-i10000	-l-e50-f75-i10000
ARF_{RE}	-e100-f75	-e100-f75
KUE	-e50-n30	-e50-n10



(a) $F1_{score}$ médio (b) *Recall* médio

Figura 18 – Resultados obtidos pelo *grid search* realizado na base *Poker hands* (*poker*).

4.4.2 Forest CoverType - *covrt*

Este *data stream* descreve 7 classes caracterizadas por 54 atributos e uma distribuição desbalanceada, contendo classes com menos de 1% do total do número de instâncias. Os experimentos conduzidos resultaram em performances para o *baseline* (ARF) de 0.904 para $F1_{score}$ médio e 0.899 para *recall* médio. A tabela 26 resume os resultados alcançados pelos métodos e a tabela 27 descreve quais parâmetros foram utilizados em cada um dos algoritmos.

Dentre os resultados obtidos, o CSARF detém os melhores resultados para as métricas $F1_{score}$ médio e *recall* médio. A diferença entre o método proposto e o *baseline* para a métrica $F1_{score}$ médio é pequena, cerca de 0.09% de ganho. Porém, se analisarmos o algoritmo ARF_{RE}, que é um algoritmo orientado a desbalanceamento, nota-se que a sua performance ficou abaixo do próprio ARF nesse *stream* de dados. O mesmo comportamento se repete para a métrica *recall* médio, só que dessa vez com o ARF_{RE} ocupando a segunda colocação. Mesmo com performances semelhantes, observa-se que os métodos oriundos do ARF não convergiram para os mesmos parâmetros listados na tabela 27. A figura 19 ilustra a busca pelo melhor resultado para as métricas $F1_{score}$ médio (Figura 19a) e *recall* médio (Figura 19b) do CSARF para o *stream* de dados *covrt*.

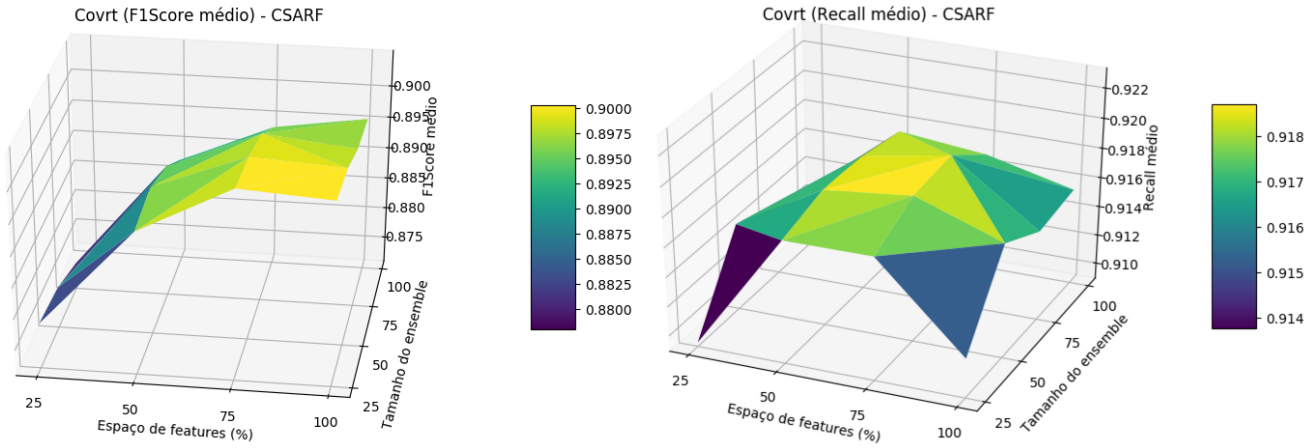
A próxima seção contém uma análise geral sobre todos os resultados obtidos pelo CSARF e os demais algoritmos para os *streams* de dados. Isto é, a seção seguinte descreve

Tabela 26 – Covrt - Resultados

	$F1_{score}$ médio	*Recall* médio
ARF (*baseline*)	0.904106	0.899831
CSARF	**0.904955**	**0.923029**
ARF_{RE}	0.897946	0.903825
KUE	0.766808	0.746529

Tabela 27 – Covrt - Configurações utilizadas

	F1Score médio	Recall médio
ARF	-e25-f75	-e25-f75
CSARF	-t-e25-f100-i10000	-l-e75-f75-i20000
ARFRE	-e50-f75	-e75-f75
KUE	-e25-n40	-e50-n40



(a) $F1_{score}$ médio (b) *Recall* médio

Figura 19 – Resultados obtidos pelo *grid search* realizado na base *Forest CoverType* (*covrt*).

em detalhes as análises estatísticas que foram conduzidas para verificar as hipóteses formuladas no protocolo experimental.

4.5 Verificação de hipótese

Nas subseções anteriores o método proposto foi comparado com os métodos concorrentes (ARF, ARF_{RE} e KUE) em diferentes níveis de desbalanceamento e problemas de classificação. Perante ao *grid search* realizado foi possível trazer a melhor solução, dentro do espaço de parâmetros explorado, proposta por cada método em cada *stream* de dados. Dessa forma, foi possível comparar em cada conjunto de dados o melhor do CSARF contra o melhor de cada algoritmo. As colocações conquistadas por cada método em cada fluxo de dados e métrica avaliada estão em detalhes na tabela 28. Ademais, o

protocolo experimental possibilitou verificar se os métodos oriundos do ARF apresentavam configurações convergentes para um determinado problema de classificação desbalanceado.

Para de fato verificar as hipóteses formuladas no presente protocolo experimental (seção 4.1) serão aceitas ou rejeitadas, conduziu-se uma análise estatística. Seguindo as instruções para comparar estatisticamente múltiplos classificadores sugeridas em (DEMŠAR, 2006), realizou-se um teste de *Friedman* e *Nemenyi* com base nos resultados obtidos nas métricas AUC-PR, $F1_{score}$ médio e *recall* médio (Tabela 28). Os gráficos de distância crítica (Figuras 20, 21 e 22) contém o ranqueamento dos métodos em que a menor colocação representa o melhor algoritmo para a métrica avaliada. Com um nível de 95% de confiança, o teste não paramétrico de *Friedman* sugere que não há diferenças entre os métodos avaliados na métrica AUC-PR (Figura 20). É possível visualizar que a distância entre o primeiro e último lugar não ultrapassa a distância crítica de 1.354, isto é, não há diferença estatística significativa de performance na métrica AUC-PR para os métodos avaliados. Entretanto, ressalta-se que o método proposto (CSARF) conquistou o primeiro lugar apresentando a melhor performance em AUC-PR, na frente do *baseline* e do ARF_{RE} que é um algoritmo orientado a desbalanceamento, nos *streams* de dados dentre os algoritmos avaliados.

Ao analisarmos estatisticamente os resultados para a métrica $F1_{score}$ médio, o teste de *Friedman* aponta que há diferença entre os métodos e que ela é significativa apenas entre o método proposto e o ARF (Figura 21). Novamente o CSARF conquista o primeiro lugar dentre os algoritmos avaliados e fica a frente do método ARF_{RE}, que nessa métrica obteve boas colocações quando comparado contra os métodos KUE e ARF. Por mais que a diferença significativa seja apenas entre o CSARF e ARF para a métrica $F1_{score}$ médio, nota-se que o método proposto detém boas performances quando comparado ao KUE nos fluxos de dados avaliados.

Por fim, o teste de *Friedman* conduzido para os resultados na métrica *recall* médio sugere que há diferença entre os métodos avaliados. Com 95% de confiança o teste *post-hoc* de *Nemenyi* aponta que há diferença estatística significativa entre o CSARF e os métodos ARF e KUE (Figura 22). Contra esses dois algoritmos, o método proposto apresenta uma distância no ranqueamento que ultrapassa a distância significativa de 1.254. O teste também sugere que não há diferença entre o CSARF e ARF_{RE} e entre o ARF e KUE para a métrica *recall* médio. De qualquer forma, o CSARF ainda é uma alternativa melhor que o ARF_{RE} pois um dos problemas desse algoritmo, apontado pelos próprios autores em (BOIKO et al., 2019), é de que o método não interpreta o fluxo de dados da maneira como ele é produzido. O ARF_{RE} utiliza reamostragem para diminuir o impacto do desbalanceamento e, isso pode ou não alterar os mecanimos de detecção de mudança de conceito oriundos do ARF fazendo com que o ARF_{RE} altere o seu comportamento adaptativo. Esse fenômeno não ocorre no CSARF pois esse altera apenas a saída dos

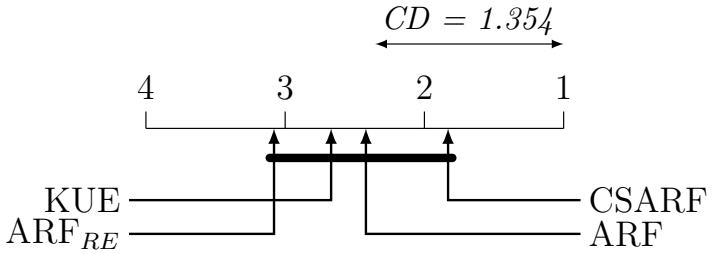


Figura 20 – Gráfico de distância crítica para os valores obtidos na métrica AUC-PR pelos métodos CSARF, ARF, ARF$_{RE}$ e KUE.

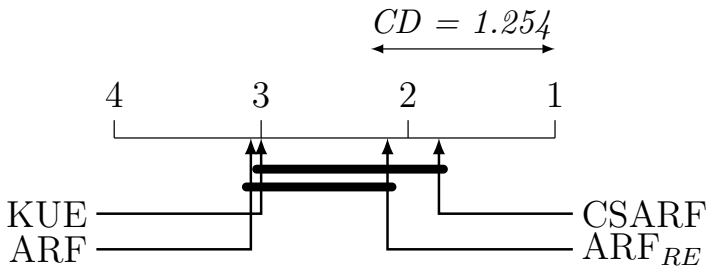


Figura 21 – Gráfico de distância crítica para os valores obtidos na métrica F1$_{score}$ médio pelos métodos CSARF, ARF, ARF$_{RE}$ e KUE.

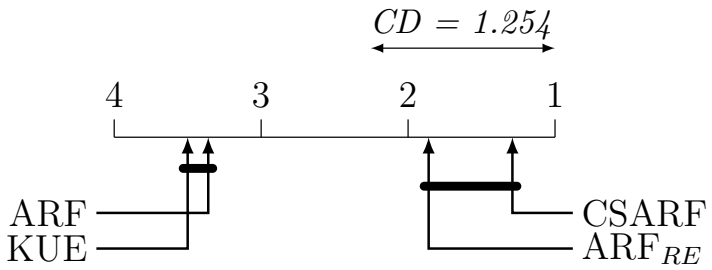


Figura 22 – Gráfico de distância crítica para os valores obtidos na métrica *recall* médio pelos métodos CSARF, ARF, ARF$_{RE}$ e KUE.

indutores base ou da combinação deles. Assim sendo, perante aos resultados apresentados rejeita-se a hipótese nula de que não há diferença estatisticamente significativa entre os métodos avaliados nas métricas AUC-PR, F1$_{score}$ médio e *recall* médio. Mesmo que o ganho na performance apresentada pelo CSARF em relação ao *baseline* seja pequeno, como demonstrado em alguns experimentos, em fluxos de dados desbalanceados o aumento no poder de detecção da classe minoritária é importante dada a sua raridade e o custo envolvido ao se classificar erroneamente uma instância dessa classe.

4.6 Análise das estratégias de uso do *threshold*

Nas seções anteriores apresentou-se o potencial do método proposto em *streams* de dados desbalanceados comparando-o com algoritmos competitivos ao estado da arte. Ressaltou-se o impacto na decisão de um classificador quando esse é sensível a custos de classificação por meio das análises estatísticas realizadas. Porém, dentro das discussões

acerca dos experimentos conduzidos não houve uma em que analisasse as três diferentes estratégias do CSARF aplicar custos de classificação: *Training*, *Local* e *Global*. Por essa razão, a presente seção traz em detalhes uma análise sobre essas estratégias e qual o potencial de cada uma para as métricas avaliadas no protocolo experimental descrito na seção 4.1.

Os experimentos conduzidos em cada *stream* de dados apresentavam não só os melhores resultados obtidos por cada algoritmo, como também os parâmetros que foram utilizados para construir os melhores modelos. Perante a essas configurações, é possível observar qual estratégia do CSARF foi utilizada para cada métrica nos *data streams* selecionados. A figura 23 ilustra o percentual de uso de cada estratégia, que configura o melhor resultado obtido, para as métricas AUC-PR, F1_{score} médio e *recall* médio. Observa-se que para cada métrica avaliada, os melhores resultados obtidos são quase que majoritariamente protagonizados por uma das estratégias.

A análise estatística realizada para os resultados na métrica AUC-PR, sugeriu que não houve diferença estatisticamente significativa entre os métodos avaliados. Nesta métrica, nota-se que a estratégia *Training* vence em cerca de 40% dos melhores resultados obtidos. Como demonstrado na subseção 3.1.2, essa estratégia aplica indiretamente sensibilidade a custos ao classificador alterando-se ligeiramente os votos probabilísticos do modelo. Isso fica evidente quando nota-se a dominância do CSARF-training na métrica AUC-PR em que, como demonstrado na seção anterior, o ARF conquistou o segundo lugar. Assim sendo, evidencia-se que a estratégia *Training* teve uma participação importante na conquista do primeiro lugar pelo CSARF para a métrica AUC-PR.

O método proposto apresentou resultados que configuram em uma diferença estatística significativa entre ele e o ARF para as métricas F1_{score} médio e *recall* médio. Observa-se que dessa vez o CSARF-local e CSARF-global protagonizaram os melhores resultados apresentados pelo CSARF (Figura 23). Ambas as estratégias alteram diretamente a saída do classificador, sendo que a *Local* induz sensibilidade a custos antes de fazer a combinação dos votos do *ensemble* e a *Global* após a combinação.

Quando um modelo probabilístico lida com um problema desbalanceado, ele tende a apresentar maiores valores de probabilidade para a classe majoritária. Isso ocorre porque o modelo retém muito mais informação sobre a classe que aparece com mais frequência. Nesses conjuntos de dados desbalanceados, o limite de decisão *default* do classificador (escolhe-se a classe com probabilidade acima de 50%) pode não ser representativo para o problema. São nesses cenários que as estratégias *Local* e *Global* demonstram o seu potencial. Sabe-se que tanto a métrica F1_{score} médio quanto o *recall* médio utilizam o limite de decisão *default* para serem calculadas. E como observado nos experimentos conduzidos, os métodos que não lidam com desbalanceamento (ARF e KUE) ocupam justamente as piores posições no ranqueamento. Dessa forma, evidencia-se o potencial das estratégias

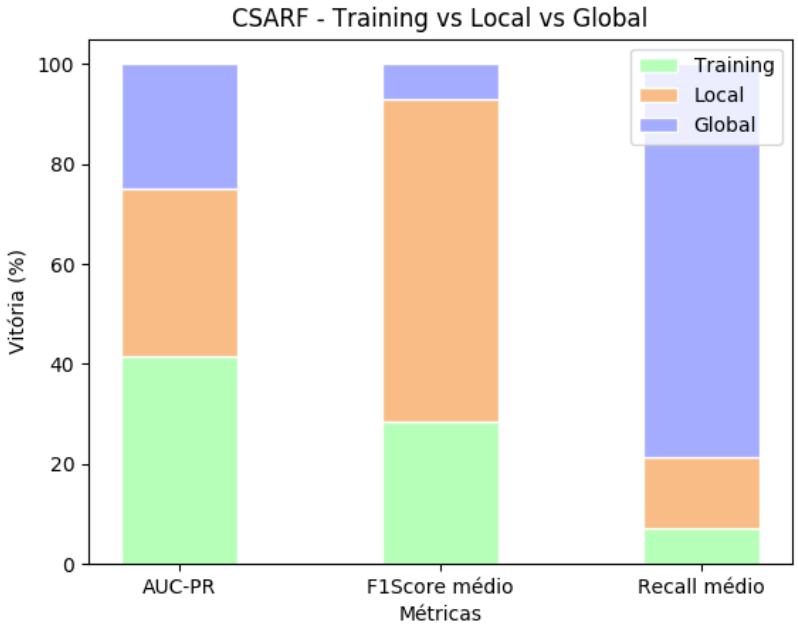


Figura 23 – Percentual de vitória das estratégias de uso do *threshold* *Training*, *Local* e *Global* em cada métrica para todas as *streams* de dados.

Training, *Local* e *Global* para os diferentes cenários apresentados.

4.7 Considerações finais

Neste capítulo analisou-se o ganho na performance de classificação utilizando a heurística *OzaCosting*, a qual foi uma tentativa de criação de uma nova heurística, contra a já existente *Picek* (Equação 3.4) (PICEK et al., 2018). Por meio dos testes conduzidos, observou-se que a heurística *Picek* apresentou os melhores resultados para os *streams* de dados avaliados. Assim sendo, escolheu-se a heurística *Picek* para avaliar o desempenho do método proposto contra os algoritmos ARF, ARF_{RE} e KUE. Comprovou-se que o CSARF detém os melhores resultados dentre os métodos avaliados com diferença estatisticamente significativa, rejeitando-se as hipóteses nulas formuladas. Por fim, analisou-se o potencial de cada estratégia de uso do *threshold* para induzir sensibilidade a custos ao CSARF. Esse estudo proporcionou salientar que a utilização de custos de classificação possibilita melhorar a assertividade na classificação de instâncias da classe minoritária.

Tabela 28 – Ranqueamento dos métodos análisados

	AUC-PR				$F1_{score}$ médio				*Recall* médio			
	CSARF	ARF	ARF_{RE}	KUE	CSARF	ARF	ARF_{RE}	KUE	CSARF	ARF	ARF_{RE}	KUE
santd	1	2	4	3	1	3	4	2	1	3	2	4
elect	3	2	4	1	2	3	1	4	1	3	2	4
privt	1	2	3	4	2	3	1	4	1	3	2	4
agrwl90	3	4	2	1	2	3	4	1	2	4	1	3
agrwl95	2	3	4	1	2	3	4	1	2	4	1	3
agrwl99	2	3	4	1	2	4	3	1	1	4	3	2
gmsc	2	3	4	1	1	4	2	3	1	4	2	3
pozzl	2	1	3	4	1	2	3	4	1	3	2	4
airli	1	2	3	4	3	2	1	4	3	2	1	4
sea90	1	3	2	4	2	3	1	4	1	3	2	4
sea95	3	2	1	4	2	4	1	3	1	4	2	3
sea99	1	2	3	4	2	4	1	3	1	4	2	3
poker	-	-	-	-	2	3	1	4	1	3	2	4
covrt	-	-	-	-	1	2	3	4	1	3	2	4
Rank médio	**1.833**	2.417	3.083	2.667	**1.786**	3.071	2.143	3	**1.286**	3.357	1.857	3.5

5 Conclusão

As abordagens baseadas em custos são utilizadas com a finalidade de incorporar a penalidade de se classificar uma classe em outra, fazendo com que o algoritmo indutor considere os custos de classificação ao se realizar uma decisão. Um dos problemas dessas abordagens reside na definição de um custo que de fato represente o problema e proporcione um ganho no desempenho na detecção da classe de maior relevância.

As penalidades de classificação podem ser incorporadas de maneiras gerais como o *threshold*, *oversampling* e *undersampling* (baseados em custo) ou específicas: redes neurais que incorporam as penalidades na função de minimização de erro e árvores de decisão que empregam os custos em suas estruturas. Todas essas estratégias podem ser utilizadas em ambientes desbalanceados se considerarmos que a classe minoritária possui a maior penalidade de classificação.

Uma das maiores dificuldades encontradas durante o desenvolvimento teórico e experimental do método, foi devido a escassez de estratégias desenvolvidas para definir de modo dinâmico as penalidades de classificação. Pelo fato do problema do presente trabalho tratar de *stream* de dados desbalanceados, orienta-se seguir os fundamentos desse paradigma para empregar uma estratégia dinâmica assim como é um fluxo contínuo de dados. Contudo, a fundamentação teórica deste trabalho contribuiu o significativamente para o desenvolvimento do CSARF.

Neste trabalho foi apresentada uma adaptação do ARF para *data streams* desbalanceados, o algoritmo CSARF. O método proposto incorpora mudanças nas individualidades do ARF para ponderar corretamente as árvores com melhores performances em ambientes desbalanceados. Além do mais, uma matriz de custos também é utilizada de modo a fazer com que o CSARF utilize as penalidades de classificação de três maneiras: *training*, *local* e *global*. Juntamentamente com o método proposto, também apresentou-se o *OzaCosting* que é uma estratégia baseada no *Online Boosting* para calcular os custos de classificação considerando a dificuldade do *ensemble* em classificar determinada classse. Por meio empírico, evidenciou-se que a heurística *OzaCosting* resultou em performances abaixo da já existente *Picek* e por essa razão, o estudo comparativo conduzido considerou apenas a heurística com melhor performance obtida.

Os experimentos realizados com o CSARF apresentaram resultados que confirmam a hipótese inicial, de que é possível melhorar a performance de classificação de S_{min} em ambientes de *streams* desbalanceados empregando custos de classificação. Houve casos em que o método proposto performou pior que eu o seu método de origem, mas as análises estatísticas conduzidas demonstraram que o CSARF tem potencial e detém os melhores

resultados. Para trabalhos futuros, pretende-se construir um *framework* para aplicação de penalidades de classificação que seja utilizável por qualquer classificador. Há também a necessidade de explorar a heurística *OzaCosting* em ambientes balanceados devido a sua estratégia de alterar os custos conforme a dificuldade do classificador. Necessita-se de estudo e desenvolvimento de novas heurísticas de custos capazes de serem utilizadas no ambiente *stream*. Em relação ao método proposto, ainda há possibilidade de investigar diferentes indutores base e métricas de ganho de informação que possam desempenhar melhor em ambientes desbalanceados.

Durante a concepção do método proposto e perante aos resultados preliminares alcançados pela primeira versão do método, publicou-se um artigo com o título *"Cost-sensitive learning for imbalanced data streams"* (DOI: 10.1145/3341105.3373949) no simpósio *The 35th ACM/SIGAPP Symposium On Applied Computing - SAC 2020*.

Referências

AGRAWAL, Rakesh; IMIELINSKI, Tomasz; SWAMI, Arun. Database mining: A performance perspective. *IEEE transactions on knowledge and data engineering*, IEEE, v. 5, n. 6, p. 914–925, 1993. Citado na página 47.

BARUA, Md Sukarna; ISLAM, Monirul; YAO, Xin; MURASE, Kazuyuki. Mwmote-majority weighted minority oversampling technique for imbalanced data set learning. *IEEE Transactions on Knowledge and Data Engineering*, IEEE, p. 1, 2012. Citado na página 34.

BATISTA, Gustavo EAPA; CARVALHO, Andre CPLF; MONARD, Maria Carolina. Applying one-sided selection to unbalanced datasets. In: SPRINGER. *Mexican International Conference on Artificial Intelligence*. [S.l.], 2000. p. 315–325. Citado 2 vezes nas páginas 31 e 36.

BHATTACHARYYA, Siddhartha; JHA, Sanjeev; THARAKUNNEL, Kurian; WESTLAND, J Christopher. Data mining for credit card fraud: A comparative study. *Decision Support Systems*, Elsevier, v. 50, n. 3, p. 602–613, 2011. Citado na página 30.

BIFET, Albert; HOLMES, Geoff; KIRKBY, Richard; PFAHRINGER, Bernhard. Moa: Massive online analysis. *Journal of Machine Learning Research*, v. 11, n. May, p. 1601–1604, 2010. Citado 6 vezes nas páginas 21, 22, 26, 45, 47 e 61.

BIFET, Albert; HOLMES, Geoff; PFAHRINGER, Bernhard. Leveraging bagging for evolving data streams. In: SPRINGER. *Joint European conference on machine learning and knowledge discovery in databases*. [S.l.], 2010. p. 135–150. Citado na página 28.

BOIKO, Luis E; GOMES, Heitor; BIFET, Albert; OLIVEIRA, Luiz S. Adaptive random forests with resampling for imbalanced data streams. *International Joint Conference on Neural Networks (IJCNN)*, 2019. Citado 3 vezes nas páginas 42, 61 e 87.

BREIMAN, Leo. Bagging predictors. *Machine learning*, Springer, v. 24, n. 2, p. 123–140, 1996. Citado 3 vezes nas páginas 25, 28 e 40.

BREIMAN, Leo. Random forests. *Machine learning*, Springer, v. 45, n. 1, p. 5–32, 2001. Citado 3 vezes nas páginas 28, 40 e 43.

CANO, Alberto; KRAWCZYK, Bartosz. Kappa updated ensemble for drifting data stream mining. *Machine Learning*, Springer, v. 109, n. 1, p. 175–218, 2020. Citado 2 vezes nas páginas 43 e 61.

CHAN, Philip K; STOLFO, Salvatore J. Toward scalable learning with non-uniform class and cost distributions: A case study in credit card fraud detection. In: *KDD*. [S.l.: s.n.], 1998. v. 1998, p. 164–168. Citado na página 41.

CHAWLA, Nitesh V; BOWYER, Kevin W; HALL, Lawrence O; KEGELMEYER, W Philip. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, v. 16, p. 321–357, 2002. Citado na página 32.

CHEN, Chao; LIAW, Andy; BREIMAN, Leo et al. Using random forest to learn imbalanced data. *University of California, Berkeley*, v. 110, p. 1–12, 2004. Citado na página 40.

CHEN, Sheng; HE, Haibo. Sera: selectively recursive approach towards nonstationary imbalanced stream data mining. In: IEEE. *2009 International Joint Conference on Neural Networks*. [S.l.], 2009. p. 522–529. Citado na página 42.

CHICCO, Davide. Ten quick tips for machine learning in computational biology. *BioData mining*, BioMed Central, v. 10, n. 1, p. 35, 2017. Citado na página 50.

CHICCO, Davide; JURMAN, Giuseppe. The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC genomics*, Springer, v. 21, n. 1, p. 6, 2020. Citado na página 50.

DAL POZZOLO, A.; BORACCHI, G.; CAELEN, O.; ALIPPI, C.; BONTEMPI, G. Credit card fraud detection and concept-drift adaptation with delayed supervised information. In: *2015 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2015. p. 1–8. ISSN 2161-4407. Citado na página 25.

DAVIS, Jesse; GOADRICH, Mark. The relationship between precision-recall and roc curves. In: ACM. *Proceedings of the 23rd international conference on Machine learning*. [S.l.], 2006. p. 233–240. Citado 3 vezes nas páginas 45, 62 e 63.

DEMŠAR, Janez. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, v. 7, n. Jan, p. 1–30, 2006. Citado 2 vezes nas páginas 64 e 87.

DITZLER, Gregory; POLIKAR, Robi. An ensemble based incremental learning framework for concept drift and class imbalance. In: IEEE. *The 2010 International Joint Conference on Neural Networks (IJCNN)*. [S.l.], 2010. p. 1–8. Citado na página 41.

DITZLER, Gregory; POLIKAR, Robi. Incremental learning of concept drift from streaming imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, IEEE, v. 25, n. 10, p. 2283–2301, 2013. Citado na página 41.

DOMINGOS, Pedro. Metacost: A general method for making classifiers cost-sensitive. In: *KDD*. [S.l.: s.n.], 1999. v. 99, p. 155–164. Citado na página 39.

DOMINGOS, Pedro; HULTEN, Geoff. Mining high-speed data streams. In: *Kdd*. [S.l.: s.n.], 2000. v. 2, p. 4. Citado na página 29.

DUA, Dheeru; GRAFF, Casey. *UCI Machine Learning Repository*. 2017. Disponível em: <http://archive.ics.uci.edu/ml>. Citado na página 47.

ELKAN, Charles. The foundations of cost-sensitive learning. In: LAWRENCE ERLBAUM ASSOCIATES LTD. *International joint conference on artificial intelligence*. [S.l.], 2001. v. 17, n. 1, p. 973–978. Citado 3 vezes nas páginas 36, 37 e 39.

ELWELL, Ryan; POLIKAR, Robi. Incremental learning of variable rate concept drift. In: SPRINGER. *International Workshop on Multiple Classifier Systems*. [S.l.], 2009. p. 142–151. Citado na página 41.

FAWCETT, Tom. An introduction to roc analysis. *Pattern recognition letters*, Elsevier, v. 27, n. 8, p. 861–874, 2006. Citado na página 44.

FREUND, Yoav; SCHAPIRE, Robert E. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, Elsevier, v. 55, n. 1, p. 119–139, 1997. Citado na página 38.

GAMA, João; SEBASTIÃO, Raquel; RODRIGUES, Pedro Pereira. Issues in evaluation of stream learning algorithms. In: ACM. *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. [S.l.], 2009. p. 329–338. Citado na página 46.

GAMA, João; SEBASTIÃO, Raquel; RODRIGUES, Pedro Pereira. On evaluating stream learning algorithms. *Machine learning*, Springer, v. 90, n. 3, p. 317–346, 2013. Citado na página 46.

GANTZ, John; REINSEL, David. The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. *IDC iView: IDC Analyze the future*, v. 2007, n. 2012, p. 1–16, 2012. Citado na página 21.

GAO, Jing; FAN, Wei; HAN, Jiawei; YU, Philip S. A general framework for mining concept-drifting data streams with skewed distributions. In: SIAM. *Proceedings of the 2007 SIAM International Conference on Data Mining*. [S.l.], 2007. p. 3–14. Citado na página 41.

GAO, Xingyu; CHEN, Zhenyu; TANG, Sheng; ZHANG, Yongdong; LI, Jintao. Adaptive weighted imbalance learning with application to abnormal activity recognition. *Neurocomputing*, Elsevier, v. 173, p. 1927–1935, 2016. Citado na página 30.

GHAZIKHANI, Adel; MONSEFI, Reza; YAZDI, Hadi Sadoghi. Online cost-sensitive neural network classifiers for non-stationary and imbalanced data streams. *Neural computing and applications*, Springer, v. 23, n. 5, p. 1283–1295, 2013. Citado na página 39.

GHAZIKHANI, Adel; MONSEFI, Reza; YAZDI, Hadi Sadoghi. Online neural network model for non-stationary and imbalanced data stream classification. *International Journal of Machine Learning and Cybernetics*, Springer, v. 5, n. 1, p. 51–62, 2014. Citado 2 vezes nas páginas 32 e 40.

GODASE, Abhijeet; ATTAR, Vahida. Classifier ensemble for imbalanced data stream classification. In: ACM. *Proceedings of the CUBE International Information Technology Conference*. [S.l.], 2012. p. 284–289. Citado na página 42.

GOMES, Heitor Murilo; BARDDAL, Jean Paul; ENEMBRECK, Fabrício; BIFET, Albert. A survey on ensemble learning for data stream classification. *ACM Computing Surveys (CSUR)*, ACM, v. 50, n. 2, p. 23, 2017. Citado 2 vezes nas páginas 27 e 40.

GOMES, Heitor M; BIFET, Albert; READ, Jesse; BARDDAL, Jean Paul; ENEMBRECK, Fabrício; PFHARINGER, Bernhard; HOLMES, Geoff; ABDESSALEM, Talel. Adaptive random forests for evolving data stream classification. *Machine Learning*, Springer, v. 106, n. 9-10, p. 1469–1495, 2017. Citado 5 vezes nas páginas 22, 28, 29, 31 e 61.

GONG, Joonho; KIM, Hyunjoong. Rhsboost: Improving classification performance in imbalance data. *Computational Statistics & Data Analysis*, Elsevier, v. 111, p. 1–13, 2017. Citado na página 22.

HAN, Hui; WANG, Wen-Yuan; MAO, Bing-Huan. Borderline-smote: a new over-sampling method in imbalanced data sets learning. In: SPRINGER. *International conference on intelligent computing*. [S.l.], 2005. p. 878–887. Citado na página 32.

HARRIES, Michael; WALES, New South. Splice-2 comparative evaluation: Electricity pricing. Citeseer, 1999. Citado na página 47.

HE, Haibo; BAI, Yang; GARCIA, Edwardo A; LI, Shutao. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In: IEEE. *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. [S.l.], 2008. p. 1322–1328. Citado na página 33.

HE, H.; GARCIA, E. A. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, v. 21, n. 9, p. 1263–1284, Sep. 2009. ISSN 1041-4347. Citado 3 vezes nas páginas 21, 43 e 44.

HULSE, Jason Van; KHOSHGOFTAAR, Taghi M; NAPOLITANO, Amri. Experimental perspectives on learning from imbalanced data. In: ACM. *Proceedings of the 24th international conference on Machine learning*. [S.l.], 2007. p. 935–942. Citado na página 31.

JORDAN, Michael I; MITCHELL, Tom M. Machine learning: Trends, perspectives, and prospects. *Science*, American Association for the Advancement of Science, v. 349, n. 6245, p. 255–260, 2015. Citado na página 21.

KOTSIANTIS, Sotiris B; ZAHARAKIS, I; PINTELAS, P. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, v. 160, p. 3–24, 2007. Citado na página 21.

KRAWCZYK, Bartosz. Learning from imbalanced data: open challenges and future directions. *Progress in Artificial Intelligence*, Springer, v. 5, n. 4, p. 221–232, 2016. Citado na página 22.

KRAWCZYK, Bartosz; GALAR, Mikel; JELEŃ, Łukasz; HERRERA, Francisco. Evolutionary undersampling boosting for imbalanced classification of breast cancer malignancy. *Applied Soft Computing*, Elsevier, v. 38, p. 714–726, 2016. Citado 3 vezes nas páginas 23, 30 e 31.

KUBAT, Miroslav; MATWIN, Stan et al. Addressing the curse of imbalanced training sets: one-sided selection. In: NASHVILLE, USA. *Icml*. [S.l.], 1997. v. 97, p. 179–186. Citado na página 35.

KUKAR, Matjaz; KONONENKO, Igor et al. Cost-sensitive learning with neural networks. In: *ECAI*. [S.l.: s.n.], 1998. p. 445–449. Citado na página 39.

LAURIKKALA, Jorma. Improving identification of difficult small classes by balancing class distribution. In: SPRINGER. *Conference on Artificial Intelligence in Medicine in Europe*. [S.l.], 2001. p. 63–66. Citado na página 36.

LAWI, A.; AZIZ, F.; SYARIF, S. Ensemble gradientboost for increasing classification accuracy of credit scoring. In: *2017 4th International Conference on Computer Applications and Information Processing Technology (CAIPT)*. [S.l.: s.n.], 2017. p. 1–4. Citado na página 25.

LI, Hu; WANG, Ye; WANG, Hua; ZHOU, Bin. Multi-window based ensemble learning for classification of imbalanced streaming data. *World Wide Web*, Springer, v. 20, n. 6, p. 1507–1525, 2017. Citado 2 vezes nas páginas 42 e 55.

LICHTENWALTER, Ryan N; CHAWLA, Nitesh V. Adaptive methods for classification in arbitrarily imbalanced and drifting data streams. In: SPRINGER. *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. [S.l.], 2009. p. 53–75. Citado 2 vezes nas páginas 32 e 42.

LING, CX; SHENG, VS. Cost-sensitive learning and the class imbalance problem. 2011. *Encyclopedia of Machine Learning: Springer*, v. 24, 2011. Citado na página 38.

LIU, Xu-Ying; WU, Jianxin; ZHOU, Zhi-Hua. Exploratory undersampling for class-imbalance learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, IEEE, v. 39, n. 2, p. 539–550, 2009. Citado na página 40.

LIU, Xu-Ying; ZHOU, Zhi-Hua. The influence of class imbalance on cost-sensitive learning: An empirical study. In: IEEE. *Sixth International Conference on Data Mining (ICDM'06)*. [S.l.], 2006. p. 970–974. Citado na página 37.

MALOOF, Marcus A. Learning when data sets are imbalanced and when costs are unequal and unknown. In: *ICML-2003 workshop on learning from imbalanced data sets II*. [S.l.: s.n.], 2003. v. 2, p. 2–1. Citado na página 37.

MATTHEWS, Brian W. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure*, Elsevier, v. 405, n. 2, p. 442–451, 1975. Citado na página 50.

NIKOLAOU, Nikolaos; EDAKUNNI, Narayanan; KULL, Meelis; FLACH, Peter; BROWN, Gavin. Cost-sensitive boosting algorithms: Do we really need them? *Machine Learning*, Springer, v. 104, n. 2-3, p. 359–384, 2016. Citado na página 38.

OZA, Nikunj C. Online bagging and boosting. In: IEEE. *2005 IEEE international conference on systems, man and cybernetics*. [S.l.], 2005. v. 3, p. 2340–2345. Citado 4 vezes nas páginas 25, 28, 53 e 54.

PICEK, Stjepan; HEUSER, Annelie; JOVIC, Alan; BHASIN, Shivam; REGAZZONI, Francesco. The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018. Citado 2 vezes nas páginas 51 e 90.

POLIKAR, Robi. Ensemble based systems in decision making. *IEEE Circuits and systems magazine*, IEEE, v. 6, n. 3, p. 21–45, 2006. Citado na página 40.

POZZOLO, Andrea Dal; JOHNSON, Reid; CAELEN, Olivier; WATERSCHOOT, Serge; CHAWLA, Nitesh V; BONTEMPI, Gianluca. Using hddt to avoid instances propagation in unbalanced and evolving data streams. In: IEEE. *2014 International Joint Conference on Neural Networks (IJCNN)*. [S.l.], 2014. p. 588–594. Citado na página 32.

QUINLAN, J. Ross. *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993. ISBN 1-55860-238-0. Citado na página 57.

SCHAPIRE, Robert E; SINGER, Yoram. Improved boosting algorithms using confidence-rated predictions. *Machine learning*, Springer, v. 37, n. 3, p. 297–336, 1999. Citado 3 vezes nas páginas 38, 40 e 53.

SPYROMITROS-XIOUFIS, Eleftherios; SPILIOPOULOU, Myra; TSOUMAKAS, Grigorios; VLAHAVAS, Ioannis. Dealing with concept drift and class imbalance in multi-label stream classification. In: *Twenty-Second International Joint Conference on Artificial Intelligence*. [S.l.: s.n.], 2011. Citado na página 31.

STREET, W Nick; KIM, YongSeog. A streaming ensemble algorithm (sea) for large-scale classification. In: ACM. *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. [S.l.], 2001. p. 377–382. Citado na página 47.

TING, Kai Ming. A comparative study of cost-sensitive boosting algorithms. In: CITESEER. *In Proceedings of the 17th International Conference on Machine Learning*. [S.l.], 2000. Citado na página 38.

TING, Kai Ming. An instance-weighting method to induce cost-sensitive trees. *IEEE Transactions on Knowledge & Data Engineering*, IEEE, n. 3, p. 659–665, 2002. Citado na página 39.

TOMEK, Ivan. Two modifications of cnn. *IEEE Trans. Systems, Man and Cybernetics*, v. 6, p. 769–772, 1976. Citado na página 35.

TOPOUZELIS, Konstantinos. Oil spill detection by sar images: dark formation detection, feature extraction and classification algorithms. *Sensors*, Molecular Diversity Preservation International, v. 8, n. 10, p. 6642–6659, 2008. Citado 2 vezes nas páginas 23 e 30.

VIOLA, Paul; JONES, Michael. Fast and robust classification using asymmetric adaboost and a detector cascade. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2002. p. 1311–1318. Citado na página 38.

WANG, Heng; ABRAHAM, Zubin. Concept drift detection for streaming data. In: IEEE. *2015 International Joint Conference on Neural Networks (IJCNN)*. [S.l.], 2015. p. 1–9. Citado na página 27.

WANG, Shuo; MINKU, Leandro L; YAO, Xin. A learning framework for online class imbalance learning. In: IEEE. *2013 IEEE Symposium on Computational Intelligence and Ensemble Learning (CIEL)*. [S.l.], 2013. p. 36–45. Citado 2 vezes nas páginas 42 e 55.

WANG, Shuo; MINKU, Leandro L; YAO, Xin. A systematic study of online class imbalance learning with concept drift. *IEEE transactions on neural networks and learning systems*, IEEE, n. 99, p. 1–20, 2018. Citado na página 27.

ZADROZNY, Bianca; LANGFORD, John; ABE, Naoki. Cost-sensitive learning by cost-proportionate example weighting. In: *ICDM*. [S.l.: s.n.], 2003. v. 3, p. 435. Citado 2 vezes nas páginas 32 e 39.

ZHANG, Chong; TAN, Kay Chen; REN, Ruoxu. Training cost-sensitive deep belief networks on imbalance data problems. In: IEEE. *2016 international joint conference on neural networks (IJCNN)*. [S.l.], 2016. p. 4362–4367. Citado na página 40.

ZHAO, H.; LI, X.; XU, Z.; ZHU, W. Cost-sensitive decision tree with probabilistic pruning mechanism. In: *2015 International Conference on Machine Learning and Cybernetics (ICMLC)*. [S.l.: s.n.], 2015. v. 1, p. 81–87. Citado na página 39.

Anexos

ANEXO A – Métrica de ponderamento do CSARF

A escolha de uma métrica de ponderamento mais representativa que a Acurácia em ambientes desbalanceados foi uma das primeiras alterações realizadas no ARF em direção ao método proposto. A primeira métrica foi escolhida com base em estudo teóricos e pela recomendação do seu uso, juntamente com o AUC-PR, para avaliar classificadores em problemas desbalanceados. O MCC foi originalmente implementado no protótipo do CSARF e chegou a render resultados que também apresentaram desempenhos melhores que o ARF para as métricas avaliadas. No entanto, a priori da execução do *grid search*, conduziu-se um experimento com o objetivo de verificar se a métrica MCC realmente era a melhor alternativa disponível à acurácia.

O protocolo experimental executado consistia em verificar se as hipóteses a seguir eram verdadeiras:

1. A métrica *Matthews Correlation Coefficient* (MCC) é a melhor métrica para ponderar os classificadores base do ARF entre as opções Acurácia, F1_{score} médio e *recall* médio.
2. Há diferença estatisticamente significativa entre as métricas avaliadas.

Para avaliar o potencial de cada métrica, utilizou-se o ARF como algoritmo base para experimentá-las como alternativas para ponderar os indutores do *ensemble*. Com base nisso, utilizou-se a estratégia *prequential* para conduzir os experimentos e avaliou-se apenas as *streams* de dados binários. A tabela 29 resume os parâmetros utilizados.

A tabela 30 traz em detalhes o ranqueamento obtido por cada alternativa, sendo elas ARF (acurácia), ARF_r (*recall* médio), ARF_f (F1_{score} médio) e ARF_m (MCC). Dentre todos os resultados obtidos, nota-se que a ARF_f conquistou a melhor posição em todas as métricas analisadas. Somente na métrica AUC-PR a versão ARF_m ficou entre os dois primeiros lugares, mas ainda assim abaixo da ARF_f. Para as métricas F1_{score} médio e *recall* médio, a versão ARF_r apresentou melhores performances que a ARF_m. Perante aos resultados obtidos, rejeita-se a primeira hipótese de que a métrica MCC é a melhor alternativa para ponderar os classificadores base do ARF. A segunda hipótese também é rejeitada pois o teste não paramétrico de *Friedman* sugeriu que não há diferença entre os métodos. Por essa razão, escolheu-se a métrica F1_{score} médio para ponderar os classificadores do método proposto (CSARF).

Tabela 29 – Parâmetros utilizados

ARF	
ensembleSize	100
lambda	6
initialTraining	1000

Tabela 30 – Ranqueamento dos métodos análisados

	AUC-PR				F1$_{score}$ médio				*Recall* médio			
	ARF	ARF$_r$	ARF$_f$	ARF$_m$	ARF	ARF$_r$	ARF$_f$	ARF$_m$	ARF	ARF$_r$	ARF$_f$	ARF$_m$
santd	4	3	1	2	1	1	1	1	1	1	1	1
elect	4	3	2	1	2	3	1	4	2	3	1	4
privt	3	4	1	2	4	2	1	3	4	2	1	3
agrwl90	4	1	2	3	4	1	2	3	4	1	2	3
agrwl95	4	1	3	2	4	1	3	2	4	1	3	2
agrwl99	4	3	1	2	1	1	1	1	1	1	1	1
gmsc	4	3	1	2	4	3	1	1	4	3	1	1
pozzl	1	2	3	4	1	1	1	1	1	1	1	1
airli	1	4	3	2	1	4	3	2	1	4	3	2
sea90	1	2	4	3	1	2	3	3	1	2	3	3
sea95	1	2	4	3	1	1	1	1	1	1	1	1
sea99	4	3	1	2	1	1	1	1	1	1	1	1
Rank médio	2.917	2.583	**2.167**	2.333	2.083	1.750	**1.583**	1.917	2.083	1.750	**1.583**	1.917

ANEXO B – Imbalanced Window

A janela deslizante possibilita o uso da heurística Picek e essa é influenciada diretamente pelo tamanho da janela. Os experimentos conduzidos exploraram os tamanhos [1,000, 5,000, 10,000 e 20,000] e as figuras 24 (AUC-PR), 25 ($F1_{score}$ médio) e 26 (*recall* médio) ilustram o impacto dessa variação nas métricas avaliadas. As linhas em cinza descrevem todos os resultados obtidos com a variação do parâmetro e a em vermelho representa a variação do tamanho da janela para a melhor configuração obtida. Dentre os resultados obtidos, observa-se que o tamanho da janela pouco afeta o desempenho da melhor configuração obtida. Há uma grande variação no desempenho alcançado em alguns *streams* de dados, porém não há uma proporcionalidade (direta ou inversa) do tamanho da janela com o desempenho na métrica obtida. Isso fica evidente quando nota-se que todas as linhas vermelhas, os melhores resultados, são ligeiramente afetados pela variação do tamanho da janela e essa variação pode estar presente em todos os resultados obtidos (Figura 25 - *stream poker*). Assim sendo, evidencia-se que a variação do parâmetro *imbalancedWindow* pouco impacta no poder de classificação da classe minoritária para os *data streams* analisados.

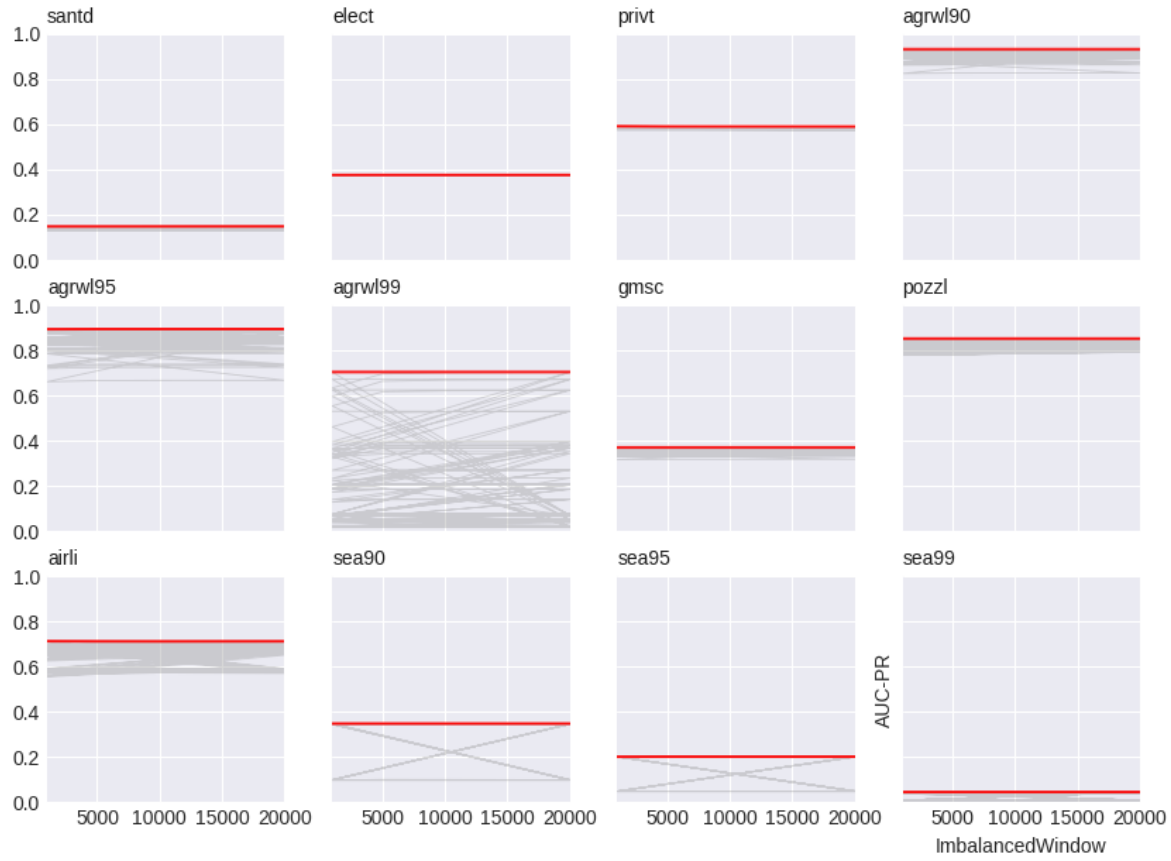


Figura 24 – Resultados obtidos com o *grid search* explorando o parâmetro *imbalancedWindow* para a métrica AUC-PR.

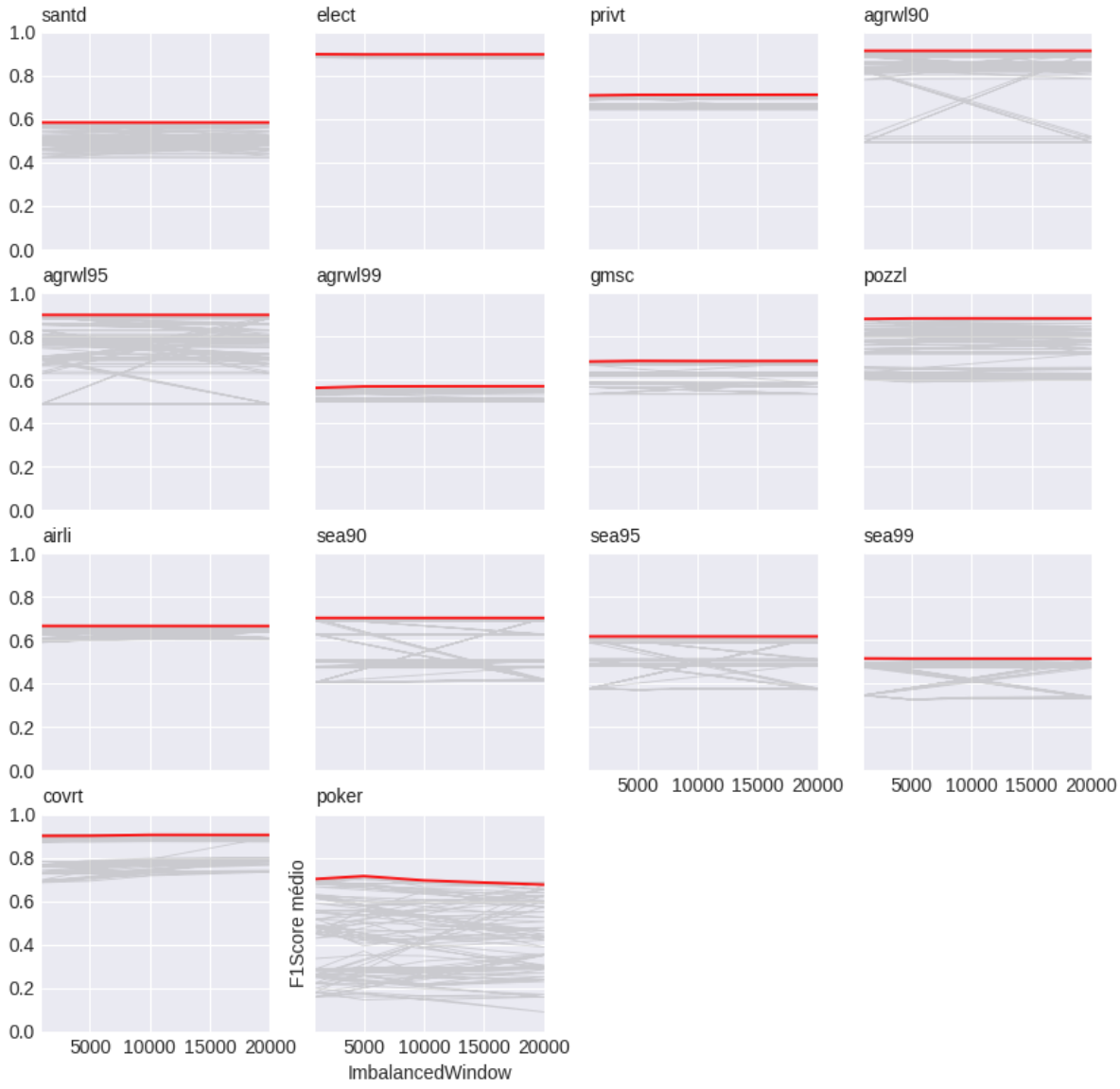


Figura 25 – Resultados obtidos com o *grid search* explorando o parâmetro *imbalancedWindow* para a métrica $F1_{score}$ médio.

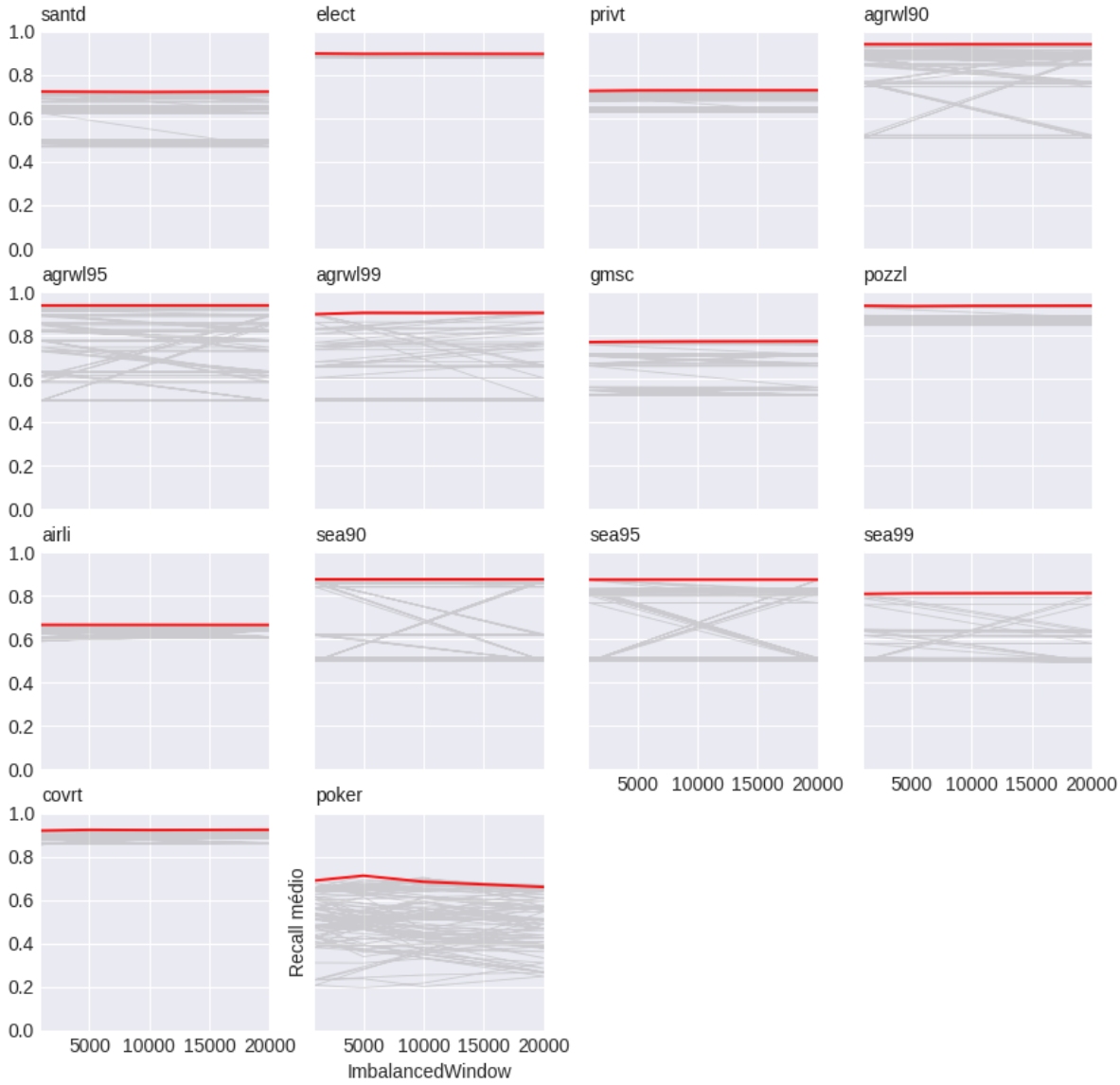


Figura 26 – Resultados obtidos com o *grid search* explorando o parâmetro *imbalancedWindow* para a métrica *recall* médio.